EECS 3215

Home Thermal Management System

Group Members:

Varsha Ragavendran, ID: 213193065

Abasifreke James, ID: 211951035

Francis Dagher, ID: 212894606

Nazia Zafor, ID: 211366424

Professor: Ebrahim Ghafar-Zadeh

Date: 17.04.17

Abstract
The Home Thermal Management System project is an wifi-enabled embedded system that measures and displays the change in temperatures around the house whilst alerting the user of unusual temperature changes via an alarm system. As engineering students, we have gained a lot of theoretical knowledge over the course of our undergraduate years, therefore we chose to build this particular system as a project, because it is an excellent combination of our knowledge of electric circuits, software, and IoT. Our overall goal is to gain deep practical experience in building embedded systems.

Introduction
The objective of the Home Thermal Management System is to measure the temperature and calculate the temperature change around the house using temperature sensors in order to maintain constant temperature and conserve energy. To implement this system, we will be using two (2) battery-powered temperature sensors for reading temperatures at different locations in the home, an ESP8266 Thing Dev microcontroller for reading from a central server and communicating with the display unit (OLED display), two other ESP8266 wifi modules for reading from the temperature sensors and writing to the server, and finally a piezoelectric speaker for sounding an alarm.

Method

Hardware Components

- 1 Microcontrollers
    - ESP8266 Sparkfun Thing Dev Board Microcontroller
- 2 WiFi modules
    - ESP8266 wifi module
- 2 Temperature sensors
    - DS18B20
- 2 4.7K Ohm Resistors
- SparkFun Micro OLED Breakout Board
- 4 AA Batteries
- 1 Alarm – piezo speaker
- 20 Jumper Wires

Software components

- Arduino IDE
- Phant.io server/database

- Software Libraries included:
    - ESP8266WiFi.h
    - Phant.h
    - ArduinoJSON.h
    - ER_MicroOLED.h
    - OneWire.h
    - DallasTemperature.h

Procedure

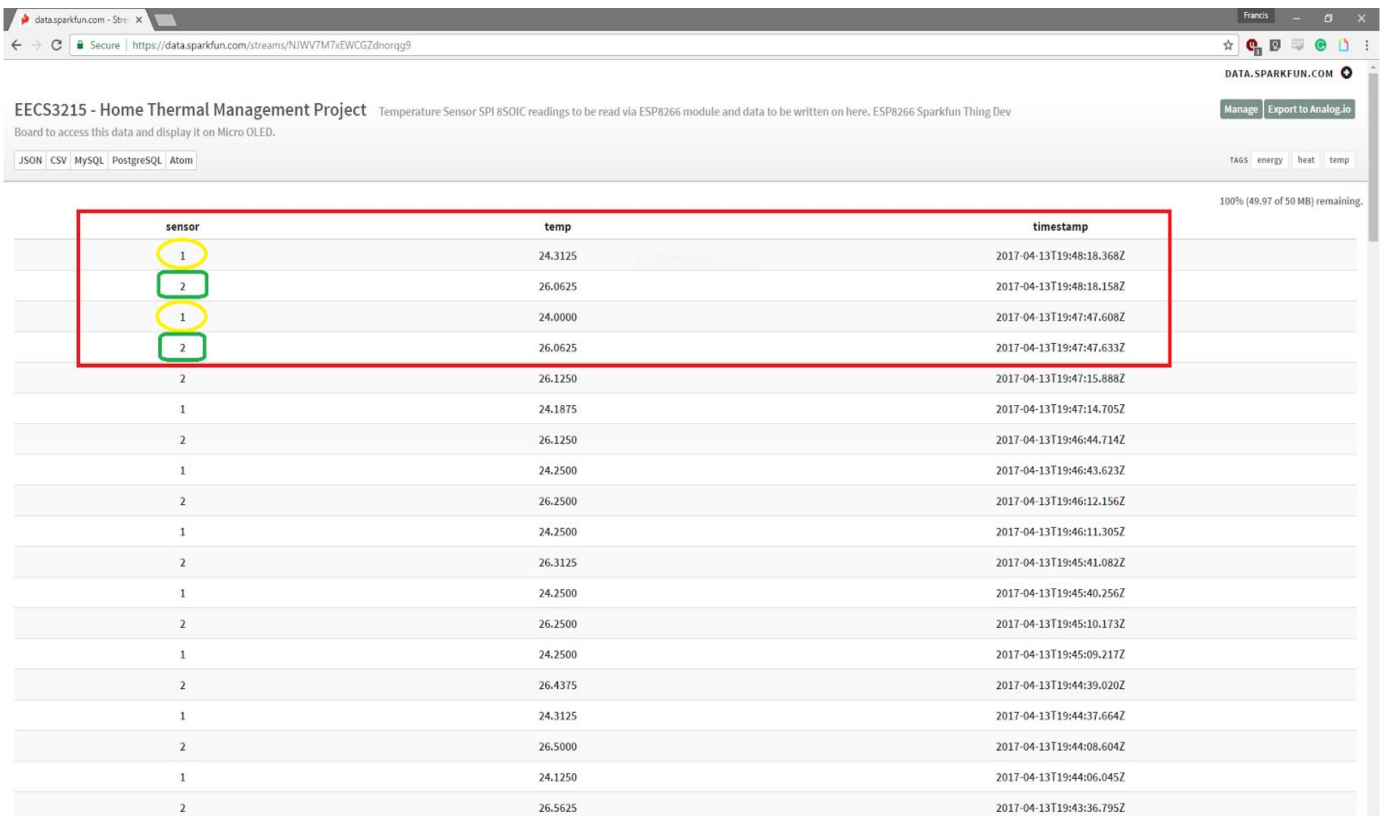
In order to demonstrate the various functions of the system, we tested on the temperatures of two different rooms. A temperature probe is placed in each room and the temperatures measured accordingly. The OLED display then displays the current temperature and the change in temperature (after a certain period of time) for each room. Following the successful display on the device, temperature of one of the rooms is increased using a hair-dryer to above the pre-programmed threshold temperature level of the device. This is done in order to test the alarm system installed with the device which is programmed to ring as soon as the temperature of the surrounding goes above a specific threshold. Thus, the demonstration tests all three functions of the system : measurement of temperature, display of temperature and temperature change and alarm system.

