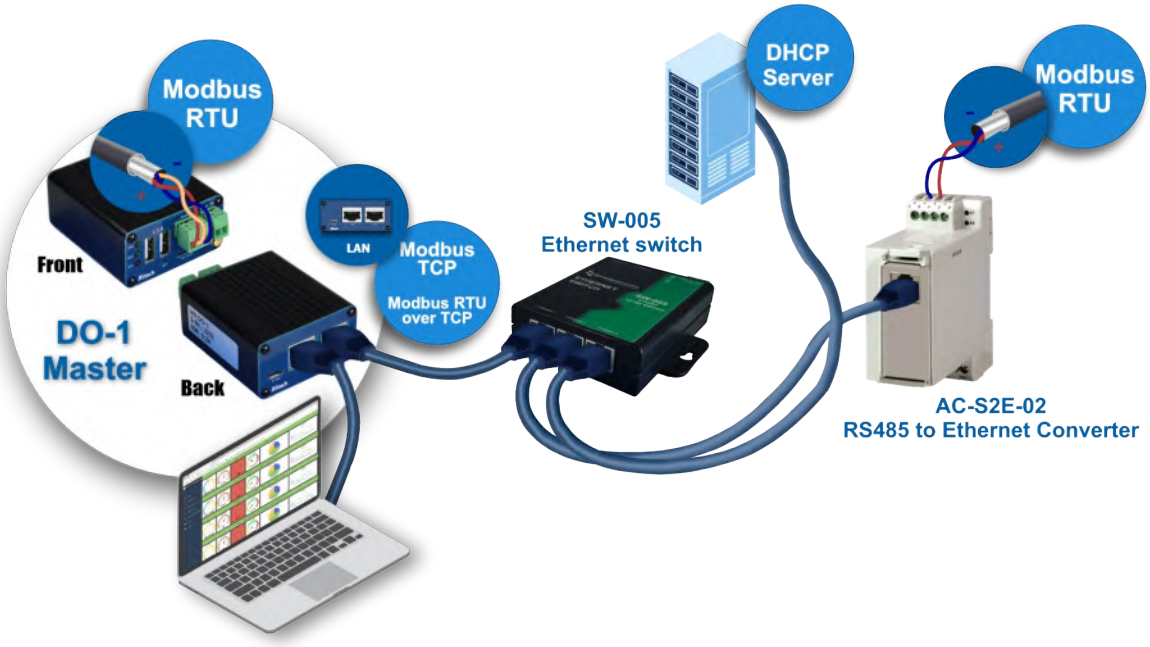
RS-485 REFERENCE GUIDE

ALTECH MEXICO

2024

ENG 05/24

Modbus RTU / Modbus TCP / Modbus RTU over TCP

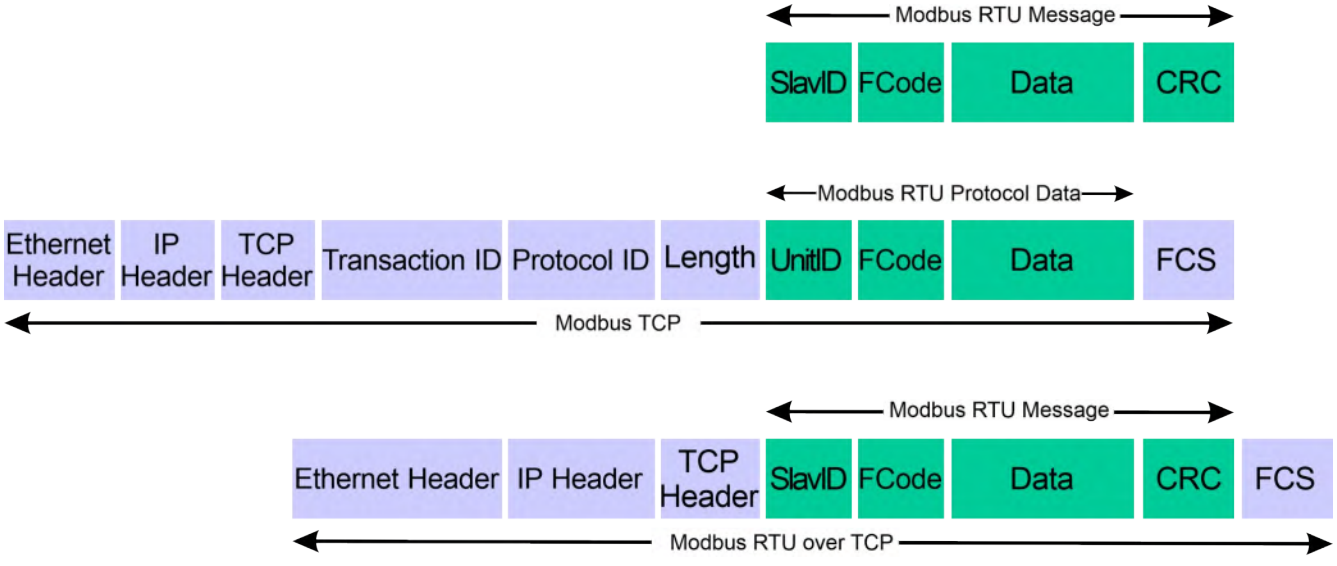


Modbus RTU is a serial communication protocol using mainly RS-485. With RS-485, the range can be up to 4000 feet (1200m) and the speed up to 115200 baud. At 115200 baud, the range is reduced. Devices can be daisy-chained with RS-485.

Modbus TCP uses TCP/IP communication. It still uses a similar Modbus protocol but communicates over the TCP/IP protocol. Modbus TCP has a specific header section and, although the core is the same, it does not use the Checksum that Modbus RTU does. Additionally, the slave ID is called Unit ID.

Modbus RTU over TCP allows you to use TCP/IP to communicate with serial Modbus devices. Modbus RTU over TCP encapsulates the Modbus RTU message within a TCP/IP package. This includes the entire Modbus RTU packet with Slave ID and Checksum. In simple terms, it's a Modbus RTU message transmitted with a TCP wrapper and sent over a network instead of serial lines.

