Ryerson University
Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

BV01 Supervisory Control and Data Acquisition System

by

Haadiya Jamil, Vidy Matadeen, Arthi Sarkar, and Anissa EL-Farkh

Electrical Engineering Capstone Design Project

Toronto Metropolitan University, 2025

Acknowledgements

This group would like to acknowledge Dr. Bala Venkatesh, Shima Bagher Zade Homayie, and Pedro Vasconcelos for their support and overall guidance throughout the fall and winter semesters.

Certification of Authorship

We hereby declare that the work titled Supervisory Control and Data Acquisition System is our original creation and has not been published or submitted for publication elsewhere. We affirm that all references and sources used in the preparation of this work have been appropriately cited and acknowledged.

Signed,

Haadiya Jamil,
Vidy Matadeen,
Arthi Sarkar, and
Anissa EL-Farkh

April 11th, 2025

Table of Contents

1. Abstract

Supervisory Control and Data Acquisition (SCADA) systems are integral to modern power distribution networks, providing centralized monitoring and control to ensure efficient and reliable power distribution. These systems support real-time data acquisition from sensors and devices on the grid. Allowing operators to make informed decisions, detect faults, and implement corrective actions promptly. SCADA systems are vital for automating grid operation, enhancing energy efficiency, and enabling continued operation during unexpected faults or events. This project proposes a SCADA system that combines hardware and software to monitor and control electrical measurements across different loads in real time. Using an ESP32 microcontroller and a user-friendly graphical interface, the system collects, processes, and displays key electrical parameters such as voltage, current, apparent power, true power, reactive power, and power factor. Users can conveniently view real-time data through graphical displays and control loads such as light bulbs, and heaters directly from the system using relays. The graphical user interface (GUI) was developed using the Tkinter, and other python libraries for image integration, data visualization and network representation. Additional functionalities included zooming, panning, adjustable window sizing, and dynamic visualization of the systems topology. The hardware test system was designed to represent different types of electrical loads within a power distribution network. By using this system, operators are guaranteed overcurrent protection and auto reset capability.

2. Introduction & Background

Supervisory Control and Data Acquisition (SCADA) systems are used widely across various industries such as telecommunications, wastewater management, traffic control, and manufacturing. This project will focus on the application of SCADA systems used in electrical networks. These systems are tools that provide users or operators with the ability to monitor and control industrial processes. Through their use, operators are provided with real-time monitoring data pertaining to electrical parameters such as voltage, current, and power for loads, which can be critical in the decision-making regarding optimizing load distributions. Alerting operators or automatically responding to issues, reducing downtime and improving efficiency. Many of the current systems available today are designed for large-scale industrial applications that monitor or connect to numerous devices, require an immense amount of hardware components, software licensing, and cloud-based or in-house data storage, which make them expensive to set up and maintain. This makes them difficult to implement in smaller-scale or educational settings. The aim of this project is to deliver a solution that can monitor and control electrical loads effectively and reliably at an affordable cost, while maintaining all the benefits of large-scale systems [4].

The goal of this project is to develop a compact solution that can interface with an existing power network, allowing the user real-time monitoring, data acquisition, and control. The system will use a Tensilica Xtensa LX6 microprocessor, a low-power and energy-efficient chip built into the ESP32 microcontroller. This microcontroller will provide the additional peripherals for the processor, such as Wi-Fi and Bluetooth capabilities, paired with an industrial-grade power meter for collecting data on various loads. The user interface will be built using Tkinter[8], providing an intuitive platform for operators to visualize and remotely interact with electrical load data. Additionally, the system will store key data in reports (text or PDF format), which will be useful for analyzing faults or abnormal trends [5].

Such systems are composed of numerous hardware and software components working together to measure, transmit, and display data. Some of the primary components that industrial systems use are sensors, Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), and Human-Machine Interfaces (HMIs) to collect and process data. Sensors used in electrical network measurement would be power meters containing voltage and current transformers that scale and measure the important electrical parameters. They are then sent to the central system using RTUs or PLCs as a means of interfacing the field equipment with the central system. The central system is where data processing takes place and is displayed for operators, who can take actions to control the system, for example, turning off loads to isolate faults [6].

Traditionally, such systems use PLCs for control, which can be expensive and limited in regard to the peripherals that can be connected to the unit. For the scope of this project, the ESP32 microcontroller will be utilized as the control system. This control system is cost-effective and offers a numerous variety of built-in data processing features along with Wi-Fi and Bluetooth compatibility out of the box. The system will interface with industrial-grade power meters to ensure accurate data collection. Real-time monitoring will be a key feature, enabling operators to track live electrical parameters and detect faults as they happen.

Additionally, the system will store data in reports, allowing operators to analyze trends and address any recurring issues [4].

Many of these systems, specifically those designed for smaller-scale applications, face limitations when implementing real-time control and feature complex user interfaces. By integrating real-time monitoring, device control, and data visualization into a single, user-friendly platform, this project aims to address these challenges. The proposed system will be cost-effective, scalable, and intuitive, making it suitable for a wide range of applications, from educational purposes to industrial use.

3. Objectives

The goals of this project are structured to develop a SCADA system for real-time monitoring and control of electrical loads through the SCADA interface. The key objectives include:

Graphical User Interface (GUI): Develop a user-friendly GUI that enables users to view real-time graphs and data for electrical measurements. This includes key parameters like supply voltage, current, apparent power, real power, reactive power, power factor, and phase angle. The GUI will also support control actions, such as toggling loads, and include features like pan, zoom, reset, and user-adjusted data window size to enhance the visualization of plotted data.

Development of Test System: Develop a physical test setup to validate the SCADA system, ensuring that it can effectively monitor and control real-world electrical loads. Simulate conditions like normal operation, overcurrent to test the system's response. Verify that SCADA can communicate with physical components (ESP32, sensors, and relays) accurately.

Load Control with Safety Features: Implement a relay system to allow users to control electrical loads through the SCADA interface. Safety features will be incorporated to automatically shut off loads in case of overcurrent or overheating, protecting both the system and connected devices.

Sensor Integration and Power Measurement: Ensuring that the measurement system is calibrated accurately for real-world conditions and capable of providing reliable data to the SCADA system. The system will use a power meter to ensure accurate raw data measurements, which will be processed by the ESP32 microcontroller.

Hardware and Software Integration: The project will combine hardware components (e.g., ESP32, sensors, relays, multiple loads) with Python-based software to provide a complete SCADA solution.

Data Processing, Transmission, and Visualization: Process raw voltage and current data using the ESP32 to calculate advanced parameters like apparent power, true power, and power factor through a power meter. Implement a reliable data transmission method (HTTP) for sending real-time data from ESP32 to SCADA software. Ensure transmitted data is reliable. Develop a system for generating reports (text or PDF) containing key metrics, fault logs, and trend analysis for further review.

4. Theory and Design

This section provides a detailed overview of the theoretical foundations and the design methodology used in developing the software and hardware components of the SCADA system.

4.1 Scientific/Engineering Principles

The ESP32 was responsible for bridging the software representation of the system with the physical hardware components in the test system. The system consisted of numerous loads, all operating using AC voltage. Our SCADA system was able to monitor the system regarding power consumption and take action to control the states of the load. Current and voltage waveforms acquired from the hardware system were processed by the SCADA system, allowing for real-time power flow monitoring. The phase angle between the signals was calculated using Python, which enabled us to determine true, reactive, and apparent power. The data processing could be done at the software level or in the ESP32, as it was capable of running Python, making the transfer of data clearer. The phase angle measurement was computed in the interfacing stage on the ESP32 by sampling both current and voltage signals using the dual ADCs and then calculating the phase difference using either zero-crossing detection or cross-correlation. Once voltage, current, and phase angle were acquired, the power-related data were transferred to the network representation for use in the SCADA system.

Table 4.1.1: Power formulas for Data Processing.

Type of Power	Formula
True Power (P)	$P = V \cdot I \cdot cos(\phi)$
Reactive Power (Q)	$Q = V \cdot I \cdot sin(\phi)$
Apparent Power (S)	$S = V \cdot I$ or $S = \sqrt{P^2 \cdot Q^2}$
Power Factor (PF)	PF = $cos(\phi)$

4.2 Technical Research

SCADA systems were widely used across various industries such as telecommunications, wastewater management, traffic control, and manufacturing. SCADA was a tool that provided users or operators with the ability to monitor and control industrial processes. Through the use of SCADA systems, operators received real-time monitoring data pertaining to electrical parameters such as voltage, current, and power for loads, which were critical in decision-making regarding optimizing load distributions. These systems could alert operators or automatically respond to issues, reducing downtime and improving efficiency. Many of the current SCADA systems available at that time were designed for large-scale industrial applications that monitored or connected to numerous devices and required an immense amount of hardware

components, software licensing, and cloud-based or in-house data storage, making them expensive to set up and maintain. This complexity made them difficult to implement in smaller-scale or educational settings. SCADA systems were composed of numerous hardware and software components that worked together to measure, transmit, and display data. Some of the primary components that industrial SCADA systems use include sensors, Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), and Human-Machine Interfaces (HMIs) to collect and process data. Sensors used in electrical network measurement included power meters containing voltage and current transformers that scaled and measured the important electrical parameters. This data was then sent to the central system using RTUs or PLCs as a means of interfacing the field equipment with the SCADA system. The central system was where data processing took place and was displayed for operators, who could take actions to control the system, such as turning off loads to isolate faults.

4.3 Application of Theory to Design

Power flow is an integral part of a SCADA system. Understanding how power flowed through different components in the system was important for designing the network representation. This allowed for the effective prediction of the behavior of electrical components (e.g., lightbulbs, heaters) to simulate a real-world power system.

Relays are helpful for providing protection to system components in case of a fault. Understanding the usage of relays in power systems was essential for the system to identify faults such as overcurrent conditions, and shut off components quickly and efficiently to protect them.

The network representation in the SCADA system served as a simulation of a real-world power system, which allowed for the testing of scenarios like overcurrent and the behavior of components turning on and off. Additionally, the network showed how components were connected in a power system: the source was connected to loads, which had switches connected to them, allowing the user to maintain control.

The GUI was created to provide the user with an easy experience to interact with the software. All electrical measurements were displayed in the main window, and graphs showing visual changes in the measurements were displayed in a separate window.

4.4 Front-End Software Development

In the initial phase of GUI development, it was necessary to design the SCADA software to be capable of real-time monitoring of electrical measurements. The primary objective was to design a user-friendly and intuitive GUI that would enable operators to efficiently monitor and manage essential electrical parameters such as voltage, current, apparent power, and power factor across various loads within a simplified power distribution network. It was important to ensure its compliance with industry standards. Ignition SCADA software and Schneider SCADA software systems were examined as a

reference. Ignition is extensively utilized in the industry for the development of sophisticated SCADA systems, and its GUI features were analyzed to gain insights into its data visualization, real-time graphing, and interactive components, as shown in **Figure 4.4.1** [7]. The software's ability to present real-time data, incorporate graphical elements, and offer interactive controls for device monitoring and management was crucial in determining the necessary features for an effective SCADA GUI.

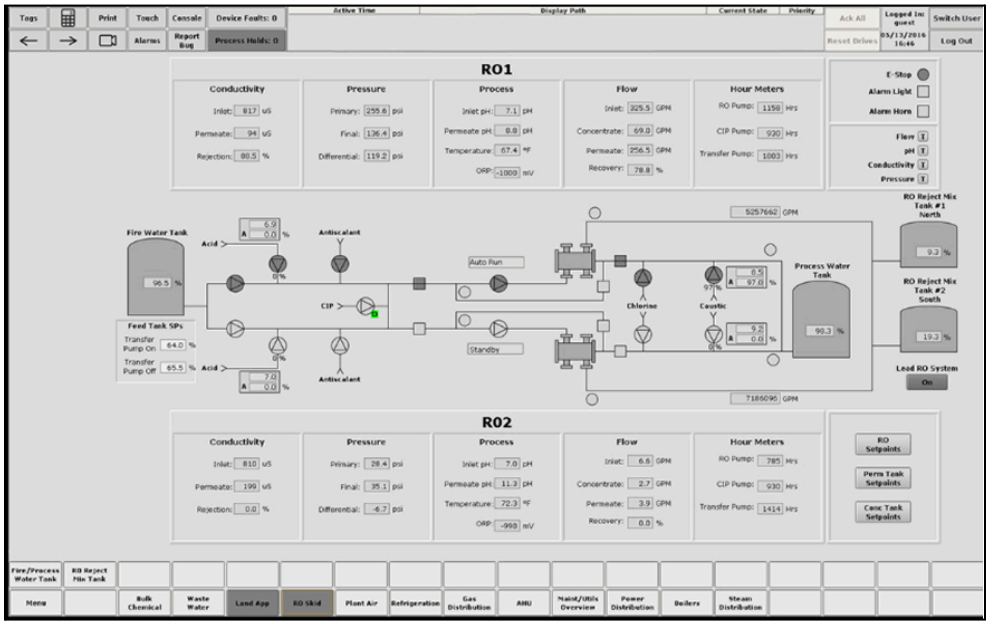


Figure 4.4.1: Inductive Automation "SCADA Software " as reference for GUI development [7].

