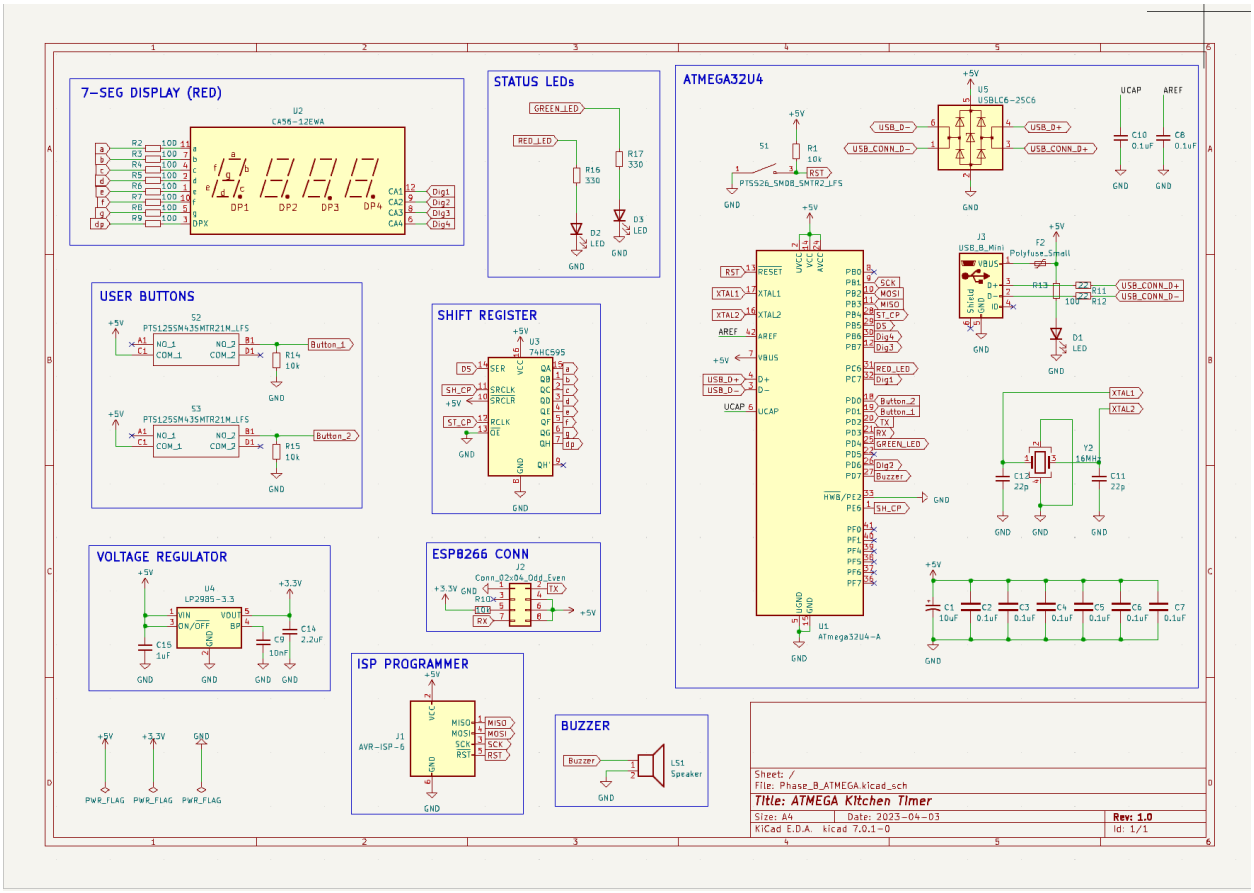
# Embedded Systems Report

Dr. Goncalo Martins

**ENCE 3231**

Raul Medina-Estrada




## Project Requirements

Embedded Systems 2023 encompasses the different components that are involved with several embedded systems. From the basic manufacturing of a kitchen timer on a PCB design software all the way to coding the microprocessor to create a kitchen timer that will be built from scratch. The requirements of this project are all laid out on the four phases of the class Phase A-D.

**Phase A:**

Phase A introduced students to the process of making a component in companies based on schematics, flowcharts and criteria all the way to the end result of a prototype of the component tested and verified with the correct code.

**Phase B:**

Phase B then gave students more freedom to create their own ideas for the same project now giving it their own flavor into the project. They created their own PCB design and designed it for the continuation of this project.

**Phase C:**

Phase C incorporated the software side of the device initially as it transitioned into assembling the device. Students looked at configuring the new STM microprocessor and soldering the ordered PCB boards to start assembling the final version of the board.

