

**International Center for Free and Open Source  
Software**



**Integration of LDR Sensor Using  
ULP LoRa**

**Azwa Harshad  
Internship  
Open IoT  
MARCH 2023**

# Contents

|  |           |
|--|-----------|
| <b>List of Figures</b>                   | <b>ii</b> |
| <b>1 Introduction</b>                    | <b>1</b>  |
| 1.1 General Background . . . . .         | 1         |
| 1.2 Components Needed . . . . .          | 2         |
| 1.3 Softwares Used . . . . .             | 3         |
| <b>2 Circuit Diagram</b>                 | <b>4</b>  |
| 2.1 Circuit Connections . . . . .        | 5         |
| 2.2 Block Diagram . . . . .              | 5         |
| <b>3 Working</b>                         | <b>7</b>  |
| 3.1 Procedure . . . . .                  | 7         |
| 3.2 ChirpStack . . . . .                 | 8         |
| 3.3 InfluxDB . . . . .                   | 9         |
| 3.4 Visualization . . . . .              | 9         |
| <b>4 Code</b>                            | <b>15</b> |
| 4.1 Code for the ULPLoRa Board . . . . . | 15        |
| <b>5 Result</b>                          | <b>25</b> |
| 5.1 Result . . . . .                     | 25        |

# List of Figures

|      |                              |    |
|------|------------------------------|----|
| 1.1  | LDR Sensor . . . . .         | 1  |
| 2.1  | Circuit Diagram . . . . .    | 4  |
| 2.2  | Block Diagram . . . . .      | 5  |
| 3.1  | Login details . . . . .      | 10 |
| 3.2  | Device profile . . . . .     | 10 |
| 3.3  | Create application . . . . . | 11 |
| 3.4  | Codec . . . . .              | 11 |
| 3.5  | Keys . . . . .               | 12 |
| 3.6  | Device data . . . . .        | 12 |
| 3.7  | Login . . . . .              | 13 |
| 3.8  | Generate token . . . . .     | 13 |
| 3.9  | Select Bucket . . . . .      | 14 |
| 3.10 | Obtain graph . . . . .       | 14 |
| 3.11 | Obtain stat . . . . .        | 14 |

# Chapter 1

## Introduction

### 1.1 General Background

The integration of LDR sensors with ULPLoRa modules and leveraging the robust infrastructure of ChirpStack, InfluxDB, and Grafana, we embark on a journey to illuminate the depths of real-time light data collection and analysis. Through this endeavor, we delve into the realms of IoT-enabled environmental monitoring, offering a glimpse into the profound insights that lie within the fluctuations of ambient light levels. This project report documents our exploration, showcasing the synergy between technology and environmental awareness as we pave the path towards a brighter, more data-driven future.

