THM | CAMPUS FRIEDBERG | IEM
TECHNISCHE HOCHSCHULE MITTELHESSEN
Informationstechnik-Elektrotechnik-Mechatronik

Interdisciplinary IoT Project

Power Consumption Analysis

Axit Kakadiya	5514902
Brieuc Pesnel	5512474
Devraj Solanki	5514890
Mary Sutharshini Thevathas	5463211
Robin Dietzel	5277715

18-01-2023

Department of Informationstechnik, Elektrotechnik und Mechatronik (IEM)
Campus Friedberg

Specialization: None/interdisciplinary

Supervisor: Prof. Dr.-Ing. Fabian Mink

Statement of Authorship

I hereby declare that I have written this report independently and exclusively using the sources and aids provided. Contents of this report that were taken verbatim or analogously from other sources are marked as such. The report or parts thereof have not been submitted in this or a comparable form to any other examination body and have not been published. *The sections of this report are marked with the initials of its responsible group member.*

Friedberg, den 18.01.2024			
Place, Date		Axit Kakadiya [AK]	Signature
Friedberg, den 18.01.2024			
Place, Date		Brieuc Pesnel [BP]	Signature
Friedberg, den 18.01.2024			
Place, Date		Devraj Solanki [DS]	Signature
Friedberg, den 18.01.2024			
Place, Date		Mary Sutharshini Thevethas [MS]	Signature
Büdingen, den 18.01.2024			
Place, Date		Robin Dietzel [RD]	Signature

Contents

Table of Acronyms

1 Introduction [MS]

Power consumption analysis has playing an important role in the technological advancement world. It's being adapted by various sectors to utilize the energy consumptions and aiming to have an understanding of the energy usage patterns. This report implements a methodology to visualize the parameters influencing power usage, which are retrieved from the power analyzer.

This report aims to analyze the power consumption patterns in the University building in real-time based. Data will be collected through several modules and monitoring systems, allowing for a comprehensive visualization of power consumption trends.

The Power measurements data such as voltages, current, Phase angles are retrieved from the DEIF MIC-2 MKII multi-instrument by using Modbus communication protocol and RS485 serial communication facilities. Then the data continuously injected to the cloud platform with the help of ESP32 microcontroller device. With the help of MQ Telemetry Transport (MQTT) protocol, the injected data in the cloud subscribed and visualized in Grafana in real time basis.

2 Objective of the Project [MS]

- Retrieve the power usage information from DEIF MIC-2 MKII module
- Publish the data to the cloud
- Subscribe the published data
- Graphical visualization of the data in Grafana

3 Basics

3.1 RS-485 [AK] [1]

RS-485 was created by the Telecommunications Industry Association and Electronic Industries Alliance (TIA/EIA) to overcome the disadvantages of the RS-232 serial communication device. It is used in two-wire data transfer. Via RS-485 serial communication protocol, the master can communicate up to 32 devices via a wire bus connection as shown in the diagram. In the OSI communication model, it lies on the first layer at the physical layer. It creates the electrical specifications for transmitting and receiving data.

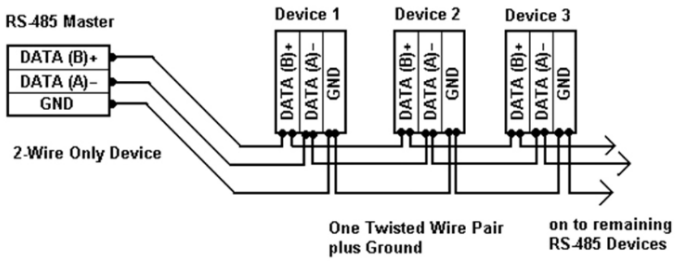


Figure 3.1: RS485 Master-Slave Connection [1]

The signal transmission of RS-485 is mentioned in the below image. Data transfer will start from High to low pulse (Mark) and then followed by 8 bits of data and then as per the configuration, it will use odd parity or even parity. At last low to high pulse (Space) for ending the data transmission.

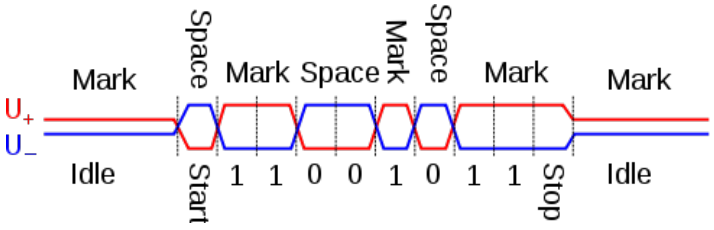


Figure 3.2: Signal Characteristic of Data Transmission [2]

3.1.1 UART [3]

A Universal Asynchronous Receiver-Transmitter (UART) is a protocol for asynchronous serial communication in which the data format and transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel.
The electric signaling levels are handled by a driver circuit external to the UART. Common signal levels are RS-232, RS-485, and raw TTL for short debugging links.In the OSI commu-

nication model, it lies on the second layer data link layer. Communication is possible in all three modes.

1. Simplex (in one direction only, with no provision for the receiving device to send information back to the transmitting device)
2. Full Duplex (both devices send and receive at the same time)
3. Half-Duplex (devices take turns transmitting and receiving)

UART frame, field length in Bits [hide]

1	5-9	0-1	1-2
Start Bit	Data Frame	Parity Bits	Stop Bits

Figure 3.3: Data Frame Structure for Transmission [3]

Details regarding the UART Frame structure:

1. Idle (logic high (1))
2. Start bit (logic low (0)): The start bit signals to the receiver that a new character is coming.
3. Data bits: The next five to nine bits, depending on the code set employed, represent the character.
4. Parity bit: The parity bit is a way for the receiving UART to tell if any data has changed during transmission.
5. Stop (logic high (1)): They signal to the receiver that the character is complete.

3.1.2 UART to RS-485 Conversion

In situations when noise immunity or long-distance communication is necessary, UART and RS-485 are frequently used in tandem. In contrast to conventional UART communication, a physical layer that facilitates communication over greater distances can be implemented using the RS-485 hardware (transceivers). In industrial applications and situations where devices must communicate over longer distances or in loud surroundings, the combination of UART and RS-485 is frequently utilized.In this project, we are using the hardware module (UART-TTL to RS485 Converter Module) which is used to convert the UART TTL signal to the RS-485 signal.

3.2 Modbus data-link layer [BP]

The following section of this report describes how the Modbus data-link layer works. Modbus works on the data-link layer to trade frames between two devices, in the previous part we focused on the electrical part here we will focus on the concept on frames exchange and on frames.

3.2.1 The protocol

Layer	ISO/OSI Model	
7	Application	Modbus Application Protocol
6	Presentation	Not used
5	Session	Not used
4	Transport	Not used
3	Network	Not used
2	Data Link	Modbus Serial Line Protocol
1	Physical	EIA/TIA-485 standard

Figure 3.4: Layer where Modbus works [4]

The Modbus protocol works on the layer 1, 2 and 7, so there is no interaction between the frame which is sent by the device and how the frame is received on the application, that's why it's important to know how this protocol works.

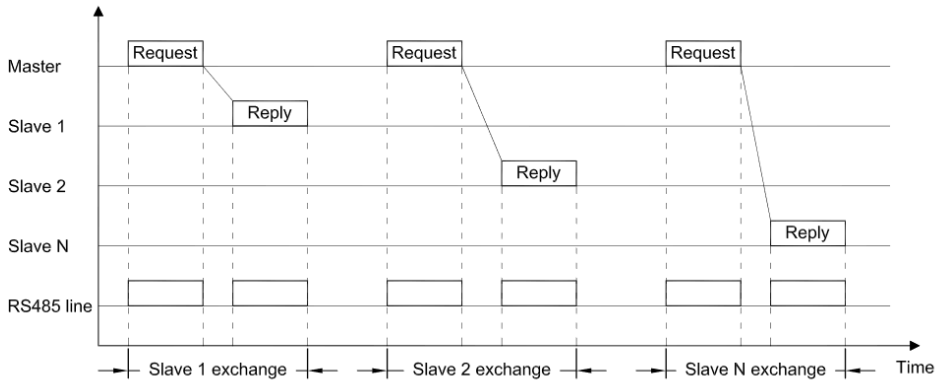
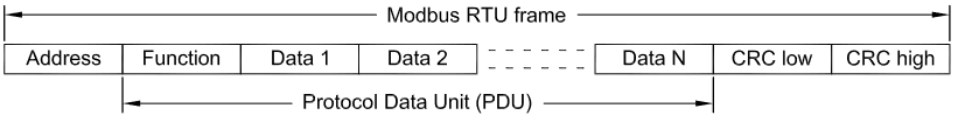


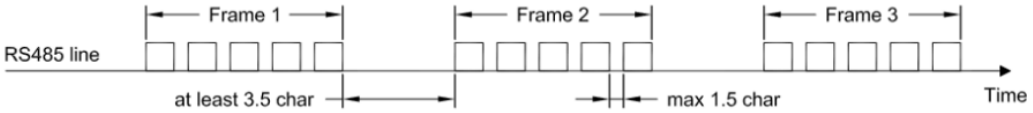
Figure 3.5: Master slave communication [4]

Modbus is based on a master slave communication, the device is consider as the master and send requests to the sensor which is consider as a slave and answer to the request. Each sensor can send informations to multiple device and device can ask different sensors.

