Abstract

Smart microgrids can be used as a remedy for outages and load control that cause the primary power grid in a region to go down, which could have a significant negative impact on several households, companies, and vital services. However, it is too expensive to create a smart microgrid. The energy demand along with the demand for the automation of microgrids is increasing with time. The aim of our project is to develop a system that automates the existing microgrid without replacing the system. And to Help plug gaps in the energy supply, Reduce wastage of energy

The ESAM (Economic Sensing & Automation in Microgrids) project aims to enhance the efficiency, reliability, and sustainability of microgrids through the application of advanced technology. By integrating real-time data analysis, load prioritization, and automation, ESAM enables optimized power distribution and improved energy management. The project focuses on addressing challenges such as load control, multiple source integration, and predictive analytics to ensure uninterrupted power supply and cost savings. With its innovative approach, ESAM offers a promising solution for the future of microgrid systems, contributing to a more resilient and sustainable energy infrastructure.

Acknowledgements

We would like to express my sincere gratitude to Dr. Archana R, Head of the Department of Electrical and Electronics Engineering for her valuable guidance throughout the project. This project would not have been possible without her help and support. I would also like to thank all the Faculty of the Department of Electrical and Electronics Engineering for their constant support without which this work would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From the beginning of the modern world in the 19th century electricity has played a significant role in all aspects of life, in future also power and energy is the key to many problems and developments. Distribution of power has its important from the beginning. A significant development in the distribution and control of electricity was visible throughout the 20th and 21st centuries. Long AC transmissions, then HVDC transmission, etc... along with distribution system control and monitoring systems also developed from ordinary grid systems to power grids to smart and microgrid concepts. With the development, demand was also rising along with challenges. Smart grids and microgrids bring many solutions to the existing problems, the Economic Sensing & Automation For Micro Grids(ESAM) brings out modern technology to the existing microgrids. The Internet of Things(IoT) refers to an interconnected network of physical devices, vehicles, buildings, and other objects that are embedded with sensors, software, and network connectivity, allowing them to collect and exchange data. The IoT allows for the automation and control of these connected devices, enabling them to communicate and interact with each other and with humans in a coordinated and intelligent manner. The IoT has the potential to transform a wide range of industries and applications, including manufacturing, transportation, agriculture, healthcare, and energy. We use IoT in microgrids to monitor the power specifications(Load, Voltage variations, Energy usage, etc..) and control the load to get more efficient and dynamic power distribution. The system is designed in a modular method that can be installed in any existing grid without affecting current

systems. Complete control and monitoring are enabled through a web user interface that gives the user complete authority to control the functioning and gives a detailed analysis of the system.

1.1 Problem

Smart microgrids can be used as a remedy for outages and load control that cause the primary power grid in a region to go down, which could significantly negatively impact several households, companies, and vital services. However, it is too expensive to create a smart microgrid in developing nations and industries.

1.2 Objectives

1. Analyze loads and supply for efficient microgrid management- Accurately assess and analyze the energy loads and supply within the microgrid to enable effective management and allocation of resources.

2. Prioritize and control loads for optimized energy consumption - Implement intelligent load prioritization and control mechanisms to ensure efficient energy consumption, focusing on high-priority loads while minimizing wastage.

3. Maximize cost savings and reduce manual effort - Reduce installation costs and enhance efficiency without the need to remove or replace existing systems, ensuring cost-effective implementation, also optimize energy usage, and automating processes to minimize costs and manual intervention, resulting in significant cost savings and reduced effort.

4. Centralize distribution control for streamlined microgrid operations - Establish a centralized control system for efficient management and coordination of microgrid operations, enabling seamless monitoring and control of power distribution.

Chapter 2

Literature Review

In traditional power systems, analog variables explain all the phenomena. The implementation of digital devices provides the power system with new forms of remote monitoring and control, which provides smartness to the system. Remote monitoring is achieved by communicating with the DER systems by using digital variables. The development of digital control is beneficial to all actors and specifically to end users in the modern power system. One important step towards the digitalization of the power system was the introduction of the Intelligent Electronic Device (IED), allowing the direct conversions of analog inputs, like voltages and currents, into digital variables and calculated results. Such devices are implemented with appropriate protocols combined with communication and computing systems. Protocols such as Modbus TCP/IP and Distributed Network Protocol3 (DNP3) are widely used in commercial DER inverters to facilitate communication with supervisory devices. The IEC 61850 standard is designed to provide a universal communication solution that allows devices from different vendors to work together seamlessly in a smart microgrid system. A multi-layer structure is often used in smart microgrids to decentralize the system's intelligence by dividing control tasks among different layers. The communication architecture of a multi-layer smart microgrid system must address two main challenges: coordinating communication with system control and coordinating communication between different layers. Testbeds for microgrids have been proposed in the literature, but most have focused on system monitoring rather than control functionality, or have been limited to specific communication protocols or hardware.

To address these issues, a multi-layer architecture for a smart microgrid system that uses the IEC 61850 standard for communication and includes a real-time control platform with a deterministic communication protocol. The proposed architecture is tested and validated through simulations and experiments. Several universities and institutes have developed their own microgrid testbeds with integrated communication architectures. These testbeds have used a variety of hardware and communication protocols, including CompactRIOs, RTDSs, EMSs, SCADAs, Modbus TCP/IP, Ethernet, LAN, and CAN bus. Some of these testbeds have focused on real-time control, while others have primarily focused on system modeling and simulation. Many of these testbeds have not addressed the challenges of coordinating communication with control and coordinating communication between different protocols in their communication architectures. The development of a communication solution for a smart microgrid system that is fast, convenient, and comprehensive, and has been validated through experiments using actual hardware rather than emulators. The proposed solution includes strategies for coordinating communication with control and for coordinating communication between different protocols. These strategies include the use of dual-core DSPs, the adjustment of time scales for different types of communication, the introduction of software protection mechanisms, the adoption of parallel computing, and the optimization of communication efficiency by considering the features of different protocols. The hardware prototypes for this solution are built with off-the-shelf components, making it ready for use in industrial applications in large distribution systems with high renewable energy penetration. This describes the 4-layer communication architecture, the hardware design of the customized Edge Intelligent Device (EID), and the implementation of the communication protocols (SPI and Modbus TCP/IP) in detail, and will present experimental results demonstrating the effectiveness of the proposed solution.[4]

The Raspberry Pi serves as the communication interface between the Process layer and the Sub-station layer in the proposed smart microgrid system. It is responsible for transmitting information collected in the Process layer to the Sub-station layer using both SPI and Modbus communication. The Raspberry Pi can operate in either serial or parallel computing mode. In the serial mode, the Raspberry Pi sends data to the EID only after receiving data from the Communication DSP, and vice versa. This

means that the SPI and Modbus communication are coupled, which can negatively impact the control of the DER inverters. The Modbus read and write operations typically take hundreds of microseconds, while the switching cycle is usually only tens of microseconds, making it impossible for the SPI communication to complete within a single switching cycle if serial computing is used. Additionally, the amount of data that can be transferred through SPI communication in a single switching cycle is limited in this mode. Using Modbus communication in series with SPI does not fully utilize its advantage of transmitting large amounts of data.[4]

There are many ways to configure networks for Internet of Things (IoT) services, and the best option depends on the specific requirements of the IoT application and the resources available. Some common network configurations for IoT services include: [3]

1. Star network: In a star network, all devices are connected to a central hub or gateway, which serves as the point of communication for the entire network. This configuration is simple and easy to set up, but the central hub can become a bottleneck if there are too many devices or if the data transfer rates are high.

2. Mesh network: In a mesh network, devices communicate with each other directly, rather than going through a central hub. This allows for more robust communication, as there are multiple paths for data to travel through the network. However, mesh networks can be more complex to set up and maintain.

3. Hybrid network: A hybrid network combines elements of both star and mesh networks, allowing for both a central hub and direct device-to-device communication. This can provide the benefits of both configurations, but may also be more complex to set up and manage.

4. Cloud-based network: In a cloud-based network, devices communicate with a central cloud-based server rather than a local hub. This can be convenient, as it allows for remote access and management, but may also be more vulnerable to security threats and require a stable internet connection.

5. Wireless network: Many IoT applications use wireless communication, such as WiFi or cellular networks, to connect devices. This allows for flexibility and

ease of deployment but may be limited by range and the availability of wireless infrastructure.

Some factors to consider when configuring communication for an IoT DEP include:[3]

1. Communication protocol: Different IoT DEPs may support different communication protocols, such as Bluetooth, WiFi, LoRaWAN, Zigbee, or cellular networks. The best protocol for an IoT DEP depends on the required data transfer rates, range, power consumption, and other factors.

2. Network topology: The overall communication architecture of the IoT system, such as whether it uses a star, mesh, hybrid, or cloud-based network, will also impact the communication configuration of the DEP.

3. Security: Ensuring secure communication is crucial for IoT DEPs, as they may be transmitting sensitive data or controlling critical systems. This may involve implementing encryption, authentication, and other security measures.

4. Quality of Service (QoS): The QoS requirements for an IoT DEP depend on the importance and timeliness of the data it is transmitting. This may involve setting priorities for different types of data or implementing mechanisms to ensure reliable delivery.

5. Interoperability: If the IoT DEP needs to communicate with other devices or systems that use different communication protocols or standards, it may be necessary to include interoperability features in the communication configuration.

A microgrid is a small-scale power system that consists of distributed energy resources (DERs), such as renewable energy sources and energy storage systems, connected to loads. Microgrids can operate in both connected and isolated modes and can vary in size and complexity. They are typically used to increase the reliability and resiliency of power distribution systems, as well as to optimize energy use and reduce costs. The control system of a microgrid is responsible for maintaining stability and ensuring that power supply and demand are balanced. It can be divided into three hierarchies: primary control, which involves the fastest actions such as islanding detection and voltage and frequency control; secondary control, which involves longer-term optimization and demand management; and tertiary control, which

involves overall coordination and planning. The primary control of a microgrid is important for stability, particularly in isolated microgrids where voltage and frequency must be maintained by the system itself. Multiple architectures have been proposed for primary control, including centralized and decentralized approaches .

The microgrid control system is responsible for maintaining stability, optimality, and reliability within the microgrid, and can be divided into three hierarchies: primary, secondary, and tertiary. Primary control is responsible for the fastest control actions in the microgrid and includes voltage and frequency control and power sharing and balance. The secondary control hierarchy is responsible for optimizing the operation of the microgrid and includes economic dispatch and demand-side management. Tertiary control is responsible for long-term planning and coordination with the main grid. There are several challenges to maintaining stability in microgrids, including primary, secondary, and tertiary stability. Primary stability challenges include frequency and voltage stability, power sharing and balance stability, and small-signal stability. Secondary stability challenges include power quality and voltage and frequency stability. Tertiary stability challenges include inter-area oscillations, power flow constraints, and reactive power compensation.[1]

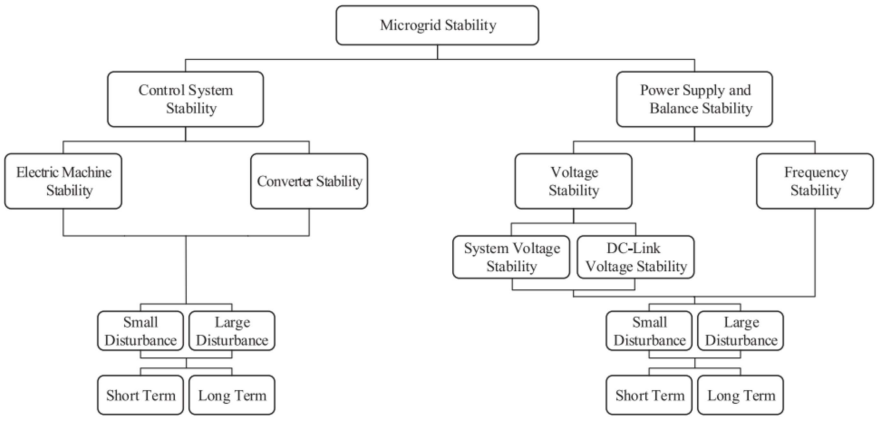


Figure 2.1: Classification of stability in microgrids

Smart metering systems are systems that collect and provide information about energy consumption to consumers and allow for two-way communication between energy devices and users. They can also be used to monitor and adjust various parameters, such as voltage, current, temperature, and moisture levels. Smart homes

are modern homes equipped with devices and appliances that can be managed remotely through the Internet of things (IoT). Smart buildings are similar to smart homes, but they apply IoT integration to commercial, industrial, public, or residential buildings to monitor and control various electrical, mechanical, environmental, and security systems. A building management system (BMS) is a computer-based system used to automatically monitor and control these systems in a building with the aim of managing the environment and reducing carbon emissions .

There are several security vulnerabilities that can arise in IoT-connected smart energy systems. These vulnerabilities can be classified into two categories: vulnerabilities related to the communication infrastructure and vulnerabilities related to the system itself.

Vulnerabilities related to the communication infrastructure include attacks on the communication channels, such as man-in-the-middle attacks and denial-of-service attacks. These attacks can disrupt the flow of information between different components of the system and can lead to system failures.

Vulnerabilities related to the system itself include attacks on the devices and systems that make up the smart energy system, such as attacks on the control systems, data storage systems, and other critical components. These attacks can result in unauthorized access to sensitive information, manipulation of system parameters and other types of damage .

To mitigate these vulnerabilities, it is important to implement security measures at multiple levels of the system. These measures can include strong encryption algorithms, secure communication protocols, and regular security updates to devices and systems. It is also important to implement robust security policies and procedures to ensure that only authorized personnel have access to the system.[2]

Chapter 3

Survey and Analysis of a microgrid

3.1 Introduction

The efficient management of electrical distribution systems is of paramount importance in educational institutions like the Federal Institute of Science And Technology(FISAT). With a diverse range of power sources, including the main grid, solar panels, UPS systems, and generators, along with various loads distributed throughout the campus, it becomes crucial to ensure optimal utilization of available resources.

The objective of this survey was to comprehensively analyze the existing electrical distribution system within our college and identify the challenges and opportunities for improvement. The survey encompassed a detailed examination of different sources, load characteristics, the current distribution system structure, maximum demand patterns, load measurements, and peak hours.

One significant issue uncovered during the survey is the imposition of penalties due to excess demand during peak times. These penalties not only result in financial losses but also hinder the smooth operation of essential facilities and negatively impact the overall functioning of the college.

In light of this finding, the focus of our project is to develop a solution that effectively reduces penalties caused by excessive demand at peak times. By implementing advanced control and optimization techniques, we aim to enhance the efficiency of our electrical distribution system, ensuring optimal load management and minimizing penalties associated with peak demand.

By addressing these challenges, our project strives to improve the reliability, sustainability, and cost-effectiveness of the electrical distribution system in our college, thereby benefiting the entire campus community.

This survey serves as a foundation for understanding the existing system and the problems it poses. The subsequent sections will delve deeper into the findings, analysis, and proposed solutions to overcome these challenges.

3.2 Methodology

1. Interaction with officers and engineers: In-depth discussions were conducted with officers and engineers responsible for the electrical distribution system within the college. These interactions provided valuable insights into the system's design, maintenance practices, load management challenges, and their experiences with peak period demand.

2. Load readings using a survey form: A survey form was developed to record load readings at various locations within the college. This involved taking physical measurements of voltage, current, and power consumption, capturing the characteristics and patterns of different loads. The collected data formed the basis for analyzing load profiles and demand fluctuations throughout the day.

3. Collection and analysis of historical data: Previous electrical bills and relevant data were gathered and analyzed. This historical information included energy consumption patterns, peak hour analysis, and any penalties or charges incurred due to excessive demand. Analyzing this data provided insights into the system's performance, load variations over time, and potential areas for optimization.

By combining these three methodological approaches-interactions with personnel, load readings using a survey form, and analysis of historical data comprehensive assessment of the electrical distribution system within the college was achieved. This approach enabled a deeper understanding of the systems operations and paved the way for identifying specific challenges and opportunities for improvement.

3.3 Findings And Observations

The institute operates a complex electrical system with the Kerala State Electricity Board's utility grid serving as the primary power source. The grid supplies a 3-phase 11KV high-tension (HT) connection. Within the campus, an indoor substation is installed, housing a 400KVA transformer with a rating of 11KV/433V. The institute has a total connected load of 1140.15kW, with a maximum demand of 221.27kW. To ensure improved performance and power-factor adjustment, a capacitor bank is incorporated into the system.

Additionally, the institute has implemented a stand-alone solar PV system and a 200KVA capacity UPS battery system. The UPS system is divided into multiple sections across the campus to cater to various points of load requirements. To ensure backup power, diesel generators with a total capacity of 320KVA are installed.

The complexity of the electrical system lies in its distribution network. The campus spans a vast area and includes multiple blocks, each with different types of connected loads. These loads vary not only in their electrical properties but also in terms of priority and specific requirements. Consequently, source lines are directed to each load based on various constraints. Each block has a distribution board (DB) that receives inputs from all the power sources. From each DB, multiple points are derived based on load characteristics. For regular loads such as classroom lights and fans, simple connections with breakers are used. However, for labs equipped with computers, a separate DB with UPS connections is established to meet the additional power needs.

The workshops and computing facilities, which operate heavy machinery, represent the highest or biggest loads on the campus. However, it's important to note that these loads are not continuously utilized or activated simultaneously. Their usage varies significantly depending on the time of day and specific days. The central computing facility and the central server, along with the office and guest rooms, are considered the major or most crucial loads within the institute. Interruptions in these areas can lead to severe issues and disruptions in operations.

It's worth mentioning that the utility company has set a maximum load limit of 250kW. Nevertheless, there are instances during peak hours where the demand surpasses this threshold, resulting in potential breaches of the allowed load capacity.

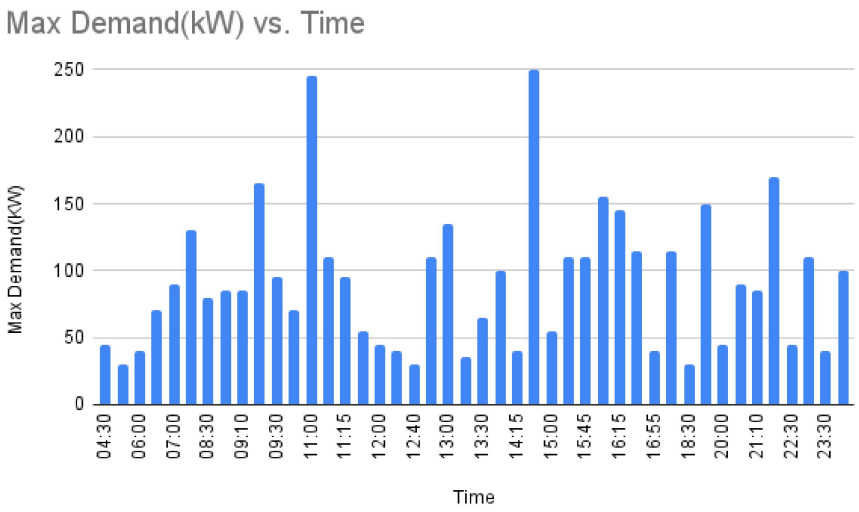


Figure 3.1: Time based data (extrapolated)

3.3.1 Data Collection & Interpretation

For data collection, measurement units for various parameters were installed at the distribution point within the campus. By analyzing these measurements, we plotted the time-demand curve for each day of the week.

