Reagan Hardy

Beta Release Phase B

Embedded Systems Programming

Dr. Goncalo Martins

04 May 2023

Project Requirements

- Cost
  - The cost was not given by the customer.
- Size/aesthetic
  - The final size of the PCB should not exceed 10cm x 10cm.
- Display screen?
  - A seven-segment display will be used to output numbers.
- Time limit
  - Due to the constraints of the given website, the time should not exceed 59 hours and 59 minutes.
- Alarm sound
  - A buzzer will be used to create an alarm like sound.
- Amount of LEDs
  - Two LEDs will be used on the PCB.
- Amount of buttons
  - Two user buttons will be used. One will start the time whereas the other will stop the time.
- Debounce done in software?
  - The buttons will need to have a debounce. This problem was tackled in the software.
- Active or passive buzzer? Or both?
  - An active buzzer was used in KiCad. This means it is polarized.
- Shift register to interface seven segment display
  - The shift register that was used is 74HC595.
- How is it powered?
  - The timer is powered using a USB that connects to a laptop.

System Design

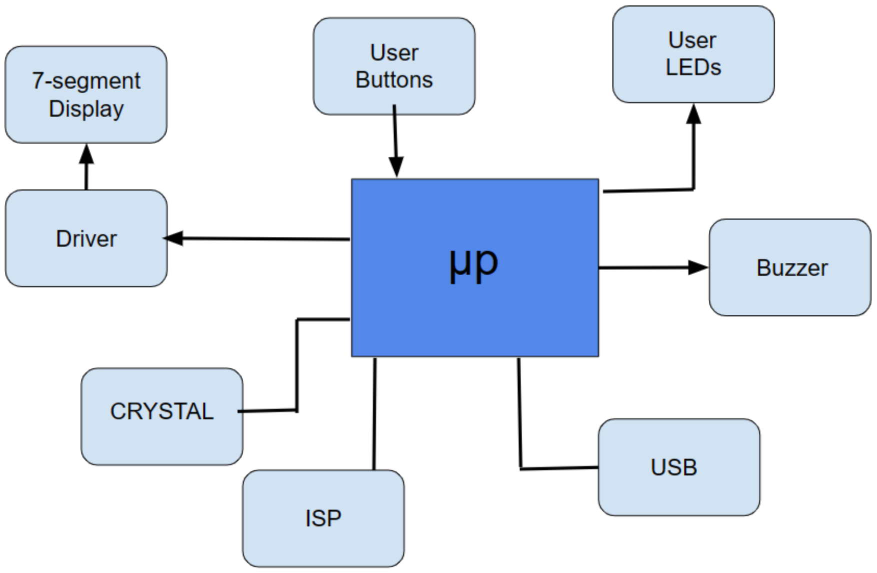

Image 1. The First Design of the Block Diagram

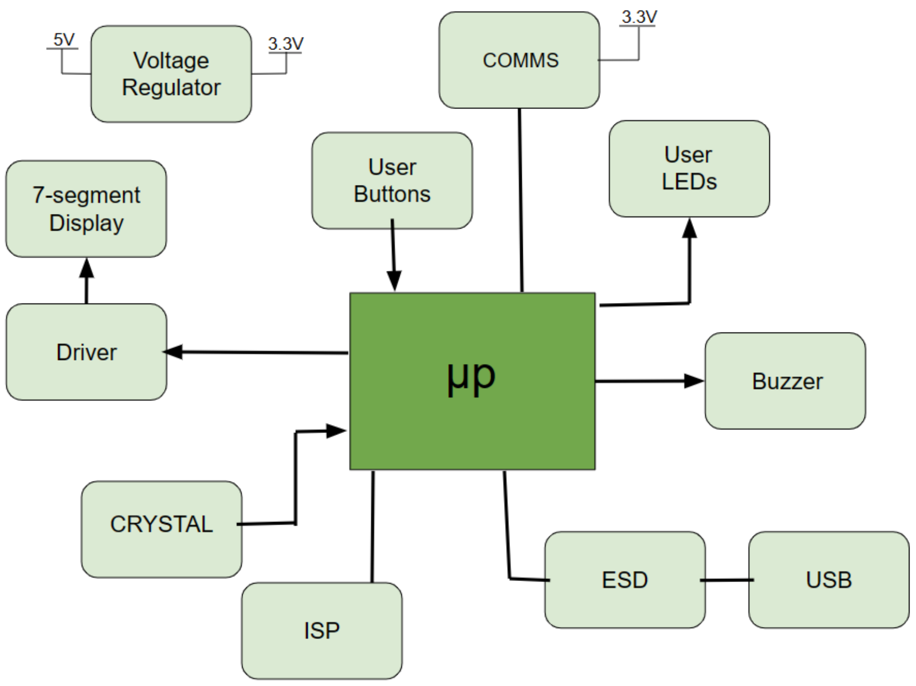

Image 2. The Second Design of the Block Diagram

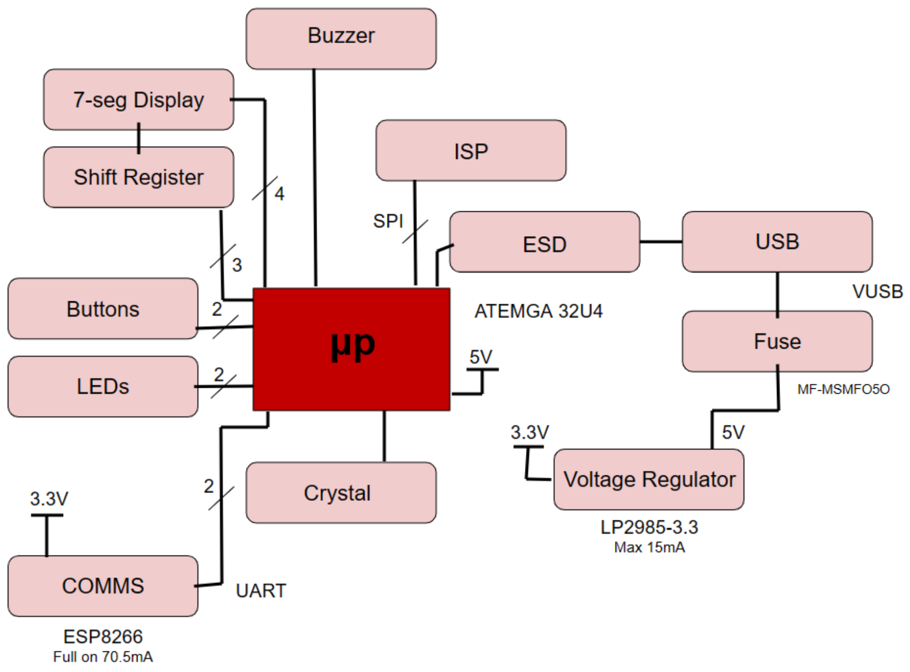

Image 3. The Final Design of the Block Diagram

