**ECE445**
Senior Design Laboratory
Final Report

# Smart Kettle Module

**Spring 2021 Team#58**
Owen Sun
Boxiang Zhao
Naomi Peng

TA: Dean Biskup

## Contents:

# 1. Introduction

## 1.1 Background

Our idea comes from our own experiences in daily life. Most people also have the same experience waiting in front of a water kettle to refill and boil water. Take one of our group members' electric kettle as an example, a 2-liter stainless steel Hamilton Beach electric kettle. It takes him 2 minutes to let the kettle fill the water and 10 minutes to boil water. If he decides to make tea with boiling water, that would be another 10 minutes for the tea to cool down at a servable temperature. To be more intuitive, "The average salary in the UK is now £26,500, which equates to £14.56 an hour for a standard 35 hour week. This means that if a typical worker makes six cups of tea or coffee in a day and spends three and a half minutes away from their desk for each cup of tea, it all adds up to 21 minutes a day and a staggering one and three-quarters of an hour over the whole week. That is £25.50 a week in lost time."[1]

We are not the only group that comes up with the smart electrical kettle idea. A website called Smarthomebit already predicted the best-selling smart kettle for 2021. In its opinion, the overall best smart kettle is called Smarter iKettle 3rd Generation. Its signature features are operation "at a diverse temperature ranging from 68-212 degrees Fahrenheit" [2], remote accessing, and preparing the steaming water at a certain time. Instead of doing some modifications on the kettle, we designed the smart kettle module that can be adapted on many different types of electrical kettle.

Our project can also pre-set different temperature levels and keep the kettle at that temperature for a certain amount of time, no longer than one hour. One of the advantages of our product is minimizing manual operation. We combined adding water, boiling water, temperature controlling, data recording all in one place. For adding water, the weight sensor will detect the water level in our kettle. As long as the water volume is less than 20 percent of the kettle-containing volume, the kettle will be refilled to a certain water level from water pumped out of the water tank. Each time, after our product automatically adds water in, the kettle starts to boil water to 100 degrees Celsius. After the water boils, the module goes to temperature control mode. Water will be cooled down to our setting temperature for a set amount of time, and the limitation is one hour. As previously mentioned, our product has four stages: Adding water stage, boiling water stage, cooling down the stage, and keeping warm stage. The LCD panel in our product is completely responsible for displaying data recorded. It will show us what stage the product is at, how long it stays in this stage, how much time is left, and the basic measurement of the water in the kettle like temperature and weight.

## 1.2 Problems and Solutions

Although there are already some types of smart kettles existing in the market [3], these kettles' functionalities are not ideal enough. The kettles require the users to stand aside and wait until the kettle is filled; otherwise, the kettle might be overfilled. Next, even though the kettles would make noise during the boiling water period and keep quiet after water is boiled, it is not informative enough to remind the users if the water is ready. There are scenarios when the users go back to the kettle and see the water is not ready yet. They will either wait next to the kettle until the water is ready or go back to work. Both ways are a waste of time. Since people nowadays live a fast-paced life[4] and try to improve the efficiency of everything [5], it would be ideal if we can add some new features to the kettles that can save us time from boiling water.

We propose a kettle-module that has three main features which provide a better user experience: 1. The module enables the kettle to be filled with water automatically. Weight sensors would be implemented and control the amount of water to prevent overflow. 2.To avoid the situations when users get back to the kettle and see the water has not fully boiled yet, the kettle-module would display the remaining time required for the water to reach the target temperature on an LCD screen. 3.The kettle-module would introduce a new way to keep warm. Unlike the traditional way of keeping warm, which maintains the temperature at a specific level, our kettle-module will not reheat the water until it is five °C lower than the target temperature. The time of keeping warm is also controllable.

