Conservation of Energy in Microgrid using IoT and Smart Switches

EE 396 : Design Lab Report

Koustub Gururaja Rao and Korada Uday Kiran
(210102051 & 210102049)

under the guidance of

Dr. Shabari Nath

to the

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

Contents

List of Figures

Chapter 1

Introduction

The integration of Internet of Things (IoT) technologies and smart switches has revolutionized the management and optimization of energy distribution within microgrids. By leveraging IoT-enabled devices and intelligent switching mechanisms, microgrids can dynamically monitor, control, and allocate energy in real-time, to maximize efficiency while minimizing waste.

1.1 Problem Statement

In microgrid systems, transitioning to generator power during outages poses a challenge in managing high electrical consumption devices like air conditioners (ACs). To address this, our project aims to use Iot (LoRa) to detect generator activation and wirelessly signal distant WiFi switches to promptly turn off such devices, ensuring optimal resource utilization and system stability.

1.2 Need for a solution

In real-life scenarios, micro-grid systems are increasingly relied upon to provide resilient and sustainable energy solutions, particularly in areas prone to power outages or with unreliable grid infrastructure. The transition to generator power during outages introduces challenges in managing energy consumption efficiently. Without proper control mechanisms, high electrical consumption devices such as air conditioners can strain the system, leading to inefficiencies, increased costs, and potential equipment damage. Therefore, there is a critical need for automated solutions that can intelligently detect generator activation and adjust device operations despite topology constraints.

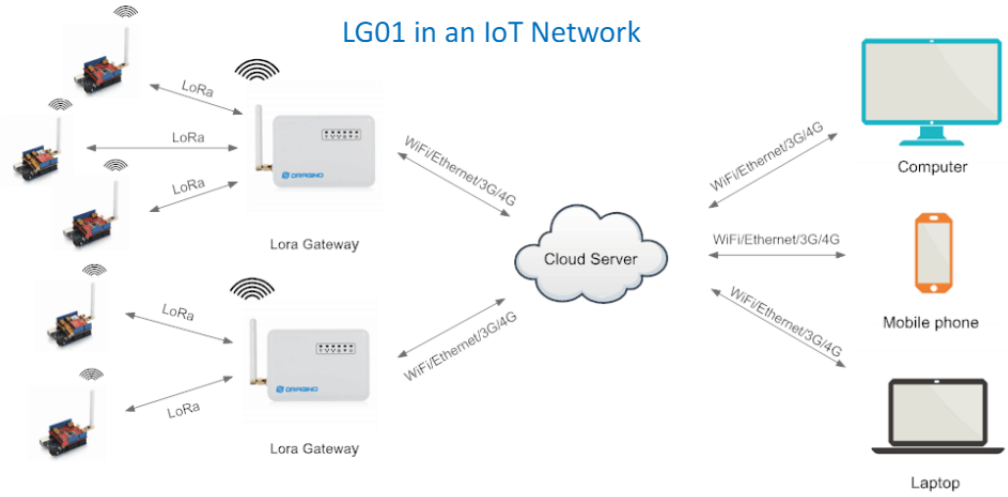

Fig. 1.1 Automation using LG01

1.3 Topology of the Circuit

We have provided a description of micro-grids, and their applications. Now, we described the model that represents the topology of circuit.
The complete circuit can be distributed into two parts:
i) The Transmitter
ii) The Receiver

The transmitter is placed at the generator location and detects when the micro-grid has switched to generator power. This is transmitted to the receiver placed somewhere near the appliances (In our case we could consider the Core Departments) through Lora.

The receiver is a commonly available WiFi-based smart switch that receives a down-link from a Lora gateway(placed in the department) through WiFi and this down-link can be propagated throughout the department by using the same WiFi.

Fig. 1.2 IITG Power Station

Fig. 1.3 IITG Core Complex

Chapter 2

The Circuit Design

Now, since we have established the topology, we zoom into the individual ends and design circuitry for the same.

