# *Communication Between PLC and Arduino Based on Modbus Protocol*

Yu-cong Kuang
Mechanical Engineering Experimental Center
Guangzhou College of South China University of Technology
Guangzhou, China
kuangyc@gcu.edu.cn

*Abstract*—**This paper explains how to set up the communication between PLC and Arduino based on Modbus Protocol. PLC is a master and Arduino is a slave. A Fx2N-48MR, a Fx2N-485BD, an Arduino mainboard and an I/O Expansion V5.0 are used to finish a test. How to organize the data package in the Modbus RTU is explained. Both the programming of PLC and Arduino are interpreted. The test result and conclusions are given at last.**

***Keywords-Modbus Protocol; Arduino; Fx2N; RS485 Communication***

## I. INTRODUCTION

PLC is one of the pillars of industrial automation, occupies a very important position. Especially Mitsubishi Fx2N series PLC is popular in China colleges and universities. Many automation laboratories are based on this PLC series. Fx2N-485BD communication board is used to implement a RS485 communication on PLC. They are showed in Figure 1.

Arduino is a kind of open source single chip computer. It has a very large user base abroad for its simple programming and rich resources. It is nice to see that Arduino is becoming more and more popular in China. Some university students make innovative projects and realize their interesting ideas through Arduino. Arduino devices can mount to each other like bricks, so it's free to add some other expansion modules to the Arduino mainboard. DFRduino Duemilanove is an Arduino mainboard and I/O Expansion V5.0 is an expansion board. They are showed in Figure 2. DFRduino Duemilanove use Atmel Atmega328 as micro processing controller. It has 13 Digital I/O, and 5 Analog I/O. [1,2]

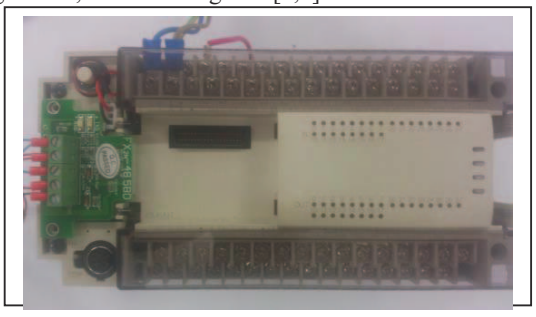

Figure 1. Fx2N-48MR & Fx2N-485BD

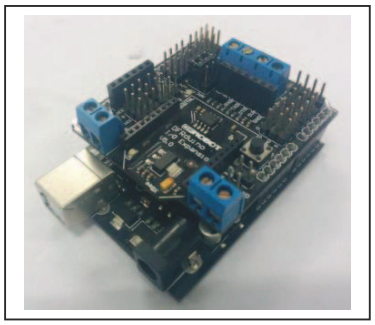

Figure 2. Arduino & I/O Expansion V5.0

## II. MODBUS PROTOCOL OVERVIEW

Modicon Company published the Modbus in 1979. It's a kind of serial communication protocol. Modbus is the standard communication protocol of the industrial field, also is widely used to connect industrial device. Modbus allows devices communicate in a same bus. There can be only one master device and up to 247 slave devices in the bus.

In the Modbus, the master queries and the slave responds. Modbus ASCII and Modbus RTU are the two different modes of Modbus. The information is transmitted by frame. TABLE I. shows us the frame structure of Modbus RTU.[3]

This paper presents the method to implement Modbus RTU protocol based on a Mitsubishi Fx2N-48MR PLC, and an Arduino board and test results are proposed.

TABLE I. FRAME STRUCTURE OF MODBUS RTU

| Start | Address | Function | Data | Check | End |
|---|---|---|---|---|---|
| T1-T2-T3-T4 | 8bits | 8bits | N × 8bits | 16bits | T1-T2-T3-T4 |

### *A. Function03:Read Holding Registers*

With Function03, master device is able to read the specified holding registers from a specified slave device. First, the master queries, and then the data would feed back to the master when the slave responds. Both the frame structures of the query and response are showed in TABLE II. & TABLE III.

