Outdoor weather station documentation

Contents

1. Product description

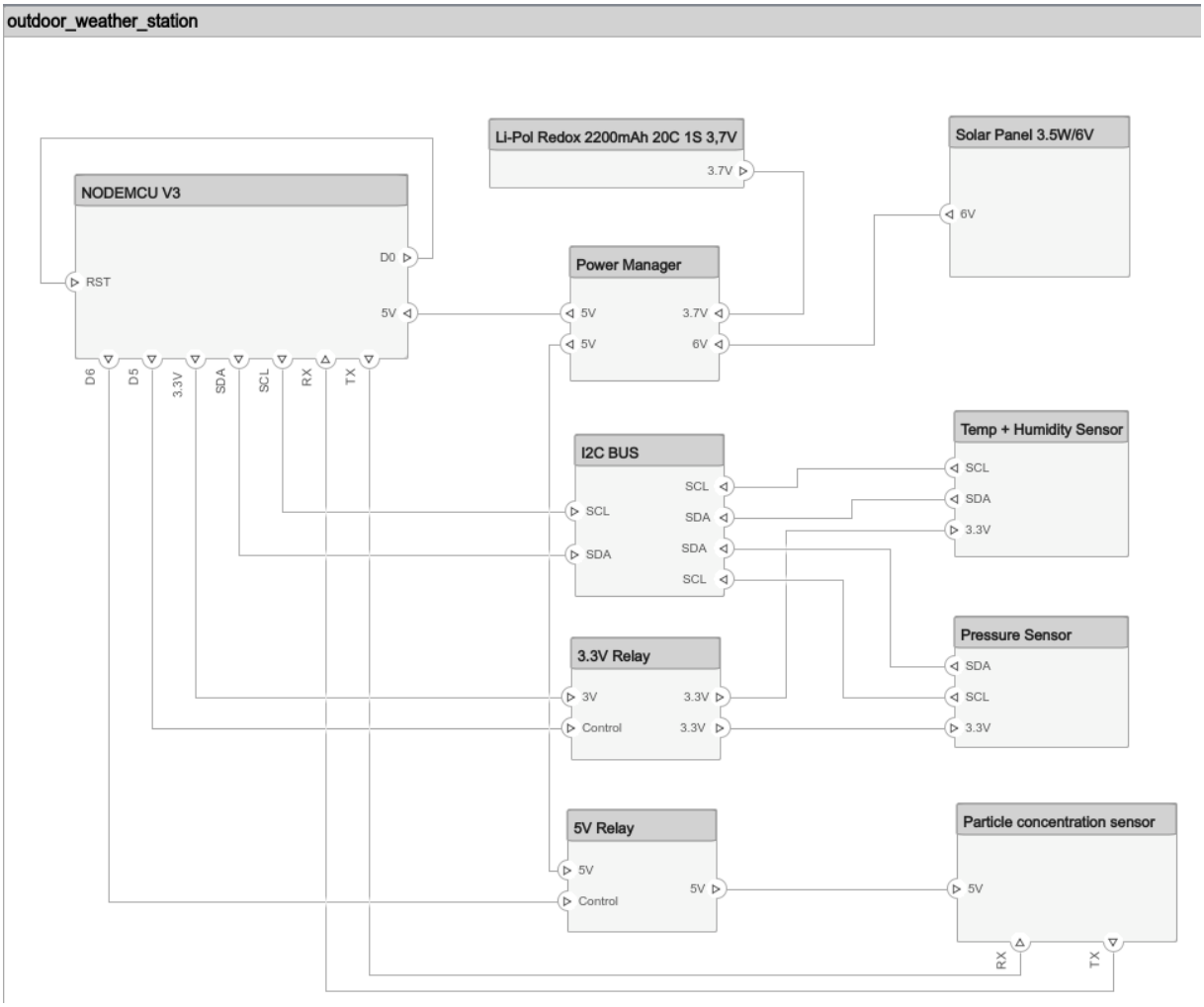
This project was designed to build fully functional and autonomous weather station which would work on the outside. Stations is divided into two main blocks, bigger one contains microcontroller, relay module and power managing board It is designed to be sealed and be safe from outdoor conditions. Second, smaller compartment contains all of the sensors. Case has ventilations holes to enable sensors work properly and read current weather condition. In addition to that, sealed cables are running from smaller box to solar panel from which battery is charged.