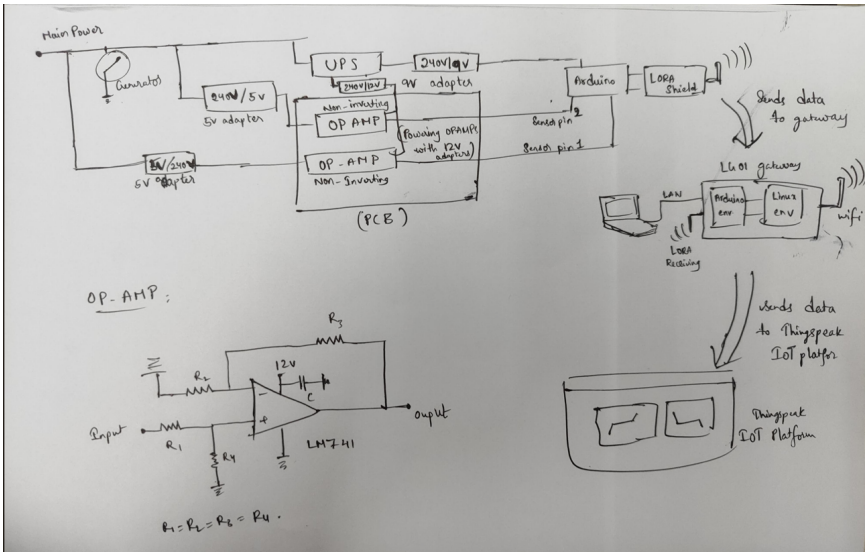

Fig. 2.1 Circuit Diagram

2.1 Detection:

The generator is a parallel line connected between the main line and ground as shown in the diagram. According to Indian Device Standards, the generator produces 240V AC current. To detect whether there is a power outage, we require two sensors, placed at Main Power(A) and Common line between power and generator(B) respectively. A basic truth table can be drawn based on sensor detecting power.

The detection is done by a micro controller(Arduino). Due to safety reasons however, it is not advisable to directly supply adapter output to the Arduino. Thus, we have designed a PCB which takes both these inputs, passes them through a differential amplifier and feeds the output to the microcontroller.

(A)	(B)	Meaning	Action
1	X(0 or 1)	Power Present	Do Nothing
0	0	Power Cut	Detect But Do Nothing
0	1	Generator Power	Turn off AC

Fig. 2.2 Truth Table for Sensor based action

2.2 Transmission:

The above data is transmitted by using a Lora shield and antenna connected to the arduino, which connects to the Lora gateway(placed in the department).

2.3 Detection:

The aim is to turn off the AC present in the department and this is achieved by using IFTTT and Wifi switches which are triggered when the gateway recieves transmission from the lora shield.

Chapter 3

PCB Design

3.1 The Differential amplifier:

To detect the presence of power in the lines, connection is made between the power lines and Arduino using a 240V AC - 5V DC adapter. However, it is not advisable to directly supply this power. We use a differential amplifier to supply sensor value. These differential amplifiers implement LM741 OP-AMP. Post calculations for power & taking amplification as 1 along with a 12V supply(for opamp) we decide in the range of 2-5 K-ohm resistors.

