lamaPLC: Simatic and Modbus

Introduction

Certainly, I am aware that numerous descriptions of Modbus can be found online and in technical literature. As the oldest and most widely used industrial communication method, it serves as the backbone of industrial connectivity. While newer, more sophisticated communication protocols have emerged, Modbus remains prevalent. In fact, you might even encounter it on the first intergalactic spacecraft.

Although this communication method is widespread and often underestimated, it can lead to unexpected issues during commissioning, usually more negatively than positively. With over 25 years of experience in automation programming, primarily with Simatic systems, I'm sharing my observations. While the following is somewhat subjective, I hope many readers will have an „aha” or facepalm moment, helping them resolve certain problems.

Modbus Fundamentals

Origin and basics of Modbus

Modbus originated in 1979 and was created by Modicon (now part of Schneider Electric). During this period, industrial automation moved from relay-based systems to digital logic. As the pioneer of the first Programmable Logic Controller (PLC) a decade earlier, Modicon developed Modbus to facilitate communication among these controllers and with external devices via serial lines. The protocol features a straightforward query-response model, in which a *“master”* (client) initiates communication with one or more *“slaves”* (servers) to transfer data.

The protocol's emergence as a worldwide industry standard was fueled by several key factors:

- **Open and Royalty-Free:** Since its inception, Modicon has made the protocol available as an open standard, enabling any manufacturer to implement it without licensing costs.
- **Technical Simplicity:** Its minimal processing requirements and simple message format facilitated its adoption by hundreds of vendors for applications ranging from sensors to motor controllers.
- **Adaptability:** Initially designed for serial interfaces such as RS-232 and RS-485, the protocol has evolved to meet industry needs. In 1999, Modbus TCP was introduced, allowing the original protocol to operate over modern Ethernet and TCP/IP networks.

In 2004, Schneider Electric officially transferred the rights to the Modbus Organization, an independent nonprofit that continues to manage and promote it as a public domain standard. Today, it is often called the “grandfather of industrial networking” due to its continued widespread use in both legacy factories and modern IoT systems.

Core application areas of Modbus

Industrial Automation & Manufacturing

- **Control Systems:** Connecting Programmable Logic Controllers (PLCs) with sensors, actuators, inverters, and motors to automate assembly lines.
- **Data Acquisition:** Using SCADA (Supervisory Control and Data Acquisition) systems to monitor real-time production data, such as oven temperatures, vibration levels, and pressure.
- **Legacy Integration:** Retrofitting older machines to communicate with modern control systems through Modbus-to-IoT gateways.

Smart Buildings & Facility Management

- **HVAC Control:** Managing heating, ventilation, and air conditioning systems based on occupancy and environmental conditions.
- **BMS Integration:** Centralizing data from lighting, security, and elevator systems for improved energy efficiency.
- **Smart Metering:** Connecting Modbus-enabled smart meters to monitor electricity, water, and gas usage across residential or commercial complexes.

Energy Management & Renewables

- **Solar & Wind:** Monitoring Photovoltaic (PV) inverters, trackers, and batteries to optimize energy generation and storage.
- **Electric Vehicles (EV):** Integrating charging infrastructure with building energy systems to manage load and prevent grid strain.
- **Smart Grids:** Enabling real-time communication between grid management systems and remote sensors at substations.

Water & Wastewater Management

- **Process Monitoring:** Automating chemical dosing units, monitoring pump station status (pressure, flow rate), and checking water quality (pH, conductivity).
- **Infrastructure Safety:** Detecting sudden pressure changes to identify pipeline leaks or bursts immediately.

Modbus Client and Server

In the Modbus protocol, the terms Client and Server specify the roles of devices during communication. These terms are the current, official replacements for the older “Master/Slave” terminology.

Modbus Client (formerly Master)

The Client is the active device that initiates all communication transactions. It's sending questions.

- **Action:** It sends a “Request” to a specific device to read or write data to it.
- **Behavior:** It should wait for a response or a timeout before issuing the next command.
- **Typical Devices:** SCADA systems, HMI panels, or a primary PLC.

Modbus Server (formerly Slave)

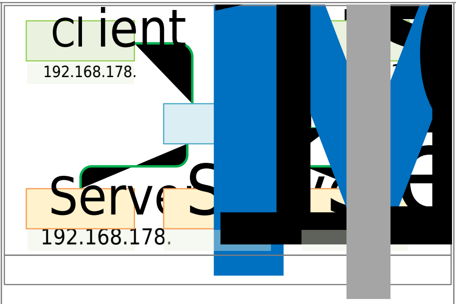
A server is a passive device that responds to client requests.

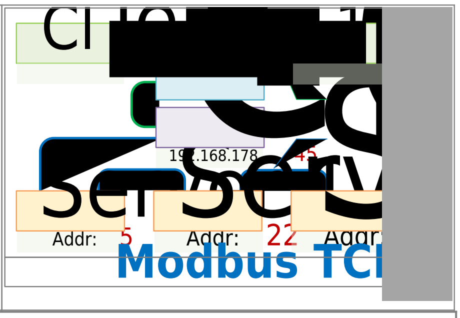
- **Action:** It waits for an incoming message, processes the request (for example, looks up a sensor value), and sends back a *“Response”*.
- **Behavior:** It never initiates a conversation; it only responds when spoken to.
- **Typical Devices:** Sensors such as temperature and humidity sensors, motor drives like VFDs, power meters, or I/O blocks.

Modbus RTU and TCP, and Hybrid

The core difference is the transport layer: Modbus RTU is designed for a physical wire (Serial), while Modbus TCP is designed for a network (Ethernet). Before Ethernet's advent, only the RTU (*Remote Terminal Unit*) protocol was available, primarily using RS-232 or RS-485. The rise of Ethernet greatly expanded options, enabling communication over the faster, more versatile TCP/IP protocol. Today, these two methods serve as the Modbus transport layer.

Although quite different, they can sometimes be integrated, such as in multimaster (hybrid) RTU scenarios. In this scenario, a signal converter needs to be integrated into the TCP network. The converter has its own IP address to receive Modbus/TCP telegrams. It's crucial that the client is aware of the Modbus/RTU network behind the converter, which uses traditional RTU addressing. Often, clients cannot manage both RTU addressing and TCP simultaneously, so verifying their ability to do so is essential. The Siemens TIA Portal is suitable for this task, and I will offer an example of its use later.

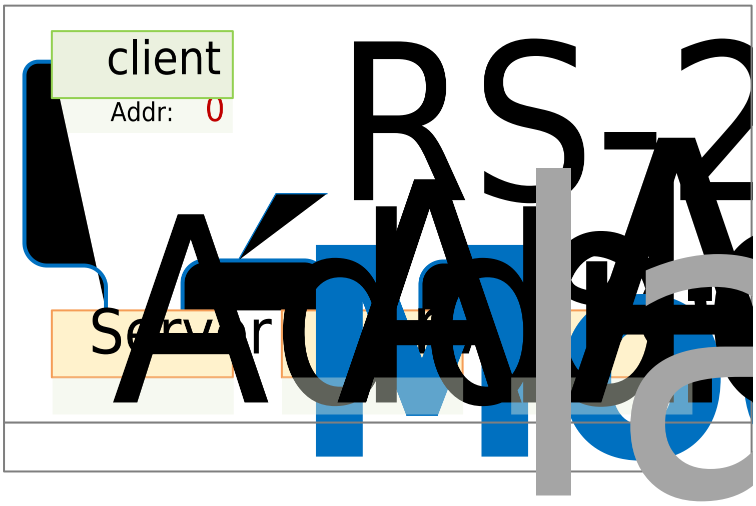




Feature	**Modbus RTU**	**Modbus TCP**
Media	Serial (RS-485/232)	Ethernet / Wi-Fi
Error Check	CRC (at the end)	TCP/IP Checksum (built-in)
Topology	Daisy-chain	Star (Switch-based)
Speed	Typically 9600 or 115200 baud	10/100/1000 Mbps
Master/Client	Only one Master	Multi-Master
Port	Serial COM Port	TCP Port 502
Segment distance	1200 meters	100 meters between switches
Addressing	Master ID: 0, slave ID: 1 to 247	IP Address (like: *192.168.178.123*)
Wiring	3-core cable with shielding or 4-core cable	Ethernet cables (RJ45) and network switches

Modbus/RTU

Modbus/RTU was the first Modbus communication method and remained the main standard until Ethernet became popular. It remains widely used today, partly because its hardware integration is simpler and more affordable than Modbus/TCP. As a result, it is likely to stay available for quite some time.



