Off-grid System Monitoring IOT Project

This project uses an off-grid property to test remote monitoring equipment. The idea is to eventually provide a blueprint of instructions to allow cottages, hobby farms, or other partially occupied locations to be monitored remotely. Eventually (hopefully) the instructions here will be detailed enough to allow anyone with basic knowledge of electronics and programming to build a complete monitoring systems with inexpensive off-the-shelf components.

- Water System Monitoring
- Home Power Consumption Monitoring
- Temperature Monitoring (indoor and outdoor)
- Simple Surveillance Camera
- Battery voltage and current draw monitor

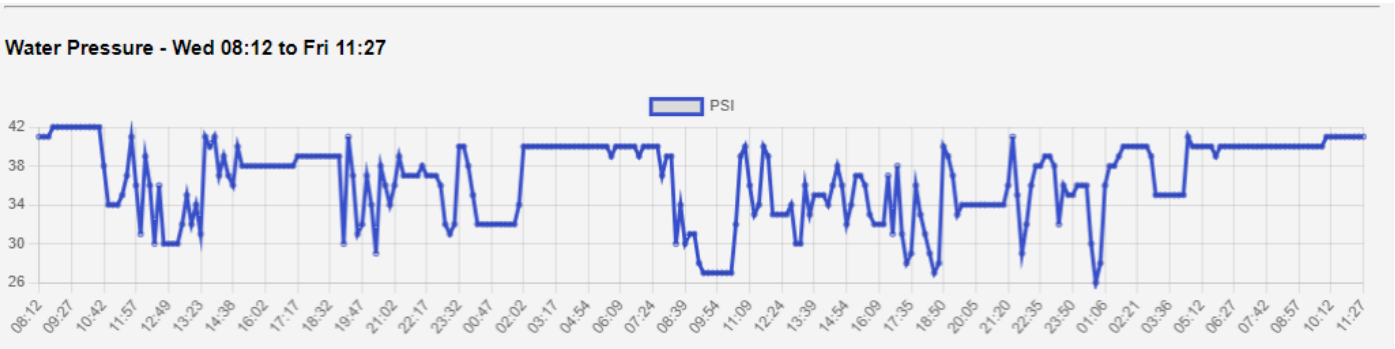
Water System Monitoring

For this project I needed a way to confirm the water pump at a remote location was functioning properly over time. The goal was to ensure the pressure is kept at a consistent 40 psi.

The solution was relatively inexpensive ($15 in parts) for an easily assembled unit using an automotive pressure sensor. The pressure sensor is wired to a ADS1115 module which then converts to voltage back to a pressure reading. The pressure reading is sent every 5 minutes to a remote server which logs the data and makes the readings available through a web page with line charts.

The server is an HP Elitedesk Pro mini pc with linux running through cloudflare (also free).

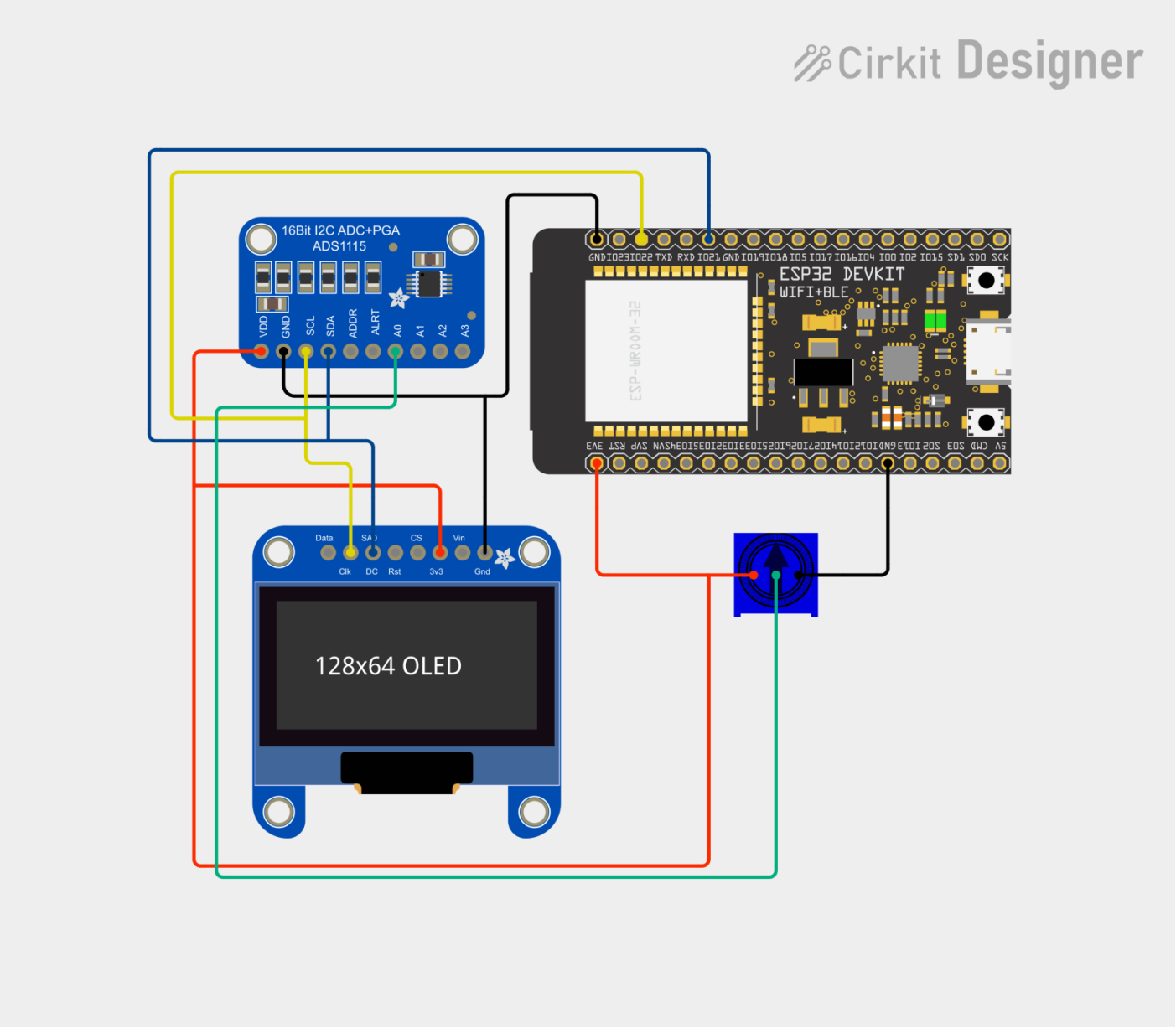
Chart on server showing pressure readings over last 2 days



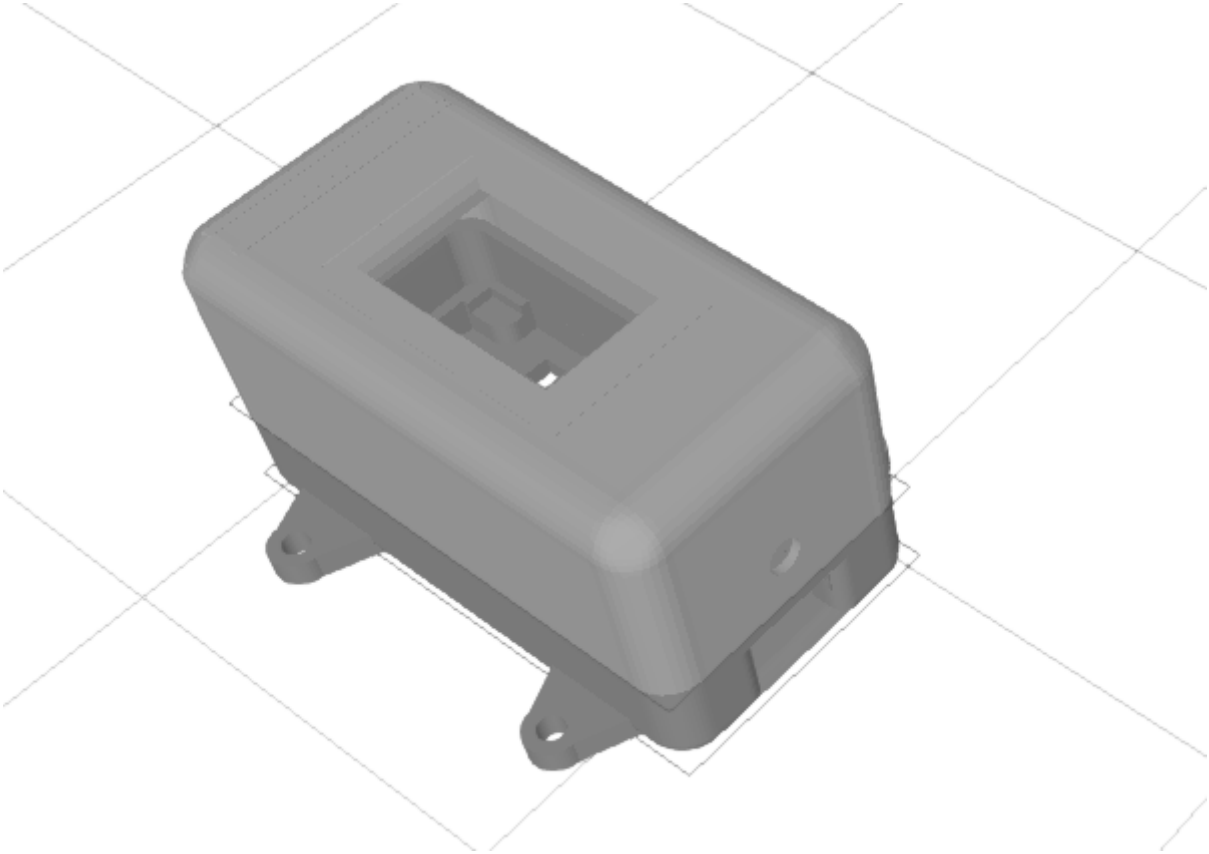
Components:

- ESP32 WROOM ($4.76 CAD) - https://www.aliexpress.com/item/4000471022528.html
- 5V 1/8NPT Pressure Transducer ($6.15 CAD) - https://www.aliexpress.com/item/1005005255271180.html
- 16 Bit I2C ADS1115 Module ($2.79 CAD) - https://www.aliexpress.com/item/32817162654.html

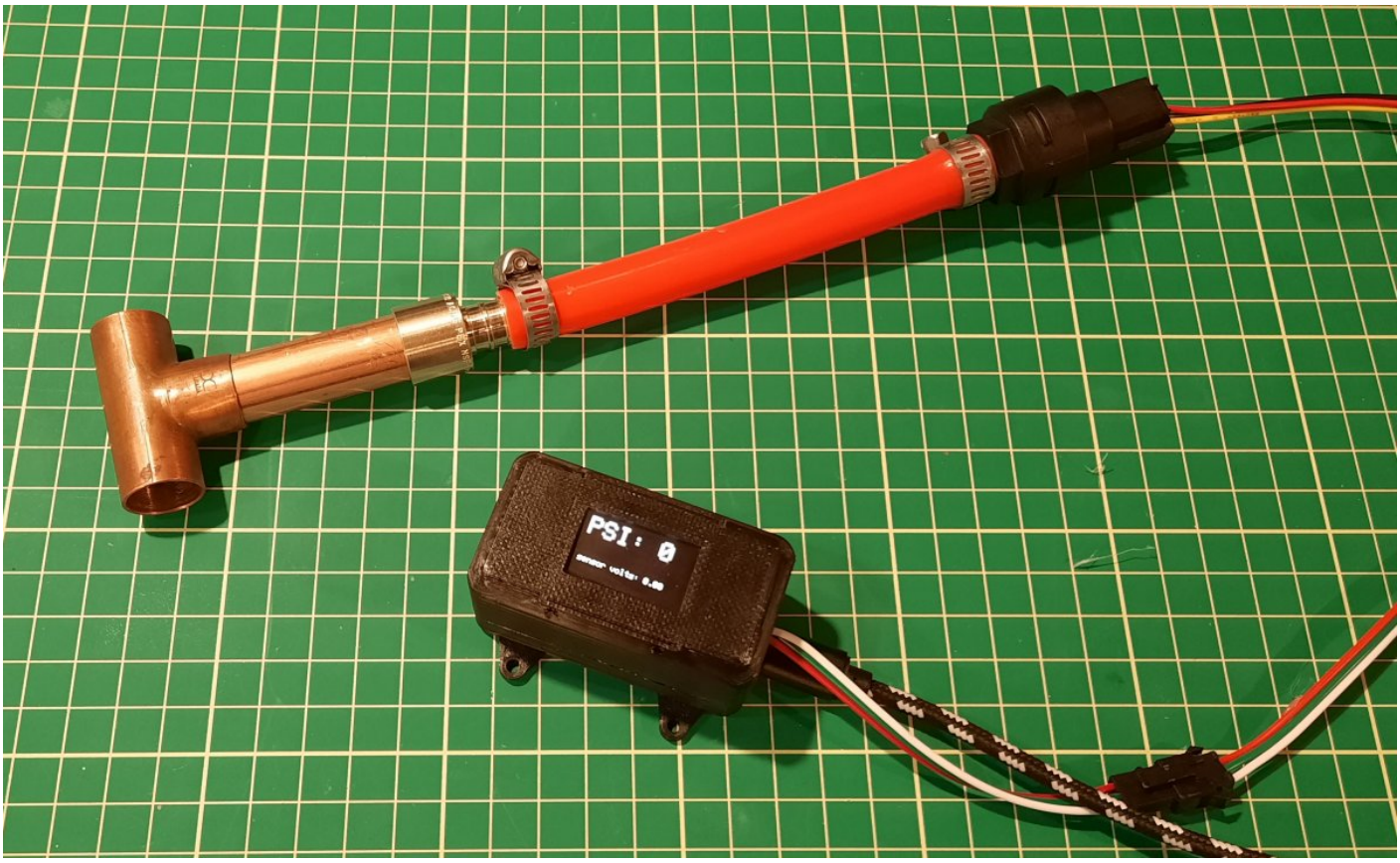
- OLED SSD1306 ($1.53 CAD) - https://www.aliexpress.com/item/32643950109.html

