International Center for Free and Open Source Software

INTEGRATION OF BMP180 SENSOR WITH ULPLoRa

Azwa Harshad

Internship

Open IoT

FEB 2023

Contents

List of Figures

Chapter 1

Introduction

1.1 General Background

In this, integrated the BMP180 sensor with the ULPLORA platform to gather atmospheric pressure and temperature data, which will be visualized and analyzed using ChirpStack, InfluxDB, and a LoRaWAN gateway. The BMP180 sensor offers high accuracy in measuring pressure and temperature, making it suitable for applications such as weather monitoring, environmental sensing, and altitude measurement. By leveraging ULPLORA's capabilities for LoRaWAN communication and ChirpStack's network server infrastructure, we can efficiently transmit sensor data to the cloud, where it will be stored and visualized using InfluxDB.

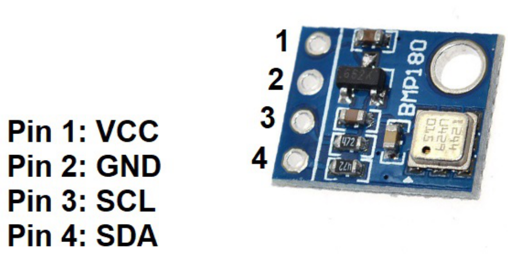

Figure 1.1: DHT11 sensor

1.2 Components Needed

1. ULPLoRa: It is a inhouse board developed by ICFOSS which integrates Arduino Pro Mini and RFM95W LoRaWAN module. The LoRaWAN module is connected to pin number 2, 3, 4, 5, 6 of the Arduino Pro Mini

2. BMP180 Sensor: This is a device that measurs the pressure and temperature. The specific type of sensor used in this system is not specified in the diagram.

   Specifications:

   - Operating Voltage: 1.8V to 3.6V
   - Pressure Measurement Range: 300 hPa to 1100 hPa (9000 m to -500 above sea level) P
   - ressure Measurement Accuracy: ±1 hPa (equivalent to ±8 m)
   - Temperature Measurement Range: 0°C to +65°C
   - Temperature Measurement Accuracy: ±1°C
   - Interface: I2C (Inter-Integrated Circuit)
   - Dimensions: 3.6 mm x 3.8 mm x 0.93 mm Low Power Consumption: Consumption in standard mode is typically 5A
   - Resolution: Pressure: 0.01 hPa, Temperature: 0.1°C
   - Library used:Adafruit BMP085 library

1.3 Softwares Used

1. Arduino IDE: The Arduino IDE (Integrated Development Environment) is a software platform used to write, compile, and upload code to Arduino boards.

2. InfluxDb: InfluxDB is an open-source time-series database developed by InfluxData. It is designed to handle high write and query loads for time-stamped data, making it particularly suitable for use cases involving monitoring, metrics, sensor data, and IoT (Internet of Things) applications.

3. Grafana: Grafana is an open-source analytics and visualization platform designed for monitoring and observability. It allows users to query, visualize, and alert on metrics and data from various sources, including time-series databases like InfluxDB, Prometheus, Graphite, and others.

4. ChirpStack: The ChirpStack LoRaWAN (Long Range Wide Area Network) is an open-source LoRaWAN network server stack, formerly known as LoRaServer.

Chapter 2

Circuit Diagram

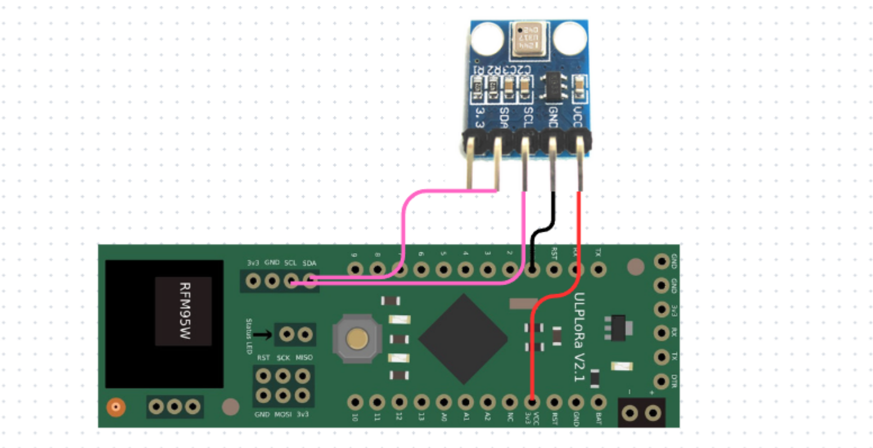

Figure 2.1: Circuit Diagram

2.1 Circuit Connections

- BMP180 Sensor VCC to ULPLoRa VCC
- BMP180 GND to ULPLoRa GND
- BMP180 SDA pin to SDA pin of ULPLoRA
- BMP180 SCL pin to SCL pin of ULPLoRa