## 1.3 Block Diagram

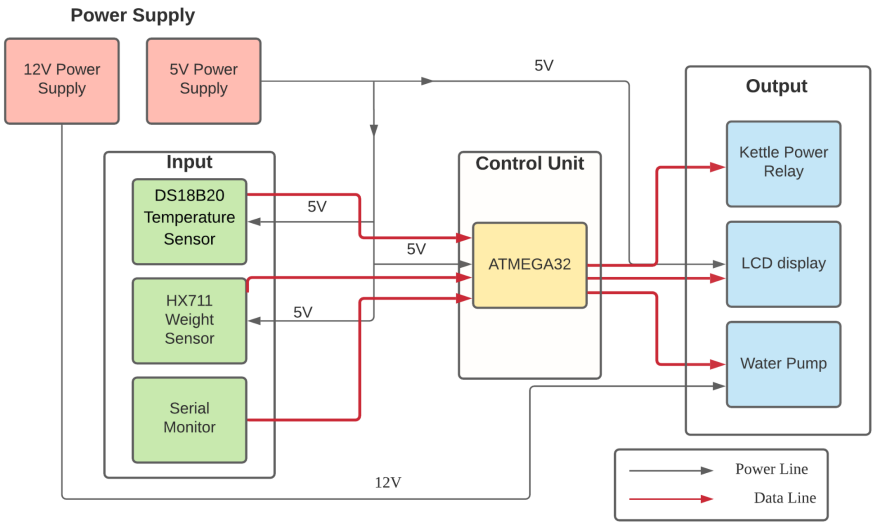



Figure1. Block Diagram

We split the project into 4 parts. There are four subsystems. They are Power Supply, Control Unit, Input, and Output. The 5V power supply is used to power our PCB, sensors, and the LCD screen. The 12V power supply is mainly used to power the water pump for high flow rate. The input subsystem consists of the real time data coming from the temperature and weight sensor. The temperature, water level, and keep warming time can be selected according to user preferences via serial monitor. Then the datas would be sent to the control unit for the algorithm. The output subsystem would receive the signals coming from the control unit and the kettle and water pump would start working when they receive a 1 from the control unit. The LCD would display the data from the sensors and control unit as well.

## 2. Design

### 2.1 Physical design & Actual project

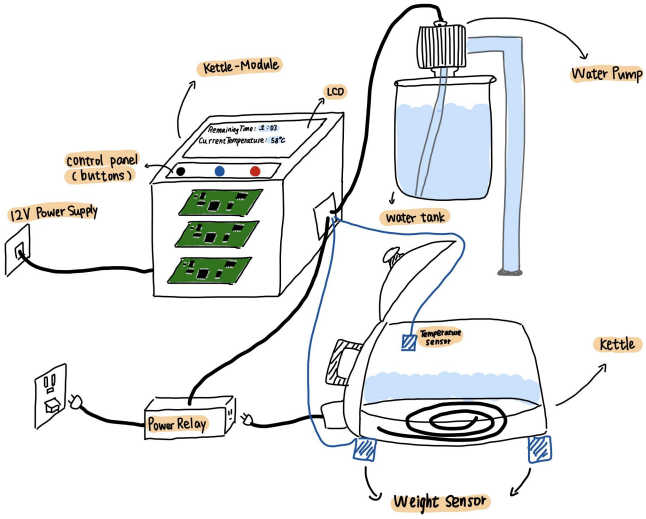



Figure2. Physical design

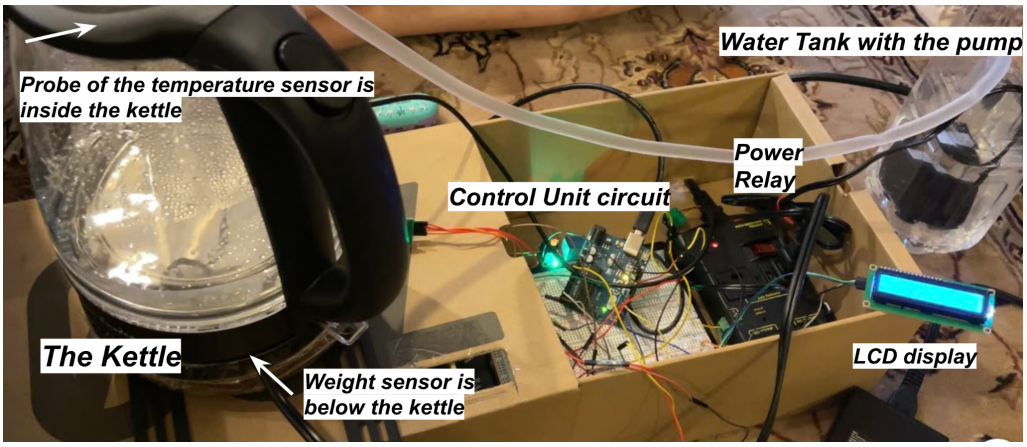



Figure3. Actual Project

## 2.2 Design procedure

### 2.2.1 Alternative Approaches

- ***Control Unit:*** **1 vs. 2 Atmega328 chips** [6]
  At the beginning of the design, we discussed whether we should use one or two atmega328 chips. Our first thought was that it would be easier to write the codes for the control unit if we were going to use two atmega328 chips because we will not need to worry about the overlapping ports.

- ***Power Supply:***
  Our first draft of the proposal suggests that a 12V battery pack would power the system. Multiple voltage regulators will be used to provide 5V for sensors and ICs. Nevertheless, considering the sustainability of the system, we decided to power the water pump and SoC separately, using a12V and a 5V power supply, which will be plugged into the power relay. This does not only simplify the control of power supply and circuit design, but it also prevents safety hazards, like heating without water in the kettle.

- ***Input:*** **Push buttons vs. Type inputs on keyboard**
  Based on our own experiences, we believed that it would be reasonable and convenient to have the push buttons on the module for the users to choose the desired functions, temperature, and time range. However, when we are designing the circuits on the breadboard and at the same time writing the codes, we realized that since we have to do the calibration for the temperature sensor HX711 every time we reboot the system on the laptop, it might be more convenient to let the users select the desired temperature and time range on the laptop. The other advantage of typing inputs on the laptop is that with reasonable wires arrangement, the users can get the kettle work without heading to the kettle. For example, they can type in the desired temperature on their laptop in the living room, and the kettle in the kitchen would start working. We thought if we would continue to work on this project in the future, we planned to develop an app that allows the users to control the kettle remotely to fit the name “Smart” more.