<u>Note:</u> Modbus TCP has a different packet structure than Modbus RTU over TCP. These protocols should be considered entirely different and are not interchangeable.



Modbus Wiring Guidelines

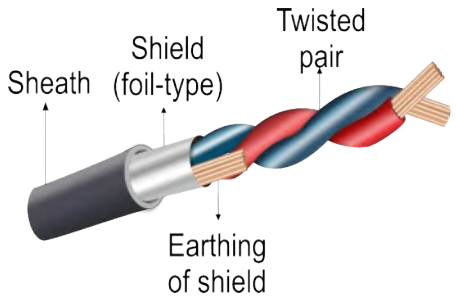
Overview

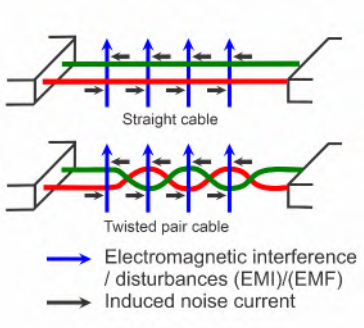
Cabling / Wiring a Modbus device to **DO-1** is unique in its way, compared to other devices which use a normal communication protocol. The installer can experience some difficulties if he is not an expert and not well informed about the factors to be considered while wiring devices in Modbus communication networks. Below are some of those factors:

1. Type of cables

The cable to be used is a shielded twisted pair (telephone type).

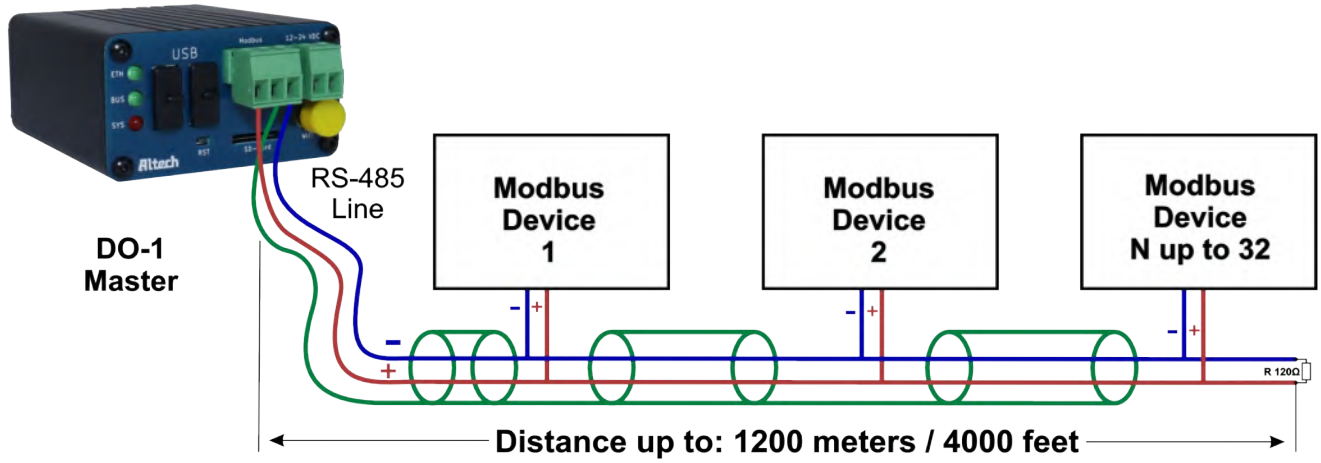
The twin consists of two conductors that are twisted together. This arrangement improves the immunity of the network because the cable forms a series of successive coils, each of which faces in the opposite direction to the next one; in this manner, any magnetic field in the environment traverses each pair of coils in opposite directions, and its effect is thus very reduced (theoretically, the effect on each coil is exactly the opposite of the effect on the next one and thus the effect is nullified). The shielding may be braided (formed by a mesh of thin conducting wires) or be a foil (consisting of a sheet of metal wound around the conductors), both types are equivalent.





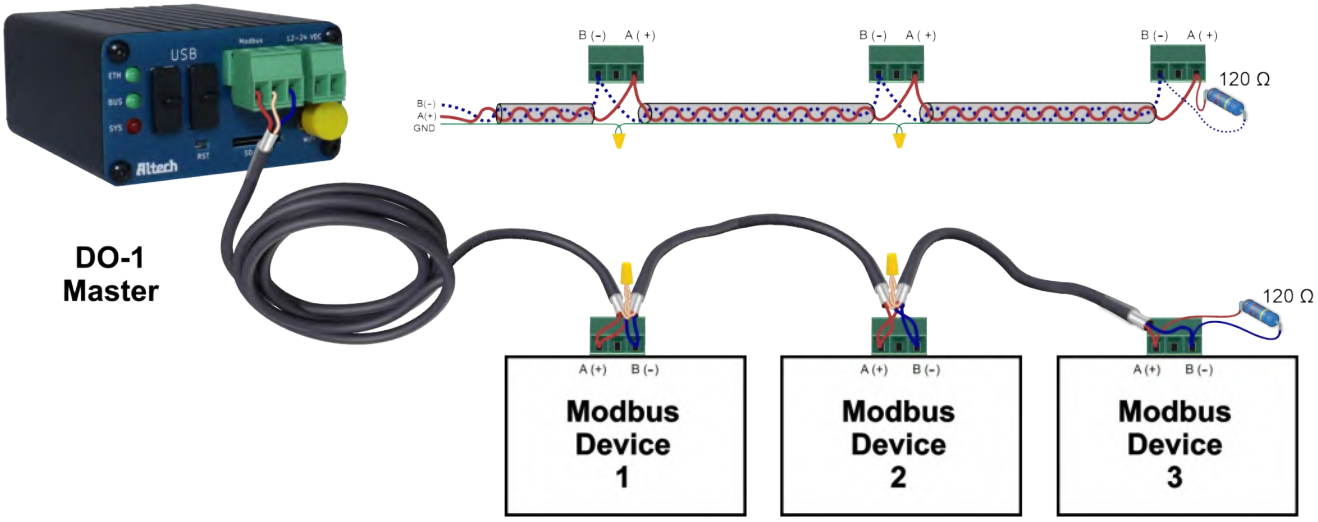
2. Connection

Each Modbus device has an 'A/+' and 'B/-' terminal. The wires of the communication cable (RS-485 system/shielded twisted pair cable) are connected to these two terminals. The connection or wiring is done in such a way that all devices that are part of the Modbus communication network are parallel to each other, as shown below.