3.2.2 The frame



(a) The frame in bytes



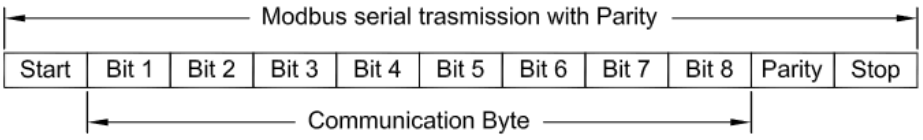
(b) The frame time

Figure 3.6: Typical frame [4]

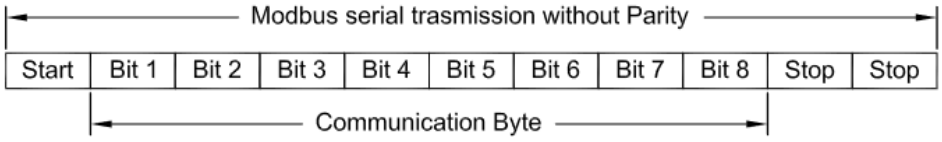
Each frame consists of one byte of addressing to connect to the sensor, one byte of protocol type, as many bytes as necessary for data transfer and two bytes of error control that are important to ensure the integration of communication. Frames are sent in regular time intervals to ensure the integrity of the link (if a delay appears is that a frame is missing). Traditionally, sensor addresses are [4]:

- 0 broadcast for slave
- 1 – 247 unicast
- 248 – 255 impossible (reserved for the manufacturer)

In details, there are two types of bytes, one with a parity cock and one without, this does not change much because it is just replaced by a stop bit.



(a) A byte with parity



(b) A byte without parity

Figure 3.7: The byte in details [4]

3.3 ESP32 [DS]

The ESP32 microcontroller family, developed by Espressif System, is very famous for providing best features which are perfectly suited wild range of IoT projects and application. There are many different microcontroller are offered by Espressif, However for this project, ESP32 family's Microcontroller are well suited as its key features are a dual-core Xtensa LX6 processor, integrated Wi-Fi and Bluetooth capabilities, a robust set of peripherals, and scalable memory options. Here, we've chosen the ESP 32-WROOM 32E microcontroller, embedded in the ESP32 Olimex board, from the ESP32 family.

3.3.1 ESP32 WROOM 32 E

Since we wish to access our data via a MQTT broker, a variety of connectivity options are important considerations when selecting a device. Here, we've chosen the ESP 32-WROOM 32E micro controller(Depicted on the left side) on the from the ESP32 family. This has an Ethernet connector, WiFi, and Bluetooth chip built right in.We decided on this ESP32-WROOM 32E rather than an Arduino since an Arduino requires the inclusion of wifi and bluetooth modules. It also has more integrated components, such as an antenna and flash memory, and is quite affordable. These points all satisfy our requirements and are most appropriate for our project.

Figure 3.8: ESP32 WROOM 32E processor [5]

3.3.2 Dual Core- Processor

The module is chipped with dual-core Xtensa LX6 processor, this configuration gives ESP32 to work on different task at a same time with correct output on two processing units. Dual core feature provide upgraded overall performance and multitasking capability. It has clock frequency range from 80 MHz to 240 MHz. This clock frequency is editable by the operator.

3.3.3 Pin Configuration

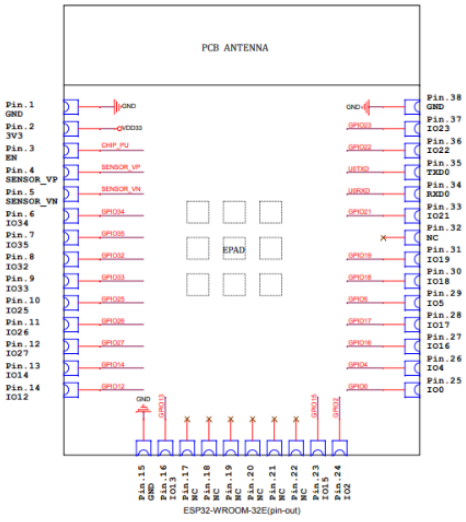


Figure 3.9: Pin configuration [5]

This module has a diverse pin configuration. Its has 40 GPIO(General Purpose Input Output) pins(e.g., GPIO04, GPIO12), which gives versatile Analog and Digital communication feature. Furthermore, it also has UART pins (TX0,RX0), by which the serial communication (e.g., RS485 modbus) become very easy. It also have ADC pins, PWM pins, boot and Reset pins, I2C pins and SPI pins.

3.3.4 Wireless Connectivity

This processor has inbuilt Wi-Fi(802.11b/g/n), which can operate as station mode and access point mode. Also, it support WiFi communication security with WPA/WPA2. During transmission of data, it can do encryption in order to ensure data security. With WiFi it also have inbuilt Bluetooth support.

3.3.5 Memory Configuration

The ESP32 WROOM 32E is devided into two different memory type, SRAM & Flash memory. SRAM(Static Random-Access Memory) it used for storing temporary data, high speed and inorder to improve overall performance of module. On the other hand, Flash Memory is Non-volatile type of memory, where the real program code, firmware and data are stored.

3.4 ESPHome Introduction [AK] [6]

Modern life is now completely dependent on smart home automation, which provides ease, improved security, and energy efficiency. Within the domain of open-source smart home systems, ESPHome is a platform designed to simplify and optimize do-it-yourself home automation projects. ESPHome, which was first developed as a component of Home Assistant, is now well-known for its user-friendly interface and seamless device connectivity. The ESP-32 device could be flashed using a variety of methods, including Platform-IO, ESP-Home, Arduino-IDE, etc. Consider your application and the problem you wish to address before choosing which platform to use.

Figure 3.10: ESPHome [7]

3.4.1 Advantages of using ESP-Home

1. ESPHome is intended to integrate easily with the ESP8266 and ESP32 microcontrollers, which are popular among makers because of their low cost, low power consumption, and integrated Wi-Fi.
2. ESPHome uses YAML configuration files, which is one of its unique characteristics. They offer a great level of flexibility and customization by defining devices, automation rules, and integrations.
3. ESPHome is compatible with several protocols for communication, such as MQTT and API-based exchanges. This allows devices and platforms from various smart homes to work together.
4. By enabling users to configure sensors, actuators, and other components using an easy-to-understand configuration language, ESPHome streamlines the device management process.
5. ESPHome easily interfaces with the well-liked open-source home automation platform as a part of the Home Assistant ecosystem. Through the provision of a single interface for controlling all linked devices, this integration improves the entire smart home experience.

3.4.2 Getting Started with ESPHome

If you want to create custom firmware for your ESP8266/ESP32 devices, ESPHome is also the possible answer. We'll walk over how to quickly and easily set up a basic "node". The ESPHome may be used in a variety of ways. Some of the services are as follows:

1. From Home Assistant
2. Using the command line
3. Migrating from Tasmota
4. Using the Docker Container

Some of the prerequisites you need to have before getting started with this method:

1. You should be familiar with the Docker container and how it is working.
2. Portainer should be running which is used as a container management tool.
3. You should be familiarized with the YAML file of the configuration.

In this project, we are using the Method using the Docker container. If you want to start with the ESP-Home via Docker container then you need to follow these steps.

3.4.3 Installation of ESP-Home Via Command Line

You need to follow the below step for installing the ESP-Home if you are using the method via the command line.

1. Open a Docker Container App
2. Create an image of the Portioner container management tool we used in this project. Follow this reference for more detail Source: https://docs.portainer.io/start/install-ce/server/docker/linux
3. Open a portainer container by typing on the browser (preferred Google Chrome). Command: https://mt-labor.iem.thm.de:9443
4. Create a stack using the YAML File of ESP-Home as follows and upload in the portioner.

```yaml
services:
  esphome:
  image: esphome/esphome
  container_name: esphome
  restart: unless-stopped
  ports:
    - "20000:6052" # Web dashboard
  networks:
    - dev-net
  volumes:
    - team4-esphome_team4_esphome_conf:/config
  environment:
    USERNAME: "admin"
    PASSWORD: "iotlab2023team4"
    ESPHOME_DASHBOARD_USE_PING: true

volumes:
  team4-esphome_team4_esphome_conf:

networks:
  dev-net:
```

5. Open the ESP-Home dashboard by typing below in the web browser(preferred Google Chrome).
   Command: `https://mt-labor.iem.thm.de:20000`
6. After opening the site, you will see the below screen as a dashboard. This is the initial setup of the ESP-Home.