The front-end of the SCADA system focuses on the GUI, which plays a crucial role in ensuring ease of use, interaction, and data visualization. The GUI was built using the Tkinter [8] library, known for its simplicity and efficiency in creating desktop GUI applications. The flowchart in **Figure 4.4.2** illustrates the organization and structure of the SCADA software's user interface, showing how the various components, such as the Login Window, Main SCADA GUI Module, and Plotting Window, are connected. The flowchart helps visualize the navigation between different functional modules, ensuring that users can easily interact with the system. The main window of the system was designed with a modular, hierarchical layout using frames, which allowed for the organization and display of various electrical parameters, such as voltage, current, reactive power, active power, apparent power, and power factor. Tkinter's [8] *grid layout manager* was employed to position widgets precisely within the GUI, making the interface responsive and user-friendly. Each widget, such as labels and buttons, was organized into frames, which were further arranged within the window. For example, the *data_frame* utilized *grid_columnconfigure* and *grid_rowconfigure* to manage the arrangement of frames, ensuring that each measurement type was clearly presented and easy to monitor.

The front-end also included a network visualization frame, created using a combination of NetworkX [12], Matplotlib [10], and Tkinter [8], to display the system's

layout. This visualization enabled users to view the connections between loads and control them interactively. Additionally, the front-end featured a panel with buttons for interacting with real-time data plots. Buttons for zooming, panning, and resizing plots were organized in a control frame, offering users the ability to adjust the plot's view dynamically.

The *open_main_scada_gui()* function is responsible for creating and displaying the main SCADA window. It first initializes a new window using Toplevel() [8], which is a new instance of the Tkinter window, and sets its properties (e.g., title and background color). Inside this window, the function organizes the layout by creating a *data_frame*, which serves as a container for displaying real-time data. This frame holds different sub-frames for voltage, current, power factor, active power, reactive power, apparent power, and phase angle for each load. Each subframe consists of labels that show the respective data for all three loads. The function also initializes buttons for controlling the loads, as well as a network visualization frame that shows the power system's topology. The *data_frame* is laid out using the grid layout manager, which ensures a flexible and responsive arrangement of widgets. This setup allows for an organized and interactive SCADA interface, where the user can easily monitor the electrical parameters and control the system through the GUI's switches.

The *measurement frame* contained seven buttons, each corresponding to a specific electric measurement. Clicking these buttons would update the plot window to display the relevant data for all three loads in real-time. Furthermore, users could specify the number of data points to be displayed in the plot through a field in the *control frame*, and the plot would adjust accordingly. The Reset button in the control frame allowed users to restore the plot to its default state.

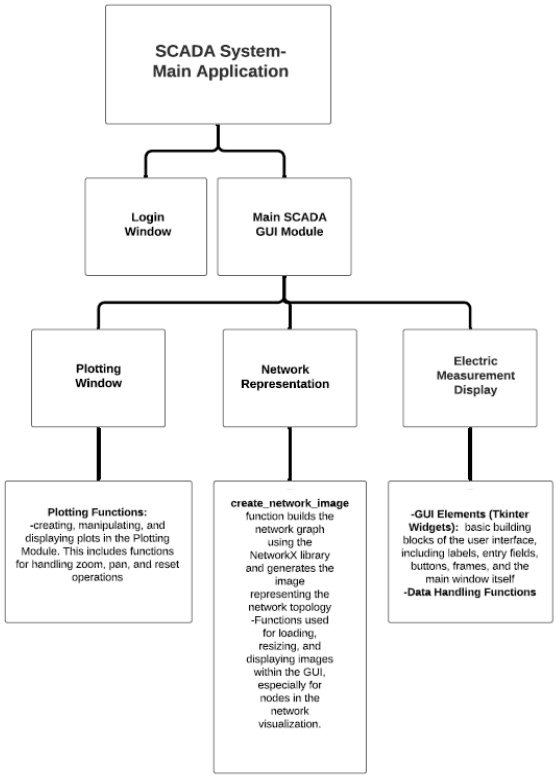


Figure 4.4.2: Software Development Flowchart.

In short, taking the appropriate approach allowed the front-end to provide a user-friendly interface for real-time interaction with the data.

4.5 Back-End Software Development

The back-end of the SCADA system is responsible for managing the data acquisition, processing, and control logic behind the GUI interactions. It was built using Python and integrated various libraries, such as Pillow (PIL) [9] for image handling, and Matplotlib [10] for data visualization. The system architecture was designed to be flexible, incorporating components for load control, data acquisition, and visualization. The flowchart in **Figure 4.1.2** also depicts the data flow and interaction between the SCADA software's components, including how the plotting window and network representation modules interact with the back-end system. This design ensures that real-time data can be efficiently handled and displayed. The Pillow library [9] allows for handling image files, enabling the resizing and manipulation of images like icons and network components that were displayed within the GUI. Matplotlib [10] was primarily used for graphing real-time data, such as power usage and voltage, which were essential for monitoring the electrical system. For the real-time data acquisition, the system retrieved data from the hardware using HTTP requests to the ESP32 microcontroller. The back-end processed the JavaScript Object Notation (JSON) responses from the hardware, which contained live measurements for each load (voltage, current, power factor, etc.).

The data was then dynamically updated and displayed in the GUI using the *after()* [11] method in Tkinter [8].

The functions *show_plot(), update_plot_data()*, and *open_plot_window()* are integral to the back-end of the SCADA system's data visualization. The *show_plot(index)* function displays the selected plot based on the chosen measurement type, updating the plot window with the relevant data for all loads. The *update_plot_data()* function ensures that the plot is continuously updated in real-time, reflecting new data points as they are received, which is essential for dynamic monitoring. Meanwhile, *open_plot_window()* initializes and opens a separate plot window that provides users with interactive controls such as zooming, panning, and resizing to better view the electrical measurements. These functions collectively manage the real-time data updates, ensuring seamless interaction with the system's GUI.

The data logging feature was integrated into the back-end of the SCADA software to capture real-time electrical measurements and save them for further analysis. A function was implemented that saved measurement data into a text file named *live_measurements_log.txt*, located in the same directory as the software file. The data was structured using tab-separated values, with all relevant parameters—such as voltage, current, power factor (PF), active power, apparent power, reactive power, and phase angle from the three loads—logged. Initially, a header row containing the labels for these parameters, along with a timestamp, was written to the file. Subsequently, within the *update_display_from_json* function, the log file was opened in append mode, and each new dataset was added as a new row. The row was formatted with the current time index and the latest measured values for each of the three loads, ensuring that the data was presented with two decimal places, separated by tabs, and followed by a newline for clarity. This operation was carried out once every second, allowing the system to build a historical record of the measurements over time.

The real-time monitoring feature was designed to continuously assess the current drawn by each load and automatically turn off any load whose current exceeded 10 A. This feature was implemented by creating the functions *check_current_load1()*, *check_current_load2()*, *check_current_load3()*, *toggle_switch_auto()*, and *show_message_box_non_blocking()*. In case of overcurrent, a warning message was presented to the user, specifying the affected load and the corresponding current value that triggered it. The logic of this feature is illustrated in **Figure 4.5.1**, which shows how the system evaluates current levels, turns off the corresponding switch if the threshold is exceeded, updates the switch colour to red, and notifies the user through a pop-up message.

For the network representation, the NetworkX [12] library was utilized to efficiently create nodes and edges, enabling a clear visualization of the loads and their interconnections. This visualization was dynamically updated with real-time data, providing users with immediate feedback regarding the operational status of each load. Additionally, toggle switch buttons were created and positioned above the network diagram by using the functions *update_button_color()* and *toggle_switch()*, allowing users to interactively control the state of each load.

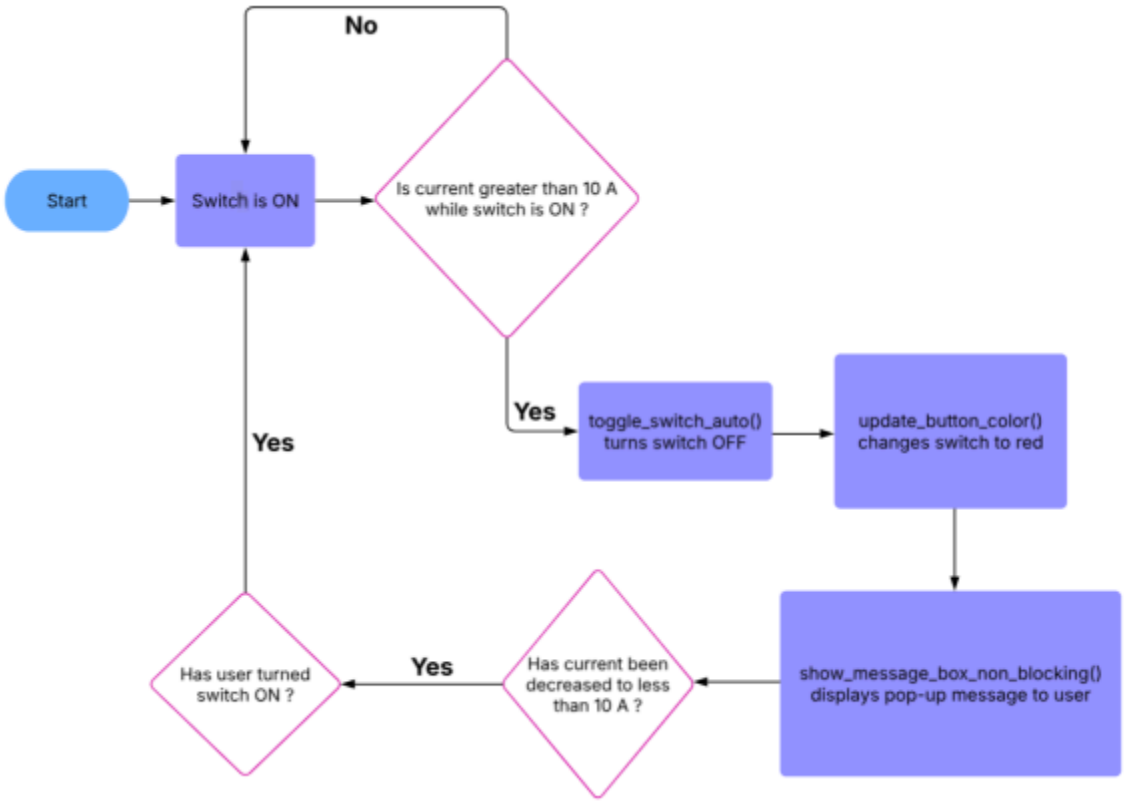


Figure 4.5.1: Flowchart of Real-Time Overcurrent Monitoring and Switch Control Logic.

By taking the appropriate approach, the back-end ensured the correct handling of data and the efficient control of the system.

4.6 Hardware

The AC Load Analyzer is the name of the hardware unit designed to measure and transmit the loads data to the SCADA system. The device consists of a step-down transformer, rectifier, current transformer, male receptacle, female receptacle, dc fan, reset push button, heatsink, ESP32 microcontroller and the PZEM-004T power meter. The following table outlines each of the components mentioned and their respective purposes in the AC Load Analyzer.

Table 4.6.1 Components used to build AC Load Analyzer and their purposes.

Component	Purpose
Step Down Transformer with Rectifier	Used to step down the 120VAC supply and convert it to 5VDC.
3.3VDC Relay	Used to control flow of power to receptacle. Also acts as an isolation device that can disconnect load in the event of an emergency or fault.

