www.elektormagazine.com

SPECIAL

elektor

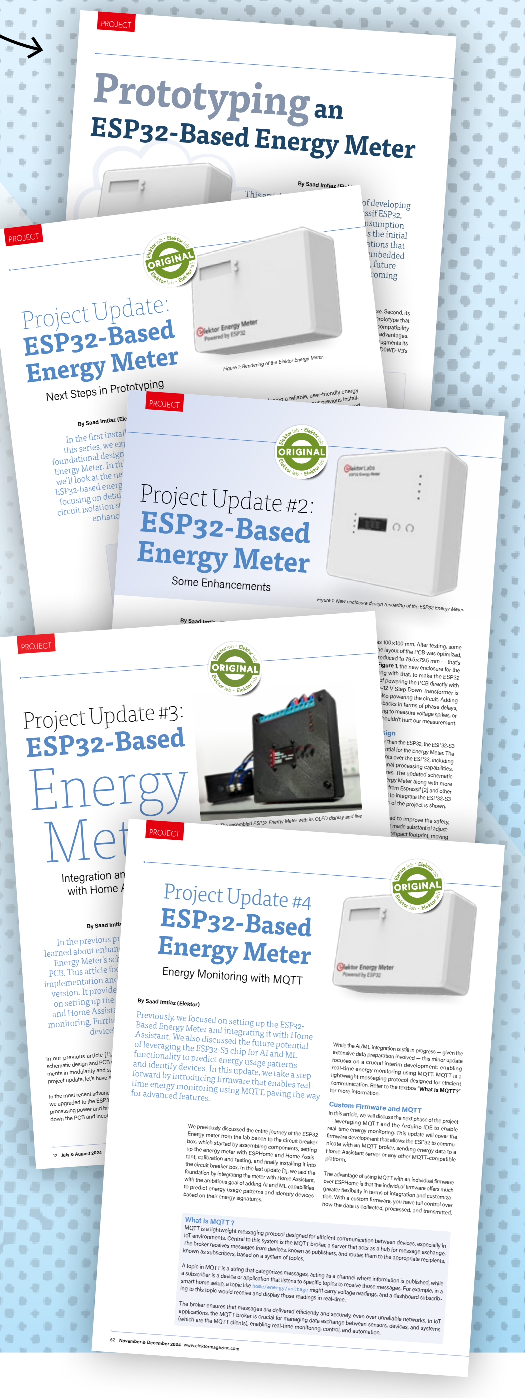
design > share > earn

SPECIAL

Elektor ESP32 Energy Meter

Elektor lab · Elektor lab · ORIGINAL · Elektor lab · Elektor lab

IN THIS EDITION


PROJECT

Prototyping an ESP32-Based Energy Meter

PROJECT

Project Update: ESP32-Based Energy Meter

Next Steps in Prototyping

PROJECT

Project Update #2: ESP32-Based Energy Meter

Some Enhancements

PROJECT

Project Update #3: ESP32-Based Energy Meter

Integration and with Home Assistant

PROJECT

Project Update #4 ESP32-Based Energy Meter

Energy Monitoring with MQTT

Prototyping an ESP32-Based Energy Meter

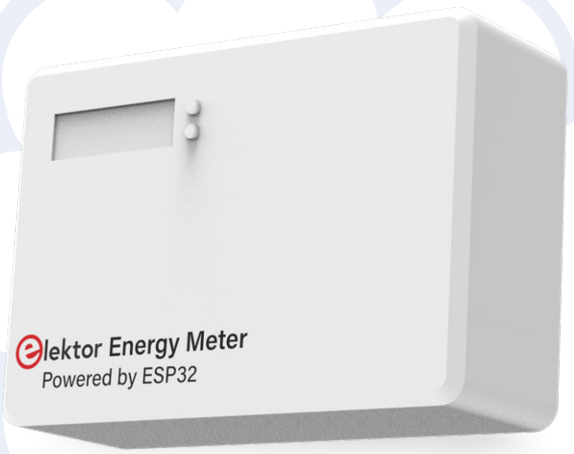


Figure 1: Test rendering of how our Single-Phase Energy Meter might look.

By Saad Imtiaz (Elektor)

This article presents the journey of developing an energy meter using an Espressif ESP32, emphasizing real-time power consumption monitoring and safety. It highlights the initial steps, requirements, and considerations that take place when embarking on an embedded project. As the project progresses, future achievements will be shared in upcoming editions of *Elektor Mag*.

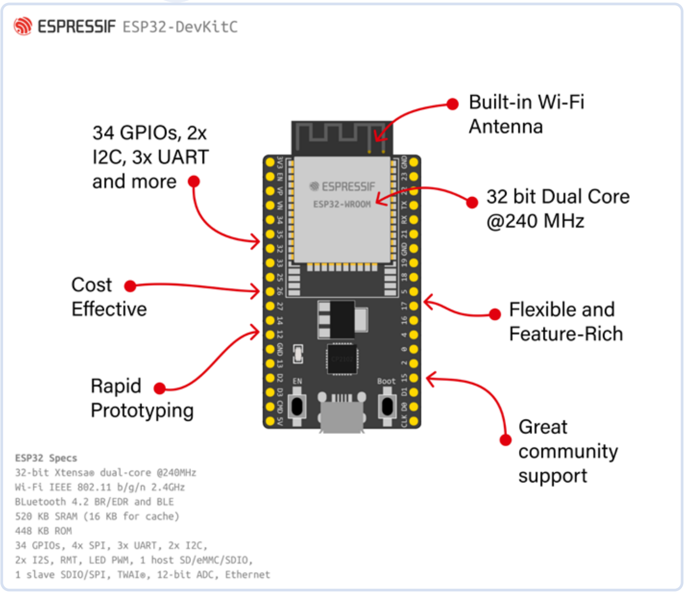
In the field of engineering, combining the right technologies can lead to significant advancements. This project aims to develop an energy meter using the Espressif ESP32 microcontroller and Microchip's ATM90E32AS energy metering IC. In this article, the beginning of this project's journey is briefly shared, from component selection to prototyping. The goal is straightforward: to create a reliable system for accurate energy measurement from your home or workshop's main circuit box. This meter will enable users to track their power consumption in real time, offering insights that can lead to more efficient energy use.

Design and Requirements

The project has clear goals and design requirements: real-time monitor single-phase power using three current transformers (CTs), keep it affordable, and make it user-friendly. The choice of ESP32 and ATM90E32AS IC components was guided by these aims, offering both cost-effectiveness and reliable performance. Another target was to keep the size smaller than 100×80×30 mm (L×W×H) to ensure that it can be accommodated in a circuit breaker box. To enhance the user experience, a mobile interface is also included for remote monitoring, as well as an OLED display with buttons for direct interaction. The design also allows for future software updates, ensuring long-term utility for the consumer. In **Figure 1**, the rendering of the current prototype enclosure is shown.

Microcontroller Selection

The choice of the ESP32 microcontroller was predicated on a detailed analysis of its capabilities. The chip excels in several areas crucial to the success of this project. First, its ease of integration into varied circuit designs provides flexibility during the engineering phase. Second, its cost-effectiveness makes it an attractive choice for a prototype that aims to balance performance and budget. Third, the compatibility with a wide range of sensors and ICs offers significant advantages. Lastly, the extensive community support for ESP32 chip augments its suitability for this project. **Figure 2** highlights the ESP32-D0WD-V3's



```
ESP32 Specs
32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet
```

Figure 2: Main features and advantages of the ESP32.

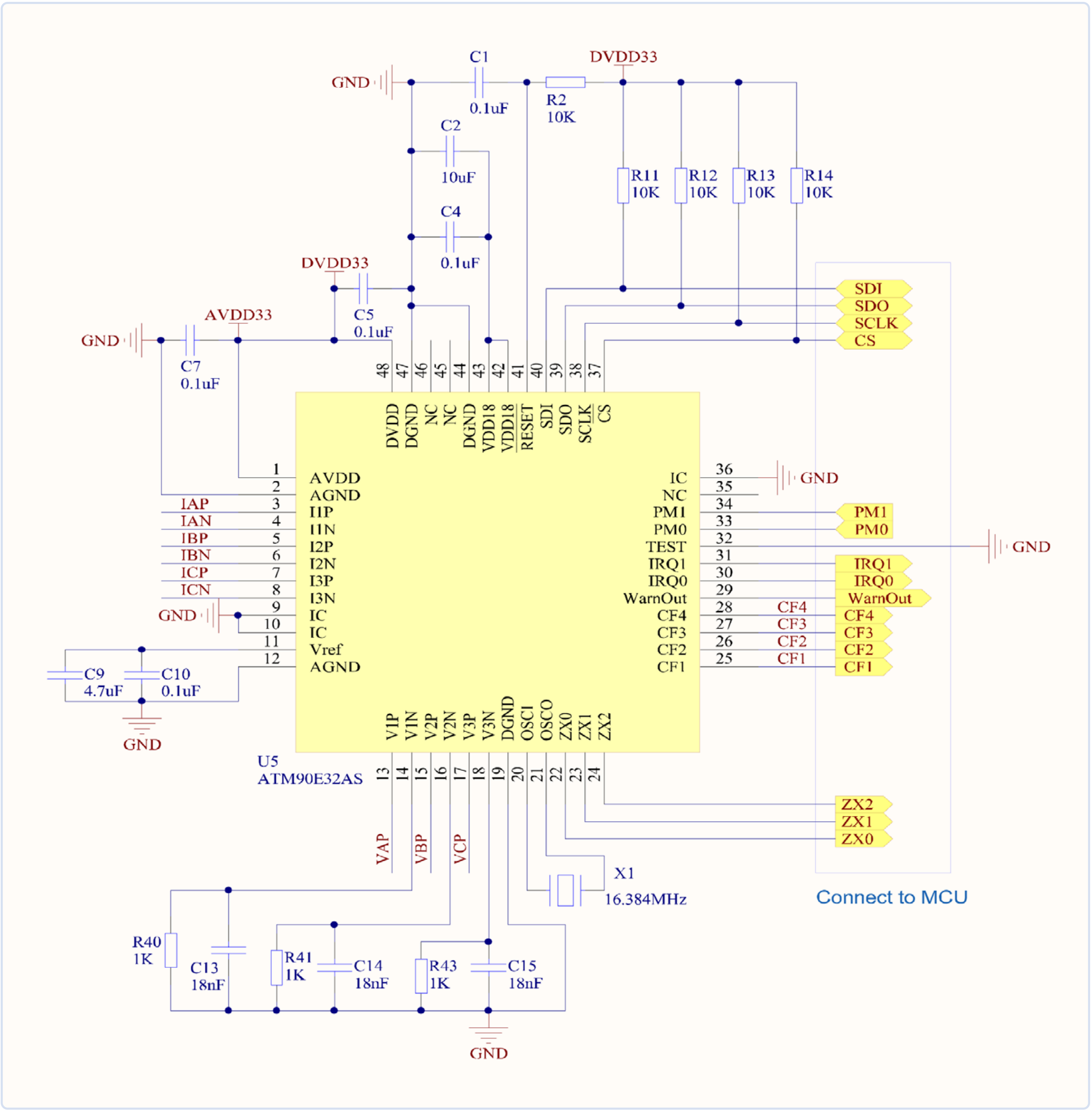


Figure 3: The energy meter is based on an application note by Atmel [2]. Here, you can see the circuitry around the metering IC.

main features and advantages resulting in its being selected for this project.

Metering IC Integration

The ATM90E32AS IC from Microchip was integrated according to the manufacturer's application note; the document served as a cornerstone in ensuring that the energy metering IC communicated seamlessly with the ESP32 microcontroller. However, this phase was not devoid of challenges. The procurement of the correct components within budget constraints required meticulous planning, given the constraints on availability. In **Figure 3**, the application note provided by Atmel (now Microchip) in shown.

Design Phase and Electrical Safety Standards

The design phase is indeed a pivotal part of the engineering process, particularly when safety is an indispensable consideration. In a device designed to interact with mains AC voltages, meticulous attention must be paid to conformance with established safety standards. In **Figure 4**, the project's block diagram is shown.

To ensure safety, several specialized electrical components were integrated into the design. Metal oxide varistors (MOVs) were used for transient voltage suppression to protect the circuitry from voltage spikes. Furthermore, fuse components were included as an essential failsafe to prevent overcurrent conditions.