Based on the observed data and analysis, we have drawn the following conclusions:

1. Total Demand Dependency: The total demand of the electrical system is influenced by multiple factors, including:
   (a) Day of the week: Different days exhibit variations in demand patterns.
   (b) Time period: Demand fluctuates throughout the day, with certain periods experiencing higher demand than others.
   (c) Weather & Climate Conditions: Environmental factors, such as temperature and weather conditions, impact the overall demand on the system.
2. Load Prioritization: To effectively manage the loads, it is crucial to establish prioritization categories based on their significance and requirements. By categorizing the loads, efficient load management and distribution can be achieved.
3. Load Peaks: The analysis of the time-demand curve has revealed certain conditions under which load peaks occur, including:

(a) Hot Sunny Days: High temperatures and sunny weather conditions tend to increase the load demand.

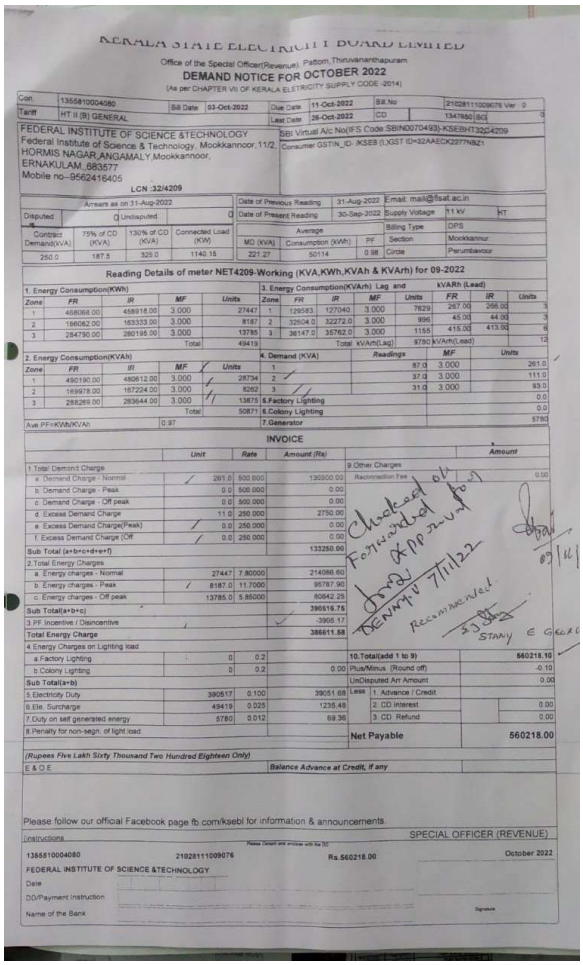
(b) Noon Time: Peak load demands are often observed during the midday hours.

(c) Simultaneous Workshops: Days, when multiple workshops operate concurrently, can lead to increased load demands.

3.3.2 Previous bills assessment

Two previous electricity bills were assessed to gather valuable data and insights for our project. The bills covered a specific time period and were obtained from the utility company records. Through careful analysis, we extracted key information from each bill, including energy consumption patterns, peak demand recorded, and cost breakdowns. The analysis revealed notable findings, such as seasonal variations in energy consumption and instances where the recorded peak demand exceeded the utility's maximum allowable load. These insights provided crucial inputs for our project, helping us understand the dynamics of energy usage and the importance of managing peak demand. By leveraging the data from the electricity bills, we were able to identify potential cost-saving opportunities and develop effective demand management strategies to mitigate excessive demand during peak periods.

September 2022



KERALA STATE ELECTRICITY BOARD LIMITED

Office of the Special Officer(Revenue), Pattom,Thiruvananthapuram

DEMAND NOTICE FOR OCTOBER 2022

(As per CHAPTER VII OF KERALA ELETRICITY SUPPLY CODE -2014)

Con.	1355810004080	Bill Date	03-Oct-2022	Due Date	11-Oct-2022
Tariff	HT II (B) GENERAL			Last Date	26-Oct-2022

Bill No 210281110090 76 Ver 0

CD 1347850 BG 0

FEDERAL INSTITUTE OF SCIENCE &TECHNOLOGY
Federal Institute of Science & Technology, Mookkannoor,11/2,
HORMIS NAGAR,ANGAMALY,Mookkannoor,
ERNAKULAM ,683577
Mobile no--9562416405

SBI Virtual A/c No(IFS Code SBIN0070493)-KSEBHT33224209
Consumer GSTIN_ID- KSEB (LXGST ID=32AAECK2277N3Z1

LCN :32/4209

Arrears as on 31-Aug-2022				Date of Previous Reading	31-Aug-2022	Email: mail@fisat.ac.in	
Disputed	0	Undisputed	0	Date of Present Reading	30-Sep-2022	Supply Voltage	11 kV HT

Contract Demand(kVA)	75% of CD (KVA)	130% of CD (KVA)	Connected Load (KW)	Average MD (kVA)	Average Consumption (kWh)	PF	Billing Type	DPS
250.0	187.5	325.0	1140.15	221.27	50114	0.98	Section / Circle	Mookkannur / Perumbavoor

Reading Details of meter NET4209-Working (KVA,KWh,KVAh & KVArh) for 09-2022

1. Energy Consumption(KWh)

Zone	FR	IR	MF	Units
1	468068.00	458918.00	3.000	27447
2	166062.00	163333.00	3.000	8187
3	284790.00	280195.00	3.000	13785
			Total	49419

2. Energy Consumption(KVAh)

Zone	FR	IR	MF	Units
1	490190.00	480812.00	3.000	28734
2	169978.00	167224.00	3.000	8262
3	288269.00	283644.00	3.000	13875
			Total	50871

Ave PF=KWh/KVAh 0.97

3. Energy Consumption(KVArh) Lag and kVARh (Lead)

Zone	FR	IR	MF	Units	Zone	FR	IR	MF	Units
1	129583	127040	3.000	7629	1	257.00	228.00	3	87
2	32604.0	32272.0	3.000	996	2	45.00	44.00	3	3
3	36147.0	35762.0	3.000	1155	3	415.00	413.00	6	6
			Total kVArh(Lag)	9780				kVArh(Lead)	12

4. Demand (KVA)

	Readings	MF	Units
1	87.0	3.000	261.0
2	37.0	3.000	111.0
3	31.0	3.000	93.0
5.Factory Lighting			0.0
6.Colony Lighting			0.0
7.Generator			5780

INVOICE

	Unit	Rate	Amount (Rs)		Amount
1.Total Demand Charge				9.Other Charges	
a. Demand Charge - Normal	261.0	500.000	130500.00	Reconnection Fee	0.00
b. Demand Charge - Peak	0.0	500.000	0.00		
c. Demand Charge - Off peak	0.0	500.000	0.00		
d. Excess Demand Charge	11.0	250.000	2750.00		
e. Excess Demand Charge(Peak)	0.0	250.000	0.00		
f. Excess Demand Charge (Off	0.0	250.000	0.00		
Sub Total (a+b+c+d+e+f)			133250.00		
2.Total Energy Charges					
a. Energy charges - Normal	27447	7.80000	214086.60		
b. Energy charges - Peak	8187.0	11.7000	95787.90		
c. Energy charges - Off peak	13785.0	5.85000	80642.25		
Sub Total(a+b+c)			390516.75		
3.PF Incentive / Disincentive			-3906.17		
Total Energy Charge			386611.58		
4.Energy Charges on Lighting load					
a.Factory Lighting	0	0.2		10.Total(add 1 to 9)	560218.10
b.Colony Lighting	0	0.2	0.00	Plus/Minus (Round off)	-0.10
Sub Total(a+b)				UnDisputed Arr Amount	0.00
5.Electricity Duty	390517	0.100	39051.68	Less 1. Advance / Credit	
6.Ele. Surcharge	49419	0.025	1235.48	2. CD interest	0.00
7.Duty on self generated energy	5780	0.012	69.36	3. CD Refund	0.00
8.Penalty for non-segn. of light load				Net Payable	560218.00

(Rupees Five Lakh Sixty Thousand Two Hundred Eighteen Only)

E & O.E

Balance Advance at Credit, if any

Checked & Forwarded for approval 7/11/22 DENNY.V

Recommended STANLY E GEORGE

Please follow our official Facebook page fb.com/ksebl for information & announcements.

Instructions

SPECIAL OFFICER (REVENUE)

1355810004080 | 210281110090 76 | Rs.560218.00 | October 2022

FEDERAL INSTITUTE OF SCIENCE &TECHNOLOGY

Date

DD/Payment Instruction

Name of the Bank

Figure 3.2: KSEB Bill for the month September 2022

Derived Data:

Maximum Contract Demand = 250 kW

Demand Charges

Total Demand(Normal) = 261 kW

Charge = 500 Rs

Excess Demand = 11 kW

$$\text{Excess Demand Charge} = 250 \text{ Rs}$$

$$\text{Excess Demand Normal Charge} = (261 - 250) \times 500 = 5500 \text{ Rs}$$

$$\text{Excess Demand Penalty Charge} = 11 \times 250 = 2750 \text{ Rs}$$

$$\text{Total Excess charges} = 5500 + 2750 = 8250 \text{ Rs}$$

For the month of September, the institute had to pay a total of Rs 8250 in additional charges due to the exceeded load demand. During our interactions with the staff, we discovered that the additional demand of 11 kW occurred twice, each time lasting for a duration of 4-10 minutes. This additional demand caused the peak demand value to exceed the set limit.

October 2022

Derived Data:

$$\text{Maximum Contract Demand} = 250 \text{ kW}$$

Demand Charges

$$\text{Total Demand(Normal)} = 267 \text{ kW}$$

$$\text{Charge} = 500 \text{ Rs}$$

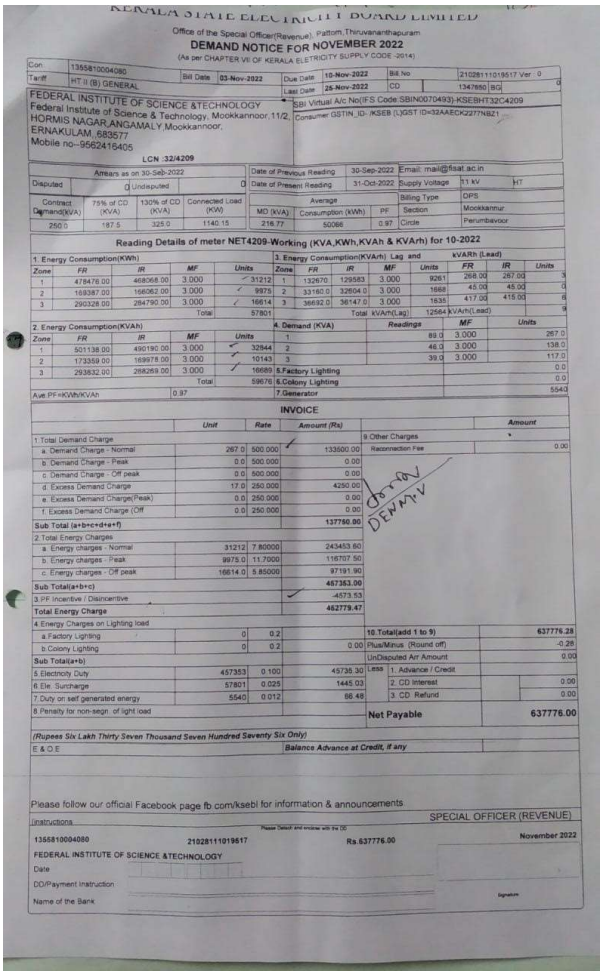
$$\text{Excess Demand} = 17 \text{ kW}$$

$$\text{Excess Demand Charge} = 250 \text{ Rs}$$

$$\text{Excess Demand Normal Charge} = (267 - 250) \times 500 = 8500 \text{ Rs}$$

$$\text{Excess Demand Penalty Charge} = 17 \times 250 = 4250 \text{ Rs}$$

$$\text{Total Excess charges} = 8500 + 4250 = 12750 \text{ Rs}$$



KERALA STATE ELECTRICITY BOARD LIMITED

Office of the Special Officer(Revenue), Pattom,Thiruvananthapuram

DEMAND NOTICE FOR NOVEMBER 2022

(As per CHAPTER VII OF KERALA ELETRICITY SUPPLY CODE -2014)

Con.	1355810004080	Bill Date	03-Nov-2022	Due Date	10-Nov-2022	Bill No	21028111019517 Ver : 0
Tariff	HT II (B) GENERAL			Last Date	25-Nov-2022	CD	1347650 BG

FEDERAL INSTITUTE OF SCIENCE &TECHNOLOGY
Federal Institute of Science & Technology, Mookkannoor,11/2, HORMIS NAGAR,ANGAMALY,Mookkannoor,
ERNAKULAM, 683577
Mobile no--9562416405

Sbi Virtual A/c No(IFS Code:SBIN0070493)-KSEBHT32C4209
Consumer GSTIN_ID- /KSEB (L)GST ID=32AAEOK2277NBZ1

LGN :32/4209

Arrears as on 30-Sep-2022					Date of Previous Reading	30-Sep-2022	Email: mail@fisat.ac.in				
Disputed	0	Undisputed	0		Date of Present Reading	31-Oct-2022	Supply Voltage	11 kV	HT		
Contract Demand(KVA)	75% of CD (KVA)	130% of CD (KVA)	Connected Load (KW)		Average			Billing Type	OPS		
					MD (KVA)	Consumption (kWh)	PF	Section	Mookkannur		
250.0	187.5	325.0	1140.15		216.77	50066	0.97	Circle	Perumbavoor		

Reading Details of meter NET4209-Working (KVA,KWh,KVAh & KVArh) for 10-2022

1. Energy Consumption(KWh)

Zone	FR	IR	MF	Units
1	478476.00	468068.00	3.000	31212
2	169387.00	166062.00	3.000	9975
3	290328.00	284790.00	3.000	16614
			Total	57801

2. Energy Consumption(KVAh)

Zone	FR	IR	MF	Units
1	501138.00	490190.00	3.000	32844
2	173359.00	169978.00	3.000	10143
3	293832.00	288269.00	3.000	16689
			Total	59676

Ave.PF=KWh/KVAh 0.97

3. Energy Consumption(KVArh) Lag and kVARh (Lead)

Zone	FR	IR	MF	Units	FR	IR	Units
1	132670	129583	3.000	9261	268.00	267.00	3
2	33160.0	32604.0	3.000	1668	45.00	45.00	0
3	36692.0	36147.0	3.000	1635	417.00	415.00	6
			Total	12564	kVArh(Lag)		

4. Demand (KVA)

	Readings	MF	Units
1	89.0	3.000	267.0
2	46.0	3.000	138.0
3	39.0	3.000	117.0

5.Factory Lighting 0.0
6.Colony Lighting 0.0
7.Generator 5543

INVOICE

	Unit	Rate	Amount (Rs)		Amount
1.Total Demand Charge				9.Other Charges	
a. Demand Charge - Normal	267.0	500.000	133500.00	Reconnection Fee	0.00
b. Demand Charge - Peak	0.0	500.000	0.00		
c. Demand Charge - Off peak	0.0	500.000	0.00		
d. Excess Demand Charge	17.0	250.000	4250.00		
e. Excess Demand Charge(Peak)	0.0	250.000	0.00		
f. Excess Demand Charge (Off	0.0	250.000	0.00		
Sub Total (a+b+c+d+e+f)			137750.00		
2.Total Energy Charges					
a. Energy charges - Normal	31212	7.80000	243453.60		
b. Energy charges - Peak	9975.0	11.7000	116707.50		
c. Energy charges - Off peak	16614.0	5.85000	97191.90		
Sub Total(a+b+c)			457353.00		
3.PF Incentive / Disincentive			-4573.53		
Total Energy Charge			452779.47		
4.Energy Charges on Lighting load					
a.Factory Lighting	0	0.2		10.Total(add 1 to 9)	637776.28
b.Colony Lighting	0	0.2	0.00	Plus/Minus (Round off)	-0.28
Sub Total(a+b)				UnDisputed Arr Amount	0.00
5.Electricity Duty	457353	0.100	45735.30	Less 1. Advance / Credit	
6.Ele. Surcharge	57801	0.025	1445.03	2. CD Interest	0.00
7.Duty on self generated energy	5540	0.012	66.48	3. CD Refund	0.00
8.Penalty for non-segn. of light load				**Net Payable**	**637776.00**

(Rupees Six Lakh Thirty Seven Thousand Seven Hundred Seventy Six Only)

E & O.E

Balance Advance at Credit, if any

Please follow our official Facebook page fb.com/ksebl for information & announcements

Instructions

SPECIAL OFFICER (REVENUE)

Please Detach and enclose with the DD

1355810004080 21028111019517 Rs.637776.00 November 2022

FEDERAL INSTITUTE OF SCIENCE &TECHNOLOGY

Date

DD/Payment Instruction

Name of the Bank

Signature

Figure 3.3: KSEB Bill for the month of October 2022

During the month of October, the institute incurred additional charges amounting to Rs 12,750 due to the exceeded load demand. Through interactions with the staff, it was revealed that the additional demand of 17 kW occurred once, lasting for a duration of 4-10 minutes. This temporary surge in demand resulted in the peak demand value surpassing the set limit, leading to the imposition of penalties and additional charges.

Based on data obtained from weather forecasting, it was observed that the months under assessment were predominantly cloudy. However, during interactions with the staff, it was highlighted that peak loads occur more frequently on hot summer days. This phenomenon can be attributed not only to the prevailing climate conditions but also to the academic schedule of the university. It was noted that the months of March to May, which coincide with the summer season, witness a higher frequency

of continuous and frequent lab sessions. These sessions contribute to increased load demands and consequently impact the overall load profile of the electrical system.

Chapter 4

System Description

4.1 Technology Stack

4.1.1 Micro grids & Smart Grids

For decades, most countries rely on large grids to fulfill electricity needs. However, with the rapid development of technology, there is a significant contribution from smaller grids all over the world in recent years, especially to fulfill the demands in remote areas.

Smaller grids comprise of mini-grids, microgrids and nano grids. Mini-grids are utility grid that is small in capacity and geography or could be a group of some microgrids. The microgrid system is a small power supply system that consists of loads and distributed energy resources (DER), such as renewable energy (RE) sources, cogeneration, combined heat and power (CHP) generation, fuel cell and energy storage systems. Nanogrids are single domains of power; single physical layers of power distribution, reliability, quality, capacity, price, and administration.

4.1.2 Internet of Things

The Internet of Things makes everyday devices smarter by enabling them to send data over the Internet and communicate with people and other IoT-enabled devices. The IoT can be found in an array of devices, industries, and settings.

4.1.3 Micro-controllers

A microcomputer made on a single semiconductor chip is called a single-chip microcomputer. Since single-chip microcomputers are generally used in control applications, they are also called microcontrollers.

The microcontroller contains all essential components of a microcomputer such as CPU, RAM, ROM/EPROM, I/O lines etc. Some single-chip microcontrollers contain devices to perform specific functions such as DMA channels, A/D converter, serial port, pulse width modulation, etc.

4.1.4 Arduino and Sensors Interface

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. The sensors are defined as a machine, modules, or devices that detect changes in the environment. The sensors transfer those changes to electronic devices in the form of a signal.

A sensor and electronic devices always work together. The output signal is easily readable by humans.

4.1.5 Web Development

Web development is the work involved in developing a website for the Internet (World Wide Web) or an intranet (a private network). Web development can range from developing a simple single static page of plain text to complex web applications, electronic businesses, and social network services. A more comprehensive list of tasks to which Web development commonly refers may include Web engineering, Web design, Web content development, client liaison, client-side/server-side scripting, Web server and network security configuration, and e-commerce development. The User communicates and interacts with the whole system using the web.

