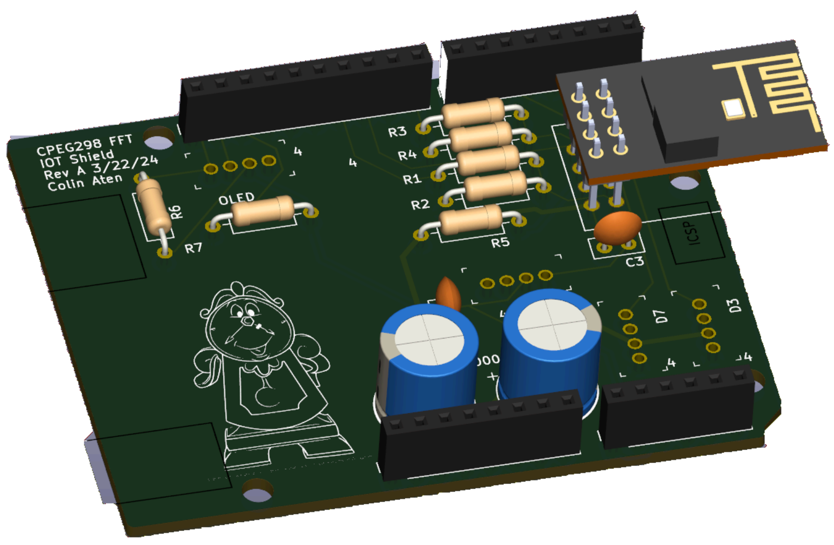


Image Rendered from KiCad

“Cogsworth”: Timer IoT Device

CPEG 298 S24 Semester

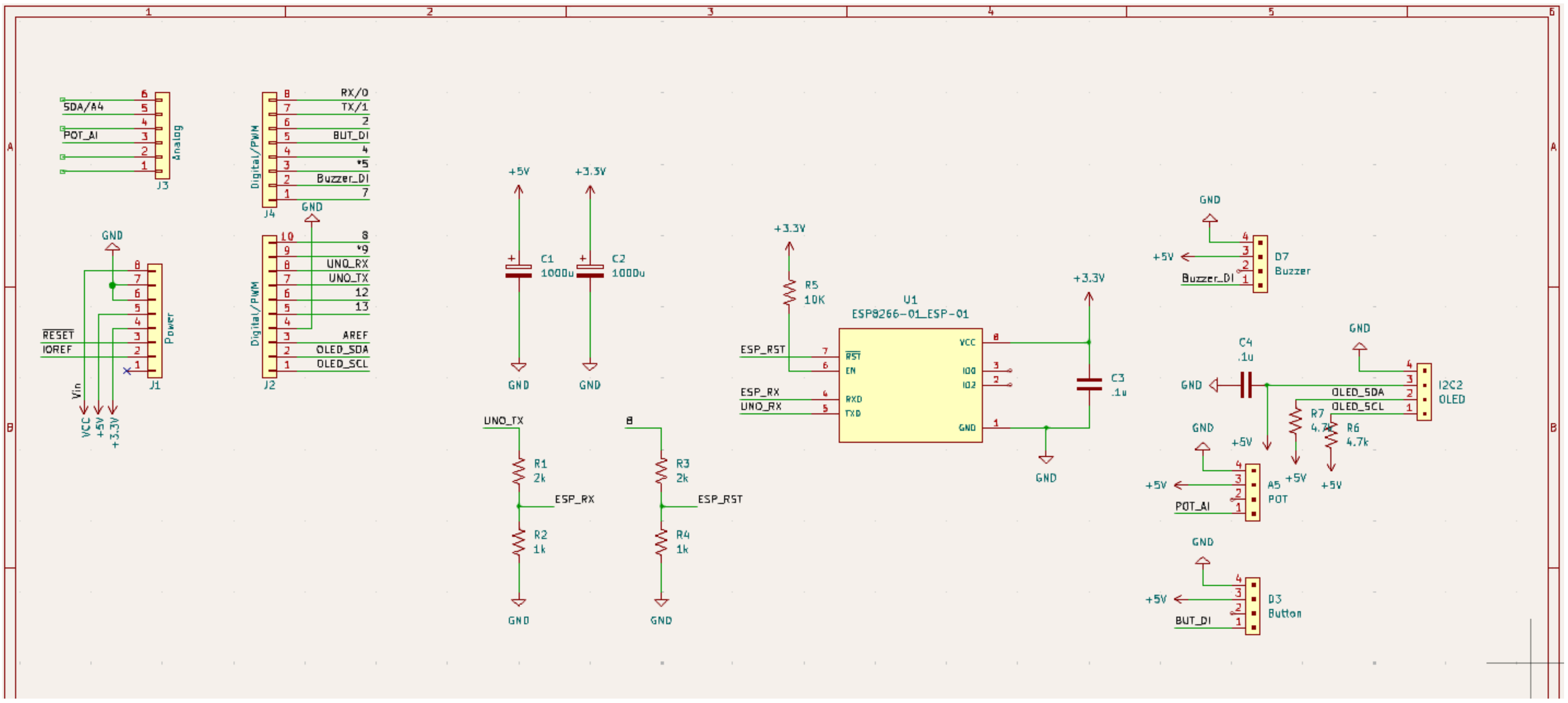
Colin Aten

Introduction / Project Description

The Timer IoT Device: “Cogsworth”, is designed as a simple adjustable timer that can be monitored from the cloud. The analog potentiometer is used to set the starting time. The analog button is used as an interrupt to trigger starting, pausing, and double clicking to reset the timer. The OLED display on the I2C2 is used to display the current time, the time being set, or the timer being finished. While the digital buzzer is used to communicate a reset as well as the timer reaching 0. The current time is sent to Adafruit.io using an ESP module.

