Khái niệm cơ bản về giao thức Modbus

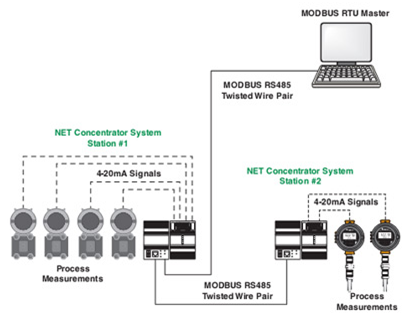
**Sau các khái niệm cơ bản về các chuẩn giao tiếp vật lý cơ bản mà chúng ta hay thấy như RS232, RS485 (còn gọi là serial), hôm nay chúng ta cùng trao đổi về giao thức Modbus, một giao thức được sử dụng rộng rãi trong công nghiệp. Nói một cách dễ hiểu, chúng ta có thể tưởng tượng RS232, RS485 là phần xác, còn giao thức như Modbus là phần hồn .**

***Vậy, giao thức Modbus là gì?***

MODBUS do Modicon (hiện nay thuộc Schneider Electric) phát triển năm 1979, là một phương tiện truyền thông với nhiều thiết bị thông qua một cặp dây xoắn đơn. Ban đầu, nó hoạt động trên RS232, nhưng sau đó nó sử dụng cho cả RS485 để đạt tốc độ cao hơn, khoảng cách dài hơn, và mạng đa điểm (multi-drop). MODBUS đã nhanh chóng trở thành tiêu chuẩn thông dụng trong ngành tự động hóa, và Modicon đã cho ra mắt công chúng như một protocol miễn phí.

MODBUS là một hệ thống “chủ - tớ”, “chủ” được kết nối với một hay nhiều “tớ”. “Chủ” thường là một PLC, PC, DCS, hay RTU. “Tớ” MODBUS RTU thường là các thiết bị hiện trường, tất cả được kết nối với mạng trong cấu hình multi-drop. Khi một chủ MODBUS RTU muốn có thông tin từ thiết bị, chủ sẽ gửi một thông điệp về dữ liệu cần, tóm tắt dò lỗi tới địa chỉ thiết bị. Mọi thiết bị khác trên mạng sẽ nhận thông điệp này nhưng chỉ có thiết bị nào được chỉ định mới có phản ứng.

Các thiết bị trên mạng MODBUS không thể tạo ra kết nối; chúng chỉ có thể phản ứng. Nói cách khác, chúng “lên tiếng” chỉ khi được “nói tới”. Một số nhà sản xuất đang phát triển các thiết bị lai ghép hoạt động như các tớ MODBUS, tuy nhiên chúng cũng có “khả năng viết”, do đó làm cho chúng trở thành các thiết bị chủ ảo.



***Các chuẩn modbus nào đang được sử dụng phổ biến?***

Hiện nay, có 03 chuẩn modbus đang được sử dụng phổ biến trong công nghiệp - tự động hóa là: Modbus RTU, Modbus ASCII, Modbus TCP

Vậy, 03 chuẩn này có gì giống và khác nhau?

Tất cả thông điệp được gửi dưới cùng một format. Sự khác nhau duy nhất giữa 3 loại MODBUS là cách thức thông điệp được mã hóa. Cụ thể:

* Modbus ASCII:

Mọi thông điệp được mã hóa bằng hexadeci-mal, sử dụng đặc tính ASCII 4 bit. Đối với mỗi một byte thông tin, cần có 2 byte truyền thông, gấp đôi so với MODBUS RTU hay MODBUS/TCP. Tuy nhiên, MODBUS ASC II chậm nhất trong số 3 loại protocol, nhưng lại thích hợp khi modem điện thoại hay kết nối sử dụng sóng radio do ASC II sử dụng các tính năng phân định thông điệp. Do tính năng phân định này, mọi rắc rối trong phương tiện truyền dẫn sẽ không làm thiết bị nhận dịch sai thông tin. Điều này quan trọng khi đề cập đến các modem chậm, điện thoại di động, kết nối ồn hay các phương tiện truyền thông khó tính khác.