- ***Output:*** **Water Tank vs. Faucet**
  Although it seems more reasonable to use the faucet directly when filling water, there are two other aspects we took into consideration. The first one is that if we connect the faucet directly to the kettle, the faucet would be restricted to only available for adding water to the kettle. In real life, we believe this would decrease the efficiency of the faucet. In addition, people nowadays pursue a healthier lifestyle, and most of them prefer purified water for several reasons. “Drinking purified water assures that the vital minerals you need to function are present in the water you are drinking. Purified water removes harmful bacteria that otherwise can lead to sickness, stomach pain and nausea” [7]. It is possible that most people would prefer to boil the water from the tank that is filled with

purified water instead of boiling the water that comes from the faucet. Thus, we chose to use the water tank instead of the faucet after consideration.

### 2.2.2 Major Design Equation

1. Firstly, we acquire water amount via the weight sensor and temperature difference via the temperature sensor.

$$\Delta T = T_{target} - T_{initial} \qquad \text{(Equation 2.2.2.1)}$$

2. Then we calculate the energy needed to heat water to the target temperature. C is the specific heat of water (4.186 joule/gram °C). m is the mass of water.

$$Q=cm\Delta T \quad \text{(Equation 2.2.2.2)}$$

3. Finally, we can calculate the remaining time using the measured power level and energy needed. P is the power level of the kettle. Considering that the kettle might not be working at a constant efficiency, we will use the real-time power level for calculation in order to get more accurate remaining time.

$$t_{remaining} = Q/P \quad \text{(Equation 2.2.2.3)}$$

Though the temperature change may lead to a change of specific heat of water, we chose to ignore the error since the calculation accuracy was not influenced obviously.

### 2.2.3 PCB Design

We tried to make the system modular. So the PCB only contains necessary components for ATMEGA328, including two 22pF capacitors and a 16MHz crystal. The rest of the pins are connected to connectors directly. A 5V power supply powers the PCB via a barrel jack connector.

Our initial idea was to make a simple version of Arduino UNO. That crystal oscillator soldered on the Arduino acts as an internal clock for the Arduino. The middle pin of it is connected to the ground to save space, and the internal capacitors can be better matched to the crystal so that it gives a more accurate frequency.

Besides, two 22pF capacitors for the oscillator are also necessary for the PCB design. According to the ATMEGA328 datasheet, the optimal value of the capacitors depends on the crystal or the resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment.

Each crystal is cut to oscillate with a specific load capacitance - if the capacitance is a different value, the frequency will be out. Two 22pF capacitors in series plus a 9pF stray capacitance equal 20pF.

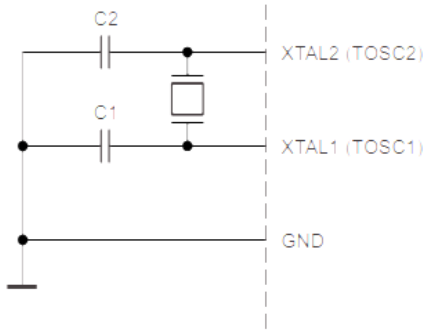



Figure4. Crystal Oscillator Connections

The barrel jack connector allows us to directly power the PCB using a 5V power supply instead of a voltage regulator or step-down converter. All the pins of ATMEGA328 are assigned to corresponding connectors for communication with sensors and LCD screens. Furthermore, we also left the RX and TX pin connectors on the PCB for uploading codes to the microcontroller from Arduino UNO.

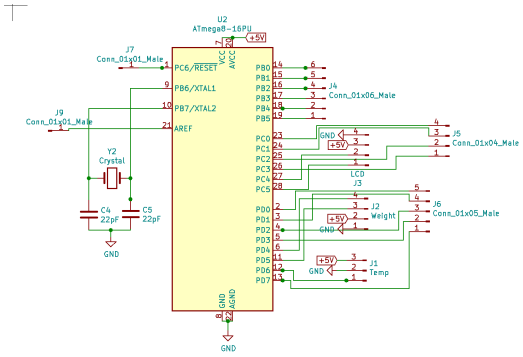
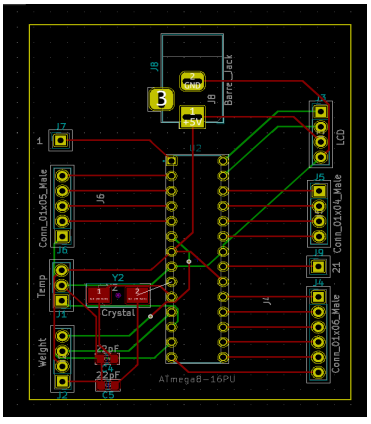