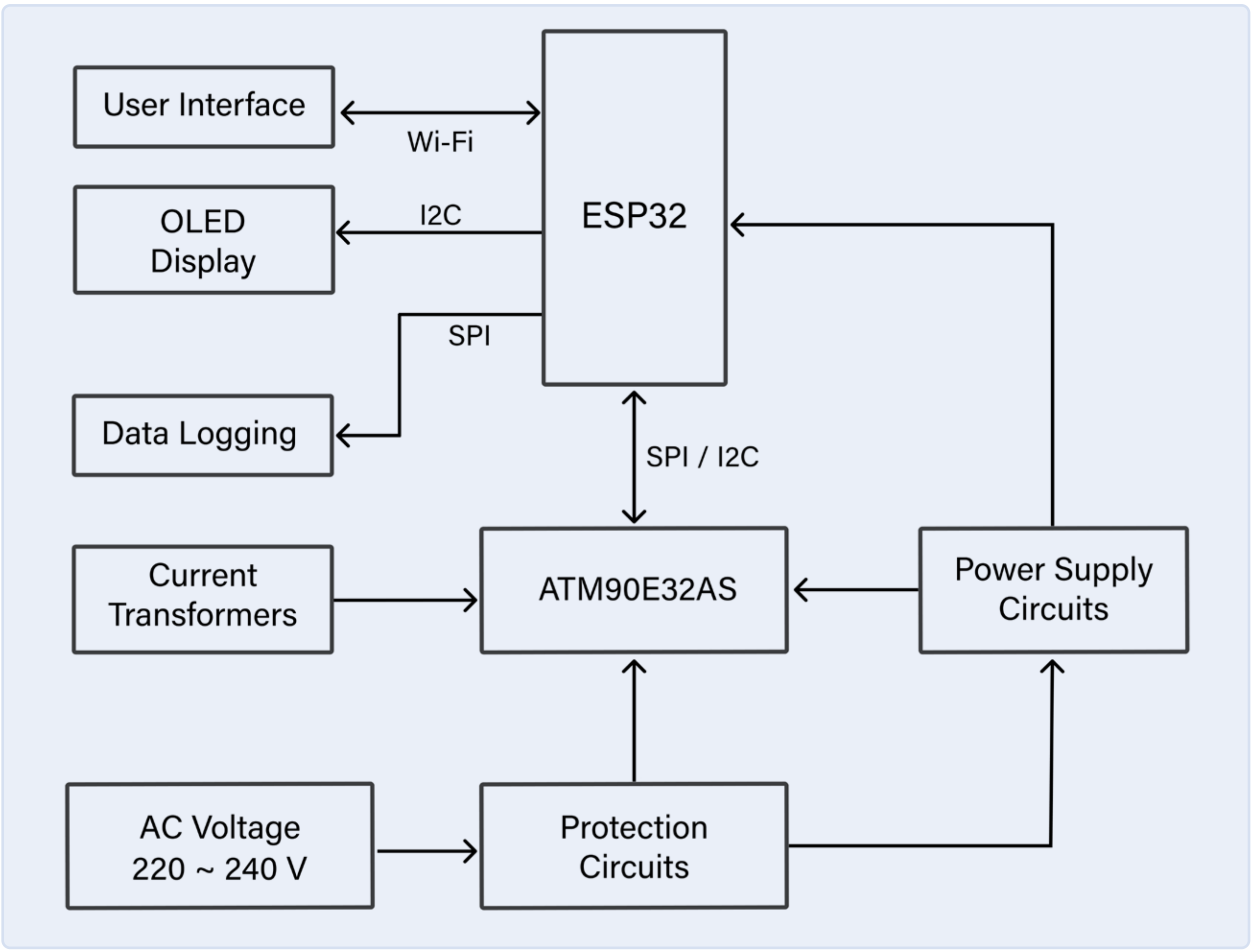


Figure 4: Block diagram of our Energy Meter project.

Beyond component selection, circuit design also focused on layout considerations that would abide by safety norms. Adequate creepage and clearance distances were maintained between the conductive elements on the PCB to prevent electrical arcing. Trace widths for AC voltage lines were calculated carefully to handle the current ratings, adhering to IPC-2221 standards [1]. This was critical in ensuring the thermal performance of the board under full-load conditions. To ensure ground integrity, a solid ground plane was used. Special attention was given to the design of differential pairs for signal integrity, making sure that the routing followed precise geometry to minimize electromagnetic interference.

Manufacturing Selection: Opting for JLC PCB

After scrutinizing various PCB assembly services, JLC PCB was selected. The principal reason for this choice was the balance of cost-effectiveness and reliability that they offer. This decision was important in keeping the project within budget without compromising on the quality of the assembled board. Currently, the prototype schematic and PCB designs are being finalized, and they will soon be sent for production.

Reflecting on the Journey and Looking Forward

In retrospect, this project shows what can be achieved when careful planning meets good engineering. The hurdles we faced helped us improve our design. As we move from making a prototype to possibly mass-producing it, we expect it to make a real difference in how people manage energy. This project will be detailed in upcoming editions of this magazine — we're still in the process of getting the prototype made, tested, and working, on the software that will run it. There's more to come, so stay tuned for updates on this project. We will near completion and share updates in the January/February 2024 edition of Elektor, which is dedicated to the topic of *Power & Energy*.

230646-01

Questions or Comments?

If you have questions about this article, feel free to email the Elektor editorial team at editor@elektor.com.

Related Products

- **ESP32-DevKitC-32E**
  www.elektor.com/20518
- **ESP32-C3-DevKitM-1**
  www.elektor.com/20324

Project Update: ESP32-Based Energy Meter

Next Steps in Prototyping

Figure 1: Rendering of the Elektor Energy Meter.

By Saad Imtiaz (Elektor)

In the first installment of this series, we explored the foundational design of the Elektor Energy Meter. In this installment, we'll look at the next phase in the ESP32-based energy meter project, focusing on detailed schematics, circuit isolation strategies, and key enhancements.

We began our journey of devloping a reliable, user-friendly energy meter leveraging the ESP32 microcontroller. In our previous installment, "Prototyping an ESP32-Based Energy Meter" [1], we discussed the initial design requirements, block diagram and the plan for starting this project. Before we give an update on it, let's have a recap. The energy meter conceptual design is shown in the rendering in **Figure 1**.

Our focus was on real-time power monitoring, with an emphasis on safety and affordability. To make the energy measurement precise, we opted for the Atmel ATM90E32AS, a polyphase energy monitoring IC [2]. This IC will get the single-phase voltage from the mains and will use split coil transformers to measure the current safely. The main application microcontroller selected was ESP32 as it has built-in Wi-Fi and very cost-effective when compared to other MCUs. In **Figure 2**,

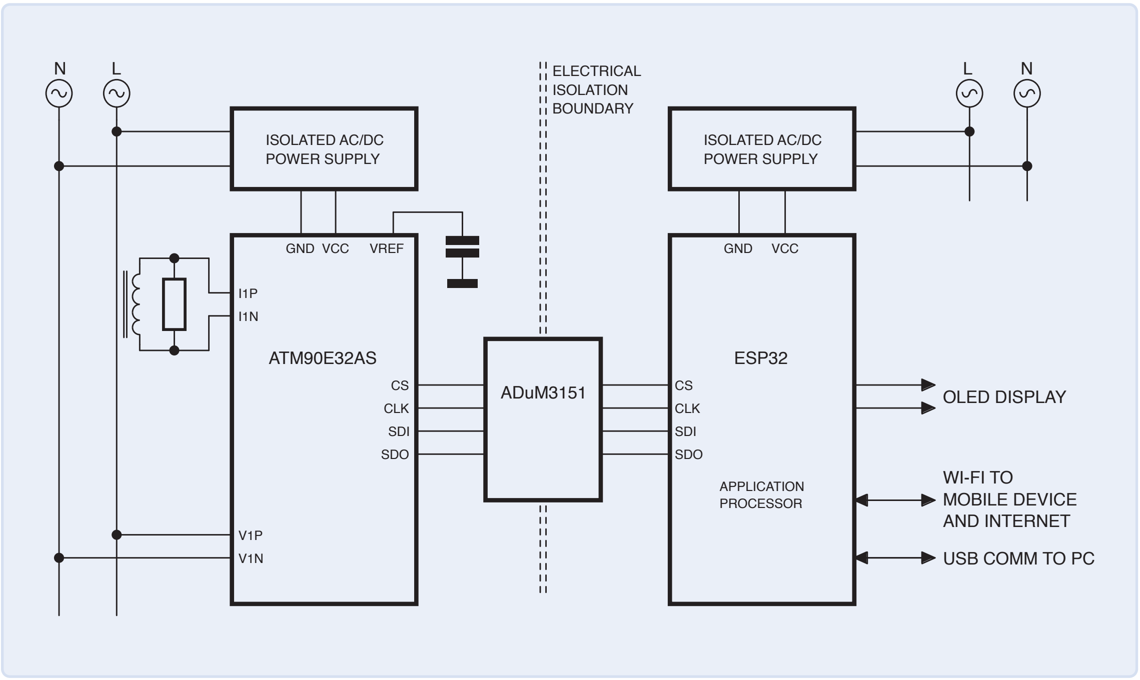


Figure 2: Block Diagram of the Elektor Energy Meter.

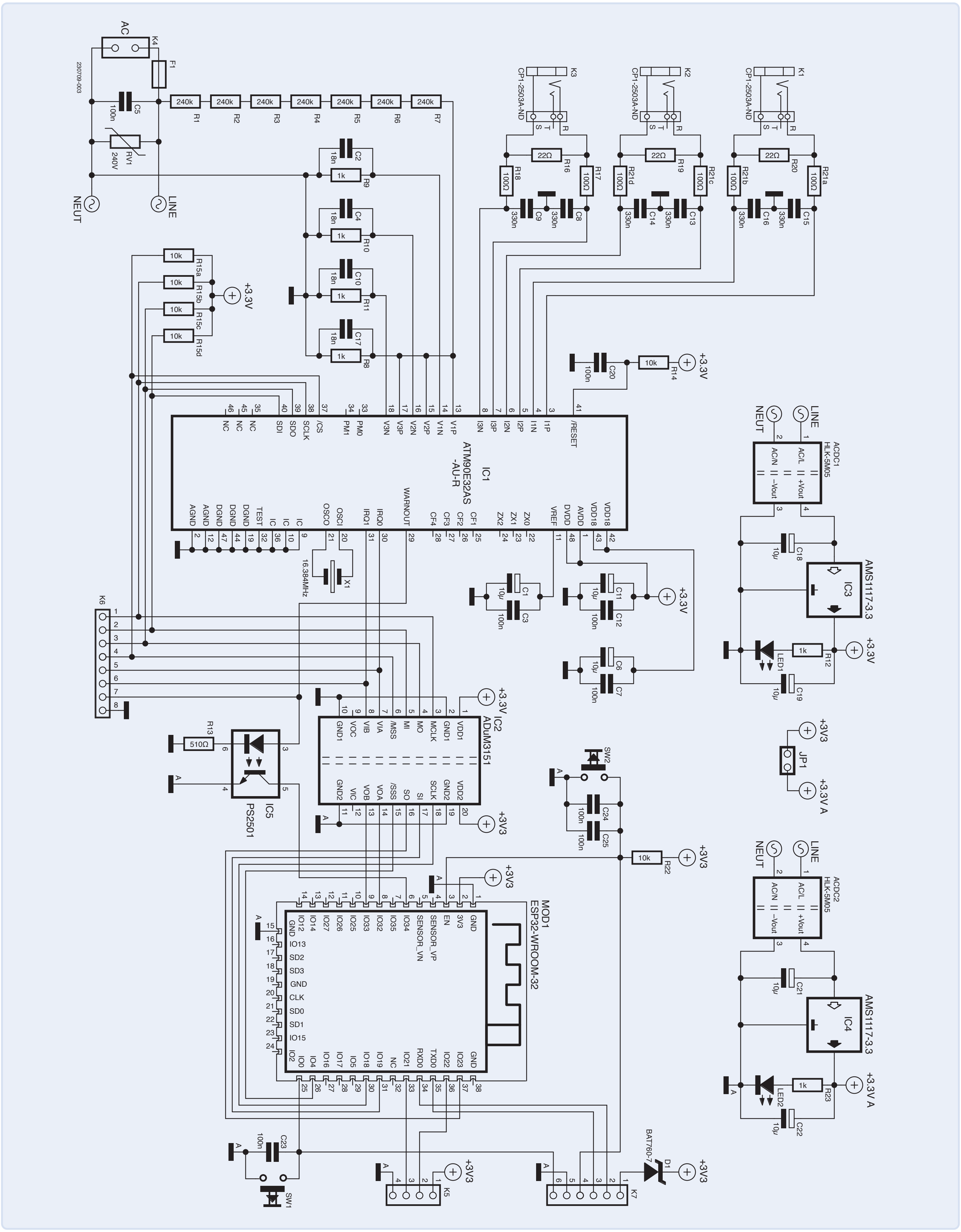


Figure 3: Schematic diagram of the project.

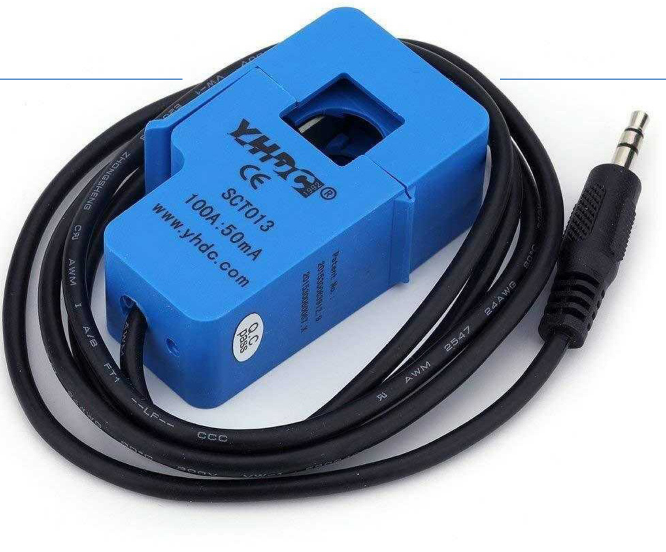


Figure 4: The YHDC SCT013 . (Source: YHDC)

the project's updated block is shown. The planned size of the final energy meter is 100×80×30 mm (L×W×H), but, for the prototype, our PCB is 100×100 mm. Our goal of this prototype is as proof-of-concept and, subsequently, if we're successful in it, we'll scale the size down to 100×80 mm or even less for the final version. The main purpose of making this energy meter was to make an IoT-enabled, budget-friendly device that can make accurate energy measurements and provide real-time energy data to the user via a mobile device, so the user can track their power consumption in real time and become more energy efficient.

In this article, we dive deeper into the project's evolution, highlighting the schematic design, the implementation of circuit isolation, and the key improvements we've integrated since our initial concept.

Schematic Design

The heart of our project lies in its schematic design. The ESP32 micro-controller remains central to our architecture, interfacing seamlessly with the ATM90E32AS for energy measurement. Our updated schematic reflects a more streamlined approach, reducing noise and enhancing signal integrity, circuit isolation, and more. In **Figure 3**, you can see the complete schematic of the project.

IC1 is the ATM90E32AS which is the brain of this entire project, it connects the mains voltage with series seven 240 k resistors (R1…R7) to pins V1P, V2P, and V3P. For keeping things simple, all these pins will be given a single-phase voltage from the mains. You might ask, why not use a transformer instead of these series of resistors? Because as we have size and cost constraints due to the approach we chose. Apart the small size benefit of using resistors there is another benefit, that is less Phase Delay. Transformers can introduce a phase delay between the primary and secondary windings, which could affect the timing and accuracy of voltage readings in energy measurements. When using resistors, this phase delay is significantly reduced, potentially leading to more accurate real-time voltage readings. But, using these series resistors has a major disadvantage, i.e., no galvanic isolation. We will talk about this later in the article.