4.2 Design

The proposed system has different levels of installation. The modular design enables the system to be installed with the existing grid. Before installation, the grid is analyzed and data is collected. The current system structure is important. Voltage, Power specifications, and loads connected are to be noted. Each load section is categorized according to its priority and load size. Loads categories are

TIME	Top priority	Medium priority	Low priority
6:00AM-6:00PM	Electrical Workshops	Office	Classrooms
	Civil & Mechanical workshops	Elevator	Staff rooms
	Auditorium	Computer lab	
	Server room	Canteen	
	Main seminar Hall		
	Labs		
6:00pm-6:00am	Hostel	Emergency lighting inside the campus	Water cooler
	Mess		Elevator
	Street lights		
	Main gate lights		
	Corridor lighting		

Table 4.1: Load Priority Categorisation

The system monitors each of the loads and sources, and the flow of power is adjusted according to the value of the maximum load possible and given priority. The detailed workflow is described in the next section.

4.2.1 ESAM Design

The complete system of ESAM consists of the following sub-systems or devices.

4.2.2 Sensor & Control Unit

This is the base module that basically completes the sensing and control unit. Sensor & Control Unit is to be installed at every load distribution point. A dedicated unit is required for each load section for control. Each device will be designed according to load and input.

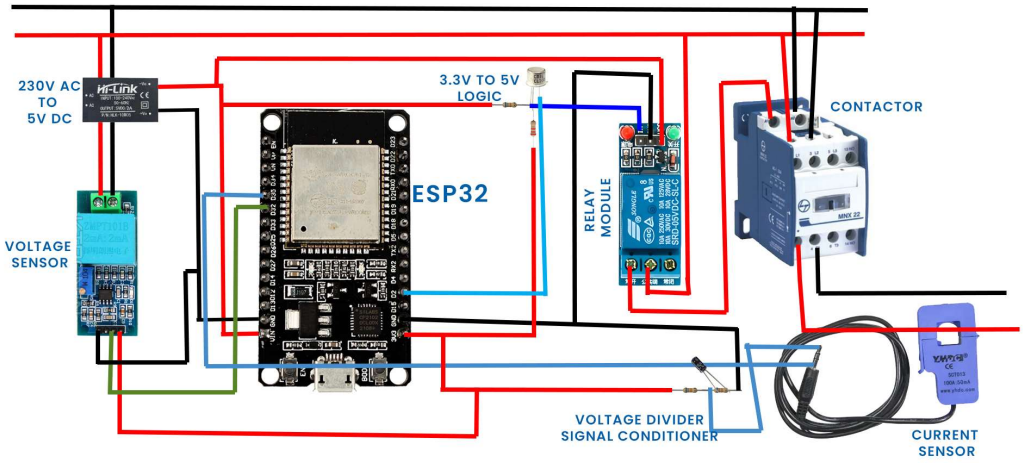


Figure 4.1: Circuit Diagram of Sensor & Control Unit(Basic Design sample)

The base design consists of a microcontroller ESP32 development board with connectivity along with current and voltage sensors and a control unit(Relay + Contactor). A design of a very basic device is shown in Figure 3.1. Sensor values and calculated power values and current control status from this device are communicated with the Intel Edison Server.

4.2.3 Intel Edison Server

Intel Edison Server is like an electronic server that gathers all information from every sensor, controller, Flask server, and Web UI and makes intelligent decisions As the name suggests the main computing device is an Intel Edison board connected to all the devices via a Wireless network.

4.2.4 Hub to Server Communication

Intel Edison Server needs to communicate with the flask server to get User inputs and to the monitoring values to the user. Since Intel Edison-to-flask server communication is important, which is a computer or system that stores and processes the data received from the intel edison server. This communication can be established using various technologies, such as WiFi, cellular, Bluetooth, or Ethernet, and can be secured using protocols such as SSL/TLS or IPSec. The specific protocol and technology used for Intel Edison-to-flask server communication will depend on the requirements of the IoT system and the devices connected to it.

4.2.5 Web User Interface

A user interface is a bridge between the user and the whole system. A web UI enables the adaptability of the system. The UI runs on a python flask server that acts as the backend to the system. The website is developed using HTML and CSS stacks. This receives the calculated values from the server and displays them in an understandable and aesthetic manner to the user. The UI includes basic monitoring values like Voltage state, Current usage, and power usage as in total and separate loads also the energy. The load priority can also be adjusted according to present situations through the UI.

4.3 Components details

4.3.1 Intel Edison

Intel Edison is a small, low-power computing module developed by Intel. It was designed for the Internet of Things (IoT) and wearable computing applications. The module integrates a dual-core Intel Atom processor, memory, storage, and Wi-Fi, and Bluetooth connectivity into a small form factor.
Key features of the Intel Edison module include:

1. Processor: It features a dual-core Intel Atom SoC (System on a Chip) with Intel Hyper-Threading Technology, providing efficient and powerful computing capabilities.

2. Memory and Storage: The module comes with 1 GB DDR3 RAM and 4 GB eMMC storage, providing enough resources for running embedded applications.

3. Connectivity: Intel Edison includes built-in Wi-Fi (802.11n) and Bluetooth 4.0 support, allowing seamless wireless connectivity with other devices and the internet.

4. Expansion: It features a 70-pin connector, which allows for various expansion options. You can connect additional modules, sensors, and peripheral devices to extend the functionality of the Intel Edison module.

5. Development Environment: Intel provides a software development kit (SDK) and integrated development environment (IDE) for programming the Intel Edison module. It supports multiple programming languages, including C/C++, JavaScript, and Python.

6. Operating System: The Intel Edison module supports various operating systems, including Linux-based Yocto Project, which is a lightweight and customizable Linux distribution specifically designed for embedded devices.

7. Power Efficiency: Intel Edison is designed for low-power applications, making it suitable for battery-powered and portable devices.

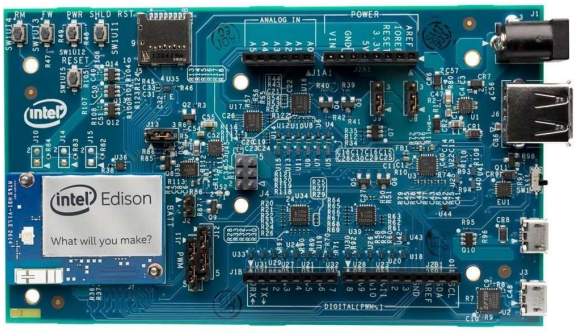


Figure 4.2: Intel Edison board with Arduino Kit

Intel Edison has found applications in various IoT projects, wearable devices, and smart home automation. It offers a compact and versatile platform for developers and makers to build innovative connected solutions. However, it's worth noting that Intel has discontinued the Edison module, and support and availability may vary.

4.3.2 ESP32

The ESP32 is a highly popular and versatile microcontroller module designed for Internet of Things (IoT) applications. Developed by Espressif Systems, it has gained significant traction among makers, hobbyists, and professionals alike.

At the heart of the ESP32 is a powerful dual-core Tensilica LX6 microcontroller, capable of running at clock speeds of up to 240 MHz. This processing power enables ESP32 to handle complex tasks and applications with ease. One of the standout features of the ESP32 is its built-in Wi-Fi and Bluetooth connectivity. With support for 2.4 GHz Wi-Fi (802.11 b/g/n) and both Bluetooth Classic and Bluetooth Low Energy (BLE), the ESP32 can seamlessly connect to networks, other devices, and the internet, opening up a wide range of possibilities for IoT projects. The ESP32 module provides ample memory and storage options, typically offering 520 KB of SRAM and various flash memory configurations ranging from 4 MB to 16 MB. This allows for storing program code, data, and web content, providing flexibility for different project requirements.

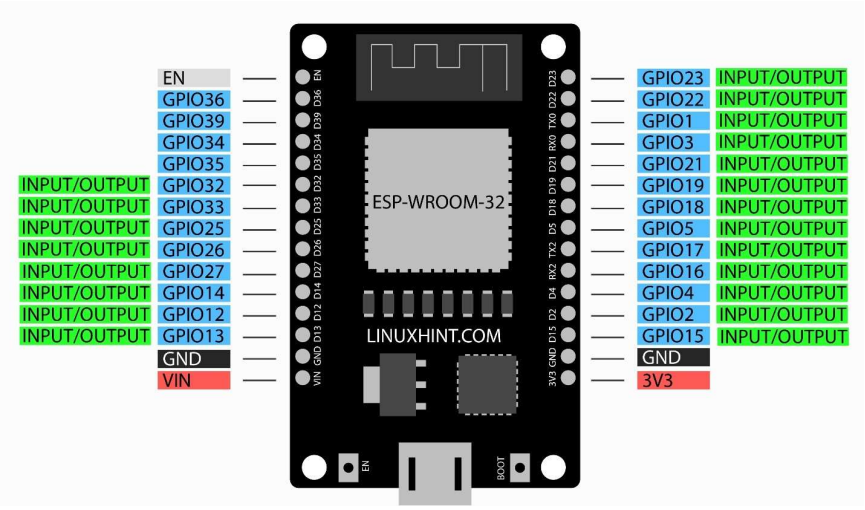


Figure 4.3: ESP32 Dev-board

With its extensive GPIO pin count, the ESP32 offers abundant opportunities for connecting to sensors, actuators, and other peripheral devices. It also includes interfaces such as I2C, SPI, UART, and ADC, simplifying communication with external components and expanding the capabilities of the module.

Developers have multiple options for programming the ESP32. It is well-supported

by the Arduino IDE, making it accessible to a broad community of developers. Additionally, Espressif provides the ESP-IDF, an official development framework that offers more advanced features and low-level access to the module.

Power efficiency is a significant consideration for IoT devices, and the ESP32 addresses this with various power-saving modes and features. This allows for optimized energy usage, making it suitable for battery-powered applications and reducing overall power consumption.

The ESP32 finds applications in a wide range of projects, including home automation systems, wearable devices, sensor networks, industrial automation, and more. Its versatility, connectivity options, processing power, and ease of development make it a go-to choice for many IoT enthusiasts and professionals.

Overall, the ESP32 has established itself as a reliable and capable microcontroller module, empowering developers to create innovative and connected solutions for the rapidly expanding world of IoT.

4.3.3 L&T (Larsen & Toubro) MNX 40 Contactor

L&T (Larsen & Toubro) MNX 40 Power Contactors are electrical devices designed for controlling and switching electrical power in industrial applications. Specifically, these contactors are rated for 415V AC voltage and feature a 3-pole configuration.

Key features of the L&T MNX 40 Power Contactors include:

1. Voltage Rating: The contactors are designed to handle a voltage of 415V AC, making them suitable for applications that require medium to high voltage control.

2. Pole Configuration: The MNX 40 Power Contactors have a 3-pole configuration. This means they have three separate contacts, enabling control of three different electrical circuits or phases.

3. Current Rating: The contactors are typically rated for a specific current, which determines their capacity to handle electrical load. The exact current rating of the MNX 40 contactors may vary based on specific models or variants.

4. Electrical Switching: These power contactors are capable of reliably switching electrical power, allowing for the control and operation of motors, pumps,

lighting systems, and other industrial equipment.

5. Coil Control: The MNX 40 contactors are typically controlled by an electromagnetic coil. When the coil is energized, it creates a magnetic field that pulls the contacts together, closing the electrical circuit. When the coil is de-energized, the contacts separate, breaking the circuit and disconnecting the power.

6. Durability and Reliability: L&T is a reputable brand known for producing high-quality electrical products. The MNX 40 Power Contactors are designed to be durable and reliable, capable of withstanding the demands of industrial environments.

7. Mounting Options: These contactors often come with various mounting options, such as DIN rail mounting or panel mounting, providing flexibility in installation and integration into electrical control systems.

Figure 4.4: L&T (Larsen & Toubro) MNX 40 Power Contactors 415V AC, 3 Pole

L&T MNX 40 Power Contactors find applications in a wide range of industrial sectors, including manufacturing, automation, HVAC (heating, ventilation, and air conditioning), and more. They are commonly used in motor control panels, power distribution systems, and other electrical control applications where reliable switching and control of electrical power is required.

Overall, the L&T MNX 40 Power Contactors offer a reliable and efficient solution for controlling and switching electrical power in industrial settings, with their 415V AC voltage rating and 3-pole configuration providing versatility and compatibility with various applications.

4.3.4 YHDC SCT-013 100A

The YHDC SCT-013 100A is a current transformer sensor used for measuring AC currents up to 100 Amperes. With its non-intrusive split-core design, it can easily clamp around the conductor carrying the current without interrupting the circuit. The sensor provides an output signal proportional to the measured current, allowing for interfacing with microcontrollers or data loggers. It offers good accuracy and ensures electrical safety during measurements. The YHDC SCT-013 100A is commonly used in energy monitoring systems, power quality analysis, and industrial applications where non-intrusive and accurate current measurements are required.

Figure 4.5: YHDC SCT-013-000 100A Non-Invasive Split Core Current Transformer

4.3.5 Hi-Link(AC to DC converter)

The HLK-PM12 is a small, plastic-enclosed power supply module that converts alternating current (AC) from a wall outlet into direct current (DC) at a voltage of 12V and a power rating of 3W. It is designed to be mounted on a printed circuit board (PCB) and can be used in a variety of applications, including home automation, communication equipment, instrumentation, and more. One of the main benefits of this module is its high efficiency and reliability, as well as its ability to maintain a stable output voltage despite fluctuations in the input voltage.

Figure 4.6: Hi-Link HLK PM12 12V/3W Switch Power Supply Module

4.3.6 ZMPT101B (Single Phase)

ZMPT101B AC Voltage Sensor is the best for the purpose of the DIY project, where we need to measure the accurate AC voltage with a voltage transformer. This is an ideal choice to measure the AC voltage using Arduino/ESP8266/Raspberry Pi like an opensource platform. In many electrical projects, engineer directly deals with measurements with few basic requirements like High galvanic isolation, Wide Range, High accuracy, Good Consistency.

Onboard precision miniature voltage transformer, The active phase AC output voltage transformer module. Onboard precision op-amp circuit, the signal sampling and appropriate compensation for precise functions. Modules can be measured within 250V AC voltage, the corresponding analog output can be adjusted. It is brand new, good quality high performance.

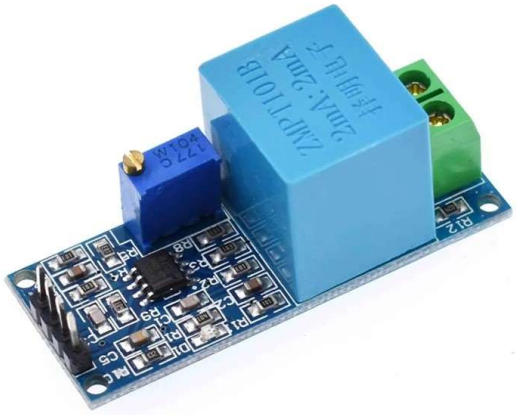


Figure 4.7: AC Voltage Sensor Module ZMPT101B (Single Phase)

4.3.7 Relay Module 5V, Single channel

The 1 Channel 5V Relay Module with High Level DC Control is a compact electronic device used to control high-power electrical circuits using low-power control signals. Operating on a 5V DC power supply, this module can effectively control a single circuit or device.

Based on the principle of electromechanical switching, the module consists of a relay and a driver circuit. When a high-level control signal is applied, the relay is energized, and its contacts close, enabling the activation or deactivation of the high-power circuit. This relay module finds applications in home automation, robotics, and industrial control systems, where the need arises to control devices or circuits operating at higher voltages or currents. By providing isolation between the low-power control circuit and the high-power circuit, it ensures safe and reliable switching with optimal protection. The 1 Channel 5V Relay Module with High Level DC Control offers a convenient and efficient solution for controlling high-power electrical circuits using low-level control signals. Its small size and easy interfacing make it suitable for a wide range of electronic projects requiring reliable and safe control of high-power devices.

Figure 4.8: 1 Channel 5V Relay Module, High Level DC Control 250V 2A with Resistive Fuse

4.4 Software(Programming languages and IDEs)

As the system in total includes IoT, cloud computing and web UI programming and algorithms play an important role in the efficiency of the system.

4.4.1 PlarformIO & ESP32 Programming

PlatformIO is an open-source ecosystem for embedded development that provides a unified platform for programming microcontrollers, including the ESP32. It simplifies the process of setting up development environments, managing libraries, and building firmware for various microcontroller platforms.

With PlatformIO, you can write code for the ESP32 microcontroller using popular programming languages such as C and C++. It provides a user-friendly interface and integrates with code editors like Visual Studio Code, making it easy to write, edit, and organize your code.

One of the key features of PlatformIO is its built-in library manager. It offers a centralized library registry where you can search for and install libraries for your ESP32 projects. The library manager takes care of resolving library dependencies and provides version control, making it effortless to include external libraries in your code.

PlatformIO also simplifies the process of building firmware for the ESP32. It handles the compilation, linking, and generating of the firmware binary, allowing

you to focus on writing the code logic without worrying about the underlying build process.

Additionally, PlatformIO provides tools for uploading the firmware to the ESP32 board and monitoring the serial output for debugging purposes. It streamlines the development workflow by offering a seamless experience from code writing to deployment and debugging.

Overall, PlatformIO is a powerful and convenient ecosystem for ESP32 programming, offering a range of features that facilitate efficient development, library management, and firmware building for embedded applications.

4.4.2 Embedded C++

Embedded C++ refers to the use of the C++ programming language in embedded systems development. Embedded systems are specialized computer systems with limited resources. C++ extends the C language with additional features such as object-oriented programming, enabling developers to write more structured and reusable code. Classes, inheritance, and polymorphism facilitate code organization and code reuse in embedded systems. However, resource constraints must be carefully considered, and certain C++ features like exceptions and dynamic memory allocation may need to be used judiciously. Many embedded systems development tools and frameworks support C++ programming, providing libraries and middleware for seamless integration with hardware drivers and real-time operating systems. Embedded C++ combines the benefits of C++'s expressive power with the efficiency and constraints of embedded systems, enabling the development of sophisticated and maintainable embedded applications.

4.4.3 Python & Python Flask

Python is a popular and easy-to-learn programming language known for its simplicity and readability. Flask, a lightweight web framework in Python, allows developers to quickly build web applications. With Flask, developers can leverage Python's extensive library ecosystem and concise syntax to create dynamic web applications. Flask provides features such as URL routing, template rendering, and form handling, making it suitable for small-scale websites, APIs, and prototypes. The combination of

Python and Flask offers a flexible and efficient solution for web development, enabling developers to build robust web applications with ease.

4.4.4 Web(HTML, CSS with Bootstrap)

HTML (HyperText Markup Language) is a programming language used to create and structure the content of a webpage. It consists of a series of tags that define the structure and content of the webpage, including headings, paragraphs, lists, and links.

CSS (Cascading Style Sheets) is a programming language used to style and layout the content of a webpage. It allows you to specify the appearance of elements on the page, such as the font, color, and size of text, the background color of a box, and the layout of elements on the page.

Bootstrap is a popular front-end framework for developing responsive, mobile-first websites. It includes a set of CSS and HTML files that provide a starting point for building webpages with a consistent look and feel, as well as a set of customizable components that can be used to build the user interface of a webpage. Bootstrap is designed to make it easy to create websites that look good on a wide range of devices, including smartphones, tablets, and desktop computers.