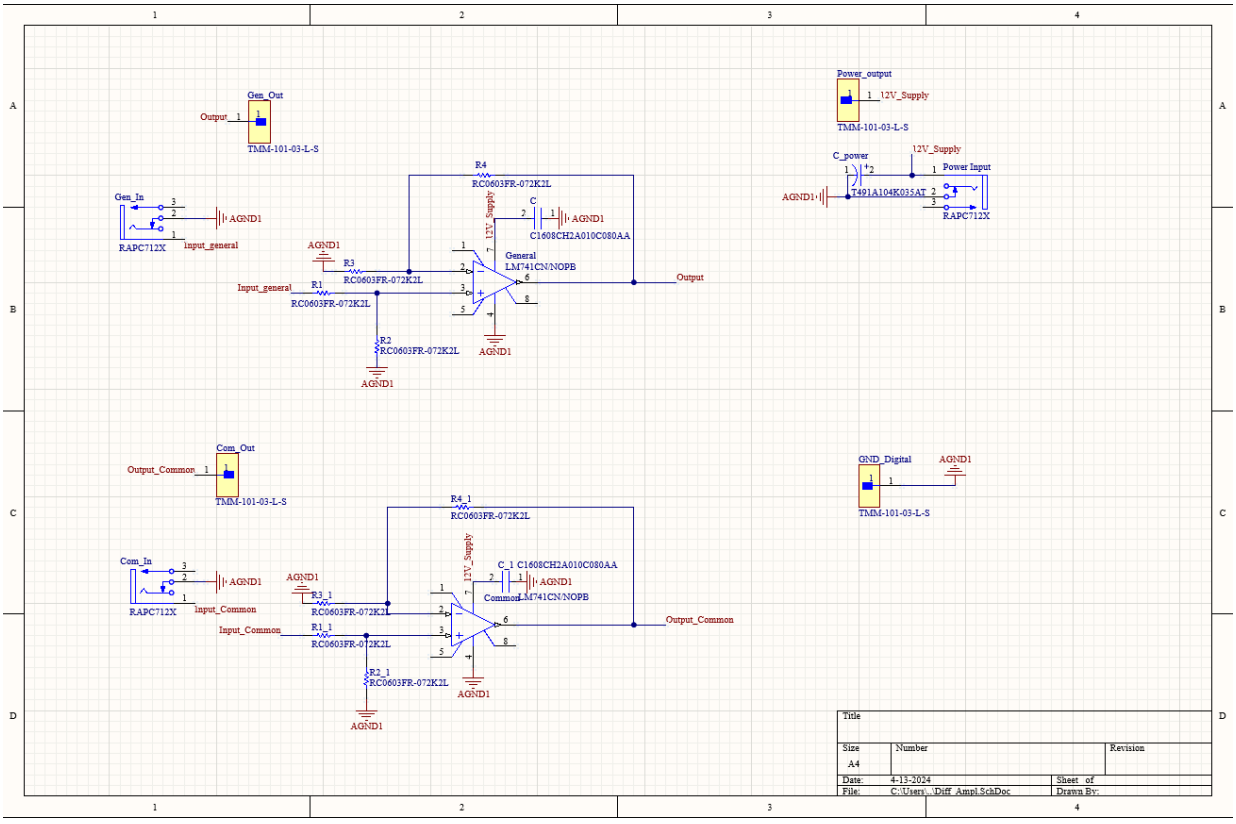

Fig. 3.1 Circuit Schematic

3.2 Altium:

We used Altium, a software platform for designing electronic circuits and printed circuit boards (PCBs) to create this PCB. The required GerberFiles, NC Drill files are also generated with ease.

Entire PCB building process involves the following steps:

- Circuit Diagram
- Components required
- Building Schematic and Schematic library
- Building PCB based on schematic while minimising size
- Troubleshooting
- Generating Gerber and NC Drill Data
- Giving PCB for manufacture

3.3 Circuit Flow:

The adapters are connected to the PCB's Gen_In and Com_In ports.
The power (12v) is supplied through power input.
The outputs are taken from the Gen_Out and Com_Out pins.

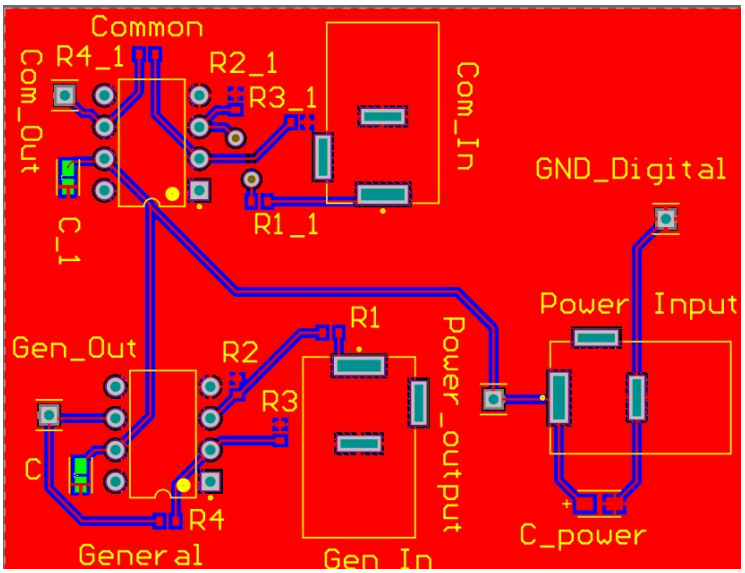

Fig. 3.2 PCB Top View

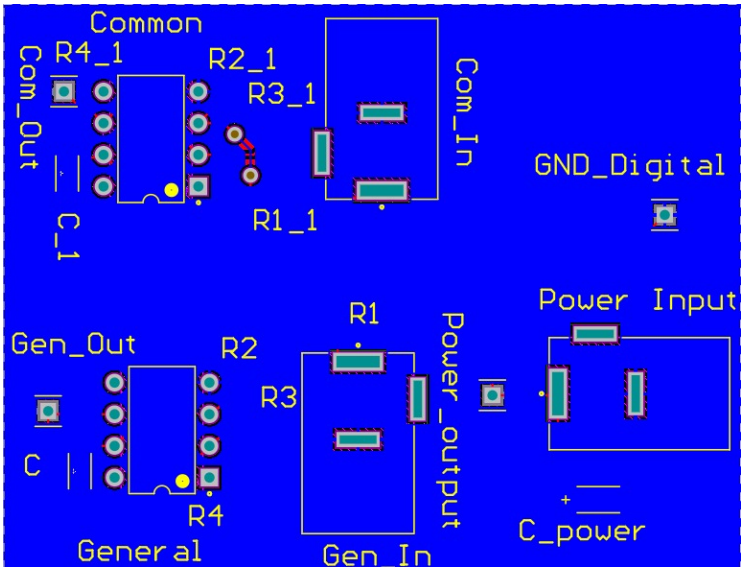

Fig. 3.3 PCB Bottom View

Chapter 4

Transmitter Design:

Post detection of power outage, we need to transmit a signal which not only indicates whether or not power is cut and the device needs to be turned off since it is running on generator power but also, it must grant the user the freedom to turn it on despite running on generator.