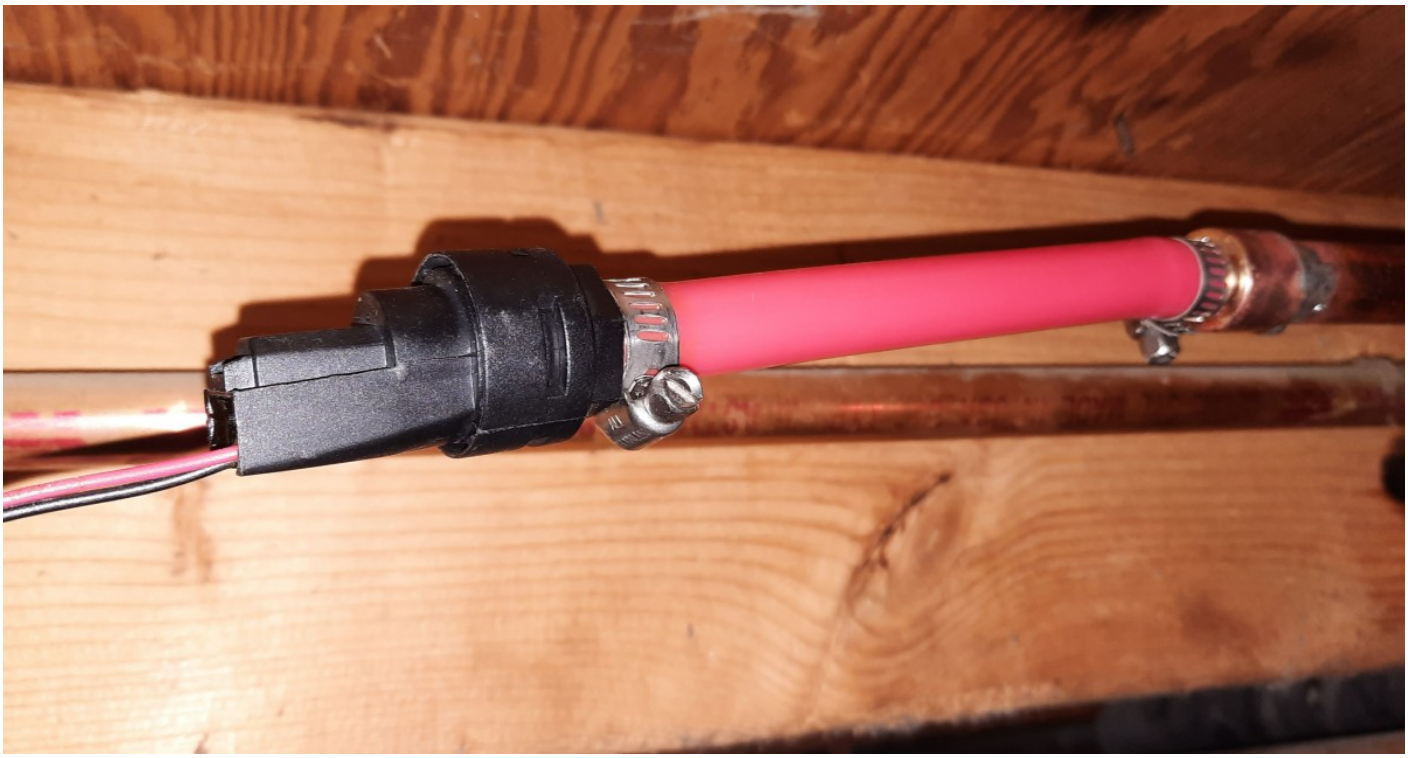


3d model for case designed for free with selfcad (www.selfcad.com)



https://www.thingiverse.com/thing:6481134





ESP32 code written in Arduino IDE (https://www.arduino.cc/en/software)

```
#include <SPI.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_ADS1X15.h>
#include <EEPROM.h>
#include "anniebitmap.h"

int activeConnection = 1;

// hansen
const String ssid1 = "xxx1";
const String password1 = "xxxxx";

// silo
const String ssid2 = "xxx2";
const String password2 = "xxxxx";

const String heartbeatUrl = "https://test.mysite.ca/silopower/heartbeat.php";
const String pressureUrl = "https://test.mysite.ca/silopower/set_water_pressure.php?data=";

#define EEPROM_SIZE 4
int eepromPingsAddress = 0;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnection = 1;

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define LOGO_HEIGHT 128
#define LOGO_WIDTH 64

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET - 1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, & Wire, OLED_RESET);

Adafruit_ADS1115 ads;
```

```
long secondsOfHour = 0;

const int secondsPerHour = 3600; // set to 60 for debugging
const int pulseRate = 1000; // loop runs once per second

const int serverSendInterval = 10; // 10 minutes between sending a voltage update to the server
const int samplesPerReading = 60 * serverSendInterval; // every x minutes send a sample to the server

int loopCount = 0;

String payload = "";
int httpCode = 0;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool debug = false; // when true server does not update
bool serverFailed = false;
int wifiConnectionAttempts = 0;

float basePressureVoltage = 0.46;
float totalledAveragePressure = 0;
float averagePressure = 0;

HTTPClient http;

void setup() {

  Serial.begin(115200);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }

  if (!ads.begin()) {
    Serial.println("Failed to initialize ADS.");
    while (1);
```

```
}

display.clearDisplay();
display.drawBitmap(0, 0, epd_bitmap_annie, 128, 64, 1);
display.display();
delay(3000);

//Init EEPROM
EEPROM.begin(EEPROM_SIZE);

activeConnection = EEPROM.read(eepromActiveConnection);

Serial.print("eepromActiveConnection :");
Serial.println(activeConnection);

if (isnan(activeConnection)) {
  activeConnection = 1;
}

if (activeConnection == 0) {
  activeConnection = 1;
}

Serial.print("activeConnection: ");
Serial.println(activeConnection);

float pingData = EEPROM.readFloat(eepromPingsAddress);
if (isnan(pingData)) {
  pingData = 0;
}
totalServerPings = pingData;
EEPROM.end();
EEPROM.begin(EEPROM_SIZE);
float pingFailData = EEPROM.readFloat(eepromFailedPingsAddress);
if (isnan(pingFailData)) {
  pingFailData = 0;
}
totalServerPingFails = pingFailData;
EEPROM.end();
```

```
  if (!connectToWiFi()) {
    delay(2000);
    WiFi.disconnect();
    delay(1000);
    if (activeConnection == 1) {
      activeConnection = 2;
    } else {
      activeConnection = 1;
    }
    connectToWiFi();
  }

  delay(2000); // Pause for 2 seconds

}

void loop() {

  secondsOfHour++;

  // after 24 hours reset this integer
  if (secondsOfHour > 86400) {
    secondsOfHour = 1;
    // Serial.println("restarting :");
    ESP.restart();
  }

  int16_t adc0;

  adc0 = ads.readADC_SingleEnded(0);
  float voltage = (adc0 * 0.1875) / 1000; //- basePressureVoltage);

  float normalizedVolatage = voltage - basePressureVoltage;

  if (normalizedVolatage < 0) {
    normalizedVolatage = 0;
  }

  float pressure = normalizedVolatage * (30 - (normalizedVolatage * 3));
```

```
if (pressure < 0) {

  pressure = 0;
}

totalledAveragePressure += pressure;

averagePressure = totalledAveragePressure / loopCount;



Serial.println("averagePressure: ");
Serial.println(averagePressure);



display.clearDisplay();
display.setTextSize(3);
display.setTextColor(WHITE);
display.setCursor(0, 0);
display.print("PSI: ");
display.println(pressure, 0);
display.println("");
display.setTextSize(1);
display.print("sensor volts: ");
display.println(normalizedVolatage, 2);
display.display();

//////////////////////
// SERVER RELAY

if (!debug && (loopCount >= samplesPerReading) || serverFailed) {

  loopCount = 0;
  totalledAveragePressure = 0;

  setWifiSleepMode(false);

  delay(2000);

  // do a heartbeat check to see if we are online...
  http.begin(heartbeatUrl);
```

```
    httpCode = http.GET();
    if (!httpCode > 0) {
      // wifi may not be alive yet so wait 3 seconds
      delay(3000);
    }

    String recordedPressure = String(averagePressure, 1);

    recordedPressure.trim();

    loopCount = 0;

    http.begin(pressureUrl + recordedPressure);
    httpCode = http.GET();
    if (httpCode > 0) {
      payload = http.getString();
      Serial.println("HTTP Response: " + payload);
      recordPingSucces();
    } else {
      recordPingFailure();
    }
    http.end();

    setWifiSleepMode(true);
  }

  loopCount++;

  delay(1000);

}

bool connectToWiFi() {

  String activeSsid = "";
  String activePassword = "";

  if (activeConnection == 1) {
    activeSsid = ssid1;
    activePassword = password1;
```

```
} else if (activeConnection == 2) {
  activeSsid = ssid2;
  activePassword = password2;
}

Serial.print("Connecting to WiFi: ");
Serial.println(activeSsid);

WiFi.begin(activeSsid, activePassword);

while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
  delay(500);
  Serial.print(".");
  wifiConnectionAttempts++;
}

wifiConnectionAttempts = 0;

if (WiFi.status() == WL_CONNECTED) {
  Serial.println("\nConnected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  wifiConnected = true;

  EEPROM.begin(EEPROM_SIZE);
  EEPROM.write(eepromActiveConnection, activeConnection);
  EEPROM.commit();
  EEPROM.end();

  Serial.print("set activeConnection to : ");
  Serial.println(activeConnection);

  return true;

} else {
  Serial.print("Connection to ");
  Serial.print(activeSsid);
  Serial.println(" failed. Trying alternative");
  return false;
}
```

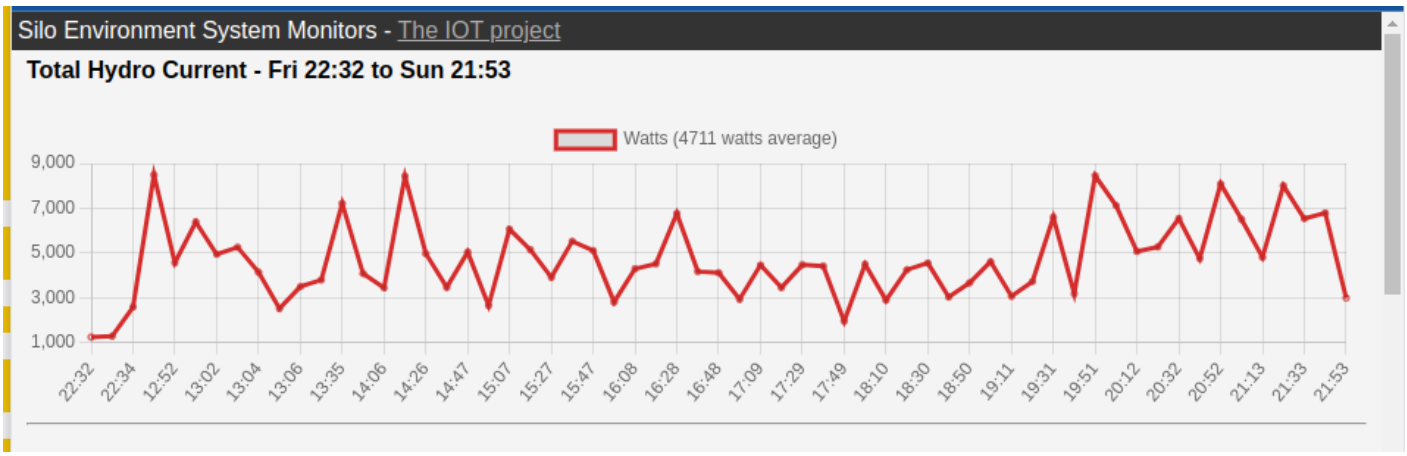
```
}

/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

  wifiSleeping = sleepMode;

  if (sleepMode) {
    WiFi.disconnect();
    WiFi.setSleep(true);
    delay(1000);
    Serial.print("sleep wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
  } else {
    WiFi.setSleep(false);
    WiFi.reconnect();
    delay(1000);
    Serial.print("awaken wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
    // Check if the connection is still active. if not trigger wait for it to come back online
    if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
      Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
      WiFi.disconnect();
      wifiPaused = true;
      wifiConnected = false;
      connectToWiFi();
    }
  }
}

/**
 * record server ping success in long term memory
 */
void recordPingSucces() {
  totalServerPings++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
```

```
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = true;
  serverFailed = false;
}

/**
 * record server ping fails in long term memory
 */
void recordPingFailure() {
  totalServerPingFails++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = false;
  serverFailed = true;
}

/**
 * ESP32 wifi card statuses
 * @param status
 * @return string
 */
String wl_status_to_string(wl_status_t status) {

  String response = "";

  switch (status) {
  case WL_NO_SHIELD:
    response = "WL_NO_SHIELD";
    break;
  case WL_IDLE_STATUS:
    response = "WL_IDLE_STATUS";
    break;
  case WL_NO_SSID_AVAIL:
    response = "WL_NO_SSID_AVAIL";
    break;
  case WL_SCAN_COMPLETED:
    response = "WL_SCAN_COMPLETED";
```

```
    break;
  case WL_CONNECTED:
    response = "WL_CONNECTED";
    break;
  case WL_CONNECT_FAILED:
    response = "WL_CONNECT_FAILED";
    break;
  case WL_CONNECTION_LOST:
    response = "WL_CONNECTION_LOST";
    break;
  case WL_DISCONNECTED:
    response = "WL_DISCONNECTED";
    break;
  }

  return response;
}
```