The Circuit / Schematic

Below is shown the KiCad schematic (.kicad_sch). On the right hand side, the Grove connectors that are used to connect the digital buzzer, digital button, I2C2 OLED, and analog potentiometer are shown with their respective pin connections. The OLED required some extra capacitors and resistors in order to ensure it functioned properly and there was no signal distortion. The center displays the ESP-module and its pin connections along with the bulk

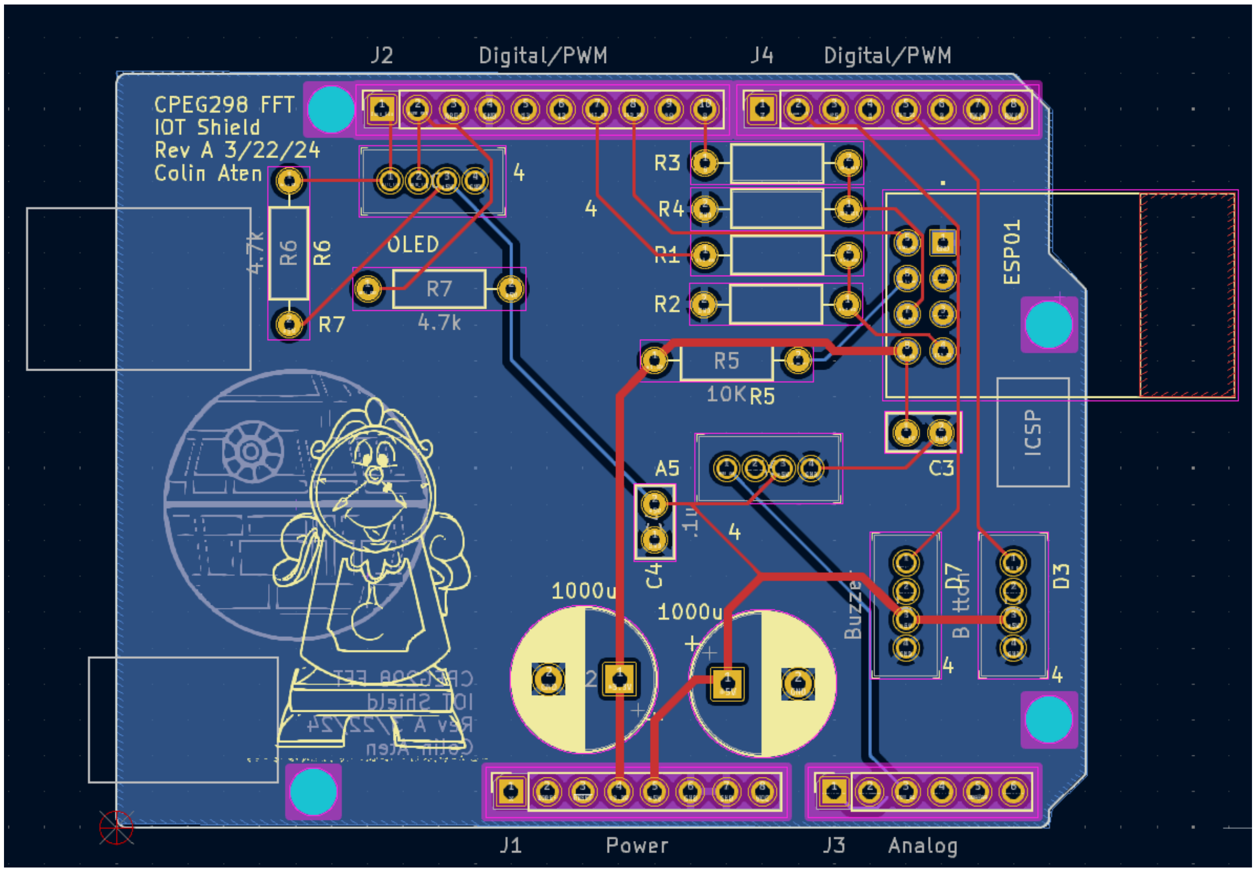


capacitors and voltage divider resistors. The left side houses the input and output pins for the arduino uno.

