Pittsburg State University, Electronics Engineering Technology

EET 640, Application Design Problems, 2016 Spring

Smart Brew

_____

Andres Chu Wu

_____

Zhong Yan Shi

Due Date: 05/02/2016

For Use of Instructor:

Date Received: ___/___/___ Grade Assigned: _____

Remarks:

Executive Summary

This report presents the work of team Solution in Engineering Technology in the Application Design Problem course, EET 640, at Pittsburg State University in the spring of 2016. Solution in Engineering Technology, formed by two on campus students in the Electronics Engineering Technology program, were assigned the task of designing an automated herbal medicine brewer by the course instructors, Prof. Clark Shaver and Prof. Jim Lookadoo.

The team first started conducting a market research on products very similar in characteristics and reviews to get a handle on the problems and solutions that exist currently in the herbal medicine brewing market. With the information gathered from our initial market research, our team determined that similar products available do not support or offer a very basic form of automation control. From there, we focused on the problem stated and began to generate a list of specification to facilitate with our initial prototype of Smart Brew. The whole project was centered on these specification; circuit design, programming, calculations, and manufacturing design.

Moving forward from the initial market research, the team began developing the different functions that our prototype would need to carry out the specifications we had generated. These functions include the ability to transmit commands from a smartphone application, read and register temperature values, output actions according the registered values, and transmit back to the smartphone application. The large number of initial concepts was then trimmed to one solution by evaluating them against our specification and also respect our team's mission statement of using open source systems. The selected solution was to create an automated control system with the aid of the Arduino environment and be implemented into our prototype.

The team determined that designing a technology that would be added to the herbal medicine field would allow for a product to stay relevant regardless of how many people do believe in holistic medicine, since there are many applications were our product may be inserted. We began working on the actual build of Smart Brew while nailing down other aspects of the project that includes: bill of materials, project schedule, and a market report. After the prototype was finished, our team then identified what further work can be done to bring the product to market. Finally, our team reflected on the prototype and process as a whole including what was done well and what could have been done more efficiently.

Contents

List of Figures

List of Tables

1 – Project Concept

1.1 – Abstract

The herbal medicine field is an ancient art that seeks to cure various illnesses with natural ingredients. Nowadays, we can observe that there's a strong demand for natural and organic products implying that, there's also an increasing need for appliances that properly prepares these special ingredients. It's known that different roots and herbs have different levels of heat sensitivity, so the preparation of an herbal tea would require time, skill and knowledge from the user in order to extract all the healing properties from these ingredients. Smart Brew is specially designed to solve these hassles; it is an automated system that will aid in the preparation of these ingredients by controlling the checking the development process through an app, therefore optimizing the quality of the drinks and minimizing waste of the ingredients. It's a product that will modernize the traditional market of herbal medicine and facilitate the lives of many consumers interested in health products.

1.2 – Introduction

Smart Brew is a completely autonomous herbal medicine brewer. This device contains four major components. The first is a heat source. The second component is the compartments in which ingredients will be stored. The third component consists of sensors in which data will be collected. And the last component is the communication between the user and machine through a smartphone application.

1.3 – Project Idea Development

At the start of the course, the team was task to think of five project concepts for this course. These five ideas include the following:

1. Smart Herbal Medicine Brewer
2. Automated Indoor Irrigation System
3. Smart Door Lock
4. Smart Water Bottle Cap
5. Ventilation Project offered from the course instructors

After presenting these ideas to the course instructors, they gave the team verbal permission to pursue the Smart Herbal Medicine Brewer project. The reason the team listed the Smart Herbal Medicine Brewer at the top at the list is due to the cultural knowledge behind herbal medicine and also the lack of this product in the current market.

1.4 – Team Organization

The first team meeting was held at the Leonard H. Axe Library at Pittsburg State University. During this meeting, several topics were discussed in order to help the team stay organized:

1. Establishing means of communication
2. Create a team charter
3. Setting project schedules and gate presentation deadlines
4. Create an account on Dropbox for the team in which each member would collectively store documentation.
5. Division of Labor and Cost
6. Team name and logo

Each member of the team was responsible for communicating with each other to ensure the each member was working as one cohesive unit. The team met according to their class schedule and availability in order to meet the deadlines for each Gate as outlined in the team charter shown in Appendix A. Also, the team settled on a team name, Solution in Engineering Technology, and a design for the logo depicted below in Figure 1.

Figure 1: Solution in Engineering Technology Logo

2 - Conceptual Design

2.1 – Primary Specification

1. Product will support 4 brewing regions with distinct temperature profile to user's preference:
    a. Brewing region for Physical Illness with a brewing temperature profile setting as given in Appendix B (± 2° C).
    b. Brewing region for Common Illness with a brewing temperature profile setting as given in Appendix B (± 2° C).
    c. Brewing region for Supplement with a brewing temperature profile setting as given in Appendix B (± 2° C).
    d. Brewing region for Custom with a brewing temperature profile setting as given in Appendix B (± 2° C).
        i. Ex:
            1. Main ingredients need 1 hour of pre immersion.
            2. Boil for 30 minutes.
            3. Simmer for 50 minutes.
            4. Dispense sensitive ingredients.
            5. Keep warm for 10 minutes.
2. Product will support 2 separate storage compartment:
    a. First compartment will store up to 4 main ingredient bags.
        i. Fist compartment will be dispensed at the start of the brewing process for pre immersion.
    b. Second compartment will store 4 sensitive ingredient bags.
        i. Second compartment will be dispensed during the remaining 10 minutes of the total brewing time.
3. Product will support a water pump to fill medical pot:
    a. Water level is filled up to 85% of the total volume to begin brewing process (± 5%).
    b. Water level cannot drop more than 35% of the total volume during brewing process (± 5%).

4. Product will support an Android based phone Application:
    a. Application will serve as a mean of communication between product and user.
        i. User can set one of the 4 brewing regions (Physical Illness, Common Illness, Supplement, and Custom).
        ii. In Custom brewing region, user can manually set:
            1. Time settings
            2. Temperature settings
            3. Override brewing process

2.2 – Secondary Specification

1. Product will support a water pump to fill medical pot for a strong or bland taste:
    a. For a stronger taste, water level is filled up to 75% of total volume (± 5%).
        i. Ex: 1.5L in a 2L medical pot.
    b. For a bland taste, water level is filled up to 85% of total volume (± 5%).
        i. Ex: 1.7L in a 2L medical pot.
2. Product will support a touch screen interface:
    a. User can set one of the 4 brewing regions (Physical Illness, Common Illness, Supplement, and Custom).
    b. In Custom brewing region, user can manually set:
        i. Time settings
        ii. Temperature settings
        iii. Override brewing process

2.3 – List of Applicable Standards

i. IEEE 802.15.4 Networking Protocol for Fast Point-to-Multipoint or Peer-to-Peer Networking
ii. IEEE 802.15.1 WPAN/ Bluetooth
iii. ASTM C1359 -13 Standard Test Method for Monotonic Tensile Strength Testing of Continuous Fiber-Reinforced Advanced Ceramics
iv. ASTM D10442-12 Standard Test Method for Linear Dimensional Changes of Plastics Caused by Exposure to Heat and Moisture
v. ASTM D50 -98 Standard Test Method for Water Absorption of Plastics

vi. ASTM D3045 -92 Standard Practice of Heat Aging of Plastics Without Load

vii. ASTM D4092 -07 (2013) Standard Terminology for Plastic: Dynamic Mechanical Properties

Note: ASTM – American Society for Testing and Materials

3 – Preliminary Design

3.1 – Conceptual Design

The design can be divided into three major functions: inputs, process, and output. Each significant function will contain several key components that will be combined and developed into a functioning prototype. First, in order to handle processing and communication between inputs and outputs, the use of microcontrollers will be integrated in the system. With the assistance of microcontrollers, it will make the project more manageable in the time assigned.

Figure 2: Basic Function Block Diagram

The inputs for the functioning prototype will be made up of three components: temperature sensor (DS18B20 Thermometer), wireless module (HC-06 Bluetooth Module), and switches. Every processing done by Smart Brew will be handled by an embedded microcontroller, specifically the Atmel Atmega 2560. This specific chip is used on the Arduino Mega development boards for its reliability and flexibility to handle basic functions. Also another key reason Smart Brew uses the Atmega 2560 is because of its reasonable price. Price is a very important factor in this prototype: if the overall costs can be reduced, so can be the price of the final product.

For the outputs section, this prototype will have four components. The first component is the LCD screen. The use of this screen will provide visual information on several functions Smart Brew. The second component is the heating element that will consist of a ceramic heater plate. For the third component, a water pump will be installed to fill the medical pot at the desired setting. For the final component, a small motor for the opening and closing of the ingredient compartments. Appendix C depicts the input/output block diagram for the Atmega 2560.

3.1.1 – Design Process

After determining the inputs and outputs the team began developing the system for Smart Brew. A simple flowchart as seen in Appendix C demonstrates how the Smart Brew will perform. The first step is to establish the wireless connection between the smartphone application and the prototype. When connection is established, the following step is to inform Smart Brew of the brewing profile the user has selected. When selected, Smart Brew will begin the brewing process and dispense the Main Ingredients. As soon as the brewing process concludes, Smart Brew will keep the concoction at an appropriate temperature in case the user is not around to consume it immediately.

3.1.1a – Wireless Connection

A smartphone application was designed in Android Studio in order to interact with Smart Brew. When the application is initialized, the first step is to ask the user to establish the Bluetooth connection with the prototype. When this connection is established, the application will allow the user to selected their desired profile. The flowchart for the Android application can be seen in Appendix C.

3.1.1b – Brewing Profile

Each brewing profile has its unique temperature profile that has been programed as seen in Appendix B. For the Physical Illness, Common Illness, and Supplement temperature profiles are profiles that are programmed into Smart Brew and no possible customization will be allowed. The Custom Profile is the only brewing profile that will allow the user set its personal settings; setting temperature (in Celsius) and brewing time length.

3.1.1c – Main Ingredient

After choosing the desired brewing profile, Smart Brew will begin filling the ceramic pot to 85% of its total volume with water. When the water level has reached, Smart Brew will begin to dispense the main medicine bag and start the pre-soak process. During the pre-soak process, the medicine bag sits inside the water-filled ceramic pot for at least an hour. Note that the heating element will not start until the one-hour mark has been reached.

3.1.1d – Brewing Process

When the main ingredient has been pre-soaked for an hour, Smart Brew will initialize the brewing process. As describe earlier in 3.1.1b, each brewing profile has a unique temperature profile. These temperature profiles are considered one of the key functions of Smart Brew. Without these profiles, the extraction of these ingredients won't be maximized. In the brewing process, Smart Brew will constantly check if the water level has not drop to a critical level at 35% of the total volume of the ceramic pot and the temperature. Also it will be checking to see if the user has paused the brewing process.

3.1.1e – Sensitive Ingredient

Apart from the main ingredients, that require extensive time to brew, there are also ingredients that do not require. These ingredients are described as 'sensitive ingredients'. Smart Brew will only dispense the sensitive ingredient medicine bag if the user had selected prior to the brewing process. If the user did select the presence of sensitive ingredients, Smart Brew will dispense during the remaining ten minutes of the total brew time.

3.1.1f – Final Brewing Process

When the brewing process has finalized, Smart Brew will begin to set the temperature at a 'keep warm' setting. This setting will set the heating element to maintain at 50° Celsius ($\pm$ 2°). The team members think that at this temperature it is ideal for consumption for the user. Smart Brew will also be constantly checking for drop in temperature inside the ceramic pot and turn the heating element when necessary.

3.2- Primary Specification Test

Specification: Test 1

Test Name: Smartphone Interface Test

Objective: Does the smartphone application have the necessary button interfaces and also test Bluetooth communication between application and board.

How is the test performed: Connect a wireless module to an array of LEDs. Connect the smartphone application to the wireless module to observe if the LED's turn on or off.

What data will be retrieved from test:

	Is Button Present?	Does Button Work?
Physical Profile	Y / N	Y / N
Common Profile	Y / N	Y / N
Supplement Profile	Y / N	Y / N
Custom Profile	Y / N	Y / N
Pause	Y / N	Y / N
Resume	Y / N	Y / N

Acceptance Criteria: All button must be present on the application and be able to turn on their assigned LED in order for the test to pass. Otherwise the test has failed.

Specification: Test 2

Test Name: Water Level Test

What is to be learned from this test: Can the prototype detect water level inside the herbal pot.

How is the test performed: On the side of the pot, two thermistor will be place at specific position, this will allow the automated herbal pot know when the water reached a certain amount.

What data will be retrieved from test?

	Initial Resistance	Reached Resistance
0.7 L (35%)		
1.7 L (85%)		

Acceptance Criteria: Comparing the initial temperature and the detected temperature. If the values did change, the test is successful. If the temperatures values did not change, the test has failed.

Specification: Test 3

Test Name: Brewing Profile Test #1

What is to be learned from this test: Can the Automated Herbal Pot perform the Physical temperature profile?

How is the test performed: A program will be written in the Arduino IDE that will mimic the physical illness temperature profile. On the development board, two LED (green and red) will be connected to indicate different temperatures. For example, if the program has reached a "temperature" of 100°C, the red LED should turn on and stay on for a determined time. When time is up, the LED should turn off.

What data will be retrieved from test?

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	OFF	ON	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	OFF	ON	OFF
120	OFF	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	OFF	ON

Table 1: Pre-determined Pattern for Physical Temperature Profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Specification: Test 4

Test Name: Brewing Profile Test #2

What is to be learned from this test: Can the Automated Herbal Pot perform the Common temperature profile?

How is the test performed: A program will be written in the Arduino IDE that will mimic the physical illness temperature profile. On the development board, two LED (green and red) will be connected to indicate different temperatures. For example, if the program has reached a "temperature" of 100°C, the red LED should turn on and stay on for a determined time. When time is up, the LED should turn off.

What data will be retrieved from test?

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	OFF	ON	OFF
90	OFF	ON	OFF
100	OFF	OFF	OFF
110	ON	OFF	OFF
120	ON	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	OFF	ON

Table 2: Pre-determined Pattern for Common Temperature Profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Specification: Test 5

Test Name: Brewing Profile Test #3

What is to be learned from this test: Can the Automated Herbal Pot perform the Supplement temperature profile?

How is the test performed: A program will be written in the Arduino IDE that will mimic the physical illness temperature profile. On the development board, two LED (green and red) will be connected to indicate different temperatures. For example, if the program has reached a "temperature" of 100°C, the red LED should turn on and stay on for a determined time. When time is up, the LED should turn off.

What data will be retrieved from test?

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	ON	OFF	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	OFF	ON	OFF
120	OFF	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	ON	OFF
180	OFF	ON	OFF
190	OFF	ON	OFF
200	OFF	ON	OFF
210	OFF	OFF	ON

Table 3: Pre-determined Pattern for Supplement Temperature Profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Specification: Test 6

Test Name: Brewing Profile Test #4

What is to be learned from this test: Can the Automated Herbal Pot perform the Custom temperature profile?

How is the test performed: A program will be written in the Arduino IDE that will mimic the physical illness temperature profile. On the development board, two LED (green and red) will be connected to indicate different temperatures. For example, if the program has reached a "temperature" of 100°C, the red LED should turn on and stay on for a determined time. When time is up, the LED should turn off.

What data will be retrieved from test?

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	ON	OFF	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	OFF	ON	OFF
120	OFF	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	OFF	ON

Table 4: Pre-determined Pattern for Custom Temperature Profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Specification: Test 7

Test Name: Heating Element Test

What is to be learned from this test: Can the prototype be able to turn on/off the heating element at a determined temperature.

How is the test performed: A simple ADC program in Arduino will be written to check a DS18B20 thermometer. The sensor will be measuring a heating element. When temperature has reached 25°C, an LED will turn on. As temperature rises to 100°C, the LED will turn off.

What data will be retrieved from test:

Temperature	Is LED on or off?	Resistance Value
25°C		
100°C		

Acceptance Criteria: The test is successful if the LED is turned on at 25 °C and also if the LED is turned off at 100 °C.

3.2 – Secondary Specification Test

Specification: Test 8

Test Name: Water Level Taste Test

What is to be learned from this test: Can the prototype detect different water level inside the herbal pot?

How is the test performed: On the side of the pot, two thermistor will be place at specific position. This will allow the automated herbal pot know when the water reached a certain amount.

What data will be retrieved from test?

	Init. Resistance	Reach Resistance
1.5 L (75%)		
1.7 L (85%)		

Acceptance Criteria: Comparing the initial resistance and reached resistance. If the values did change, the test is successful. If the resistance value did not change, the test has failed.

Specification: Test 9

Test Name: Touchscreen Brew Profile Selection Test

What is to be learned from this test: Does the touch screen have the necessary button interface and do each of these buttons properly function?

How is the test performed: Connect a wireless module to an array of LEDs. Connect the smartphone application to the wireless module to observe if the LED's turn on or off.

What data will be retrieved from test?

	Is Button Present?	Does Button Work?
Physical Profile	Y / N	Y / N
Common Profile	Y / N	Y / N
Supplement Profile	Y / N	Y / N
Custom Profile	Y / N	Y / N
Time	Y / N	Y / N
Temperature	Y / N	Y / N
Home	Y / N	Y / N
Pause	Y / N	Y / N
Resume	Y / N	Y / N

Acceptance Criteria: All button must be present on the application and turn on the LEDs in order for the test to pass. Otherwise the test has failed.