PZEM-004T Power Meter	Used to analyze voltage and current from the receptacle and calculate power parameters. Contains built-in voltage transformer for voltage measurements.
ESP 32 with Heatsink	Used for data transmission and control of the load. Heatsink was added to help with heat dissipation from the main chip.
Current Transformer	Used to step down the current to a suitable level for the PZEM-004T to analyze.
Fan	Used to circulate air flow throughout the unit inorder to cool all components.
Reset Button	Used to restore the device to its original state as a response to any malfunction or fault.
Male Receptacle	Designed with prongs that will be used to plug into any standard outlet for power.
Female Receptacle	Custom made receptacle which allows for loads to be plugged into for analyzing,

These components are wired as illustrated in the following wiring diagram to work off of the power supplied from a 120VAC wall outlet. The following circuit diagram showcases the implementation of the built in voltage transformer (PZEM-004T), current transformer and the control relay for one load. The relay is used to control the power supplied to the load. The relay is provided with 3.3V from the ESP32 to power the module, and either a high or low voltage to the input terminal to close or open the contact. When the contact is closed the neutral side of the 120VAC is passed through the relay and when the contact is open the neutral is disconnected. This neutral line is connected through the current transformer which is powered using 5VDC and is providing an analog input to the microcontroller. The voltage is measured across the female receptacle in parallel, and powered by 5VDC supplied from the step-down transformer/rectifier. For each load there are a total of 2 analog inputs from the microcontroller to the PZEM-004T and 1 digital output from the microcontroller to the relay. For the scope of this project, the intention is to have 3 loads and measure and control the primary side of the power supply to each of the loads.

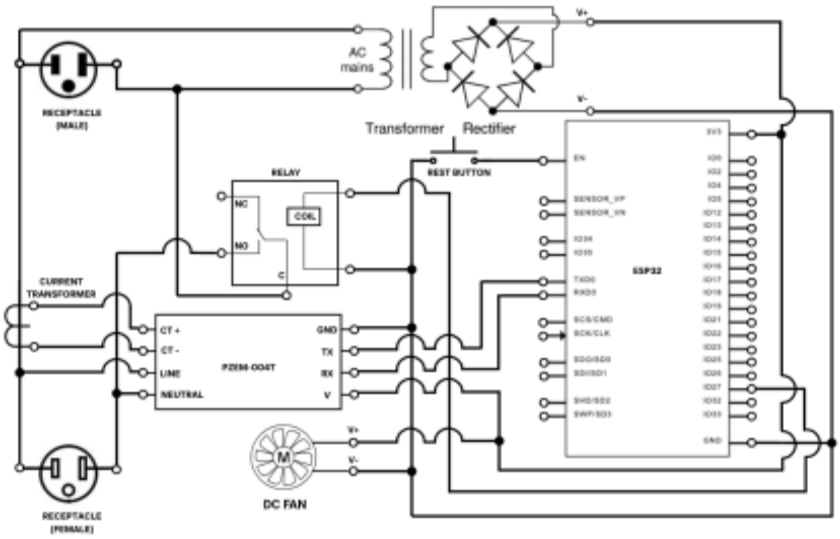


Figure 4.6.1: Wiring Schematic for AC Load Analyzer.

Each AC Load Analyzer is housed in a custom designed enclosure. The design is catered to fit each component perfectly based on its dimensions. The front of the unit houses the custom made receptacle which allows for loads to be plugged into for analyzing, while the back is designed with prongs that will be used to plug into any standard outlet. The right side of the unit has a meshed design to allow for airflow in and out of the unit. In order to print the enclosure of the AC load Analyzer, support material was also designed to help hold up floating material in the design for example where holes were placed for closing the enclosure, for ventilation, and for the front and back receptacle holes. Also because of the extensive print times adding a brim to the first layer of the print of the enclosure helped the unit adhere to the print bed more securely and prevent material from lifting up, vibrating or warping during the first layer print process. Below is an illustration of the three different materials used. The orange represents the design of the actual enclosure, the neon green is the support for floating material, and the dark green is the brim for the first layer.

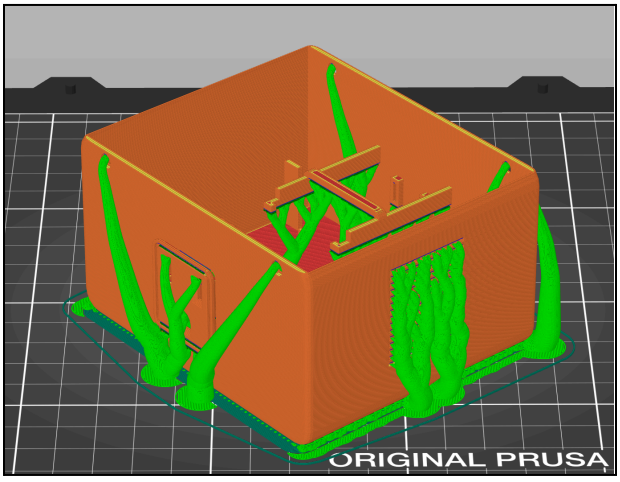


Figure 4.6.2: Slicer Design of AC Load Analyzer with Support Material.

Below are various side perspectives of the unit, showcasing the enclosures features. The images include front, rear, top and back views of the device, each providing a detailed look into how the AC Load Analyzer was assembled.

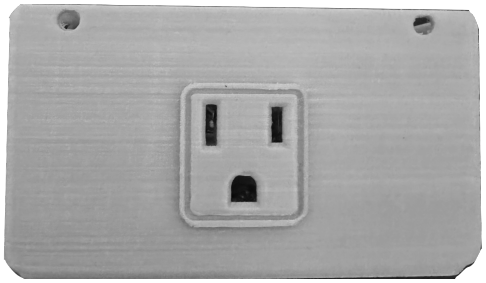


Figure 4.6.3: Front View (Female Receptacle).

Figure 4.6.4: Rear View (Ventilation and Reset Button).

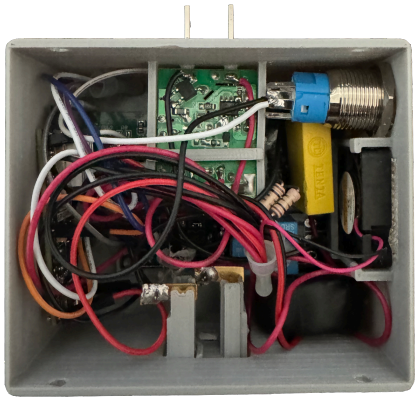


Figure 4.6.5: Top View (Internal Circuitry).

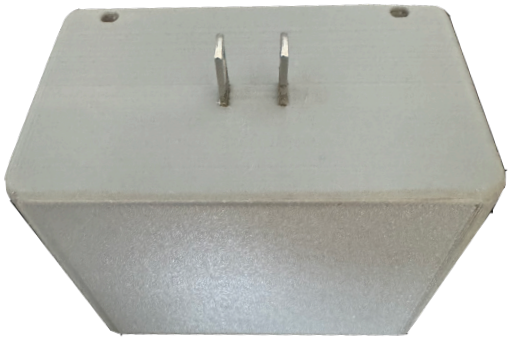


Figure 4.6.6: Back View (Male Plug).

4.7 Data Transmission

For monitoring each load we will have the loads connected to the PZEM-004T and the data will be directly transferred to the connected ESP32. Each ESP32 will function as a transmitter transmitting the measured data to the specified receiver. The communication between transmitter and receiver will be via ESP-NOW protocol which provides data integrity throughout the entire measuring phase.

The process of transferring data from transmitter to receiver follows a four step process:

1. **Initialization:** The ESP32 (transmitter) establishes the serial communication with the power meter and also sets up the ESP-NOW protocol which uses one of the free Wi-Fi channels on the ESP32 that best suits the compatibility of the receiver based on the SSID to reduce interference.

2. **Data Collection:** The ESP32 (transmitter) periodically reads the voltage current and other power readings every 200 ms and then encapsulates the data in a structured JSON message
3. **Transmission:** By using the receiver MAC address the data is sent every 200 ms to the receiver with the structured message packet.
4. **Error Handling:** Each transmission of data is followed by a send response. This message indicates whether the data has been received successfully or if an error occurred. This helps with troubleshooting reliability issues and also allows us to see how often packets are missed.

The following flowcharts showcase the functionality of the transmitter and retriever within the scope of our project. The transmitter is programmed using the following flowchart as the bases for the state machine. Below we can see the expected behaviour of the transmitter after powering up the device. The device continues to loop until power is disconnected.

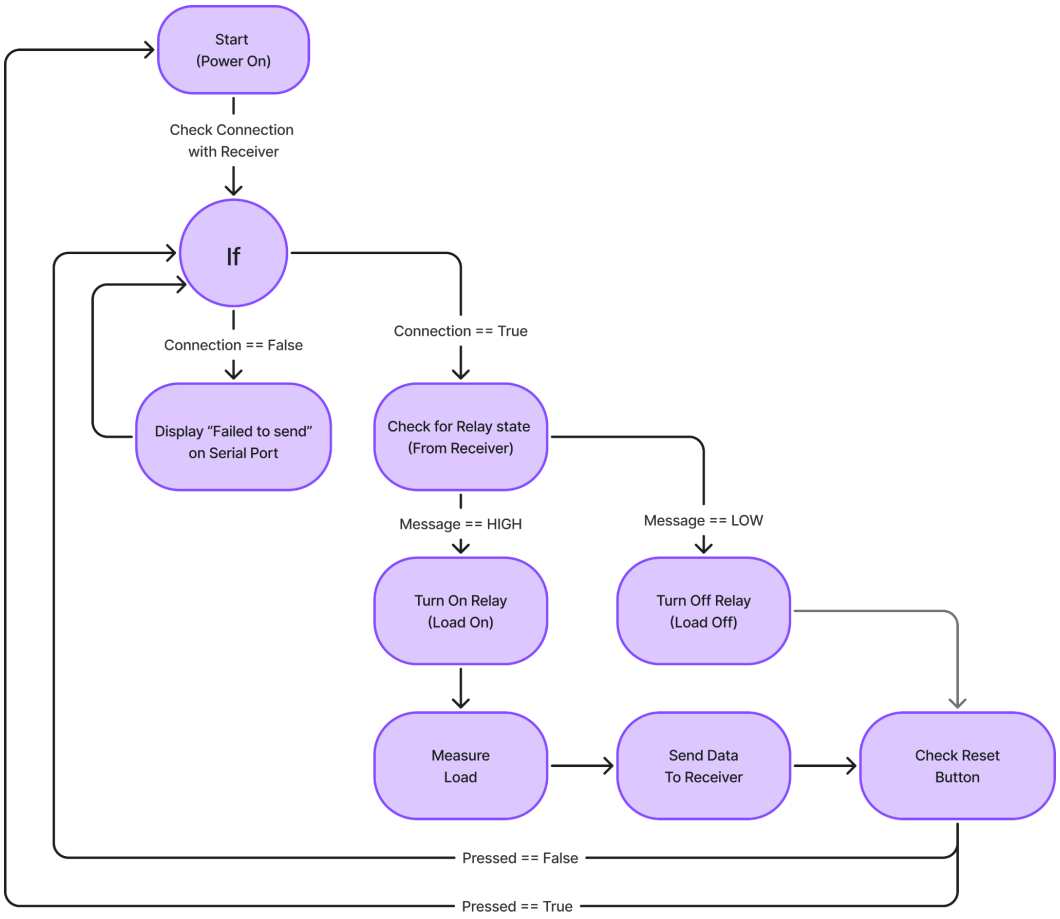


Figure 4.7.1: Transmitter Flowchart.

The following flowchart shows the functionality of the receiver serving as the bridge between the SCADA system and the transmitter. Once powered up the receiver continuously attempts to connect and maintain a connection to the WiFi. After connecting to WiFi the receiver retrieves any data sent from the transmitters and pushes the data to the web server. After pushing data to the server the receiver checks for changes in relay state made by the SCADA system and depending on the state transmits the changes to the according transmitter before starting over the loop.