Throughout the design process, three different block diagrams were created. They linearly improved throughout the lab as more information was considered. The final design was used to create the schematic in KiCad. These diagrams were very important in the design process because they show the entire system in a simple and visually appealing way.

Components Selection

- Arduino Uno (Optional) / Arduino Mega (Preference) Qt: 1
- 7-Segment Display (Sparkfun) ($1.60) Qt: 1
- Shift Register 8-Bit (SN74HC595) (Sparkfun) ($1.05) Qt: 1
- Buttons (Sparkfun) ($0.55/unit) Qt: 2
- LEDs Qt: 2
- Resistors
- Capacitors
- Mini Speaker (Sparkfun) ($2.10)
- ESP8266 (Amazon) ($9 for 3 units)
- Voltage Regulator

Most of these parts were carried over from Phase A. This made the component selection a simple process. It resulted in a cost of less than $60. This is an important step because the data for these parts can easily be found and stored for future use.

Build Prototype

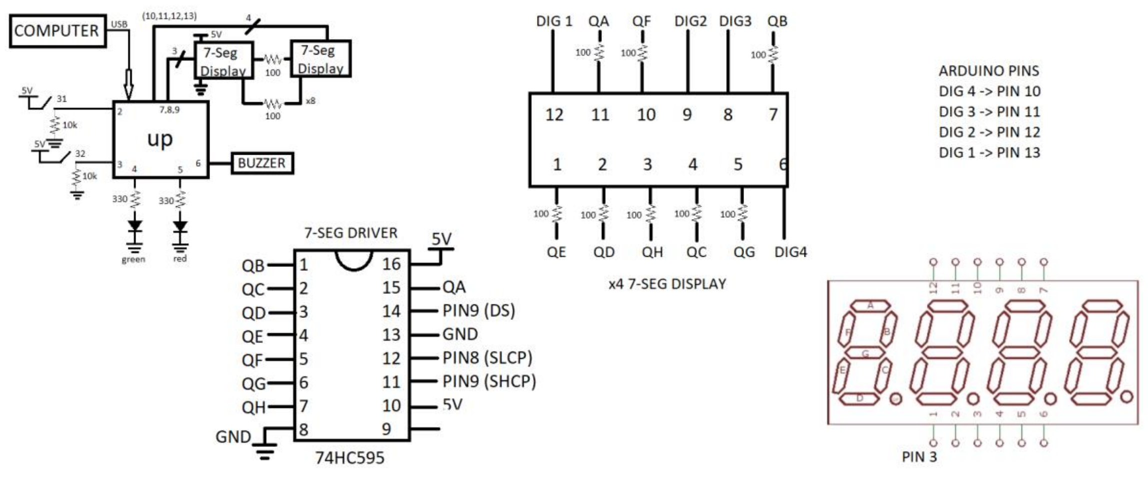

Image 4. Arduino Uno Schematic

During this Phase, an Arduino Shield will be used in conjunction with the created PCB. Arduino Shields provides hardware and controller. The board itself sends commands to the shield. The shield then reacts based on what it was told to do. They communicate via SPI in this case. By using this, it provides ease of use. However, a schematic was drawn out so that there are visuals on pin connections. This came in handy when designing the whole schematic on KiCad.