4 - Detailed Design

4.1 – Engineering Simulation

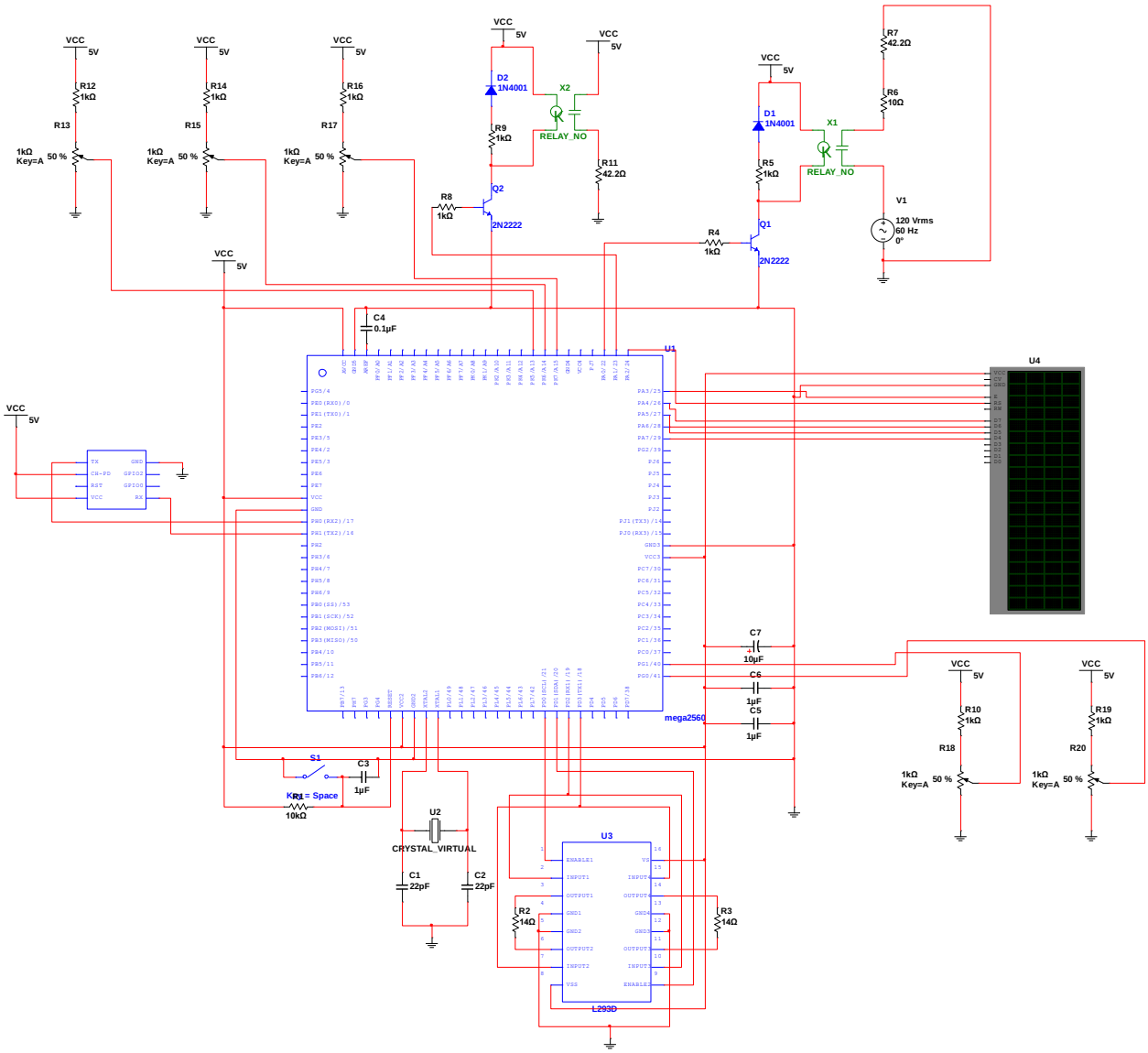


Figure 3: Smart Brew Circuit Design.

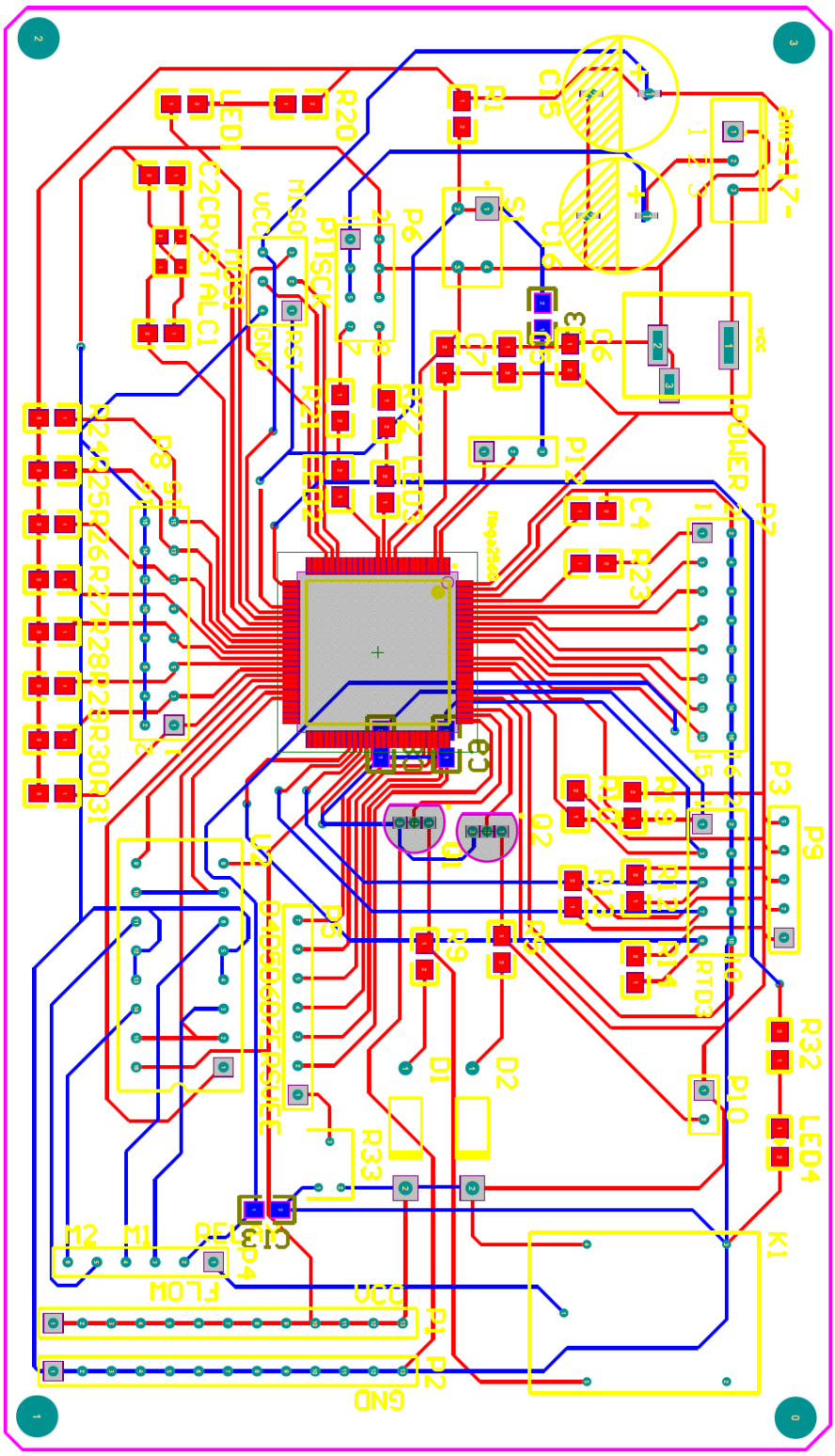


Figure 4: Smart Brew PCB Schematic with Altium Designer

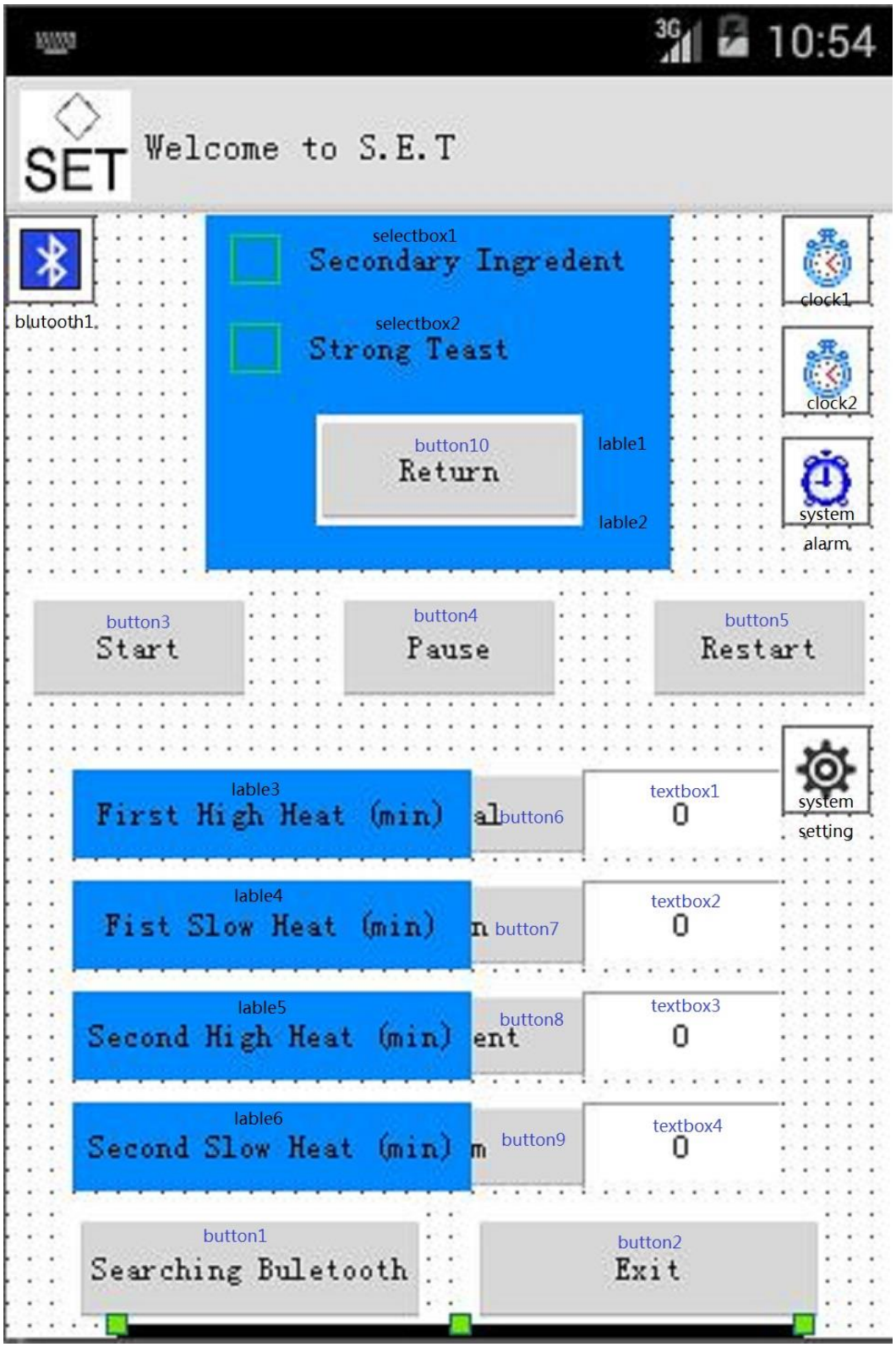


Figure 5: Smart Brew Android Application.

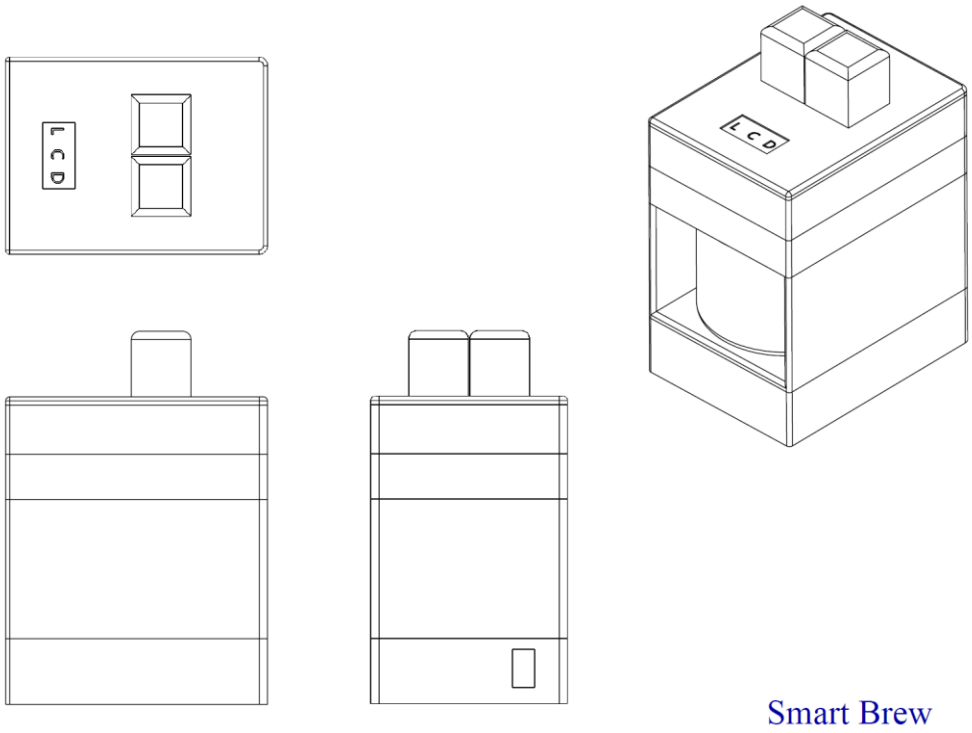


Figure 6: Smart Brew Solidwork Design

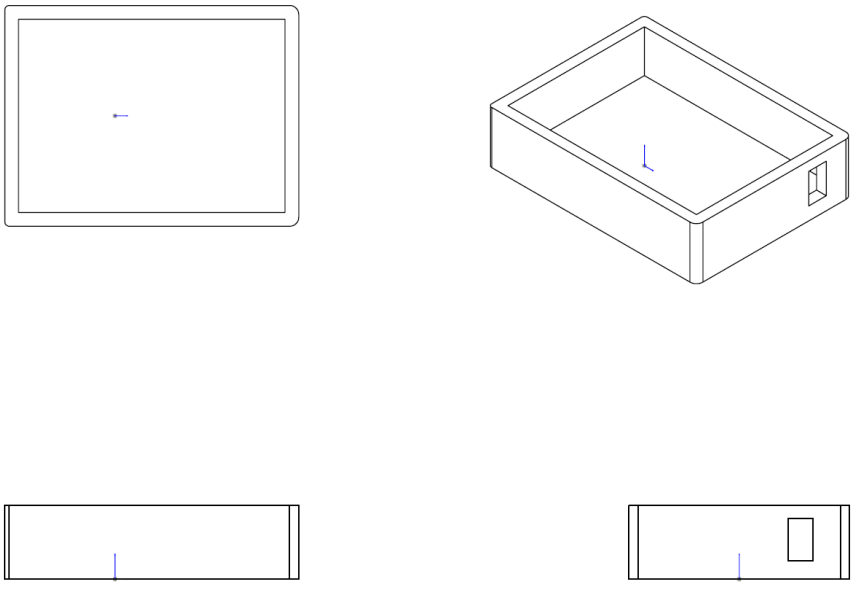


Figure 7: Smart Brew Base 1 Design in Solidworks.

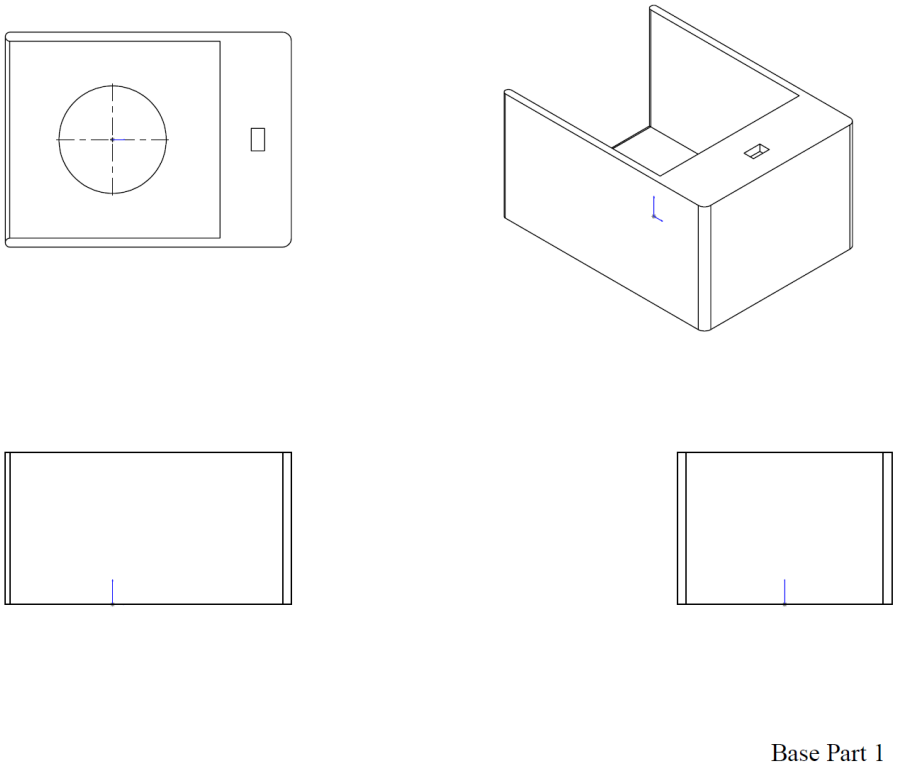


Figure 8: Smart Brew Base 2 Solidwork Design.

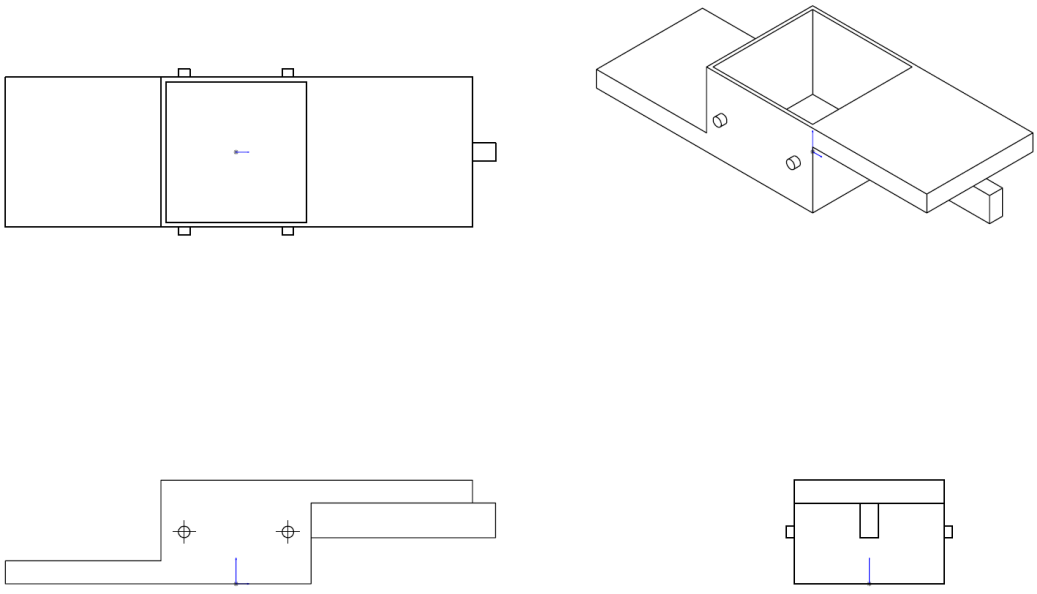


Figure 9: Smart Brew Ingredient Cart Solidwork Design.

