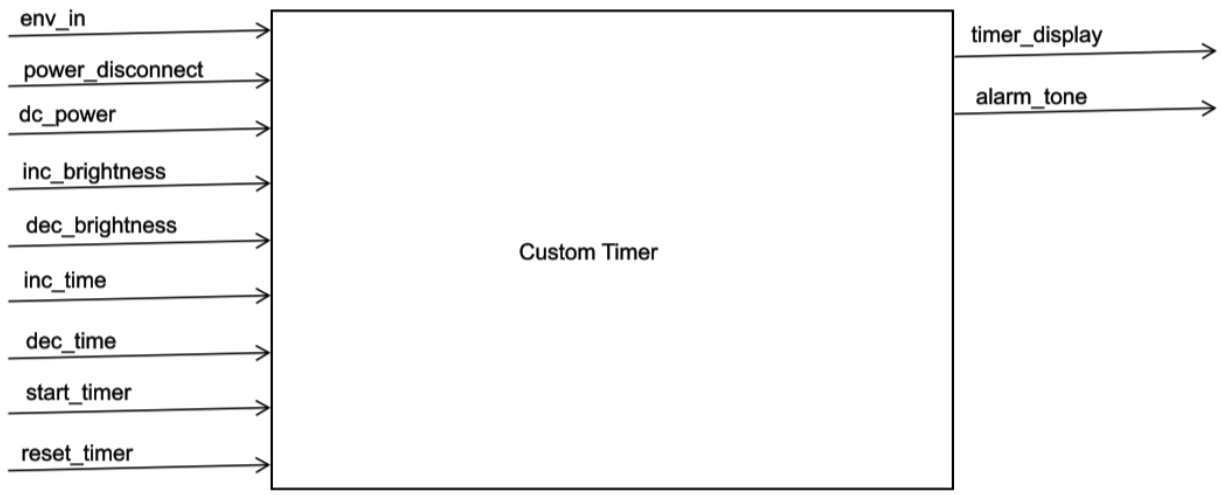
Joelle Czarissa Perez
Portia Bailey
Custom Timer 004-2

System Level Verification

Black Box Diagram:



System Function Description:
The system will perform a countdown starting from the user's desired custom time. This countdown will be no more than 1 second off every minute. When the countdown reaches 10 seconds remaining, the system will warn the user of this with a beeping tone at 440 ± 1 Hz. When the countdown reaches 0 seconds remaining, the system will indicate this to the user with a constant tone at 440 ± 1 Hz. Along with inputs that control the countdown, the system will be able to output three different levels of brightness on the seven segment display. The level of brightness is determined by user input. The system will provide a legible user interface that can be seen from at least 3 feet away. In addition, the system is provided with a disconnect switch which will cut off power to the system in the case of an emergency.

Interface Definitions:

Interface Name	Interface Properties
env_in	● IP43 ○ Dust - Protection against objects over 1mm ○ Liquid - Protection against vertical water spray up to 60 degrees ● Dimensions ○ (5.20 x 3.66 x 6.10)'' ± .1''
power_disconnect	● 1 - HIGH (Flipped/Activated) ○ System is ON

	● 0 - LOW (Unflipped/Neutral) ○ System is OFF
dc_power	● Vmax - 12V ± 1% ● Vmin - 5V ± 1% ● Inominal - 500mA ● Ipeak - 1A
inc_brightness	● 1 - HIGH (Pressed/Activated) ● 0 - LOW (Unpressed/Neutral)
dec_brightness	● 1 - HIGH (Pressed/Activated) ● 0 - LOW (Unpressed/Neutral)
inc_time	● 1 - HIGH (Pressed/Activated) ● 0 - LOW (Unpressed/Neutral)
dec_time	● 1 - HIGH (Pressed/Activated) ● 0 - LOW (Unpressed/Neutral)
start_timer	● 1 - HIGH (Pressed/Activated) ● 0 - LOW (Unpressed/Neutral)
reset_timer	● 1 - HIGH (Pressed/Activated) ● 0 - LOW (Unpressed/Neutral)
timer_display	● 1 - HIGH (Segment Light ON) ● 0 - LOW (Segment Light OFF) ● Brightness Range: -100 to 100
alarm_tone	● Tone ○ ON - 440 ± 1 Hz ○ OFF - 0 Hz

Testing Requirements Plan:

Testing Plan #1 (Requirement 1):

1. Plug in USB DC power into power source
2. Set up timer on phone to 60 seconds
3. Press inc_time push button until seven segment display shows 60 seconds
4. Press start_time push button on custom timer and start on phone timer at the same time
5. Observe both the custom timer and the phone timer, verify that the countdowns are in sync or only 1 second apart.