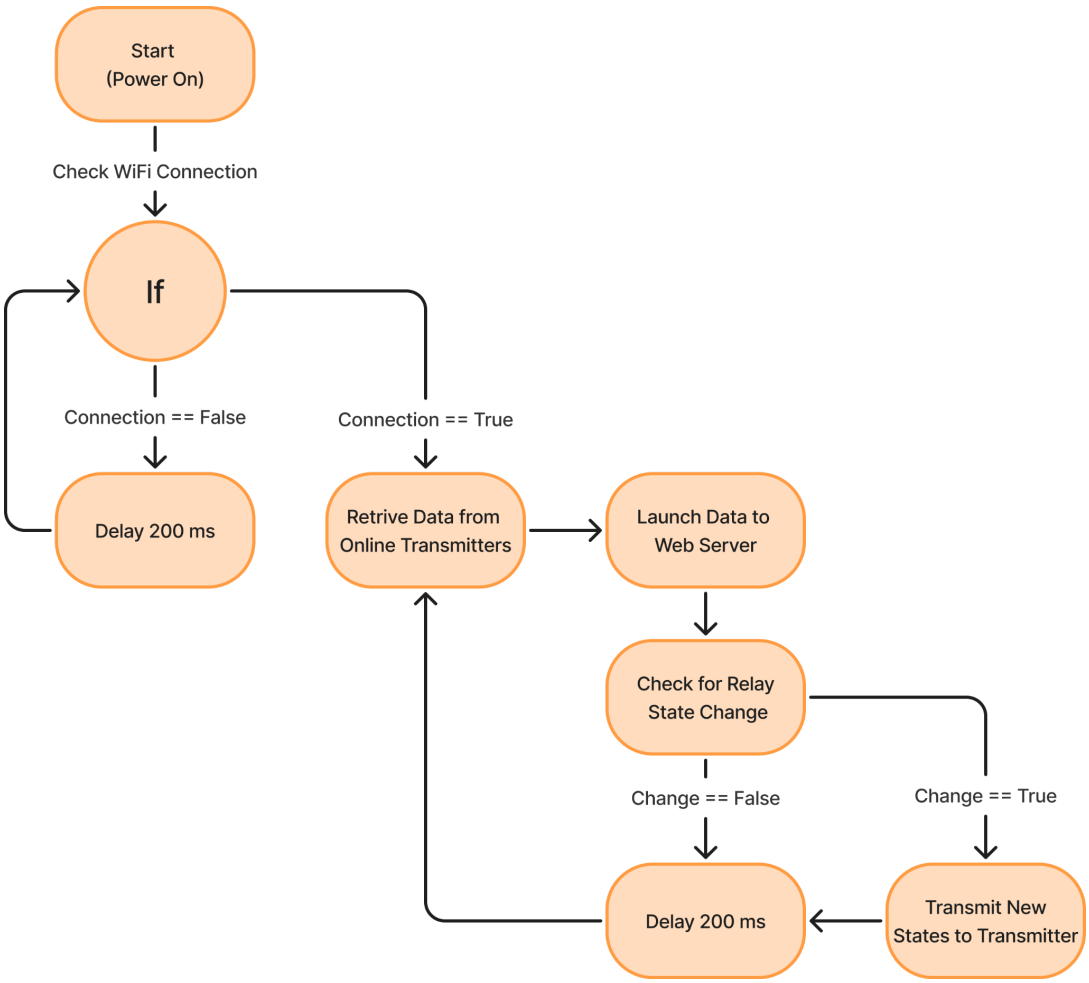


Figure 4.7.2: Receiver Flowchart.

Once data is received by the receiver the receiver creates a web server and transmits data using HTTP request in the form of JSON objects to SCADA. Below is the console for the webserver. The console can be used to see the log of events that the receiver has processed.

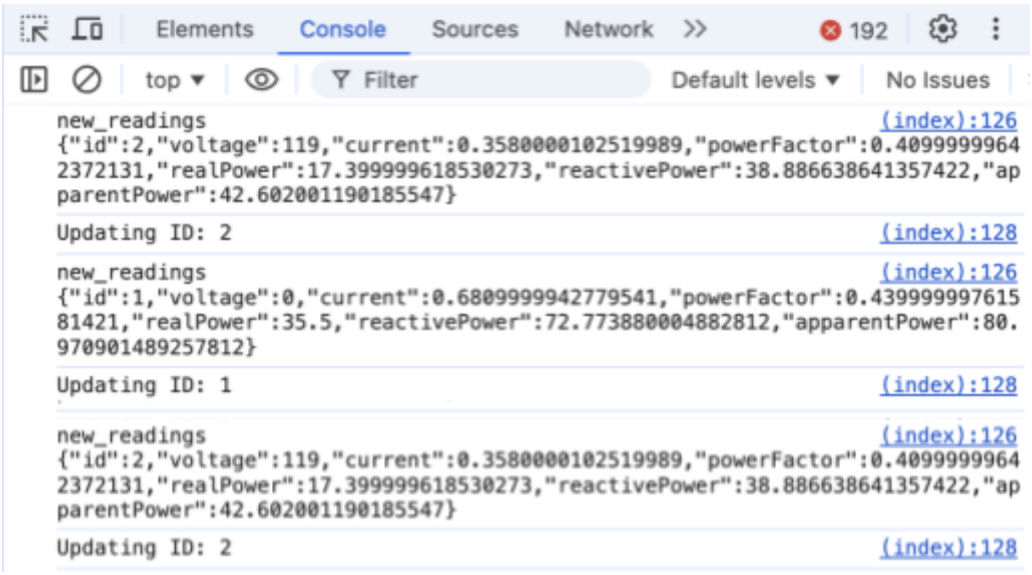


Figure 4.7.3: Web Server Console showing incoming data.

The choice of using JSON format was due to the ease of readability, ease of parsing and the compatibility with web interfaces. Since the ESP32 outputs its own unique webserver JSON format allows for a structure representation of each measurement and control state in a way that is easy to manipulate and extend. An example of the JSON structure is illustrated below.


```
{
 "VoltagePeak": 120.0,
 "RMSVoltage": 115.0,
 "CurrentPeak": 15.0,
 "RMSCurrent": 10.0,
 "DelayIndex": 2,
 "PhaseAngle": 30.0,
 "PowerFactor": 0.95,
 "RealPower": 1100.0,
 "ApparentPower": 1150.0,
 "ReactivePower": 300.0,
 "RelayStatus": "ON"
}
```


Figure 4.7.4: JSON Structure.

When changes are made in the SCADA system, the web server interface will update. Every time the web server is updated, a command will be sent from the web server to the transmitter, updating the local relay state variable in each of the loads. Once the AC Load Analyzer receives the new relay state, it will adjust the relay accordingly. Communication is bidirectional which ensures that changes made in the SCADA interface are accurately reflected in real-time. Below are images of the actual device that was used for testing bidirectional communication for load 2.

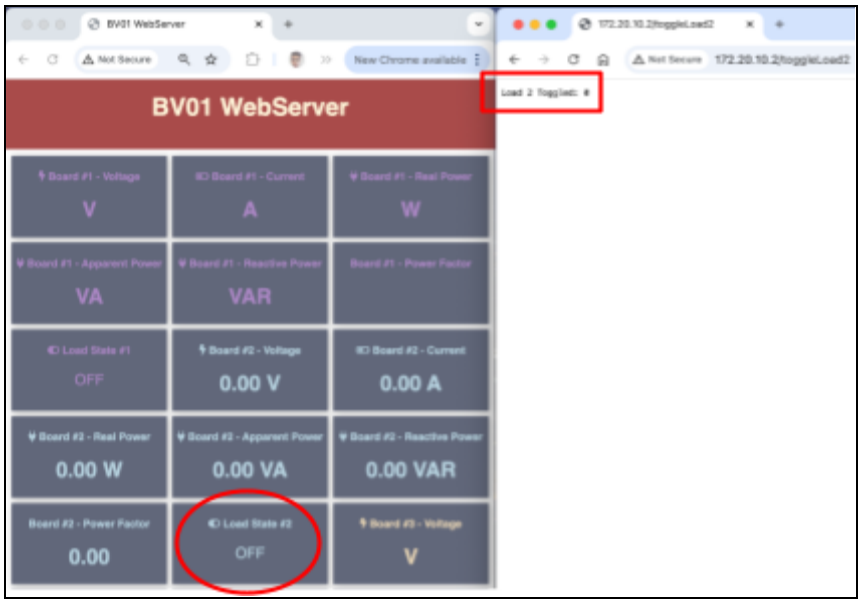


Figure 4.7.5: GET request for toggling (in the off state).

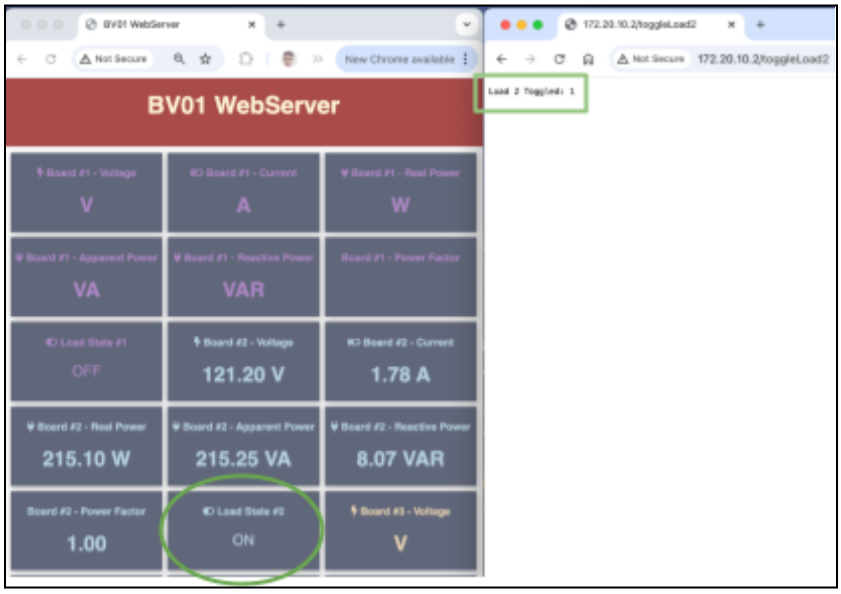


Figure 4.7.6: GET request for toggling (in the ON state).

Each load has its own specific GET request strand and will provide a structured JSON layout of the data. Each load can be accessed by sending a request to the IP address/specific load. For the scope of this project, the 3 GET requests that were used for the SCADA display are as follows.

1. *172.20.10.2/Load1*
2. *172.20.10.2/Load2*
3. *172.20.10.2/Load3*

Below, we can see the GET request made through the browser and the transmitted data that's been sent as a response to the request. For this GET request, the user receives back all of the information pertaining to the load.

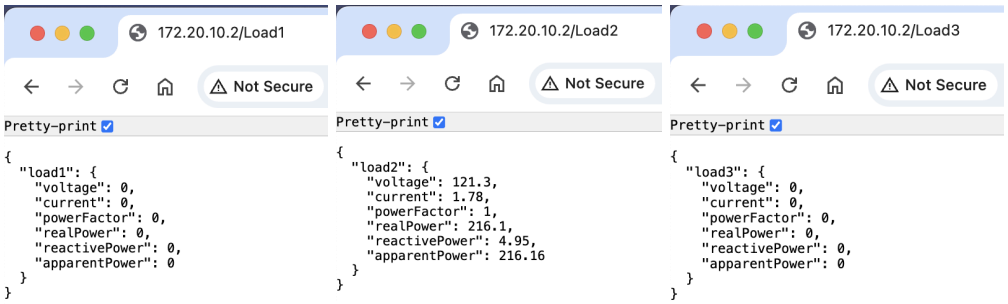


Figure 4.7.7: GET requests to retrieve the load data.

For the SCADA control element of the project, the operator will have the ability to control the state of all 3 loads. In regards to controlling the loads, typically a POST request would be used. However, the nature of a POST request introduces security issues as it makes variables within the web server vulnerable to changes. Since the loads are simply being turned on and off, we can use GET requests as a way to toggle the states. The web server will see the request made by the SCADA system and change the state of the variables accordingly. The three GET requests used for toggling the states are as follows.

1. 172.20.10.2/toggleLoad1
2. 172.20.10.2/toggleLoad2
3. 172.20.10.2/toggleLoad3

By using only GET requests this simplifies the design of the client to interact with the server, without needing form submissions or the client to do any intensive backend for data transmission. GET requests can be efficiently cached by the browser, and caching is beneficial for scalability and performance of the server. Caching is also ideal when making the same request frequently, speeding up response times for the end users. GET requests work well across various platforms and devices without needing special handling. They are supported by all HTTP clients, making them universally applicable for web services, IoT devices, and traditional web applications.