4.4.5 MySQL Database

MySQL is an open-source relational database management system (RDBMS) known for its simplicity, reliability, and performance. It organizes data into tables consisting of rows and columns, with each table representing a specific entity or concept. Using Structured Query Language (SQL), users can create tables, insert data, query data, update records, and delete data. MySQL supports various data types such as numeric, string, and date/time types. It provides indexing capabilities for faster data retrieval and supports transactions following the ACID properties to ensure data integrity. Security features include user authentication, access control, and data encryption. MySQL is scalable and can handle large data volumes and high traffic loads using replication, clustering, and partitioning techniques. It integrates with popular programming languages and frameworks using connectors or APIs. MySQL offers

command-line tools and graphical user interfaces for database management and administration. With a vibrant community and extensive documentation, MySQL is widely used for a range of applications from small-scale projects to enterprise-level systems.

4.5 Work Flow of designed prototype

The detailed structure about the system was mentioned in previous section. The sensing and control device is the base unit to be installed at each load point. It consists of a current sensor(YHDC 013-100A) and a voltage sensor(ZMPT101B) connected to power coming to the load, the values from these sensors are read by the microcontroller connected(ESP32). The device also includes a control system which has two major components a 5v relay and a contactor. The relay module controls a 230V AC circuit from the AC bus which is connected through the contactor. This circuit controls the Contactor which is in series with the supply and load. The load is controlled using this contactor circuit. The microcontroller calculates voltage, current, and power from the sensor values and communicates the values to the Intel Edison server. The Intel Edison server receives input from these microcontrollers and Websites to prioritize loads and analyze the system. Considering a time when the load crosses the threshold value set, the server gives control signals to each microcontroller to cut the low-priority loads on a capacity basis, until the system is under control, if not yet controlled medium loads are also removed. When the system is under threshold value, if the system is not controlled manually, it will wait until a certain period and keep analyzing for the decrease in the top and medium priority loads to be normal again. Finally when the correct load is achieved the system will automatically bring back the low-priority loads live. The Intel Edison server communicates to the Web through flask server communication, and the web UI displays the analysis, current condition, etc...

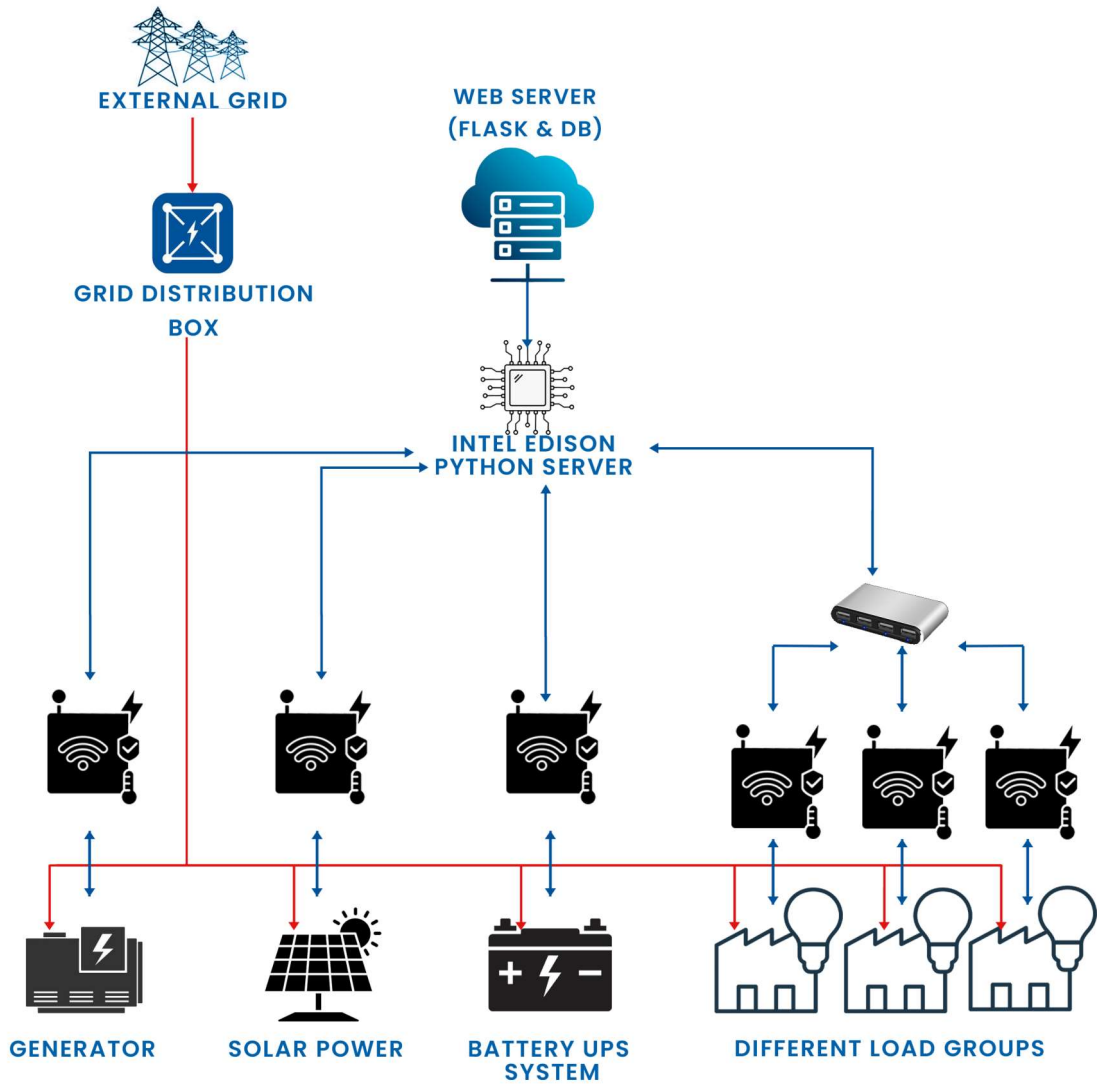


Figure 4.9: System design structure

4.6 Additional Circuits Used

4.6.1 3.3V to 5V Logic level converter

ESP32 works on 3.3V logic, means for high signal it give a 3.3V signal and for a low signal it gives 0V with reference to ground pin. But the relay modules we use works on 5V logic, means to activate the relay we require 5V and to turn off, it require 0V with reference to ground. So a logic level converter is designed using an npn transistor. The basic idea is to use the transistor as a switch to control the voltage level.

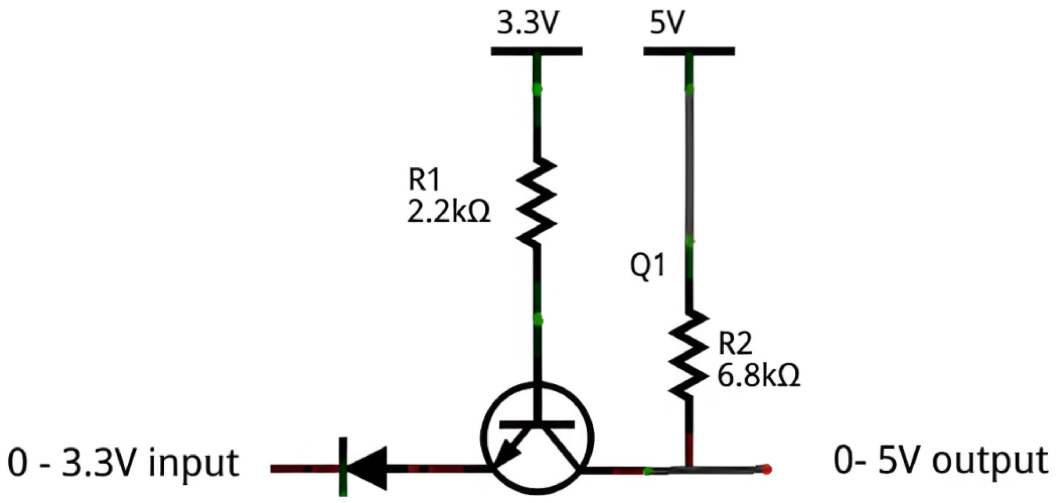


Figure 4.10: 3.3V logic to 5V logic circuit using an NPN transistor

Here's how the circuit works step-by-step:

1. The Signal_IN represents your 3.3V logic signal that you want to convert to 5V logic.

2. R1 is a resistor connected between the Signal_IN and the transistor's base. This resistor acts as a current limiter to protect the transistor from excessive current.

3. The base of the NPN transistor is biased using the voltage divider formed by R1 and R2. R2 is connected between the base and the +5V supply. The voltage at the base will be around 0.7V (the forward voltage drop across the base-emitter junction) lower than the voltage at the emitter.

4. When Signal_IN is at a logic high level (3.3V), the current flows through R1, forward biases the base-emitter junction of the transistor, and allows current to flow from the collector to the emitter.

5. The voltage at the collector will be approximately Vout, which is close to the +5V supply voltage.

6. When Signal_IN is at a logic low level (0V or ground), no current flows through R1, and the transistor is in the cutoff region. The collector-emitter path is effectively open, resulting in Vout being pulled to GND.

By using this circuit, the transistor acts as a level shifter, converting the 3.3V logic signal to a 5V logic signal at the output (Vout) based on the input logic levels.

4.6.2 Signal Conditioner

The SCT013 CT sensor is a non-invasive current sensor that measures the alternating current (AC) flowing through a conductor without directly contacting it. It typically produces a voltage output that is proportional to the measured current. However, the sensor's voltage output might have a high level of noise or a varying DC offset, depending on the specific application and environmental factors.

The purpose of the signal conditioning circuit you described is to filter out unwanted noise and adjust the voltage level to a more suitable range for further processing or measurement.

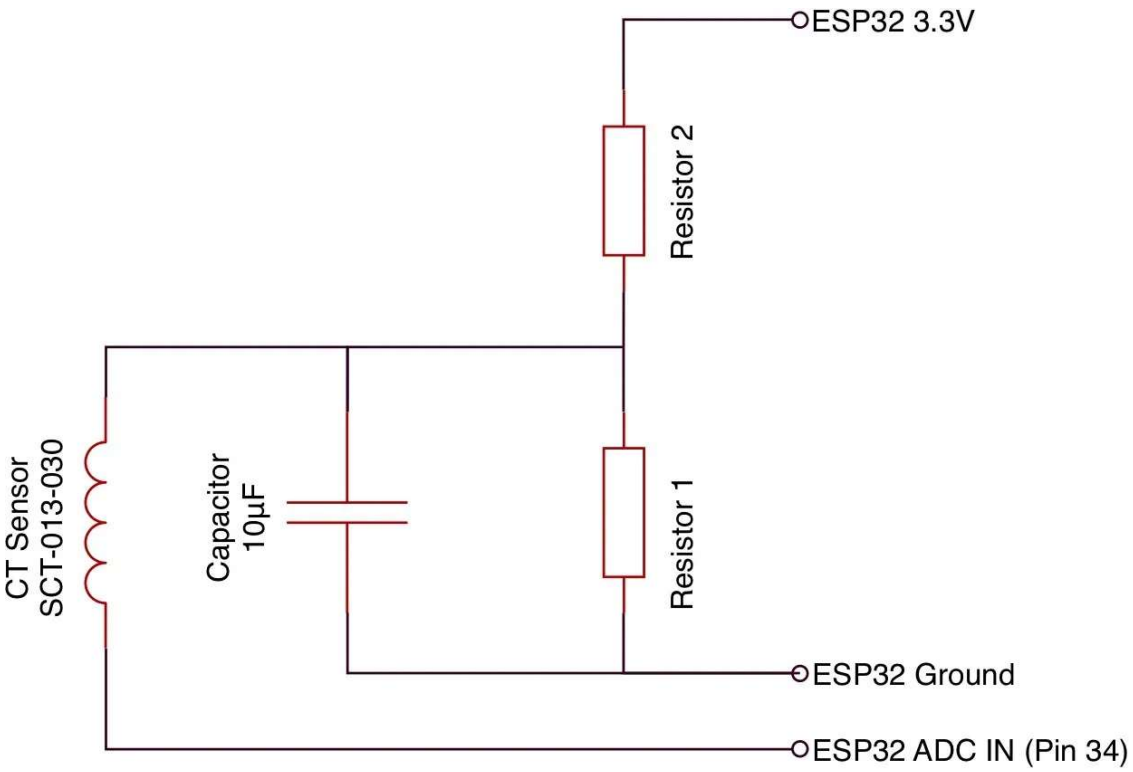


Figure 4.11: Voltage divider signal conditioner

1. Sensor_OUT represents the voltage output of the SCT013 CT sensor, which may have noise or DC offset.

2. R1 and R2 form a voltage divider. The purpose of the voltage divider is to

reduce the voltage level of Sensor_OUT and scale it down to a more suitable range. The values of R1 and R2 are selected based on the desired voltage scaling.

3. The 10uF capacitor (C) is connected in parallel with R2. This capacitor acts as a low-pass filter, allowing low-frequency AC signals to pass while attenuating higher-frequency noise components.

4. The combination of R2 and the capacitor forms an RC filter. The capacitor charges and discharges through R2, smoothing out the voltage variations and reducing high-frequency noise.

5. The filtered voltage at the junction of R1 and R2 (output of the voltage divider) can be connected to further processing circuits or measurement devices to obtain a cleaner and more stable representation of the original current signal.

By using this signal conditioning circuit, the voltage output of the SCT013 CT sensor is adjusted, filtered, and scaled to provide a more reliable and usable signal for subsequent processing or measurements. The capacitor across R2 helps remove noise and stabilize the output voltage, improving the overall performance of the system.

Chapter 5

Simulation Analysis

5.1 Simulating the system

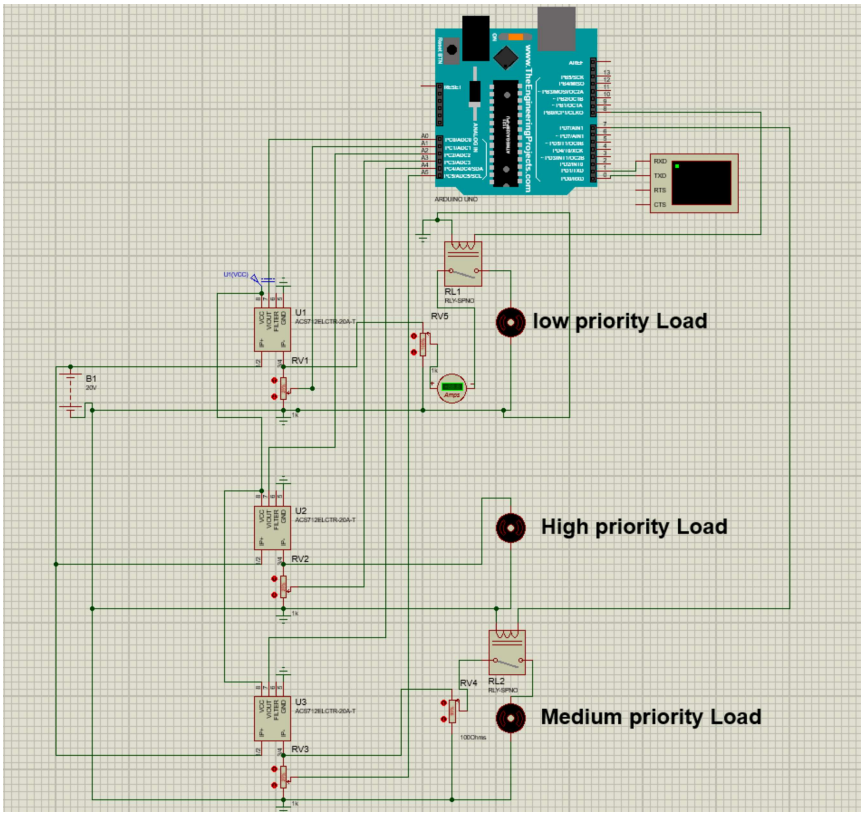


Figure 5.1: Schematic Diagram Proteus Model

The system is designed to control AC systems, but due to limitations in simulations the entire system is converted to DC system for simulation. But it does not affect

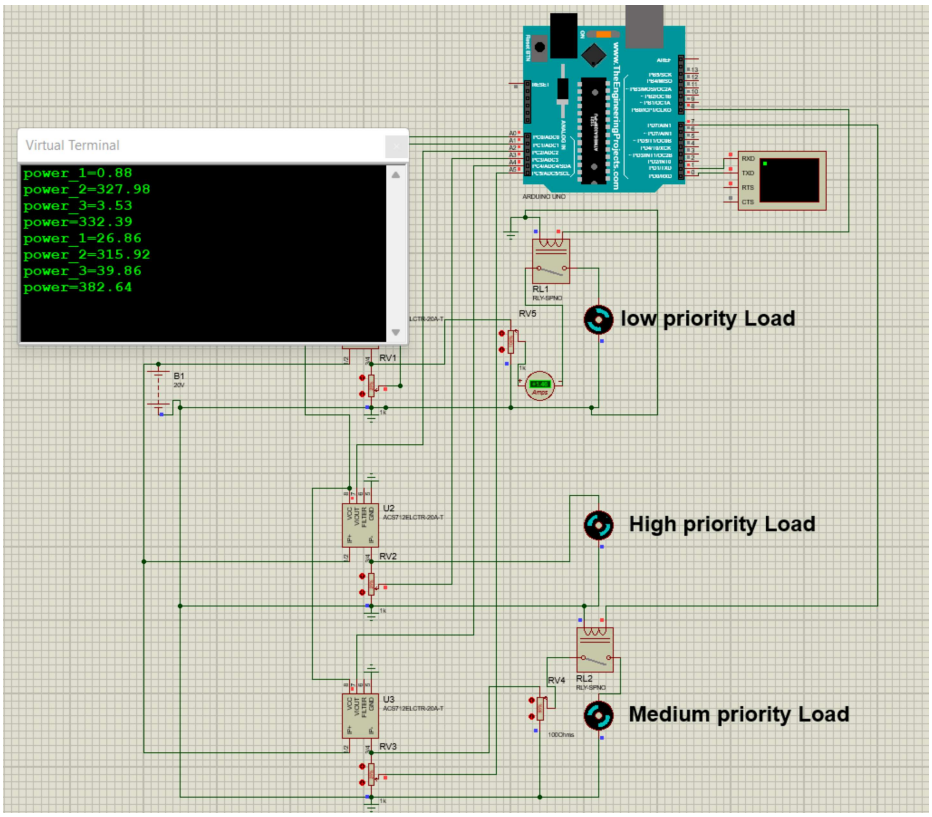


Figure 5.2: Simulation in Proteus

the algorithm or work flow. The whole system is constructed in Proteus design suit. The system includes a 20V source and 3 loads categorised in top, medium and low priorities. the system is controlled with an arduino Uno module. The schematic of the system is shown in figure 4.1 and 4.2.

