AUTOMATIC TEA BREWING THERMOS

By

Joseph Niemerg

Vincent Murphy

Danny Yi

Final Report for ECE 445, Senior Design, Spring 2018

TA: Nicholas Ratajczyk

2 May 2018

Project No. 60

Abstract

For our final project we decided on an automatic tea brewing thermos. This thermos will be able to take boiling water and regulate the time of steeping and temperatures for both steeping and drinking. In this document we will be discussing the design process, verification, and conclusions we can draw.

Contents

1. Introduction

In modern day America there is a sever lack of tea drinking. Linked to many health benefits it is a shame that so many go without it. This project is the solution to this problem by taking out all the hassles of drinking tea. Such as forgetting to remove the tea bag and over steeping it making the drink bitter and unbearable. In addition, not drinking the tea quick enough and having it go cold and be unsatisfying. Our device however will counteract these issues by brewing the tea automatically removing the tea bag from the water after a set amount of time. Additionally, our device will maintain a user set drinking temperature to keep the tea at a satisfying temperature. In the upcoming sections the objectives of our device will be further explored. Followed by the physical design work done than the verification of our design. Lastly the cost of the product will be analyzed followed by a conclusion. A conclusion which will lay out our accomplishments, uncertainties, and the ethical considerations we considered for the cup.

1.1 Team Project Overview

The Automatic Tea Brewing Thermos is an easy to use, mobile, and automated tea steeping thermos. It requires the user to input hot-boiling water, his/her favorite teabag, along with a desired steeping temperature, steeping time, and drinking temperature for a customized personal experience.

It uses active heating to steep and keep the tea at the desired temperatures and a motor to steep the teabag.

Active heating is done by comparing the user's value with a temperature sensor's reading value and sends current, via a power relay, through a nichrome wire when the temperature falls below a threshold value.

It will use a DC motor and an H-bridge to reel the tea bag up and down and will use the user's steep time to decide when to turn the motor on. The user's inputs will be obtained by either an LCD screen with buttons or a Bluetooth App.

1.2 High Level Requirements

- Thermos must be able to automatically brew tea with the user either pressing buttons or using the Bluetooth connected phone app.
- Thermos must keep water at desired comfort level ranging from 70-110°F for up to 20 min ideally.
- Thermos must be able to operate from battery to keep it mobile.

2 Design

In this section we will be discussing the various design processes we undertook to complete our tea brewer. We broke up in the same way we did with our block diagram being modularly with the first

section dealing with the Bluetooth and the associated software. We will then discuss the microcontroller and all control components, and then the power. Then finally we will discuss the physical design.

2.1 Circuit Schematic

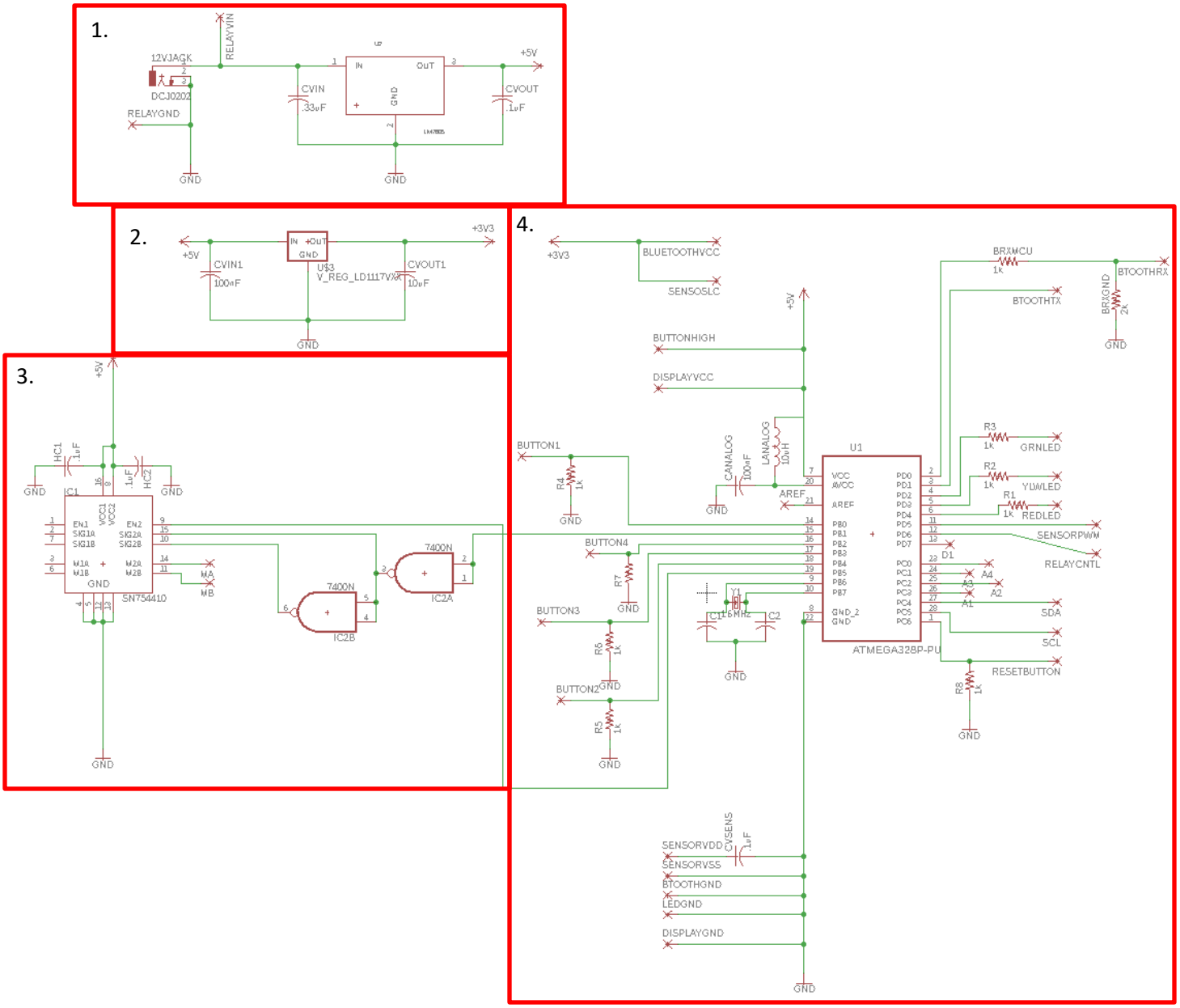


Figure 1 Full Circuit Schematic

Figure 1 above shows all the design work done for the circuit schematic. Including boxes splitting up the circuit into four different blocks. In box 1 there is the 12v supply with the relay connections for the nichrome heating element along with the 12-5v voltage regulator. To connect the 12V supply to the PCB board a simple DC barrel-jack will be used. From there the 12v signal is split to a wire pad that will connect to an external relay and then to the input of our LM7805 voltage regulator. Lastly, capacitors were added to the input and output terminals of the regulator to stabilize the voltages

and were placed according to the datasheet.[1] The purpose for the LM7805 is to supply power to our microcontroller, other IC chips, and LCD screen that require 5v.

Box three contains the 5-3.3v Voltage regulator. This regulator is the LD1117 model and like the LM7805 in box one will have input and output capacitors as specified in the datasheet.[2] The reason for the LD117's implementation is for our supply voltage for the HM-10 Bluetooth chip and MLX90614 IR temperature sensor. The design decision for a voltage regulator over a resistor voltage divider circuit was the issues with injecting more heat into the system. Since a resistive heating element produces power in heat equal to the current squared times the resistance or $P = I^2R$. [3]

Box three contains the H-bridge motor controls that will control the ascent and descent of our tea bag. A control signal is passed through a SN7400 NAND gate chip where the signal is split into two. One that is inverted, and one that is not. This is necessary for our SN754410 H-Bridge IC chip as it requires two signals to operate.[4] In addition, a third signal is passed to the H-Bridge chip as an enable bit. The two signals from the NAND chip work in conjunction with one another and give direction to which way the DC motor should spin. If signal A is high and B is low the motor will spin one direction, and if reversed the motor will also spin in the reverse direction. This allows us to use the motor as a reel system for the tea bag. Raising and lowering it into the water at specified times. The implementation of the NAND gate has been added after the design review due to a misunderstanding of the H-bridge datasheet.

Lastly box four which houses the microcontroller, wire pads for the buttons, LCD, temperature sensor, Bluetooth, analog inputs, the encoder connections, LEDs, along with the communal ground plane. The microcontroller that is used in our design is the ATMEGA 328P with a 28-pin configuration. This was used as it was low power and gave us plenty of processing power for the sensors. Connected to the digital I/O pins are five buttons; four for the user interface, and one for the code's system reset. Also, three status LEDs that let the user know what part of the brewing cycle the thermos is in. In addition, the control signals for the motor, relay, and data from the encoder are connected via the digital I/O. Next the analog I/O pins have two pins that are used for the communication with the LCD display and the temperature sensor. The two devices can share the same pins due to the I2C bus on analog pins four and five. With the MCU set up as a master and the LCD and temperature sensor set up as slaves. The MCU can communicate to each device individually using their slave identification. This allows us to use less pins on our board in case more devices need to be added. There are then four excess analog pins that are connected to a wire pad in case components need to be added in the future. Finally, a crystal oscillator is connected using the ATMEGA's data sheet, along with a inductor/capacitor circuit for the analog voltage input.[5]