Home Power Consumption Monitoring

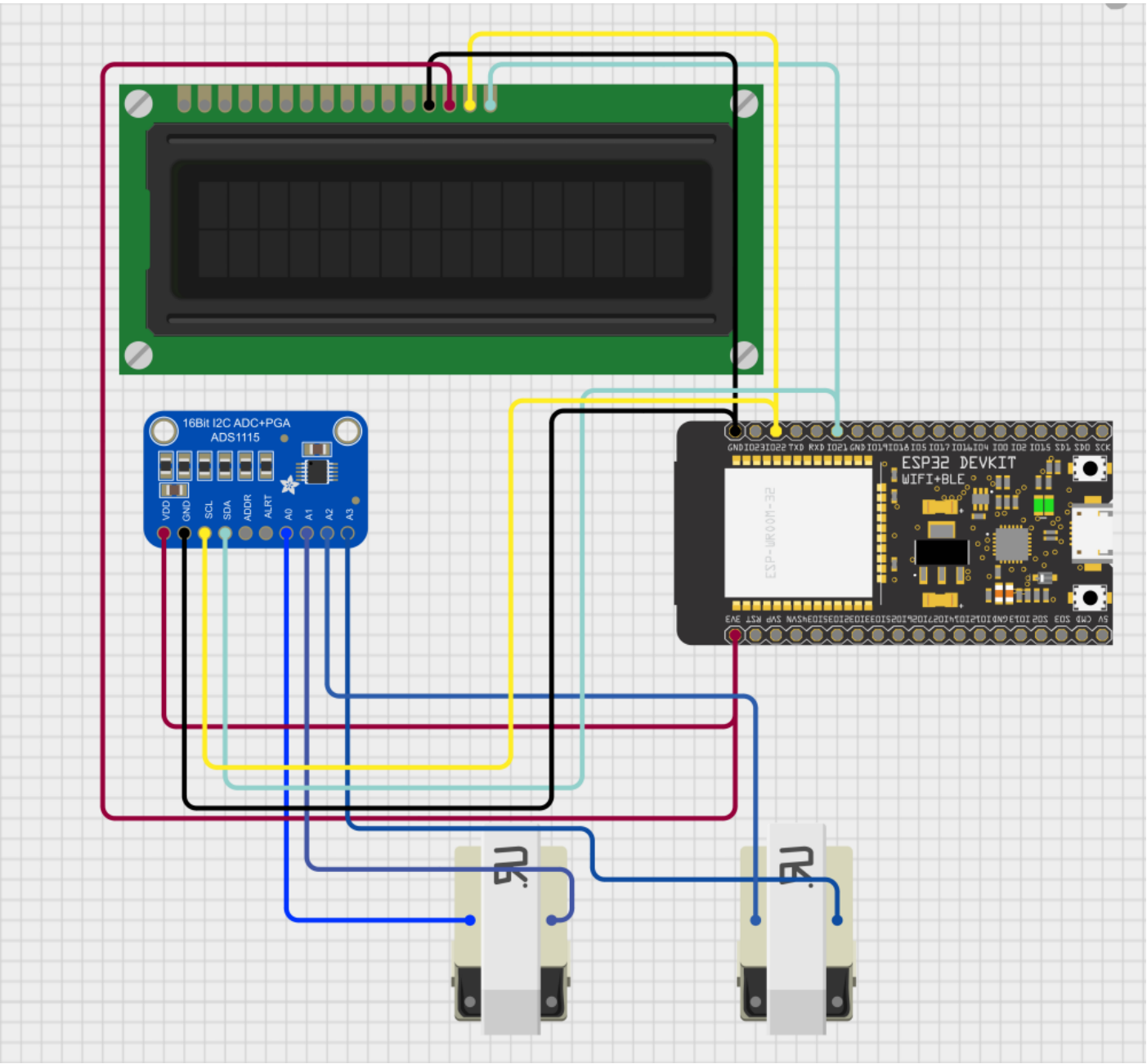
For this project the goal is to know exactly how much current is being used for the entire property. To collect this data I placed 2 SCT013 non-invasive split core current transformers around the phase 1 and 2 wires in the main electrical panel. Using some simple math this can convert the small voltage in the transformers into an accurate current reading.



Components (total cost $20 CAD):

- ESP32 WROOM ($5 CAD) - https://www.aliexpress.com/item/4000471022528.html
- 2 x SCT013 split core transducers ($8 CAD) - https://www.aliexpress.com/item/1005006318596840.html
- 16 Bit I2C ADS1115 Module ($3 CAD) - https://www.aliexpress.com/item/32817162654.html
- OLED SSD1306 ($1.50 CAD) - https://www.aliexpress.com/item/32643950109.html
- 3d printer filament ($1 CAD)

Wiring Diagram:



A clamp meter was used to confirm the calculated values were valid for a wide range of current (1 amp up to 60)

Relay unit mounted on the wall:

And the code to make it all happen:

```
#include <SPI.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
```

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_ADS1X15.h>
#include <EEPROM.h>
#include "splashscreenbitmap.h" // just make it look fun on bootup

int activeConnection = 1;

// network 1
const String ssid1 = "xxx";
const String password1 = "xxxxxxx";

// network 2
const String ssid2 = "xxx";
const String password2 = "xxxxx";

const String heartbeatUrl = "https://xxx.xxx.com/silopower/heartbeat.php";
const String currentUrl = "https://xxx.xxx.com/silopower/set_hydro_current.php?data=";

#define EEPROM_SIZE 4
int eepromPingsAddress = 0;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnection = 1;

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define LOGO_HEIGHT 128
#define LOGO_WIDTH 64

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET - 1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, & Wire, OLED_RESET);

Adafruit_ADS1115 ads;

long secondsOfHour = 0;

const int pulseRate = 1000; // loop runs once per second
```

```
const int serverSendInterval = 10; // 10 minutes between sending a pressure update to the server
const int samplesPerReading = 12 * serverSendInterval; // reading current takes 5 seconds so 60 / 5 = 12

int loopCount = 0;

String payload = "";
int httpCode = 0;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool debug = false; // when true server does not update
bool serverFailed = false;
int wifiConnectionAttempts = 0;

float basePressureVoltage = 0.46;
float totalledAveragePressure = 0;
float averagePressure = 0;

const float FACTOR = 9;

const int secondsPerHour = 3600; // set to 60 for debugging

HTTPClient http;

void setup() {

  Serial.begin(115200);


  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }

  if (!ads.begin()) {
    Serial.println("Failed to initialize ADS.");
    while (1);
  }
```

```
// ads.setGain(GAIN_FOUR);

 display.clearDisplay();
 display.drawBitmap(0, 0, epd_bitmap_annie, 128, 64, 1);
 display.display();
 delay(3000);

 EEPROM.begin(EEPROM_SIZE);

 activeConnection = EEPROM.read(eepromActiveConnection);

 Serial.print("eepromActiveConnection :");
 Serial.println(activeConnection);

 if (isnan(activeConnection)) {
   activeConnection = 1;
 }

 if (activeConnection == 0) {
   activeConnection = 1;
 }

  if (activeConnection > 2) {
   activeConnection = 2;
 }

 Serial.print("activeConnection: ");
 Serial.println(activeConnection);

 float pingData = EEPROM.readFloat(eepromPingsAddress);
 if (isnan(pingData)) {
   pingData = 0;
 }
 totalServerPings = pingData;
 EEPROM.end();
 EEPROM.begin(EEPROM_SIZE);
 float pingFailData = EEPROM.readFloat(eepromFailedPingsAddress);
 if (isnan(pingFailData)) {
   pingFailData = 0;
```

```
  }
  totalServerPingFails = pingFailData;
  EEPROM.end();

  if (!connectToWiFi()) {
    delay(2000);
    WiFi.disconnect();
    delay(1000);
    if (activeConnection == 1) {
      activeConnection = 2;
    } else {
      activeConnection = 1;
    }
    connectToWiFi();
  }

  delay(2000); // Pause for 2 seconds

}

void loop() {

  secondsOfHour++;

  // after 24 hours reset this integer
  if (secondsOfHour > 86400) {
    secondsOfHour = 1;
    ESP.restart();
  }

  float amps = getAmps();
  float watts = amps * 120;

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.print("W:");
  display.println(watts, 0);
  display.print("A:");
```

```
display.println(amps, 0);
display.setTextSize(1);
display.println("internet:");
display.println(wl_status_to_string(WiFi.status()));
display.display();

//////////////////////
// SERVER RELAY

if (!debug && (loopCount >= samplesPerReading) || serverFailed) {

  loopCount = 0;
  totalledAveragePressure = 0;

  setWifiSleepMode(false);

  delay(2000);

  // do a heartbeat check to see if we are online...
  http.begin(heartbeatUrl);
  httpCode = http.GET();
  if (!httpCode > 0) {
    // wifi may not be alive yet so wait 3 seconds
    delay(3000);
  }

  String recordedWatts = String(watts, 1);

  recordedWatts.trim();

  loopCount = 0;

  http.begin(currentUrl + recordedWatts);
  httpCode = http.GET();
  if (httpCode > 0) {
    payload = http.getString();
    Serial.println("HTTP Response: " + payload);
    recordPingSucces();
  } else {
    recordPingFailure();
```

```
    }
    http.end();

    setWifiSleepMode(true);
  }

  loopCount++;

}

bool connectToWiFi() {

  String activeSsid = "";
  String activePassword = "";

  if (activeConnection == 1) {
    activeSsid = ssid1;
    activePassword = password1;
  } else if (activeConnection == 2) {
    activeSsid = ssid2;
    activePassword = password2;
  }

  Serial.print("Connecting to WiFi: ");
  Serial.println(activeSsid);

  WiFi.begin(activeSsid, activePassword);

  while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
    delay(500);
    Serial.print(".");
    wifiConnectionAttempts++;
  }

  wifiConnectionAttempts = 0;

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
```

```
    wifiConnected = true;

    EEPROM.begin(EEPROM_SIZE);
    EEPROM.write(eepromActiveConnection, activeConnection);
    EEPROM.commit();
    EEPROM.end();

    Serial.print("set activeConnection to : ");
    Serial.println(activeConnection);

    return true;

  } else {
    Serial.print("Connection to ");
    Serial.print(activeSsid);
    Serial.println(" failed. Trying alternative");
    return false;
  }
}

/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

  wifiSleeping = sleepMode;

  if (sleepMode) {
    WiFi.disconnect();
    WiFi.setSleep(true);
    delay(1000);
    Serial.print("sleep wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
  } else {
    WiFi.setSleep(false);
    WiFi.reconnect();
    delay(1000);
    Serial.print("awaken wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
```

```
    // Check if the connection is still active. if not trigger wait for it to come back online
    if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
      Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
      WiFi.disconnect();
      wifiPaused = true;
      wifiConnected = false;
      connectToWiFi();
    }
  }
}

/**
 * record server ping success in long term memory
 */
void recordPingSucces() {
  totalServerPings++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = true;
  serverFailed = false;
}

/**
 * record server ping fails in long term memory
 */
void recordPingFailure() {
  totalServerPingFails++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = false;
  serverFailed = true;
}

/**
 * ESP32 wifi card statuses
 * @param status
```

```
 * @return string
 */
String wl_status_to_string(wl_status_t status) {

  String response = "";

  switch (status) {
  case WL_NO_SHIELD:
    response = "WL_NO_SHIELD";
    break;
  case WL_IDLE_STATUS:
    response = "WL_IDLE_STATUS";
    break;
  case WL_NO_SSID_AVAIL:
    response = "WL_NO_SSID_AVAIL";
    break;
  case WL_SCAN_COMPLETED:
    response = "WL_SCAN_COMPLETED";
    break;
  case WL_CONNECTED:
    response = "WL_CONNECTED";
    break;
  case WL_CONNECT_FAILED:
    response = "WL_CONNECT_FAILED";
    break;
  case WL_CONNECTION_LOST:
    response = "WL_CONNECTION_LOST";
    break;
  case WL_DISCONNECTED:
    response = "WL_DISCONNECTED";
    break;
  }

  return response;
}



/**
 * Get the current in amps coming from the hall sensor
 * @return float
```

```
 */
float getAmps() {

  float sensor1Reading;
  float sensor2Reading;
  float amps1 = 0;
  float amps2 = 0;

  float combinedReading;
  float sum = 0;
  long time_check = millis();
  int counter = 0;

  while (millis() - time_check < 5000) {

    sensor1Reading = ads.readADC_Differential_0_1();
    sensor2Reading = ads.readADC_Differential_2_3();

