Department of Electronic & Telecommunication Engineering, University of Moratuwa, Sri Lanka.

Industrial Power Monitoring System

Design Documentation

EN2160

Group Members:

210153H	EDIRISINGHE E.A.D.D.D.
210703V	WICKRAMARATHNE M.P
210679B	WANNIARACHCHI W.A.D.N.M.
210222U	HEWAGAMAGE K.L.N

Updated
7 July 2024

Contents

1 Introduction

Energy efficiency is crucial for industrial facilities to manage operating costs and reduce environmental impact. While many machines have their own power monitoring systems, the scattered nature of this data makes it difficult to analyze effectively. Our project proposes an industrial energy monitoring system to address this issue, centralizing and analyzing energy data for deeper insights. By providing real-time data transmission to a centralized system, our solution aims to offer valuable insights into energy consumption patterns and the performance of industrial equipment. This, in turn, empowers facilities to optimize energy usage, reduce waste, and improve overall operational efficiency.

The purpose of this document is to provide a comprehensive overview of the design and development process of our Industrial Power Monitoring System. It includes detailed descriptions of the system architecture, schematics, PCB layout, mechanical design, firmware development, and testing procedures. Additionally, this design document serves as a thorough record of our project, illustrating the methodologies and decisions that guided us from initial concept to final implementation.

2 System Overview

The Industrial Power Monitoring System is designed to provide comprehensive monitoring and analysis of power parameters in industrial settings. The system comprises several key components: voltage and current sensors, power measuring IC, microcontroller, and a cloud-based data management and visualization interface.

2.1 System Architecture

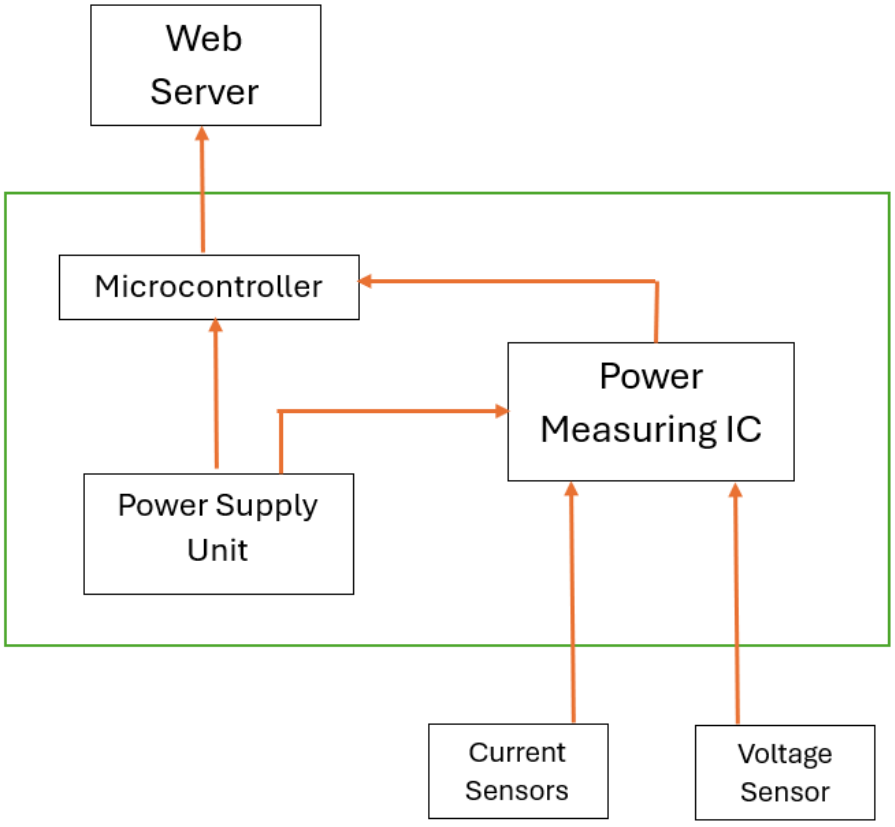


Figure 1: Block Diagram

2.2 Function of each Subsystems and its Components

2.2.1 Power Measuring IC

The heart of the monitoring system is the power measuring IC. We selected ATM90E36A for this. It is responsible for processing the raw data from the voltage and current sensors.

The ATM90E36A is a high-performance energy metering IC designed for three-phase power systems. It has 3 channels with measurement capabilities and Fourier analysis functions which can used for power monitoring instruments. It can provide accurate measurements of various parameters such as voltage, current, total harmonic distortion (THD), discrete Fourier transform (DFT), mean power etc.

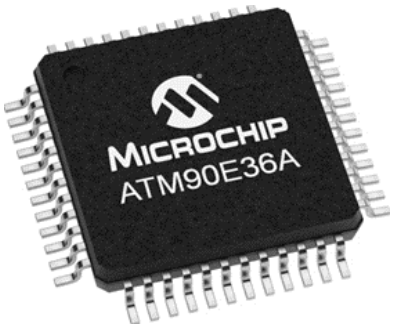


Figure 2: Metering IC

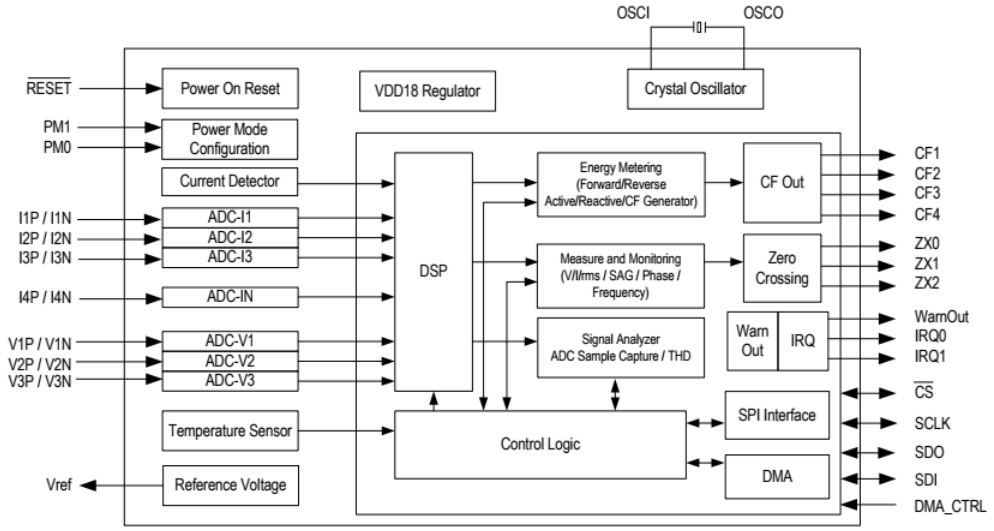


Figure 3: M90E36A Block Diagram

2.2.2 Microcontroller

The microcontroller acts as the central processing unit of the system. It receives the processed data from the power measuring IC and manages the communication with the cloud. The microcontroller is programmed to handle data acquisition, processing, and transmission, ensuring real-time data availability for analysis. It is also responsible for establishing and maintaining a stable WiFi connection for data transmission.

We used ESP32-S3 because it provides reliable data communication, which is important for sending real-time data to the cloud. It has built-in WiFi and Bluetooth, making it easy to connect. Also, there are many libraries and resources available, which makes development easier and faster. The ESP32 is also commonly used in industrial products due to its reliability, performance, and cost-effectiveness, making it a great choice for our project.

ESP32-S3 is a dual-core XTensa LX7 MCU, capable of running at 240 MHz. Apart from its 512 KB of internal SRAM, it also comes with integrated 2.4 GHz, 802.11 b/g/n Wi-Fi and Bluetooth 5 (LE) connectivity that provides long-range support. It has 45 programmable GPIOs and supports a rich set of peripherals. ESP32-S3 supports larger, high-speed octal SPI flash, and PSRAM with configurable data and instruction cache.

- AI Acceleration and Security:

  ESP32-S3 includes AI acceleration support through vector instructions in the MCU, enhancing capabilities for neural network computing and signal processing tasks. It leverages ESP-DSP and ESP-NN libraries for optimized application development. Security features include AES-XTS-based flash encryption, RSA-based secure boot, digital signature, HMAC, and a "World Controller" peripheral supporting two fully-isolated execution environments. These features enable the implementation of trusted execution environments and robust security schemes without requiring external components.

- Software and Development Support:

  Supported by Espressif's ESP-IDF platform, ESP32-S3 benefits from mature software infrastructure used in millions of devices worldwide. ESP-IDF ensures rigorous testing, regular updates, and comprehensive support policies, empowering developers to build and deploy applications with confidence. The platform's familiar tools and APIs facilitate seamless development and migration of applications to the ESP32-S3 platform.

Figure 4: Microcontroller

2.2.3 Power Supply Unit

The system is powered by a robust and efficient power supply unit, ensuring reliable operation in industrial environments. The power supply is designed to handle the electrical demands of the sensors, power measuring IC, and microcontroller, providing stable voltage and current for optimal performance.

1. Hi-Link HLK-20M05 220VAC to 5VDC 20W, a 220VAC to 5VDC 20W Step-Down Power Supply Module. This is cost effective and compact compared to making a SMPS circuit.
   - All voltage input (AC: 90 264V)
   - Low ripple and low noise
   - Output overload and short circuit protection
   - High efficiency, high power density

- The product is designed to meet the requirements of EMC and Safety Test
- Low power consumption, environmental protection

Figure 5: Hi-Link

2. TPS75933KTTR - Voltage Regulators 3.3-V. This is a low Dropout Voltage Regulators which is able to convert voltage from 5 to 3.3V while maintaining required current.

Figure 6: Voltage Regulator

2.2.4 Voltage and Current Sensors

The system utilizes external voltage and current sensors to accurately measure the electrical parameters of the industrial equipment. These sensors are connected externally to the main device to mitigate the risk of high currents flowing through the measuring apparatus, enhancing safety and reliability.

- **Clip on current transformer**

  These transformers are non-intrusive and can be easily clipped around the conductors without the need to disconnect any wiring. This design enhances safety and convenience, making installation and maintenance quick and straightforward in industrial environments.

Figure 7: Clip on CT

2.2.5 Cloud-Based Data Management and Visualization

The data collected by the system is transmitted to a cloud-based platform where it is stored, managed, and visualized. The cloud platform provides a user-friendly web interface that allows stakeholders to monitor power consumption and quality metrics in real time. The interface includes graphical representations of the data, historical trends, and customizable alerts for critical parameters, enhancing the decision-making process for energy management.

2.2.6 Additional Components

1. Protection Components

    - 3 Ampere fuse (30mm x 6mm) – Over current protection

Figure 8: 3 Ampere Fuse

    - MOV-10D561K Varistor – Gives over voltage protection for voltage surges over 560V

Figure 9: MOV-10D561K Varistor

2. Connectors

   - Through hole PCB mount Fuse Holder - Open Fuse design allow to change the fuse easily.

Figure 10: PCB mount Fuse Holder

   - PCB mount screw terminal blocks - Used to connect voltage reference wires.

Figure 11: Terminal Block

   - PCB Mount Mono Socket - Used to connect current transformers which can connect and disconnect CT easily.

Figure 12: PCB mount Mono Socket

- Male headers - Used for testing power, metering ic, serial interface and spi interface.

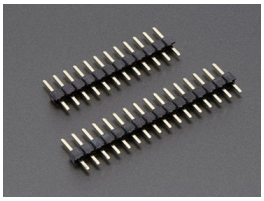


Figure 13: Male Headers

3. Antennas

- 2118909-2 Antenna: Wi-Fi 6E L273 mm MHF (Cabled PCB)

  Offering a length of 273 mm and utilizing MHF connectors for cabled PCB installations. This antenna supports triple bands, including the 6GHz band, enhancing wireless communication capabilities with improved bandwidth and reduced interference.

  Features - Adds 1200MHz of the frequency spectrum at 6GHz band

  -Accommodates 14 additional 80MHz channels or seven additional 160MHz channels

  -Enables greater network flexibility and faster speeds with reduced latency

  -TE's RF engineering competence can offer optimal throughput

  -System design flexibility with customizable solutions and multiple product options

  -Customizable cable length, connector types, and multi-input and multi-output (MIMO) arrays

Figure 14: Antenna

- RF Cable Assemblies (CSJ-RGFB-100-MHF3): RP-SMA female bulkhead to right angle IPEX MHF3 plug with 100mm of 0.81mm cable

Figure 15: Antenna Connector

3 Detailed Design

3.1 Schematics and Circuit Diagrams

A hierarchical schematic approach was employed, utilizing a main schematic with six sub-schematics dedicated to specific functionalities like the MCU, voltage and current sensing, and power management.

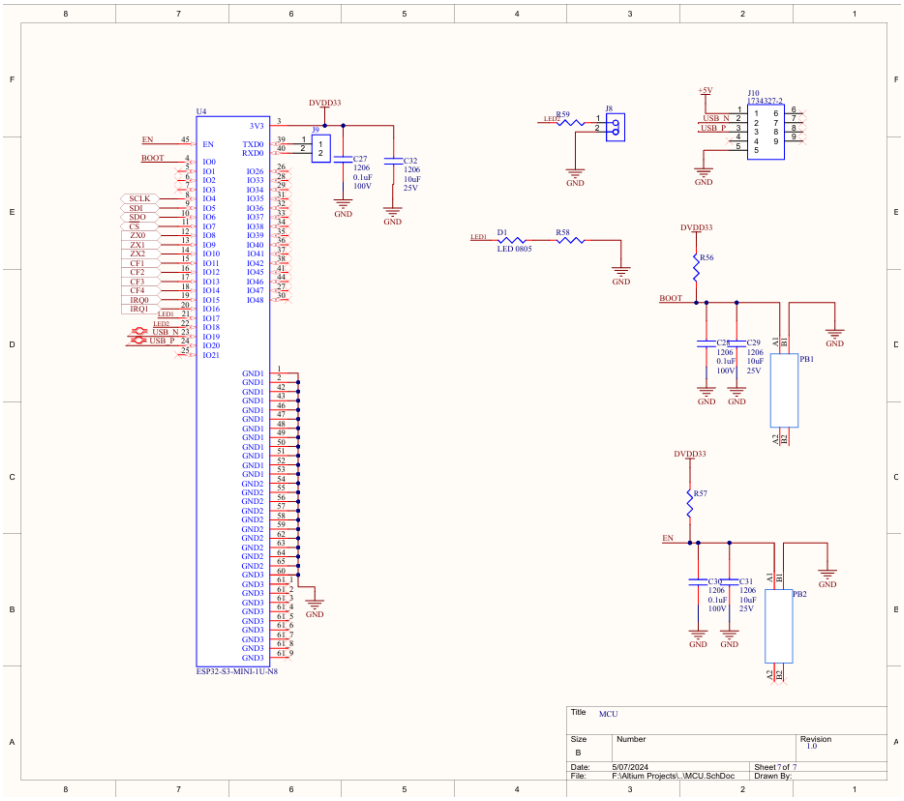


Figure 16: Schematic- MCU

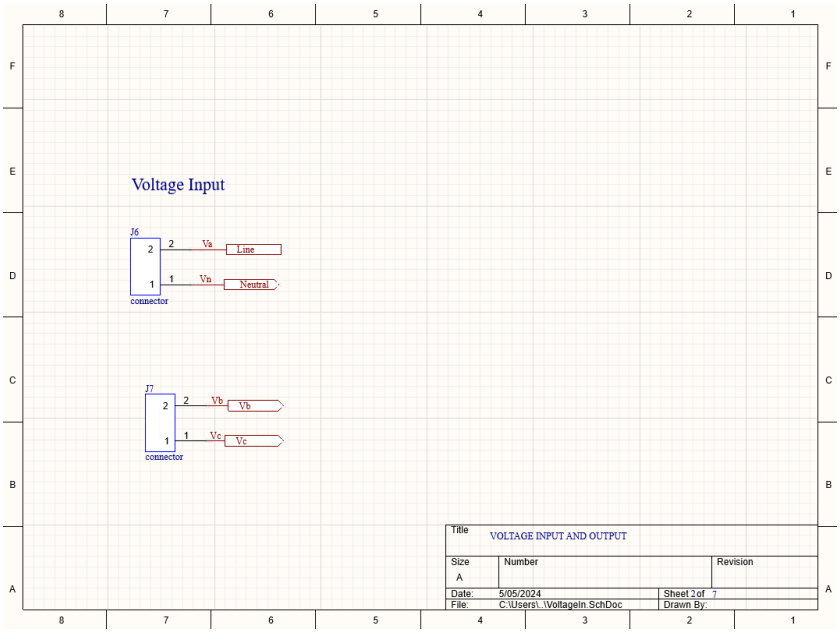


Figure 17: Schematic- Voltage In and Out

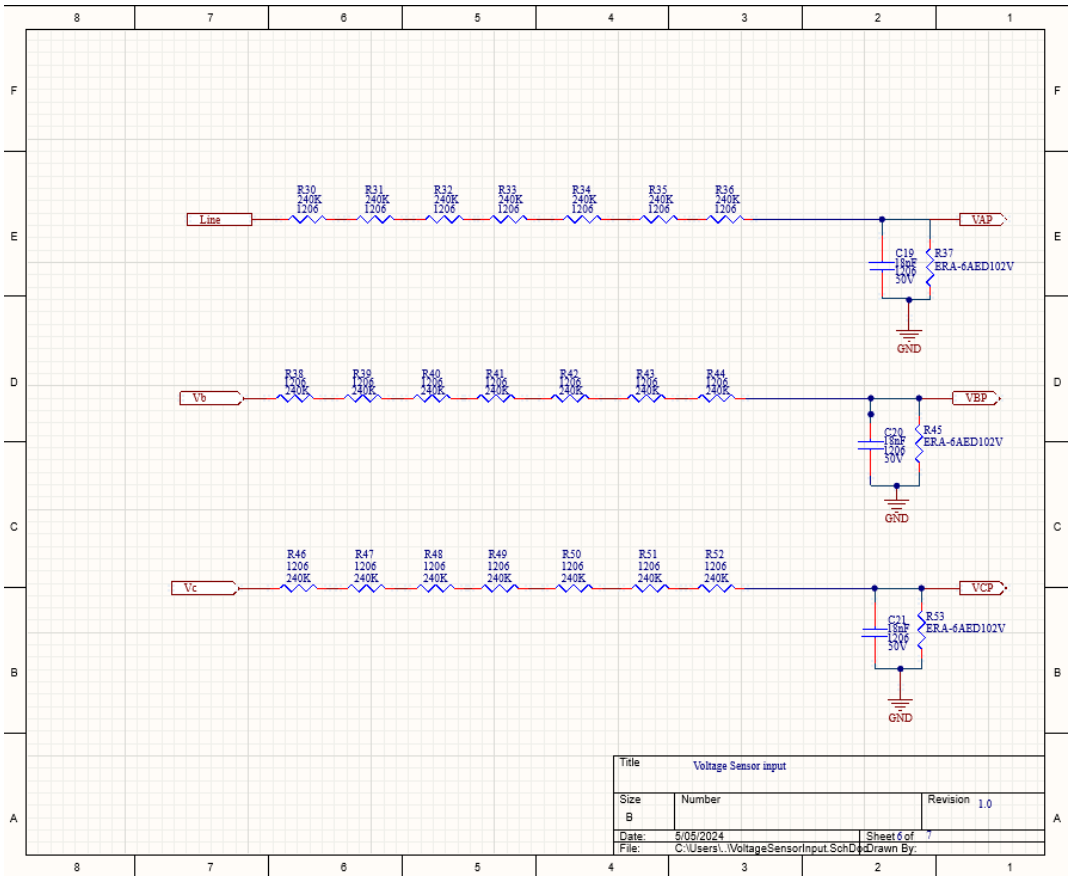


Figure 18: Schematic- Voltage sensor input

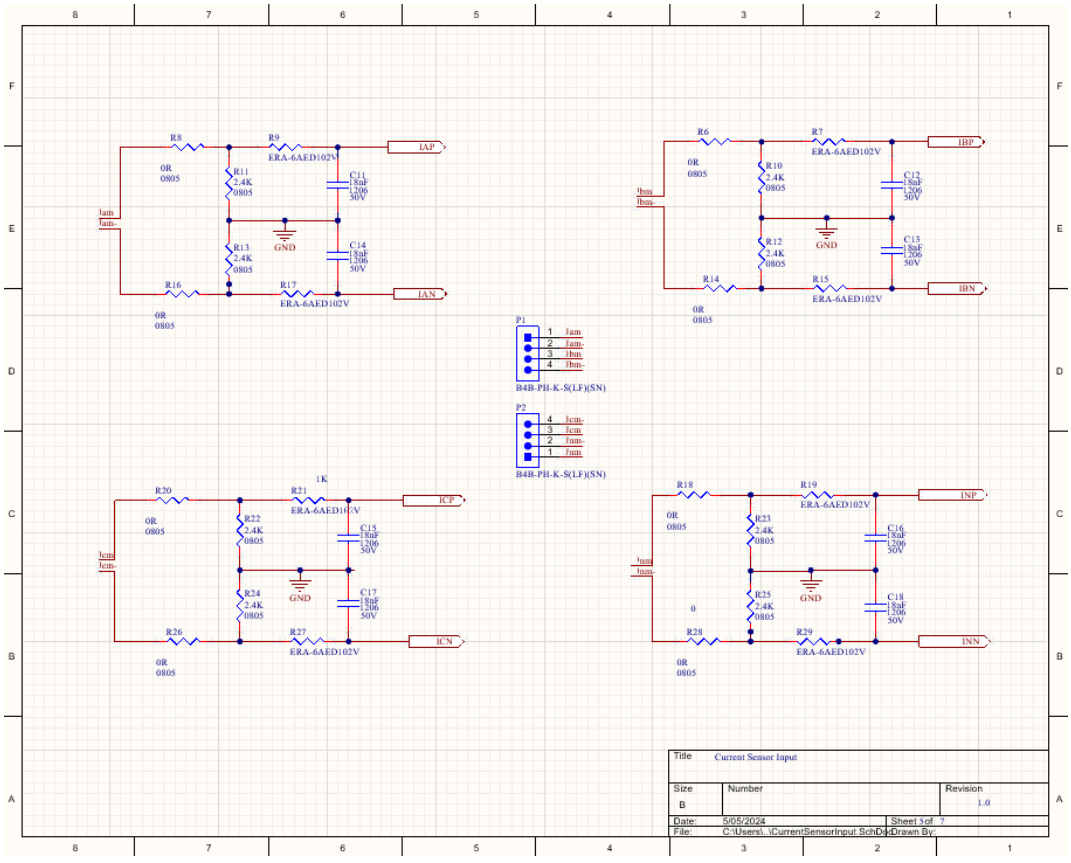


Figure 19: Schematic- Current sensor input

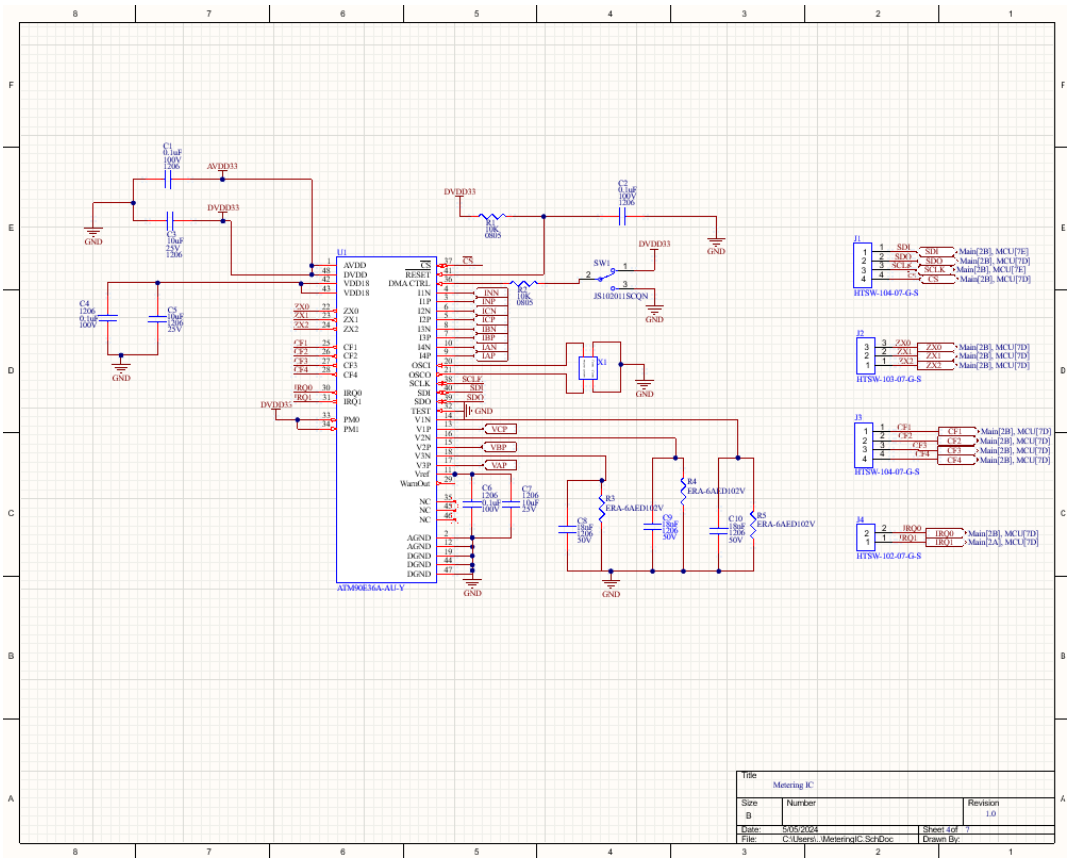


Figure 20: Schematic- Metering IC

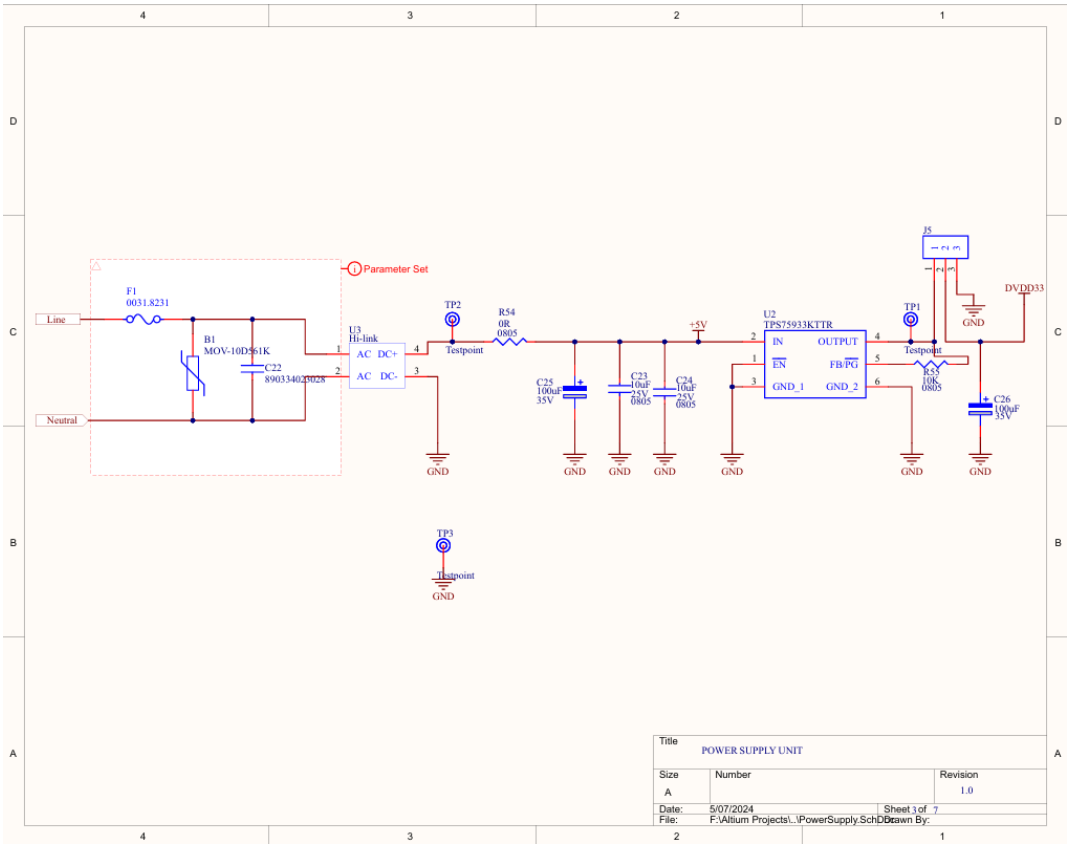


Figure 21: Schematic- Power supply unit

3.2 PCB Layout

PCB design and layout were performed using Altium, adhered to established design rules, ensuring optimal functionality and signal integrity.