2.1 PCB

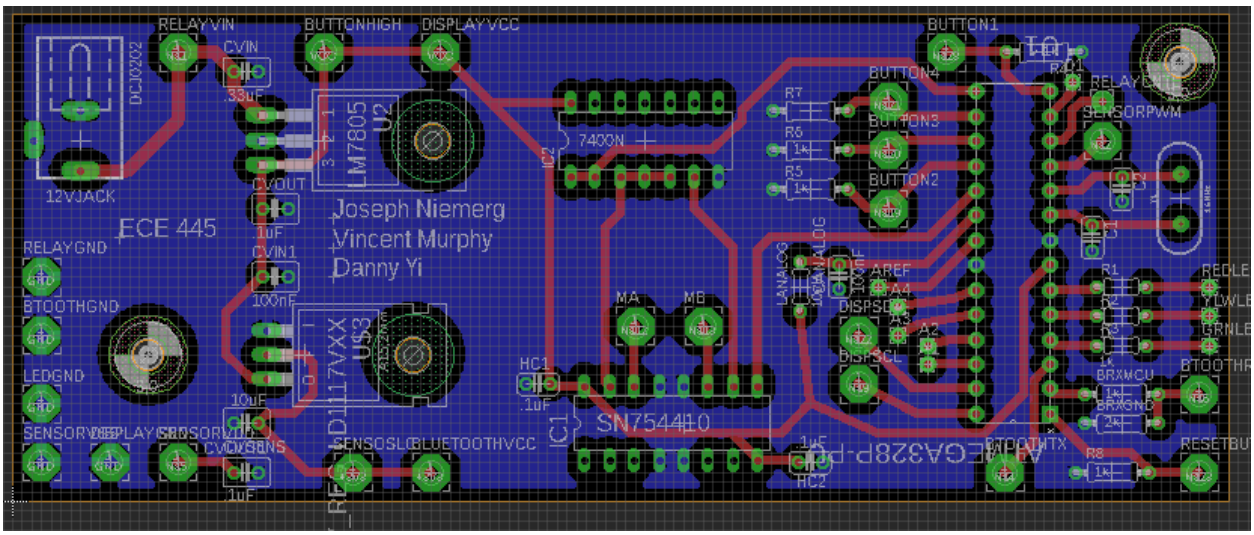


Figure 2 Final PCB Design

By using the circuit schematic shown in figure 1 the following PCB board was made as seen in figure 2. The PCB is a 5-inch by 2-inch design to fit in our electrical housing box to put a gap between the PCB and LCD screen for buttons to be mounted. The board uses a single large grounding plane. Since there is no component with a necessity on high accuracy precision there is no downfall to making a ground plane this way. Instead it allows for easier routing of the ground connections.

2.2 User Interface

There are two (2) ways for users to input a desired steep temperature, steep time, and drinking temperature. The first way is through the navigation of the LCD screen via four (4) buttons. The other way is through a phone application that will communicate with the microcontroller via Bluetooth.

2.2.1 Design Considerations of LCD Screen and Buttons

A 20 x 4 LCD with and I2C adapter was chosen to reduce the number of pins required from six (6) to two (2): SCL and SDA.

How many buttons to use:

Minimum buttons for the user to easily understand and to progress through different screens:

1. Back
2. Decrease value
3. Increase value
4. Next/Submit

States/different screens (see appendix C):

1. Steep Temperature Input
2. Steep Time Input
3. Drink Temperature Input
4. Verification

Size of LCD screen (see fig. 3-7):

The largest characters on a single screen: Verification Screen

A total of *four* (4) lines are needed (see fig. 7):

- *Three* (3) lines for each input variable (steep temperature, steep time, drink temperature)
- *One* (1) line for prompting an option to go back to edit values or submitting values to microcontroller.

The width consists of *twenty* (20) characters:

- Input variable name consists of ten (10) characters:
  - "Drink" and "Steep" are both five (5) characters long
  - "Temp" and "Time" are both (4) characters long
  - A single (1) space character
- Units will consist of a maximum of five (5) characters:
  - The units for temp is "(F)" which is three (3) characters long
  - The units for time is "(min)" which is five (5) characters long
- Additional spaces and special characters for easy interface to read

Design Layout for LCD Screen:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2																			
3																			
4																			

Figure 3. 20 x 4 Character Screen

S	T	E	E	P		T	E	M	P	(	F	)	:				1	8	5
															N	E	X	T	>

Figure 4. Steep Temperature Input Layout

S	T	E	E	P		T	I	M	E	(	M	I	N	)	:				4
<	B	A	C	K											N	E	X	T	>

Figure 5. Steep Time Input Layout

D	R	I	N	K		T	E	M	P	(	F	)	:					9	0
<	B	A	C	K											N	E	X	T	>

Figure 6. Drink Temperature Input Layout

S	T	E	E	P		T	E	M	P	(	F	)	:				#	#	#
S	T	E	E	P		T	I	M	E	(	M	I	N	)	:				#
D	R	I	N	K		T	E	M	P	(	F	)	:					#	#
<	B	A	C	K									S	U	B	M	I	T	>

Figure 7. Verification Layout

2.2.2 Design Considerations for Phone Application

The phone application will transmit all three (3) of the user's custom values, in the order of the state diagram, in a form a string and separate them by the comma character. The microcontroller will utilize the comma character and separate the string into three (3) substrings and convert them into its corresponding integer variables. After obtaining integer values from the phone app, the state diagram will transition directly to verification screen. This accounts for errors that may occur either by corruption of data or user error of hitting the transmit button when not ready.

The data will be sent to a HM-10 Bluetooth chip that utilizes the Universal Asynchronous Receiver/Transmitter (UART) protocol. It converts the incoming bytes of data into a serial bit stream with a start bit and a stop bit. Since it is asynchronous, there is no incoming clock signal. Instead, the UART generates its clock internally to the microcontroller and synchronizes with the data stream's start bit [6]. For this to happen, the receiver would need to know the baud rate ahead of time, which was initialized as 9600.

The phone application interface is constructed using the MIT App Inventor online application [7]. To use the functionality of BLE devices, a separate extension had to be downloaded and installed [8].

User Interface

It will consist of thirteen (13) important features (see fig. 13):

(1) Scan for DSD TECH Button: Click this button to scan for "DSD TECH", which is the name of or Bluetooth module. Clicking this button generates and displays a list of all nearby BLE devices.
(2) Stop Scan Button: Click this button to stop the list from scrolling.
(3) *Connect Button*: Clicking this button establish a connection to the BLE device.
(4) *Disconnect Button:* Clicking this button will disconnect the phone app from the selected BLE device.
(5) *Status*:
   a. Scanning: Scanning for all nearby BLE devices
   b. Stopped Scanning: Stop scanning and updating list of all nearby BLE devices
   c. Connected: Successfully connected to the Bluetooth device.
   d. Not Connected: Unsuccessfully/not connected to any Bluetooth device.
(6) Bluetooth Device: Displays which BLE device is connected to account for any user error in choosing the BLE device.
(7) *Steep Temperature Slider Value Text Box*: numerical value read from the slider (4). The default value of 185°F is the suggested temperature value designated by us.

(8) *Steep Temperature Slider*: an interactive module that slides left (to decrease) or right (to increase) the user's desired steep temperature. The default position of 185°F, the minimum value of 160°F and maximum value of 210°F is designated by us.
(9) *Steep Time Slider Value Text Box*: numerical value read from the slider (6). The default value of 3min is the suggested steep time value designated by us.
(10) *Steep Time Slider*: an interactive module that slides left (to decrease) or right (to increase) the user's desired steep time in minutes. The default position of 3min, the minimum value of 2 min and maximum value of 10 min is designated by us.
(11) *Drink Temperature Slider Value Text Box*: numerical value read from the slider (8). The default value of 90°F is the suggested drink temperature value designated by us.
(12) *Drink Temperature Slider*: an interactive module that slides left (to decrease) or right (to increase) the user's desired steep temperature. The default position of 90°F, the minimum value of 70°F and maximum value of 110°F is designated by us.
(13) *Submit button*: Once the user has chosen the desired values and is connected to the HM-10 Bluetooth module, this button will send the values to the microcontroller.

2.3 Design Considerations for Control Software

The next system we will be focusing on is the control software loops. We will be running two separate control loops. One will be for the motor control and the other will be our temperature control. Both control loops will be proportional control and their values determined by one sensor.

The first control system we will discuss is the motor control. We knew in order to introduce the tea bag to the water we needed a solution which would be able to raise and lower the tea bag outside of the cup and it made sense to us to have a DC motor do this as we will be able to easily power it with the 5V line we are using to power the majority of our other modules. The motor control consisted of the following components: 7400 NAND gate, SN754410 H-bridge chip, a DC motor, Keyes – 040 Rotary Encoder, and a potentiometer. The latter two were added late into the design of this module and resulted in us having to make a circuit differ a bit from what we had originally intended. We decided on the addition of the rotary encoder due to the control loop being open without some feedback on the motor's position and if we had used a timer as we had originally planned, there is a chance after repeated usages we would be off by quite a bit and have no way to tell if the motor had been pulled manually by the user. With the problem of position out of the way our next major problem was the motor speed. We found through testing the motor and the encoder through Arduino that the encoder was unable to update fast enough to keep track of the motor at full speed. This in turn required us to create a new motor circuit which included a potentiometer between the two leads of the motor in order to limit current into the motor and slow it down to the point where we could efficiently measure the number of rotations.

The next control system which we had to implement was the heating element control. As seen in the figure 1 in order to control the heating element we chose to use the circuit in only an ON/OFF mode where the relay completes the heating circuit. We were able to use this since as seen in Appendix 1 we are able to do this since it takes a very long time for the water to heat up as seen in that test. We are able to read the temperature using an IR sensor specifically the MLX90614. This gave us some

flexibility when it came to how to implement the sensor. It gives both object and ambient temperature which gave us flexibility on placement since on the time of choosing a part we were unsure of the placement of the sensor. The next portion of the heating element control was ensuring we had a timer so we knew when to change the desired temperature. The Appendix C diagram shows how we needed a timer to reach the next state being the raise tea bag state. We did this in Arduino using the pre-made millis() function which returns the time the processor has been running in ms and recording a start time when states change and then comparing a new millis() entry to the start time entered. This allowed us to get a timer for our control software. These made our requirements for the control software given in Appendix A.

2.4 Design Considerations for Power

The next major hardware module we needed to design was the power module. We needed a power supply which would meet our high level requirements of being both portable and able to power resistance heating wire. This left us with a problem of weight and size. It was hard to find a battery option which is both above 5V and compact. We wanted a Lithium Ion battery so the brewer was rechargeable. We ended up finding one which is 12V, 9800mAH, and has 2A max. This allowed us to use this to directly power the resistance wire and still have enough current draw left to power our circuit components. As for other components for power we needed two separate voltage levels to power our remaining circuit components. For this we used a series of voltage dividers and capacitors shown in Figure 1 .