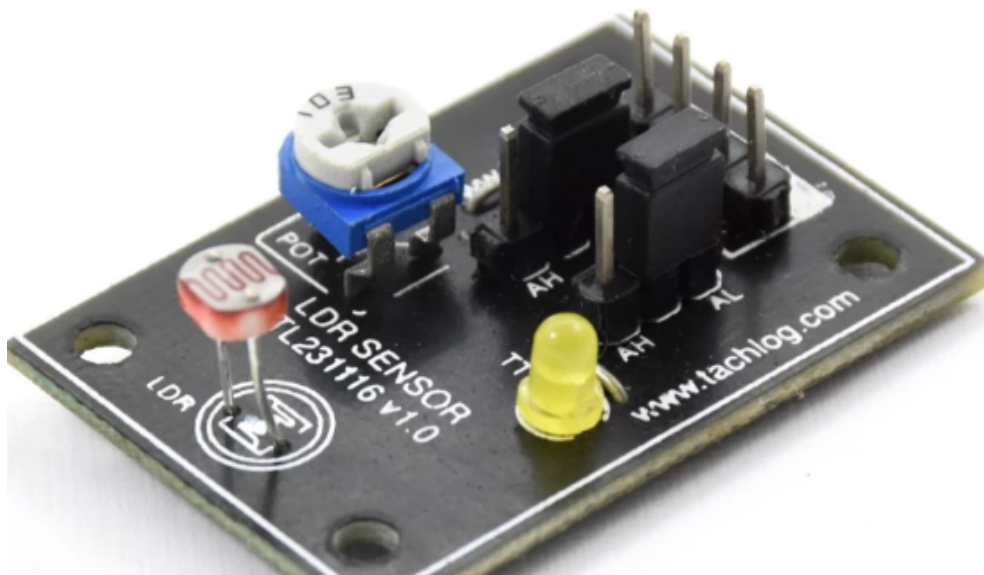


Figure 1.1: LDR Sensor

## 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. LDR Sensor: The Light Dependent Resistor (LDR), also known as a photoresistor, is a passive electronic component that exhibits a change in resistance based on the intensity of incident light. As light levels fluctuate, the resistance of the LDR varies inversely, making it a valuable tool for detecting changes in ambient light conditions.

Specifications:

- Operating Principle: A passive component whose resistance changes with incident light intensity.
- Material: Typically made of semiconductor materials such as cadmium sulfide (CdS) or lead sulfide (PbS).
- Resistance Range: Can vary from several kilohms in darkness to a few hundred ohms in bright light.
- Response Time: Generally quick, with changes in resistance occurring almost instantaneously with changes in light intensity.
- Wavelength Sensitivity: Responsive to a wide spectrum of light wavelengths, depending on the specific material used.
- Temperature Dependence: The resistance can be affected by temperature changes, though less significantly compared to other types of resistors.
- Size: Compact and lightweight, making them suitable for integration into small electronic devices.
- Cost: Relatively inexpensive, making them cost-effective for various applications. Operating Voltage: Typically compatible with low-voltage circuits, suitable for integration into battery-powered devices.

