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	1A. during active state, measure voltage drop across 1k resistor.	Y

8.5mA during active state. 2. Maintain thermal stability below 125˚C 3. Communicates in a range of 5-10m.	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
"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 submit values via Bluetooth for the steep temp/time, and drinking temp.	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. 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.	Y Y
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.	Y Y

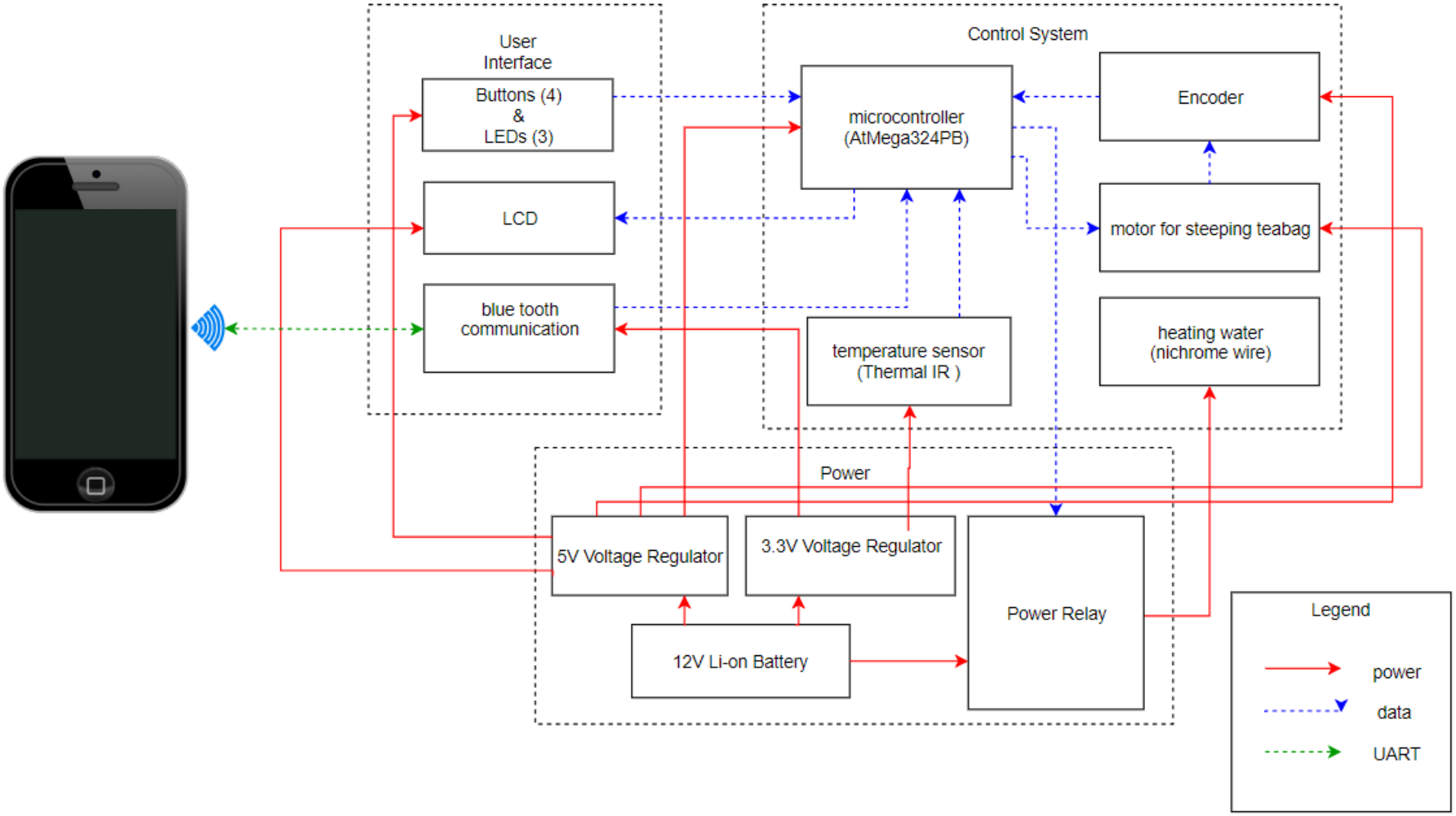
	3A. During normal operation we will monitor the current draw from the power source powering the chip and ensure we do not exceed .3 mA.	Y
Control Software		
Requirements	**Verification**	**Verification status (Y or N)**
1. Read temperature sensor data and be able to compare it to values obtained from 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 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	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. 7A. Send signals to the enable pin of the H-bride chip and send a high and a low signal to the NAND	Y Y Y Y Y

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.	gate and verify that the motor rotates in two directions.	Y 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		