Reversing the 'A/+' and 'B/-' connections of a device not only stops communication between these devices, but also stops communication in the entire network of the system due to the incorrect DC (polarization) voltage of the terminals of the incorrectly connected device. To prevent this from happening, wires of the same color are used to connect one type of terminal (e.g. red for A/+ and blue for B/-).

3. Connection between devices



<u>Note:</u> The unshielded wires of the shielded cable should be kept as short as possible. It should also be noted that a secure connection of the cable shield to the Modbus devices is necessary to ensure proper shielding over the individual cable sections. This will ensure interference-free data transmission.

4. Maximum distance of cable and maximum number of devices

The main cable must not be longer than 1200 m. This distance does not include the branches (which must still be short). The maximum number of devices that can be connected to the main cable is 32, including the master.

5. Repeaters

To extend the Modbus network, repeaters can be used, which are signal amplifying and regenerating devices with two communication ports that transmit what they receive from the device at one end to the other and vice versa. By using a repeater, the main cable is divided into different segments, each of which can be up to 1000m long and connect 32 devices (this number includes the repeaters). The maximum number of repeaters that should be connected in series is 3. A higher number will cause excessive delays in the communication system.

6. Connection at the terminal end

In some countries, it is permitted to insert two cables into the same screw terminal. In this case, it is possible to connect the main input and output terminals directly to the terminals of an instrument without creating a branch. If, on the other hand, each terminal can accept only a single cable, a proper branch must be created using three auxiliary terminals for each instrument to be connected.

7. Earth Connection

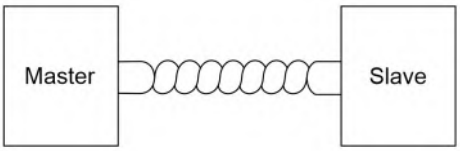
The cable shield must be earthed only at one point (GND). Normally, this connection is made at one end of the main cable.

8. Termination Resistance

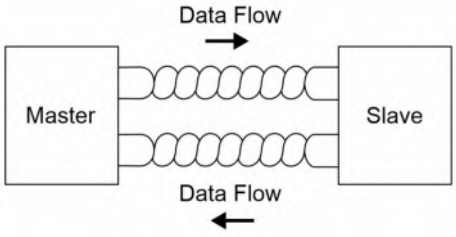
To prevent signal reflections, a 120 Ohm termination must be placed at each end of the trunk cable. The termination must be used only at the ends of the trunk cable. If the total length of the trunk cable is less than 5 m (16 ft.), the use of terminating resistors at the ends of the trunk cable can be avoided.

RS-485 – Full / Half Duplex

Half Duplex:



Full Duplex:



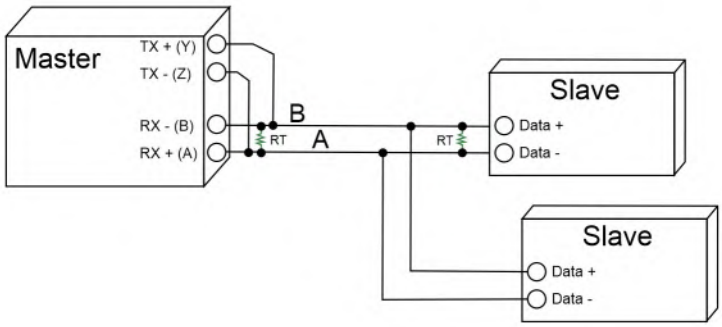
Half Duplex (2-Wire Communication)

RS-485 is designed so that only one transmitter on a twisted pair can be active at a time. With this limitation, the master can send data to the slave, or the slave can send data to the master, but they can't both send data at the same time. This is called "half duplex".

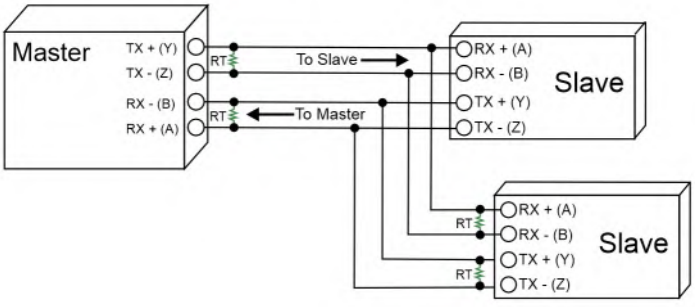
Full Duplex (4-Wire Communication)

A full-duplex system, on the other hand, would allow communication in both directions simultaneously. Full-duplex systems can be designed using RS-485, but require two twisted-pair cables between the master and slave. One twisted pair is used to transmit information in one direction, and the other twisted pair is used to transmit data in the opposite direction.