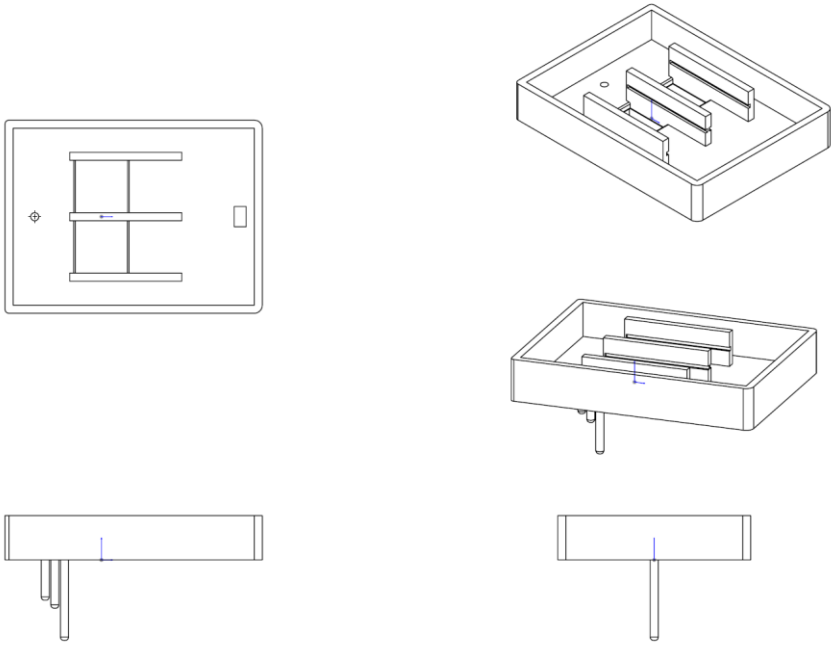


Figure 10: Smart Brew Top Encasing 1 Solidwork Design.

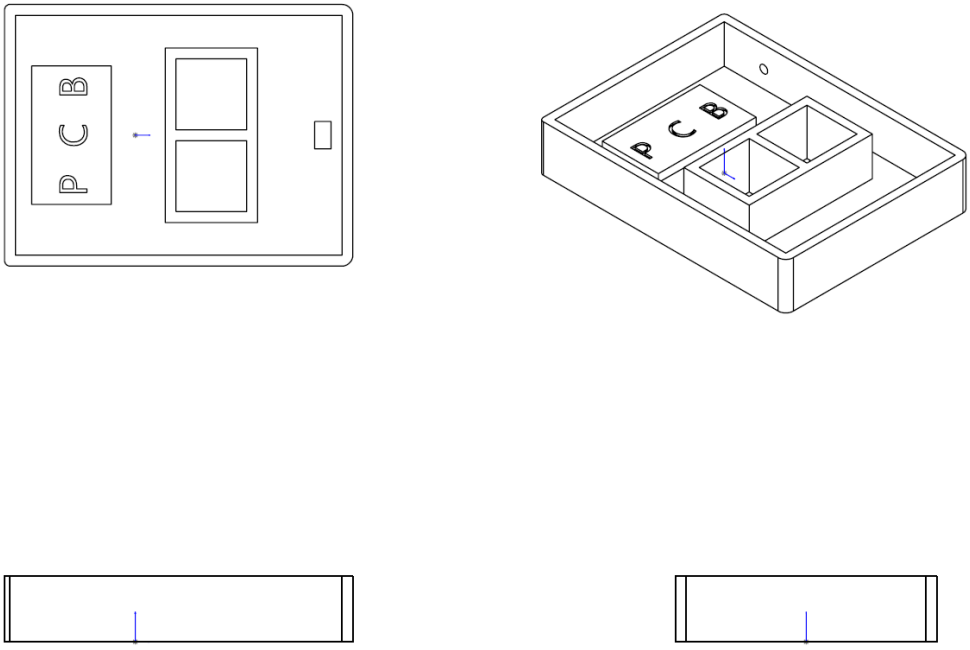


Figure 11: Smart Brew Top Encasing 2 Solidwork Design.

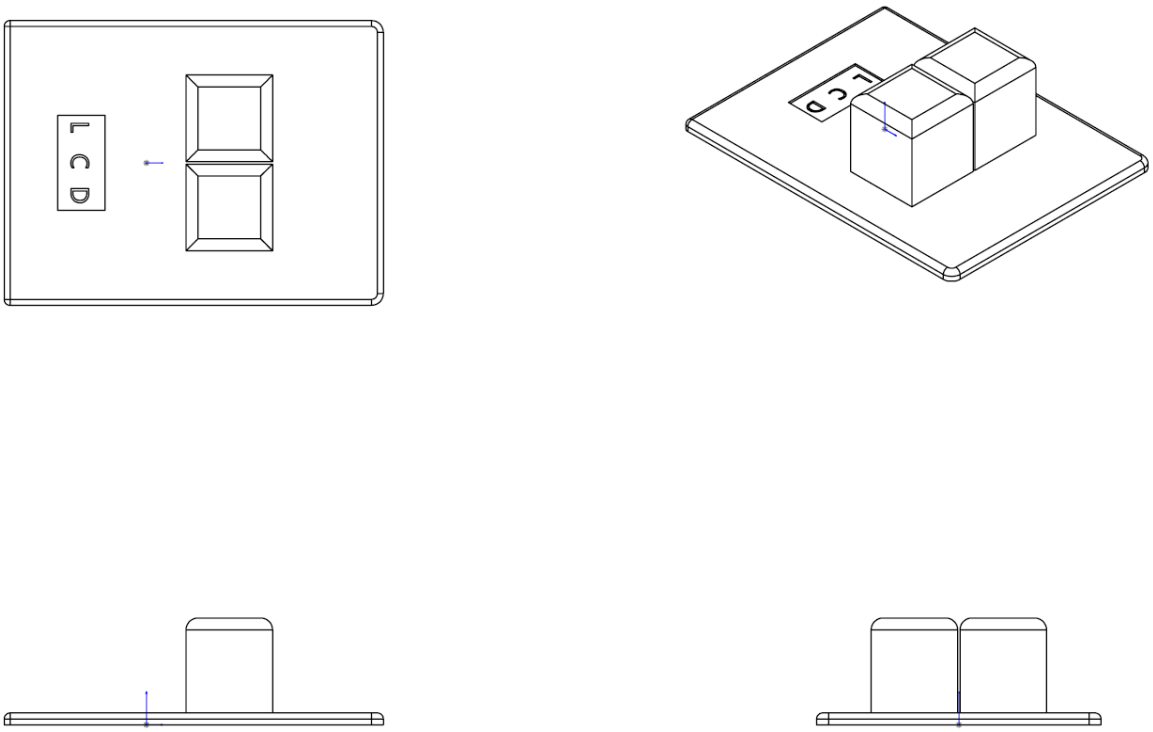


Figure 12: Smart Brew Top Encasing 3 Solidwork Design.

4.2 – Programming Code

4.2.1 – Smart Brew Arduino Code

The controls section of Smart Brew was programmed using Arduino IDE. The following code is located in Appendix D.

4.2.2 – Android Studio Code

For Smart Brew's companion App, the team decide to use an Android platform. The reason for this platform was due to the free-of-charge development of the application and also due to its flexibility with open source hardware. The following code is located in Appendix E.

5 – Qualification Test Report

Specification: Test 1

Test Name: Smartphone Interface Test

Objective: Does the smartphone application have the necessary button interfaces and also test Bluetooth communication between application and board.

Acceptance Criteria: All button must be present on the application and be able to turn on their assigned LED in order for the test to pass. Otherwise the test has failed.

Equipment List:

- Arduino Uno
- 220 Ω Resistor (6)
- 3 Green LED
- 2 Red LED
- 1 Yellow LED
- HC-06 Bluetooth Module
- Android Smartphone Application

Procedure:

1. Include Wire library in Arduino Program

2. Set Arduino UNO digital pin (13 – 8) to Output

3. Connect 220 Ω Resistors and LEDs to each of the following digital pins in Procedure 2.

4. Wire HC-06 Bluetooth Module to each pin of the Arduino UNO:

    - VCC to 5V

    - GND to GND

    - Rx to Tx

    - Tx to Rx

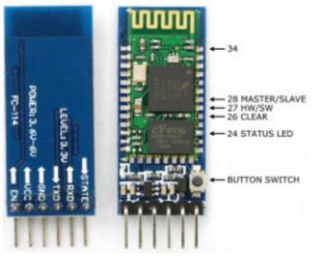


Figure 13: HC-06 Bluetooth Module Pin Out. http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-first-look/

5. Initialize simple Android Smartphone Application with necessary button interfaces.

Test Data:

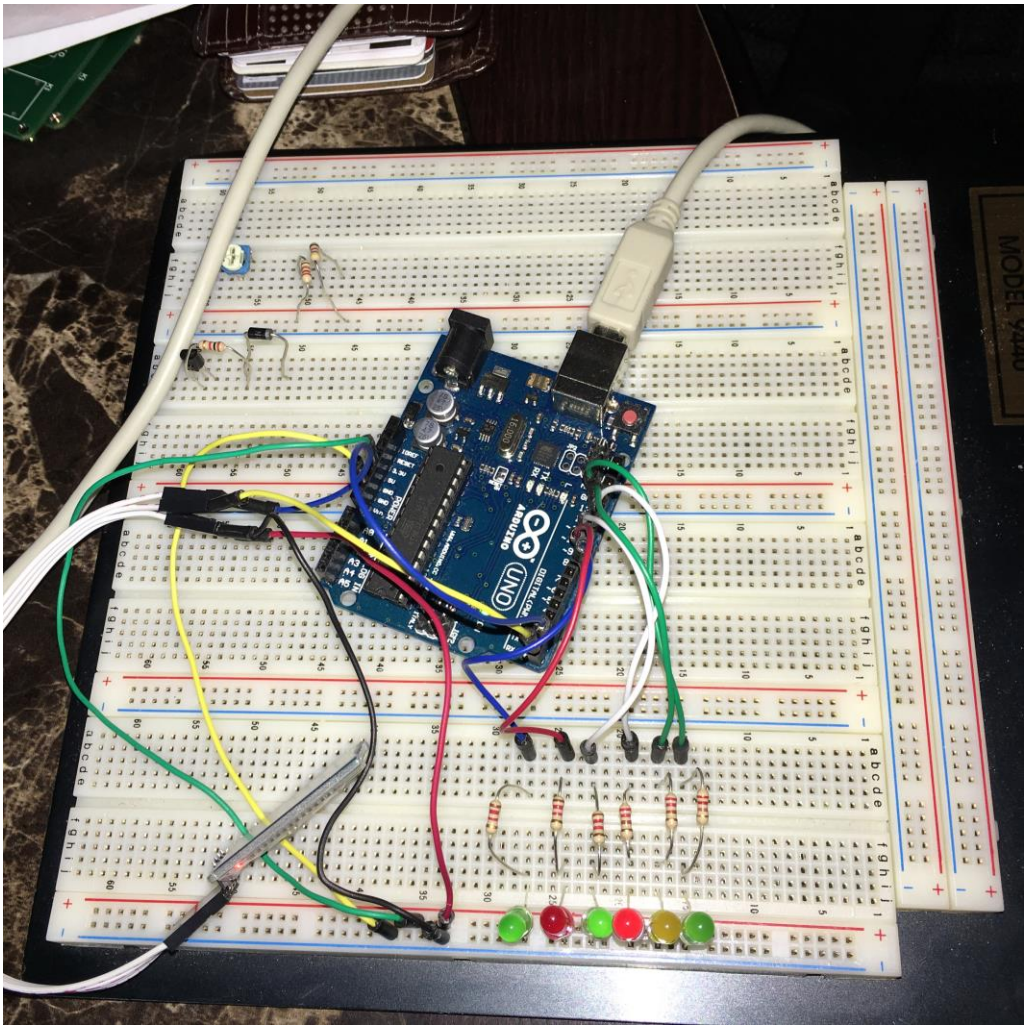


Figure 14: Smartphone Interface Test Built Circuit (Arduino Mega, HC-06 BT Module, LEDs)

	Is Button Present?	Does Button Work?
Physical Profile	Y/N	Y/N
Common Profile	Y/N	Y/N
Supplement Profile	Y/N	Y/N
Custom Profile	Y/N	Y/N
Pause	Y/N	Y/N
Resume	Y/N	Y/N

Table 5: Gather Results from Smartphone Interface Test.

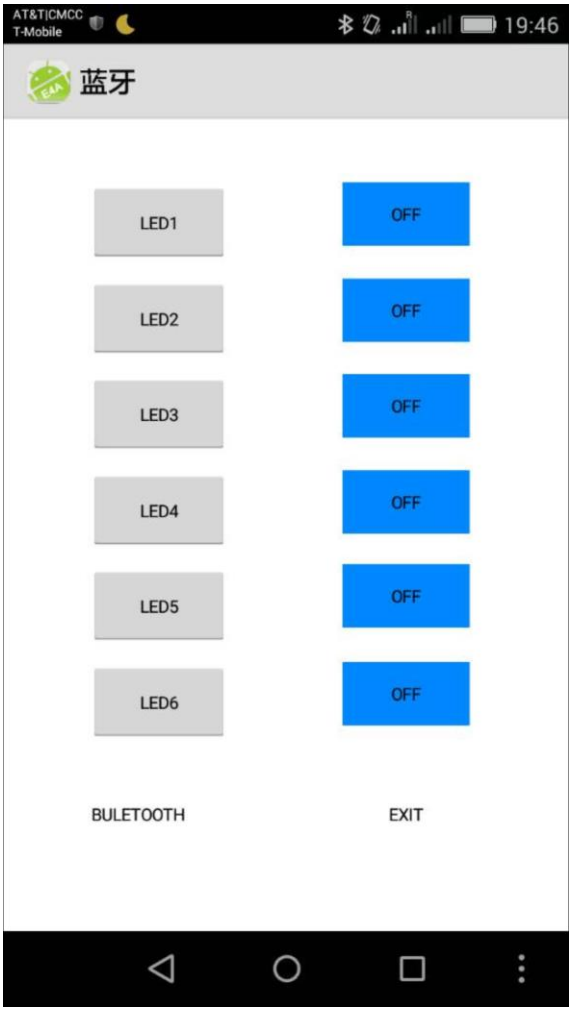


Figure 15: Simple Android Smartphone Application with six button (each representing: Physical, Common, Supplement, Custom, Start, and Pause).

Conclusion

From the data collected in Test 1, we have determined that the Acceptance Criteria was met. Our simple Android Smartphone application was set up to have six button. Each of the six buttons represented the Physical, Common, Supplement, and Custom brewing profile. The other two buttons left was for the Start and Pause button. After establishing wireless connection between the Smartphone Application and the Arduino UNO board, each button was pressed to see if they were able to turn on and off each of their assigned LED. Table 1 displays information gathered.

Arduino Code: Refer to Appendix F.

Specification: Test 2

Test Name: Water Level Test

What is to be learned from this test: Can the prototype detect water level inside the herbal pot.

Acceptance Criteria: Comparing the initial temperature and the detected temperature. If the values did change, the test is successful. If the temperatures values did not change, the test has failed.

Equipment List:

- Arduino Uno
- USB Cable
- Liquid Crystal Display
- 10 kΩ Potentiometer
- DS18B20 Thermometer
- Green L.E.D
- 220 Ω Resistor (1)
- 4.7 kΩ Resistor (1)
- Wires

Procedure

1. Setup Arduino Program and include DallasTemperature and LiquidCrystal libraries.

2. Wire to +5 V and GND

3. Wire LCD pins to the Arduino Uno Board (refer to Figure 1):

    - LCD RS Pin to digital pin 12

    - LCD Enable Pin to digital pin 11

    - LCD D4 pin to digital pin 5

    - LCD D5 pin to digital pin 4

    - LCD D6 pin to digital pin 3

    - LCD D7 pin to digital pin 2

    - 10 kΩ Potentiometer to pin 16

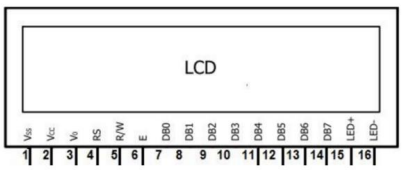


Figure 16: 20X4 LCD Pin Layout. (https://www.hackster.io/windows-iot/smartfaire-on-raspberry-pi-2-bda1dc)

4. Wire DS18B20 Thermometer according to Figure 2. Data line is the wire to digital pin 7 on the Arduino Uno.

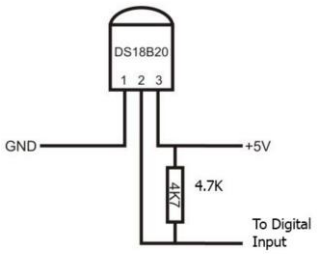


Figure 17: Wire Diagram of DS18B20.

https://arduino-info.wikispaces.com/Brick-Temperature-DS18B20

5. Wire 220 Ω Current Limit Resistor and Green L.E.D to pin 13 on the Arduino Uno.

Test Data:

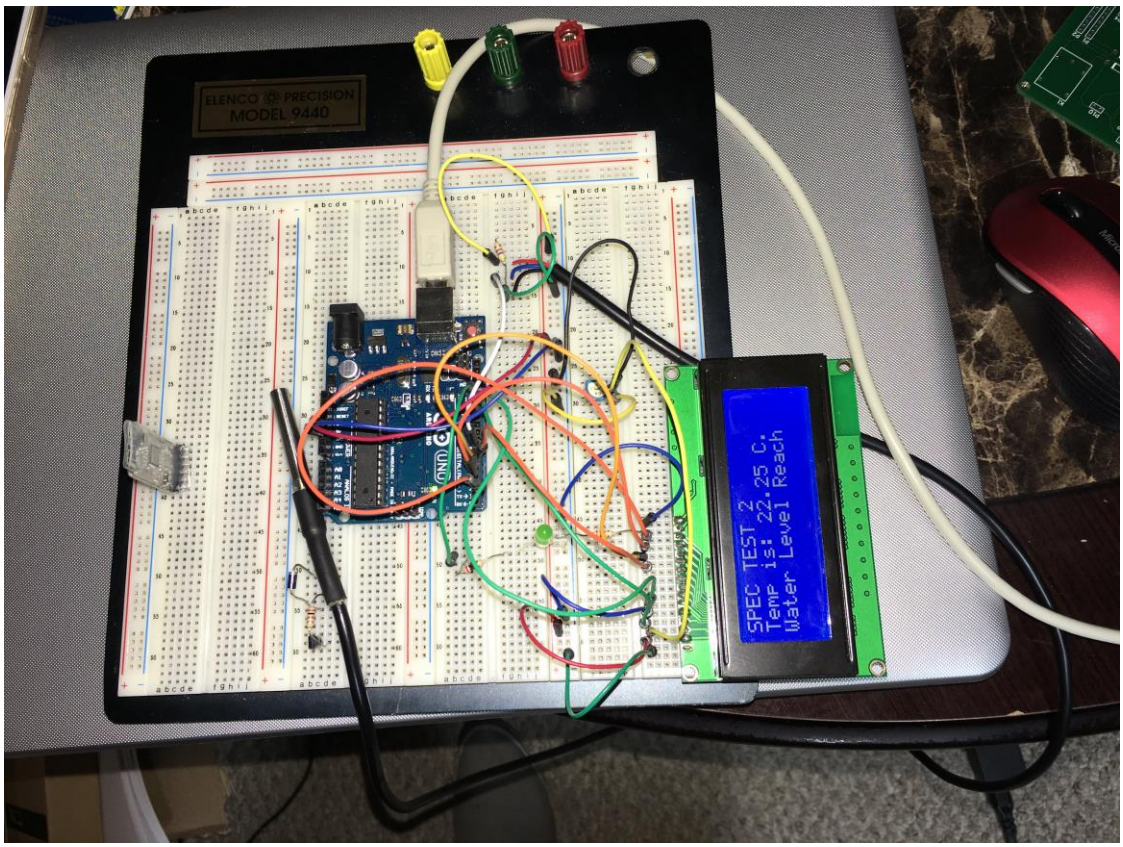


Figure 18: Arduino Uno using a DS18B20 Transducer and Displaying Temperature Values on LCD.