Now moving to the current measurement: For that we will be using coil transformers (CTs). Connectors K1-K3 are audio jack connectors where the SCT013 by YHDC will be connected, which is a split core type CT, shown in **Figure 4**. The reason for selecting CT was that it is cost-effective easy to use, and non-invasive.

The energy metered is powered by two Hi-Link HLK-5M05 modules ACDC1/2, to ensure galvanic isolation between the MCU and energy meter circuitry, protecting against high-voltage risks. AMS1117-3.3 regulators provide stable 3.3 V power, essential for the ESP32 and other low-voltage parts. Safety is further bolstered by fuses (F1) for overcurrent protection and a metal oxide varistor (MOV) (R23) against voltage spikes. For diagnostics, LED1 and LED2 indicate power and operational status. Connector K6 connects to all the outputs for the MCU for debugging operations.

Circuit Isolation

In the schematic, you might have noticed two DC grounds, GND and GNDA. The ground terminal (GND) is connected to IC1 and is also connected to the AC mains neutral. GNDA is an isolated ground terminal that is connected to the ESP32-WROOM-32D, which is MOD1. To ensure safety, it is imperative to isolate the ESP32 from the AC mains neutral. As the IC1 lacks galvanic isolation, it is imperative to isolate these components from each other. Now, the question arises as to how the SPI between these two chips will be communicated. Here is when IC2, an Analog Devices ADuM3151, comes into play.

The ADuM3151 is pivotal in ensuring safe communication between IC1 and the ESP32-WROOM-32D, providing galvanic isolation for SPI lines. In **Figure 5**, you can see the functional block diagram of IC2 [3]. It uses inductive couplers to transfer digital signals across an isolation barrier, effectively shielding the computer-connected ESP32

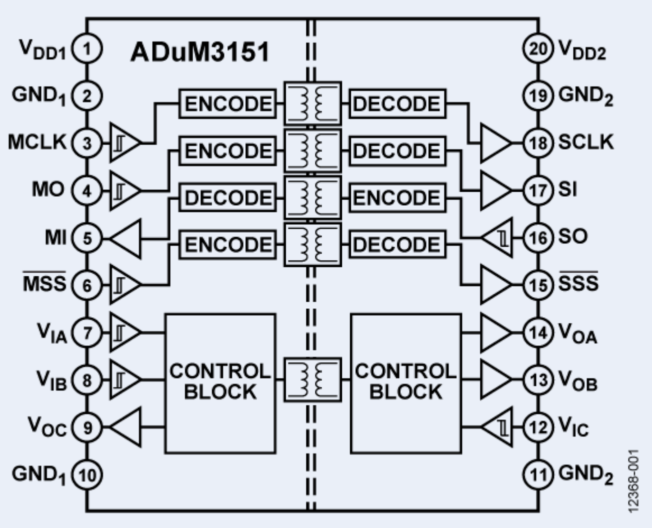


Figure 5: ADuM3151 SPIsolator functional block diagram. (Source: Analog Devices [3])

from the AC mains' high-voltage transients. This choice is crucial for preventing damage during coding and debugging, while its capability of supporting multiple isolated channels ensures reliable and secure SPI communication, maintaining data integrity and aligning with the project's safety and performance goals.

User Interface and Interaction

The user interface of the ESP32 Energy Meter project is designed to be informative and user-friendly. An OLED Display, connected to connector K5, which is interfacing with the I2C pins of the ESP32, will serve as the primary display medium. This display will show all relevant data to the user in real-time, including energy consumption metrics and system status. The choice of OLED technology ensures clear visibility and a responsive interface.

In addition to the hardware display, the project incorporates a web server hosted on the ESP32. This web interface will mirror the data displayed on the OLED screen, offering users an alternative way to monitor their energy usage. The development team is dedicated to creating a web UI and UX that is both user-friendly and detailed, ensuring accessibility and comprehensiveness in data presentation. This dual-interface approach allows users to interact with the energy meter both physically and remotely, enhancing the overall usability of the system.

Next Steps and Future Plans

As the project moves forward, the initial PCB design has been sent off for manufacture. Upon its return, the focus will shift to the firmware side of the project. The firmware development will involve programming the ESP32 to accurately process and display energy consumption data, manage the web server, and ensure smooth communication between all components.

Looking ahead, there are plans to integrate additional features to enhance the energy meter's functionality. These may include:

- Remote monitoring capabilities: Allowing users to check their energy consumption data from anywhere via the web interface.
- Alerts and notifications: Implementing a system to alert users about unusual energy consumption patterns or potential system issues.
- Data analysis tools: Incorporating analytical tools in the web interface to help users understand their energy usage trends and identify areas for efficiency improvements.

We are committed to continuous improvement and innovation, with a focus on user feedback to guide future enhancements. The goal is to not only provide a reliable energy monitoring tool, but also to empower users with insights into their energy usage, fostering awareness and efficiency.

230709-01

Questions or Comments?

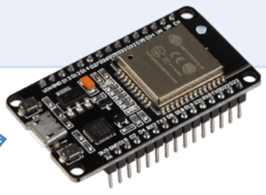
If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.

About the Author

Saad Imtiaz (Senior Engineer, Elektor) is a mechatronics engineer with five years of experience in embedded systems, mechatronic systems, and product development. He has collaborated with numerous companies, ranging from startups to enterprises globally, on product prototyping and development. Saad has also spent time in the aviation industry and has led a technology startup company. Recently, he joined Elektor and drives project development in both software and hardware.

Related Products

- **LILYGO T-Display-S3 ESP32-S3 Development Board**
  www.elektor.com/20299
- **ESP-C3-12F-Kit Development Board (4 MB Flash)**
  www.elektor.com/19855
- **Joy-IT NodeMCU ESP32**
  www.elektor.com/19973



WEB LINKS

[1] Saad Imtiaz, "Prototyping an ESP32-Based Energy Meter," Elektor Guest Edition 2023: http://www.elektormagazine.com/230646-01
[2] ATM90E32AS Poly-Phase Energy Metering IC: https://www.microchip.com/en-us/product/atm90e32as
[3] Analog Devices Inc. ADuM3151 SPIsolator™ Digital Isolators: https://eu.mouser.com/new/analog-devices/adi-adum3151-isolators

Project Update #2: ESP32-Based Energy Meter

Some Enhancements

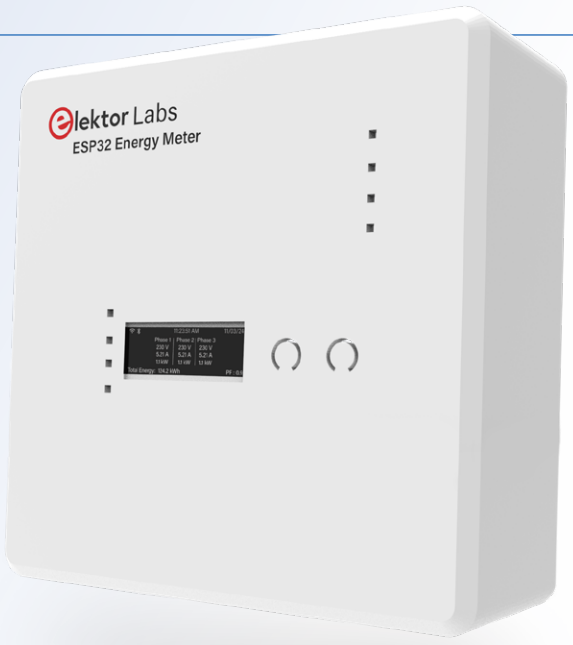


Figure 1: New enclosure design rendering of the ESP32 Energy Meter.

By Saad Imtiaz (Elektor)

In the previous installment of this series, we discussed the schematics, and circuit isolation strategies of the ESP32 Energy Meter. Now let's discuss further enhancements, a PCB design, and more.

In 2023, we started with the goal of creating a reliable, user-friendly energy meter using an Espressif ESP32 microcontroller. In our last article, "Project Update: ESP32-Based Energy Meter" [1], we went over the block diagram, the schematics, circuit isolation strategy, features, and project strategy. Let's start with a small recap before we get into the next update.

The main idea revolved around developing a precise and efficient energy meter leveraging the capabilities of the Espressif ESP32 microcontroller and the ATM90E32AS IC for energy measurement. The project aimed to enhance user experience and reliability through meticulous schematic design and circuit isolation using the ADuM3151 to provide safe communication between ESP32 and the ATM90E32AS by Atmel (now Microchip). It emphasized safety and efficiency by incorporating noise reduction techniques, signal integrity enhancements, and protective mechanisms such as fuses and MOVs. With a focus on future-ready features, the plan included integrating remote monitoring and data analysis tools for improved energy management and efficiency insights.

In this article, our main goals remain the same, and plenty of changes have been made to make the project safer to use, reduce its production cost, and reduce its size. As mentioned in the previous article, the size of the prototype PCB was 100×100 mm. After testing, some components were removed and the layout of the PCB was optimized, hence the size for this version is reduced to 79.5×79.5 mm — that's about 20% less from last time. In **Figure 1**, the new enclosure for the new version of PCB is shown. Along with that, to make the ESP32 Energy Meter safer to use, instead of powering the PCB directly with the mains voltage, now a 220 V-to-12 V Step Down Transformer is required for voltage sampling and also powering the circuit. Adding a transformer does have some drawbacks in terms of phase delays, but safety first! Since we are not looking to measure voltage spikes, or fast sags or surges, but energy, this shouldn't hurt our measurement.

The Updated Schematic Design

We made some upgrades, and now, rather than the ESP32, the ESP32-S3 is on board. This also unlocks more potential for the Energy Meter. The ESP32-S3 offers significant enhancements over the ESP32, including improved processing power, AI and signal processing capabilities, more memory, and better security features. The updated schematic further improves the capability of the Energy Meter along with more functionality. The design references guides from Espressif [2] and other useful internet resources [3...6] were used to integrate the ESP32-S3 into the project. In **Figure 2**, the schematic of the project is shown.

The circuit board layout has been optimized to improve the safety, usability, and efficacy of the ESP32-S3. We've made substantial adjustments by lowering the PCB size for a more compact footprint, moving to transformer-based power for increased safety, and adding versatility with single and three-phase compatibility. The utilization of a more efficient AP63203WU-7 buck converter in place of Hi-Link modules, along with the addition of user-friendly features such as a USB-C connector and a Qwiic connector for expandability, all contributed to the advancement of the project. These improvements build on the ESP32-S3's capabilities, focusing on providing a practical, adaptable, and safer energy monitoring solution.

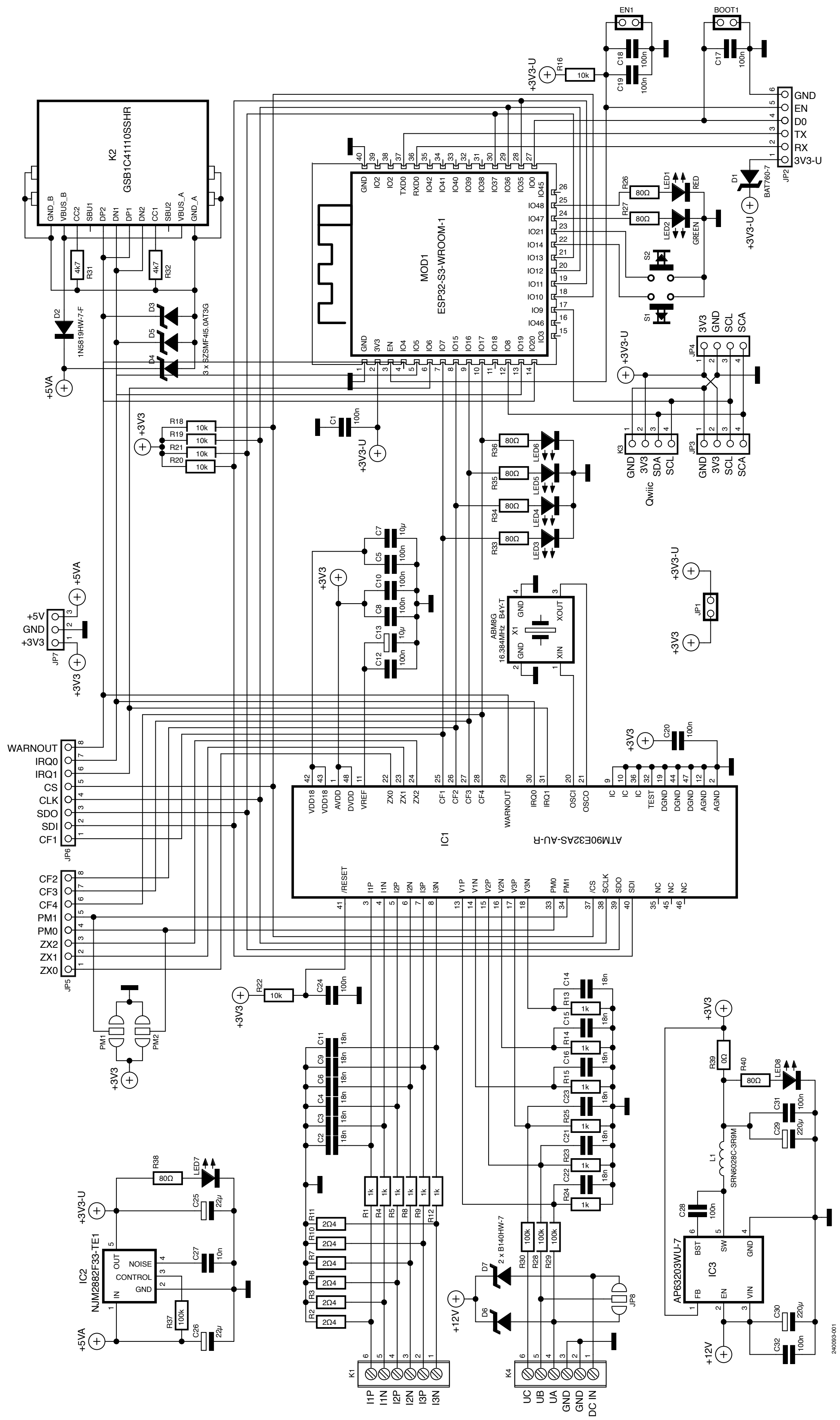


Figure 2: Project schematic.