The following paragraph talks about the connections of the embedded system.
The Sparkfun Thing Dev board is connected to the 'Micro OLED breakout' via an I2C connection. Pins D2, and D14 of the Sparkfun board are connected to D1, and D0 of the OLED respectively. The other pins we needed to use of both units are power and ground. These connections allow for Sparkfun to display onto the OLED. The speaker is connected to pin 4 of the Sparkfun. This can be seen in our Result section. In order for the unit to function as we want it, we need to program it. We used Arduino IDE and its library in order to program the ESP8266 Sparkfun.
The Thing Dev MC read the latest records from our central server (Phant.io). Each record is a tuple of sensor and temp. values (sensor, temp), and the MC used these to calculate the temperature changes for each room housing the temp sensors. These values are read in 30 seconds intervals, and if the temperature rises or drops below a threshold of +/- 5 ℃, it signals the alarms to alert the user.

The other two ESP8266 Micro Controllers are connected to the DS18B20 temperature sensor. The middle pin of the sensor is a 'One Wire' connection. Thus it is connected to the GPIO pin of the ESP8266. This together with the OneWire.h and DallasTemperature.h libraries allow the ESP8266 to read the temperature from the temperature sensors. The other important pins used for both is power and ground, and a 'pull up' resistor measuring at 4.7K Ohms is used in the circuitry to ensure that the signal will be a valid logic level. In the code module written for these two ESP8266 MCs, we also included the ESP8266WIFI.h and Phant.h libraries which help us write the readings gotten from the temperature sensors, to our central server (Phant.io).

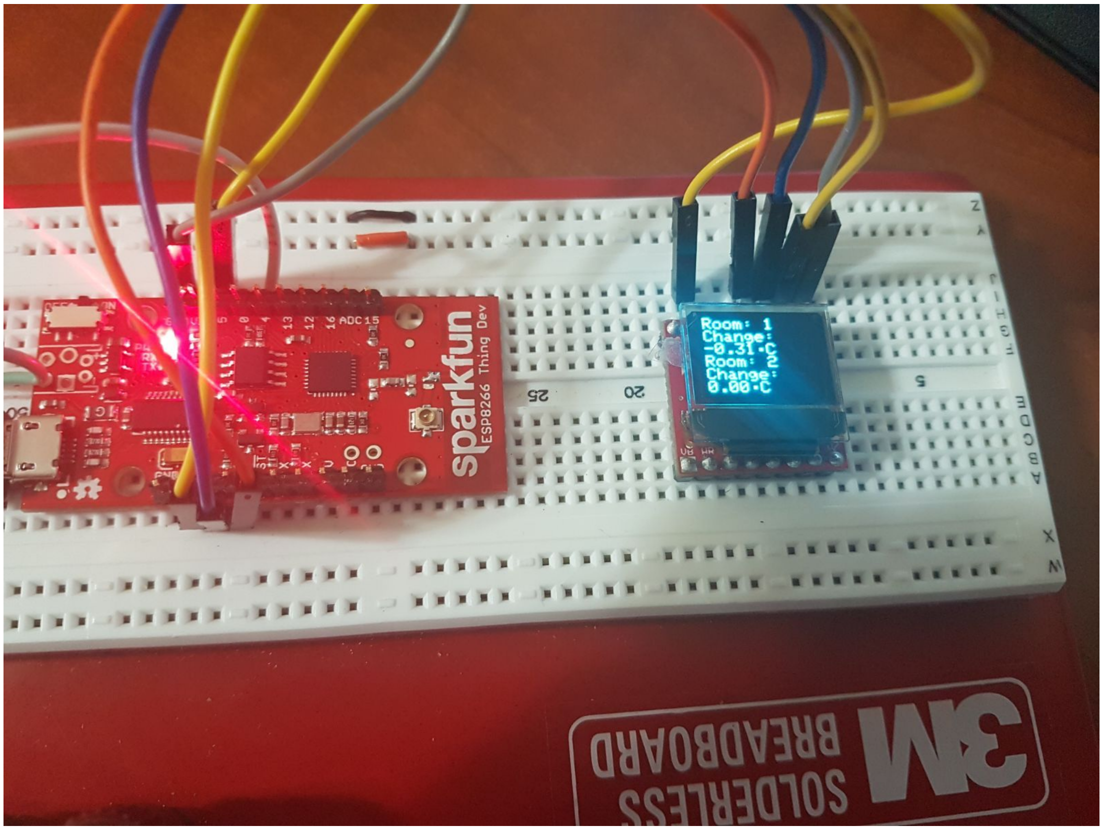
Results

The two main integral parts of our results are the uploaded data (temperature) on the server and the display of the results on the OLED screen. The following two figures show the results accordingly:



sensor	temp	timestamp
1	24.3125	2017-04-13T19:48:18.368Z
2	26.0625	2017-04-13T19:48:18.158Z
1	24.0000	2017-04-13T19:47:47.608Z
2	26.0625	2017-04-13T19:47:47.633Z
2	26.1250	2017-04-13T19:47:15.888Z
1	24.1875	2017-04-13T19:47:14.705Z
2	26.1250	2017-04-13T19:46:44.714Z
1	24.2500	2017-04-13T19:46:43.623Z
2	26.2500	2017-04-13T19:46:12.156Z
1	24.2500	2017-04-13T19:46:11.305Z
2	26.3125	2017-04-13T19:45:41.082Z
1	24.2500	2017-04-13T19:45:40.256Z
2	26.2500	2017-04-13T19:45:10.173Z
1	24.2500	2017-04-13T19:45:09.217Z
2	26.4375	2017-04-13T19:44:39.020Z
1	24.3125	2017-04-13T19:44:37.664Z
2	26.5000	2017-04-13T19:44:08.604Z
1	24.1250	2017-04-13T19:44:06.045Z
2	26.5625	2017-04-13T19:43:36.795Z

