World Journal of Engineering Research
Print ISSN: 2959-9865
Online ISSN: 2959-9873
DOI: https://doi.org/10.61784/wjer3038

RESEARCH ON DUAL PLC SIMULATION TECHNOLOGY BASED ON MODBUS TCP COMMUNICATION

Hong Chen
College of Intelligent Construction and Manufacturing, Sichuan University of Technology and Business, Chengdu 610000, Sichuan, China.

Abstract: With the rapid development of industrial automation technology, efficient communication between PLCs has become the key to realizing complex control systems. Based on the MODBUS TCP protocol, this paper studies the simulation method of dual PLC communication, aiming to improve the reliability and real-time performance of industrial control networks. By constructing a dual PLC communication simulation platform based on MODBUS TCP, the data frame structure and communication mechanism of the protocol are analyzed. The logic design and communication configuration of the PLC program are implemented using TIA Portal software, and the simulation platform adopts S7-PLCSIM Advanced. The experimental results show that the simulation system can stably realize data exchange between dual PLCs, providing a reference for multi-PLC collaborative control in industrial fields. This research provides theoretical support and practical basis for the application of MODBUS TCP in industrial automation.
Keywords: PLC; MODBUS TCP; TIA portal; S7-PLCSIM advanced simulation

1 INTRODUCTION

With the rapid development of industrial automation technology, PLC (Programmable Logic Controller) plays a core role in industrial control systems. In complex control scenarios, multi PLC collaborative work has become a trend, and efficient communication protocols are the key to achieving data exchange between devices. MODBUS TCP, as an open and standard industrial communication protocol, is widely used in industrial control networks due to its simplicity[1], reliability, and ease of implementation. However, in practical applications, there are still challenges in the compatibility, real-time communication, and stability of PLCs from different manufacturers. Therefore, researching dual PLC simulation technology based on MODBUS TCP is of great significance[2]. At present, domestic and foreign scholars have achieved certain results in the fields of MODBUS TCP communication and PLC simulation. Foreign research mainly focuses on protocol optimization and real-time performance improvement, such as improving frame structure to enhance communication efficiency; Domestic research focuses on protocol implementation and system integration, such as the development of SCADA systems based on MODBUS TCP. However, there is still limited research on dual PLC collaborative simulation, especially in the areas of communication compatibility, data synchronization, and fault diagnosis for different brands of PLCs, which urgently need to be further explored. Therefore, this article studies the optimization strategy of MODBUS TCP communication by building a dual PLC simulation platform, providing theoretical support and technical reference for the stable operation of industrial control systems.

2 MODBUS TCP PROTOCOL

The MODBUS TCP protocol is an extension of the MODBUS protocol over TCP/IP networks[3]. It encapsulates the data frames of the MODBUS protocol into TCP/IP data packets for transmission. The MODBUS TCP protocol adopts a Client-Server communication mode, where the client sends a request frame to the server, and the server processes it according to the content of the request frame and returns a response frame.
The data frame format of the MODBUS TCP protocol mainly includes the MBAP (MODBUS Application Protocol) header and the data field[4]. The MBAP header contains information such as transaction identifier, protocol identifier, length, and unit identifier, which are used to identify and manage communication transactions. The data field contains specific operation codes and data information, such as operations like reading coils and writing registers.

2.1 Communication Mode

Modbus network communication adopts a master-slave mode, where there can only be one master station in the network and multiple slave stations. Each communication is initiated by the master station, and only one slave station can respond in the network. Each node in the network (including the master station) has a fixed station address, which is unique in the network. It should be noted that the number of nodes in the network is limited, and the specific number depends on the hardware interface. The structure of the Modbus network communication is shown in Figure 1:

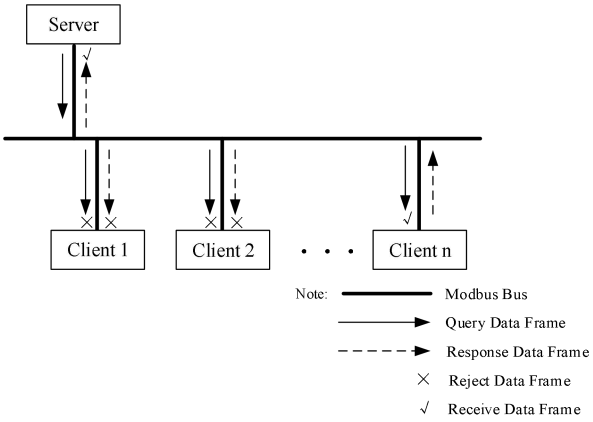

Figure 1 Modbus Network Structure

In Figure 1, it shows a typical Modbus bus network structure. A master-slave communication process is described as follows:
a) When the master station needs to query slave station n, it sends a data frame to the network;
b) All nodes in the network can receive this data frame. These nodes first check whether it is a data frame sent to themselves; if not, they do not respond;
c) If slave station n does not respond, the master station considers that the slave station to be queried is not in the network, and this communication ends;
d) If it is slave station n, it receives this data frame and responds to the master station;
e) Slave station n sends a response data frame to the network, and only the master station receives this response data frame, and this communication ends.
Each node in the Modbus network communicates data in a polling manner, that is, the master station communicates with slave stations one by one in a certain order in a master-slave manner, and each master-slave communication follows the above process.

2.2 Data Frame

This paper mainly discusses Modbus RTU and TCP. Since the data frames of RTU and TCP are basically the same, the data frame of Modbus RTU is mainly described here. The data frame uses hexadecimal encoding, divided into two formats: request data frame and response data frame.T he formats of the register reading frames are shown in Tables 1 and 2:

Table 1 Master Station Read Request Data Frame

Slave Address	Function Code	Starting Address of Register		Number of Registers		CRC Check	
		Hi	Lo	Hi	Lo	Hi	Lo
1 Byte	1 Byte	2 Bytes		2 Bytes		2 Bytes	

Table 2 Slave Station Response to Read Request Data Frame

Slave Address	Function Code	Total Number of Data Bytes	Data 1		⋯		Data n		CRC Check	
			Hi	Lo	Hi	Lo	Hi	Lo	Hi	Lo
1 Byte	1 Byte	1 Byte	2 Bytes		2 Bytes		2 Bytes		2 Bytes	

The format of the register write frame is shown in Table 3:

Table 3 Slave Station Write Request Data Frame

Slave Address	Function Code	Starting Address of Register		Number of Registers		Total Number of Data Bytes	Data 1		⋯		Data n		CRC Check	
		Hi	Lo	Hi	Lo		Hi	Lo	Hi	Lo	Hi	Lo	Hi	Lo
1 Byte	1 Byte	2 Bytes		2 Bytes		1 Byte	2 Bytes		2Bytes		2Bytes		2Bytes	

The Modbus protocol has multiple function codes, each implementing different functions. For example, 0x04 indicates reading values from multiple consecutive registers, and 0x10 indicates writing preset values to multiple consecutive registers. In a data frame, the total number of data bytes represents the number of bytes of the registers to be operated on, which is twice the number of registers.

2.3 Bus Hardware Interface

The experimental platform uses Siemens S7-1500 series PLC (specific model: CPU 1513-1 PN), which supports the Profinet communication protocol and has a built-in string processing function[5]. The software development environment is TIA Portal V16, whose integrated data block editor supports structured variable definition.

3 PROGRAM DESIGN

This research aims to realize that the server is read 10 holding register values and written 10 holding register values by the client, with the data type being integer.

3.1 Server

Add the "MB_SEVER" instruction block to "Program Blocks > OB1" through the path "Communication > Others > MODBUS TCP". A background DB block will be automatically generated when adding, just click OK, as shown in Figure 2 This instruction is used to handle connection requests from Modbus TCP clients and receive responses. The definitions of each pin are shown in Table 4, where the CONNECT pin needs to be filled in by means of symbolic addressing.

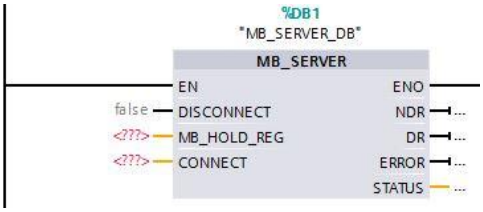

Figure 2 MB_SERVER Instruction Block

Table 4 Slave Station Write Request Data Frame

Parameter	Description
DISCONNET	0 (default): passively establish a communication connection with the client; 1: terminate the connection.
MB_HOLD_REG	Points to the data area of Modbus holding registers. There is no requirement for non-optimized data blocks, and it is generally input in the form of a P# pointer.
CONNECT	Pointer to the connection description structure. Use the TCON_IP_v4 data type.
NDR	0: no newly written data; 1: new data has been written by the Modbus client.
DR	0: data not read; 1: data has been read by the Modbus client.
ERROR	Error bit: 0: no error; 1: error occurs, check STATUS for the cause.
STATUS	Detailed status information of the instruction.