Once the webserver is running the data can be viewed by accessing the static address 172.20.10.2 on the local network. By using a static IP address this ensures the ESP32 web server is always accessed at the same network address. This is critical with the SCADA system, where communication is frequent and a change in IP address can disrupt operations and cause operation downtime while the addresses are re-configured for a dynamic network. Static IPs ensure better compatibility with legacy systems and other components of the network that require fixed addresses for integration. This

SCADA application may not necessarily be a legacy system, however, the hardcoded IP aspect does introduce limitations that are characteristic of legacy systems, particularly in how well it can adapt to changes in the network infrastructure. The server is designed to work across platforms and adjust the sizing automatically to fit the device for the best viewing experience. Below is an example of viewing the web server on an iphone. In the first image the load is off and all parameters are zero. By pressing the toggle button on the server the operator can test the responses of the load and get the load readings before the data is transmitted to the SCADA system.

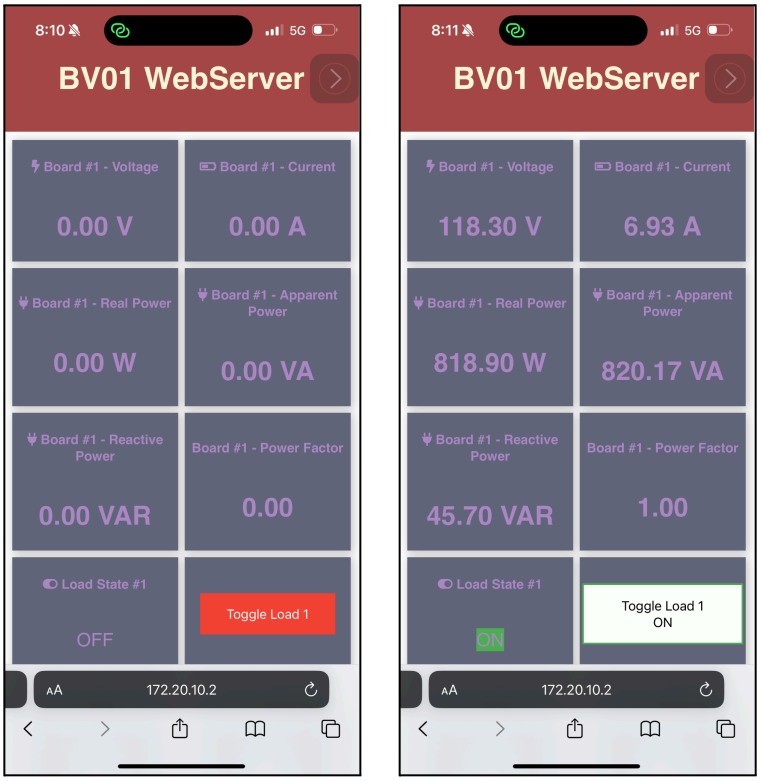


Figure 4.7.8: Iphone Interface Viewing and Controlling Load.

To summarize, the theoretical principles and systematic design process for software and hardware components helped establish a strong foundation for the development of a reliable and efficient SCADA system.

5. Alternative Designs

This section presents alternative design solutions for both the software and hardware components of the SCADA system and includes a comparative analysis of the proposed designs.

5.1 Software

An alternative design for the network representation included a heater, fan, lightbulb, two toggle switches, and a slider switch. In this configuration, the toggle switches were designated to control the fan and heater, while the slider switch was intended specifically for controlling the fan. As illustrated in **Figure 5.1.1**, the design followed a vertical layout, with the power source positioned at the bottom, switches in the middle, and loads (fan, heater, lightbulb) at the top. This configuration was ultimately not selected due to the limited portability and relatively heavy weight of the fan, which made it less practical. Additionally, a horizontal network layout was deemed more user-friendly and preferable, as it provides improved accessibility and greater space above the network diagram for displaying electrical measurements.

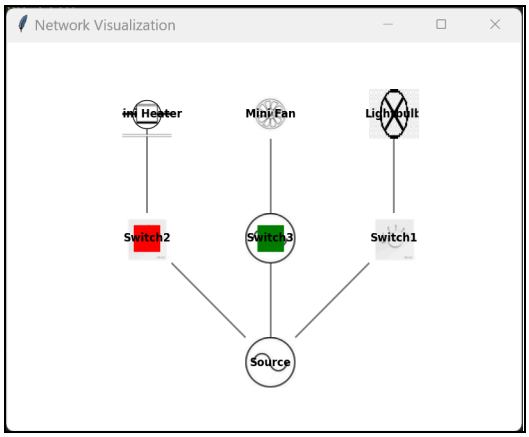


Figure 5.1.1: Alternative design for network representation in the GUI.

5.2 Hardware

In the development of the hardware for measuring and wirelessly transmitting the data to the SCADA system, a key decision was whether to use a pre-assembled power meter or to design a custom power meter. Ultimately, the decision was made to use the pre-assembled board as it saved development time and claimed to offer better reliability. However, in the case of an alternative power meter, the design would consist of voltage and current transformers. These transformers are crucial for stepping down the voltage and current from the main supply to levels that are manageable and safe for the microcontroller to handle for detailed measurements. There are a total of three transformers used within this design: two for sampling voltage and current, and the other used to power the microcontroller. Below is a schematic of the theoretical circuit for powering the ESP32.

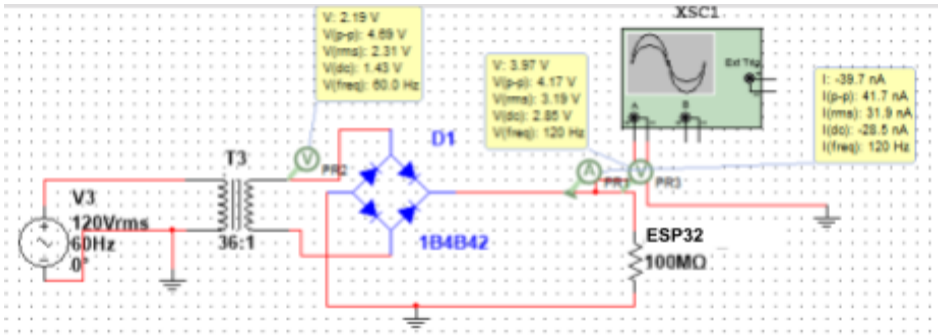


Figure 5.2.1: Schematic of the power supply for ESP32.

One side of the main power would go to the load, while the other side passes through the relay so that the SCADA system can control it. From here, the other end of the relay will pass through the current transformer so that the current can be measured and the voltage transformer must be connected in parallel to the ac supply to measure the voltage. A layout of the circuit is illustrated below.

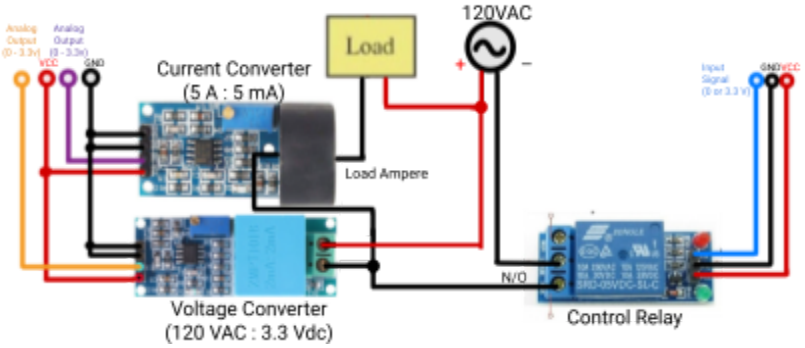


Figure 5.2.2: Layout of measurement transformers and control relay.

A prototype of the design was put together within a custom 3D modeled enclosure to show how the unit would look. Below is a labeled illustration of the prototype.

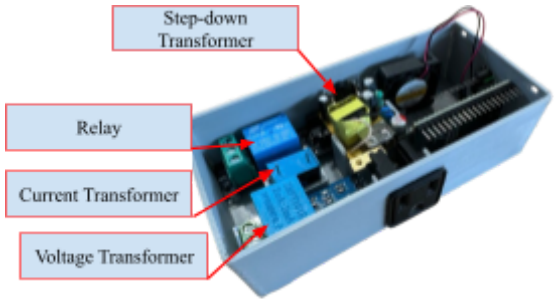


Figure 5.2.3: Prototype of Alternative Hardware design.

The software alternative design featured a vertically layout network with both toggle and slider switches to control the loads. This design was later scrapped and altered due to hardware limitations and usability concerns, leading to the current horizontal layout with only toggle switches. For hardware, the custom meter using voltage and current transformers was replaced by the pre-assembled board which was ultimately chosen to save time and enhance reliability.

6. Material/Component list

This section outlines the costs of all project components, including unit costs and quantities.

Table 6.1 Cost breakdown for purchasing three PZEM-004T V3 power meters.

Components	Price	Quantity	Total
PZEM-004T V3	$24.45	3	$73.35

Table 6.2 Cost breakdown for purchasing alternative power meter components.

Components	Price	Quantity	Total
ZMPT101B (Voltage Sensor)	$0.40	3	$1.20
ZMCT103C (Current Sensor)	$0.76	3	$2.28

Table 6.3 Cost breakdown for purchasing AC load analyzer components.

Components	Price	Quantity	Total
ESP32 with WiFi and Bluetooth	$3.00	3	$9.00
Step-down Transformer with Rectifier	$1.75	3	$5.25
5 Volt DC Fan	$1.30	3	$3.90
SRD-5VDC-SL-C (Relay)	$0.19	3	$0.57
Momentary Push Button	$0.95	3	$2.85
Male Receptacle Hardware	$0.21	3	$0.63
Female Receptacle Hardware	$0.20	3	$0.60
Heatsink	$0.50	3	$1.50
1 Enclosure (Quantity in grams)	$0.15	76	$11.40

Table 6.4 Price Comparison for Different Configurations.

Configuration	Price for 1 Unit	Price for 3 Units
Pricing without Power Meter	$19.50	$58.50
Pricing with PZEM-004T V3	$43.95	$131.85
Pricing with Alternative Design	$20.66	$61.98

By calculating the total costs for all materials and components, it was easier to manage budget restrictions and make financial decisions.

7. Measurement and Testing Procedures

This section describes the methodologies used to measure and test the functionality and performance of the software and hardware components of the SCADA system.

7.1 Software

A proper design of the SCADA system software was critical to ensure that the system would perform as intended under varying operating conditions. The SCADA system was tested in two phases: initially, using simulated mock data, and later with real hardware data. This approach ensured the system's functionality and reliability before integrating it with the actual hardware setup.

Phase 1: Testing SCADA Software with Simulated Data

Initially, the SCADA system was tested using simulated data. The mock data generation function, *generate_mock_reading()*, was created to simulate electrical readings such as voltage, current, power, energy, frequency, and power factor. These readings were formatted as JSON strings to mimic the data structure used by the ESP32. The mock data was processed by the system in the same way as real hardware data, ensuring that the system's display logic, data handling, and dynamic updates worked seamlessly. During this phase, the SCADA system was able to dynamically update the GUI with simulated measurements. The data is updated periodically using the Tkinter [8] *after()* [11] method, which calls *update_display_from_json()* every second to refresh the displayed values. This ensures that the SCADA system continuously reflects the latest readings from the hardware as well. The simulated data allowed for extensive testing of the GUI, including the responsiveness of dynamic plots, switch states, and the overall interaction with the measurement parameters.

Figure 7.1.1 illustrates the main SCADA window, where mock data was used to display key parameters such as voltage, current, power factor, active power, apparent power, reactive power, and phase angle for each of the three loads. These values were dynamically updated to simulate real-time system monitoring, providing a clear interface for monitoring the performance of each load in the power distribution network. The plot window, shown in **Figure 7.1.2**, was tested using the same simulated data to display the active power of each load over time. The graph demonstrates how the active power fluctuates for each load (Load 1, Load 2, Load 3) during a 20-second period. The system correctly visualized the real-time changes for all the electric measurements being monitored, supporting the expected system functionality for zooming, panning, resetting, and data point adjustments.