3.2.1 Design Rules

1. Trace Width and Spacing: Minimum trace width and spacing were set at 0.09mm respectively, based on material and anticipated current requirements.

2. Annular Rings: Annular rings around component pads were defined as 0.3mm to ensure proper solder connections.

3. Clearances: Minimum clearances between traces, components, and vias were set at 0.5mm to prevent short circuits and crosstalk. Hole to hole clearance(Different nets) - 0.5mm Pad to Pad clearance(Pad with hole, Different nets) - 0.5mm Via to Track - 0.254mm Pad to Track - 0.2mm

4. Via Rules: Minimum via diameter, drill size, and pad size were specified as 0.15mm, 0.25mm, and 1.0mm respectively, ensuring reliable connections between layers.

3.2.2 Routing

1. Signal Routing: Signal traces were routed with controlled impedance using manual routing techniques. Critical high-speed signals were prioritized for shortest paths and minimal cross-talk.

   Used 10 mils as the trace width and for differential pair routing we used the co-planar differential pair routing technique with impedance matched at 90Ω .

   Differential pair trace width was calculated according to the manufacturer standards at the JLCPCB.

   We used the calculated Width of 6.5 mil and 8 mil separation to route differential pairs.

2. Power and Ground Planes: Dedicated power and ground planes were incorporated for efficient power distribution and noise reduction. The ground plane was placed on both inner layers and outer layers for optimal shielding. Stitching vias were used to ensure a proper connectivity of the ground planes and reduce the noises.

3.2.3 Stackup

1. Layer Count: A four-layer stackup was chosen, consisting of two signal layers, a power plane, and a ground plane to accommodate the signal complexity, power requirements, and cost considerations.

#	Name	Material	Type	Weight	Thickness	Dk	Df
	Top Overlay		Overlay				
	Top Solder	Solder Resist	Solder Mask		0.0127mm	3.8	
1	Top Layer		Signal	1oz	0.035mm		
	Dielectric 1	3313	Prepreg		0.09939mm	4.1	
2	second layer		Signal	1/2oz	0.01519mm		
	Core	0.55mm H/...	Core		0.45001mm	4.6	
3	Third layer		Signal	1/2oz	0.01519mm		
	Dielectric 2	3313	Prepreg		0.09939mm	4.1	
4	Bottom Layer		Signal	1oz	0.035mm		
	Bottom Solder	Solder Resist	Solder Mask		0.0127mm	3.8	
	Bottom Overlay		Overlay				

Figure 22: Layer Stackup

2. Material Selection: Considering the manufacturing cost and the routable track width, we have selected JLC04081H-3313 as the pre preg material. The board Thickness and differential pair width was set as 0.8mm and 6.5mils respectively.

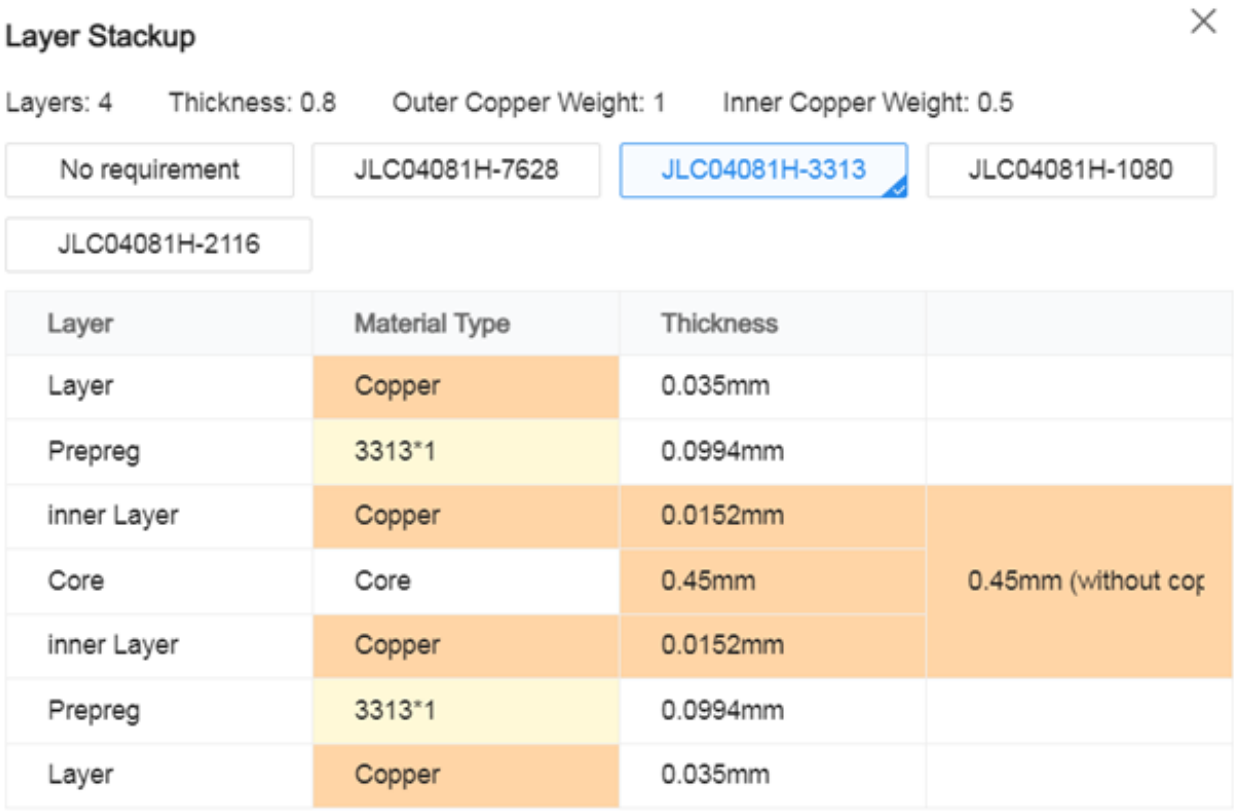


Figure 23: Layer Stackup

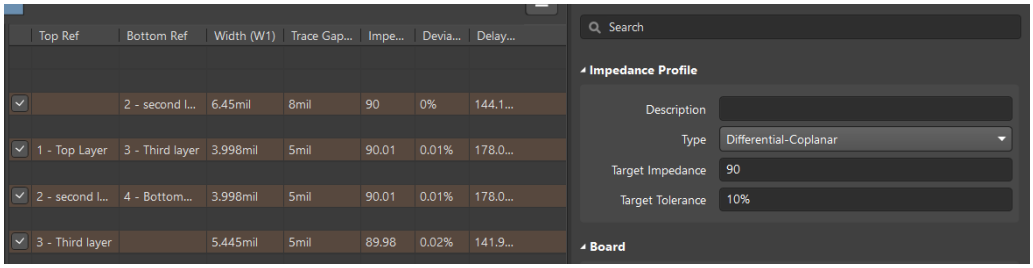


Figure 24: Impedance profile

Components were strategically placed considering heat dissipation, signal routing paths, and ease of assembly. High-power components were positioned near the ground plane for optimal heat transfer.