RS-485 Half Duplex Wiring

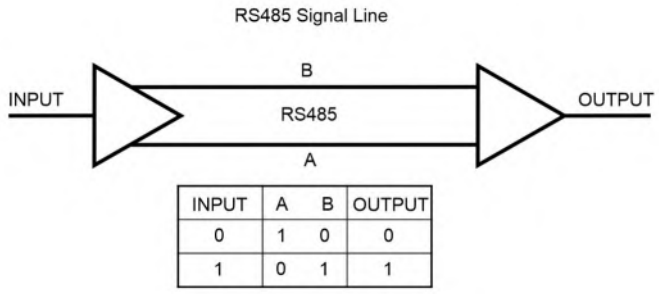


RS-485 Full Duplex Wiring



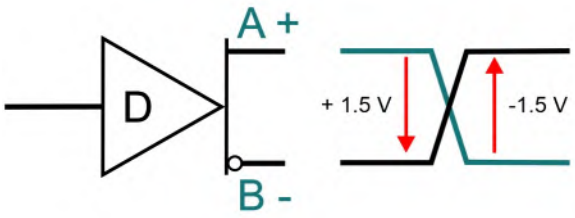
Signals / Voltage Levels on a RS-485 2-Wire Network

Signals:



INPUT	A	B	OUTPUT
0	1	0	0
1	0	1	1

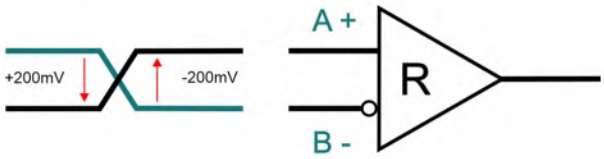
Voltage level for data transmitting:



Voltage (A+) - Voltage (B-) => 1.5Vdc : Logic 0 **|** Voltage (B-) - Voltage (A+) => 1.5Vdc : Logic 1

For transmitting data the minimun voltage differnce should be 1.5 Vdc and not exceed 5 Vdc

Voltage level for data receiving:



Voltage (A+) - Voltage (B-) => 200mVdc : Logic 0 **|** Voltage (B-) - Voltage (A+) => 200mVdc : Logic 1

For receiving data the minimun voltage difference should be 200mVdc.

Note: RS-485 networks can typically maintain correct data with a difference of -7 to +12 Volts.

Tri-State (For waiting):

A not active devices on a RS-485 network is going to a high impedance state (idle).

RS-485 allows multiple devices on the bus for true bidirectional transmission over a single cable, so each Transceiver (TRX) must have tri-state output capability:

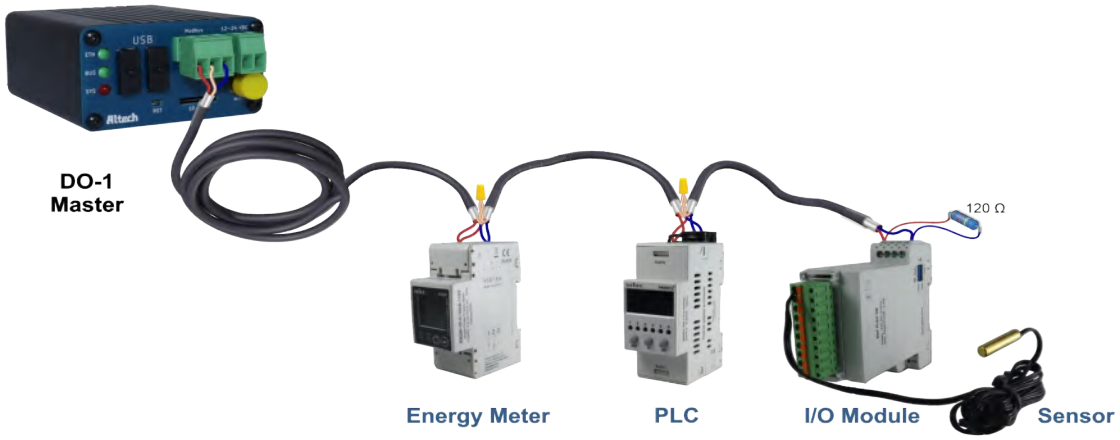
Mark - Sending or Receiving a Logic 1 **|** Space - Sending or Receiving a Logic 0 **|** Idle - High Impedance

Troubleshooting RS-485 Communication Issues

When encountering communication breakdowns in a Modbus RS-485 network, various factors can be at play. Generally, most RS-485-related problems can be categorized into two primary areas:

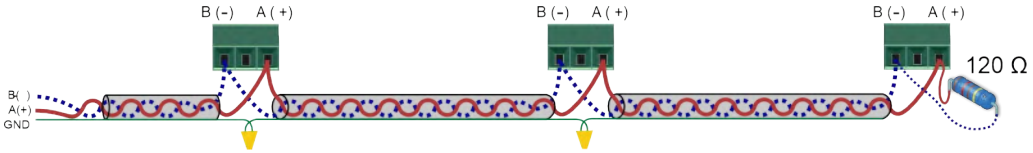
1. Physical Wire Connection Issues

Proper physical connections between devices are essential for RS-485 communication. Devices that communicate via RS-485 should be equipped with an RS-485 connector. This port can be a simple screw terminal, a DB9 connector, or an RJ45 connector. Regardless of the physical implementation, you'll typically be dealing with three key pin assignments: +, -, and ground. Some devices may label these pinouts as A, B, and Ground, while in rarer cases you may encounter Tx/Rx+, Tx/Rx-, and Ground as labels. The diagram below provides a visual representation of how RS-485 devices are typically connected in a daisy-chain configuration.



If your network is configured in this manner and is experiencing communication disruptions or unreliable communication, it is imperative to conduct the following diagnostic checks:

1st Check for reverserd polarity on the RS8485 Line



2nd Examine the physical connections:

Initiate the troubleshooting process by verifying that all connections have been appropriately terminated and securely fastened. Despite its apparent simplicity, loose wire connections can lead to intermittent communication issues within an RS-485 network, posing a challenging problem to resolve.

3rd Test RS-485 Ports:

It is possible that one of the devices in the network has a malfunctioning RS-485 port. To isolate this issue, consider replacing RS-485 devices individually with known functional ones, particularly in multi-drop RS-485 configurations. A single device with a defective serial port can disrupt communication for all devices on the same cable.

4th Address electrical noise:

While RS-485 is known for its resilience to electrical noise, the proximity of communication cables to machinery or equipment generating substantial electrical interference can be problematic. In such cases, it is advisable to reroute the cables to minimize exposure to the sources of interference.