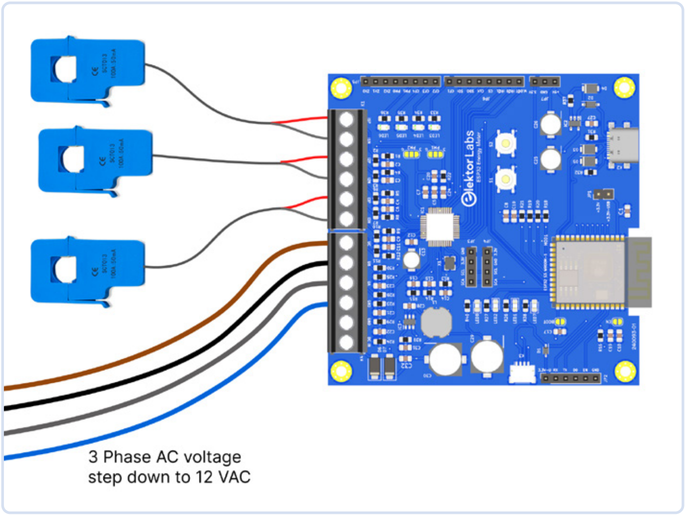


Figure 3: Overall wiring of a three-phase voltage system and coil transformers with the ESP32 Energy meter.

Refined Voltage and Current Sampling

IC1 remains the same ATM90E32AS, but the change is that it now requires a 220 VAC-to-12 VAC step-down transformer, between it and the mains. This change has been made to make the project more safe to test and use, as the transformers provide galvanic isolation. In my testing there was no notable difference in doing so.

Thus, for each of IC1's voltage sampling inputs, there is now only one 100 kΩ resistor (R27 to R29). Last time we combined all the phase voltages into one input, and a lot of feedback was given by the readers to have the option to use it with either three-phase or single-phase power if needed. We thought about it, and now we can use it with both. By default, three-phase mode is configured, but if one wants to make it single-phase, jumper JP8 needs to be shorted. **Figure 3** shows the general wiring illustration for a three-phase system. Note that the phase wires are connected after the step-down to 12 VAC from a transformer — using a 12 VAC doorbell transformer can be useful in this case.

For current sampling and measurement, instead of using the headphone jack as the connector, a 5.08 mm pitch screw terminal block is used, i.e., K1. This adds to the overall ruggedness of the energy meter. For current coil sensors, the YHDC SCT013 100 A : 50 mA is selected and the resistors R1 to R12 for all three current sensing inputs are calibrated accordingly.

Power Supply Optimization

The energy meter is now powered with buck switching regulator IC3, i.e., the AP63203WU-7 by Diodes Incorporated. Previously, Hi-Link HLK5M05 modules were used, but they are much bulkier and more expensive than this buck converter. This is done as buck converters are more efficient than these Hi-Link modules, they cost less, and their size is much smaller. Using IC3 also lets us power the circuit with 12 VDC at K4 for development purposes and also from the UA, i.e., voltage phase 1 of from the same connector, K4, for normal operation.

Interactive and Modular Features

For active, reactive, apparent, active fundamental, and harmonic energy pulse outputs CF1 to CF4, LEDs have been added [7][8]. For the power mode selection of IC1, jumpers PM1 and PM2 are added. In this version, all the output pins of IC1 ATM90E32AS for MCU are provided on terminal JP5 and JP6. This enables the energy meter to be used as a module as well with another MCU if the onboard ESP32-S3 is not required.

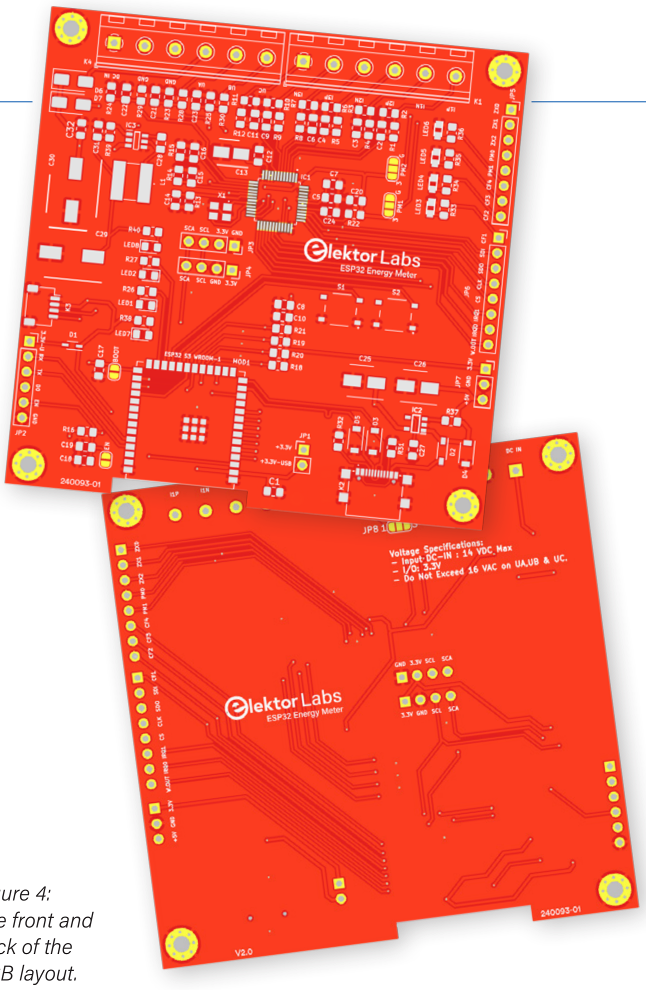


Figure 4: The front and back of the PCB layout.

The ESP32-S3 has the USB feature built in, so it's really convenient to program the MCU this way, which is why we added USB-C connector K2. For troubleshooting, terminal JP2 has been added. Status LEDs LED1 and LED2, which can be controlled by the ESP32-S3, and push buttons S1 and S2 are added for interacting with the OLED screen, which can be connected to JP3 and JP4. Why two connection points? Some I^{2}C OLED screens have ground as the first pin and some have 3V3 supply instead. This way, both types of OLED pinout variants can be worked with.

Finally, the Qwiic connector at K3 has also been added to enhance the functionality of the Energy Meter, in case one wishes to add some additional sensors or modules to this project.

The PCB Layout

The PCB layout has been meticulously optimized for compactness, and easy soldering, shown in **Figure 4**. At the top, voltage and current sampling connections are strategically positioned in one place for DIN rail format integration. On the right side, connections for any external microcontroller (MCU) are facilitated through 2.54 mm pitch headers, ensuring ease of access and modularity. Centrally located is the connection for the OLED screen, flanked by push buttons for intuitive interaction. Adjacent to the OLED display, power and status LEDs provide immediate visual feedback, while energy pulse output LEDs are conveniently situated near the MCU output terminals for direct monitoring.

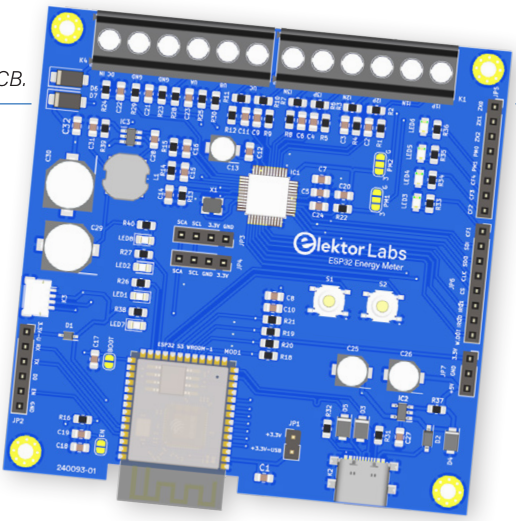


Figure 5: 3D model of the assembled PCB.

At the foundation of the design, the USB-C port and the ESP32-S3 module are positioned away from the AC voltage areas to improve safety. A ceramic capacitor, placed adjacent to the 3 V input of the ESP32-S3, serves to decouple and significantly reduce any potential noise. Additionally, electrolytic capacitors are incorporated into the design, further stabilizing the power supply and ensuring the circuit's reliability and performance. This layout streamlines the assembly process and enhances functionality and user experience by providing a clear and logical component arrangement. In **Figure 5**, you can see the rendering of the assembled PCB.

This design requires the use of mains-powered transformers. People inexperienced with mains voltages should not attempt this project or should ask someone with experience who can help them with this project!

Next Steps and Prospects

Following the prototype phase with the original schematic, we've made several enhancements to increase the reliability of the ESP32 Energy Meter. Currently, we are also focusing on the further development of its firmware.

The latest PCB design has been dispatched for production, and we anticipate conducting extensive tests upon its receipt to ensure system reliability. Concurrently, software development is progressing, aimed at maximizing the capabilities of the ESP32-S3 module within our energy meter.

Looking forward, we plan to integrate the ESP32 Energy Meter with Home Assistant, aiming for simplified user engagement. Nevertheless, we are equally committed to developing bespoke firmware to fully utilize the device's potential.

In summary, the project is moving forward with both hardware improvements and software advancements. Our goal remains to provide a dependable and efficient energy metering solution. This project is also on the Elektor Labs platform [9], so feel free to comment and contribute there!

240093-01

Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.

About the Author

Saad Imtiaz (Senior Engineer, Elektor) is a mechatronics engineer with experience in embedded systems, mechatronic systems, and product development. He has collaborated with numerous companies, ranging from startups to enterprises globally, on prototyping and development. Saad has also spent time in the aviation industry and has led a technology startup company. At Elektor, he drives project development in both software and hardware.

WEB LINKS

[1] Saad Imtiaz, "Project Update: ESP32-Based Energy Meter," Elektor 1/2024: https://elektormagazine.com/magazine/elektor-324/62641
[2] ESP32 S3 DevKit-C Schematic: https://dl.espressif.com/dl/schematics/SCH_ESP32-S3-DevKitC-1_V1.1_20221130.pdf
[3] ESP32 S3 Pinout Help Guide: https://luisllamas.es/en/which-pins-can-i-use-on-esp32-s3
[4] SCH_ESP32-S3-USB-Bridge-MB_V2.1 Schematic: https://tinyurl.com/usbbridgeschematic
[5] ESP32-S3 Pin Reference: http://wiki.fluidnc.com/en/hardware/ESP32-S3_Pin_Reference
[6] ESP32-S3: Which Pins Should I Use?: https://atomic14.com/2023/11/21/esp32-s3-pins.html
[7] Application Note Poly-Phase Energy Metering IC M90E32AS: https://tinyurl.com/polyphasemetering
[8] Atmel M90E32AS | Datasheet: https://eu.mouser.com/datasheet/2/268/Atmel_46003_SE_M90E32AS_Datasheet-1368788.pdf
[9] ESP32 Energy Meter | Elektor Labs:
https://elektormagazine.com/labs/esp32-energy-meter-an-open-source-solution-for-real-time-energy-monitoring

Project Update #3: ESP32-Based Energy Meter

Integration and Testing with Home Assistant

By Saad Imtiaz (Elektor)

In the previous project update, you learned about enhancements to the ESP32 Energy Meter's schematic design and PCB. This article focuses on the practical implementation and integration of the new version. It provides a step-by-step guide on setting up the meter with ESPHome and Home Assistant for effective energy monitoring. Furthermore, we deal with the device's calibration.

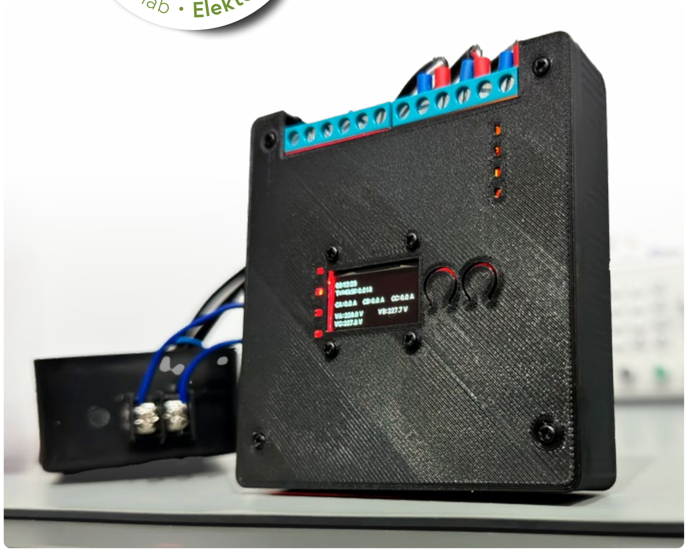


Figure 1: The assembled ESP32 Energy Meter with its OLED display and live status indicators.

In our previous article [1], we focused on enhancements to the schematic design and PCB of the ESP32 Energy Meter, with improvements in modularity and safety features. Before we get into the next project update, let's have a brief overview.

In the most recent advancements of the ESP32 Energy Meter project, we upgraded to the ESP32-S3 microcontroller, introducing enhanced processing power and broader functionality. The new design slimmed down the PCB and incorporated a transformer-based power system, using a 230-V-to-12-V Step Down Transformer for voltage sampling and running of the system. This significantly improves safety while maintaining flexibility for both single- and three-phase installations.

Other improvements included the integration of a more efficient AP63203WU-7 buck converter, making the PCB more modular, calibration of the current transformer sampling circuit, and more. This not only optimizes the energy meter's performance and functionality, but also reduces both cost and size.

In this article, we will discuss the steps taken to get this energy meter up and running and the journey it took from the lab bench to the circuit breaker box. Furthermore, we will also discuss how to set it up, calibrate it, and finally, how to integrate it with Home Assistant with ESP Home to show and monitor the collected data from the energy meter. In **Figure 1**, a vivid snapshot of the ESP32 Energy Meter project in action is captured, encased in a 3D-printed enclosure with an OLED display. The image highlights live status indicators that seamlessly track and display real-time power consumption.

Assembly

The new PCB was designed to be more compact and straightforward to solder and the layout had adequate spacing for each component which accommodated the soldering process of it. To facilitate project replication and modifications by enthusiasts and professionals alike, the complete Bill of Materials (BOM) in Mouser format, and the production files are shared on the Elektor's Lab GitHub repository [2].

