DOI: 10.2478/ijasitels-2025-0018

Wireless Controller for Modbus RS485

Mihai-Liviu Stroila[1],

[1]*Department of Computer Science and Electrical Engineering,*
Faculty of Engineering, University "Lucian Blaga" of Sibiu, Sibiu, Romania

Abstract

The purpose of this paper is to propose a modern and efficient solution for controlling and analysing RS-485 communication networks, overcoming the limitations of traditional methods. In high-voltage systems, isolating RS-485 nodes is essential to protect both circuits and human operators from high voltages and transient phenomena. Conventional solutions, such as high-speed optocouplers and transformers, are effective but require numerous external components, resulting in bulky devices with high production costs. The proposed solution introduces a rechargeable, battery-powered control device operated wirelessly via a smartphone or laptop. This ensures complete isolation for the operator while simplifying the system's design.

Compared to traditional galvanic isolation methods, this solution significantly reduces production costs and device size, with the total size being comparable to a computer mouse. It offers a cost-effective, compact, and future-ready alternative for industrial applications requiring efficient communication and safety in high-voltage environments. Additionally, the integration of IoT technology enhances functionality by enabling remote monitoring, real-time control, and data analysis. This IoT-driven approach supports predictive maintenance and scalability, making it ideal for modern industrial and automation systems.

Keywords: RS-485, communication networks, IoT, industrial applications.

1.Proposed method

The concept of the developed application consists of creating a control module primarily used for monitoring an RS-485 serial interface. In its final form, the designed equipment should ideally be capable of operating in multiple modes:

- Sniffer mode, acting only as a passive component in the system, intercepting messages transmitted on the line without responding or influencing communication in any way.
- Controller mode, functioning as the master in the network, with the ability to take over the role of the existing master node and send requests to the network nodes to observe how they respond.

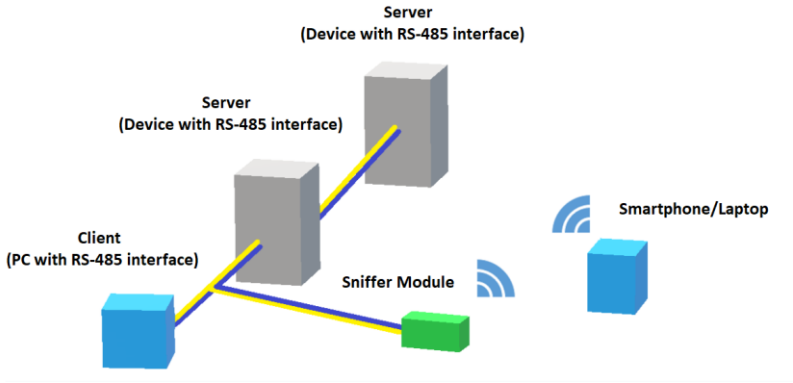

Figure 1 Working Diagra

1.1 The design of the system with a microcontroller from the ESP family.

In order to develop a complex system, the most effective method is to break it down into smaller subsystems with dedicated functions. For each individual block, the desired function is determined, and the components required to achieve this function are selected. In Figure 2 below, the connection method of the component blocks can be observed.

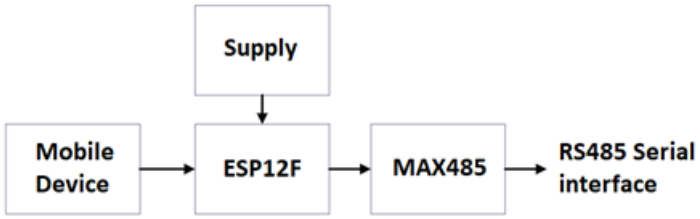

Figure 2 Block Diagram

To establish a connection with the RS-485 serial interface, a driver specifically designed for this interface is required. The MAX485 block consists of this integrated circuit along with the necessary components for its proper functioning, such as capacitors, resistors, and connectors. The data to be transmitted over the interface is received from the microcontroller via a UART-type serial communication.

The ESP12F block serves as the central unit of the entire system and includes the microcontroller of the same name, along with other passive components. This microcontroller runs all the logic required for processing the data that will be sent and received through the serial interface.