## 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

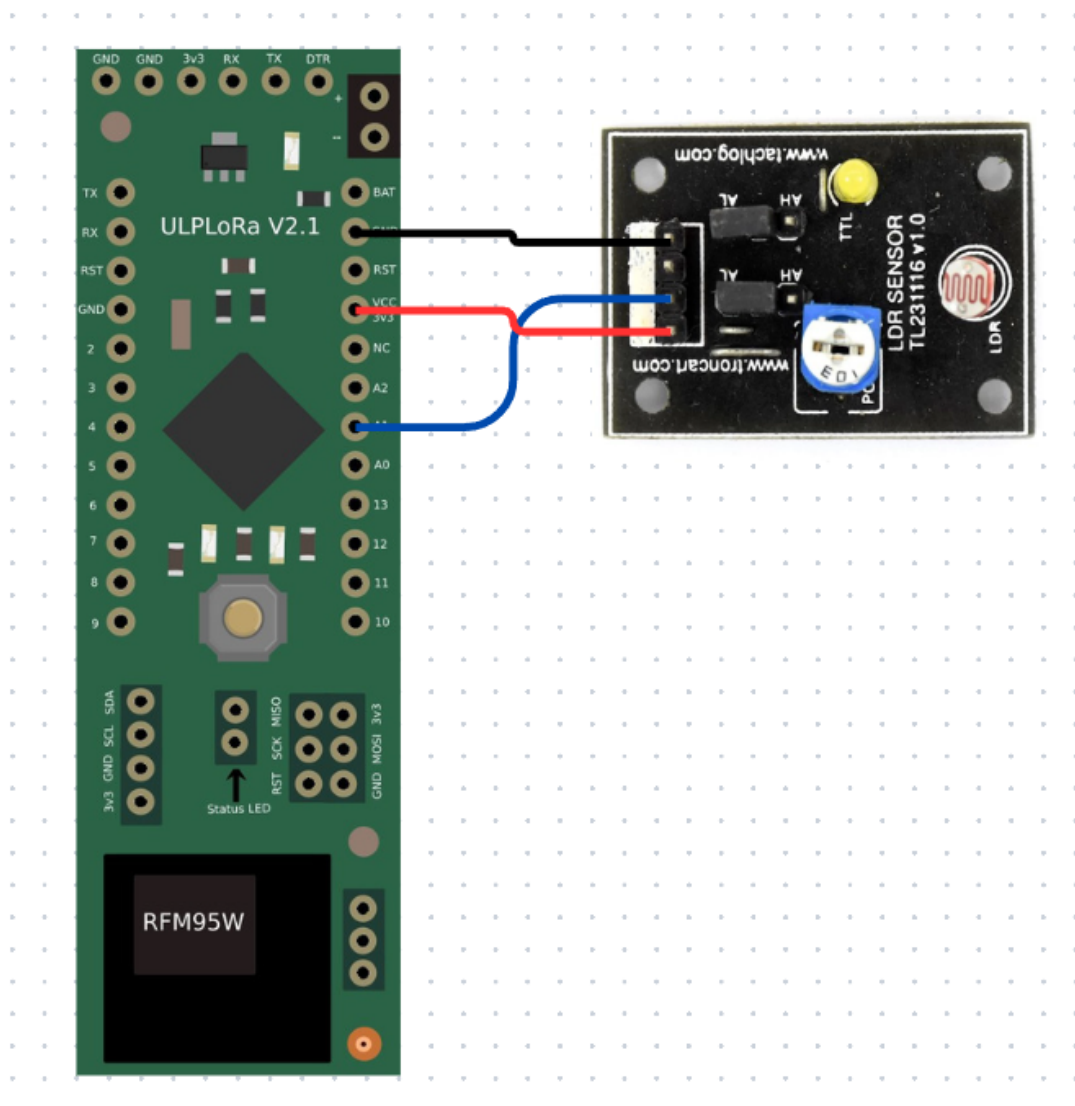


Figure 2.1: Circuit Diagram

## 2.1 Circuit Connections

- Connect one end of the LDR (Light Dependent Resistor) to the positive voltage supply (Vcc).
- Connect the other end of the LDR to A1 pin of board.
- Connect the other terminal of the resistor to the ground (GND) or the negative terminal of the power supply.

## 2.2 Block Diagram

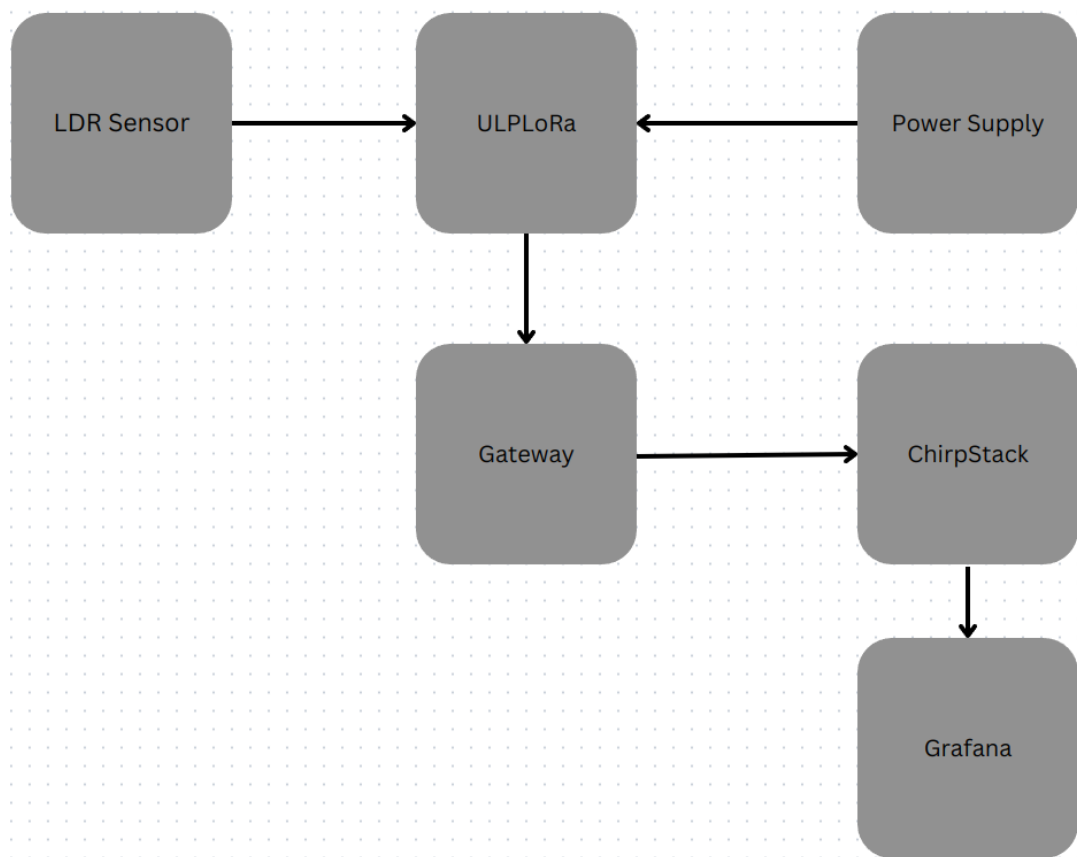


Figure 2.2: Block Diagram

- LDR Sensor: The Light Dependent Resistor (LDR), also known as a photoresistor, is a passive electronic component that exhibits a change in resistance based on the intensity of incident light. As light levels fluctuate, the resistance of the



LDR varies inversely, making it a valuable tool for detecting changes in ambient light conditions.