Sparkfun (Phant) Server. Saves which sensor and its temperature



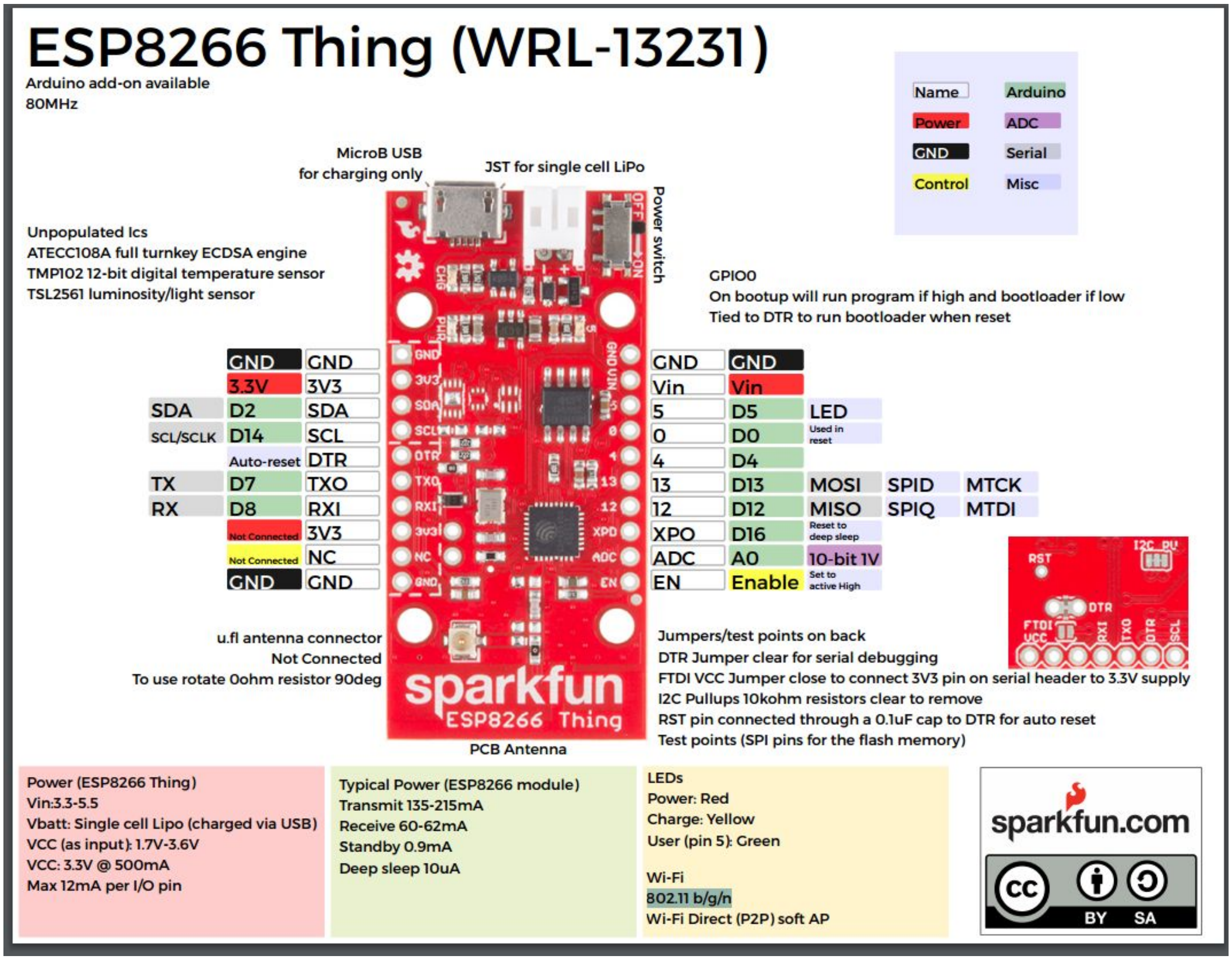
The main display, reading the difference between the two readings of each sensor