4.1 The Lora shield:

An Arduino UNO paired with a LoRa shield is employed to transmit messages. The LoRa shield facilitates long-range communication, making it well-suited for IoT applications. Operating on unlicensed bands, it ensures reliable communication across vast distances. This capability is particularly beneficial for transmitting messages between distant powerhouses and WiFi switches in homes or offices. The LoRa gateway receives these transmissions and relays them to the WiFi switches. In this setup, the SX127x Shield, based on the Semtech SX1276/SX1278 chip, is utilized. It is tailored for professional wireless sensor network applications such as irrigation systems, smart metering, smart cities, and building automation.

4.2 Libraries for Lora shield:

This code makes use of two libraries: SPI.h and RH_RF95.h. The SPI.h library offers functions for interacting with devices through the Serial Peripheral Interface (SPI) protocol, typically employed for communication between microcontrollers and peripherals like sensors and displays. Meanwhile, the RH_RF95.h library is specifically designed for the RFM95 LoRa transceiver module, providing functions to initialize, configure, and communicate with the module.

4.3 Dragino Lora Gateway :

The LG01 is an open source single channel LoRa Gateway. It lets you bridge LoRa wireless network to an IP network base on WiFi, Ethernet, 3G or 4G cellular. LG01 runs on open

source embedded Linux system; it has a USB host port and has full Ethernet and 802.11 b/g/n WiFi capabilities.

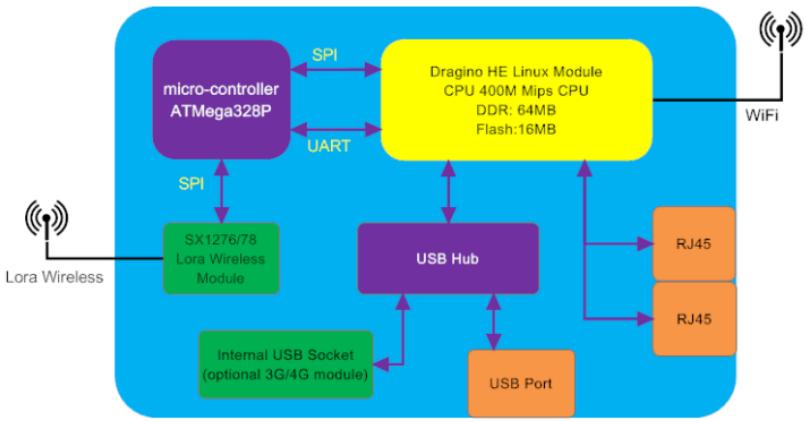

Fig. 4.1 LG01 System Overview

4.4 Configure IoT Server

A RESTful API is a type of application program interface (API) that utilizes HTTP requests to perform operations such as GET, PUT, POST, and DELETE on data. Many servers, including ThingSpeak, support RESTful APIs, offering intuitive charts to visualize test results. The method described here is versatile and can be applied to connect with other IoT servers using RESTful communication. To utilize ThingSpeak, a channel needs to be created, and its information, including the unique Channel ID, is inputted. Additionally, API keys and the API call method are required for communication with the channel, which can be found in the API Keys page of ThingSpeak.

4.5 Linux System

The LG01 operates on the OpenWrt Linux System, providing an open-source platform where users have the freedom to configure and adjust the internal Linux settings as needed. Users can connect to the Linux console using an SSH tool like PuTTY. This console allows users to view received results, make API calls, and perform other necessary tasks.

4.6 RESTful API call

We can send RESTful API call via two ways.
Example API url - https://api.thingspeak.com/update?api_key=B9Z0R25QNVEBKIFY&field1=0
api_key: Define th channel on which data is to be uploaded.
Field: Each channel has a maximum of 8 fields. Field1, Field2 define which exactly field to be updated.
Value: Field1=0, means implies the Field1 to update value to 0.