- 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 working principle of the LDR sensor circuit is based on the characteristic behavior of the Light Dependent Resistor. As ambient light levels change, the resistance of the LDR varies inversely, leading to a corresponding change in the voltage across the fixed resistor connected in series. This voltage variation is then detected at the output terminal of the circuit, typically connected to an analog pin of a microcontroller. By measuring this voltage using the microcontroller's analog-to-digital converter (ADC), the intensity of light falling on the LDR can be accurately determined. Thus, the circuit effectively converts changes in light intensity into measurable electrical signals, allowing for real-time monitoring and control in various applications such as automatic lighting systems, environmental monitoring, and smart agriculture.

### 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) Install library: Tools-Manager-libraries-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](http://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

ChirpStack Login

Username / email \*

demouser@icfoss.org

Password \*

\*\*\*\*\*

LOGIN

Figure 3.1: Login details

GENERAL JOIN (OTAA / ABP) CLASS-B CLASS-C CODEC TAGS

Device-profile name \*

azwa dht

A name to identify the device-profile.

LoRaWAN MAC version \*

1.0.3

The LoRaWAN MAC version supported by the device.

LoRaWAN Regional Parameters revision \*

A

Revision of the Regional Parameters specification supported by the device.

ADR algorithm \*

Default ADR algorithm (LoRa only)

The ADR algorithm that will be used for controlling the device data-rate.

Max EIRP \*

0

Maximum EIRP supported by the device.

Uplink interval (seconds) \*

3000

The expected interval in seconds in which the device sends uplink messages. This is used to determine if a device is active or inactive.

UPDATE DEVICE-PROFILE

Figure 3.2: Device profile

| DETAILS  | CONFIGURATION | KEYS (OTAA) | ACTIVATION | DEVICE DATA | LORAWAN FRAMES |
|--|---------------|-------------|------------|-------------|----------------|
| <div> <div>GENERAL</div> <div>VARIABLES</div> <div>TAGS</div> </div>   |               |             |            |             |                |
| <div>Device name *</div> <div>azwadht</div> <div>The name may only contain words, numbers and dashes.</div>  |               |             |            |             |                |
| <div>Device description *</div> <div>training</div>  |               |             |            |             |                |
| <div>Device-profile *</div> <div>azwa dht</div> <div></div>  |               |             |            |             |                |
| <div><input checked="" type="checkbox"/> Disable frame-counter validation</div> <div>Note that disabling the frame-counter validation will compromise security as it enables people to perform replay-attacks.</div> |               |             |            |             |                |
| <div><input type="checkbox"/> Device is disabled</div> <div>ChirpStack Network Server will ignore received uplink frames and join-requests from disabled devices.</div>  |               |             |            |             |                |
| <div>UPDATE DEVICE</div>   |               |             |            |             |                |

Figure 3.3: Create application

| GENERAL  | JOIN (OTAA / ABP) | CLASS-B | CLASS-C | CODEC | TAGS |
|--|-------------------|---------|---------|-------|------|
| <div>Payload codec</div> <div>Custom JavaScript codec functions</div> <div>By defining a payload codec, ChirpStack Application Server can encode and decode the binary device payload for you.</div>   |                   |         |         |       |      |
| <pre> 1 // Decode decodes an array of bytes into an object. 2 // - fPort contains the LoRaWAN fPort number 3 // - bytes is an array of bytes, e.g. [225, 230, 255, 0] 4 // - variables contains the device variables e.g. {"calibration": "3.5"} (both the key / value are of type string) 5 // The function must return an object, e.g. {"temperature": 22.5} 6 function Decode(fPort, bytes, variables) { 7   var decoded={}; 8   decoded.myVal = (bytes[0] &lt;&lt; 8) 9     + bytes[1]; 10   return decoded; 11 } </pre> |                   |         |         |       |      |
| <div>The function must have the signature <code>function Decode(fPort, bytes)</code> and must return an object. ChirpStack Application Server will convert this object to JSON.</div>  |                   |         |         |       |      |
| <pre> 1 // Encode encodes the given object into an array of bytes. 2 // - fPort contains the LoRaWAN fPort number 3 // - obj is an object, e.g. {"temperature": 22.5} 4 // - variables contains the device variables e.g. {"calibration": "3.5"} (both the key / value are of type string) 5 // The function must return an array of bytes, e.g. [225, 230, 255, 0] 6 function Encode(fPort, obj, variables) { 7   return []; 8 } </pre>   |                   |         |         |       |      |