* Modbus RTU: Dữ liệu được mã hóa theo hệ nhị phân, và chỉ cần một byte truyền thông cho một byte dữ liệu. Đây là thiết bị lí tưởng đối với RS 232 hay mạng RS485 đa điểm, tốc độ từ 1200 đến 115 baud. Tốc độ phổ biến nhất là 9600 đến 19200 baud. MODBUS RTU là protocol công nghiệp được sử dụng rộng rãi nhất, do đó hầu như trong bài viết này chỉ tập trung đề cập đến cơ sở và ứng dụng của nó.
* Modbus TCP: MODBUS/TCP đơn giản là MODBUS qua Ethernet. Thay vì sử dụng thiết bị này cho việc kết nối với các thiết bị tớ, do đó các địa chỉ IP được sử dụng. Với MODBUS/TCP, dữ liệu MODBUS được tóm lược đơn giản trong một gói TCP/IP. Do đó, bất cứ mạng Ethernet hỗ trợ MODBUS/ IP sẽ ngay lập tức hỗ trợ MODBUS/TCP.

***Modbus gateway là gì?***

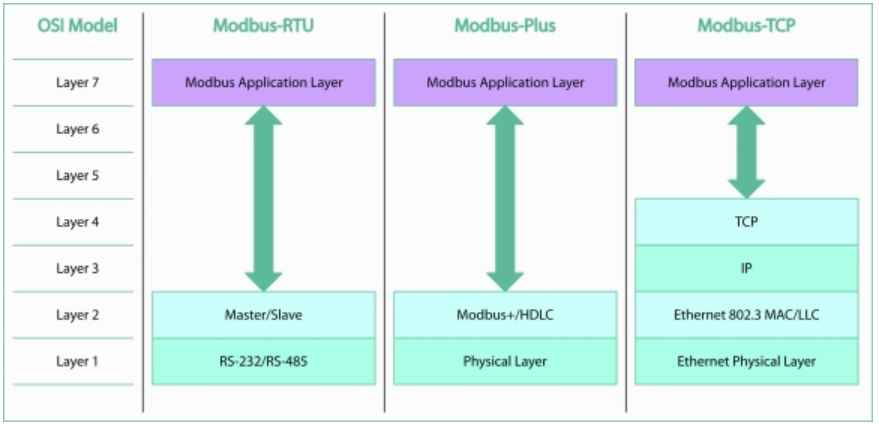
Modbus gateway là một thiết bị cho phép chuyển đổi qua lại giữa giao thức [Modbus RTU và Modbus TCP](http://bkaii.com.vn/san-pham/usr/usr-tcp232-410s-bo-chuyen-doi-rs232-rs485-sang-ethernet). Thông thường thiết bị sẽ có 01 cổng serial (RS232/RS485) và 01 cổng Ethernet.

Tại sao lại là 2 giao thức này? vì hai giao thức này phổ biến, đại diện cho 02 loại cổng vật lý là serial (RS232/RS485) và ethernet (cổng RJ45). Thiết bị công nghiệp dùng ở thị trường hiện nay, nhiều thiết bị chỉ hỗ trợ Modbus RTU, hoặc chỉ hỗ trợ Modbus TCP, hoặc hỗ trợ cả 2. Do đó, để kết nối các thiết bị trường này vào hệ thống modbus chung của nhà máy, xí nghiệp thì sẽ cần 01 thiết bị phiên dịch được gọi là Modbus Gateway.

Khái niệm cơ bản về giao thức Modbus RTU

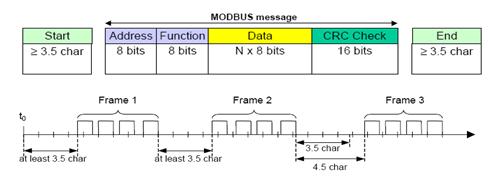
Giao thức Modbus RTU là một giao thức mở, sử dụng đường truyền vật lý RS-232 hoặc RS485 và mô hình dạng Master-Slave. Đây là một giao thức được sử dụng rộng rãi trong nhiều lĩnh vực như BMS (Building Management Systems), tự động hóa, công nghiệp, điện lực,.... Chắc hẳn sẽ có bạn tự hỏi, tại sao giao thức Modbus này lại thông dụng như thế, đi đến đâu, đụng vào thiết bị gì thì hầu như cũng có giao thức này? Vâng, câu trả lời cho câu hỏi trên chắc chỉ cần gói gọn trong vài từ: Ổn định - Đơn giản - dễ dùng.

