SOLUTION NOTE

Modbus RTU Master-to-Master Communication Solution

Exchanging data between two Modbus RTU masters through an X2Modbus gateway

迅饶 sunfull 专注自控 物物互联

Shanghai Sunfull Automation Co., Ltd

Solution note · March 2017

PREPARED BY

Shanghai Sunfull Automation Co., Ltd

Technical Support: +86-21-20252795 | sales@opcmaster.com

March 2017 · English edition

Contents

1. Project Background 1

2. Requirement Analysis 1

2.1 Problem Statement 1

2.2 The X2Modbus Gateway 2

3. Solution Design 4

3.1 Solution Overview 4

3.2 Communication Topology 4

3.3 How the Data Path Works 5

4. Configuration Procedure 6

4.1 Creating the Simulator Driver 6

4.2 Defining the Intermediate Tags 7

4.3 Enabling the RTU Server Channels 7

4.4 Setting COM1 for the PLC 8

4.5 Setting COM2 for the Host Computer 8

4.6 Optional Modbus TCP Access 9

5. Verification 10

1. Project Background

Industrial control systems are usually built around a master-slave relationship: one master polls a group of slaves, and each slave answers only when it is asked. That model covers the majority of field installations, but it is not the only one. Occasionally a project requires two masters to exchange data with each other, and Modbus itself provides no native mechanism for master-to-master traffic. The bus arbitration rules assume a single master per segment, so the two devices cannot simply be wired together.

This note describes the field problem, explains why it cannot be solved by wiring alone, and presents a working solution built on the X2Modbus gateway family (MOD1002 and MOD2004). The solution keeps both devices in their original master role and introduces no changes to their firmware.

2. Requirement Analysis

2.1 Problem Statement

A typical case looks like this. A workstation host computer needs to monitor a PLC at the end of the line. Normally the PLC would act as a Modbus slave and answer read requests from the host. This particular PLC, however, is a special model that can only operate as a Modbus master: instead of answering requests, it continuously writes values outward, using the write as its only means of publishing data.

Placing such a device on the same RS-485 segment as a host computer produces a bus with two masters. Both devices initiate transactions, both expect to control the timing of the frame, and neither will answer a request from the other. The result is colliding frames and no usable data, as illustrated in the figure below.

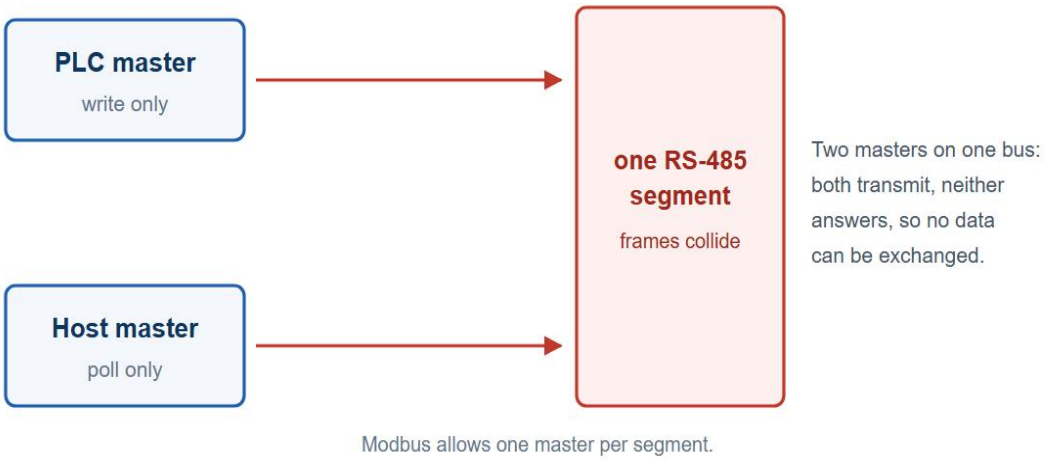

Figure 1 Two masters sharing one segment

Seen from the data's point of view, the requirement is modest. The PLC needs a place to deposit values, and the host needs to read those values whenever it wants. What is missing is neutral memory that neither device has to arbitrate for, and that both can reach with ordinary Modbus requests. Supplying that memory is exactly what a gateway does.