Discussion

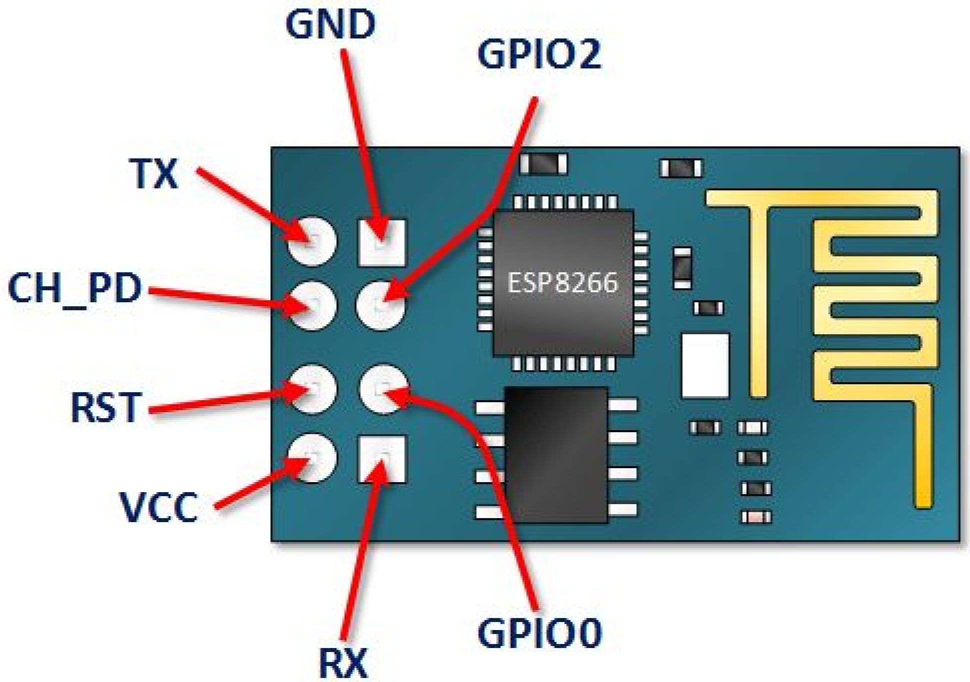
The Arduino IDE was used to program the ESP8266 Sparkfun thing dev microcontroller. First, a wifi connection was established with the use of the *ESP8266WiFi* Library. Once connected to WiFi, a connection to SparkFun's Phant server was established with the help of the *Phant* library by issuing an HTTP GET request with the public key for the data stream our team created on the Phant server. In the GET request, we programmed the microcontroller to limit the data we retrieve to just the last 4 data inputs that was logged on the server. Two of the data inputs will be from sensor 1 and the other two will be from sensor 2. Data is retrieved in the form of JSON, therefore we took advantage of the *ArduinoJSON* library. This library is used to parse the retrieved data in order to find which sensor sent the data to the server and the temperature reading from that sensor.

After this point, the difference of the two temperatures received from each sensor is calculated. If the difference is more than +/- 5 ℃, the piezo speakers are programmed to play a sound and alarm that there is a huge difference in temperature. The *ER_MicroOLED* library acts as the communication link between our microcontroller and the OLED in order to display the most recent temperature readings on the OLED display.

The Sparkfun ESP8266 Micro Controller is able to download data via the WiFi protocol 802.11 b/g/n. Below is a graphical datasheet of the Sparkfun board.

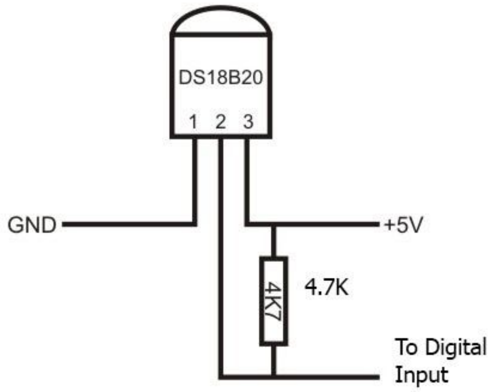


If an SPI connection would need to be made, pins 13, 12 and SCL/D14. These pins are needed because SPI connections need MOSI, MOSI, and SCLK. The micro USB is used for powering the device as well as programming it.The microcontroller has an integrated USB-to-serial chip thus the USB-to-serial converter allows to program the device without the need of any peripheral components. If the microcontroller has a program loaded onto it, once powered, it will immediately start running the code.

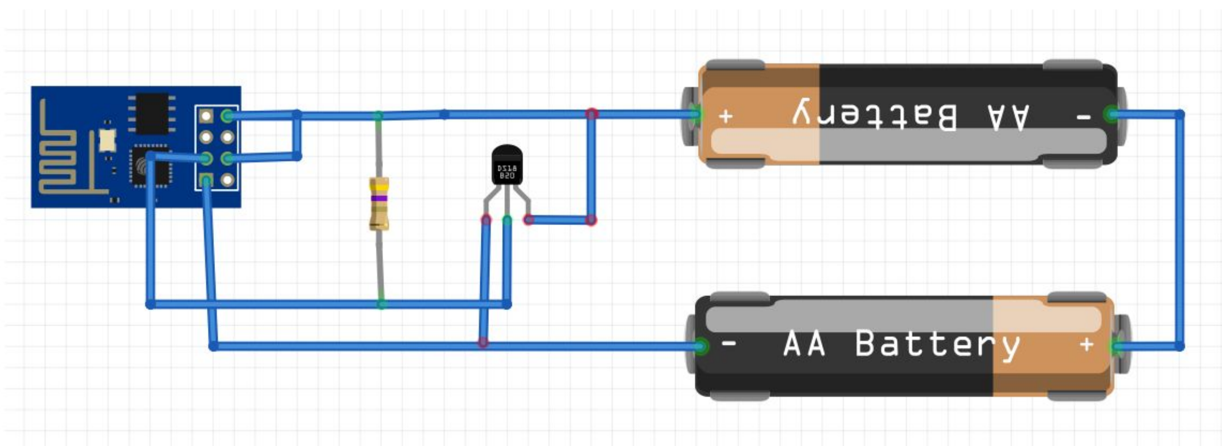


The ESP8266 WiFi controller has a total of 8 pins. One for power and ground, and two for general purpose I/O (GPIO0 and GPIO2). TX and CH_PD allowed us to program the ESP8266 through an Arduino board. The center chip that says ESP8266 is the CPU of the microcontroller. On the left of the board is the WiFi antenna. The WiFi protocol is the same as of the Sparkfun controller 802.11 b/g/n.

Originally, the temperature sensor of choice for our project was the ADT7310 IC which connects to the microcontroller through an SPI connection.. However, after learning in the course about different protocols and standards of ICs, we realized that we cannot have any temperature sensor to work with the ESP8266. This model of the ESP8266 does not support SPI, and thus needed to change our sensor. The first instinct was to get a regular analog temperature sensor, but we then realized that this ESP does not have an ADC either. We decided on getting the DS18B20 sensor instead. This sensor only has 3 pins: Power, Ground and the ‘One Wire’ connection. We connect the middle pin to one of our GPIO pins and the ESP is able to read the temperature readings. The advantage of One Wire over SPI is that it only needs one pin while SPI needs 4. SPI, however, is a much faster connection than ‘One Wire’. Since we only need to read temperature and not much data is being transferred between the sensor and the ESP, One Wire in our case is much beneficial, even if our ESP supported SPI.