    // ac current flows in 2 directions so grab the flow in each direction
    if(sensor1Reading < 0) {
      sensor1Reading = sensor1Reading * -1;
    }

    if(sensor2Reading < 0) {
      sensor2Reading = sensor2Reading * -1;
    }

    if(sensor1Reading < 2) {
      sensor1Reading = 0;
    }

    if(sensor2Reading < 2) {
      sensor2Reading = 0;
    }

    amps1 += sensor1Reading;
    amps2 += sensor2Reading;

    combinedReading = amps1 + amps2;
```

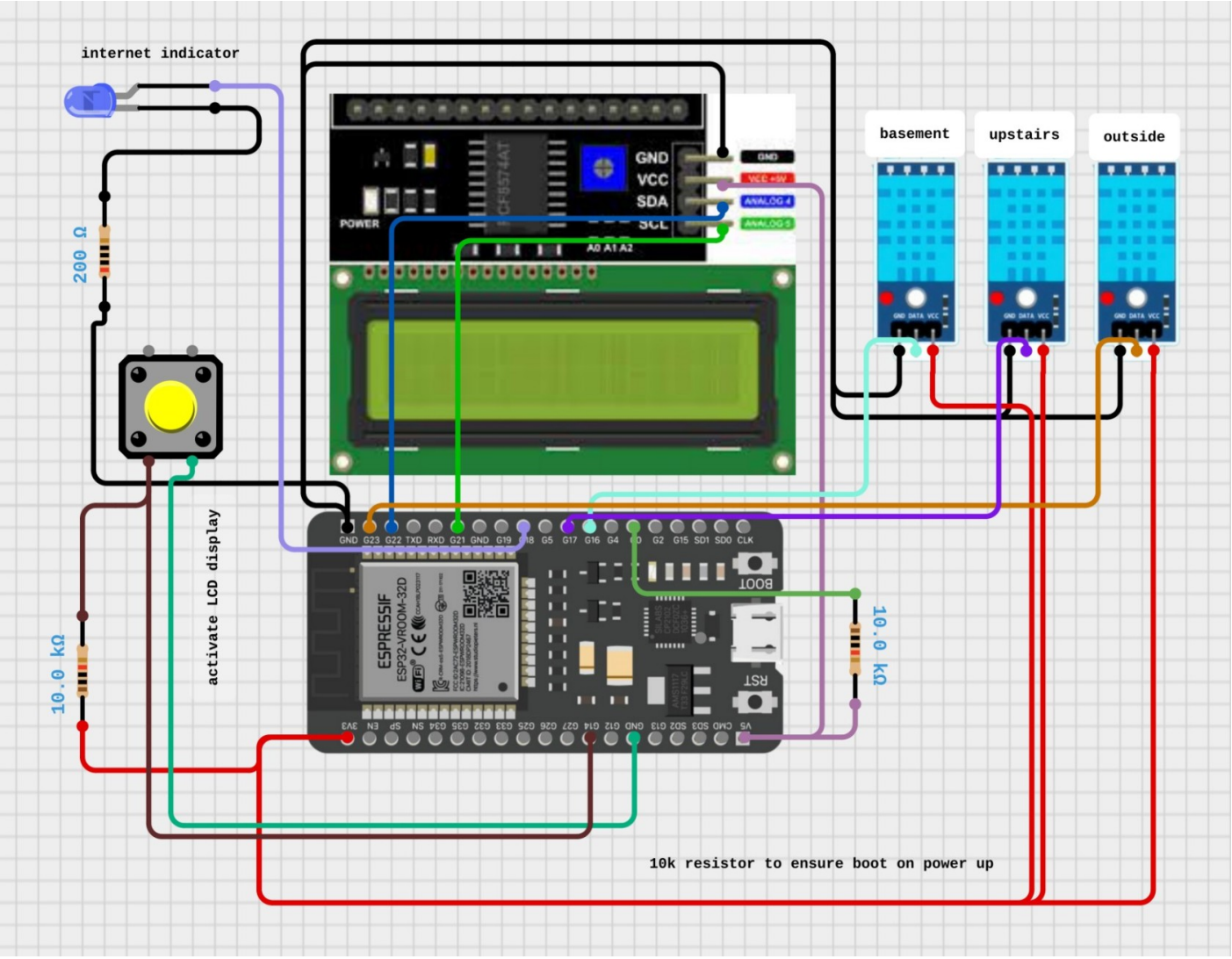
```
 counter = counter + 1;
}

float reading = (combinedReading / counter);

float averageAmps1 = (amps1 / counter);
float averageAmps2 = (amps2 / counter);

// some adjustments for variations in readings
float divider = .155;
averageAmps1 = averageAmps1 * divider + (averageAmps1 * 0.015);
averageAmps2 = averageAmps2 * divider + (averageAmps2 * 0.015);

return averageAmps1 + averageAmps2;