2. Hardware

1. General hardware information

Project was aiming to achieve autonomous work being fully powered by solar panel and battery. To achieve this, microcontroller ESP32 in form of NodeMCU V3 was used. Reason was its deep sleep mode and Wi-Fi connectivity. Additionally relay modules were used to cut off power from all sensors for time of sleeping and turning it on for time of measurements.

2. Simplified station diagram



3. Parts list

Name	Catalog number	Store Website
Microcontroller	ESP8266 + NodeMCU v3	https://botland.com.pl/moduly-wifi-esp8266/8241-modul-wifi-esp8266-nodemcu-v3-5904422300630.html
Power Converter	DFRobot Solar Power Manager	https://botland.com.pl/akcesoria-do-paneli-slonecznych/14423-dfrobot-solar-power-manager-modul-zarzadzania-energia-sloneczna-5v-5903351243841.html
Solar Panel	Solar Panel 3,5W/6V	https://botland.com.pl/panele-sloneczne-malej-mocy/6988-ogniwo-sloneczne-35w6v-165x135x3mm-5904422335328.html
Battery	Li-Pol Redox 2200mAh 20C 1S 3,7V	https://botland.com.pl/akumulatory-li-pol-1s-37v-/1313-ogniwo-li-pol-redox-2200mah-20c-1s-37v-5903754000690.html
Temp + Humidity Sensor	SHT31-F - DFRobot	https://botland.com.pl/czujniki-temperatury/17380-sht31-f-cyfrowy-czujnik-temperatury-i-wilgotnosci-i2c-dfrobot-sen0334-5904422378578.html
Pressure Sensor	BMP280 - DFRobot	https://botland.com.pl/czujniki-cisnienia/19198-cyfrowy-czujnik-cisnienia-atmosferycznego-barometr-bmp280-i2c-dfrobot-sen0372-5904422361655.html
Particle concentration sensor	PMS3003	https://botland.com.pl/czujniki-czystosci-powietrza/13442-czujnik-pylu-czystosci-powietrza-pm10-pm25-pm10-pms3003-5v-uart-5904422366735.html
Relays Module	Module with two SRD-05 relays	https://botland.com.pl/moduly-przekaznikow/14266-modul-przekaznikow-iduino-2-kanaly-z-optoizolacja-styki-10a250vac-cewka-5v-5903351242332.html
Case for sensors	Kradex Z123	https://botland.com.pl/obudowy/10124-obudowa-plastikowa-wentylowana-kradex-z123-76x76x30-czarna-5904422376444.html
Case for Main Electronics	Kradex Z80J IP54	https://botland.com.pl/obudowy/6199-obudowa-plastikowa-kradex-z80j-ip54-120x90x38mm-jasna-5904422302276.html

3. Software

1. General software information

Software was written in Arduino IDE with usage of external libraries. Mainly library to handle NodeMCU V3 was used. Libraries to make communication with sensors was problematic due to the differences between ESP32 and Arduino pinout. This required big changes in library for humidity and temperature sensor and writing our own library to handle UART communication with PMS sensor.