2.5 Physical Design

Figure 8 Finished Cup Design

Shown in figure 8 is the outcome of our physical design work. The design consists of two cups; an outer and inner cup. The outer cup will act as a shell for the design and will be where most of our external electronics are mounted. The cup has a 5-inch diameter and a 7-inch height. The inner cup is where the tea will be stored and flares out at the opening to allow for a portion of it to rest on the top of the outer cup. This flaring allows us to place the lid on top and with pressure from the screw on lid can keep it secured to the assembly without the need for screws or other methods of attachment. This promotes ease of access to the cup and lets the user take the inner cup out easily to wash it. The inner cups dimensions will be a 5-inch height (flare included) and a 3-inch diameter. This will house roughly 16-20 ounces of water depending on how far up the flared cone the water is poured. In between the outer and inner cup there are plans to add pipe insulation to protect the outer cup from the heat produced by the inner cup and nichrome wire. This is a new addition since the design document that was thought of for added thermal protection.

The handle seen in figure 8 is a simple design with a 4-inch inner gap height, a 2-inch inner gap width, and a half-inch thickness. This was chosen arbitrarily as a comfortable distance for hands. The handle is screw mounted to the outer cup.

At the base of the outer cup is where a nichrome wire heating element will be mounted. For ease of access the plan is to have the outer cup's base be screw mounted with flush screw heads to make access to the nichrome element easy. This element will fit in a 3-inch diameter with a 2-inch height and will provide the heating for the inner cup.

Lastly, the electrical box can be seen in figure 8. This box will be mounted to the side of the cup and will have a 5x5x2(LxWxD)-inch design. Inside this is where the battery, PCB, relay, temperature sensor, encoder, and LCD screen along with LEDs will be mounted. In addition, the temperature sensor will be mounted on the connecting wall to the cup with a hole through both the box and outer shell of the cup to measure the inner cup directly with IR.

3. Design Verification

This section we will use to discuss our testing methods and our verification of the various components. We will begin with the LCD screen and buttons being verified together through the usage of the software in the user input stage. Next we will discuss our testing methods of the control software and that has an overarching verification of our other components.

3.1 LCD and Buttons Verification

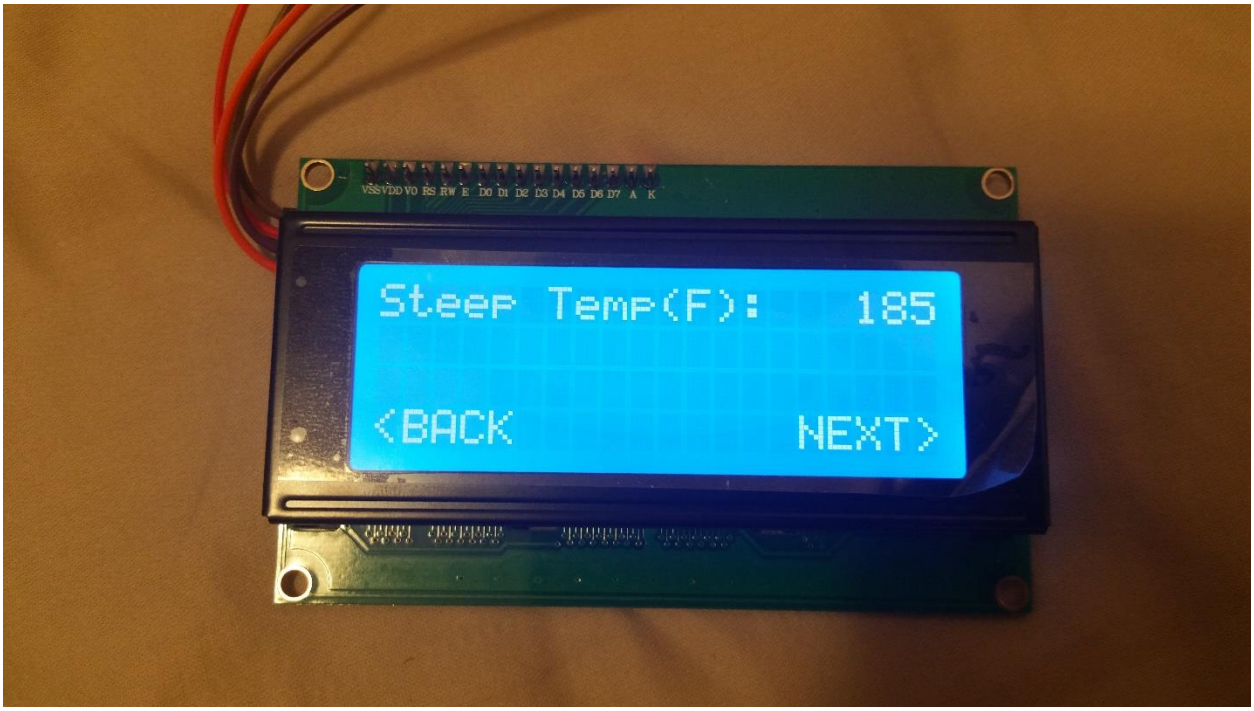


Figure 9. Testing of the LCD Screen

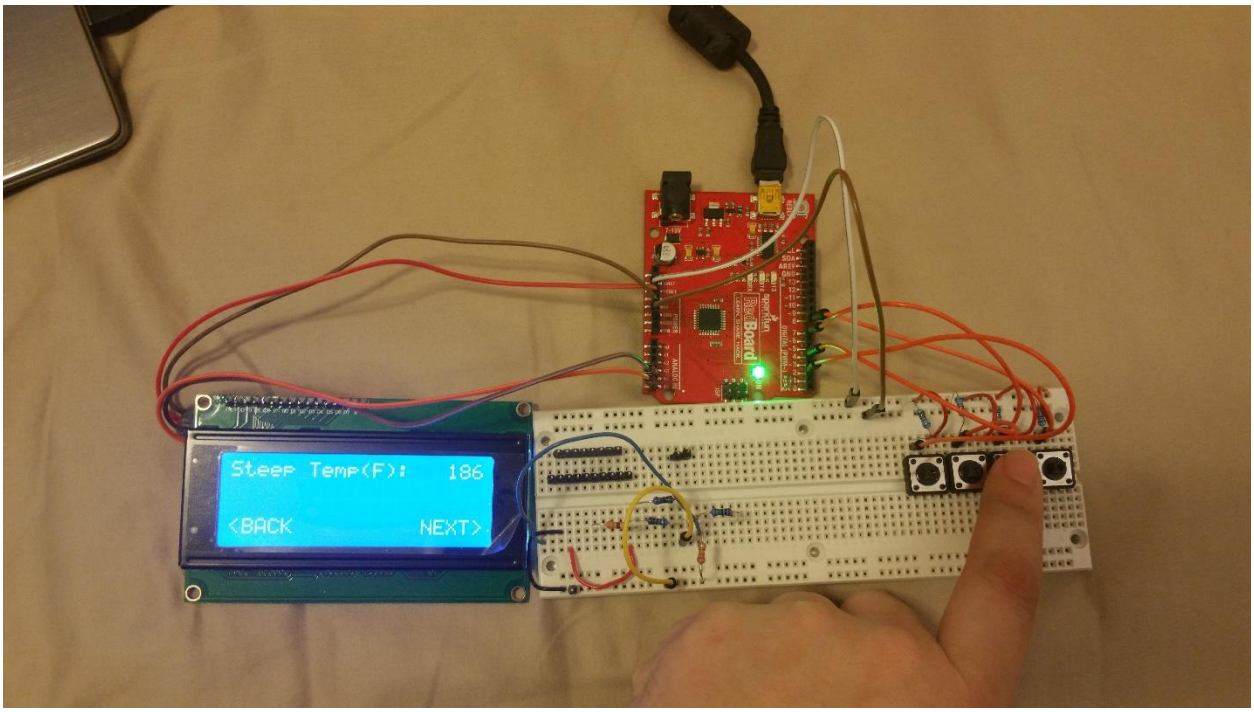


Figure 10. Testing of the Increase button (from 185 to 186)

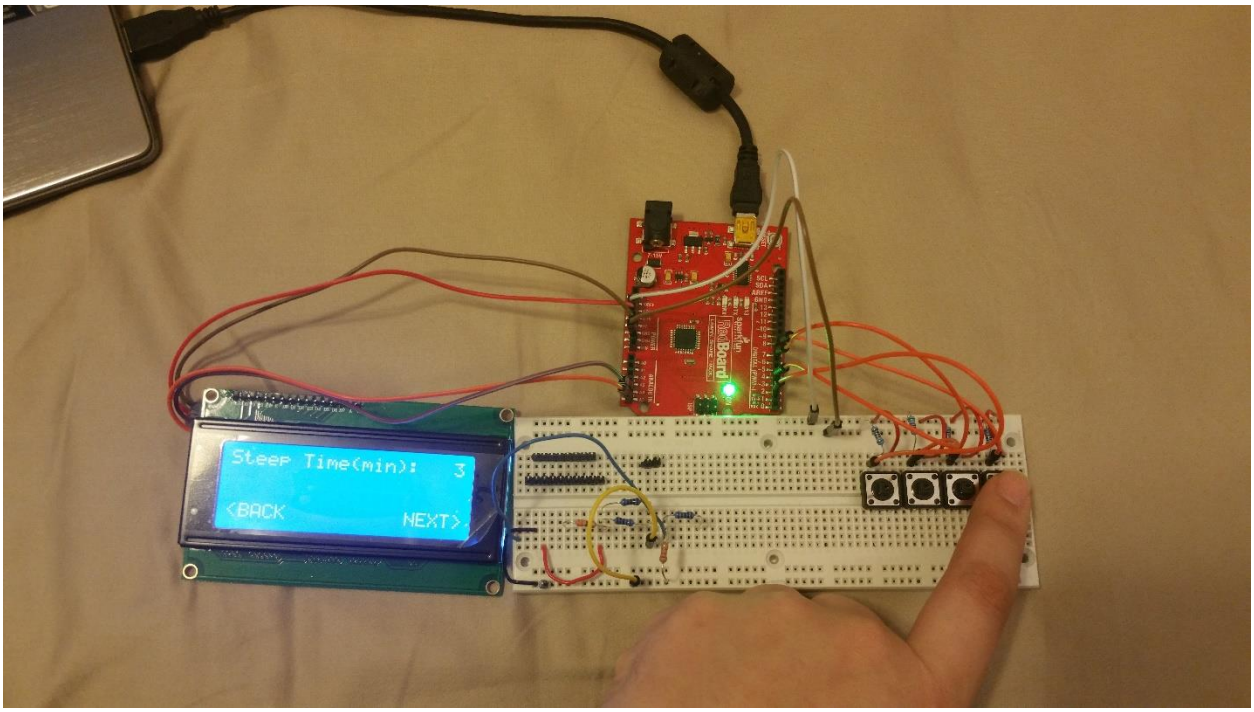


Figure 11. Testing of the Next Button (Steep Temp Input Screen to Steep Time Input Screen)