Arduino is the master control which follows a simple algorithm to control and monitor the system. The arduino program is given below

```
1          #pragma GCC push_options
2          #pragma GCC optimize ("Os")
3          #pragma GCC pop_options
4          // Peripheral Constructors
5          void peripheral_setup () {
6          }
7          void peripheral_loop() {
8          }
9
10         float volt_1=0.0;
```

```
11          float volt_2=0.0;
12          float volt_3=0.0;
13
14          float current_1=0.0;
15          float current_2=0.0;
16          float current_3=0.0;
17
18          float power_1=0.0;
19          float power_2=0.0;
20          float power_3=0.0;
21
22          float total_power=0.0;
23          float load_max=400.0;
24          float load_offset=50.0;
25
26          void setup () {
27          peripheral_setup();
28             pinMode(A0, INPUT);
29             pinMode(A1, INPUT);
30             pinMode(A2, INPUT);
31             pinMode(A3, INPUT);
32             pinMode(A4, INPUT);
33             pinMode(A5, INPUT);
34             Serial.begin(9600);
35             pinMode(7,OUTPUT);
36             pinMode(8,OUTPUT);
37             digitalWrite(7,HIGH);
38             digitalWrite(8,HIGH);
39          }
40
41
42          void loop() {
43          peripheral_loop();
44             power_calc();
```

```
45            if(total_power>=load_max){
46               Serial.println("low priority cut");
47               digitalWrite(8,LOW);
48               power_calc();
49               if(total_power>=load_max){
50                  Serial.println("medium priority cut");
51                  digitalWrite(7, LOW);
52               }
53               while(total_power > (load_max-load_offset)){
54                  Serial.println("power_save_mode");
55                  power_calc();
56                  delay(1000);
57               }
58               Serial.println("back to safe mode");
59               digitalWrite(7,HIGH);
60               digitalWrite(8,HIGH);
61            }
62            delay(1000);
63         }
64         float power_calc(){
65            volt_1=(analogRead(A1) * (5.0 / 1023.0))*4.0;
66            volt_2=(analogRead(A3) * (5.0 / 1023.0))*4.0;
67            volt_3=(analogRead(A5) * (5.0 / 1023.0))*4.0;
68
69            current_1=(analogRead(A0) - 512)/20.6;
70            current_2=(analogRead(A2) - 512)/20.6;
71            current_3=(analogRead(A4) - 512)/20.6;
72
73            power_1=volt_1*current_1;
74            power_2=volt_2*current_2;
75            power_3=volt_3*current_3;
76            Serial.print("power_1=");
77            Serial.println(power_1);
78            Serial.print("power_2=");
```

```
79            Serial.println(power_2);
80            Serial.print("power_3=");
81            Serial.println(power_3);
82
83            total_power= power_1+power_2+power_3;
84            Serial.print("power=");
85            Serial.println(total_power);
86            delay(100);
87        }
```

5.2 Result of simulation

The simulation with the DC system was successful, monitoring voltage and current in real-time and calculating power and load was efficient and correct. Also, the threshold load limit was maintained without crossing by controlling the relays installed.

Chapter 6

Program & Server Architecture

6.1 Python flask server

The Python flask server implements a web application using the Flask framework in Python. The application interacts with a MySQL database and communicates with an external server using the Flask-SocketIO extension.

```
1   from flask import Flask, render_template,
        request, redirect, url_for, Response
2   from flask_socketio import SocketIO, emit
3   import mysql.connector
4   import random, json
5   from threading import Thread
6   import time
7   from flask import current_app
8   import requests
9   app = Flask(__name__)
10  app.config['SECRET_KEY'] = 'secret!'
11  socketio = SocketIO(app)
12
13  # Database connection details
14  db_host = 'localhost'
15  db_user = 'root'
16  db_password = 'Gsw@1924'
```

```
17              db_name = 'esamproject'
18
19              data = {'voltage1': 0, 'current1': 0, 'energy1':
                    0, 'voltage2': 0, 'current2': 0, 'energy2':
                   0, 'voltage3': 0, 'current3': 0, 'energy3':
                   0}
20              data_1 = {'voltage1': 0, 'current1': 0, 'energy1
                   ': 0, 'voltage2': 0, 'current2': 0, 'energy2'
                   : 0, 'voltage3': 0, 'current3': 0, 'energy3':
                    0, 'total_energy':0}
21              state={'button_id':'load1', 'status':'OFF'}
22              mode={'mode':'Manual'}
23              # Function to validate user credentials
24              def validate_user(username, password):
25                  # Create a database connection
26                  conn = mysql.connector.connect(
27                      host=db_host,
28                      user=db_user,
29                      password=db_password,
30                      database=db_name
31                  )
32
33                  # Create a cursor object to execute SQL
                       queries
34                  cursor = conn.cursor()
35
36                  # Execute the query to check if the user
                       exists
37                  query = "SELECT * FROM users WHERE username
                       = %s AND password = %s"
38                  cursor.execute(query, (username, password))
39
40                  # Fetch the result
41                  result = cursor.fetchone()
```

```
42
43                  # Close the cursor and database connection
44                  cursor.close()
45                  conn.close()
46
47                  # Return True if user exists, False
                        otherwise
48                  if result:
49                      return True
50                  else:
51                      return False
52
53              @app.route('/')
54              def login():
55                  return render_template('login.html')
56
57              # Route to handle the login page
58              @app.route('/login', methods=['GET', 'POST'])
59              def handle_login():
60                  if request.method == 'POST':
61                      # Get the username and password from the
                            form
62                      username = request.form['username']
63                      password = request.form['password']
64
65                      # Validate the user credentials
66                      if validate_user(username, password):
67                          return redirect(url_for('home'))
68                      else:
69                          return 'Invalid Credentials'
70                  else:
71                      return render_template('login.html')
72
73              @app.route('/home')
```

```
74              def home():
75                  return render_template('home.html')
76
77
78              @app.route('/receive_data', methods=['POST'])
79              def receive_data():
80                  global data
81                  data = request.get_json()
82                  print(data)  # Do something with the
                        received data
83                  # Extract individual values
84                  return 'Data received successfully'
85
86              def send_data():
87                  # Send random voltage, current, and energy
                        values
88                  global data
89                  voltage1 = data['voltage1']
90                  current1 = data['current1']
91                  energy1 = data['energy1']
92                  voltage2 = data['voltage2']
93                  current2 = data['current2']
94                  energy2 = data['energy2']
95                  voltage3 = data['voltage3']
96                  current3 = data['current3']
97                  energy3 = data['energy3']
98                  total_energy = energy1+ energy2 + energy3
99                  total_energy = round(total_energy, 3)
100
101                 conn = mysql.connector.connect(
102                     host=db_host,
103                     user=db_user,
104                     password=db_password,
105                     database=db_name
```

```
106                 )
107
108                 # Create a cursor object to execute SQL
                        queries
109                 cursor = conn.cursor()
110
111                 # Execute the query to check if the user
                        exists
112                 query = "INSERT INTO measurements (voltage1,
                         current1, energy1, voltage2, current2,
                        energy2, voltage3, current3, energy3,
                        total_energy ) VALUES (%s, %s, %s, %s, %s
                        , %s, %s, %s, %s, %s)"
113                 cursor.execute(query, (voltage1, current1,
                        energy1, voltage2, current2, energy2,
                        voltage3, current3, energy3, total_energy
                        ))
114                 conn.commit()
115                 rowcount=cursor.rowcount
116                 print(f"{rowcount} row(s) affected")
117
118                 cursor.close()
119                 conn.close()
120                 data_1 = data.copy()  # Create a copy of the
                        original data dictionary
121                 data_1.update({'total_energy': total_energy
                        })
122                 with app.app_context():
123                     try:
124                         socketio.emit('data', data_1)
125                     except RuntimeError:
126                         current_app.logger.error('Unable to
                                emit data to SocketIO clients.')
127                     time.sleep(1)
```

```
128                 # Schedule the next data send in 1 second
129                 socketio.start_background_task(send_data)
130
131             @app.route('/get_data', methods=['GET'])
132             def get_data():
133                 connection = mysql.connector.connect(
134                     host=db_host,
135                     user=db_user,
136                     password=db_password,
137                     database=db_name
138                 )
139                 cursor = connection.cursor(dictionary=True)
140                 query = "SELECT id,voltage1,current1,energy1
                        ,voltage2,current2,energy2,voltage3,
                        current3,energy3 FROM measurements ORDER
                        BY id DESC LIMIT 100"  # Update with your
                         table name and column names
141                 cursor.execute(query)
142                 data = cursor.fetchall()
143                 connection.commit()
144                 connection.close()
145                 json_data = json.dumps(data)
146                 return Response(json_data, content_type='
                        application/json')
147
148             @socketio.on('connect')
149             def handle_connect():
150                 print('Client connected')
151                 t1 = Thread(target=send_data)
152                 t1.daemon = True
153                 t1.start()
154
155             @socketio.on('send_status')
156             def handle_send_status(state):
```

```
157                 id = None
158                 button_id = state['button_id']
159                 status=state['status']
160                 # Process the received status
161                 # For example:
162                 if button_id == 'load1':
163                     # Do something for Load 1 with the
                            received status
164                     id = "relay1"
165                     pass
166                 elif button_id == 'load2':
167                     # Do something for Load 2 with the
                            received status
168                     id = "relay2"
169                     pass
170                 elif button_id == 'load3':
171                     # Do something for Load 3 with the
                            received status
172                     id = "relay3"
173                     pass
174                 conn = mysql.connector.connect(
175                     host=db_host,
176                     user=db_user,
177                     password=db_password,
178                     database=db_name
179                 )
180                 cursor = conn.cursor()
181                 query = "UPDATE loads SET state = %s WHERE
                        Load_id = %s"
182                 cursor.execute(query, (status, button_id))
183                 conn.commit()
184                 cursor.close()
185                 conn.close()
186                 print(state)
```

```
187                 send_relay_state(id, status)
188
189             @socketio.on('send_mode')
190             def handle_send_mode(mode):
191                 Mode=mode['mode']
192                 print(Mode)
193                 send_mode_control(Mode)
194
195             @socketio.on('list_order')
196             def handle_list_order(jsonData):
197                 data_list = json.loads(jsonData)
198                 conn = mysql.connector.connect(
199                     host=db_host,
200                     user=db_user,
201                     password=db_password,
202                     database=db_name
203                 )
204                 cursor = conn.cursor()
205                 query = "UPDATE priority SET Load_id = %s
                        WHERE id = %s"
206                 # Iterate over the data_list and update each
                        row in the table
207                 for i, item in enumerate(data_list, start=1)
                        :
208                     cursor.execute(query, (item, i))
209                 conn.commit()
210                 cursor.close()
211                 conn.close()
212                 url = 'http://192.168.43.67:5000/endpoint'
213                 priority_list = {'id': 'priority_list'}
214                 priority_list.update({'priority_' + str(i):
                        load_id for i, load_id in enumerate(
                        data_list, start=1)})
215                 headers = {'Content-Type': 'application/json
```

```
                    '}
216                 response = requests.post(url, json=
                        priority_list, headers=headers)
217
218                 if response.status_code == 200:
219                     print('sent successfully to Python 2
                            server')
220                 else:
221                     print('Failed to send  to Python 2
                            server')
222                 # Process the received list order data
223                 # For example, you can print it or perform
                        any other desired action
224                 print(priority_list)
225
226             def send_relay_state(id, stage):
227                 url = 'http://192.168.43.67:5000/endpoint'
228                 payload = {'id':id, 'state': stage}
229                 headers = {'Content-Type': 'application/json
                        '}
230                 response = requests.post(url, json=payload,
                        headers=headers)
231
232                 if response.status_code == 200:
233                     print('Relay state sent successfully to
                            Python 2 server')
234                 else:
235                     print('Failed to send relay state to
                            Python 2 server')
236
237             def send_mode_control(modes):
238                 url = 'http://192.168.43.67:5000/endpoint'
239                 load = {'id':'mode', 'mode': modes}
240                 headers = {'Content-Type': 'application/json
```

```
                    '}
241                 response = requests.post(url, json=load,
                        headers=headers)
242
243                 if response.status_code == 200:
244                     print('mode state sent successfully to
                            Python 2 server')
245                 else:
246                     print('Failed to send relay state to
                            Python 2 server')
247
248             if __name__ == '__main__':
249                 socketio.run(app, host='0.0.0.0', port=5000,
                        debug=True)
```

Listing 6.1: Python Flask server Program

The code begins by importing the required modules and setting up the Flask application and SocketIO instance. It also defines the database connection details and initializes some global variables.

The application consists of several routes that handle different functionalities. The /login route handles user login, validating the user credentials against the MySQL database. Upon successful login, the user is redirected to the home page (/home).

The /receive_data route receives data in JSON format and updates the global data variable. It also inserts the received data into the MySQL database.

The send_data function continuously sends data to the connected clients using SocketIO. It retrieves the data from the global data variable, performs database operations to store the data, and emits the updated data to the clients.

The /get_data route retrieves the last 100 data entries from the MySQL database and returns them in JSON format.

The SocketIO events connect, send_status, send_mode, and list_order handle client-server communication. They receive data from the clients and perform corresponding actions such as updating relay states, handling mode changes, and updating the priority list.

The code also includes functions to send relay states and mode controls to an external server using HTTP requests.

Overall, this code sets up a web application using Flask and Flask-SocketIO, handles user login, stores and retrieves data from a MySQL database, and enables real-time communication with clients through SocketIO.

6.1.1 Web UI

The Web UI consist of two pages Login page and home page which runs on the python flask server. Login Page:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Login</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <h1>Login</h1>
      <form action="/login" method="POST">
        <div class="form-group">
          <label for="username">Username:</label>
          <input type="text" class="form-control" id="username" name="username" required>
        </div>
        <div class="form-group">
          <label for="password">Password:</label>
```

```
                        >
19                        <input type="password" class="form-
                            control" id="password" name="
                            password" required>
20                      </div>
21                      <button type="submit" class="btn btn-
                            primary">Login</button>
22                    </form>
23                  </div>
24                </body>
25              </html>
```

Listing 6.2: HTML Code

Home Page:

```
1           <!DOCTYPE html>
2           <html>
3             <head>
4               <meta charset="utf-8">
5               <title>Home Page</title>
6               <link rel="stylesheet" href="https://maxcdn.
                    bootstrapcdn.com/bootstrap/4.5.2/css/
                    bootstrap.min.css">
7               <script src="https://cdn.jsdelivr.net/npm/chart.
                    js"></script>
8               <!-- Include jQuery -->
9               <script src="https://code.jquery.com/jquery
                    -3.5.1.min.js"></script>
10
11              <!-- Include Bootstrap JS -->
12              <script src="https://maxcdn.bootstrapcdn.com/
                    bootstrap/4.0.0/js/bootstrap.min.js"></script
                    >
13
14
```

```
15              <style>
16               <!--Style css for page-->
17              </style>
18            </head>
19            <body class="bg-image">
20              <div>
21                <nav class="navbar navbar-expand-lg">
22                  <a class="navbar-brand" href="#"> <img src="
                      {{ url_for('static', filename='img/icon.
                      png') }}" alt="Logo" height="70px" > </a>
23                  <button class="navbar-toggler" type="button"
                       data-toggle="collapse" data-target="#
                      navbarNav" aria-controls="navbarNav" aria
                      -expanded="false" aria-label="Toggle
                      navigation">
24                    <span class="navbar-toggler-icon"></span>
25                  </button>
26                  <div class="collapse navbar-collapse" id="
                      navbarNav">
27                    <ul class="navbar-nav ml-auto">
28                      <li class="nav-item active">
29                        <a class="nav-link navbar-nav li a"
                            href="#Home">Home</a>
30                      </li>
31                      <li class="nav-item">
32                        <a class="nav-link navbar-nav li a"
                            href="#Manual">Manual</a>
33                      </li>
34                      <li class="nav-item">
35                        <a class="nav-link navbar-nav li a"
                            href="#Priority">Priority</a>
36                      </li>
37                      <li class="nav-item">
38                        <a class="nav-link navbar-nav li a"
```

```
                        href="#footer">About</a>
39                    </li>
40                  </ul>
41                </div>
42              </nav>
43              <h2 class="poppins-text" id="Home">ESAM</h2>
44              <div>
45                <!-- meters -->
46                <div>
47                  <h3 class="poppins-text_med">Live
                       monitoring</h3>
48                  <div class="row">
49                    <div class="col-md-4">
50                      <div class="card bg-blue text-white">
51                        <div class="card-body">
52                          <h5 class="card-title">Load 1</h5>
53                          <canvas id="chart1" height="100"><
                               /canvas>
54                          <p class="card-text">Voltage: <
                               span id="load1Voltage"></span>
                               V</p>
55                          <p class="card-text">Current: <
                               span id="load1Current"></span>
                               A</p>
56                          <p class="card-text">Energy: <span
                                id="load1Energy"></span> W</p>
57                        </div>
58                      </div>
59                    </div>
60                    <div class="col-md-4">
61                      <div class="card bg-blue text-white">
62                        <div class="card-body">
63                          <h5 class="card-title">Load 2</h5>
64                          <canvas id="load2Chart" height="
```

```
                    100"></canvas>
65                  <p class="card-text">Voltage: <
                    span id="load2Voltage"></span>
                    V</p>
66                  <p class="card-text">Current: <
                    span id="load2Current"></span>
                    A</p>
67                  <p class="card-text">Energy: <span
                     id="load2Energy"></span> W</p>
68                </div>
69              </div>
70            </div>
71            <div class="col-md-4">
72              <div class="card bg-blue text-white">
73                <div class="card-body">
74                  <h5 class="card-title">Load 3</h5>
75                  <canvas id="load3Chart" height="
                    100"></canvas>
76                  <p class="card-text">Voltage: <
                    span id="load3Voltage"></span>
                    V</p>
77                  <p class="card-text">Current: <
                    span id="load3Current"></span>
                    A</p>
78                  <p class="card-text">Energy: <span
                     id="load3Energy"></span> W</p>
79                </div>
80              </div>
81            </div>
82          </div>
83          <div class="">
84            <div class="card bg-blue text-white">
85              <button type="button" class="btn btn-
                  modal" data-toggle="modal" data-
```

```
                target="#myModal">Previous loads</
                button>
86
87              <div class="modal fade" id="myModal"
                tabindex="-1" role="dialog" aria-
                labelledby="myModalLabel" aria-
                hidden="true">
88                <div class="modal-dialog modal-lg">
89                  <div class="modal-content">
90                    <div class="modal-header">
91                      <h5 class="modal-title" id="
                        myModalLabel">Table from
                        MySQL Database</h5>
92                      <button type="button" class="
                        close" data-dismiss="modal"
                         aria-label="Close">
93                        <span aria-hidden="true">&
                          times;</span>
94                      </button>
95                    </div>
96                    <div class="modal-body">
97                      <table id="data-table" class="
                        table table-bordered table-
                        striped">
98                        <thead>
99                          <tr>
100                           <th>ID</th>
101                           <th>Voltage 1</th>
102                           <th>Current 1</th>
103                           <th>Energy 1</th>
104                           <th>Voltage 2</th>
105                           <th>Current 2</th>
106                           <th>Energy 2</th>
107                           <th>Voltage 3</th>
```

```
108                                 <th>Current 3</th>
109                                 <th>Energy 3</th>
110                                 <!-- Add more column
                                        headers as needed -->
111                               </tr>
112                             </thead>
113                             <tbody>
114                               <!-- Table rows will be
                                      inserted here -->
115                             </tbody>
116                           </table>
117                         </div>
118                         <div class="modal-footer">
119                           <button type="button" class="
                                  btn btn-secondary" data-
                                  dismiss="modal">Close</
                                  button>
120                         </div>
121                       </div>
122                     </div>
123                   </div>
124                 </div>
125               </div>
126             </div>
127
128             <!-- meters -->
129             <div class="container_gauge">
130               <div class="gauge">
131                 <div class="gauge-inner">
132                   <div id="gauge-needle" class="gauge-
                        needle"></div>
133                   <div class="gauge-label">Total Load</
                        div>
134                   <div id="gauge-value" class="gauge-
```