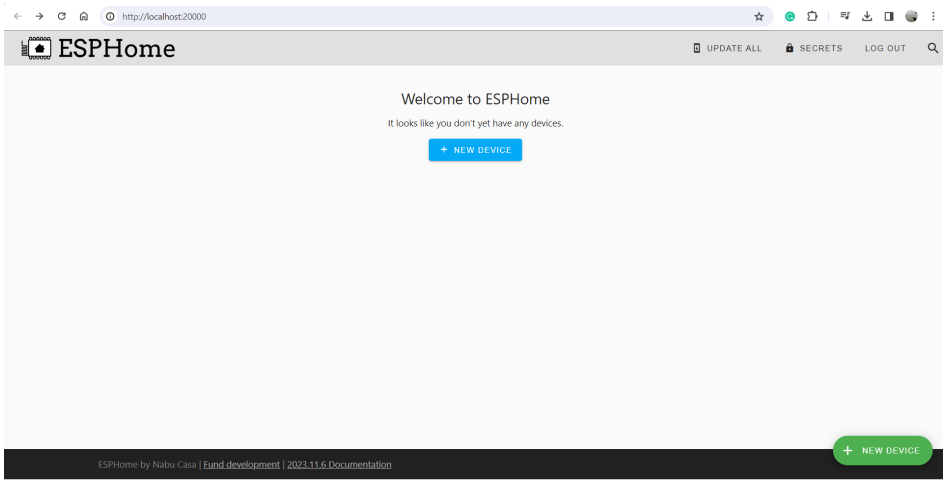


Figure 3.11: ESPHome Dashboard

Please refer to the port detail in the GitLab for more details regarding the configured port.

3.5 MQTT [MS]

Message Queuing Telemetry Transport (MQTT) is open-source, lightweight, and publish-subscribe messaging protocol. Primarily designed for the communication among the devices those contain the resource.

Publish-Subscribe-Architecture
It uses publish-subscribe model, which allows the devices to act as publishers or subscribers.

Broker-Based Communication
MQTT communication relies on a broker that acts as a mediator between publishers and subscribers. The broker efficiently manages message routing to ensure messages reach the intended recipients.

Topics
Communication in MQTT uses topics, hierarchical strings used to categorize messages. Subscribers express interest in specific topics, and publishers send messages to these topics.

Quality of Service (QoS) Levels
MQTT provides configurable levels of reliability (QoS Levels 0, 1, 2)

Security Features
Transport Layer Security (TLS) encryption, can be implemented to secure MQTT communication.

Retained messages
the broker stores a last known good value for new subscribers

Persistent sessions
the broker stores messages on behalf of a subscriber that is temporarily offline

Heartbeats
if the broker does not receive a PINGREQ for some time, it will close the connection and send the Last Will and Testament (LWT)

Last Will Testament (LWT)
A predefined message sent to all subscribers by the broker in case a device stops responding.

3.6 DEIF MIC-2 MKII [MS]

DEIF MIC-2 MKII is a multi-instrument used to monitor and control the operations. The unit continuously updates metering results and allows users online access to monitor. It has the capability of logging from single low voltage to multiple high voltage parameters. All metering data and setting parameters can be accessed by using the front panel keys or with the communication port. Setting parameters are stored in the EEPROM.

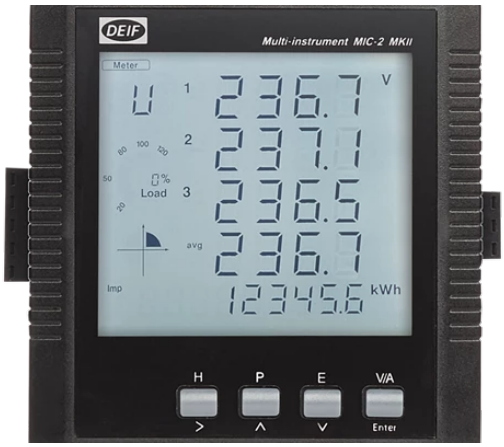


Figure 3.12: DEIF MIC 2 MKII [8]

- Multiple connected devices- Uses RS-485 Modbus communication (Up to 32 devices can be connected on a RS485 bus)
- Customized alarm settings- facilitate to customize up to 16 different parameters
- Password protected setting- Counter reset and change of settings can be password-protected
- Display- Large LCD screen with white backlight
- Control and Monitoring- It provides a user interface for operators to monitor the parameters and control its operation.
- Communication- the instrument support various communication modules - Ethernet (Modbus TCP, HTTP, SMTP), Profibus DP
- Remote Access- Remote monitoring and control capabilities allow users to access and manage the generator set from a distance.
- Inputs and Outputs- Optional Input and Output modules - Relay , Analogue I/O, Digital I/O
- Data Logging-It may include data logging features, allowing operators to analyze historical performance data and identify trends.
- Customizable Configurations- Users may have the ability to configure the controller to meet specific application requirements, adapting it to different measurements and operational scenarios.

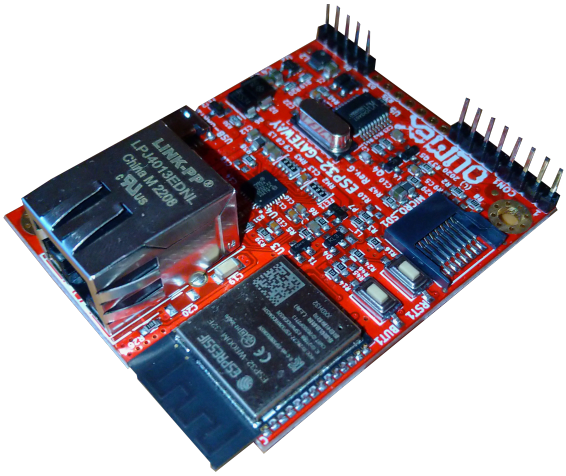
4 Hardware Suggestion [RD]

The following section of this report describes the hardware and software suggestion for interfacing with the `MIC-2 MKII's` modbus interface. The hardware/software stack should be able to read out several registers of the device via Modbus. Then the data should be sent via MQTT to a broker within the THM network for further processing and visualization.

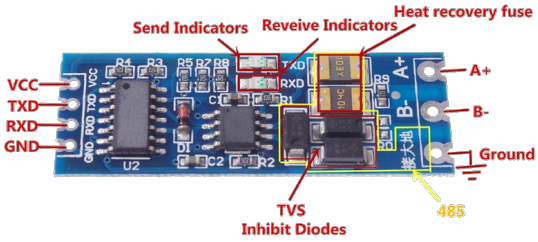
4.1 Development Board

As the microcontroller development-board was picked the `OLIMEX ESP32-GATEWAY` board. Olimex is a provider of development boards, embedded device programmers and other development tools [9]. The board itself is depicted within fig. 4.1 on the left side. The following list gives a brief overview about the features of this development board.

- ESP32-WROOM32 module
- CH340B USB-UART converter for programming
- Ethernet 100MB interface (LANA710A phy chip)
- MicroSD card slot
- 20-Pin GPIO connector with all ESP32 ports



(a) OLIMEX ESP32-GATEWAY



(b) RS485 module [10]

Figure 4.1: Selected hardware for connection with the MIC-2 MKII

The board seems to be well suited for this application because it's quite cheap with an average price of 16.59€ (in 01-11-2023) and has an onboard Ethernet transceiver and port. The Ethernet port was one of the main criteria when picking the board because wireless communication form inside the wiring closet, where the device should be placed, might not be a good choice for reliable communication. The onboard CH340B chip brings the advantage to easily

flash the board without an external Universal Serial Bus (USB) to Universal Asynchronous Receiver Transceiver (UART) converter. This gives more flexibility when developing the firmware for the board. The onboard SD-card slot is not really necessary in the context of this work.
At least there is one other advantage to mention here: The board has open-source hardware, so the schematics can be freely browsed on the internet.

4.2 RS485 interface

The picked RS485 interface is depicted in fig. 4.1 on the right hand. The module then connects to the development board with 3.3 V supply voltage and ground. The `TXD` and `RXD` pins connect the UART interface between the board and the converter. On the right edge of the converter module you can see the `A+` and `B-` pins which later will be connected to the Modbus device. They communicate over RS485 levels. Unfortunately the chip on the converter was milled of so that it was impossible to look up the Integrated Curcuit (IC) used on those types of boards. An internet researched had resulted in a common use of the MAX485 or MAX3485 IC.