Create the pointer type for the CONNECT pin: first create a new global data block DB1, named "Sever Data", and uncheck "Optimize Block Access". Create a variable "CONNECT" of data type "TCON_IP_v4" in the data block. This type is a system data type, not created in PLC data types, as shown in Figure 3. Among them, "InterfaceId" is the network port hardware identifier, which is 64 (16#40) for the local network port； "ID" is the connection ID for communication between the two PLCs, which is set to 14 here, and must be the same for both communication parties to connect; "Connection Type" is the connection type, which is 16#0B by default for TCP connections; "ActiveEstablished" is the connection establishment type, which is 0 for passive connection as the server; "Remote Address" is the IP address of the client, which can be unspecified; "RemotePort" is the remote port number, generally using the default value 0, meaning not specifying the client port. "LocalPort" is the local port number, which is generally set to a fixed value of 502 for the server.

This simulation aims to realize that the server is read 10 holding register values and written 10 holding register values by the client, so an array of 20 elements with integer data type needs to be added to the "Server Data" data block, and the array name is "Register" as shown in Figure 3.

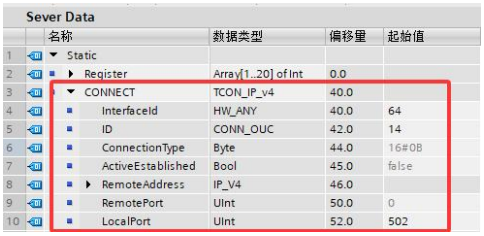

Figure 3 MB_SERVER Sever Data

If only simple data communication is required without observing the data transmission process and completion status, then only the three ports "DISCONNECT", "MB_HOLD_REG" and "CONNECT" need to be assigned parameters, as shown in Figure 4.

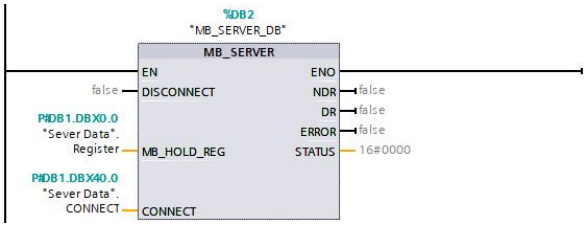

Figure 4 Server-side Program

3.2 Client

Add the "MB_CLIENT" instruction block to the main program Main through the path "Communication > Others > MODBUS TCP", as shown in Figure 5. This instruction is used to handle connection requests from Modbus TCP clients and receive responses. The definitions of each pin are shown in Table 5, among which the 4 pins DISCONNECT, CONNECT, ERROR and STATUS have the same definitions as in Table 4, so they are not listed again.

Table 5 Definition Description of Each Pin of MB_CLIENT

Parameter	Description
REQ	Set input (REQ=true), and the instruction will send a communication request.
MB_MODE	Select the Modbus request mode (read, write or diagnose) or directly select the Modbus function.
MB_DATA_ADDR	Depends on MB_MODE.
MB_DATA_LENTH	Data length.
MB_DATA_PTR	Pointer to the data buffer where data to be received from or sent to the Modbus server is located.
DONE	If the last Modbus job is successfully completed, this bit in the output parameter DONE will be set to "1" immediately.
BUSY	0: no ongoing Modbus request; 1: processing Modbus request.

Create the pointer type for the CONNECT pin: first create a new global data block DB1, named "Client Data", and also uncheck "Optimize Block Access". Create three data in the data block, namely "Write", "Read" and "CONNECT", as shown in Figure 5. Among them, the "Write" variable is an array containing 10 integer data, used to store the data written by the client to the server; the "Read" variable is also an array containing 10 integer data, used to store the data read by the client from the server; the "CONNECT" variable is of "TCON_IP_v4" type, used to store communication configuration parameters. Among them, "InterfaceId" is the network port hardware identifier, which is still 64 (16#40); "ID" is the same as that of the server, i.e., 14; "ConnectionType" is still 16#0B; "ActiveEstablished" is the connection establishment type, which is 1 for active connection as the client; "Remote Address" is the IP address of the server, which needs to be specified as "192.168.2.10" here, set as shown in the red box in Figure 5; "RemotePort" specifies the server port as 502. "LocalPort" is the local port number, which is 0.