PCB Design

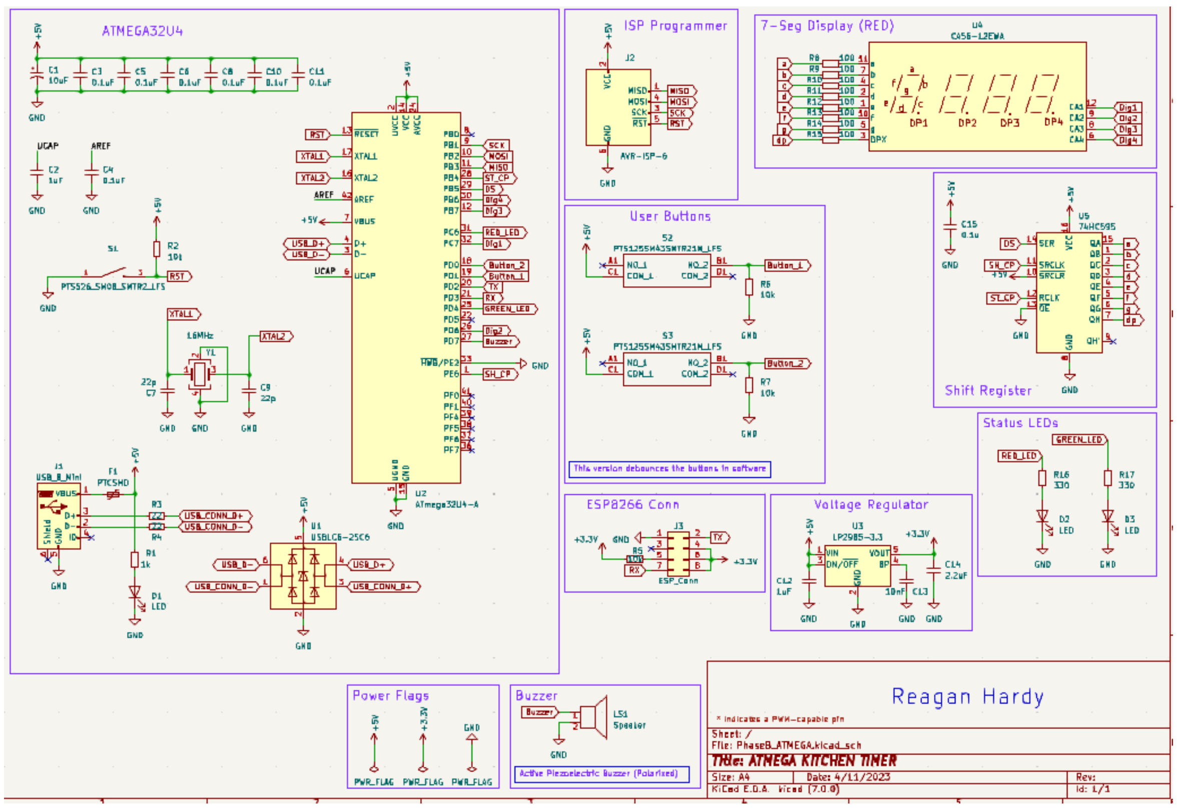
ATMEGA32U4
ISP Programmer
7-Seg Display (RED)
User Buttons
Shift Register
Status LEDs
ESP8266 Conn
Voltage Regulator
Power Flags
Buzzer
This version debounces the buttons in software
Active Piezoelectric Buzzer (Polarized)
Reagan Hardy
Title: ATMEGA KITCHEN TIMER
File: PhaseB_ATMEGA.kicad_sch
Date: 4/11/2023
KiCad E.D.A. kicad (7.0.0)

Image 5. Whole Atmega Kitchen Timer Schematic

This was designed using KiCad. To build this, many images were used including images 4 and 3. The components that were built are ATMEGA32U4, power flags, buzzer, ESP8266 Conn, voltage regulator, status LEDs, user buttons, shift register, ISP programmer, and a 7-segment display. Resistors, capacitors, ground, and switches were also used.