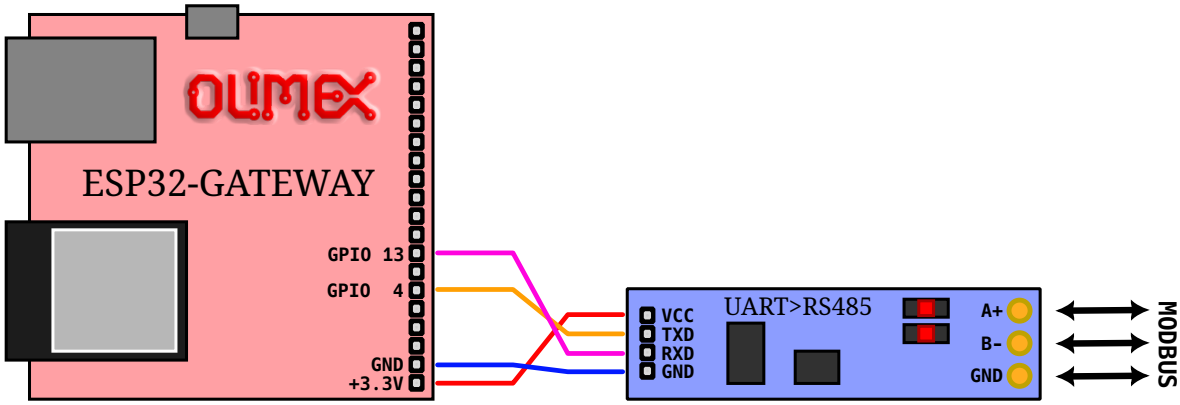


Figure 4.2: Schematic of connection between ESP32-GATEWAY board and the RS485 converter

The precedent fig. 4.2 outlines the wired connections between the UART to RS485 converter used in this project with the ESP32-GATEWAY board. For mounting and installing those two connected boards in the wiring cabinet it was additionally necessary to design an enclosure. This will be further described in section 4.4. The GPIO-Pins 4 and 13 were chosen because they aren't occupied by specific hardware functions already present on the development board (e.g. SD-Card slot). One additional challenge was not to use pin 12 because it's a „strapping“ pin. Those pins must be left floating or pulled to ground when flashing the device for the flashing procedure to work properly.

4.3 Modification of RS485 transceiver

Unfortunately the first real hardware connection of the transceiver board described in section 4.2 to the DEIF instrument did **not** work properly. The console of the software on the microcontroller logged the reception of „duplicate Modbus commands“. This issue was caused by a bad RX-TX-switching design on the board.

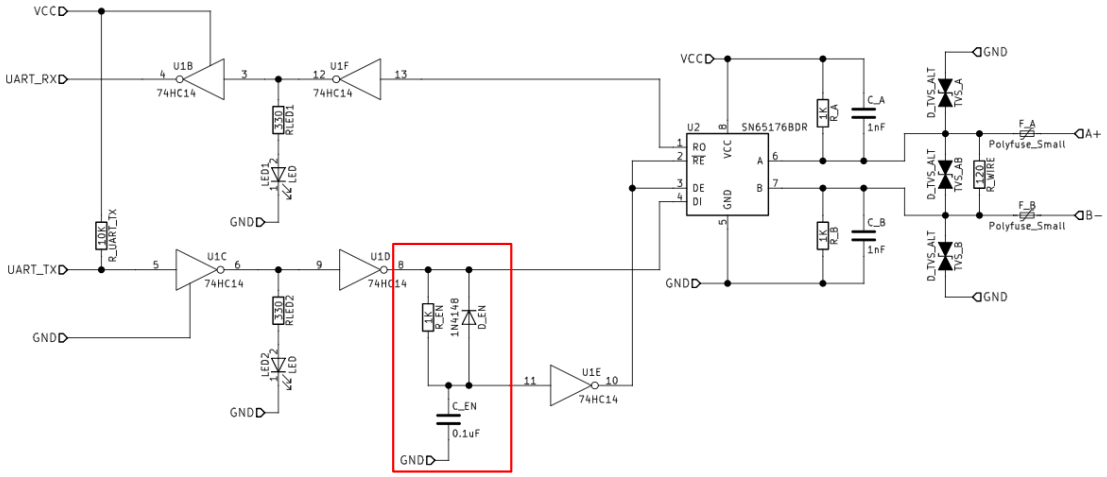


Figure 4.3: RS-485 module schematic [10]

When looking at fig. 4.3 it becomes clear what actually happened. Unfortunately it was not possible to verify if the depicted schematic exactly matches to the board used. The two inputs on the transceiver chip on the right hand of the picture (labeled `RE` and `DE`) are responsible for setting the chip into transmitting or receiving mode. This is actually necessary because RS-485 works only in half-duplex mode. So when `UART_TX` is pulled to LOW level for transmitting data `DE` becomes HIGH and the chip switches to transmitting mode. To keep the chip within this mode, even when some bits are transferred that forces `UART_TX` to HIGH, an RC-circuit is formed in the box marked red to keep the transceiver within transmitting mode for some time.

Now the real issue was that the time-constant of this RC-circuit was not suitable for the baud-rate used with the DEIF instrument. The transceiver chip switched back to receiving mode just before the whole sending command had finished. So it received back (parts) of its own command and displayed the error message mentioned before.It was not possible to

determine the exact configuration of the RC-circuit on this board, but when using the values from the schematic in fig. 4.3 we come to a time constant of:

$$\tau = C \cdot R = 0.1\mu\text{F} \cdot 1\text{k}\Omega = 100\mu\text{s} \tag{4.1}$$

When calculating how long one byte needs to be transferred (in between the chip must stay in transmit mode) with the baud-rate used by the DEIF instrument, we find the mismatch:

$$\Delta t = \frac{8\text{Bit}}{19200\text{Bit}\,\text{s}^{-1}} \approx 417\mu\text{s} \tag{4.2}$$

Those two equations prove the issue. When not pulling `UART_TX` to LOW level often enough the capacitor `C_EN` discharges to quickly and allows the transceiver chip to go to receiving mode. The time constant τ should at least be greater than the transmission duration of one byte, since all data bits of one transferred byte and the parity could be LOW bits but at least the start bit of the next byte will always be HIGH. The second constraint is that it must be smaller than the time the DEIF instrument will reply on the bus, because if it keeps the chip too long in transmitting mode an answer could not be received on the bus. Thus, the capacitor and the resistor on the board were replaced trough 220kΩ and 10nF that form an RC circuit with $\tau = 2.2$ms and the bus communication worked pretty well.

4.4 Case-design

For installing the device a kind of case should be designed to avoid short-circuits or other damage to the hardware when placing it somewhere. Therefore, a very simple base-plate was designed and 3D printed in order to protect the bottom of the ESP32-GATEWAY's board.

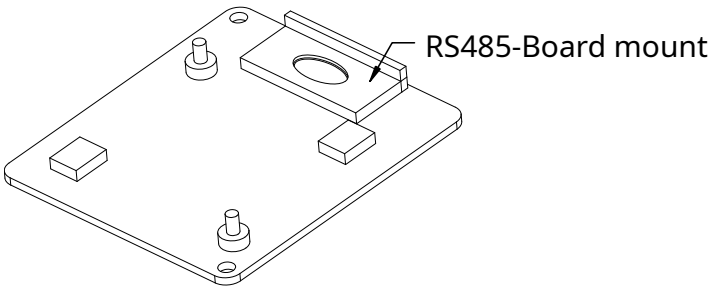


Figure 4.4: ESP32-GATEWAY + RS485 Converter Board mount

Figure 4.4 shows an exported 3D drawing created with the CAD-Application FreeCAD. The real mount was printed on a Prusa Mini+ 3D printer with PLA[1] filament. After mounting the board the bolts were molten with a soldering iron to hold the board in-place. The RS485 Board was glued onto the mount.

[1] Polylactic acid, thermoplastic polyester made of fermented corn or sugarcane which has good characteristics for 3D printing.

5 Software Suggestion [RD]

This section contains a basic explanation which software/firmware was chosen for operating the ESP32-GATEWAY board described in section 4. Thus this section contains the basic configuration used to do Over The Air (OTA) updates and configuration provisioning.
For building and installing the ESPHome firmware (see section 3.4), an ESPHome server must have been set up in a containerized environment. This procedure is described in section 6.2. The following listing contains the basic configuration in the Yet Another Markup Language (yaml) format that was used to build the ESPHome firmware for the device. It configures OTA-updates and the basic connection to the MQTT server. Some parts of the configuration were left out because they are not relevant in this scope. The second listing provides the configuration for a single Modbus register as an example. All other registers were configured in the same scheme.