Figure 3.4: Codec

DETAILS

CONFIGURATION

KEYS (OTAA)

ACTIVATION

DEVICE DATA

LORAWAN FRAMES

CLEAR DEVNONCE

Device address \*

fc 00 95 4c

MSB

While any device address can be entered, please note that a LoRaWAN compliant device address consists of an AddrPrefix (derived from the NetID) + NwkAddr.

Network session key (LoRaWAN 1.0) \*

cf a9 d9 ea 40 fd 8e d4 9c 06 c5 c2 6f b5 98 77

MSB

Hex String

Hex Array

Application session key (LoRaWAN 1.0) \*

\*\*\*\*\*

Uplink frame-counter \*

82

Downlink frame-counter (network) \*

284

(RE)ACTIVATE DEVICE

Figure 3.5: Keys

longitude: 76.8611742

altitude: 0

source: "UNKNOWN"

accuracy: 0

fineTimestampType: "NONE"

context: "US22YQ=="

uplinkID: "b54527b2-9770-4271-8a2c-b37fd004fbc5"

crcStatus: "CRC\_OK"

txinfo: {} 3 keys

frequency: 865062500

modulation: "LORA"

loRaModulationInfo: {} 4 keys

bandwidth: 125

spreadingFactor: 7

codeRate: "4/5"

polarizationInversion: false

adr: true

dr: 5

fCnt: 81

fPort: 1

data: "AmJ="

objectJSON: {} 1 key

myVal: 613

tags: {} 0 keys

confirmedUplink: false

devAddr: "fc00954c"

publishedAt: "2024-02-21T11:56:42.917068876Z"

deviceProfileID: "05ddd02b-c203-4fc1-9e62-ac8b3aa0b7f2"

deviceProfileName: "azwa dht"