Figure5. PCB design



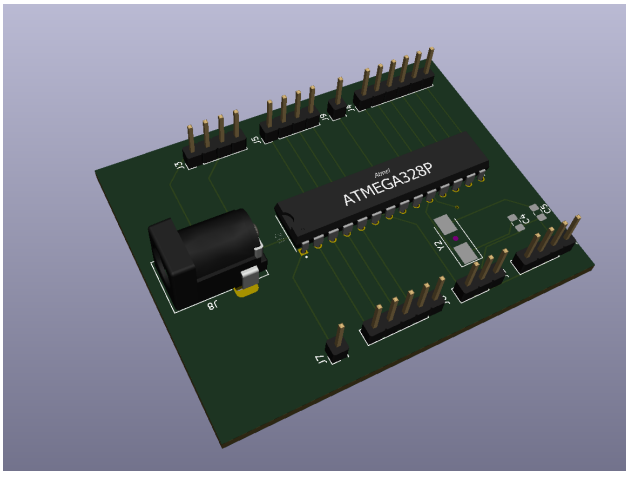



Figure6. & Figure7. PCB design

### 2.2.4 RESET Button (Debounce Switch Circuit)

For the RESET function, instead of using a push button, which we believe would be less stable because it is easy to touch the button unintentionally, and meanwhile, the signal is not constant, we chose to use a debounce switch to replace the push button.

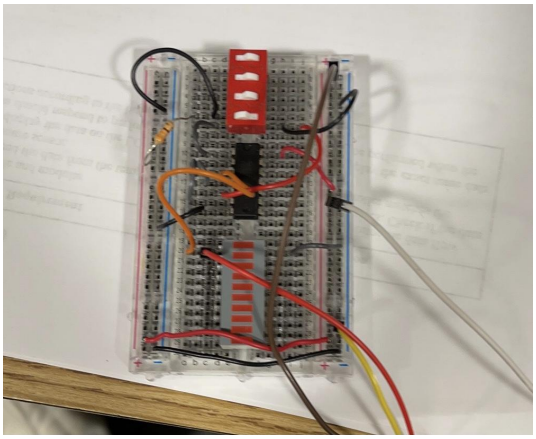


Figure8. Debounce switch circuit for the RESET

## 2.3 Design Details

### 2.3.1 Hardware Design

The main idea of the design is modular. All the peripherals (including the two sensors, LCD screen, and water pump) are not soldered to the system. This is for easy replacement once the component is malfunctioning. The water tank is placed beside the module. Moreover, the size of the water tank can be adjusted according to different needs. The reason we chose soft tubing for water refilling is due to better compatibility. Soft tubings are also easier to install and store compared to hard tubing. All the power supplies are integrated into the large power relay. So there are no extra wires and plugs needed. Since the project is heating the water, we can not have exposed components. All the components are contained in a water-resist box, so the water spill will not harm the stability and functionality of the whole system.

**Input:**

1. DS18B20 Temperature Sensor [8]: The sensor is used to measure water temperature from 0 - 100°C. It also needs to be waterproof. So we chose the DS18B20 with the waterproof probe.
2. HX711 Weight Sensor [9]: This sensor receives analog signals and converts them to digital via the ADC on the chip. Also, the sensor comes with a loading cell and tray. We chose this sensor because it is easy to be installed.

**Power Supply:**

1. IoT Power Relay: This is a power cord that switches 120 VAC (up to 15 amps) with a DC control voltage of 3 VDC @ 3mA to 12 VDC @ 30 mA. The reason we chose this large

power relay is due to safety concerns. This power relay has many safety features, including eliminating shock hazards and preventing relay chatter. With four power outlets on the relay, we can power the whole system using one power relay. No extra wires and plugs are needed.

2. 5V Power Relay: We install the 5V power relay on the power line of the water pump. The control of the two relays is the same. We use this relay because we want to make sure of the isolation of each output component. The water pump will not be working while the kettle is heating and vice versa.

**Output:**

1. LCD screen: The LCD screen came with an I2C module, so we only need to connect Vcc, GND, SDA, and SCL instead of all 14 pins. The backlight can be adjusted according to different needs. We chose this screen as a display because LCD screens with I2C modules are easy to be programmed. Furthermore, the readings are very intuitive.
2. Water Pump [10] : We initially tried to unify the power supply voltage and bought a 5V water pump. However, the flow speed did not meet our requirement (fully refill the kettle within 30 seconds). So we changed it to a 12V water pump, which has a flow speed of 160GPH (Gallon per hour). This water pump can refill the kettle within 10 seconds. It also came with a power cord, which simplified our power control.

### 2.3.2 Software Design

The control unit let us implement all the functionalities we stated in our proposal and demonstrated in our official demo. The control unit includes the software part and hardware part. We used a UNO Arduino with an Atmega328 chip on it. A debounce switch acts as a reset button for our hardware part, and Arduino IDE code to implement all functionalities illustrated in the following flowchart software-wise.