```
                        value">0</div>
135                   </div>
136                 </div>
137               </div>
138
139                 <br>
140                 <div class="container_switch">
141                   <h2 class="title">Control Mode Selection
                          </h2>
142
143                   <div class="slider-switch">
144                     <input type="checkbox" id="mode-switch
                            ">
145                     <label for="mode-switch"></label>
146                     <span class="control-label">Auto</span
                            >
147                     <span class="control-label">Manual</
                            span>
148                   </div>
149                 </div>
150
151                 <div id="Manual" class="control-div">
152                   <h3 class="poppins-text_med">Manual
                          Controls</h3>
153                   <div class="row switch">
154                     <div class="col-md-4">
155                       <button id="load1" class="round-
                              button off-button">OFF</button>
156                       <h5 class="font-load">Load 1</h5>
157                       <!-- <button class="round-button off
                              -button">OFF</button> -->
158                     </div>
159                     <div class="col-md-4">
160                       <button id="load2" class="round-
```

```
                    button off-button">OFF</button>
161                 <h5 class="font-load">Load 2</h5>
162                 <!-- <button class="round-button off
                    -button">OFF</button> -->
163               </div>
164               <div class="col-md-4">
165                 <button id="load3" class="round-
                    button off-button" onclick="
                    sendStatus()">OFF</button>
166                 <h5 class="font-load">Load 3</h5>
167                 <!-- <button class="round-button off
                    -button">OFF</button> -->
168               </div>
169             </div>
170           </div>
171           <br>
172           <div id="Priority" class="control-div">
173             <h4 class="poppins-text_med" >Load
                  Priority</h4>
174             <div class="row">
175               <div class="col-md-3">
176                 <div class="button-container">
177                   <button id="saveButton" class="
                      button save">Save</button>
178                   <button class="button reset">Reset
                      </button>
179                 </div>
180               </div>
181                 <div class="col-md-6">
182                   <ul class="list-group" id="
                      sortableList">
183                     <li class="list-group-item list-
                        font">load1</li>
184                     <li class="list-group-item list-
```

```
                        font">load2</li>
185                     <li class="list-group-item list-
                        font">load3</li>
186                     <!-- <li class="list-group-item"
                        >Item 4</li>
187                     <li class="list-group-item">Item
                         5</li> -->
188                   </ul>
189                 </div>
190               </div>
191             </div>
192
193             </div>
194             <footer id="footer">
195               <div class="footer-top">
196                 <div class="container">
197                   <div class="row">
198
199                     <div class="col-lg-8 col-md-8">
200                       <div class="footer-info">
201                         <h3>ESAM</h3>
202                         <p>
203                           Smart microgrids can be used
                              as a remedy for outages and
                               load control that cause
                              the primary power grid in a
                               region to go down,
204                           which could have a significant
                               negative impact on several
                               households, companies, and
                               vital services. However,
205                           it is too expensive to create
                              a smart microgrid. Economic
                               Sensing and Automation in
```

```
                Micro-grids(ESAM) Our
                project focuses on the
                design and implementation
                of the smart microgrid in a
                 more economic manner.
206             Taking the present situation
                of Kerala as well as India,
207             the requirement for the smart
                microgrid is inevitable,
                but the financial
                capability is lesser.
208             Also, considering the current
                designs of electricity
                distribution,
209             which are more complex than in
                 any other location, it is
                difficult to install a
                completely new system. So
                to address these problems,
210             a design variant of sensor
                module hubs, servers, and
                computations is proposed
                through the "Economic
                sensing and automation of
                micro grids" solution.
211             The modular design concept is
                the main feature that helps
                 reduce the cost.
212             The installation feature over
                the existing distribution
                system is the key to the
                advantages.
213           </p>
214         </div>
```

```
215                     </div>
216
217                     <div class="col-lg-2 col-md-8 footer
                            -links">
218                       <h4>Features</h4>
219                       <ul>
220                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">Live
                                monitoring</a></li>
221                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">
                                Remote Controlling of loads</
                                a></li>
222                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">Load
                                History Recording</a></li>
223                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">
                                Variable Load limit and
                                Priority</a></li>
224                       </ul>
225                     </div>
226
227                     <div class="col-lg-2 col-md-8 footer
                            -links">
228                       <h4>Benefits</h4>
229                       <ul>
230                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">Easy
                                Control and Analysis</a></li>
231                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">
                                Modular Design</a></li>
232                         <li><i class="bx bx-chevron-
```

```
                                right"></i> <a href="#">No
                                need to reset existing system
                                </a></li>
233                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">
                                Reduction of Penalities</a></
                                li>
234                         <li><i class="bx bx-chevron-
                                right"></i> <a href="#">
                                Energy Efficiency</a></li>
235                       </ul>
236                     </div>
237                   </div>
238                 </div>
239
240
241               <div class="container">
242                 <div class="copyright">
243                   &copy; Copyright <strong><span>Day</
                          span></strong>. All Rights Reserved
244                 </div>
245                 <div class="credits">
246                   Designed by <a href="https://fisat.ac.
                          in/">EEEFISAT</a>
247                 </div>
248               </div>
249             </footer><!-- End Footer -->
250         </div>
251         <script src="https://ajax.googleapis.com/ajax/
              libs/jquery/3.5.1/jquery.min.js"></script>
252         <script src="https://maxcdn.bootstrapcdn.com/
              bootstrap/4.5.2/js/bootstrap.min.js"></script
              >
253         <script src="https://code.jquery.com/ui/1.12.1/
```

```
    jquery-ui.js"></script>
<script>
</script>
<script>
  $(document).ready(function() {
    $('.btn-modal').click(function() {
      $.ajax({
        url: 'http://127.0.0.1:5000/get_data',
           // Update the URL to your Flask route
        type: 'GET',
        success: function(data) {
          var tableBody = '';
          for (var i = 0; i < data.length; i++)
             {
            var row = '<tr>' +
              '<td>' + data[i].id + '</td>' +
              '<td>' + data[i].voltage1 + '</td>
                 ' +
              '<td>' + data[i].current1 + '</td>
                 ' +
              '<td>' + data[i].energy1 + '</td>'
                 +
              '<td>' + data[i].voltage2 + '</td>
                 ' +
              '<td>' + data[i].current2 + '</td>
                 ' +
              '<td>' + data[i].energy2 + '</td>'
                 +
              '<td>' + data[i].voltage3 + '</td>
                 ' +
              '<td>' + data[i].current3 + '</td>
                 ' +
              '<td>' + data[i].energy3 + '</td>'
                 +
```

```
276                         '</tr>';
277                       tableBody += row;
278                     }
279                     $('#data-table tbody').html(tableBody)
                            ;
280                   }
281                 });
282               });
283             });
284
285
286         </script>
287         <script>
288           function updateValues(data) {
289             // Update the voltage value
290             document.getElementById('load1Voltage').
                    textContent = data.voltage1;
291
292             // Update the current value
293             document.getElementById('load1Current').
                    textContent = data.current1;
294
295             // Update the energy value
296             document.getElementById('load1Energy').
                    textContent = data.energy1;
297             // Update the voltage value
298             document.getElementById('load2Voltage').
                    textContent = data.voltage2;
299
300             // Update the current value
301             document.getElementById('load2Current').
                    textContent = data.current2;
302
303             // Update the energy value
```

```
304                 document.getElementById('load2Energy').
                       textContent = data.energy2;
305                 // Update the voltage value
306                 document.getElementById('load3Voltage').
                       textContent = data.voltage3;
307
308                 // Update the current value
309                 document.getElementById('load3Current').
                       textContent = data.current3;
310
311                 // Update the energy value
312                 document.getElementById('load3Energy').
                       textContent = data.energy3;
313               }
314             </script>
315             <script src="https://cdn.socket.io/4.0.1/socket.
                   io.min.js"></script>
316             <script>
317
318               // Load 2 chart
319               var load2Chart = new Chart(document.
                     getElementById("load2Chart"), {
320               // ...
321               type: 'line',
322               data: {
323                 labels: [],
324                 datasets: [{
325                   label: 'Voltage',
326                   data: [],
327                   borderColor: '#ff6384',
328                   fill: false
329                 }, {
330                   label: 'Current',
331                   data: [],
```

```
                    borderColor: '#36a2eb',
                    fill: false
                  }, {
                    label: 'Energy',
                    data: [],
                    borderColor: '#ffce56',
                    fill: false
                  }]
                },
                options: {
                  responsive: true,
                  scales: {
                    xAxes: [{
                      display: true
                    }],
                    yAxes: [{
                      display: true,
                      ticks: {
                        beginAtZero: true
                      }
                    }]
                  }
                }
              });

              // Load 3 chart
              var load3Chart = new Chart(document.
                 getElementById("load3Chart"), {
                // ...
                type: 'line',
                data: {
                  labels: [],
                  datasets: [{
                    label: 'Voltage',
```

```
                    data: [],
                    borderColor: '#ff6384',
                    fill: false
                  }, {
                    label: 'Current',
                    data: [],
                    borderColor: '#36a2eb',
                    fill: false
                  }, {
                    label: 'Energy',
                    data: [],
                    borderColor: '#ffce56',
                    fill: false
                  }]
                },
                options: {
                  responsive: true,
                  scales: {
                    xAxes: [{
                      display: true
                    }],
                    yAxes: [{
                      display: true,
                      ticks: {
                        beginAtZero: true
                      }
                    }]
                  }
                }
              });

              // Load 1 chart
              var chart1 = new Chart(document.getElementById
                 ("chart1"), {
```

```
398                 type: 'line',
399                 data: {
400                   labels: [],
401                   datasets: [{
402                     label: 'Voltage',
403                     data: [],
404                     borderColor: '#ff6384',
405                     fill: false
406                   }, {
407                     label: 'Current',
408                     data: [],
409                     borderColor: '#36a2eb',
410                     fill: false
411                   }, {
412                     label: 'Energy',
413                     data: [],
414                     borderColor: '#ffce56',
415                     fill: false
416                   }]
417                 },
418                 options: {
419                   responsive: true,
420                   scales: {
421                     xAxes: [{
422                       display: true
423                     }],
424                     yAxes: [{
425                       display: true,
426                       ticks: {
427                         beginAtZero: true
428                       }
429                     }]
430                   }
431                 }
```

```
});

var socket = io.connect('http
    ://127.0.0.1:5000');

//function to receive data
socket.on('data', function(data) {
  var time = new Date().toLocaleTimeString();
  if (chart1.data.labels.length >= 10) {
    chart1.data.labels.shift();
    chart1.data.datasets[0].data.shift();
    chart1.data.datasets[1].data.shift();
    chart1.data.datasets[2].data.shift();
  }
  chart1.data.labels.push(time);
  chart1.data.datasets[0].data.push(data.
     voltage1);
  chart1.data.datasets[1].data.push(data.
     current1);
  chart1.data.datasets[2].data.push(data.
     energy1);
  chart1.update();

  if (load2Chart.data.labels.length >= 10) {
    load2Chart.data.labels.shift();
    load2Chart.data.datasets[0].data.shift();
    load2Chart.data.datasets[1].data.shift();
    load2Chart.data.datasets[2].data.shift();
  }
  load2Chart.data.labels.push(time);
  load2Chart.data.datasets[0].data.push(data.
     voltage2);
  load2Chart.data.datasets[1].data.push(data.
     current2);
```

```
460                 load2Chart.data.datasets[2].data.push(data.
                        energy2);
461                 load2Chart.update();
462
463                 if (load3Chart.data.labels.length >= 10) {
464                   load3Chart.data.labels.shift();
465                   load3Chart.data.datasets[0].data.shift();
466                   load3Chart.data.datasets[1].data.shift();
467                   load3Chart.data.datasets[2].data.shift();
468                 }
469                 load3Chart.data.labels.push(time);
470                 load3Chart.data.datasets[0].data.push(data.
                        voltage3);
471                 load3Chart.data.datasets[1].data.push(data.
                        current3);
472                 load3Chart.data.datasets[2].data.push(data.
                        energy3);
473                 load3Chart.update();
474                 var totalLoad = data.total_energy; //
                        Replace with your actual total load value
475                 $('#gauge-value').text(totalLoad);
476                 updateValues(data);
477               });
478
479               //Function for button status sending
480               function sendStatus(buttonId, state) {
481                 socket.emit('send_status', { button_id:
                        buttonId, status: state});
482               }
483               $('.round-button').click(function() {
484                 var state=null;
485                 if ($(this).hasClass('on-button')) {
486                   $(this).removeClass('on-button');
487                   $(this).addClass('off-button');
```

```
488                     $(this).text('OFF');
489                     state='OFF';
490                   } else {
491                     $(this).removeClass('off-button');
492                     $(this).addClass('on-button');
493                     $(this).text('ON');
494                     state='ON';
495                   }
496                   var buttonId = $(this).attr('id');
497                   sendStatus(buttonId, state);
498                 });
499
500                 //sending list
501                 $(function() {
502                   $("#sortableList").sortable();
503                   $("#sortableList").disableSelection();
504
505                   $('.save').click(function() {
506                     var items = $("#sortableList li").map(
                          function() {
507                       return $(this).text().trim();
508                     }).get();
509
510                     var jsonData = JSON.stringify(items);
511                     console.log(jsonData);
512                     // Emit the jsonData to the server via
                          SocketIO
513                     socket.emit('list_order', jsonData);
514                   });
515                 });
516
517                 //control state
518                 function sendMode(mode) {
519                   socket.emit('send_mode', { mode : mode});
```

```
520             }
521               // Handle switch change event
522             $('#mode-switch').change(function() {
523               if ($(this).is(':checked')) {
524                 // Manual mode
525                 $('#Manual').addClass('disabled');
526                 $('#Priority').removeClass('disabled');
527                 $('.control-label:first-child').text('
                      Manual');
528                 $('.control-label:last-child').text('Auto
                      ');
529                 Mode = 'Auto';
530               } else {
531                 // Automatic mode
532                 $('#Manual').removeClass('disabled');
533                 $('#Priority').addClass('disabled');
534                 $('.control-label:first-child').text('Auto
                      ');
535                 $('.control-label:last-child').text('
                      Manual');
536                 Mode = 'Manual';
537               }
538                 // Disable/enable manual control on page
                      load based on initial switch state
539                 if ($('#mode-switch').is(':checked')) {
540                   $('#Manual').addClass('disabled');
541                 } else {
542                   $('#Priority').addClass('disabled');
543                 }
544                 sendMode(Mode);
545               });
546
547           </script>
548         </body>
```

```
549                </html>
```

Listing 6.3: HTML Code

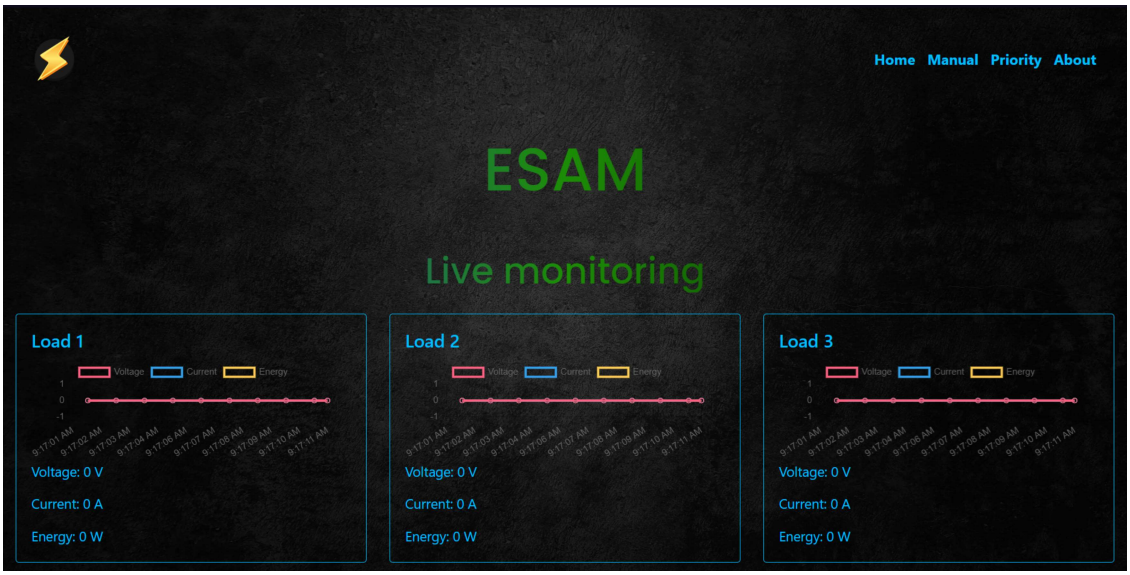


Figure 6.1: Home page Monitoring and live plotting

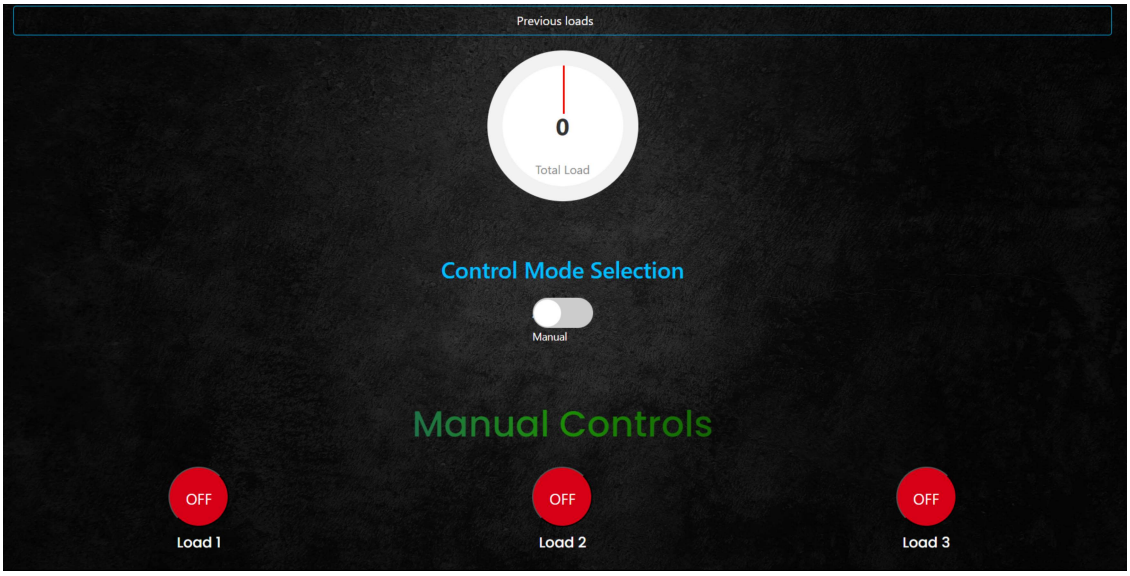


Figure 6.2: Home page Manual Controls and previous loads history

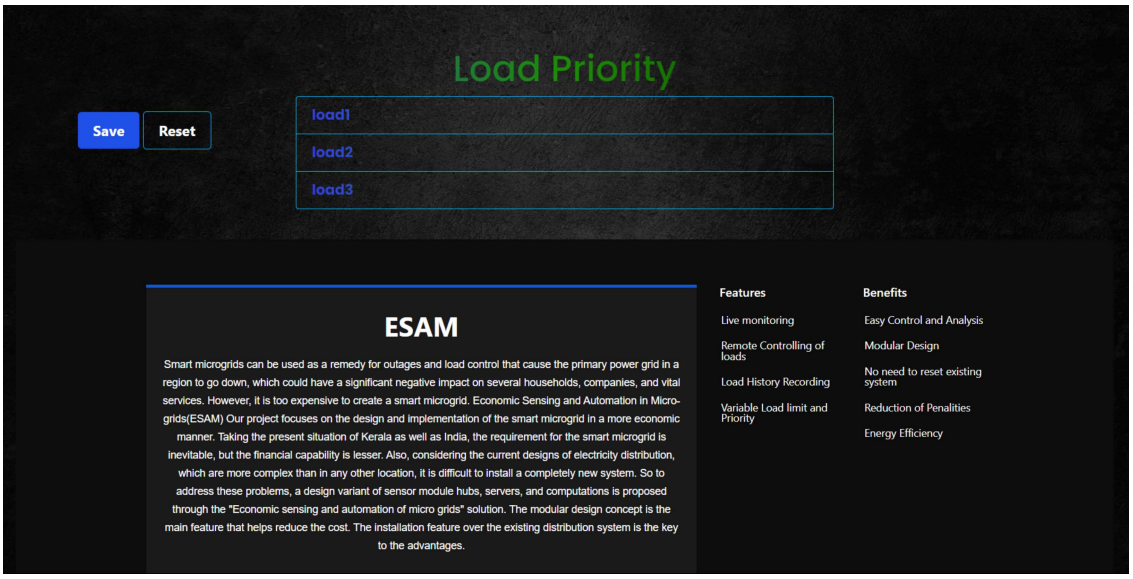


Figure 6.3: Home page Priority list

The code is an HTML document with embedded JavaScript and CSS. It sets up a webpage that displays real-time data and allows user interaction. Let's break down the key features:

Data Retrieval and Display: The code includes an AJAX request that fetches data from a Flask server when a button with the class "btn-modal" is clicked. The retrieved data is used to populate an HTML table with the received values, providing a visual representation of the data. Real-time Updates using Socket.IO: The code establishes a WebSocket connection using the Socket.IO library. It listens for a 'data' event emitted by the server whenever new data is available. Upon receiving the event, the code updates three line charts (chart1, load2Chart, load3Chart) with the latest values for voltage, current, and energy. The line charts use the Chart.js library to dynamically update and visualize the data. User Interaction Controls: The code includes buttons with the class "round-button" that can be clicked to toggle between 'ON' and 'OFF' states. Clicking these buttons triggers an event that sends the button's ID and state ('ON' or 'OFF') to the server using Socket.IO. The server can handle these events to control the corresponding devices or perform any required actions. The code also includes a sortable list that allows users to rearrange items. Clicking the 'save' button triggers an event that sends the updated list order to the server. Overall, the webpage combines data retrieval, real-time updates, and user interaction to create an interac-

tive dashboard displaying data, controlling device states, and allowing customization of the list order.

6.2 Python server in Intel edison

The Edison board acts as the off-grid controller that controls all the modules installed. The board runs on Ubuntu Yocto software with python2 services. So a python2 server is running for the system.

The server serves as a bridge between IoT devices (ESP32 devices) and a Flask server/web UI, enabling communication and control. Here's a brief explanation of the code's functions with respect to the provided requirements:

```
1          import random
2          import time
3          import httplib
4          import json
5          from BaseHTTPServer import BaseHTTPRequestHandler,
              HTTPServer
6
7          voltage1 = 0
8          current1 = 0
9          energy1 = 0
10         voltage2 = 0
11         current2 = 0
12         energy2 = 0
13         voltage3 = 0
14         current3 = 0
15         energy3 = 0
16         relay_state1 = 1
17         relay_state2 = 1
18         relay_state3 = 1
19         relay_id = None
20         load_1 =0
21         load_2 =0
22         load_3 = 0
```

```
23          operation_mode = 'Manual'
24          priority_list = ['load1','load2','load3']  #
               Priority list to be updated from Flask server
25          energy_limit = 200  # Set your desired energy limit
               here
26
27          class RequestHandler(BaseHTTPRequestHandler):
28              def do_POST(self):
29                  global voltage1, current1, energy1, voltage2
                       , current2, energy2, voltage3, current3,
                       energy3, relay_state1, relay_state2,
                       relay_state3, relay_id, priority_list,
                       operation_mode
30                  content_length = int(self.headers['Content-
                       Length'])
31                  data = self.rfile.read(content_length)
32                  json_data = json.loads(data)
33                  #print(json_data)  # Process the received
                       JSON data as desired
34
35                  if json_data.get('id') == 'load1':
36                      voltage1 = json_data.get('voltage')
37                      current1 = json_data.get('current')
38                      energy1 = json_data.get('power')
39                  elif json_data.get('id') == 'load2':
40                      voltage2 = json_data.get('voltage')
41                      current2 = json_data.get('current')
42                      energy2 = json_data.get('power')
43                  elif json_data.get('id') == 'load3':
44                      voltage3 = json_data.get('voltage')
45                      current3 = json_data.get('current')
46                      energy3 = json_data.get('power')
47                  elif json_data.get('id') == 'mode':
48                      operation_mode = json_data.get('mode')
```

```
49                      print(operation_mode)
50                  elif json_data.get('id') == 'relay1':
51                      relay_stage = json_data.get('state')
52                      print(relay_stage)
53                      relay_id = 'relay1'
54                      if relay_stage == "ON":
55                          relay_state1 = 1
56                      elif relay_stage == "OFF":
57                          relay_state1 = 0
58                  elif json_data.get('id') == 'relay2':
59                      relay_id = 'relay2'
60                      relay_stage = json_data.get('state')
61                      #print(relay_stage)
62                      if relay_stage == "ON":
63                          relay_state2 = 1
64                      elif relay_stage == "OFF":
65                          relay_state2 = 0
66                  elif json_data.get('id') == 'relay3':
67                      relay_id = 'relay3'
68                      relay_stage = json_data.get('state')
69                      #print(relay_stage)
70                      if relay_stage == "ON":
71                          relay_state3 = 1
72                      elif relay_stage == "OFF":
73                          relay_state3 = 0
74                  elif json_data.get('id') == 'priority_list':
75                      next_priorities = []
76                      for i in range(1, len(json_data)):
77                          priority_key = 'priority_' + str(i)
78                          priority_load = json_data.get(
                               priority_key)
79                          if priority_load:
80                              next_priorities.append(
                                   priority_load)
```

```
81                      #print("Next priorities:",
                            next_priorities)
82                      priority_list.extend(next_priorities)
83                      #print("Updated priority list:",
                            priority_list)
84
85
86                  self.send_response(200)
87                  self.send_header('Content-type', 'text/plain
                        ')
88                  self.end_headers()
89
90              def do_GET(self):
91                  global relay_state1, relay_state2,
                        relay_state3, relay_id
92                  if self.path == '/endpoint/state':
93                      response_data = {
94                          'relay1': relay_state1,
95                          'relay2': relay_state2,
96                          'relay3': relay_state3
97                      }
98                      self.send_response(200)
99                      self.send_header('Content-type', '
                            application/json')
100                     self.end_headers()
101
102                     self.wfile.write(json.dumps(
                            response_data).encode('utf-8'))
103
104         def run_server():
105             server_address = ('', 5000)
106             httpd = HTTPServer(server_address,
                    RequestHandler)
107             print('Starting server on port 5000...')
```

```
108             httpd.serve_forever()
109
110         def send_data():
111             global voltage1, current1, energy1, voltage2,
                    current2, energy2, voltage3, current3,
                    energy3
112             while True:
113                 data = {
114                     'voltage1': voltage1,
115                     'current1': current1,
116                     'energy1': energy1,
117                     'voltage2': voltage2,
118                     'current2': current2,
119                     'energy2': energy2,
120                     'voltage3': voltage3,
121                     'current3': current3,
122                     'energy3': energy3
123                 }
124
125                 # Convert data to JSON format
126                 json_data = json.dumps(data)
127
128                 # Send the data to the server
129                 conn = httplib.HTTPConnection('
                        192.168.43.244', 5000)
130                 headers = {'Content-type': 'application/json
                        '}
131                 conn.request('POST', '/receive_data',
                        json_data, headers)
132                 response = conn.getresponse()
133
134                 if response.status == 200:
135                     #print('Data sent successfully')
136                     flag=0
```

```
137                 else:
138                     print('Failed to send data')
139
140                 conn.close()
141
142                 time.sleep(1)  # Delay for 1 second before
                       sending the next data
143
144         def manage_loads():
145             global energy_limit, priority_list, relay_state1
                   , relay_state2, relay_state3, load_1, load_2,
                   load_3, energy1, energy2, energy3,
                   operation_mode
146
147             def cut_load(load_id):
148                 global relay_state1, relay_state2,
                       relay_state3, load_1, load_2, load_3
149
150                 if load_id == 'load1' and relay_state1 == 1:
151                     relay_state1 = 0
152                     load_1 = energy1
153                     print("Turning off load1")
154                 elif load_id == 'load2' and relay_state2 ==
                       1:
155                     relay_state2 = 0
156                     load_2 = energy2
157                     print("Turning off load2")
158                 elif load_id == 'load3' and relay_state3 ==
                       1:
159                     relay_state3 = 0
160                     load_3 = energy3
161                     print("Turning off load3")
162
163             def activate_load(load_id):
```

```
164                 global relay_state1, relay_state2,
                        relay_state3, load_1, load_2, load_3
165
166                 if load_id == 'load1' and relay_state1 == 0:
167                     relay_state1 = 1
168                     load_1 = 0
169                     print("Turning on load1")
170                 elif load_id == 'load2' and relay_state2 ==
                        0:
171                     relay_state2 = 1
172                     load_2 = 0
173                     print("Turning on load2")
174                 elif load_id == 'load3' and relay_state3 ==
                        0:
175                     relay_state3 = 1
176                     load_3 = 0
177                     print("Turning on load3")
178
179             while True:
180                 if operation_mode == 'Auto':
181                     LOAD_REDUCTION_AMOUNT = load_1 + load_2
                            + load_3
182                     total_energy = energy1 + energy2 +
                            energy3
183                     print("total_energy: ", total_energy )
184
185                     if total_energy > energy_limit:
186                         print("Auto operation started")
187                         # Excess load detected, start
                                shedding loads
188                         for load_id in reversed(
                                priority_list):
189                             cut_load(load_id)
190                             total_energy -= globals()['
```

```
                        energy' + load_id[-1]]
191                     time.sleep(1)
192
193                     if total_energy <= energy_limit:
194                         break
195
196                 elif total_energy +
                    LOAD_REDUCTION_AMOUNT <= energy_limit
                    :
197                     # Remaining load decreased by a
                        certain amount, start restoring
                        loads
198                     for load_id in (priority_list):
199                         if globals()['energy' + load_id
                            [-1]] <= energy_limit -
                            total_energy:
200                             activate_load(load_id)
201                             total_energy += globals()['
                                energy' + load_id[-1]]
202             print(load_1 ,':', load_2,':', load_3)
203             if total_energy+load_1 <= energy_limit:
204                 activate_load('load1')
205                 print("Activating Load1")
206                 time.sleep(1)
207             if total_energy+load_2 <= energy_limit:
208                 activate_load('load2')
209                 print("Activating Load2")
210                 time.sleep(1)
211             if total_energy+load_3 <= energy_limit:
212                 activate_load('load3')
213                 print("Activating Load3")
214                 time.sleep(1)
215
216             time.sleep(1)  # Delay for 1 second
```

```
                        before checking again
217
218         if __name__ == '__main__':
219             # Start the server in a separate thread
220             import threading
221             server_thread = threading.Thread(target=
                    run_server)
222             server_thread.daemon = True
223             server_thread.start()
224
225             # Start sending data
226             send_data_thread = threading.Thread(target=
                    send_data)
227             send_data_thread.daemon = True
228             send_data_thread.start()
229
230             # Start load management
231             manage_loads_thread = threading.Thread(target=
                    manage_loads)
232             manage_loads_thread.daemon = True
233             manage_loads_thread.start()
234
235             while True:
236                 time.sleep(1)
```

Listing 6.4: Python2 server Program

1. Data Reception and Forwarding: The code receives voltage, current, and energy values from ESP32 devices. These values are updated in variables (voltage1, current1, energy1, etc.) based on the received data. The received data is sent to the Flask server by making a POST request to the '/receive_data' endpoint. The data is converted to JSON format and included in the request payload. The Flask server can process this data and use it for further analysis or display it in the web UI.

2. Operation Modes (Auto and Manual): The code includes an 'operation_mode' variable that represents the current mode of operation ('Auto' or 'Manual'). When the mode is set to 'Auto', the code monitors the energy consumption and load priorities to manage the loads automatically. In the 'Auto' mode, if the total energy consumption exceeds a predefined limit ('energy_limit'), the code executes a load shedding algorithm based on the priority list. If the total energy consumption is below the limit, the code restores loads based on the priority list until the limit is reached. When the mode is set to 'Manual', the code relies on the manage_loads algorithm to control the loads without automatic load shedding.

3. Communication with ESP32 Devices: The code interacts with ESP32 devices by sending commands to turn the loads on or off. The Flask server can send commands to the code, specifying the load ID ('relay1', 'relay2', 'relay3') and the desired state ('ON' or 'OFF'). The code updates the relay states (relay_state1, relay_state2, relay_state3) accordingly. The commands are processed through the 'do_POST' function and can be sent to the ESP32 devices for load control.

In summary, the code receives voltage, current, and energy values from ESP32 devices, sends the data to the Flask server, and provides two modes of operation: 'Auto' and 'Manual'. In 'Auto' mode, it manages loads based on energy consumption and priorities, while in 'Manual' mode, it operates according to the manage_loads algorithm. The code also facilitates communication between the Flask server and ESP32 devices for load control.

6.2.1 Manage loads algorithm

This algorithm plays a major role in controlling the peaks.

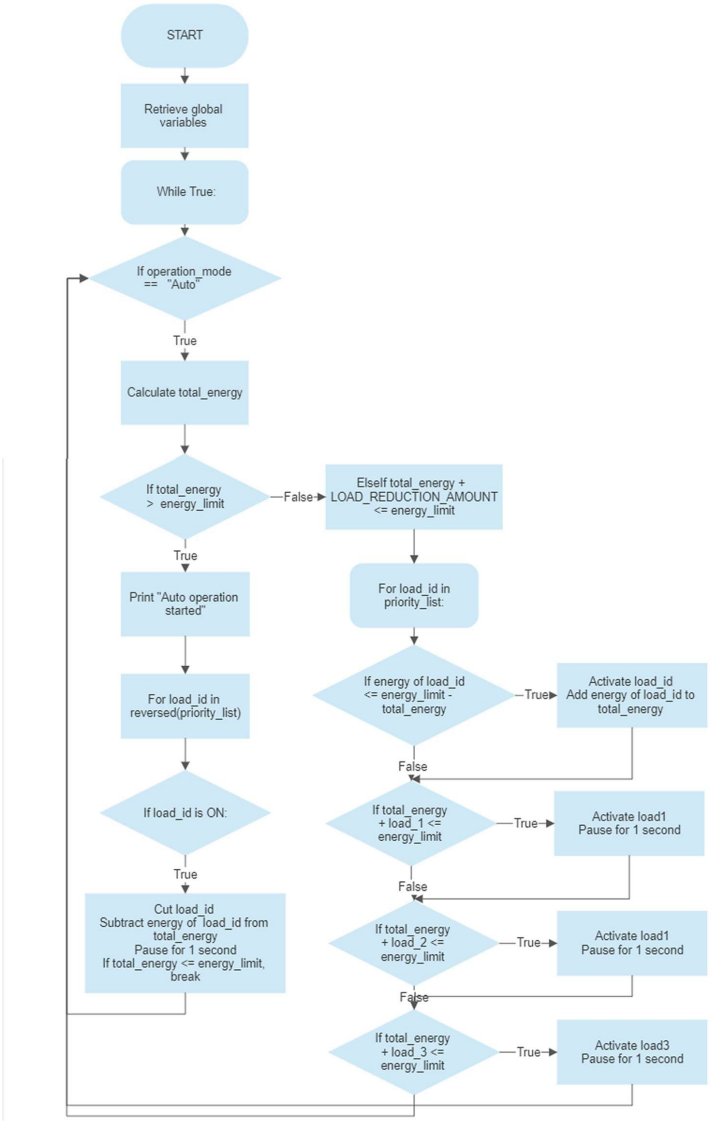


Figure 6.4: Flowchart for Manage loads algorithm

The manage_loads function implements an algorithm to manage the loads based on the energy limit and priority list. Here is a high-level algorithm model for the function:

Retrieve the global variables: energy_limit, priority_list, relay_state1, relay_state2, relay_state3, load_1, load_2, load_3, energy1, energy2, energy3, and operation_mode.

Define two helper functions:

cut_load(load_id): If the load is currently ON, turn it OFF and store the current energy value in the corresponding load variable. activate_load(load_id): If the load is currently OFF, turn it ON and set the load variable to 0. Enter an infinite loop:

Check if the operation_mode is set to "Auto". Calculate the total energy by summing the energy values of all loads: total_energy = energy1 + energy2 + energy3. Energy Limit Check:

If the total energy exceeds the energy limit: Print a message indicating that auto operation has started. Iterate over the priority_list in reverse order to shed loads starting from the lowest priority: Call cut_load(load_id) for each load to turn it OFF and store its current energy value in the corresponding load variable. Subtract the energy of the load from the total_energy. Pause execution for 1 second. If the total_energy becomes less than or equal to the energy limit, break out of the loop. Energy Restoration Check:

If the total energy plus the load reduction amount (sum of load variables) is less than or equal to the energy limit: Iterate over the priority_list in the original order to restore loads starting from the highest priority: If the energy of the load is less than or equal to the remaining energy needed to reach the energy limit: Call activate_load(load_id) for each load to turn it ON and set the corresponding load variable to 0. Add the energy of the load to the total_energy. Check if any loads can be activated individually:

If the total_energy plus the load variable load_1 is less than or equal to the energy limit, activate load1. If the total_energy plus the load variable load_2 is less than or equal to the energy limit, activate load2. If the total_energy plus the load variable load_3 is less than or equal to the energy limit, activate load3. Pause execution for 1 second after activating each load. Pause execution for 1 second before rechecking the load management.

This algorithm continuously checks the energy level and manages the loads accordingly, shedding or activating them based on the energy limit and priority list.

6.3 Sensor Module Code

The sensor module or the ESP32 modules installed works with embedded C++ code along with PlatformIO.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <WiFiClient.h>
#include <WiFiAP.h>

#include <EmonLib.h> // Include Emon Library
EnergyMonitor emon1; // Create an instance
int state = LOW;
const char* ssid = "AndroidAP";
const char* password = "gsw@1234";
const char* serverUrl = "http://192.168.43.67:5000/
   endpoint";
WiFiServer server(1234);

int getRelayState();
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println("Connected to WiFi");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
```

```
28          emon1.voltage(32, 234.26, 1.7);  // Voltage: input
                pin, calibration, phase_shift
29          emon1.current(35, 0.2);       // Current: input
               pin, calibration.
30          pinMode(2, OUTPUT);
31          digitalWrite(2, LOW);
32          // Change the sensitivity value based on value you
                got from the calibrate
33          // example.
34          server.begin();
35        }
36
37        void loop() {
38          // Read relay state
39          int relayState = getRelayState();
40
41          // Control the relay based on the received state
42          digitalWrite(2, relayState);
43          emon1.calcVI(20, 1024);         // Calculate all.
               No.of half wavelengths (crossings), time-out
44          emon1.serialprint();           // Print out all
               variables (realpower, apparent power, Vrms,
               Irms, power factor)
45
46          float realPower = emon1.realPower;        //
               extract Real Power into variable
47          float apparentPower = emon1.apparentPower;    //
               extract Apparent Power into variable
48          float powerFActor = emon1.powerFactor;      //
               extract Power Factor into Variable
49          float voltage = emon1.Vrms;             //extract
               Vrms into Variable
50          float Irms = emon1.Irms;             //extract
               Irms into Variable
```

```
51            if (voltage < 20)
52              voltage = 0;
53            Serial.print("Voltage=");
54            Serial.println(voltage);
55            if (Irms < 0.1)
56              Irms = 0;
57            Serial.print("Current=");
58            Serial.println(Irms); // Irms
59
60            if (apparentPower < 0.1)
61              apparentPower = 0;
62            Serial.print("Power=");
63            Serial.println(apparentPower);
64
65            if (WiFi.status() == WL_CONNECTED) {
66              // Create JSON payload
67              DynamicJsonDocument json(1024);
68              json["id"] = "load1";
69              json["voltage"] = voltage;
70              json["current"] = Irms;
71              json["power"] = apparentPower;
72
73              // Convert JSON to string
74              String payload;
75              serializeJson(json, payload);
76
77              // Send HTTP POST request
78              HTTPClient http;
79              http.begin(serverUrl);
80              http.addHeader("Content-Type", "application/json
                    ");
81              int httpResponseCode = http.POST(payload);
82
83              if (httpResponseCode > 0) {
```

```
84              Serial.print("HTTP Response code: ");
85              Serial.println(httpResponseCode);
86            } else {
87              Serial.print("HTTP Error code: ");
88              Serial.println(httpResponseCode);
89            }
90
91            http.end();
92          }
93
94          delay(500); // Send data every 0.5 seconds
95        }
96        int getRelayState() {
97
98          if (WiFi.status() == WL_CONNECTED) {
99            HTTPClient http;
100           http.begin("http://192.168.43.67:5000/endpoint/
                state");
101
102           int httpResponseCode = http.GET();
103           if (httpResponseCode == 200) {
104             String response = http.getString();
105             DynamicJsonDocument json(1024);
106             deserializeJson(json, response);
107             state = json["relay1"];
108             Serial.print("state:");
109             Serial.println(state);
110           } else {
111             Serial.print("HTTP Error code: ");
112             Serial.println(httpResponseCode);
113           }
114
115           http.end();
116         }
```

```
117             return state;
118         }
```

Listing 6.5: Embedded C++ Code

Explanation of how the program works:

1. Libraries: The program includes several libraries required for WiFi connectivity, HTTP communication, handling JSON data, and energy monitoring using the EmonLib library.