}
```

Temperature Monitoring (indoor and outdoor)

This project aims to relay indoor and outdoor temperature (and optionally humidity) to a server log. This project is fairly simple because we can use off-the-shelf DHT11 and DHT22 sensors (DHT22 is for below 0c).

Note to self: avoid non-oem DHT22 modules. You only save a couple of dollars and they are in my experience completely unreliable. The cheap clones either don't work or die after a few hours. I pay about $6 per unit for OEM (original) AM2302 chips, which have been running without issue for months.

The diagram below includes a 10k resistor between VIN (5v) and GPIO 0 because the esp32 unit I was using would not boot on power-up without having to hit the reset button. This is a common flaw with the esp32 units but so far for me only happened with one of the dozens I've used so far.

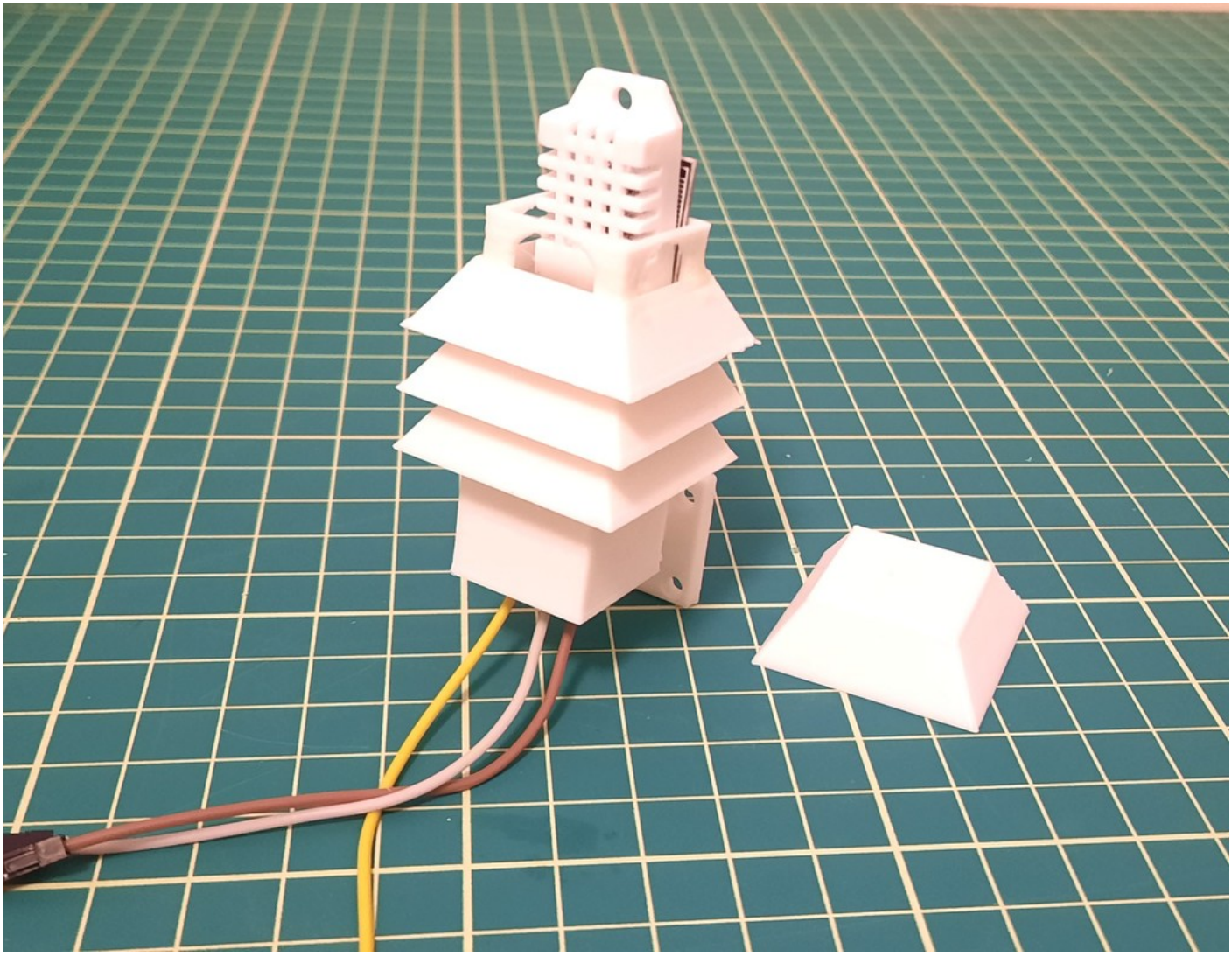


For the indoor sensor I used this case:

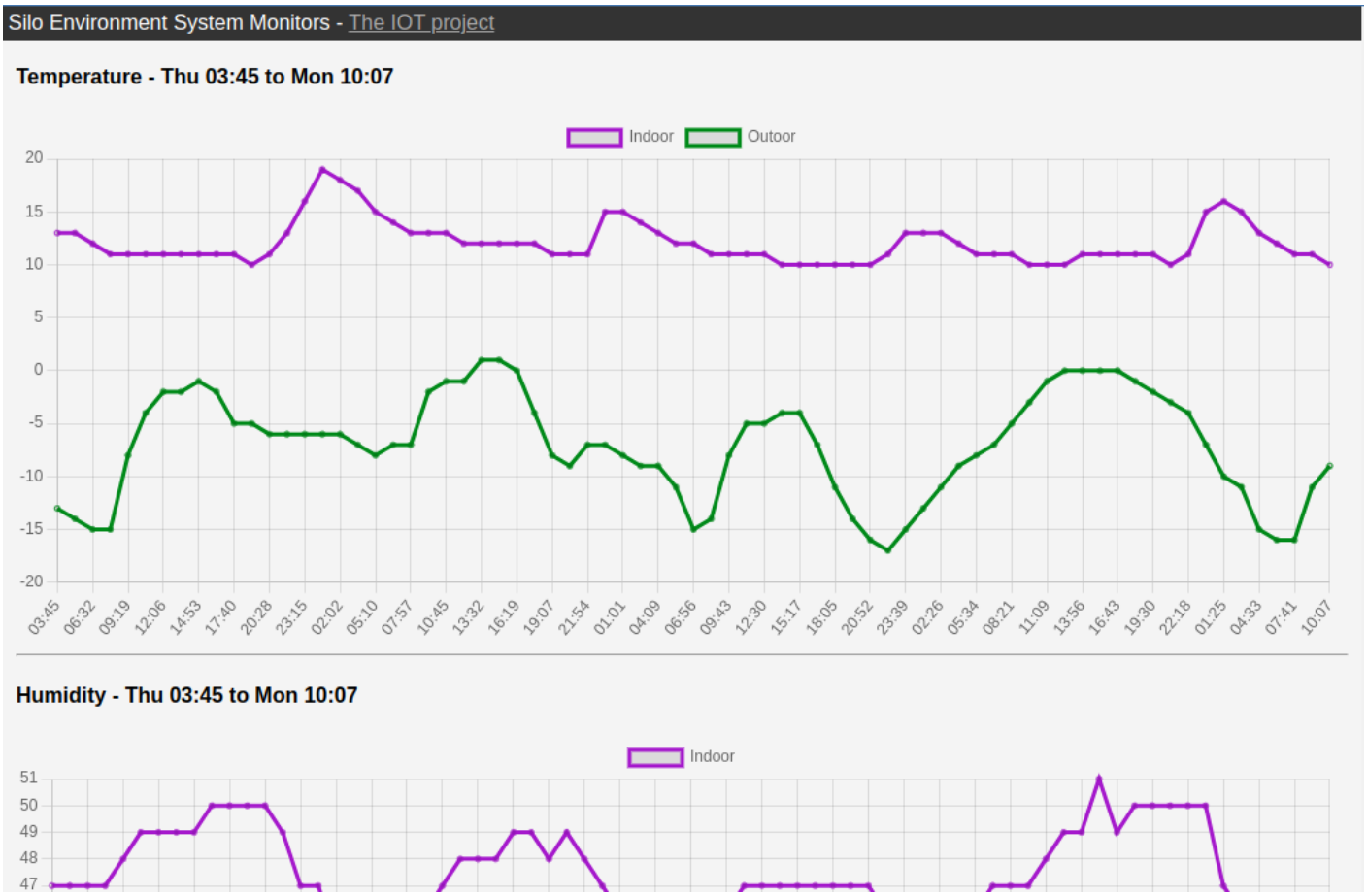
https://www.thingiverse.com/thing:2497711

And for the outdoor sensor I used this case:

https://www.thingiverse.com/thing:6400888



The end result in my case is a chart showing the temperature over a couple of days. If I see the temperature drop significantly (like pipes freezing zone) I at least can get out there to remedy the situation before it becomes an issue.



And the code. It ain't pretty but it works. For the server-side code I am just using free web hosting since the load is almost nothing. For the ESP32 module, the code below is about as minimal as I dare go. I do record the server success/fail rates but if everything seems stable after a few weeks I will remove that too.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#include <DHT_U.h>
#include <EEPROM.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"


////////////////////////////////
// EDIT THIS SECTION
// last updated June 23, 2024
```

```
int activeConnection = 1;

// lab
const String ssid1 = "asiuy2G";
const String password1 = "654765";

// other place
const String ssid2 = "cheytr";
const String password2 = "87654";

const String temperatureUrl = "https://homelab.htd.org/set_temp_humidity.php?data=";

// end edit section
//////////////////

#define EEPROM_SIZE 48
int eepromPingsAddress = 0;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnectionAddress = 1;

String prevTemperatureText = "";
String prevHumidifyText = "";
String prevConnectivityText1 = "";
String prevConnectivityText2 = "";

const int basementTemperaturePin = 16;
const int upstairsTemperaturePin = 17;
const int outdoorTemperaturePin = 23;



const int buttonPin = 14;
int buttonState = 0;
int displayMode = 1;  // 1: temp, humidiy  2: internet

const int pulseRate = 2;  // 2 seconds

const int temperatureServerSendInterval = 1;                                    // 10 minutes between sending a
```

```
temperature update to the server
const int temperatureSamplesPerReading = (60 * temperatureServerSendInterval) / pulseRate;  // every x
minutes send a sample to the server

int temperatureLoopCount = 0;

int lcdBacklightOnCounter = 0;



long secondsOfHour = 0;

const int secondsPerHour = 3600;  // set to 60 for debugging

const int hoursPerDay = 24;









float currentBasementTemperature = 0;
float currentUpstairsTemperature = 0;
float currentOutdoorTemperature = 0;

float currentBasementHumidity = 0;
float currentUpstairsHumidity = 0;
float currentOutdoorHumidity = 0;



bool connectingToWifi = false;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool debug = false;  // when true server does not update
bool serverFailed = false;
int wifiConnectionAttempts = 0;

HTTPClient http;
```

```
DHT_Unified dhtBasement(basementTemperaturePin, DHT11);
DHT_Unified dhtUpstairs(upstairsTemperaturePin, DHT11);
DHT_Unified dhtOutdoor(outdoorTemperaturePin, DHT11);  //DHT22 is actually in ouse outside

LiquidCrystal_I2C lcd_i2c(0x27, 16, 2);

sensors_event_t basementSensorEvent;
sensor_t basementSensor;
sensors_event_t upstairsSensorEvent;
sensor_t upstairsSensor;
sensors_event_t outdoorSensorEvent;
sensor_t outdoorSensor;

#define LED_PIN 18  // ESP32 pin GPIO18 connected to LED



void setup() {

  // We sometimes run into brownouts due to main hydro line voltage drops. For this situation set BOD to 3 volts
  // WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);  //disable brownout detector
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0x03);  // 3 volts
  //WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0x04); // set brownout detector to 3.2 volts
  // WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0x07); // Set 3.3V as BOD level

  Serial.begin(115200);  // this needs to match the value in the serial monitor

  //Init EEPROM
  EEPROM.begin(EEPROM_SIZE);

  pinMode(buttonPin, INPUT); // button for displaying info on LCD
  pinMode(LED_PIN, OUTPUT); // led to indicate internet connectivity

  delay(2000);

  dhtBasement.begin();
  dhtUpstairs.begin();
  dhtOutdoor.begin();

  dhtBasement.temperature().getSensor(&basementSensor);
```

```
dhtBasement.humidity().getSensor(&basementSensor);

dhtUpstairs.temperature().getSensor(&upstairsSensor);
dhtUpstairs.humidity().getSensor(&upstairsSensor);

dhtOutdoor.temperature().getSensor(&outdoorSensor);
dhtOutdoor.humidity().getSensor(&outdoorSensor);

getSensorReadings();

EEPROM.begin(EEPROM_SIZE);

activeConnection = EEPROM.read(eepromActiveConnectionAddress);

Serial.print("eepromActiveConnection :");
Serial.println(activeConnection);

if (isnan(activeConnection)) {
  activeConnection = 1;
}

if (activeConnection == 0) {
  activeConnection = 1;
}

if (activeConnection > 2) {
  activeConnection = 2;
}
Serial.print("activeConnection: ");
Serial.println(activeConnection);

if (!connectToWiFi()) {
  delay(2000);
  WiFi.disconnect();
  delay(1000);
  if (activeConnection == 1) {
    activeConnection = 2;
  } else {
    activeConnection = 1;
  }
```

```
    connectToWiFi();
  }

  // in case we did a reboot allow remote server a moment
  delay(5000);

  // if wifi connect failed again just reboot
  if (WiFi.status() != WL_CONNECTED) {
    ESP.restart();
  }

  lcd_i2c.init();
  lcd_i2c.backlight();
}


void loop() {

  secondsOfHour += pulseRate;

  // after 24 hours reset this integer
  if (secondsOfHour > 86400) {
    secondsOfHour = 1;
    // Serial.println("restarting :");
    ESP.restart();
  }

  if (lcdBacklightOnCounter == 0) {
    lcd_i2c.noBacklight();
  } else {
    // if lcb backlight counter is over zero it is active so increment
    lcdBacklightOnCounter++;
    // turn off backlight after 300 5 mintes of inactivity
    if (lcdBacklightOnCounter % 300 == 0) {
      lcd_i2c.noBacklight();
      lcdBacklightOnCounter = 0;
    }
  }

  buttonState = digitalRead(buttonPin);
```

```
  if (buttonState == LOW) {
    lcd_i2c.clear();
    lcd_i2c.backlight();

    if (lcdBacklightOnCounter == 0) {
      displayMode = 1;
    } else {
      displayMode++;
      if (displayMode > 2) {
        displayMode = 1;
      }
    }
    lcdBacklightOnCounter = 1;
  }

  sensors_event_t basementSensorEvent;

  ////////////////////////////
  // TEMPERATURE AND HUMIDITY

  // no need to read the temperature every second
  if (temperatureLoopCount % 10 == 0) {
    getSensorReadings();
  }


  String temperatureText = "T U:" + String(currentUpstairsTemperature, 0) + " B:" +
String(currentBasementTemperature, 0) + " E:" + String(currentOutdoorTemperature, 0) + "c";
  String humidifyText = "H U:" + String(currentUpstairsHumidity, 0) + " B:" + String(currentBasementHumidity,
0) + " E:" + String(currentOutdoorHumidity, 0) + "";

  if (wifiPaused) {
    if (wifiPausedTick > 60 && wifiConnectionAttempts == 0) {  // wait 1 minute before attempting to reconnect
      wifiPaused = false;
      wifiPausedTick = 0;
      connectToWiFi();
    } else {
      wifiPausedTick++;
    }
```

```
  }

  //////////////////////
  // SERVER RELAY


  if (!debug && (temperatureLoopCount >= temperatureSamplesPerReading) || serverFailed) {
    String recordedUpstairsTemperature = String(currentUpstairsTemperature, 1);
    String recordedUpstairsHumidity = String(currentUpstairsHumidity, 1);
    String recordedBasementTemperature = String(currentBasementTemperature, 1);
    String recordedBasementHumidity = String(currentBasementHumidity, 1);
    String recordedOutdoorTemperature = String(currentOutdoorTemperature, 1);
    String recordedOutdoorHumidity = String(currentOutdoorHumidity, 1);

    recordedBasementTemperature.trim();
    recordedBasementHumidity.trim();
    recordedOutdoorTemperature.trim();
    recordedOutdoorHumidity.trim();

    temperatureLoopCount = 0;
    setWifiSleepMode(false);
    http.begin(temperatureUrl + recordedBasementTemperature + "," + recordedBasementHumidity + "," +
recordedUpstairsTemperature + "," + recordedUpstairsHumidity + "," + recordedOutdoorTemperature + "," +
recordedOutdoorHumidity);
    int httpCode = http.GET();
    if (httpCode > 0) {
      String payload = http.getString();
      Serial.println("HTTP Response: " + payload);
      recordPingSucces();
    } else {
      recordPingFailure();
    }
    http.end();

    setWifiSleepMode(true);
  }

  temperatureLoopCount++;
```

```
//////////////////////////
// LCD DISPLAY

switch (displayMode) {
  case 1:
    temperatureText.trim();
    humidifyText.trim();

    if (prevTemperatureText != temperatureText || prevHumidifyText != humidifyText) {
      lcd_i2c.clear();
    }

    prevTemperatureText = temperatureText;
    prevHumidifyText = humidifyText;

    lcd_i2c.setCursor(0, 0);
    lcd_i2c.print(temperatureText);
    lcd_i2c.setCursor(0, 1);
    lcd_i2c.print(humidifyText);
    break;

  case 2:
    String connectivityText1 = "";
    if (wifiConnected) {
      connectivityText1 += "WF ON";
    }
    if (wifiConnected && !serverFailed) {
      connectivityText1 += " SERVER ON";
    }

    String serverPings = String(totalServerPings, 0);
    serverPings.trim();

    String serverPingFails = String(totalServerPingFails, 0);
    serverPingFails.trim();

    String connectivityText2 = serverPings + "/" + serverPingFails + " PINGS";

    connectivityText1.trim();
    connectivityText2.trim();
```

```
    if (prevConnectivityText1 != connectivityText1 || prevConnectivityText2 != connectivityText2) {
      lcd_i2c.clear();
    }

    prevConnectivityText1 = connectivityText1;
    prevConnectivityText2 = connectivityText2;

    lcd_i2c.setCursor(0, 0);
    lcd_i2c.print(connectivityText1);
    lcd_i2c.setCursor(0, 1);
    lcd_i2c.print(connectivityText2);
    break;
  }

  //delay(pulseRate * 1000);
  pulseLed(!wifiPaused);

  // analogWrite(LED_PIN, 255);
}




/**
* pulse the led in and out over a span of 2 seconds
*
*/
void pulseLed(bool active) {

  // analogWrite(LED_PIN, 255);

  int dutyCycle = 166;
  int delayMs = pulseRate * 1000 / dutyCycle;

  // Fade in the LED
  for (int brightness = 0; brightness <= dutyCycle; brightness++) {
    if (active) {
      analogWrite(LED_PIN, brightness);
    }
```

```
    delay(delayMs);  // Adjust this value to change the fade duration
  }

  // Fade out the LED
  for (int brightness = dutyCycle; brightness >= 0; brightness--) {
    if (active) {
      analogWrite(LED_PIN, brightness);
    }
    delay(delayMs);  // Adjust this value to change the fade duration
  }

  // delay(1000);
}

bool connectToWiFi() {

  if (connectingToWifi || wifiConnected) {
    return true;
  }

  connectingToWifi = true;

  String activeSsid = "";
  String activePassword = "";

  if (activeConnection == 1) {
    activeSsid = ssid1;
    activePassword = password1;
  } else if (activeConnection == 2) {
    activeSsid = ssid2;
    activePassword = password2;
  }

  Serial.print("Connecting to WiFi: ");
  Serial.println(activeSsid);

  WiFi.begin(activeSsid, activePassword);

  while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
    delay(500);
```

```
    Serial.print(".");
    wifiConnectionAttempts++;
  }

  wifiConnectionAttempts = 0;

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());

    EEPROM.write(eepromActiveConnectionAddress, activeConnection);
    EEPROM.commit();

    Serial.print("set activeConnection to : ");
    Serial.println(activeConnection);

    // 5 flashes means intenet is connected
    for (int i = 0; i < 5; i++) {
      digitalWrite(LED_PIN, HIGH);
      delay(10);
      digitalWrite(LED_PIN, LOW);
      delay(200);
    }

    wifiConnected = true;
    connectingToWifi = false;
    return true;

  } else {
    Serial.print("Connection to ");
    Serial.print(activeSsid);
    Serial.println(" failed. Trying alternative");

    wifiConnected = false;
    connectingToWifi = false;

    return false;
  }
}
```

```
/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

  wifiSleeping = sleepMode;

  if (sleepMode) {
    WiFi.disconnect();
    WiFi.setSleep(true);
    wifiConnected = false;
    delay(1000);
    Serial.print("sleep wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
  } else {
    WiFi.setSleep(false);
    WiFi.reconnect();
    delay(2000);
    Serial.print("awaken wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
    // Check if the connection is still active. if not trigger wait for it to come back online
    if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
      Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
      WiFi.disconnect();
      wifiPaused = true;
      wifiConnected = false;
      connectToWiFi();
    }
  }
}

/**
 * record server ping success in long term memory
 */
void recordPingSucces() {
  totalServerPings++;
  EEPROM.begin(EEPROM_SIZE);
```

```
  EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = true;
  serverFailed = false;
}

/**
 * record server ping fails in long term memory
 */
void recordPingFailure() {
  totalServerPingFails++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = false;
  serverFailed = true;
}

/**
 * ESP32 wifi card statuses
 * @param status
 * @return string
 */
String wl_status_to_string(wl_status_t status) {

  String response = "";

  switch (status) {
    case WL_NO_SHIELD:
      response = "WL_NO_SHIELD";
      break;
    case WL_IDLE_STATUS:
      response = "WL_IDLE_STATUS";
      break;
    case WL_NO_SSID_AVAIL:
      response = "WL_NO_SSID_AVAIL";
      break;
    case WL_SCAN_COMPLETED:
```

```
      response = "WL_SCAN_COMPLETED";
      break;
    case WL_CONNECTED:
      response = "WL_CONNECTED";
      break;
    case WL_CONNECT_FAILED:
      response = "WL_CONNECT_FAILED";
      break;
    case WL_CONNECTION_LOST:
      response = "WL_CONNECTION_LOST";
      break;
    case WL_DISCONNECTED:
      response = "WL_DISCONNECTED";
      break;
  }

  return response;
}



void getSensorReadings() {


  float newTempReading;
  float newHumidityReading;


  // BASEMENT

  dhtBasement.temperature().getEvent(&basementSensorEvent);
  if (isnan(basementSensorEvent.temperature)) {
    Serial.println(F("Error reading basement temperature!"));
  } else {
    newTempReading = basementSensorEvent.temperature;
    if (newTempReading > -10 && newTempReading < 50) {
      currentBasementTemperature = basementSensorEvent.temperature;
    }
    Serial.print("basement temp: ");
    Serial.println(currentBasementTemperature);
```