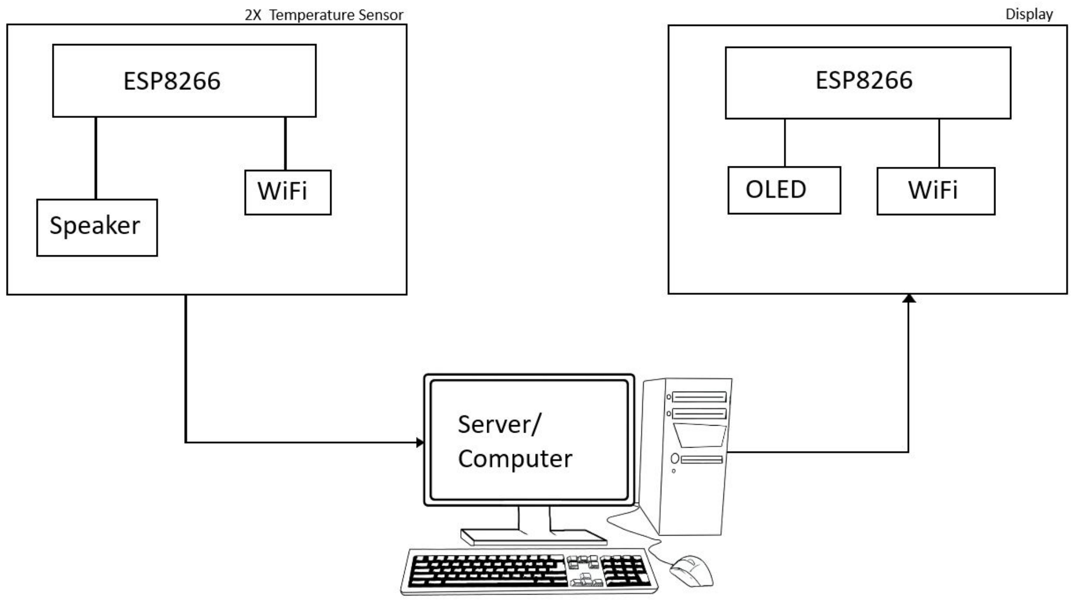
We connected the sensor to our microcontroller as follows:



The batteries are used to make the whole system portable.

The ESP8266 and the Sparkfun ESP8266 both have an energy saving mode called deep sleep and standby. This allows the microcontroller to not use energy when it doesn't need to. In deep sleep mode, the ESP is almost off. We can take advantage of this by sending a Real-Time Interrupt every 5-10 min to read the temperature. This will save battery and the user will not need to replace them often.

The overall view of our embedded system is as seen below:



Conclusion

The Home Thermal Management system is a system in which it lets the user know the change in temperature in a specific room. This project builds on the theme of IoT, Internet of Things. Since IoT is becoming very important in everyday life, it is important to learn and understand what we can achieve from this field. This project can be expanded by adding energy saving features such as deep sleep mode in the ESP microcontroller. Also, it can be expanded into a fully automated Home Energy/Thermal system in which the sensor can see which rooms need more heat or cooling than others. By working on this project, we learned how to connect different ICs in order to communicate with each other. We also learned the importance of reading data sheets. By reading each components data sheet, we understood why the original temperature sensor did not function with the ESP8266 and knew which type of sensor should be used.