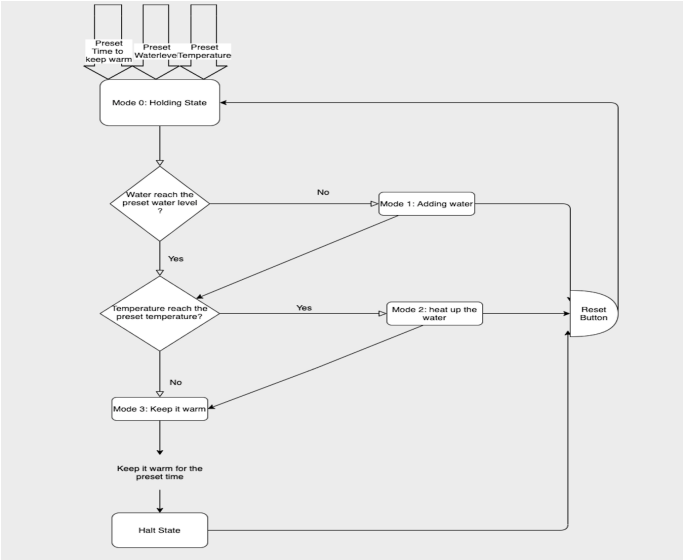



Figure 9. Flow Chart

The basic writing style of the Arduino IDE code is very similar to C++, but it has its uniqueness. C++ has one main function called "int main()," which most likely would come to an end. IDE code, instead, has two main functions, which are "void setup()" and "void loop()." The "setup()" function only goes through one time per run, and loop() in Arduino IDE does not quite stop since it is an infinite loop.

```
void setup()
```

Figure 10. void setup function

```
void loop()
```

Figure 11. void loop function

Except for those two main functions, we wrote some other helper functions to fulfill certain requirements to implement all the functionalities of our project, including int "setup_temperature()," "int setup_waterlevel()," "int setup_warmtime()," and "void calibrate()."

- **Helper functions**

  *void calibrate():*

  This function is coming with the HX711 GitHub library. HX711 is the weight sensor we used in our project. "Calibrate()" function makes the weight sensor calibrate to know what the weight of an object is. It first asks you to place nothing on the weight sensor and press "t" on your keyboard. It will know what it is going to feel like when nothing is placed on the weight sensor. The char in Byte is to store everything read from the serial monitor. "Serial.read()" is the function to read the letter the users type on the keyboard in the Serial monitor. This is how we capture the input char from the keyboard to set up all the pre-set variables from the other helper functions: setup_temperature(), setup_waterlevel(), and setup_warmtime().

```
boolean _resume = false;
while (_resume == false) {
  LoadCell.update();
  if (Serial.available() > 0) {
    if (Serial.available() > 0) {
      char inByte = Serial.read();
      if (inByte == 't') LoadCell.tareNoDelay();
    }
  }
  if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
    _resume = true;
  }
}
```

Figure 12. Functions to capture char from keyboard

Then the system will ask you on the serial monitor to put on a known weight and type in the weight value. “Serial.parseFloat()” is the function to read a float number from the serial monitor, in other words, the keycode input. After doing that, the weight sensor will calculate and give us a calibration value which would normally be around 447 with an

error range of less than 2. If the calibration number is larger than that, it means a small weight would cause a large weight value increase in the perspective of the weight sensor, vice versa. Furthermore, 447 is the value most close to the real world; one gram of weight will be one gram detected by the weight sensor.

```
Serial.println(F("Now, place your known mass on the loadcell."));
Serial.println(F("Then send the weight of this mass (i.e. 100.0) from serial monitor."));

float known_mass = 0;
_resume = false;
while (_resume == false) {
  LoadCell.update();
  if (Serial.available() > 0) {
    known_mass = Serial.parseFloat();
    if (known_mass != 0) {
      Serial.print("Known mass is: ");
      Serial.println(known_mass);
      _resume = true;
    }
  }
}
```

Figure 13.Functions to track the known mass

```
LoadCell.refreshDataSet(); //refresh the dataset to be sure that the known mass is measured correct
float newCalibrationValue = LoadCell.getNewCalibration(known_mass); //get the new calibration value

Serial.print(F("New calibration value has been set to: "));
Serial.print(newCalibrationValue);
Serial.println(F(", use this as calibration value (calFactor) in your project sketch."));
Serial.print(F("Save this value to EEPROM adress "));
Serial.print(calVal_eepromAdress);
Serial.println(F("? y/n"));
```