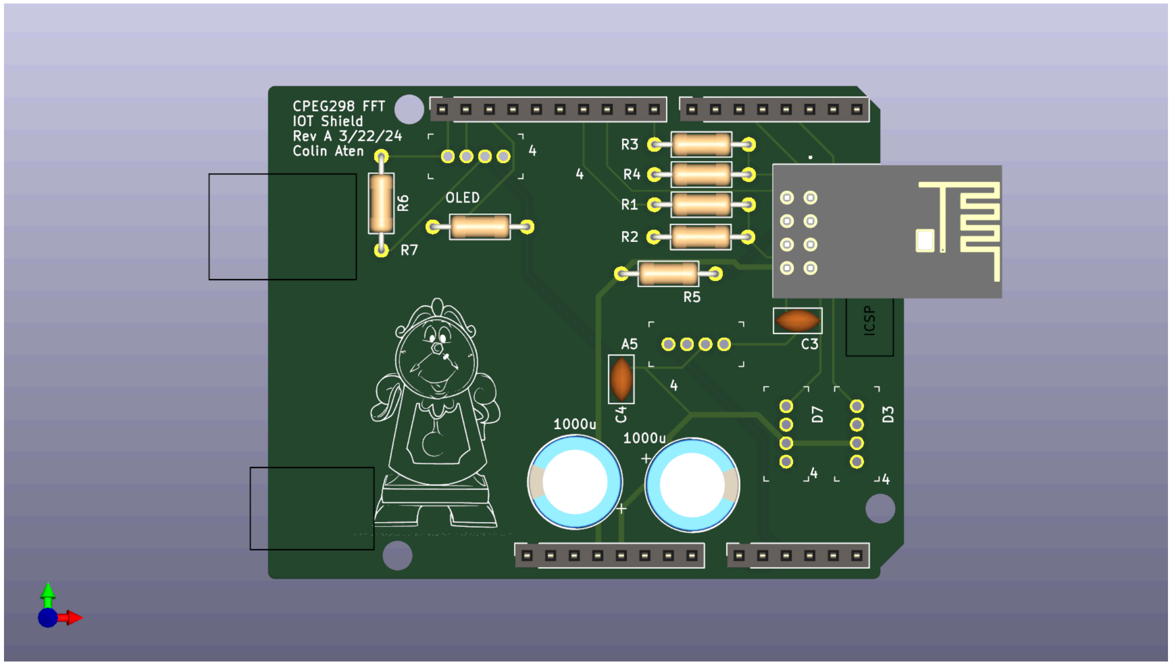
The ESP 8266 that was used for sending data to the internet is a 3.3V device. We used the 3.3V DC from the arduino for the VCC and EN/CH_PD pins. We used a 2kΩ / 1kΩ voltage divider to take the 5V UART signal that the Arduino sends to 3.3V. To get past the Arduino only having one hardware UART, a software serial UART was created using the GPIO10 and 11 pins. Special software replaced the AT Command firmware to allow us to use the ESP8266. We utilized another 2kΩ / 1kΩ to change the GPIO8 to reset the ESP-01S model, so that it boots up with the new firmware that is stored in the flash memory.

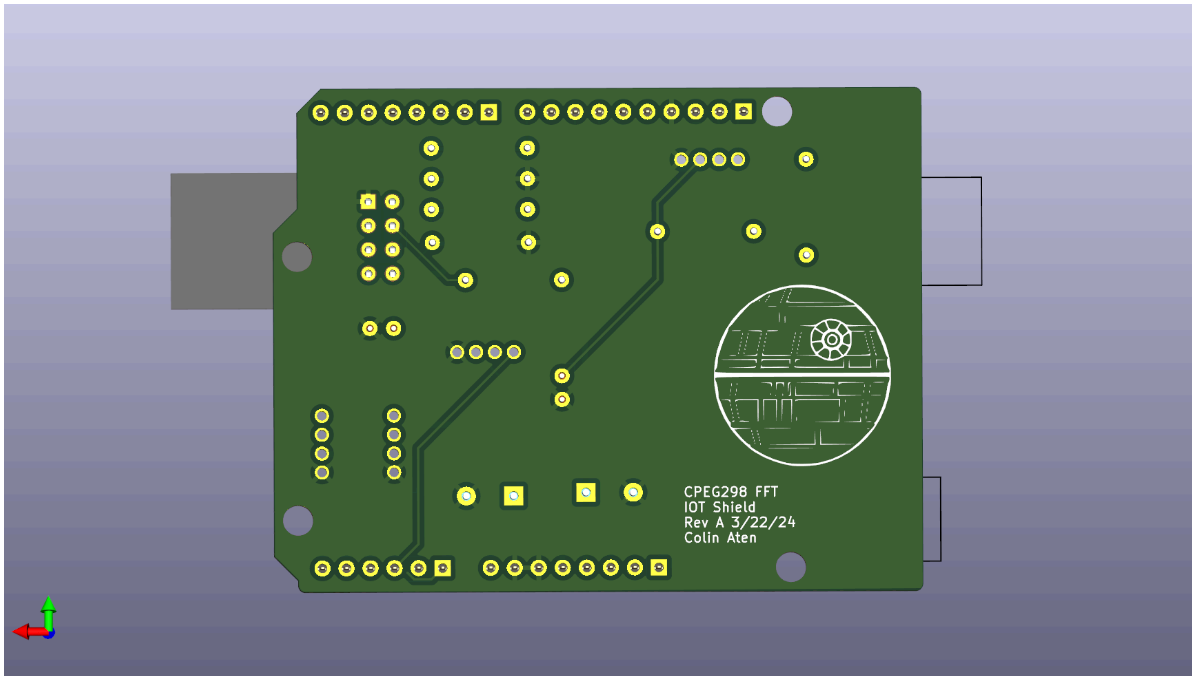
In order to connect up the OLED, button, potentiometer, and buzzer, four 4-pin, 2mm pitch Grove connectors were used. The Grove OLED Display 0.96” is a white display of 128x64 pixels that we used to show the time.