For the voltage and current sampling connections, screw type terminal blocks by CUI Devices were used, the quality of these terminal blocks were much better than the cheap blue colored terminal blocks seen on most sensor modules. As we are dealing with AC Voltages and energy metering, it is vital to have secure and reliable connections.

Noise reduction is a critical aspect of the PCB design, addressed by integrating both electrolytic and ceramic capacitors around the ATM90E32S energy metering chip. This arrangement helps to filter out both low- and high-frequency noise, ensuring more accurate and stable energy measurement. The board is depicted in **Figure 2**.

As mentioned earlier, we decided to use a step-down transformer for voltage sampling and the main source of powering the entire system. Finding such a transformer is easy and cheap, but most of these step down transformers take up a lot of space when used in a custom enclosure, as shown in **Figure 3**. So, it is best that DIN Rail Bell transformers are used in this case to make the setup more clean and safe; such transformers can easily be found online. Moreover, the accuracy of voltage measurements depends on the characteristics of the transformers, including their voltage ratio accuracy, phase shift, and linearity.

You might have noticed in the images that there is only one transformer attached to the energy meter. As the energy meter was configured to be used in Single Phase mode, by sorting the jumper JP8 on the back side of the PCB, as seen in **Figure 4.** To operate the energy meter in three-phase mode or to sample voltage from each phase in a three-phase system using three step-down transformers, you must connect the primary sides of three transformers to the respective phases (L1, L2, L3). On the secondary side, connect one end of each transformer's winding to a common neutral point, forming a star (Y) configuration. The free ends of the secondary windings (V1, V2, V3) will then provide the voltage outputs (UA, UB and UC) on the PCB for each phase. Key considerations include ensuring the transformers are properly rated for the system's voltage and current, maintaining strict isolation between primary and secondary circuits for safety, and securing a stable and well-balanced neutral connection to prevent measurement inaccuracies.

Setting up with ESPHome and Home Assistant

As part of the development plan, a specific firmware is being created to leverage the capabilities of the energy metering chip and the advanced AI features of the ESP32-S3. Although developing such tailored firmware requires a significant amount of time and is still underway, this does not restrict the usability of the energy meter. The device can be fully functional with existing platforms like Home Assistant, providing an immediate solution for energy monitoring. Therefore, in this article, the focus is on the integration of the ESP32 Energy Meter with Home Assistant [3] and ESPHome [4]. This section will guide you through setting up the energy meter within the Home Assistant environment to utilize its complete functionality.

To set up the ESP32 Energy Meter with the ESPHome firmware and integrate it into Home Assistant, you should start by installing Home Assistant. (See a comprehensive guide in an article by my colleague Clemens Valens [5].) After that, add the ESPHome integration from the *Add-on Store*; then create a new project in ESPHome for your ESP32 device. This automatically generates a basic YAML configuration file (such a YAML file specifies a distinct ESPHome project with all the sensors used and many other options). You have to download this default file, before taking the next steps.

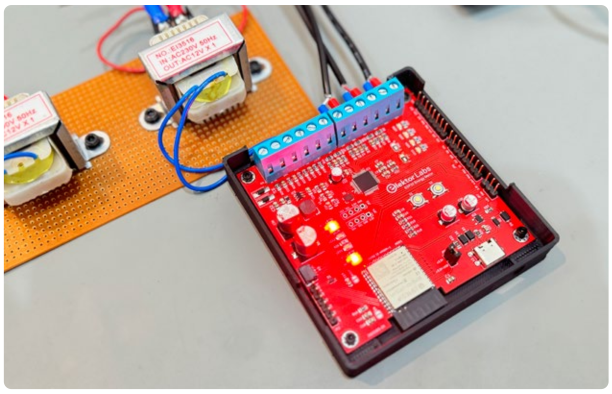


Figure 2: PCB of the fully assembled ESP32 Energy Meter.

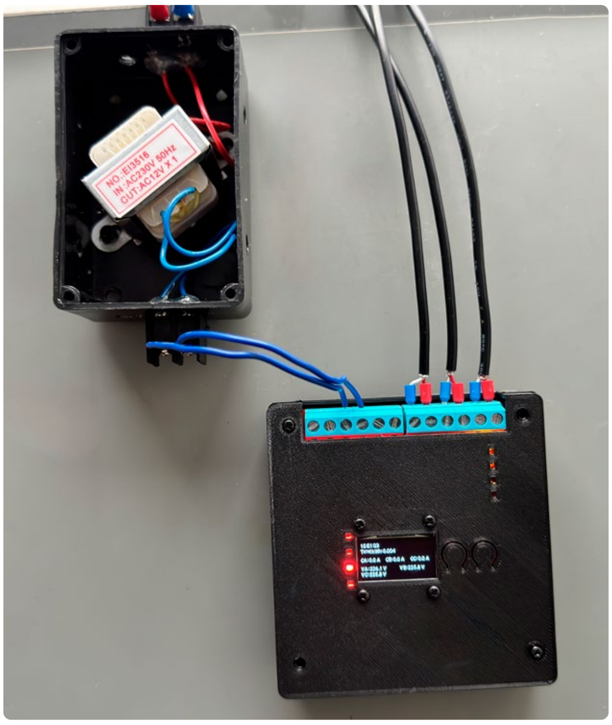


Figure 3: 220 V to 12 V Step-down transformer setup within a custom enclosure.

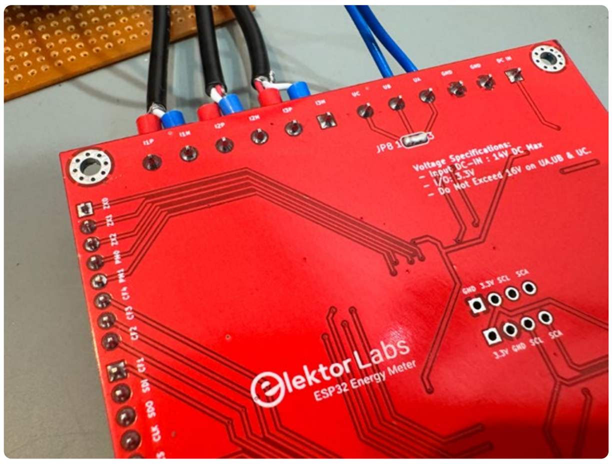
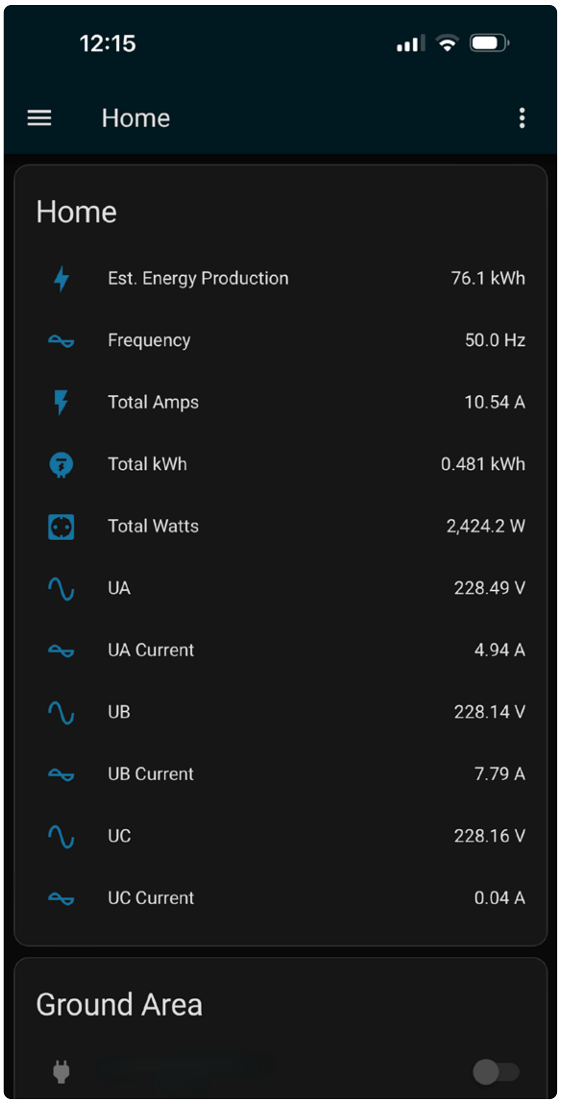


Figure 4: Single-phase jumper configuration on the ESP32 Energy Meter's PCB.

Connect your ESP32 to your computer and select the correct COM Port for the ESP32-S3. In the ESPHome dashboard, click *Install* and choose the *.bin* file to flash the firmware. Once the firmware is successfully uploaded, and your ESP32 Energy Meter is recognized by Home Assistant, proceed to edit the initial YAML configuration. To do this, open the ESPHome dashboard in Home Assistant, find your device, and click on the *Edit* option on the energy meter's card. Replace the existing configuration with the YAML content provided in the GitHub repository [2]. Make sure to properly configure your API, OTA, and WiFi credentials in this new YAML setup.

Install the new configuration wirelessly onto your ESP32 Energy Meter. Once this is done, the device will be active and connected. To display the energy meter data on your Home Assistant dashboard, simply assign the ESPHome device to a specific area in Home Assistant. This helps organize your dashboard by grouping devices according to their physical or logical location in your home. For a visual representation of what you can achieve, refer to the **Figure 5**, which displays the energy meter data on the Home Assistant dashboard.

Integrating the ESP32 Energy Meter with Home Assistant not only simplifies the process of monitoring energy usage but also unlocks a suite of powerful features provided by the platform. Home Assistant offers an intuitive interface for real-time data visualization, control automation, and seamless integration with other smart devices in your home. This integration allows for the creation of detailed history graphs and analytics within Home Assistant, providing an in-depth look at power consumption patterns over time, as shown in **Figure 6**. These insights enable users to make informed decisions about their energy use, identify potential savings, and optimize their home's energy efficiency.



12:15

Home

Home

Est. Energy Production	76.1 kWh
Frequency	50.0 Hz
Total Amps	10.54 A
Total kWh	0.481 kWh
Total Watts	2,424.2 W
UA	228.49 V
UA Current	4.94 A
UB	228.14 V
UB Current	7.79 A
UC	228.16 V
UC Current	0.04 A

Ground Area

Figure 5: Home Assistant dashboard displaying real-time energy data from the ESP32 Energy Meter.

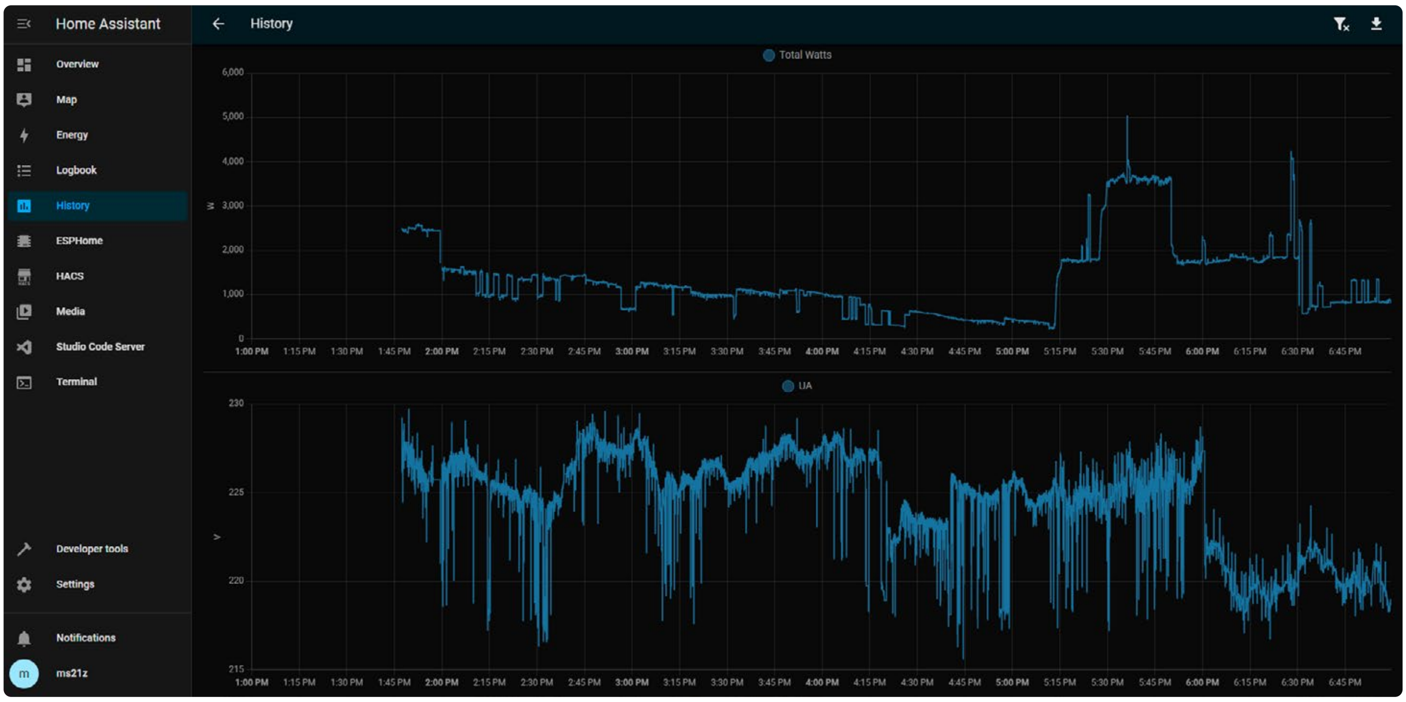


Figure 6: Detailed history graphs of energy consumption in Home Assistant.

By following the setup process described above, users can take full advantage of these capabilities, turning the ESP32 Energy Meter into a central component of their smart home ecosystem. This integration not only enhances the functionality of the energy meter but also enriches the overall smart home experience with comprehensive energy monitoring and management tools.

YAML Configuration

The provided YAML configuration sets up the ESP32 Energy Meter with ESPHome, enabling the monitoring of essential electrical parameters like voltage, current, power across all three phases. It leverages the capabilities of the ATM90E32 sensor, with detailed definitions for SPI communication and individual sensors for each phase. This setup not only measures but also calculates total consumption metrics, integrating a daily energy counter for kWh and an OLED display for real-time data visualization. These configurations are made according to the instructions on the ESPHome page for the ATM90E32 sensor [6].