Comparison of RS-232 and RS-485

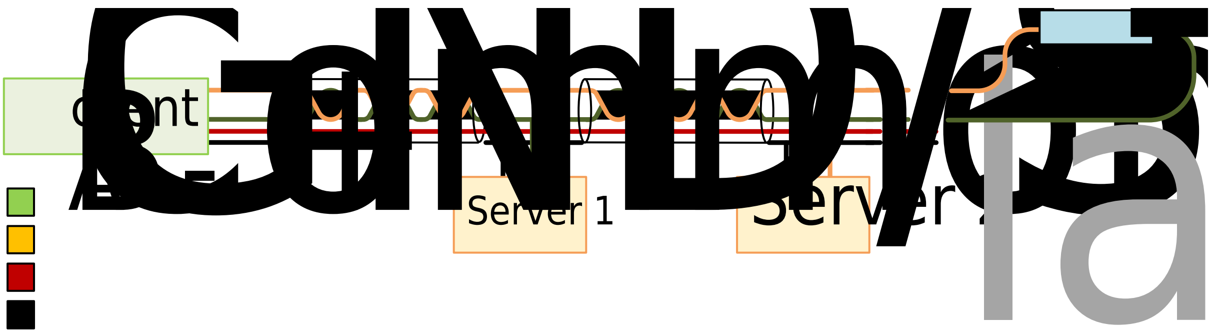
The Modbus/RTU transmission options depend on the physical layer: RS-232 or RS-485. RS-232 is quite uncommon because it only supports point-to-point connections, meaning one Client and one Server. In contrast, RS-485 is a more flexible option; its technical specifications are outlined in the table below.

	RS-232	**RS-485**
Operating mode	asynchronous transmission	synchronous transmission
Number of drives and receivers per line	1 drive 1 receiver (point-to-point)	32 stations per segment

	RS-232	**RS-485**
Data transfer method	half-duplex, full-duplex	half duplex
Data transmission	p2p	multipoint
Max. cable length	15 m	1200 m
Max. data transfer **12 m** **1200 m**	20 kbps (1 kbps)	35 Mbps 100 kbps
Receiver input resistance	3..7 kΩ	≧ 12 kΩ
Drive Load-Impedance	3..7 kΩ	54 Ω
Receiver “dead band”	±3 V	±200 mV
Receiver voltage level	±15 V	-7..+12 V
Drive output voltage max.	±25 V	–9..+14 V
Drive output voltage min. (with load)	±5 V	±1.5 V
Drive output short circuit current limit	500 mA to Vdc or Ground	150 mA to Ground 250 mA to Vdc
Receiver Hysteresis	1.15 V	50 mV

Modbus/RTU wiring

For Modbus/RTU communication, use RS-485 with either a 3-wire with shield or a 4-wire cable. To improve noise immunity, using shielded cables and twisted pairs is recommended. The units should be connected in a daisy-chain layout, though star topology can also work in some cases with a few units.



In a Modbus RTU network over RS-485, it is vital to terminate the bus to prevent signal reflections that can cause communication errors and CRC failures.

Guidelines on Proper Terminator Placement

- **Both Ends Only:** Install a termination resistor at the first and last device of the trunk cable.
- **Do Not Terminate Middle Devices:** Adding resistors at intermediate nodes can overload drivers and cause signal loss.
- **Daisy Chain Topology:** RS-485 should be wired in a continuous line. Avoid star or tree configurations with long branches or stubs, as these cannot be correctly terminated.

Resistor Specifications

- **Value:** Usually **120 Ω** to match the characteristic impedance of standard twisted-pair cable.
- **Power Rating:** Select a resistor with a power rating of at least 0.25W to 0.5W.
- **Connection:** Attach the resistor directly between the A (D0/-) and B (D1/+) data lines.

Modbus/RTU RS-485 Signaling

In a Modbus/RTU network (which uses the RS-485 physical layer), **A** and **B** represent the two wires of a differential pair used to transmit data.

- **Differential Signaling:** The receiver measures the voltage difference between two lines instead of against a common ground. This helps the signal resist electrical noise because interference usually impacts both wires equally.
- **Half-Duplex:** Modbus/RTU uses these two wires for transmission and reception, but only one device can communicate at a time.

The Naming Confusion (A vs. B)

Since there is no universal naming standard, wiring errors are common. Terminal labels vary by manufacturer as follows:

Labeling Convention	Non-Inverting Signal	Inverting Signal
Common/Modbus	B or B+	A or A-
TIA/EIA-485 Standard	A (Negative)	B (Positive)
Alternate Labels	D+, Data+, Tx+	D-, Data-, Tx-

- **Standard Rule:** In the official TIA-485 standard, **A is negative (A-)** and **B is positive (B+)**.
- **Industry Practice:** Many Modbus device manufacturers (such as those following Modbus Organization guidelines) use A for negative (-) and B for positive (+).

Voltage Levels

- **Logic 1 (Idle/Marking):** Occurs when the voltage at B is higher than A by at least 200mV.
- **Logic 0 (Active/Spacing):** Occurs when the voltage at A is higher than B by at least 200mV.
- **Idle State:** When no device is transmitting, the bus is *“idle.”* Biasing resistors are often used to keep the B line slightly more positive than the A line, preventing noise from being misinterpreted as data.

If your devices have the correct baud rate and address but still can't communicate, try swapping the A and B wires on one side. This is a common cause of Modbus RTU setup failure and won't harm your hardware.

Modbus/RTU Speed (Baud Rate)

This is the transmission speed in bits per second (bps).

- **Standard speeds** are typically 9600 and 19200 bps. For quicker data updates over short distances, higher speeds such as 38400, 57600, or 115200 bps are often used.
- **Lower baud rates**, such as 9600, provide greater stability over lengthy cable runs—up to 1200 meters—because they are less affected by signal reflection and noise.
- **Timing Requirement:** Modbus RTU depends on silent intervals (3.5 character times) to mark the end of a frame. If the baud rate is incorrect, devices cannot *“see”* where one message ends and the next begins.
- **Manufacturer specification:** The primary focus should be on the speed setting. If not specified, it is recommended to start at 9600 baud and 8N1.

Modbus/RTU Data Format ("8N1")

Data Format 8N1:

This describes the structure of a single byte (character) sent over the wire. Totaling 10 bits per character:

- **8 (Data Bits):** Modbus RTU consistently employs 8 bits per byte, enabling direct transmission of binary data, whereas Modbus ASCII converts data to text.
- **N (Parity):** No parity bit is employed for error detection at the byte level. Note: While the Modbus standard technically specifies Even Parity as the default, the industry predominantly uses No Parity (8N1).
- **1 (Stop Bit):** A single stop bit indicates the conclusion of the byte.
  - **Constraint:** If *“No Parity”* is selected, the standard recommends using 2 stop bits (8N2) to keep the character length at 11 bits, but most modern devices default to **8N1** regardless.

Modbus/RTU Error Checking (CRC)

Even though *“No Parity”* (**8N1**) lacks bit-level checking, Modbus RTU is still secure because every full packet ends with a 16-bit **CRC** (*Cyclic Redundancy Check*). If a single bit is flipped during transmission due to noise, the CRC will fail, and the receiving device will ignore the command.