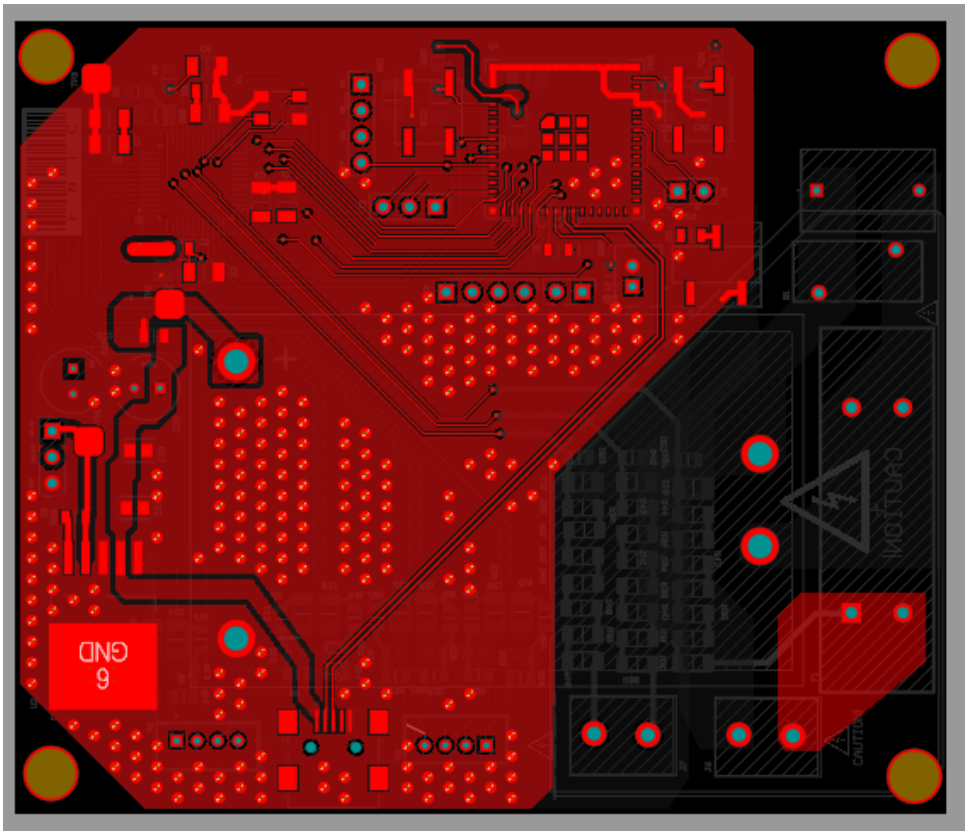


Figure 25: Top Layer

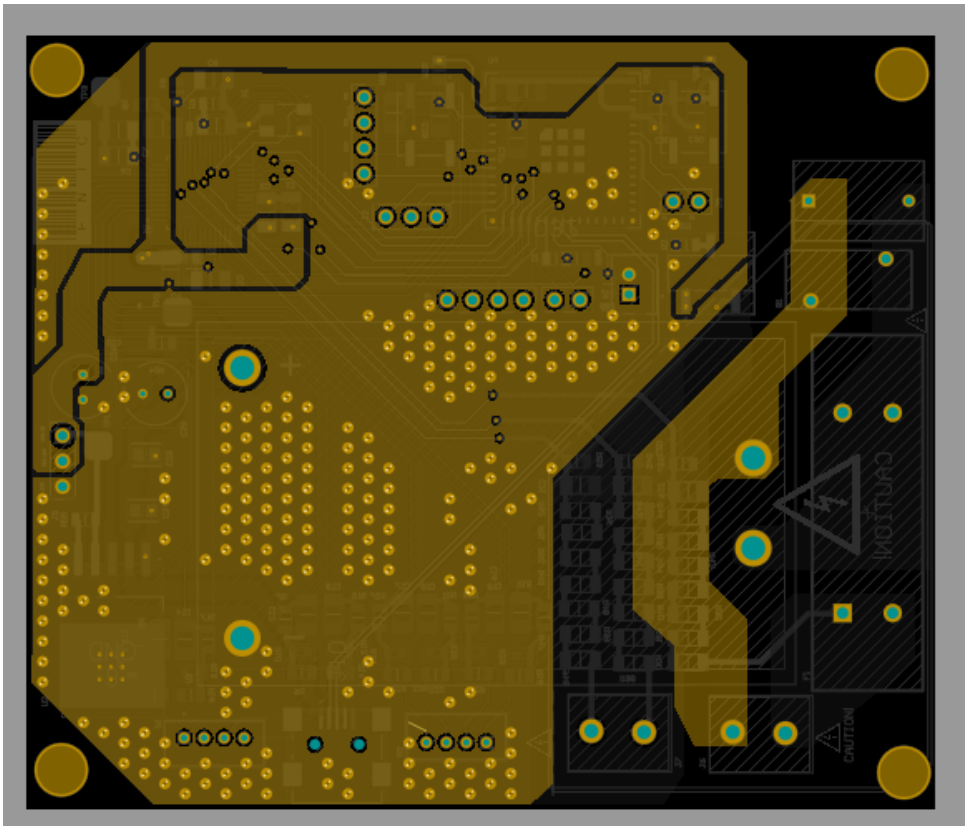


Figure 26: Second Layer

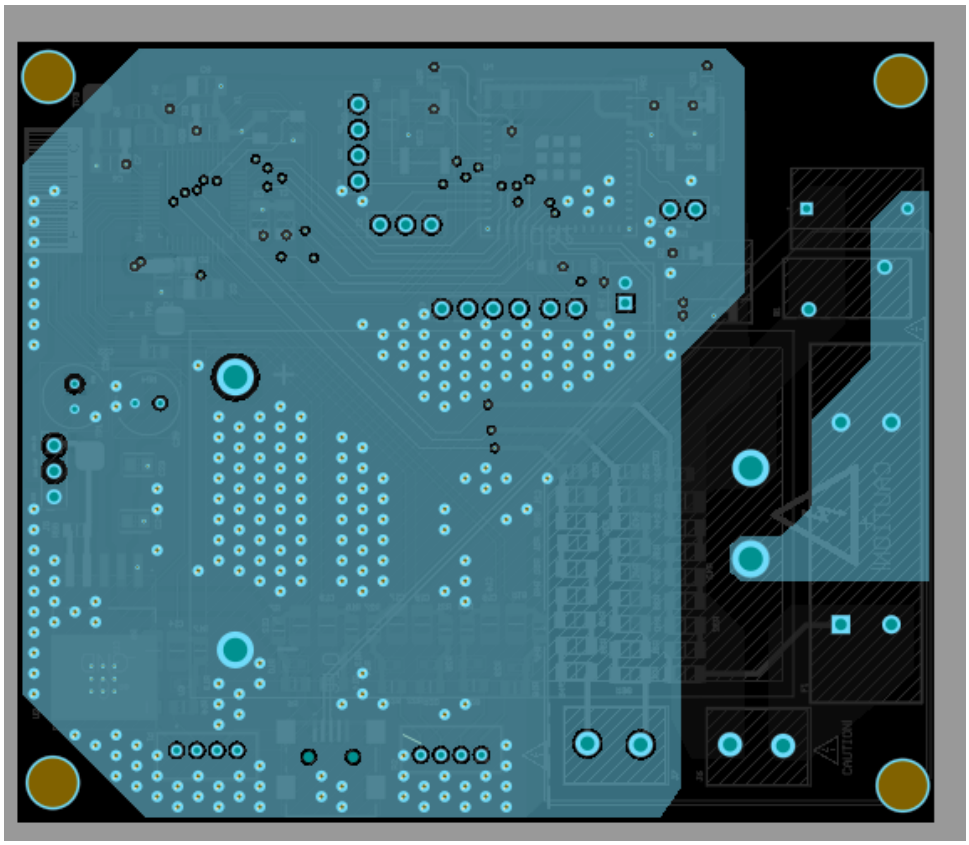


Figure 27: Third Layer

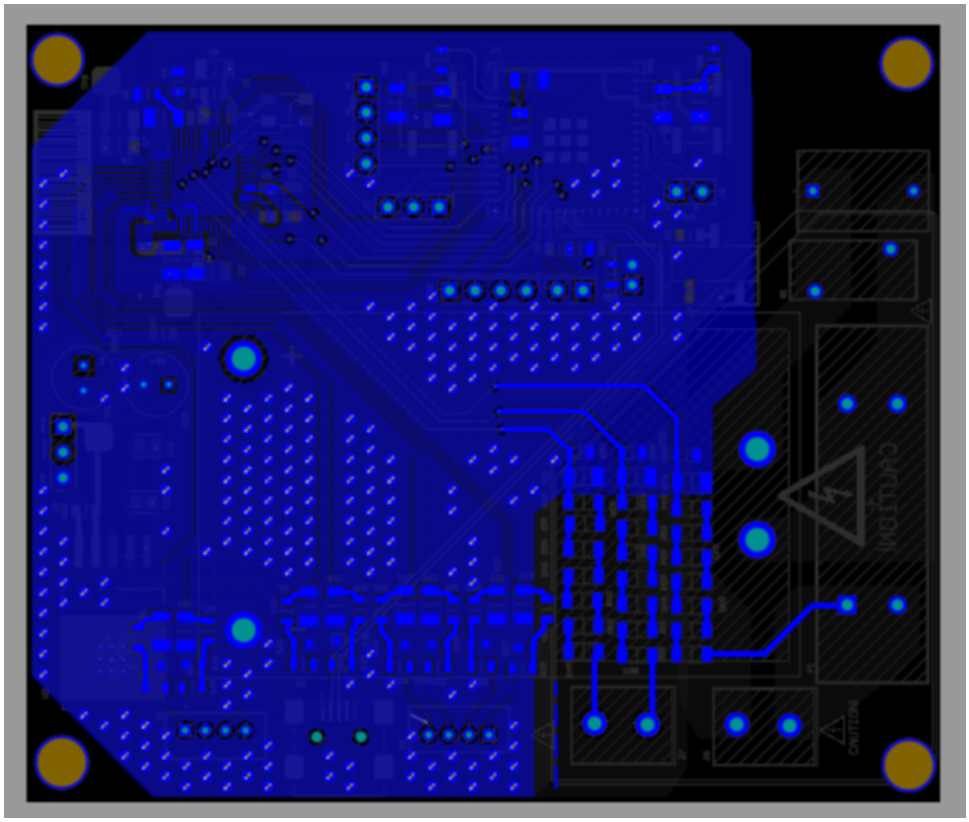


Figure 28: Bottom Layer

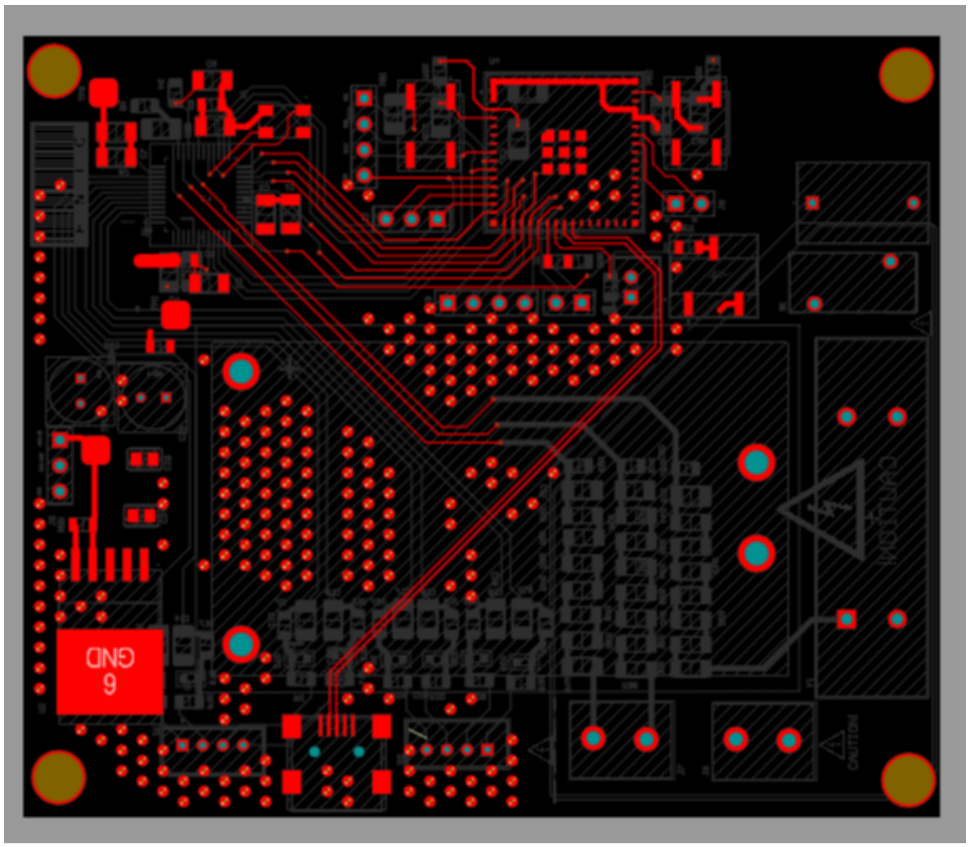


Figure 29: TOP Layer Traces

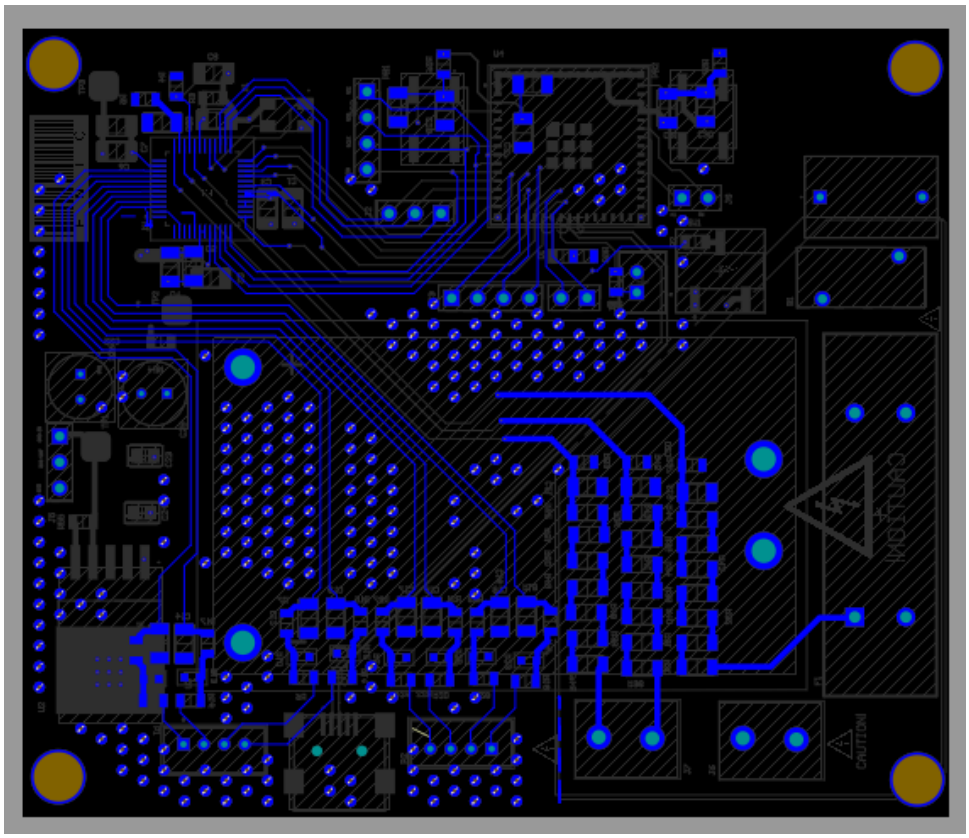


Figure 30: Bottom Layer Traces

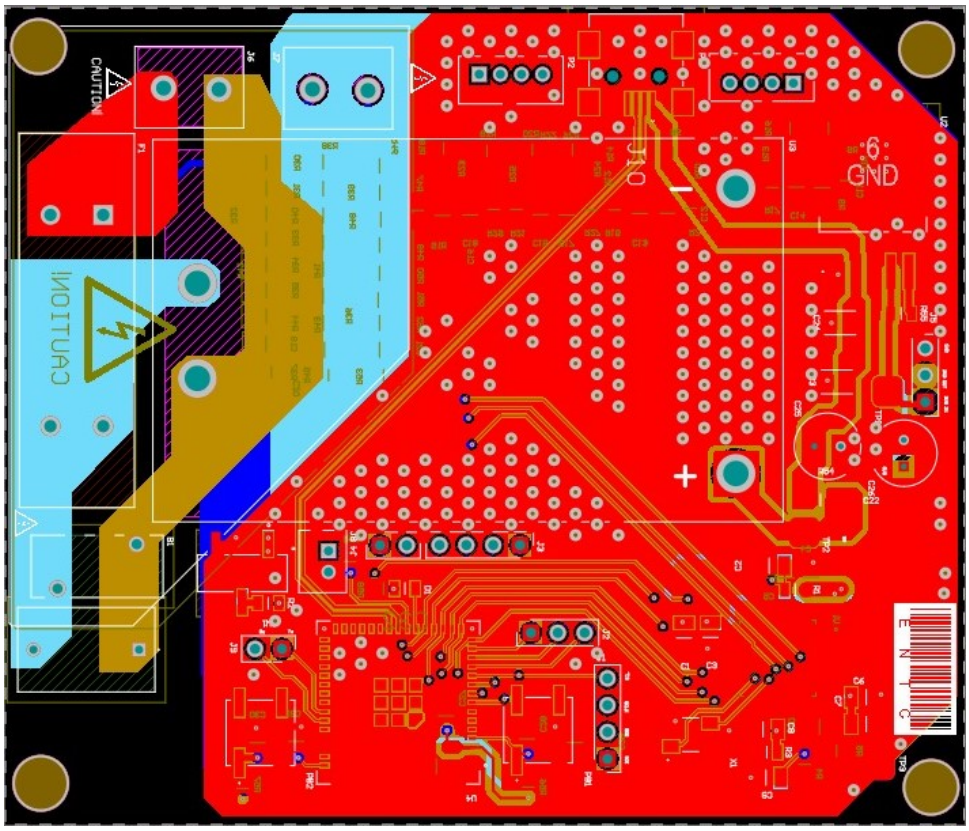


Figure 31: PCB Layout Fig1

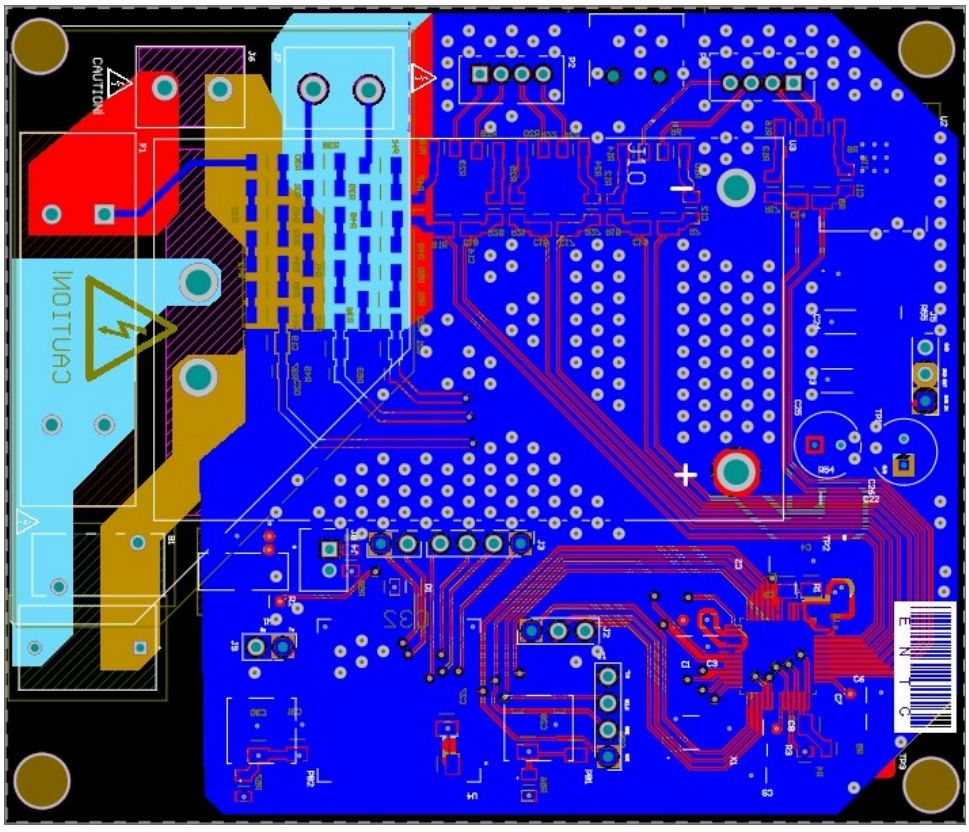


Figure 32: PCB Layout Fig2

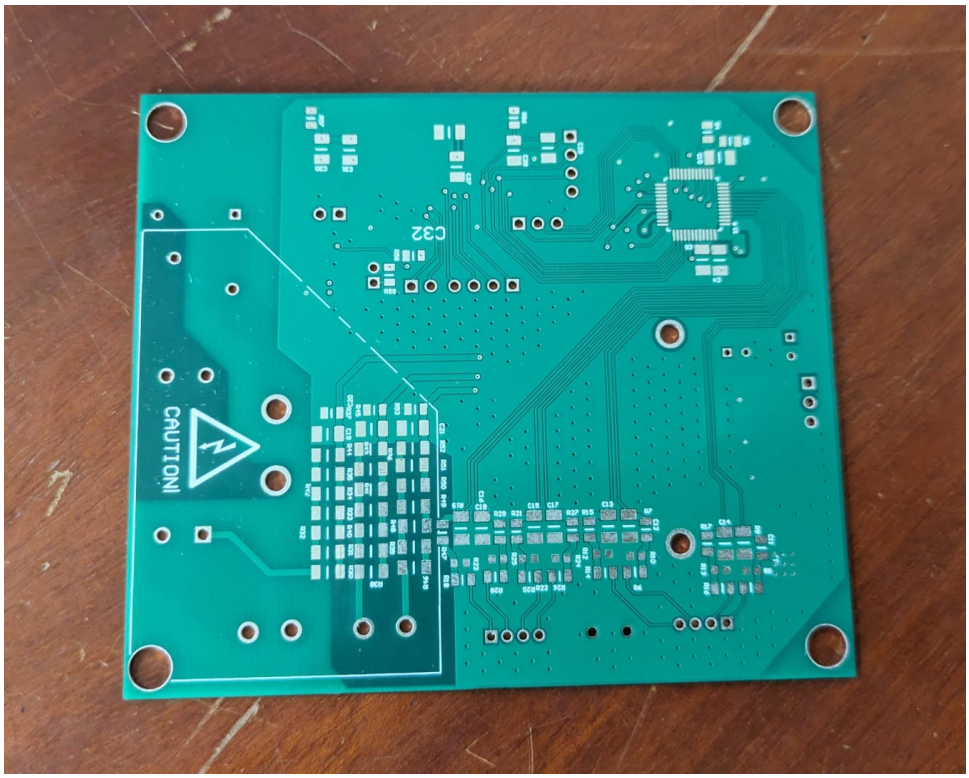


Figure 33: Bare PCB Fig1

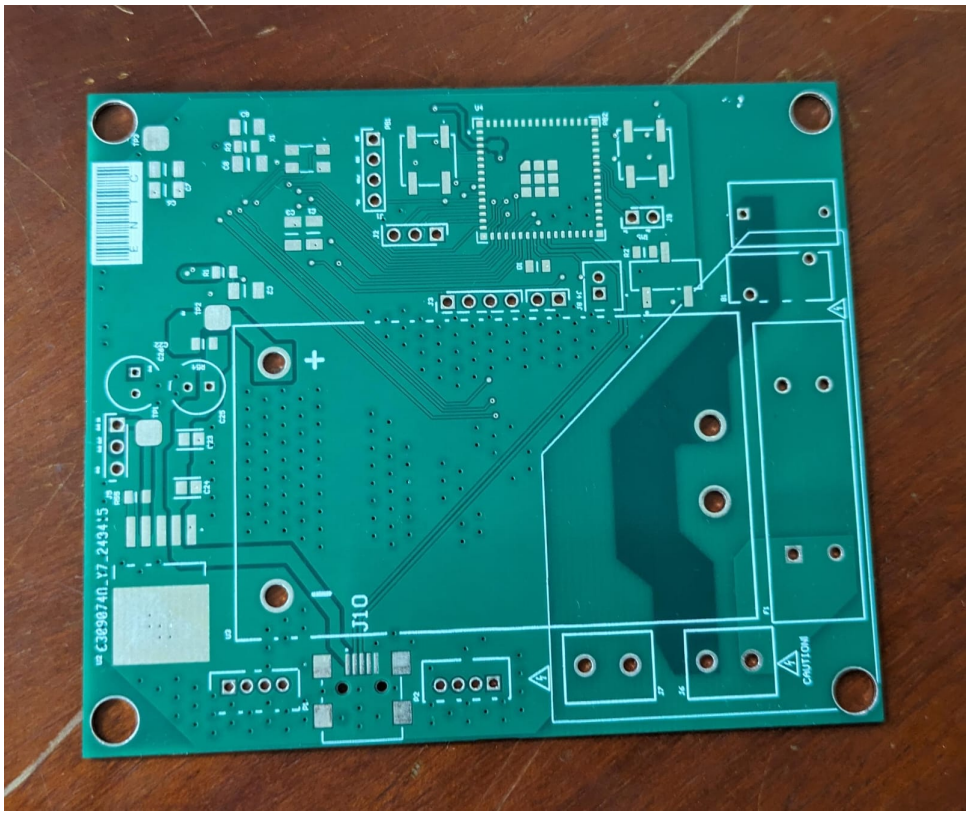


Figure 34: Bare PCB Fig2

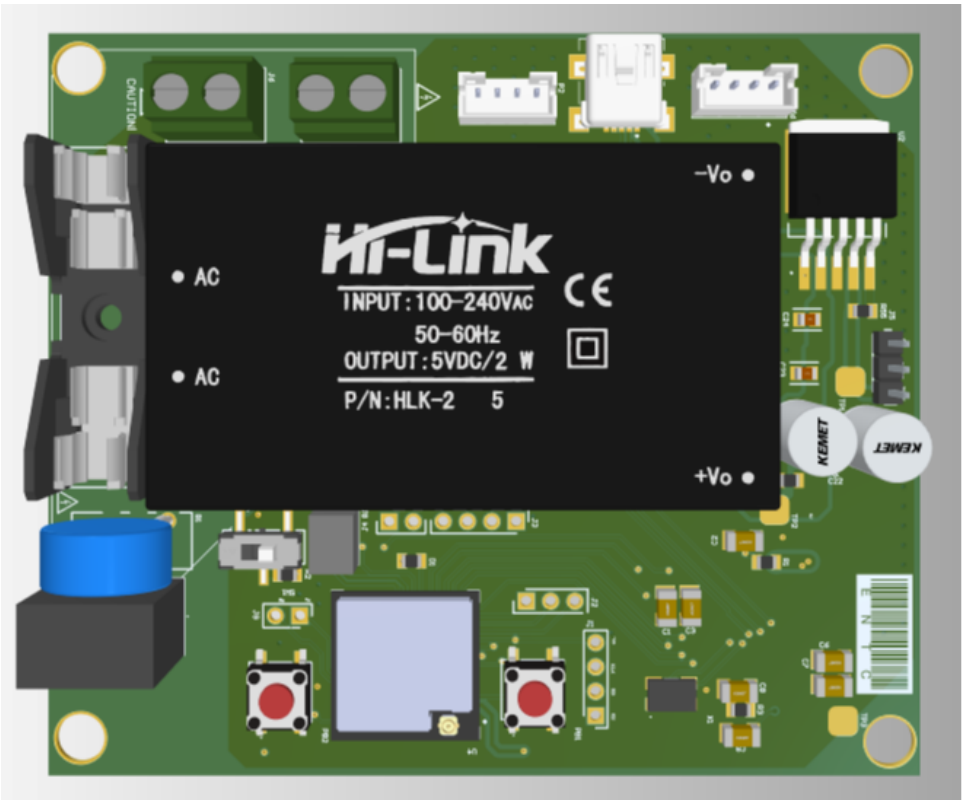
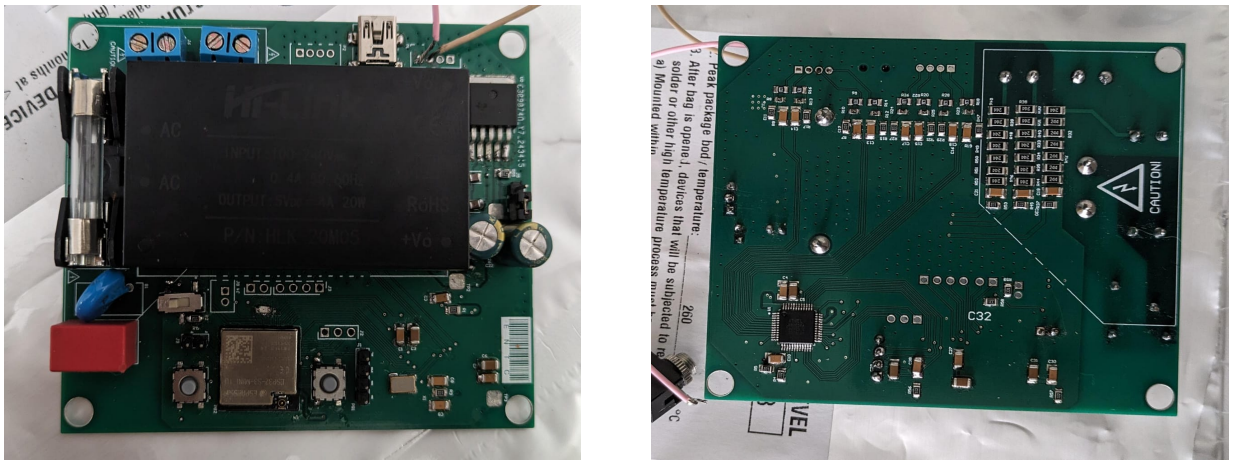


Figure 35: 3D View Fig9

Figure 36: Assembled PCB



(a) (b)

Figure 37: Soldered PCB

3.3 Bill of Materials (BOM)

A comprehensive BOM lists all components used in the design, including part numbers, descriptions, quantities, and suppliers. This ensures easy procurement and assembly.

	Part Number	Order Qty.	Price (USD/$)	Ext.: (USD/$)
1	693-0031.8231	1	1.85	1.85
2	652-MOV-10D561K	1	0.41	0.41
3	710-890334023028	2	0.51	1.02
4	71-WSL080500000ZEA9	13	0.616	8.01
5	80-ESC107M035AE3AA	4	0.31	1.24
6	81-GRM21B71E106KE01L	4	0.32	1.28
7	595-TPS75933KTTR	2	6.02	12.04
8	71-CRCW201010K0JNFIF	10	0.438	4.38
9	80-C1206C104J1RAUTO	11	0.175	1.93
10	963-BAST31LAB7106KT	10	0.256	2.56
11	556-ATM90E36A-AU-Y	2	4.88	9.76
12	611-JS102011SCQN	2	0.94	1.88
13	520-163.84-1030BCKMT	2	0.76	1.52
14	80-C1206C183J5R	28	0.237	6.64
15	667-ERA-6AED102V	32	0.138	4.42
16	667-ERA-8AEB244V	30	0.279	8.37
17	356-ESP32S3MINI1UN8	2	3.1	6.2
18	667-ERA-6AED331V	4	0.22	0.88
19	667-EVP-BV2C3L000	4	0.52	2.08
20	571-1734327-2	1	1.36	1.36
21	279-CPF-A-0603B2K4E1	16	0.248	3.97
22	595-TL084QDR	2	0.87	1.74
23	571-2118909-2	1	2	2
24	712-CSJ-SGFB100-MHF3	1	5.08	5.08
	Total	**185**	**31.537**	**90.62**

3.4 SolidWorks Design

The enclosure was designed using SolidWorks, ensuring it meets industrial standards for durability and functionality. Detailed CAD models and drawings with precise dimensions and tolerances were created.

1. **Mounting Features:** Used four hardware boss heads to secure the lid to the box and four pin bosses to mount the PCB.

2. **Cooling Vents:** Included holes at the side to ensure adequate heat dissipation.

3. **Lid attachment:** Used lip and groove to tighten the lid to the box.

4. **Draft Angles and Fillets:** Draft angles and fillets were incorporated into the design to facilitate plastic molding and prevent mold release issues.

5. **Enclosure Material:** The Fused Deposition Modeling (FDM) technique was employed using Polylactic Acid (PLA) filament for 3D printing the enclosure. PLA was selected as the material because of its lightweight and durability.

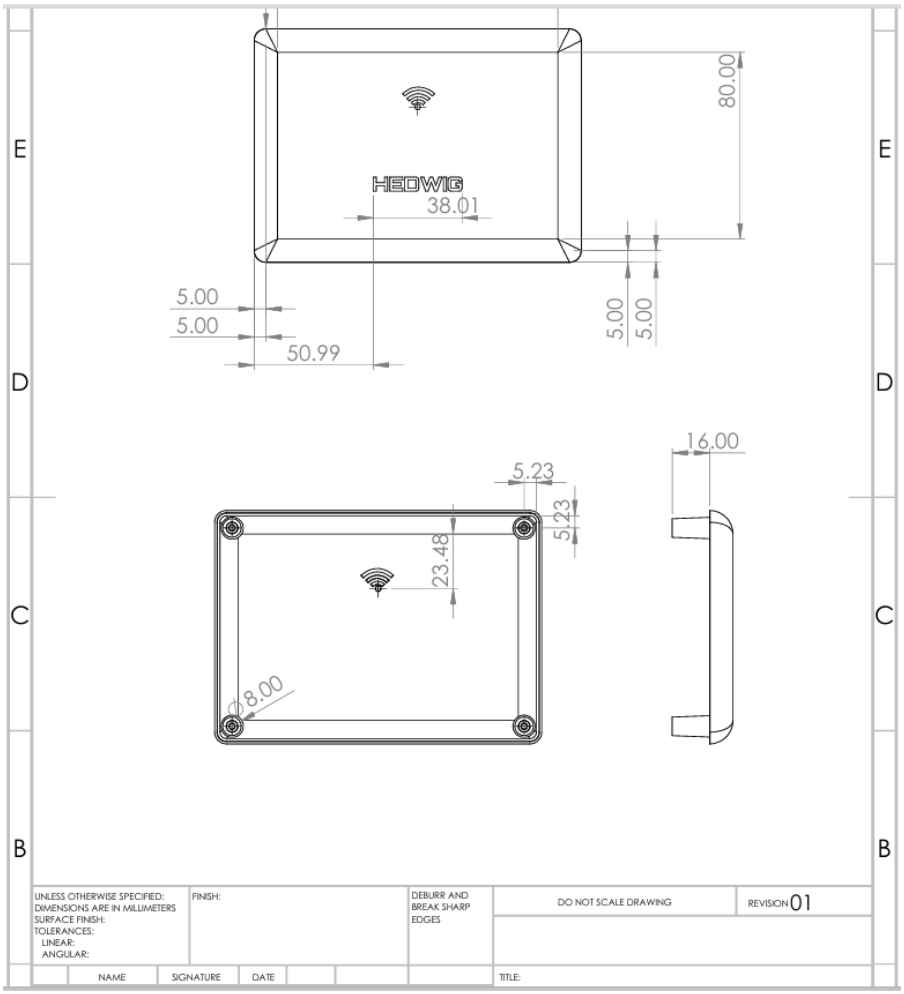


Figure 38: Dimensions of Lid

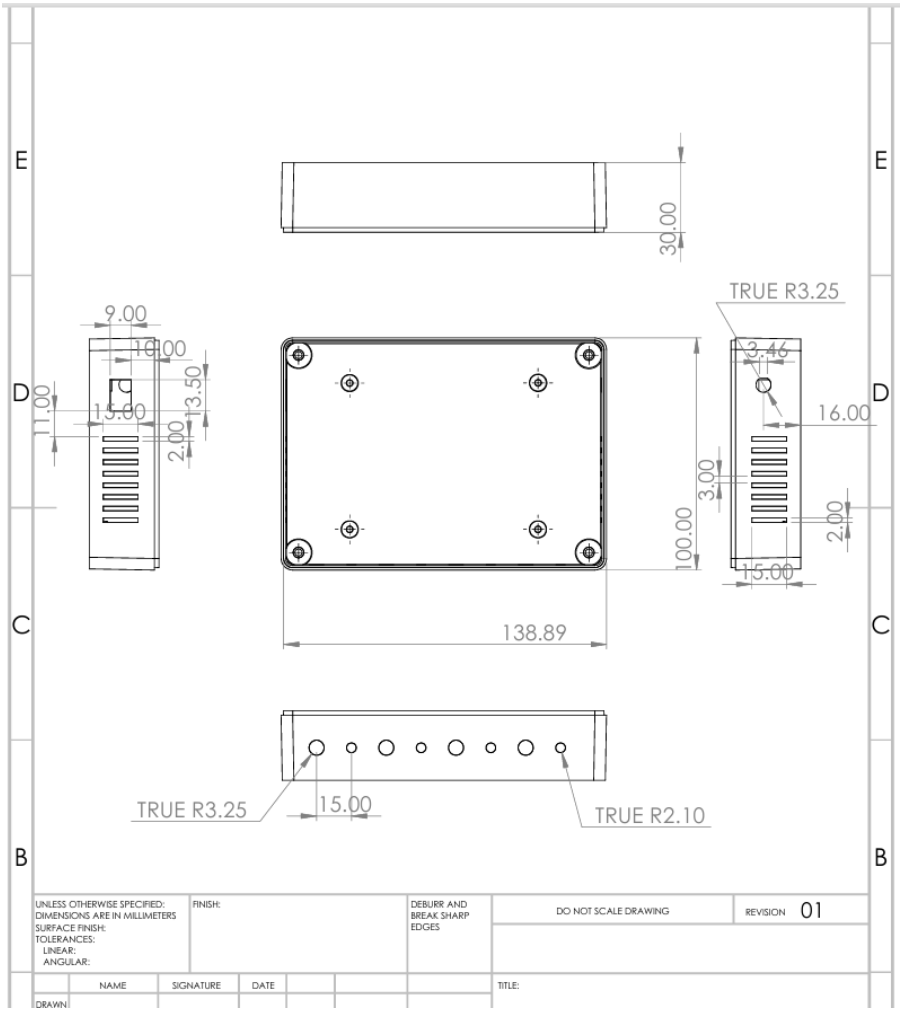
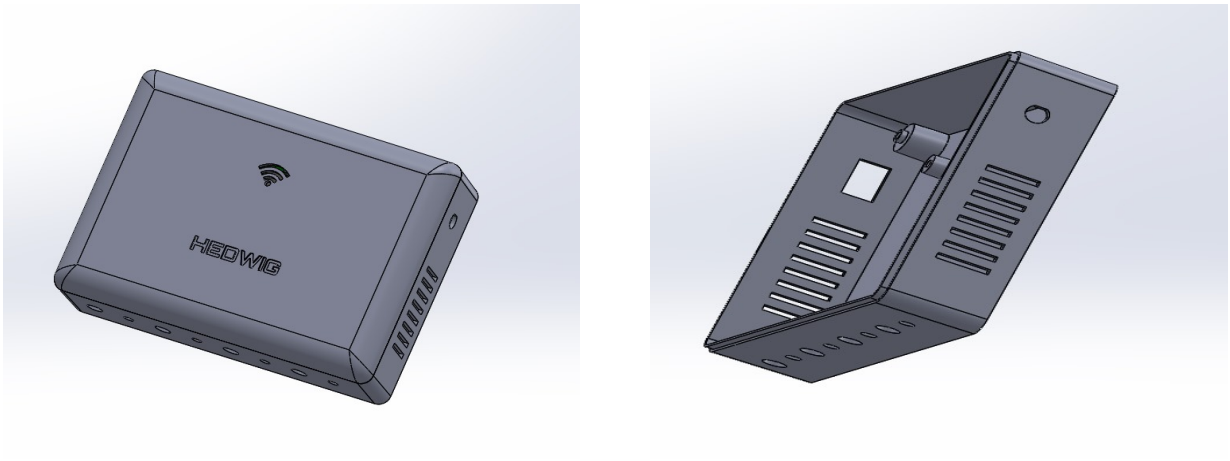


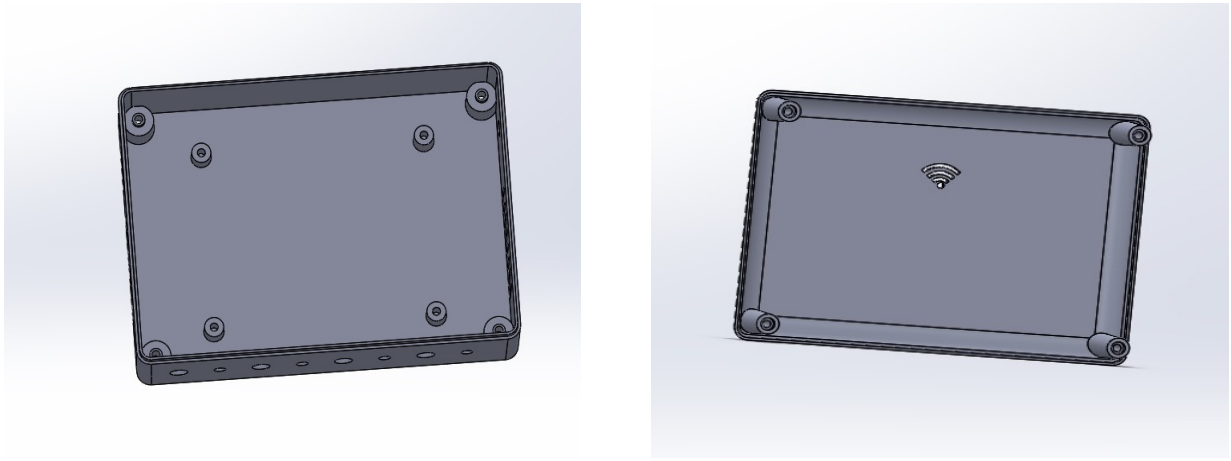
Figure 39: Dimensions of Box



(a) Enclosure Fig1

(b) Enclosure Fig2

Figure 40: Enclosure



(a) Enclosure Fig3 (b) Enclosure Fig4

Figure 41: Enclosure

Figure 42: Physically built Enclosure

(a) Lid (b) Box

Figure 43: Physically built Enclosure

4 Firmware

The firmware is designed to retrieve parameters from the Metering IC using SPI and then publish the data to an MQTT Broker.

4.1 System Architecture

The firmware system operates as follows:

- **Initialization**: Upon startup, the firmware initializes the SPI interface and establishes communication with the Metering IC.
- **Data Acquisition**: It sends commands to the Metering IC via SPI to request measurements (voltage, current, etc.).
- **Data Processing**: Received data from the Metering IC is processed, potentially formatted, and prepared for transmission.
- **MQTT Communication**: The firmware connects to the MQTT broker, publishes formatted data as MQTT messages to designated topics.