1. Through pasting the API url in any web browser.
2. Through Putty by calling the curl command.

Command for the Above URL:

curl -d "key=B9Z0R25QNVEBKIFY&field1=0" -k http://api.thingspeak.com/update

4.7 Transmitter Side Architecture From Shield to Thingspeak

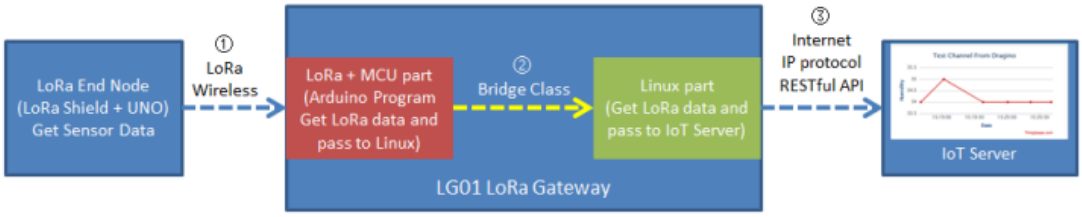

Fig. 4.2 Transmission Architecture

4.8 Gateway Libraries:

This code employs SPI.h, RH_RF95.h, Bridge, and Process libraries to receive and transmit data to the ThingSpeak IoT platform. Utilizing SPI.h and RH_RF95.h libraries enables the reception of packets sent by the LoRa Shield and sends acknowledgments back to the shield. The data is transmitted concurrently from the Mega328P MCU (Arduino Environment) to the Dragino HE AR9331 Module (Linux Environment) via the Bridge Library. Subsequently, using the Process Library and Linux commands within the code, the data is forwarded from the Linux Environment to the IoT Server through API calls.

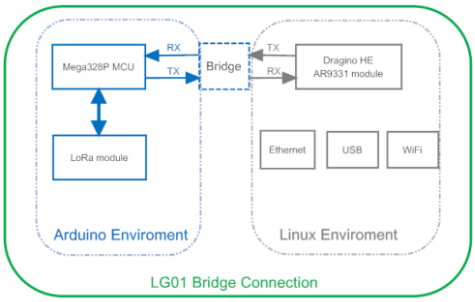

Fig. 4.3 Connection between Arduino and Linux environments

Note: In this process, we must ensure that both the gateway and Programming computer are on the same network.