2. Global Variables: state: Stores the current state of the relay (LOW or HIGH). ssid and password: Stores the WiFi network credentials. serverUrl: Specifies the URL for sending the power consumption data. WiFiServer server(1234): Initializes a WiFi server on port 1234.

3. Function: getRelayState() This function is used to retrieve the state of the relay by making an HTTP GET request to a specific endpoint (/endpoint/state) on a server. It uses the HTTPClient library to perform the GET request and receives the response as a JSON string. The JSON response is parsed using the ArduinoJson library, and the state of the relay is extracted from the JSON data. The extracted state is stored in the state variable and returned.

4. setup() function: Initializes the serial communication and connects to the specified WiFi network. Sets up the voltage and current measurement using the EmonLib library. Sets the initial state of the relay to LOW (OFF). Starts the WiFi server on port 1234.

5. loop() function: Reads the current state of the relay using the getRelayState() function. Controls the relay by setting the corresponding GPIO pin (pin 2) to the received state. Measures voltage, current, real power, apparent power, and power factor using the EmonLib library. Prints the measured values to the serial monitor. If the WiFi connection is established, the program creates a JSON payload with the measured values. The JSON payload is sent as an HTTP POST request to the specified server URL. The HTTP response code is printed on the serial monitor.

The program continuously loops, collecting power consumption data and sending it to the server every 0.5 seconds. It also checks the state of the relay periodically by making an HTTP GET request to retrieve the state from the server. The relay can be remotely controlled by updating the state on the server's endpoint.

6.4 Libraries used

The project utilized several external libraries to enhance its functionality and enable the successful implementation of various features. These libraries played a crucial role in achieving the project objectives.

1. Flask: The Flask library was used to develop the web-based user interface. It provided a lightweight and versatile web framework for creating interactive and user-friendly interfaces. Flask allowed users to access and control the microgrid system conveniently through their web browsers.

2. MySQL Connector: The MySQL Connector library was used to establish a connection with the MySQL database and perform database operations. It facilitated seamless integration with the database, enabling efficient storage and retrieval of data related to the microgrid system.

3. Python Requests: The Python Requests library was utilized for making HTTP requests to external APIs. It allowed the project to interact with external services and retrieve relevant data to enhance the functionality of the system.

4. Flask-SocketIO: The Flask-SocketIO library was employed for real-time communication between the server and clients. It enabled the transmission of data between the microgrid system and the user interface, facilitating instant updates and responsiveness.

5. ArduinoJson: The ArduinoJson library was utilized for parsing and generating JSON data. It provided convenient functions for handling JSON objects, making it easier to process and exchange data between different components of the system.

6. EmonLib: The EmonLib library was used for energy monitoring. It provided functions and algorithms for measuring energy consumption and monitoring load patterns, allowing for efficient management and analysis of power usage.

7. WiFi: The WiFi library was employed for establishing and managing Wi-Fi connections on the ESP32 microcontroller. It allowed the microgrid system to connect to local Wi-Fi networks, enabling communication with other devices and internet access.

8. HTTPClient: The HTTPClient library facilitated making HTTP requests from the ESP32 microcontroller. It enabled communication with external APIs and services, allowing the microgrid system to fetch data or send requests to external resources.

9. WiFiClient: The WiFiClient library provided the necessary functionalities for creating a TCP client connection over Wi-Fi. It allowed the microgrid system to communicate with other devices or servers via TCP/IP protocols.

10. WiFiAP: The WiFiAP library enabled the ESP32 microcontroller to function as an access point. It allowed other devices to connect to the microgrid system directly, creating a local network for communication and control.

Chapter 7

Results

The objective of this project was to develop a prototype model for Economic Sensing and Automation in Microgrids (ESAM). The prototype aimed to achieve the following objectives:

1. Analysis of the loads and supply system
2. Load prioritization and control
3. Cost savings and reduction of manual effort
4. Centralized distribution control system

7.1 Prototype Model Implementation

The implementation of the prototype model involved a series of meticulous steps to ensure its successful development and functionality. The first stage included the design and fabrication of sensor-control modules on printed circuit boards (PCBs). The modules were carefully designed, taking into consideration the specific requirements of the project.

After the design phase, the PCBs were selected and the components were soldered onto the boards. This process required precision and attention to detail to ensure the modules' reliability and stability. Once the soldering process was completed, thorough testing and inspection were carried out to verify the integrity of the connections and components.

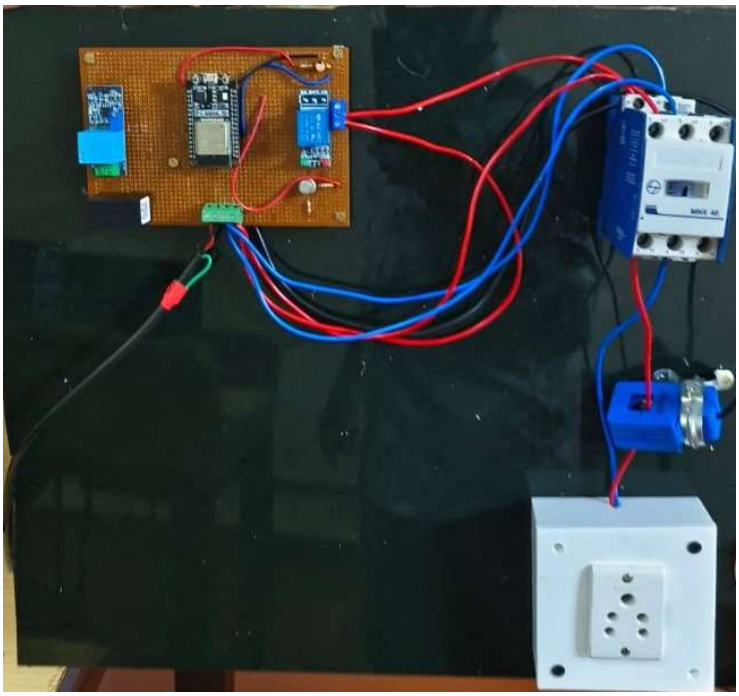


Figure 7.1: Sensor Control module prototype

Three fully assembled modules were prepared for testing purposes. These modules were strategically placed in different locations within the microgrid system to capture load data from various sources. The selection of these locations was based on the diversity of loads and their representation of the overall system.

To establish communication between the modules and the server, a local server infrastructure was set up. This allowed for efficient data transmission and storage. Rigorous testing was conducted to ensure the stability and reliability of the communication system. Data transmission between the modules and the server was validated, and any potential issues were addressed and resolved.

In order to validate the accuracy of load measurements, a range of loads were sampled using the prototype model. This included minor loads such as high-power bulbs and inductive loads. The prototype model demonstrated its capability to accurately capture and record load data from these sources. The collected data was stored in a MySQL database for further analysis and visualization.

Furthermore, a web user interface (UI) was developed to provide a user-friendly platform for accessing and visualizing the load data. The web UI displayed real-time load information, including graphical representations of load profiles. Both manual

and automatic load control functions were successfully implemented and tested, further enhancing the usability and effectiveness of the prototype model.

Overall, the implementation of the prototype model was a significant milestone in the project. The successful design, fabrication, testing, and validation of the sensor-control modules, along with the establishment of the server infrastructure and the development of the web UI, demonstrated the feasibility and potential of the proposed solution. The prototype model showcased its capability to accurately measure and monitor loads within the microgrid system, paving the way for further advancements and improvements in the project.

7.2 Load Monitoring and Analysis

The load measuring units effectively captured load data from various sources within the microgrid. The recorded load measurements were stored in a MySQL database for comprehensive analysis. The web-based UI displayed the load and measurement data through intuitive plots and graphs, facilitating insights into load patterns and fluctuations. This load monitoring and analysis capability enhanced the understanding and optimization of the microgrid system.

7.3 Load Prioritization and Control

The prototype model successfully prioritized and controlled loads based on the recorded data. The auto control function utilized the load measurements to automatically adjust load priorities and optimize energy consumption. It effectively responded to load fluctuations, thereby reducing penalties caused by excessive demand during peak times. The manual control function provided users with the flexibility to intervene and make adjustments based on specific requirements or unforeseen circumstances. The load prioritization and control features proved to be efficient and reliable.

7.4 Cost Savings and Reduction of Manual Effort

The prototype model demonstrated its capability to achieve cost savings and reduce manual effort. By optimizing load prioritization and consumption, the model effec-

tively minimized penalties associated with excess demand during peak times, resulting in cost savings for the microgrid. Additionally, the automated control function minimized the need for manual intervention, saving time and effort for the operators.

7.5 Centralized Distribution Control System

The prototype model centralized the distribution control system, enabling efficient management of the microgrid. The integrated system facilitated seamless communication between the load measuring units, Edison board, and the web-based UI. This centralized control system streamlined operations and improved overall microgrid performance.

7.6 Validation and Testing

The prototype model underwent rigorous validation and testing to ensure accuracy and reliability. Various testing scenarios and methodologies were employed to evaluate its performance. The validation results confirmed the successful operation of the prototype model in achieving the project objectives. The model proved to be a robust solution for Economic Sensing and Automation in Microgrids (ESAM).

The prototype model for ESAM showcased its effectiveness in achieving the project objectives. The integration of the Edison board, load measuring units, and the web-based UI allowed for load monitoring, analysis, prioritization, and control. The model successfully achieved cost savings, reduced manual effort, and centralized the distribution control system. While encountering some limitations during the project, such as the need for further scalability and system integration, the prototype model established a solid foundation for future improvements and advancements.

The results of this project contribute to the field of microgrid management, demonstrating the potential of Economic Sensing and Automation to enhance energy efficiency and streamline operations. The successful implementation of the prototype model sets the stage for further research and development in this area, providing opportunities for future innovation and practical applications.

Chapter 8

Benefits of the solution

ESAM (Economic Sensing and Automation in Microgrids) is a groundbreaking technology that brings numerous benefits to microgrids, enhancing their efficiency, reliability, and sustainability. The implementation of ESAM in microgrids yields the following advantages:

1. Enhanced Efficiency: ESAM leverages real-time data to optimize power distribution within the microgrid. By constantly monitoring and analyzing the energy flow, ESAM minimizes waste, ensuring that energy is utilized in the most efficient manner. This leads to significant improvements in the overall efficiency of the grid, resulting in reduced energy losses and increased cost-effectiveness.

2. Improved Reliability: With its advanced sensing and automation capabilities, ESAM is capable of swiftly detecting and responding to outages or other grid-related issues. By promptly identifying problems, ESAM enables rapid restoration efforts and minimizes downtime. The enhanced reliability of the microgrid ensures uninterrupted power supply to critical loads, offering a reliable energy source for various applications.

3. Empowered Consumer Control: ESAM empowers consumers by providing them with real-time information about their energy consumption. Through intuitive interfaces and detailed energy usage data, consumers can make informed decisions regarding their energy usage patterns. This increased awareness allows for

better load management and the ability to adjust energy consumption based on individual preferences and needs.

4. Economic Benefits: The implementation of ESAM brings significant economic advantages to both consumers and utilities. By optimizing the power distribution and reducing energy losses, ESAM helps decrease operational costs for utilities. Moreover, by enabling consumers to actively manage their energy usage, ESAM promotes energy conservation and reduces overall energy bills for end-users. Additionally, ESAM minimizes the need for costly infrastructure changes by optimizing the existing grid system, resulting in cost savings for all stakeholders involved.

5. Environmental Sustainability: ESAM contributes to the overall sustainability of microgrids by promoting energy efficiency and reducing carbon emissions. Through its intelligent load management and optimization capabilities, ESAM facilitates the integration of renewable energy sources, such as solar and wind, into the microgrid. This supports the transition towards a cleaner and greener energy ecosystem, reducing the reliance on fossil fuels and mitigating environmental impacts.

By harnessing the power of ESAM, microgrids can unlock these compelling benefits, leading to a more efficient, reliable, and sustainable energy system for communities and industries alike.

Chapter 9

Future Scope

While ESAM is currently in its initial stages as a robust and efficient system, there is scope for further enhancements and refinements. Ongoing research and development efforts can focus on optimizing system controls, improving fault detection and response mechanisms, and enhancing overall system performance. By continuously evolving ESAM, it can become an even more powerful tool for microgrid management.

1. Integration of Multiple Energy Sources: Currently, ESAM focuses on load control within the microgrid. However, future developments can explore the integration of multiple energy sources, such as solar photovoltaic (PV) systems, uninterruptible power supplies (UPS), and generators. This integration can introduce sophisticated re-routing techniques, allowing for seamless transitions between energy sources based on availability, demand, and other factors. By effectively managing and utilizing these diverse energy sources, ESAM can ensure uninterrupted power supplies and optimize energy utilization.

2. Utilizing Machine Learning Algorithms: The integration of machine learning (ML) algorithms into the ESAM system opens up exciting possibilities for predictive analytics and intelligent decision-making. By collecting and analyzing data from the microgrid over a specific period, ML algorithms can identify patterns, predict energy losses, and optimize system operations accordingly. These intelligent algorithms can enable proactive measures to prevent or mitigate potential issues, improve energy efficiency, and optimize load distribution. ML-driven insights can also facilitate better planning and resource allocation in the

microgrid.

3. Grid-Level Energy Management: As ESAM evolves, there is potential for expanding its scope beyond microgrid-level management to grid-level energy management. By connecting multiple microgrids and leveraging advanced communication and control systems, ESAM can contribute to the development of smart grids. This would enable efficient energy exchange, load balancing, and overall grid optimization, leading to a more sustainable and resilient energy infrastructure.

4. Integration with Energy Storage Systems: The integration of energy storage systems, such as batteries, into the ESAM framework can further enhance the reliability and flexibility of the microgrid. Energy storage systems can store excess energy during periods of low demand and release it during peak demand, ensuring a stable and uninterrupted power supply. ESAM can incorporate intelligent algorithms to optimize the charging and discharging cycles of energy storage systems, maximizing their efficiency and extending their lifespan.

The future scope of ESAM encompasses continuous improvements in system performance, the integration of multiple energy sources, the utilization of machine learning algorithms for predictive analytics, grid-level energy management, and integration with energy storage systems. These advancements hold the potential to revolutionize microgrid management, enabling more efficient, reliable, and sustainable energy systems for the future.

Chapter 10

Conclusion

In conclusion, the automation of grids through technologies like ESAM (Economic Sensing & Automation in Microgrids) offers immense potential for the power industry. By harnessing advanced technology, smart grids can revolutionize power generation and distribution, leading to improved efficiency and reliability. ESAM's cost-effective and user-friendly design makes it a viable solution for transitioning to automated microgrid systems.

The benefits of ESAM are significant. It empowers consumers with greater control over their energy usage, allowing them to make informed decisions and optimize their consumption patterns. This not only benefits individuals but also contributes to overall energy conservation and sustainability. Additionally, ESAM enables seamless integration of multiple power sources, such as solar, UPS, and generators, which enhances the resiliency of the microgrid and ensures uninterrupted power supply.

Furthermore, future enhancements in ESAM can leverage machine learning algorithms to analyze data collected from the microgrid. This data-driven approach can provide valuable insights into energy losses, predictive maintenance, and system optimization. By intelligently adapting to changing conditions and demands, ESAM can continuously improve the efficiency and performance of the microgrid.

While challenges exist, such as the need for investment in technology and infrastructure, the automation of grids represents a promising direction for the power industry. ESAM's ability to address key challenges, provide cost-effective solutions, and enhance the overall grid operation makes it a vital component in the transition towards a more efficient and sustainable energy future.

In summary, ESAM brings together the power of automation, control, and optimization to revolutionize microgrid systems. It offers a pathway to address current challenges, improve energy efficiency, and ensure reliable power supply. With ongoing advancements and continued research, ESAM is poised to play a crucial role in shaping the future of the power industry.

Bibliography

[1] Mostafa Farrokhabadi et al. "Microgrid Stability Definitions, Analysis, and Examples". In: *IEEE Transactions on Power Systems* 35.1 (2020), pp. 13–29. DOI: 10.1109/TPWRS.2019.2925703.

[2] S. M. Abu Adnan Abir et al. "IoT-Enabled Smart Energy Grid: Applications and Challenges". In: *IEEE Access* 9 (2021), pp. 50961–50981. DOI: 10.1109/ACCESS.2021.3067331.

[3] Nick Farid and Roghoyeh Salmeh. "Network Configurations for IoT Services in Smart Grid". In: *2021 9th International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*. 2021, pp. 15–18. DOI: 10.1109/ICSGCE52779.2021.9621636.

[4] Yunpeng Si et al. "A High Performance Communication Architecture for a Smart Micro-Grid Testbed Using Customized Edge Intelligent Devices (EIDs) With SPI and Modbus TCP/IP Communication Protocols". In: *IEEE Open Journal of Power Electronics* 2 (2021), pp. 2–17. DOI: 10.1109/OJPEL.2021.3051327.