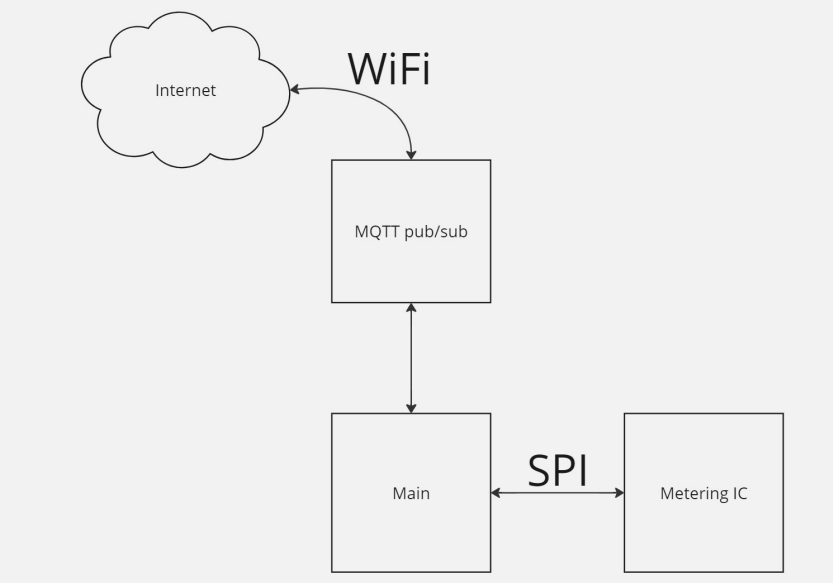


Figure 44: Firmware architecture

The firmware system effectively interfaces with a Metering IC using SPI, acquires real-time electrical measurements, and publishes this data to an MQTT broker for further processing and utilization. Libraries used in the firmware include SPI.h, ATM90E36.h, WiFi.h, and PubSubClient.h, each serving specific functionalities such as SPI communication, metering IC interfacing, Wi-Fi connectivity, and MQTT client functionalities.

Note: *The full code and additional programming information are provided in the Programming Information section. (Page 28)*

5 Software

1. **Mosquitto**

   We are using Mosquitto version 2.0.11 and the MQTT Broker in a local server. We're using Mosquitto version 2.0.11 as our MQTT broker on a local server. Mosquitto handles the messaging between devices in our setup. It supports MQTT versions 3.1, 3.1.1, and 5,

which is great because it ensures compatibility with our firmware's MQTT client (using the PubSubClient library).

Setting up Mosquitto involves configuring a file called `mosquitto.conf` where we specify things like network ports, security settings (like TLS encryption for secure communication), and access controls to manage who can publish or subscribe to which topics.

It's lightweight and efficient, which is perfect for our IoT applications. Plus, it has features for monitoring and managing connections and messages, ensuring everything runs smoothly.

In short, Mosquitto helps us handle reliable communication between our devices and our local server, making sure data gets where it needs to go securely and efficiently.

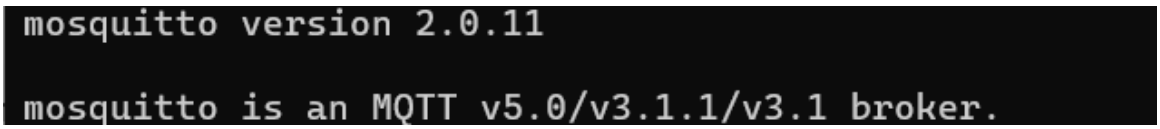


Figure 45: Mosquitto

2. **Node-red**

We are using Node-red to display the dashboard Node-RED serves as our dashboarding tool, providing a visual interface for real-time data visualization and management. It enables us to effortlessly connect and display data. Integrated with our MQTT broker (Mosquitto), Node-RED supports customization with graphs, gauges, and other visual elements, facilitating clear and insightful data representation tailored to our operational needs. Its intuitive interface streamlines the process of creating and maintaining a comprehensive dashboard for monitoring critical metrics and ensuring operational efficiency.

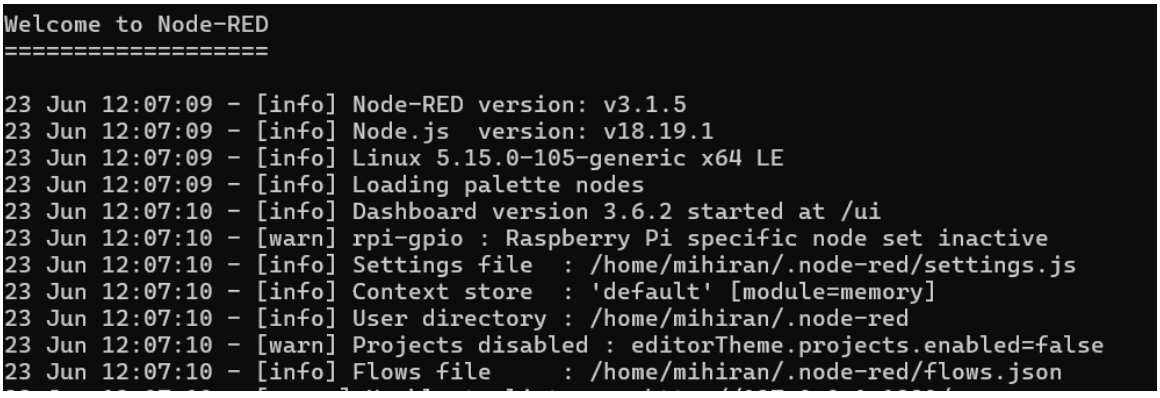


Figure 46: Node-red

3. **Apache2**

We utilize Apache2 as our HTTP server, configured with reverse proxy capabilities. This setup enables us to efficiently manage and route incoming web traffic to backend services, ensuring secure and optimized delivery of web applications and resources. Apache2's robust features and configuration flexibility support our needs for scalable and reliable web hosting and application deployment.

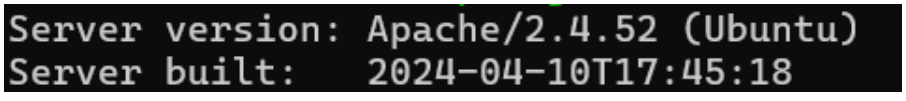


Figure 47: Apache2

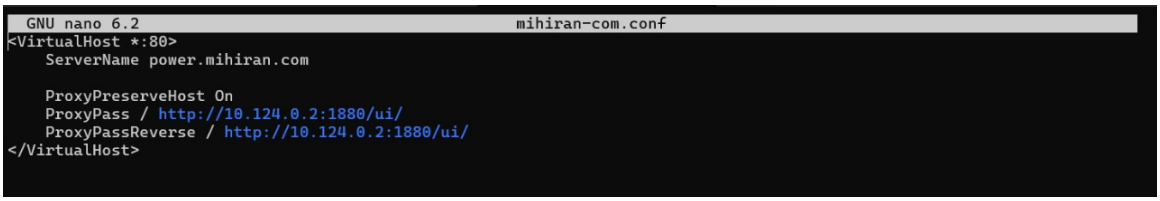


Figure 48: Apache2 Configurations

Our server is proxied through Cloudflare, leveraging their globally distributed network to enhance our web infrastructure. Cloudflare acts as a reverse proxy, serving as a protective shield between our origin server and the internet. This setup improves security by filtering malicious traffic, optimizing performance through caching and content delivery network (CDN) capabilities, and ensuring reliable HTTPS connections with their SSL/TLS encryption. Additionally, Cloudflare provides real-time analytics and insights, allowing us to monitor and optimize our web traffic effectively. This integration helps us deliver a secure, fast, and reliable web experience to our users worldwide

Figure 49: Cloudflare proxy

Digital Ocean Droplet details

Configuration:

- **Memory**: 1 GB
- **CPU**: 1 AMD vCPU
- **Storage**: 25 GB Disk
- **Datacenter**: SFO3 (San Francisco 3)
- **Operating System**: Ubuntu 22.04 (LTS) x64

This Digital Ocean Droplet configuration provides a balanced setup with 1 GB of memory and a single AMD vCPU. It includes a 25 GB disk capacity and is located in the SFO3 datacenter, ensuring efficient performance and reliability. The choice of Ubuntu 22.04 LTS x64 as the operating system ensures long-term support and compatibility with a wide range of applications and services.

This setup is well-suited for hosting services such as Node-RED for real-time data visualization and management, leveraging its robust resources and the stability offered by Digital Ocean's infrastructure.

Figure 50: Droplet

4. **Final Dashboard**

Figure 51: Dashboard

6 Programming Information

To program the MCU we have used several basic utility libraries provided by Espressif IDF and several custom libraries. Here are several important files in the codebase.

- **main.c**: This code responsible for overall device functionality, including SPI register configuration, controling metering IC, network connectivity and publishing measurement data to the server.
- **ATM90E36.h and ATM90E36.c** These files includes register addresses for each measurements in the metering IC and definitions of metering functions.
- **wifi.c and mttq.c** These files include configurations and definitions for wifi connections and MTTQ server connections to enable IoT functionalities.
- **creds.h** Includes network credentials required to access the network.

6.1 main.c

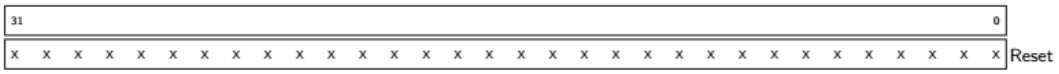
```
// Copyright (c) 2024 Mihiran Wickramarathne. All rights reserved.

#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_log.h"
#include "driver/gpio.h"

#include "ATM90E36.h"
#include "wifi.h"
#include "mqtt.h"

// GPIO pin definitions for SPI communication
#define GPIO_MOSI 5
#define GPIO_MISO 6
#define GPIO_SCLK 4
#define GPIO_CS 7

// Base address for ESP32-S3 GPIO registers
#define GPIO_Base 0x60004000
#define GPIO_OUT_W1TS_REG (GPIO_Base + 0x0008)
#define GPIO_OUT_W1TC_REG (GPIO_Base + 0x000C)
#define GPIO_ENABLE_REG   (GPIO_Base + 0x0020)

// Define the built-in LED pin
#define LED 17

void app_main(void) {
    // Pointers to GPIO registers
    volatile uint32_t* gpio_out_w1ts_reg = (volatile uint32_t*) GPIO_OUT_W1TS_REG;
    volatile uint32_t* gpio_out_w1tc_reg = (volatile uint32_t*) GPIO_OUT_W1TC_REG;
    volatile uint32_t* gpio_enable_reg = (volatile uint32_t*) GPIO_ENABLE_REG;

    // Enable GPIO outputs for LED and SPI pins
    *gpio_enable_reg = (1 << LED) | (1 << GPIO_CS) | (1 << GPIO_SCLK) | (1 << GPIO_MOSI);

    // Turn on the built-in LED
    *gpio_out_w1ts_reg = (1 << LED);

    // Initialize NVS for storing WiFi credentials
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    // Initialize WiFi in station mode
    wifi_init_sta();
    vTaskDelay(1000 / portTICK_PERIOD_MS);

    // Initialize SPI and ATM90E36 energy meter IC
    start_spi();
    begin_ATM90E36();

```

```
62      // Start the MQTT client
63      mqtt_app_start();
64
65      while (1) {
66
67
68
69
70          // Publishing measurment data to MQTT topics
71          double LineVoltageA = GetLineVoltageA();
72          printf("Line Voltage A: %f\n", LineVoltageA);
73          mqtt_publish("Energy_monitor/monitor_1/Phase_A/Line_Voltage_A",
    LineVoltageA);
74
75          double LineVoltageB = GetLineVoltageB();
76          printf("Line Voltage B: %f\n", LineVoltageB);
77          mqtt_publish("Energy_monitor/monitor_1/Phase_B/Line_Voltage_B",
    LineVoltageB);
78
79          double LineVoltageC = GetLineVoltageC();
80          printf("Line Voltage C: %f\n", LineVoltageC);
81          mqtt_publish("Energy_monitor/monitor_1/Phase_C/Line_Voltage_C",
    LineVoltageC);
82
83          double LineCurrentA = GetLineCurrentA();
84          printf("Line Current A: %f\n", LineCurrentA);
85          mqtt_publish("Energy_monitor/monitor_1/Phase_A/Line_Current_A",
    LineCurrentA);
86
87          double LineCurrentB = GetLineCurrentB();
88          printf("Line Current B: %f\n", LineCurrentB);
89          mqtt_publish("Energy_monitor/monitor_1/Phase_B/Line_Current_B",
    LineCurrentB);
90
91          double LineCurrentC = GetLineCurrentC();
92          printf("Line Current C: %f\n", LineCurrentC);
93          mqtt_publish("Energy_monitor/monitor_1/Phase_C/Line_Current_C",
    LineCurrentC);
94
95          double LineCurrentN = GetLineCurrentN();
96          printf("Line Current N: %f\n", LineCurrentN);
97          mqtt_publish("Energy_monitor/monitor_1/Neutral/Line_Current_N",
    LineCurrentN);
98
99          double ActivePowerA = GetActivePowerA();
100         printf("Active Power A: %f\n", ActivePowerA);
101         mqtt_publish("Energy_monitor/monitor_1/Phase_A/Active_Power_A",
    ActivePowerA);
102
103         double ActivePowerB = GetActivePowerB();
104         printf("Active Power B: %f\n", ActivePowerB);
105         mqtt_publish("Energy_monitor/monitor_1/Phase_B/Active_Power_B",
    ActivePowerB);
106
107         double ActivePowerC = GetActivePowerC();
108         printf("Active Power C: %f\n", ActivePowerC);
109         mqtt_publish("Energy_monitor/monitor_1/Phase_C/Active_Power_C",
    ActivePowerC);
110
111         double TotalActivePower = GetTotalActivePower();
112         printf("Total Active Power: %f\n", TotalActivePower);
113         mqtt_publish("Energy_monitor/monitor_1/Total_Active_Power",
    TotalActivePower);
114
115         double ReactivePowerA = GetReactivePowerA();
116         printf("Reactive Power A: %f\n", ReactivePowerA);
117         mqtt_publish("Energy_monitor/monitor_1/Phase_A/Reactive_Power_A",
```

```
    ReactivePowerA);

        double ReactivePowerB = GetReactivePowerB();
        printf("Reactive Power B: %f\n", ReactivePowerB);
        mqtt_publish("Energy_monitor/monitor_1/Phase_B/Reactive_Power_B",
    ReactivePowerB);

        double ReactivePowerC = GetReactivePowerC();
        printf("Reactive Power C: %f\n", ReactivePowerC);
        mqtt_publish("Energy_monitor/monitor_1/Phase_C/Reactive_Power_C",
    ReactivePowerC);

        double TotalReactivePower = GetTotalReactivePower();
        printf("Total Reactive Power: %f\n", TotalReactivePower);
        mqtt_publish("Energy_monitor/monitor_1/Total_Reactive_Power",
    TotalReactivePower);

        double ApparentPowerA = GetApparentPowerA();
        printf("Apparent Power A: %f\n", ApparentPowerA);
        mqtt_publish("Energy_monitor/monitor_1/Phase_A/Apparent_Power_A",
    ApparentPowerA);

        double ApparentPowerB = GetApparentPowerB();
        printf("Apparent Power B: %f\n", ApparentPowerB);
        mqtt_publish("Energy_monitor/monitor_1/Phase_B/Apparent_Power_B",
    ApparentPowerB);

        double ApparentPowerC = GetApparentPowerC();
        printf("Apparent Power C: %f\n", ApparentPowerC);
        mqtt_publish("Energy_monitor/monitor_1/Phase_C/Apparent_Power_C",
    ApparentPowerC);

        double TotalApparentPower = GetTotalApparentPower();
        printf("Total Apparent Power: %f\n", TotalApparentPower);
        mqtt_publish("Energy_monitor/monitor_1/Total_Apparent_Power",
    TotalApparentPower);

        double Frequency = GetFrequency();
        printf("Frequency: %f\n", Frequency);
        mqtt_publish("Energy_monitor/monitor_1/Frequency", Frequency);

        double PowerFactorA = GetPowerFactorA();
        printf("Power Factor A: %f\n", PowerFactorA);
        mqtt_publish("Energy_monitor/monitor_1/Phase_A/Power_Factor_A",
    PowerFactorA);

        double PowerFactorB = GetPowerFactorB();
        printf("Power Factor B: %f\n", PowerFactorB);
        mqtt_publish("Energy_monitor/monitor_1/Phase_B/Power_Factor_B",
    PowerFactorB);

        double PowerFactorC = GetPowerFactorC();
        printf("Power Factor C: %f\n", PowerFactorC);
        mqtt_publish("Energy_monitor/monitor_1/Phase_C/Power_Factor_C",
    PowerFactorC);

        double TotalPowerFactor = GetTotalPowerFactor();
        printf("Total Power Factor: %f\n", TotalPowerFactor);
        mqtt_publish("Energy_monitor/monitor_1/Total_Power_Factor",
    TotalPowerFactor);

        double PhaseA = GetPhaseA();
        printf("Phase A: %f\n", PhaseA);
        mqtt_publish("Energy_monitor/monitor_1/Phase_A/Phase_A", PhaseA);

        double PhaseB = GetPhaseB();
        printf("Phase B: %f\n", PhaseB);
```

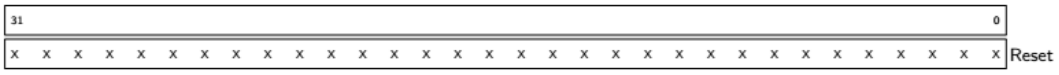
```
173         mqtt_publish("Energy_monitor/monitor_1/Phase_B/Phase_B", PhaseB);
174
175         double PhaseC = GetPhaseC();
176         printf("Phase C: %f\n", PhaseC);
177         mqtt_publish("Energy_monitor/monitor_1/Phase_C/Phase_C", PhaseC);
178
179         double Temperature = GetTemperature();
180         printf("Temperature: %f\n", Temperature);
181         mqtt_publish("Energy_monitor/monitor_1/Temperature", Temperature);
182
183 //This limits Speeds
184         // Blink the LED
185         *gpio_out_w1ts_reg = (1 << LED);
186         vTaskDelay(100 / portTICK_PERIOD_MS);
187
188         *gpio_out_w1tc_reg = (1 << LED);
189         vTaskDelay(100 / portTICK_PERIOD_MS);
190     }
191 }
```

Register 4.2. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS_REG GPIO0-31 output set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_REG will be set. (WO)

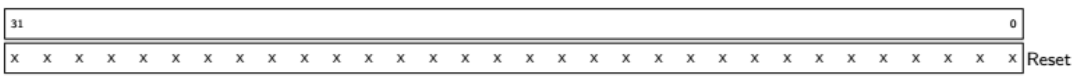
Register 4.3. GPIO_OUT_W1TC_REG (0x000c)



GPIO_OUT_W1TC_REG GPIO0-31 output clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_REG will be cleared. (WO)

Figure 52: GPIO_OUT_W1TS_REG and GPIO_OUT_W1TC_REG Registers

Register 4.7. GPIO_ENABLE_REG (0x0020)



GPIO_ENABLE_REG GPIO0-31 output enable. (R/W)