Total height of the herbal pot (3.0 L) is 14.7 cm.

Volume (L)	Height (cm)
1.00	2.8
1.50	4.3
2.00	5.7
2.25	6.6
2.55	7.7
3.00	9.5

Table 6: Distance from Base of the Herbal Pot to Different Water Volume.

	75% (6.6 cm)	35% (2.8cm)
Initial Temperature	24.50 °C	25.75 °C
Water Reached	19.50 °C	20.75 °C

Table 7: Initial and Measured Temperature at Different Water Volume.

Conclusion

From the data collect in Test 2, we have determined that the Acceptance Criteria was met. At the initial of this test, several measurement were done to determine the water level height from the base of the herbal pot. With a measuring cup, different water volume were poured into the herbal pot. At each volume, a ruler was used to measure the height of the water level from the base, shown in Table 1. This information would serve in a later purpose for our prototype. Specifically for this test, we used the following heights: 6.6 cm and 2.8cm. These height were selected due to one of our primary specification. Table 2, presents the different temperature registered when initialized and when the water had reach a specific height. A noticeable difference in temperature between the initial measured temperature and when the water level reach the transducer.

Arduino Code: Refer to Appendix G.

Specification: Test 3

Test Name: Brewing Profile Test #1

What is to be learned from this test: Can the Automated Herbal Pot perform the Physical temperature profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Equipment List:

- Arduino MEGA
- 220 Ω Resistor (3)
- Green LED (1)
- Yellow LED (1)
- Red LED (1)
- HC-06 Bluetooth Module
- Android Smartphone Application

Procedure:

1. Include Wire library in Arduino Program.

2. Set Arduino MEGA digital pin (7 – 9) to Output

3. Connect 220 Ω Resistors and LEDs to each of the following digital pins in Procedure 2.

4. Wire HC-06 Bluetooth Module to each pin of the Arduino UNO:

    - VCC to 5V

    - GND to GND

    - Rx to Tx

    - Tx to Rx

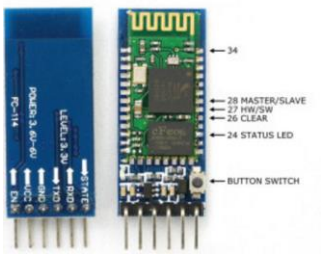


Figure 19: HC-06 Bluetooth Module Pin Out. http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-first-look/

5. Initialize Android Smartphone Application.

Test Data:

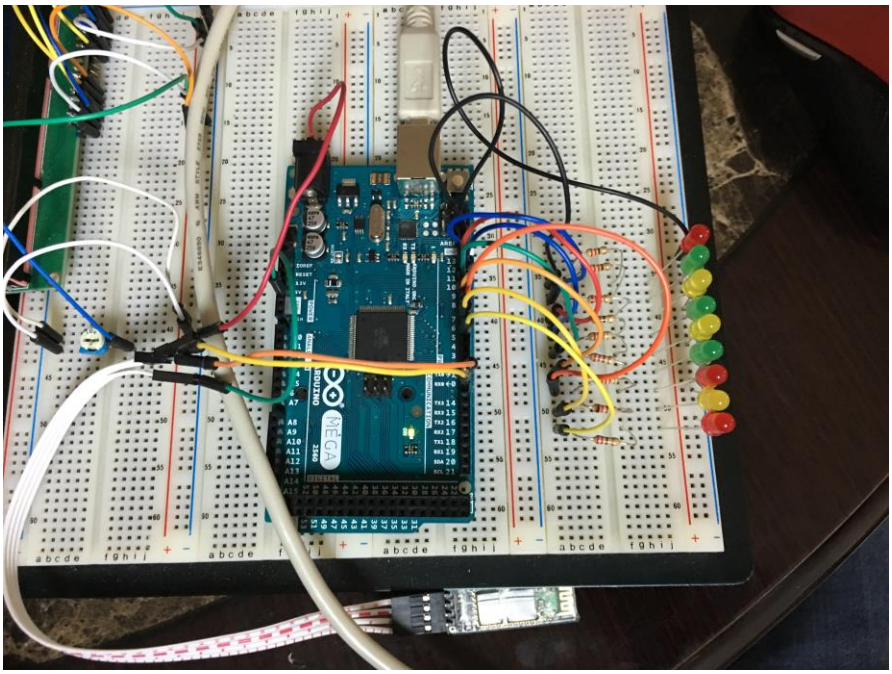


Figure 20: Smartphone Interface Test Built Circuit (Arduino UNO, HC-06 BT Module, LEDs)

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	OFF	ON	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	OFF	ON	OFF
120	OFF	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	OFF	ON

Table 8: Pre-determined Pattern for Physical Temperature Profile.

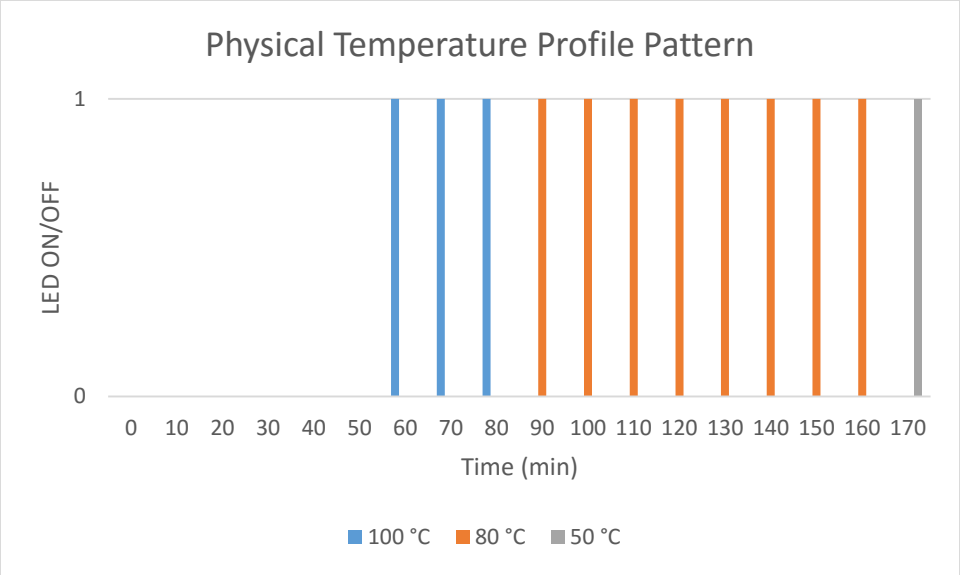


Figure 21: Observed LEDs State for Physical Temperature Profile.

Conclusion

From the data collected in Test 3, we have determined that the Acceptance Criteria was met. Several adjustments were made in order to speed the test. In our program, our counter was set that 2 seconds equals 1 minute in real time. As the program begin the Physical temperature profile, we observed 3 different LEDs. The Yellow LED symbolized when the temperature of the water inside the herbal pot has reached 100 °C. Also the Green LED for 80 °C and the RED LED for 50 °C. As seen in Table 2, our recorded observation of the on/off state of each LED did match the pre-determined on/off states of the LED, refer to Table 1.

Arduino Code: Refer to Appendix H.

Specification: Test 4

Test Name: Brewing Profile Test #2

What is to be learned from this test: Can the Automated Herbal Pot perform the Common temperature profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Equipment List:

- Arduino MEGA
- 220 Ω Resistor (3)
- Green LED (1)
- Yellow LED (1)
- Red LED (1)
- HC-06 Bluetooth Module
- Android Smartphone Application

Procedure:

1. Include Wire library in Arduino Program.

2. Set Arduino MEGA digital pin (7 – 9) to Output

3. Connect 220 Ω Resistors and LEDs to each of the following digital pins in Procedure 2.

4. Wire HC-06 Bluetooth Module to each pin of the Arduino UNO:

    - VCC to 5V

    - GND to GND

    - Rx to Tx

    - Tx to Rx

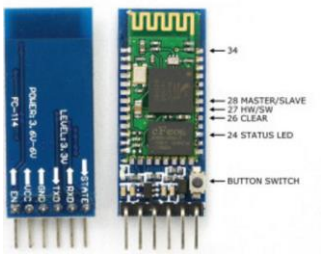


Figure 22: HC-06 Bluetooth Module Pin Out. http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-first-look/

5. Initialize Android Smartphone Application.

Test Data:

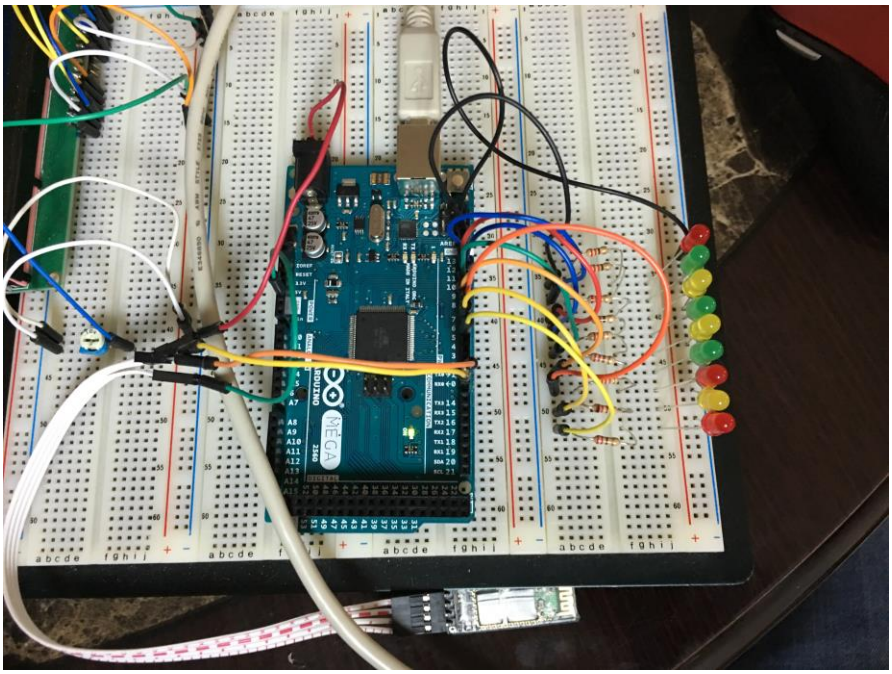


Figure 23: Smartphone Interface Test Built Circuit (Arduino UNO, HC-06 BT Module, LEDs)

Time (min)	100 °C (Yellow LED	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	OFF	ON	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	ON	OFF	OFF
120	ON	OFF	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	OFF	ON

Table 9: Pre-determined Pattern for Common Temperature Profile.

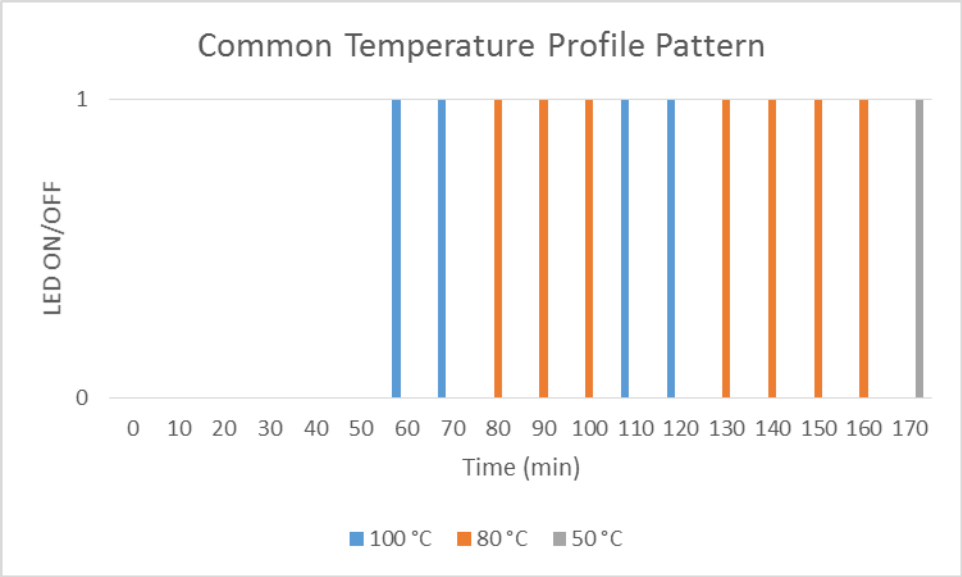


Figure 24: Observed LEDs State for Common Temperature Profile.

Conclusion

From the data collected in Test 3, we have determined that the Acceptance Criteria was met. Several adjustments were made in order to speed the test. In our program, our counter was set that 2 seconds equals 1 minute in real time. As the program begin the Common temperature profile, we observed 3 different LEDs. The Yellow LED symbolized when the temperature of the water inside the herbal pot has reached 100 °C. Also the Green LED for 80 °C and the RED LED for 50 °C. As seen in Table 2, our recorded observation of the on/off state of each LED did match the pre-determined on/off states of the LED, refer to Table 1.

Arduino Code: Refer to Appendix H.

Specification: Test 5

Test Name: Brewing Profile Test #3

What is to be learned from this test: Can the Automated Herbal Pot perform the Supplement temperature profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Equipment List:

- Arduino MEGA
- 220 Ω Resistor (3)
- Green LED (1)
- Yellow LED (1)
- Red LED (1)
- HC-06 Bluetooth Module
- Android Smartphone Application

Procedure:

1. Include Wire library in Arduino Program.

2. Set Arduino MEGA digital pin (7 – 9) to Output

3. Connect 220 Ω Resistors and LEDs to each of the following digital pins in Procedure 2.

4. Wire HC-06 Bluetooth Module to each pin of the Arduino UNO:

    - VCC to 5V

    - GND to GND

    - Rx to Tx

    - Tx to Rx

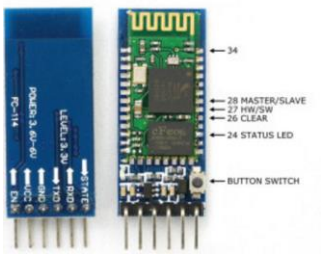


Figure 25: HC-06 Bluetooth Module Pin Out. http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-first-look/

5. Initialize Android Smartphone Application.

Test Data:

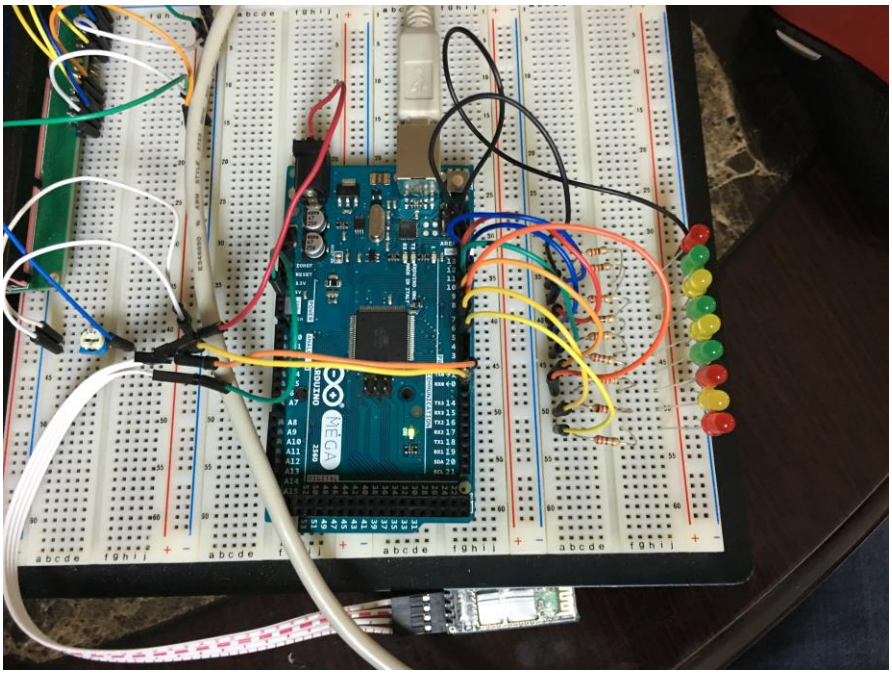


Figure 26: Smartphone Interface Test Built Circuit (Arduino UNO, HC-06 BT Module, LEDs)

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	ON	OFF	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	OFF	ON	OFF
120	OFF	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	ON	OFF
180	OFF	ON	OFF
190	OFF	ON	OFF
200	OFF	ON	OFF
210	OFF	OFF	ON

Table 10: Pre-determined Pattern for Supplement Temperature Profile.