Modbus được coi là giao thức truyền thông hoạt động ở tầng "Application", cung cấp khả năng truyền thông Master/Slave giữa các thiết bị được kết nối thông qua các bus hoặc network. Trên mô hình OSI, Modbus được đặt ở lớp 7. Modbus được xác định là một giao thức hoạt động theo "hỏi/đáp" và sử dụng các "function codes" tương ứng để hỏi đáp.



**Cấu trúc bản tin Modbus RTU**

Một bản tin Modbus RTU bao gồm: 1 byte địa chỉ -  1 byte mã hàm - n byte dữ liệu - 2 byte CRC như hình ở dưới:



Chức năng và vai trò cụ thể như sau:

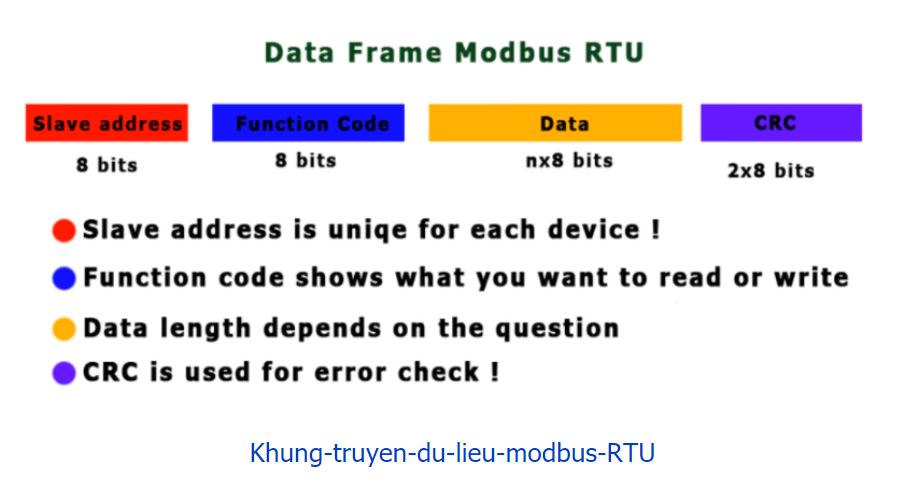
* **Byte địa chỉ**: xác định thiết bị mang địa chỉ được nhận dữ liệu (đối với Slave) hoặc dữ liệu nhận được từ địa chỉ nào (đối với Master). Địa chỉ này được quy định từ 0 - 254
* **Byte mã hàm**: được quy định từ Master, xác định yêu cầu dữ liệu từ thiết bị Slave. Ví dụ mã 01: đọc dữ liệu lưu trữ dạng Bit, 03: đọc dữ liệu tức thời dạng Byte, 05: ghi dữ liệu 1 bit vào Slave, 15: ghi dữ liệu nhiều bit vào Slave ...
* **Byte dữ liệu**: xác định dữ liệu trao đổi giữa Master và Slave.
  + Đọc dữ liệu:
    - Master:  2 byte địa chỉ dữ liệu - 2 byte độ dài dữ liệu
    - Slave: 2 byte địa chỉ dữ liệu - 2 byte độ dài dữ liệu - n byte dữ liệu đọc được
  + Ghi dữ liệu:
    - Master: 2 byte địa chỉ dữ liệu  - 2 byte độ dài dữ liệu - n byte dữ liệu cần ghi
    - Slave: 2 byte địa chỉ dữ liệu - 2 byte độ dài dữ liệu
* **Byte CRC**: 2 byte kiểm tra lỗi của hàm truyền. cách tính giá trị của Byte CRC 16 Bit

ModBus RTU khung Frame data.

Frame: trong chế độ RTU chứa 8 bit địa chỉ , 8 bit mã hàm , data < 253 byte , 16 bit check , Mỗi Byte không trễ quá 1,5 T (T thời gian một kí tự truyền đi )

Truyền một Byte đi bằng giao thức UART với chế độ truyền 8 bit.

CRC Check: (Cyclic Redundancy – Check) sử dụng thuật toán CRC để tạo mã check lỗi.