Figure 53: GPIO_ENABLE_REG Register

6.2 ATM90E36.c

```
// MIT License

// Copyright (c) 2016 whatnick and Ryzee

// Copyright (c) 2024 Mihiran Wickramarathne

// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

// The above copyright notice and this permission notice shall be included in
    all
// copies or substantial portions of the Software.

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#include "ATM90E36.h"
#include <stdio.h>
#include <string.h>
#include "driver/spi_master.h"


//Pins in the project
#define GPIO_MOSI 5
#define GPIO_MISO 6
#define GPIO_SCLK 4
#define GPIO_CS 7

#define WRITE 0 // WRITE SPI
#define READ 1  // READ SPI

//SPI Configuration
spi_transaction_t t;
spi_device_handle_t spi;
esp_err_t ret;


void start_spi(void){
spi_host_device_t spi_host = SPI3_HOST;

spi_bus_config_t buscfg;
memset(&buscfg, 0, sizeof(buscfg));
buscfg.mosi_io_num = GPIO_MOSI;
buscfg.miso_io_num = GPIO_MISO;
buscfg.sclk_io_num = GPIO_SCLK;
buscfg.quadwp_io_num = -1;
buscfg.quadhd_io_num = -1;
buscfg.max_transfer_sz = 1;
buscfg.flags = SPICOMMON_BUSFLAG_MASTER ; /*| SPICOMMON_BUSFLAG_IOMUX_PINS ;*/
buscfg.intr_flags = 0;

spi_device_interface_config_t devcfg;
memset(&devcfg, 0, sizeof(devcfg));
devcfg.command_bits = 0;
devcfg.address_bits = 16;
devcfg.dummy_bits = 0;
```

```
devcfg.mode = (uint8_t) 3; //SPI_MODE;
devcfg.duty_cycle_pos = 0;
devcfg.cs_ena_pretrans = 0;
devcfg.cs_ena_posttrans = (uint8_t) 0; //CS_ENA_POSTTRANS;
devcfg.clock_speed_hz = 300 * 1000; //SPI_HZ;
devcfg.input_delay_ns = 50; // INPUT_DELAY_NS;
devcfg.spics_io_num = GPIO_CS;
devcfg.flags = SPI_DEVICE_NO_DUMMY ;
devcfg.queue_size = 1;
devcfg.pre_cb = 0;
devcfg.post_cb = 0;
devcfg.input_delay_ns = 400;

ret = spi_bus_initialize(spi_host, &buscfg, 0); // No DMA
ESP_ERROR_CHECK(ret);

ret =  spi_bus_add_device(spi_host, &devcfg, &spi);
ESP_ERROR_CHECK(ret);

memset(&t, 0, sizeof(t));
t.flags = SPI_TRANS_USE_TXDATA | SPI_TRANS_USE_RXDATA  ;
t.length = 32;
t.rxlength = 16;
}


unsigned short CommEnergyIC(unsigned char RW, unsigned short address, unsigned
    short val)
    {
        //Sets the adress and read and write
        uint16_t addrss;
        uint16_t response;
        uint16_t data;
        if(RW){
            addrss = 0x1<<15 | address; // Read
            data = 0x0;
        }
        else{
            addrss = 0x0000 | address;//Write
            data = val;
        }
        spi_device_acquire_bus(spi, portMAX_DELAY);
        *((uint16_t*)t.tx_data) = SPI_SWAP_DATA_TX(data, 16); // Writing the
    data to the TX buffer
        t.addr = addrss;// Writing the address to the address buffer
        ret =  spi_device_polling_transmit(spi, &t);// Transmitting the data
        response = SPI_SWAP_DATA_RX(*((uint16_t*)t.rx_data), 16);// Reading the
    data from the RX buffer
        spi_device_release_bus(spi);
        vTaskDelay(10);
        return response;
    }


// Parameters Functions

/*
- Gets main electrical parameters,
such as: Voltage, Current, Power, Energy,
and Frequency
- Also gets the temperature
*/


// VOLTAGE
double  GetLineVoltageA() {
  unsigned short voltage = CommEnergyIC(READ, UrmsA, 0xFFFF);
```

```
  return (double)voltage / 100;
}

double  GetLineVoltageB() {
  unsigned short voltage = CommEnergyIC(READ, UrmsB, 0xFFFF);
  return (double)voltage / 100;
}

double GetLineVoltageC() {
  unsigned short voltage = CommEnergyIC(READ, UrmsC, 0xFFFF);
  return (double)voltage / 100;
}


// CURRENT
double GetLineCurrentA() {
  unsigned short current = CommEnergyIC(READ, IrmsA, 0xFFFF);
  return (double)current / 1000;
}
double GetLineCurrentB() {
  unsigned short current = CommEnergyIC(READ, IrmsB, 0xFFFF);
  return (double)current / 1000;
}
double GetLineCurrentC() {
  unsigned short current = CommEnergyIC(READ, IrmsC, 0xFFFF);
  return (double)current / 1000;
}
double GetLineCurrentN() {
  unsigned short current = CommEnergyIC(READ, IrmsN0, 0xFFFF);
  return (double)current / 1000;
}


// ACTIVE POWER
double GetActivePowerA() {
  signed short apower = (signed short) CommEnergyIC(READ, PmeanA, 0xFFFF);
  if (apower & 0x8000) {
    apower= (apower & 0x7FFF) * -1;
  }
  return (double)apower / 1000;
}
double GetActivePowerB() {
  signed short apower = (signed short) CommEnergyIC(READ, PmeanB, 0xFFFF);
  if (apower & 0x8000) {
    apower= (apower & 0x7FFF) * -1;
  }
  return (double)apower / 1000;
}
double GetActivePowerC() {
  signed short apower = (signed short) CommEnergyIC(READ, PmeanC, 0xFFFF);
  if (apower & 0x8000) {
    apower= (apower & 0x7FFF) * -1;
  }
  return (double)apower / 1000;
}
double GetTotalActivePower() {
  signed short apower = (signed short) CommEnergyIC(READ, PmeanT, 0xFFFF);
  if (apower & 0x8000) {
    apower= (apower & 0x7FFF) * -1;
  }
  return (double)apower / 250;
}


// REACTIVE POWER
double GetReactivePowerA() {
  signed short apower = (signed short) CommEnergyIC(READ, QmeanA, 0xFFFF);
```

```
196   if (apower & 0x8000) {
197     apower= (apower & 0x7FFF) * -1;
198   }
199   return (double)apower / 1000;
200 }
201 double GetReactivePowerB() {
202   signed short apower = (signed short) CommEnergyIC(READ, QmeanB, 0xFFFF);
203   if (apower & 0x8000) {
204     apower= (apower & 0x7FFF) * -1;
205   }
206   return (double)apower / 1000;
207 }
208 double GetReactivePowerC() {
209   signed short apower = (signed short) CommEnergyIC(READ, QmeanC, 0xFFFF);
210   if (apower & 0x8000) {
211     apower= (apower & 0x7FFF) * -1;
212   }
213   return (double)apower / 1000;
214 }
215 double GetTotalReactivePower() {
216   signed short apower = (signed short) CommEnergyIC(READ, QmeanT, 0xFFFF);
217   if (apower & 0x8000) {
218     apower= (apower & 0x7FFF) * -1;
219   }
220   return (double)apower / 250;
221 }
222
223
224 // APPARENT POWER
225 double GetApparentPowerA() {
226   signed short apower = (signed short) CommEnergyIC(READ, SmeanA, 0xFFFF);
227   if (apower & 0x8000) {
228     apower= (apower & 0x7FFF) * -1;
229   }
230   return (double)apower / 1000;
231 }
232 double GetApparentPowerB() {
233  signed short apower = (signed short) CommEnergyIC(READ, SmeanB, 0xFFFF);
234   if (apower & 0x8000) {
235     apower= (apower & 0x7FFF) * -1;
236   }
237   return (double)apower / 1000;
238 }
239 double GetApparentPowerC() {
240   signed short apower = (signed short) CommEnergyIC(READ, SmeanC, 0xFFFF);
241   if (apower & 0x8000) {
242     apower= (apower & 0x7FFF) * -1;
243   }
244   return (double)apower / 1000;
245 }
246 double GetTotalApparentPower() {
247   signed short apower = (signed short) CommEnergyIC(READ, SmeanT, 0xFFFF);
248   if (apower & 0x8000) {
249     apower= (apower & 0x7FFF) * -1;
250   }
251   return (double)apower / 250;
252 }
253
254
255 // FREQUENCY
256 double GetFrequency() {
257   unsigned short freq = CommEnergyIC(READ, Freq, 0xFFFF);
258   return (double)freq / 100;
259 }
260
261
262 // POWER FACTOR
```

```
263 double GetPowerFactorA() {
264   short pf = (short) CommEnergyIC(READ, PFmeanA, 0xFFFF);
265   //if negative
266   if (pf & 0x8000) {
267     pf = (pf & 0x7FFF) * -1;
268   }
269   return (double)pf / 1000;
270 }
271 double GetPowerFactorB() {
272   short pf = (short) CommEnergyIC(READ, PFmeanB, 0xFFFF);
273   if (pf & 0x8000) {
274     pf = (pf & 0x7FFF) * -1;
275   }
276   return (double)pf / 1000;
277 }
278 double GetPowerFactorC() {
279   short pf = (short) CommEnergyIC(READ, PFmeanC, 0xFFFF);
280   //if negative
281   if (pf & 0x8000) {
282     pf = (pf & 0x7FFF) * -1;
283   }
284   return (double)pf / 1000;
285 }
286 double GetTotalPowerFactor() {
287   short pf = (short) CommEnergyIC(READ, PFmeanT, 0xFFFF);
288   //if negative
289   if (pf & 0x8000) {
290     pf = (pf & 0x7FFF) * -1;
291   }
292   return (double)pf / 1000;
293 }
294
295
296 // PHASE ANGLE
297 double GetPhaseA() {
298   signed short apower = (signed short) CommEnergyIC(READ, PAngleA, 0xFFFF);
299   return (double)apower / 10;
300 }
301 double GetPhaseB() {
302   signed short apower = (signed short) CommEnergyIC(READ, PAngleB, 0xFFFF);
303   return (double)apower / 10;
304 }
305 double GetPhaseC() {
306   signed short apower = (signed short) CommEnergyIC(READ, PAngleC, 0xFFFF);
307   return (double)apower / 10;
308 }
309
310
311 // TEMPERATURE
312 double GetTemperature() {
313   short int apower = (short int) CommEnergyIC(READ, Temp, 0xFFFF);
314   return (double)apower;
315 }
316
317
318 /* BEGIN FUNCTION */
319 void begin_ATM90E36(void)
320 {
321   CommEnergyIC(WRITE, SoftReset, 0x789A);    // Perform soft reset
322   CommEnergyIC(WRITE, FuncEn0, 0x0000);      // Voltage sag
323   CommEnergyIC(WRITE, FuncEn1, 0x0000);      // Voltage sag
324   CommEnergyIC(WRITE, SagTh, 0x0001);        // Voltage sag threshold
325
326   /* SagTh = Vth * 100 * sqrt(2) / (2 * Ugain / 32768) */
327
328   //Set metering config values (CONFIG)
329   CommEnergyIC(WRITE, ConfigStart, 0x5678); // Metering calibration startup
```

```
330   CommEnergyIC(WRITE, PLconstH, 0x0861);     // PL Constant MSB (default)
331   CommEnergyIC(WRITE, PLconstL, 0xC468);     // PL Constant LSB (default)
332   CommEnergyIC(WRITE, MMode0, 0x1087);       // Mode Config (60 Hz, 3P4W)
333   CommEnergyIC(WRITE, MMode1, 0x1500);       // 0x5555 (x2) // 0x0000 (1x)
334   CommEnergyIC(WRITE, PStartTh, 0x0000);     // Active Startup Power Threshold
335   CommEnergyIC(WRITE, QStartTh, 0x0000);     // Reactive Startup Power Threshold
336   CommEnergyIC(WRITE, SStartTh, 0x0000);     // Apparent Startup Power Threshold
337   CommEnergyIC(WRITE, PPhaseTh, 0x0000);     // Active Phase Threshold
338   CommEnergyIC(WRITE, QPhaseTh, 0x0000);     // Reactive Phase Threshold
339   CommEnergyIC(WRITE, SPhaseTh, 0x0000);     // Apparent  Phase Threshold
340   CommEnergyIC(WRITE, CSZero, 0x4741);       // Checksum 0
341
342   //Set metering calibration values (CALIBRATION)
343   CommEnergyIC(WRITE, CalStart, 0x5678);     // Metering calibration startup
344   CommEnergyIC(WRITE, GainA, 0x0000);        // Line calibration gain
345   CommEnergyIC(WRITE, PhiA, 0x0000);         // Line calibration angle
346   CommEnergyIC(WRITE, GainB, 0x0000);        // Line calibration gain
347   CommEnergyIC(WRITE, PhiB, 0x0000);         // Line calibration angle
348   CommEnergyIC(WRITE, GainC, 0x0000);        // Line calibration gain
349   CommEnergyIC(WRITE, PhiC, 0x0000);         // Line calibration angle
350   CommEnergyIC(WRITE, PoffsetA, 0x0000);     // A line active power offset
351   CommEnergyIC(WRITE, QoffsetA, 0x0000);     // A line reactive power offset
352   CommEnergyIC(WRITE, PoffsetB, 0x0000);     // B line active power offset
353   CommEnergyIC(WRITE, QoffsetB, 0x0000);     // B line reactive power offset
354   CommEnergyIC(WRITE, PoffsetC, 0x0000);     // C line active power offset
355   CommEnergyIC(WRITE, QoffsetC, 0x0000);     // C line reactive power offset
356   CommEnergyIC(WRITE, CSOne, 0x0000);        // Checksum 1
357
358   //Set metering calibration values (HARMONIC)
359   CommEnergyIC(WRITE, HarmStart, 0x5678);    // Metering calibration startup
360   CommEnergyIC(WRITE, POffsetAF, 0x0000);    // A Fund. active power offset
361   CommEnergyIC(WRITE, POffsetBF, 0x0000);    // B Fund. active power offset
362   CommEnergyIC(WRITE, POffsetCF, 0x0000);    // C Fund. active power offset
363   CommEnergyIC(WRITE, PGainAF, 0x0000);      // A Fund. active power gain
364   CommEnergyIC(WRITE, PGainBF, 0x0000);      // B Fund. active power gain
365   CommEnergyIC(WRITE, PGainCF, 0x0000);      // C Fund. active power gain
366   CommEnergyIC(WRITE, CSTwo, 0x0000);        // Checksum 2
367
368   //Set measurement calibration values (ADJUST)
369   CommEnergyIC(WRITE, AdjStart, 0x5678);     // Measurement calibration
370   CommEnergyIC(WRITE, UgainA, 0x0002);       // A SVoltage rms gain
371   CommEnergyIC(WRITE, IgainA, 0xFD7F);       // A line current gain
372   CommEnergyIC(WRITE, UoffsetA, 0x0000);     // A Voltage offset
373   CommEnergyIC(WRITE, IoffsetA, 0x0000);     // A line current offset
374   CommEnergyIC(WRITE, UgainB, 0x0002);       // B Voltage rms gain
375   CommEnergyIC(WRITE, IgainB, 0xFD7F);       // B line current gain
376   CommEnergyIC(WRITE, UoffsetB, 0x0000);     // B Voltage offset
377   CommEnergyIC(WRITE, IoffsetB, 0x0000);     // B line current offset
378   CommEnergyIC(WRITE, UgainC, 0x0002);       // C Voltage rms gain
379   CommEnergyIC(WRITE, IgainC, 0xFD7F);       // C line current gain
380   CommEnergyIC(WRITE, UoffsetC, 0x0000);     // C Voltage offset
381   CommEnergyIC(WRITE, IoffsetC, 0x0000);     // C line current offset
382   CommEnergyIC(WRITE, IgainN, 0xFD7F);       // C line current gain
383   CommEnergyIC(WRITE, CSThree, 0x02F6);      // Checksum 3
384
385   // Done with the configuration
386   CommEnergyIC(WRITE, ConfigStart, 0x5678);
387   CommEnergyIC(WRITE, CalStart, 0x5678);     // 0x6886 //0x5678 //8765);
388   CommEnergyIC(WRITE, HarmStart, 0x5678);    // 0x6886 //0x5678 //8765);
389   CommEnergyIC(WRITE, AdjStart, 0x5678);     // 0x6886 //0x5678 //8765);
390   CommEnergyIC(WRITE, SoftReset, 0x789A);    // Perform soft reset
391 }
```

Access type of SPI interface is determined by first bit on SDI and has 15 bit address (total 16 bits).

Access type:

The first bit on SDI defines the access type as below:

Instruction	Description	Instruction Format
Read	read from registers	1
Write	write to registers	0

Address:

Fixed 15-bit, following the access type bits. The lower 10-bit is decoded as address; the higher 5 bits are 'Don't Care'.

Read/Write data:

Fixed as 16 bits.

Figure 54: SPI access Type of Metering IC

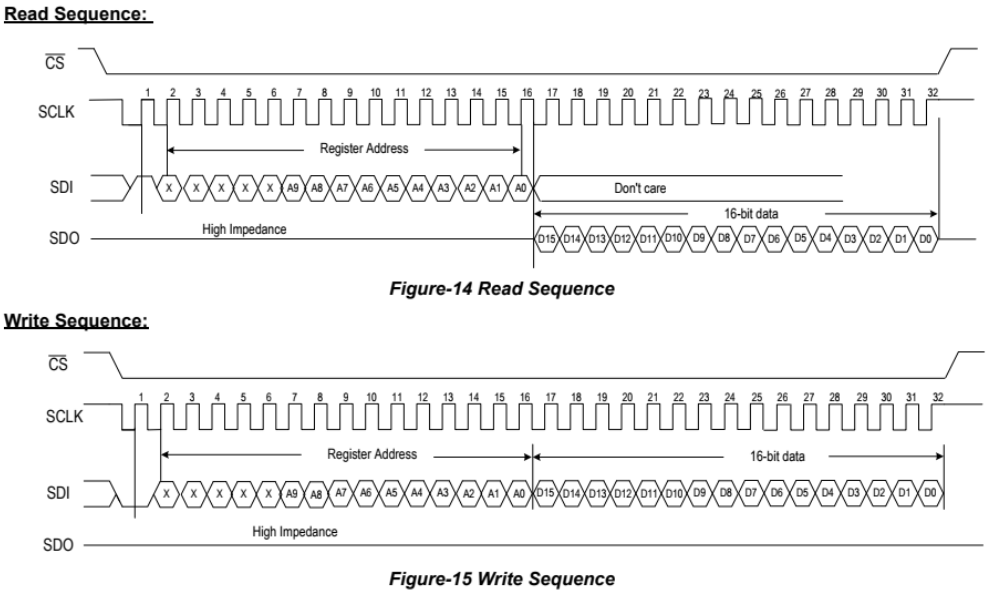


Figure 55: SPI Timing Diagram

SPI clock frequency is determined by the following equation.

$$f_{\text{SCLK}} = \frac{f_{\text{sys\_clk}}}{\text{CLK\_DIV} \times 2 + 2} \tag{1}$$

With the crystal oscillator of 16.384MHz and clock division rate of 25 (recommended), we have used $\approx$ 30MHz.

Note: *Additional information about the measurment registers can be found in Atmel M90E36A Datasheet. (Page 53)*

6.3 wifi.c

```c
// Copyright (c) 2015-2023 Espressif Systems (Shanghai) CO LTD
// Copyright (c) 2024 Mihiran Wickramarathne. All rights reserved.

#include "wifi.h"
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "lwip/err.h"
#include "lwip/sys.h"
#include "creds.h"

// WiFi credentials from the "creds.h" file
#define ESP_WIFI_SSID      WIFI_SSID
#define ESP_WIFI_PASS      WIFI_PASSWORD
#define ESP_MAXIMUM_RETRY  30  // Maximum number of retry attempts for connecting to the WiFi

// FreeRTOS event group to signal when the device is connected to WiFi
static EventGroupHandle_t s_wifi_event_group;

// Define bits for different WiFi events
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static const char *TAG = "wifi station";  // Tag for logging
static int s_retry_num = 0;  // Retry counter

// Event handler for WiFi and IP events
static void event_handler(void* arg, esp_event_base_t event_base,
                          int32_t event_id, void* event_data)
{
    // Event when WiFi station starts
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    }
    // Event when WiFi station is disconnected
    else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_num < ESP_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;  // Increment retry counter
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            esp_restart();  // Restart the ESP32 if maximum retries are reached
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG, "connect to the AP fail");
    }
    // Event when WiFi station gets an IP address
    else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;  // Reset retry counter
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

// Function to initialize WiFi in station mode
void wifi_init_sta(void)
```

```
{
    // Create the event group
    s_wifi_event_group = xEventGroupCreate();

    // Initialize the underlying TCP/IP stack
    ESP_ERROR_CHECK(esp_netif_init());

    // Create the default event loop
    ESP_ERROR_CHECK(esp_event_loop_create_default());

    // Create default WiFi station interface
    esp_netif_create_default_wifi_sta();

    // Initialize WiFi with default configuration
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    // Register event handlers for WiFi and IP events
    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_got_ip));

    // Configure WiFi connection settings
    wifi_config_t wifi_config = {
        .sta = {
            .ssid = ESP_WIFI_SSID,
            .password = ESP_WIFI_PASS,
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));  // Set WiFi mode to
    station
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));  // Set
    WiFi configuration
    ESP_ERROR_CHECK(esp_wifi_start());  // Start WiFi

    ESP_LOGI(TAG, "wifi_init_sta finished.");

    // Wait for WiFi connection or failure event
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
                                           pdFALSE,
                                           pdFALSE,
                                           portMAX_DELAY);

    // Check which event occurred
    if (bits & WIFI_CONNECTED_BIT) {
        ESP_LOGI(TAG, "connected to ap SSID:%s password:%s",
                 WIFI_SSID, ESP_WIFI_PASS);
    } else if (bits & WIFI_FAIL_BIT) {
        ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",
                 ESP_WIFI_SSID, ESP_WIFI_PASS);
    } else {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }
}
```

6.4 mttq.c

```c
// Copyright (c) 2015-2023 Espressif Systems (Shanghai) CO LTD
// Copyright (c) 2024 Mihiran Wickramarathne. All rights reserved.

#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"

#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"

#include "esp_log.h"
#include "mqtt_client.h"

#include "mqtt.h"
#include "creds.h"

static const char *TAG = "MQTT_TCP";  // Tag for logging

// MQTT credentials from the "creds.h" file
char mqtt_username[] = MQTT_USERNAME;
char mqtt_password[] = MQTT_PASSWORD;

esp_mqtt_client_handle_t client;  // MQTT client handle

// Callback function to handle different MQTT events
static esp_err_t mqtt_event_handler_cb(esp_mqtt_event_handle_t event)
{
    esp_mqtt_client_handle_t client = event->client;  // Get the client handle from the event
    switch (event->event_id)  // Handle different event IDs
    {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG, "MQTT_EVENT_DATA");
        printf("\nTOPIC=%.*s\r\n", event->topic_len, event->topic);
        printf("DATA=%.*s\r\n", event->data_len, event->data);
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
```

```
65          break;
66      default:
67          ESP_LOGI(TAG, "Other event id:%d", event->event_id);
68          break;
69      }
70      return ESP_OK;
71  }
72
73  // Event handler function to route events to the callback
74  static void mqtt_event_handler(void *handler_args, esp_event_base_t base,
        int32_t event_id, void *event_data)
75  {
76      mqtt_event_handler_cb(event_data);
77  }
78
79  // Function to start the MQTT client
80  void mqtt_app_start(void)
81  {
82      // MQTT configuration structure
83      esp_mqtt_client_config_t mqtt_cfg = {
84          .broker.address.uri = "mqtt://137.184.9.146",  // MQTT broker URI
85          .credentials.username = &mqtt_username,  // MQTT username
86          .credentials.authentication.password = &mqtt_password,  // MQTT password
87      };
88
89      client = esp_mqtt_client_init(&mqtt_cfg);  // Initialize the MQTT client
90      esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler,
        client);  // Register event handler
91      esp_mqtt_client_start(client);  // Start the MQTT client
92  }
93
94  // Function to publish data to an MQTT topic
95  void mqtt_publish(char *topic, double value)
96  {
97      char data[100];
98      sprintf(data, "%f", value);  // Convert double to string
99      esp_mqtt_client_publish(client, topic, data, 0, 1, 0);  // Publish data to
        the specified topic
100 }
```

6.5 References

1. SPI Master: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/spi$_{m}aster.html$
2. ESP WiFi: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp$_{w}ifi.html$
3. MQTT: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/mqtt.html
4. ESP S3 Manual: https://www.espressif.com/sites/default/files/documentation/esp32-s3$_{t}echnical_{r}eference_{m}anual_{e}n.pdf$

7 System Integration

Figure 56

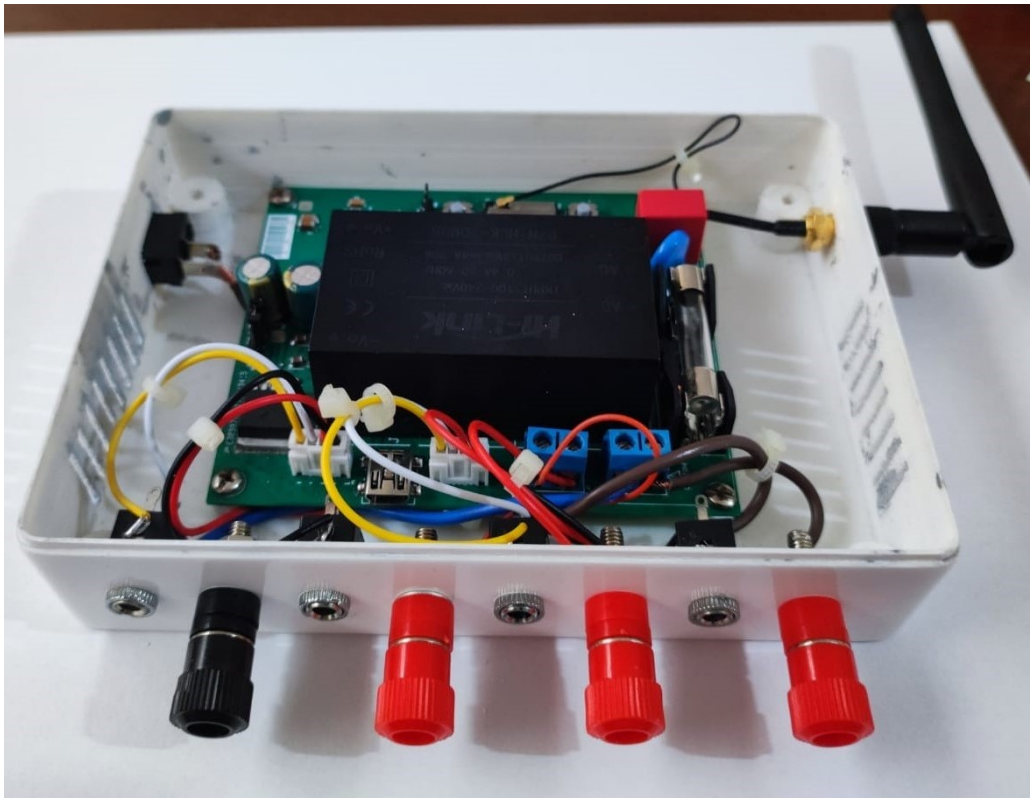


Figure 57

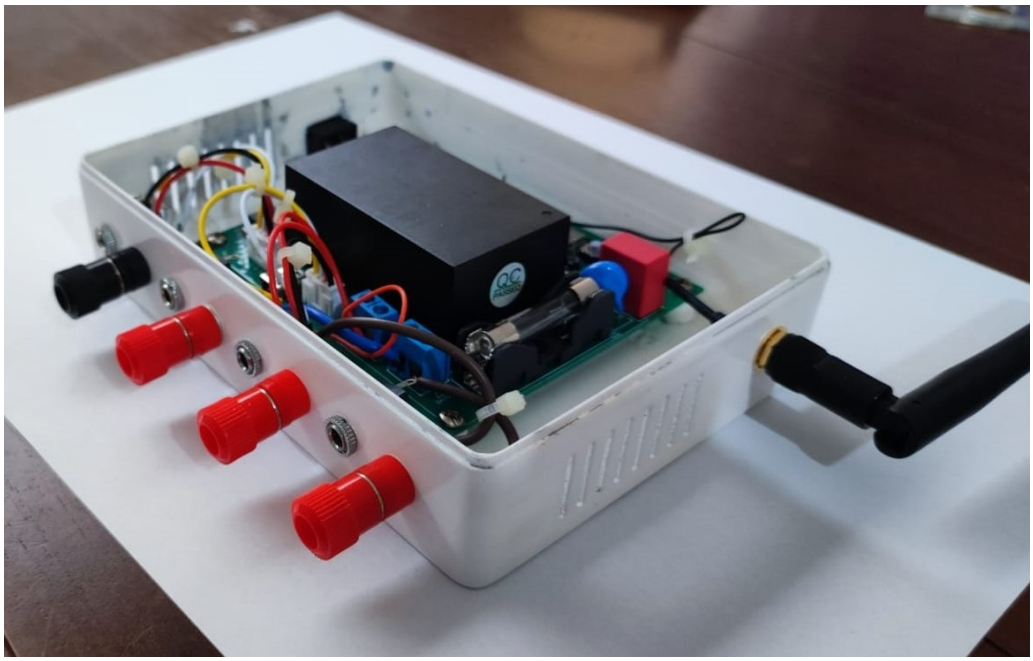


Figure 58

Figure 59

8 Conclusion

Throughout the design process of the Industrial Power Monitoring System, we focused on creating a reliable and efficient solution tailored for industrial applications. Key outcomes include the successful integration of sensors, data processing capabilities, and robust communication channels for real-time data transmission and analysis. Our modular approach ensures scalability and adaptability to diverse industrial environments, facilitating seamless deployment and operation.

Looking ahead, potential enhancements could further refine the system's performance and capabilities. Future improvements may include enhancing sensor accuracy, expanding data analytics capabilities for deeper insights, and integrating predictive maintenance features to optimize operational efficiency. Continuous iteration based on user feedback and technological advancements will be crucial to staying at the forefront of industrial energy management solutions.

In conclusion, the Industrial Power Monitoring System not only addresses current industry needs but also sets a foundation for ongoing innovation and improvement in energy monitoring and management practices. We are grateful to Prof. Jayasinghe for the support and guidance that enabled us to navigate challenges and achieve our design goals effectively.

9 Appendix

This section presents a comprehensive record of the activities undertaken throughout the design process of the Industrial Power Monitoring System on a weekly basis.

9.1 Initialization Process (17 - 23 February 2024)

We thoroughly analyzed the project's scope, objectives, and target user market. This involved identifying the core problem and defining clear objectives aligned with our vision. Market research was conducted to assess conditions and competition, while detailed user personas were developed to understand user needs and preferences. These insights guided our decision-making process, ensuring our project addressed relevant market demands and user requirements effectively.

9.2 Field Visit to Observe Users (24th of Feruary 2024)

We visited a plastic molding factory in order to observe users and find their specific requirements. Throughout our time at the facility, we observed the operations and interactions of workers with the existing machinery and systems. We noted various aspects such as workflow patterns, common tasks performed, challenges faced by operators, and areas where improvements could be made. By closely engaging with the users and witnessing their workflow firsthand, we were able to make several key assumptions.

- Machines are distributed over the factory premises.
- Some high power machines may have built in PLCs with limited energy monitoring capabilities.
- Techniques such as power factor correction is required to increase the efficiency.
- Multiple Power distribution boards exists, but may not have the same configuration.

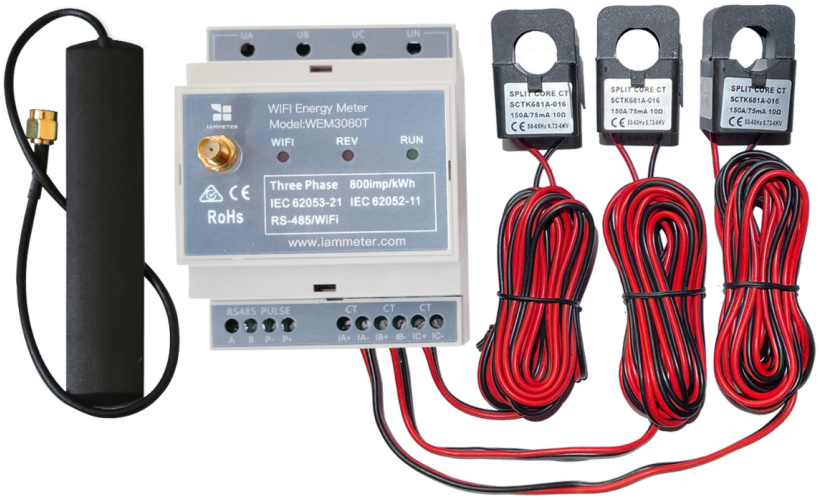
With these observations, we have identified several user requirements for our device.

- Device needs a modular configuration in order to ease the adoption process in to existing machinery.
- It need to be standalone as it need to operate in a distributed configuration.
- Device need to provide data required for power quality analysis.

