N9H20/M487 Modbus Quick Guide

V1.00.001

Publication Release Date: Mar. 2020

Support Chips:

W55FA Series
N9H2 Series

Support Platforms:

Windows

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1. Introduction

1.1. Modbus Introduction

Modbus is a serial communications protocol originally published by Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs). Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices. Modbus is popular in industrial environments because it is openly published and royalty-free. Currently Modbus has four kinds of protocols, Modbus RTU, Modbus ASCII, Modbus TCP/IP and Modbus plus.

In the system, N9H20 is the master and two M487s are the slave. The ID of slave address are 1 and 2. They use the protocol Modbus RTU for communication of RS485. The commands are compatible with Modbus protocol.

2. Opertaions

2.1. The Solution of N9H20

N9H20 works as master, There are three files RS485_demo.c, RS485_UART.c and ModbusMaster.c.

Within the file RS485_demo.c, it is the main entry and hardware setting. The UART protocol is 9600 baud rate, 8 data bit, 2 stop bits and none parity. We use UART input to send the command to the slave as follows. There are some APIs OpenDevice(uint8_t id), ReadDevice(uint8_t id),DeviceLedOn(uint8_t id) and DeviceLedOff(uint8_t id) will execute some functions from the file ModbusMaster.c

```
while (1)
    {
                    sysprintf(" [1] Open sensor 1\n");
                    sysprintf(" [2] Read sensor 1\n");
                    sysprintf(" [3] LED ON 1\n");
                    sysprintf(" [4] LED OFF 1\n");
                    sysprintf(" [5] Open sensor 2\n");
                    sysprintf(" [6] Read sensor 2\n");
                    sysprintf(" [7] LED ON 2\n");
                    sysprintf(" [8] LED OFF 2\n");

                    u32Item = sysGetChar();
                    if (u32Item == '1')
                    {
                            :
                            :
                    }
                    :
                    :
    }
```

Within the file RS485_UART.c, its purpose executes the RS485 protocol by using UART hardware, RS485 is s half-duplex.

Within the file ModbusMaster.c, the source code is from the website https://github.com/4-20ma/ModbusMaster, we modify the code to meet our request. The request command of open device is as follows.

Field Name	(Hex)
Slave address	1 or 2
Function code	5
Address high byte	20
Address low byte	02
Output value high byte	FF
Output value low byte	00

CRC high byte	TBD
CRC low byte	TBD

The correct response is the same as the request command.

The error response will be shown as follows.

Field Name	(Hex)
Slave address	1 or 2
Error code	85
Exception code	01 or 02 or 03 or 04
CRC high byte	TBD
CRC low byte	TBD

The request command of “device LED ON” is as follows.

Field Name	(Hex)
Slave address	1 or 2
Function code	5
Address high byte	20
Address low byte	03
Output value high byte	FF
Output value low byte	00
CRC high byte	TBD
CRC low byte	TBD

The correct response is the same as the request command.

The error response will be shown as follows.

Field Name	(Hex)
Slave address	1 or 2
Error code	85
Exception code	01 or 02 or 03 or 04
CRC high byte	TBD
CRC low byte	TBD

The request command of “device LED OFF” is as follows.

Field Name	(Hex)
Slave address	1 or 2
Function code	5
Address high byte	20
Address low byte	03
Output value high byte	00
Output value low byte	00
CRC high byte	TBD
CRC low byte	TBD

The correct response is the same as the request command.

The error response will be shown as follows.

Field Name	(Hex)
Slave address	1 or 2
Error code	85

Exception code	01 or 02 or 03 or 04
CRC high byte	TBD
CRC low byte	TBD

The request command of "Read sensor of device" is as follows.

Field Name	(Hex)
Slave address	1 or 2
Function code	4
Address high byte	20
Address low byte	00
Output value high byte	00
Output value low byte	02
CRC high byte	TBD
CRC low byte	TBD

The correct response will be shown as follows.

Field Name	(Hex)
Slave address	1 or 2
Function code	4
Byte count	4
Data 1 high byte	TBD
Data 1 low byte	TBD
Data 2 high byte	TBD
Data 2 low byte	TBD
CRC high byte	TBD
CRC low byte	TBD

Data1/100 is the degree of Celsius.
Data2/100 is the degree of Fahrenheit.

The error response will be shown as follows.