978-1-4799-6575-5/14 $31.00 © 2014 IEEE
DOI 10.1109/IMCCC.2014.83

TABLE II. QUERY OF FUNCTION03

| Field Name | Example (Hex) |
|---|---|
| *Slave Address* | 10 |
| *Function* | 03 |
| *Starting Address Hi* | 00 |
| *Starting Address Lo* | 00 |
| *No. of Registers Hi* | 00 |
| *No. of Registers Lo* | 02 |
| *Error Check: CRC Lo* | C7 |
| *CRC Hi* | 4A |

TABLE III. RESPONSE OF FUNCTION03

| Field Name | Example (Hex) |
|---|---|
| *Slave Address* | 10 |
| *Function* | 03 |
| *Byte Count* | 04 |
| *Data Hi (Register 1)* | 00 |
| *Data Lo (Register 1)* | 01 |
| *Data Hi (Register 2)* | 00 |
| *Data Lo (Register 2)* | 00 |
| *CRC Lo* | AA |
| *CRC Hi* | F2 |

### B. *Function16:Write Holding Registers*

With Function16, master device is able to write the holding registers to a specified slave device. Both the frame structures of the query and response are showed in TABLE IV. & TABLE V. In fact, if no error happens in the communication, the response data is no used for us. So, we won't give much attention to the response data of Function16.

## III. TEST DESCRIPTION

We have a test here: Button1 can light up a LED, meanwhile button2 can turn off it. This LED is famous to the Arduino users, it's an on board LED, which is connected to Arduino's pin13. Both button1 and button2 are connected to PLC. Button3 and button4 are connected to Arduino's pin3 and pin4 respectively. They can control the Y0 of PLC on or off, just like what button1 and button2 do to LED.

TABLE IV. QUERY OF FUNCTION16

| Field Name | Example (Hex) |
|---|---|
| *Slave Address* | 11 |
| *Function* | 10 |
| *Starting Address Hi* | 00 |
| *Starting Address Lo* | 02 |
| *No. of Registers Hi* | 00 |
| *No. of Registers Lo* | 02 |
| *Byte Count* | 04 |
| *Data Hi* | 00 |
| *Data Lo* | 02 |
| *Data Hi* | 00 |
| *Data Lo* | 04 |
| *CRC Lo* | 86 |
| *CRC Hi* | B5 |

TABLE V. RESPONSE OF FUNCTION16

| Field Name | Example (Hex) |
|---|---|
| *Slave Address* | 11 |
| *Function* | 10 |
| *Starting Address Hi* | 00 |
| *Starting Address Lo* | 02 |
| *No. of Registers Hi* | 00 |
| *No. of Registers Lo* | 02 |
| *CRC Lo* | E2 |
| *CRC Hi* | 98 |

## IV. COMMUNICATION SET-UP

### A. *Communication Parameters*

It must be confirmed that devices on the same bus have the same communication parameters. We use Modbus RTU protocol here, and the communication parameters are: start-bit 1, stop-bit 2, no parity check, baud rate 9600bps. Fx2N-48MR is the master device and the Arduino is the slave device, which has a slave address 2.

### B. *Wiring*

Figure 3. shows the wiring. The RS485 communication between Fx2N-485BD and the I/O Expansion board is in a half-duplex way.

We use a pull-down resistor between pin3 (pin4) and the GND, use a button between pin3 (pin4) and the +5V. Just like Figure 4. So, pressing button sets input to 5V. That is, press is high, and release is low.[4,5]

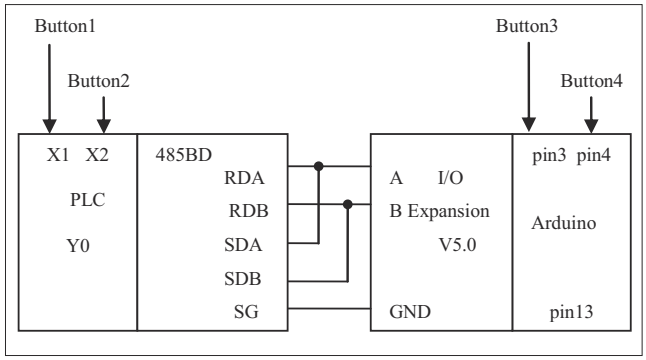


Figure 3. System Wiring

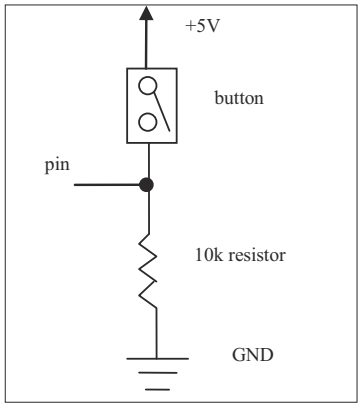


Figure 4. Button Wiring to Arduino

## V. POINTS OF PLC PROGRAMMING

We are going to set up two holding registers in Modbus. The first one is called PTA_VAL, it should be at the address