The **CRC** (*Cyclic Redundancy Check*) is an error-detection method that ensures data integrity in Modbus RTU. It is a 16-bit (2-byte) value appended to each message.

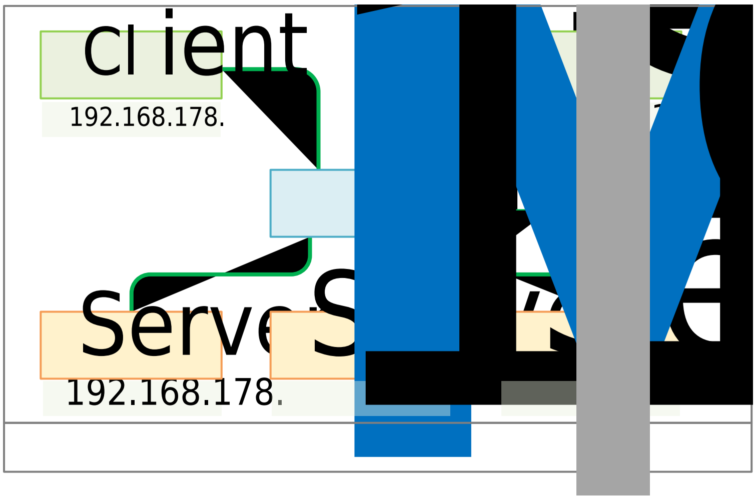
Key Characteristics

- **Algorithm:** Modbus uses the CRC-16-ANSI (also known as **CRC-16-IBM**) polynomial: $X^{16}+X^{15}+X^{2}+1$
- **Verification:** The sender computes the CRC and adds it to the frame. When the receiver gets

the frame, it recalculates the CRC; if the values differ, the receiver discards the packet and does not reply.
- **Efficiency:** It is far more dependable than a basic *“Checksum,”* as it can identify all single and double-bit errors and the majority of burst errors.

Modbus/TCP

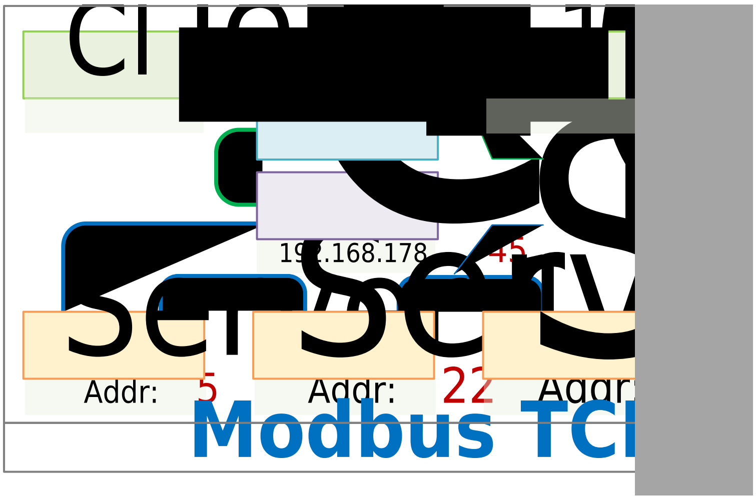
Modbus/TCP (also known as Modbus TCP/IP) is a version of the Modbus protocol designed for Ethernet network communication. It wraps standard Modbus messages into TCP/IP packets, enabling reliable, high-speed data exchange.



Key Characteristics

- **Architecture:** Uses a Client-Server model (previously called Master-Slave). The Client sends requests, and the Server replies with data or action confirmations. Network Port: Default is TCP Port 502.
- **Reliability:** Depends on the TCP/IP stack for error detection and reliable delivery, removing the necessity of **CRC** (*Cyclic Redundancy Check*) used in serial Modbus RTU.
- **Addressing:** Uses IP addresses to identify devices instead of the 1-byte Slave ID used in serial versions.

Modbus/TCP and Modbus/RTU Hybrid



Modbus Registers and Coins

Modbus data is structured into four main *“data banks”* or *“storage units”*, categorized by whether the data is a single bit or a 16-bit word, and whether it is read-only or read-write.

Coils and **Discrete Inputs** are 1-bit variables. While a Coil can be read and written, a Discrete Input is only a read-only indicator of status.

Registers are 16-bit storage units: **Holding registers** are both writable and readable, and **Input registers** are read-only.

Although the Modbus standard allows Holding registers to be both read and written, many manufacturers restrict them to read-only. Always check the manufacturer’s specifications for accurate details.

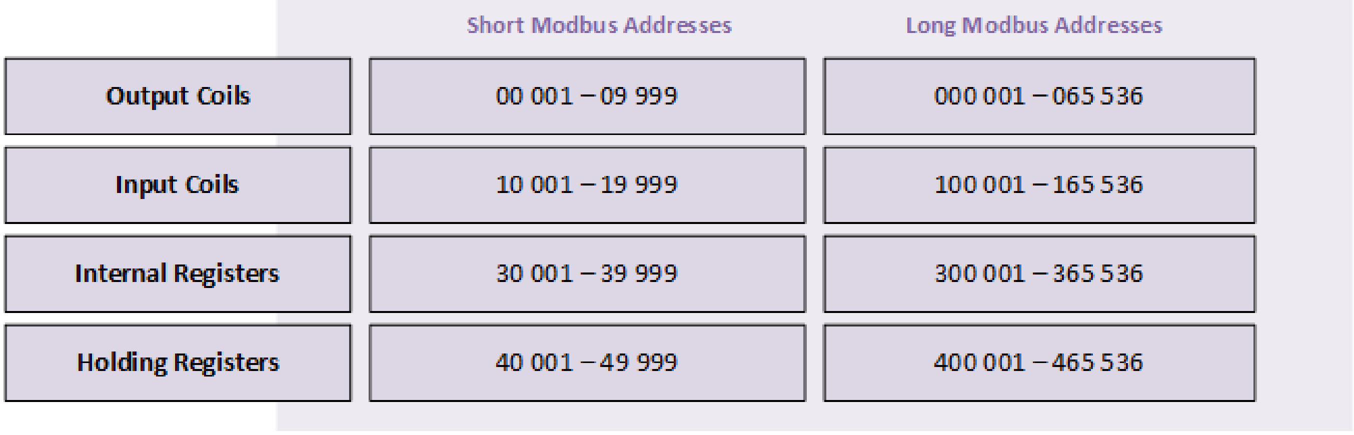
Key Characteristics

- **16-Bit Architecture:** All Modbus registers are 16 bits (2 bytes) in size. To handle larger data types such as 32-bit floating points or integers, systems usually allocate two registers in sequence.
- **Big-Endian Format:** Modbus typically transmits the most significant byte (MSB) first.
- **Zero-Based Addressing:** Although documentation often refers to addresses using “1-based” numbering (e.g., Holding Register 40001), the actual transmission uses 0-based addresses (e.g., Address 0).
- **Manufacturer Flexibility:** Devices are not required to support all four register types. Many modern devices simplify data handling by mapping all information, including status bits, into Holding Registers.
- **Read/Write Permissions:** “Input” types (Discrete Inputs and Input Registers) are strictly for data sent from field devices to the controller and cannot be modified by a Modbus master.

Modbus Coin and Register Addressing

To index Modbus address ranges, a 5-digit address (e.g., 40001) was initially used. Over time, this was insufficient, as it allowed only 9,999 addresses per type. This was expanded to 6-digit addresses (e.g., 400001), offering 65,536 addresses aligned with Word boundaries. The first digit indicates the area type: 0x for Coils, 1x for Discrete Inputs, 3x for Input Registers, and 4x for Holding Registers.

Addressing within each range begins at 1, so the first holding register is 40001 or 400001. Users should note that many manufacturers use hexadecimal addresses, while Modbus/RTU uses decimal addresses. Larger data types (>16 bit, such as REAL, LREAL, DT, or STRING, WSTRING) span multiple registers for a single variable; thus, both the start address (offset, e.g., 400012) and the area length are specified.