The Printed Circuit Board Design



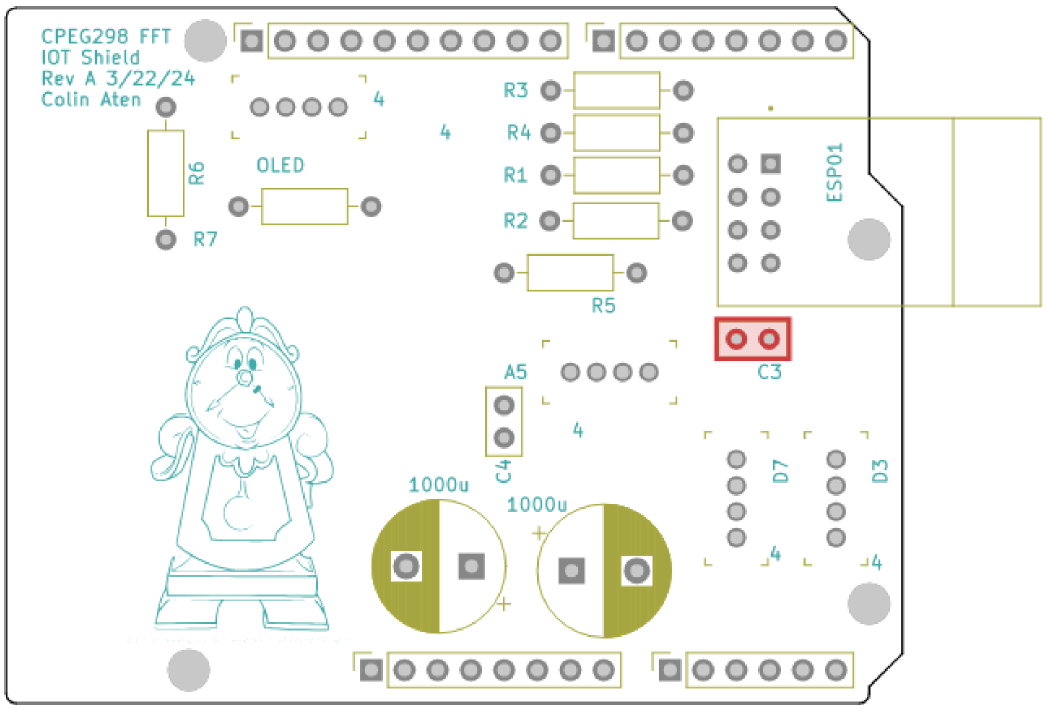
Above is a screen capture of the KiCad PCB design. A two-layer board with a ground plane on the bottom layer was used. The design is built off an Arduino UNO template from KiCad. The ESP and the voltage divider resistors are placed in the top right corner. The two digital grove connectors are placed in the bottom right corner. The OLED and its resistors are placed in the top left. Finally, in the right center is the analog grove connector for the potentiometer. The board was manufactured by JLCPCB (https://jlcpcb.com/). A render of the board is shown below.





Component List / Bill of Materials

Below is a component placement diagram – all the components are placed on the top side of the board. Additionally, the BOM table is displayed below. At the bottom is a table sourced from Digikey with prices and descriptions of the unique parts we used. All parts were sourced from the University of Delaware undergraduate labs.



