Alex Shah
Project 5 - ESP8266 Weather Station
2024-03-17
EN.605.715.81.SP24

Table of Contents

Requirements

The requirements for the weather station are to assemble the weather station kit, with a temperature and humidity sensor dht11, an arduino with esp8266 wifi, and oled screen. The wifi should connect to a network in order to receive openweathermap data to display from the api, and send thingspeak data to the thingspeak api. The oled screen should display weather data.

Design

I implemented the weather station kit with the dht11 sensor and arduino with esp8266 on a breadboard with wires connecting the screen to sda, scl, vcc, and ground to arduino pins d3, d4. The dht11 is connected with signal wire in the middle to d5 pin, and power and ground on the external pins. The other sensors were connected as described in the figure included in the weather station documentation, but not utilized.

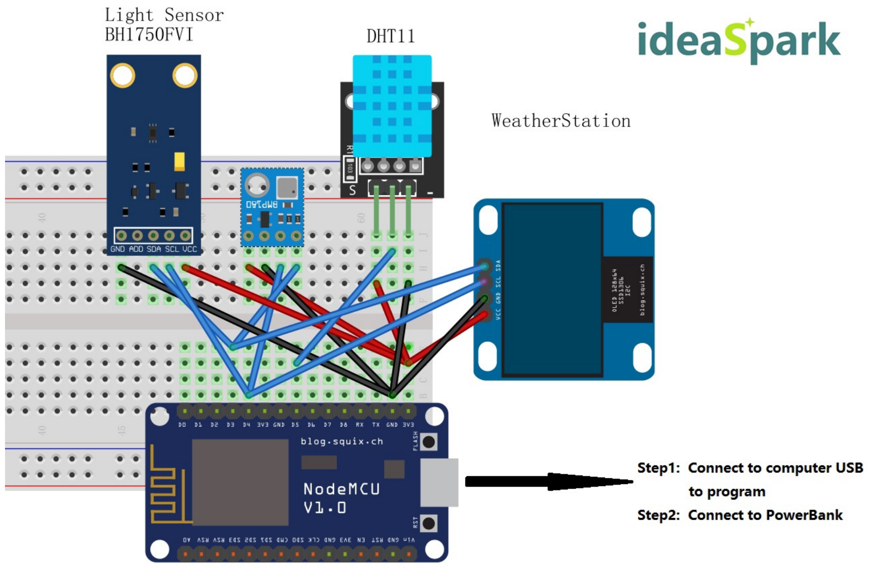

Figure 1. Board Design

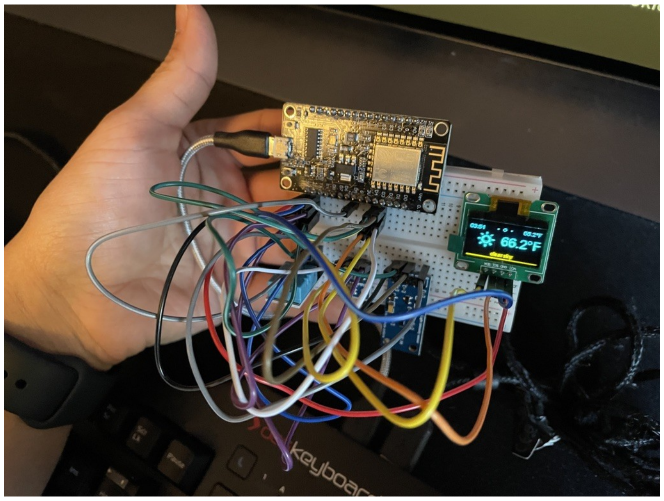

Figure 2. Board Implementation

Implementation

The project was breadboarded and assembled, then the code was abstracted from the weather station example WeatherStationDemo.ino in order to display weather data from openweathermap, as well as send back temperature and humidity data to thingspeak.

Please see WeatherStation.ino in the appendix.