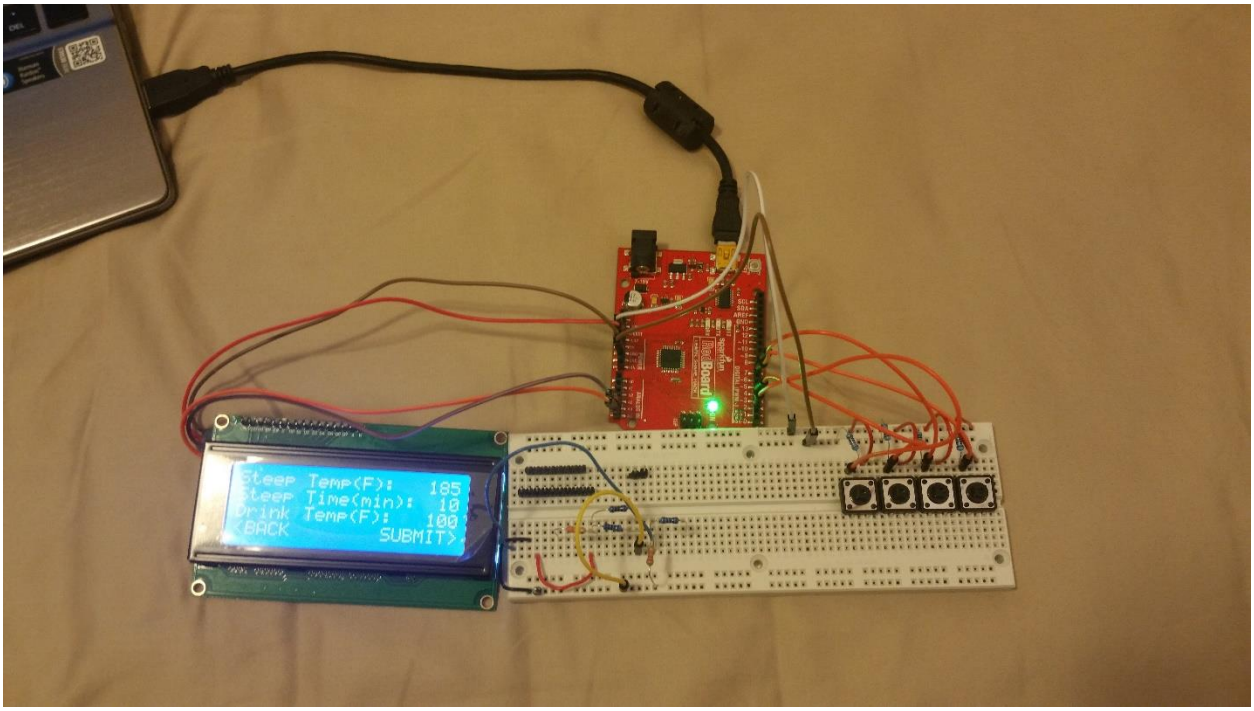


Figure 12. Verification Screen

3.2 Phone Application Verification

Numbers correspond to section 2.2.2 User Interface Design.

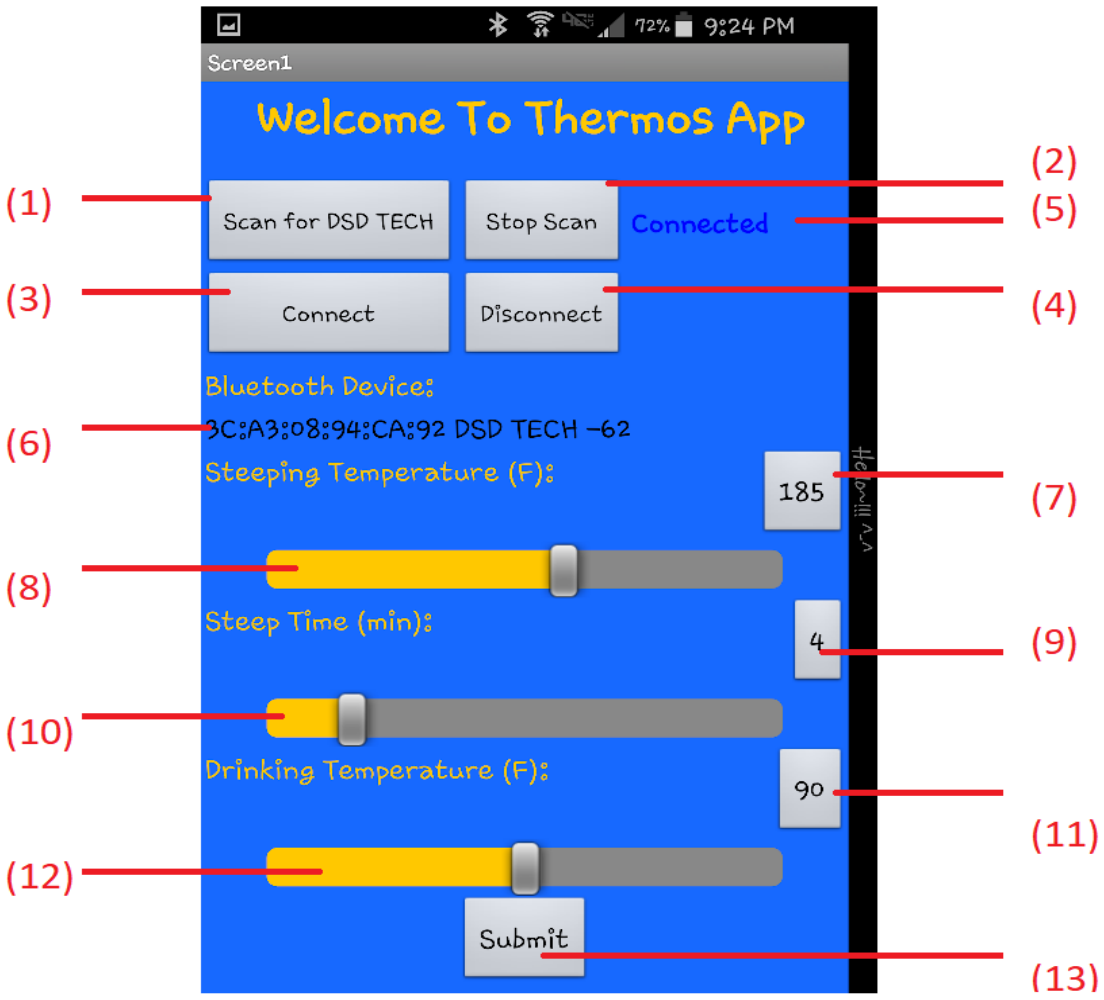


Figure 13. Phone App Interface

3.3 5 V Voltage Regulator

For this test we set up the circuit shown in Appendix *. We then sweep the voltage from 12-14V the following table recorders the data we have on that.

Table 1 5V Unit Test Results.

Vin (V)	Vout (V)	Iout (A)
12	5.1	0.79
12.5	5.1	0.8
13	5.2	0.78
13.5	5.3	0.77
14	5.3	0.78

3.4 3.3 V Voltage Regulator

For this test the circuit was set up as seen in Figure 1. We then swept the voltage from 5.5v to 4.5v since the input of the device would be the direct output of the 5v voltage regulator.

Table. 2 3.3V Unit Test Results

Vin(V)	Vout(V)
5.5	3.3
5.3	3.3
5	3.3
4.8	3.3
4.6	3.3
4.5	3.3

3.5 Control Software & Control Components Verification

In order to prove the software works we took a video of the operation of the software while hooked up according to the circuit shown in Figure 1.

Software Test: https://www.youtube.com/watch?v=M04rv0zZy5s

IR Sensor Test: https://www.youtube.com/watch?v=5fE5v2imhqQ

3.6 Nichrome Wire

For this test the nichrome wire was wired in directly to a 12v power supply and run continuously. The wire used was 30 gauge that was 15 inches in length and at the 12v input would consume 1.42A. From the data shown below the device was found to raise the water temperature by 2˚F every 5 min.

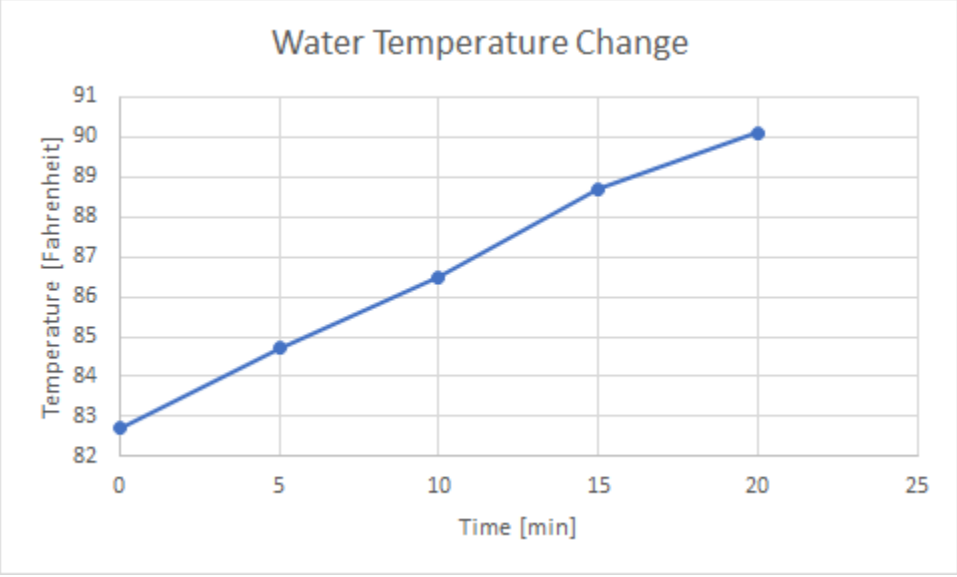


Figure 15. Temp Results

4. Costs

4.1 Parts

Table 3 Parts Costs

Part	Retail Cost ($)	Actual Cost ($)
12V Li-Ion Battery	40.99	40.99
5V Voltage Regulator	5.99	5.99
3.3V Voltage Regulator	1.95	1.95
Power Relay	5.95	5.95
30 Gauge Nichrome Wire	6.89	6.89
DC Gear Motor	3.95	3.95
Encoder	1.99	1.99
H Bridge	6.50	6.50
NAND chip	0.95	0.95
HM-10 Bluetooth Chip	9.99	9.99
IR Temperature Sensor	19.95	19.95
20x4 LCD	8.95	8.95
Push Button (4)	0.85	3.40
LED (3)	0.95	2.85
ATmega328p Microcontroller	2.20	2.20
Total		122.50

4.2 Labor

Table 4 Labor Costs

Labor Hours	$/hr	Laborer(s)	Total
100	$25.00	3	$7,500.00

5. Conclusion

5.1 Accomplishments

We have successfully integrated the different modules to perform the task of brewing tea with custom temperature and time values- all while maintaining its portability. The different modules include:

- Bluetooth module to transmit custom values from a phone application to the microcontroller
- Sensor module to constantly read current temperature readings of the inner cup
- Motor control module to raise and lower the tea bag according to the encoder value
- Heating module to turn on when the temperature
- Controller module to send proper signals to all other modules
- Power module to power on all the other modules

5.2 Uncertainties

During the testing of the motor, depending on how the tea bag is wrapped around the motor affected the encoder value. The distance between the temperature probe and inner cup affected the accuracy. The ideal distance was marked, but mounting the probe into the hole of the outside cup shifted the probe. The amount of liquid that would leak from the inner cup to the outer cup would vary depending on how well the inner cup sat in the insulation.