Figure 14.Functions to get weight calibration value

*setup_temperature(), setup_waterlevel(), setup_warmtime():*

We talk about these three helper functions together since they are using the same logic but return different values for different preset variables. The main function we used is "Serial.read()", which we talked about how it works in the previous paragraph. The Arduino will get the letter(char) from the Serial Monitor, then let the letter decide which value to return. Take "setup_temperature" as an example; char A will return 60, char B will return 70, and so on. Of course, some global variables declared will catch those values in our "setup()" function. "setup_waterlevel()" is the function that will return the percentage of the pre-set water level like 50% or 100%. "setup_warmtime()" is the function that will return the remaining time to keep the water at the preset temperature (mode 3).

```
int setup_temperature() {
  // put your setup code here, to run once:
  int set_temp;

boolean temp_flag = false;
Serial.println();
Serial.print(F("choose the preset tempature, please typein the captial char: A->50,B->60,C->70,D->80,E->90,F->100"));
while(temp_flag==false){
   if (Serial.available() > 0){
  char temp_char = Serial.read();
  if(temp_char=='A'){set_temp=50;temp_flag=true;}
  else if(temp_char=='B'){set_temp=60;temp_flag=true;}
  else if(temp_char=='C'){set_temp=70;temp_flag=true;}
  else if(temp_char=='D'){set_temp=80;temp_flag=true;}
  else if(temp_char=='E'){set_temp=90;temp_flag=true;}
  else if(temp_char=='F'){set_temp=99;temp_flag=true;}//using 99 instead of 100
  else{
    Serial.println();
    Serial.print(F("No such option, please re-choose one"));
    }
  }
}
Serial.println();
Serial.print(F("Water Temp is set to "));
Serial.print(set_temp);
Serial.print(F( " Celsius degrees"));
return set_temp;
}
```

Figure 15. Sample function of the setup helper functions

- **Setup function:**

  *Void setup():*

  This function only goes through once, and it prepares everything before the control unit goes to each mode in the loop() function. It sets up all the preset variables, sensors, LCD, and serial monitor. Though all the required information will be printed on the LCD, the serial monitor plays an important role in debugging. We keep track of those variables' values by using "Serial. print()." Of course, we called out the calibration function here in the "setup()," since calibration is part of the weight sensor setting. At the end of the "setup()" function, we defined the input pin eight and output pin twelve and thirteen. Pin eight is connected to the debounced switch(the RESET button) for the purpose of reset and starts another new round (Every new round starts from checking if adding water is needed and ends at the keep warm state). Pin 12 and 13 would receive the signals that could be either 1 or 0 to evoke the electric kettle and water pump.

- **Loop function:**

  *Void loop():*

  At the beginning of the loop function, the temperature sensor and the weight sensor would measure the current temperature and weight. Since the loop function is an infinite loop and the UNO Arduino will go through it quickly and repeatedly, so all the data is updated each time the Arduino goes through the loop. The following figures showed what functions we used to get the data based on the sensor's library.

```
sensors.requestTemperatures();
Celsius = sensors.getTempCByIndex(0);
```

Figure 16. Functions to get the temperature sensor data

```
mass = LoadCell.getData();
```

Figure 17. Functions to get the weight sensor data

The functional logic for our control unit is to take data input from sensors and give the correct output signal to open or close the water pump and the electric kettle. We start with mode 0; if the digital pin eight is high, we use a "while()" function to trap the program. Therefore the program will never go out of it and finish the loop until pin eight input switches to low. This is how our reset and hold state means.

```
//adding the reset button/////////////////
if(digitalRead(8)==HIGH){
  mode=0;
  println(F("Reset holding"));
  lcd.clear();
  lcd.print("RESET HOLDING");
  while(digitalRead(8)==HIGH){
    }
  }
```

Figure18. Reset and Holding state

After finishing the process in mode 0, the system would go to mode one unconditionally. Mode one is the state to add water until the water level reaches the preset water level. Mode two would have the electric kettle start working until the water temperature reaches the preset water temperature, then the kettle would be switched off. Timers for mode one and mode two have the same style. We have separate formulas to calculate them every time in each loop. This example could explain why we are doing the calculations in every loop. When people download some files from the internet, the remaining time is calculated based on the downloading speed, which is floating. The floated remaining time is more realistic than the fixed remaining time that the speed of the internet is not constant, and some other factors might affect the download speed as well. Thus, it is scientific to calculate the remaining time in each loop rather than just calculating once at the beginning of the loop. The following figures are the equations we used to calculate the remaining time.

```
mode_1_timer =  (preset_waterlevel*17)/160; //just assmue the water pump goes 160ml per sec
```

Figure 19. Timer of mode 1

```
mode_2_timer =  4.1868*(mass-933)*(preset_temp-Celsius)/1200;//empty water bottle weight 933g
```

Figure 20. Timer of mode 2

Mode three timer is different from the timer we used in modes one and two in that it is a chronological timer. The time remaining will not float since we are not calculating the time using the changing data. The timer for mode three is just like a timer we have on our phone that is simply counting down at the same pace as the clock. We used a variable called timer_3_start to record the time just before the program goes into mode three. Next, we used mode_3_timer to record the current time and then do the calculation with

preset_timewarm to get the time remaining in mode three. "millis()" is the function to record the current time.

```
timer_3_start = millis();//at the end of mode 2
mode_3_timer =millis();// each time in mode 3
```

Figure 21. Timer of mode 3

In modes 1,2,3, we stored all the required data into some variables. At the end of the "loop()" function, we used many "if," "else if," and "else" functions and displayed the result data on the LCD screen and serial monitor.