To evaluate the system's automatic switch toggling functionality in response to overcurrent conditions, the *generate_mock_reading()* function was modified to simulate four distinct scenarios: normal operation, overcurrent, reset, and recovery. For each scenario, the current value was randomly selected within a specific range to reflect the expected behavior of the system. Specifically, values between 6A and 9A represented normal operation; values between 11A and 13A indicated overcurrent conditions; a value

of exactly 0A simulated an ESP32 reset or absence of current; and values between 5.5A and 7.5A represented system recovery following an ESP32 reset.

This phase allowed for the early identification of potential issues and provided a solid foundation for further testing with real hardware data in later stages of the project.

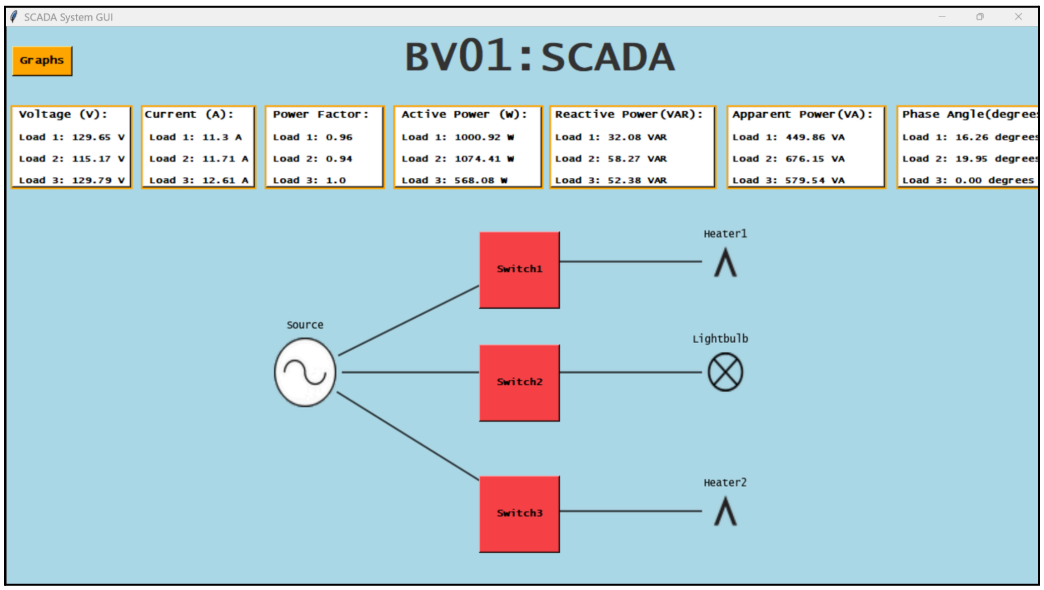


Figure 7.1.1: GUI simulation showing main SCADA window for three loads using mock data.

Figure 7.1.2: GUI simulation showing plot window for three loads using mock data.

Phase 2: Testing SCADA Software with Hardware Data

Once the SCADA system was fully tested with simulated data, it was integrated with the real hardware setup. The ESP32 devices, acting as web servers, were programmed to send real-time measurements in JSON format, which the SCADA system retrieved using HTTP requests. The functions *load_json_data1(), load_json_data2()*, and *load_json_data3()* were used to fetch the real-time data from the ESP32 for each load (Load1, Load2, and Load3). These functions retrieved the actual values for parameters like voltage, current, power factor, real power, apparent power, reactive power, and phase angle. The data from the ESP32 devices was parsed and processed in real-time, updating the SCADA system's GUI, resembling the method that was utilized for testing during the simulated phase. This transition to real hardware allowed for verification of the system's ability to handle live data, ensure communication between the hardware and the software, and validate the system's performance under actual operating conditions. During the testing phases, the system's ability to log the data into a text file for historical record-keeping was also tested, ensuring that all real-time measurements were continuously saved for future analysis.

To evaluate the system's automatic switch toggling functionality in response to overcurrent conditions within the hardware, the *toggle_load_state()* function was developed and integrated into both the *toggle_switch()* and *toggle_switch_auto()* functions. The primary purpose of *toggle_load_state()* was to initiate an HTTP GET request to the designated URL corresponding to a specific load, thereby changing its operational state to either on or off. This function was invoked after the execution of *update_button_color()* within both *toggle_switch()* and *toggle_switch_auto()*, ensuring that the visual representation of the switch state within the network was updated prior to modifying the actual load state on the back-end.

By first testing with simulated data and then validating the system with real hardware, it was possible to ensure that the SCADA system would operate as expected in a real-world environment, with both mock and live data sources.

7.2 Hardware

Initial Design for Communication Testing:

The first design was developed to test the communication between the microcontroller (ESP32) and the power meter (PZEM-004T V3). In this setup, the power connections were carefully established to ensure proper operation. The 5V output pin on the PZEM-004T power meter was connected to the 3.3V input pin on the ESP32, as the ESP32 operates at a 3.3V logic level. Additionally, the ground (GND) pins on both the ESP32 and the PZEM-004T were connected to establish a common reference, ensuring proper communication and preventing potential voltage-related issues.

For the communication interface, the RX (Receive) pin on the power meter was connected to GPIO pin 5 on the ESP32, while the TX (Transmit) pin on the power meter was connected to GPIO pin 4 on the ESP32. These connections enabled bidirectional data exchange between the two devices. The communication protocol used was Modbus RTU, a widely adopted and robust protocol for real-time monitoring and control applications.

Modbus operates on a master-slave architecture, where the ESP32 functions as the master and the PZEM-004T as the slave device.

This design was specifically implemented to test the compatibility and communication reliability between the two components. Proper wiring and configuration were critical for ensuring successful data exchange. By establishing this setup, the foundational communication framework for the system was verified, laying the groundwork for the integration and monitoring of electrical loads. Communication between the power meter and the microcontroller was successfully established, with the resulting data output illustrated in **Figure 7.2.1**, which shows the initial circuit configuration without a connected load.

Following successful communication as shown in **Figure 7.2.3**, the next step involved verifying the accuracy of the measurements obtained from the power meter. To achieve this, a load in the form of a mini heater was connected to the system, as shown in **Figure 7.2.2**. The measured values for voltage, current, power, and power factor (PF), illustrated in **Figure 7.2.4**, were found to be accurate when compared with the specifications provided on the mini heater's packaging. This confirmed the system's capability to deliver reliable and precise real-time electrical measurements under load conditions.

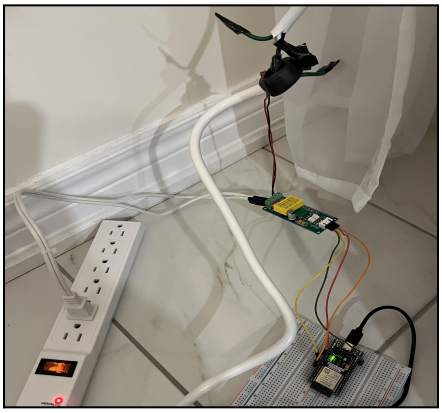


Figure 7.2.1: Circuit Setup for testing the power meter with no load.

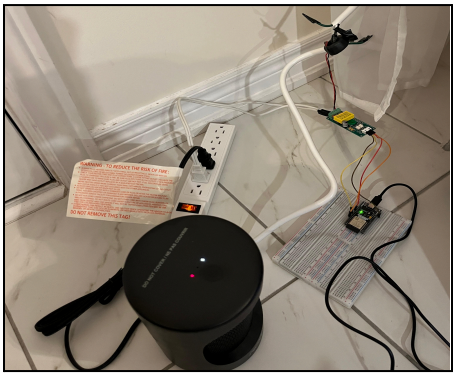


Figure 7.2.2: Circuit Setup for testing the power meter with a mini heater.

```
Custom Address:1
Voltage: 119.90V
Current: 0.02A
Power: 0.50W
Energy: 0.000kWh
Frequency: 60.0Hz
PF: 0.18
```

Figure 7.2.3: Serial monitor output when no loads are connected.

```
Custom Address:1
Voltage: 118.20V
Current: 5.00A
Power: 589.00W
Energy: 0.007kWh
Frequency: 60.0Hz
PF: 1.00
```

Figure 7.2.4: NOMA Mini heater serial monitor output.

7.3 Data Transmission

The efficiency and reliability of data transmission between the transmitter and receiver was tested. The goal for testing was to make sure that the data packets arrive at the receiver correctly. In order to verify this, it was necessary to implement additional information pertaining to the packet status at each point of the transmission process. At the transmitter end after data is measured, the packet is transmitted to the receiver. Based on this interaction with the receiver, the transmitter is able to identify whether the transmission took place successfully, and if the packet delivered successfully. In **Figure 7.3.1**, transmission takes place roughly 200ms apart. At this sampling rate it is observed that all packets deliver without failure. Earlier in the project when sampling at rates faster than 200ms, the packets were not able to send fast enough and on average 6 out 10 packets would be sent successfully.

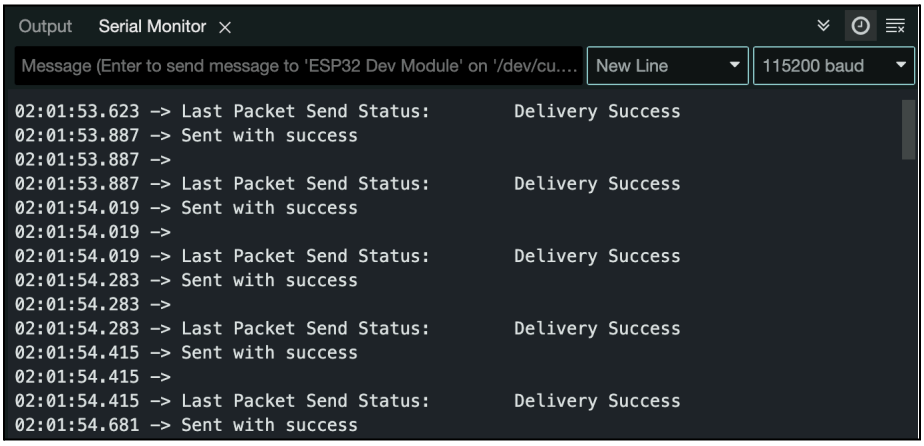


Figure 7.3.1: Transmitter Console when Receiver is Connected.

If the receiver is outside the range of the transmitter or not connected to power the data fails to be received and the transmitter displays a failed message. The data will not be marked as received because of the missing acknowledgement from the receiver. Below the transmitter outputs the message “Delivery Fail” when the data is not processed by the receiver.

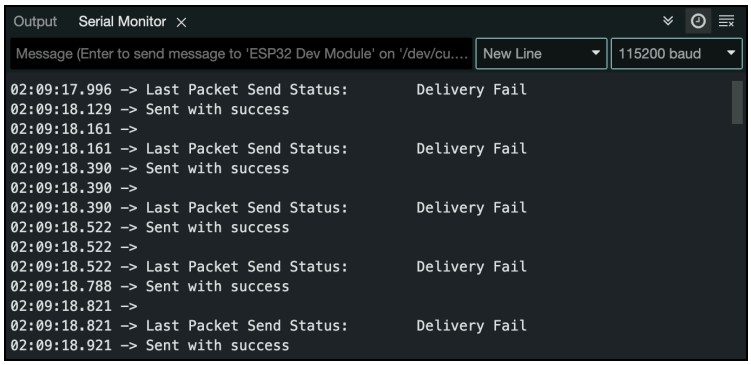


Figure 7.3.2: Transmitter Console when Receiver is Disconnected.

Looking at the receiver side, since there are three loads that transmit to the receiver the formatting is slightly different. Instead of having a static approach to sending data the receiver is responsive and only takes actions after being prompted by a transmitter. The receiver displays the MAC address of the transmitter along with the designated board id and measured data from the power meter. The receiver only displays data whenever a transmission occurs. Until a transmission occurs the receiver waits and listens for a potential peer device.