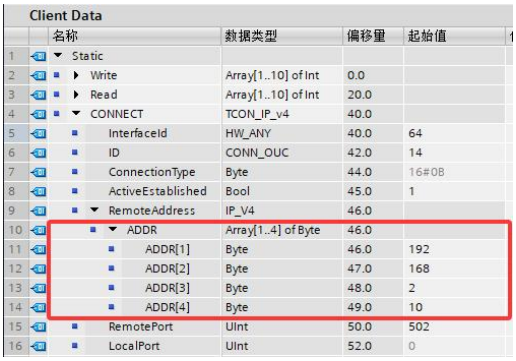

Figure 5 MB_CLIENT Client Data

This research aims to realize that the client reads 10 and writes 10 integer data from/to the server respectively, so two "MB_CLIENT" instruction blocks need to be called here, and read and write operations are performed respectively in a read-write polling manner. The parameters of the two instruction blocks are shown in Figure 6. The first "MB_CLIENT" instruction block is used for write operation, so "MB_MODE is set to 1". The client needs to write 10 consecutive integer data to the server, and the written data is stored in the "Write" array variable in the "Client Data" data block. The mapping interval corresponding to these 10 written data is 40001-40010, so "MB_DATA_ADDR" is set to 40001, and "MB_DATA_LEN" is set to 10; the second "MB_CLIENT" instruction block is used for read operation, "MB_MODE is set to 0". The client needs to read 10 consecutive integer data from the server and store them in the "Read" variable in the "Client Data" data block. The mapping interval corresponding to these 10 read data is 40011-40020, so "MB_DATA_ADDR" is set to 40011, and "MB_DATA_LEN" is set to 10. Due to the adoption of read-write polling communication mode, the "DONE" and "BUSY" of the first instruction block are read out and stored in bit addresses "M 0.0" and "M 0.1" respectively, and the specific polling program is shown in Figure 6.

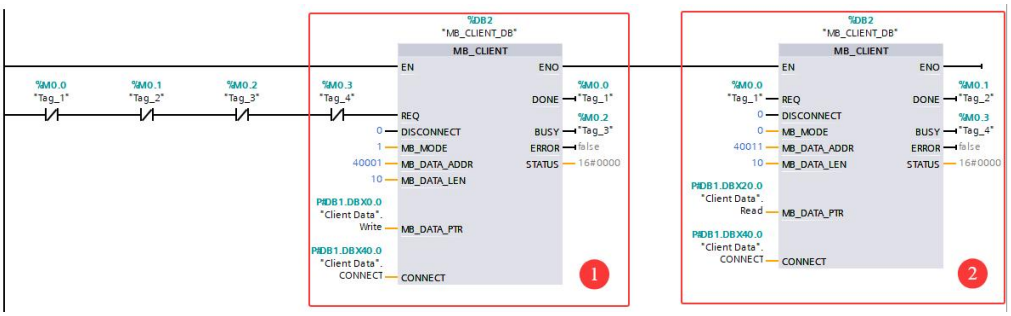

Figure 6 Client-side Program with Read-Write Polling Mode

4 SIMULATION VERIFICATION

This paper selects S7-PLCSIM Advanced developed by Siemens as the core simulation platform[6-8], which represents the most advanced virtual PLC simulation technology in the current industrial automation field. Compared with the traditional S7-PLCSIM simulator, S7-PLCSIM Advanced has significant advantages in architecture design and function implementation.

The construction process of the dual PLC joint simulation environment is as follows:

1) PLCSIM Advanced environment configuration

After starting the PLCSIM Advanced simulation platform, the network parameter configuration must be completed first. Set the IP address of the virtual PLC of the server to 192.168.2.10, the IP address of the virtual PLC of the client to 192.168.2.11, and the IP address of the computer to 192.168.2.1. The IP addresses of the three devices are in the same network segment.

2) TIA Portal project property configuration

In the TIA Portal project properties, the "Block compilation supports simulation" option must be checked (path: Project Tree > Project Name > Properties > Protection > Compilation Support).

3) Virtual PLC operation and data monitoring

After compilation and download, start the virtual CPU through PLCSIM Advanced. Monitor the value changes of the background data block DB1 of the two PLCs. First, test the client writing data to the server. As shown in ① in Figure 7, modify the 5th data value in the "Write" array of the client's background data block OB1 to "23". At this time, the 5th data in the data register of the server's background data block OB1 becomes "23", indicating that the client successfully writes data; then test the client reading data from the server. As shown in ② in Figure 7, modify the 15th data value in the server's register to "11". At this time, the data read by the client is stored in the 5th data of the "Read" array variable in the client's background data block.