4.9 Lora shield Code and Gateway Source Code:

The following section shows the code between Lora shield and gateway in which Shield acts as Trasmitter and also waits for the achknowledgement for the gateway. The serial monitor is used to display communication logs between during the process. Snippets 1,2 and 3 are for transmitter and 4 and 5 for gateway.

```
#include <SPI.h>
#include <RH_RF95.h>

RH_RF95 rf95;

#define node_id_length 3 // Length of Node ID
#define sensor_1_pin A0  // Pin for sensor 1
#define sensor_2_pin A1  // Pin for sensor 2

char node_id[node_id_length] = {1, 1, 1}; // LoRa End Node ID
float frequency = 868.0;
unsigned int count = 1;
bool acknowledgmentReceived = false; // Flag to track acknowledgment status
byte prevcombinedbits = 0b11;

void setup()
{
    Serial.begin(9600);
    if (!rf95.init())
        Serial.println("init failed");
    // Setup ISM frequency
    rf95.setFrequency(frequency);
    // Setup Power,dBm
    rf95.setTxPower(13);
    rf95.setSyncWord(0x34);

    Serial.println("LoRa End Node Example --");
    Serial.println("    Sensor Data Sending\n");
    Serial.print("LoRa End Node ID: ");

    for (int i = 0; i < node_id_length; i++)
    {
        Serial.print(node_id[i], HEX);
    }
    Serial.println();
}

// Function to read sensor value and determine bit based on threshold
byte readSensorAndAssignBit(int sensorPin)
{
    int sensorValue = analogRead(sensorPin);
    return sensorValue > 125 ? 1 : 0;
}

int solve(byte bits, bool &ack, byte prev)
{
    if (bits == 0b11)
    {
        ack = false;
        return 0;
    }
    else if (bits == 0b01)
```

```
  {
    if (ack == false)
    {
      return 50;
    }
    else
    {
      return 20;
    }
  }
  return 0;


oid loop()

  Serial.print("###########    ");
  Serial.print("COUNT=");
  Serial.print(count);
  Serial.println("    ###########");
  count++;

  // Read sensor values and assign bits
  byte bit1 = readSensorAndAssignBit(sensor_1_pin);
  byte bit2 = readSensorAndAssignBit(sensor_2_pin);

  // Combine the bits into one byte
  byte combinedBits = (bit1 << 1) | bit2;

  char data[node_id_length + 1] = {0}; // Payload Length (Node ID + Sensor Data)
  // Use node_id as the first bytes of data
  for (int i = 0; i < node_id_length; i++)
  {
    data[i] = node_id[i];
  }

  // Decide data to be sent based on acknowledgment status and combined bits
  int dataToSend = 0;
  dataToSend = solve(combinedBits, acknowledgmentReceived, prevcombinedbits);
  prevcombinedbits = combinedBits;

  // Print sensor values and combined bits
  Serial.print("Sensor 1 value: ");
  Serial.println(analogRead(sensor_1_pin));
  Serial.print("Sensor 2 value: ");
  Serial.println(analogRead(sensor_2_pin));
  Serial.print("Combined Bits: ");
  Serial.println(combinedBits, BIN);

  // Print data to be sent
  Serial.print("Data to be sent: ");
  Serial.println(dataToSend);
```

```
Serial.println(dataToSend);

// Set the last byte of data as the data to be sent
data[node_id_length] = dataToSend;

rf95.send((uint8_t *)data, sizeof(data)); // Send LoRa Data

// Wait for acknowledgment

if (rf95.waitAvailableTimeout(3000))
{
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN]; // receive data buffer
    uint8_t len = sizeof(buf);            // data buffer length
    if (rf95.recv(buf, &len))             // check if reply message is correct
    {
        if (buf[0] == node_id[0] && buf[1] == node_id[2] && buf[2] == node_id[2]) // Check if reply message has the our node ID
        {

            Serial.print("Got Reply from Gateway: "); // print reply
            Serial.println((char *)buf);
            acknowledgmentReceived = true;
        }
    }
    else
    {
        Serial.println("recv failed"); //
    }
}
else
{
    Serial.println("No reply, is LoRa gateway running?"); // No signal reply
}
delay(3000); // Send sensor data every 3 seconds
Serial.println("");
```

```
#include <SPI.h>
#include <RH_RF95.h>
#include <Console.h>
#include <Process.h>
RH_RF95 rf95;

#define BAUDRATE 115200

String myWriteAPIString = "P07KVY59P5QEY6M6";
float frequency = 868.0;

void setup()
{
    Bridge.begin(BAUDRATE);
    Console.begin();
    if (!rf95.init())
    |   Console.println("init failed");
    ;
    // Setup ISM frequency
    rf95.setFrequency(frequency);
    // Setup Power,dBm
    rf95.setTxPower(13);
    rf95.setSyncWord(0x34);

    Console.println("LoRa Gateway Example  --");
    Console.println("    Upload Single Data to ThinkSpeak");
}

void loop()
{
    if (rf95.waitAvailableTimeout(2000)) // Listen Data from LoRa Node
    {
    |   uint8_t buf[RH_RF95_MAX_MESSAGE_LEN]; // receive data buffer
    |   uint8_t len = sizeof(buf);            // data buffer length
    |   if (rf95.recv(buf, &len))            // Check if there is incoming data
    |   {
    |   |   Console.print("Get LoRa Packet: ");
    |   |   for (int i = 0; i < len; i++)
    |   |   {
    |   |   |   Console.print(buf[i], HEX);
    |   |   |   Console.print(" ");
    |   |   }
    |   |   Console.println();
```

```
            if (len >= 4 && buf[0] == 1 && buf[1] == 1 && buf[2] == 1) // Check if the ID matches the LoRa Node ID
            {
                uint8_t randomValue = buf[3]; // Extract random value from received data
                uploadData(randomValue);      // Forward random value to ThingSpeak
            }
        }
    }
}

void uploadData(int randomValue)
{ // Upload Data to ThingSpeak
    // form the string for the API header parameter:

    // form the string for the URL parameter, be careful about the required "
    String upload_url = "https://api.thingspeak.com/update?api_key=";
    upload_url += myWriteAPIString;
    upload_url += "&field1=";
    upload_url += randomValue;

    Console.println("Call Linux Command to Send Data");
    Process p; // Create a process and call it "p", this process will execute a Linux curl command
    p.begin("curl");
    p.addParameter("-k");
    p.addParameter(upload_url);
    p.run(); // Run the process and wait for its termination

    Console.print("Feedback from Linux: ");
    // If there's output from Linux,
    // send it out the Console:
    while (p.available() > 0)
    {
        char c = p.read();
        Console.write(c);
    }
    Console.println("");
    Console.println("Call Finished");
    Console.println("###################################");
    Console.println("");
}
```

Chapter 5

Down-link to WiFi Switch:

Thingspeak is a MATLAB based IoT platform that allows users to collect, analyze, and visualize data from sensors or other devices in real-time. With its React functionality, users can trigger actions or alerts based on predefined conditions, enabling automated responses to data inputs.