For ensuring the accuracy of the data reported by the ESP32 Energy Meter, calibration is a crucial step. The specifics of how to adjust the gain settings for current transformers and voltage inputs will be detailed in the upcoming section.

Test Setup and Calibration

The test setup employed a multi-step heat and speed hair dryer as the load, covering a range from 0.7 A to 8 A. The power cord of an extension power strip was stripped to allow placement of the split coil transformers on the live or neutral wire, facilitating direct monitoring under various conditions as shown in **Figure 7**.

I conducted the current and voltage calibration of the ESP32 Energy Meter using my UT201+ multimeter. This offers a resolution of 0.001 A and an accuracy specification of ±4% +10 digits for current and a resolution of 0.001 V with accuracy of ±1% +5 digits for voltage.

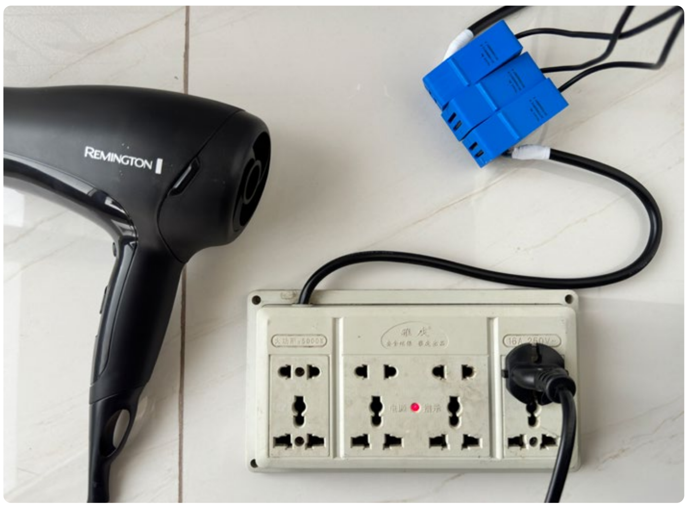


Figure 7: Setup for testing and calibrating the ESP32 Energy Meter using a variable load.

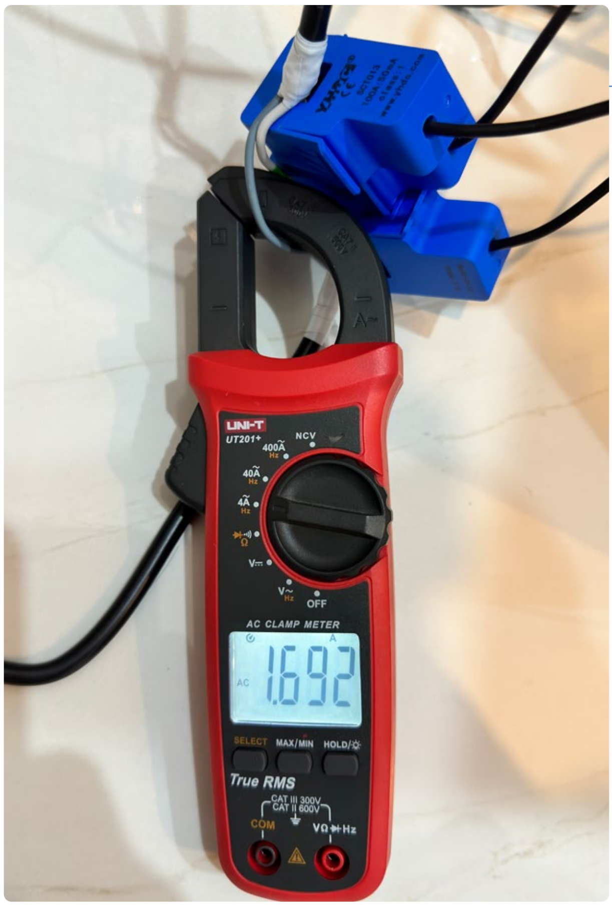


Figure 8: Clamp meter setup for calibration.

This level of precision is adequate for most projects, but is slightly less precise than professional-grade meters.

During the calibration, a comparison of current readings was observed: the clamp meter reading was 1.692 A shown in **Figure 8**, while the readings calculated by the energy meter displayed 1.70 to 1.73 A after calibration, as shown in **Figure 9**. Given the specifications of the UT201+ and the SCT-013-000, a Class 1 split core transformer which guarantees an accuracy within 1% of the actual value, this small discrepancy falls within the expected error margin. However, for even greater accuracy, a more precise clamp meter could be used.

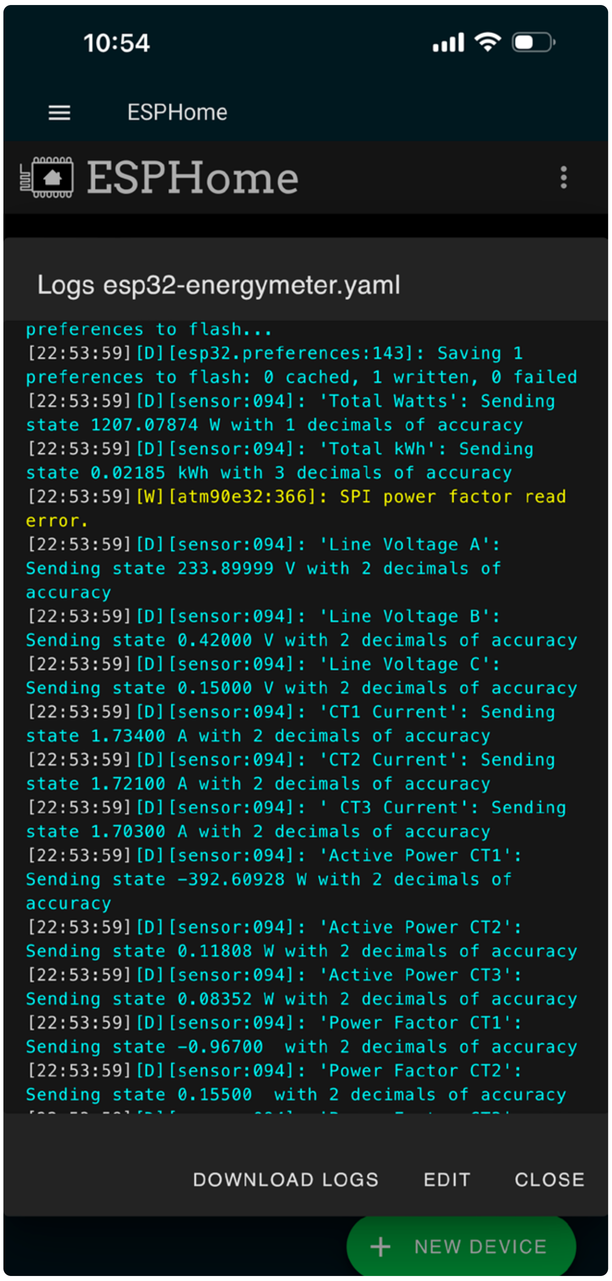
To fine-tune the ESP32 Energy Meter's accuracy further, adjustments were made to the gain settings for both voltage and current measurements. For voltage, the sensor was calibrated using the formula:

*New gain_voltage = (your voltage reading /
ESPHome voltage reading) * existing gain_voltage value*

Similarly, for current adjustments:

*New gain_ct = (your current reading /
ESPHome current reading) * existing gain_ct value*

These new gain values were then updated in the ESPHome YAML configuration file, followed by recompiling and uploading the firmware. This process can be repeated as necessary to ensure optimal accuracy. These calibrated values help refine the measurements and are crucial for accurate reporting and analysis in any energy monitoring setup.


10:54

ESPHome

ESPHome

Logs esp32-energymeter.yaml

```
preferences to flash...
[22:53:59][D][esp32.preferences:143]: Saving 1
preferences to flash: 0 cached, 1 written, 0 failed
[22:53:59][D][sensor:094]: 'Total Watts': Sending
state 1207.07874 W with 1 decimals of accuracy
[22:53:59][D][sensor:094]: 'Total kWh': Sending
state 0.02185 kWh with 3 decimals of accuracy
[22:53:59][W][atm90e32:366]: SPI power factor read
error.
[22:53:59][D][sensor:094]: 'Line Voltage A':
Sending state 233.89999 V with 2 decimals of
accuracy
[22:53:59][D][sensor:094]: 'Line Voltage B':
Sending state 0.42000 V with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'Line Voltage C':
Sending state 0.15000 V with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'CT1 Current': Sending
state 1.73400 A with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'CT2 Current': Sending
state 1.72100 A with 2 decimals of accuracy
[22:53:59][D][sensor:094]: ' CT3 Current': Sending
state 1.70300 A with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'Active Power CT1':
Sending state -392.60928 W with 2 decimals of
accuracy
[22:53:59][D][sensor:094]: 'Active Power CT2':
Sending state 0.11808 W with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'Active Power CT3':
Sending state 0.08352 W with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'Power Factor CT1':
Sending state -0.96700  with 2 decimals of accuracy
[22:53:59][D][sensor:094]: 'Power Factor CT2':
Sending state 0.15500  with 2 decimals of accuracy
```

Figure 9: Final calibration results showing improved measurement accuracy.

WARNING: Working inside a circuit breaker box carries inherent risks, including the potential for electrical shock or fire. It is vital to turn off all power before starting the installation. In most countries, this work may only be carried out by a qualified electrician!

Breaker Box Installation for the ESP32 Meter

Installing the ESP32 Energy Meter into my circuit breaker box proved to be a manageable task that required meticulous attention to detail to ensure both safety and functionality. I started by selecting a circuit with the lowest amp limit. This choice was strategic, as it provided a safety buffer; the circuit breaker would trip in the event of any unexpected surges or transformer failures, thus protecting the system.

Using split core current transformers was particularly beneficial due to their ease of installation. These transformers can be quickly clamped onto any load, but it was crucial to pay attention to the direction of current flow to guarantee accurate readings. It's important to note that if the direction of the current and the orientation of the current transformer are not aligned correctly, the power readings will appear negative, which indicates incorrect installation.

For a visual demonstration of the ESP32 Energy Meter in action within the circuit breaker panel, refer to **Figure 10**. This image shows the energy meter displaying real-time current, voltage measurements, and the corresponding load in kilowatts, illustrating its functionality in a live setting.

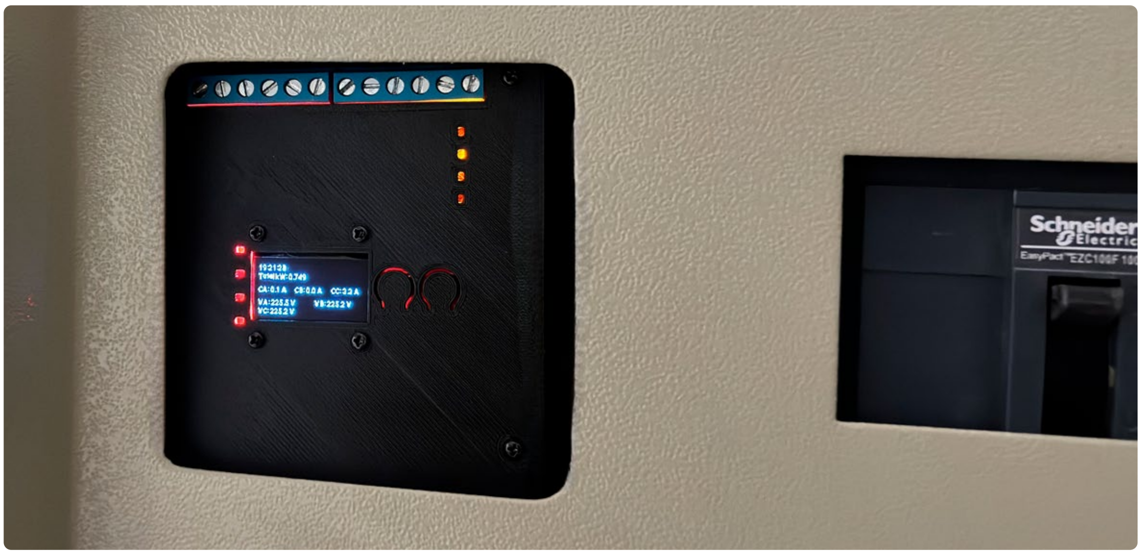


Figure 10: ESP32 Energy Meter installed in a circuit breaker panel, monitoring real-time power consumption.

Development and Prospects

While the current software configuration runs on ESPHome, there is ongoing development to expand the capabilities of the ESP32 Energy Meter. The project is looking forward to being integrated with a new firmware specifically designed to harness the full potential of the ESP32-S3 chip. This future firmware is expected to include advanced features such as detailed energy analytics and potentially groundbreaking AI/ML functionalities that could predict energy usage patterns and identify the device according to its load footprint.

Although the core design and operational aspects of the project have been completed, the development of these sophisticated features is a complex and time-consuming endeavor. I am excited about the possibilities and committed to pushing the boundaries of what this energy meter can achieve.

The ESP32 Energy Meter project is continuously evolving, incorporating more features with each update. Community members who are interested in the upcoming AI and ML functionalities, or those who would like to contribute to the development, are encouraged to get involved. Collaboration will help accelerate progress and result in a more robust and feature-rich energy monitoring solution. Keep an eye out for further advancements as the project aims to refine and elevate this versatile energy management tool to new heights.

240244-01

Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.

About the Author

Saad Imtiaz (Senior Engineer, Elektor) is a mechatronics engineer with experience in embedded systems, mechatronic systems, and product development. He has collaborated with numerous companies, ranging from startups to enterprises globally, on prototyping and development. Saad has also spent time in the aviation industry and has led a technology startup company. At Elektor, he drives project development in both software and hardware.