2. Global variables and imports

This part of code defies important code snippets to later use them in dedicated functions.

```
#include "SHT3x.h"
#include "DFRobot_BMP280.h"
#include  <ESP8266WiFi.h>
#include "HTTPSRedirect.h"
#include "DebugMacros.h"
#include "PMS.h"

#define SHT3_ADDRESS (0x45)
#define SEA_LEVEL_PRESSURE    1015.0f   // sea level pressure

// Power Switches Pins
uint8_t ThreeVPowerSwitch = D5;
uint8_t FiveVPowerSwitch = D6;

// Struct for collected data
struct GoogleData {
  float temperature;
  float humidity;
  uint32_t pressure;
  uint32_t airQuality_PM_AE_UG_1_0;
  uint32_t airQuality_PM_AE_UG_2_5;
  uint32_t airQuality_PM_AE_UG_10_0;
};
GoogleData collectedData = {404,404,404,404,404,404};


// Sensors globals
typedef DFRobot_BMP280_IIC BMP;
BMP bmp(&Wire, BMP::eSdoLow);
const int httpsPort = 443;
HTTPSRedirect* client = nullptr;

// External configuration
extern const char* ssid;
extern const char* password;
extern const char* host;
extern const char *GScriptId;

// Google writing string
String google_write_url = String("/macros/s/") + GScriptId + "/exec?cal";
```

3. Main function

Due to using DeepSleep mode all functions were used in Arudino's *Setup* function, rather than using them in *loop*, because after DeepSleep NodeMCU is resetting itself and starts program from the beginning. Due to this, no memory is being kept and record must be pushed to external server after the data gathering.

```
void setup() {
  pinMode(ThreeVPowerSwitch, OUTPUT);
  pinMode(FiveVPowerSwitch, OUTPUT);
  TurnOff3VPowerSwitch();
  TurnOff5VPowerSwitch();

  ConnectWiFi();
  if (WiFi.status() == WL_CONNECTED){
      if (ConnectToGoogleHost()){
      CollectData();
      String payload = CreatePayload();
      client->POST(google_write_url, host, payload, false);
    }
  }
  ESP.deepSleep(36e8);
}

void loop() {}
```

4. Managing Relays

As mentioned before, to keep power consumption at minimum, relay module was used. To better describe program behaviour, these functions were written to handle all necessary operations. Used pins were specified in global variables. Also delays were added in case of sensors longer turning on time.

```
void TurnOn5VPowerSwitch(){
  digitalWrite(FiveVPowerSwitch, LOW);
  delay(100);
}

void TurnOff5VPowerSwitch(){
  digitalWrite(FiveVPowerSwitch, HIGH);
}

void TurnOn3VPowerSwitch(){
  digitalWrite(ThreeVPowerSwitch, LOW);
  delay(100);
}

void TurnOff3VPowerSwitch(){
  digitalWrite(ThreeVPowerSwitch, HIGH);
}
```

5. Connecting to Wi-Fi

First step of program way of working is to connect to Wi-Fi. If no internet connections is available, no sensor measurements will be taken in order to save power. Function is written to make few attempts to connect to network. After that one more test is present in main function to fully determine if Wi-Fi is connected or not.

```
void ConnectWiFi(){
  WiFi.begin(ssid, password);
  for (int i = 0; i < 20; ++i){
    if (WiFi.status() != WL_CONNECTED){
      delay(500);
    }
    else{
      break;
    }
  }
}
```

6. Connecting to google host

As mentioned, data is being collected on google servers due to them being free and always available. However connection is not always reached between microcontroller and host, so few attempts are made. If connection was not established microcontroller makes no measurements to save power. To connect to google, first google script must be written and published. Script for weather station is presented in next snippet.

```
var SS = SpreadsheetApp.openById('1lqyVAOhAJ_WA8Mnh_lmnWStzlGGpyXeYKLyWZuVnLfY');

function doPost(e) {
  var parsedData;
  try {
    parsedData = JSON.parse(e.postData.contents);
  }
  catch(f){
    return ContentService.createTextOutput("Error in parsing request body: " + f.m
essage);
  }
  switch (parsedData.command) {
    case "appendRow":
      var current_sheet = SS.getSheetByName(parsedData.sheet_name);
      var dataArr = parsedData.values.split(",");
      var datetime_now = Utilities.formatDate(new Date(), "GMT+2", "yyyy-MM-
dd HH:mm:ss");
      var appendData = [datetime_now, dataArr[0], dataArr[1], dataArr[2], dataArr[
3], dataArr[4], dataArr[5]];
      current_sheet.appendRow(appendData);
      SpreadsheetApp.flush();
      break;
    }
  return ContentService.createTextOutput("End");
}
```