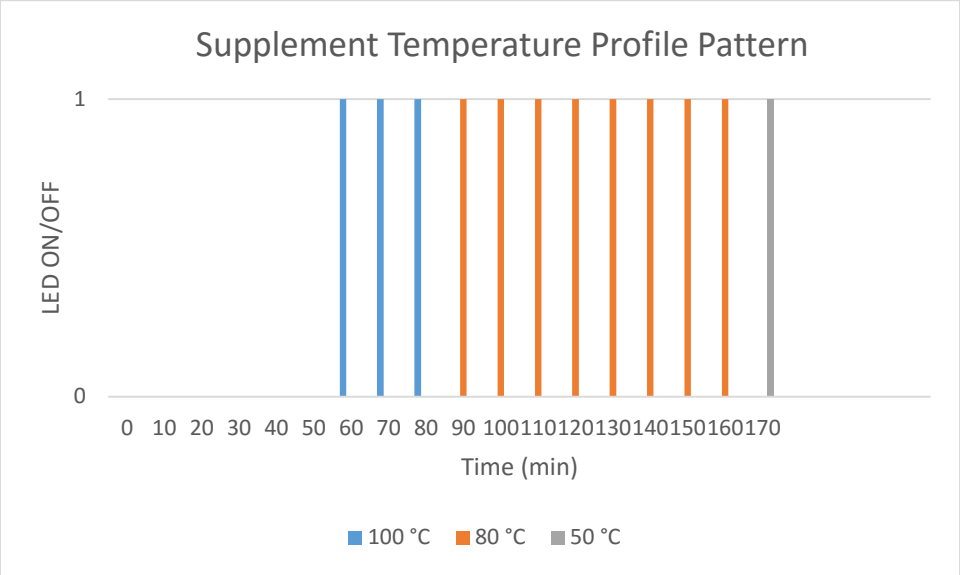


Figure 27: Observed LEDs State for Supplement Temperature Profile.

Conclusion

From the data collected in Test 3, we have determined that the Acceptance Criteria was met. Several adjustments were made in order to speed the test. In our program, our counter was set that 2 seconds equals 1 minute in real time. As the program begin the Supplement temperature profile, we observed 3 different LEDs. The Yellow LED symbolized when the temperature of the water inside the herbal pot has reached 100 °C. Also the Green LED for 80 °C and the RED LED for 50 °C. As seen in Table 2, our recorded observation of the on/off state of each LED did match the pre-determined on/off states of the LED, refer to Table 1.

Arduino Code: Refer to Appendix H.

Specification: Test 6

Test Name: Brewing Profile Test #4

What is to be learned from this test: Can the Automated Herbal Pot perform the Custom temperature profile.

Acceptance Criteria: If the LEDs does follow the pre-determined state of the LEDs, the test has passed, otherwise the test has failed.

Equipment List:

- Arduino MEGA
- 220 Ω Resistor (3)
- Green LED (1)
- Yellow LED (1)
- Red LED (1)
- HC-06 Bluetooth Module
- Android Smartphone Application

Procedure:

1. Include Wire library in Arduino Program.

2. Set Arduino MEGA digital pin (7 – 9) to Output

3. Connect 220 Ω Resistors and LEDs to each of the following digital pins in Procedure 2.

4. Wire HC-06 Bluetooth Module to each pin of the Arduino UNO:

    - VCC to 5V

    - GND to GND

    - Rx to Tx

    - Tx to Rx

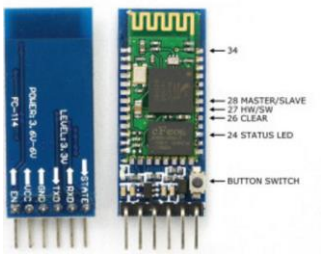


Figure 28: HC-06 Bluetooth Module Pin Out. http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-first-look/

5. Initialize Android Smartphone Application.

Test Data:

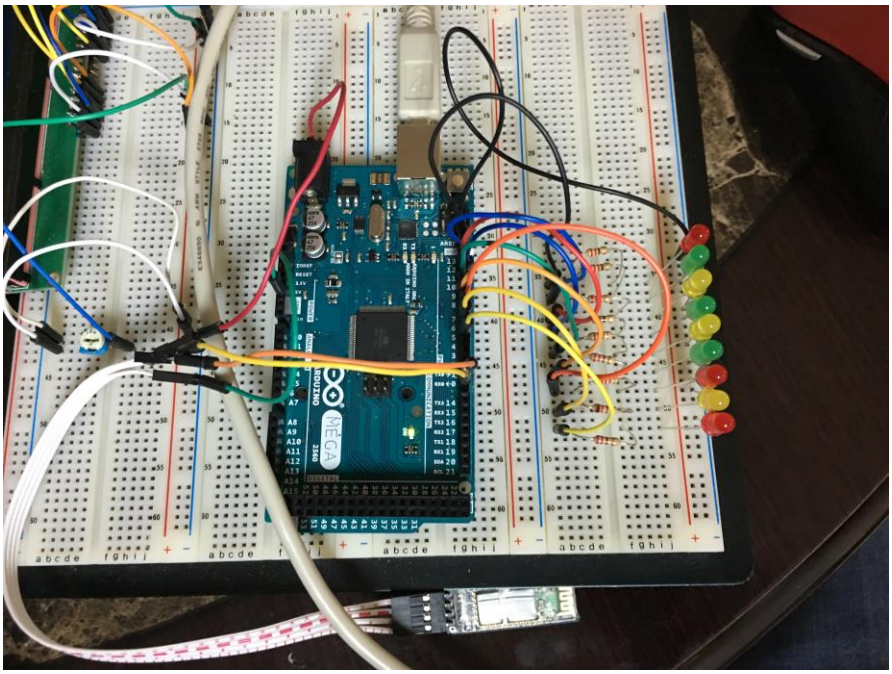


Figure 29: Smartphone Interface Test Built Circuit (Arduino UNO, HC-06 BT Module, LEDs)

Time (min)	100 °C (Yellow LED)	80 °C (Green LED)	50 °C (Red LED)
0	OFF	OFF	OFF
10	OFF	OFF	OFF
20	OFF	OFF	OFF
30	OFF	OFF	OFF
40	OFF	OFF	OFF
50	OFF	OFF	OFF
60	ON	OFF	OFF
70	ON	OFF	OFF
80	ON	OFF	OFF
90	OFF	ON	OFF
100	OFF	ON	OFF
110	OFF	ON	OFF
120	OFF	ON	OFF
130	OFF	ON	OFF
140	OFF	ON	OFF
150	OFF	ON	OFF
160	OFF	ON	OFF
170	OFF	OFF	ON

Table 11: Pre-determined Pattern for Custom Temperature Profile.

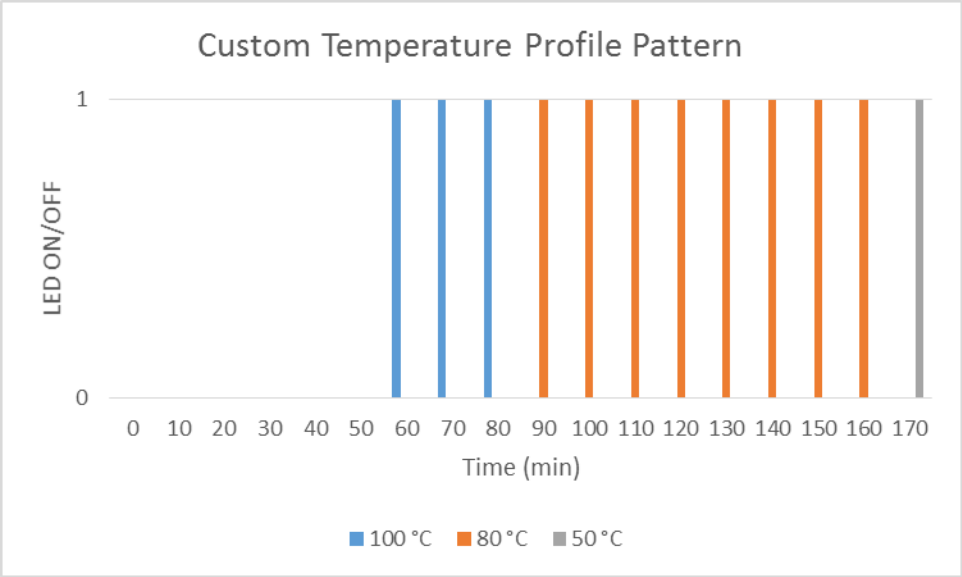


Figure 30: Observed LEDs State for Custom Temperature Profile.

Conclusion

From the data collected in Test 3, we have determined that the Acceptance Criteria was met. Several adjustments were made in order to speed the test. In our program, our counter was set that 2 seconds equals 1 minute in real time. As the program begin the Custom temperature profile, we observed 3 different LEDs. The Yellow LED symbolized when the temperature of the water inside the herbal pot has reached 100 °C. Also the Green LED for 80 °C and the RED LED for 50 °C. As seen in Table 2, our recorded observation of the on/off state of each LED did match the pre-determined on/off states of the LED, refer to Figure 31.

Arduino Code: Refer to Appendix H.

Specification: Test 7

Test Name: Heating Element Test

What is to be learned from this test: Can the prototype be able to turn on/off the heating element at a determined temperature.

Acceptance Criteria: The test is successful if the LED is turned on at 25 °C and also if the LED is turned off at 100 °C.

Equipment List:

- Arduino MEGA
- 220 Ω Resistor (1)
- Red LED (1)
- 2N2222 Transistor (1)
- 1N4001 Diode (1)
- 20x4 LCD
- DS18B20 Thermometer (1)
- JQC-3F Relay
- Heating Element

Procedure:

1. Include OneWIre, DallasTemperature, and LiquidCrystal library in Arduino Program.

2. Wire the circuit as shown in Figure 1.

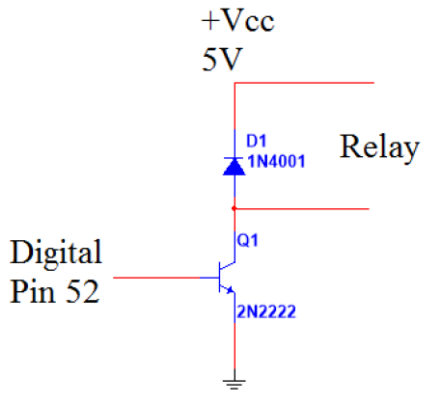


Figure 31: Diode and Transistor Circuit Diagram.

3. Set Arduino MEGA digital pin 7 to Output.

4. Wire 220 Ω current limit resistor and Red LED to digital pin 7.

5. Wire LCD pins to the Arduino Uno Board (refer to Figure 1):

- LCD RS Pin to digital pin 24

- LCD Enable Pin to digital pin 25

- LCD D4 pin to digital pin 29
- LCD D5 pin to digital pin 28
- LCD D6 pin to digital pin 27
- LCD D7 pin to digital pin 26
- 10 kΩ Potentiometer to pin 16

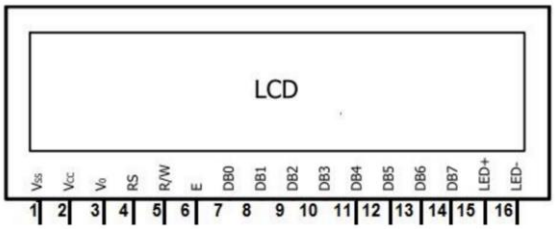


Figure 32: 20X4 LCD Pin Layout. (https://www.hackster.io/windows-iot/smartfaire-on-raspberry-pi-2-bda1dc)

Test Data:

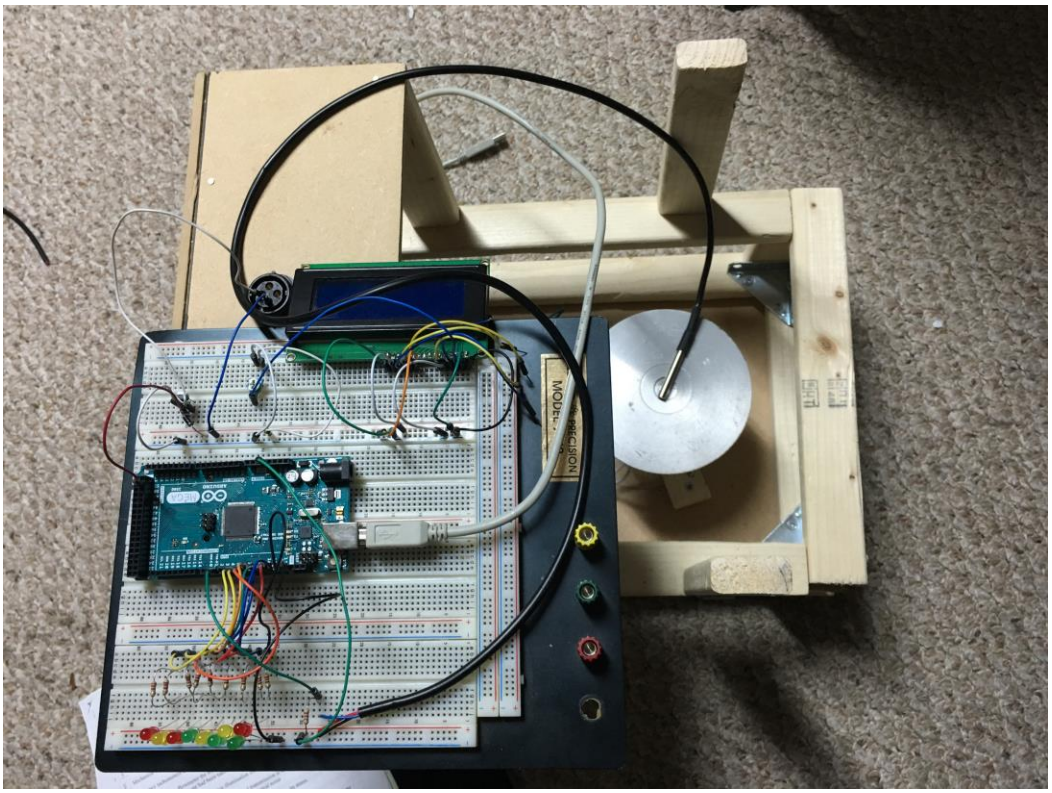


Figure 33: Arduino MEGA using a DS18B20 Transducer and Displaying Temperature Values on LCD.

Temperature (°C)	Displayed Temperature (°C)	LED ON/OFF
25	25.25	ON
100	100.50	OFF

Table 12: Results from Test Display Current Temperature and State of LED.

Conclusion

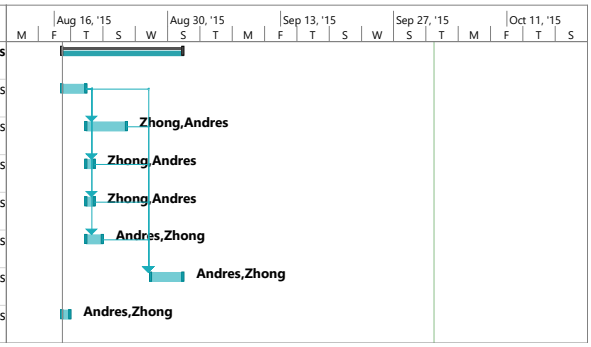
From the data collected in Test 7, we have determined that the Acceptance Criteria was met. As the Arduino program initialized, the heating element begins to heat up. The led should turn on when the temperature reaches 25 °C. As it continues to heat up and reaches 100 °C, the LED turns off. This result can be displayed in Table 1.

Arduino Code: Refer to Appendix I.