5.1 IFTTT integration with Thingspeak:

IFTTT stands for "IF This Then That" and implements the functionality as the name suggests. On the IFTTT website under the applets section and under "If this", we choose webhooks which triggers when HTTP requests are sent. Then, under "If that" we select the "Smart Life" App which contains the trigger - turn off the WiFi switch. Additionally we receive an API Key from maker web-hooks which will be used in further steps.

5.2 Creating ThingHTTP:

ThingHTTP improves integration and automation possibilities by enabling Thingspeak users to make HTTP queries to other web services using information gathered from IoT devices. We enter the created webhook data and required data to be sent to the react functionality through the above using the GET method.

5.3 React triggers:

The React app on Thingspeak allows users to trigger actions based on specified conditions in their channel data. In this instance, a numerical value is sent from the receiver and is compared with a threshold value and if greater than threshold, we trigger the HTTP request to turn off the WiFi switch. Thus, complete automation of process is complete.

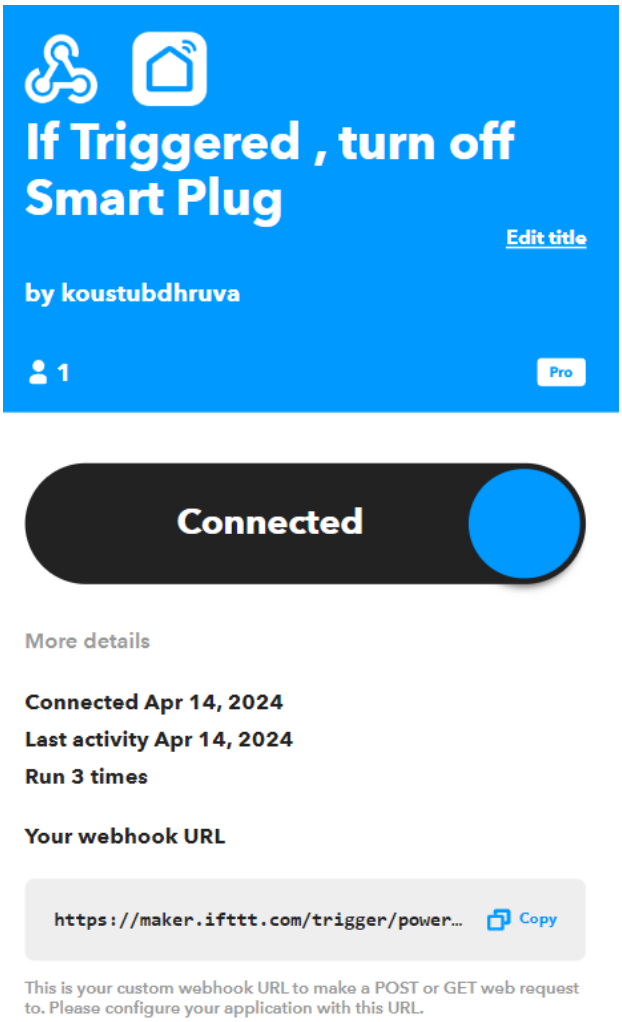

Fig. 5.1 IFTTT Webhook Applet

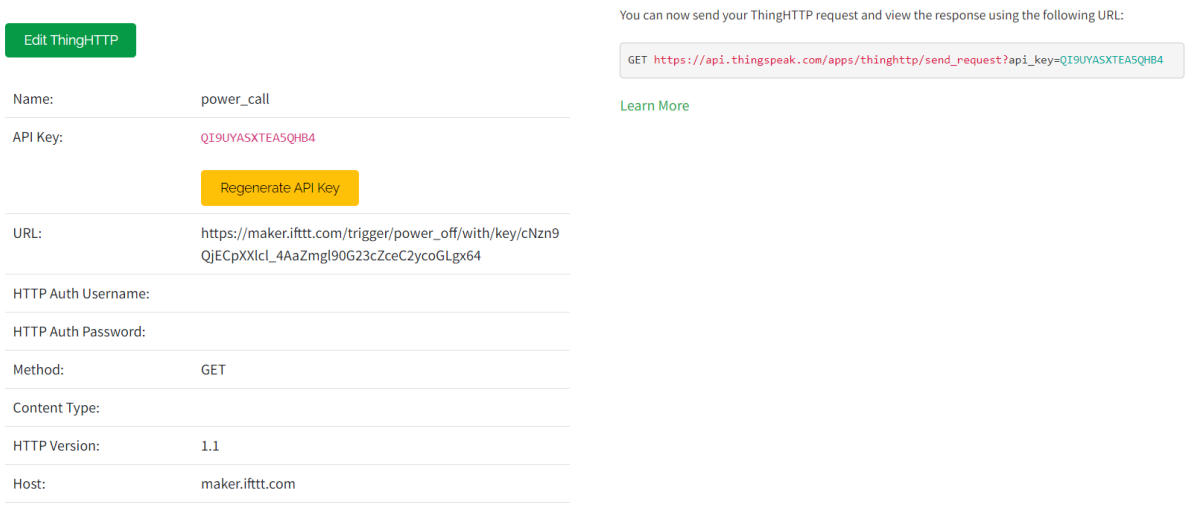

Fig. 5.2 ThingHTTP Example

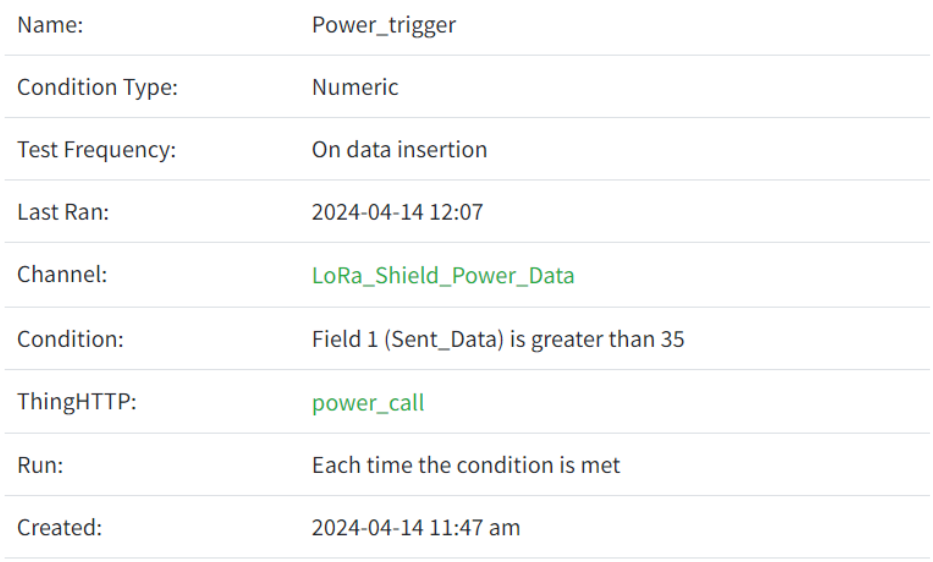

Fig. 5.3 Thingspeak React Example

Chapter 6

Conclusion and Future Work

The project aims towards automation of power-grids and the applications are wide from healthcare (such as huge hospitals to communicate between doctors), to agriculture (such as automation of irrigation) just through the change of the sensors present at the transmitter.

Fig. 6.1 LoRa Applications

6.1 Conclusions:

- Successfully implemented the objective to conserve energy in Microgrid using IoT and smart switches.