5.3 Ethical considerations

As for the safety of our design we will be greatly concerned with our Li-Ion battery. We will have to make sure to keep it within the correct operating temperatures, 0 ~ 45℃ for charging and -20 ~ 60℃ for discharging. We will attempt to avoid these risks by mounting the battery in the handle the area furthest from the bottom where the heating elements are mounted in order to give us a lot of separation in case of any failure in the insulation around the heating elements.

Along with this we know as state in the article “Safety of Lithium-ion batteries” [9] that as well as operating temperature, we must also ensure we regulate the current flow to and from the battery. The specific values are 6A maximum continuous discharge/charge. We want to ensure we are following the IEEE Ethics part 9 [10] “avoid injuring others, their property, reputation, or employment by false or malicious action”. Any mishaps with the battery would certainly cause injury or property damage. This is why we will take our testing very seriously with the battery as we are aware if mishandled these batteries are very dangerous. We are going to ensure that these values are not reached in our circuit by testing the circuit rigorously.

Some important ethical issues intertwined with safety also are included. We have to make sure our heating element is properly insulated. We want to ensure that is does not cause any harm to others as state in the ACM Code of Ethics “loss of property, property damage, or unwanted environmental impacts.” [11] In our case if we do not properly make the heating element safe for the user to not burn or start a fire. We can do this by properly regulating the voltage across our heating element. We will include a safety that will shut off power to the heating element if a certain voltage is reached. This will ensure the heating element, regardless of sensor failure, will not reach certain temperatures.

We will also to make sure our hot tea is safely insulated and safe to drink. Mention in the IEEE code of conduct it is stated we are "to hold paramount the safety, health, and welfare of the public" [10] . We must do this by ensuring that our temperature sensors are functioning at all times to alert the user if the drink is too hot to drink. If our microcontroller receives data from our sensors which is not within its set range the user will be notified, also the sensor we are using sends a fault signal and this will also notify the user notified in this case that the sensor isn't working, and all heating elements will be shut off.

We also want to ensure all testing in the lab is safe. We are going to test all parts of our design as we receive them after buying them to verify they meet rated criteria. After this we will test our design modularly without attaching to PCB. Then integrate it outside of the PCB, then finally we will integrate our design and test again to ensure our integration was conducted properly. Throughout this process we will have to test the individual components and modules in a safe, controlled way. We will ensure all rated constraints are not exceeded in our tests to make sure we minimize part failure. We also will isolate our heating element in a way such that it will not cause harm to us or our testing equipment as well as other students in the lab.

5.4 Future work

For future work, we would like to improve on the aesthetics by reducing the size of the electrical box. This would require the sacrifice of the LCD, LEDs, and buttons. In return, it would allow us to concentrate on improving the Phone app's interface to display everything the LCD would have. We would also need to find a smaller battery with matched power of the current battery and an optimized PCB to fit in a smaller housing unit.

References

[1]'LM7805 Datasheet'. [online]. Available. https://www.sparkfun.com/datasheets/Components/LM7805.pdf [Accessed 3/21/2018]

[2]'LD117V33 Datasheet'. [Online] Available https://www.sparkfun.com/datasheets/Components/LD1117V33.pdf [Accessed 3/21/2018]

[3]'Resistive heating explained in details'. 7/27/2011. [Online]. Available http://electrical-engineering-portal.com/resistive-heating-explained-in-details [Accessed 3/21/2018]

[4]'SN754410 Quadruple Half-H Driver' [Online] Available http://www.ti.com/lit/ds/symlink/sn754410.pdf [Accessed 3/21/2018]

[5]'ATmeage328/P Datasheet Complete' [Online] Available http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf [Accessed 3/21/2018]

[6]'USART vs UART: Know the difference' [Online] Available: https://www.edn.com/electronics-blogs/embedded-basics/4440395/USART-vs-UART--Know-the-difference [Accessed: 2/4/2018]

[7]'BluetoothLE' [Online] Available: http://iot.appinventor.mit.edu/#/bluetoothle/bluetoothleintro [Accessed: 3/20/2018]

[8]'How To Build Custom Android App for your Arduino Project using MIT App Inventor' [Online] Available: www.howtomechatronics.com/tutorials/arduino/how-to-build-custom-android-app-for-your-arduino-project-using-mit-app-inventor [Accessed: 2/4/2018]

[9]'Safety of lithium-ion batteries' June 2013 [Online]. Available: http://www.rechargebatteries.org/wp-content/uploads/2013/07/Li-ion-safety-July-9-2013Recharge-
.pdf [Accessed: 2/4/2018]

[10]'IEEE Code of Ethics' [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html [Accessed: 2/5/2018]

[11]'ACM Code of Ethics and Professional Conduct' 10/16/92. [Online]. Available:

https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct [Accessed: 2/2/2018]

Appendix A Requirement and Verification Table

Table X System Requirements and Verifications

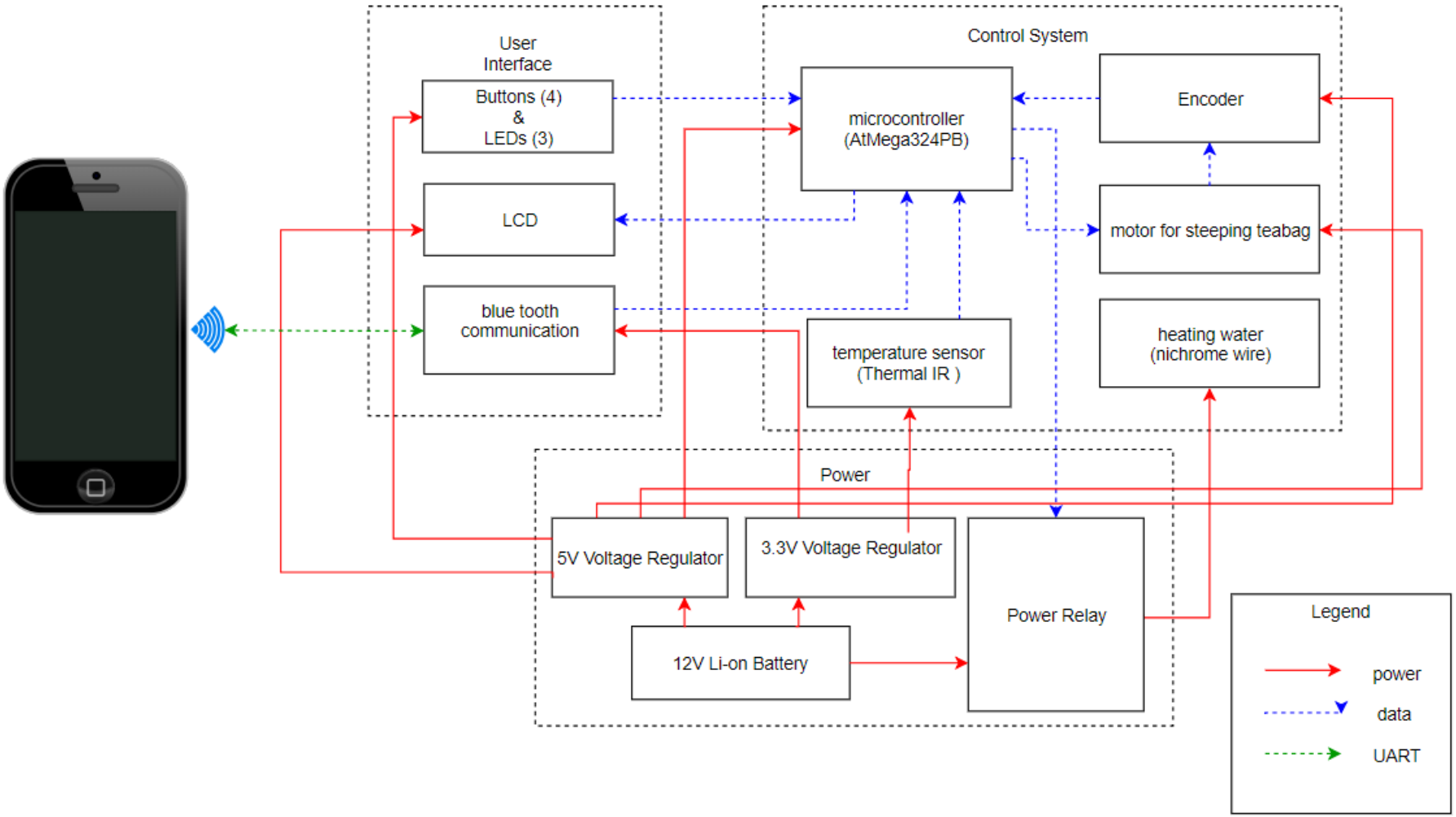
Buttons and LED		
Requirements	**Verification**	**Verification status (Y or N)**
1. Buttons must be easily press able and accessible by the user. 2. LED must be visible from up to 15 ft away.	1A. Do qualitative testing on the buttons and LEDs making sure they work. 1B. Use a voltmeter to test high side voltage to be within 5±1%V. 2A. Stand at a distance of 15ft away and determine if the LED are visible.	Y Y Y
LCD Display		
Requirements	**Verification**	**Verification status (Y or N)**
1. LCD must be easy for user to understand with a maximum of 80 characters. 2. LCD screen will show a readout of the current temperature and other valid info in the standby phases.	1A. Ensure LCD is legible and has good contrast between background and characters, and also having proper fitting words. 2A. While doing the verification for the design code, and for 1A the screen will be qualitatively analyzed to make sure the proper data is displayed.	Y Y
HM10 Bluetooth Controller		
Requirements	**Verification**	**Verification status (Y or N)**
1. Can operate with current of around 8.5mA during active state. 2. Maintain thermal stability below 125˚C 3. Communicates in a range of 5-10m.	1A. during active state, measure voltage drop across 1k resistor. 2A. During verification of 1A use an IR thermometer to read the temperature of the device. 3A. Connect and pair the device with the phone and measure the range between them and check to see if the temperature values transmitted from the phone to the device at the 10m maximum..	Y Y Y
“Thermos” Phone app		
Requirements	**Verification**	**Verification status (Y or N)**
1. Phone app must be usable on an android phone and have an easy to follow UI. 2. The phone app must be able to generate and	1A. Qualitatively assess the accessibility of the app on a group members phone and be able to use the app’s UI to control the temperature for drinking/steeping and steeping time.	Y Y