□□□	Sourced	Placed	References	Value	Footprint
1	☐	☐	C1	1000u	CP_Radial_D10.0mm_P5.00mm
2	☐	☐	C2	1000u	CP_Radial_D10.0mm_P5.00mm
3	☐	☐	C3	.1u	C_Disc_D5.0mm_W2.5mm_P2.50mm
4	☐	☐	C4	.1u	C_Disc_D5.0mm_W2.5mm_P2.50mm
5	☐	☐	R1	2k	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
6	☐	☐	R3	2k	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
7	☐	☐	R2	1k	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
8	☐	☐	R4	1k	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
9	☐	☐	R6	4.7k	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
10	☐	☐	R7	4.7k	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
11	☐	☐	R5	10K	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
12	☐	☐	D3	Button	PinHeader_1x4_P2mm_Drill1mm
13	☐	☐	D7	Buzzer	PinHeader_1x4_P2mm_Drill1mm
14	☐	☐	U1	ESP8266-01_ESP-01	XCVR_ESP8266-01_ESP-01
15	☐	☐	A5	POT	PinHeader_1x4_P2mm_Drill1mm
16	☐	☐	I2C2	OLED	PinHeader_1x4_P2mm_Drill1mm
17	☐	☐	J1	Power	PinSocket_1x08_P2.54mm_Vertical
18	☐	☐	J2	Digital/PWM	PinSocket_1x10_P2.54mm_Vertical
19	☐	☐	J3	Analog	PinSocket_1x06_P2.54mm_Vertical
20	☐	☐	J4	Digital/PWM	PinSocket_1x08_P2.54mm_Vertical

Component	DidKey part #	Price/each ($)	Description
ESP8266-01/ESP-01	3647-Ai-ThinkerESP-01 SESP8266-ND	3.96	Ai-Thinker ESP-01S ESP8266 Wi-Fi
Grove OLED	1597-104020208-ND	5.50	GROVE - OLED DISPLAY 0.96" (SSD1)
Grove Button	1597-1285-ND	2.10	GROVE BUTTON
Grove Buzzer	1597-1346-ND	2.10	GROVE BUZZER
Grove Potentiometer	1597-1099-ND	3.20	GROVE ROTARY ANGLE SENSOR

Software / Arduino Code and Libraries

The full Arduino sketch is displayed in an Appendix following this report. Within the sketch, the following libraries were utilized.

```
C/C++
      #include "Arduino.h"         //Arduino library
      #include <SoftwareSerial.h> //Allows us to use two GPIO pins for a second
UART
      #include "Arduino_SensorKit.h" //Library for controlling the OLED
      #include "TimerInterrupt.h"    //Library for defining the timer interrupt
      #include "ISR_Timer.h"         //Library for setting up the ISR Timer
```

Pre-directives were used to define the pins and global variables were set.

```
C/C++
//Pin Assignments
#define BUTTON_PIN 3
#define BUZZER_PIN 6
#define POT_PIN A2
#define RST_PIN 8
```

```
//Global Variables
unsigned int current_time = 6000; // Variable to keep track of the time in seconds
unsigned int next_time=0; // variable to keep hold the set time from potentiometer
unsigned int old_time=0;  // variable to check if there is a change in the time
char state = 0;            // keeps track of our state for what to do in loop()
bool timer_running_flag = 0;      //variable for if timer is running
unsigned int pot_value = 0;     // Value for holding the value of the potentiometer
byte timer_flip=1;                // Flag to flip every time timer is run
bool debug = 0;                   // Variable to enable debug options or not
bool new_state_flag = 1;        // Variable to make new state debugging cleaner
bool display_debug = 0;           // To enable or disable debug in display
```

Within setup(), Timer 1 is setup and initialized as paused, the OLED is setup, the Serial is setup, the all the pins are set to appropriately be inputs or outputs, the button is attached to its interrupt handler, and the ESP board is setup to transmit data on one feed to Adafruit IO.

```
C/C++
// Set up Timer 1:
  ITimer1.init();
  ITimer1.attachInterruptInterval(TIMER1_INTERVAL_MS, TimerHandler1);
  ITimer1.pauseTimer();//start with timer paused

  // Set up OLED not flipped and with a font
  Oled.begin();
  Oled.setFlipMode(0);
  Oled.setFont(u8x8_font_chroma48medium8_r);

  // Begin Serial for debugging
  Serial.begin(115200);

  // Configure pins for the inputs and outputs
  pinMode(BUTTON_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(POT_PIN,INPUT);
  pinMode(RST_PIN, OUTPUT);

resp = espData("setup_feed=2, current_time", 2000, false);  //start the second
data feed
```

When the button is pressed, the button_interrupt() is run. First it checks for debounce to avoid errors, then it calls the button_press function to increment the state by 1. Additionally, if the button is pressed twice within a short period, the state is set to 5, causing the timer to reset.