**Phase D:**

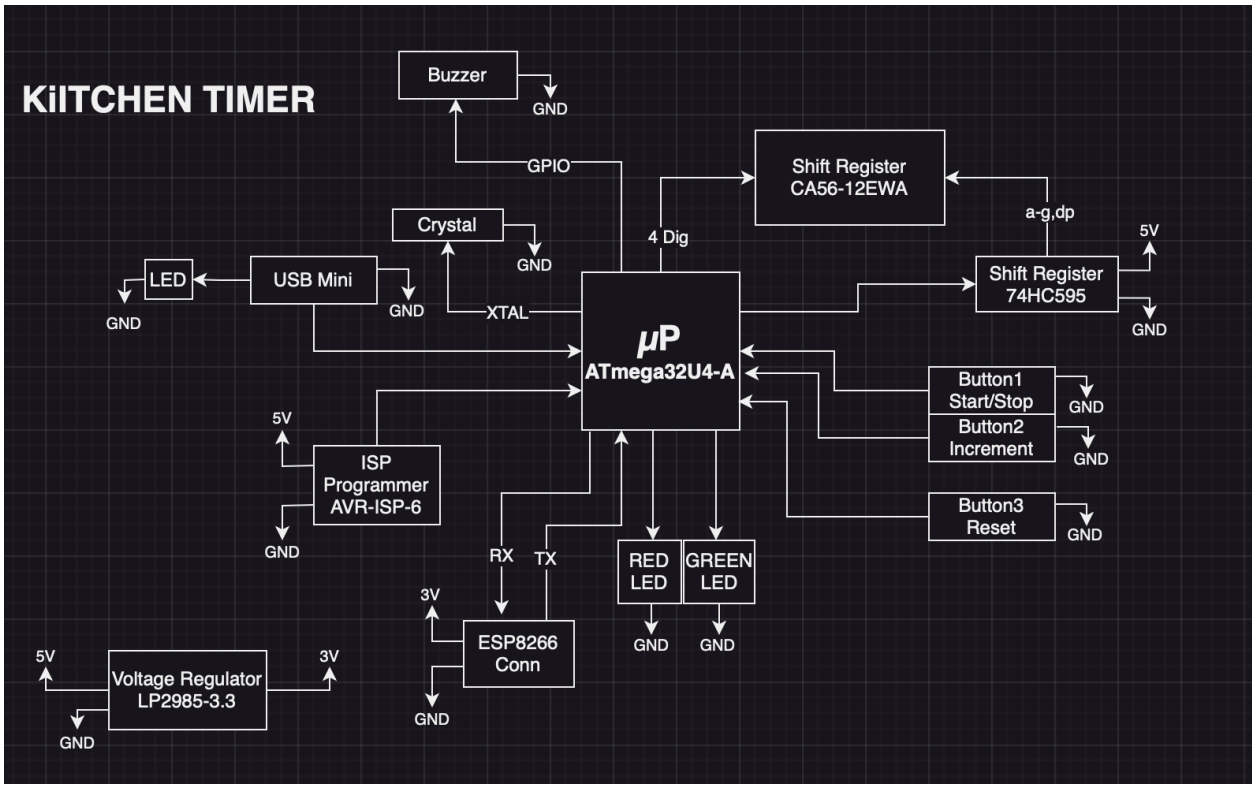
This phase was focused on our own personal projects that were able to be accomplished now that we have done phases A-D.

**System Design** (Block Diagrams)

Link For FlowChart
Image Below

https://app.code2flow.com/HEZqe5sGI6Bg

Block Diagram:

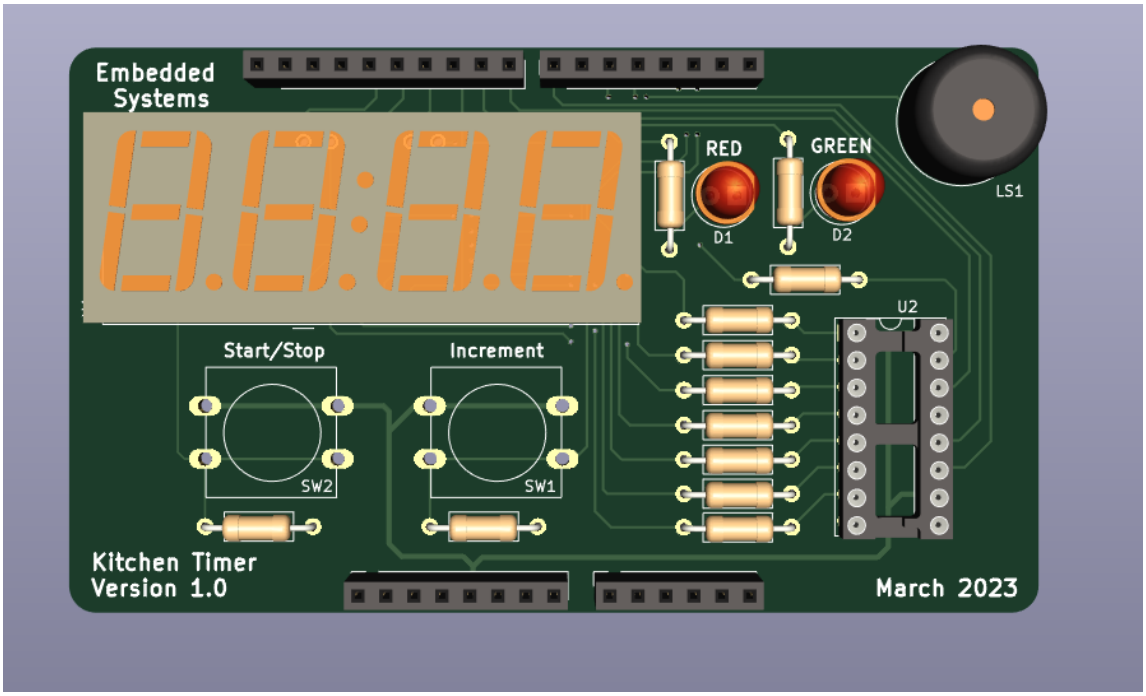



**Component Selection**

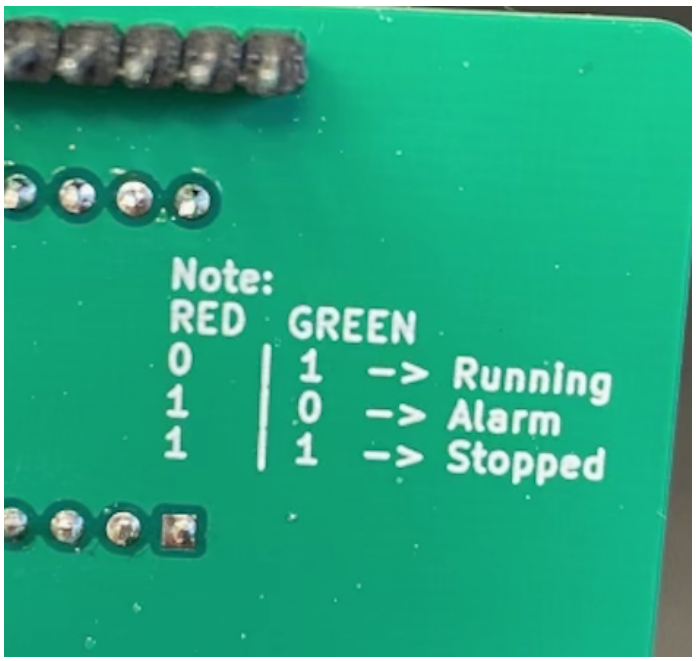
The components for the kitchen timer were selected prior to the lab by the professor including items that each accounted for the parts available at the university to the parts that best suited our applications needed for the most efficient timer the class can create.

**Build Prototype** (Arduino Shield Description)

The prototype included a PCB design that allowed for integrating pins into the arduino code that we already had in our components list. The class builds upon ideas learned in mechatronics and engineering integration to create a concurrent code that essentially creates a kitchen timer that allows it to increment the time, to start and stop the timer, and to turn a green led and red led on and off depending on the state.




The following states is given but a binary 1 and 0 system that is engraved on the back of of the PDB on this design and the final version of the PCB that uses its own microprocessors the states are listed below.