5th Mitigate ground loops:

Ground loops can negatively affect RS-485 signal integrity if multiple devices on the RS-485 cable connect the shield to ground. This interference can distort the RS-485 signal. Good practice dictates that the shield of the communication cable should only be grounded at one end to prevent ground loop problems. In addition, accidental grounding of the shield in the center of the cable should be avoided.

***6th* Consider termination and impedance matching:**

A termination resistor is used in RS-485 communication systems to ensure signal integrity and communication reliability. RS-485 is a serial communication interface often used in industrial applications where data must be transmitted over relatively long distances. A termination resistor is used for several reasons:

1. Prevention of Reflections: When data is transmitted over long cable lengths, signal reflections can occur at the ends of the line. These reflections can disrupt the original signal and lead to data errors. The termination resistor absorbs these reflections, thus minimizing disruptions.

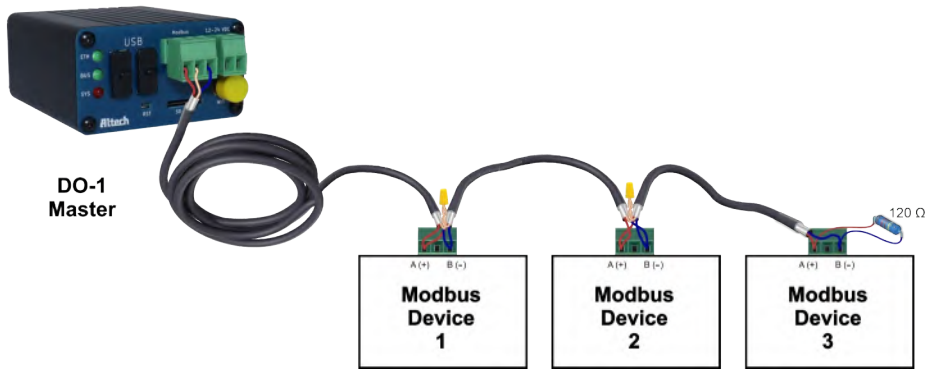
2. Impedance matching: RS-485 lines have a characteristic impedance, typically 120 ohms. A termination resistor with the same value is used to terminate the line's impedance. This helps maintain signal quality and minimize reflections.

3. Immunity to Interference: RS-485 is known for its resistance to interference, making it ideal for industrial environments. Using a termination resistor helps improve the system's ability to combat interference by preserving signal integrity.

4. Extend communication range: The use of termination resistors can increase the communication range by minimizing signal losses due to reflections.

It is important to note that the terminator is typically connected only at the ends of the RS-485 line, not at each device. If you have multiple devices in an RS-485 communication chain, you should use the terminator only at the ends of the line, not at each device in between. The exact resistor value may vary depending on the specific requirements of your RS-485 system, but in most cases it is 120 Ohms to match the characteristic impedance of the line.

Important: In the **DO-1**, only one termination resistor is required at the end of the line, since the **DO-1** already has an integrated 120 Ohm termination resistor.



In summary: These diagnostic measure address fundamental aspects of RS-485 network troubleshooting, including physical connections, device functionality, noise interference, ground loops, and impedance matching through termination and biasing. Careful implementation of these checks can help identify and correct communication problems within the RS-485 network, ultimately improving its reliability.

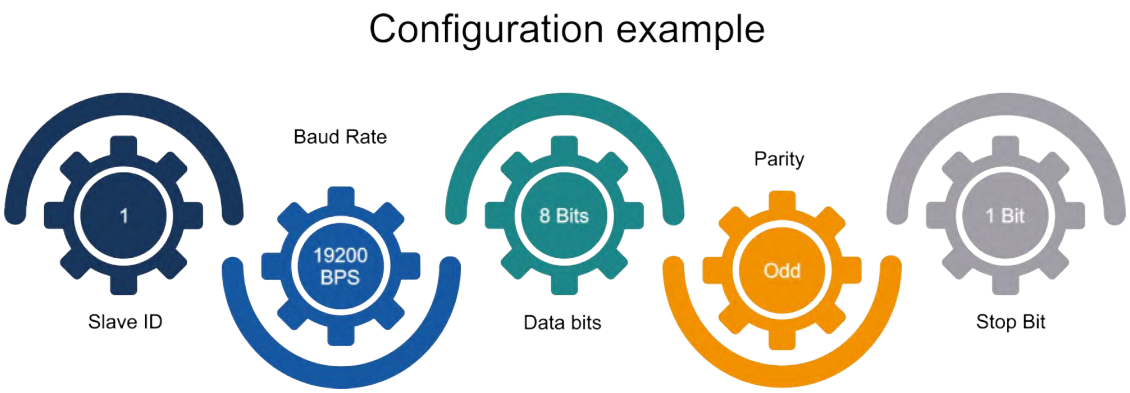
2. Mismatched Communication Settings

Another common source of problems with RS-485 networks is misalignment of communication settings between devices connected to the network:

Once all physical cabling connections have been confirmed to be sound, the next area of concern is to examine the communication settings of all devices on the network. These communication settings primarily include the RS-485 port configurations on each individual device.

Crucial data parameters as: ***1. Baud rate*** ***2. Data bits*** ***3. Parity bit*** ***4. Stop bits***

need to be identical for each device on the communication cable.



It should be noted that these settings can include a variety of different combinations, which is indeed permissible. However, the critical task is to carefully verify that these settings remain identical from device to device.

Mismatched port configurations typically lead to communication breakdowns within the network. In such cases, a master device may issue a request to a slave, only to receive no response from the slave.

To illustrate a potentially worse scenario, there have been documented cases where a master device has sent a query to a slave device. Even in cases where the slave device had different port settings, it would respond with a Modbus error code. This response was due to the mismatch in port settings, causing the slave to misinterpret the transmitted data as an irregular Modbus query.