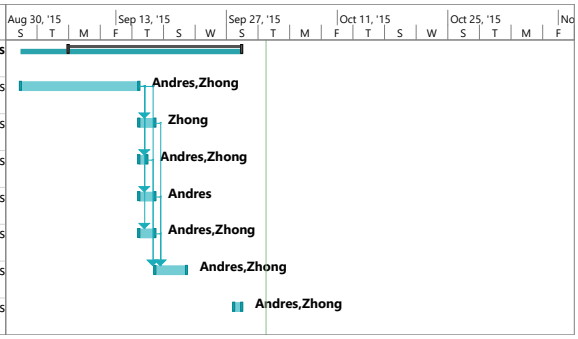
Appendix A

Gantt Charts

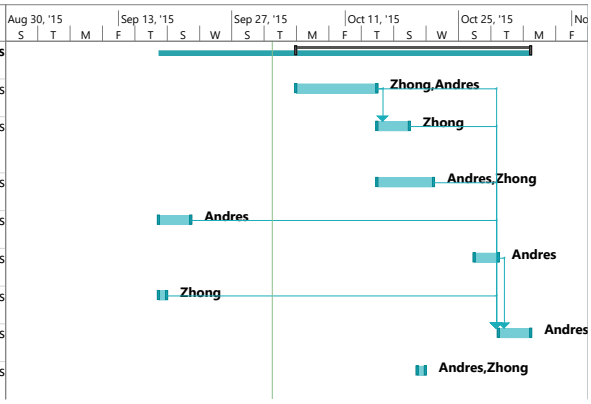
ID	Task Name	Duration	Start	Finish	Predecessors	Work
2	**Gate 1**	**11 days**	**Aug 17 '15**	**Aug 31 '15**		**160 hrs**
3	Project Concept	3 days	Aug 17 '15	Aug 19 '15		0 hrs
4	Abstract of Project	3 days	Aug 20 '15	Aug 24 '15	3	48 hrs
5	Team Name, Logo	1 day	Aug 20 '15	Aug 20 '15	3	16 hrs
6	Bios	1 day	Aug 20 '15	Aug 20 '15	3	16 hrs
7	30 Sec. Elevator Pitch	2 days	Aug 20 '15	Aug 21 '15	3	32 hrs
8	Gate 1 Presentation	2 days	Aug 28 '15	Aug 31 '15	3,4,5,6,7	32 hrs
9	Purchase Parts	1 day	Aug 17 '15	Aug 17 '15		16 hrs



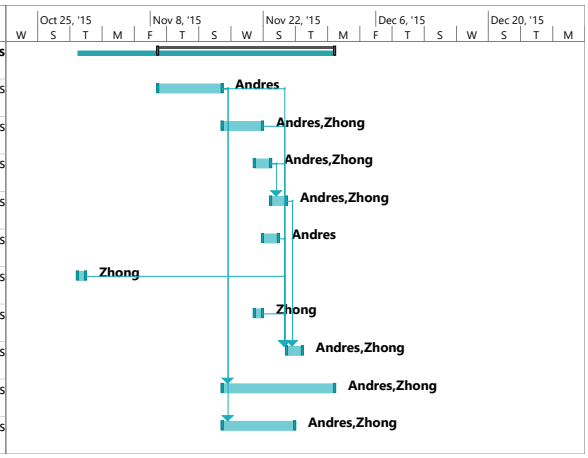
ID	Task Name	Duration	Start	Finish	Predecessors	Work
10	**Gate 2**	**16 days**	**Sep 7 '15**	**Sep 28 '15**	**2**	**304 hrs**
11	Signed Specification	11 days	Sep 1 '15	Sep 15 '15		176 hrs
12	Concep. Design Report	2 days	Sep 16 '15	Sep 17 '15	11	16 hrs
13	List of Standards	1 day	Sep 16 '15	Sep 16 '15	11	16 hrs
14	Init. Market Report	2 days	Sep 16 '15	Sep 17 '15	11	16 hrs
15	Project Schedule	2 days	Sep 16 '15	Sep 17 '15	11	32 hrs
16	Gate 2 Presentation	2 days	Sep 18 '15	Sep 21 '15	15,14,13,12,11	32 hrs
17	Purchase Parts	1 day	Sep 28 '15	Sep 28 '15		16 hrs



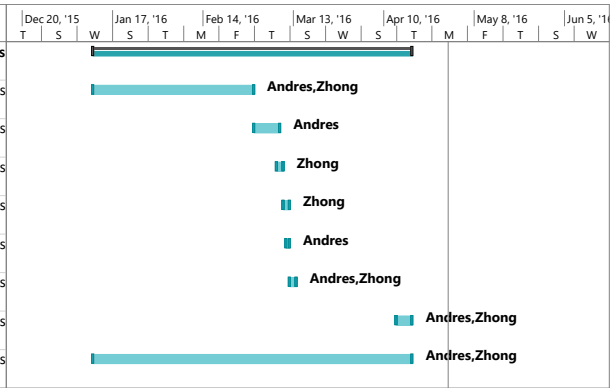
ID	Task Name	Duration	Start	Finish	Predecessors	Work
18	**Gate 3**	**21 days**	**Oct 5 '15**	**Nov 2 '15**	**10**	**328 hrs**
19	Signed Crit. Design Review	8 days	Oct 5 '15	Oct 14 '15		128 hrs
20	Preliminary Design Documents	3 days	Oct 15 '15	Oct 18 '15	19	24 hrs
21	Qualification Test Plan	5 days	Oct 15 '15	Oct 21 '15		80 hrs
22	Updated Marketing Report	2 days	Sep 18 '15	Sep 21 '15	14	16 hrs
23	Project Budget Estimate	3 days	Oct 27 '15	Oct 29 '15		24 hrs
24	Project Sched. Update	1 day	Sep 18 '15	Sep 18 '15	15	8 hrs
25	Gate 3 Presentation	2 days	Oct 30 '15	Nov 2 '15	19,20,21,22,23,2	32 hrs
26	Purchase Parts	1 day	Oct 20 '15	Oct 20 '15		16 hrs



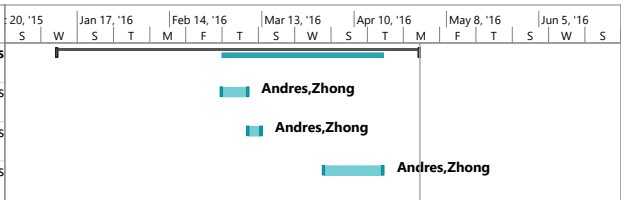
ID	Task Name	Duration	Start	Finish	Predecessors	Work
27	**Gate 4**	**16 days**	**Nov 9 '15**	**Nov 30 '15**	**18**	**528 hrs**
28	Detailed Design Documents	6 days	Nov 9 '15	Nov 16 '15		48 hrs
29	Engineering Simulation	5 days	Nov 17 '15	Nov 21 '15		80 hrs
30	Marketing Website	2 days	Nov 21 '15	Nov 22 '15		32 hrs
31	Promo. Video	2 days	Nov 23 '15	Nov 24 '15	30	32 hrs
32	Updated Marketing Report	2 days	Nov 22 '15	Nov 23 '15	22	16 hrs
33	Project Budget Update	1 day	Oct 30 '15	Oct 30 '15	23	8 hrs
34	Project Schedule Update	1 day	Nov 21 '15	Nov 21 '15	24	8 hrs
35	Gate 4 Presentation	2 days	Nov 25 '15	Nov 26 '15	28,29,30,31,32,3	32 hrs
36	Programming	10 days	Nov 17 '15	Nov 30 '15	28	160 hrs
37	Order PCB	7 days	Nov 17 '15	Nov 25 '15	28	112 hrs



ID	Task Name	Duration	Start	Finish	Predecessors	Work
38	**Gate 5**	**71 days**	**Jan 11 '16**	**Apr 18 '16**	**27**	**1,880 hrs**
39	Working Prototype	36 days	Jan 11 '16	Feb 29 '16		576 hrs
40	Marketing Presentation	6 days	Mar 1 '16	Mar 8 '16		48 hrs
41	Project Budget Update	2 days	Mar 8 '16	Mar 9 '16		16 hrs
42	Project Schedule Update	2 days	Mar 10 '16	Mar 11 '16		16 hrs
43	One Page Leaflet	1 day	Mar 11 '16	Mar 11 '16		8 hrs
44	Tradeshow Poster	2 days	Mar 12 '16	Mar 13 '16		32 hrs
45	Gate 5 Presentation	3 days	Apr 14 '16	Apr 18 '16		48 hrs
46	Programming	71 days	Jan 11 '16	Apr 18 '16		1,136 hrs



ID	Task Name	Duration	Start	Finish	Predecessors	Work
47	**Gate 6**	**80 days**	**Jan 11 '16**	**Apr 29 '16**	**38**	**352 hrs**
48	Final Engineering Report	6 days	Mar 1 '16	Mar 8 '16		96 hrs
49	Business Plan	4 days	Mar 9 '16	Mar 12 '16		64 hrs
50	Service and Product Manuals	12 days	Apr 1 '16	Apr 18 '16		192 hrs



Appendix B

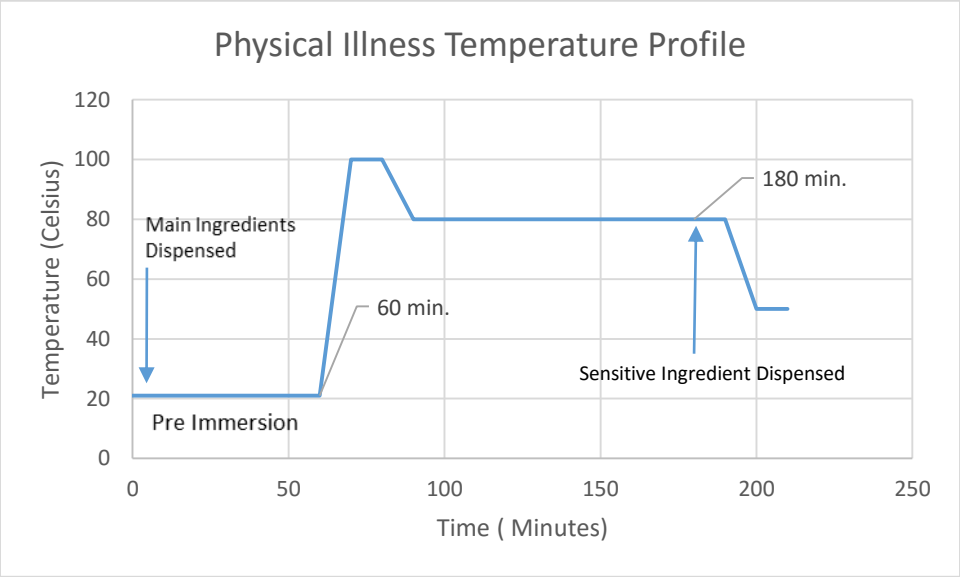


Figure 34: Temperature vs Time Brewing Profile

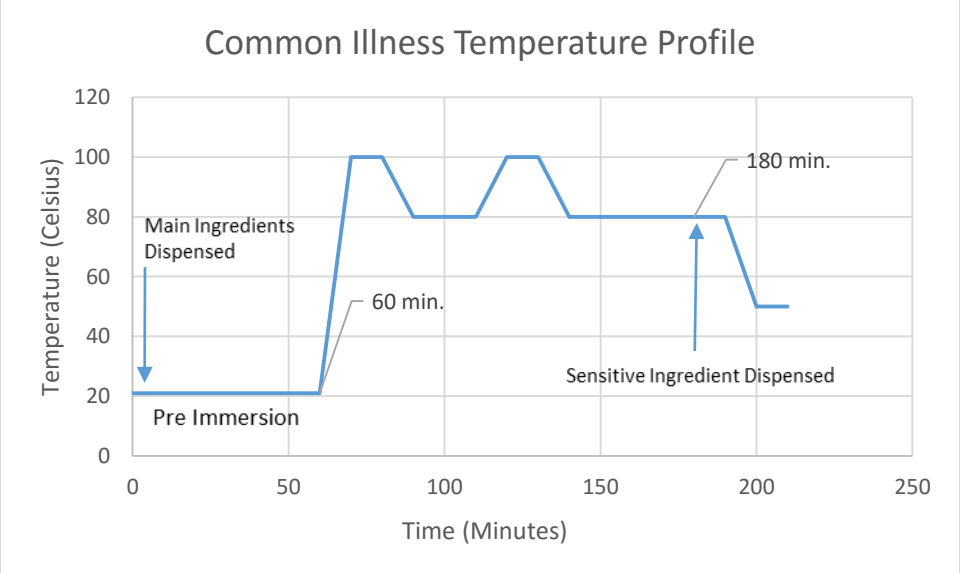


Figure 35: Temperature vs Time Brewing Profile

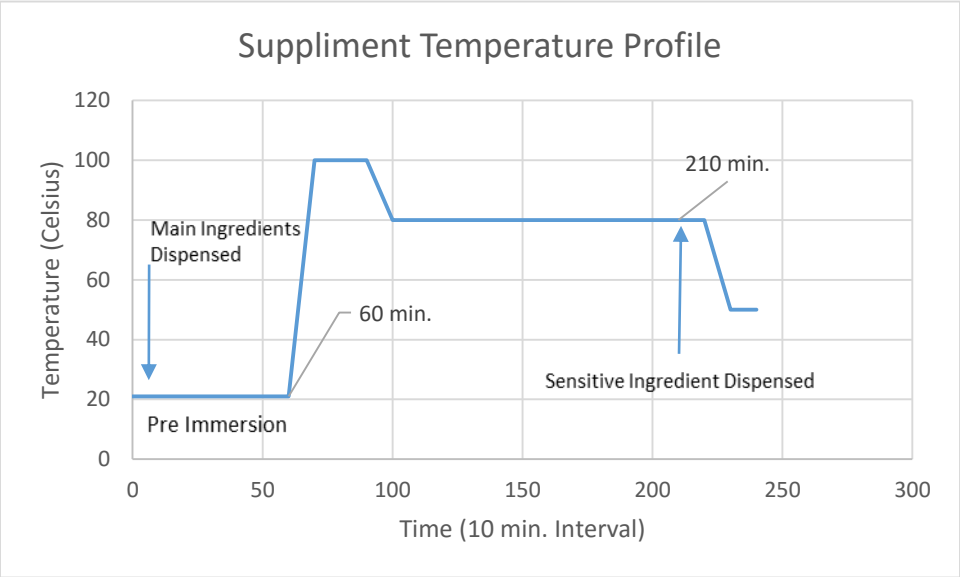


Figure 36: Temperature vs Time Brewing Profile

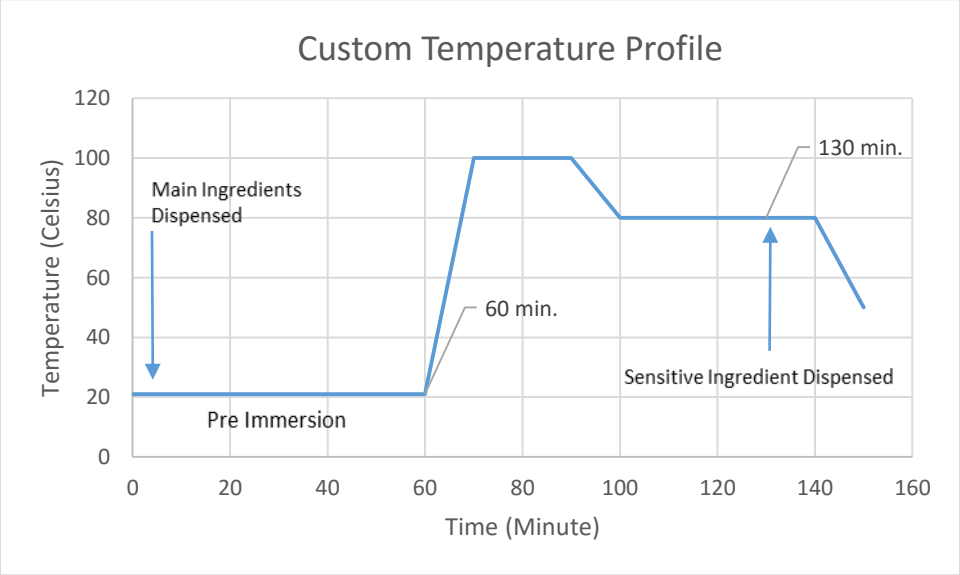


Figure 37: Temperature vs Time Brewing Profile

Appendix C

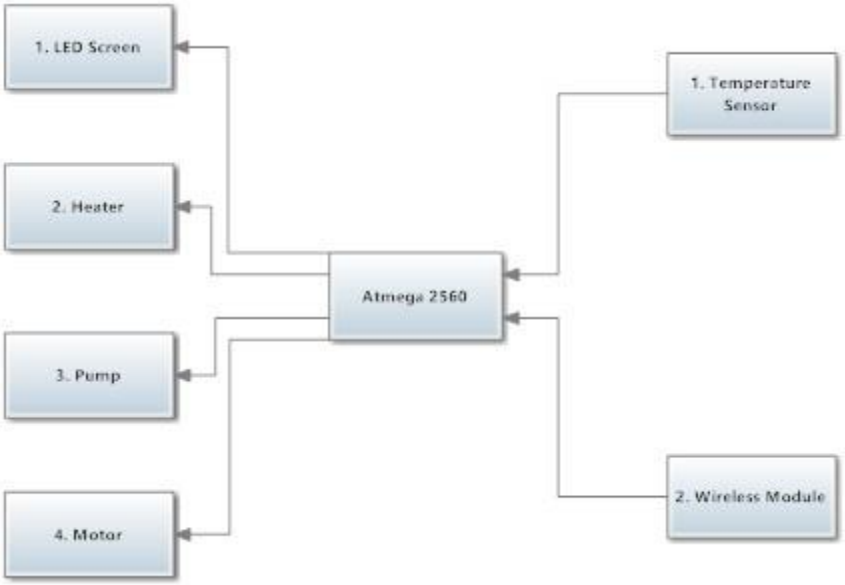


Figure 38: Smart Brew Input and Output Block Diagram.

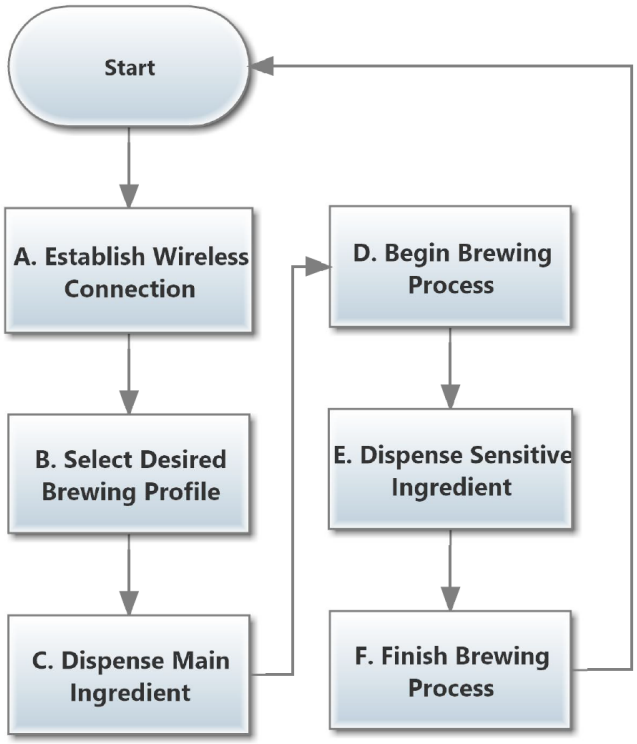


Figure 39: Smart Brew Simple Flow Chart.