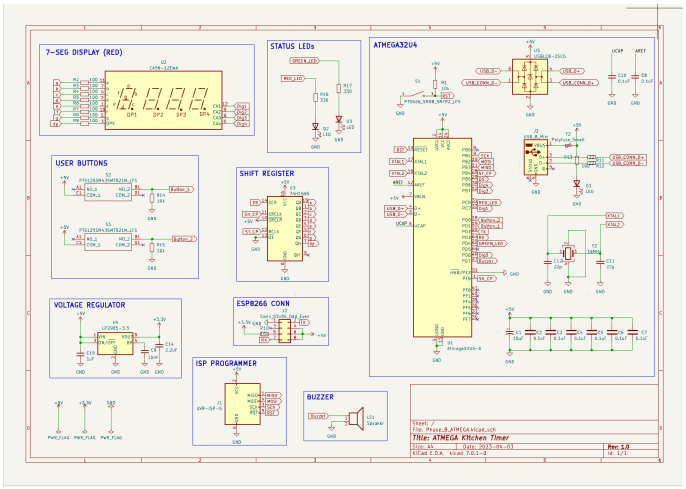


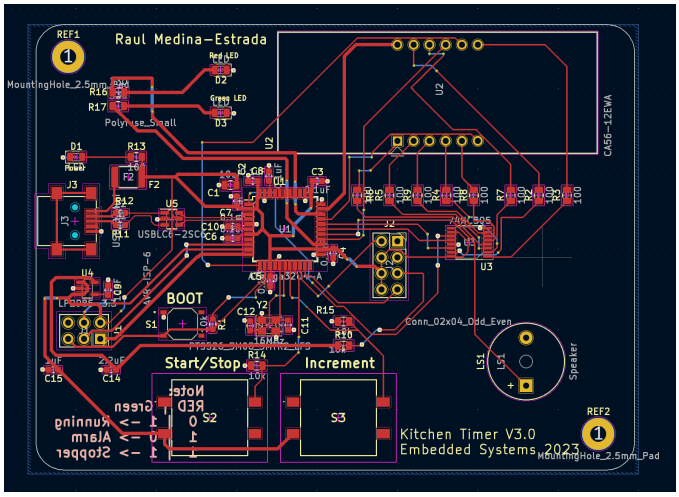
## COMPONENTS

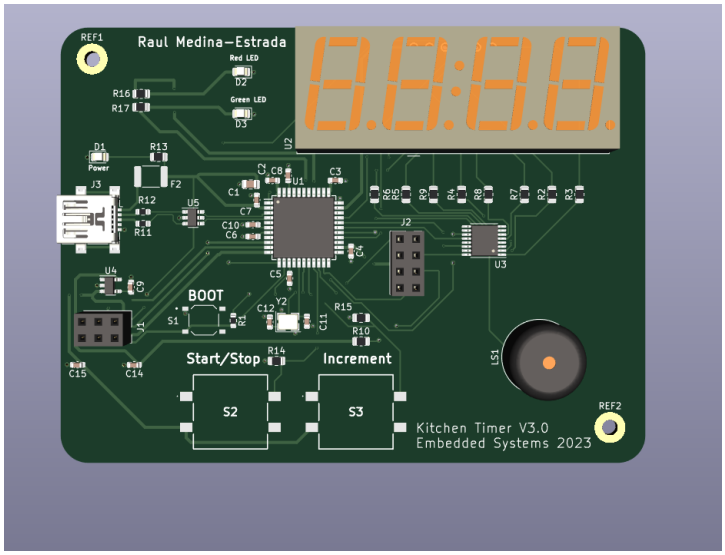
| null | Placed | References | Value | Footprint | Quantity |
|---|---|---|---|---|---|
| 1 | | C1, C4, C5 | 0.1uF | C_0603_16 | 8 |
| 2 | | C2, C3 | 22p | C_0603_16 | 2 |
| 3 | | C9, C13 | 1uF | C_0603_16 | 2 |
| 4 | | C8 | 10uF | C_0805_20 | 1 |
| 5 | | C14 | 2.2uF | C_0603_16 | 1 |
| 6 | | C15 | 10nF | C_0603_16 | 1 |
| 7 | | R5, R6, R7 | 100 | R_0805_20 | 8 |
| 8 | | R1, R2 | 10k | R_0805_20 | 2 |
| 9 | | R3, R4 | 330 | R_0805_20 | 2 |
| 10 | | R13, R17 | 10k | R_0603_16 | 2 |
| 11 | | R14, R15 | 22 | R_0603_16 | 2 |
| 12 | | R16 | 1k | R_0603_16 | 1 |
| 13 | | D1, D2, D3 | LED | LED_0805_ | 3 |
| 14 | | U1 | CA56-12EV | CA56-12EV | 1 |
| 15 | | U2 | 74HC595 | TSSOP-16_ | 1 |
| 16 | | U3 | ATmega32 | TQFP-44_1 | 1 |
| 17 | | U4 | USBLC6-2S | SOT-23-6 | 1 |
| 18 | | U5 | LP2985-3.3 | SOT-23-5 | 1 |
| 19 | | Y1 | 16MHz | Crystal_SM | 1 |
| 20 | | F1 | PTCSMD | Fuse_1812 | 1 |
| 21 | | S1, S2 | PTS125SM | PTS125_SI | 2 |
| 22 | | LS1 | Speaker | Buzzer_12 | 1 |
| 23 | | S3 | PTS526_SI | PTS526_SI | 1 |
| 24 | | J1 | AVR-ISP-6 | PinSocket_ | 1 |
| 25 | | J2 | USB_B_Mi | USB_Mini-I | 1 |
| 26 | | J3 | ESP_Conn | PinSocket_ | 1 |