9.3 Observing Existing Products (25th of February - 2nd of March 2023)

As part of our market research initiative, we conducted a thorough analysis of existing energy monitoring devices to gain insights into their design and architecture. This involved studying a diverse range of products available in the market, including smart meters, energy monitoring systems, and IoT-enabled devices. During this process, we have identified several noticeable designs.

- WEM3080T by IAMMETER
  - Have 3 phase energy metering capabilities and embedded WiFi module for data transmission.
  - Only support single channel power monitoring



- 3-Phase Power Meter by VP instrument
  - Have 3 phase energy metering capabilities and have RS485 (Modbus RTU)output.
  - Only support single channel power monitoring and no WiFi communication

- Green 3-Phase Smart Energy Meter by Renesas
  - Have 3 phase energy metering capabilities and have remote data acquisition capabilities.
  - Only support single channel power monitoring.

- VTX-34 is Electronic Three Phase Energy Meter
  - Have 3 phase energy metering capabilities.
  - Only support single channel power monitoring and no Wi-Fi communication

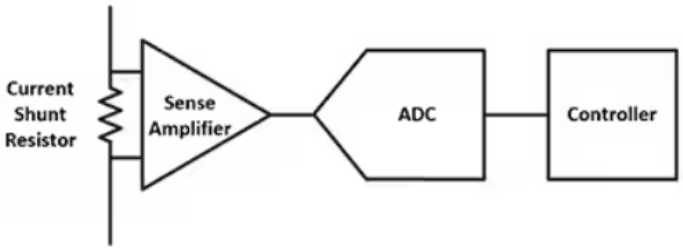
One of the key observation is that all of these devices uses current transformers for current sensing. That prompt us to search for current sensing methods.

9.4 Current Sensor Selection (3 - 9 March 2024)

For current sensor, we considered 2 approaches:

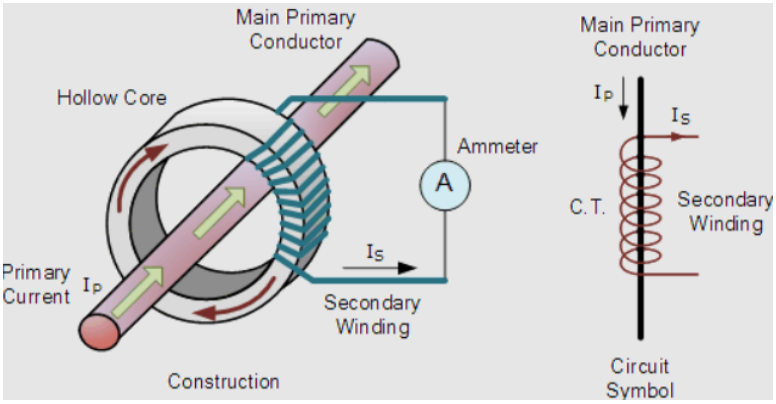
9.4.1 Shunt Resistors

Current sensing resistors , also known as shunt resistors, are utilized for measuring the current flow within a circuit. Their primary function involves detecting the current and converting it into a measurable output voltage using Ohm's law. Typically, these resistors are characterized by low values and high power ratings. Their design prioritizes low resistance to minimize power dissipation and mitigate the potential for short circuits, which could otherwise cause harm to surrounding components.



9.4.2 Current Transformer

A Current Transformer (CT) is employed for the measurement of current within another circuit. These devices have widespread use internationally for the monitoring of high-voltage lines within national power grids. The CT operates by generating an alternating current in its secondary winding, which is directly proportional to the current being measured in its primary winding. This process effectively reduces a high current to a lower value, thereby enabling safe monitoring of electrical current flow within an AC transmission line.



After several consideration we decided to use current transformers.

- Safety: Transformers play a crucial role in high-voltage systems where directly measuring current can pose safety risks. Current transformers are therefore employed to lower the current to a level that is safe and feasible for measurement.
- Accuracy: Current transformers ensure precise and dependable meter readings, even when dealing with high currents. This accuracy is particularly important in applications where exact measurements are essential.
- Isolation: The primary winding of a current transformer is electrically isolated from its secondary winding. This isolation provides an additional layer of protection against potential faults in the primary circuit.

- Reduced Power Losses: By stepping down the current before measurement, three-phase current transformers significantly reduce power dissipation in the metering circuit. This approach effectively minimizes power losses.

9.5 Metering IC Selection (10 - 16 March 2024)

In this designs we have considered 3 different metering ICs.

- Atmel M90E36A
- Microchip MCP3903
- ADE7752A

9.5.1 Atmel M90E36A

The ATM90E36A is a high-performance energy metering IC designed for three-phase power systems. It has 3 channels with measurement capabilities and Fourier analysis functions which can used for power monitoring instruments. It can provide accurate measurements of various parameters such as voltage, current, total harmonic distortion (THD), discrete Fourier transform (DFT), mean power etc.

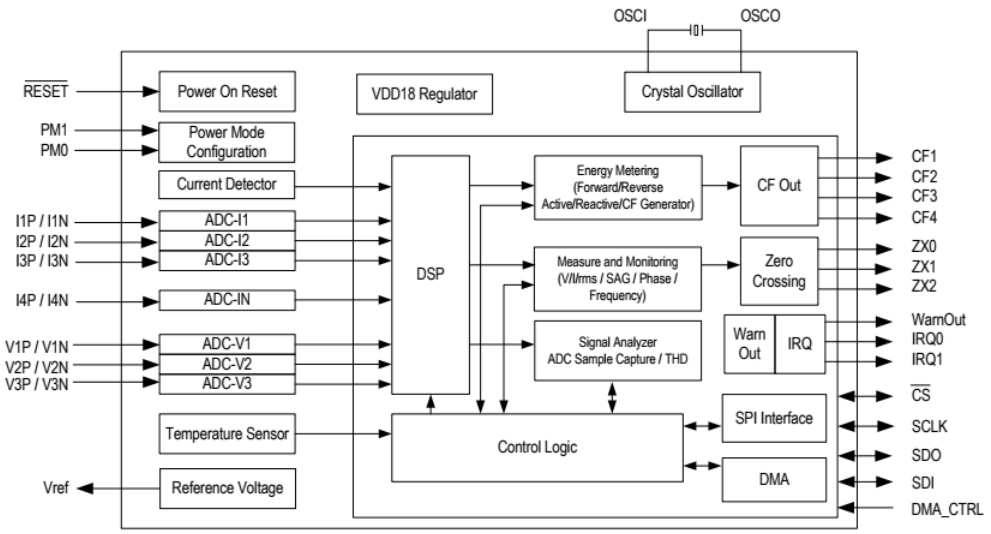


Figure 60: M90E36A Block Diagram

9.5.2 Microchip MCP3909

The MCP3909 is an energy measurement IC designed to comply with the IEC 62053 international energy metering specification. Its output includes a frequency proportional to the average active (real) power at the inputs, alongside a simultaneous serial SPI interface for accessing the ADC channels and multiplier output data. With a dynamic range of 1000:1, the MCP3909 integrates two 16-bit delta-sigma ADCs, offering programmable gain up to 16.

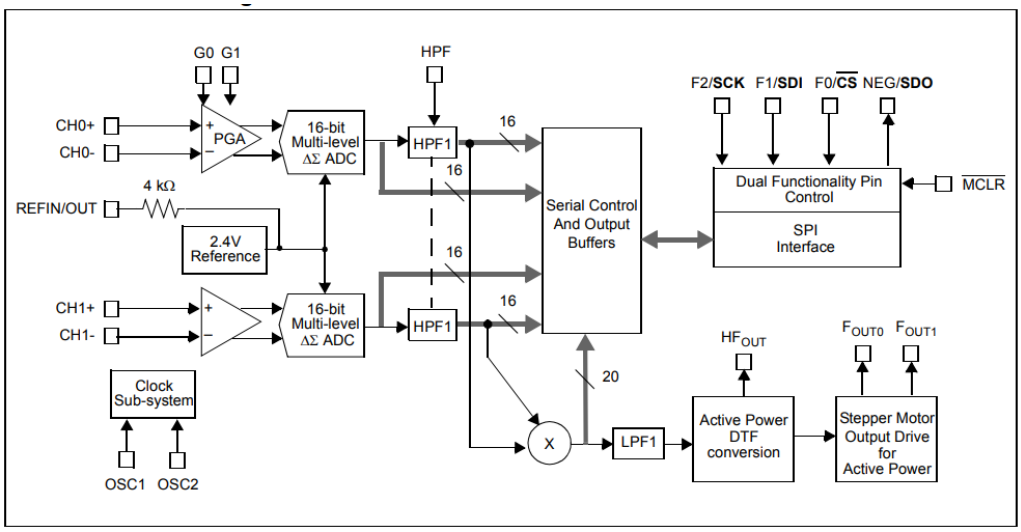


Figure 61: MCP3909 Block Diagram

9.5.3 Analog Devices ADE7752A

ADE7752 is a high accuracy polyphase electrical energy measurement IC.It includes dual-channel analog-to-digital converters (ADCs) for simultaneous sampling of voltage and current waveforms, enabling precise measurement of active energy, reactive energy, and instantaneous voltage and current. The IC also integrates digital signal processing capabilities to perform complex calculations and metering functions. On-chip reference oscillators ensure stable operation, while features like temperature compensation and calibration enhance accuracy across varying environmental conditions. Communication is facilitated through a serial interface, typically SPI, allowing for easy integration with external microcontrollers or host systems.

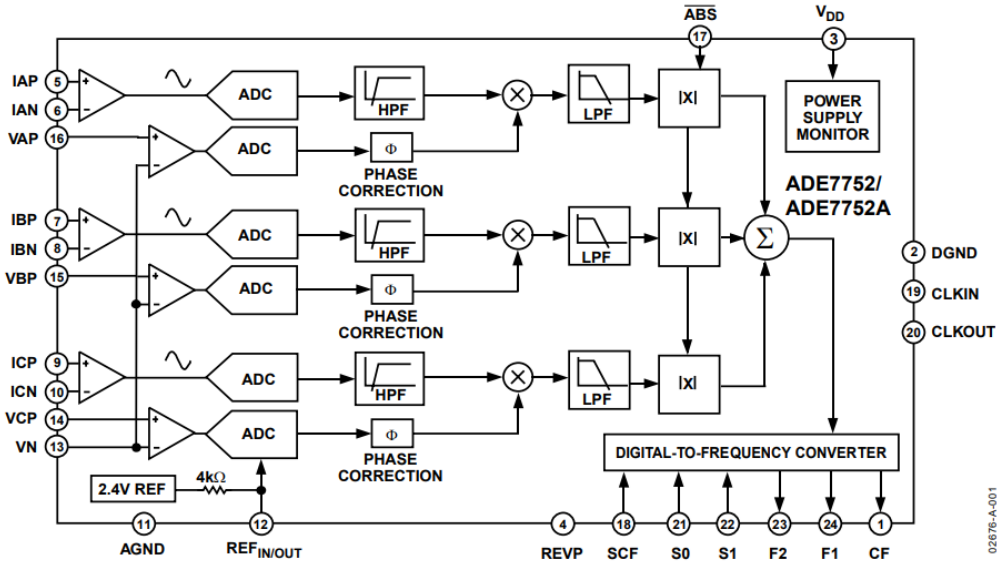


Figure 62: ADE7752A Block Diagram

We rejected MCP3909 since it only has 2 ADC, which will force us to use 3 metering IC to in a signle device in order to measure 3 phase energy. We selected M90E36A over ADE7752A because not only it has sufficient number of ADCs but also built in functionality to measure various parameters such as voltage, current, total harmonic distortion (THD), discrete Fourier transform (DFT), mean power etc.

9.6 M90E36A Register Configuration (17 - 23 March 2024)

In order to get required data from the metering IC, we need to access following Register.

Register Address	Register Name	Read/ Write Type	Functional Description
			Energy Register
80H	APenergyT	R/C	Total Forward Active Energy
81H	APenergyA	R/C	Phase A Forward Active Energy
82H	APenergyB	R/C	Phase B Forward Active Energy
83H	APenergyC	R/C	Phase C Forward Active Energy
84H	ANenergyT	R/C	Total Reverse Active Energy
85H	ANenergyA	R/C	Phase A Reverse Active Energy
86H	ANenergyB	R/C	Phase B Reverse Active Energy
87H	ANenergyC	R/C	Phase C Reverse Active Energy
88H	RPenergyT	R/C	Total Forward Reactive Energy
89H	RPenergyA	R/C	Phase A Forward Reactive Energy
8AH	RPenergyB	R/C	Phase B Forward Reactive Energy
8BH	RPenergyC	R/C	Phase C Forward Reactive Energy
8CH	RNenergyT	R/C	Total Reverse Reactive Energy
8DH	RNenergyA	R/C	Phase A Reverse Reactive Energy
8EH	RNenergyB	R/C	Phase B Reverse Reactive Energy
8FH	RNenergyC	R/C	Phase C Reverse Reactive Energy
90H	SAenergyT	R/C	Total (Arithmetic Sum) Apparent Energy
91H	SenergyA	R/C	Phase A Apparent Energy
92H	SenergyB	R/C	Phase B Apparent Energy
93H	SenergyC	R/C	Phase C Apparent Energy
94H	SVenergyT	R/C	(Vector Sum) Total Apparent Energy
95H	EnStatus0	R	Metering Status 0
96H	EnStatus1	R	Metering Status 1
98H	SVmeanT	R	(Vector Sum) Total Apparent Power
99H	SVmeanTLSB	R	LSB of (Vector Sum) Total Apparent Power

Figure 63: Energy Registers

Register Address	Register Name	Read/ Write Type	Functional Description
			Power and Power Factor Registers
B0H	PmeanT	R	Total (all-phase-sum) Active Power
B1H	PmeanA	R	Phase A Active Power
B2H	PmeanB	R	Phase B Active Power
B3H	PmeanC	R	Phase C Active Power
B4H	QmeanT	R	Total (all-phase-sum) Reactive Power
B5H	QmeanA	R	Phase A Reactive Power
B6H	QmeanB	R	Phase B Reactive Power
B7H	QmeanC	R	Phase C Reactive Power
B8H	SAmeanT	R	Total (Arithmetic Sum) apparent power
B9H	SmeanA	R	phase A apparent power
BAH	SmeanB	R	phase B apparent power
BBH	SmeanC	R	phase C apparent power
BCH	PFmeanT	R	Total power factor
BDH	PFmeanA	R	phase A power factor
BEH	PFmeanB	R	phase B power factor
BFH	PFmeanC	R	phase C power factor

Figure 64: Power and PF Registers

Register Address	Register Name	Read/ Write Type	Functional Description
			THD+N, Frequency, Angle and Temperature Regis
F1H	THDNUA	R	phase A voltage THD+N
F2H	THDNUB	R	phase B voltage THD+N
F3H	THDNUC	R	phase C voltage THD+N
F5H	THDNIA	R	phase A current THD+N
F6H	THDNIB	R	phase B current THD+N
F7H	THDNIC	R	phase C current THD+N
F8H	Freq	R	Frequency
F9H	PAngleA	R	phase A mean phase angle
FAH	PAngleB	R	phase B mean phase angle
FBH	PAngleC	R	phase C mean phase angle
FCH	Temp	R	Measured temperature
FDH	UangleA	R	phase A voltage phase angle
FEH	UangleB	R	phase B voltage phase angle
FFH	UangleC	R	phase C voltage phase angle

Figure 65: Phase angle Registers

To calibrate and read data, we will be using SPI interface of the device. For the that IC have to work in slave mode. For that DMA-CTRL pin need to be HIGH. Then the calibration process can be stared by writing 5678H to the ConfigStart register.

9.7 ATM90EX GUI Software (24 - 31 March 2024)

For the ease of calibration and debugging, we searched for an alternative method to access IC's registers. Then we discovered that ATM90EX GUI, a software developed Atmel. It is capable of callibrating ATM90EXX series chips using GUI. Therefore we contacted Microchip through email and requested a copy of the software.

Hi [redacted]

You can find all the information in the webpage of the ATM90E36A or in the webpage of the evaluation kit (ATM90E36A-DB):

ATM90E36A

https://www.microchip.com/en-us/product/atm90e36a

ATM90E36A DEMO BOARD

https://www.microchip.com/en-us/development-tool/ATM90E36A-DB

The related software which image you attached is the Metrology Demo GUI and it is included in the ATM90E3x Demo Board Design Documentation package.

Regards,

Jorge

Required software and all the documentation can be found in ATM90E36A DEMO BOARD web page, under ATM90E3x Demo Board Design Documentation.
ATM90EX GUI has a register editor that can be use to edit register values for calibration. It also has other functions such as Measurement, Fourier Analysis, Meter Configuration, Temperature compensation etc.

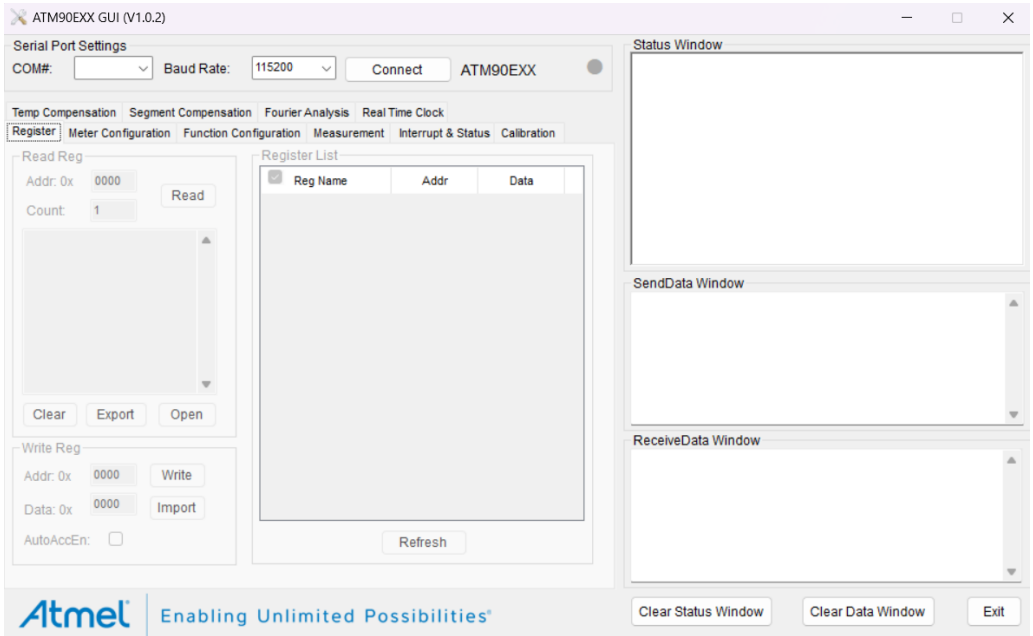


Figure 66: ATM90EX GUI

9.8 Conceptual Designs (24 - 31 March 2024)

Following the project requirements, we developed three conceptual designs for the industrial power monitoring system.

9.8.1 Design 1 - Centralized Design

This design represents a modular wireless sensor network consisting of a central unit and multiple distributed measuring devices. The central device collects data from the measuring devices, potentially performing initial processing and communication with a remote system. Measuring devices connect via a wired medium. The ability to connect multiple sensors allows for scalability, data redundancy, and diverse data collection.

- Block diagram

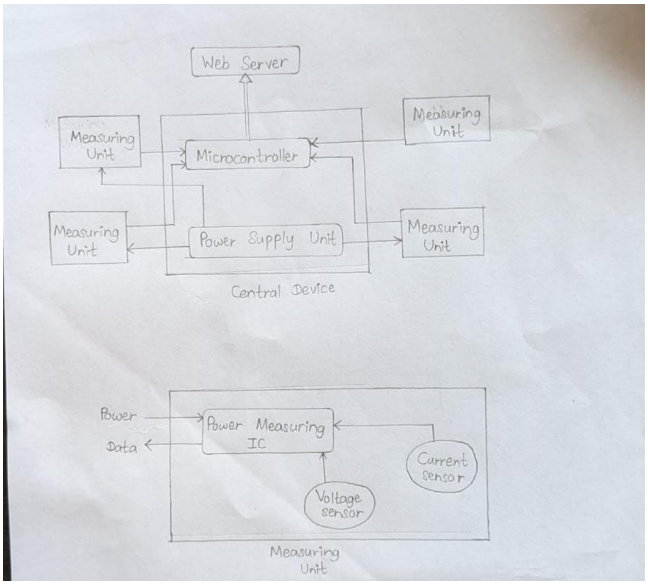


Figure 67: Design1 Block Diagram

- Enclosure design

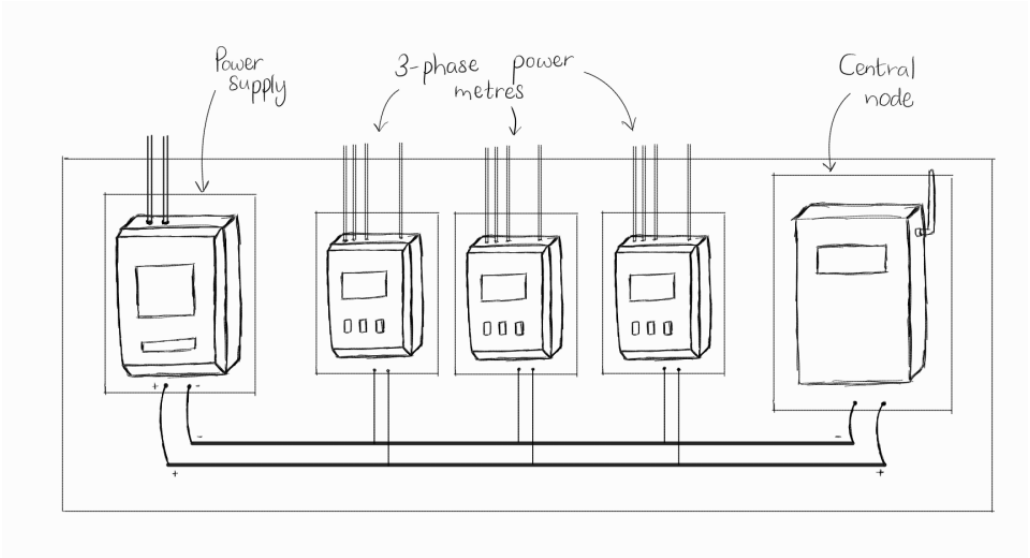


Figure 68: Design1 Enclosure Design

9.8.2 Design 2 - Standalone Device with External Sensors

In this proposed design, the incorporation of separate current and voltage sensors connected externally to the main device serves to mitigate the risk of high currents flowing through the measuring apparatus. By employing distinct current and voltage sensors, the system enables precise measurement of currents and voltages without subjecting the measuring device to potentially damaging levels of current. This segregation of measurement components enhances the safety and efficacy of the overall system, ensuring accurate readings while safeguarding the integrity of the measuring apparatus.

- Block diagram

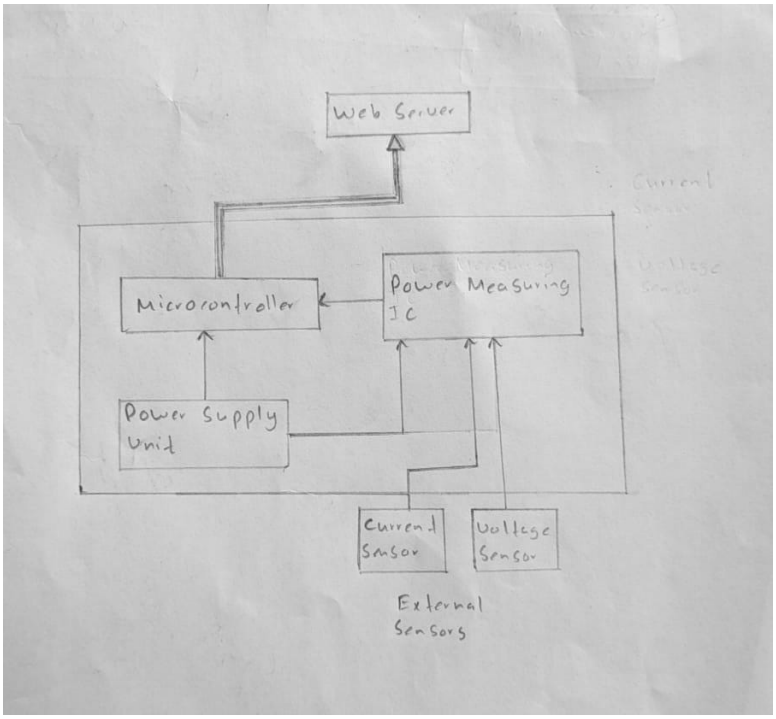


Figure 69: Design2 Block Diagram

- Enclosure design

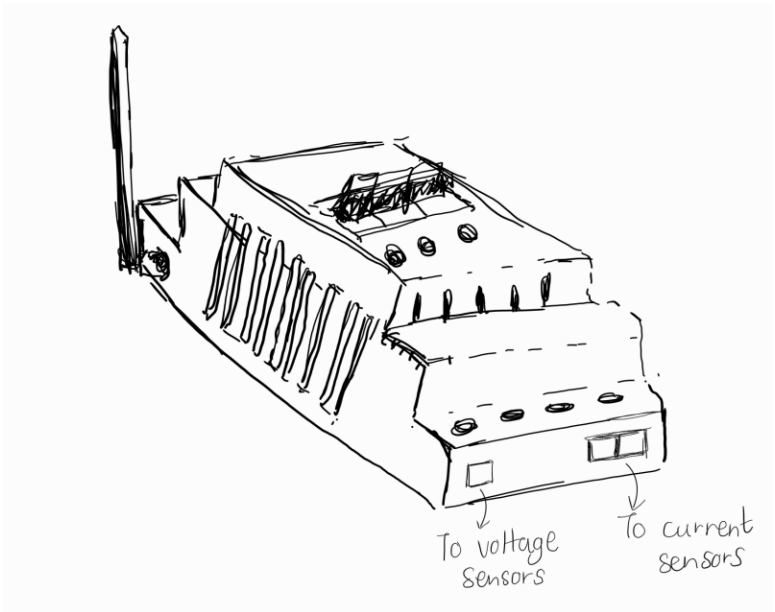


Figure 70: Design2 Enclosure Design

9.8.3 Design 3 - Standalone Device with Internal Sensors

This design entails the utilization of a singular, multifunctional device capable of connecting seamlessly to both a power source and a corresponding machine. This integrated approach streamlines usage and enhances safety protocols. Leveraging Wi-Fi connectivity, the device enables remote management and monitoring of the connected machinery, thereby offering flexibility and convenience in operation. However, in scenarios necessitating the simultaneous measurement of multiple devices, the deployment of additional instances of this device becomes imperative to ensure comprehensive data acquisition and analysis.