In summary: It is imperative that the RS-485 port settings of all devices connected to the network be carefully scrutinized and consistent. This diligence serves as a fundamental measure to ensure consistent and effective communication within the RS-485 network.

By addressing these two key areas, you can effectively troubleshoot and resolve RS-485 communication problems, enhancing the reliability of your network.

Modbus Reference Information

Modbus Data Address Format

This is one of the most confusing parts of working with Modbus devices. Because manufacturers of Modbus-compatible devices deviate from the Modbus standard, each manufacturer interprets the standard in its own way. And they use either the traditional or extended convention as well as register (entity) addresses or register (entity) numbers for coils or registers.

- ⇨ *It is important to make a distinction between register (entity) number and register (entity) address!*
- ⇨ *Do the numbers in your documentation refer to the register number or register address?*

Coil and Register number and data offset

- Traditional convention is 5 digits and 1-(one)-based for coil or register Number
- Extended convention is 6 digits and 1-(one)-based for coil or register Number

The traditional numbering convention is offset from the extended protocol by 1 digit. This can sometimes cause confusion!

Memory Map

Traditional	Extended	Register Type	Size and Access	Notes	
Coil or Register Number	Coil or Register Number				
00001 – 09999	000001 – 065536	Coils	1-Bit, read-write	DO	Discrete (digital) Output
10001 – 19999	100001 – 165536	Inputs	1-Bit, read	DI	Discrete (digital) Input
30001 – 39999	300001 – 365536	Input Register	16-Bit word, read	AI	Analog Input
40001 – 49999	400001 – 465536	Holding Register	16-Bit word, read-write	AO	Analog Output

***Register (Entity) number* , traditional convention 5 digits:**

The entity numbers start with a digit representing the entity (object type), followed by four digits representing the entity location. A total of 5 digits:

- coils numbers start with **0** and span from **0**0001 to **0**9999
- discrete input numbers start with **1** and span from **1**0001 to **1**9999
- input register numbers start with **3** and span from **3**0001 to **3**9999
- holding register numbers start with **4** and span from **4**0001 to **4**9999

***Register (Entity) number* , extended convention 6 digits:**

The entity numbers start with a digit representing the entity (object type), followed by five digits representing the entity location. A total of 6 digits:

- coils numbers start with **0** and span from **0**00001 to **0**65536
- discrete input numbers start with **1** and span from **1**00001 to **1**65536
- input register numbers start with **3** and span from **3**00001 to **3**65536
- holding register numbers start with **4** and span from **4**00001 to **4**65536

***Register (Entity) address* :**

The Entity address is used to pinpoint a specific register within a Modbus device. These addresses indicate the location of data that you want to read from or write to within the device.

The entity address is the starting address; 0 to 9998 for traditional and 0 to 65535 for extended convention.

Confusing Modbus Register Addresses

Sometimes "entity numbers" are used in the documentation of Modbus devices. These numbers are typically five or six digits long and contain information about both the register (object) type and the register address. The first digit describes the register (object) type, e.g. 4 is a holding register. The remaining digits describe the address plus one (yes, very confusing).

Note: Documentation for Modbus is not well standardized. Actually, there is a standard, but it is not well followed when it comes to documentation. You will have to do one or more of the following to decipher which register a manufacturer is really referring to.

Some guidelines to follow: ***Look at the numbers themselves !***

1st For traditional or extended convention: Look for 5 or 6 digits to see if you are using the traditional or extended convention. If you see the first register on the list with the number 40001, that really tells you that register #1 is a holding register and you are using the traditional (5-digit) convention. This form of notation is often referred to as the old Modicon convention.

2nd 0-(zero)-based addressing: Look at the number itself for 0 (zero) or 1 (one) based addressing. Look at the last digit to the right of the first register (entity) number or address. If the last digit is 0 (zero), this means that 0-(zero)-based addressing is being used.

3rd Do the numbers in your documentation refer to the register number or address?
- Look for keywords in the register description, such as analog, digital, descrete, holding register, input register, coil, input, output, etc. If the documentation says #1 and tells you they are holding registers, then you have holding register #1.
- Look for a definition of the function codes to be used. If you see a register #1, along with notation telling you to use function codes 3 and 16, that also tells you it is a holding register #1.

4th As well, look in the documentaion if binary or hexadecimal numbering is used for register numbers or addresses.

Variables (Data Types)

Variable Type	Abbreviation	Variable Range
1-Bit	BOOL	0 and 1
16-Bit Unsigned Integer	UINT16	0 to 65535
16-Bit Signed Integer	INT16	-32768 to 32767
32-Bit Unsigned Integer	UINT32	0 to 4,294,967,295
32-Bit Signed Integer	INT32	-2,147,483,648 to 2,147,483,647
64-Bit Unsigned Integer	UINT64	0 to (2^64)-1
64-Bit Signed Integer	INT64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
32-Bit Float	Float32, FP32	-3.4028E+38... +3.4028E+38
64-Bit Float	Float64, FP64	-3.4028E+308... +3.4028E+308

Some examples of documentation variantes:

The Modbus slave vendor determines and documents what data is available as 0X, 1X, 3X, or 4X references (i.e. Coils, Discrete Inputs, Input Registers, and Holding Registers, respectively).

Simple Modbus slaves may expose all of their data as only **4X** references (Holding Registers), regardless of whether they are analog or digital values (up to 16 digital values may be bit-packed into a register) and regardless of whether they are inputs (read-only by the master) or outputs (read/write by the master).