Demo

Please see an example video showing the project running, and data sent back to thingspeak api appearing on screen.

https://drive.google.com/file/d/1L-ZHL_B-4WEMn7UQs-o77NeVDBxbv2BV/view?usp=sharing

References

https://github.com/GJKJ/WSKS/blob/master/Guide%20Manual(Read%20Me%20First).doc

Appendix

```WeatherStation.ino
```

```
#include <Arduino.h>
#if defined(ESP8266)
#include <ESP8266WiFi.h>
#include <coredecls.h> // settimeofday_cb()
#else
#include <WiFi.h>
#endif
#include <ESPHTTPClient.h>
#include <JsonListener.h>
// time
#include <time.h>     // time() ctime()
#include <sys/time.h> // struct timeval
#include "SSD1306Wire.h"
#include "OLEDDisplayUi.h"
#include "Wire.h"
#include "OpenWeatherMapCurrent.h"
#include "OpenWeatherMapForecast.h"
#include "WeatherStationFonts.h"
#include "WeatherStationImages.h"
#include <Adafruit_BMP085.h>

/***************************
 * WIFI Settings
 **************************/
const char *WIFI_SSID = "####";
const char *WIFI_PWD = "####";

/***************************
 * Begin DHT11 Settings
 **************************/
```

```
WiFiClient client;
const char *host = "api.thingspeak.com";  // IP address of the thingspeak server
const char *api_key = "####"; // Your own thingspeak api_key
const int httpPort = 80;
#define pin 14 // ESP8266-12E  D5 read emperature and Humidity data
int temp = 0;  // temperature
int humi = 0;  // humidity
void readTemperatureHumidity();
void uploadTemperatureHumidity();
long readTime = 0;
long uploadTime = 0;

/***************************
 * Begin Atmosphere and Light Sensor Settings
 **************************/
// void readLight();
// void readAtmosphere();
// Adafruit_BMP085 bmp;
// const int Light_ADDR = 0b0100011;   // address:0x23
// const int Atom_ADDR = 0b1110111;  // address:0x77
// int tempLight = 0;
// int tempAtom = 0;

/***************************
 * Begin Settings
 **************************/
#define TZ 5      // (utc+) TZ in hours
#define DST_MN 60 // use 60mn for summer time in some countries

// Setup
```

```
const int UPDATE_INTERVAL_SECS = 20 * 60; // Update every 20 minutes
// Display Settings
const int I2C_DISPLAY_ADDRESS = 0x3c;
#if defined(ESP8266)
// const int SDA_PIN = D1;
// const int SDC_PIN = D2;

const int SDA_PIN = D3;
const int SDC_PIN = D4;
#else
// const int SDA_PIN = GPIO5;
// const int SDC_PIN = GPIO4

const int SDA_PIN = GPIO0;
const int SDC_PIN = GPIO2
#endif

// OpenWeatherMap Settings
// Sign up here to get an API key:
// https://docs.thingpulse.com/how-tos/openweathermap-key/
const boolean IS_METRIC = false;
// Add your own thingpulse ID
String OPEN_WEATHER_MAP_APP_ID = "####";
String OPEN_WEATHER_MAP_LOCATION = "New York,NY,US";

// Pick a language code from this list:
// Arabic - ar, Bulgarian - bg, Catalan - ca, Czech - cz, German - de, Greek - el,
// English - en, Persian (Farsi) - fa, Finnish - fi, French - fr, Galician - gl,
// Croatian - hr, Hungarian - hu, Italian - it, Japanese - ja, Korean - kr,
// Latvian - la, Lithuanian - lt, Macedonian - mk, Dutch - nl, Polish - pl,
// Portuguese - pt, Romanian - ro, Russian - ru, Swedish - se, Slovak - sk,
// Slovenian - sl, Spanish - es, Turkish - tr, Ukrainian - ua, Vietnamese - vi,
// Chinese Simplified - zh_cn, Chinese Traditional - zh_tw.

String OPEN_WEATHER_MAP_LANGUAGE = "en";
const uint8_t MAX_FORECASTS = 4;

// Adjust according to your language
const String WDAY_NAMES[] = {"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};
const String MONTH_NAMES[] = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};

/***************************
 * End Settings
 **************************/
// Initialize the oled display for address 0x3c
SSD1306Wire display(I2C_DISPLAY_ADDRESS, SDA_PIN, SDC_PIN);
OLEDDisplayUi ui(&display);

OpenWeatherMapCurrentData currentWeather;
OpenWeatherMapCurrent currentWeatherClient;

OpenWeatherMapForecastData forecasts[MAX_FORECASTS];
OpenWeatherMapForecast forecastClient;

#define TZ_MN ((TZ) * 60)
#define TZ_SEC ((TZ) * 3600)
#define DST_SEC ((DST_MN) * 60)
time_t now;