Code

I. Code of the (2) ESP8266 WIFI Module MicroController (used for Temperature Sensing:

```
// Including the ESP8266 WiFi library. (Works a lot like the Arduino WiFi library.)
#include <ESP8266WiFi.h>
// Including the SparkFun Phant library.
#include <Phant.h>
// Libraries for working with the DS18B20 Digital temperature sensor
#include <OneWire.h>
#include <DallasTemperature.h>

//////////////////////
// WiFi Definitions //
//////////////////////
const char WiFiSSID[] = "YOUR_WIFI_SSID";
const char WiFiPSK[] = "YOUR_WIFI_PASSWORD";

// Data wire is plugged into pin D1 on the ESP8266 01S - GPIO 2
#define ONE_WIRE_BUS 2
// Setup a oneWire instance to communicate with any OneWire devices (not just
Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature.
DallasTemperature DS18B20(&oneWire);
char temperatureCString[6];
char temperatureFString[6];

/////////////////////
// Pin Definitions //
/////////////////////
const int LED_PIN = 5; // Thing's onboard, green LED
const int ANALOG_PIN = A0; // The only analog pin on the Thing
const int DIGITAL_PIN = 12; // Digital pin to be read

////////////////
// Phant Keys //
////////////////
const char PhantHost[] = "data.sparkfun.com";
const char PublicKey[] = "NJWV7M7xEWCGZdnorqg9";
const char PrivateKey[] = "5dPvwjwE7PIVKJGkNR6M";

/////////////////
// Post Timing //
/////////////////
const unsigned long postRate = 30000;
unsigned long lastPost = 0;

void setup()
{
  initHardware();
```

```
  DS18B20.begin(); // IC Default 9 bit. If you have troubles consider upping it 12. Ups
the delay giving the IC more time to process the temperature measurement
  connectWiFi();
  digitalWrite(LED_PIN, HIGH);
}

void loop()
{
  if (lastPost + postRate <= millis())
  {
    if (postToPhant())
      lastPost = millis();
    else
      delay(100);
  }
}

void connectWiFi()
{
  byte ledStatus = LOW;

  // Set WiFi mode to station (as opposed to AP or AP_STA)
  WiFi.mode(WIFI_STA);

  // WiFI.begin([ssid], [passkey]) initiates a WiFI connection
  // to the stated [ssid], using the [passkey] as a WPA, WPA2,
  // or WEP passphrase.
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WiFiSSID);

 // WiFi.begin(ssid, password);
  WiFi.begin(WiFiSSID, WiFiPSK);

  // Use the WiFi.status() function to check if the ESP8266
  // is connected to a WiFi network.
  while (WiFi.status() != WL_CONNECTED)
  {
    // Blink the LED
    digitalWrite(LED_PIN, ledStatus); // Write LED high/low
    ledStatus = (ledStatus == HIGH) ? LOW : HIGH;

    // Delays allow the ESP8266 to perform critical tasks
    // defined outside of the sketch. These tasks include
    // setting up, and maintaining, a WiFi connection.
    delay(100);
    // Potentially infinite loops are generally dangerous.
    // Add delays -- allowing the processor to perform other
    // tasks -- wherever possible.
  }
```

```
  Serial.println("");
  Serial.println("WiFi connected");

}

float getTemperature() {
  float tempC;
  float tempF;
  do {
    DS18B20.requestTemperatures();
    tempC = DS18B20.getTempCByIndex(0);
    dtostrf(tempC, 2, 2, temperatureCString);
    tempF = DS18B20.getTempFByIndex(0);
    dtostrf(tempF, 3, 2, temperatureFString);
    delay(100);
  } while (tempC == 85.0 || tempC == (-127.0));

  return tempC;
}
void initHardware()
{
  Serial.begin(9600);
  pinMode(DIGITAL_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  // Don't need to set ANALOG_PIN as input,
  // that's all it can be.
}

int postToPhant()
{
  // LED turns on when we enter, it'll go off when we
  // successfully post.
  digitalWrite(LED_PIN, HIGH);

  // Declare an object from the Phant library - phant
  Phant phant(PhantHost, PublicKey, PrivateKey);

  // Do a little work to get a unique-ish name. Append the
  // last two bytes of the MAC (HEX'd) to "Thing-":
  uint8_t mac[WL_MAC_ADDR_LENGTH];
  WiFi.macAddress(mac);
  String macID = String(mac[WL_MAC_ADDR_LENGTH - 2], HEX) +
                 String(mac[WL_MAC_ADDR_LENGTH - 1], HEX);
  macID.toUpperCase();
  String postedID = "ThingJames-" + macID;

  // Add the four field/value pairs defined by our stream:
//  phant.add("id", postedID);
//  phant.add("analog", analogRead(ANALOG_PIN));
```

```
//  phant.add("digital", digitalRead(DIGITAL_PIN));
//  phant.add("time", millis());
  float temp = getTemperature();
  Serial.print("The temperature is :");
  Serial.println(temp);
  phant.add("sensor", 2);
  phant.add("temp", temp);

  // Now connect to data.sparkfun.com, and post our data:
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(PhantHost, httpPort))
  {
    // If we fail to connect, return 0.
    return 0;
  }
  // If we successfully connected, print our Phant post:
  client.print(phant.post());

  // Read all the lines of the reply from server and print them to Serial
  while(client.available()){
    String line = client.readStringUntil('\r');
    //Serial.print(line); // Trying to avoid using serial
  }

  // Before we exit, turn the LED off.
  digitalWrite(LED_PIN, LOW);

  return 1; // Return success
}
```