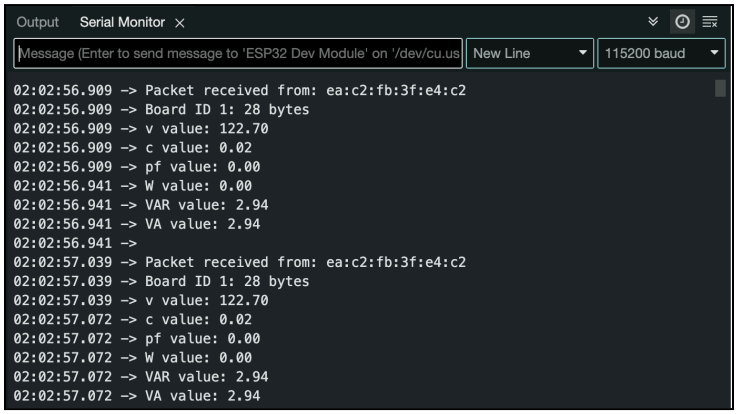


Figure 7.3.3: Receiver Console when data is sent.

This data is then broadcasted to the web server which can then be accessed from the SCADA software.

This section served to provide the methodology for measuring and controlling the loads by combining software and hardware components of the SCADA system.

8. Performance Measurement Results

This section presents the results obtained from testing the software and hardware components of the SCADA system under a range of testing scenarios.

8.1 Software

The SCADA system successfully achieved real-time monitoring and visualization of electrical parameters across three loads. The user interface, developed using Tkinter, displayed clearly labeled sections for voltage, current, power factor, real power, reactive power, apparent power, and phase angle. These values were updated every second, ensuring smooth and timely data presentation, as shown in **Figure 8.1.2**.

Upon launch, the system presented a secure login screen shown in **Figure 8.1.1**, providing controlled access to the monitoring environment. Data was fetched from local hardware servers via JSON files and parsed using dedicated functions. The results were consistently reflected in both the GUI and internal arrays, enabling synchronized real-time updates across both display and plotting components.

The plotting interface, shown in **Figure 8.1.3**, delivered high-resolution, live-updating graphs for all electrical parameters. It supported interactive controls such as zoom, pan, and reset, allowing users to analyze trends within a default 10-second time window. Throughout testing, the plotting system maintained stable performance with no graphical lag, even under continuous operation.

To allow for further analysis and ensure recordkeeping is accurate, the updated parameter values were logged to a text file every second, as shown in **Figure 8.1.4**. This logging function continuously operated in the background without causing interruptions to the responsiveness of the GUI or interfering with real-time updates to the plots. The ability to consistently capture data confirmed that the system remains stable under continuous operation and provided a reliable record of the electrical behavior monitoring.

Along with manual switch control, the SCADA system integrated an automatic toggling mechanism for overcurrent protection. This functionality continuously monitored the current drawn by each load and reliably triggered a protective response when the measured value exceeded the predefined threshold of 10 A. During testing, the system responded within one second of detecting an overcurrent event by automatically turning off the affected switch, updating the button color from green to red in the GUI, and displaying a warning message to the user to indicate the specific load and the current value that caused the shutdown. This performance was consistent through rigorous testing, confirming that the protection logic was reliable and responsive. **Figure 8.1.5** shows an example of a load being automatically turned off due to an overcurrent condition, and the pop-up message that is displayed to the user. These observations highlight that the system is able to consistently operate safely and provide the user with timely feedback.

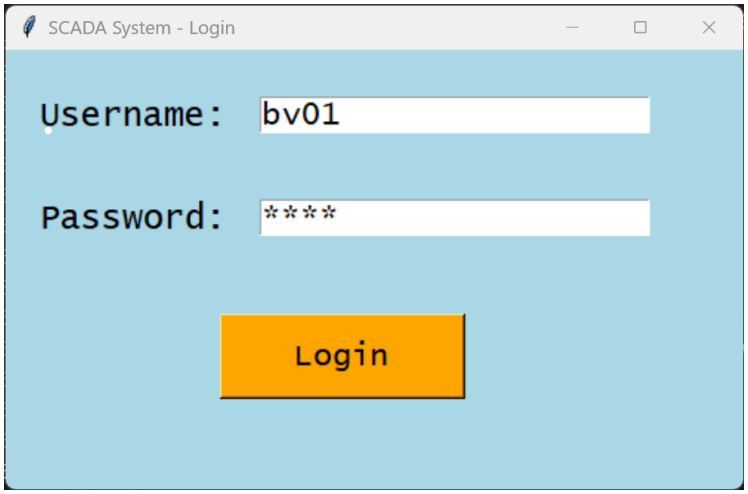


Figure 8.1.1: Login window of the SCADA software during hardware testing with three loads.

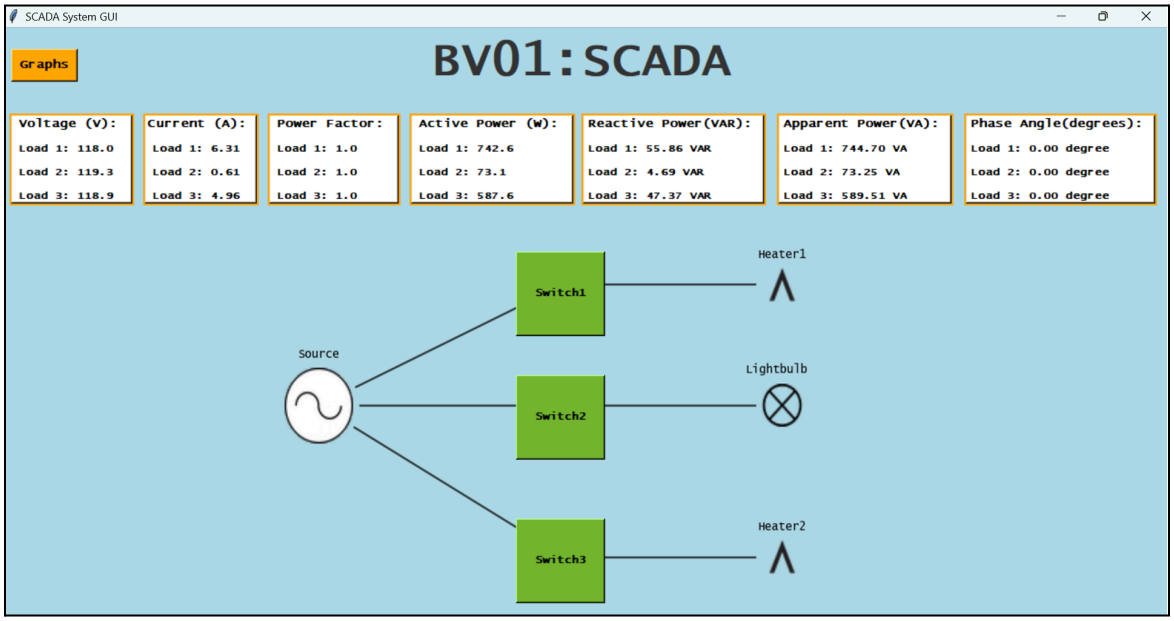


Figure 8.1.2: Main SCADA window of the SCADA software during hardware testing with three loads.

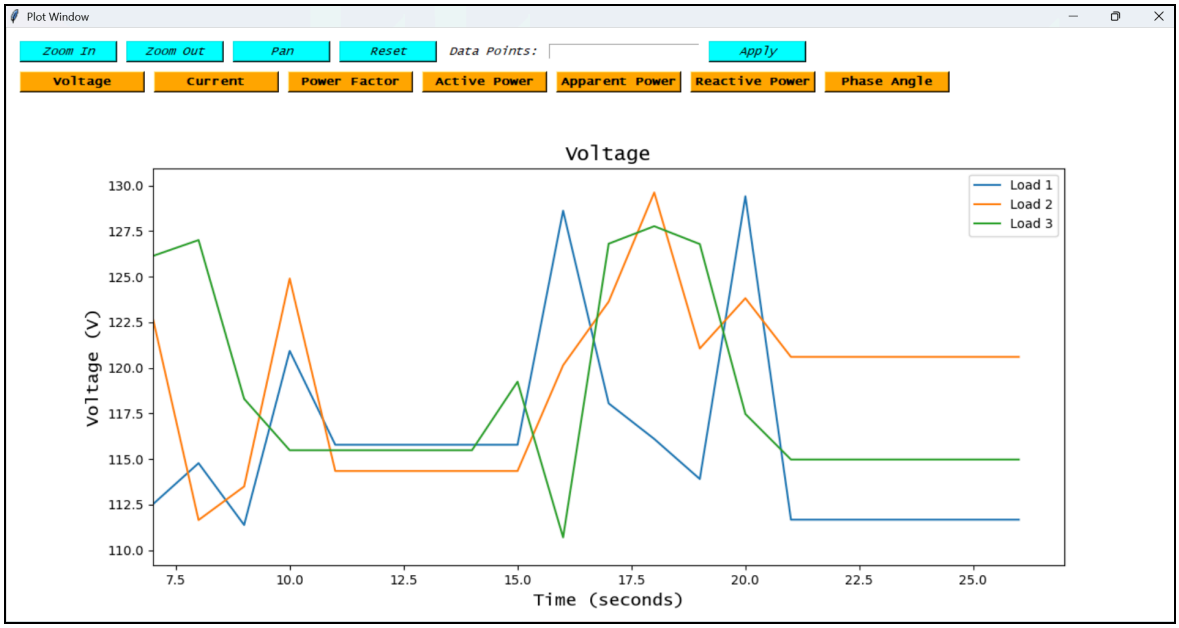
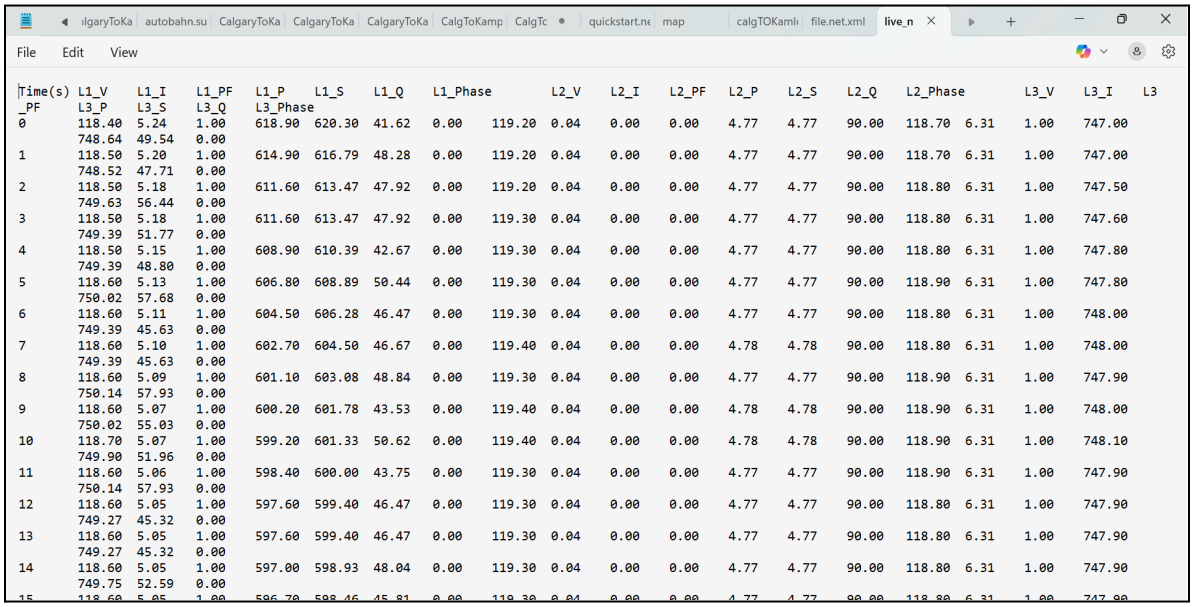


Figure 8.1.3: Plot window of the SCADA software during hardware testing with three loads.