5.1 General configuration

Listing 1: Basic esphome configuration

```yaml
esphome:
  name: $dev_name
  friendly_name: $dev_name

esp32:
  board: esp32-gateway
  framework:
    type: esp-idf

ethernet:
  type: LAN8720
  mdc_pin: GPIO23
  mdio_pin: GPIO18
  clk_mode: GPIO17_OUT
  phy_addr: 0
  use_address: $current_ip

ota:
  password: "82c8244b10a089f5f56db37378ed616c"

web_server:
  port: 80
  auth:
    username: $auth_username
    password: $auth_password

time:
```

```yaml
  - platform: sntp
    id: sntp_time
    timezone: Europe/Berlin
    servers: $ntp_server

text_sensor:
  - platform: ethernet_info
    ip_address:
    name: "IP-Address"

mqtt:
  broker: $mqtt_broker
  port: $mqtt_port
  username: $mqtt_auth_username
  password: $mqtt_auth_password
  client_id: $dev_name
  topic_prefix: "/IoT"
  log_topic: "/system/log"
  keepalive:
    seconds: 5
  skip_cert_cn_check: False
  idf_send_async: False
  certificate_authority: |
    "PLAINTEXT CERTIFICATE IS PLACED HERE"

uart:
  id: modbus_ua
  tx_pin: 4
  rx_pin: 13
  baud_rate: $mb_baudrate
  data_bits: 8
  stop_bits: 1
  parity: NONE

modbus:
  id: modbus0
  uart_id: modbus_ua
  send_wait_time: 250ms
  disable_crc: False

modbus_controller:
  - id: deif_mic2
    address: $mb_slave
    modbus_id: modbus0
    update_interval: $update_interval
    command_throttle: 0ms
```

The general configuration described in listing 1 sets up all necessary base components for the ESPHome firmware on the device. All values with a `$` prefix are replaced during compilation with predefined values. This gives us the ability to configure those values in a central place (top of the configuration, left out here). This is comparable to preprocessor defines in many programming languages. Most of the configuration options here speak for their selves. Some additional explanation might be done for the `ota` option in line 18. This API-key specified here is hard-coded into the firmware on the device. for doing OTA-updates the same key must be present in a configuration for ESPHome to be able to install via the network. This is a security mechanism that only an authorized user can install updates or modify the firmware over the network.
The configuration in line 33 configures a text sensor entity that later on publishes the IP address of the device itself to the MQTT broker. Thereby it's possible to determine the IP address of the device even without physical access.
With the configuration in line 50 the internal MQTT client is forced to use Secure Sockets Layer (SSL) encryption when connecting to the server[2]. The Certificate Authority (CA) can be supplied here in plain text format and was left in the listing for better readability.
The last important configuration sections starting from line 53 are responsible for setting up the pin configuration for the RS-485 converter and configuring the baud-rate for the Modbus interface.

5.2 Register configuration

Listing 2: Modbus register configuration example

```
1  sensor:
2   -  platform: modbus_controller
3    state_topic:
4    id: current_l1
5    name: "Current L1"
6    address: 0x3012
7    modbus_controller_id: deif_mic2
8    register_type: holding
9    value_type: FP32
10   accuracy_decimals: $cfg_decimal_accuracy
11   unit_of_measurement: "A"
12   filters:
13     - multiply: $cfg_current_multiplicator
```

[2]SSL encryption with certificate validation can only be used when using the esp-idf framework. This was set in line 8.

Listing 2 demonstrates how each of the registers of interest from the DEIF instrument were configured within the ESPHome configuration. Within the `sensors` section all registers were defined in this format. The sensor's platform is always `modbus_controller` and the `modbus_controller_id` must be specified each time for the sensor to use the correct hardware Modbus. The option in line 3 was intentionally left blank so that the sensor itself is not automatically published on the MQTT broker[3]. Then the `filters` section in line 12 is besides important. It allows applying different types of filtering functions on the incoming sensor values. Here a simple multiplication filter with a fixed factor was applied to correct the current values coming from the DEIF.

5.3 MQTT publishing

Listing 3: MQTT publishing script

```
1  script:
2    -   id: publish_to_broker
3      then:
4        - mqtt.publish_json:
5          topic: "/IoT/combined"
6          payload: |-
7            root["_ts"] = id(sntp_time).now().timestamp;
8            root["frequency"] = id(frequency).state;
9            root["phasevoltage_v1"] = id(phasevoltage_v1).state;
10           root["phasevoltage_v2"] = id(phasevoltage_v2).state;
11           root["phasevoltage_v3"] = id(phasevoltage_v3).state;
12           root["phasevoltage_avg"] = id(phasevoltage_avg).state;
13           root["linevoltage_v12"] = id(linevoltage_v12).state;
14           root["linevoltage_v23"] = id(linevoltage_v23).state;
15           root["linevoltage_v31"] = id(linevoltage_v31).state;
16           root["linevoltage_avg"] = id(linevoltage_avg).state;
17
18 interval:
19   - interval: $update_interval
20     then:
21       - script.execute: publish_to_broker
```

[3]Due to a bug in a previous ESPHome version the configuration validator did not allow a blank state topic. After an upgrade to a more recent version this malicious behavior was fixed.

Listing 3 shows the process how the different register values are aggregated and published in a single JavaScript Object Notation (JSON) string on the MQTT broker. The `script` component defines a new script that uses the `mqtt.publish_json` action to build and publish a JSON string. The payload is built using the syntax starting from line 7. The sensor components itself were retrieved using their ID and then the `state` method was used to get the state value of the sensor. The first line of the payload is responsible for appending the UNIX timestamp to the data.
The last configuration section in line 18 creates an interval component that executes the publish script within a fixed interval. For the production environment the variable `$update_interval` was set to 1s.

6 Software (Server Side) [RD]

On the server side of this project a flexible and secure approach was picked to deploy several services easily. To fulfill these conditions we make use of containerization using `Docker`.
Within `Docker` a container is an isolated environment for running code. The container has no knowledge of the operating system or the file-system. A container includes everything needed for the containerized application to run, including all library dependencies and the base operating system [11].
The docker daemon[4] was deployed on a `Debian 6.1` virtual server provided by the THM. At the beginning of this project only SSH-Access to this server was available. The setup of the `Docker` daemon and `Portainer`[5] was done together with IIOT-Team1 and Professor Mink. The following subsection describes briefly how this was accomplished.

6.1 Docker and Portainer setup

The enumeration outlines the basic installation and setup process:

1. Upgrade software on the server to the latest releases
2. Installation of docker daemon according to the official instructions [12]
3. Join ssh-user into docker group (rootless control of Docker daemon)
4. Deploy the `Portainer` container
5. Configure basic Portainer settings[6]
6. Create user accounts for team members

This setup provides options to easily deploy docker containers for all team-members without having ssh access onto the server. Portainer also adds user management on top of the Docker daemon. Thus, simple accounting rules can be applied on the different users so their usable resources on the server can be limited.

6.2 Esphome deployment

This section explains the deployment of an ESPHome server used to configure and build the firmware for the hardware described in section 4. Esphome was deployed using `docker compose`. Docker compose is a kind of wrapper for the docker command line interface. In general, you define a „stack“ in the yaml file format where you configure all containers, which ports to expose and which volumes/directories to mount into the container. Such a yaml file was set up within the Portainer web-interface containing the following configuration:

[4] Managing background process for docker containers
[5] Webinterface for managing a docker installation
[6] It's best-practice to disable bind-mounts in Portainer so that Portainer users cannot mount and access the root directory within a container

Listing 4: docker-compose.yml configuration for ESPHome container

```yaml
esphome:
  image: esphome/esphome
  container_name: esphome
  restart: unless-stopped
  ports:
    - "20000:6052" # Web dashboard
  networks:
    - dev-net
  volumes:
    - team4-esphome_team4_esphome_conf:/config
  environment:
    USERNAME: "admin"
    PASSWORD: "iotlab2023team4"
    ESPHOME_DASHBOARD_USE_PING: true
volumes:
  team4-esphome_team4_esphome_conf:
  external: true
```

In this configuration file a new container with the name `esphome` is deployed. The virtual docker volume `team4-esphome_team4_esphome_conf` is mounted according to the ESPHome documentation within the container under the path `/config`. The default ESPHome port `6052` is exposed as port `20000` to not overlap with ports assigned to team 1.
After deploying that container this way the web interface of ESPHome was available under the given port. From that web interface our team was able to deploy and build the yaml-based configuration for the device and also flash it onto the device via the OTA update feature. The configuration itself is explained in detail in section 5. Of special interest is the environment variable `ESPHOME_DASHBOARD_USE_PING`. This enables the dashboard to discover the ESPHome device's online status by using ping and not Multicast DNS (mDNS)[7]. This configuration change is essential because mDNS would not work in the THM network.

6.3 Telegraf

Telegraf is a software developed by Influxdb and has the main task to scrape data from via different source plugins and push them into an Influxdb instance. This tool is highly integrated with Influxdb, thus configuration for a Telegraf process cand be directly generated from an Influxdb. In the Influxdb's web interface you can add a new „Telegraf“ as data source. Then a configuration editor pops up, where the previously deployed MQTT-server was configured. The retrieved data is published in a single MQTT topic in a JSON encoded string. Thus, the Telegraf configuration was set up so that it parses the JSON string and stores all time series data correctly in the Influxdb.

[7] mDNS is the default discovery process of ESPHome.

The next step consisted of deploying the Telegraf instance with docker. listing 5 shows the configuration. The Telegraf agent in the container is started with the environment variable `INFLUX_TOKEN` set to the access token provided by the Influxdb Telegraf integration and the base docker command is appended with the `--config` option and the URL provided by the integration that serves the configuration file. Thus, the configuration could be edited dynamically in the Influxdb web-interface and the Telegraf container re-fetches the configuration on each startup. This way we save ourselves to provide static configuration files for telegraf.