More complex Modbus slaves may use all reference types and will typically map data as follows:

Digital Outputs => **0X** references (Coils)
Digital Inputs => **1X** references (Discrete Inputs)
Analog Inputs => **3X** references (Input Registers)
Analog Outputs => **4X** references (Holding Registers)

"**Register reading MSB to (→) LSB**" refers to Big Endian
"**Register reading LSB to (→) MSB**" refers to Little Endian
"0X, 1X, 3X, 4X" type is referring to register (object) type like coil, input, input register or holding register.
"4X0000" is referring to a holding register (4) , (X) for hexadecimal and to 0-(zero) based address register 0.
"Signed" or **"INT"** refers to a signed integer data type
"Unsigned" or **"USINT"** or **"UINT"** refers to a unsigned integer data type
"Unsigned short" or **" UINT16"** refers 16-bit unsigned integer data type
"Short" refers to a 16-bit integer register type
"INT16" refers to a signed 16-bit integer data type
"1-Word" or **"2 Bytes"** refers to a 16 bit-register
"INT32" or **"DINT"** refers to a signed 32-bit integer data type
"UINT32" or **"UDINT"** refers to a unsigned 32-bit integer register type
"Long" refers to a 32-bit integer register type
"Unsigned long" refers to a 32-bit unsigned integer data type
"2-Word" or **"DWORD"** refers to 32-bits or two consecutive 16-bit registers
"Float" or **"Float32"** or **"FP32"** or **"REAL"** refers to 32-bits floating point register
"Double" or **"Float64"** or **"FP64"** or **"LREAL"** refers to 64-bits floating point register
"BOOL" refers to 1-bit register (0 or 1)
"HEX" refers to hexadecimal representation
"**4-Word**" or "**LWORD**" refers to 64-bits or four consecutive 16-bit registers

Endiannity

Is represented in two ways, ***Big-Endian*** and ***Little-Endian***.

Example below:

Decimal Number: 61455 = Binary Number: 11110000 00001111 = Hexadecimal Number: F0 0F

	Least Significant number		Least Significant byte (LSbyte)
Most Significate number		Most Significate byte (MSbyte)	

Register reading left to right	**Binary Numbers**	**Hexadecimal Numbers**
	---------->	---------->
Big-Endian (most common)	11110000 00001111	F0 0F
	MSbyte LSbyte	
	---------->	---------->
Little-Endian	00001111 11110000	0F F0
	LSbyte MSbyte	

Register reading right to left	**Binary Numbers**	**Hexadecimal Numbers**
	<----------	<----------
Big-Endian Reversed	00001111 11110000	0F F0
	LSbyte MSbyte	
	<----------	<----------
Little-Endian Reversed	11110000 00001111	F0 0F
	MSbyte LSbyte	

<u>Note:</u> Big-endian is the most common because it corresponds to the way humans write numbers (most significant digits on the left). It is used in network protocols such as IP and TCP, which follow a standard called network byte order (big-endian). Early computer systems, such as the IBM 360, preferred big-endian, and its consistency makes it easier to write software for different systems without making changes.

Modbus Typical Query-Response Cycle

The master will send a request (query) in the form of a byte stream, the slave will respond with a similar (but different) byte stream, with 4 different sections consisting of bites that serve different purposes.

Query of master

Device address	Unit ID of target slave device (1 Byte) *Modbus unit ID*
Function code	Tells the slave what function to perform (1 Byte) *Function Code*
Eight-Bit data bytes	Specifies address block range (4 Bytes) *Start Memory block address + number of blocks*
Error check	CRC - Cyclical Redundancy Check (2 Bytes) *for Data Transmission Error Detection*

Response of slave

Device address	Unit ID of target slave device (1 Byte) *Mirror of Master Query*
Function code	Returns the same function code (1 Byte) *Mirror of Master Query*
Eight-Bit data bytes	Data is returned (4 Bytes)
Error check	CRC - Cyclical Redundancy Check (2 Bytes) *for Data Transmission Error Detection*

Example for analog input data reading (electrical current in mA)

Query of Master: 01 03 00 00 00 01 C5 C8 (hex)

Data	Byte	Description	Remarks
01	1	Device address	Address range
03	1	Function code	03 - Read holding register
00 00	2	Register address	Starting register address
00 01	2	Register number	0001 - Read 1 register (electrical current)
C5 C8	2	CRC Check Code	CRC Check Code created by Modbus

Response of Slave: 01 03 02 07 69 B6 26 (hex)

Data	Byte	Description	Remarks
01	1	Device address	Address range
03	1	Function code	03 - Read holding register
02	1	Byte of data	02 - Read 2 bytes
07 69	2	Read data	0769 - Analog input channel 1 (electrical current)
B6 26	2	CRC Check Code	CRC Check Code created by Modbus

Analog value of Channel 1 : 0769 (hex) = 1897 (dec). Electrical current (I) = 1897 * 0.005 (Gain/Multipicator) = 9.485mA

Modbus commands (Most common function codes)

Function Code	Action	Description	Value Type	Access Type
01	Read DO	Read coil status	1-Bit	Read
02	Read DI	Read input status	1-Bit	Read
03	Read AO	Read holding register	16-Bit	Read
04	Read AI	Read input register	16-Bit	Read
05	Write one DO	Force single coil	1-Bit	Write
06	Write one AO	Preset single register	16-Bit	Write
15	Multiple DO recording	Force multiple coils	1-Bit	Write
16	Multiple AO recording	Preset nultiple registers	16-Bit	Write
17	Report Slave ID	Read holding register	16-Bit	Read

Modbus Exception Codes

01	**Illegal function**	The function code received in the query is not an allowable action for the slave. If a Poll Program Complete command was issued, this code indicates that no program function preceeded it.
02	**Illegal data address**	The data address received in the query is not an allowable address for the slave.
03	**Illegal data value**	A value contained in the query data field is not an allowable value for the slave.
04	**Slave device failure**	An unrecoverable error occurred while the slave was attempting to perform the requested action.
05	**Acknowledge**	The slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the master. The master can then issue a Poll Program Complete message to determine if processing is complete.
06	**Slave device busy**	The slave is engaged in processing a long-duration program command. The master should re-transmit the message later when the slave is free.
07	**Negative acknowledge**	The slave cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 decimal. The master should request diagnostic or error information from the slave.
08	**Memory parity error**	The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device.