Guangdong Province Colleges and Universities Construction Project of Demonstration Experiment Teaching Center 2012 (SZL13001) ; The Quality Project of Guangzhou College South China University of Technology 2013 (JY130351)

H0000. The second one is ATP_VAL, it is at the address H0001.

### A. Communication Parameters of PLC

According to the communication parameters above, we use the MOV instruction to send the value of H4C89 to the data register D8120. And make sure M8161 is always on, after this, we can use the data register under the 8 bits processing mode. If these instructions are downloaded to the PLC for the first time, communication parameters work after PLC power off and on again.

### B. RS Instruction

Modbus protocol can't be used directly in Fx2N-48MR, so RS instruction is needed here. RS instruction is used to declare the addresses of sending data and receiving data. These addresses are data registers in PLC actually. E.g., [RS D100 K13 D130 K10] means sending data addresses are D100 to D112, and receiving data addresses are D130 to D139.

### C. Write to a Slave

The main thought is using Function16 to change the value of PTA_VAL. If button1 (X1) is pressed, PTA_VAL shall be written to 1. If button2 (X2) is pressed, PTA_VAL shall be written to 0.Then, Arduino reads the value of PTA_VAL. LED turns on whenever PTA_VAL is 1, LED turns off whenever PTA_VAL is 0.

Use MOV to send the value to PLC data registers D100-D110 according to TABLE VI. After this so called package is finished, set M8122. These data in D100-D110 will transmit to Arduino. Arduino writes the value of PTA_VAL, and then responds. At this time, PLC resets M8122 and sets M8123 automatically, even though there are no these instructions in the ladder chart. In order to prepare for the next communication, rst M8123 is needed in the ladder chart.

TABLE VI. DATA VALUE WHEN MASTER WRITES TO SLAVE

| Field Name | Value: X1 pressed (Hex) | Value : X2 pressed (Hex) | PLC data registers |
|---|---|---|---|
| *Slave Address* | 02 | 02 | D100 |
| *Function* | 10 | 10 | D101 |
| *Starting Address Hi* | 00 | 00 | D102 |
| *Starting Address Lo* | 00 | 00 | D103 |
| *No. of Registers Hi* | 00 | 00 | D104 |
| *No. of Registers Lo* | 01 | 01 | D105 |
| *Byte Count* | 02 | 02 | D106 |
| *Data Hi* | 00 | 00 | D107 |
| *Data Lo* | 01 | 00 | D108 |
| *CRC Lo* | 73 | B2 | D109 |
| *CRC Hi* | 60 | A0 | D110 |

### D. Read from a Slave

The main thought is using Function03 to query the value of ATP_VAL from Arduino. The frame structure is shown in TABLE VII. After this, Arduino responds the value with the frame structure like TABLE VIII. So we can read the ATP_VAL from D133 and D134, if the ATP_VAL is 1, PLC turns Y0 on. If the ATP_VAL is 0, PLC turns Y0 off.

TABLE VII. DATA VALUE WHEN MASTER QUERIES TO READ A SLAVE

| Field Name | Value | PLC data registers |
|---|---|---|
| *Slave Address* | 02 | D100 |
| *Function* | 03 | D101 |
| *Starting Address Hi* | 00 | D102 |
| *Starting Address Lo* | 01 | D103 |
| *No. of Registers Hi* | 00 | D104 |
| *No. of Registers Lo* | 01 | D105 |
| *CRC Lo* | D5 | D106 |
| *CRC Hi* | F9 | D107 |

TABLE VIII. DATA VALUE HEN MASTER READS A SLAVE

| Field Name | Value:button3 pressed (Hex) | Value:button4 pressed (Hex) | PLC data registers |
|---|---|---|---|
| *Slave Address* | 02 | 02 | D130 |
| *Function* | 03 | 10 | D131 |
| *Byte Count* | 02 | 00 | D132 |
| *Data Hi* | 00 | 00 | D133 |
| *Data Lo* | 01 | 00 | D134 |
| *CRC Lo* | 3D | FC | D135 |
| *CRC Hi* | 84 | 44 | D136 |