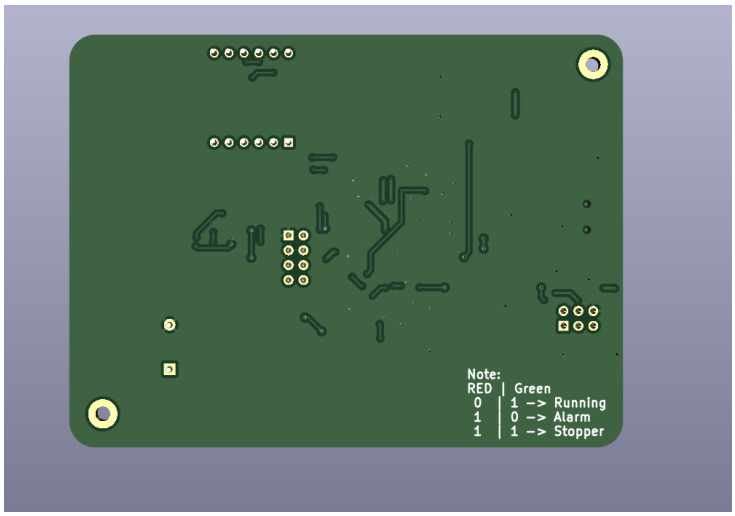
Components used come from the following template and accounted for the specific individual components located within the component. The use the new ESP wifi module and the ATmega microprocessor module within kiCad. The class will also be usng more slick smaller components such as the LED's and resistors, along with two new buttons.

**PCB Design** (Kitchen Timer design, Schematics & PCB layout description)




Raul Medina-Estrada
BOOT
Start/Stop
Increment
S2
S3
Kitchen Timer V3.0
Embedded Systems 2023


Raul Medina-Estrada
BOOT
Start/Stop
Increment
S2
S3
Kitchen Timer V3.0
Embedded Systems 2023


Note:
RED | Green
0 | 1 -> Running
1 | 0 -> Alarm
1 | 1 -> Stopper

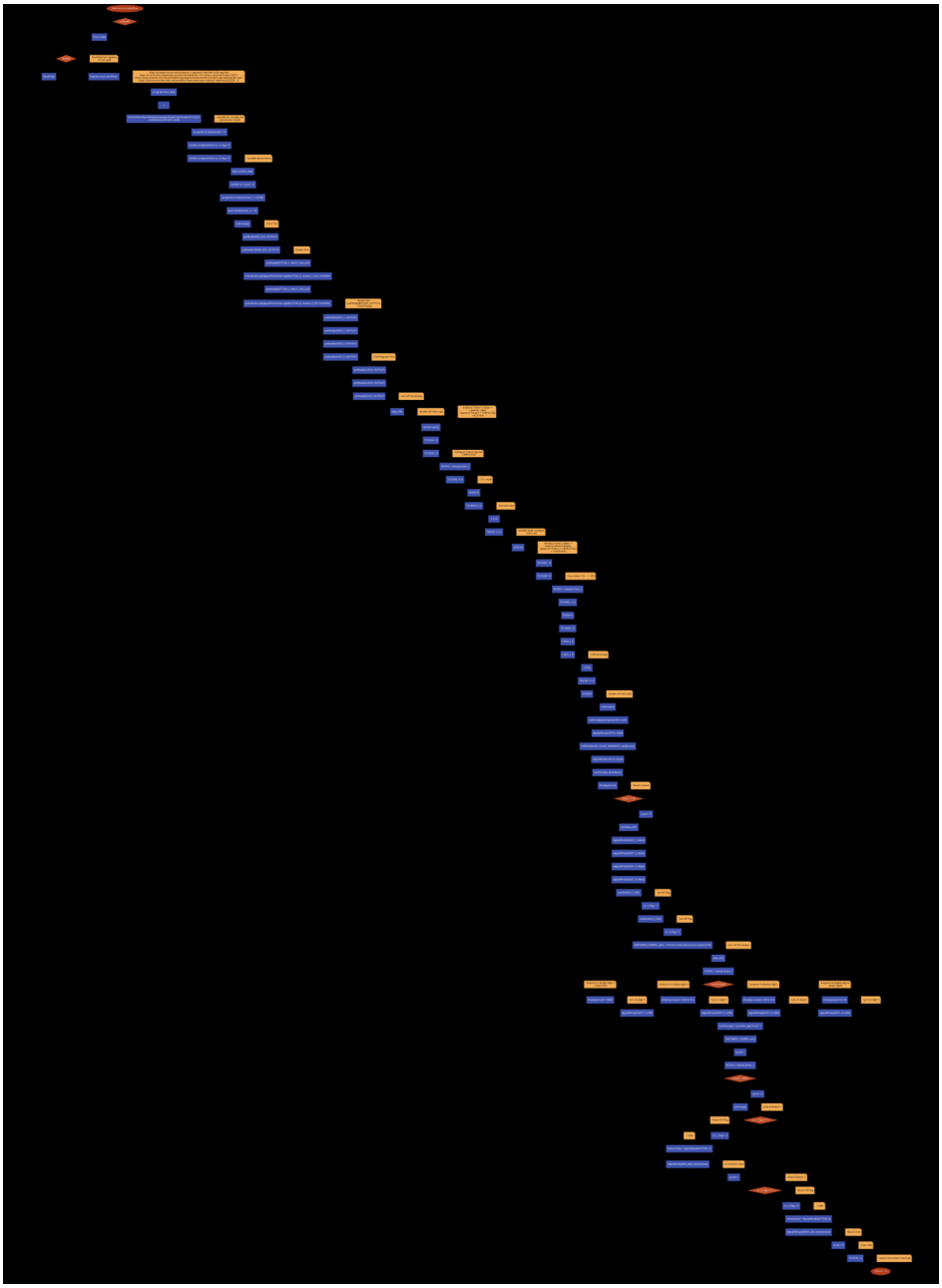
**Assemble Stage** (Documentation of assemble process)