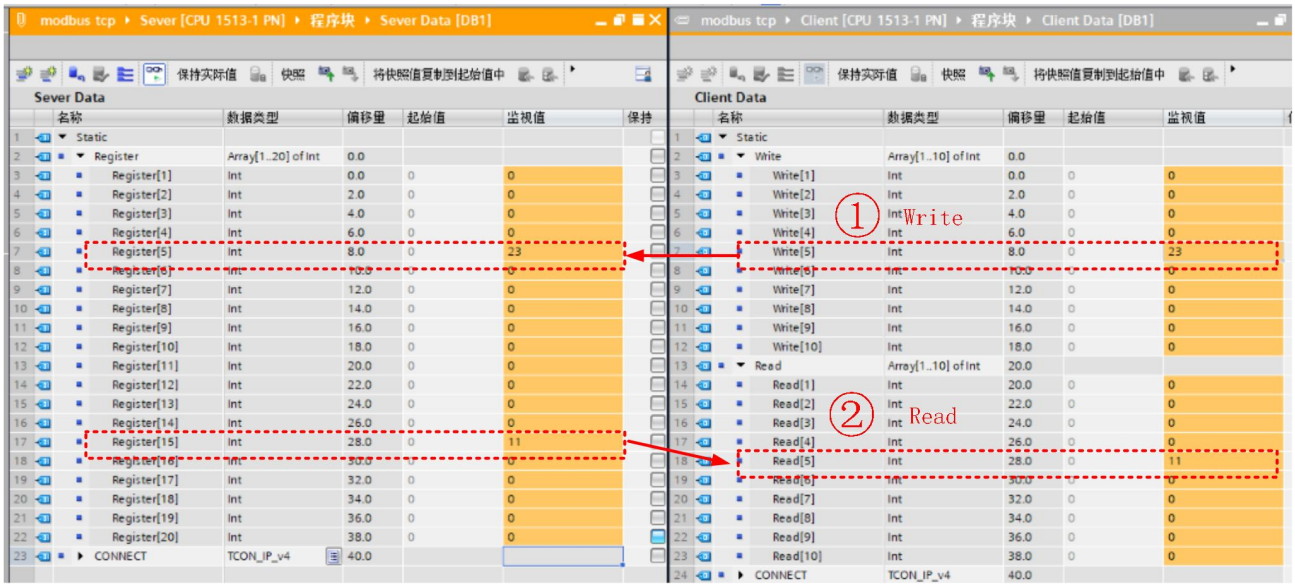

Figure 7 Simulation Test Results

5 CONCLUSION

Based on the interconnection requirements of Industry 4.0 intelligent equipment, this paper constructs a MODBUS TCP communication simulation platform for dual Siemens S7-1500 PLCs. Through the PLCSIM Advanced simulation environment, real-time data interaction between multiple PLC devices is realized. The experimental results show that the simulation system can stably realize data exchange between dual PLCs, providing a reference for multi-PLC collaborative control in industrial fields.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Zhang H, Wang Y. Application of Modbus - TCP Protocol in Industrial Automation Integration. Journal of Industrial Electronics, 45(3): 456-468.

[2] Liu C, Li X. Research on Modbus TCP Communication Optimization for Multi - PLC Systems. Automation and Instrumentation, 34(7): 89-93.

[3] Wu J, Zheng H. Modbus TCP - Based Communication Interface Design for Industrial Robots and PLCs. Robotics and Computer - Integrated Manufacturing, 2024, 58: 101832.

[4] Chen W, Zhao L. Analysis of Modbus TCP Protocol Data Frame Structure and Its Influence on Communication Efficiency. Journal of Communication Technology, 52(4): 678-689.

[5] Zhao Xia. Communication between Siemens PLC and host computer based on TCP/IP protocol. Industrial Control Computer, 2024, 37(11): 33+36.

[6] Wang Q, Sun F. PLC Communication Simulation Based on Modbus TCP Protocol: A Comparative Study of Different Simulation Platforms. Journal of System Simulation, 33(6): 1123-1135.

[7] Chen Hong, Application of TCP/IP Communication between Siemens S7-1500 and ABB Robot. Internet of Things Technology, 2025.

[8] Cao Yuxin, Yang Xiao, Li Jian, et al. Simulation and Debugging Application of Conveying System in a Digital Factory Based on S7-PLCSIM Advanced. Standardization and Quality of Machinery Industry, 2024(09): 43-46+50.