Field Name	(Hex)
Slave address	1 or 2
Error code	84
Exception code	01 or 02 or 03 or 04
CRC high byte	TBD
CRC low byte	TBD

Firstly the definition of one frame for Modbus is shown as follows. ADU (Application Data Unit) and PDU (Protocol Data Unit).

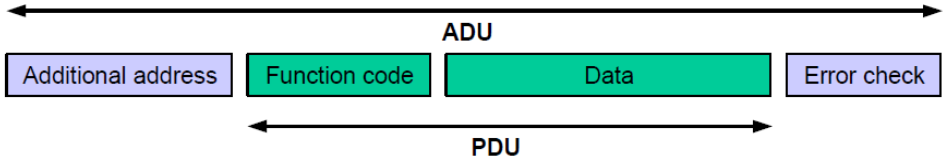
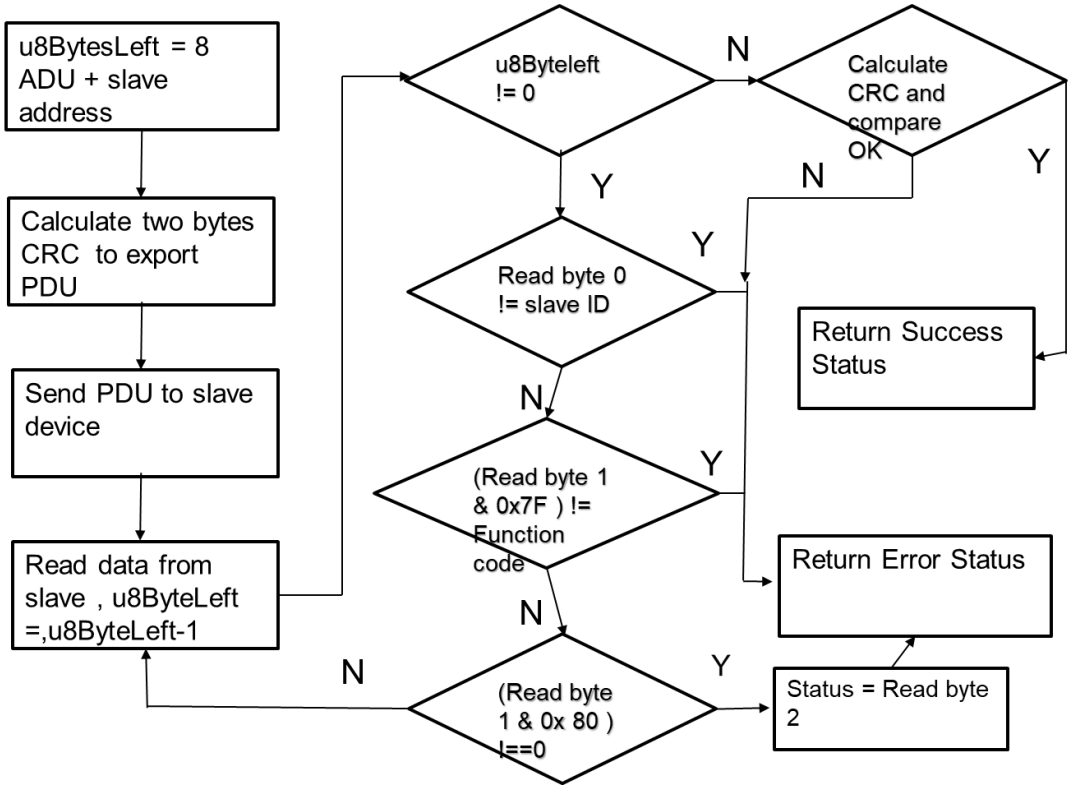


Figure 3: General MODBUS frame

Within the file ModbusMaster.c, there is one function ModbusMasterTransaction(). Its purpose is to send the request command and obtain the feedback command. It is the kernel of Modbus master. the flowchart is shown as follows.