**Software Replacement** (Code needs to be formatted; Mutex)

- The following code follows the initial prototype stage used for the arduino PCB design from the previous phase. It uses ISR flags and if statements to manipulate the seven segment display attached to the PCB board.

Link For FlowChart

Image Below



**Code:**

Macros: the pin assignments for various components such as buttons, LEDs, the buzzer, and the 7-segment display.
The program initializes the necessary pins as inputs or outputs.
It sets up interrupts for the two buttons (BUTTON_1 and BUTTON_2) using attachInterrupt(). These interrupts will be triggered when the buttons are pressed.
The code defines an array table[] that holds the hexadecimal values for displaying numbers on a 7-segment display.

```
#define BUTTON_2  2  // Inc Timer
#define BUTTON_1  3  // Start/Stop Timer and Stop Buzzer
#define GREEN_LED 4
#define RED_LED   5
#define BUZZER    6

#define DATA      9 // DS
#define LATCH     8 // ST_CP
#define CLOCK     7 // SH_CP

#define DIGIT_4   10
#define DIGIT_3   11
#define DIGIT_2   12
#define DIGIT_1   13

// 7-Seg Display Variables
unsigned char table[]=
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c
,0x39,0x5e,0x79,0x71,0x00};
byte current_digit;

// Volatile Variables
volatile unsigned char isr_1_flag = 0;
volatile unsigned char buzzer_flag = 0;

// Timer Variables
#define DEFAULT_COUNT 30     // default value is 30secs
volatile int  count = DEFAULT_COUNT;
unsigned char timer_running = 0;

unsigned int reload_timer_1 = 62500; // corresponds to 1 second
byte reload_timer_2 = 10;  // display refresh time

```

Timer-related variables are declared, including count (the timer value in seconds) and timer_running (a flag indicating if the timer is running).
The program sets up Timer2 to refresh the display periodically and Timer1 to keep track of time.
**The Display()** function is used to show a number on the 7-segment display.
The Button_2_ISR() interrupt service routine is triggered when BUTTON_2 is pressed, incrementing the timer value (count).

```
void Display(unsigned char num, unsigned char dp)
{
  digitalWrite(LATCH, LOW);
  shiftOut(DATA, CLOCK, MSBFIRST, table[num] | (dp<<7));
  digitalWrite(LATCH, HIGH);
}

void disp_off()
{
    digitalWrite(DIGIT_1, HIGH);
    digitalWrite(DIGIT_2, HIGH);
    digitalWrite(DIGIT_3, HIGH);
    digitalWrite(DIGIT_4, HIGH);
}
```