2.2 The X2Modbus Gateway

X2Modbus is a Modbus gateway protocol suite developed in-house. It works as a communication bridge: it converts common Modbus traffic, or a variety of non-standard protocols, into one or more Modbus slave devices, so that different host masters can access field equipment through standard Modbus requests. The suite runs in two application modes.

Table 1 The two application modes of X2Modbus

Mode	Description
Local mode	The software runs on a mainstream Windows platform (Windows XP / 2000 / 2003 / 7 / 8 / 10) and converts various protocols into standard Modbus for upstream reading. It can also replicate one Modbus slave into several Modbus slaves, so that multiple Modbus masters can access the same data.

Mode	Description
Gateway mode	The software acts as a configuration tool. A project is built on the PC, downloaded to a Modbus hardware gateway, and then runs standalone in the gateway to perform protocol conversion on site.

The hardware gateway used in this solution is a fanless DIN-rail unit. The MOD2004 shown below provides four electrically isolated RS-485 channels and two 100M/1000M Ethernet ports, and is rated for 1,024 points, 128 devices and more than 500 acquisition protocols, which it forwards as Modbus RTU/TCP slaves or as an OPC UA server. The MOD1002 is the smaller member of the same family, with two RS-485 channels and 512 points, and is sufficient when only two serial channels are needed.



Figure 2 MOD2004-ARM gateway and its key specifications

Table 2 MOD2004 hardware specification

Operating system	Linux
Processor	1 core, 1 GHz
Memory	512 MB RAM, 4 GB Flash
Power supply	AC/DC 12–24 V, approx. 8 W
Ethernet	2 × 100M/1000M

Operating system	Linux
Serial ports	4 × RS-485 (COM1–COM4), electromagnetic isolation
Mounting	35 mm DIN rail; galvanised steel, 300 g
Operating temperature	−20 … +70 °C

3. Solution Design

3.1 Solution Overview

The requirement is met with a single MOD1002 or MOD2004 gateway placed between the two masters. Each master is connected to its own serial channel of the gateway, and the gateway presents itself as an independent Modbus RTU slave on each of those channels. Behind the two slave interfaces sits one shared register area, so a value written by one master is immediately visible to the other.

Because the two masters never share a physical segment, neither of them is aware that a second master exists. The PLC continues to write values at its own pace, the host computer continues to poll at its own pace, and the gateway absorbs the difference in timing.

3.2 Communication Topology

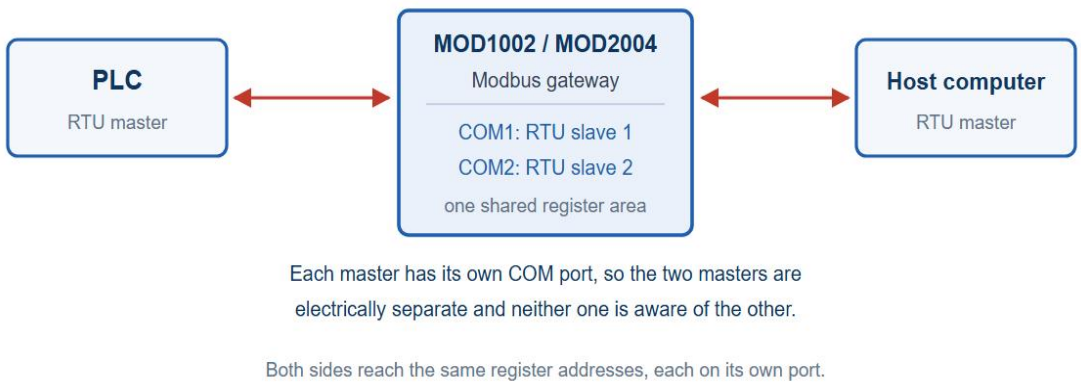

Figure 3 Communication topology: PLC on COM1, host computer on COM2

The topology is deliberately simple. COM1 of the gateway is wired to the PLC and behaves as Modbus RTU slave 1; COM2 is wired to the host computer and behaves as Modbus RTU slave 2. Both channels may use the same station number, because they are electrically separate segments and are never addressed at the same moment.