The Mobile Device block is represented by a web-based graphical interface that can be accessed through a browser on a mobile device such as a smartphone or any other device with WIFI connectivity and browser access.

The power supply block is divided into two subsystems. The microcontroller is powered by a rechargeable LiPo battery.

- The first subsystem is responsible for regulating the battery voltage to a nominal 3.3V using a linear voltage regulator.
- The second subsystem is in charge of charging the battery. This is achieved using a specialized integrated circuit designed for charging LiPo batteries. The necessary energy for charging is supplied externally via a USB-C connector, which is used exclusively for charging.

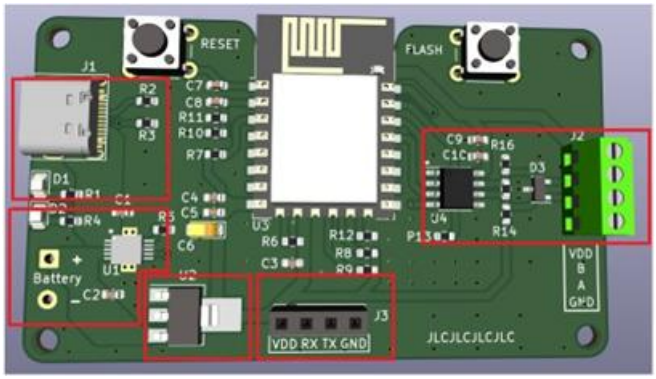

Figure 3 Layout

1.2 Software development of the application

A graphical interface is a means of communication between a user and a system that is meant to be controlled. This allows for easier use of complex systems through various objects such as icons and buttons that replace the old text- or command-based method of communication.

Interaction is carried out via a pointer that acts as a navigator within the interface. The user can enter or select the desired icon or button with a click, which will trigger a series of actions.

The graphical interface used to control the developed application is in the form of a web interface. To create this interface, we used three programming languages: HTML for the structure of the interface, CSS to describe how the elements of the HTML file are displayed and to style the interface, and JavaScript to program the interface's behavior, with its main function being to process requests sent to the backend and display the responses within the interface.

To create the graphical interface, the three corresponding files need to be accessed by the C++ functions. The chosen method was using the LittleFS library.

The files (.html, .css, .js) are stored as data in the microcontroller's flash memory, being flashed separately from the compiled code. These files are accessed using a file access system.

Using specific methods from the LittleFS library, the files are loaded into the microcontroller's RAM. From there, they are retrieved by other functions to initialize the actual web interface.

For the ModBus protocol, the necessary settings are configured using the ModBus RTU library. The choice of this library was explained at the beginning of the section.

The transmission and reception pins are set, along with the baud rate, transmission mode data, and the operating mode, which in this case is master mode.

Once a client is connected, the system waits for a request from the frontend. When a request arrives, it is analyzed and processed by the corresponding backend function, depending on its type.

The communication between front and backend is done using JSON (JavaScript Object Notation). It plays crucial role in this code by serving as the standard format for structuring and transmitting data between the ESP8266 (acting as a backend) and the web client (frontend).

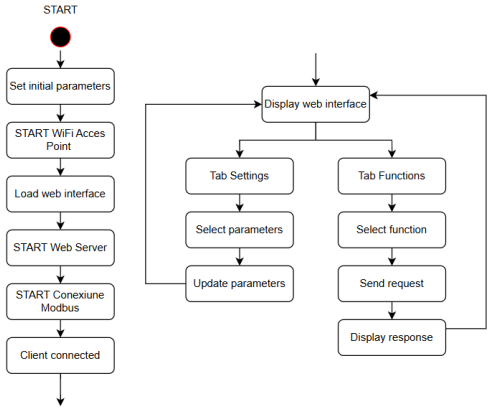

Figure 4 Software workflow

JSON provides a lightweight, human-readable way to organize data into key-value pairs and arrays. In this code, various endpoints (e.g., /settings, /readCoils, /readHolding, etc.) use a global StaticJsonDocument to assemble response data such as transaction statuses, modbus readings, and configuration values. This structure makes it easy for the frontend to parse and understand the response.