Testing Plan #2 (Requirement 2):

1. Open enclosure, observe and verify the use of connectors and split loom on wires
2. Place a piece of paper inside the enclosure.
3. Close enclosure
4. Spray water onto system at a vertical angle of 60 degrees
5. Open enclosure, observe and verify the paper is dry

Testing Plan #3 (Requirement 3):

1. Take a ruler/yardstick and measure a distance of three feet
2. Place the system on one end of the distance and stand on the opposite end
3. Observe the labels on the system and verify that the labels are legible from three feet away

Testing Plan #4 (Requirement 4):

1. Plug in USB DC power into power supply
2. Press inc_time push button until seven segment display shows 5 seconds
3. Set up tuner app on phone
4. Press start_time push button on the system
5. Wait for countdown to finish
6. Listen to the speaker and look at the tuner, verify alarm_tone is HIGH at a constant 440 hz

Testing Plan #5 (Requirement 5):

1. Plug in USB DC power into power supply
2. Press dec_brightness push button, observe seven segment display and verify that the brightness has decreased to “low brightness”
3. Press inc_brightness push button, observe seven segment display and verify that the brightness has increased to “medium brightness”
4. Press inc_brightness push button, observe seven segment display and verify that the brightness has increased to “high brightness”

Testing Plan #6 (Additional Requirement):

1. Plug in USB DC power into system
2. Press inc_time push button until seven segment display shows 15 seconds
3. Set up tuner app on phone
4. Press start_time push button on the system
5. Wait for countdown to reach 10 seconds remaining
6. Listen to the speaker and look at the tuner, verify alarm_tone is alternating between HIGH at 440 hz and LOW at 0 hz

Testing Interfaces Plan:

1. Use a ruler to verify the height, width, and length of the enclosure
2. Open enclosure and place a paper inside the enclosure
3. Close enclosure
4. Spray water onto system at a vertical angle of 60 degrees
5. Open enclosure, observe and verify the paper is dry
6. Dry the outside of the enclosure
7. Plug in USB DC power into system

8. Press dec_brightness push button and observe brightness decrease in seven segment display
9. Press inc_brightness push button two times and observe brightness increase in seven segment display
10. Press inc_time push button and observe the display updating the time
11. Stop when display shows 1 min and 0 seconds
12. Press dec_int push button and observe the display updating the time
13. Stop when display shows 0 min and 30 seconds
14. Press the start_time pushbutton and observe the display updating and decreasing in time every second
15. Set up tuner app on phone
16. Wait for countdown to reach 10 seconds remaining
17. Listen to the speaker and look at the tuner, verify alarm_tone switches between HIGH at 440 hz and LOW at 0 hz.
18. Wait for countdown to finish
19. Listen to the speaker and look at the tuner, verify alarm_tone is HIGH at a constant 440 hz
20. Press the reset_time button. Observe the display updating back to the user chosen time and listen to the speaker and verify that it is LOW at 0 hz.
21. Unplug USB DC power
22. Use a DMM and measure the voltage outputted by the USB DC power supply and verify that it is between 5V to 12V ± 1%

Video Demos:
Engineering Requirement 1: The timer must be less than 1 second off every minute.
Link:
https://drive.google.com/file/d/1i0nz7VHzuMKnHqGsG2a4Mp_Cuqhi2TjU/view?usp=drivesdk

Engineering Requirement 2: The system must use connectors for every module used in the timer system, have a power disconnect switch, and not have any exposed conductors. Wires must be organized in split loom or other protective materials. All devices must be rated at least IP43 (https://en.wikipedia.org/wiki/IP_Code).
Link:
Connectors :
https://drive.google.com/file/d/1p44Ua1vaomNh-j51voObiZ6TRsVk1JBK/view?usp=drivesdk
Spray Test :
https://drive.google.com/file/d/15VZpl15f9soW9BWr5WyWIu5odOfXeZce/view?usp=drivesdk

Engineering Requirement 3: Every switch and potentiometer on the user interface should have a label that can be read from three feet away by at least 2 people other than the project designer.
Link:
https://drive.google.com/file/d/1P3HIV99WSHvBftcdseaxI-Qy2W2KVENA/view?usp=drivesdk