## VI. POINTS OF ARDUINO PROGRAMMING

### A. SimpleModbusSlave Library

SimpleModbus is a collection of Arduino libraries that enables you to communicate serially using the Modicon Modbus RTU protocol. Both SimpleModbusMaster & SimpleModbusSlave implement Function 03 and 16. There are only two required functions: modbus_update() and modbus_configure().[6]

Download SimpleModbusSlave library and then import it to the Arduino IDE.

modbus_update() is described later. modbus_configure() sets the communication parameters. The Modbus setting part is shown in Figure 5.

```
#include <SimpleModbusSlave.h>
enum{
  PTA_VAL,
  ATP_VAL,
  HOLDING_REGS_SIZE
};
unsigned int holdingRegs[HOLDING_REGS_SIZE];
void setup( ){
  modbus_configure(&Serial,9600,SERIAL_8N2,2,2,
  HOLDING_REGS_SIZE, holdingRegs);
  pinMode(13,OUTPUT);
  pinMode(3,INPUT);
  pinMode(4,INPUT);
}
```

Figure 5. Modbus Setting & Enum Part

### B. Set Up Two Holding Registers

Use eunm, make the PTA_VAL be at the address H0000, make the ATP_VAL be at the address H0001. This part is shown in Fig. 5. After setting up the two holding registers, and the Modbus configuring, it comes to the void loop() part, like Figure 6. In this part, we do some logic control.

```
int val3;
int val4;
void loop(){
  modbus_update();
  if (holdingRegs[PTA_VAL]==0){
  digitalWrite (13,LOW);}
  if (holdingRegs[PTA_VAL]==1){
  digitalWrite(13,HIGH);}
  val3=digitalRead(3);
  val4=digitalRead(4);
  if (val3==HIGH){
  holdingRegs[ATP_VAL]=1;}
  if (val4==HIGH){
  holdingRegs[ATP_VAL]=0;}
}
```

Figure 6. Void loop() part

### C. *Modbus_update()*

This method updates the holdingRegs register array and checks if there is any error happens in the communication.

### D. *Judging the Value of PTA_VAL*

Judge the value of holdingRegs[PTA_VAL], if it is 1, then make the pin13 output high. If it is 0, then make the pin13 output low.

### E. *Giving the Value to ATP_VAL*

If button3 of Arduino is pressed, we give the value H0001 to holdingRegs[ATP_VAL]. If button4 is pressed, give the value H0000 to holdingRegs[ATP_VAL].

## VII. TEST RESULTS AND CONCLUSIONS

### A. *Test Result*

PLC and Arduino communicates to each other successfully. However, there is some data echo in this half-duplex way RS485 communication. So, when PLC queries and reads the data from Arduino, we do some treatment to the data.

### B. *Downloading Program*

Remove the I/O Expansion board from the Arduino mainboard when downloading the program. Because it needs the pin0 and pin1 of Arduino to the download process. Arduino uses these two pins to communicate with PC which we use to do programming. However, I/O Expansion board uses these two pins to communicate with Arduino. If don't remove the I/O Expansion board when downloading the program, the download process won't success.

### C. *Pay Attention to Pin2*

Pin2 is used for the communication control. It's called the enable pin. When it is low, the Arduino is under the receiving mode. When it is high, the Arduino is under the sending mode. Since the pin2 is already used for communication, it can't be used as I/O. So we just leave it alone.


## ACKNOWLEDGMENT

Thanks to Mechanical Engineering Experimental Center, I could finish this paper.


## REFERENCES


[1] M. Banzi: Getting Started with Arduino (O'Reilly Media, America 2009 ).

[2] M. Margolis: Arduino Cookbook (O'Reilly Media, America 2011).

[3] Information on http://zh.wikipedia.org/wiki/Modbus

[4] M. McRoberts: Beginning Arduino (Apress, America 2010).

[5] J.Oxer and H.Blemings: Practical Arduino (Apress, America 2009).

[6] Information on https://code.google.com/p/simple-modbus/