To handle the timer counting down, Timer 1 is set up. When unpaused, the timer will decrement current time by 1 every 1000ms.

```C/C++
//Timer 1 definition
#define TIMER1_INTERVAL_MS 1000 //Timer interrupt runs every 1 second
void TimerHandler1() {
  current_time += -1;
}
```

The main loop behaves as a state machine. The initial state (state = 0) is simply for waiting for the first button press. In the first state (state =1), the value of the potentiometer is read and stored as next_time. That value is then displayed when it changes.

```C/C++
next_time = (analogRead(POT_PIN)*5.8);   // Get potentiometer value
if(old_time != next_time){
          display_time(next_time);
          old_time = next_time;
        }
```

The second state sends the value of next_time to current_time, starts the timer, then goes immediately to state 3.

```
C/C++
current_time = next_time;
ITimer1.resumeTimer();
timer_running_flag = 1;
state++;
new_state_flag = 1;
```

The third state runs continually. It changes the display when the value of current_time changes(every second) and sends the updated value of current_time to Adafruit (roughly every 2 seconds).

```
C/C++
if(old_time != current_time){ // Only update display if time changes
          old_time = current_time;
          display_time(current_time);
          espData("send_data=2," + String(current_time), 2000, false);  //send
feed to cloud
```

The fourth state runs once to pause or unpause the timer, then it immediately goes back to state 3.

```
C/C++
if(timer_running_flag){
          ITimer1.pauseTimer();
          timer_running_flag = 0;
        }else{
          ITimer1.resumeTimer();
          timer_running_flag = 1;
        }
```

The fifth state runs when the button is double-pressed. It sets the state back to 1 and clears the display, pausing the timer if necessary.

There is also a display_time function that serves to take an unsigned int (the time to be displayed) and break it into a char array where each digit corresponds to mm:ss. Then that char array is displayed on the OLED.

Another function is the buzzer, which takes in a byte for how many times it should buzz. Then the buzzer will play a tone for 100ms and then off for 100ms for as many loops as given.

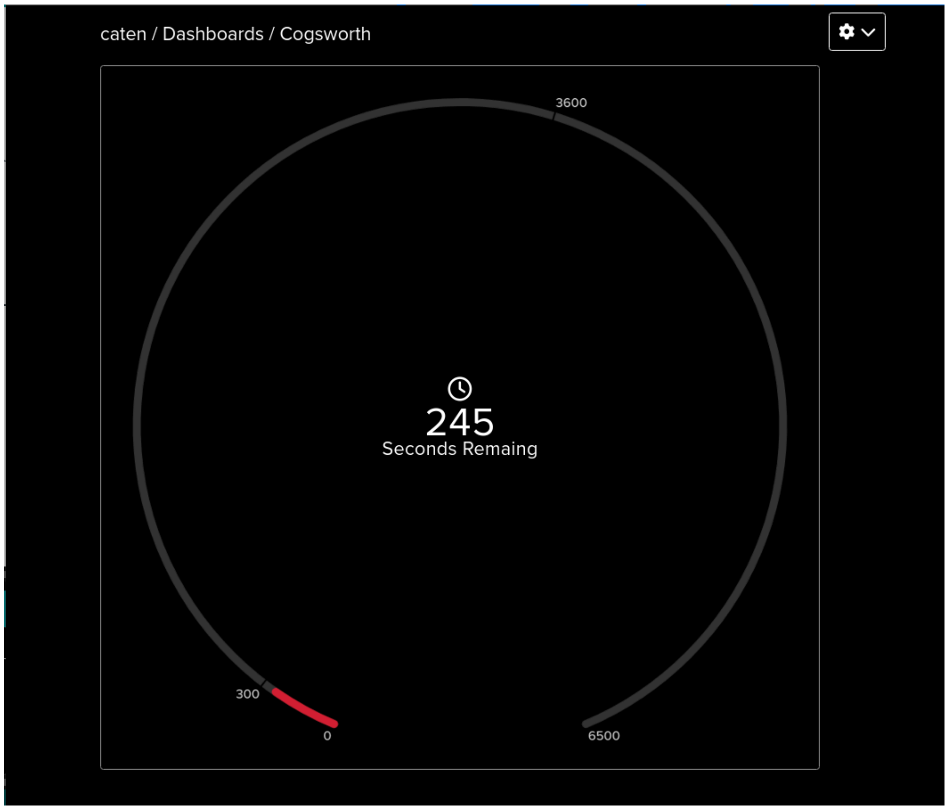
```
C/C++
tone(BUZZER_PIN, 2000); // Set the voltage to high and makes a noise
delay(100);             // Waits for 100 milliseconds
noTone(BUZZER_PIN);     // Sets the voltage to low and makes no noise
delay(100);
```

Finally, we have a helper function display_done that is run when the current_time=0, meaning the timer has counted all the way down. The function beeps the buzzer 5 times and displays a message on the OLED.

```
C/C++
Oled.print("Timer done! \n Get Moving!");
Oled.refreshDisplay();
buzzer(5);            //Beep Buzzer 5 times
```

Adafruit IO dashboard and cloud data

Below is a screenshot of the Adafruit IO dashboard that displays the current time. The slider turns red when there is less than 5 minutes left, meaning the timer is almost up. The dashboard is updated roughly every two seconds when the board is running.



Conclusion / Possible Changes

The device functions mostly as expected. The biggest hiccup is that sending data to the cloud seems to interfere with displaying the time, resulting in the device seeming to only display the time every 2 seconds. Though at times it will display back to back seconds. This is likely due to the time it takes for the esp to send data. One way to improve this device is to add additional separate timers running concurrently. For example, having a 5 minute timer and a 10 minute timer both counting down. Another way to improve the device would be to add a stopwatch feature, allowing the user to have the device count up instead of down. Overall, the device does serve to demonstrate data to the cloud, but not in a major way.