```

```
// flag changed in the ticker function every 10 minutes
bool readyForWeatherUpdate = false;
String lastUpdate = "--";
long timeSinceLastWUpdate = 0;
// declaring prototypes
void drawProgress(OLEDDisplay *display, int percentage, String label);
void updateData(OLEDDisplay *display);
void drawDateTime(OLEDDisplay *display, OLEDDisplayUiState *state, int16_t x, int16_t y);
void drawCurrentWeather(OLEDDisplay *display, OLEDDisplayUiState *state, int16_t x, int16_t
y);
void drawForecast(OLEDDisplay *display, OLEDDisplayUiState *state, int16_t x, int16_t y);
void drawForecastDetails(OLEDDisplay *display, int x, int y, int dayIndex);
void drawHeaderOverlay(OLEDDisplay *display, OLEDDisplayUiState *state);
void setReadyForWeatherUpdate();

// Add frames
// this array keeps function pointers to all frames
// frames are the single views that slide from right to left
FrameCallback frames[] = {drawDateTime, drawCurrentWeather, drawForecast};
int numberOfFrames = 3;

OverlayCallback overlays[] = {drawHeaderOverlay};
int numberOfOverlays = 1;

void setup()
{
   Serial.begin(115200);

   Wire.begin(0, 2);

```

```
//  Wire.beginTransmission(Atom_ADDR);
//  //initialize Atmosphere sensor
//  if (!bmp.begin()) {
//    Serial.println("Could not find BMP180 or BMP085 sensor at 0x77");
//  }else{
//    Serial.println("Find BMP180 or BMP085 sensor at 0x77");
//  }
//  Wire.endTransmission();
//
//  //initialize light sensor
//  Wire.beginTransmission(Light_ADDR);
//  Wire.write(0b00000001);
//  Wire.endTransmission();

// initialize dispaly
display.init();
display.clear();
display.display();

// display.flipScreenVertically();
display.setFont(ArialMT_Plain_10);
display.setTextAlignment(TEXT_ALIGN_CENTER);
display.setContrast(255);

WiFi.begin(WIFI_SSID, WIFI_PWD);

int counter = 0;
while (WiFi.status() != WL_CONNECTED)
{
   delay(500);
```

```
    Serial.print(".");
    display.clear();
    display.drawString(64, 10, "Connecting to WiFi");
    display.drawXbm(46, 30, 8, 8, counter % 3 == 0 ? activeSymbole : inactiveSymbole);
    display.drawXbm(60, 30, 8, 8, counter % 3 == 1 ? activeSymbole : inactiveSymbole);
    display.drawXbm(74, 30, 8, 8, counter % 3 == 2 ? activeSymbole : inactiveSymbole);
    display.display();

    counter++;
  }
  // Get time from network time service
  configTime(TZ_SEC, DST_SEC, "pool.ntp.org");
  ui.setTargetFPS(30);
  ui.setActiveSymbol(activeSymbole);
  ui.setInactiveSymbol(inactiveSymbole);
  // You can change this to
  // TOP, LEFT, BOTTOM, RIGHT
  ui.setIndicatorPosition(BOTTOM);
  // Defines where the first frame is located in the bar.
  ui.setIndicatorDirection(LEFT_RIGHT);
  // You can change the transition that is used
  // SLIDE_LEFT, SLIDE_RIGHT, SLIDE_TOP, SLIDE_DOWN
  ui.setFrameAnimation(SLIDE_LEFT);
  ui.setFrames(frames, numberOfFrames);
  ui.setOverlays(overlays, numberOfOverlays);
  // Inital UI takes care of initalising the display too.
  ui.init();
  Serial.println("");
  updateData(&display);
  while (!client.connect(host, httpPort))
```

```
    {
        Serial.println("Connection Failed");
    }
}

void loop()
{
    // Read Temperature Humidity every 5 seconds
    if (millis() - readTime > 5000)
    {
        readTemperatureHumidity();
        // readLight();
        // readAtmosphere();
        readTime = millis();
    }
    // Upload Temperature Humidity every 60 seconds
    if (millis() - uploadTime > 60000)
    {
        uploadTemperatureHumidity();
        uploadTime = millis();
    }

    if (millis() - timeSinceLastWUpdate > (1000L * UPDATE_INTERVAL_SECS))
    {
        setReadyForWeatherUpdate();
        timeSinceLastWUpdate = millis();
    }

    if (readyForWeatherUpdate && ui.getUiState()->frameState == FIXED)
    {
        updateData(&display);
    }

    int remainingTimeBudget = ui.update();

    if (remainingTimeBudget > 0)
    {
        // You can do some work here
        // Don't do stuff if you are below your
        // time budget.
        delay(remainingTimeBudget);
    }
}

void drawProgress(OLEDDisplay *display, int percentage, String label)
{
    display->clear();
    display->setTextAlignment(TEXT_ALIGN_CENTER);
    display->setFont(ArialMT_Plain_10);
    display->drawString(64, 10, label);
    display->drawProgressBar(2, 28, 124, 10, percentage);
    display->display();
}

void updateData(OLEDDisplay *display)
{
    drawProgress(display, 10, "Updating time...");
    drawProgress(display, 30, "Updating weather...");
    currentWeatherClient.setMetric(IS_METRIC);
    currentWeatherClient.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
    currentWeatherClient.updateCurrent(&currentWeather, OPEN_WEATHER_MAP_APP_ID, OPEN_WEATHER_MAP_LOCATION);
    drawProgress(display, 50, "Updating forecasts...");
    forecastClient.setMetric(IS_METRIC);
    forecastClient.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
    uint8_t allowedHours[] = {12};
    forecastClient.setAllowedHours(allowedHours, sizeof(allowedHours));
    forecastClient.updateForecasts(forecasts, OPEN_WEATHER_MAP_APP_ID, OPEN_WEATHER_MAP_LOCATION, MAX_FORECASTS);
    readyForWeatherUpdate = false;
    drawProgress(display, 100, "Done...");
    delay(1000);
}

void drawDateTime(OLEDDisplay *display, OLEDDisplayUiState *state, int16_t x, int16_t y)
{
    now = time(nullptr);
    struct tm *timeInfo;
    timeInfo = localtime(&now);
    char buff[16];

    display->setTextAlignment(TEXT_ALIGN_CENTER);
    display->setFont(ArialMT_Plain_10);
    String date = WDAY_NAMES[timeInfo->tm_wday];

    sprintf_P(buff, PSTR("%s, %02d/%02d/%04d"), WDAY_NAMES[timeInfo->tm_wday].c_str(), timeInfo->tm_mday, timeInfo->tm_mon + 1, timeInfo->tm_year + 1900);
    display->drawString(64 + x, 5 + y, String(buff));
    display->setFont(ArialMT_Plain_24);

    sprintf_P(buff, PSTR("%02d:%02d:%02d"), timeInfo->tm_hour, timeInfo->tm_min, timeInfo->tm_sec);
    display->drawString(64 + x, 15 + y, String(buff));
    display->setTextAlignment(TEXT_ALIGN_LEFT);
}

void drawCurrentWeather(OLEDDisplay *display, OLEDDisplayUiState *state, int16_t x, int16_t y)
{
    display->setFont(ArialMT_Plain_10);
    display->setTextAlignment(TEXT_ALIGN_CENTER);
    display->drawString(64 + x, 38 + y, currentWeather.description);

    display->setFont(ArialMT_Plain_24);
    display->setTextAlignment(TEXT_ALIGN_LEFT);
    String temp = String(currentWeather.temp, 1) + (IS_METRIC ? "°C" : "°F");
    display->drawString(60 + x, 5 + y, temp);

    display->setFont(Meteocons_Plain_36);
    display->setTextAlignment(TEXT_ALIGN_CENTER);
    display->drawString(32 + x, 0 + y, currentWeather.iconMeteoCon);
}

void drawForecast(OLEDDisplay *display, OLEDDisplayUiState *state, int16_t x, int16_t y)
{
    drawForecastDetails(display, x, y, 0);
    drawForecastDetails(display, x + 44, y, 1);
    drawForecastDetails(display, x + 88, y, 2);
}