II. <u>Code of the ESP8266 SparkFun Thing Dev Board MicroController module:</u>

```
#include <ESP8266WiFi.h>
#include <Phant.h>
#include <ER_MicroOLED.h>
#include <ArduinoJson.h>

#define PIN_RESET 5
#define PIN_DC 8
#define DC_JUMPER 1
MicroOLED oled(PIN_RESET, DC_JUMPER); // Example I2C declaration

//////////////////////
// WiFi Definitions //
//////////////////////
const char WiFiSSID[] = "************"; // username of WIFI network
const char WiFiPSK[] = "**********"; // password of WIFI network
```

```
/////////////////////
// Pin Definitions //

////////////////
// Phant Keys //
////////////////
const char PhantHost[] = "data.sparkfun.com";
const char PublicKey[] = "********";  //public key of our phant server data stream
const char PrivateKey[] = "*********"; // private key of our phant server data stream

/////////////////
// Post Timing //
/////////////////
const unsigned long postRate = 30000;
unsigned long lastPost = 0;
int speakerPin = 4;


int length = 2; // the number of notes
char notes[] = "ab"; // a space represents a rest
int beats[] = { 1, 1, };
int tempo = 300;
void setup()
{
  initHardware();
  connectWiFi();
  pinMode(speakerPin, OUTPUT);
}

void loop()
{
  if (lastPost + postRate <= millis())
  {
    if (postToPhant())
      lastPost = millis();
    else
      delay(100);
  }
}

void connectWiFi()
{
 // byte ledStatus = LOW;

  // Set WiFi mode to station (as opposed to AP or AP_STA)
  WiFi.mode(WIFI_STA);

  // WiFI.begin([ssid], [passkey]) initiates a WiFI connection
  // to the stated [ssid], using the [passkey] as a WPA, WPA2,
  // or WEP passphrase.
```

```
  WiFi.begin(WiFiSSID, WiFiPSK);

  // Use the WiFi.status() function to check if the ESP8266
  // is connected to a WiFi network.
  while (WiFi.status() != WL_CONNECTED)
  {
    // Blink the LED
   // digitalWrite(LED_PIN, ledStatus); // Write LED high/low
    //ledStatus = (ledStatus == HIGH) ? LOW : HIGH;

    // Delays allow the ESP8266 to perform critical tasks
    // defined outside of the sketch. These tasks include
    // setting up, and maintaining, a WiFi connection.
    delay(100);
    // Potentially infinite loops are generally dangerous.
    // Add delays -- allowing the processor to perform other
    // tasks -- wherever possible.
  }
}

void initHardware()
{
  Serial.begin(9600);
  //pinMode(DIGITAL_PIN, INPUT_PULLUP);
  //pinMode(LED_PIN, OUTPUT);
  //digitalWrite(LED_PIN, LOW);
  // Don't need to set ANALOG_PIN as input,
  // that's all it can be.
}

int postToPhant()
{

  // Declare an object from the Phant library - phant
  Phant phant(PhantHost, PublicKey, PrivateKey);


  // Add the four field/value pairs defined by our stream:
  //phant.add("temp", 85);
  //phant.add("sensor",1);

    // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(PhantHost, httpPort)) {
    Serial.println("connection failed");
    return 0;
  }

  // If we successfully connected, print our Phant post:
```

```
  //client.print(phant.post());

  // We now create a URI for the request
  String url = "/output/";
  url += PublicKey;
  url += ".json?limit=4";

  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + PhantHost + "\r\n" +
               "Connection: close\r\n\r\n");

  // Read all the lines of the reply from server and print them to Serial
  // Response is shown between the lines
  Serial.println("_____________________________");
  String line = "";
 // const char *b[2];
  while(client.available()){
    line = client.readString();
    Serial.print(line);
   // b[0] = line.c_str();
  }
  Serial.println("PRINTING LINES OF ARRAY");
 /* int i = 0;
  for(i=0; i < 4 ; i++){
    Serial.println(b[i]);
  }*/
 // Serial.println(b[0]);
if(line.length() == 0){
  return 0;
}
int firstbracket = line.indexOf('[');
  int secondbracket = line.indexOf(']', firstbracket + 1);
  line.remove(secondbracket+1);
  line.remove(0,firstbracket);
 /* char json[512];
//  json = line;

//------------------------------
  StaticJsonBuffer<200> jsonBuffer; //JSON
  // Root of the object tree.
  //
  // It's a reference to the JsonObject, the actual bytes are inside the
  // JsonBuffer with all the other nodes of the object tree.
  // Memory is freed when jsonBuffer goes out of scope.
    Serial.println(line);
```

```
  JsonObject& root = jsonBuffer.parseObject(line);

  // Test if parsing succeeds.
  if (!root.success()) {
    Serial.println("parseObject() failed");
    return 0;
  }

  // Fetch values.
  //
  // Most of the time, you can rely on the implicit casts.
  // In other case, you can do root["time"].as<long>();
  const char* sensor = root["sensor"].asString();
const char* temp = root["temp"].asString();
  // Print values.
  Serial.println(sensor);
  Serial.println(temp); */

    int firstOpenBracket = line.indexOf('{');
    int secondOpenBracket = line.indexOf('{', firstOpenBracket+1);
    int thirdOpenBracket = line.indexOf('{', secondOpenBracket+1);
    int fourthOpenBracket = line.indexOf('{', thirdOpenBracket+1);

    String firstValue = line.substring(firstOpenBracket, secondOpenBracket-5);
    String secondValue = line.substring(secondOpenBracket, thirdOpenBracket-5);
    String thirdValue = line.substring(thirdOpenBracket, fourthOpenBracket-5);
    String fourthValue = line.substring(fourthOpenBracket, line.length()-4);

    //{"sensor":"15","temp":"5","timestamp":"2017-03-26T09:02:22.702Z"}


    /** Get time of first sensor **/
    int firstSCharacter = firstValue.indexOf('s');

    String timeOfValue = line.substring(firstSCharacter + 52, firstSCharacter + 57);
    Serial.print("Time stamp: ");
    Serial.println(timeOfValue);

    /** FIRST SENSOR & TEMPERATURE VALUE **/

    String valueOfFirstSensor ="";
    if(!firstValue.charAt(firstSCharacter +10) == '"'){

        valueOfFirstSensor = firstValue.substring(firstSCharacter + 9, firstSCharacter
+ 10);

    }
    else{
```

```
        valueOfFirstSensor = firstValue.substring(firstSCharacter + 9,firstSCharacter +
10);

    }
    Serial.print("VALUE OF 1st SENSOR: ");
    Serial.println(valueOfFirstSensor);
    int firstTCharacter = firstValue.indexOf('t');
    String valueOfFirstTemp ="";
    if(!firstValue.charAt(firstTCharacter + 9) == '"'){

        valueOfFirstTemp = firstValue.substring(firstTCharacter + 7, firstTCharacter +
12);

    }
    else{

        valueOfFirstTemp = firstValue.substring(firstTCharacter + 7,firstTCharacter +
12);

    }
    Serial.print("VALUE OF 1st TEMP: ");
    Serial.println(valueOfFirstTemp);

    /** SECOND SENSOR & TEMPERATURE VALUE **/
    int secondSCharacter = secondValue.indexOf('s');
    String valueOfSecondSensor ="";
    if(!secondValue.charAt(secondSCharacter +10) == '"'){
      valueOfSecondSensor = secondValue.substring(secondSCharacter + 9,
secondSCharacter + 10);
    }
    else{
      valueOfSecondSensor = secondValue.substring(secondSCharacter + 9,secondSCharacter
+ 10);
    }
    Serial.print("VALUE OF 2nd SENSOR: ");
    Serial.println(valueOfSecondSensor);
    int secondTCharacter = secondValue.indexOf('t');
    String valueOfSecondTemp ="";
    if(!secondValue.charAt(firstTCharacter + 9) == '"'){
      valueOfSecondTemp = secondValue.substring(secondTCharacter + 7, secondTCharacter
+ 12);
    }
    else{
      valueOfSecondTemp = secondValue.substring(secondTCharacter + 7,secondTCharacter +
12);
    }
    Serial.print("VALUE OF 2nd TEMP: ");
    Serial.println(valueOfSecondTemp);

     /** THIRD SENSOR & TEMPERATURE VALUE **/
```

```
    int thirdSCharacter = thirdValue.indexOf('s');
    String valueOfThirdSensor ="";
    if(!thirdValue.charAt(thirdSCharacter +10) == '"'){
      valueOfThirdSensor = thirdValue.substring(thirdSCharacter + 9, thirdSCharacter +
10);
    }
    else{
      valueOfThirdSensor = thirdValue.substring(thirdSCharacter + 9,thirdSCharacter +
10);
    }
    Serial.print("VALUE OF 3rd SENSOR: ");
    Serial.println(valueOfThirdSensor);
    int thirdTCharacter = thirdValue.indexOf('t');
     String valueOfThirdTemp ="";
    if(!thirdValue.charAt(firstTCharacter + 9) == '"'){
      valueOfThirdTemp = thirdValue.substring(thirdTCharacter + 7, thirdTCharacter +
12);
    }
    else{
      valueOfThirdTemp = thirdValue.substring(thirdTCharacter + 7,thirdTCharacter +
12);
    }
    Serial.print("VALUE OF 3rd TEMP: ");
    Serial.println(valueOfThirdTemp);

    /** FOURTH SENSOR & TEMPERATURE VALUE **/
    int fourthSCharacter = fourthValue.indexOf('s');
    String valueOfFourthSensor ="";
    if(!fourthValue.charAt(fourthSCharacter +10) == '"'){

        valueOfFourthSensor = fourthValue.substring(fourthSCharacter + 9,
fourthSCharacter + 10);

    }
    else{
      valueOfFourthSensor = fourthValue.substring(fourthSCharacter + 9,fourthSCharacter
+ 10);
    }
    Serial.print("VALUE OF 4th SENSOR: ");
    Serial.println(valueOfFourthSensor);
    int fourthTCharacter = fourthValue.indexOf('t');
    String valueOfFourthTemp ="";
    if(!fourthValue.charAt(firstTCharacter + 9) == '"'){
      valueOfFourthTemp = fourthValue.substring(fourthTCharacter + 7, fourthTCharacter
+ 12);
    }
    else{
      valueOfFourthTemp = fourthValue.substring(fourthTCharacter + 7,fourthTCharacter +
12);
    }
```

```
    Serial.print("VALUE OF 4th TEMP: ");
    Serial.println(valueOfFourthTemp);

    /*****************Temperature Difference**************************/

    float roomOne = valueOfThirdTemp.toFloat() - valueOfFirstTemp.toFloat();
    float roomTwo = valueOfFourthTemp.toFloat() - valueOfSecondTemp.toFloat();

  if( roomOne < 5 && roomOne > -5 || roomTwo < 5 && roomTwo > -5  ) {

  for (int i = 0; i < length; i++) {
    if (notes[i] == ' ') {
      delay(beats[i] * tempo); // rest
    } else {
      playNote(notes[i], beats[i] * tempo);
    }

    // pause between notes
    delay(tempo / 1);
}
  }

    /** For now OLED to display just 1 reading **/
    oled.begin();
    oled.clear(PAGE);
    oled.clear(ALL);
    oled.setCursor(0,0);
    oled.setTextColor(WHITE);
    oled.setTextSize(0);
    /********* Room 1*****************/
    oled.print("Room: ");
    oled.println(valueOfFirstSensor);
    oled.println("Change: ");
    oled.print(roomOne);
    oled.print(char(248));
    oled.println("C");
    /********* Room 2****************/
    oled.print("Room: ");
    oled.println(valueOfSecondSensor);
    oled.println("Change: ");
    oled.print(roomTwo);
    oled.print(char(248));
    oled.println("C");
    oled.display();

  Serial.println("_____________________________");

  Serial.println();
  Serial.println("closing connection");
```

```
  return 1; // Return success
}
void playTone(int tone, int duration) {
  for (long i = 0; i < duration * 1000L; i += tone * 2) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(tone);
    digitalWrite(speakerPin, LOW);
    delayMicroseconds(tone);
  }
}

void playNote(char note, int duration) {
  char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
  int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };

  // play the tone corresponding to the note name
  for (int i = 0; i < 8; i++) {
    if (names[i] == note) {
      playTone(tones[i], duration);
    }
  }
}
```