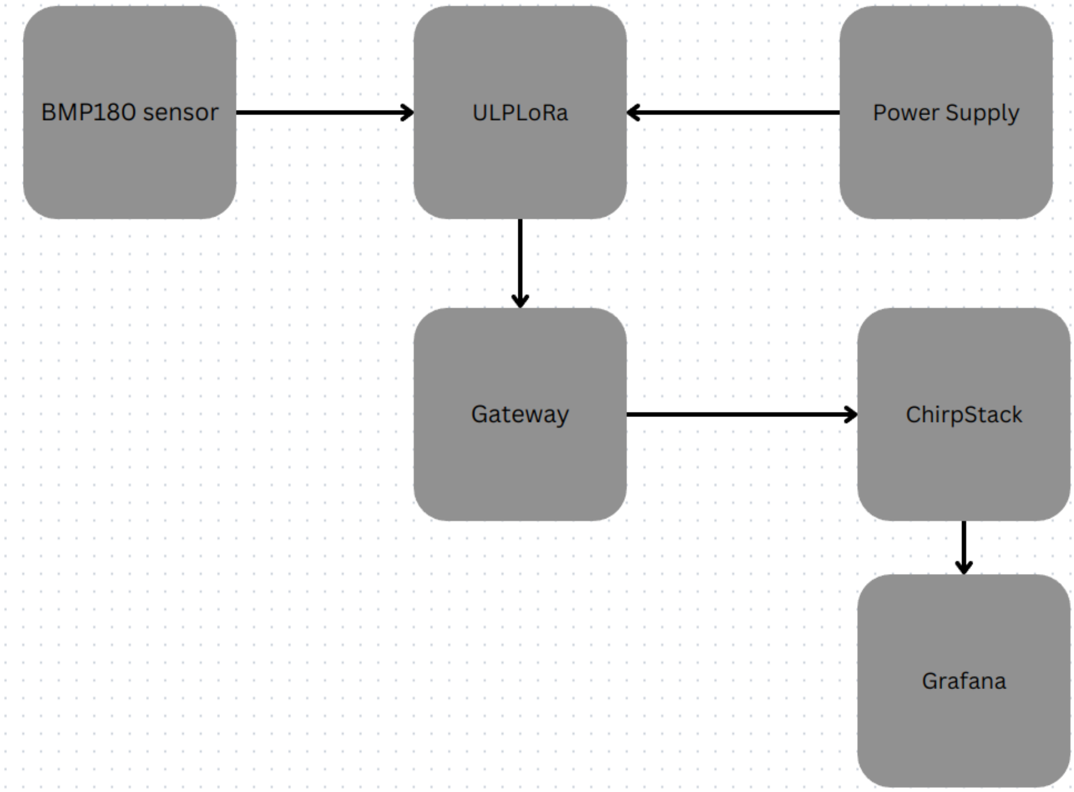

Figure 2.2: Block Diagram

2.2 Block Diagram

- BMP180 sensor: This is a device that measurs the pressure and temperature. The specific type of sensor used in this system is not specified in the diagram.
- Power supply: This block provides power to the entire system. It is likely a battery or a solar panel.
- ULP LoRa: This block refers to the communication protocol used by the system. ULP stands for Ultra Low Power and LoRa stands for Long Range. This combination of technologies allows the sensor to transmit data over long distances while consuming very little power.
- Gateway: This block acts as a bridge between the sensor and the internet. It receives data from the sensor and transmits it to the cloud.
- ChirpStack: This is an open-source platform that can be used to manage and monitor LoRaWAN networks. In this system, ChirpStack is likely used to receive data from the gateway and store it in a database.

- Grafana: This is an open-source platform that can be used to create visualizations of data. In this system, Grafana is likely used to visualize the soil moisture data collected by the sensor.

Chapter 3

Working

The BMP180 sensor operates on the principle of piezoresistive sensing, utilizing a pressure-sensitive element to measure atmospheric pressure and temperature. When exposed to air pressure changes, the internal sensor element deforms, causing a corresponding change in the resistance of the piezoresistive material. This change in resistance is converted into an electrical signal, which is then amplified and processed by the sensor's internal circuitry. By measuring the resistance changes, the BMP180 can accurately determine the atmospheric pressure, which can be further calibrated to provide altitude measurements. Additionally, the sensor incorporates a temperature sensor to compensate for temperature variations, ensuring accurate pressure readings under different environmental conditions. Through precise pressure and temperature measurements, the BMP180 sensor facilitates various applications, including weather forecasting, altitude sensing, and indoor navigation.