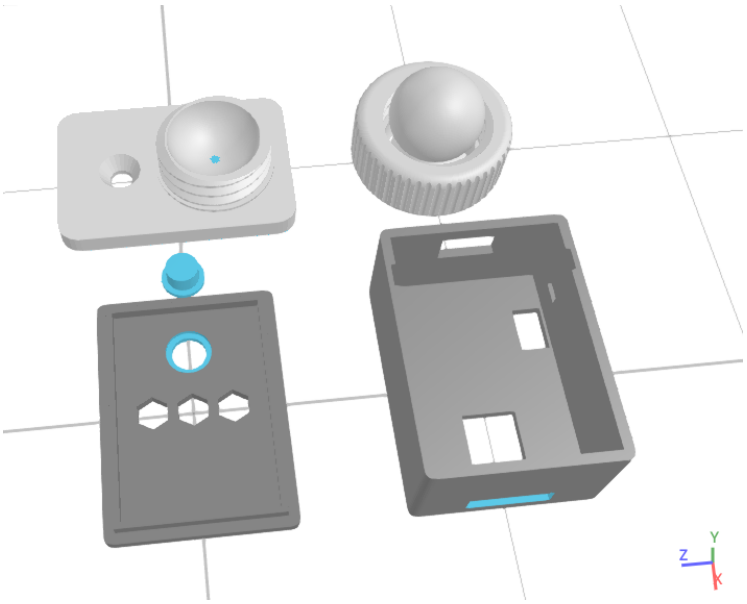
```
}


dhtBasement.humidity().getEvent(&basementSensorEvent);
if (isnan(basementSensorEvent.relative_humidity)) {
  Serial.println(F("Error reading basement humidity!"));
} else {

  newHumidityReading = basementSensorEvent.relative_humidity;
  if (newHumidityReading > -10 && newHumidityReading < 100) {
    currentBasementHumidity = basementSensorEvent.relative_humidity;
  }

  Serial.print("basement humidity: ");
  Serial.println(currentBasementHumidity);
}


// UPSTAIRS
dhtUpstairs.temperature().getEvent(&upstairsSensorEvent);
if (isnan(upstairsSensorEvent.temperature)) {
  Serial.println(F("Error reading upstairs temperature!"));
} else {
  newTempReading = upstairsSensorEvent.temperature;
  if (newTempReading > -10 && newTempReading < 50) {
    currentUpstairsTemperature = upstairsSensorEvent.temperature;
  }
  Serial.print("upstairs temp: ");
  Serial.println(currentUpstairsTemperature);
}


dhtUpstairs.humidity().getEvent(&upstairsSensorEvent);
if (isnan(upstairsSensorEvent.relative_humidity)) {
  Serial.println(F("Error reading upstairs humidity!"));
} else {

  newHumidityReading = upstairsSensorEvent.relative_humidity;
  if (newHumidityReading > -10 && newHumidityReading < 100) {
    currentUpstairsHumidity = upstairsSensorEvent.relative_humidity;
```

```
    }

    Serial.print("upstairs humidity: ");
    Serial.println(currentUpstairsHumidity);
  }



  // OUTDOOR

  dhtOutdoor.temperature().getEvent(&outdoorSensorEvent);
  if (isnan(outdoorSensorEvent.temperature)) {
    Serial.println(F("Error reading outdoor temperature!"));
  } else {
    newTempReading = outdoorSensorEvent.temperature;
    if (newTempReading > -10 && newTempReading < 50) {
      currentOutdoorTemperature = outdoorSensorEvent.temperature;
    }
    Serial.print("outdoor temp: ");
    Serial.println(currentOutdoorTemperature);
  }



  dhtOutdoor.humidity().getEvent(&outdoorSensorEvent);
  if (isnan(outdoorSensorEvent.relative_humidity)) {
    Serial.println(F("Error reading outdoor humidity!"));
  } else {

    newHumidityReading = outdoorSensorEvent.relative_humidity;
    if (newHumidityReading > -10 && newHumidityReading < 100) {
      currentOutdoorHumidity = outdoorSensorEvent.relative_humidity;
    }

    Serial.print("outdoor humidity: ");
    Serial.println(currentOutdoorHumidity);
  }
}
```

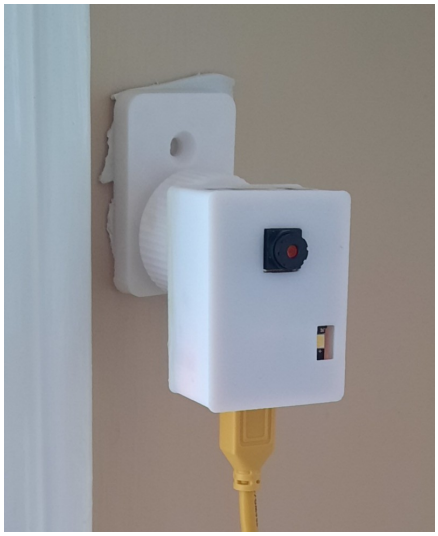
Simple Surveillance Camera

This probably sounds dumb but I wanted to make sure my birds were not starving to death anytime I needed to travel. The solution was 2 very inexpensive (less than $10) ESP32 CAM modules. These are pretty much plug and play regarding hardware. I just needed to print a case and write some code (code at bottom of this page), and the images are sent to my server in 5 minute intevals. For night time there is a flash built into the ESP32 CAM which actually works well, even in larger areas.

Here is the 3d printer file for the case: esp32_case.zip



The printed case assembled:



And here is what I see on my webpage:

So there you have it - the ultimate cheap-ass bird cam!

In case you want to do this yourself here's the code I used:

```
#include <WiFi.h>
#include <EEPROM.h>
#include <esp_sleep.h>
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"
#include "esp_camera.h"

int activeConnection = 1;



const String ssid1 = "xxxx";
const String password1 = "xxxx";
```

```
// silo
const String ssid2 = "xxxxx";
const String password2 = "xxxxxx";

String source = "camera1";

String serverName = "xx.xxx.ca";
String serverPath = "/xxxx/setphoto.php?source=" + source;

const int serverSendInterval = 5; // 5 minutes between sending a photo update to the server

const int serverPort = 80;

#define LED_BUILTIN 4
#define EEPROM_SIZE 4
int eepromPingsAddress = 0;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnection = 1;
long secondsOfHour = 0;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool serverFailed = false;
int wifiConnectionAttempts = 0;

WiFiClient client;

void setupLedFlash(int pin);

void setup() {

  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
```

```
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sccb_sda = SIOD_GPIO_NUM;
config.pin_sccb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_UXGA;
config.pixel_format = PIXFORMAT_JPEG; // for streaming
config.grab_mode = CAMERA_GRAB_LATEST;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 12;
config.fb_count = 3;

pinMode(LED_BUILTIN, OUTPUT);

// camera init
esp_err_t err = esp_camera_init( & config);
if (err != ESP_OK) {
  Serial.printf("Camera init failed with error 0x%x", err);
  delay(1000);
  ESP.restart();
}

//Init EEPROM
EEPROM.begin(EEPROM_SIZE);

activeConnection = EEPROM.read(eepromActiveConnection);
```

```
Serial.print("eepromActiveConnection :");
Serial.println(activeConnection);

if (isnan(activeConnection)) {
  activeConnection = 1;
}

if (activeConnection == 0) {
  activeConnection = 1;
}

if (activeConnection > 2) {
  activeConnection = 2;
}
Serial.print("activeConnection: ");
Serial.println(activeConnection);

float pingData = EEPROM.readFloat(eepromPingsAddress);
if (isnan(pingData)) {
  pingData = 0;
}
totalServerPings = pingData;
EEPROM.end();
EEPROM.begin(EEPROM_SIZE);
float pingFailData = EEPROM.readFloat(eepromFailedPingsAddress);
if (isnan(pingFailData)) {
  pingFailData = 0;
}
totalServerPingFails = pingFailData;
EEPROM.end();

if (!connectToWiFi()) {
  delay(2000);
  WiFi.disconnect();
  delay(1000);
  if (activeConnection == 1) {
    activeConnection = 2;
  } else {
    activeConnection = 1;
  }
```

```
    connectToWiFi();
  }

}

void loop() {

  delay(serverSendInterval * 60 * 1000);

  secondsOfHour += serverSendInterval * 60;

  // after 24 hours reset this integer
  if (secondsOfHour > 86400) {
    secondsOfHour = 1;
    ESP.restart();
  }

  sendPhoto();

}

void sendPhoto() {

  String getAll;
  String getBody;

  camera_fb_t * fb = NULL;
  digitalWrite(LED_BUILTIN, HIGH);
  delay(500); // allow time for led to fully illuminate
  fb = esp_camera_fb_get();
  delay(50);
  digitalWrite(LED_BUILTIN, LOW);

  if (!fb) {
    Serial.println("Camera capture failed");
    delay(1000);
    ESP.restart();
  }

  setWifiSleepMode(false);
```

```
  delay(5000);

  Serial.println("Connecting to server: " + serverName);

  if (client.connect(serverName.c_str(), serverPort)) {
    Serial.println("Connection successful!");
    String head = "--ImpresstoDocs\r\nContent-Disposition: form-data; name=\"imageFile\"; filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";
    String tail = "\r\n--ImpresstoDocs--\r\n";

    uint32_t imageLen = fb -> len;
    uint32_t extraLen = head.length() + tail.length();
    uint32_t totalLen = imageLen + extraLen;

    client.println("POST " + serverPath + " HTTP/1.1");
    client.println("Host: " + serverName);
    client.println("Content-Length: " + String(totalLen));
    client.println("Content-Type: multipart/form-data; boundary=ImpresstoDocs");
    client.println();
    client.print(head);

    uint8_t * fbBuf = fb -> buf;
    size_t fbLen = fb -> len;
    for (size_t n = 0; n < fbLen; n = n + 1024) {
      if (n + 1024 < fbLen) {
        client.write(fbBuf, 1024);
        fbBuf += 1024;
      } else if (fbLen % 1024 > 0) {
        size_t remainder = fbLen % 1024;
        client.write(fbBuf, remainder);
      }
    }
    client.print(tail);

    esp_camera_fb_return(fb);

    int timoutTimer = 10000;
    long startTimer = millis();
    boolean state = false;
```

```
    while ((startTimer + timoutTimer) > millis()) {
      Serial.print(".");
      delay(100);
      while (client.available()) {
        char c = client.read();
        if (c == '\n') {
          if (getAll.length() == 0) {
            state = true;
          }
          getAll = "";
        } else if (c != '\r') {
          getAll += String(c);
        }
        if (state == true) {
          getBody += String(c);
        }
        startTimer = millis();
      }
      if (getBody.length() > 0) {
        break;
      }
    }
    Serial.println();
    client.stop();
    Serial.println(getBody);

    setWifiSleepMode(true);
  } else {
    getBody = "Connection to " + serverName + " failed.";
    Serial.println(getBody);
    client.stop();

    setWifiSleepMode(true);

  }
}

bool connectToWiFi() {
```

```
String activeSsid = "";
String activePassword = "";

if (activeConnection == 1) {
  activeSsid = ssid1;
  activePassword = password1;
} else if (activeConnection == 2) {
  activeSsid = ssid2;
  activePassword = password2;
}

Serial.print("Connecting to WiFi: ");
Serial.println(activeSsid);

WiFi.begin(activeSsid, activePassword);

while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
  delay(500);
  Serial.print(".");
  wifiConnectionAttempts++;
}

wifiConnectionAttempts = 0;

if (WiFi.status() == WL_CONNECTED) {
  Serial.println("\nConnected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  wifiConnected = true;

  EEPROM.begin(EEPROM_SIZE);
  EEPROM.write(eepromActiveConnection, activeConnection);
  EEPROM.commit();
  EEPROM.end();

  Serial.print("set activeConnection to : ");
  Serial.println(activeConnection);

  return true;
```

```
  } else {
    Serial.print("Connection to ");
    Serial.print(activeSsid);
    Serial.println(" failed. Trying alternative");
    return false;
  }
}

/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

  wifiSleeping = sleepMode;

  if (sleepMode) {
    WiFi.disconnect();
    WiFi.setSleep(true);
    delay(1000);
    Serial.print("sleep wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
  } else {
    WiFi.setSleep(false);
    WiFi.reconnect();
    delay(2000);
    Serial.print("awaken wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
    // Check if the connection is still active. if not trigger wait for it to come back online
    if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
      Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
      WiFi.disconnect();
      wifiPaused = true;
      wifiConnected = false;
      connectToWiFi();
    }
  }
}

/**
```

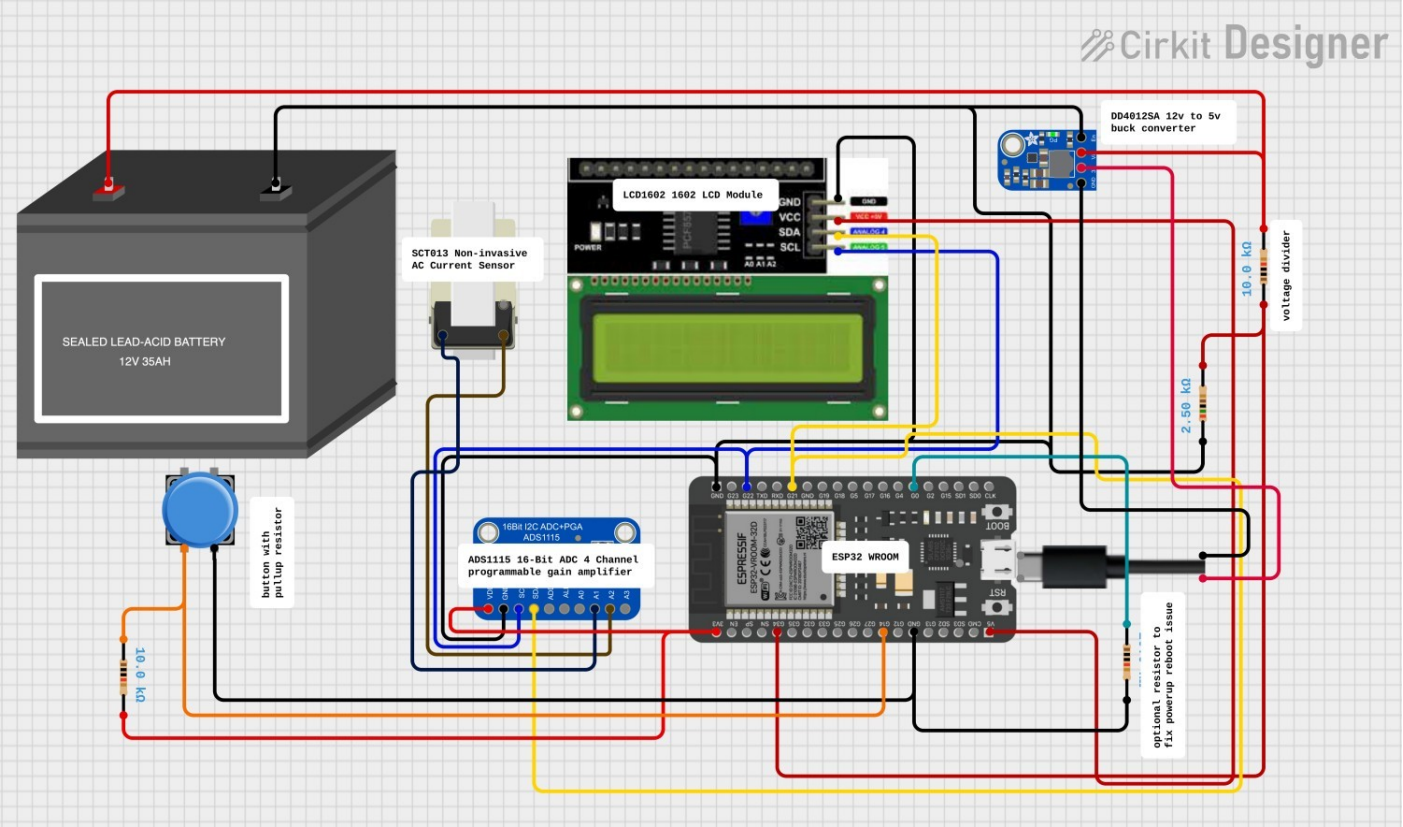
```
 * record server ping success in long term memory
 */
void recordPingSucces() {
  totalServerPings++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = true;
  serverFailed = false;
}

/**
 * record server ping fails in long term memory
 */
void recordPingFailure() {
  totalServerPingFails++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = false;
  serverFailed = true;
}

/**
 * ESP32 wifi card statuses
 * @param status
 * @return string
 */
String wl_status_to_string(wl_status_t status) {

  String response = "";

  switch (status) {
  case WL_NO_SHIELD:
    response = "WL_NO_SHIELD";
    break;
  case WL_IDLE_STATUS:
    response = "WL_IDLE_STATUS";
```

```
    break;
  case WL_NO_SSID_AVAIL:
    response = "WL_NO_SSID_AVAIL";
    break;
  case WL_SCAN_COMPLETED:
    response = "WL_SCAN_COMPLETED";
    break;
  case WL_CONNECTED:
    response = "WL_CONNECTED";
    break;
  case WL_CONNECT_FAILED:
    response = "WL_CONNECT_FAILED";
    break;
  case WL_CONNECTION_LOST:
    response = "WL_CONNECTION_LOST";
    break;
  case WL_DISCONNECTED:
    response = "WL_DISCONNECTED";
    break;
  }

  return response;
}
```

Battery voltage and current draw monitor

For this project the goal was to keep track of backup batteries during float and active states - and also to save a few $$$. Total cost for this setup was about $20 CAD in parts. This monitor helps to ensure the trickle (solar) charger is still functioning, and when in a power outage, how much current is being drawn from the system and how much capacity is left. The system can be used to shut the inverter down remotely if it appears the batteries are at critical low voltage. An email alert is also sent when the voltage goes below 11 volts.