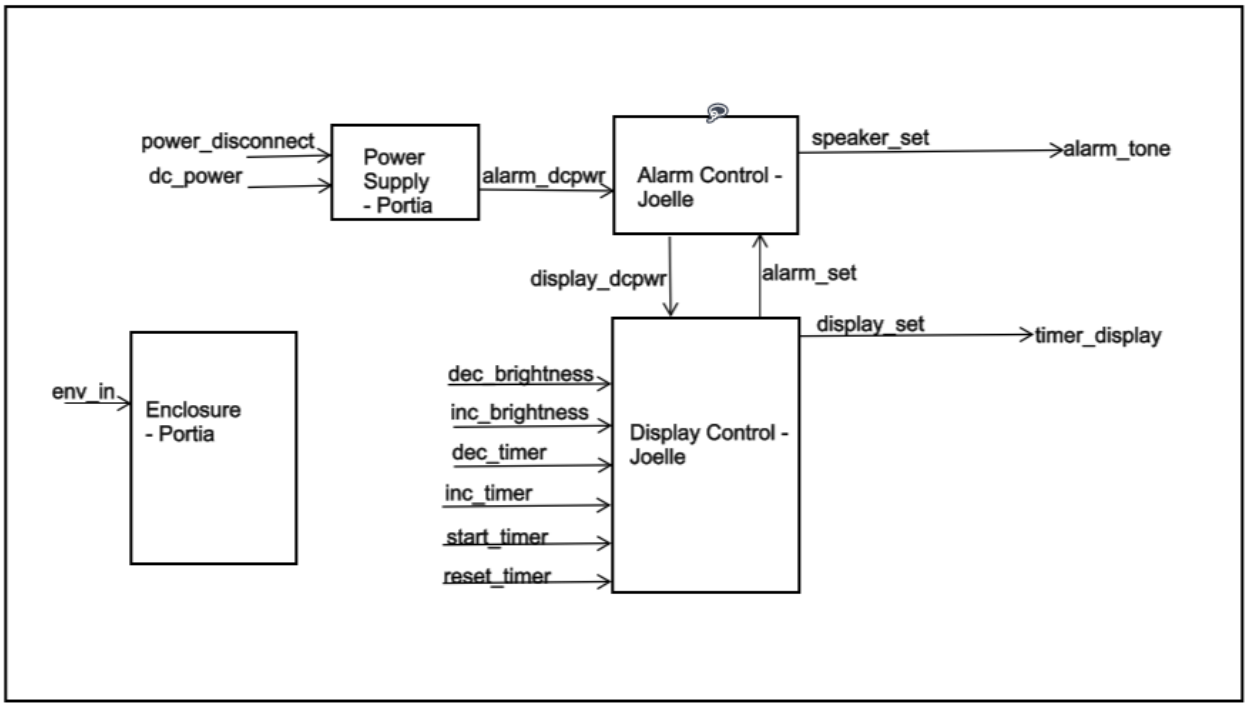
Engineering Requirement 4: The alarm should be 440 Hz +/- 1 Hz.
Link:
https://drive.google.com/file/d/1dUmiJVezMH5NVgN7b5MF1Amdgs1z2jT8/view?usp=drivesdk

Engineering Requirement 5: The LEDs on the timer display must have 3 brightness levels. The brightness level will be selected by a switch, potentiometer, or photosensor.
Link:
https://drive.google.com/file/d/1hv0mS8Agf8gPOp8n2bvS_ikvjNEp2RgJ/view?usp=drivesdk

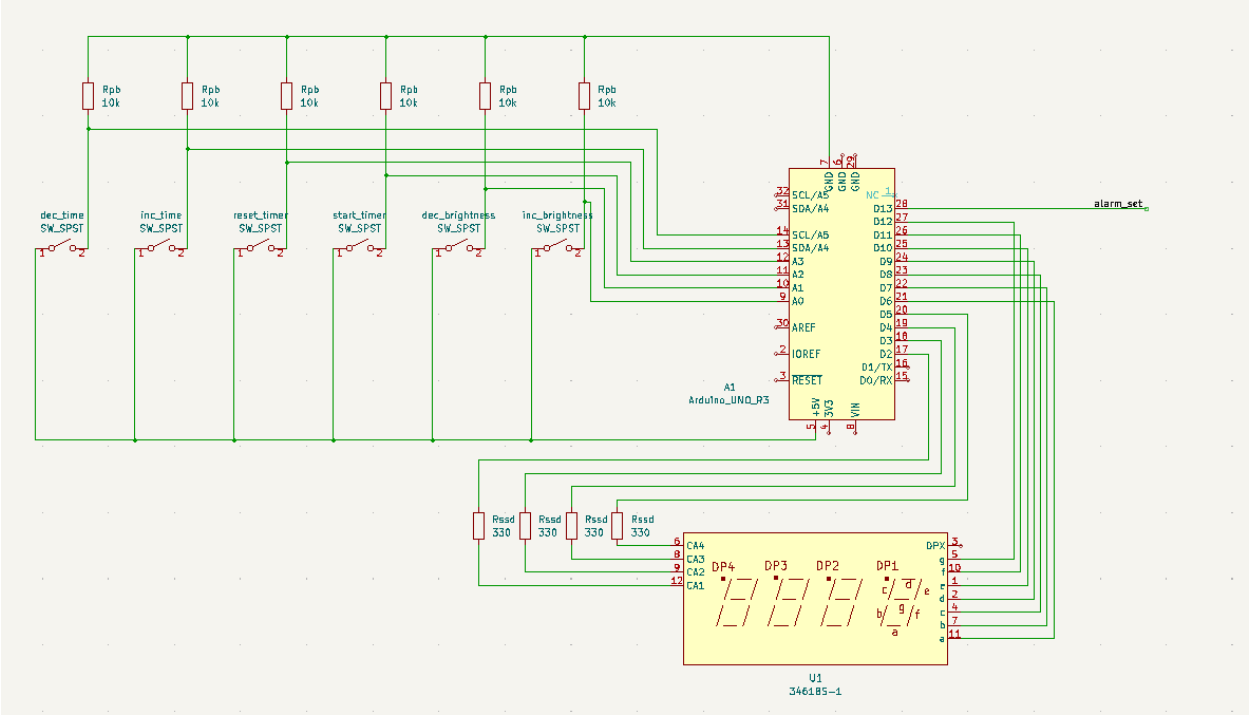
Additional Engineering Requirement: The system will play an alert sound to indicate that 10 seconds of the countdown remains.
Link:
https://drive.google.com/file/d/1GnJTSgQUnQSWmj4RXbF9yE8vgYzLhFGy/view?usp=drivesdk