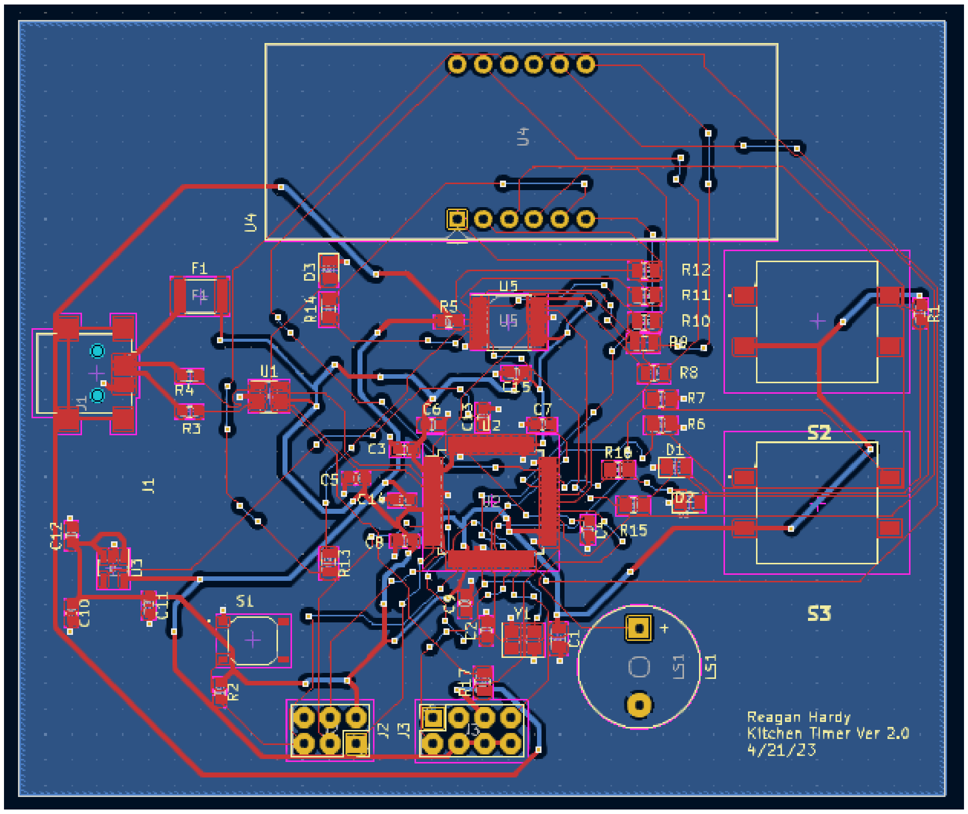

Image 6. PCB Layout

This layout was taken after the whole process was finished. This includes designing, routing, and labeling. The red lines are on the top plane of copper. Whereas the blue lines are beneath the plane. This helps so that the routes do not cross one another. Labels were added to help with clarity and to add a name and date.