### 2.3.3 Operation of the Project

To use this Smart-Kettle-Module, the users can connect any electric kettles. First of all, the user should plug the kettle into the power relay. Then, on the laptop connected to the control unit circuits, the Arduino board in our case, the users would need to do the calibration for the weight sensor first. The users need to put an object with a known mass on the weight sensor and then tell the system the weight of that object. After that, the system would ask the user to select the required temperature and amount of water and keep warm time length. Next, the users can simply put the kettle on the base just like they usually do, and the kettle module should handle everything else.

When the user places the kettle on the base, which has the weight sensor implemented below, the weight sensor would measure the real-time weight. We defined the minimum weight as the sum of the weight of the base, the kettle, and the minimum water amount for boiling water. In mode one, if the current weight is lower than the minimum weight, the water pump would receive a signal of 1 and start pumping a specific amount of water.

Then, the system would go to the next mode, mode two, automatically. In this mode, the water pump would be switched off while the kettle would receive a signal of 1 and start working. See the figure 22 below. On the right top of the LCD screen is the remaining time to reach the preset temperature. On the left bottom is the current temperature of the water and the right bottom is the weight of the water and the kettle. The weight should be ml, but due to the limitation of the size of the LCD screen. The kettle would receive a signal of 0 and stop working immediately when the current temperature reaches the target temperature and goes to the next mode to keep warm function unconditionally.

In mode three, both the kettle and the pump should not be working. Unlike mode two, the remaining time on the right top is not the remaining time to reach the target temperature. Instead, it is the time left of the keeping warm function. In this mode, if the current temperature drops below the selected temperature, it would go back to mode two to reheat the water.

Besides, when the RESET switch is triggered, the system would go from mode one to mode three again. The module will check the weight in mode one to see if there is still some minimum

water left. If not, the pump would start working again. Otherwise, if there is still enough water left, the system would go through the rest of the modes just like before.

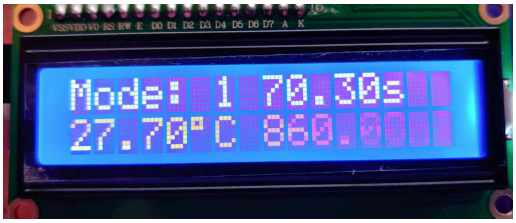



Figure 22. Mode 1


Figure 23. Mode 2


Figure 24. Mode 3

## 2.4 Verifications

- **LCD display**
  LCD unit test is basically for the three lines of the code shown on the screen. "lcd.begin()" defines the columns and rows displayed in LCD. Based on what we bought, we have to choose 16 columns and two rows for LCD size. We tried other numbers, but the content would disappear if the column number extended 16, same for rows extending 2. The Setcursor() is to set the cursor at a certain position. If we do the print, the printed content will appear in the position we just set.

```
lcd.begin(16, 2);// set up the LCD's number of columns and rows:

                    lcd.setCursor(8,1);

                    lcd.print("Mode: ");
```

Figure 25. Unit Test of LCD

- **DS18B20 Temperature Sensor**
  The reason we chose this sensor is the compatibility. The 1-Wire bus system makes the hardware configuration simpler since only one data line is connected. According to our unit test, the accuracy of the sensor is way better than we need. The data feedback speed is fast enough to deal with rapid changes in water temperature, which is our requirement

for the sensor since we would like the kettle to stop working immediately when the target temperature is reached.

- **HX711 Weight Sensor**
  We tried to make the whole design simple. And we did not need the sensors with very high accuracy. This specific sensor meets our requirements, including weighting range, price, and the difficulty for programming and installation. In the unit test, the sensor measured the real time weight fast enough that although there was around four seconds delay. The delay is acceptable since we allow some buffering time for the weight sensor.
- **Water Pump**
  Our first choice of the water pump was a 5V DC water pump. But during the unit test, we found out that the speed of pumping water is not fast enough. Then we switched to a 12V water pump that has a rated speed of 160GPH. The 12V water pump reached the rated speed when we were using the hard tubing. However, the pumping speed dropped to 140GPH when we replaced the hard pumping. It is a little bit lower than our expectation but still acceptable.

## 3. Cost

- Labor cost / teammate = 15 [$/hr] * 150 [hrs] * 2.5 = 5625 [$]
- Parts

| **Component** | **Manufacturer** | **Vendor** | **Quantity** | **Cost/unit** | **Total cost** |
|---|---|---|---|---|---|
| DS18B20 | Maxim Integrated | Sparkfun | 1 | $12 | $12 |
| HX711 | AVIA Semiconductor | Amazon | 1 | $13 | $13 |
| Water Pump | N/A | Amazon | 1 | $12 | $12 |
| LCD screen | N/A | Amazon | 1 | $20 | $20 |
| Kettle | Taylor Swoden | Amazon | 1 | $25 | $25 |
| Power Relay | Lot Relay | Amazon | 1 | $30 | $30 |

Table1. Retail price of the components

## 4. Conclusion

- **Success**

  We fulfilled the high-level requirements we wrote in the Design Document. All the add-on features work perfectly. Specifically, the kettle stops at the exact right target temperature. Furthermore, the water amount is accurate. Most importantly, the remaining time calculation has less than one-second error; once the timer countdown to zero, the kettle stops working immediately and enters mode three, and the time display will turn to the remaining time for the warm-keeping function.