Additional Artifacts:

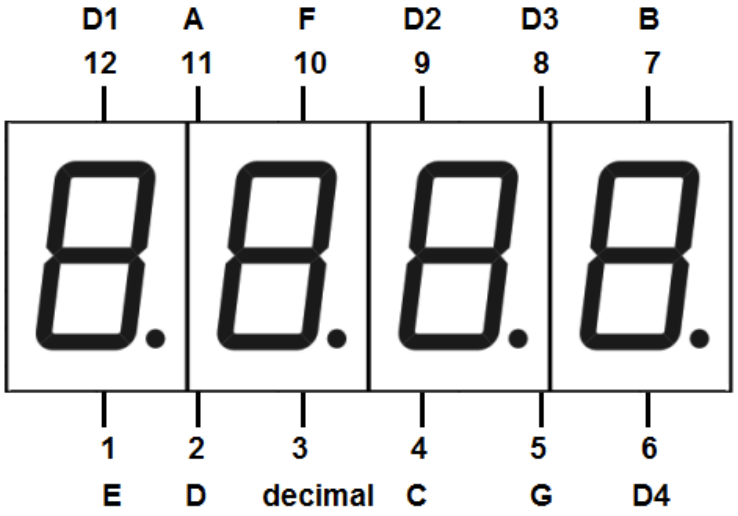
Top Level Diagram:



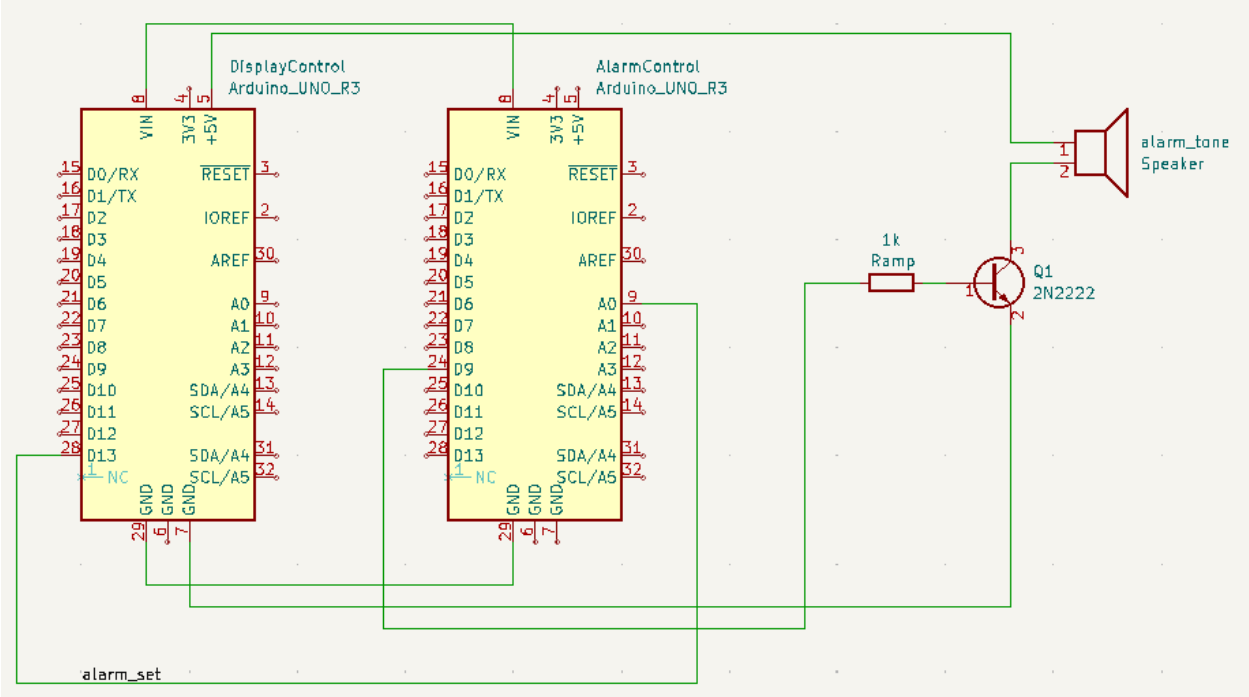
Display Control Block Wiring Diagram:



7 Segment Display Pinout (3461BS1):



Alarm Control Block Wiring Diagram:



Display Control Block Code (C):

```
#include"SevSeg.h"
SevSeg sevseg;
int inc_brightness = A0; //Light adjustment switch 1
int dec_brightness = A1; //Light adjustment switch 2
int start_timer = A2; //Start Timer PB
int reset_timer = A3; //Reset Timer PB
int inc_time = A4;  //Increment countdown PB
int dec_time = A5;  //Decrement countdown PB
int alarmState = 13; //Alarm control pin

bool timerRunning; //Indicates if the timer is countingdown
long startTime = 0; //The initial time for the countdown
int brightness;  //The brightness level of the display

const long secInterval = 1000;
const long toneInterval = 500;
const long displayInterval = 150;
long prevTimerTime = 0;
long prevToneTime = 0;
long prevDisplayTime = 0;


voidsetup()
{
  //Setup input and output pins
 pinMode(inc_time, INPUT);
 pinMode(dec_time, INPUT);
 pinMode(start_timer, INPUT);
 pinMode(reset_timer, INPUT);
 pinMode(inc_brightness, INPUT);
 pinMode(dec_brightness, INPUT);
 pinMode(alarmState, OUTPUT);

  //7 Segment Display Setup
  byte numDigits = 4;
  byte digitPins[] = {2, 3, 4, 5};
  byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12};
 bool resistorsOnSegments = false; // 'false' means resistors are on digit pins
 byte hardwareConfig = COMMON_ANODE; // See README.md for options
 bool updateWithDelays = false; // Default 'false' is Recommended
  bool leadingZeros = true; // Use 'true' if you'd like to keep the leading zeros
 bool disableDecPoint = true; // Use 'true' if your decimal point doesn't exist or isn't connected. Then, you
only need to specify 7 segmentPins[]

sevseg.begin(hardwareConfig,numDigits,digitPins,segmentPins,resistorsOnSegments,
updateWithDelays,leadingZeros,disableDecPoint);

sevseg.setBrightness(0);
}


voiddisplayTime(longusrTime)
{
 //Break usrTime into minutes and seconds
  int minute = 100 * (usrTime/60);
  int second = usrTime % 60;

  int countdownTime = minute + second;

ievseg.setNumber(countdownTime,0);
sevseg.refreshDisplay();
}


voidadjustLight()
{
```