File Edit View

Time(s)	L1_V	L1_I	L1_PF	L1_P	L1_S	L1_Q	L1_Phase		L2_V	L2_I	L2_PF	L2_P	L2_S	L2_Q	L2_Phase		L3_V	L3_I	L3_PF	L3_P	L3_S	L3_Q	L3_Phase
0	118.40	5.24	1.00	618.90	620.30	41.62	0.00	119.20	0.04	0.00	0.00	4.77	4.77	90.00	118.70	6.31	1.00	747.00		748.64	49.54	0.00	
1	118.50	5.20	1.00	614.90	616.79	48.28	0.00	119.20	0.04	0.00	0.00	4.77	4.77	90.00	118.70	6.31	1.00	747.00		748.52	47.71	0.00	
2	118.50	5.18	1.00	611.60	613.47	47.92	0.00	119.20	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	747.50		749.63	56.44	0.00	
3	118.50	5.18	1.00	611.60	613.47	47.92	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	747.60		749.39	51.77	0.00	
4	118.50	5.15	1.00	608.90	610.39	42.67	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	747.80		749.39	48.80	0.00	
5	118.60	5.13	1.00	606.80	608.89	50.44	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.90	6.31	1.00	747.80		750.02	57.68	0.00	
6	118.60	5.11	1.00	604.50	606.28	46.47	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	748.00		749.39	45.63	0.00	
7	118.60	5.10	1.00	602.70	604.50	46.67	0.00	119.40	0.04	0.00	0.00	4.78	4.78	90.00	118.80	6.31	1.00	748.00		749.39	45.63	0.00	
8	118.60	5.09	1.00	601.10	603.08	48.84	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.90	6.31	1.00	747.90		750.14	57.93	0.00	
9	118.60	5.07	1.00	600.20	601.78	43.53	0.00	119.40	0.04	0.00	0.00	4.78	4.78	90.00	118.90	6.31	1.00	748.00		750.02	55.03	0.00	
10	118.70	5.07	1.00	599.20	601.33	50.62	0.00	119.40	0.04	0.00	0.00	4.78	4.78	90.00	118.90	6.31	1.00	748.10		749.90	51.96	0.00	
11	118.60	5.06	1.00	598.40	600.00	43.75	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.90	6.31	1.00	747.90		750.14	57.93	0.00	
12	118.60	5.05	1.00	597.60	599.40	46.47	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	747.90		749.27	45.32	0.00	
13	118.60	5.05	1.00	597.60	599.40	46.47	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	747.90		749.27	45.32	0.00	
14	118.60	5.05	1.00	597.00	598.93	48.04	0.00	119.30	0.04	0.00	0.00	4.77	4.77	90.00	118.80	6.31	1.00	747.90		749.75	52.59	0.00	

Figure 8.1.4: Snippet of live_measurements_log.txt showing logged measurement data.

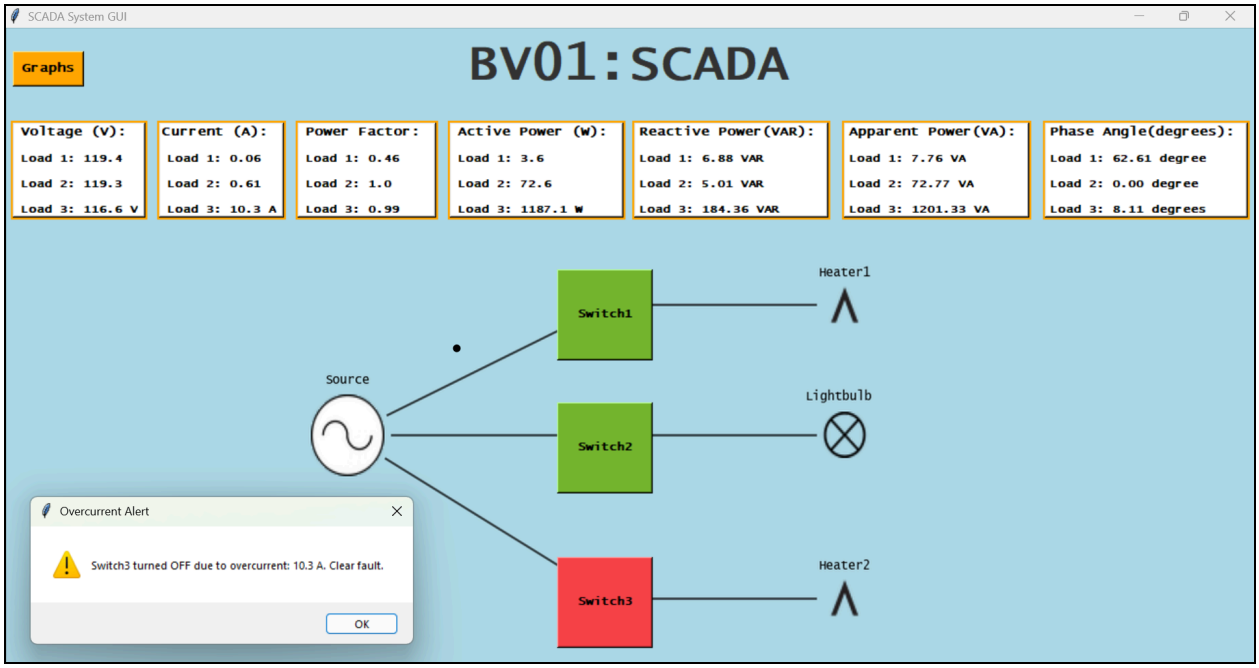


Figure 8.1.5: Automatic switch-off of a load due to overcurrent detection, with warning message.

8.2 Hardware

The first design was implemented to validate the communication between the microcontroller (ESP32) and the power meter (PZEM-004T V3), serving as the foundational step for the real-time monitoring capabilities required in the SCADA system. Once the physical connections and communication protocol (Modbus RTU) were properly configured, successful data exchange was confirmed.

To verify the accuracy of the measurements, a mini heater was connected to the system as a test load. The voltage, current, power, and power factor (PF) readings were cross-validated with the specifications provided on the mini heater's packaging. The results demonstrated a high level of accuracy, confirming that the system can reliably monitor electrical parameters under load conditions.

In parallel with the physical testing, a simulation was conducted to further validate system performance. Within the simulated environment, the Modbus RTU protocol was modeled, and data communication between the ESP32 and a virtual power meter was tested under various load scenarios. The readings obtained during simulation—for example, voltage around 120V, current near 5A, power approximately 590W, and a power factor close to 1.00—closely mirrored the real-world results, reinforcing the system's consistency and reliability.

These findings play a crucial role in the broader context of the SCADA software developed for this project. The SCADA system relies on accurate real-time data acquisition to provide meaningful visualizations and control capabilities. The verified communication and measurement accuracy ensure that the software can display precise electrical parameters on its

user interface, enabling operators to monitor system health, detect anomalies, and make informed decisions in real time. This seamless integration between hardware and software not only validates the SCADA system's effectiveness but also sets the stage for future scalability and the incorporation of additional smart monitoring features.

9. Analysis of Performance

This section analyzes the overall performance of the SCADA system based on key metrics such as responsiveness, reliability, accuracy, and real-time data handling capabilities.

The SCADA system demonstrated effective real-time monitoring and responsive visualization of electrical parameters across all three loads. The performance of the system remained consistent throughout testing, with smooth data updates occurring every second. The frequent updates were effective for tracking in real-time without overwhelming users or the GUI. All key electrical parameters—voltage, current, power factor, real power, reactive power, apparent power, and phase angle—were clearly displayed and synchronously updated with internal data structures.

The stable user interface allowed the software to maintain system responsiveness. The creation of the GUI using Tkinter [8], ensured an interactive and visually consistent interface, even as various display sections and plots were simultaneously being updated. The login screen added an additional layer of security to ensure that only authorized users could initiate monitoring.

Modular functions efficiently handled data parsing from JSON files, to ensure the synchronization of GUI values, internal data arrays, and real-time plots. The reliable performance of the live-plotting window offered smooth zoom, pan, and reset functionalities. There was no graphical lag or data loss observed when the system was operating for long periods, which ensured the system was well-optimized for continuous usage.

The logging mechanism of the system further validated its performance. By recording values every second in a background process, it allowed users to analyze the data without causing interruptions to front-end performance. The seamless operation of the logging function alongside the GUI and plotting window confirmed the ability of the system to handle multiple tasks occurring simultaneously.

The automatic toggling mechanism was essential for creating a safe and reliable SCADA system. By continuously monitoring the current drawn by each load, the system was able to quickly detect and respond to overcurrent conditions. The automatic shutdown was consistently triggered when the current exceeded the 10 A threshold, and the affected switch was immediately turned off, with its visual state updated in the GUI. The corresponding warning message was displayed without causing interruptions to GUI, which confirms that the protection logic was able to operate effectively. This behavior showed that the system can handle abnormal electrical conditions and alert users in real time, even during continuous monitoring. The successful handling of overcurrent events validated the responsiveness and robustness of the automatic protection feature, showing that the SCADA system can enforce safety protocols as well as visualize the behaviour of the system.

Overall, the SCADA system was able to maintain consistent updates, preserve the quality of user interactions, and continuously capture data without conflicts.

10. Conclusion

This section provides a comprehensive conclusion to the SCADA system project by summarizing the work completed, evaluating the alignment between the initial objectives and final outcomes, and outlines potential areas for future improvement and continued development.

A complete SCADA system was successfully designed with the integration of hardware and software to allow users to monitor and control real-world electrical loads. The final design aligns with the initial project objectives, fulfilling key functional and performance requirements. The implementation of a user-friendly interface based in Python provided real-time visualization of key parameters such as voltage, current, apparent power, real power, reactive power, power factor, and phase angle, along with interactive features like pan, zoom, reset, and control of loads. The validation using a physical test setup demonstrated the system's performance under normal and fault conditions, confirming reliable communication between the ESP32, sensors, and relays. To protect both the system and connected devices, safety mechanisms, including automatic shutdown during overcurrent or overheating, were incorporated. The combination of sensor calibration and ESP32 processing allowed for the transmission of data via HTTP for real-time display of accurate power measurements. Additionally, the system was able to generate text reports with key metrics, trend analysis, and fault logs, all of which in combination enhance the SCADA system to be comprehensive and reliable.

In conclusion, the SCADA system can be further enhanced by expanding support for additional loads and incorporating more advanced interactive features within the GUI.

11. References

[1] A. Rehman, “SCADA and its application in Electrical Power Systems,” AllumiaX Engineering, https://www.allumiax.com/blog/scada-and-its-application-in-electrical-power-systems (Accessed Nov. 25, 2024).

[2] EEP - Electrical Engineering Portal and E. Csanyi, “SCADA as the heart of a distribution management system: EEP,” EEP - Electrical Engineering Portal, https://electrical-engineering-portal.com/scada-as-heart-of-distribution-management-system (Accessed Nov. 25, 2024).

[3] "PZEM-004T V3.0 Datasheet and User Manual," InnovatorsGuru, [Online]. Available: https://innovatorsguru.com/wp-content/uploads/2019/06/PZEM-004T-V3.0-Datasheet-User-Manual.pdf. [Accessed: Nov. 28, 2024].

[4] S.-A. TcaciucGherasim, "A Solution for an Industrial Automation and SCADA System," in Proc. 2022 Int. Conf. Exposition on Electrical and Power Engineering (EPE), Iasi, Romania, 2022, pp. 297-300, doi: 10.1109/EPE56121.2022.9959775.

[5] J. P. Romer, "Supervisory control and data acquisition systems," IEEE Spectrum, vol. 31, no. 8, pp. 42-46, Aug. 1994.

[6] R. Kumar, M. L. Dewal, and K. Saini, "Utility of SCADA in power generation and distribution system," in Proc. 2010 3rd Int. Conf. Computer Science and Information Technology, Chengdu, China, 2010, pp. 648-652, doi: 10.1109/ICCSIT.2010.5564689.

[7] Inductive Automation, "SCADA Software," Inductive Automation. [Online]. Available: https://www.inductiveautomation.com/scada-software/. [Accessed: Feb. 6, 2025].

[8] Python Software Foundation, “tkinter — Python interface to Tcl/Tk — Python 3.7.2 documentation,” *python.org*, 2019. https://docs.python.org/3/library/tkinter.html

[9] Pillow, “Pillow — Pillow (PIL Fork) 6.2.1 Documentation,” *Readthedocs.io*, 2011. https://pillow.readthedocs.io/en/stable/

[10] Matplotlib, “Users guide — Matplotlib 3.5.0 documentation,” *matplotlib.org*. https://matplotlib.org/stable/users/index.html

[11] Python Assets, “Tk after() Function (tkinter),” *Python Assets*, Apr. 11, 2022. https://pythonassets.com/posts/tk-after-function-tkinter/

[12] Reference#. Reference - NetworkX 3.4.2 documentation. (n.d.). https://networkx.org/documentation/stable/reference/index.html