```

```
void drawForecastDetails(OLEDDisplay *display, int x, int y, int dayIndex)
{
    time_t observationTimestamp = forecasts[dayIndex].observationTime;
    struct tm *timeInfo;
    timeInfo = localtime(&observationTimestamp);
    display->setTextAlignment(TEXT_ALIGN_CENTER);
    display->setFont(ArialMT_Plain_10);
    display->drawString(x + 20, y, WDAY_NAMES[timeInfo->tm_wday]);

    display->setFont(Meteocons_Plain_21);
    display->drawString(x + 20, y + 12, forecasts[dayIndex].iconMeteoCon);
    String temp = String(forecasts[dayIndex].temp, 0) + (IS_METRIC ? "°C" : "°F");
    display->setFont(ArialMT_Plain_10);
    display->drawString(x + 20, y + 34, temp);
    display->setTextAlignment(TEXT_ALIGN_LEFT);
}

void drawHeaderOverlay(OLEDDisplay *display, OLEDDisplayUiState *state)
{
    now = time(nullptr);
    struct tm *timeInfo;
    timeInfo = localtime(&now);
    char buff[14];
    sprintf_P(buff, PSTR("%02d:%02d"), timeInfo->tm_hour, timeInfo->tm_min);

    display->setColor(WHITE);
    display->setFont(ArialMT_Plain_10);
    display->setTextAlignment(TEXT_ALIGN_LEFT);
    display->drawString(0, 54, String(buff));
    display->setTextAlignment(TEXT_ALIGN_RIGHT);
```

```
  String temp = String(currentWeather.temp, 1) + (IS_METRIC ? "°C" : "°F");
  display->drawString(128, 54, temp);
  display->drawHorizontalLine(0, 52, 128);
}

void setReadyForWeatherUpdate()
{
  Serial.println("Setting readyForUpdate to true");
  readyForWeatherUpdate = true;
}

// read temperature humidity data
void readTemperatureHumidity()
{
  int j;
  unsigned int loopCnt;
  int chr[40] = {0};
  unsigned long time1;
bgn:
  delay(2000);
  // Set interface mode 2 to: output
  // Output low level 20ms (>18ms)
  // Output high level 40μs
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  delay(20);
  digitalWrite(pin, HIGH);
  delayMicroseconds(40);
  digitalWrite(pin, LOW);
  // Set interface mode 2: input
```

```
pinMode(pin, INPUT);
// High level response signal
loopCnt = 10000;
while (digitalRead(pin) != HIGH)
{
    if (loopCnt-- == 0)
    {
        // If don't return to high level for a long time, output a prompt and start over
        Serial.println("HIGH");
        goto bgn;
    }
}
// Low level response signal
loopCnt = 30000;
while (digitalRead(pin) != LOW)
{
    if (loopCnt-- == 0)
    {
        // If don't return low for a long time, output a prompt and start over
        Serial.println("LOW");
        goto bgn;
    }
}
// Start reading the value of bit1-40
for (int i = 0; i < 40; i++)
{
    while (digitalRead(pin) == LOW)
    {
    }
    // When the high level occurs, write down the time "time"
```

```
        time1 = micros();
        while (digitalRead(pin) == HIGH)
        {
        }
        // When there is a low level, write down the time and subtract the time just saved
        // If the value obtained is greater than 50μs, it is ‘1’, otherwise it is ‘0’
        // And save it in an array
        if (micros() - time1 > 50)
        {
            chr[i] = 1;
        }
        else
        {
            chr[i] = 0;
        }
    }

    // Humidity, 8-bit bit, converted to a value
    humi = chr[0] * 128 + chr[1] * 64 + chr[2] * 32 + chr[3] * 16 + chr[4] * 8 + chr[5] * 4 + chr[6] * 2 + chr[7];
    // Temperature, 8-bit bit, converted to a value
    temp = chr[16] * 128 + chr[17] * 64 + chr[18] * 32 + chr[19] * 16 + chr[20] * 8 + chr[21] * 4 + chr[22] * 2 + chr[23];

    Serial.print("temp:");
    Serial.print(temp);
    Serial.print("    humi:");
    Serial.println(humi);
}