- Block diagram

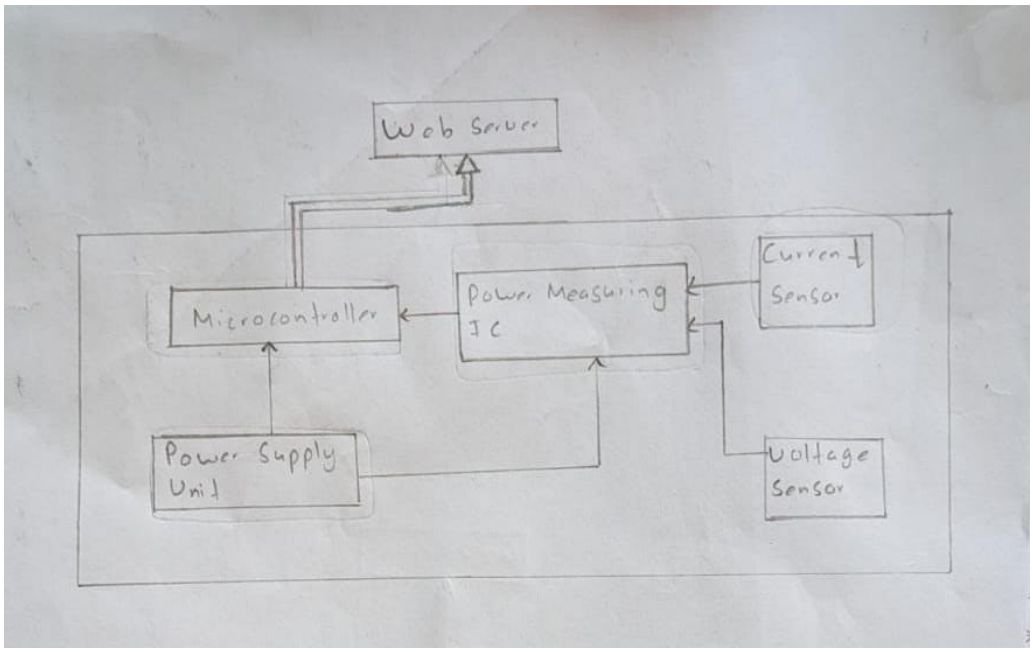


Figure 71: Design3 Block Diagram

- Enclosure design

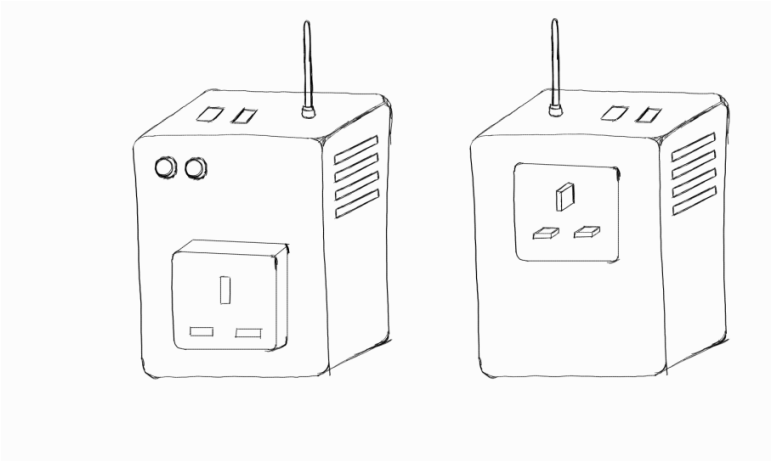


Figure 72: Design3 Enclosure Design

9.8.4 Design 4 - Standalone Device with External Sensors - With Ergonomic Enclosure

This design incorporates separate current and voltage sensors externally connected to the main device, effectively mitigating the risk of high currents flowing through the measuring apparatus. By employing distinct sensors, the system enables precise measurement of currents and voltages without exposing the measuring device to potentially damaging currents. Additionally, the design includes an ergonomic enclosure, robust enough for industrial environments, while also being easy to assemble and manufacture, ensuring both durability and user convenience.

- Block diagram

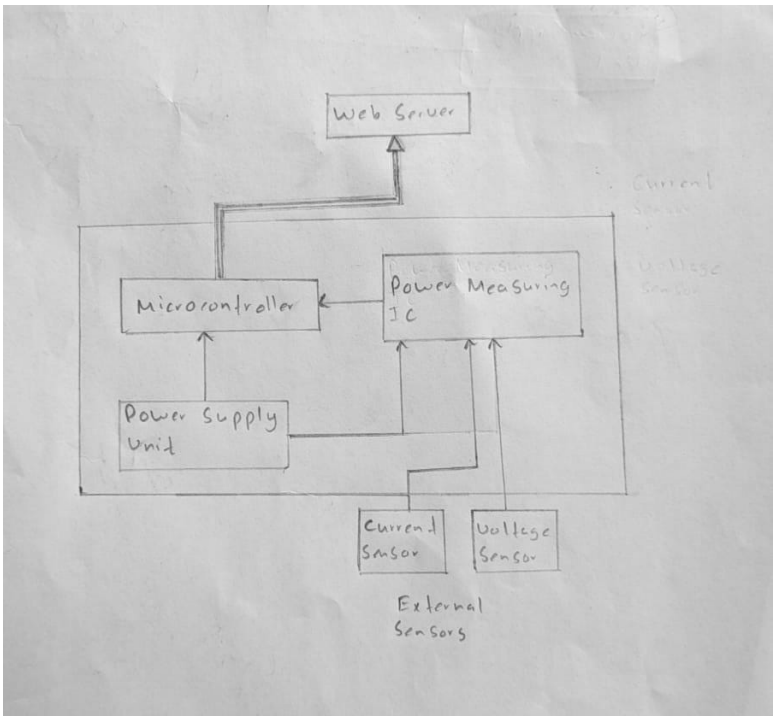


Figure 73: Design4 Block Diagram

- Enclosure design

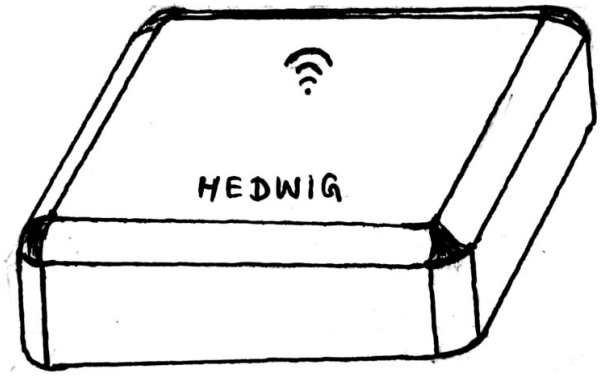


Figure 74: Design4 Enclosure Design

9.8.5 Evaluation of the Designs

1. Functional block criteria

    - Functionality – How the given design helps to achieve given functionalities.
    - Robustness – How suitable the enclosure is to an industrial environment.
    - Safety – How the enclosure incorporates safety measures.
    - Assembly – How much the enclosure is suitable for Manufacturing and assembly line.
    - Repairability- How easy to repair when it comes to a fault.
    - Ergonomics - How the design is designed for user comfort, ease of handling, and accessibility to ports, buttons, and other interactive elements.
    - Customization – How easy to customize the product to user preferences.

2. Enclosure Criteria

    - Functionality - How well each functional block contributes to the overall purpose and features of the product.
    - Manufacturing Feasibility - How is the complexity of manufacturing and compatibility with production methods
    - Cost -How much implementing each block cost, and associated expenses.
    - Performance – How well the given functional block performs.
    - Reliability – how reliable the functional block diagram is.
    - Power Efficiency – How much power does it use?
    - Repairability – How easy to diagnose a problem and repair it.

3. Selection:

    After thorough evaluation we selected the Standalone Device with External Sensors design (design 4) due to its well-balanced approach that prioritises functionality, robustness, user-friendliness, which allows for independent sensor placement, enhanced safety, easier repairability, and cost-effective customization, making it the most adaptable and practical solution for industrial power monitoring.

Score allocation

		Design 1 /10	Design 2 /10	Design 3 /10	Design 4 /10
Enclosure design criteria comparison	Functionality	8	8	8	9
	Robustness	9	9	9	10
	Safety	8	8	8	9
	Assembly	7	8	9	9
	Repairability	7	8	8	10
	Ergonomics	7	9	8	10
	Customization	7	8	9	8
Functional block design criteria comparison	Functionality	9	9	9	9
	Manufacturing feasibility	8	8	8	8
	Cost	7	9	7	9
	Performance	8	9	9	9
	Power Efficiency	7	8	8	8
	Repairability	8	10	7	10
Total		**100**	**111**	**107**	**118**

9.9 Schematics (1 - 6 April 2024)

Following a thorough analysis of the system requirements, the schematics were meticulously designed with deliberate component selection. A hierarchical schematic approach was employed, utilizing a main schematic with six sub-schematics dedicated to specific functionalities like the MCU, voltage and current sensing, and power management.

1. MCU: This sub-schematic focused on the microcontroller unit, responsible for overall system control, data processing, and communication.

2. Voltage In and Out: This sub-schematic handled the voltage input and output stages, ensuring proper power supply and distribution throughout the system.

3. Voltage Sensor Input: This sub-schematic detailed the circuitry for voltage sensors, responsible for accurate measurement of input voltages.

4. Current Sensor Input: This sub-schematic outlined the circuitry for current sensors, crucial for monitoring current flow within the system.

5. Metering IC: This sub-schematic focuses on the ATM90E36, a dedicated metering integrated circuit. It serves as the heart of the data acquisition system, processing sensor inputs (voltages and currents) from multiple phases. The ATM90E36 performs intricate calculations to derive various power parameters like power in each phase, energy consumption, Total Harmonic Distortion (THD), power factors, phase angles, and even temperature. These processed data points are then transmitted to the controller circuit for further processing and potential cloud transmission.

6. Power Supply Unit: This take AC power as input and efficiently converts it to stable DC voltage using Hi-Link module, providing the necessary electrical energy to operate the entire circuit. This pre-designed solution ensures consistent and reliable power delivery, crucial for accurate measurements and system stability.

9.10 Component Selection (1 - 6 April 2024)

Careful component selection, including the utilization of a pre-designed AC-DC power supply module Hi-Link, metering IC, antennas and other connectors, was used to ensure optimal performance and reliability within the industrial environment.

9.10.1 Protection Components

1. 3 Ampere fuse (30mm x 6mm) – Over current protection

2. MOV-10D561K Varistor – Gives over voltage protection for voltage surges over 560V

9.10.2 Power Components

1. Hi-Link HLK-20M05 220VAC to 5VDC 20W 220VAC to 5VDC 20W Step-Down Power Supply Module. This is cost effective and compact compared to making a SMPS circuit.

    - All voltage input (AC: 90 264V)
    - Low ripple and low noise
    - Output overload and short circuit protection
    - High efficiency, high power density
    - The product is designed to meet the requirements of EMC and Safety Test
    - Low power consumption, environmental protection

2. TPS75933KTTR - Voltage Regulators 3.3-V. This is a low Dropout Voltage Regulators which is able to convert voltage from 5 to 3.3V while maintaining required current.

9.10.3 Metering IC

This is a high-performance energy metering IC designed for three-phase power systems. It has 3 channels with measurement capabilities and Fourier analysis functions which can be used for power monitoring instruments. Also, it can provide accurate measurements of various parameters such as voltage, current, total harmonic distortion (THD), discrete Fourier transform (DFT), mean power, etc.

9.10.4 Connectors

1. Through hole PCB mount Fuse Holder - Open Fuse design allow to change the fuse easily.

2. PCB mount screw terminal blocks - Used to connect voltage reference wires.

3. PCB Mount Mono Socket - Used to connect current transformers which can connect and disconnect CT easily.

4. Male headers - Used for testing power, metering ic, serial interface and spi interface.

9.10.5 Antenna

1. 2118909-2 Antenna: Wi-Fi 6E L273 mm MHF (Cabled PCB) The 2118909-2 antenna was chosen due to its targeted features that enhance performance and integration within the industrial environment. Here's a breakdown of the key factors influencing this decision:

    - Industrial Design: Based on information from TE Connectivity, the 2118909-2 is a PCB antenna specifically designed for industrial applications. This suggests a rugged construction built to withstand dust, moisture, and extreme temperatures, ensuring reliable operation in harsh industrial settings. This characteristic is crucial to maintain consistent data transmission from the monitoring system.
    - Multi-Band Compatibility: While specific details are limited, the antenna is likely compatible with multiple frequency bands, offering flexibility for Wi-Fi connectivity. Common options for industrial applications include 2.4 GHz, 5 GHz, and potentially Bluetooth. This multi-band capability allows for adaptation to specific network requirements within the industrial environment.
    - Modular Design Consideration: If the 2118909-2 utilizes a modular design with an external antenna element connected to a PCB mount, it offers significant placement advantages. This configuration allows for positioning the antenna element within the enclosure for optimal signal reception, while the PCB mount remains easily accessible for seamless integration with the system's circuitry. This potential modularity simplifies installation and optimizes signal strength within the enclosure.

    The final antenna placement can be optimized during the enclosure design phase to ensure adequate space allocation.

    Overall, the 2118909-2 antenna effectively balances reliable data transmission ensuring reliable data transmission even in a harsh industrial environment. This makes it a suitable choice for the industrial power monitoring system.

2. RF Cable Assemblies (CSJ-RGFB-100-MHF3): RP-SMA female bulkhead to right angle IPEX MHF3 plug with 100mm of 0.81mm cable

9.11 PCB Designing (1 - 6 April 2024)

PCB design and layout were performed using Altium, adhered to established design rules, ensuring optimal functionality and signal integrity. Here's a breakdown of the key considerations:

The design rules for the PCB include a minimum trace width and spacing of 0.09mm, annular rings of 0.3mm for solder connections, and clearances of 0.5mm between traces, components, and vias to prevent short circuits and crosstalk. Specific clearances include 0.5mm for hole-to-hole and pad-to-pad (different nets), 0.254mm for via-to-track, and 0.2mm for pad-to-track. Via specifications are a 0.15mm diameter, 0.25mm drill size, and 1.0mm pad size. Signal routing utilized manual techniques for controlled impedance, prioritizing critical high-speed signals, with a 10 mil trace width and an 18 mil width with 8 mil separation for differential pairs. Power and ground planes were dedicated for efficient distribution and noise reduction, with stitching vias for connectivity. A four-layer stackup was chosen, consisting of two signal layers, a power plane, and a ground plane to balance signal complexity, power needs, and cost.

Components were strategically placed considering heat dissipation, signal routing paths, and ease of assembly. High-power components were positioned near the ground plane for optimal heat transfer.

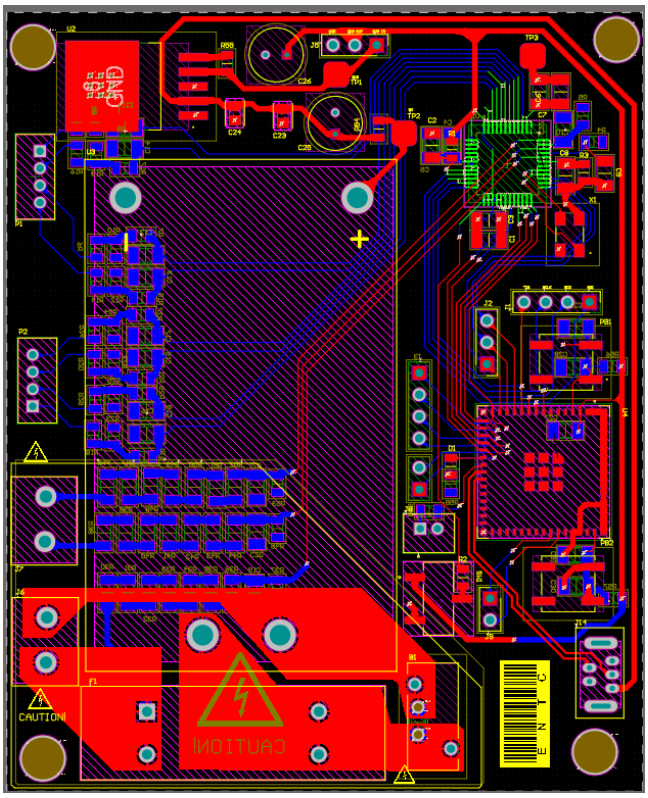


Figure 75: PCB initial

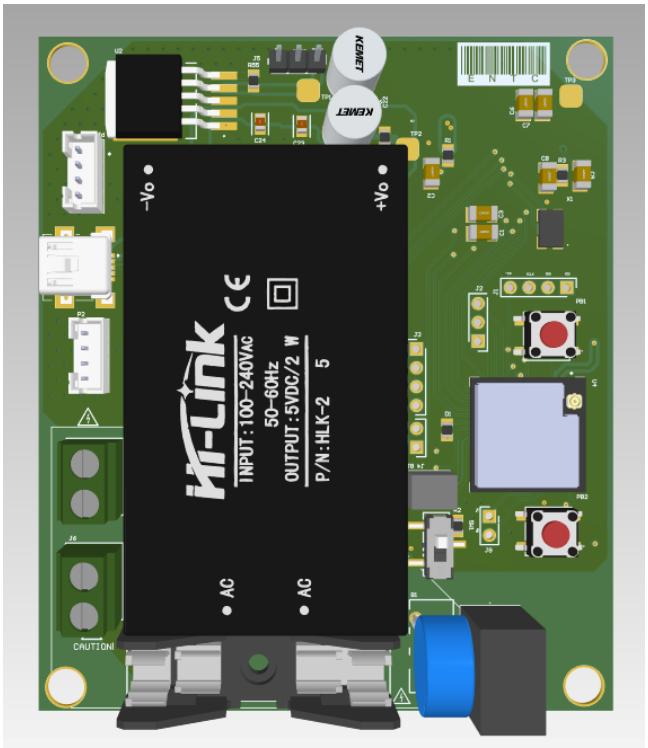


Figure 76: 3D view - Top

9.12 Finalizing the PCB and Placing the order (7 - 13 April 2024)

After further revisions, PCB design was improved. We added stitching vias for better connectivity of ground layers And we isolated different sections of the PCB using zero /*ohm* resistors. Following the finalization of the PCB design, component procurement commenced. Surface-mount device (SMD) components and other main components were ordered from Mouser Electronics, a reputable supplier known for its extensive inventory and reliable delivery. The rest of the components were procured from Tronics, catering to the remaining component needs.

Furthermore, five PCB units were ordered from JLCPCB, a cost-effective and efficient PCB manufacturer. This ensured timely acquisition of the necessary components and circuit boards for prototype development.

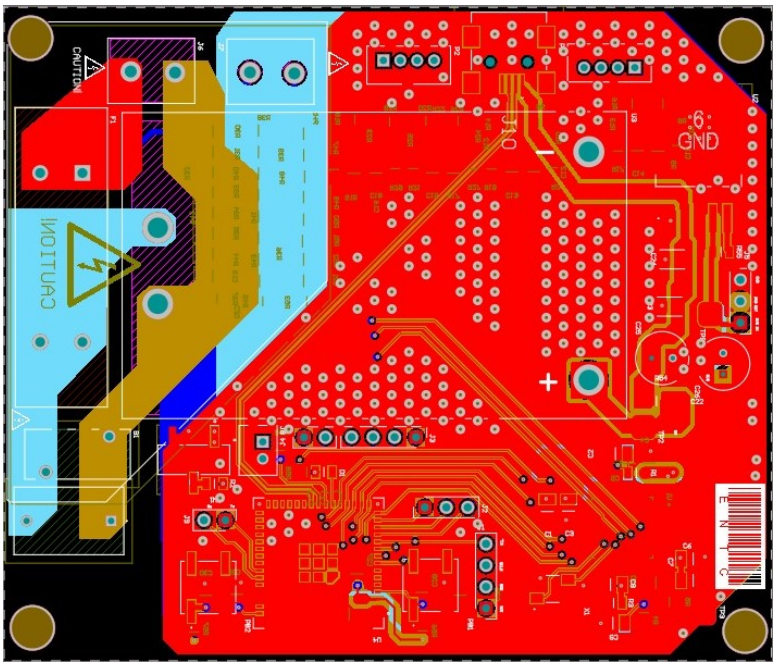


Figure 77: Revised PCB Fig1

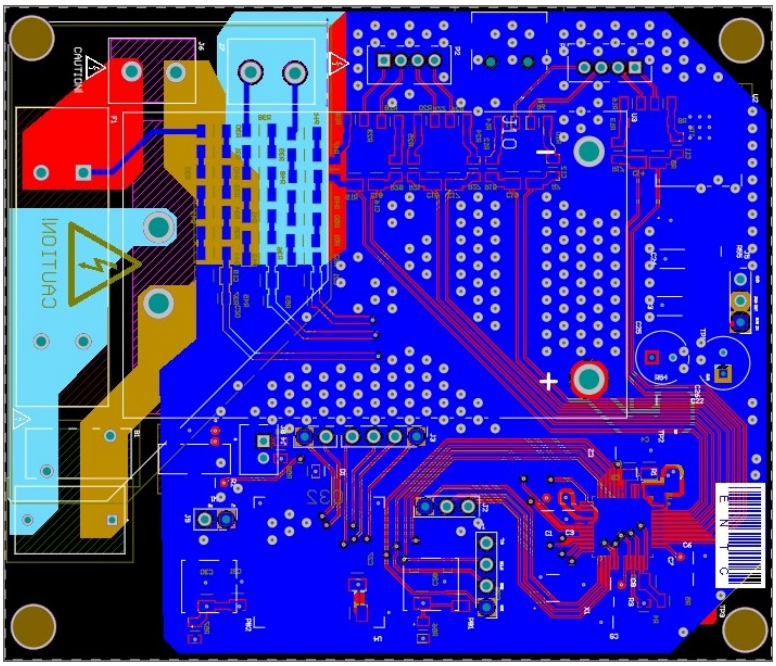


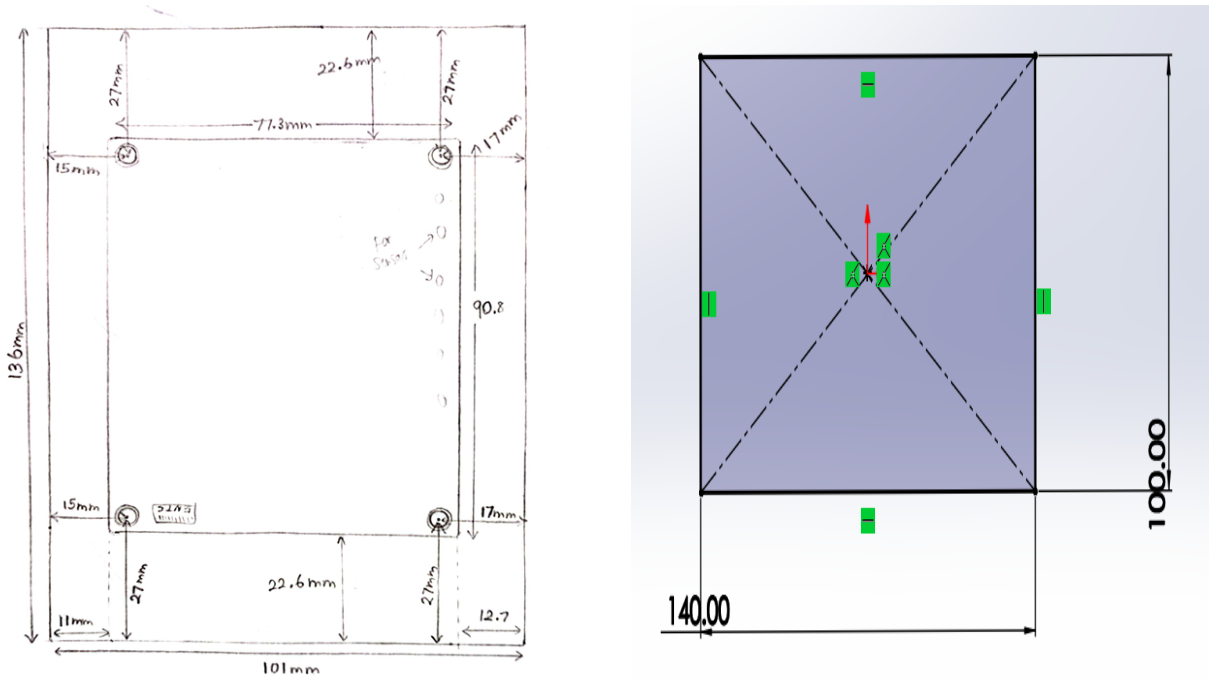
Figure 78: Revised PCB Fig2

9.13 SolidWorks Design (14 - 20 April 2024)

The industrial enclosure for the power monitoring system was designed using SolidWorks, 3D modeling software while adhering to industrial design principles.

Initial Approach:

Considering the size of the PCB and and other components we finalized the dimensions and positions of the parts of the enclosure. Initially, the design involved creating two separate parts: the main body of the box and the lid. While functional, this approach lacked efficiency for future modifications.



(a) Hand sketched Dimensions

(b) Dimensions of the enclosure

Figure 79: Initial sketches

Optimized Design Strategy:

Following guidance from our lecturer, a more efficient and industrial-standard design strategy was adopted. This involved creating a single enclosure with a partition incorporated into the lid itself. This approach offered several advantages.