Ví dụ: Nếu tôi muốn hỏi một câu hỏi để đọc dữ liệu của 5 input số, nó là một danh sách bắt đầu từ một danh sách số. Đơn giản tôi muốn đọc số từ 1, 2, 3, 4, và 5, slaves ở địa chỉ 17.

Request: 11 01 00 00 00 05 FE 99 from Mater (Computer)

Respond: 11 01 01 00 94 88 from Slaves (Arduino)

**Sự khác nhau giữa Modbus RTU và Modbus TCP**

Sự khác nhau cơ bản giữa MODBUS RTU và MODBUS TCP ( còn được gọi là modbus IP, Modbus Ethernet hay Modbus TCP/IP) là Modbus TCP chạy ở cổng vật lý Ethernet và Modbus RTU thì chạy ở cổng vật lý serial nối tiếp ( RS232 hoặc RS485).

**MODBUS RTU Memory Map**

|  |  |  |
| --- | --- | --- |
| Modbus RTU  Data Type | Common name | Starting address |
| Modbus Coils | Bits, binary values, flags | 00001 |
| Digital Inputs | Binary inputs | 10001 |
| Analog Inputs | Binary inputs | 30001 |
| Modbus Registers | Analog values, variables | 40001 |

**Sự khác nhau giữa Modbus RTU và Modbus ASCII**

Đây là hai chế độ truyền dữ liệu cơ bản trong giao thức Modbus khi sử dụng đường truyền nối tiếp là: ASCII và RTU.  Mỗi một chuẩn sẽ có một cách mã hóa tin nhắn khác nhau, cho dù đều alf chuẩn Modbus chung. Ví dụ, Modbus ASCII cho phép người đọc có thể đọc trực tiếp tin nhắn trong bản tin. Nhưng với Modbus RTU, thì nội dung data đã được mã hóa nhị phân và không thể đọc được trong quá trình giám sát. Một điểm đặc biệt trong giao thức Modbus là trong một đường truyền dẫn Modbus, tất cả các giao thức phải là giống nhau, có nghĩa là Modbus ASCII không thể giao tiếp với modbus RTU và ngược lại.

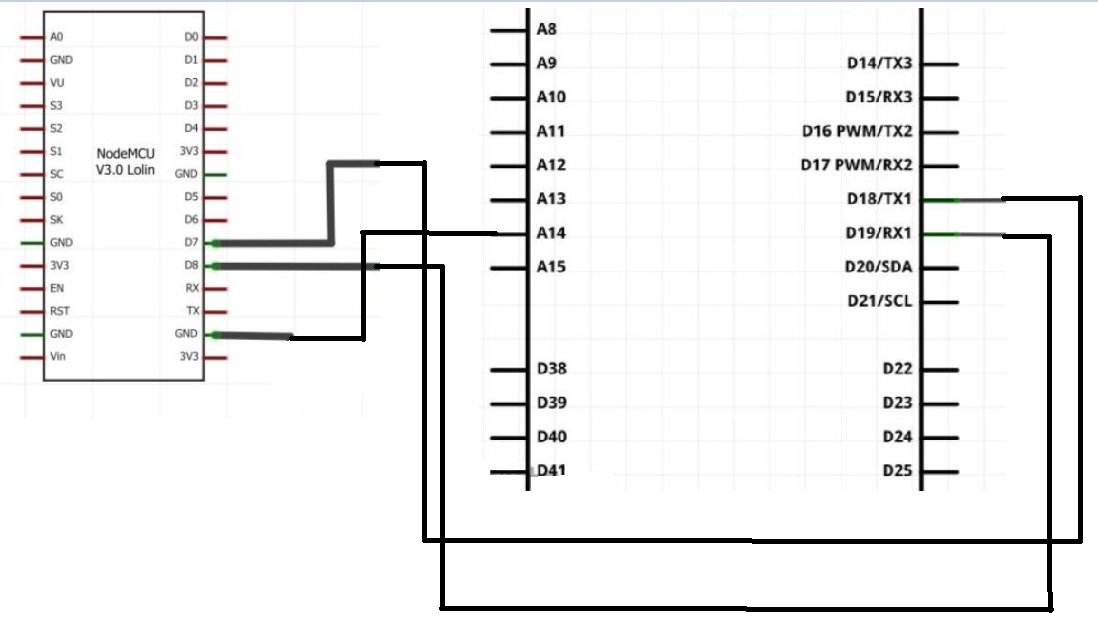
**Các tính chất của Modbus ASCII và Modbus RTU**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Modbus/ASCII** | | **Modbus/RTU** | |
| Characters | ASCII 0…9 and A..F | | Binary 0…255 | |
| Error check | LRC Longitudinal Redundancy Check | | CRC Cyclic Redundancy Check | |
| Frame start | character ‘:‘ | | 3.5 chars silence | |
| Frame end | characters CR/LF | | 3.5 chars silence | |
| Gaps in message | 1 sec | | 1.5 times char length | |
| Start bit | 1 | | 1 | |
| Data bits | 7 | | 8 | |
| Parity | even/odd | none | even/odd | none |
| Stop bits | 1 | 2 | 1 | 2 |