3.1 Procedure

1. HardWare Setup:
   (a) Gather all necessary components.
   (b) Connect them according to the circuit diagram.
2. Arduino IDE:

(a) Install Arduino IDE: Download and install Arduino IDE from the official website.

(b) Open Arduino IDE: Launch the Arduino IDE software.

(c) Write Code: Compose your program using Arduino programming language (based on C/C++). Write setup and loop functions.

(d) Installlibrary: Tools-Managelibraries-MCCI LMIC LoRaWAN library-Install

(e) Verify Code: Click on the Verify button (checkmark icon) to check for any errors in the code

(f) Upload Code: Connect Arduino board to the computer via USB. Select the correct board and port from Tools menu. Click on the Upload button (right arrow icon) to upload the code to the Arduino board.

(g) Test: Make sure the hardware is powered on.

3.2 ChirpStack

(a) Link: lorawandev.icfoss.org

(b) Login to chirpstack as in Figure 3.1

(c) Create Device-profile as in Figure 3.2: Device-profiles-Create-Add details-Update device profile

(d) Create Applications as in Figure 3.3: Application-Create-Fill application details-Create application-Update device details

(e) Update Codec as in Figure 3.4: Device-profiles-Select your profile-Codec-Write codec in Java script-Update device-profiles

(f) Copy keys as in Figure 3.5: Application-Select application name-Activation-copy device address and keys as Hexarray-Reactivate device¿¿paste in arduino sketch

(g) Check Device data as in Figure 3.6: Application-Select application-Device data

3.3 InfluxDB

(a) Integrate Influxdb with chirpstack as in Figure 3.7: Application-Select application-Integration-Influxdb-Enter details(generate token in influxdb)-Update integration

(b) Link:117.223.185.200.8086

(c) Login to influxdb as in Figure 3.8

(d) Generate token as in Figure 3.9: Generate API token-custom API token¿¿buckets-select bucket¿¿enable read,write-copy the generated token-paste in chirpstack

(e) Select bucket-application name-select the data for which you need graph-submit as in Figure 3.10

(f) Script editor-copy query-paste it in grafana for visualization as in Figure 3.11