Feb 21 5:25:40 PM

up

865.4025 MHz

SF7

BW125

FCnt: 80

FPort: 1

Unconfirmed

Feb 21 5:24:38 PM

up

865.985 MHz

SF7

BW125

FCnt: 79

FPort: 1

Unconfirmed

Feb 21 5:23:35 PM

up

865.0625 MHz

SF7

BW125

FCnt: 78

FPort: 1

Unconfirmed

Figure 3.6: Device data

DEVICES   MULTICAST GROUPS   APPLICATION CONFIGURATION   **INTEGRATIONS**

---

### Update InfluxDB integration

InfluxDB version \*  
**InfluxDB 2.x**

---

API endpoint (write) \*  
**http://localhost:8086/api/v2/write**

---

Organization \*  
**icfoss**

---

Bucket \*  
**loradevdb**

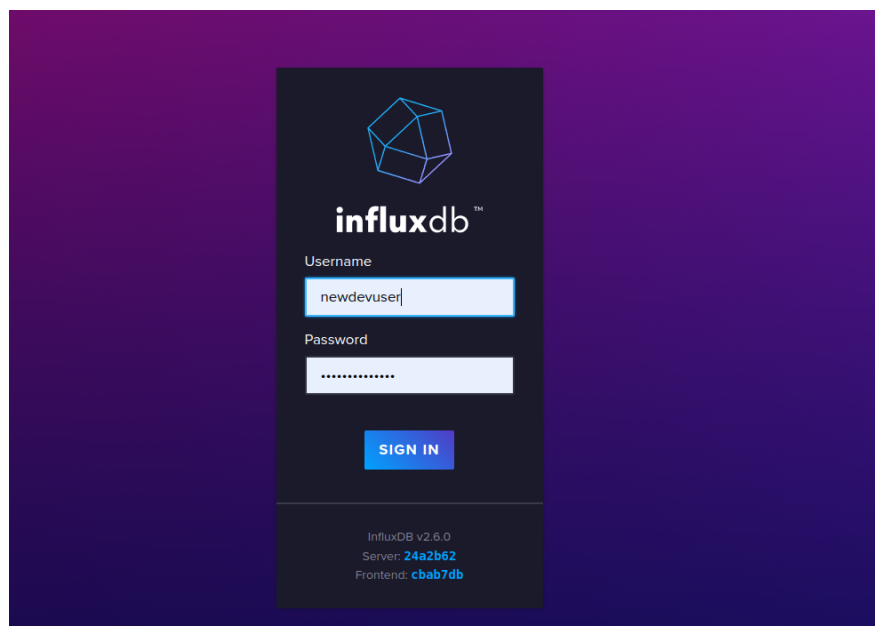
---

Token \*

.....

UPDATE INTEGRATION

Figure 3.7: Login



The image shows the InfluxDB login interface. It features a dark purple background with a central dark grey panel. At the top of the panel is the InfluxDB logo, a white wireframe cube. Below the logo is the text 'influxdb™'. Underneath are two input fields: 'Username' with the text 'newdevuser' and 'Password' with masked characters. A blue 'SIGN IN' button is positioned below the password field. At the bottom of the panel, it displays 'InfluxDB v2.6.0', 'Server: 24a2b62', and 'Frontend: cbab7db'.

influxdb™

Username  
newdevuser

Password  
.....