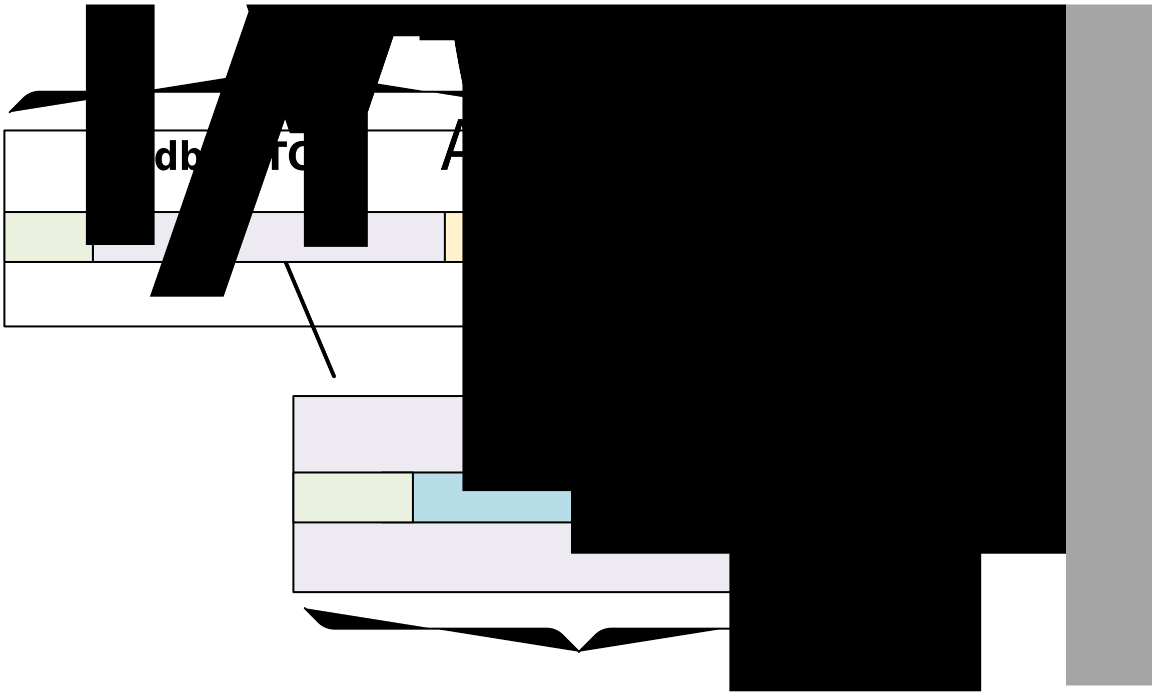
	Short Modbus Addresses	Long Modbus Addresses
Output Coils	00 001 – 09 999	000 001 – 065 536
Input Coils	10 001 – 19 999	100 001 – 165 536
Internal Registers	30 001 – 39 999	300 001 – 365 536
Holding Registers	40 001 – 49 999	400 001 – 465 536

lamaPLC.com / Vamos 2026

Modbus Telegram Structure, ADU/PDU

In Modbus, the Application Data Unit (ADU) represents the complete message frame sent over a physical network. It functions as an *“envelope”* that encloses the Protocol Data Unit (PDU)—the main message with commands and data—while also including addressing and error-checking fields specific to the communication method.

The ADU's structure changes based on whether you're using Modbus RTU (serial) or Modbus TCP (Ethernet):



[redacted] [redacted] [redacted] [redacted]

Modbus/TCP ADU (Ethernet)	**Modbus/RTU** ADU (Serial)

MBAP Header (7 Bytes): - Transaction ID (2 Bytes): Matches requests with responses. - Protocol ID (2 Bytes): Always 0 for Modbus. - Length (2 Bytes): Number of remaining bytes. - Unit ID (1 Byte): Used for routing to serial devices through a gateway. **PDU** (Function Code + Data): The core command. Max Size: 260 bytes.	Slave **Address** (1 Byte): Identifies the target device (1–247). **PDU** (Function Code + Data): The core command. **CRC** (2 Bytes): A Cyclic Redundancy Check used to detect transmission errors. Max Size: 256 bytes.

Error checking by Modbus/TCP is managed by the TCP layer and is not included in the ADU.

Modbus Protocol Data Unit (PDU)

The *Modbus Protocol Data Unit* (**PDU**) is the core message structure common to all Modbus variants (RTU, ASCII, and TCP). It defines the actual command and data being exchanged between a client and a server, independent of the network medium. The PDU consists of two primary fields with a maximum combined size of 253 bytes.

Field	Size	Description
Function Code	1 Byte	Tells the server which action to perform, such as Read, Write, Diagnostic.
Data Field	0–252 Bytes	Contains request details such as register addresses, quantities, or the actual data values being transmitted or returned.

Function Code Types

The function code ranges from 1 to 255 and is categorized by its purpose:

- **Public Codes (1–64, 73–99, 111–127):** Standardized, well-defined commands like **03** (*Read Holding Registers*) or **16** (*Write Multiple Registers*).
- **User-Defined Codes (65–72, 100–110):** Reserved for custom device functions not defined by the standard.
- **Exception Responses (128–255):** When an error occurs, the server returns the original function code with its highest bit set (*Original Code + 0x80*).

Data Field Structure

The structure of the data field changes depending on whether the PDU is part of a Request or a Response:

- In a **Request**, the Starting Address (2 bytes) and the Quantity of items to read or write (2 bytes) are typically included.
- In a **Response**, it presents a Byte Count along with the requested data values.
- **Addressing:** All addresses within the PDU data field are 0-based offsets ranging from 0 to 65.535.

Data Encoding (Endianness)

Modbus uses Big-Endian representation for all 16-bit values within the PDU. This means the *Most*

Significant Byte (**MSB**) is transmitted before the *Least Significant Byte* (**LSB**). *Example:* A register value of 0x1234 is sent as 0x12 followed by 0x34.

Modbus Function Codes

The table below details the standard Modbus function codes, which cover data access, diagnostics, and advanced functions used in both RTU and TCP variants. The first byte in a PDU is the Function Code, indicating the operation that the telegram performs.

Code (Hex)	Code (Dec)	Function Name	Data Type	Access
0x01	01	Read Coils	Bit (0x)	Read
0x02	02	Read Discrete Inputs	Bit (1x)	Read
0x03	03	Read Holding Registers	16-bit (4x)	Read
0x04	04	Read Input Registers	16-bit (3x)	Read
0x05	05	Write Single Coil	Bit (0x)	Write
0x06	06	Write Single Register	16-bit (4x)	Write
0x07	07	Read Exception Status	Serial Only	Read
0x08	08	Diagnostics	Internal	Read
0x0B	11	Get Comm Event Counter	Serial Only	Read
0x0C	12	Get Comm Event Log	Serial Only	Read
0x0F	15	Write Multiple Coils	Bit (0x)	Write
0x10	16	Write Multiple Registers	16-bit (4x)	Write
0x11	17	Report Server ID	Serial Only	Read
0x14	20	Read File Record	File	Read
0x15	21	Write File Record	File	Write
0x16	22	Mask Write Register	16-bit (4x)	Write
0x17	23	Read/Write Multiple Registers	16-bit (4x)	R/W
0x18	24	Read FIFO Queue	16-bit	Read
0x2B	43	Read Device Identification	Internal	Read

Function Code Categories

- **Public Function Codes (1–64, 73–99, 111–127):** These are standard codes established by the Modbus community.
- **User-Defined Codes (65–72, 100–110):** These are designated for custom implementations created by manufacturers.
- **Exception Codes (128–255):** These codes are reserved for error responses. When a server encounters an error, it responds by adding 0x80 to the original function code; for example, a failed Read 0x03 returns 0x83.