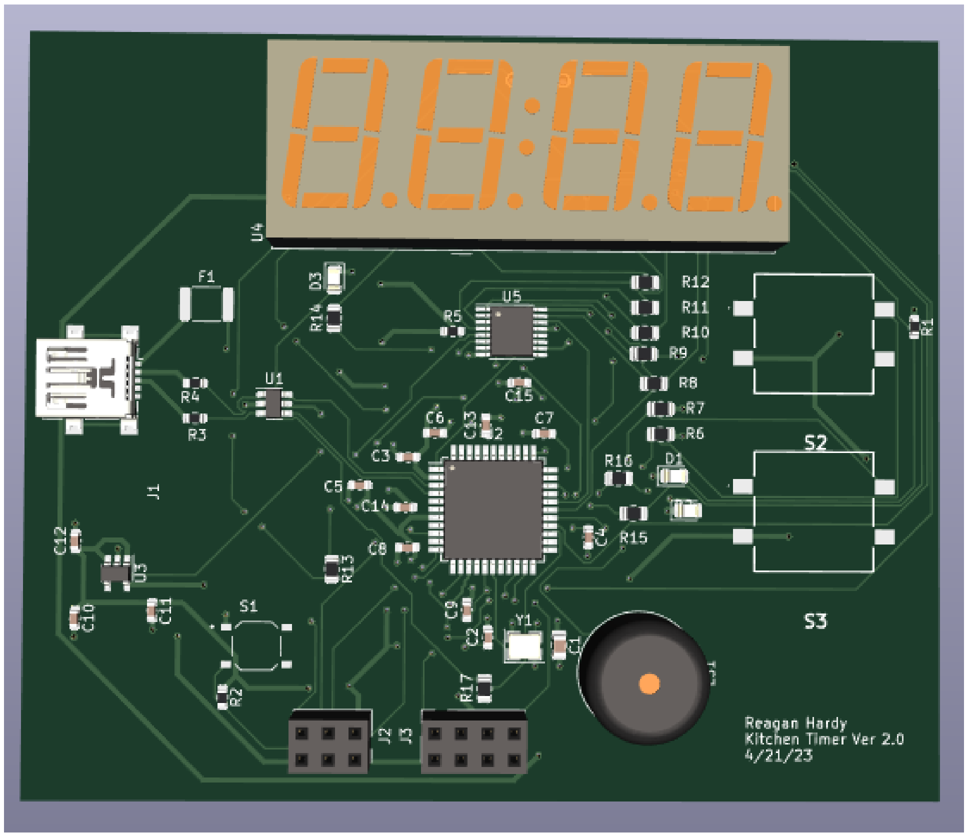

Image 7. Front of the Atmega Board

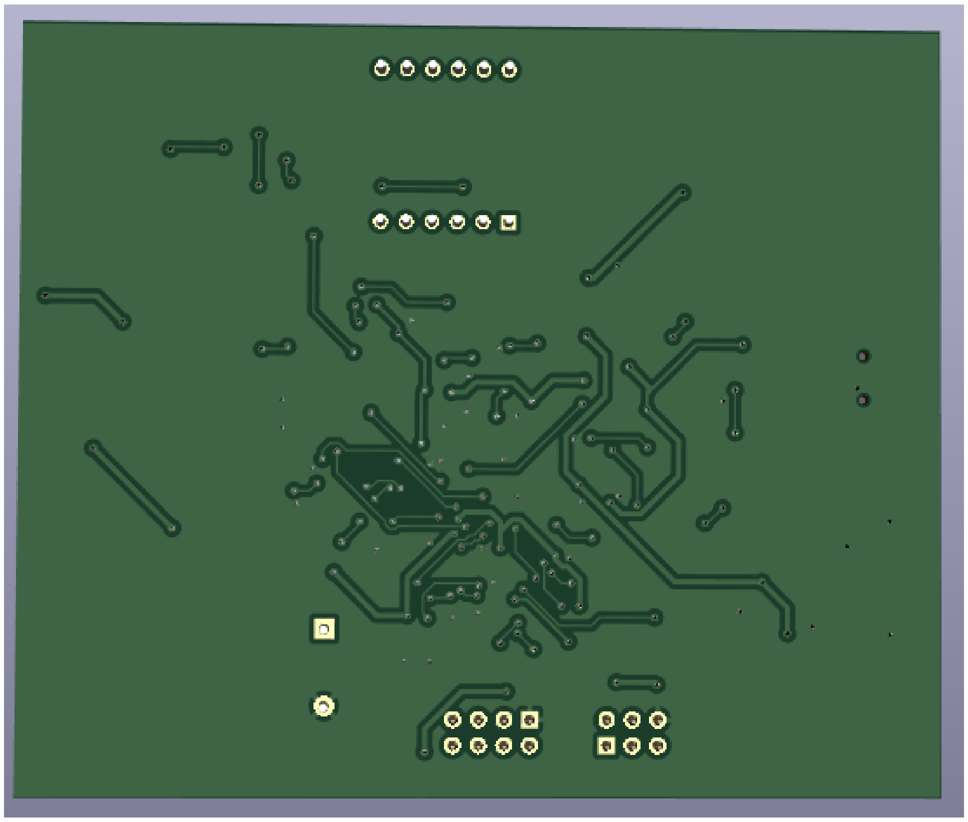

Image 8. Back of the Atmega Board

Images 7 and 8 show both sides of the Atmega board. The board itself could not be bigger than 10cm x 10cm. This board is 7cm x 9cm and in a rectangular shape. The 7-segmet display is at the top of the board to eliminate any distractions and to be clearly visible. The USB is on the left-hand side of the board. This was done so that the cable wouldn't be in the way or covering anything. The rest of the components were placed according to the routing. This process was done so that it wasn't as complex to route while still staying in the size guidelines.