submit values via Bluetooth for the steep temp/time, and drinking temp.	2A. While verifying 1A ensure that the phone app can transmit the data via Bluetooth to our device and have the data displayed on the LCD screen.	
Microcontroller		
Requirements	**Verification**	**Verification status (Y or N)**
1. The microcontroller must be able to take in sensor data from the IR sensor and use that to control voltage across the heating element via the relay. This must work with no user input controlling the relay. 2. Ensure that the microcontroller can receive data from the Bluetooth phone app. 3. Keep current draw under .3 mA.	1A. We can verify this by testing the temperature sensor on a controlled surface with another IR thermometer to verify the accuracy. We can then change the temperature and see if the voltage on the relay changes accordingly. 2A. Set phone app on and have it transmit a series of commands to the microcontroller. Then verify that these specific commands are read through a simple program by switching the digital relay of our circuit on and off. 3A. During normal operation we will monitor the current draw from the power source powering the chip and ensure we do not exceed .3 mA.	Y Y Y
Control Software		
Requirements	**Verification**	**Verification status (Y or N)**
1. Read temperature sensor data and be able to compare it to values obtained from the two different UI to check if certain conditions have been met and respond accordingly. 2. Read encoder data to determine when the motor has spun the required distance to steep the tea bag. 3. Display requested variables on the LCD display. 4. Take in data from the Bluetooth device and be able to integrate the data to set bounds on	1A. During verification of the temperature sensor check to make sure that the data being read is able to be stored in a variable and use conditionals and output if the two values match to an LED and see if it turns on. 2A. Hook the encoder up to the microprocessor and turn it until a set value is reached. Upon reaching the value have the controller turn on an LED. 3A. Send words to the LCD screen and see if they are displayed properly. 4A. Use the phone app to set values for our thermos and use the serial printout that the Arduino code must verify that the values sent from the phone app match the values received. 5A. with the Nichrome wire send a signal to the motor and see if the conditional n the software turns off the relay disconnecting the nichrome wire from power. 6A. While verifying 5A see if the relay is responsive to the microcontrollers command.	Y Y Y Y Y Y

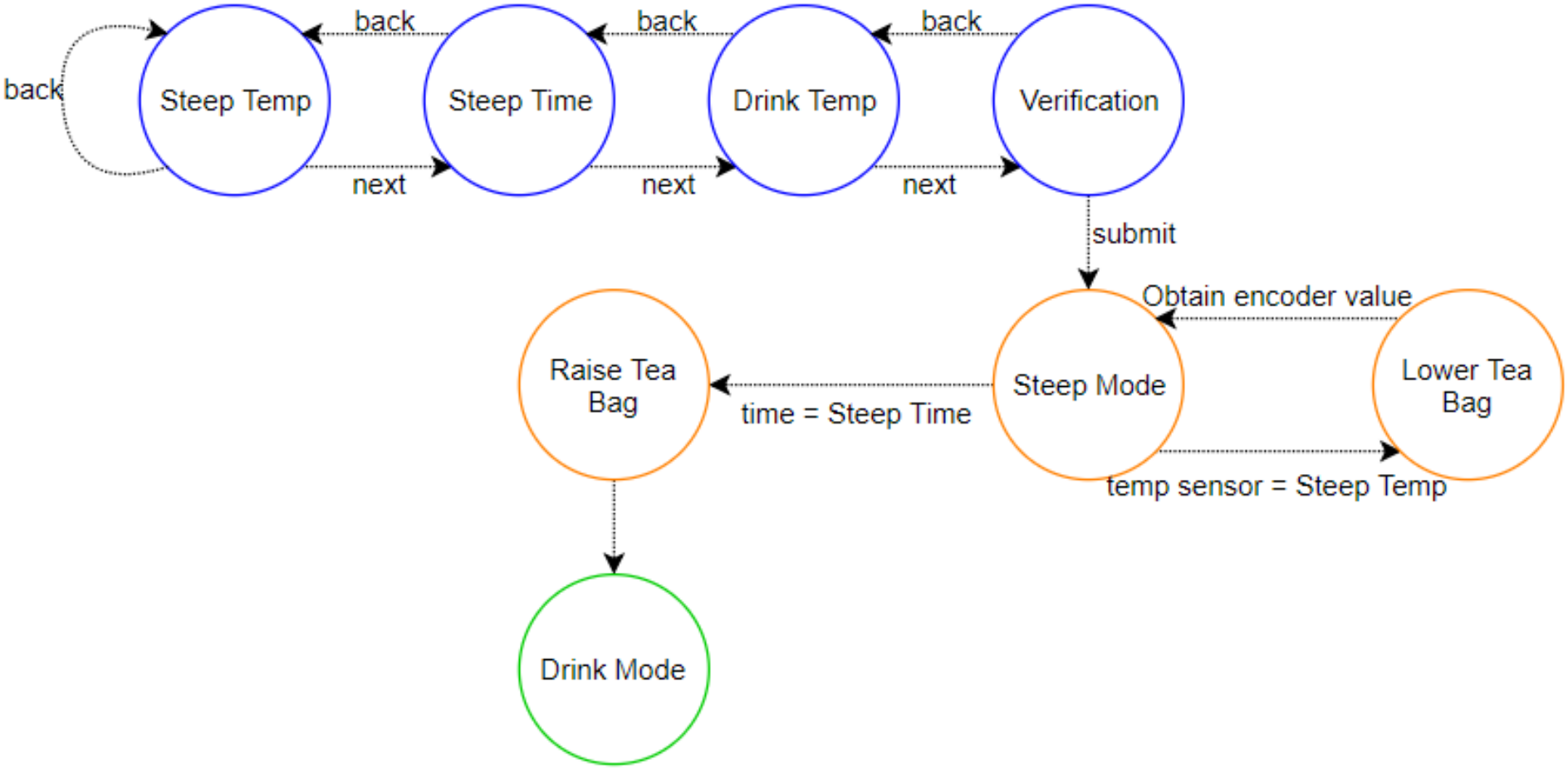
the steeping temp/time and drinking temp. 5. Ensure that the motor and Nichrome wire are never activated at the same time giving priority to the motor. 6. Can control the relay for the nichrome wire based off the inputs from the temperature sensor. 7. Send correct signals to the H-bride and NAND gate chips to properly drive the motor.	7A. Send signals to the enable pin of the H-bride chip and send a high and a low signal to the NAND gate and verify that the motor rotates in two directions.	Y
Nichrome Heating Element		
Requirements	**Verification**	**Verification status (Y or N)**
1. Ensure tolerance analysis is current and warm a 16 oz. cup of water 2˚F in 9 minutes. 2. Maintain a custom drinking temperature for 20 min ranging from 70-110 ˚F with a ±2˚F allowable error. 3. Draw less than or equal to 1.5A of current through itself.	1A. Run the nichrome wire for 9 minutes and take temperature readings with an IR thermometer during that interval of time. 2A. During testing of 1A have water within the range specified and verify that the thermal stability could be maintained within those 20 min. 3A. During Verification of 1A and 2A read the current draw from the power supply.	Y Y Y
IR Temperature Sensor		
Requirements	**Verification**	**Verification status (Y or N)**
1. The sensor must be accurate to within 2˚F at the center point of the sensor.	1A. In conjunction with an IR thermometer the temperature of the water inside the cup will be measured. The two read values will be compared and determine the accuracy of our IR temperature sensor.	Y
Rotary Encoder		
Requirements	**Verification**	**Verification status (Y or N)**
1. Be able to detect a change in rotational	1A. Hook up the rotary encoder to our microcontroller and manually rotate the encoder.	Y

direction and transmit the data to our microcontroller.	1B. Determine if the signal from the rotary encoder gives insight into a direction via positive numbers going clockwise and negative numbers going counter clockwise.	Y
LM7805 5V voltage regulator		
Requirements	**Verification**	**Verification status (Y or N)**
1. Provide 5V ±.5% from a 11.5-12.5 V source. 2. Maintain thermal stability below 125˚C	1A. Measure the output voltage using a voltmeter and ensure the output is within the specified range. 2A. During verification of 1 use and IR thermometer to read the temperature of the device to ensure it is below the threshold.	Y Y
LD117 3.3V Voltage Regulator		
Requirements	**Verification**	**Verification status (Y or N)**
1. Provide 3.3V ±.5% from the output of the LM7805 Voltage regulator. 2. Maintain thermal stability below 125˚C	1A. Measure the output voltage using a voltmeter and ensure the output is within the specified range. 2A. During verification of 1 use and IR thermometer to read the temperature of the device to ensure it is below the threshold.	Y Y
12V Lithium Ion Battery		
Requirements	**Verification**	**Verification status (Y or N)**
1. Maintain thermal stability below 100˚F 2. Provide up to 1.9 A of power for a 20 min period. 3. Provide 12.6 to 10.8V for 9800mAh	1A. Use an IR thermometer to ensure the levels never go above that value during operation. 2A. during verification of nichrome heating element use a current meter and voltage probe connected to oscilloscope to make sure 12v and up to 1.9A are provided. 3A fully charge batter then discharge at 300 mA and record time it takes for the battery to completely drain.	Y Y Y