Related Products

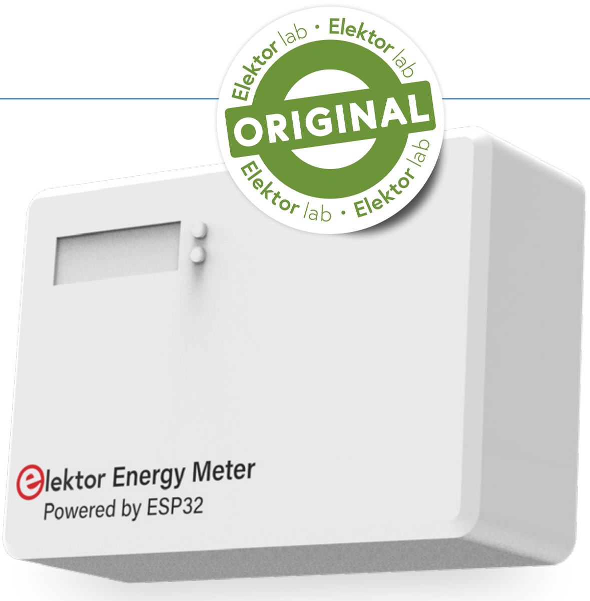
- **PeakTech 4350 Clamp Meter**
  www.elektor.com/18161
- **Siglent SDM3045X Multimeter**
  www.elektor.com/17892

WEB LINKS

[1] Saad Imtiaz, "Project Update #2: ESP32-Based Energy Meter", Elektor 5-6/2024 : https://www.elektormagazine.com/magazine/elektor-341/62892
[2] ESP32 Energy Meter Github Repository: https://github.com/ElektorLabs/esp32-energymeter
[3] Home Assistant: https://home-assistant.io/
[4] ESPHome: http://esphome.io
[5] Clemens Valens, "Home Automation Made Easy," Elektor Magazine 9-10/2020 : https://www.elektormagazine.com/200019-01
[6] ATM90E32 Power Sensor: https://esphome.io/components/sensor/atm90e32.html

Project Update #4 ESP32-Based Energy Meter

Energy Monitoring with MQTT



By Saad Imtiaz (Elektor)

Previously, we focused on setting up the ESP32-Based Energy Meter and integrating it with Home Assistant. We also discussed the future potential of leveraging the ESP32-S3 chip for AI and ML functionality to predict energy usage patterns and identify devices. In this update, we take a step forward by introducing firmware that enables real-time energy monitoring using MQTT, paving the way for advanced features.

We previously discussed the entire journey of the ESP32 Energy meter from the lab bench to the circuit breaker box, which started by assembling components, setting up the energy meter with ESPHome and Home Assistant, calibration and testing, and finally installing it into the circuit breaker box. In the last update [1], we laid the foundation by integrating the meter with Home Assistant, with the ambitious goal of adding AI and ML capabilities to predict energy usage patterns and identify devices based on their energy signatures.

While the AI/ML integration is still in progress — given the extensive data preparation involved — this minor update focuses on a crucial interim development: enabling real-time energy monitoring using MQTT. MQTT is a lightweight messaging protocol designed for efficient communication. Refer to the textbox "**What Is MQTT?**" for more information.

Custom Firmware and MQTT

In this article, we will discuss the next phase of the project — leveraging MQTT and the Arduino IDE to enable real-time energy monitoring. This update will cover the firmware development that allows the ESP32 to communicate with an MQTT broker, sending energy data to a Home Assistant server or any other MQTT-compatible platform.

The advantage of using MQTT with an individual firmware over ESPHome is that the individual firmware offers much greater flexibility in terms of integration and customization. With a custom firmware, you have full control over how the data is collected, processed, and transmitted,

What Is MQTT ?

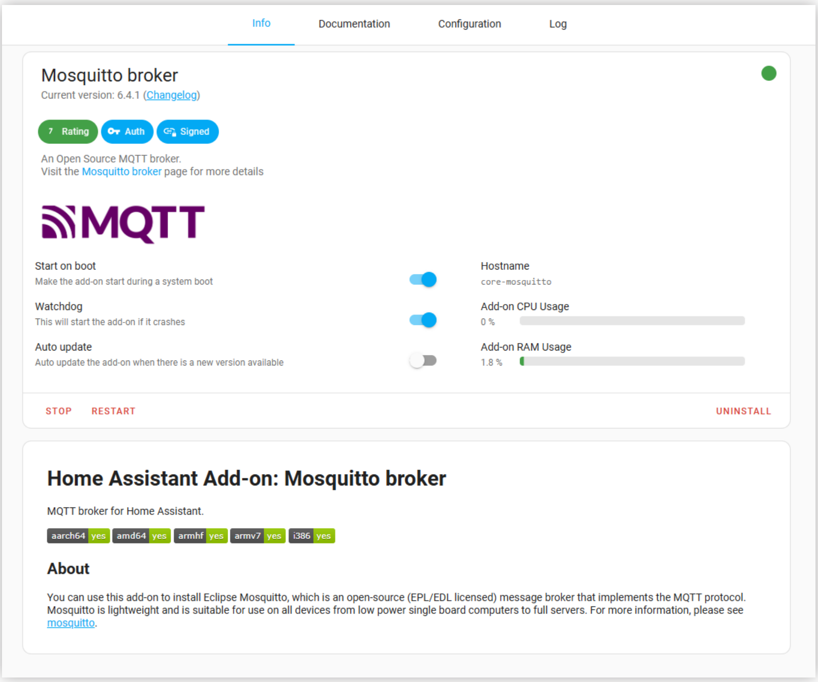
MQTT is a lightweight messaging protocol designed for efficient communication between devices, especially in IoT environments. Central to this system is the MQTT broker, a server that acts as a hub for message exchange. The broker receives messages from devices, known as publishers, and routes them to the appropriate recipients, known as subscribers, based on a system of topics.

A topic in MQTT is a string that categorizes messages, acting as a channel where information is published, while a subscriber is a device or application that listens to specific topics to receive those messages. For example, in a smart home setup, a topic like `home/energy/voltage` might carry voltage readings, and a dashboard subscribing to this topic would receive and display those readings in real-time.

The broker ensures that messages are delivered efficiently and securely, even over unreliable networks. In IoT applications, the MQTT broker is crucial for managing data exchange between sensors, devices, and systems (which are the MQTT clients), enabling real-time monitoring, control, and automation.

Listing 1: The firmware (excerpt).

```
#include <WiFi.h>
#include <SPI.h>
#include <ATM90E32.h>
#include <MQTTPubSubClient.h>
#include <config.ino> // Include configuration file for WiFi and MQTT details

// WiFi Credentials
const char* ssid = WIFISSID;       // Your WiFi SSID
const char* pass = WIFIPASSWORD;   // Your WiFi Password

WiFiClient client;
MQTTPubSubClient mqtt;

ATM90E32 energymeter{};

void setup() {

...

  /* Initialize the ATM90E32 energy meter with the specified parameters */
  energymeter.begin(CS_PIN, LINEFREQ, PGA_GAIN, VOLTAGE_GAIN, GAIN_CT_A, GAIN_CT_B, GAIN_CT_C);

...

  /* Begin the WiFi connection using the provided SSID and password */
  WiFi.begin(ssid, pass);

  /* Initialize the MQTT client */
  mqtt.begin(client);

  /* Connect to WiFi, MQTT broker, and Home Assistant */
  connect();

...
 }

void loop() {
  /* Keep the MQTT client updated */
  mqtt.update();

  /* Reconnect to the MQTT broker if the connection is lost */
  if (!mqtt.isConnected()) {
    connect();
  }

  /* Check and send energy data to Home Assistant every 3 seconds */
  static uint32_t prev_ms = millis();
  if (millis() > prev_ms + 3000) {
    prev_ms = millis();
    getEnergyData(); // Retrieve energy data and send via MQTT
  }
}

void getEnergyData() {

  // Retrieve system status from the ATM90E32
  unsigned short sys0 = energymeter.GetSysStatus0();   //EMMState0
  unsigned short sys1 = energymeter.GetSysStatus1();   //EMMState1
  unsigned short en0 = energymeter.GetMeterStatus0();  //EMMIntState0
  unsigned short en1 = energymeter.GetMeterStatus1();  //EMMIntState1

  // Print system and meter status for debugging
  Serial.println("Sys Status: S0:0x" + String(sys0, HEX) + " S1:0x" + String(sys1, HEX));
  Serial.println("Meter Status: E0:0x" + String(en0, HEX) + " E1:0x" + String(en1, HEX));
  delay(10);

  // Check if the MCU is not receiving data from the energy meter
  if (sys0 == 65535 || sys0 == 0) Serial.println("Error: Not receiving data
                                                  from energy meter - check your connections");

  // Retrieve all parameters from the ATM90E32
  float lineVoltageA = energymeter.GetLineVoltageA();
  float lineVoltageB = energymeter.GetLineVoltageB();
  float lineVoltageC = energymeter.GetLineVoltageC();
       ...

  // Send all the collected energy data via MQTT to Home Assistant
  mqtt.publish("esp32energymeter/lineCurrentA", String(lineCurrentA).c_str());
  mqtt.publish("esp32energymeter/lineCurrentB", String(lineCurrentB).c_str());
  mqtt.publish("esp32energymeter/lineCurrentC", String(lineCurrentC).c_str());
  mqtt.publish("esp32energymeter/totalCurrent", String(totalCurrent).c_str());

...
}
```



Info | Documentation | Configuration | Log

Mosquitto broker
Current version: 6.4.1 (Changelog)

7 Rating | Auth | Signed

An Open Source MQTT broker.
Visit the Mosquitto broker page for more details

Start on boot Make the add-on start during a system boot	on	Hostname `core-mosquitto`
Watchdog This will start the add-on if it crashes	on	Add-on CPU Usage 0 %
Auto update Auto update the add-on when there is a new version available	off	Add-on RAM Usage 1.8 %

STOP | RESTART | UNINSTALL

Home Assistant Add-on: Mosquitto broker

MQTT broker for Home Assistant.

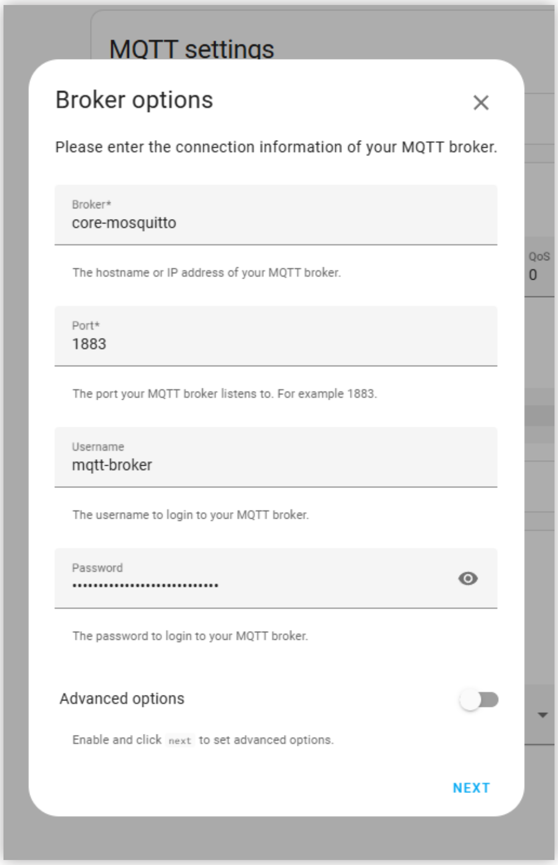
aarch64 yes | amd64 yes | armhf yes | armv7 yes | i386 yes

About

You can use this add-on to install Eclipse Mosquitto, which is an open-source (EPL/EDL licensed) message broker that implements the MQTT protocol. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers. For more information, please see mosquitto.

▲

Figure 1: Configuration options for the MQTT broker in Home Assistant, including the Start on boot option to ensure automatic startup.



MQTT settings

Broker options

Please enter the connection information of your MQTT broker.

Broker*
core-mosquitto
The hostname or IP address of your MQTT broker.

Port*
1883
The port your MQTT broker listens to. For example 1883.

Username
mqtt-broker
The username to login to your MQTT broker.

Password
••••••••••••••••••••••••••••••
The password to login to your MQTT broker.

Advanced options
Enable and click `next` to set advanced options.

NEXT

Figure 2: Setting up MQTT integration in Home Assistant with the IP address, port, and user credentials.

▶

allowing you to tailor the system to meet specific project requirements. This level of control is particularly beneficial for complex applications where you need to optimize performance, integrate with non-standard hardware, or implement advanced features like AI and machine learning algorithms.

Additionally, custom firmware allows for easier integration with a wide range of platforms beyond just Home Assistant, such as cloud-based services, custom dashboards, and other IoT systems. You can also implement more granular security measures, such as custom encryption protocols or advanced authentication mechanisms, ensuring that your data is secure across the network. Furthermore, individual firmware can be optimized for specific use cases, reducing overhead and improving system efficiency, which is especially important in resource-constrained environments.

The Software

The firmware is written to connect the ESP32 to a Wi-Fi network and utilize MQTT for communication, allowing the energy data to be sent to a Home Assistant server for monitoring and automation purposes. In **Listing 1**, you can see an excerpt version of the code, as the entire code and all hardware files can be accessed on the GitHub repository of this project [2].

The project relies on two key libraries: the *ATM90E32* library, provided by CircuitSetup [3], which handles the communication with the energy metering IC, and the *MQTTPubSubClient* library [4], which manages the communication as an MQTT client. The *ATM90E32* library is essential for collecting data from the energy meter chip, including metrics like voltage, current, and power. During the development process, a significant challenge was encountered when integrating MQTT with username and password authentication. While many MQTT libraries can handle basic tasks, *MQTTPubSubClient* stood out as one of the few that was compatible with the ESP32 and supported the necessary authentication features.

The software begins by including the necessary libraries for network connectivity, SPI communication, interfacing with the energy meter IC, and managing MQTT communication. Configuration details for Wi-Fi and MQTT are stored in a separate configuration file. The `setup()` function initializes the serial port for debugging, sets up the ATM90E32 energy meter with the specified parameters, and establishes connections to the Wi-Fi network and MQTT broker. This initial setup ensures that the ESP32 is ready to communicate with Home Assistant and other MQTT-compatible platforms.