3.3 How the Data Path Works

Inside the gateway, an intermediate variable layer decouples the two sides. The variables are defined once in the X2Modbus project, and each of them is exposed on both slave channels at the same register address. The flow of a single value is shown below.

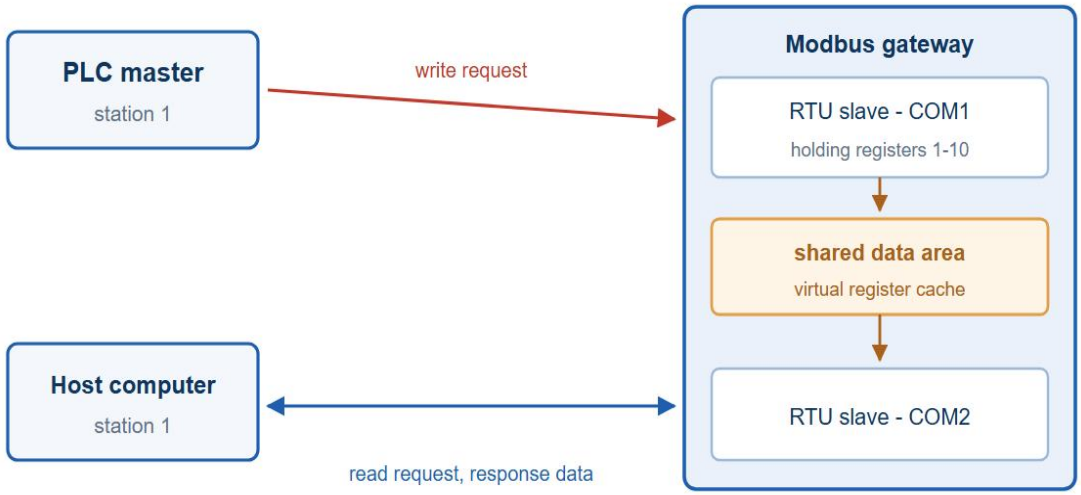

Figure 4 Write on COM1, read on COM2, through the shared register area

1. The PLC issues a write request to slave 1 on COM1, addressing the holding registers that it was configured to use (for example 40001 to 40010).
2. The gateway stores the incoming values in its intermediate variables, which form a shared data area inside the device.
3. The host computer issues a read request to slave 2 on COM2, and receives the same values from the same addresses.

4. The cycle repeats. The gateway's registers behave as a virtual buffer shared by both masters.

A practical consequence: the two sides do not need to agree on any protocol handshake. The only contract between them is the register map. As long as both devices use the same addresses and data types defined in the project, the exchange works, and either side can be replaced without touching the other.

4. Configuration Procedure

The procedure below was carried out in the X2Modbus configuration tool. The screenshots show the original Chinese user interface; the English equivalents of the relevant fields are given in the text and figure captions.

4.1 Creating the Simulator Driver

Create a project and add a virtual driver named **Simulator**. On the driver properties dialog, confirm the driver name and set the polling interval; 1000 ms is the default and is adequate for this application. The Simulator driver does not communicate with any physical device — it only provides the variable container that the two slave channels will expose.