**The Button_1_ISR()** interrupt service routine is triggered when BUTTON_1 is pressed, indicating the start/stop action of the timer. It uses switch cases loops to go through a timer that uses a counting system with sixty seconds in a minute.

```
void Button_1_ISR()
{
  // Set ISR Flag
  isr_1_flag = 1;
}

ISR(TIMER2_COMPA_vect)   // Timer2 interrupt service routine (ISR)
{
  disp_off();  // turn off the display
  OCR2A = reload_timer_2;

  switch (current_digit)
  {
    case 1: //0x:xx
      Display( int((count/60) / 10) % 6, 0 );   // prepare to display digit 1 (most left)
      digitalWrite(DIGIT_1, LOW);  // turn on digit 1
      break;

    case 2: //x0:xx
      Display( int(count / 60) % 10, 1 );   // prepare to display digit 2
      digitalWrite(DIGIT_2, LOW);     // turn on digit 2
      break;

    case 3: //xx:0x
      Display( (count / 10) % 6, 0 );   // prepare to display digit 3
      digitalWrite(DIGIT_3, LOW);    // turn on digit 3
      break;

    case 4: //xx:x0
      Display(count % 10, 0); // prepare to display digit 4 (most right)
      digitalWrite(DIGIT_4, LOW);  // turn on digit 4
  }

  current_digit = (current_digit % 4) + 1;
}
```

```

ISR(TIMER1_COMPA_vect)  // Timer1 interrupt service routine (ISR)
{
  count--;
  OCR1A = reload_timer_1;

  if(count == 0)
  {
      // Stop Timer
      stopTimer1();

      // Raise Alarm
      buzzer_flag = 1;
      timer_running = 0;
  }
}
```

```
ISR(TIMER1_COMPA_vect)  // Timer1 interrupt service routine (ISR)
{
  count--;
  OCR1A = reload_timer_1;

  if(count == 0)
  {
      // Stop Timer
      stopTimer1();

      // Raise Alarm
      buzzer_flag = 1;
      timer_running = 0;
  }
}
```

**The TIMER2_COMPA_vect** interrupt service routine updates the display by cycling through the four digits of the timer and showing the corresponding numbers on the 7-segment display.
**The TIMER1_COMPA_vect** interrupt service routine decreases the count value each second and raises an alarm (buzzer) when the count reaches zero.
The **stopTimer1()** function stops Timer1 by disabling the clock and canceling the timer interrupt.

```
void stopTimer1()
{
  // Stop Timer
  TCCR1B = 0; // stop clock
  TIMSK1 = 0; // cancel clock timer interrupt
}
```

The **startTimer1()** function initializes and starts Timer1 for counting time.

```
void startTimer1()
{
  // Initialize Timer1 (16bit) -> Used for clock
  // Speed of Timer1 = 16MHz/256 = 62.5 KHz
  noInterrupts();
  TCCR1A = 0;
  TCCR1B = 0;
  OCR1A = reload_timer_1; // compare match register 16MHz/256
  TCCR1B |= (1<<WGM12);   // CTC mode
  TCCR1B |= (1<<CS12);    // 256 prescaler
  TIMSK1 |= (1<<OCIE1A);  // enable timer compare interrupt
  interrupts();
}
```

The **Active_Buzzer()** function generates a buzzing sound using the buzzer component.
In the loop() function, the program checks for button presses and controls the timer based on the button actions. It also handles the alarm (buzzer) when the timer reaches zero.

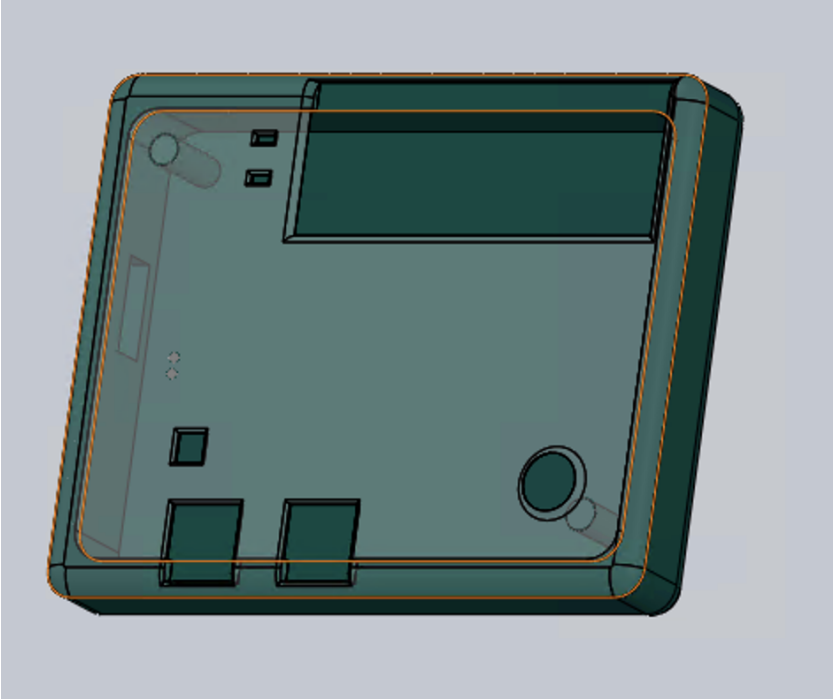
```
void Active_Buzzer()
{
  unsigned char i;
  unsigned char sleep_time = 1; // ms

  for(i=0;i<100;i++)
   {
    digitalWrite(BUZZER,HIGH);
    delay(sleep_time);//wait for 1ms
    digitalWrite(BUZZER,LOW);
    delay(sleep_time);//wait for 1ms
    }
}
```