In the main `loop()`, the MQTT client is continuously updated to maintain the connection with the broker. If

Defining the MQTT Data as Sensors in Home Assistant

To monitor the data sent by your ESP32 Energy Meter via MQTT, you need to define these data points as sensors in Home Assistant. Follow these steps:

Access the Configuration File:
Open your Home Assistant configuration file (*configuration.yaml*) using the File Editor or any text editor.

Define MQTT Sensors:
In the *configuration.yaml* file, add the following configuration to define your MQTT sensors:

```
mqtt:
  sensor:
    - name: Line Voltage A
      unique_id: esp32_voltage_a
      state_topic: "esp32energymeter/lineVoltageA"
      unit_of_measurement: "V"

    - name: Line Current A
      unique_id: esp32_current_a
      state_topic: "esp32energymeter/lineCurrentA"
      unit_of_measurement: "A"
```

Customize Your Sensors:
Replace `Line Voltage A`, `Line Current A`, etc., with names that suit your needs.
Ensure the `state_topic` matches the topic used in your ESP32 firmware for publishing the data. The `unique_id` should be a unique identifier for each sensor, allowing Home Assistant to track and manage them properly.

Save and Restart Home Assistant:
After adding the sensor definitions, save the *configuration.yaml* file and restart Home Assistant to apply the changes.

View Your Sensors:
Once Home Assistant restarts, your MQTT sensors should appear in the dashboard, allowing you to monitor the energy data in real-time.

the connection is lost at any point, the code automatically attempts to reconnect. Additionally, the software is programmed to retrieve energy data from the ATM90E32 chip and send this data to the MQTT broker. This setup enables Home Assistant to monitor the energy consumption in near real-time, providing valuable insights for home automation.

The `getEnergyData()` function plays a crucial role in the software by collecting various energy metrics from the ATM90E32 chip. These metrics include line voltage, current, power (active, reactive, and apparent), power factor, phase angle, frequency, and temperature. The collected data is then published to specific MQTT topics, making it accessible for monitoring and analysis within Home Assistant. For developers and users who enable debugging, the energy data is also printed to the Serial Monitor, allowing for easy troubleshooting and validation of the data.

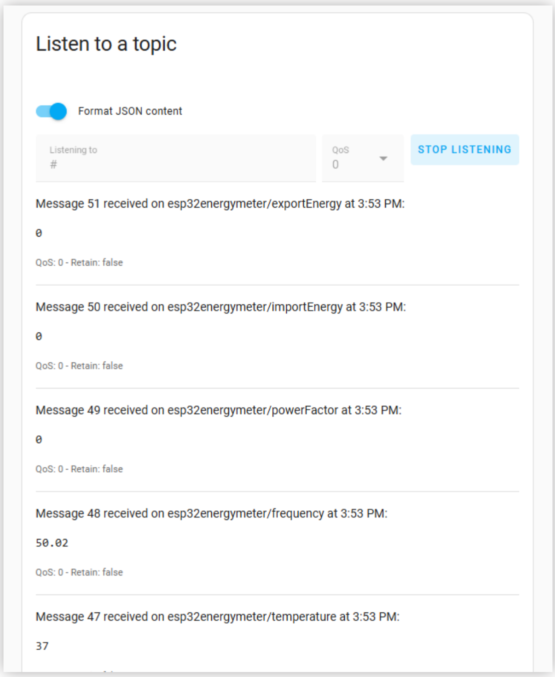
To ensure that the ESP32 maintains a stable connection to the network and the MQTT broker, the `connect()` function is included. This function handles the reconnection process if either the Wi-Fi or MQTT connection is lost. Debugging messages provide real-time feedback on the connection status, and an LED is used to indicate successful connections visually.

Setting Up an MQTT Broker on Home Assistant

To enable the ESP32 Energy Meter to use MQTT protocol to send the meter reading, you'll need to first set up an MQTT broker. An MQTT broker can be set up on almost any computer connected to your home network. It can be configured on your PC using a Docker container, installed directly on a Raspberry Pi, or even hosted on a cloud server for remote access. However, to keep things simple and integrate seamlessly with your smart home setup, we're going to set it up on Home Assistant.

To install the MQTT Add-on in Home Assistant, start by accessing your Home Assistant dashboard by navigating to the URL where your instance is running. From the sidebar, go to *Settings*, then *Add-ons*, and open the *Add-on Store*. Search for *MQTT*, where you should find the official Mosquitto broker in the results. Click on it

Figure 3: Testing the MQTT connection by subscribing to all topics to ensure messages are being received by Home Assistant.


Listen to a topic

Format JSON content

Listening to
#

QoS
0

STOP LISTENING

Message 51 received on esp32energymeter/exportEnergy at 3:53 PM:

```
0
```

QoS: 0 - Retain: false

Message 50 received on esp32energymeter/importEnergy at 3:53 PM:

```
0
```

QoS: 0 - Retain: false

Message 49 received on esp32energymeter/powerFactor at 3:53 PM:

```
0
```

QoS: 0 - Retain: false

Message 48 received on esp32energymeter/frequency at 3:53 PM:

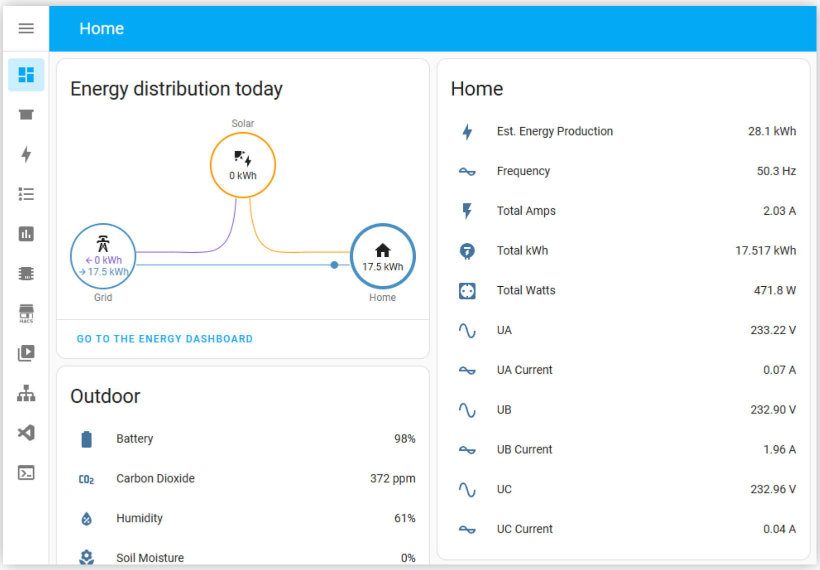
```
50.02
```

QoS: 0 - Retain: false

Message 47 received on esp32energymeter/temperature at 3:53 PM:

```
37
```

Figure 4: Custom dashboard in Home Assistant, visualizing real-time energy data from the ESP32 Energy Meter.



Outdoor

Battery	98%
Carbon Dioxide	372 ppm
Humidity	61%
Soil Moisture	0%

Home

Est. Energy Production	28.1 kWh
Frequency	50.3 Hz
Total Amps	2.03 A
Total kWh	17.517 kWh
Total Watts	471.8 W
UA	233.22 V
UA Current	0.07 A
UB	232.90 V
UB Current	1.96 A
UC	232.96 V
UC Current	0.04 A

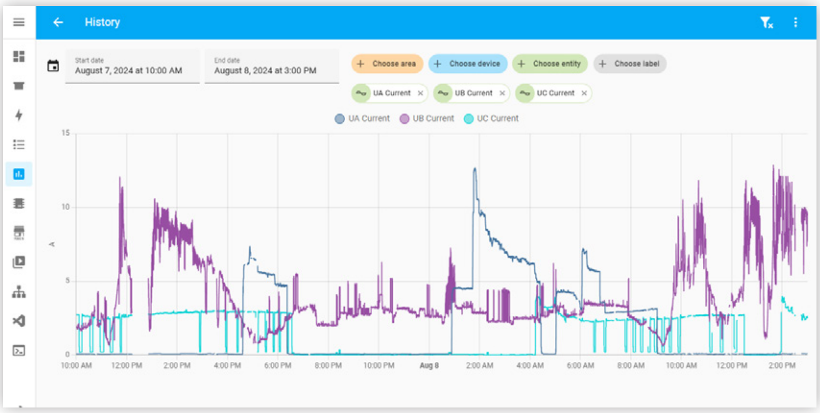


Figure 5: Example of graphs created in Home Assistant to monitor energy consumption in the History dashboard.

and then select *Install*; the installation may take a few moments. Once installed, you can configure the MQTT broker by editing the configuration options — typically, the default settings are sufficient, but you can and should set up a specific username and password. After configuration, click the *Start* button to activate the MQTT broker, and enable the *Start on boot option* to ensure it starts automatically whenever Home Assistant restarts, as shown in **Figure 1**.

Next, you'll need to configure MQTT integration within Home Assistant. Go to *Settings*, then *Devices & Services*, and select *Integrations*. Click on *Add Integration*, search for *MQTT*, and select it. Home Assistant will automatically detect the running MQTT broker. If prompted, configure the MQTT settings, such as the IP address of the broker — usually localhost if it's running on the same device as Home Assistant (you can also enter *core-mosquito* to get the same result). For the port, keep it default *1833*, and for the username as password, make sure that user credentials are of a current user in Home Assistant as shown in **Figure 2**. You can also make a separate user in Home Assistant just for MQTT in this case.

Now, we have to ensure Home Assistant is set up to receive messages from your ESP32 Energy Meter. You can test the connection by subscribing to a topic "#" to receive all the messages sent to your MQTT broker, as shown in **Figure 3.** This page is accessible by clicking *Configure* in *Integration entities* in the MQTT integration menu item.

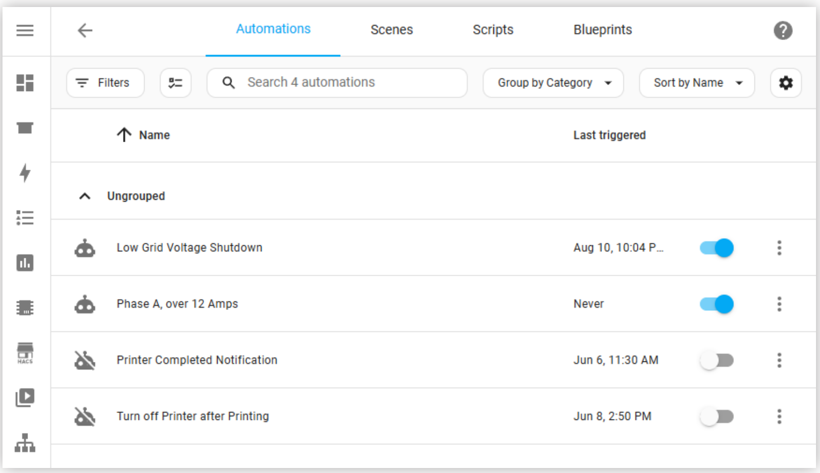
With the MQTT broker running on Home Assistant, you can now connect your ESP32 Energy Meter. In the *config.ino* file of your ESP32 firmware, set the `HOMEASSISTANT_IP` to the IP address of your Home Assistant instance. Then configure the `DEVICE_NAME`, `USER_ID`, and `PASSWORD` if you have set up authentication on the MQTT broker. After configuring these settings, flash the ESP32 with the firmware and ensure it connects to the MQTT broker successfully. Once connected, the ESP32 Energy Meter will begin publishing energy data to the MQTT broker, which you can monitor in Home Assistant by subscribing to the appropriate MQTT topics.

Finally, to visualize energy data in Home Assistant, the MQTT entities will be automatically created for each topic your ESP32 Energy Meter publishes. If not, what can happen sometimes, you will have to define the MQTT entities in the *configurations.yaml* file in Home Assistant. You can follow the instructions mentioned in the text box **Defining the MQTT Data as Sensors in Home Assistant**.

After that, you can find the sensor entities under *Settings* in Home Assistant, then *Devices & Services*, and *Entities*. Use these entities to create custom dashboards in Home Assistant, allowing you to visualize real-time energy data, to create graphs, and to set up alerts based on consumption thresholds as shown in **Figure 4** and **Figure 5**. With MQTT and Home Assistant, you can also automate actions based on energy data as shown in **Figure 6**, integrate with other smart devices, and gain valuable insights into your home's energy usage.

For beginners, I recommend checking out the *Getting Started* Guide by Home Assistant [5], which provides a comprehensive introduction to setting up and using Home Assistant. Additionally, you can explore the Home Assistant MQTT Integration Documentation [6][7] for detailed instructions on configuring MQTT and integrating your devices effectively. These resources will help you get up and running with Home Assistant and MQTT, making your smart home setup more efficient and user-friendly.

240349-01



Name	Last triggered
Ungrouped	
Low Grid Voltage Shutdown	Aug 10, 10:04 P...
Phase A, over 12 Amps	Never
Printer Completed Notification	Jun 6, 11:30 AM
Turn off Printer after Printing	Jun 8, 2:50 PM

Figure 6: Automated actions in Home Assistant based on energy data received from the ESP32 Energy Meter via MQTT.

About the Author

Saad Imtiaz, Senior Engineer at Elektor, is a mechatronics engineer with extensive experience in embedded systems and product development. His journey has seen him collaborate with a diverse array of companies, from innovative startups to established global enterprises, driving forward-thinking prototyping and development projects. With a rich background that includes a stint in the aviation industry and leadership of a technology startup, Saad brings a unique blend of technical expertise and entrepreneurial spirit to his role at Elektor. Here, he contributes to project development in both software and hardware.

Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.

WEB LINKS

[1] Saad Imtiaz, "Project Update #3: ESP32-Based Energy Meter," Elektor 7-8/2024: https://elektormagazine.com/240244-01
[2] ESP32 Energy Meter Github Repository: https://github.com/ElektorLabs/esp32-energymeter
[3] ATM90E32 Arduino Library by CircuitSetup: https://github.com/CircuitSetup/ATM90E32
[4] MQTTPubSubClient Library by hideakitai: https://github.com/hideakitai/MQTTPubSubClient
[5] Getting started, Home Assistant: https://www.home-assistant.io/getting-started/
[6] MQTT Integration, Home Assistant: https://www.home-assistant.io/integrations/mqtt/
[7] MQTT Sensor, Home Assistant : https://www.home-assistant.io/integrations/sensor.mqtt/