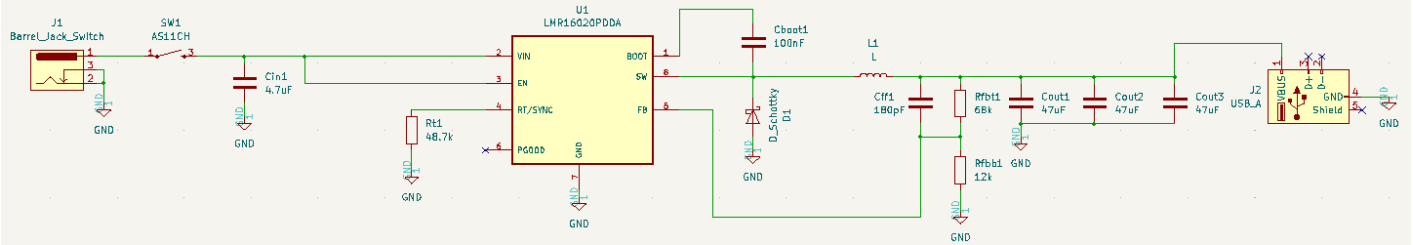
```
 //Increase or decrease brightness depending on user input
 if (digitalRead(inc_brightness) == HIGH && digitalRead(dec_brightness) == LOW
     && brightness < 100)
  {
       brightness += 100;
       delay(500);
  }
 if (digitalRead(dec_brightness) == HIGH && digitalRead(inc_brightness) == LOW
     && brightness > -100)
  {
     brightness -= 100;
     delay(500);
  }
evseg.setBrightness(brightness);
sevseg.refreshDisplay();
}


intstartTimer(longstartTime)
{
 startTime++;
 while(startTime >= 0)
  {
   //Decrease countdown by 1 second per second
   timerRunning = 1;
   long currentTimerTime = millis();
  if(currentTimerTime - prevTimerTime >= secInterval)
      {
       startTime--;
      prevTimerTime = currentTimerTime;
      }

   long currentToneTime = millis();

   //Start beeping tone when countdown has 10 seconds remaining
   if(startTime <= 10 && startTime > 0)
    {
    if(currentToneTime - prevToneTime >= toneInterval)
      {

      if(digitalRead(alarmState) == HIGH)
        digitalWrite(alarmState, 0);
      else if(digitalRead(alarmState) == LOW)
        digitalWrite(alarmState, 1);

      prevToneTime = currentToneTime;
      }
    }

   //Start constant tone when countdown has 0 seconds remaining
   if(startTime == 0)
    {
     //Play alarm
    digitalWrite(alarmState, 1);
     timerRunning = 0;

    while(digitalRead(reset_timer) == LOW)
       displayTime(0);

    if(digitalRead(reset_timer) == HIGH && digitalRead(alarmState) == HIGH)  //Resets timer
      {
       timerRunning = 0;
      digitalWrite(alarmState, 0);   //Stop alarm
      delay(500);   //PB debouncer
       return 0;
      }
```

```
    }

  displayTime(startTime);
  }

}


voidloop()
{
 //Allow user to adjust countdown time when timer has not started/is not running
 long currentDisplayTime = millis();
 if(timerRunning != 1 && (currentDisplayTime - prevDisplayTime) >= displayInterval)
  {
  if(digitalRead(inc_time) == HIGH && digitalRead(dec_time) == LOW
     && startTime < 600)
      {
       startTime++;
      }
  else if(digitalRead(inc_time) == LOW && digitalRead(dec_time) == HIGH
          && startTime > 0)
      {
       startTime--;
      }

  prevDisplayTime = currentDisplayTime;
   displayTime(startTime);  //Display the time onto 7Seg
  }

 //Starts timer
 if(digitalRead(start_timer) == HIGH && timerRunning != 1)
  {
   startTimer(startTime);   //Start Timer
   delay(500);  //PB debouncer
  }

 //Allow user to adjust brightness of display when timer is not running
 if(timerRunning != 1)
   adjustLight();

}
```

Alarm Control Block Code (C):

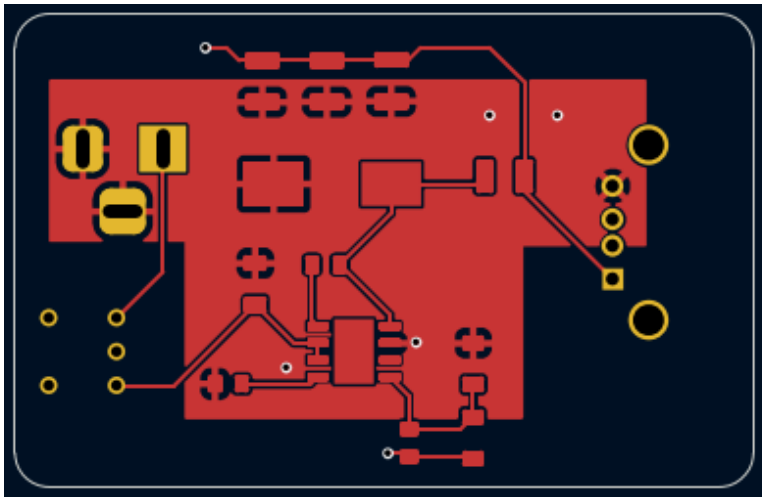
```
//Define  pin  variables
int  Alarm_Set  =  A0;
int  Alarm_Tone  =  9;

void  setup()
{
   //Setup  input  and  output  pins
   pinMode(Alarm_Set,  INPUT);
   pinMode(Alarm_Tone,  OUTPUT);
}

void  loop()
{
   //Check  if  alarm_tone  should  output  440Hz  or  0Hz
   if  (digitalRead(A0)  ==  HIGH)
   {
      tone(Alarm_Tone,  440,  100);
   }
   else
   {
      tone(Alarm_Tone,  0,  100);
   }
}
```

PCB Schematic:

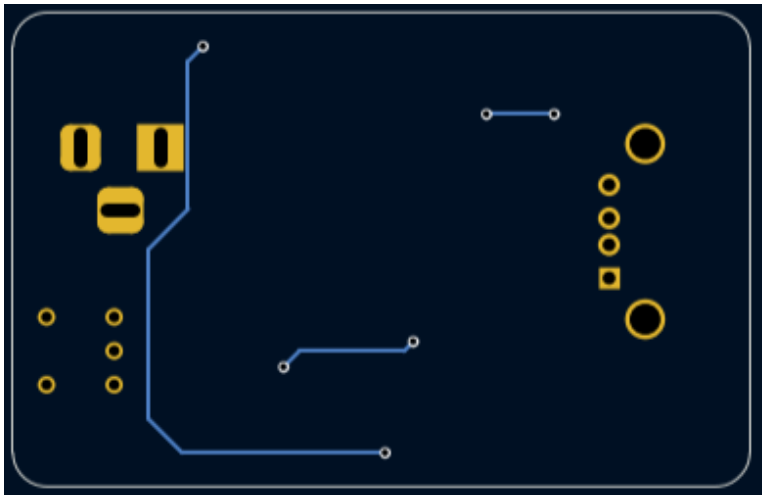


PCB Layers:

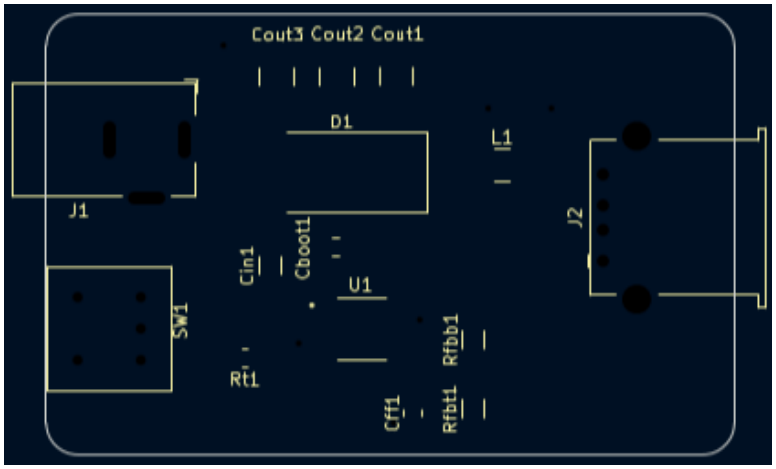
Front Copper -



Back Copper -



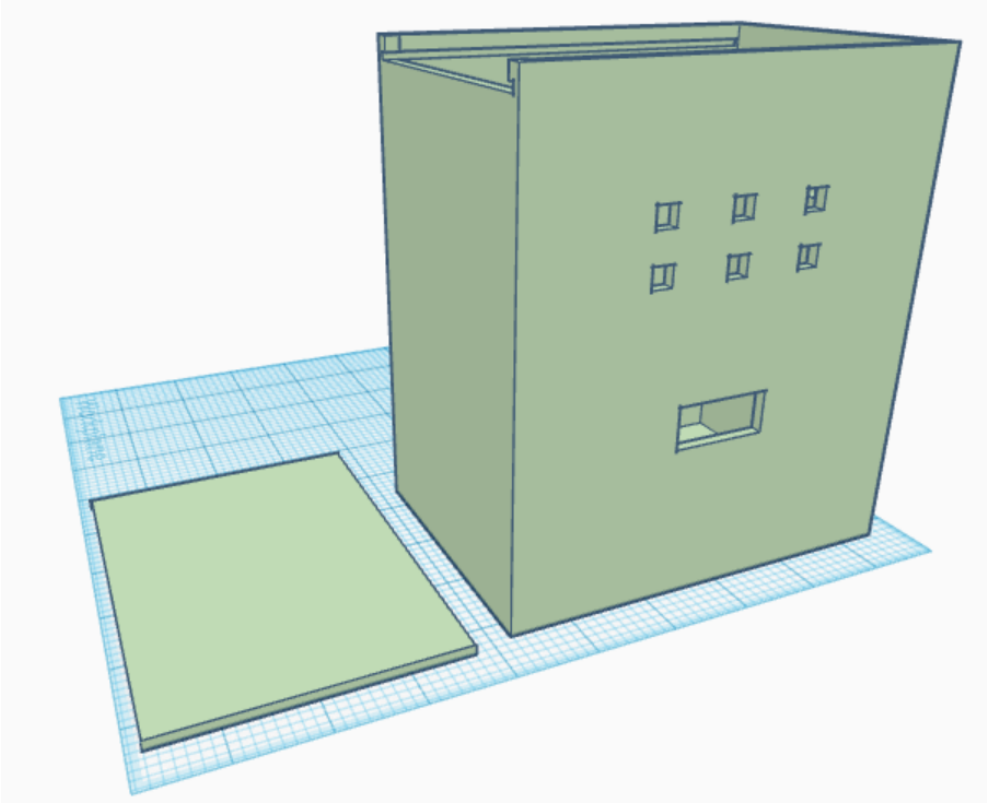
Front Silkscreen -



Bill of Materials:

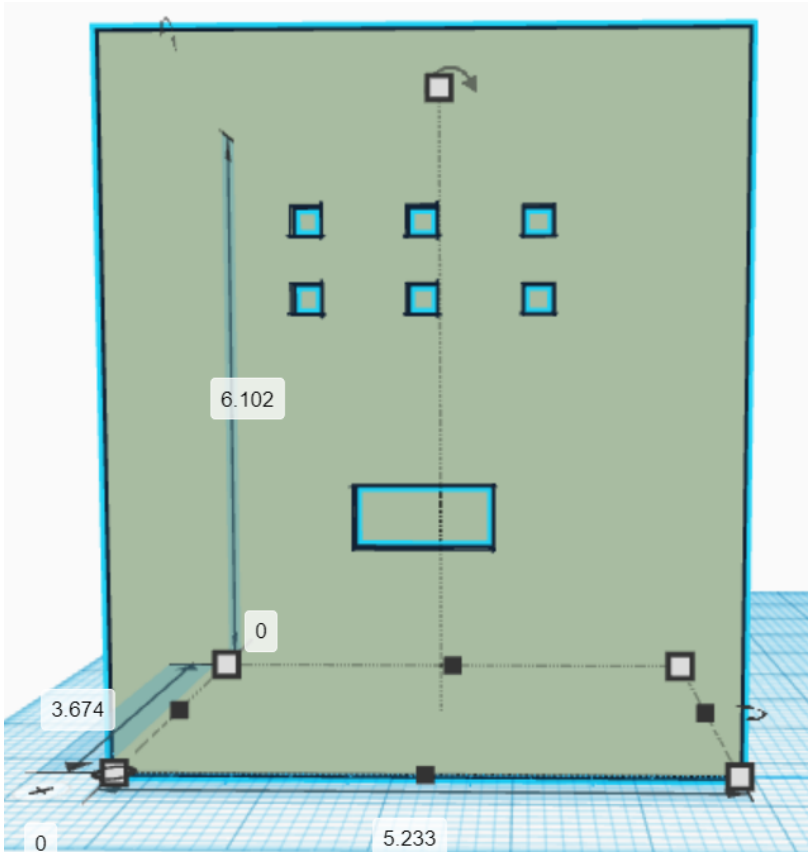
Component	Quantity
Arduino Uno	2
USB-A to USB-B cable	1
Student Designed PCB	1
12 V DC Wall Adapter	1
Pushbutton	6
Bundle of Jumper Wires	1
Male Header Pins x 40	2
2n2222 Transistor	1
8 Ohm Speaker	1
4 Digit 7 Segment Display	1
10k Ohm Resistor	6
330 Ohm Resistor	4
1k Ohm Resistor	1
Enclosure	1
Protoboard	3
¼“ Diameter Split Loom	1

Isometric Enclosure 3D Design

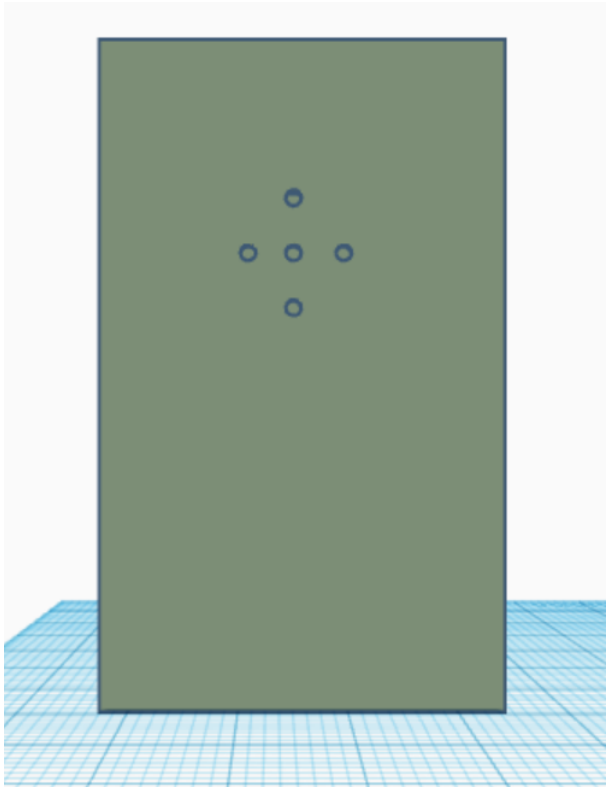


3D Enclosure Design Front View

- All units listed in inches



3D Enclosure Design Side View



3D Enclosure Design Back View