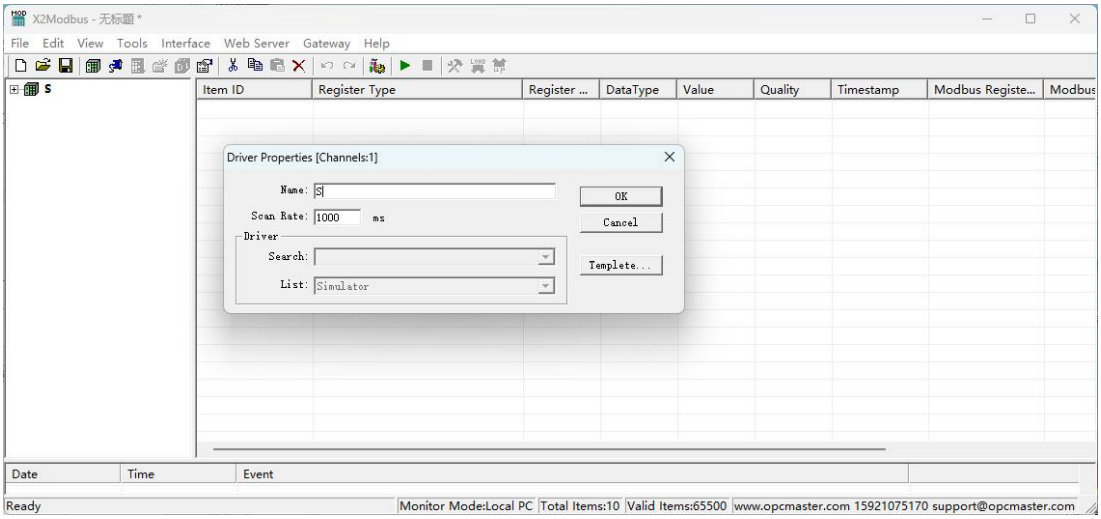

Figure 5 Driver properties: driver name Simulator, polling interval 1000 ms

4.2 Defining the Intermediate Tags

Create an intermediate tag in the tag table and set the register type to **Const**. The register address starts at 1 and continues upward without gaps or duplicates; the data type can be chosen to suit the actual values (Word is used in this example). Ten tags are created here, mapping to addresses 40001 to 40010 on the forwarding side.

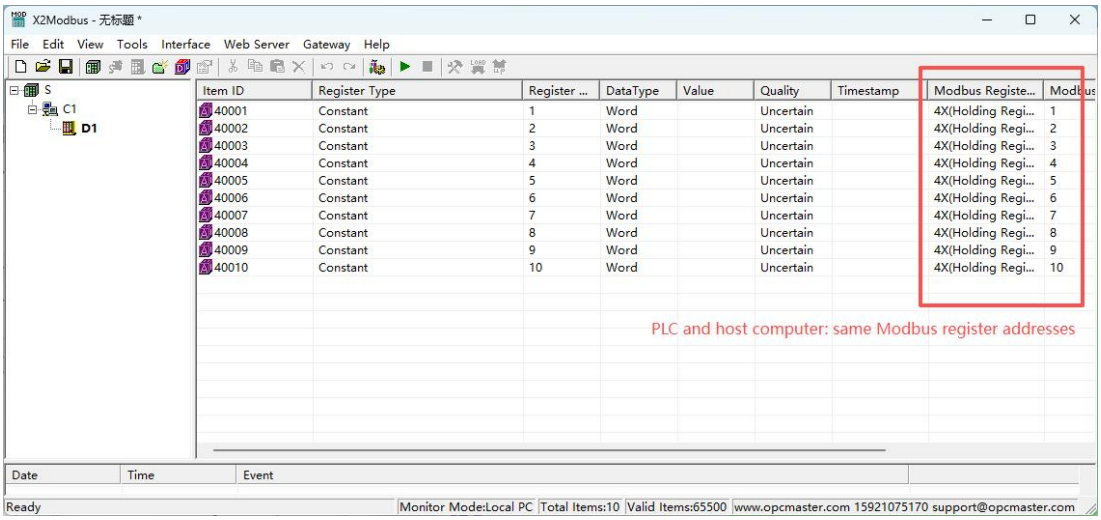

Figure 6 Ten intermediate tags mapped to holding registers 40001–40010

The Modbus register type column in the highlighted area lists each tag as a 4X (holding) register, numbered 1 to 10. These are the addresses that both the PLC and the host computer will operate on, so the register map should be written down before commissioning and used verbatim on both sides. Note that X2Modbus counts addresses from 1 on both ends — decimal base 1 on the acquisition side, and base 1 for the base address of the forwarding side — whereas many third-party tools count from 0. The one-offset difference is the usual reason a correctly wired system still returns the neighbouring register.

4.3 Enabling the RTU Server Channels

The gateway can serve two serial channels as slaves at the same time. In the Modbus Server menu, enable **Modbus RTU1** and **Modbus RTU2**; RTU3 and RTU4

remain available for other channels if the project later needs them.

