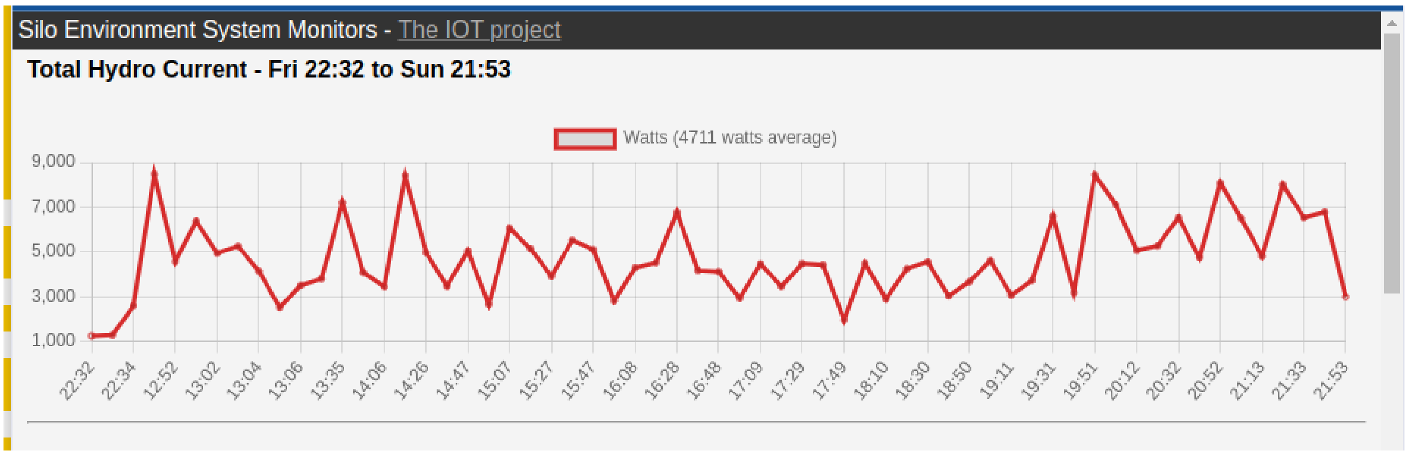
Home Power Consumption Monitoring

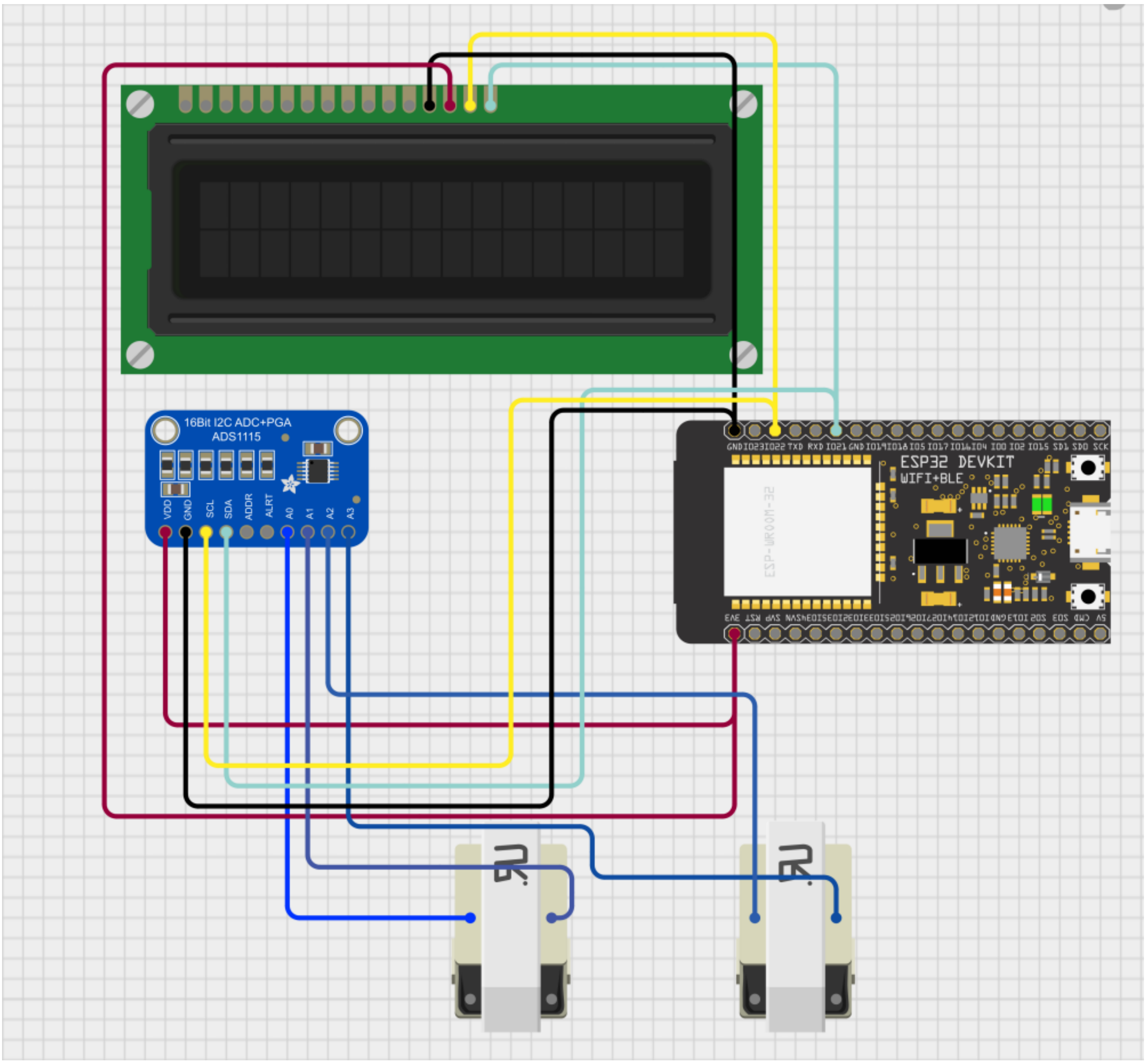
For this project the goal is to know exactly how much current is being used for the entire property. To collect this data I placed 2 SCT013 non-invasive split core current transformers around the phase 1 and 2 wires in the main electrical panel. Using some simple math this can convert the small voltage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


16Bit I2C ADC+PGA
ADS1115
VDD
GND
SCL
SDA
ADDR
ALRT
A0
A1
A2
A3
ESP32 DEVKIT
WIFI+BLE
ESP-WROOM-32
GND IO23 IO22 TXD RXD IO21 GND IO19 IO18 IO5 IO17 IO16 IO4 IO0 IO2 IO15 SD1 SD0 SCK
3V3 EN SVP SVN IO34 IO35 IO32 IO33 IO25 IO26 IO27 IO14 IO12 GND IO13 SD2 SD3 CMD 5V

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;

 }
```

Revision #19
Created 16 February 2024 21:35:12 by peter drinnan
Updated 23 June 2024 17:22:11 by peterd