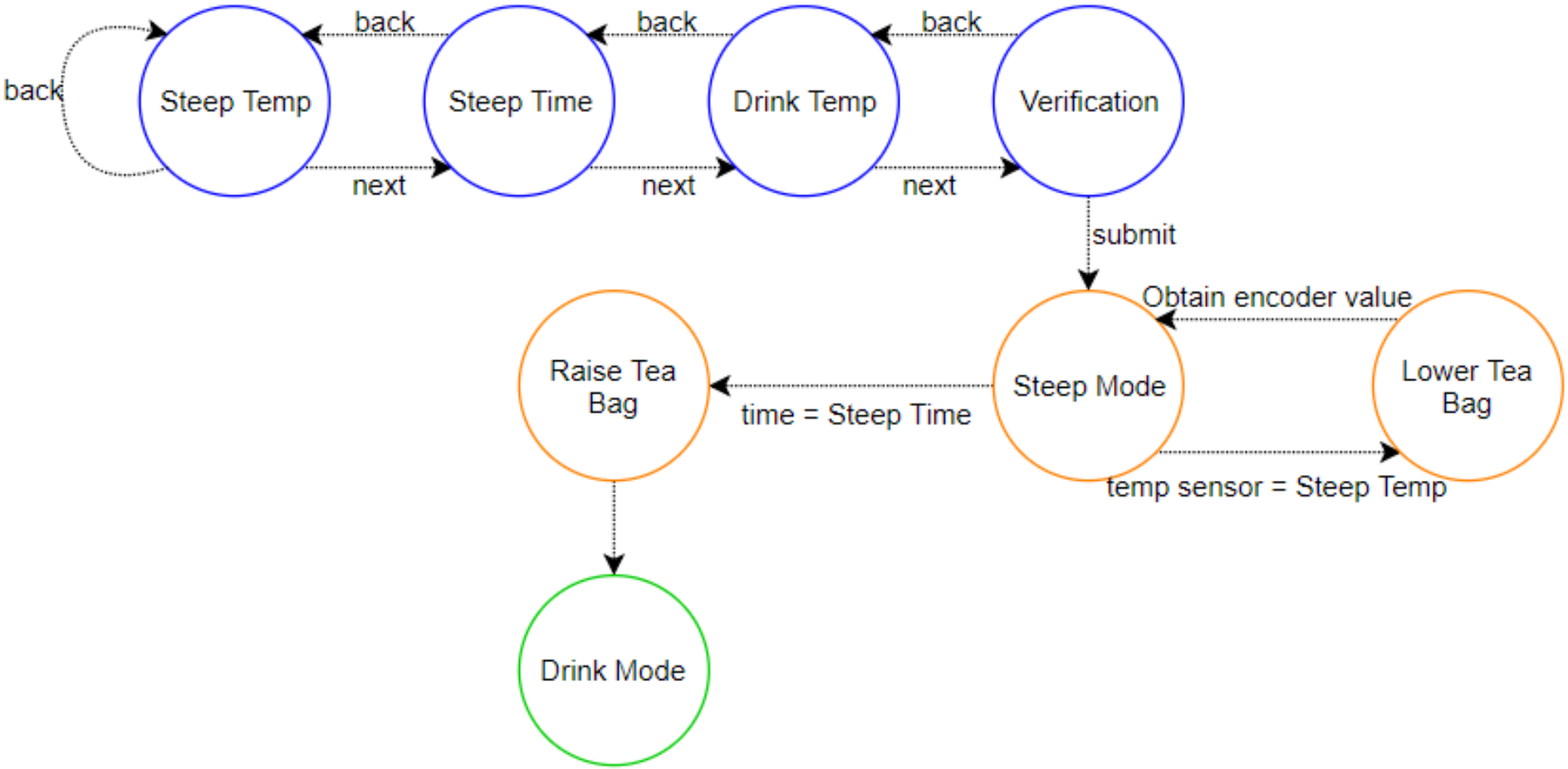
Requirements	Verification	Verification status (Y or N)
1. Be able to detect a change in rotational direction and transmit the data to our microcontroller.	1A. Hook up the rotary encoder to our microcontroller and manually rotate the encoder. 1B. Determine if the signal from the rotary encoder gives insight into a direction via positive numbers going clockwise and negative numbers going counter clockwise.	Y 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