References

1. "Micro OLED Hookup guide for ESP8266 and Particle Photon." *Learn with Edwin Robotics*. N.p., n.d. Web. 17 Mar. 2017. <http://learn.edwinrobotics.com/micro-oled-hookup-guide-for-esp8266-and-particle-photon/>.

2. "SparkFun ESP8266 Thing - Dev Board." *WRL-13711 - SparkFun Electronics*. N.p., n.d. Web. 17 Apr. 2017. <https://www.sparkfun.com/products/13711>

3. User, Super. "ESP8266 WiFi DS18B20 Temperature Sensor (Arduino IDE)." Iot-playground. N.p., n.d. Web. 17 Apr. 2017. <http://iot-playground.com/blog/2-uncategorised/41-esp8266-ds18b20-temperature-sensor-arduino-ide>.

4. ESP8266 Thing (WRL-13231) (n.d.): n. pag. Sparkfun. Web. 16 Apr. 2017. <https://cdn.sparkfun.com/datasheets/Wireless/WiFi/ESP8266ThingV1.pdf>

.

5. "ESP8266 ESP-01 Pin Outs and Schematics." Henry's Bench. N.p., 06 Feb. 2017. Web. 16 Apr. 2017. <http://henrysbench.capnfatz.com/henrys-bench/arduino-projects-tips-and-more/esp8266-esp-01-pin-outs-and-schematics/>.

Datasheets

ESP8266: http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf

DS18B20: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

Piezo Buzzer: https://product.tdk.com/info/en/catalog/datasheets/ef532_ps.pdf

ADT7310: http://www.analog.com/media/en/technical-documentation/data-sheets/ADT7310.pdf