Modbus test programs, test methods

Modbus Problems and errors

Simatic and Modbus

Scheme of Simatic

Simatic and Modbus RTU and/or TCP

Modbus Installation examples, step by step

S7-1500 and Easton Energymeter

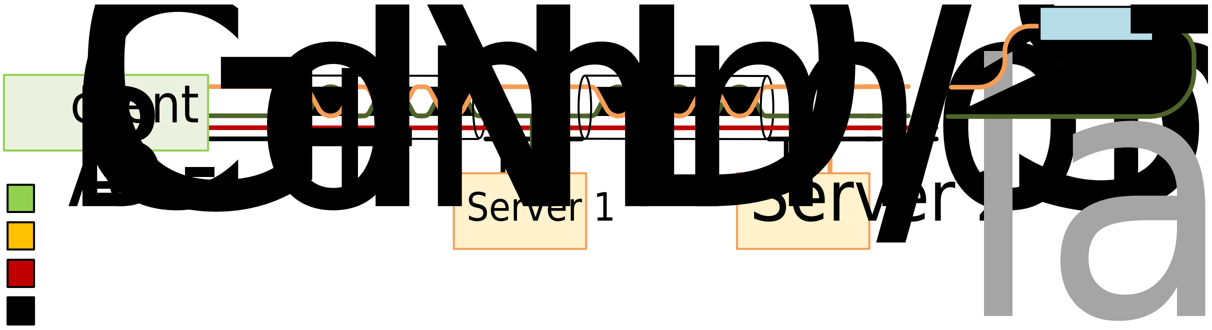
S7-1500 and Arduino Uno R4

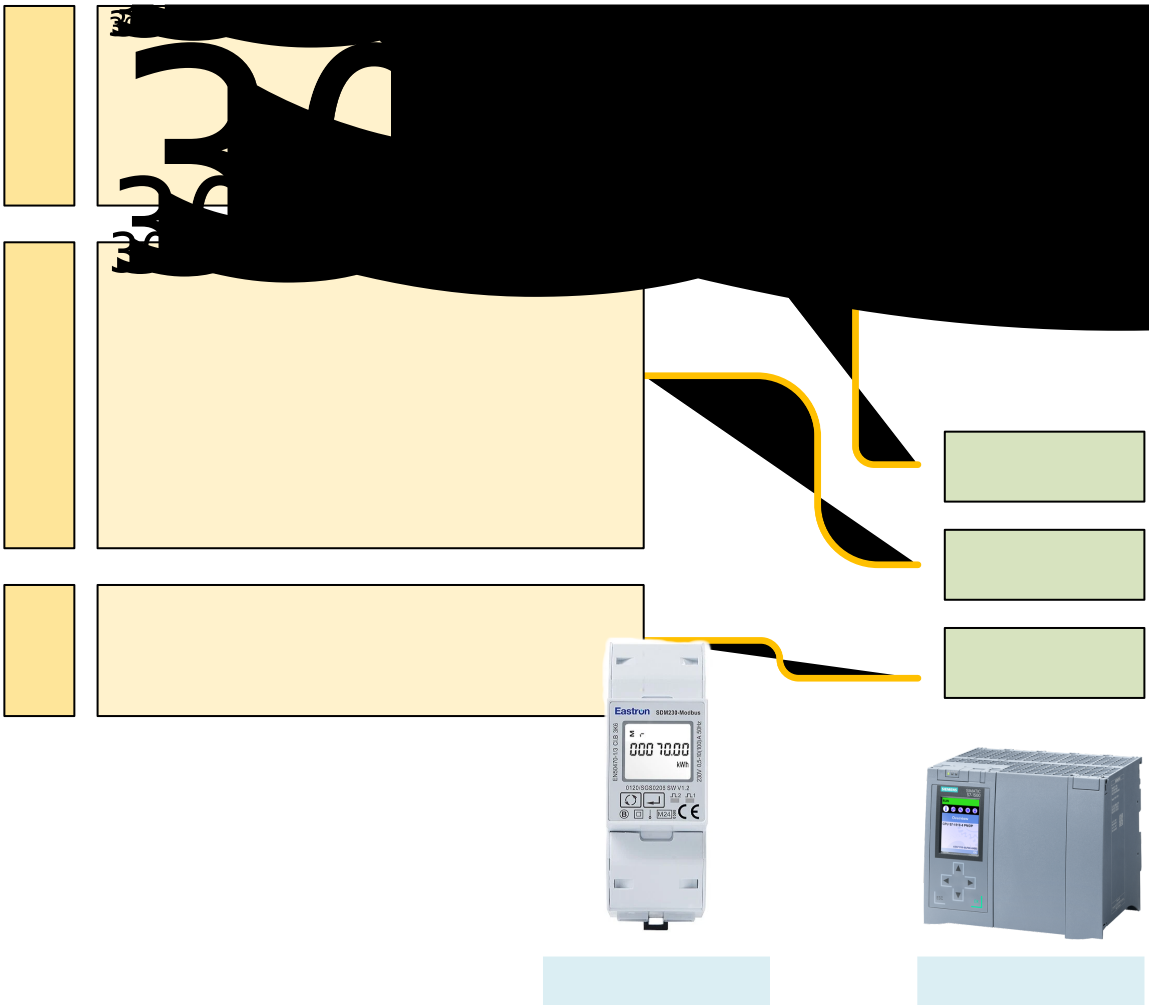
Arduino and Modbus

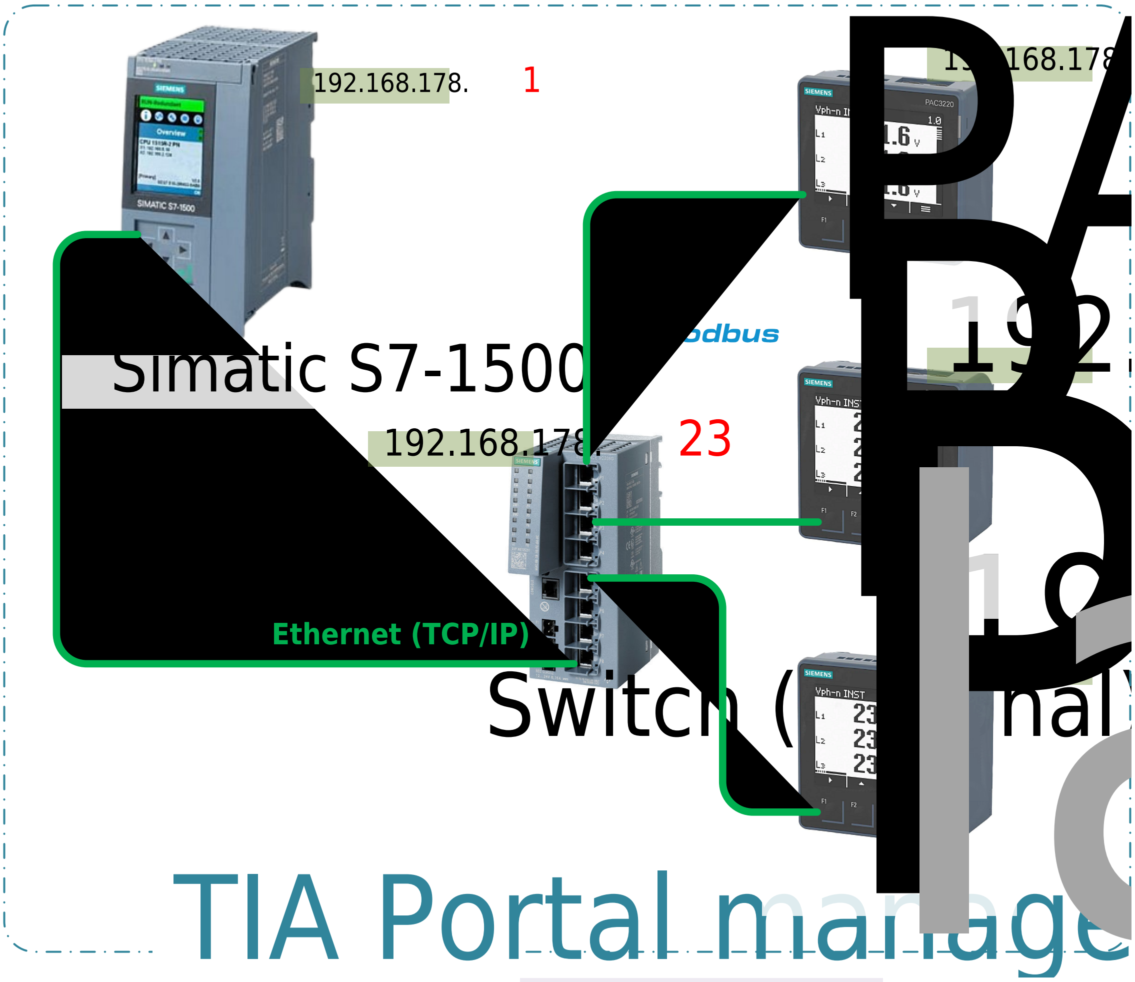
Arduino and Modbus RTU and/or TCP