```

```
// void readLight(){
//   // reset
//   Wire.beginTransmission(Light_ADDR);
//   Wire.write(0b00000111);
//   Wire.endTransmission();
//
//   Wire.beginTransmission(Light_ADDR);
//   Wire.write(0b00100000);
//   Wire.endTransmission();
//   // typical read delay 120ms
//   delay(120);
//   Wire.requestFrom(Light_ADDR, 2); // 2byte every time
//   for (tempLight = 0; Wire.available() >= 1; ) {
//     char c = Wire.read();
//     tempLight = (tempLight << 8) + (c & 0xFF);
//   }
//   tempLight = tempLight / 1.2;
//   Serial.print("light: ");
//   Serial.println(tempLight);
// }
//
//
// void readAtmosphere(){
//   tempAtom = bmp.readPressure();
//   Serial.print("Pressure = ");
//   Serial.print(tempAtom);
//   Serial.println(" Pascal");
// }

// upload temperature humidity data to thinkspak.com
```

```
void uploadTemperatureHumidity()
{
    if (!client.connect(host, httpPort))
    {
        Serial.println("connection failed");
        return;
    }
    // Three values(field1 field2 field3 field4) have been set in thingspeak.com
    client.print(String("GET ") + "/update?api_key=" + api_key + "&field1=" + temp + "&field2=" + humi + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: close\r\n\r\n");
    while (client.available())
    {
        String line = client.readStringUntil('\r');
        Serial.print(line);
    }
}
```