- **Challenge**

  Our initial idea was to make a simple version of Arduino UNO and program the microcontroller externally. All the corresponding ports are labeled for each connection from sensors. During the first test, the microcontroller is burnt due to a shorted power supply. After we fixed the power supply, we found that we cannot upload our code to the microcontroller, even though all the connections (RX, TX, and RST) are connected correctly.

  Secondly, it is about the installation of the temperature probe. We put it into the kettle directly from the spout, which may lead to some inconvenience removing them. We tried to install the probe internally, but we failed.

- **Improvement**

  Since the system takes input from Arduino IDE, we can transplant the software to mobile devices and make an application software for the user interface.

  Due to the inconvenience of removing the temperature probe every time, we plan to replace our temperature sensor with a wireless one integrated into the kettle.

  We plan to replace our soft tubing with hard tubing and stabilize the tubing to the module box. Moreover, the lid of the kettle can be modified to fit the tubing. By filling the kettle externally (above the water level), we can avoid siphon during the water refilling stage.

## 5. Citations


[1] “Are You Wasting Time Boiling A Kettle?,” *Glug Glug Glug*, 28-May-2019. [Online]. Available: https://www.glugglugglug.com/news/are-you-wasting-time-boiling-a-kettle/. [Accessed: 17-Feb-2021].

[2] B. Spicer, “Best Smart Kettles on the market 2021?,” *SmartHomeBit*, 20-Dec-2020. [Online]. Available: https://www.smarthomebit.com/which-smart-kettle-should-you-buy-in-2020/. [Accessed: 19-Feb-2021].

[3] “Black + Decker™ 1.7-Liter Rapid Boil Electric Kettle,” *Bed Bath & Beyond*. [Online]. Available: https://www.bedbathandbeyond.com/store/product/black-decker-trade-1-7-liter-rapid-boil-electric-kettle/. [Accessed: 19-Feb-2021].

[4] President Dieter F. UchtdorfSecond Counselor in the First Presidency, *Living in a Fast-Paced World*. [Online]. Available: https://www.churchofjesuschrist.org/study/new-era/2015/06/living-in-a-fast-paced-world?lang=eng. [Accessed: 18-Feb-2021].

[5] Z. Cvijetic, “Discover How To Drastically Increase Your Efficiency & Effectiveness In Life,” *Medium*, 25-Apr-2020. [Online]. Available: https://medium.com/@zdravko/become-a-master-executioner-discover-how-to-drastically-increase-your-efficiency-effectiveness-8bd735727f2e. [Accessed: 18-Feb-2021].

[6] “ATMEGA32(L) Summary Datasheet by Microchip Technology,” *Digi*. [Online]. Available: https://www.digikey.com/htmldatasheets/production/54000/0/0/1/atmega32-l-summary.html?utm_adgroup=General&utm_source=google&utm_medium=cpc&utm_campaign=Dynamic+Search_EN_RLSA_Cart&utm_term=&utm_content=General&gclid=Cj0KCQiAvbiBBhD-ARIsAGM48bx_rs5l9CSfTQnZPhxpaQS8Sbz9_Fku-z1CdrvbdfdatfO2KIQM8gMaAqr3EALw_wcB. [Accessed: 19-Feb-2021].

[7] “Why Purified Water Is Better,” *Hanson Beverage Service*, 28-Jun-2019. [Online]. Available: https://hansonbeverage.com/2019/04/why-purified-water-is-better/#:~:text=Drinking%20purified%20water%20assures%20that,it%20smell%20and%20taste%20cleaner. [Accessed: 05-May-2021].

[8] “DS18B20+T&R,” *DigiKey*. [Online]. Available: https://www.digikey.com/htmldatasheets/production/54000/0/0/1/atmega32-l-summary.html?. [Accessed: 19-Feb-2021].

[9] “LTC2485, 24-Bit ΔΣ Analog-to-Digital Converter (ADC),” *DigiKey*. [Online]. Available: https://www.digikey.com/en/product-highlight/l/linear-tech/ltc2485-24-bit-analog-to-digital-converter. [Accessed: 19-Feb-2021].

[10] “GP,” *Amazon*, 2011. [Online]. Available: https://www.amazon.com/gp/product/B010LY7P3Y/ref=ox_sc_act_title_1?smid=A3VVL1M01MGYPX&psc=1. [Accessed: 19-Feb-2021].