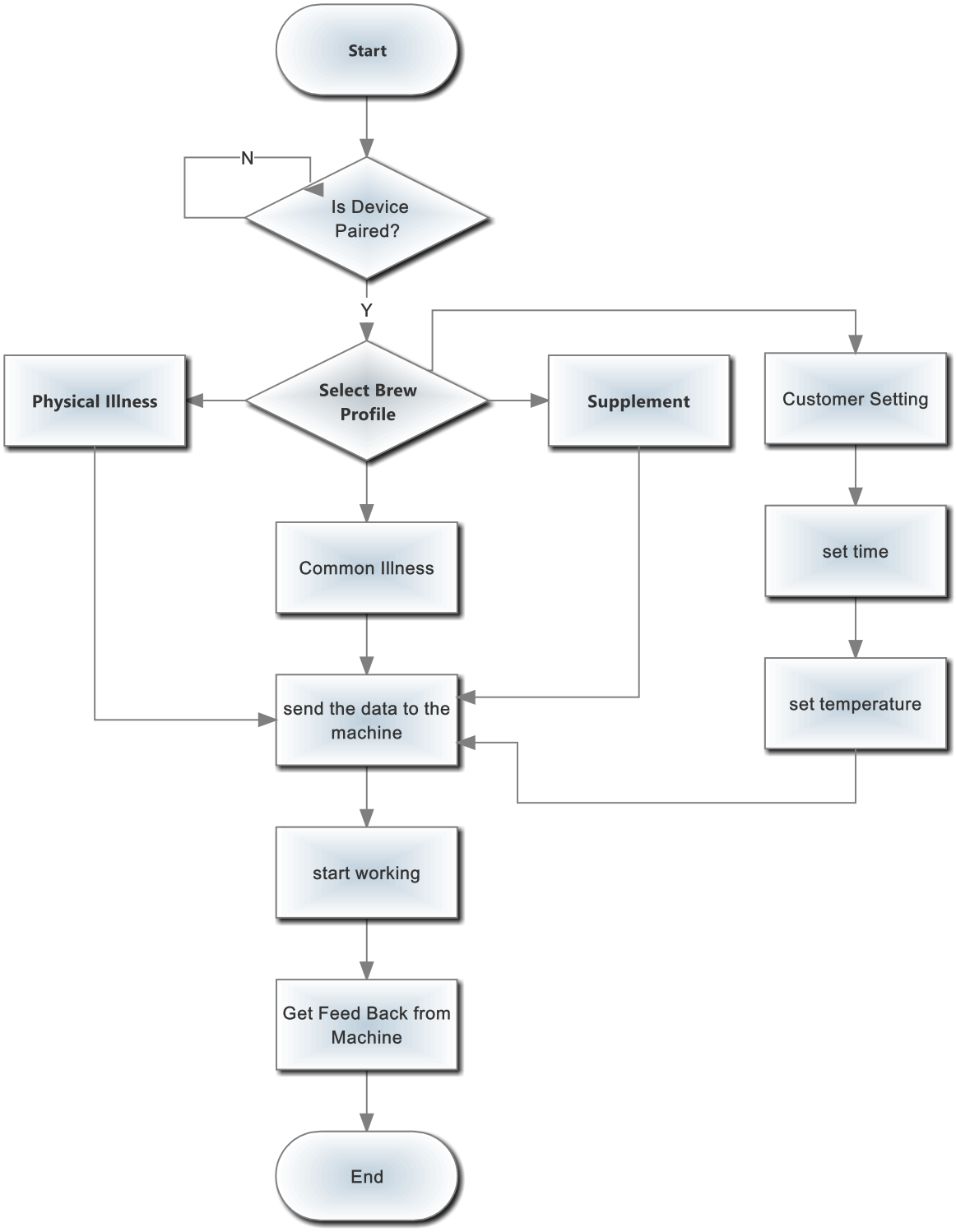


Figure 40: Android Application Flowchart.

Appendix D

```
//these are library which used in below program

#include <Wire.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include <LiquidCrystal.h>


//set up all global variable and const number

const int EN1 = 21;

const int EN2 = 20;

const int MR = 19;

const int MB = 18;

const int PUMP = 23;

const int HEATER = 22;

const int SW1 = 42;

const int SW2 = 43;

const int SW3 = 44;

const int SW4 = 45;

float a;

float b;

float c;

float tempture1;

int p;
```

```
int water = 0;

int buttonState = 0;

bool check1 = false;

bool check2 = false;

bool check3 = false;

bool motor = false;

bool pu = false;

bool start = false;

bool pause = false;

bool restart = false;

bool highheat = false;

bool slowheat = false;

bool keepheat = false;

bool motor2 = false;

bool strong = false;

bool motormove = false;

bool heat = false;


// set up LCD screen

LiquidCrystal lcd(24, 25, 29, 28, 27, 26);
```

```
// set up RTD sensor
#define ONE_WIRE_BUS 30
OneWire oneWire (ONE_WIRE_BUS); // on digital pin 30
DallasTemperature sensors(&oneWire);
DeviceAddress RTD1 = {0x28, 0xFF, 0xE3, 0x0E, 0x91, 0x15, 0x01, 0x07};
DeviceAddress RTD2 = {0x28, 0xFF, 0x50, 0x8D, 0x65, 0x15, 0x02, 0xA6};
DeviceAddress RTD3 = {0x28, 0xFF, 0xAD, 0x5E, 0x90, 0x15, 0x03, 0x83};

//start the program setup
void setup() {
  pinMode(PUMP, OUTPUT);
  digitalWrite(PUMP, HIGH);
  Serial.begin(9600);
  lcd.begin(20, 4);
  sensors.begin();
  sensors.setResolution(RTD1, 10);
  sensors.setResolution(RTD2, 10);
  sensors.setResolution(RTD3, 10);
  pinMode(EN1, OUTPUT);
  pinMode(EN2, OUTPUT);
  pinMode(MR, OUTPUT);
  pinMode(MB, OUTPUT);
  pinMode(HEATER, OUTPUT);
```

```
  pinMode(SW1,INPUT);
  pinMode(SW2,INPUT);
  pinMode(SW3,INPUT);
  pinMode(SW4,INPUT);
  lcd.setCursor(0, 1);
  lcd.print("Welcome to Smartbrew");
}

// the program main loop, this loop mainly tests the subroutine can be used or not by Boolean variable, then decide to go to which subroutine
void loop() {
    if (pause == true){
       lcd.clear();
       lcd.setCursor(7, 1);
       lcd.print("Pause");
       PauseSET();
    }
    if (motor == true){
       lcd.clear();
       lcd.setCursor(2, 1);
       lcd.print("Loading Ingredient");
       MOTORstart();
       lcd.clear();
```

```
  lcd.setCursor(3, 1);
  lcd.print("Finish Loading");
}
if (pu == true){
  lcd.clear();
  lcd.setCursor(3, 1);
  lcd.print("Filling Water");
  PUMPstart();
  lcd.clear();
  lcd.setCursor(4, 1);
  lcd.print("Water Filled");
}
if (highheat == true){
  lcd.clear();
  lcd.setCursor(5, 1);
  lcd.print("High Heat");
  HighHeater();
}
if (slowheat == true){
  lcd.clear();
  lcd.setCursor(4, 1);
  lcd.print("Medium Heat");
  SlowHeater();
```

```
    }
    if (keepheat == true){
      lcd.clear();
      lcd.setCursor(5, 1);
      lcd.print("Keep Warm");
      KeepHeater();
    }
    if (motor2 == true) {
     MOTOR2start();
    }
    if (restart == true) {
     RestartSET();
    }
    recive();
}
```

//This is the received subroutine, it mainly receives data from Bluetooth and changes the global Boolean variable to open or close the other subroutine in the main loop

```
void recive(){
  while (Serial.available() > 0) {
       p = Serial.parseInt();
       if (Serial.read() == 'X') {
            switch (p) {
```

```
case 116:
  if (motormove == false){
    if (motor == false){
    motor = !motor;
    }
    if (pu == false){
      pu = !pu;
    }
    if (strong == false) {
      strong = !strong;
    }
  }
    if (start == false){
      start = !start;
    }
    break;
case 117:
  if (motormove == false){
    if (motor == false){
    motor = !motor;
    }
    if (pu == false){
      pu = !pu;
```

```
    }
  }
    if (start == false){
      start = !start;
    }
    break;
case 118:
    if (pause == false){
      pause = !pause;
    }
    break;
case 119:
    if (restart == false){
      restart = !restart;
    }
    if (slowheat == true){
      slowheat = !slowheat;
    }
    if (highheat == true){
      highheat = !highheat;
    }
    if (keepheat == true){
      keepheat = !keepheat;
```

```
    }
    break;
case 120:
    if (highheat == false){
      highheat = !highheat;
    }
    if (slowheat == true){
      slowheat = !slowheat;
    }
    break;
case 121:
    if (slowheat == false){
      slowheat = !slowheat;
    }
    if (highheat == true){
      highheat = !highheat;
    }
    break;
case 122:
    if (keepheat == false){
      keepheat = !keepheat;
    }
    if (slowheat == true){
```

```
                slowheat = !slowheat;
               }
               if (highheat == true){
                highheat = !highheat;
               }
               break;
            case 130:
               if (motor2 == false){
                motor2 = !motor2;
               }
               break;
            default:
               break;
          }
       }
    }
}

//this is the primary ingredient subroutine, which controls the primary ingredient cart
void MOTORstart(){
  digitalWrite(EN1, HIGH);
  digitalWrite(MR, HIGH);
  buttonState = digitalRead(SW1);
```

```
 while(buttonState == LOW){
   digitalWrite(MR, HIGH);
    buttonState = digitalRead(SW1);
  }
  digitalWrite(MR, LOW);
  delay(3000);
  digitalWrite(MB, HIGH);
  buttonState = digitalRead(SW2);
  while(buttonState == LOW){
    digitalWrite(MB, HIGH);
    buttonState = digitalRead(SW2);
  }
  digitalWrite(MB, LOW);
  digitalWrite(EN1, LOW);
  motormove = !motormove;
  motor = !motor;
  Serial.println(151);
}

void MOTOR2start(){
  digitalWrite(EN2, HIGH);
  digitalWrite(MR, HIGH);
  buttonState = digitalRead(SW3);
```

```
 while(buttonState == LOW){
   digitalWrite(MR, HIGH);
    buttonState = digitalRead(SW3);
  }
  digitalWrite(MR, LOW);
  delay(3000);
  digitalWrite(MB, HIGH);
  buttonState = digitalRead(SW4);
  while(buttonState == LOW){
    digitalWrite(MB, HIGH);
    buttonState = digitalRead(SW4);
  }
  digitalWrite(MB, LOW);
  digitalWrite(EN2, LOW);
  motor2 = !motor2;
}


//this is the pump subroutine which controls the pump to open/close
void PUMPstart(){
  digitalWrite(PUMP, LOW);
  float temp;
  if (strong == true){
      sensors.requestTemperatures();
```

```
if (check1 == false) {
    a = sensors.getTempC(RTD1);
    check1 = !check1;
}
sensors.requestTemperatures();
temp = sensors.getTempC(RTD1);
while ((temp > (a - 0.5)) && (temp < (a + 0.5))){
    sensors.requestTemperatures();
    temp = sensors.getTempC(RTD1);
}
sensors.requestTemperatures();
if (check2 == false) {
    b = sensors.getTempC(RTD2);
    check2 = !check2;
}
sensors.requestTemperatures();
temp = sensors.getTempC(RTD2);
while ((temp > (b - 0.5)) && (temp < (b + 0.5))){
    sensors.requestTemperatures();
    temp = sensors.getTempC(RTD2);
}
check1 = !check1;
check2 = !check2;
```

```
}
else{
    sensors.requestTemperatures();
    if (check1 == false) {
        a = sensors.getTempC(RTD1);
        check1 = !check1;
    }
    sensors.requestTemperatures();
    temp = sensors.getTempC(RTD1);
    while ((temp > (a - 0.5)) && (temp < (a + 0.5))){
        sensors.requestTemperatures();
        temp = sensors.getTempC(RTD1);
    }
    sensors.requestTemperatures();
    if (check3 == false) {
        c = sensors.getTempC(RTD3);
        check3 = !check3;
    }
    sensors.requestTemperatures();
    temp = sensors.getTempC(RTD3);
    while ((temp > (c - 0.5)) && (temp < (c + 0.5))){
        sensors.requestTemperatures();
        temp = sensors.getTempC(RTD3);
```

```
      }
      check1 = !check1;
      check3 = !check3;
  }
  digitalWrite(PUMP, HIGH);
  if (pu == true) {
     Serial.println(152);
     pu = !pu;
  }
}

//this is the high heat subroutine which is to control the heating element to keep around 90 C
void HighHeater() {
    sensors.requestTemperatures();
    tempture1 = sensors.getTempC(RTD1);
    Serial.println(tempture1);
    lcd.setCursor(3, 2);
    lcd.print("Temperature is");
    lcd.setCursor(7, 3);
    lcd.print(tempture1);
    lcd.print(" C");
    delay(1000);
    if ((tempture1 < 89) && (heat == false)){
```

```
        digitalWrite(HEATER, HIGH);
        heat = !heat;
    }
    else if (tempture1 > 89) {
        digitalWrite(HEATER, LOW);
        if (heat == true){
            heat = !heat;
        }
        water = water + 1;
    }
    if ((water > 0) && (tempture1 < 79)){
       PUMPstart();
    }
}

//this is the slow heat subroutine which is to control the heating element to keep around 80 C
void SlowHeater() {
    sensors.requestTemperatures();
    tempture1 = sensors.getTempC(RTD1);
    Serial.println(tempture1);
    lcd.setCursor(3, 2);
    lcd.print("Temperature is");
    lcd.setCursor(7, 3);
```

```
    lcd.print(tempture1);
    lcd.print(" C");
    delay(1000);
    if ((tempture1 < 80) && (heat == false)){
        digitalWrite(HEATER, HIGH);
        heat = !heat;
    }
    else if (tempture1 > 81) {
        digitalWrite(HEATER, LOW);
        if (heat == true){
            heat = !heat;
        }
        water = water + 1;
    }
    if ((water > 0) && (tempture1 < 71)){
       PUMPstart();
    }
}

//this is the keep warm subroutine which is to control the heating element to keep around 50 C
void KeepHeater(){
    Serial.println(154);
    sensors.requestTemperatures();
```

```
    tempture1 = sensors.getTempC(RTD1);
    lcd.setCursor(3, 2);
    lcd.print("Temperature is");
    lcd.setCursor(7, 3);
    lcd.print(tempture1);
    lcd.print(" C");
    delay(1000);
    if ((tempture1 <49) && (heat == false)){
        digitalWrite(HEATER, HIGH);
        heat = !heat;
    }
    else if (tempture1 > 50) {
        digitalWrite(HEATER, LOW);
        if (heat == true){
            heat = !heat;
        }
    }
}

//this is the Pause subroutine when it is pressed, all process are paused.
void PauseSET() {
  if (start == true) {
    start = !start;
```

```
  }
  digitalWrite(HEATER, LOW);
  while (start == false){
    recive();
  }
  pause = !pause;
  if (heat == true){
     heat = !heat;
  }
}

//This is restart subroutine.
void RestartSET() {
  restart = !restart;
  a = 0;
  b = 0;
  c = 0;
  tempture1 = 0;
  p = 0;
  water = 0;
  buttonState = 0;
  digitalWrite(HEATER, LOW);
  if (strong == true){
```

```
    strong = !strong;
  }
  if (motormove == true){
    motormove = !motormove;
  }
  if (heat == true){
    heat = !heat;
  }
  if (start == true){
    start = !start;
  }
  lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print("Welcome to Smartbrew");
}
```