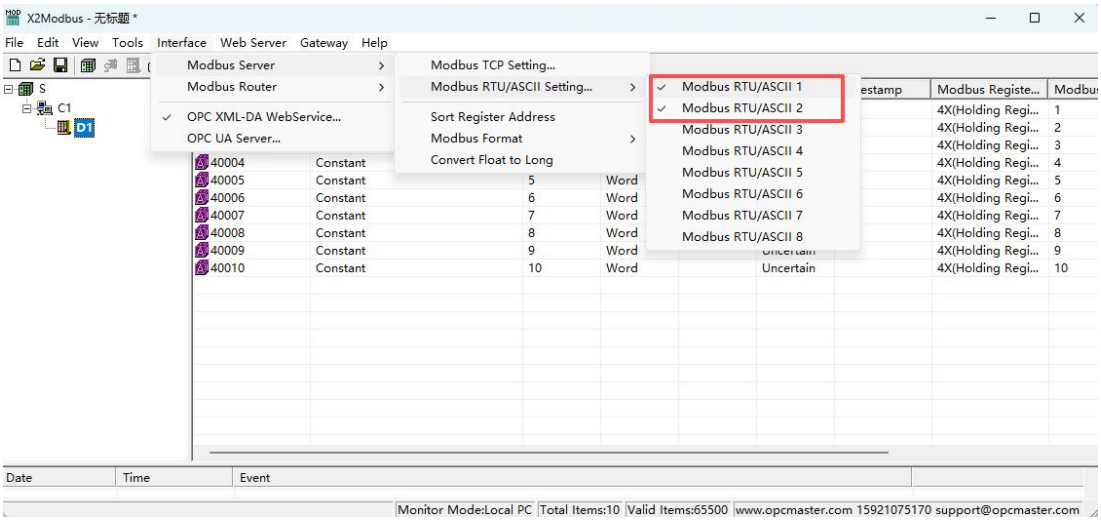

Figure 7 Modbus Server menu: RTU1 and RTU2 enabled

4.4 Setting COM1 for the PLC

Open the Modbus RTU server settings for the first channel. Tick *Enable Modbus RTU server*, select station number **1** and port **COM1**, and set the serial parameters to match the PLC: 9600 bps, 8 data bits, 1 stop bit, no parity. These values must agree with the PLC's own communication settings, otherwise no frame will be accepted.

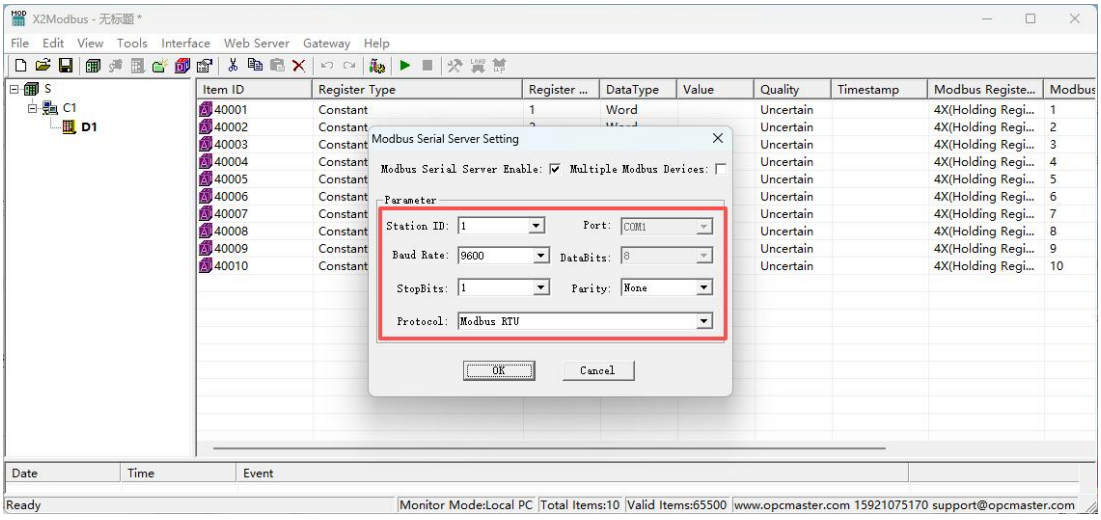

Figure 8 Modbus RTU server settings for COM1

4.5 Setting COM2 for the Host Computer

Repeat the same steps for the second channel, using port **COM2**. The station

number may be identical to the first channel, since the two ports are independent buses. The baud rate and frame format should match whatever the host computer expects; in the verified setup both channels were left at 9600-8-1.