Troubleshooting Techniques for Modbus Communication

This chapter focuses on troubleshooting techniques specific to Modbus protocol issues. Assuming that the cabling and RS-485 port settings have already been checked, we will look at methods for diagnosing and correcting problems within Modbus networks. Whether you are dealing with an existing Modbus RS-485 network or setting up a new one, these checks and techniques will be invaluable.

1. Unit ID Verification

In a Modbus network, each device must have a unique Unit ID. When troubleshooting network problems, the first step is to confirm that there are no duplicate Unit IDs among the devices. It is important to repeat this check because some devices configure their Unit IDs through dip switches or local displays, while others require computer-based configuration using the manufacturer's software.

2. Modbus Master Configuration

Understanding that the Modbus master is responsible for coordinating all network communication, it is imperative to ensure proper configuration. The Modbus master should be configured to recognize:

- The Unit ID of each slave on the network.
- The specific registers to read from and write to for each slave.

This configuration typically occurs via a computer using the manufacturer's software. It's vital to confirm that the Modbus master possesses an accurate list of slave Unit IDs on the network and the correct register listing for each respective slave.

3. Modbus Master Substitution

This method requires a computer, preferably a laptop, and a USB to RS-485 converter. Run a Modbus master application on your computer to send Modbus polling messages to the network. The goal is to replace the original Modbus master device with the laptop acting as the Modbus master to communicate with each slave on the network. Effective communication confirms that the problem is with the original Modbus master device, allowing its configuration to be reevaluated. If the Modbus master application software fails to communicate with the slave devices, disconnect and reconnect all slaves one at a time to identify the problem devices.

If communication to the slaves is established, but no data is retrieved, examine the response data streams for exception responses and review the associated exception codes.

4. Modbus Slave Substitution

This method is useful when dealing with new equipment that is set to act as a Modbus master. To verify the accuracy of the configuration instructions in the manual, use your laptop and a USB to RS-485 converter. Run a Modbus slave simulator application, such as Modsim32 from win-tech, on your laptop to simulate a Modbus slave. Connect the Modbus master to the RS-485 converter and the laptop, configure the Modbus master to read data from the simulated slave, and modify data values within the simulation. Verify that the Modbus master reads these data values correctly.

Using the Modbus Master and Modbus Slave Simulator application software, you can create controlled test scenarios to effectively troubleshoot network problems.

A list of simulation software and (vendors) that offer Modbus solutions:

Modbus Master Application:

ModScan32 (Win-TECH)

Simply Modbus Master (Simply Modbus)

CAS Modbus Scanner (Chipkin Automation Systems)

Modbus Poll (modbus tools)

QModMaster (libmodbus)

Winlog Lite (SCADA)

ModbusView TCP (Ocean Controls)

Simply Modbus TCP (Simply Modbus)

Modbus Slave Simulator Application:

Modsim32 from (Win-TECH)

Simply Modbus Slave (Simply Modbus)

CAS Modbus Slave (Chipkin Automation Systems)

Modbus Slave (modbus tools)

Some helpful links:

https://rapidscada.net/modbus/

These applications play a vital role in facilitating the troubleshooting techniques discussed in this chapter.

Glossary

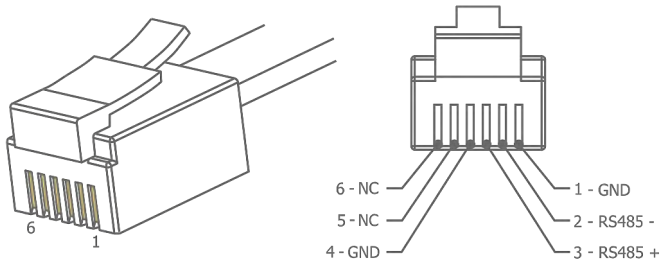
0X — Hexadecimal
BIT— binary digit
BYTE— By eight (group of 8 bits)
BPS — bits per second
BIN— binary
CRC — Cyclic Redundancy Check
DEC— decimal
DHCP — Dynamic Host Configuration Protocol
HEX — Hexadecimal
IEC — International Electrotechnical Commission
IP — Internet Protocol
PoE — Power over Ethernet
TCP/IP — Transmission Control Protocol/Internet Protocol
TRX —Transceiver contains both a transmitter and a receiver
RTU — Remote Terminal Unit

RS-485 – Two wire connections

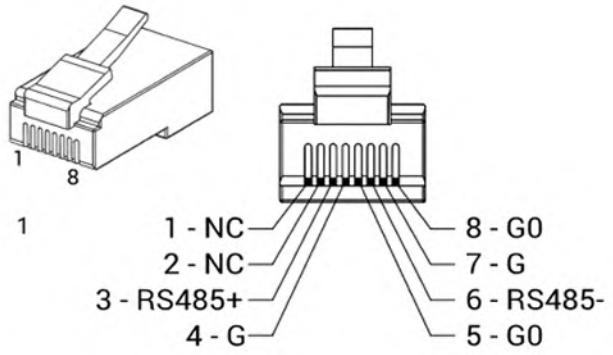
The RS485 standard does not specify a specific connector type or pin layout, but there are several widely used conventions in practice. Here are some example connectors and pin layouts:

Terminal Blocks/Screw Terminals: RS485 devices often use terminal blocks for connecting the RS485 twisted pair wires. These terminal blocks typically have screw terminals for easy connection of the RS485 wires.

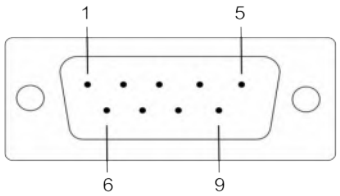
J11 (example):



RJ45 (example):



DB9 (example):



Male DB9	Pin	RS485-2W
	1	- -
	2	- -
	3	Data -
	4	Data +
	5	GND
	6	- -
	7	- -
	8	- -
	9	- -

Note: It is important tp note that while these are example conventions, the actual pinout and connector type used for RS485 communication may vary depending on the specific device or manufacturer. Therefore, it's always a good practice to consult the documentation provided by the device manufacturer for the correct pinout and connector type.