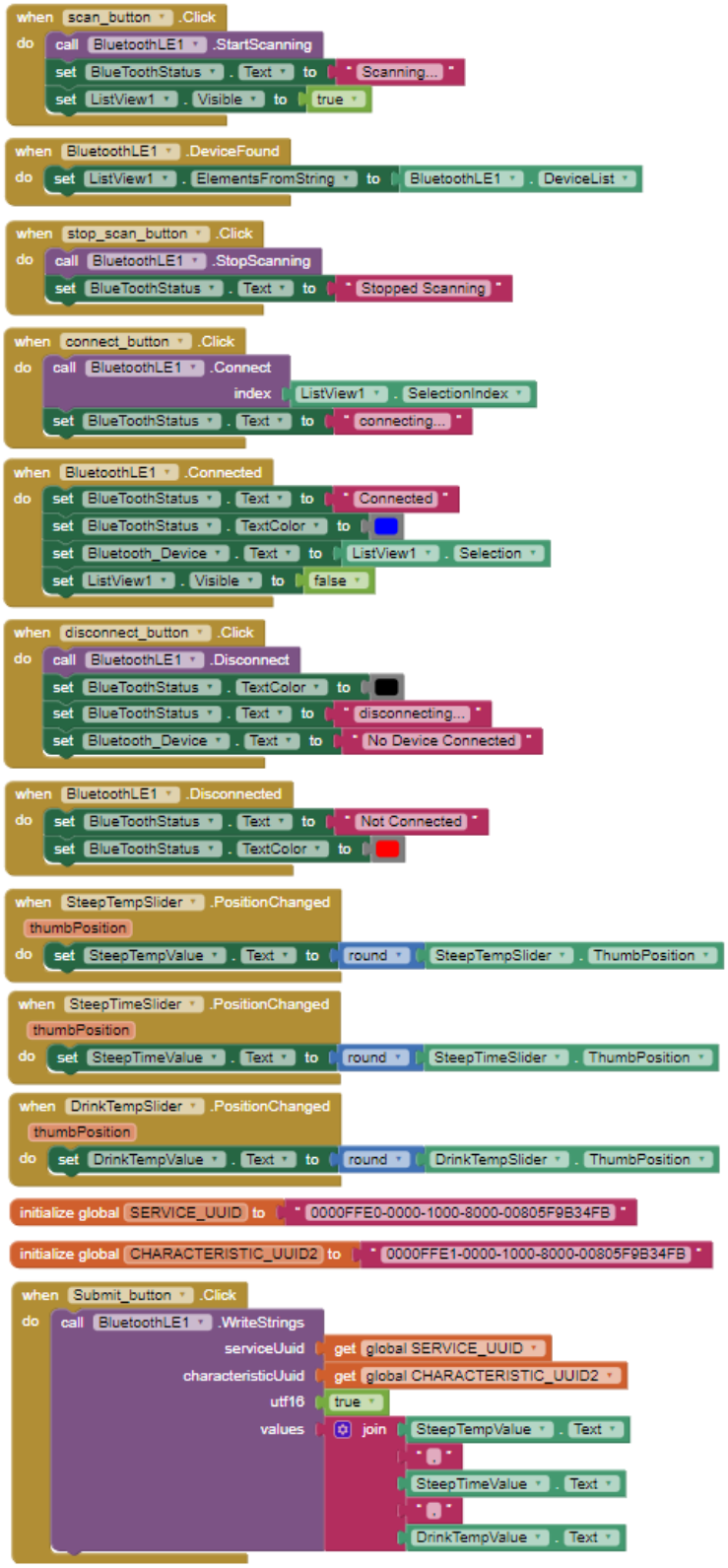
Appendix B Block Diagram



Appendix C State Diagram

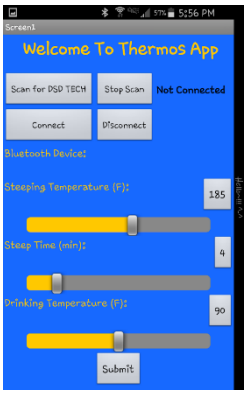


Appendix D Software Code for Phone Application

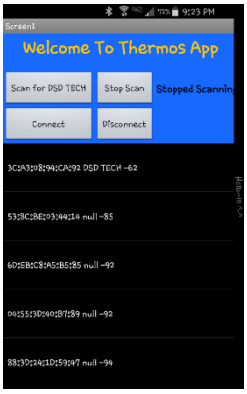


Appendix E Phone App Process

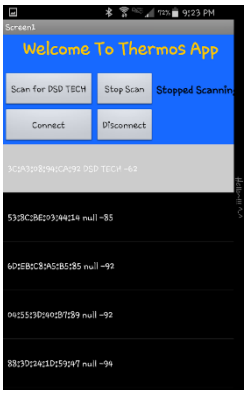
1. Load up the thermos app and ensure the Bluetooth is enabled.



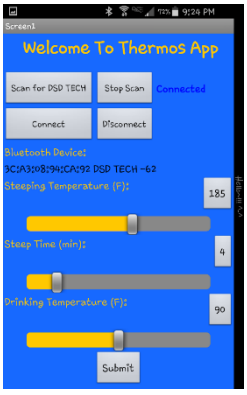
2. Click on the “Scan for DSD TECH” button and click “Stop Scan” when DSD TECH appears on the list.



3. Select DSD TECH from the list and hit the “Connect” button.



4. Verify the correct Bluetooth Device. Choose custom Steeping temperature, Steep Time, and Drinking Temperature and click submit.



Appendix F Software Code for Microcontroller

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#include <SoftwareSerial.h>

#include <stdlib.h>     /* atoi */

#include <Adafruit_MLX90614.h>

#include <SparkFunMLX90614.h> // SparkFunMLX90614 Arduino library


IRTherm therm;


//initialize HM-10

SoftwareSerial mySerial(0, 1);

//RXD: 1

//TXD: 0

//GND: GND

//VCC: 3.3V


// Set the LCD address to 0x27 for a 20 chars and 4 line display

LiquidCrystal_I2C lcd(0x27, 20, 4); //20 columns, 4 lines

//GND:GND

//VCC:5V

//SDA: Analog pin 4

//SCL: Analog pin 5



//IR sensor setup

Adafruit_MLX90614 mlx = Adafruit_MLX90614();

int ProbeTemp;


//set default customizable values

int SteepTemp = 79; //185

int SteepTime = 1;  //4

int DrinkTemp = 79; //90

String number;
```

```
//microcontroller pins
const int backPin = 10;
const int decrPin = 11;
const int incrPin = 12;
const int nextPin = 8;    // 13
const int greenLEDPin = 2;
const int yellowLEDPin = 3;
const int redLEDPin = 4;
const int pinA = 5; // CLK on KY-040
const int pinB = 7; // DT on KY-040
const int motorconPin = 9;
const int motorenPin = 13;  //8
const int nichPin = 6;

//encoder set up
volatile int encoderPosCount = 0;
volatile int aVal;
volatile int pinALast;
int TeaPos = -40;
boolean bCW;

//set default state variables
static unsigned int state;
int backState = 0;
int decrState = 0;
int incrState = 0;
int nextState = 0;
boolean flag = false;

//timer default
unsigned long current_time = 0;
unsigned long start_time = 0;
unsigned long steep_time = 0;
```

```
////////////////////////////////////////////////////////////////////////////////

void setup()
{
  //initialize buttons
  pinMode(backPin,INPUT);
  pinMode(decrPin,INPUT);
  pinMode(incrPin,INPUT);
  pinMode(nextPin,INPUT);
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  pinMode(redLEDPin, OUTPUT);
  pinMode(yellowLEDPin, OUTPUT);
  pinMode(greenLEDPin, OUTPUT);
  pinMode(nichPin, OUTPUT);
  pinMode(motorconPin, OUTPUT);
  digitalWrite(motorconPin,LOW);
  pinMode(motorenPin, OUTPUT);
  digitalWrite(motorenPin,LOW);

  pinALast = digitalRead(pinA);

                // initialize the LCD
                lcd.begin();
                lcd.backlight(); // Turn on the blacklight
  state = 1;

  Serial.begin(9600);
  mySerial.begin(9600);
  mlx.begin();
}

////////////////////////////////////////////////////////////////////////////////
```

```
void loop()
{
//bluetooth module
  String string;
  if (Serial.available())
  {
    Serial.print("blah");
    //mySerial.print(string);
  }
  if (mySerial.available())
  {
    //Serial.println("2");
    string = mySerial.readString();
    int CommaIndex = string.indexOf(',');
    int SecondCommaIndex = string.indexOf(',', CommaIndex+1);
    String s1 = string.substring(0, CommaIndex);
    String s2 = string.substring(CommaIndex+1, SecondCommaIndex);
    String s3 = string.substring(SecondCommaIndex+1);

    Serial.println("String: ");
    Serial.println(string);
    Serial.println(CommaIndex);
    Serial.println(SecondCommaIndex);
    Serial.println("Testing Strings:");
    Serial.println(s1);
    Serial.println(s2);
    Serial.println(s3);

    int n1 = atoi(s1.c_str());
    int n2 = atoi(s2.c_str());
    int n3 = atoi(s3.c_str());

    Serial.println("Testing Integers:");
```

```
  Serial.println(n1);
  Serial.println(n2);
  Serial.println(n3);

  SteepTemp = n1;
  SteepTime = n2;
  DrinkTemp = n3;
  state = 4;
 }

 backState = digitalRead(backPin);
 decrState = digitalRead(decrPin);
 incrState = digitalRead(incrPin);
 nextState = digitalRead(nextPin);

//State Definitions
 switch (state)
 {
 case 1: //steep temp
 //increase or decrease button pressed
  digitalWrite(greenLEDPin, LOW);
  digitalWrite(yellowLEDPin, LOW);
  digitalWrite(redLEDPin, HIGH);

  if(incrState == HIGH && (SteepTemp+106) < 210)
  {
   SteepTemp = SteepTemp + 1;
   delay(200);
  }
  if(decrState == HIGH && (SteepTemp+106) > 150)
  {
   SteepTemp = SteepTemp - 1;
   delay(200);
  }
```