The code involves interactions with hardware components via Modbus and then relays the results back to the web interface. By packaging the hardware communication results (like sensor readings, coil states, or error codes) into a JSON object, the data is sent in a format that web clients (typically using JavaScript) can readily process and display. The ArduinoJson library is specifically designed for embedded systems with constrained resources. Its efficient handling of JSON data makes it an ideal choice for an ESP8266 environment where memory and processing power are limited. This allows the microcontroller to handle both Modbus communication and HTTP server tasks without significant overhead.

1.3 Functionality demonstration

To test the project, a system will be used consisting of a device acting as the master (represented by the project itself), an Arduino Uno functioning as a slave, and a device with a dual role—either as a slave or as a monitor—to oversee transmissions on the serial interface using the Windows application ModLink VCL Demo. In Figure 5 below, the block diagram of the system used for the practical demonstration is shown.

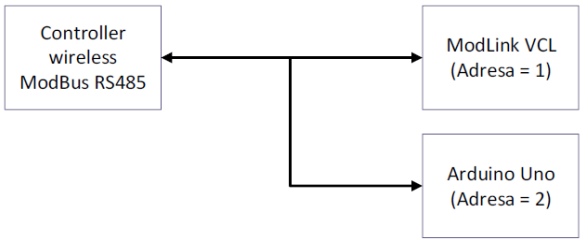

Figure 5 Block diagram of the testing system

To test compatibility with other systems, an Arduino Uno running a ModBus slave code based on the "SimpleModbusSlave.h" library was used, with a slave address set to 2. A TMP36 temperature sensor is connected to the Arduino, monitoring the ambient temperature. The read value is stored in a Holding register at address 0x00.

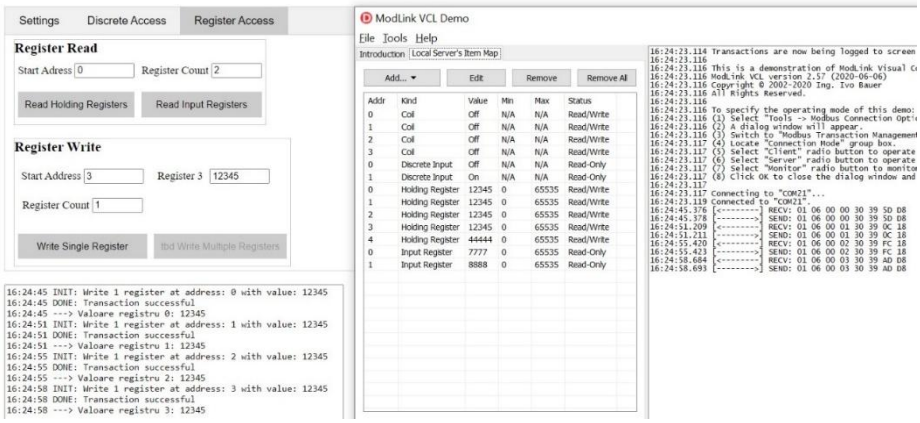

Figure 6 Testing the functionality

2 Comparison with other available solution

In examining the available solutions for establishing wireless connectivity to RS485 servers, it becomes evident that a diverse range of options exists, each designed with specific applications in mind. These alternatives vary in terms of design, performance, and intended use, spanning from simple point-to-point configurations to more sophisticated networked systems optimized for industrial automation, remote monitoring, and control. Each solution addresses the unique challenges of converting

RS485 communication to a wireless medium, ensuring robust and reliable data transfer in different operational environments. Evaluating these options allows for the selection of the most appropriate approach tailored to the particular requirements and constraints of a given project.