Listing 5: docker-compose.yml configuration for telegraf

```
1 telegraf1:
2   image: telegraf
3   container_name: telegraf1
4   restart: unless-stopped
5   networks:
6     - dev-net
7   environment:
8     INFLUX_TOKEN: "boY1r6bdGmwU00Sou-f-
          VrFuh4zlQovnLbRDaqRQyZNkybBSSg9nwRAlQBL9LlLFr8qNUmdt5dLQAF6IgmouXw=="
9   command: "--config http://influxdb:8086/api/v2/telegrafs/0c4903c473f8e000"
```

The configuration for the Telegraf agent, that is automatically fetched from the Telegraf during startup, is available under the URL in line 9 within listing 5. The following listing 6 shows this configuration:

Listing 6: Telegraf configuration

```
1  [agent]
2  interval = "10s"
3  round_interval = true
4
5  [[outputs.influxdb_v2]]
6    urls = ["http://mt-labor.iem.thm.de:20002"]
7    token = "$INFLUX_TOKEN"
8    organization = "iot-team4"
9    bucket = "iot-data-telegraf"
10
11 [[inputs.mqtt_consumer]]
12   servers = ["ssl://emqx:8883"]
13   topics = [
14     "/IoT/combined"
15   ]
16   client_id = "telegraf-scraper"
17   username = "admin"
18   password = "iotlab2023team4"
19   insecure_skip_verify = true
20
```

```
21   data_format = "json"
22   json_time_key = "_ts"
23   json_time_format = "unix"
```

As you can see in listing 6 the Telegraf agent is configured to write data to the database each 10 s and to round the collected data within this interval. The section beginning from line 5 configures the connection to the indluxdb running within the docker stack. The section beginning at line 11 describes the connection to the MQTT broker. It was configured to listen only to the json encoded topic `/IoT/combined`. The option `insecure_skip_verify` skips the certificate check[8]. Of special interest is the option `json_time_key` that defines a JSON key in the object received at the provided topic that holds the timestamp to write to the Influxdb. **By using this option the timestamp is not appended when writing to the Influxdb, but rather originates from the interface board's local clock itself.** This clock was synchronized to a Network Time Protocol (NTP) server as described in section 5.3 and represents the real timestamps of when sensor values were aggregated on the interface board.

[8]Telegraf still uses an SSL encrypted connection to the broker. Only the CA check is skipped. This saved a bit of work to import the CA certificate into the docker container. For real production use this should be configured later.

7 Visualization with Python [BP]

In this section we will discuss about a visualization with python in the objective to show on figure the angle of voltage and current as well as their values. This code use the package matplotlib which is a mathematic python package with some tools to build some graphs, the package patho to colect data via MQTT and the package JSON to work with the data.

7.1 The code to connect EMQX

To connect EMQX via python, we have to set up session information, create a connection and maintain it.

Listing 7: Connection to EMQX

```
# Set EMQX broker parameters
broker_address = "mt-labor.iem.thm.de"
port = 20005
topic = "IoT/random_data/v1"

# Set client ID, username, and password
client_id = "admin"
username = "admin"
password = "iotlab2023team4"

# Configure the MQTT client with client ID, username, and password
client = mqtt.Client(client_id)
client.username_pw_set(username, password)
client.on_connect = on_connect
client.on_message = on_message

# Connect to the EMQX broker
client.connect(broker_address, port, 60)

# Maintain the connection and process messages
client.loop_forever()
```

7.2 The code to create the figure

The code to create the figure is a bit too long to put it in this section and can be found on git. but here is the principle.

- Create and define the window where we will put the figures
- Create the two figures and create the axis
- When a message is received, received the data from EMQX and convert them in the right format to work with
- Draw on the figure
- Show it in the window

7.3 The result

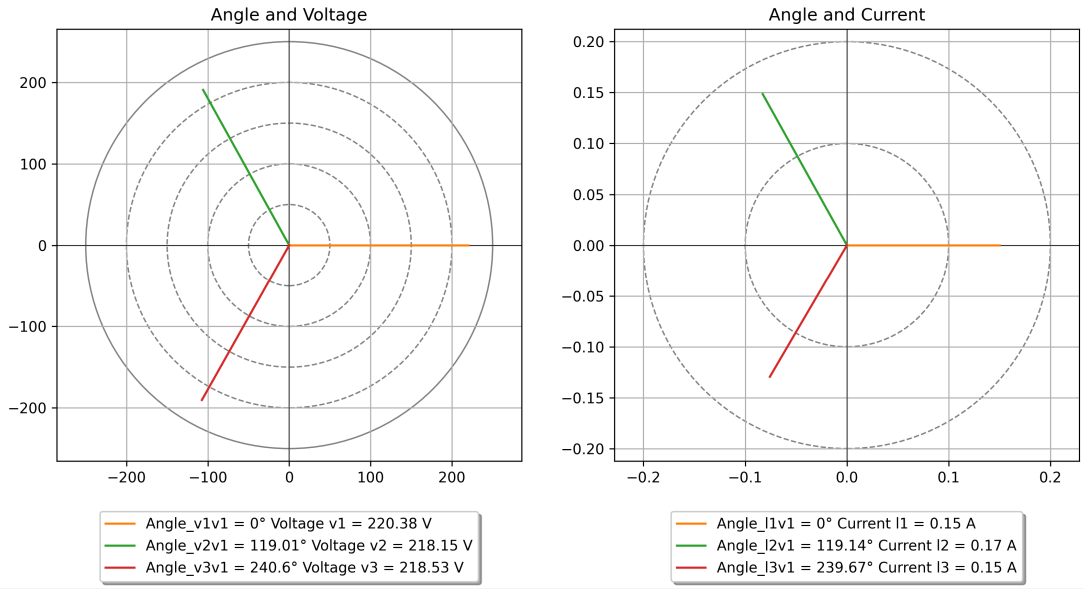


Figure 7.1: Result with matplotlib

To quickly describe this result, there is two figure, one with all the angle of the voltage, the other with the angle of the current. The length of the line represent respectively the voltage or the current. This example have been made with random data (that explain the high values for the current).

8 Visualization with Node-Red & Random Data [DS]

We had to constantly enable and disable the gadget in order to view real-time data and use it for other project purposes. which could take a while, and if something goes wrong, we might not be able to see it. Thus, we provide random data in the same JSON format that the device is providing. Therefore, we can visualize that data without the need for a bus or device by using it as dummy data. Here, The Node-Red has generated the random data. Next, the visualization is completed on two separate platforms: Grafana and Node-Red. Several variables are made, and random numbers are generated in various ranges using JavaScript. Here is an illustration of a single variable: frequency.

Listing 8: Random generation of frequency variable

```
var frequency = parseFloat((Math.random() * 1.111 + 49.999).toFixed(2));
var payload = {
  frequency: frequency,
};

msg.payload = payload;
return msg;
```

The full code of the function block can be found in the git repository of this project (See section 10).

Like this there are more than 20 variables including timestamp. The Function node, a highly helpful node that lets user write custom JavaScript programs to process and modify message data within a flow, is where this JavaScript is employed. To provide input at a time interval of one second, use the Inject node. Then, identical to the real system, created random data is delivered to EMQX MQTT broker.

8.1 Node-Red Visualization

A set of nodes known as a Node-Red dashboard allows a user to create a dashboard for real-time data representation in many formats, such as a chart, gauge, line graph, etc. However, the user must install the node set to use it.The dashboard node set can be installed by the user by searching for node-red-dashboard under Menu - Manage palette option. This is how it appears:

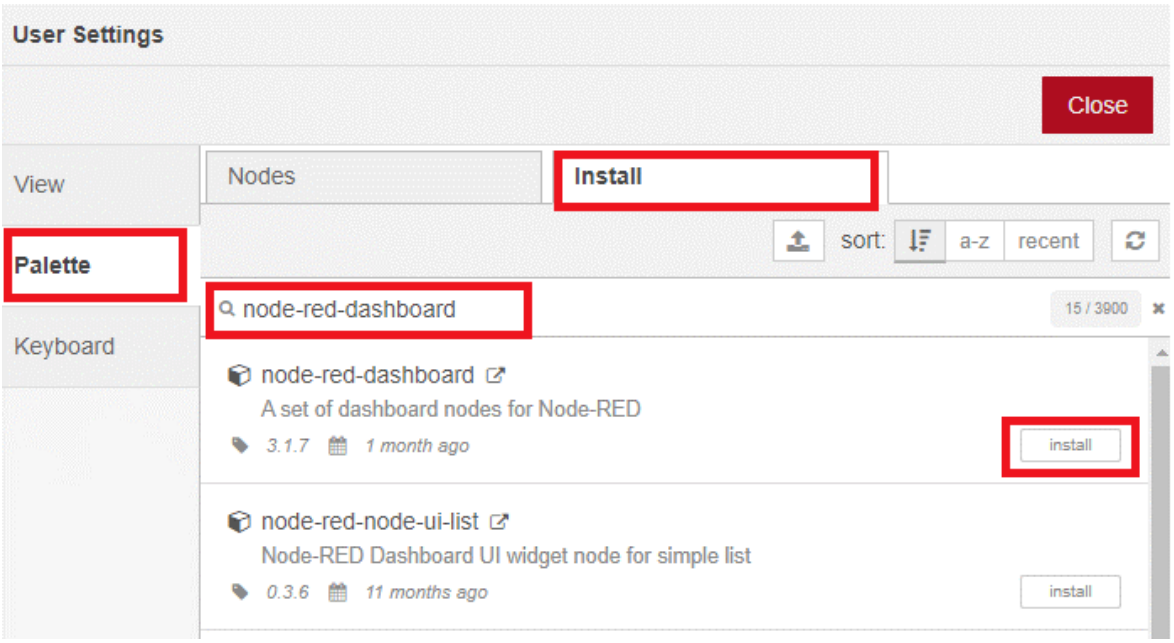


Figure 8.1: Guide NodeRed-dashoard [13]

the user can now access every ui_node, which is useful in a variety of ways to create a dashboard or visualization.
Here, the ui_guage and ui_chart nodes are used to generate the dashboard. The user must define Value format: `{{msg.payload.frequency}}` in the gauge node configuration in order to select certain variables, such as frequency, in the ui_guage. Additionally, the user must choose one specific variable value for the ui_chart by using the function node and JavaScript directly before the ui_chart. Moreover, the user can customize their dashboard by choosing a new tab with a different group for the gauges and charts. This is the appearance of a Node-Red dashboard with random data.

Figure 8.2: NodeRed Visualization

8.2 Grafana Visualization

for Grafana visualization, the user must first navigate through InfluxDB(To store time-stamped data in an Influx bucket). In Grafana, choosing a data source is a prerequisite, and Influxdb is chosen. Subsequently, the necessary details (URL, API pin, bucket-organizer name) must be entered. Now Grafana and InfluxDB is linked. The creation of a dashboard is necessary to visualize a single or more graphs. To obtain certain data from InfluxDB, created InfluxDB queries is copied and used the Grafana Query Editor.Then user can Personalize dashboard panels to display the data as graphs, tables, or gauges. This is the appearance of a Grafana dashboard with random data.

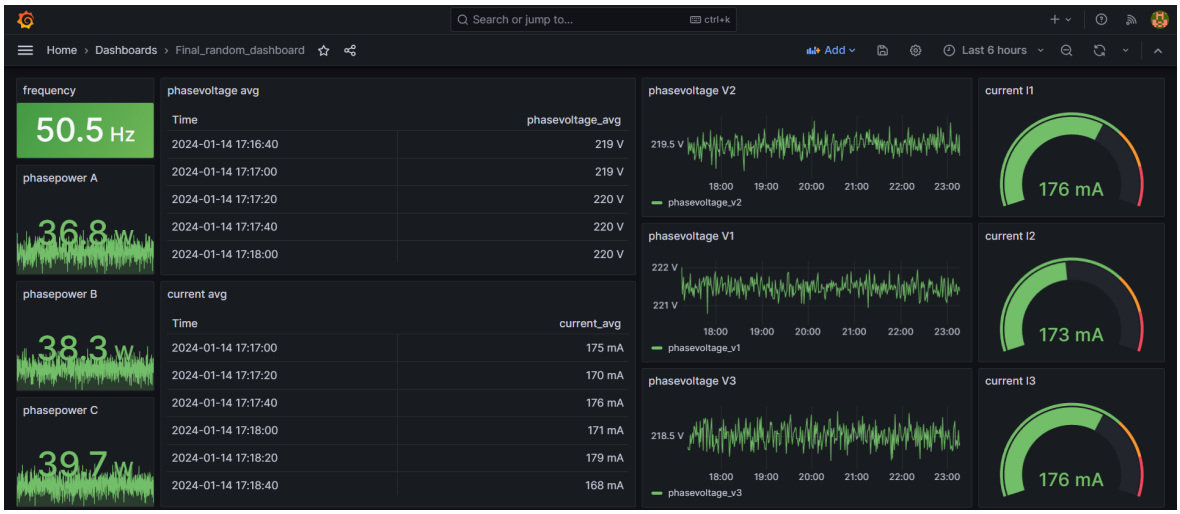


Figure 8.3: Grafana Visualization

9 Visualization with MATLAB [AK]

9.1 Introduction to MATLAB

Figure 9.1: MATLAB Symbol [14]

MATLAB (an abbreviation of MATrix Laboratory) is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.
Although MATLAB is intended primarily for numeric computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.
In MATLAB, you can use the C language programming, although it supports some other programming languages like Fortran and Java. For our work, we are using the C language. For this representation of the data, we are using MATLAB 2023b version.

9.2 Installation of the MATLAB Software

You will find all the details regarding how to install MATLAB and how to start with it in the following source [15].
You will find at the below-mentioned location about how to add a module in MATLAB. In this project, we are using the Industrial communication toolbox module for MATLAB to MQTT communication [16].
When you go through the details and install MATLAB, you will find the below screen when you open the first time MATLAB application.

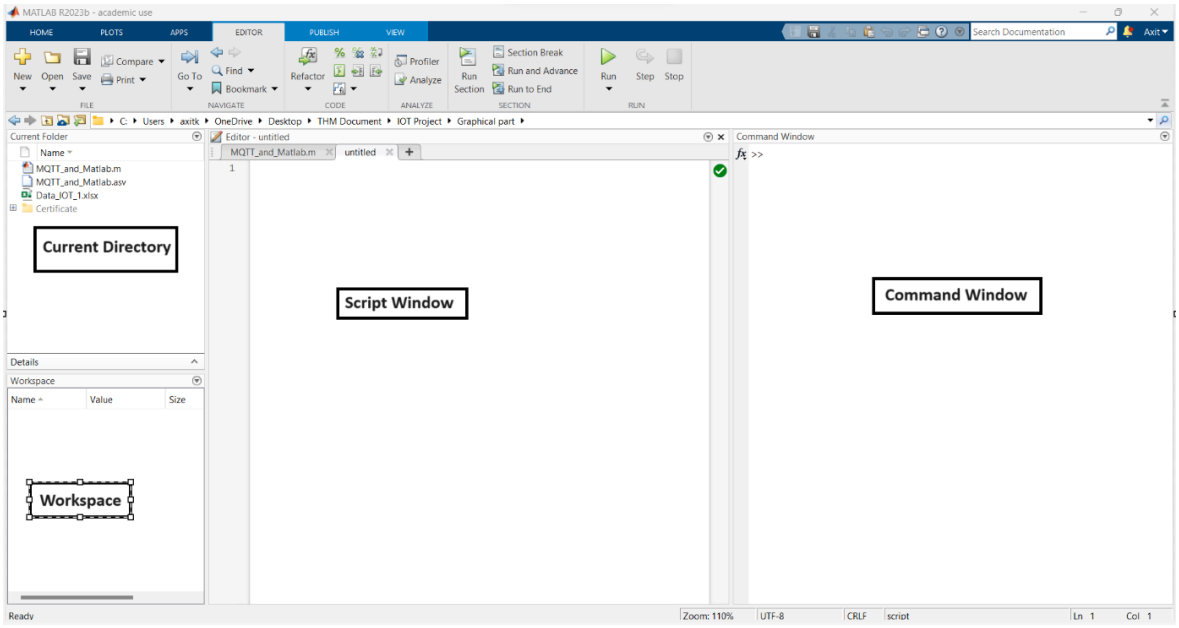


Figure 9.2: Initial Screen for MATLAB

All the different parts of the MATLAB home screen are listed below.

1. Current Folder: It will show the current directory of the file where it is located.
2. Script Window: You can write your script here in this window.
3. Command Window: You can write your command in this window.
4. Workspace: This window will show the variables you have used in the script.

9.3 Script for the Data Representation

You will find the script we had configured regarding the data visualization at the below-mentioned location in the git repository (See section 10).
Location: `software/matlab-demo/MQTT_and_Matlab.m` You will find all the details regarding which part of the code is used for which application in the code itself.

9.4 Result of the Data Representation

Here I am sharing some of the screenshots of the result that we had captured.