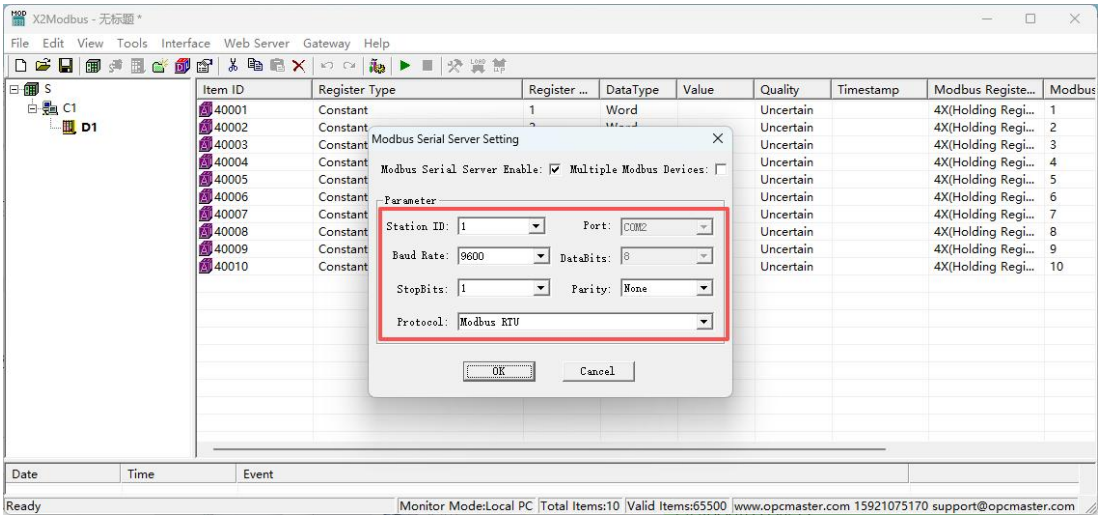

Figure 9 Modbus RTU server settings for COM2

4.6 Optional Modbus TCP Access

The gateway can also publish the same register area over Ethernet. In the Modbus Server menu, open the Modbus TCP settings and enable the server on port 502 with the required station number. Once enabled, a SCADA system or a second host can read the values over the network while the serial channels remain in service.

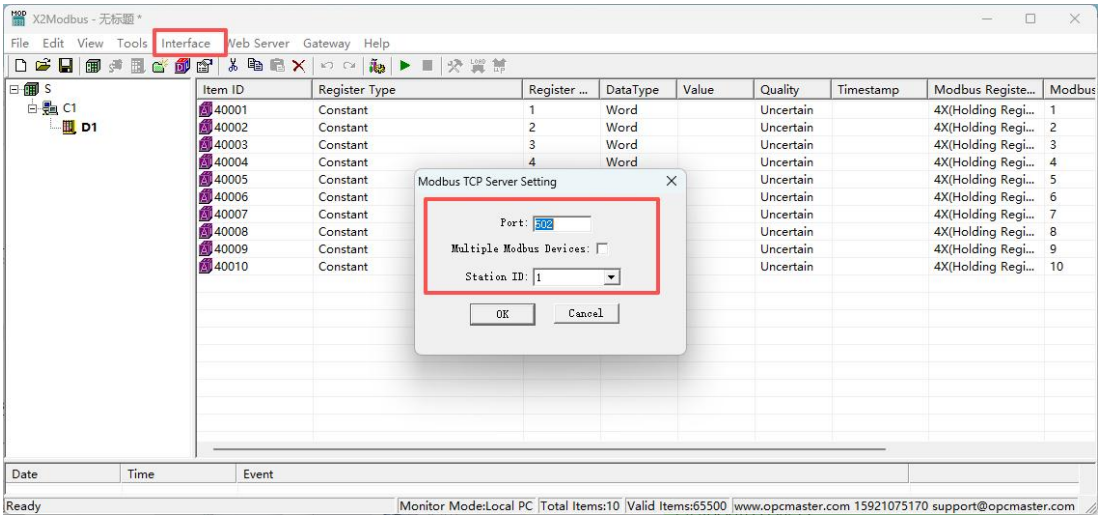

Figure 10 Modbus TCP server settings on port 502

5. Verification

The configuration was verified by simulation before being downloaded to the hardware gateway. Two instances of a Modbus polling tool were started on the same PC: one connected to COM1 to emulate the PLC master, the other connected to COM2 to emulate the host computer. The PLC side was used to write values with a single-register write, and the host side polled the same addresses continuously.