3.4 Visualization

(a) Link:visualizadev.icfoss.org

(b) Login to grafana

(c) New dashboard-add a new panel-select data source-paste the query -apply

(d) Panel title-edit-choose the type of visualization-eg:stat

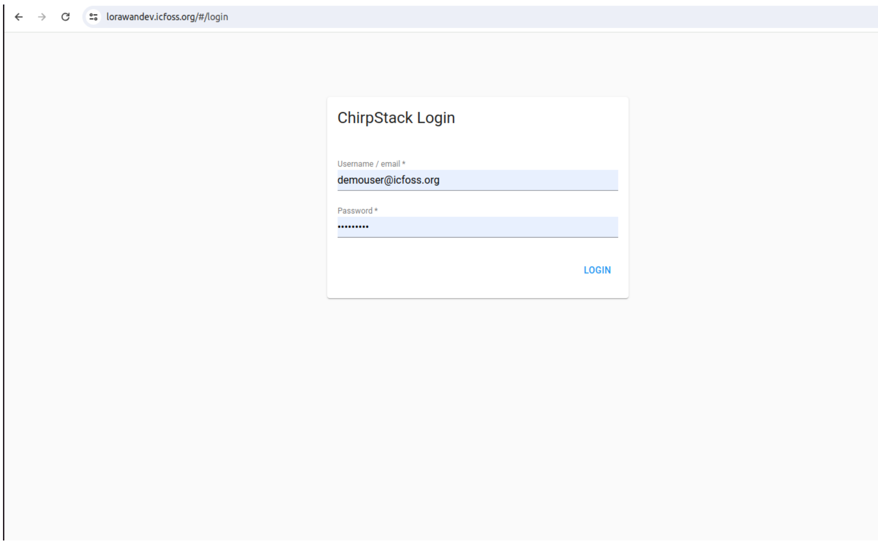

Figure 3.1: Login details

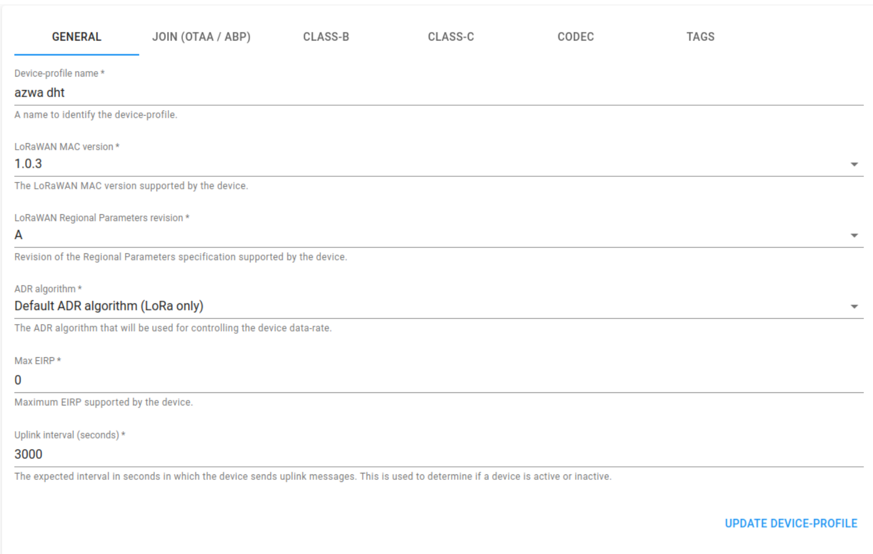

Figure 3.2: Device profile

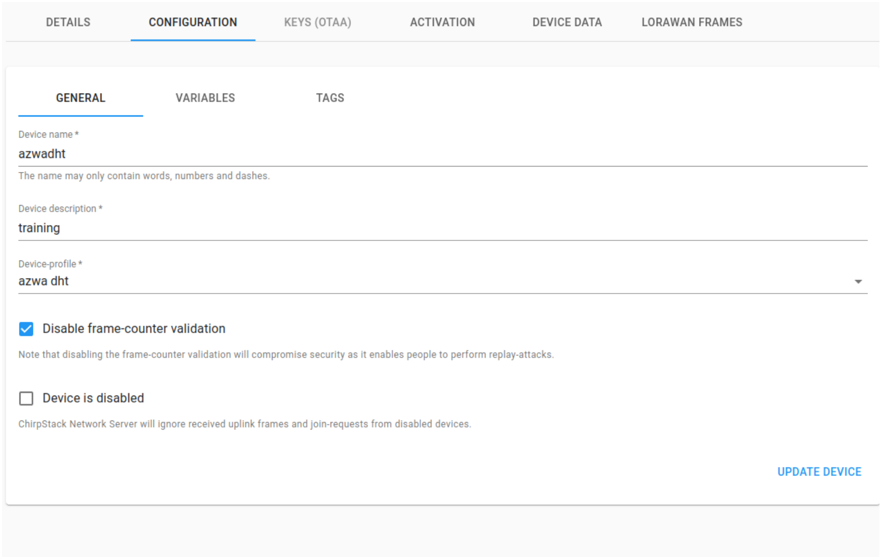

Figure 3.3: Create application

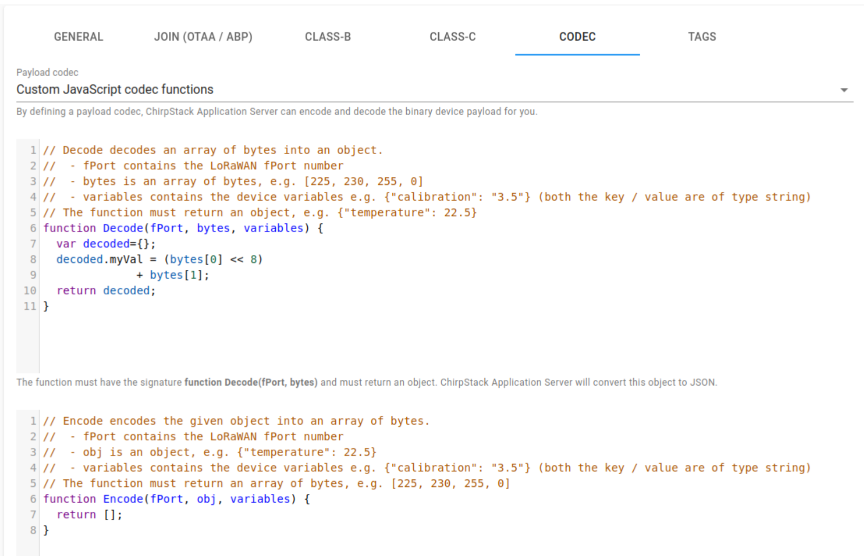

Figure 3.4: Codec

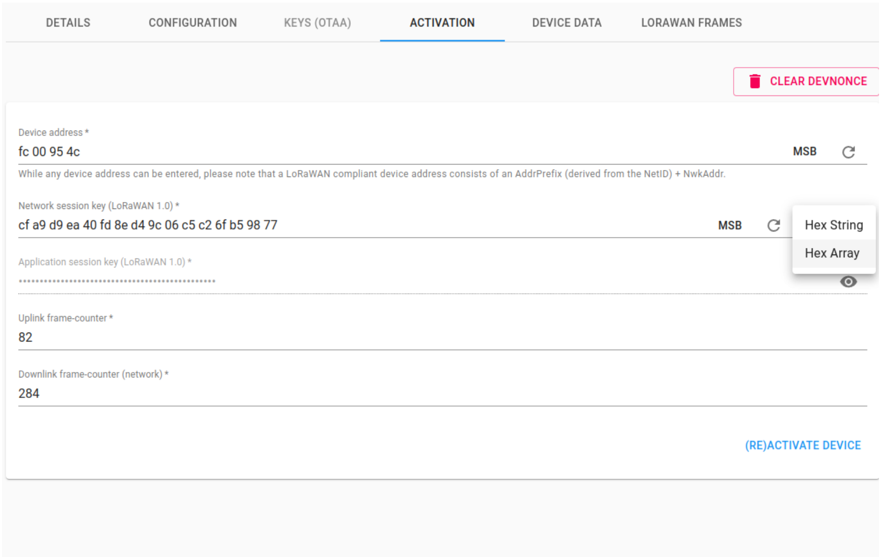

Figure 3.5: Keys

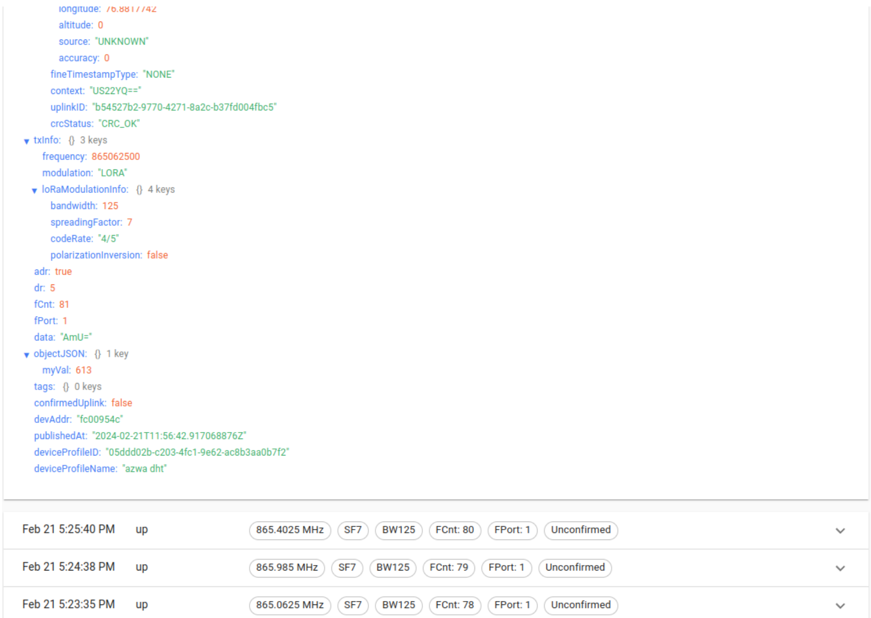

Figure 3.6: Device data

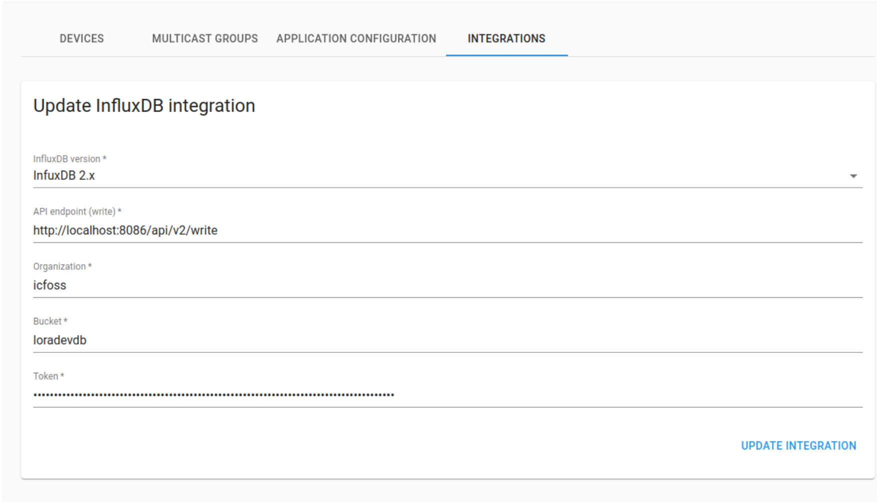

Figure 3.7: Login

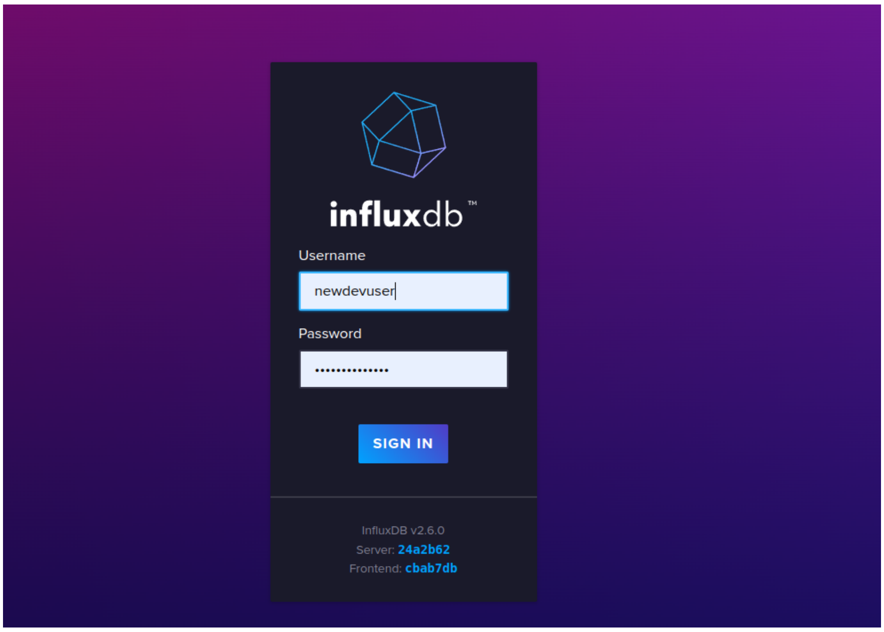

Figure 3.8: Generate token

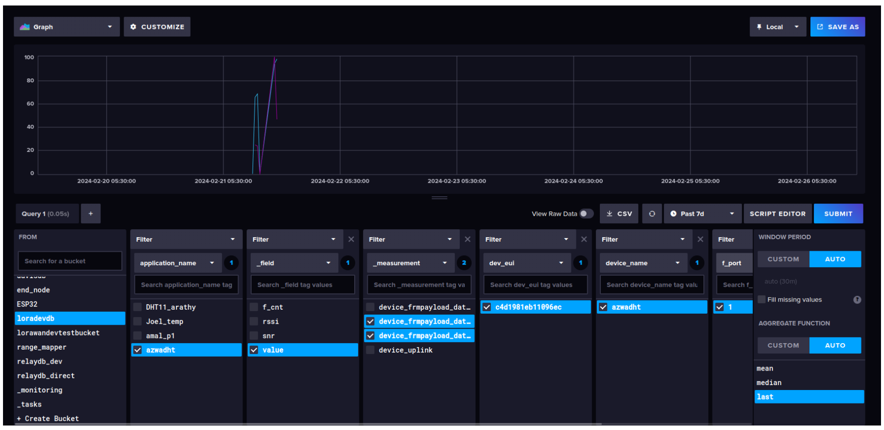

Figure 3.9: Select Bucket

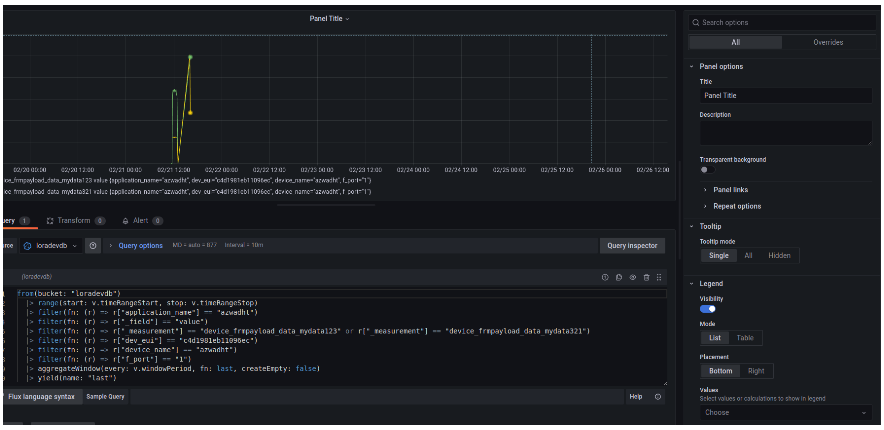

Figure 3.10: Obtain graph

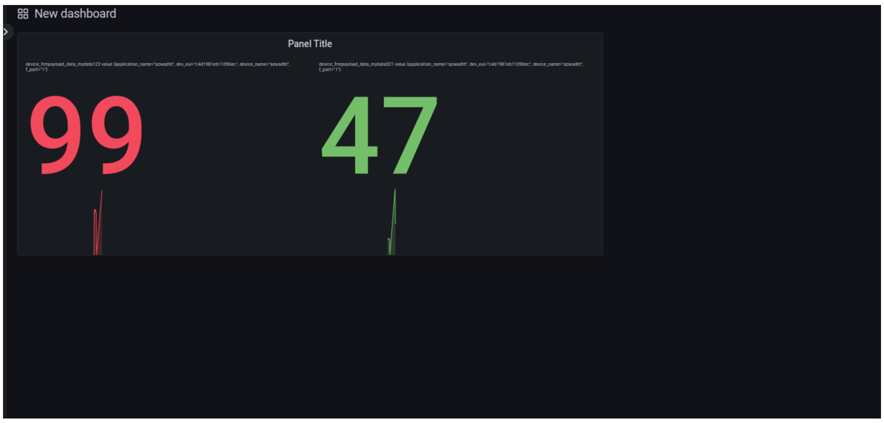

Figure 3.11: Obtain stat

Chapter 4

Code

4.1 Code for the ULPLoRa Board

It is done through ArduinoIDE

The library is used for LoRaWAN communication

Code

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <Adafruit_BMP085.h>

byte val[3];
int pressure;
int temperature;

//