Wiring:

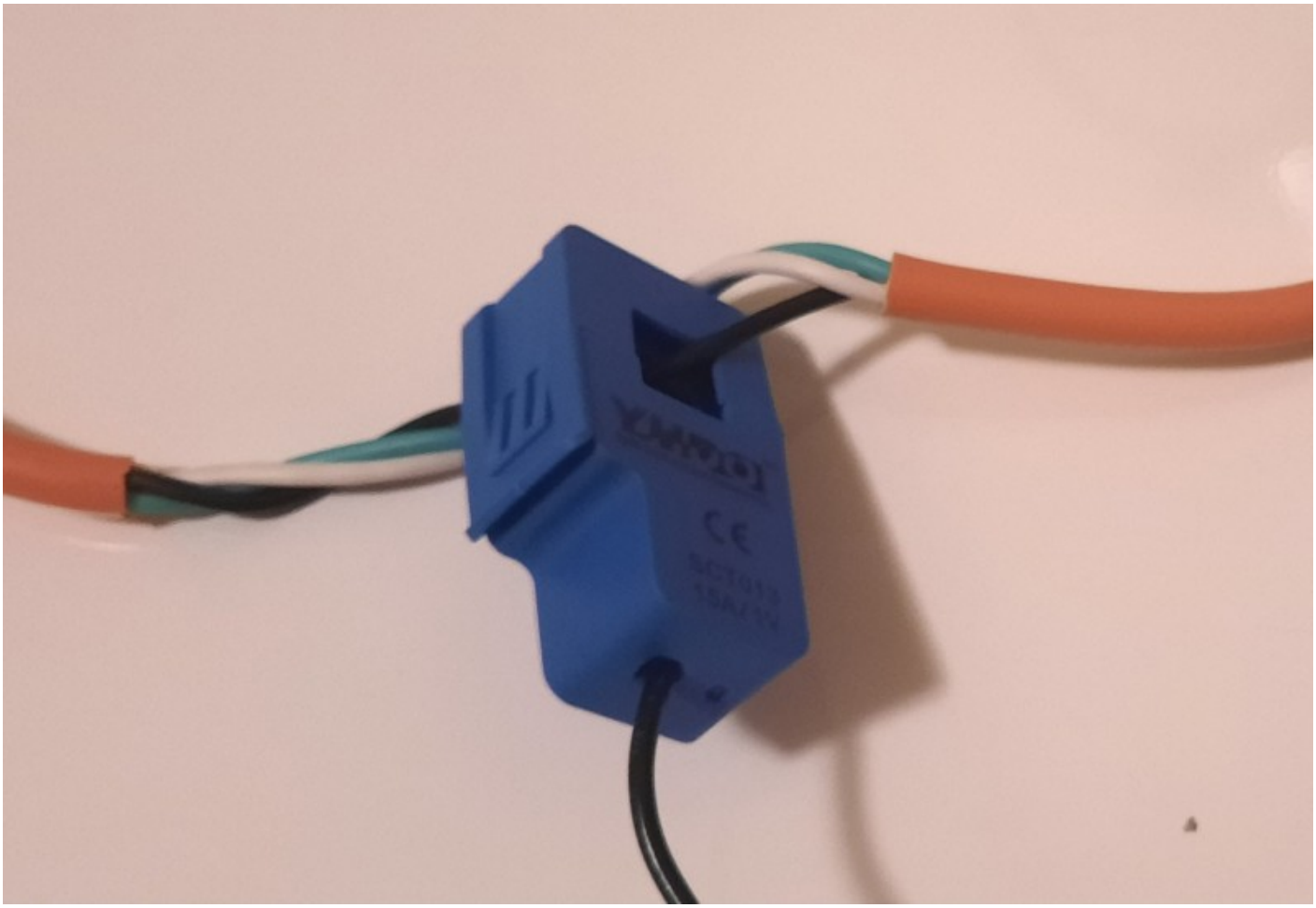


parts:

- Voltage divider - 10 kohm / 2.5 kohm resistors will get us down from 15 to a safe readable 3 volts - https://www.digikey.ca/en/resources/conversion-calculators/conversion-calculator-voltage-divider
- SCT-013 current sensor - https://www.aliexpress.com/item/32708887594.html
- ADS1115 high precision multiplexer - https://www.aliexpress.com/item/32817162654.html
- L2C display - https://www.aliexpress.com/item/32649621944.html
- ESP32 (microcontroller with wifi) - https://www.aliexpress.com/item/1005006246777139.html

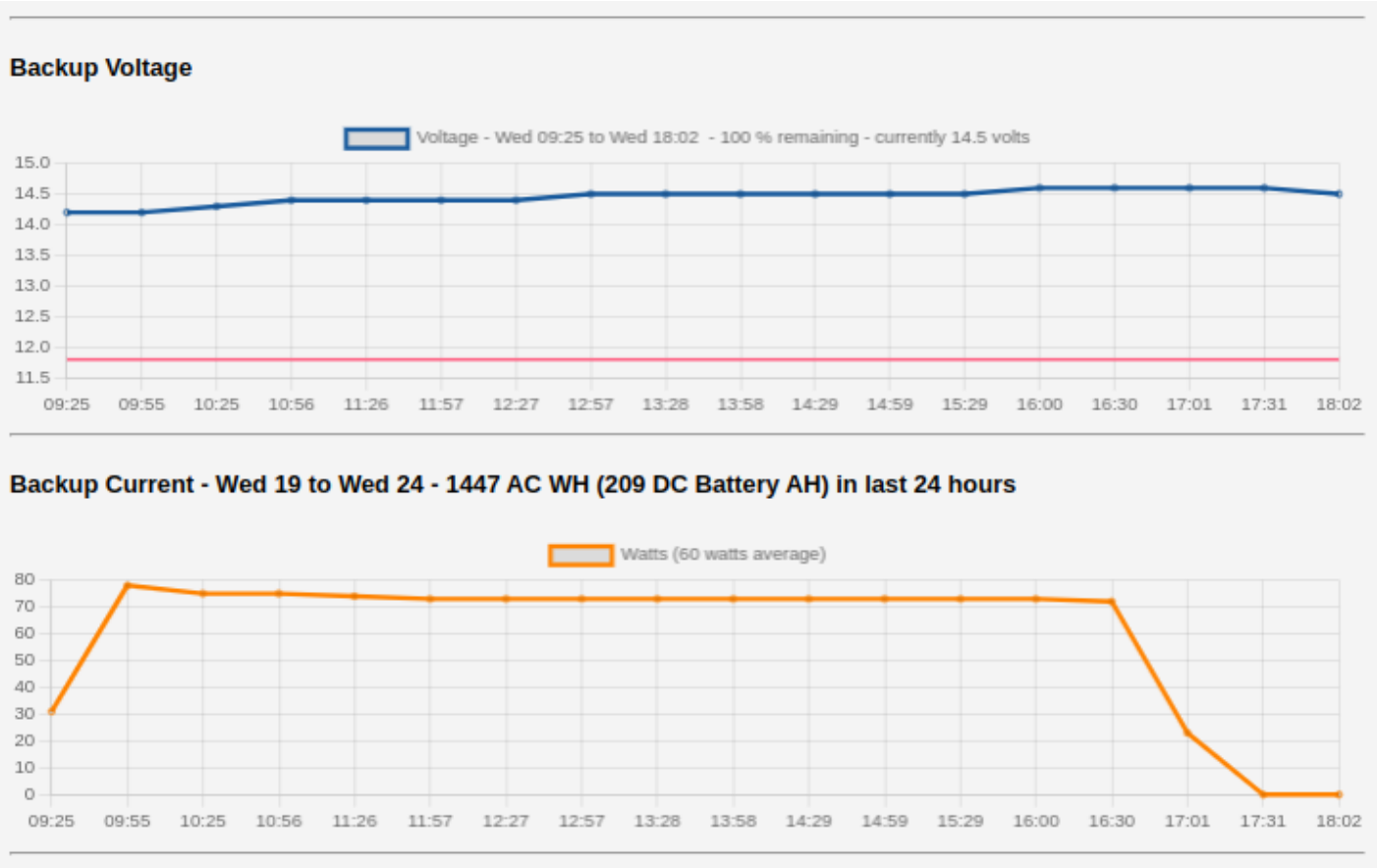
- buck converter - 12 volts > 5 volts - https://www.aliexpress.com/item/1005006186831162.html
- 3d printed case - https://www.thingiverse.com/thing:6684072

The current sensor is just a transformer coil wrapped around the AC cable which comes out of the inverter. It is much easier in my experience than dealing with shunts that produce unreliable readings. This simple coil provides readings with a 99% accuracy range over a wide range of currents ranging from 10 watts to 1800 watts (15 amps@120volts).



	I set a "fudge" in the code to get accurate readings by comparing the real-time readings compared to an outlet wall meter which I knew to be accurate. The readings simply needed to be boosted by a tiny fraction (X 0.0000444) to get near 100% accuracy.

Online view of realtime data:



code:

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <Adafruit_ADS1X15.h>
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>


// last updated July 1, 2024

//////////////////////////////
// EDIT THIS SECTION


int activeConnection = 1;

// cottage 1
const String ssid1 = "rewa";
const String password1 = "gfggf";
```

```
// cottage 2
const String ssid2 = "fdsa";
const String password2 = "rreter";

const String voltateUrl = "http://batterymonitor.uytr.ca/set_current_voltage.php?data=";

// end edit section
//////////////////////


#define EEPROM_SIZE 8
int eepromActiveConnectionAddress = 0;
int eepromPingsAddress = 2;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnection = 1;

String prevWattagePerHourText = "";
String prevWattageVoltageText = "";
String prevConnectivityText1 = "";
String prevConnectivityText2 = "";

const int potPin = 34;
const int buttonPin = 14;
int buttonState = 0;
int displayMode = 1;  // 1: watts, watts per hours, voltage   2: internet

int ACDCVoltage = 120;

const float criticalVoltage = 11.8;  // this is considered the critical point for deep cycle sla batteries

const float deadBatteryVoltage = 9;  //  set to 9

const float potOffset = 880;            // offset accounts for base reading on pot value -  should be zero in theory
but in reality it is not
const float voltageMultiplier = 0.025;  // adjust for non-linear increase in pot value vs actual voltage
const float voltageDividerRatio = 4;    // 10K / 2.5k voltage divider
const int pulseRate = 1000;             // loop runs once per second
```

```
const int serverSendInterval = 15;                    // 15 minutes between sending a voltage update to the server
const int samplesPerReading = 60 * serverSendInterval;  // every x minutes send a sample to the server

float averagePotValue = 0;
float potValue = 0;
float accumulatedPotValues = 0;
int voltageLoopCount = 0;

int lcdBacklightOnCounter = 0;

float accumulatedAmps = 0;
float averageCurrentValue = 0;
float averageWattage = 0;

float wattsPerHour = 0;
float ampHoursPerDay = 0;

long secondsOfHour = 0;

const int secondsPerHour = 3600;  // set to 60 for debugging
int houryAmpsArrayCurrentIndex = 0;
int dailyAmpsArrayCurrentIndex = 0;

float houryAmpsArray[secondsPerHour];

const int hoursPerDay = 24;
float dailyAmpsArray[hoursPerDay];

float voltage;

bool connectingToWifi = false;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool debug = false;  // when true server does not update
bool serverFailed = false;
int wifiConnectionAttempts = 0;
```

```
HTTPClient http;

Adafruit_ADS1115 ads;

LiquidCrystal_I2C lcd_i2c(0x27, 16, 2);

const float FACTOR = 20;                  //20A/1V from the CT
const float hallSensorFudge = 0.0000472;  // fudge for inaccuracy in the hall sensor



void setup() {


 // We sometimes run into brownouts due to main hydro line voltage drops. For this situation set BOD to 3 volts
 // WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);  //disable brownout detector
 WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0x03);  // 3 volts
 //WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0x04); // set brownout detector to 3.2 volts
 // WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0x07); // Set 3.3V as BOD level