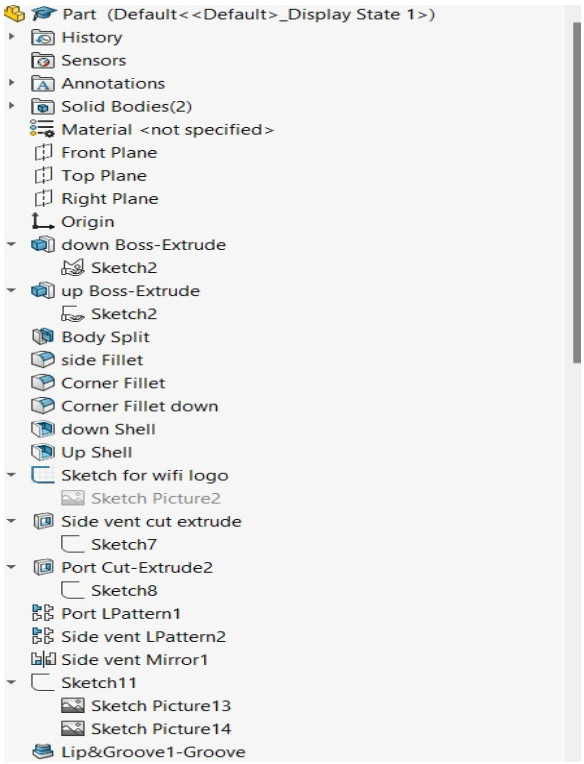
1. Design Efficiency: Any changes made to the lid automatically update the partition within the box, significantly reducing design time and effort for future modifications.

2. Improved Design Integrity: The single-part design with an internal partition ensures a more robust and seamless enclosure, potentially enhancing its structural integrity.

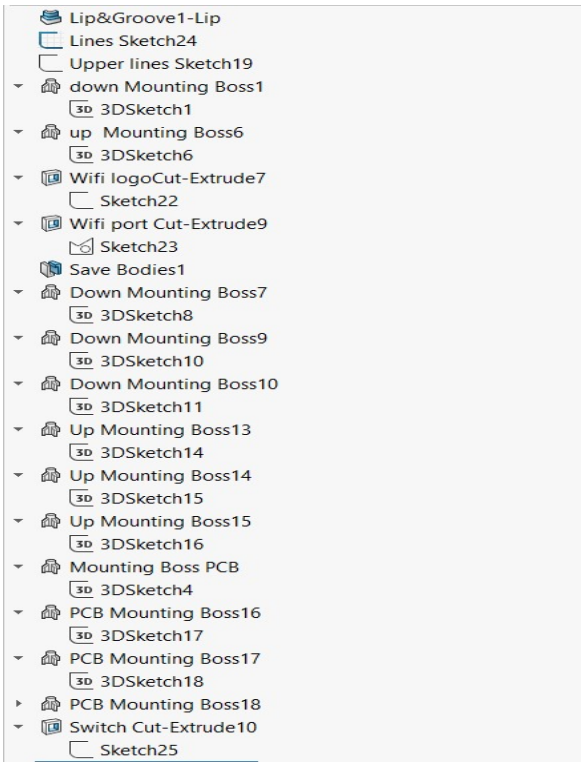
The enclosure design incorporated features crucial for the system's operation and user interaction:

- **Standard Design Parameters and Measurements:** The enclosure design adhered to established design parameters and standard measurements, ensuring compatibility with readily available components and manufacturing processes. Standard bolt and screw sizes readily available in the market were chosen to simplify assembly and maintenance processes.

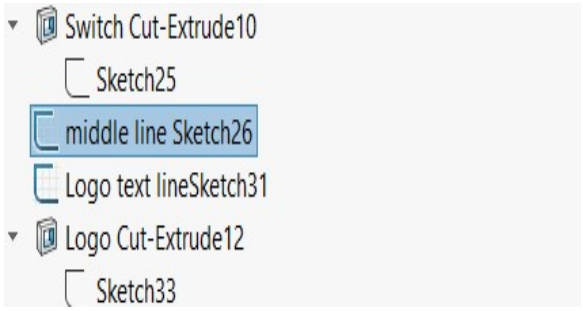
- **Draft Angles and Fillets:** Draft angles and fillets were incorporated into the design to facilitate plastic molding and prevent mold release issues.



(a)



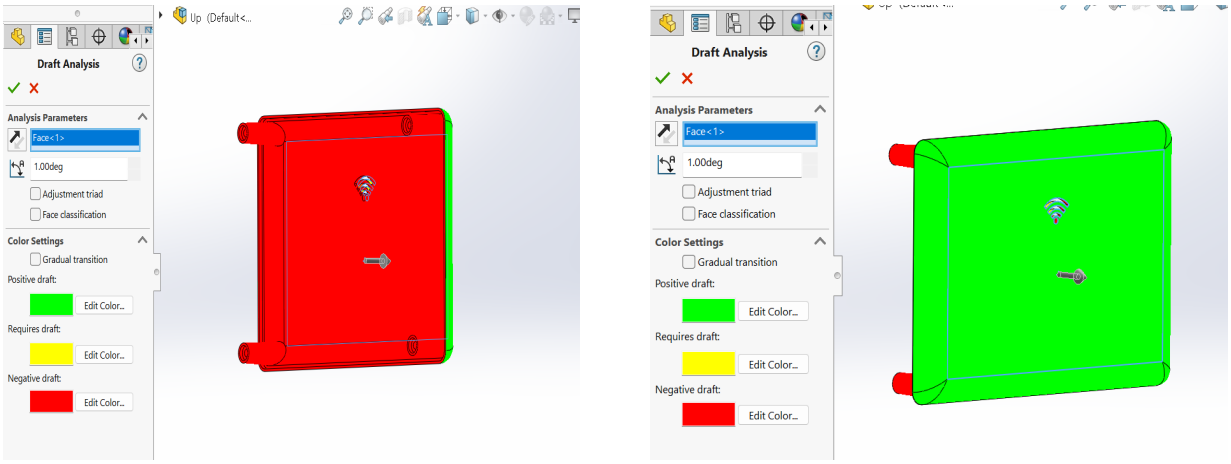
(b)



(c)

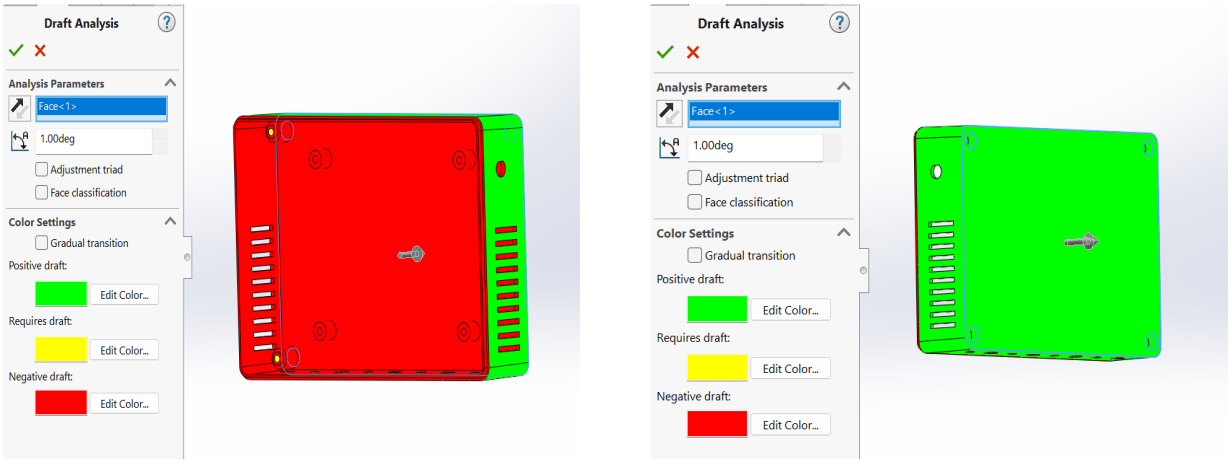
Figure 80: Model Tree

- **Mounting Points:** 4 mounting bosses were placed each for securing the lid to the box and for mounting the PCB. A major challenge involved finding the optimal placement for the mounting bosses due to the slightly asymmetrical dimensions of the PCB. Careful analysis and adjustments were necessary to ensure proper alignment and stability.
- **Lid Attachment:** The lid was designed to securely attach to the box body using lip and groove feature, protecting the internal components from environmental factors while allowing for easy access during maintenance.
- **Ventilation Openings:** Strategically positioned ventilation openings facilitated airflow, preventing excessive heat buildup within the enclosure and ensuring optimal operating temperature for the PCB components.
- **Sensor Access:** Dedicated holes were incorporated into the sides of the enclosure to accommodate the antenna, voltage sensors, and current sensors, providing access for external connections and measurements.
- **Material Selection:** The enclosure material is yet to be selected after a thorough search for available printers. We will consider factors like durability, environmental resistance, and cost-effectiveness.



(a) Draft Analysis Lid 1 (b) Draft Analysis Lid 2

Figure 81: Draft analysis of Lid



(a) Draft Analysis Box 1 (b) Draft Analysis Box 2

Figure 82: Draft analysis of Box

Figure 83: Enclosure

9.14 Library Customization and Serial Peripheral Interface (SPI) Testing (21 - 27 April 2024)

9.14.1 Adapt Chosen Libraries

We incorporated an open source project that was under developemnt. Therefor we did a throuigh code review and necessary changes for the Library.

1. **Modifications for the Exsisting Library**

   To tailor the microcontroller libraries for our specific power monitoring needs, we identified several necessary modifications. We incorporated additional functionalities to support the Atmel M90E36A power measuring IC and ensured seamless integration with the ESP32 microcontroller. This involved:

2. **Development Considerations**

   During the development and customization of the libraries, we adhered to strict coding standards to ensure code quality and maintainability. Key considerations included:

   - Coding Standards:
     We followed industry best practices for coding, including proper documentation, modular design, and consistent naming conventions. This approach facilitated easier debugging and future enhancements.
   - Testing Approaches:
     We employed unit testing to verify individual functions and integration testing to ensure smooth interaction between the customized libraries and other system components. Rigorous testing helped identify and resolve issues early in the development process.

   We did not have the completed PCB with us when developing the firmware. Therefor we had to use some alternative methods to Test the firmware in development phase.

3. **Test Plan for SPI Communication**

   To verify the reliability of SPI communication, we developed a comprehensive test plan that included:

   - Data Transmission/Reception Verification:
     We conducted tests to ensure accurate data transmission and reception between the micro-controller and the power measuring IC. This involved sending known data patterns and verifying the received data for correctness.
   - Signal Integrity Analysis:
     We used an oscilloscope to monitor the SPI signals, ensuring that the clock, MOSI, MISO, and CS lines operated as expected.

By adapting the chosen libraries to meet our specific needs and rigorously testing the SPI communication, we ensured that our Industrial Power Monitoring System operates reliably and accurately, providing valuable real-time data for power quality analysis and operational efficiency.

9.15 Firmware Development (21 - 27 April 2024)

9.15.1 Git Repository Setup

To manage our firmware development efficiently, we established a private Git repository. This repository served as a centralized platform for version control, allowing us to track changes, collaborate effectively, and maintain a history of modifications. Utilizing Git ensured that our development process was organized and that we could easily revert to previous versions if needed.

9.15.2 Library Integration and Customization

For the project's SPI communication, we leveraged an existing library to expedite development. However, several customizations were necessary to align the library with our specific requirements:

- **SPI Mode Support:**

  We modified the library to support any SPI Mode, optimizing data transfer for our application. This adjustment was crucial for ensuring reliable communication with our hardware components. We used SPI Mode 3 in this project

- **Custom I/O Pins:**

  To enhance hardware flexibility, we adapted the library to allow the use of custom I/O pins. This adaptation was essential for integrating our device with various sensors and peripherals.

- **SPI Pin Configuration:**

9.15.3 Initial Testing with Alternative MCU

Due to the unavailability of the target PCB during initial stages, we conducted preliminary testing using a different MCU. This approach allowed us to validate the core functionalities of our firmware and ensure that the basic operations were working correctly before transitioning to the final hardware.

9.15.4 WiFi Integration

To enable IoT capabilities and remote data access, we integrated WiFi connectivity into the firmware. This integration involved:

  – **Firmware Testing with WiFi:**
  Thorough testing was conducted to ensure that the WiFi functionality was stable and reliable. We validated the connection to local networks, data transmission, and handling of network interruptions.

  – **IoT Capabilities:**
  WiFi integration allowed our device to transmit data to remote servers, enabling real-time monitoring and analysis.

9.15.5 MQTT Publishing Capability

To facilitate data transmission to remote servers, we implemented MQTT publishing capability within the firmware. This feature enabled our device to send power monitoring data to a central system, providing stakeholders with continuous access to critical information.

9.15.6 Heartbeat Mechanism

To ensure continuous functionality and monitor the operational status of the device, we added a heartbeat mechanism to the firmware. This mechanism regularly sent status signals to broker, confirming that the device was functioning correctly and alerting stakeholders to any potential issues.

We used numerous other libraries excluding ATM90E36.h library

- SPI.h - For SPI Commiuncation with ATM90E36

- WiFi.h - For WiFi Connectivity
- PubSubClient.h - For MQTT Communication

9.16 Dashboard Development (28 April - 4 May 2024)

We first Considered Frameworks and platforms to build the dashboard including

- React.js
- Vue.js
- Node-red
- Blynk
- OpenHAB We choose to go with Node-red after extensive research because it is highly customizable and modular. Then we considered options to host Node-red. Some exsaples given below
- IBM Cloud
- Heroku
- Balena
- GCP
- AWS
- Raspberry Pi
- Microsoft Azure
- Digital Ocean We decided to buy a Droplet from Digital ocean and use it to host the Service. We also bought a Microsoft Azure instance.

  We also got the public IPv4 for Digital Ocean droplet

  We decided to get a subdomain from a domain(mihiran.com) that one of our team member owned. We intially did the developments with power.mihiran.com

  We also incorparated Cloudfare CDN network and their services to incease the real-abitly,reachabilty and the security of the dash-board related services.

  The dashboard for the hedwig.mihiran.com monitoring system will be developed using the Node-RED platform to leverage its capabilities for real-time data visualization and interaction. It was built with the following key-features;

9.16.1 Key Features

1. Real-time Measurements: The dashboard will display critical power system parameters in real-time, including:
   - Power (total and per phase)
   - Energy consumption (total and per phase)
   - Power factor (per phase)
   - Total Harmonic Distortion (THD)
   - Phase angle
   - Temperature

2. User-Friendly Interface: The design will prioritize user-friendliness and aesthetics. The interface will be intuitive and easy to navigate, allowing users to quickly access and understand the displayed information.
3. Data Visualization Techniques: Data will be presented through a combination of visually appealing and informative elements:
   - Graphs: Line graphs will be used to depict trends in power consumption, temperature, and other parameters over time.
   - Line Charts: Real-time variations in power (per phase), THD, and phase angle will be effectively visualized using live-updating line charts.
   - Gauges: Gauges will be employed to provide a clear and concise representation of key measurements like power factor and temperature.

9.16.2 Implementation with Node-RED

The dashboard involves a combination of the following nodes:

1. MQTT nodes: These nodes will subscribe to relevant MQTT topics for receiving real-time sensor data.
2. Data processing nodes: Depending on the data format, processing nodes might be required to manipulate and prepare data for visualization.
3. Chart nodes: Node-RED offers various chart nodes for creating line graphs, bar charts, and other visual representations.
4. UI nodes: A range of UI nodes will be utilized to build the user interface, including elements for displaying gauges, text information, and buttons for historical data exploration.

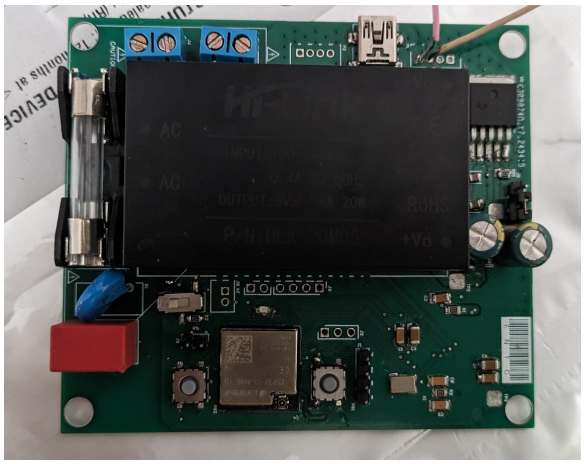
By adding these Node-RED's capabilities and focusing on user-friendly design principles, we could make this dashboard more comprehensive and visually appealing for visualizing and analyzing real-time and historical power system data.

9.17 PCB Assembly and Functional Testing (5 - 11 May 2024)

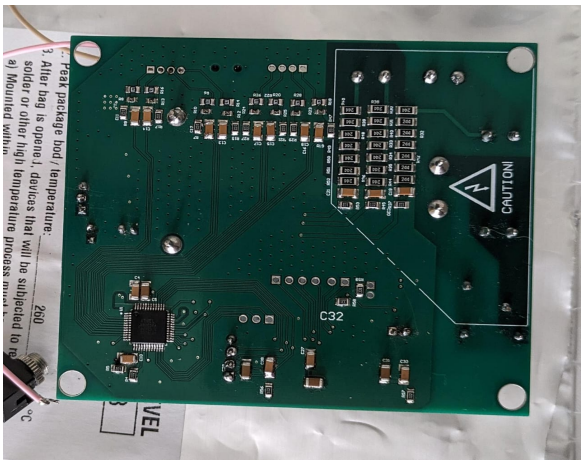
The PCBs and electronic components were received this week. So, we could start soldering the components. In preparation for soldering, ESD-safe tweezers were used to handle the delicate components, particularly the ATM90E36A IC. To ensure the safety of the ICs during soldering, a combination of a hot air gun and soldering iron was employed, maintaining a controlled temperature of 280°C at the joints.

Following the soldering process, the assembled PCB was inspected for any potential defects like excess solder causing short circuits, weak solder joints, or missing or misaligned components.

Testing the assembled PCB followed a systematic approach. First, the power circuit was isolated by disconnecting it from the main circuit. Then, 5V was directly supplied to the LDO (Low Dropout Regulator) to verify its voltage conversion capability from 5V to 3.3V. The high-link transformer, responsible for converting 230V AC to 5V DC, underwent a separate functionality test. Finally, the power circuit was reintegrated with the main circuit by soldering a 0-ohm resistor, effectively completing the connection. This comprehensive hardware assembly and testing process ensured the final product was of high quality and functioned as intended.



(a) Fig1



(b) Fig2

9.18 Establishing System Connectivity (5 - 18 May 2024)

9.18.1 Initial Functionality Testing

To ensure the basic functionality of our microcontroller unit (MCU), we began with a simple test using a blink code and the onboard LED. This initial step verified that the MCU was operational and correctly programmed, serving as a fundamental validation before proceeding with more complex integrations.

9.18.2 WiFi Connectivity

The next step involved verifying the WiFi connectivity of our MCU. We configured the MCU to connect to a local WiFi network and tested its ability to maintain a stable connection. This step was crucial for enabling the Internet of Things (IoT) capabilities of our system, allowing for remote data access and monitoring.

9.18.3 SPI Interface Testing

To ensure proper communication between the MCU and the power measuring IC via the SPI interface, we used another MCU development kit. By testing the SPI communication with this setup, we validated the configuration settings, such as clock speed and data mode, ensuring reliable data transfer between the devices.

9.18.4 Metering IC Testing

For the metering IC, we conducted specific tests by checking the sysstatus and meterstatus registers. These tests confirmed that the IC was correctly initialized and functioning, capable of accurately measuring and reporting power parameters. Monitoring these status registers provided assurance that the IC was correctly integrated and responsive to commands from the MCU.

9.18.5 Hardware Configuration for Data Acquisition

We configured the hardware for data acquisition by establishing secure connections between the sensors and the MCU. This involved careful wiring and configuration to ensure accurate and reliable data capture. Key considerations included proper grounding, shielding of cables to minimize interference, and ensuring robust connections to avoid data loss.

9.18.6 Testing Procedures for Data Transmission/Reception

To verify the integrity of data transmission and reception, we implemented a comprehensive testing plan:

- Data Transmission Verification:
  We sent known data patterns from the sensors to the MCU and checked the received data for accuracy. This step confirmed that the Metering IC is working correctly transmitting data and that the MCU was receiving it without errors.
- Reception Testing:
  We conducted similar tests in reverse, sending data from the MCU to another system (e.g., a server or another MCU) to verify that the data was received correctly and could be processed.
- End-to-End Testing:
  We performed end-to-end tests, simulating real-world conditions to ensure that the entire data acquisition and transmission process worked seamlessly. This included capturing data from sensors, processing it through the MCU, transmitting it via WiFi, and finally verifying it on a remote server.

9.19 Finalizing the Enclosure and 3D Printing (26 May - 1 June 2024)

We employed the Fused Deposition Modeling (FDM) technique using Polylactic Acid (PLA) filament for 3D printing the enclosure. PLA was chosen for its balance of durability, ease of printing, and environmental friendliness. The enclosure design was finalized to accommodate all system components, ensuring proper fit and alignment for seamless integration.

Figure 85: Final Enclosure

9.20 Test Code

```
1
2 // This code is provided solely for testing purposes.
3 // It is intended to be used as a demonstration or for experimental purposes
     only.
4 // Please refrain from using this code in any production or application
    environment as it may not meet the necessary standards
5 // or requirements for such use. * The code may contain errors,
    inefficiencies, or other issues that could lead to unexpected behavior
6 // or performance.
7 #include <SPI.h>
8 #include "ATM90E36.h"
9 #include <WiFi.h> //IoT capabilites
10 #include <PubSubClient.h>
11
12 //Change here for differencet Wifi
13 // WiFi
14 const char *ssid = "Wi Likes Fi"; // Enter your Wi-Fi name
15 const char *password = "mihiran1";  // Enter Wi-Fi password
16
17 // MQTT Broker
18 const char *mqtt_broker = "137.184.9.146";
19 const char *mqtt_username = "mihiran";
20 const char *mqtt_password = "mihiran";
21 const int mqtt_port = 1883;
22
23 const char *topic = "Energy_moniter/moniter_1/phase_3";
24 const char *phase_A_Power = "Energy_moniter/moniter_1/phase_A/Power";
25
26
27 //MQTT Topics
28
29 //Device Heart beat. Keep to inform weather the device is connected and
    working to Dashboard
30 const char *heart_beat = "Energy_moniter/moniter_1/heart_beat";
31
32 //voltage topics
33 const char *phase_A_Volatge = "Energy_moniter/moniter_1/phase_A/Voltge";
34 const char *phase_B_Volatge = "Energy_moniter/moniter_1/phase_B/Voltge";
35 const char *phase_C_Volatge = "Energy_moniter/moniter_1/phase_C/Voltge";
36
37 //Current Topics
38 const char *phase_A_Current = "Energy_moniter/moniter_1/phase_A/Current";
39 const char *phase_B_Current = "Energy_moniter/moniter_1/phase_B/Current";
40 const char *phase_C_Current = "Energy_moniter/moniter_1/phase_C/Current";
41
42
43 //Enabling Wifi
44 WiFiClient espClient;
45 PubSubClient client(espClient);
46
47 //SPI
48 // Define MISO, MOSI, and SCK pins
49 const int misoPin = 6;
50 const int mosiPin = 5;
51 const int sckPin = 4;
52 const int ss = 7;
53
54 //Making the eic object - this should go below. check
55 ATM90E36 eic(7);
56
57 void setup() {
58   pinMode(17,OUTPUT);
59   pinMode(18,OUTPUT);
60   digitalWrite(17,HIGH);
61   digitalWrite(18,HIGH);
```

```
  Serial.begin(115200);

  // Start SPI with specified settings
  SPI.begin(sckPin, misoPin, mosiPin, ss);

  // Set SPI settings
  SPISettings settings(200000, MSBFIRST, SPI_MODE0);
  SPI.beginTransaction(settings);


  /* Initialize the serial port to host */
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }


  Serial.println("Start ATM90E36");
  /*Initialise the ATM90E36 + SPI port */
  eic.begin();
  delay(1000);
}

void callback(char *topic, byte *payload, unsigned int length) {
  String word;
 for (int i = 0; i < length; i++) {
        word += ((char) payload[i]);
    }
    Serial.println(word);
}


void loop() {
  digitalWrite(17,HIGH);
  digitalWrite(18,HIGH);
   client.publish(topic, "Working");

  /*Repeatedly fetch some values from the ATM90E36 */
  double voltageA,freq,voltageB,voltageC,currentA,currentB,currentC,power,pf
    ,new_current,new_power;
  int sys0=eic.GetSysStatus0();
  int sys1=eic.GetSysStatus1();
  int en0=eic.GetMeterStatus0();
  int en1=eic.GetMeterStatus1();

  String message = "S0:0x" + String(sys0, HEX);

  //client.publish(topic, message.c_str());
  Serial.println("S0:0x"+String(sys0,HEX));
  delay(10);
  Serial.println("S1:0x"+String(sys1,HEX));
  message = "S1:0x"+String(sys1,HEX);
  //client.publish(topic, message.c_str());
  delay(10);
  Serial.println("E0:0x"+String(en0,HEX));
   message = "E0:0x"+String(en0,HEX);
  //client.publish(topic, message.c_str());
  delay(10);
  Serial.println("E1:0x"+String(en1,HEX));
  message = "E1:0x"+String(en1,HEX);
  //client.publish(topic, message.c_str());

  //VolatgeA RMS
  voltageA=eic.GetLineVoltageA();
  Serial.println("VA:"+String(voltageA)+"V");
  message =  "VA:"+String(voltageA)+"V";
  //client.publish(phase_A_Volatge,message.c_str());
```

```
128
129   //VoltageB RMS
130   voltageB=eic.GetLineVoltageB();
131   Serial.println("VB:"+String(voltageB)+"V");
132   message =  "VB:"+String(voltageB)+"V";
133   //client.publish(phase_B_Volatge,message.c_str());
134
135   //VoltageC Rms
136   voltageC=eic.GetLineVoltageC();
137   Serial.println("VC:"+String(voltageC)+"V");
138   message =  "VC:"+String(voltageC)+"V";
139   //client.publish(phase_C_Volatge,message.c_str());
140
141   //Currents
142   currentA = eic.GetLineCurrentA();
143   Serial.println("IA:"+String(currentA)+"A");
144   message =  String(currentA);
145   //client.publish(phase_A_Current,message.c_str());
146
147   currentB = eic.GetLineCurrentB();
148   Serial.println("IB:"+String(currentB)+"A");
149   message =  String(currentB);
150   //client.publish(phase_B_Current,message.c_str());
151
152   currentC = eic.GetLineCurrentC();
153   Serial.println("IC:"+String(currentC)+"A");
154   message =  String(currentC);
155   //client.publish(phase_C_Current,message.c_str());
156
157   digitalWrite(17,LOW);
158   digitalWrite(18,LOW);
159   delay(1000);
160 }
```

Checked by,

Sayuru Amugoda Kaveendra Alwis