// For normal use, we require that you edit the sketch to replace FILLMEIN
// with values assigned by the TTN console. However, for regression tests,
// we want to be able to compile these scripts. The regression tests define
// COMPILE_REGRESSION_TEST, and in that case we define FILLMEIN to a non-
```

```
// working but innocuous value.
//
#ifdef COMPILE_REGRESSION_TEST
# define CFG_in866 1
#else
# warning "You must replace the values marked FILLMEIN with real values fro
# define FILLMEIN (#dont edit this, edit the lines that use FILLMEIN)
#endif

// LoRaWAN NwkSKey, network session key
// This should be in big-endian (aka msb).
static const PROGMEM u1_t NWKSKEY[16] = { 0x1A, 0xA6, 0xB1, 0x91, 0x5C, 0x

// LoRaWAN AppSKey, application session key
// This should also be in big-endian (aka msb).
static const u1_t PROGMEM APPSKEY[16] = { 0x12, 0x0D, 0x85, 0x72, 0xE4, 0x

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
// The library converts the address to network byte order as needed, so th
static const u4_t DEVADDR = 0xfc0095b4 ; // <-- Change this address for ev

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmic/project_config/lmic_project_config.
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

//static uint8_t mydata[];
static osjob_t sendjob;
```

```
int BMP180_ADDRESS = 0x77; // BMP180 I2C address
Adafruit_BMP085 bmp;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 60;

// Pin mapping
// Adapted for Feather M0 per p.10 of [feather]
const lmic_pinmap lmic_pins = {
    .nss = 6,                       // chip select on feather (rf95module)
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 5,                       // reset pin
    .dio = {2, 3, 4}, // assumes external jumpers [feather_lora_jumper]
                                    // DIO1 is on JP1-1: is io1 - we connec
                                    // DIO1 is on JP5-3: is D2 - we connect
};

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
```

```
    Serial.println(F("EV_BEACON_TRACKED"));
    break;
case EV_JOINING:
    Serial.println(F("EV_JOINING"));
    break;
case EV_JOINED:
    Serial.println(F("EV_JOINED"));
    break;
/*
|| This event is defined but not used in the code. No
|| point in wasting codespace on it.
||
|| case EV_RFU1:
||     Serial.println(F("EV_RFU1"));
||     break;
*/
case EV_JOIN_FAILED:
    Serial.println(F("EV_JOIN_FAILED"));
    break;
case EV_REJOIN_FAILED:
    Serial.println(F("EV_REJOIN_FAILED"));
    break;
case EV_TXCOMPLETE:
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windo
    if (LMIC.txrxFlags & TXRX_ACK)
      Serial.println(F("Received ack"));
    if (LMIC.dataLen) {
      Serial.println(F("Received "));
      Serial.println(LMIC.dataLen);
      Serial.println(F(" bytes of payload"));
    }
    // Schedule next transmission
```

```
        os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTE
        break;
    case EV_LOST_TSYNC:
        Serial.println(F("EV_LOST_TSYNC"));
        break;
    case EV_RESET:
        Serial.println(F("EV_RESET"));
        break;
    case EV_RXCOMPLETE:
        // data received in ping slot
        Serial.println(F("EV_RXCOMPLETE"));
        break;
    case EV_LINK_DEAD:
        Serial.println(F("EV_LINK_DEAD"));
        break;
    case EV_LINK_ALIVE:
        Serial.println(F("EV_LINK_ALIVE"));
        break;
    /*
    || This event is defined but not used in the code. No
    || point in wasting codespace on it.
    ||
    || case EV_SCAN_FOUND:
    ||    Serial.println(F("EV_SCAN_FOUND"));
    ||    break;
    */
    case EV_TXSTART:
        Serial.println(F("EV_TXSTART"));
        break;
    case EV_TXCANCELED:
        Serial.println(F("EV_TXCANCELED"));
        break;
```