**Void Loop:**
The loop function in this code is a very simple code that checks for isr flags and turns off the corresponding flags to zero as it resets them. It also turns on the respective LEDs when the indicated button is pressed. This is the code that connects the interrupt flags to the actual timer and hardware that controls the kitchen timer.

```
196  void loop()
197  {
198    // Attend Button 2 ISR
199    if(isr_1_flag == 1)
200    {
201      // Reset ISR Flag
202      isr_1_flag = 0;
203
204      if(timer_running == 0)
205      {
206        // Start Timer
207        timer_running = 1;
208
209        if(count == 0)
210          count = DEFAULT_COUNT;
211
212        if(buzzer_flag == 1)
213        {
214          buzzer_flag = 0;
215
216          // LEDs -> Timer Stopped
217          digitalWrite(RED_LED, HIGH);
218          digitalWrite(GREEN_LED, HIGH);
219        }
220        else
221        {
222          startTimer1();
223          // LEDs -> Timer Running
224          digitalWrite(RED_LED, LOW);
225          digitalWrite(GREEN_LED, HIGH);
226        }
227      }
```

```
228      else
229      {
230        // Stop Timer
231        stopTimer1();
232        timer_running = 0;
233
234        // LEDs -> Timer Running
235        digitalWrite(RED_LED, HIGH);
236        digitalWrite(GREEN_LED, HIGH);
237      }
238    }
239
240    // Attend buzzer_flag
241    if(buzzer_flag == 1)
242    {
243      // Make Noise...
244      digitalWrite(RED_LED, HIGH);
245      digitalWrite(GREEN_LED, LOW);
246      Active_Buzzer();
247    }
248  }
249
```

**Enclosure Design** (CAD Designs, 3D printed Model)



Enclosure Design includes a clear top side with openings for the Start/Stop button, the Increment button, and the newly added reset button. It also includes two LED holes for the smaller new LEDs ane the opening for the original seven segment display and buzzer. There are also two included mounting holes on the actual PCB design. The button allows for the pins on the bottom of the PCB and the USB port on the side included.

Once I printed the case, I realized that the top side had miscalculations. I had measured starting from the edge of the case and not from the start of where the IC would be which led to the buttons and buzzer holes to be lower and further in than intended. From here I wanted to pivot. I originally wanted the top side to be seen through so I could see my custom IC board but the colors selection did not offer a clear option. Therefore, instead of reprinting the case, I wanted to get creative with it. I will be doing this with epoxy. There are multiple projects that use epoxy for electronics as they do a really good job with preserving the component. I will do this by first using a hot glue gun to cover the parts that I don't want the epoxy to ruin in the kitchen timer. This includes the port, buttons and buzzer. From here I will add the epoxy up until the buttons are still reachable with fingers. The final product leaves the actual  IC board still visible in order to be able to see my first IC board project while still having a kitchen timer that is functional and efficient.

With the time this document needs to be submitted my case will not be complete due to wanting to add the ESP8266 which was not offered during the class. I ordered it and is arriving after the due date. I need to solder the device before applying the resin that is meant to be on the component indefinitely.


**Conclusion:**

This course was perfectly structured to teach students how to be able to start from an idea and manifest the actual component from it. It starts with an idea on paper and putting together components and a schematic that will be able to be put together on a breadboard. In our case the professor was able to get the bread board prototype out of the way to make sure it was working. From this idea we worked with the professor to be able to create a working pcb design on KiCad. From there the boards were ordered and soldered. A kitchen timer enclosure was then created for it. The original kitchen timer used an Arduino interface so we spent Phase C defveloping knowledge on an STM development software since it uses the same microprocessor software as the ATMega microprocessor used in the kitchen timer. Once everything was put together the students have finally finished their own kitchen timer made from the brilliant mind of the professor. These type of classes include the type of projects that is necessary in order for students to be able to show something for their future prospects in jobs and research opportunities.