Code to connect to google host from microcontroller:

```
bool ConnectToGoogleHost(){
  client = new HTTPSRedirect(httpsPort);
  client->setInsecure();
  client->setPrintResponseBody(true);
  client->setContentTypeHeader("application/json");

  // Try to connect for a maximum of 5 times
  for (int i=0; i<5; i++){
    int retval = client->connect(host, httpsPort);
    if (retval == 1) {
       return true;
    }
    else
      delay(500);
  }
  return false;
}
```

7. Preparing sensors

All sensors are using prepared libraries can be found on project repository. Before being fully operational, sensors must be prepared and communication must be established. I2C bus is being activated and on Serial interface, after activation with specified baud rate, signal for PMS sensor to enter passive mode is sent to turn off automatic data gathering and sensing to minimize energy usage.

```
void PrepareSensors(){
  Serial.begin(9600);
  Wire.begin();
  delay(100);
  if(!sht3x_softReset(SHT3_ADDRESS)){
     delay(100);
  }

  bmp.reset();
  for (int i = 0; i < 5; ++i){
    if(bmp.begin() != BMP::eStatusOK){
      break;
    }
    delay(100);
  }
  PMS_SetPassiveMode();
}
```

8. Collecting sensors data

After establishing connection between the microcontroller and Wi-Fi and google server, sensors can be activated to collect all available data.

```
void CollectData(){
  TurnOn5VPowerSwitch();
  TurnOn3VPowerSwitch();
  PrepareSensors();

  // Getting temperature and humidity from sensor
  sRHAndTemp_t sht3x_data = sht3x_readTemperatureAndHumidity(SHT3_ADDRESS,
eRepeatability_High);
  collectedData.temperature = sht3x_data.TemperatureC;
  collectedData.humidity = sht3x_data.Humidity;

  // Getting air pressure from sensor
  // float    bmp_temp = bmp.getTemperature();
  uint32_t bmp_press = bmp.getPressure()/100;
  // float    bmp_alti = bmp.calAltitude(SEA_LEVEL_PRESSURE, bmp_press);
  collectedData.pressure = bmp_press;
  TurnOff3VPowerSwitch();

  // Function to get air quality
  //Delay for PMS sensor (30s from datasheet)
  delay(30000);
  ReadPMS();
  Serial.end();
  TurnOff5VPowerSwitch();
}
```

9. Converting collected data to google host

After gathering all data from connected sensors it must be converted to string format with specified prefixes and suffixes.

```
String CreatePayload(){
  String google_payload_prefix = "{\"command\": \"appendRow\", \
                            \"sheet_name\": \"Datalogger\", \
                            \"values\": \"";
  String google_payload_suffix = "\"}";
  String google_payload = "";

  String payload = (google_payload_prefix + collectedData.temperature + "," +
collectedData.humidity + \
                    "," + collectedData.pressure + "," +
collectedData.airQuality_PM_AE_UG_1_0 + "," + \
                    collectedData.airQuality_PM_AE_UG_2_5 + "," +
collectedData.airQuality_PM_AE_UG_10_0 + google_payload_suffix);
  return payload;
}
```

10. Sending gathered data and entering sleep mode

After gathering and converting sensors' data, it is send to google host with HTTP POST request.

```
client->POST(google_write_url, host, payload, false);
```

This request is received by google script described before. After sending data, microcontroller enters deep sleep mode for specified amount of time to save energy.