```
        case EV_RXSTART:
            /* do not print anything -- it wrecks timing */
            break;
        case EV_JOIN_TXCOMPLETE:
            Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
            break;
        default:
            Serial.print(F("Unknown event: "));
            Serial.println((unsigned) ev);
            break;
    }
}

void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
             int temperature = bmp.readTemperature();
        int pressure = bmp.readPressure() / 100;
        // Pressure in hPa




        Serial.println(temperature);




        Serial.println(pressure);
        send();
        // Prepare upstream data transmission at the next possible time.
```

```
        LMIC_setTxData2(1, val, sizeof(val), 0);
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
//    pinMode(13, OUTPUT);
    while (!Serial); // wait for Serial to be initialized
    Serial.begin(115200);
    delay(100);     // per sample code on RF_95 test
    Serial.println(F("Starting"));

    if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP085 sensor, check wiring!");
    }

    #ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be disc
    LMIC_reset();

    // Set static session parameters. Instead of dynamically establishing a
```

```
    // by joining the network, precomputed session parameters are be provi
    #ifdef PROGMEM
    // On AVR, these values are stored in flash and only copied to RAM
    // once. Copy them to a temporary buffer here, LMIC_setSession will
    // copy them into a buffer of its own again.
    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x13, DEVADDR, nwkskey, appskey);
    #else
    // If not running an AVR with PROGMEM, just use the arrays directly
    LMIC_setSession (0x13, DEVADDR, NWKSKEY, APPSKEY);
    #endif

    #if defined(CFG_eu868)
    // Set up the channels used by the Things Network, which corresponds
    // to the defaults of most gateways. Without this, only three base
    // channels from the LoRaWAN specification are used, which certainly
    // works, so it is good for debugging, but can overload those
    // frequencies, so be sure to configure the full frequency range of
    // your network here (unless your network autoconfigures them).
    // Setting up channels should happen after LMIC_setSession, as that
    // configures the minimal channel set. The LMIC doesn't let you change
    // the three basic settings, but we show them here.
//    LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
//    LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_
//    LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
//    LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
//    LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
//    LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
//    LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
```

```
//    LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_
//    LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK),  BAND_
//    // TTN defines an additional channel at 869.525Mhz using SF9 for clas
    // devices' ping slots. LMIC does not have an easy way to define set th
    // frequency and support for class B is spotty and untested, so this
    // frequency is not configured here.
    #elif defined(CFG_us915) || defined(CFG_au915)
    // NA-US and AU channels 0-71 are configured automatically
    // but only one group of 8 should (a subband) should be active
    // TTN recommends the second sub band, 1 in a zero based count.
    // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-glob
    LMIC_selectSubBand(1);
    #elif defined(CFG_as923)
    // Set up the channels used in your country. Only two are defined by de
    // and they cannot be changed.  Use BAND_CENTI to indicate 1% duty cycl
    // LMIC_setupChannel(0, 923200000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BANI
    // LMIC_setupChannel(1, 923400000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BANI

    // ... extra definitions for channels 2..n here
    #elif defined(CFG_kr920)
    // Set up the channels used in your country. Three are defined by defau
    // and they cannot be changed. Duty cycle doesn't matter, but is conver
    // BAND_MILLI.
    // LMIC_setupChannel(0, 922100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BANI
    // LMIC_setupChannel(1, 922300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BANI
    // LMIC_setupChannel(2, 922500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BANI

    // ... extra definitions for channels 3..n here.
    #elif defined(CFG_in866)
    // Set up the channels used in your country. Three are defined by defau
    // and they cannot be changed. Duty cycle doesn't matter, but is conver
    // BAND_MILLI.
```

```
     LMIC_setupChannel(0, 865062500, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_I
     LMIC_setupChannel(1, 865402500, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_I
     LMIC_setupChannel(2, 865985000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_I

    // ... extra definitions for channels 3..n here.
    #else
    # error Region not supported
    #endif

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power for uplink
    LMIC_setDrTxpow(DR_SF7,14);

    // Start job
    do_send(&sendjob);
}

void loop() {



    os_runloop_once();

}
void send() {



  val[0]=highByte(pressure);
```

```
  val[1]=lowByte(pressure);

  //val[2]=byte(temperature);
}
```

Chapter 5

Result

5.1 Result

Integrated BMP180 sensor with ULPLoRa communication, As a result, the system can collect environmental data and transmit it wirelessly to a central hub or monitoring station, enabling applications such as weather monitoring, precision agriculture, environmental conservation, and industrial automation.