Modbus Installation examples, step by step

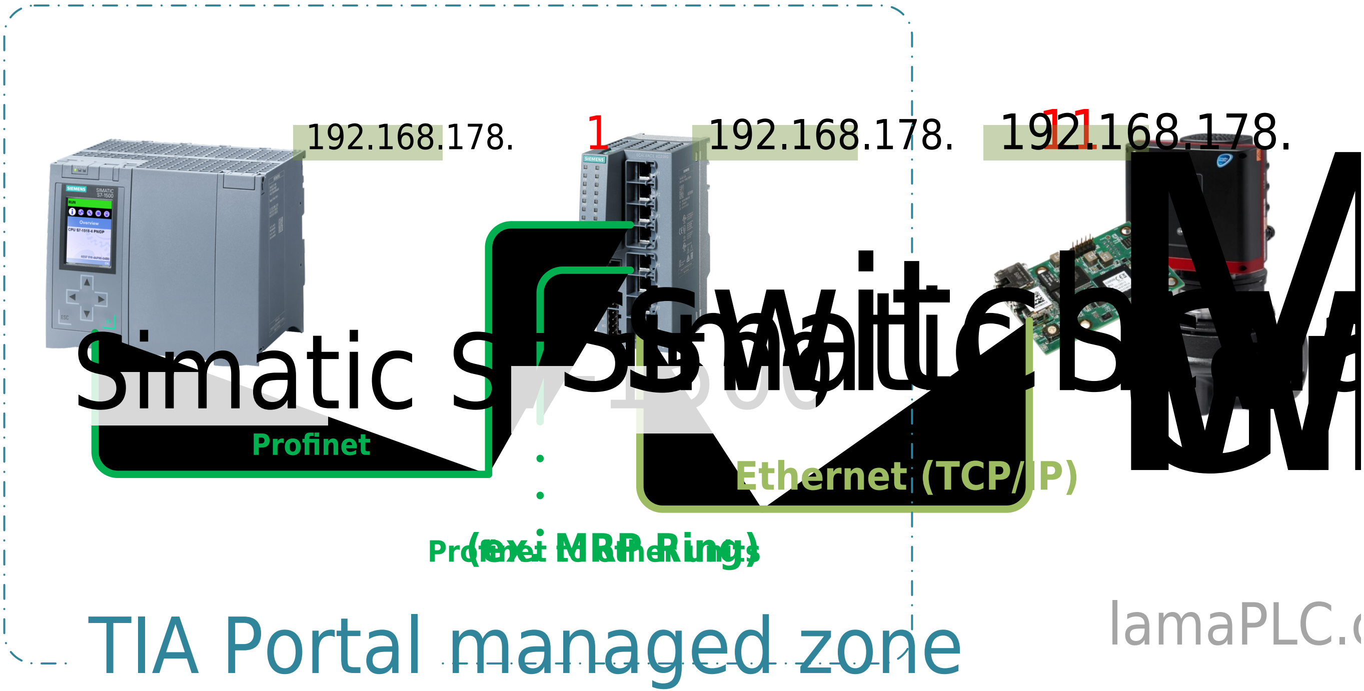
Appendix

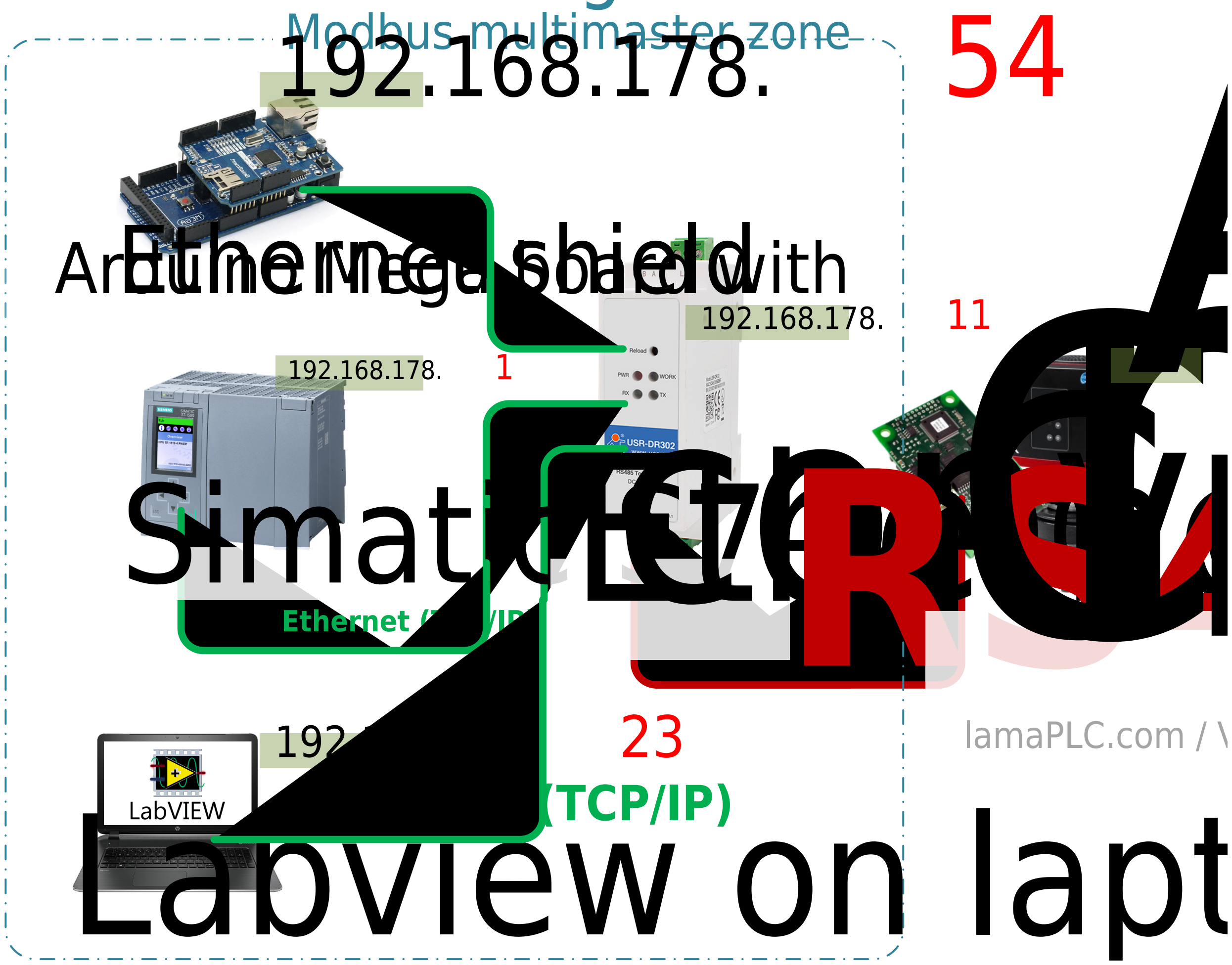


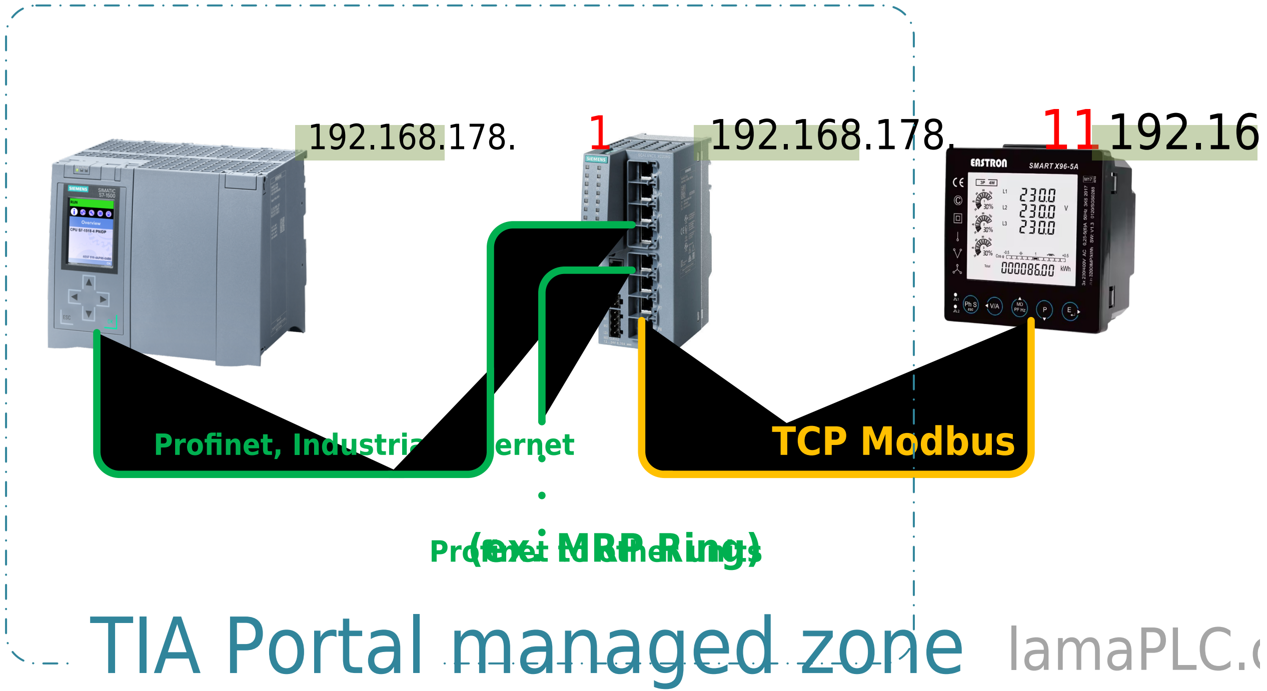

Eastron
SDM230-Modbus
00070.00
kWh

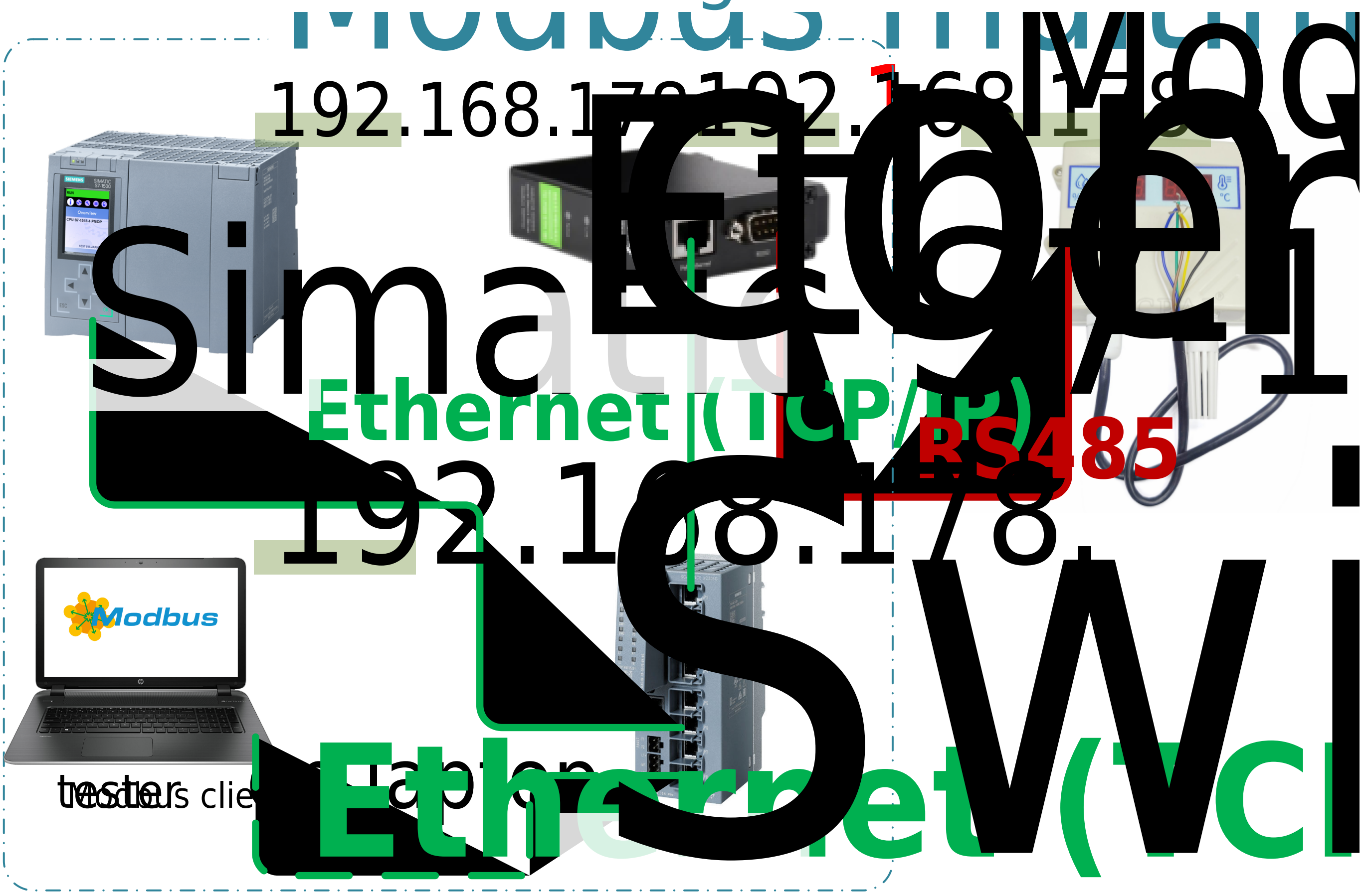


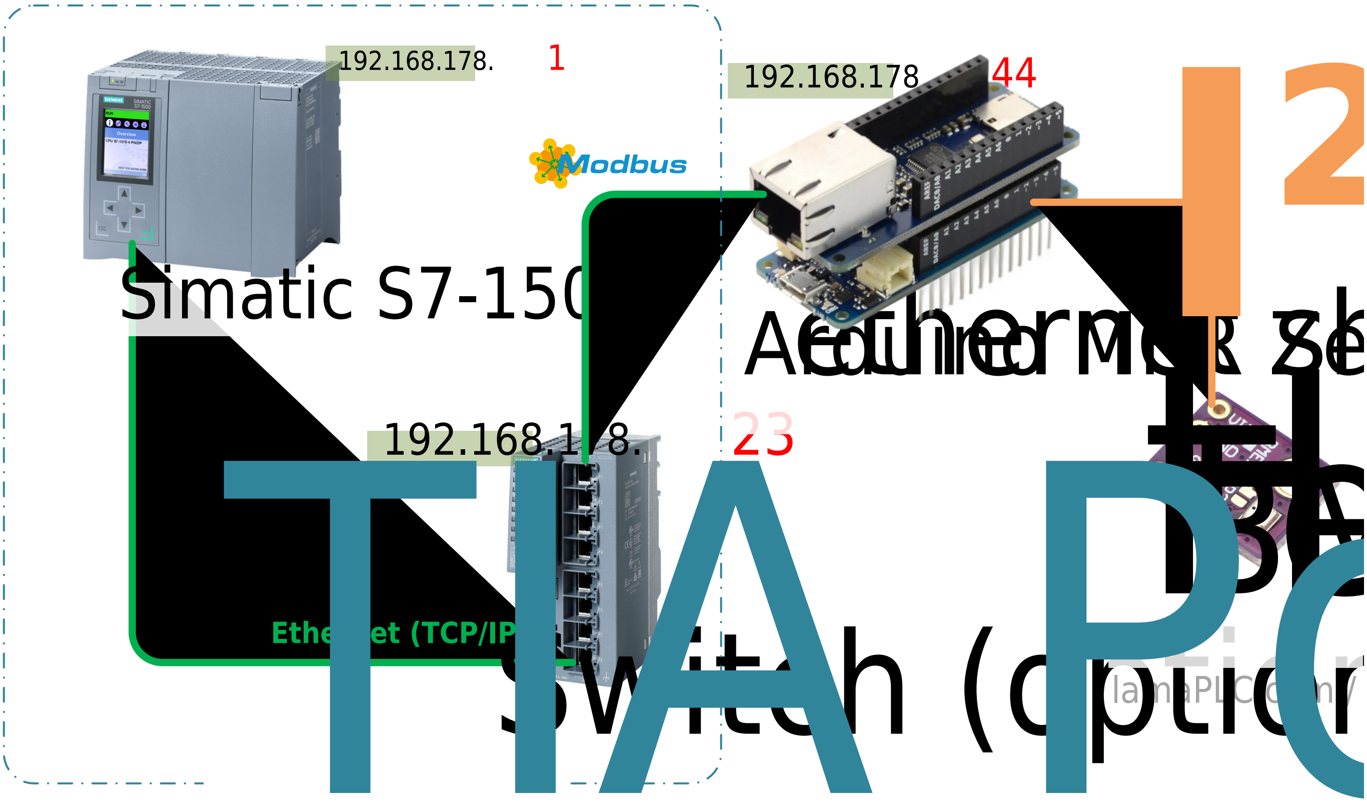
	Addresses
Output Coil	000 00
Input Coi	100 00
Internal	300 0
Holding	400

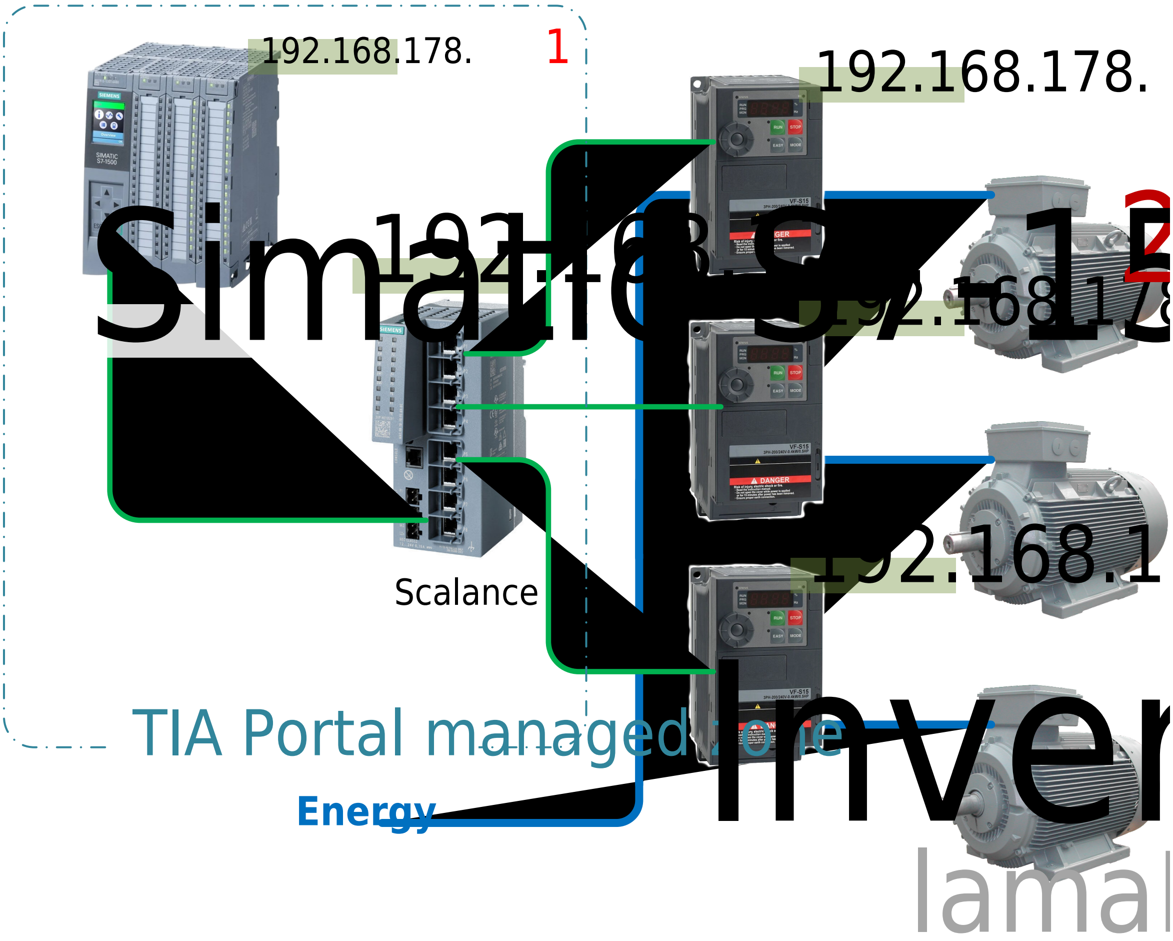

192.168.178.
1
192.168.178.
192.168.178.