C/C++
```
//Colin Aten
//CPEG298 Arduino Uno Shield IoT device
//May 22 2024

#include "Arduino.h"
#include <SoftwareSerial.h>        //Allows us to use two GPIO pins for a second
UART
#include "Arduino_SensorKit.h"    //Library for controlling the OLED

//Library for using timers on Arduino Uno
//Timer setup
#define TIMER_INTERRUPT_DEBUG         1
#define _TIMERINTERRUPT_LOGLEVEL_     0

#define USE_TIMER_1     true

#include "TimerInterrupt.h"  //Library for defining the timer interrupt

#include "ISR_Timer.h" //Library for setting up the ISR Timer
#define USE_TIMER_1     true

//Headers for OLED
#include <Arduino.h>

// Setup for ESP
SoftwareSerial espSerial(10, 11);  //Create software UART to talk to the ESP8266
String IO_USERNAME = "caten";
String IO_KEY = "aio_omId10dcdZu53XPNmWFgAdC1qHOq";
String WIFI_SSID = "UD Devices";  //Only need to change if using other network,
eduroam won't work with ESP8266
String WIFI_PASS = "";            //Blank for open network
```

```
//Pin Assignments
#define BUTTON_PIN 3
#define BUZZER_PIN 6
#define POT_PIN A2
#define RST_PIN 8

//Function declarations
void display_time(unsigned int);
void button_interrupt();
void button_press();
void buzzer(byte);
void display_done();
void button_up();
void clear_disp();

//Global Variables
unsigned int current_time = 6000; // Variable to keep track of the time in
seconds
unsigned int next_time=0;         // variable to keep hold the set time from
potentiometer
unsigned int old_time=0;          // variable to check if there is a change in
the time
char state = 0;                   // keeps track of our state for what to do in
loop()
bool timer_running_flag = 0;
unsigned int pot_value = 0;       // Value for holding the value of the
potentiometer
byte timer_flip=1;                // Flag to flip every time timer is run

bool debug = 0;                   // Variable to enable debug options or not
bool new_state_flag = 1;          // Variable to make new state debugging
cleaner
bool display_debug = 0;           // To enable or disable debug in display
```

```
//Timer 1 definition
#define TIMER1_INTERVAL_MS 1000 //Timer interrupt runs every 1 second
void TimerHandler1() {
  current_time += -1;
  if(debug){
  Serial.print("Timer1 running");
  }
}

void setup() {
  // Set up Timer 1:
  ITimer1.init();
  ITimer1.attachInterruptInterval(TIMER1_INTERVAL_MS, TimerHandler1);
  ITimer1.pauseTimer();//start with timer paused

  // Set up OLED not flipped and with a font
  Oled.begin();
  Oled.setFlipMode(0);
  Oled.setFont(u8x8_font_chroma48medium8_r);

  // Begin Serial for debugging
  Serial.begin(115200);

  // Configure pins for the inputs and outputs
  pinMode(BUTTON_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(POT_PIN,INPUT);
  pinMode(RST_PIN, OUTPUT);

  // Setup the button interrupt
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN),button_interrupt,FALLING);

  // ESP Borad setup
  digitalWrite(RST_PIN,LOW);
  delay(500);
  digitalWrite(RST_PIN,HIGH);
  espSerial.begin(9600);  // set up software UART to ESP8266 @ 9600 baud rate
  Serial.println("setting up");
  String resp = espData("get_macaddr", 2000, true);      //get MAC address of
8266
  resp = espData("wifi_ssid=" + WIFI_SSID, 2000, true);  //send Wi-Fi SSID to
connect to
  resp = espData("wifi_pass=" + WIFI_PASS, 2000, true);  //send password for
Wi-Fi network
  resp = espData("io_user=" + IO_USERNAME, 2000, true);  //send Adafruit IO info
  resp = espData("io_key=" + IO_KEY, 2000, true);
  resp = espData("setup_io", 30000, true);  //setup the IoT connection
  if (resp.indexOf("connected") < 0) {
    Serial.println("\nAdafruit IO Connection Failed");
    while (1)
      ;
  }
  resp = espData("setup_feed=1,current_time2", 2000, false);  //start the data
feed
  resp = espData("setup_feed=2, current_time", 2000, false);  //start the second
data feed
  Serial.println("------ Setup Complete ----------");
}

void loop() {
  if(current_time<=1){              // Special Statement to check if time ran
out
    if(timer_running_flag){       // Pause timer if neccessary
        ITimer1.pauseTimer();
        timer_running_flag = 0;
    }
    display_done();               // Calls function to show timer is done
  }
  else{
    switch (state){
      case 0: //Default state, for waiting for input
        new_state_flag =1;
        break;
      case 1: // State for checking potentiometer to set value of timer
        next_time = (analogRead(POT_PIN)*5.8);   // Get potentiometer value
        if(old_time != next_time){
          display_time(next_time);
          old_time = next_time;
        }
        if(new_state_flag){                    // Send to Serial for debugging
          Serial.println("state 1");
          Serial.print("Next_time =");
          Serial.println(next_time);