 Serial.begin(115200);  // this needs to match the value in the serial monitor

 //Init EEPROM
 EEPROM.begin(EEPROM_SIZE);

 pinMode(buttonPin, INPUT);

 for (int i = 0; i < secondsPerHour; i++) {
  houryAmpsArray[i] = -1;
 }

 for (int i = 0; i < hoursPerDay; i++) {
  dailyAmpsArray[i] = -1;
 }

 delay(2000);

 EEPROM.begin(EEPROM_SIZE);

 activeConnection = EEPROM.read(eepromActiveConnectionAddress);
```

```
Serial.print("eepromActiveConnection :");
Serial.println(activeConnection);

if (isnan(activeConnection)) {
  activeConnection = 1;
}

if (activeConnection == 0) {
  activeConnection = 1;
}

if (activeConnection > 2) {
  activeConnection = 2;
}
Serial.print("activeConnection: ");
Serial.println(activeConnection);


if (!connectToWiFi()) {
  delay(2000);
  WiFi.disconnect();
  delay(1000);
  if (activeConnection == 1) {
    activeConnection = 2;
  } else {
    activeConnection = 1;
  }
  connectToWiFi();
}

// in case we did a reboot allow remote server a moment
delay(5000);

// if wifi connect failed again just reboot
if (WiFi.status() != WL_CONNECTED) {
  ESP.restart();
}

if (!ads.begin()) {
  Serial.println("Failed to initialize ADS.");
```

```
    while (1)
      ;
  }

  ads.setGain(GAIN_FOUR);


  lcd_i2c.init();
  lcd_i2c.backlight();
}

void loop() {

  secondsOfHour++;

  // after 24 hours reset this integer
  if (secondsOfHour > 86400) {
    secondsOfHour = 1;
    // Serial.println("restarting :");
    ESP.restart();
  }

  if (lcdBacklightOnCounter == 0) {
    //lcd_i2c.noBacklight();
    lcdBacklightOnCounter++;
    displayMode = 1;
  } else {
    // if lcb backlight counter is over zero it is active so increment
    lcdBacklightOnCounter++;

    Serial.print("lcdBacklightOnCounter: ");
    Serial.println(lcdBacklightOnCounter);

    // turn off backlight after 5 mintes of inactivity
    if (lcdBacklightOnCounter % 150 == 0) {

      Serial.println("backlight off");

      displayMode = 2;  // ensure when clicking for light we see temp again
      lcd_i2c.noBacklight();
```

```
    lcdBacklightOnCounter = 1;
  }
}


buttonState = digitalRead(buttonPin);

Serial.println("buttonState: ");
Serial.println(buttonState);

if (buttonState == LOW) {
  lcd_i2c.clear();
  lcd_i2c.backlight();
  lcdBacklightOnCounter = 1;
  displayMode++;
  if (displayMode > 2) {
    displayMode = 1;
  }
}



////////////////////////////
// CURRENT AND VOLTAGE

// CURRENT

float amps = getAmps();
accumulatedAmps += amps;

cycleHourlyAmpsArray(amps);

float ampsPerHour = getAverageAmps(houryAmpsArray, secondsPerHour);

// to get average wattage get the last 10 reading divided by 10

float wattsAverage = getLastXWattReadings(5) * ACDCVoltage;
float wattsPerHour = ampsPerHour * ACDCVoltage;

float ampHoursPerDay = getAverageAmps(dailyAmpsArray, hoursPerDay);
```

```
if (secondsOfHour % secondsPerHour == 0) {
  cycleDailyAmpsArray(ampsPerHour);
}

float dailyAmpHours = getDailyAmpHours();

float wattagePerDay = dailyAmpHours * ACDCVoltage;

// VOLTAGE
potValue = analogRead(potPin);
accumulatedPotValues += potValue;

Serial.print("pot value:");
Serial.println(potValue);

voltage = (potValue / (potOffset + (potValue * voltageMultiplier)) * voltageDividerRatio);

Serial.print("volts");
Serial.println(voltage);

String wattsAverageString = String(wattsAverage, 0);
wattsAverageString.trim();

String wattsPerHourString = String(wattsPerHour, 0);
wattsPerHourString.trim();

String wattsPerDayString = String(wattagePerDay, 0);
wattsPerDayString.trim();

String wattagePerHourText = wattsPerHourString + "WH " + wattsPerDayString + "DWH ";
String wattageVoltageText = wattsAverageString + "W BV:" + String(voltage, 1);

// keep the noise down
if (voltageLoopCount % 10 == 0) {
  averagePotValue = accumulatedPotValues / voltageLoopCount;
  voltage = (averagePotValue / (potOffset + (averagePotValue * voltageMultiplier)) * voltageDividerRatio);
  float currentVoltage = (potValue / (potOffset + (potValue * voltageMultiplier)) * voltageDividerRatio);
}
```

```
  averagePotValue = accumulatedPotValues / voltageLoopCount;
  voltage = (averagePotValue / (potOffset + (averagePotValue * voltageMultiplier)) * voltageDividerRatio);

  averageWattage = (accumulatedAmps / voltageLoopCount) * ACDCVoltage;

  //////////////////////
  // SERVER RELAY



  // if voltage is below 9 the batteries are basically dead!
  if (!debug && (voltageLoopCount >= samplesPerReading && voltage >= deadBatteryVoltage) || serverFailed)
{

    accumulatedPotValues = 0;
    accumulatedAmps = 0;
    voltageLoopCount = 0;

    setWifiSleepMode(false);

    String recordedAverageVoltage = String(voltage, 2);
    String recordedAverageWattage = String(averageWattage, 0);

    recordedAverageVoltage.trim();
    recordedAverageWattage.trim();

    http.begin(voltateUrl + recordedAverageVoltage + "," + recordedAverageWattage);
    int httpCode = http.GET();
    if (httpCode > 0) {
      String payload = http.getString();
      Serial.println("HTTP Response: " + payload);
      recordPingSucces();
    } else {
      Serial.println("HTTP Request failed with error code: " + String(httpCode));
      recordPingFailure();
    }
    http.end();



    setWifiSleepMode(true);
  }
```

```
voltageLoopCount++;


//////////////////////////
// LCD DISPLAY

switch (displayMode) {
  case 1:
    wattagePerHourText.trim();
    wattageVoltageText.trim();

    if (prevWattagePerHourText != wattagePerHourText || prevWattageVoltageText != wattageVoltageText) {
      lcd_i2c.clear();
    }

    prevWattagePerHourText = wattagePerHourText;
    prevWattageVoltageText = wattageVoltageText;

    lcd_i2c.setCursor(0, 0);
    lcd_i2c.print(wattagePerHourText);
    lcd_i2c.setCursor(0, 1);
    lcd_i2c.print(wattageVoltageText);

    break;

  case 2:
    String connectivityText1 = "";
    if (wifiConnected) {
      connectivityText1 += "WF ON";
    }
    if (wifiConnected && !serverFailed) {
      connectivityText1 += " SERVER ON";
    }

    String serverPings = String(totalServerPings, 0);
    serverPings.trim();

    String serverPingFails = String(totalServerPingFails, 0);
    serverPingFails.trim();
```

```
      String connectivityText2 = serverPings + "/" + serverPingFails + " PINGS";

      connectivityText1.trim();
      connectivityText2.trim();

      if (prevConnectivityText1 != connectivityText1 || prevConnectivityText2 != connectivityText2) {
        lcd_i2c.clear();
      }

      prevConnectivityText1 = connectivityText1;
      prevConnectivityText2 = connectivityText2;

      lcd_i2c.setCursor(0, 0);
      lcd_i2c.print(connectivityText1);
      lcd_i2c.setCursor(0, 1);
      lcd_i2c.print(connectivityText2);
      break;
    }


  delay(pulseRate);
}

bool connectToWiFi() {

  if (connectingToWifi || wifiConnected) {
    return true;
  }

  connectingToWifi = true;

  String activeSsid = "";
  String activePassword = "";

  if (activeConnection == 1) {
    activeSsid = ssid1;
    activePassword = password1;
  } else if (activeConnection == 2) {
    activeSsid = ssid2;
```

```
  activePassword = password2;
}

Serial.print("Connecting to WiFi: ");
Serial.println(activeSsid);

WiFi.begin(activeSsid, activePassword);

while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
  delay(500);
  Serial.print(".");
  wifiConnectionAttempts++;
}

wifiConnectionAttempts = 0;

if (WiFi.status() == WL_CONNECTED) {
  Serial.println("\nConnected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  EEPROM.write(eepromActiveConnectionAddress, activeConnection);
  EEPROM.commit();

  Serial.print("set activeConnection to : ");
  Serial.print(activeConnection);

  wifiConnected = true;
  connectingToWifi = false;
  return true;

} else {
  Serial.print("Connection to ");
  Serial.print(activeSsid);
  Serial.println(" failed. Trying alternative");

  wifiConnected = false;
  connectingToWifi = false;

  return false;
```

```
  }
}



/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

  wifiSleeping = sleepMode;

  if (sleepMode) {
    WiFi.disconnect();
    WiFi.setSleep(true);
    wifiConnected = false;
    delay(1000);
    Serial.print("sleep wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
  } else {
    WiFi.setSleep(false);
    WiFi.reconnect();
    delay(2000);
    Serial.print("awaken wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
    // Check if the connection is still active. if not trigger wait for it to come back online
    if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
      Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
      WiFi.disconnect();
      wifiPaused = true;
      wifiConnected = false;
      connectToWiFi();
    }
  }
}

/**
 * record server ping success in long term memory
 */
void recordPingSucces() {
```

```
  totalServerPings++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = true;
  serverFailed = false;
}

/**
 * record server ping fails in long term memory
 */
void recordPingFailure() {
  totalServerPingFails++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = false;
  serverFailed = true;
}

/**
 * ESP32 wifi card statuses
 * @param status
 * @return string
 */
String wl_status_to_string(wl_status_t status) {

  String response = "";

  switch (status) {
    case WL_NO_SHIELD:
      response = "WL_NO_SHIELD";
      break;
    case WL_IDLE_STATUS:
      response = "WL_IDLE_STATUS";
      break;
    case WL_NO_SSID_AVAIL:
      response = "WL_NO_SSID_AVAIL";
```

```
      break;
    case WL_SCAN_COMPLETED:
      response = "WL_SCAN_COMPLETED";
      break;
    case WL_CONNECTED:
      response = "WL_CONNECTED";
      break;
    case WL_CONNECT_FAILED:
      response = "WL_CONNECT_FAILED";
      break;
    case WL_CONNECTION_LOST:
      response = "WL_CONNECTION_LOST";
      break;
    case WL_DISCONNECTED:
      response = "WL_DISCONNECTED";
      break;
  }

  return response;
}

/**
 * Get the current in amps coming from the hall sensor
 * @return float
 */
float getAmps() {
  float hallSensorVoltage;
  float current;
  float sum = 0;
  long time_check = millis();
  int counter = 0;

  while (millis() - time_check < 1000) {
    hallSensorVoltage = ads.readADC_Differential_0_1() * hallSensorFudge;  // get voltage from hall sensor with
fugdge
    current = hallSensorVoltage * FACTOR;                                  // 1 volt = 20 amps with current sensor

    sum += sq(current);
    counter = counter + 1;
  }
```

```
  current = sqrt(sum / counter);
  return (current);
}


/**
 * read the accumulated amps divided by readings
 */
float getAverageAmps(float array[], int size) {

  float accumulatedValues = 0;
  int ampCounts = 0;

  for (int i = 0; i < size - 1; i++) {
    if (array[i] >= 0) {
      ampCounts++;
      accumulatedValues += array[i];
    }
  }

  return accumulatedValues / ampCounts;
}


/**
 * read the total amp over a 24 hour periods
 */
float getDailyAmpHours() {

  float accumulatedValues = 0;
  for (int i = 0; i < hoursPerDay - 1; i++) {
    if (dailyAmpsArray[i] >= 0) {
      accumulatedValues += dailyAmpsArray[i];
    }
  }

  return accumulatedValues;
}
```

```
/**
 * get average wattage from samples
 * @param int sampleCount
 * @return float
 */
float getLastXWattReadings(int sampleCount) {

  float accumulatedValues = 0;

  int countedValues = 0;

  for (int i = secondsPerHour; i > 0; i--) {
    if (houryAmpsArray[i - 1] >= 0) {
      accumulatedValues += houryAmpsArray[i - 1];
      countedValues++;
    }
    if (countedValues >= sampleCount) {
      break;
    }
  }

  return accumulatedValues / sampleCount;
}



/**
 * remove first item from array, shift all value to left and add new value to end.
 */
void cycleHourlyAmpsArray(float newValue) {

  // this means we the array if full so we can begin shifting
  if (houryAmpsArray[secondsPerHour - 1] >= 0) {
    for (int i = 0; i < secondsPerHour - 1; i++) {
      houryAmpsArray[i] = houryAmpsArray[i + 1];
    }
    houryAmpsArray[secondsPerHour - 1] = newValue;
```

```
  } else {
    // allow the array to initialize with real values
    houryAmpsArray[houryAmpsArrayCurrentIndex] = newValue;
    houryAmpsArrayCurrentIndex++;
    if (houryAmpsArrayCurrentIndex > secondsPerHour) {
      houryAmpsArrayCurrentIndex = secondsPerHour - 1;
    }
  }
}



/**
 * remove first item from array, shift all value to left and add new value to end.
 */
void cycleDailyAmpsArray(float newValue) {

  // this means we the array if full so we can begin shifting
  if (dailyAmpsArray[hoursPerDay - 1] >= 0) {
    for (int i = 0; i < hoursPerDay - 1; i++) {
      dailyAmpsArray[i] = dailyAmpsArray[i + 1];
    }
    dailyAmpsArray[hoursPerDay - 1] = newValue;

  } else {
    // allow the array to initialize with real values
    dailyAmpsArray[dailyAmpsArrayCurrentIndex] = newValue;
    dailyAmpsArrayCurrentIndex++;
    if (dailyAmpsArrayCurrentIndex > hoursPerDay) {
      dailyAmpsArrayCurrentIndex = hoursPerDay - 1;
    }
  }
}
```

https://www.youtube.com/embed/pHUaLFN6yoc?si=XX7-Mq9xULj_qPtG