Assemble Stage

This section is not applicable as the PCB's have not been delivered

Software Development

```
/* author: Reagan Hardy
created: 04-25-2023
updated:
how updated:
worked with: Noe Sheridan
*/
#define LED 13 //assigns value LED to pin 13

int iscommand;
char array[4];
bool start;

void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600); //starts the serial
}
void loop() {
  // put your main code here, to run repeatedly:
  if(Serial.available()){
    //Serial.print(" I received ");
    Serial.write("Received command"); //writes what the serial reads
    //Serial.write(Serial.read());
    char data = (Serial.read());
    if(data == '$'){
      iscommand = 1;
    }
    if(data == '\n'){
      //safety measure
    }
    if(iscommand){
      iscommand = 0;
      int i = 0;
      array[i]= data;
      i++;
    }
    if(array == 'STR'){
      start = true;
    } else if(array == 'STP'){
      start = false;
    }
    blink(start);
    for(int i= 0; i<4; i++){
        array[i]=0; //clear array
    }
  }
}
void blink(bool){
  if(start){
    digitalWrite(LED, HIGH);
  }else{
    digitalWrite(LED, LOW);
  }
}
```
Code 1. Initial Code Attempt

```
/* author: Reagan Hardy
created: 05-03-2023
updated:
how updated:
worked with: Noe Sheridan
*/
#define LED 13 //assigns value LED to pin 13

volatile unsigned char gISRFlag2 =0;
unsigned int reloadtimer1 = 100; //0.4 ms

int buffer_size = 20;
char array[4];
bool start;
char inChar;
char gCommsMsgBuff[buffer_size];
int ibuffer = 0;
byte packageflag = 0;
byte processdataflag = 0;
```

```
char compareArrays(char a[], char b[], int size){//function compares arrays
  int i;
  char result = 1;  // default: the arrays are equal

  for(i = 0; i<size; i++)
  {
    if(a[i]!=b[i])
    {
      result = 0;
      break;
    }
  }
  return result;
}

void setup() {
// put your setup code here, to run once:
pinMode(LED, OUTPUT);

Serial.begin(9600);
// Initialize Timer1 for Serial Comms
// Speed of Timer1 = 16MHz/64 = 250 KHz
TCCR1A = 0;
TCCR1B = 0;
OCR1A = reloadtimer1;            //  65535
TCCR1A |= (1<<WGM11);
TCCR1B = (1<<CS11) | (1<<CS10);   // 64 prescaler
TIMSK1 |= (1<<OCIE1A);
interrupts();
}
void loop() {
  char altgCommsMsgBuffer[buffer_size];
  int altcount =0;
  unsigned char altdigit = '0';

  if(gISRFlag2 == 1){
    gISRFlag2 = 0;
    inChar = (Serial.read());

    if(packageflag == 1){
      gCommsMsgBuff[ibuffer] = inChar; //reccords the character as part of the
message
      ibuffer++;

      if(ibuffer == buffer_size){
        packageflag =0;
        processdataflag =1;
      }
    }
  }

  if(inChar == '$'){ //start of package
    packageflag =1;

    for(int i=0;i< buffer_size; i++){ //clears previous package
      gCommsMsgBuff[i]=0;
    }
    ibuffer = 0;
  }
  if((inChar == '\n')&(packageflag ==1 )){//end of package
    packageflag =0;
    processdataflag = 1;
  }
  if(processdataflag ==1){//proccess serial commands

    processdataflag = 0;

    if(compareArrays(gCommsMsgBuff, "STR", 3) ==1 ){
      //turn on LED
      //start command
      digitalWrite(LED, HIGH);
    }
    if(compareArrays(gCommsMsgBuff, "STP", 3) == 1){
      //stop command
      digitalWrite(LED, LOW);
    }
    if(compareArrays(gCommsMsgBuff, "GET", 3) == 1){
      //send clock
      Serial.print("$00:01\n");
    }
  }
}
/**
 * @brief Timer 1 ISR
 * @param TIMER1_COMPA_vect
 * @return
 */
ISR(TIMER1_COMPA_vect)  // Timer1 interrupt service routine (ISR)
{
  if(Serial.available()>0)
  {
    gISRFlag2 = 1;
  }
}
```

Code 2. Final Complete Code

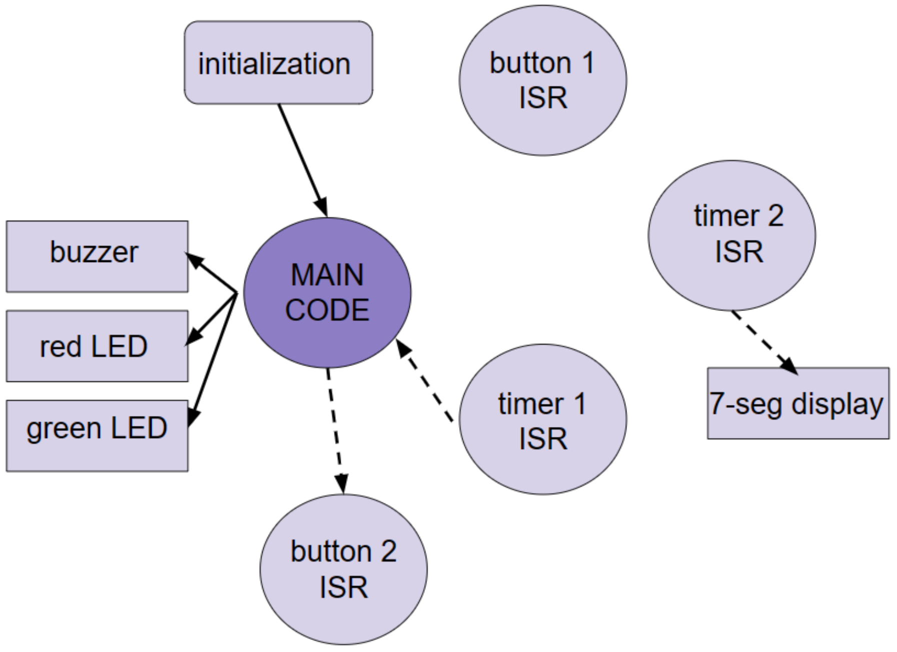

Image 9. Code Block Diagram

The block diagram in Image 9 was created to help write the code. The code itself has two commands. The first one is when the start button is pressed, turn on an LED. The second one is when the stop button is pressed, turn of the LED. This is done through a website and Wi-Fi connector. On the website, there are three buttons. The first one is labeled “Increments” and the serial command that responds to this is “$INC\n.” The second one is labeled “Start” and the serial command that responds to this is “$STR\n.” The third one is labeled “Stop” and the serial command that responds to this is “$STP\n.” The fourth one is labeled “Set Timer” and the serial commands that responds to this is “$SET, TMRS, mm:ss\n.” The last serial command that is used is “$GET.” This gets sent every 1 second.