**SIGN IN**

InfluxDB v2.6.0  
Server: 24a2b62  
Frontend: cbab7db

Figure 3.8: Generate token



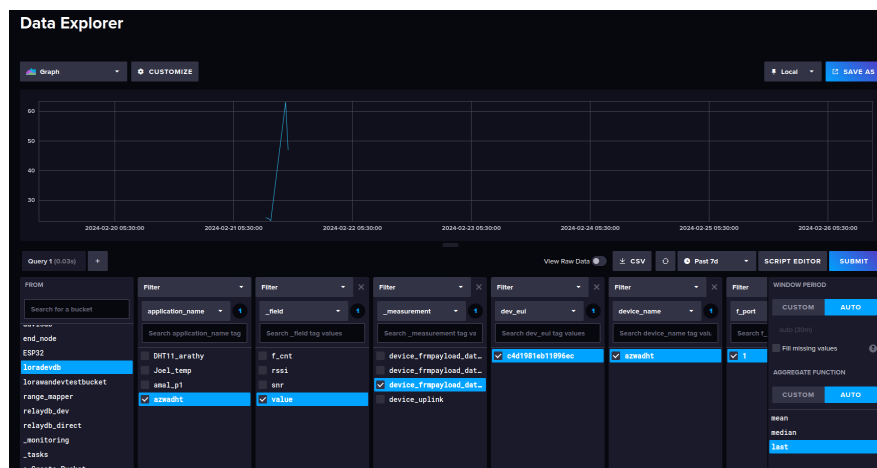


Figure 3.9: Select Bucket

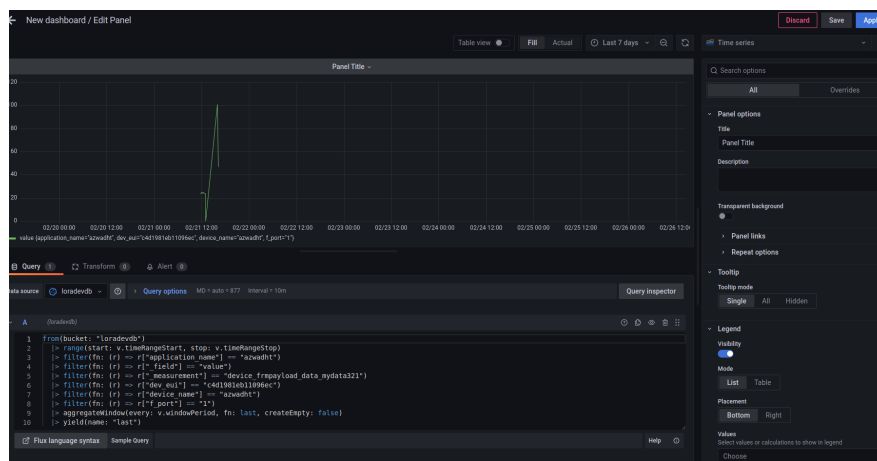


Figure 3.10: Obtain graph

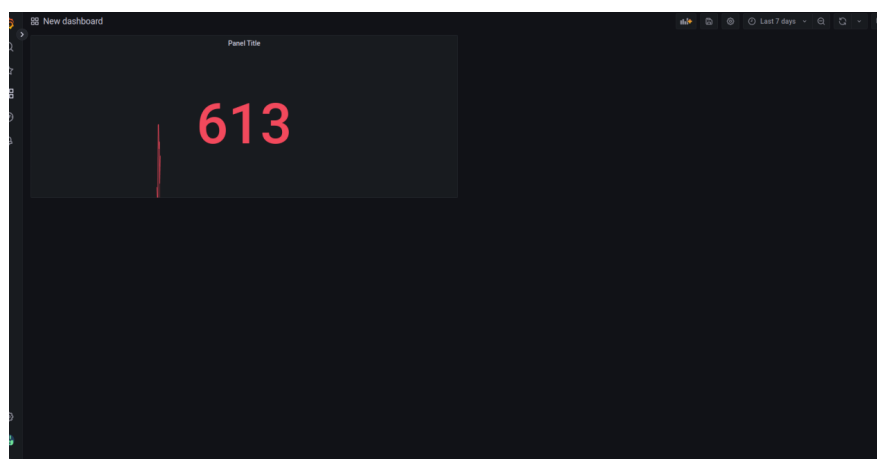


Figure 3.11: Obtain stat

# 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>

//
// 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 fr
# 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, 0x1

// 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.h
// 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;

// 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 connect
                                // 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 window)"));
        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_INTERVAL), 1);
        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 {
        // Prepare upstream data transmission at the next possible time.
        int ldrPin = A1;
        int ldrValue = analogRead(ldrPin);
        Serial.println(ldrValue);
        byte ldr[2];
        ldr[0]=highByte(ldrValue);
        ldr[1]=lowByte(ldrValue);
        LMIC_setTxData2(1, ldr, sizeof(ldr), 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"));

#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 discarded.
LMIC_reset();

// Set static session parameters. Instead of dynamically establishing a session
// by joining the network, precomputed session parameters are be provided here.
#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-global.conf
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 cyc
// LMIC_setupChannel(0, 923200000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND
// LMIC_setupChannel(1, 923400000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND

// ... extra definitions for channels 2..n here
#elif defined(CFG_kr920)

```

```

// Set up the channels used in your country. Three are defined by default
// and they cannot be changed. Duty cycle doesn't matter, but is converted
// BAND_MILLI.
// LMIC_setupChannel(0, 922100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_MILLI);
// LMIC_setupChannel(1, 922300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_MILLI);
// LMIC_setupChannel(2, 922500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_MILLI);

// ... extra definitions for channels 3..n here.
#elif defined(CFG_in866)
// Set up the channels used in your country. Three are defined by default
// and they cannot be changed. Duty cycle doesn't matter, but is converted
// BAND_MILLI.
    LMIC_setupChannel(0, 865062500, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_MILLI);
    LMIC_setupChannel(1, 865402500, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_MILLI);
    LMIC_setupChannel(2, 865985000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_MILLI);

// ... 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();  
  
}
```

# Chapter 5

## Result

### 5.1 Result

The implementation of the LDR sensor circuit within the broader project framework yielded promising results, showcasing its efficacy in capturing real-time variations in ambient light levels. Through meticulous calibration and integration with the chosen microcontroller platform, the circuit accurately detected and quantified changes in light intensity across different environmental conditions. The gathered data, seamlessly transmitted via LoRaWAN connectivity and meticulously stored in the InfluxDB database, provided valuable insights into light patterns over time. Visualization of this data in Grafana facilitated intuitive analysis, empowering stakeholders to make informed decisions regarding energy management, security systems, and environmental monitoring. The successful deployment of the LDR sensor circuit underscores its utility in diverse IoT applications, emphasizing its role as a reliable tool for harnessing environmental data for actionable insights and sustainable solutions.