```
  else
  {
    SteepTemp = SteepTemp;
  }

 //print steep temp value on screen
  lcd.setCursor(0,0); //index starts at (0,0):(x,y) at top left of screen
  if(SteepTemp >= 100)
  {
    lcd.print("Steep Temp(F):   " + String(SteepTemp));
  }
  else
  {
    lcd.print("Steep Temp(F):    " + String(SteepTemp));
  }
  lcd.setCursor(0,3);
  lcd.print("<BACK          NEXT>");
 //next or back button pressed
  if(nextState == HIGH)
  {
    state = 2;
    delay(200);
    lcd.clear();
  }
  else
  {
    state = 1;
  }
 break;
/*****************************************************************************************************/
 case 2: //steep time
 //increase or decrease button pressed
  digitalWrite(greenLEDPin, LOW);
  digitalWrite(yellowLEDPin, LOW);
```

```
  digitalWrite(redLEDPin, HIGH);
  if(incrState == HIGH && SteepTime < 10)
  {
    SteepTime = SteepTime + 1;
    delay(200);
  }
  if(decrState == HIGH && SteepTime > 3)
  {
    SteepTime = SteepTime - 1;
    delay(200);
  }
//print steep time on screen
  lcd.setCursor(0,0); //index starts at (0,0):(x,y) at top left of screen
  if(SteepTime >= 10)
  {
    lcd.print("Steep Time(min):  " + String(SteepTime));
  }
  else
  {
    lcd.print("Steep Time(min):   " + String(SteepTime));
  }
  lcd.setCursor(0,3);
  lcd.print("<BACK          NEXT>");
//back or next button pressed
  if(backState == HIGH)
  {
    state = 1;
    delay(200);
    lcd.clear();
  }
  if(nextState == HIGH)
  {
    steep_time = SteepTime * 60000;
    state = 3;
```

```
    delay(200);
    lcd.clear();
   }
 break;
/*************************************************************************************************/
 case 3: //drink temp
 //increase or decrease button pressed
   digitalWrite(greenLEDPin, LOW);
   digitalWrite(yellowLEDPin, LOW);
   digitalWrite(redLEDPin, HIGH);
   if(incrState == HIGH && DrinkTemp < 110)
   {
    DrinkTemp = DrinkTemp + 1;
    delay(200);
   }
   if(decrState == HIGH && DrinkTemp > 70)
   {
    DrinkTemp = DrinkTemp - 1;
    delay(200);
   }
 //print drink temp on screen
   lcd.setCursor(0,0); //index starts at (0,0):(x,y) at top left of screen
   if(DrinkTemp >= 100)
   {
    lcd.print("Drink Temp(F):   " + String(DrinkTemp));
   }
   else
   {
    lcd.print("Drink Temp(F):    " + String(DrinkTemp));
   }
   lcd.setCursor(0,3);
   lcd.print("<BACK          NEXT>");
 //back or next button pressed
   if(backState == HIGH)
```

```
  {
    state = 2;
    delay(200);
    lcd.clear();
  }
  if(nextState == HIGH)
  {
    state = 4;
    delay(200);
    lcd.clear();
  }
 break;
/*************************************************************************************************/
 case 4: //verification screen
 //print out verification screen
  digitalWrite(greenLEDPin, LOW);
  digitalWrite(yellowLEDPin, LOW);
  digitalWrite(redLEDPin, HIGH);
  lcd.setCursor(0,0); //index starts at (0,0):(x,y) at top left of screen
  if(SteepTemp >= 100)
  {
    lcd.print("Steep Temp(F):   " + String(SteepTemp));
  }
  else
  {
    lcd.print("Steep Temp(F):    " + String(SteepTemp));
  }
  lcd.setCursor(0,1); //index starts at (0,0):(x,y) at top left of screen
  if(SteepTime >= 10)
  {
    lcd.print("Steep Time(min):  " + String(SteepTime));
  }
  else
  {
```

```
    lcd.print("Steep Time(min):   " + String(SteepTime));
   }
   lcd.setCursor(0,2); //index starts at (0,0):(x,y) at top left of screen
   if(DrinkTemp >= 100)
   {
    lcd.print("Drink Temp(F):   " + String(DrinkTemp));
   }
   else
   {
    lcd.print("Drink Temp(F):    " + String(DrinkTemp));
   }
   lcd.setCursor(0,3);
   lcd.print("<BACK        SUBMIT>");

  //if back or next button is pressed
   if(backState == HIGH)
   {
    state = 3;
    delay(200);
    lcd.clear();
   }
   if(nextState == HIGH)
   {
    state = 5;
    delay(200);
    lcd.clear();
   }
  break;
/*************************************************************************************************/
  case 5: //steep mode
   digitalWrite(greenLEDPin, LOW);
   digitalWrite(yellowLEDPin, HIGH);
   digitalWrite(redLEDPin, LOW);
```

```
    ProbeTemp = mlx.readObjectTempF();
    //Serial.println(aVal);
    lcd.setCursor(0,0); //index starts at (0,0):(x,y) at top left of screen
    lcd.print("Preparing to Steep..");
    lcd.setCursor(0,1);

    if(SteepTemp >= 100)
    {
      lcd.print("Steep Temp(F):   " + String(SteepTemp));
    }
    else
    {
      lcd.print("Steep Temp(F):    " + String(SteepTemp));
    }
    lcd.setCursor(0,2);
    if(ProbeTemp >= 100)
    {
      lcd.print("Current Temp(F): " + String(ProbeTemp));
    }
    else
    {
      lcd.print("Current Temp(F):  " + String(ProbeTemp));
    }

//Reached Desired Steep Temperature
    if(ProbeTemp == SteepTemp && digitalRead(nichPin)== LOW && flag == false)//steeptemp)
    {
      //Serial.println(ProbeTemp);
      //print LCD screen: "steeping..."
      lcd.setCursor(0,3); //index starts at (0,0):(x,y) at top left of screen
      lcd.print("Steeping...         ");
```

```
      state = 7;
    }

//Colder than Desired Steep Temperature
    else if(ProbeTemp < SteepTemp && digitalRead(motorenPin) == LOW)//SteepTemp)//need to add limit value - how much battery is consumed to raise temp and be worth it
    {
      //Serial.println(ProbeTemp);
      digitalWrite(nichPin,HIGH); //turn on power relay
    }

//Hotter than Desired Steep Temperature
    else //if((ProbeTemp) > SteepTemp) //need to add limit value
    {
      //Serial.println(ProbeTemp);
      digitalWrite(nichPin, LOW); //turn off power relay
    }

//Reaches Steep Time
   current_time= millis();
   current_time= current_time - start_time;
   Serial.println(current_time/1000);

   if(current_time >= steep_time){
    state = 8;
   }
  break;
/*****************************************************************************************************/
  case 6: //Drink Mode
    digitalWrite(greenLEDPin, HIGH);
    digitalWrite(yellowLEDPin, LOW);
    digitalWrite(redLEDPin, LOW);
    ProbeTemp = mlx.readObjectTempF();
    lcd.setCursor(0,0); //index starts at (0,0):(x,y) at top left of screen
```

```
lcd.print("Prepare Drink Temp..");
lcd.setCursor(0,1);

if(SteepTemp >= 100)
{
  lcd.print("Drink Temp(F):   " + String(DrinkTemp));
}
else
{
  lcd.print("Drink Temp(F):    " + String(DrinkTemp));
}
lcd.setCursor(0,2);
if(ProbeTemp >= 100)
{
  lcd.print("Current Temp(F): " + String(ProbeTemp));
}
else
{
  lcd.print("Current Temp(F):  " + String(ProbeTemp));
}

if((ProbeTemp) == DrinkTemp)
{
  //print LCD screen: "Ready!"
  lcd.clear();
  lcd.setCursor(0,3); //index starts at (0,0):(x,y) at top left of screen
  lcd.print("      Ready!!!      ");
  //print LCD screen: "Next button for new cup"
}
else if((ProbeTemp) < DrinkTemp) //need to add limit value
{
  digitalWrite(nichPin,HIGH); //turn on power relay
}
else
```

```
  {
    digitalWrite(nichPin, LOW); //turn off power relay
  }
 break;
/*************************************************************************************************/
 case 7:
  //motor on to lower teabag

  aVal = digitalRead(pinA);
  if (aVal != pinALast){ // Means the knob is rotating
   // if the knob is rotating, we need to determine direction
   // We do that by reading pin B.
   if (digitalRead(pinB) != aVal) {  // Means pin A Changed first - We're Rotating Clockwise
     encoderPosCount++;
     bCW = true;
   } else {// Otherwise B changed first and we're moving CCW
     encoderPosCount--;
     bCW = false;
   }
   Serial.print ("Rotated: ");
   if (bCW){
   Serial.println ("clockwise");
   }else{
   Serial.println("counterclockwise");
   }
   Serial.print("Encoder Position: ");
   Serial.println(encoderPosCount);
  }
  pinALast = aVal;

  if(encoderPosCount > TeaPos)
  {
    digitalWrite(nichPin, LOW);
    digitalWrite(motorenPin, HIGH);
```

```
    digitalWrite(motorconPin, HIGH);
   }
   else
   {
    digitalWrite(motorenPin, LOW);
    digitalWrite(motorconPin, LOW);
    flag = true;
    start_time = millis();//start timer
    state = 5;
   }
  break;
/*****************************************************************************************************/
  case 8:
  //motor on to raise tea bag

  aVal = digitalRead(pinA);
  if (aVal != pinALast){ // Means the knob is rotating
   // if the knob is rotating, we need to determine direction
   // We do that by reading pin B.
   if (digitalRead(pinB) != aVal) {  // Means pin A Changed first - We're Rotating Clockwise
    encoderPosCount++;
    bCW = true;
   } else {// Otherwise B changed first and we're moving CCW
    encoderPosCount--;
    bCW = false;
   }
   Serial.print ("Rotated: ");
   if (bCW){
   Serial.println ("clockwise");
   }else{
   Serial.println("counterclockwise");
   }
   Serial.print("Encoder Position: ");
   Serial.println(encoderPosCount);
```

```
  }
  pinALast = aVal;

   if(encoderPosCount < 0)// && digitalRead(nichPin) == LOW)
   {
     digitalWrite(nichPin, LOW);
     digitalWrite(motorenPin, HIGH);
     digitalWrite(motorconPin, LOW);
   }
   else
   {
     digitalWrite(motorenPin, LOW);
     digitalWrite(motorconPin, LOW);
     lcd.clear();
     state = 6;
   }
  break;
 }
}
```