*This section is missing a flow chart as well as the correct formatting for the codes. The plan for the flow chart is to design the first version by going over the final code. This will then be compared to others in class to make sure it has as much information as possible. These comments will be used to improve the flow chart and to make the final version. The code formatting needs to be double checked in order to have confidence in the program. This will be done by using the slides posted on Canvas. There is a link to the manual there. Once both are done, the software development section will be completed. *

Enclosure Design

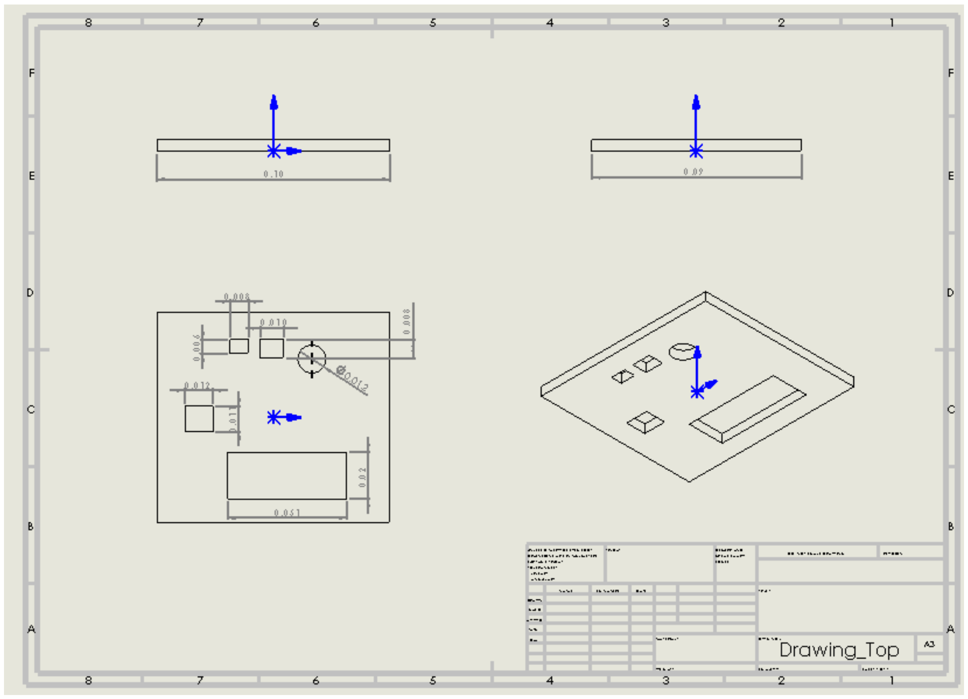

Image 10. Drawing of the Top Piece

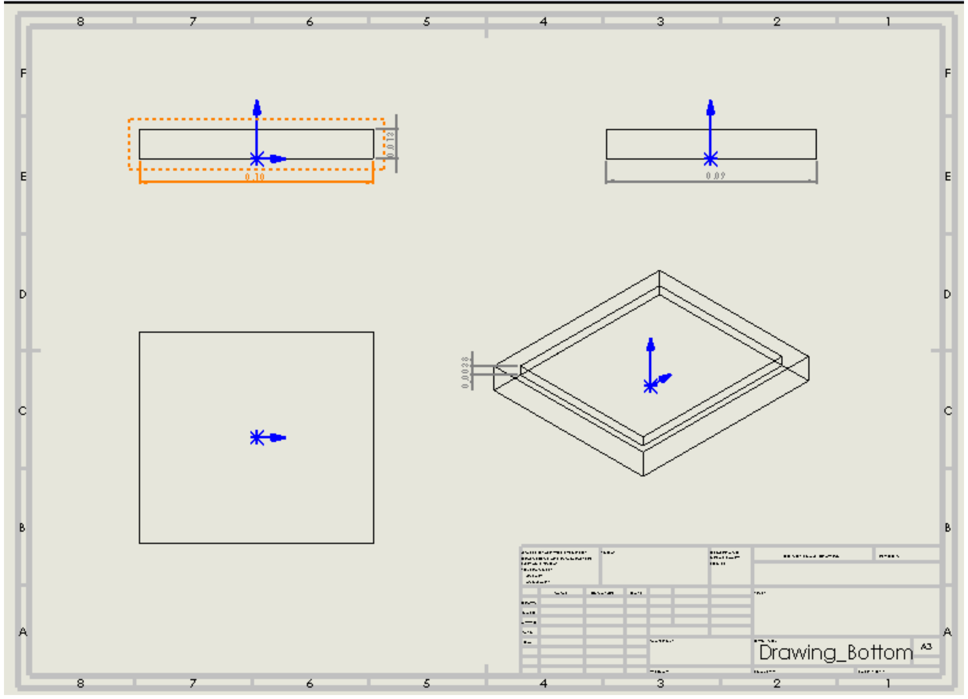

Image 11. Drawing of the Bottom Piece