	Connectivity	**Power supply**	**Isolation Protection**	**Work mode**
Controller wireless for Modbus RS485	WIFI	5V USBC, 3.7V battery	Power isolation	Web browser interface
Waveshare Rail-Mount Serial Server RS485 to WIFI/Ethernet Module	ethernet, WIFI	5 - 36V DC screw terminal, POE	Power isolation, Signal isolation	Web Browser, MQTT
RS485 to Ethernet converter	ethernet, WIFI	5 - 36V DC	Not mentioned	TCP Server, TCP Client, UDP Server, UDP Client, MQTT Server, MQTT Client
Industrial RS485 IoT Gateway	ethernet, WIFI	6 - 35V DC	Signal isolation	MQTT
Modbus to Lorawan Sensornode for Industrial IoT Application	Lorawan,	USB, Solar Panel, Battery	Not mentioned	MQTT

The alternative projects mentioned above are dedicated to integrating RS485 networks into broader systems for enhanced monitoring capabilities. This approach delivers a permanent, always-connected solution that, once installed becomes a critical component of the overall system architecture. Such a solution is typically intended for continuous, long-term monitoring and control within industrial or building automation environments, where reliability and seamless integration are paramount.

In contrast, my proposed solution is specifically designed as a portable tool for debugging and calibration purposes. This solution is engineered to maximize portability while maintaining a high level of functionality and ease of use. The intent is to provide a flexible, temporary asset that can be deployed quickly for on-site troubleshooting, system verification, and calibration without the need for a fixed installation. Its design emphasizes rapid deployment, allowing technicians to address issues, fine-tune system parameters, and verify network communications directly at the point of need.

Furthermore, the control interface of this portable solution is implemented as a robust web-based platform, accessible via any device equipped with wireless connectivity. This ensures that system diagnostics, updates, and real-time monitoring can be performed from virtually anywhere, significantly enhancing operational efficiency. The web interface is designed to be intuitive and user-friendly, reducing the learning curve for new users and streamlining the maintenance process.

Additionally, the modular design of the proposed solution supports scalability and future enhancements. The system can be easily integrated with other IoT devices, enabling comprehensive monitoring and control across various network segments. This flexibility not only facilitates immediate debugging and calibration tasks but also allows for gradual system upgrades and the incorporation of new functionalities as technological needs evolve.

3 Conclusion

Due to the simplicity and reliability of the protocol—and the fact that it is open source—ModBus remains one of the preferred options in industrial automation and beyond. Almost every industrial automation device with serial communication offers a ModBus option in one of its forms (RTU, ASCII), if not as the standard.

I started with the idea of creating a complete controller, primarily used for monitoring an RS-485 serial interface on which messages are transmitted using the ModBus protocol. Out of the two approaches I considered (a sniffer for intercepting messages, or a controller in which the equipment acts as the master on the network), I managed to implement the latter by finding a solution for implementing a master device on the serial interface. The main objective was to demonstrate that the idea could be implemented, and bringing the project to a functional stage was more important than the number of features and functions the device could perform.

The final result is the successful presentation of a wireless controller for Modbus RS-485, with limited functionality, but the possibility of adding new features through software development. for a few seconds

The final result is the successful demonstration of a wireless controller for Modbus RS-485, with limited functionalities, but with the possibility of adding new functions through software development.

References

[1] Modbus Organisation, Modbus Application Protocol Specification v1.1b3, Modbus.org, 2012.

[2] MODICON, Inc., Industrial Automation Systems, Modbus Protocol - Reference Guide, 1996.

[3] modbus-esp8266, https://github.com/emelianov/modbus-esp8266.

[4] Modlink, https://www.ozm.cz/ivobauer/modlink/introduction.htm.

[5] Espressif Systems Inc, ESP8266EX Datasheet, 2020.

[6] Microchip Technology Inc., Lithium-Ion Battery Charging: Techniques and Trade-offs, 2004.

modbus-esp8266, https://github.com/emelianov/modbus-esp8266.

[7] SimpleModbus NG, https://github.com/angeloc/simplemodbusng.

[8] Rail-Mount Serial Server, RS485 to WIFI/Ethernet Module, Modbus MQTT Gateway https://www.waveshare.com/rs485-to-wifi-eth.htm

[9] Industrial RS485 IoT Gateway https://www.pcbway.com/project/shareproject/Industrial_RS485_IoT_Gateway.html

[10] RS485 to Ethernet converter https://hackaday.io/project/179981-rs485-to-ethernet-converter

[11] Modbus to Lorawan Sensornode for Industrial IoT Application

[12] https://www.hackster.io/dadanugm07/modbus-to-lorawan-sensornode-for-industrial-iot-application-b399e3#toc-hardware-diagram-2