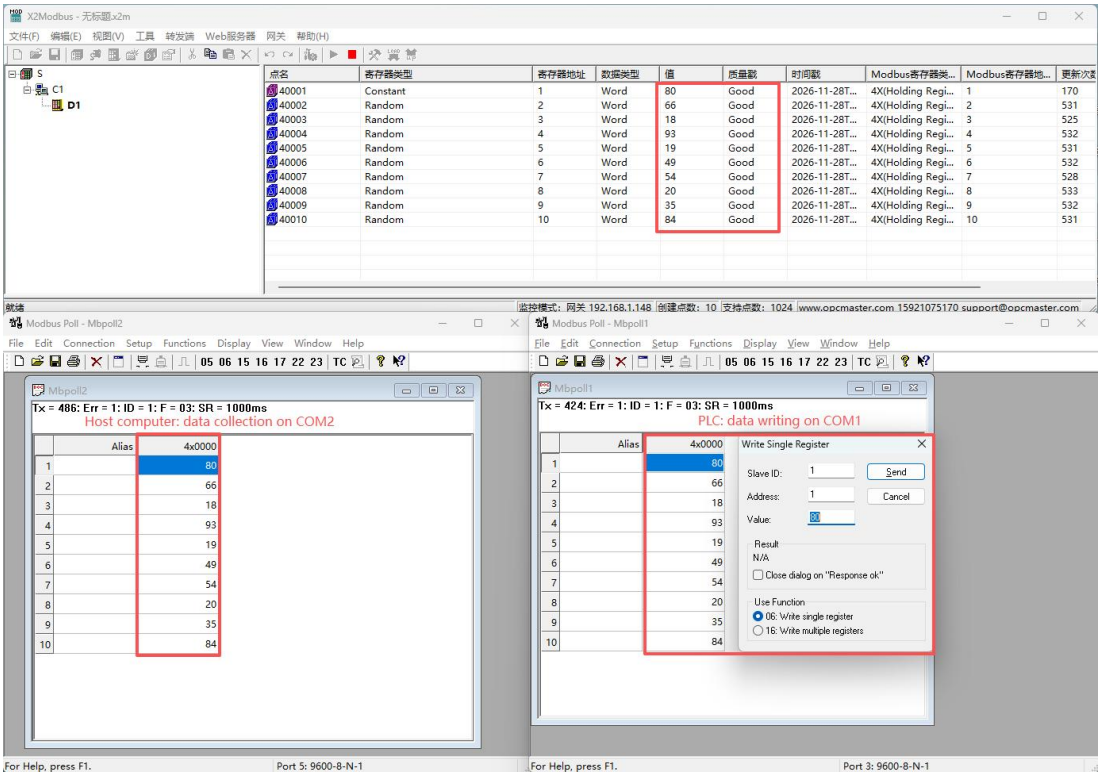

Figure 11 Simulation: values written on COM1 read back on COM2

The result is visible in the tag monitor of the configuration tool as well. Values written from the PLC side (11, 22, 33 and so on) appear in the tag table and are read back on the host side without delay or distortion. The registers inside the gateway act as a virtual buffer shared by both masters, which resolves the master-to-master communication problem.

In production, the same project is downloaded to the MOD1002 or MOD2004 hardware gateway and runs without a PC. The PLC keeps writing values to the gateway,

the host computer keeps collecting them in real time, and neither device needs to know that the other exists. Two additional points are worth checking during commissioning: the register map on both sides must be identical, and the serial parameters of each channel must match its own master rather than the other one.

> **Deployment note.** Because COM1 and COM2 are separate electrical segments, the two masters may use the same station number. If a single bus is used instead, assign different station numbers and keep only one master active per segment.

Document reference: Modbus RTU Master-to-Master Communication Solution, Shanghai Sunfull Automation Co., Ltd, March 2017. English edition prepared for international customers.