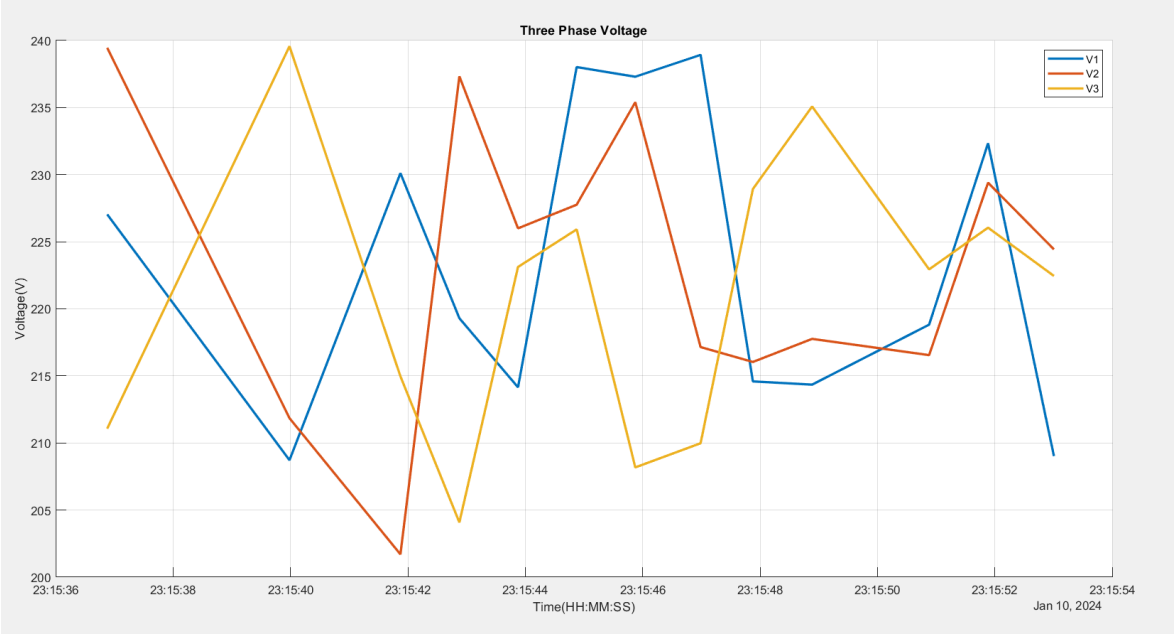


Figure 9.3: Three Phase Voltage Representation in MATLAB

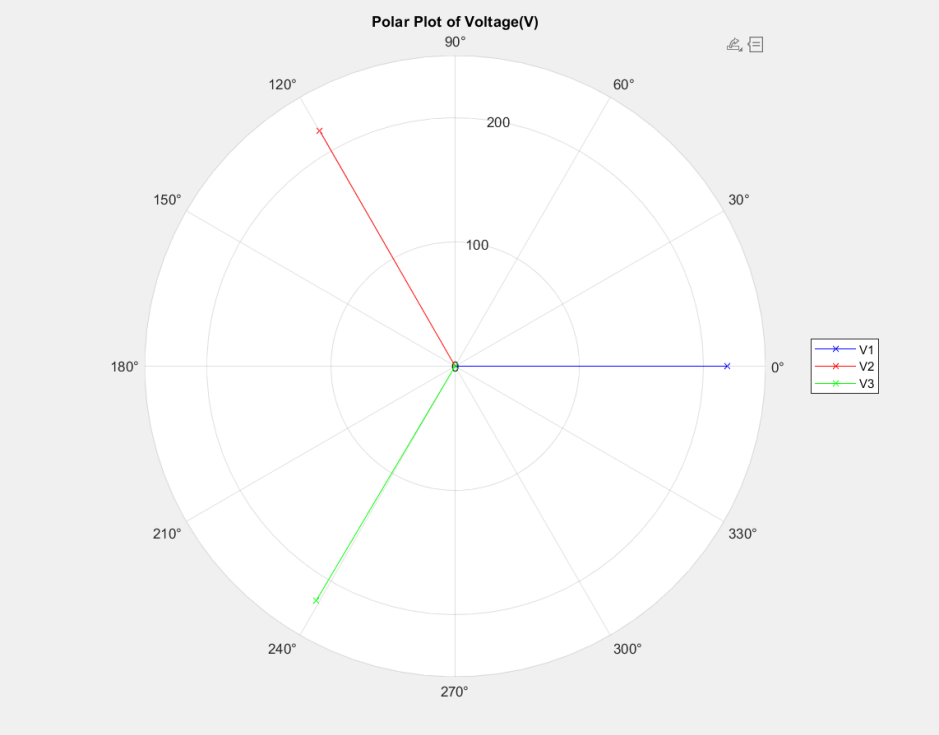


Figure 9.4: Three Phase Vector Diagram Representation of Voltage in MATLAB

10 Conclusion [MS]

The Power Consumption data Analysis Project mainly focuses on providing a clear understanding of power usage through real-time data visualization. By using a combination of advanced tools and techniques, this project primarily aims to develop the analysis of power consumption patterns and trends in conventional graphs and polar diagram.

Live Data Collection

Implemented a system to capture real-time power consumption data from the DEIF power analyzer module by using Modbus communication protocol and with the help of ESP32 microcontroller the collected information transferred in real time basis to the emqx server by using Node red platform.

Data Integration

Integrated data from emqx server to the influxdb and create a comprehensive dataset for the power usage parameters.

Visualization Interface

Integrated Node-RED with several visualization tools of Grafana, MATLAB, and Matplotlib and visualize by transmitting data to these platforms for customize the dashboard, and customizable plots. Grafana for real-time dashboards, MATLAB for extensive scientific visualization, or Matplotlib for Python-based plotting.

Tools and Technologies

Utilized several tools such as Python for data processing, InfluxDB for real-time database systems and storage, Modbus communication protocol for data transmission, Node red for connect the platforms, Grafana, Matlab, Matplotlib for data visualization

Polar Diagram Visualization

Implemented an effective visualization graph using a polar diagram for users to interpret complex power consumption patterns easily and with clarity.

References

[1] Advantech Co., Ltd. „RS-485 Connections FAQ“, [Online]. Available under: https://www.advantech.com/en/resources/white-papers/02cb2f4e-4fb2-4a87-be3b-508325bd61d6 (Accessed on: 01/11/2024).

[2] et. al. „Wikipedia Entry RS-485“, [Online]. Available under: https://en.wikipedia.org/wiki/RS-485 (Accessed on: 01/11/2024).

[3] et. al. „Wikipedia Entry UART“, [Online]. Available under: https://en.wikipedia.org/wiki/Universal-asynchronous-receiver-transmitter (Accessed on: 01/11/2024).

[4] A. Omiccioli. „Modbus protocol over su RS485 – Part 3 – Data Link layer“. (2017), [Online]. Available under: https://web-plc.com/blog/2017/06/01/rtu-modbus/ (Accessed on: 01/09/2024).

[5] Espressif Systems. „ESP32-WROOM-32E Datasheet“, [Online]. Available under: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf (Accessed on: 01/11/2024).

[6] Nabu Casa Inc. „ESPHome“, [Online]. Available under: https://esphome.io (Accessed on: 01/11/2024).

[7] Nabu Casa Inc. „ESPHome Logo“, [Online]. Available under: https://esphome.io (Accessed on: 01/11/2024).

[8] DEIF A/S. „DEIF MIC-2 MKII product image“, [Online]. Available under: https://deif-cdn-umbraco.azureedge.net/media/hwcj24l1/mic-2-mkii-front.png (Accessed on: 01/11/2024).

[9] „Olimex website“, [Online]. Available under: https://www.olimex.com/About/ (Accessed on: 11/01/2023).

[10] R. Sharma. „How is this RS485 Module Working?“ (2016), [Online]. Available under: https://electronics.stackexchange.com/questions/244425/how-is-this-rs485-module-working (Accessed on: 11/01/2023).

[11] „What is a container?“ (2023), [Online]. Available under: https://docs.docker.com/guides/walkthroughs/what-is-a-container/ (Accessed on: 12/06/2023).

[12] „Install Docker Engine on Debian“. (2023), [Online]. Available under: https://docs.docker.com/engine/install/debian (Accessed on: 12/06/2023).

[13] RandomNerdTutorials.com. „Getting Started with Node-RED Dashboard on Raspberry Pi“. (2023), [Online]. Available under: https://randomnerdtutorials.com/getting-started-node-red-dashboard/ (Accessed on: 01/15/2024).

[14] MathWorks Inc. „Matlab logo“, [Online]. Available under: https://www.mathworks.com/ (Accessed on: 01/16/2024).

[15] MathWorks Inc. „Matlab install products“, [Online]. Available under: https://www.mathworks.com/help/install/install-products.html (Accessed on: 01/16/2024).

[16] MathWorks Inc. „Matlab install toolboxes“, [Online]. Available under: https://www.mathworks.com/matlabcentral/answers/101885-how-do-i-install-additional-toolboxes-into-an-existing-installation-of-matlab (Accessed on: 01/16/2024).

Appendix

All code and other software components developed during this project were tracked and published on the THM Git server. The project itself is available at:

https://git.thm.de/rtdt35/iiot-project

Further explanations on the purpose of the sub-folders and the repository structure are within the `README.md` file contained in the projects root.