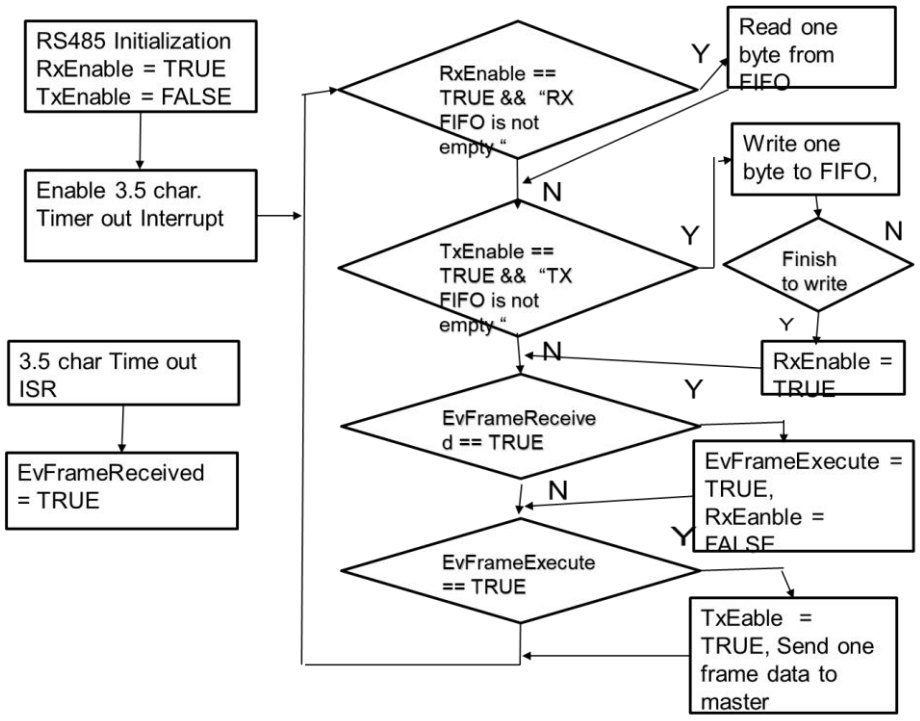

u8BytesLeft = 8
ADU + slave address
Calculate two bytes CRC to export PDU
Send PDU to slave device
Read data from slave , u8ByteLeft =,u8ByteLeft-1
u8Byteleft != 0
N
Y
Calculate CRC and compare OK
N
Y
Read byte 0 != slave ID
Y
N
Return Success Status
(Read byte 1 & 0x7F) != Function code
Y
N
Return Error Status
(Read byte 1 & 0x 80) !==0
N
Y
Status = Read byte 2

2.2. The Solution of M487

M487 works as slave, the source code of Modbus slave is from FreeModbus. The code is more complex than the master, but we only focus one the three files, main.c, portserial.c and porttimer.c.
Within the file main.c, there is one main statements as follows.

```
if ( (eStatus = eMBInit( MB_RTU, usSlaveID, 0, 9600, MB_PAR_NONE )) !=    MB_ENOERR )
      goto FAIL_MB;
else if ( (eStatus = eMBEnable(    ) ) != MB_ENOERR )
      goto FAIL_MB_1;
else {
      for( ;; )
      {
            xMBPortSerialPolling();
            if ( eMBPoll( ) != MB_ENOERR ) break;
      }
}
```

It includes one loop, we use the polling instead of interrupt for RS485 access.
For the loop, we use the following flow chart to describe roughly.



You could refer to the source code for more details.

There is one thing to notice, when the master sends the address value ADDR, the slave receives the value to be ADDR+1. Modbus has four kinds of data model as follows

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

So there are four callback functions to response the four data model. The API eMBRegInputCB( ) is for Input Registers, the API eMBRegHoldingCB( ) is for Hold Registers, the API eMBRegCoilsCB( ) is for Coils, and the API eMBRegDiscreteCB( ) is for Discretes Input. User could add more functions within the related call back function. For example, N9H20 sends the command “device LED ON”, then the call back function eMBRegCoilsCB( ) must be programmed to feedback. N9H20 sends the command “Read sensor of device”, the call back function eMBRegInputCB( ) must be programmed to feedback.

Within the file portserial.c, we set RS485 access by polling instead of interrupt because of the source code of FreeModbus.

Within the file porttimer.c, if the execution time of received character is greater than of 3.5 character timer, then the received character does not belong to the frame data. The frame data of Modbus RTU is shown as follows.

Start	Address	Function	Data	CRC	End
3.5 Char time	8 Bit	8 Bit	N * 8Bit	16 Bit	3.5 Char time

3. Revision History

Version	Date	Description
V1.00.001	Mar. 31, 2020	● Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.