- Designed differential amplifier PCB to add layer of safety between Power lines and sensing Arduino + Dragino Lora Shield.
- Established automated uplink communication between Lora shield and Gateway.
- Additionally implemented communication between Gateway and Thingspeak to process received data, and trigger React functionality to send HTTP request.
- Implemented IFTTT webhooks to receive HTTP request and in turn automate the Wifi switch using the Smart Life app.
- LG01-N is an open source single channel LoRa Gateway. The website suggests that it is not recommend for LoRaWAN use. Thus, integration with commonly available resources such as The Things Network(TTN) is not suggestible for the same.

6.2 Future Work:

- Here we utilised IFTTT to integrate LoRa and Wifi Switch communications. Since our switch enables IFTTT integration feature, we are able to implement this feature.
- We can also control our WiFi switch using a more generic feature called Tasmota. Tasmota is used for configuring the wifi devices such that we can control the device. Implementation of MQTT is possible in this route.
- The above circuit can further be connected to the generator and automate it to work only during power outages by modifications in code.

6.3 References:

- LoRa and LoRaWan Thoery: Dragino Documention, Dragino LG01- P Manual, and Internet
- Tramsmitter Design: Dragino LG01- P Manual
- Gateway and Transmitter Architecture Pictures: Dragino LG01- P Manual
- Downlink to wifi Switch: https://in.mathworks.com/help/thingspeak/use-ifttt-to-send-text-message-notification.html
- Downlink to wifi switch pictures: Screenshots taken from the respective websites
- Tasmota Documentation: https://tasmota.github.io/docs/Getting-Started/
- Tasmota Devices List : https://templates.blakadder.com/

References