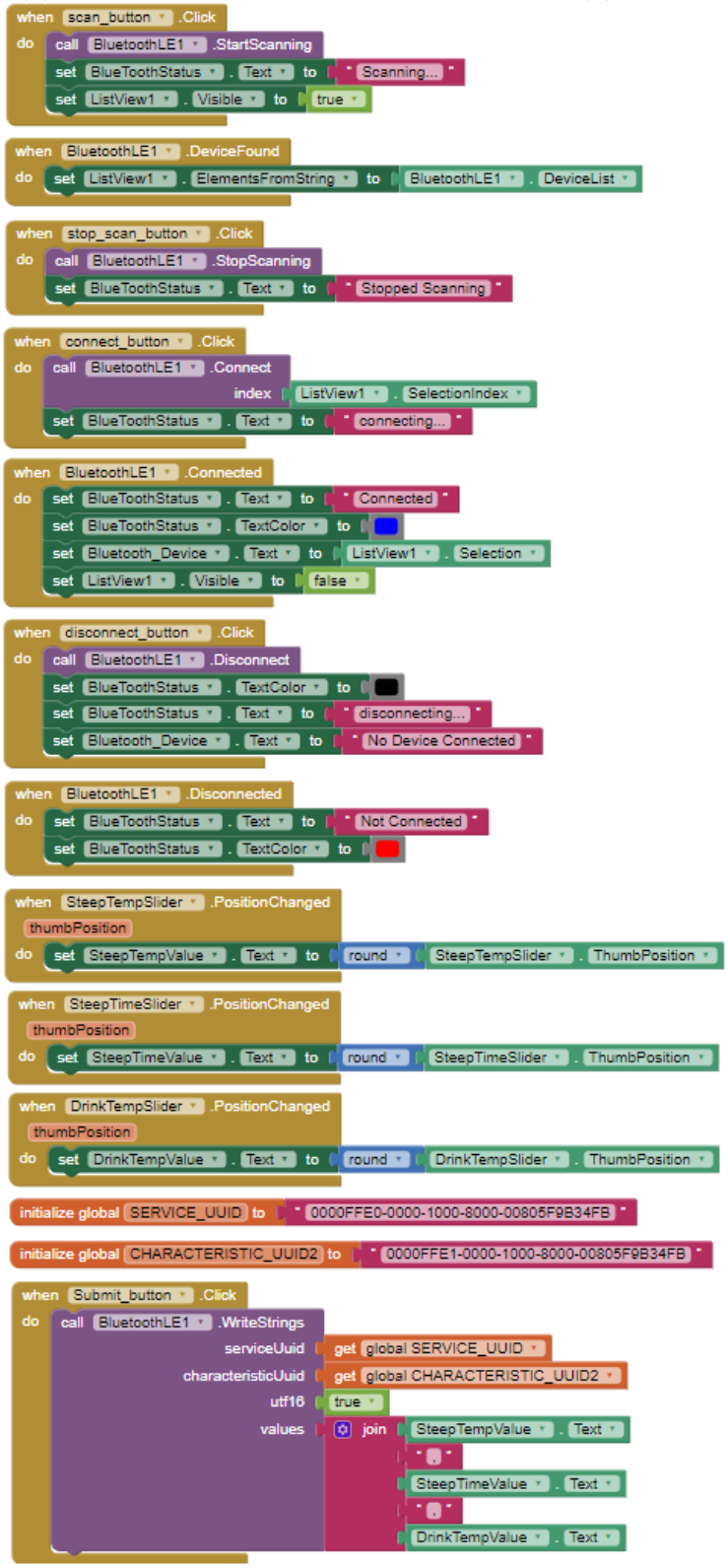
Appendix B Block Diagram



Appendix C State Diagram

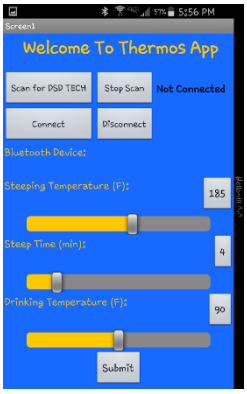


Appendix D Software Code for Phone Application

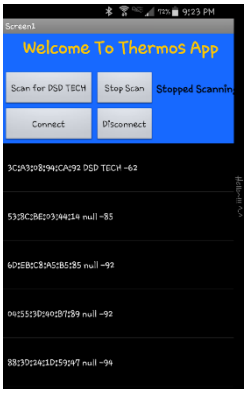


Appendix E Phone App Process

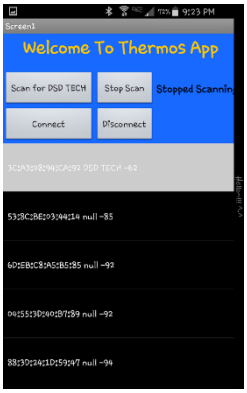
1. Load up the thermos app and ensure the Bluetooth is enabled.



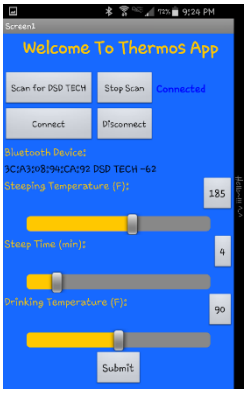
2. Click on the “Scan for DSD TECH” button and click “Stop Scan” when DSD TECH appears on the list.



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;
 }
}
```