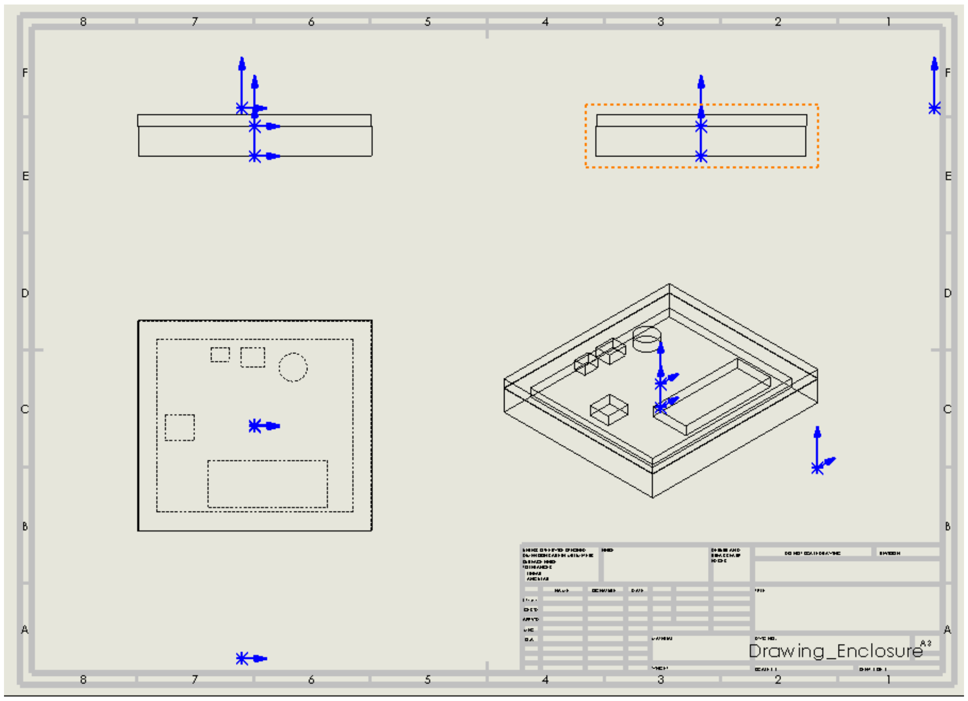

Image 12. Drawing of the Full Enclosure

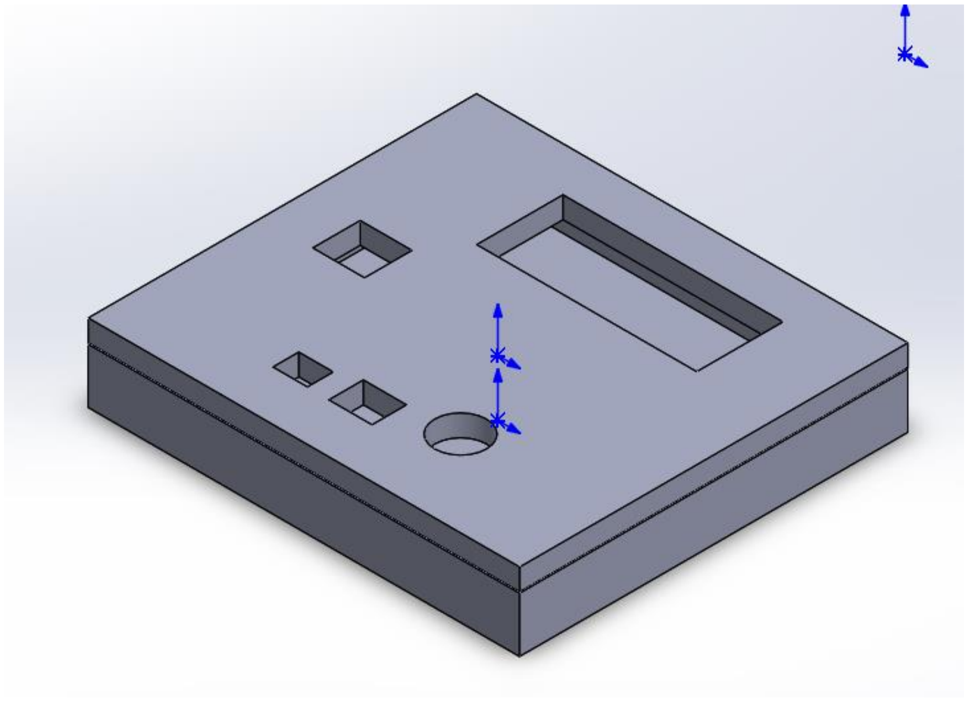

Image 13. PCB Enclosure

After designing the PCB, an enclosure was built so that the PCB had a place for storage. This was done using SolidWorks. There are two different parts, the top and bottom. An assembly was done so that the two parts could be mated together. There are five holes in the top so that parts of the PCB will be displayed and accessed easily. This includes the 7-segment display, two buttons, a USB, and the buzzer. This is the current final design but there are always improvements that can be made to increase productivity. This could include holes in the bottom so the PCB can be mounted. Topics like this will be considered heavily before it gets 3D printed.