11
Simatic S7-1500
Switch
Profinet
Ethernet (TCP/IP)
Profinet Ethernet rings
(exi MRP Ring)
TIA Portal managed zone
lamaPLC.c


Modbus multimaster zone
192.168.178.
54
Arduino Mega board with Ethernet shield
192.168.178.
11
192.168.178.
1
Simatic S7-1200
Ethernet (TCP/IP)
RS
USR-DR302
192.
23
(TCP/IP)
LabVIEW
Labview on lapt
lamaPLC.com / \


192.168.178.
1
192.168.178.
11
192.16
Profinet, Industrial Ethernet
TCP Modbus
(ex: MRP Ring)
Profinet Ethernet Rings
TIA Portal managed zone
lamaPLC.c


Modbus
192.168.178.
192.168.178.
Simatic
Ethernet (TCP/IP)
RS485
192.168.178.
Modbus
Modbus client
tester
laptop
Ethernet(TC
SWI


192.168.178. 1
Modbus
Simatic S7-150
192.168.178 44
Arduino MKR
192.168.178. 23
Ethernet (TCP/IP)
Switch (option
TIA Po
lamaPLC.com


192.168.178. 1
192.168.178.
Simatic S7-15
Scalance
192.168.1
Inver
TIA Portal managed zone
Energy
lamaPL