Appendix E

Boolean pump

Boolean motor

Boolean motor2

Boolean Afterrelease

Boolean completion

Boolean start

Boolean pause

Boolean Restart

Boolean one

Boolean two

Boolean three

Boolean Four

Boolean Lowwater

Boolean heatingpause

Int input1

Int input2

Int input3

Int input4

Int Totaltime

Int temperature

String return

Int count

Int add1

Int add2

Int add3

Int Secondinverted

```
Main event mainwindow.created ()
        Pump = false
        Motor = false
        Motor 2 = false
        Afterrelease = false
        completion = False
        Start = False
        Pause = False
        Heatingpause = false
        Restart = false
        one = false
        Two = false
        Three = false
        Four = false
        Lowwater = false
        Input1 = 0
        Input2 = 0
        Input3 = 0
        Input4 = 0
        Totaltime = 0
        Temperature = 0
        Count = 0
        add1 = 0
        add2 = 0
        add3 = 0
        Secondinverted = 0
      return = ""
```

```
        If Bluetooth1 is turned on () = false then
                Bluetooth 1. TurnonBluetooth ()
        End If
End Event


Event mainwindow. started (String startupparameter)
        If the startupparameter = "wake up" the
           SystemSetup1. Alwayskeepscreen ()
               SystemSetup1. screenunlock ()
        End If
End Event


Event Bluetooth 1. Bluetooth is set up (Int setto)
    Switch setto
            Branch 1
                Messagebox ( "Bluetooth already open")
                Bluetooth1. canfound()
            Branch 2
                Messagebox ( "message", "Bluetooth not open, Exit", "ok")
                Me.close ()
            Branch 3
                Messagebox ( "Bluetooth avaliable")
            Branch 4
                Messagebox ( "message", "Bluetooth not avaliable, Exit", "ok")
                Me.close ()
    End Switch
```

```
        Bluetooth1. Setmode (1)
End Event


event button1.clicked ()
        Bluetooth1. Searchdevices ()
        Messagebox ( "Searching")
End Event


event Bluetooth1. devicediscovery (String devicename, String address, Boolean paired)
        listbox1. Visible= True
        listbox1.items.Add (device name & "_" & Device Address & "_" & paired) '
End Event


event listbox1.clicked (Int index)
        String deviceinformation
        String Textarray()
        String deviceaddress
        deviceinformation = listbox1.item.read(index)
        Textarray = split (deviceinformation, "_")
        deviceaddress = Textarray (1)
        Messagebox (device address)
     Bluetooth1.Connectdevice (deviceaddress)
        Messagebox ( "Connecting")
End Event
```

```
event Bluetooth1. connection (Boolean result, String devicename, String deviceaddress, Int mode)
    If result = true
            Listbox1. visible = false
            Mainwindow.text = "Connect:" & Device Name
            messagebox ( "Connect Successful")
            Button6. Visible = true
            Button7. Visible = true
            Button8. Visible = true
            Button9. Visible = true
    else
            messagebox ( "Connect error")
    End If
End Event


Event Bluetooth1.receive (byte data(), String devicename, String deviceaddress)
    return = return & bytetoString (data, "UTF8")
    If text.readright (return, 1) = "\ n"
       Switch text.readleft(return, lengh(return)-2)
            Branch "151"
                Motor = true
                mainwindow. text = "MOTOR OVER"
            Branch "152"
                Pump = true
                mainwindow. text = "PUMP OVER"
            Branch "154"
                completion = true
```

```
            Others
                temperature = text.readleft (return, lengh(return)-2)
        End Switch
        If temperature> 100
            temperature = 100
        End If
            return = ""
    End If
End Event


event button2.clicked ()
    Bluetooth1.send (Stringtobyte ("119X", "UTF8"))
    Bluetooth 1. Disconnect ()
    Me.close ()
End Event


event button9.clicked ()
    Input1 = 0
    Input2 = 0
    Input3 = 0
    Input4 = 0
    button6.visible = false
    button7.visible = false
    button8.visible = false
    button9.visible = false
    button3.visible = true
```

```
    button5.visible = true
    lable3.visible = true
    lable4.visible = true
    lable5.visible = true
    lable6.visible = true
    textbox1.visible = true
    textbox2.visible = true
    textbox3.visible = true
    textbox4.visible = true
    selectbox1.visible = true
    selectbox2.visible = true
    lable1.visible = true
    lable2.visible = true
End Event


event button8.clicked ()
    Input1 = 30
    Input2 = 120
    Input3 = 0
    Input4 = 0
     add1 = 59 + input1
    add2 = add1 + Input2
    add3 = add2 + Input3
    button6.visible = false
    button7.visible = false
    button8.visible = false
    button9.visible = false
```

```
    button3.visible = true
    button5.visible = true
    selectbox1.visible = true
    selectbox2.visible = true
    lable1.visible = true
    lable2.visible = true
End Event


event button7.clicked ()
    Input1 = 20
    Input2 = 30
    Input3 = 15
    Input4 = 60
     add1 = 59 + input1
    add2 = add1 + Input2
    add3 = add2 + Input3
    button6.visible = false
    button7.visible = false
    button8.visible = false
    button9.visible = false
    button3.visible = true
    button5.visible = true
    selectbox1.visible = true
    selectbox2.visible = true
    lable1.visible = true
    lable2.visible = true
End Event
```

```
event button6.clicked ()
    Input1 = 20
    Input2 = 90
    Input3 = 0
    Input4 = 0
     add1 = 59 + input1
    add2 = add1 + Input2
    add3 = add2 + Input3
    button6.visible = false
    button7.visible = false
    button8.visible = false
    button9.visible = false
    button3.visible = true
    button5.visible = true
    selectbox1.visible = true
    selectbox2.visible = true
    lable1.visible = true
    lable2.visible = true
End Event


event button3.clicked ()
    If textbox1.visible = true
        If textbox1.text = 0
            messagebox( "Please, enter the time in first high heat")
             else
```

```
If textbox2.text = 0
      messagebox( "Please, enter the time in first slow heat")
           else
       If textbox3.text = 0
                              Input2 = Input2 + Input4
           Input4 = 0
           clock2. clockcycle = 1
           Start = True
           add1 = 59 + input1
           add2 = add1 + Input2
           add3 = add2 + Input3
           Totaltime = add3 + input4
           Secondinverted = Totaltime -10
           textbox1.visible = false
           textbox2.visible = false
           textbox3.visible = false
           textbox4.visible = false
           lable3.visible = false
           lable4.visible = false
           lable5.visible = false
           lable6.visible = false
           button3.visible = false
           button2.visible = false
           Systemalarm1. Alarmcycle (1, 1000, "wake up")
           If lowwater = true
                Bluetooth1.send (stringtobyte ( "116X", "UTF8"))
           else
                Bluetooth1.send (stringtobyte ( "117X", "UTF8"))
```

```
            End If
        else
            clock2. clockcycle = 1
            Start = True
            add1 = 59 + input1
            add2 = add1 + Input2
            add3 = add2 + Input3
            Totaltime = add3 + input4
            Secondinverted = Totaltime -10
            textbox1.visible = false
            textbox2.visible = false
            textbox3.visible = false
            textbox4.visible = false
            lable3.visible = false
            lable4.visible = false
            lable5.visible = false
            lable6.visible = false
            button3.visible = false
            button2.visible = false
            Systemalarm1. Alarmcycle (1, 1000, "wake up")
            If lowwater = true
                Bluetooth1.send (stringtobyte ( "116X", "UTF8"))
            else
                Bluetooth1.send (stringtobyte ( "117X", "UTF8"))
            End If
        End If
    End If
End If
```

```
else
    If start = false
        clock2. clockcycle = 1
        Start = True
        add1 = 59 + input1
        add2 = add1 + Input2
        add3 = add2 + Input3
        Totaltime = add3 + input4
        Secondinverted = Totaltime -10
        button3.visible = false
        button2.visible = false
        Systemalarm1. Alarmcycle (1, 1000, "wake up")
        If lowwater = true
            Bluetooth1.send (stringtobyte ( "116X", "UTF8"))
        else
            Bluetooth1.send (stringtobyte ( "117X", "UTF8"))
        End If
    else
        button3.visible = false
        button4.visible = true
        Pause = False
        Heatingpause = false
        If lowwater = true
            Bluetooth1.send (stringtobyte ( "116X", "UTF8"))
        else
            Bluetooth1.send (stringtobyte ( "117X", "UTF8"))
        End If
    End If
```

```
    End If
End Event


event button4.clicked ()
    Bluetooth1.send (stringtobyte ( "118X", "UTF8"))
    Pause = true
    Heatingpause = true
    Clock1. clockcycle = 0
    button3.visible = true
    button4.visible = false
End Event

event button5.clicked ()
    Bluetooth1.send (stringtobyte ( "119X", "UTF8"))
    Pump = false
    Motor = false
    Motor 2 = false
    Afterrelease = false
    completion = False
    Start = False
    Pause = False
    Heatingpause = false
    Restart = false
    one = false
    Two = false
    Three = false
    Four = false
```

```
Lowwater = false
Input1 = 0
Input2 = 0
Input3 = 0
Input4 = 0
Totaltime = 0
Temperature = 0
Count = 0
add1 = 0
add2 = 0
add3 = 0
Secondinverted = 0
return = ""
button1.visible = true
button2.visible = true
button3.visible = false
button4.visible = false
button5.visible = false
button6.visible = true
button6.useable = true
button7.visible = true
button7.useable = true
button8.visible = true
button8.useable = true
button9.visible = true
button9.useable = true
button10.visible = false
selectbox1.visible = false
```

```
        selectbox2.visible = false
        lable1.visible = false
        lable2.visible = false
        lable3.visible = false
        lable4.visible = false
        lable5.visible = false
        lable6.visible = false
        textbox1.visible = false
        textbox2.visible = false
        textbox3.visible = false
        textbox4.visible = false
        textbox1.text = ( "0")
        textbox2.text = ( "0")
        textbox3.text = ( "0")
        textbox4.text = ( "0")
         listbox1.items.clear()
         lable2.text = ( "Not start to heat")
       lable1.text = ( "0 mins")
       Mainwindow.text = ( "SELECT THE BREW PROFILE")
       systemalarm1. Destruction (1)
       Clock2. clockcycle = 0
       Clock1. clockcycle = 0
End Event


event button10.clicked ()
       systemalarm1. Destruction (1)
       Clock2. clockcycle = 0
```

```
    Bluetooth1.send (stringtobyte ( "119X", "UTF8"))
    Bluetooth1. Disconnect ()
        Me.close()
End Event


event clock1.clockcycle ()
    Count = count + 1
    lable1.text = (Integertostring(count) & "mins")
End Event


event clock2.clockcycle ()
    If count > 59
        lable2.text = (Integer to text (temperature) & "C")
    else
        lable2.text = ( "Not start to heat")
    End If
    If count > Secondinverted
        If motor2 = true
            Bluetooth1.send (stringtobyte ( "130X", "UTF8"))
        End If
    End If
    If pump = true
        If pause = false
            Button 4. visible = true
            Button 5. visible = true
            If Clock1. clockcycle = 0
```

```
            Clock1. clockcycle = 6000
        End If
    else
        Button 4. visible = false
            If Clock1. clockcycle = 6000
                Clock1. clockcycle = 0
            End If
    End If
End If
If input3 = 0
    If count > Totaltime
        if three= False
            Bluetooth1.send (stringtobyte ( "122X", "UTF8"))
            Clock1. clockcycle = 0
            Count = 0
            Pause = true
            mainwindow. Text = "FINISH"
            completion = true
            Three = true
        End If
    Else
        If count > add1
            If Two = false
                Bluetooth1.send (stringtobyte ( "121X", "UTF8"))
                Clock1. clockcycle = 0
                Pause = true
                mainwindow. Text = "HEAT TO SLOW HEAT"
                Two = true
```

```
            End If
        Else
            If count > 59
                                if one = false
                Bluetooth1.send (stringtobyte ( "120X", "UTF8"))
                Clock1. clockcycle = 0
                Pause = true
                mainwindow. Text = "HEAT TO HIGH HEAT"
                one = true
                End If
            End If
        End If
    End If
else
    If count > Totaltime
        If completion = False
            Bluetooth1.send (stringtobyte ( "122X", "UTF8"))
            Clock1. clockcycle = 0
            Count = 0
            Pause = true
            mainwindow. Text = "FINISH"
            completion = true
        End If
    else
        If count > add3
            If Four = false
                Bluetooth1.send (stringtobyte ( "121X", "UTF8"))
                Clock1. clockcycle = 0
```

```
            Pause = true
            mainwindow. Text = "HEAT TO SLOW HEAT"
            Four = true
        End If
    Else
        If count > add2
            if Three = false
                Bluetooth1.send (stringtobyte ( "120X", "UTF8"))
                Clock1. clockcycle = 0
                Pause = true
                mainwindow. Text = "HEAT TO HIGH HEAT"
                Three = true
            End If
        Else
            If count > add1
                If Four = false
                    Bluetooth1.send (stringtobyte ( "121X", "UTF8"))
                    Clock1. clockcycle = 0
                    Pause = true
                    mainwindow. Text = "HEAT TO SLOW HEAT"
                    Two = true
                End If
            else
                if one = false
                    Bluetooth1.send (stringtobyte ( "120X", "UTF8"))
                    Clock1. clockcycle = 0
                    Pause = true
                    mainwindow. Text = "HEAT TO HIGH HEAT"
```

```
                                one = true
                            End If
                        End If
                    End If
                End If
            End If
        End If
    End If
    If temperature > 88 and heatingpause = false
        If one = true and Two = false
            If Clock1. clockcycle = 0
                Clock1. clockcycle = 6000
                Pause = False
            End If
        End If
    End If
    If temperature < 82 and heatingpause = false
        If Two = true and Three = false
            If Clock1. clockcycle = 0
                Clock1. clockcycle = 6000
                Pause = False
            End If
        End If
    End If
    If temperature > 88 and heatingpause = false
        If Three = true and Four = false
            If Clock1. clockcycle = 0
                Clock1. clockcycle = 6000
                Pause = False
```

```
            End If
        End If
End If
If temperature < 82 and heatingpause = false
        If Four = true
            If Clock1. clockcycle = 0
                Clock1. clockcycle = 6000
                Pause = False
            End If
        End If
End If
If completion = true
        Pump = false
        Motor = false
        Motor 2 = false
        Afterrelease = false
        completion = False
        Start = False
        Pause = False
        Heatingpause = false
        Restart = false
        one = false
        Two = false
        Three = false
        Four = false
        Lowwater = false
        Input1 = 0
        Input2 = 0
```

```
        Input3 = 0
        Input4 = 0
        Totaltime = 0
        Temperature = 0
        Count = 0
        add1 = 0
        add2 = 0
        add3 = 0
        Secondinverted = 0
        return = ""
    End If
End Event


event textbox1.changed (String newcontent)
    Input1 = textbox1. text
End Event


event textbox2.changed (String newcontent)
    Input2 = textbox2. text
End Event

event textbox3.changed (String newcontent)
    Input3 = textbox3. text
End Event

event textbox4.changed (String newcontent)
```

```
    Input4 = textbox4. text
End Event
event selectbox1.changes ()
    If selectbox1. Select = true
        Motor2 = true
    else
        Motor2 = false
    End If
End Event


event selectbox2.changes ()
    If selectbox2. Select = true
        Lowwater = true
    else
        Lowwater = false
    End If
End Event
```

Appendix F

Arduino Program

```
#include <Wire.h>
int p;

void setup() {
Serial.begin(9600);
pinMode (13,OUTPUT);    // Physical LED
pinMode (12,OUTPUT);    // Common LED
pinMode (11,OUTPUT);    // Supplement LED
pinMode (10,OUTPUT);    // Custom LED
pinMode (9,OUTPUT);     // Pause LED
pinMode (8,OUTPUT);     // Resume LED
}

void loop() {
  while (Serial.available() > 0) {
    p = Serial.parseInt();
    if (Serial.read() == 'X') {
      switch (p) {
        case 101:
          digitalWrite(13, HIGH);   // Physical is pressed
          Serial.println(51);
          break;
        case 102:
          digitalWrite(12, HIGH);   // Common is pressed
          Serial.println(52);
          break;
```

```
case 103:
  digitalWrite(11, HIGH);   // Supplement is pressed
  Serial.println(53);
  break;
case 104:
  digitalWrite(10, HIGH);   // Custom is pressed
  Serial.println(54);
  break;
case 105:
  digitalWrite(9, HIGH);    // Pause is pressed
  Serial.println(55);
  break;
case 106:
  digitalWrite(8, HIGH);    // Resume is pressed
  Serial.println(56);
  break;
case 107:
  digitalWrite(13, LOW);    // Physical is NOT pressed
  Serial.println(57);
  break;
case 108:
  digitalWrite(12, LOW);    // Common is NOT pressed
  Serial.println(58);
  break;
case 109:
  digitalWrite(11, LOW);    // Supplement is NOT pressed
  Serial.println(59);
  break;
```

```
      case 110:
        digitalWrite(10, LOW);    // Custom is NOT pressed
        Serial.println(60);
        break;
      case 111:
        digitalWrite(9, LOW);     // Pause is NOT pressed
        Serial.println(61);
        break;
      case 112:
        digitalWrite(8, LOW);     // Resume is NOT pressed
        Serial.println(62);
        break;
      default:
        p = map(p,0,100,0,255);
        analogWrite(11,p);
    }
  }
 }
}
```

Appendix G

Arduino Program:

```
// Library Codes:
#include <DallasTemperature.h>
#include <LiquidCrystal.h>
float a;
bool check = false;
#define ONE_WIRE_BUS 7



// initialized the LC Library
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

OneWire oneWire (ONE_WIRE_BUS); // on digital pin 7
DallasTemperature sensors(&oneWire);
DeviceAddress insideThermometer = {0x28, 0xFF, 0xAD, 0x5E, 0x90, 0x15, 0x03, 0x83};

void setup() {
  // initial thermistor
  Serial.begin(9600);
  sensors.begin();
  sensors.setResolution(insideThermometer, 10);

 // set LCD number of columns and rows:
 lcd.begin (20,4);
 //Print a message to the LCD
 lcd.print("SPEC TEST 2");
 pinMode(13, OUTPUT);   // pin 13 to set LED indicator
```

```
}
void loop() {
  // set initial temperature
  sensors.requestTemperatures();
  if (check == false) {
    a = sensors.getTempCByIndex(0);
    check = !check;
  }
  // allocate current temperature in memory
  float temp = 0;
  // set the cursor to column 0, line 1
  lcd.setCursor(0, 1);
  lcd.print("Temp is: ");
  lcd.print(sensors.getTempCByIndex(0));
  temp = sensors.getTempCByIndex(0);
  lcd.print(" C.");
  // check water level with temperature
  if (temp == a) {
    digitalWrite(13, HIGH);
    lcd.setCursor(0, 2);
    lcd.print("ADD Water");
  }
   else if ((temp < (a - 0.5)) || (temp > (a + 0.5))){
    digitalWrite(13, LOW);
    lcd.setCursor(0, 2);
    lcd.print("Water Level Reach");
  }
}
```

Appendix H

Arduino Code:

```
#include <Wire.h>
int p;
bool motor = false;

void setup() {
Serial.begin(9600);

pinMode (13,OUTPUT);    // Physical LED
pinMode (12,OUTPUT);    // Common LED
pinMode (11,OUTPUT);    // Supplement LED
pinMode (10,OUTPUT);    // Custom LED
pinMode (9,OUTPUT);     // Pause LED
pinMode (8,OUTPUT);     // Resume LED
pinMode (7,OUTPUT);
pinMode (6,OUTPUT);
pinMode (5,OUTPUT);
}

void loop() {
  while (Serial.available() > 0) {
    p = Serial.parseInt();
    if (Serial.read() == 'X') {
      switch (p) {
        case 116:
          if (motor == false){
              digitalWrite(13, HIGH);
```

```
      delay(5000);
      Serial.println(151);
      delay(5000);
      Serial.println(152);          // Physical is pressed
      digitalWrite(13, LOW);
      motor = !motor;
   }
   digitalWrite(6, HIGH);
   digitalWrite(11, LOW);
   break;
 case 117:
   if (motor == false){
      digitalWrite(12, HIGH);
      delay(5000);
      Serial.println(151);
      delay(5000);
      Serial.println(152);          // Physical is pressed
      digitalWrite(12, LOW);
      motor = !motor;
   }
   digitalWrite(6, HIGH);
   digitalWrite(11, LOW);
   break;
 case 118:
   digitalWrite(11, HIGH);   // Supplement is pressed
   digitalWrite(6, LOW);
   break;
 case 119:
```

```
  digitalWrite(10, HIGH);
  delay(5000);
  digitalWrite(13, LOW);
  digitalWrite(12, LOW);
  digitalWrite(11, LOW);
  digitalWrite(10, LOW);
  digitalWrite(9, LOW);
  digitalWrite(8, LOW);
  digitalWrite(7, LOW);
  digitalWrite(6, LOW);
  digitalWrite(5, LOW);
  if (motor ==true) {
    motor = !motor;
  }
  break;
case 120:
  digitalWrite(9, HIGH);
  digitalWrite(8, LOW);
  delay(5000);
  Serial.println(100);
  break;
case 121:
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  delay(5000);// Resume is pressed
  Serial.println(80);
  break;
case 122:
```

```
        digitalWrite(7, HIGH);
        digitalWrite(9, LOW);
        digitalWrite(8, LOW);  // Physical is NOT pressed
        Serial.println(154);
        break;
      case 130:
        digitalWrite(5, HIGH);
        delay(5000);
        digitalWrite(5, LOW);// Common is NOT pressed
        break;
      default:
        p = map(p,0,100,0,255);
    }
  }
 }
}
```

Appendix I

Arduino Code:

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(24, 25, 29, 28, 27, 26);
#define ONE_WIRE_BUS 2
OneWire oneWire (ONE_WIRE_BUS); // on digital pin 2
DallasTemperature sensors(&oneWire);
DeviceAddress RTD1 = {0x28, 0xFF, 0xE3, 0x0E, 0x91, 0x15, 0x01, 0x07};
void setup() {
  Serial.begin(9600);
  lcd.begin(20, 4);
  sensors.begin();
  sensors.setResolution(RTD1, 10);

  pinMode(52, OUTPUT);
  pinMode(7, OUTPUT);
}

void loop() {
    float tempture1;
    sensors.requestTemperatures();
    tempture1 = sensors.getTempC(RTD1);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(tempture1);
    if ((tempture1 > 25) && (tempture1 < 100)) {
```

```
    digitalWrite(7, HIGH);
    digitalWrite(52, HIGH);
  }
  else if (tempture1 > 100){
    digitalWrite(7, LOW);
    digitalWrite(52, LOW);
    delay(5000);
  }
  else{
    digitalWrite(7, LOW);
    digitalWrite(52, HIGH);
  }
}
```