          new_state_flag = 0;
        }

        break;
      case 2: // Send value of next_time to current_time and activate timer,
              //run once then go to watch state
        Serial.println("state 2");
        current_time = next_time;
        if(old_time != current_time){ // Only show on OLED if new
          old_time = current_time;
          display_time(current_time);
        }
        ITimer1.resumeTimer();
        timer_running_flag = 1;
        state++;
        new_state_flag = 1;
        break;
      case 3: // Watch state, timer and display are running and waiting for
button
```

```
        if(new_state_flag){
          Serial.println("state 3");
          new_state_flag = 0;
        }
        if(old_time != current_time){ // Only update display if time changes
          old_time = current_time;
          display_time(current_time);
          espData("send_data=2," + String(current_time), 2000, false);  //send
feed to cloud
        }
        break;
      case 4: // Paused state, pause or resume timer (flip it), and send back to
state 3.
        Serial.println("state 4");
        if(timer_running_flag){
          ITimer1.pauseTimer();
          timer_running_flag = 0;
        }else{
          ITimer1.resumeTimer();
          timer_running_flag = 1;
        }
        new_state_flag = 1;
        state = 3;

        break;
      case 5: // State for resetting timer on DOUBLE press, sends to state 1
        Serial.println("state 5");
        if(timer_running_flag){
          ITimer1.pauseTimer();
          timer_running_flag = 0;
        }
        current_time = 2;
        state = 1;     //Sets state to starting state
        buzzer(1);     //Beeps buzzer once
        clear_disp();  //Sets the Oled to all spaces
        break;
    }
  }
}

void display_time(unsigned int t) {
  char seperated_time[9] = "";
// Calculate minutes and seconds
  if(debug){
    Serial.print("value passed into disply_time:  ");
    Serial.println(t);
  }
  int minutes = t / 60;
  int seconds = t  % 60;

//Convert to chars for the display
  seperated_time[0] = (minutes/10) + 48;
  seperated_time[1] = (minutes % 10) + 48;
  seperated_time[2] = ':';
  seperated_time[3] = (seconds/10) + 48;
  seperated_time[4] = (seconds%10) + 48;
  seperated_time[5] = '\0';

  if(debug){
  Serial.println(seperated_time);
  display_debug = 0;
  }
  //Oled.print("               0");
  Oled.setCursor(0, 0);
  Oled.print(seperated_time);
  Oled.refreshDisplay();
```

```
}

void button_press(){
  // Function to call from button_interrupt in order to limit the logic in the
interrupt
  state++;   // increment the state
  if(debug){ // Send to Serial for debug
    Serial.println("Button pressed function");
  }
}

void button_interrupt(){ // Button interrupt function to run every time the
button is pressed
  noInterrupts();
  // variables for checking debounce and double press
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();
  unsigned long time_between = (interrupt_time - last_interrupt_time);

  // If interrupts come faster than 200ms, assume it's a bounce and ignore
  if ((time_between > 200)){
    if(time_between > 1000){
      // If the interrupt comes in twice within one second,
      button_press();
    }else{
      if(debug){
        Serial.println("long button press");
      }
      state = 5;
      current_time=1;
    }
  }

  last_interrupt_time = interrupt_time;
  interrupts();
  return;
}



void buzzer(byte i){
  while(i){
  tone(BUZZER_PIN, 2000); // Set the voltage to high and makes a noise
  delay(100);             // Waits for 100 milliseconds
  noTone(BUZZER_PIN);     // Sets the voltage to low and makes no noise
  delay(100);             // Waits for 100 milliseconds
  if(debug){
    Serial.print("buzzer beep");
  }
  i--;
  }
  return;
}

void display_done(){      // Function to run when timer is done
  // Display text
  Oled.setCursor(0,0);
  Oled.print("Timer done! \n Get Moving!");
  Oled.refreshDisplay();
  buzzer(5);            //Beep Buzzer 5 times
  delay(1000);
}

void clear_disp(){      //Sets the Oled to all spaces
  Oled.setCursor(0,0);
  Oled.print("                    \n                 ");
  Oled.refreshDisplay();
```

```
}


String espData(String command, const int timeout, boolean debug) {
  String response = "";
  espSerial.println(command);  //send data to ESP8266 using serial UART
  long int time = millis();
  while ((time + timeout) > millis()) {  //wait the timeout period sent with the
command
    while (espSerial.available()) {      //look for response from ESP8266
      char c = espSerial.read();
      response += c;
      Serial.print(c);  //print response on serial monitor
    }
  }
  if (debug) {
    Serial.println("Resp: " + response);
  }
  response.trim();
  return response;
}
```