Ở đây project của mình sẽ gồm:

Truyền theo kiểu modbus.

Phần nối dây



Ở phần Node mcu slaves

#include <SimpleTimer.h>

#include <modbus.h>

#include <modbusDevice.h>

#include <modbusRegBank.h>

#include <modbusSlave.h>

#include <SoftwareSerial.h>// Modbus RTU pins D7(13),D8(15) RX,TX

modbusDevice regBank;

modbusSlave slave;

int s = 0;

int t = 0;

void setup()

{

// You can also specify server:

//Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 80);

//Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,100), 8080);

timer.setInterval(1000L, processes);//Real sensor in 1s

//Assign the modbus device ID.

regBank.setId(1);

// Holding registers

regBank.add(40005);

regBank.add(40006);

regBank.add(40007);

regBank.add(40008);

regBank.add(40009);

regBank.add(40010);

slave.\_device = &regBank;

Serial.begin(9600);

delay(100);

// Initialize the serial port for coms at 9600 baud

slave.setBaud(9600);

delay(100);

Serial.println("Connected ");

Serial.println("Modbus RTU Slave Online");

}

void loop()

{

//Goi du lieu cho mater

SetAlarm = String(SGio)+ ":" +String(SPhut)+ ":" +String(SGiay);

NotAlarm = String(FGio)+ ":" +String(FPhut)+ ":" +String(FGiay);

// Set Alarm

Serial.println("Set Alarm music: ");

Serial.print(SetAlarm);

regBank.set(40005,SetAlarm);

delay(20);

// Not Alarm

Serial.println("Not Alarm music: ");

Serial.print(NotAlarm);

regBank.set(40006,NotAlarm);

delay(20);

// Set volume music

Serial.println("Volume music: ");

Serial.print(vl);

regBank.set(40007,vl);

delay(20);

// Set control music

Serial.println("Control music: ");

Serial.print(control);

regBank.set(40008,control);

delay(20);

// Set Floder music

Serial.println("Floder music: ");

Serial.print(Floder);

regBank.set(40010,Floder);

delay(20);

slave.run();

timer.run();

// Set Song music

Serial.println("Song music: ");

Serial.print(song);

regBank.set(40010,song);

delay(20);

slave.run();

timer.run();

}

Mater Arduino lấy dữ liệu

#include <ModbusMaster232.h>

#include <SoftwareSerial.h> // Modbus RTU pins D7(13),D8(15) RX,TX

// Instantiate ModbusMaster object as slave ID 1

ModbusMaster232 node (1);

int Floder = 0;

int song = 0;

int control = 0;

int vl = 0;

int SetAlarm = 0;

int NotAlarm = 0;

void setup() {

Serial.begin (9600);

Delay (100);

node.begin (9600); // Modbus RTU

delay (1000);

Serial.println ("Connected ");

Serial.println ("Modbus RTU Master Online");

}

void loop(){

//Lay nhung gia tri tu slave goi ve

//Set Alarm

Serial.println (" ");

node.readHoldingRegisters (4, 1);

Serial.print ("[4] Set Alarm music ");

SetAlarm = node.getResponseBuffer (0);

Serial.println (SetAlarm);

node.clearResponseBuffer ();

//Not Alarm

Serial.println (" ");

node.readHoldingRegisters (5, 1);

Serial.print ("[5] Not Alarm music ");

NotAlarm = node.getResponseBuffer (0);

Serial.println (NotAlarm);

node.clearResponseBuffer ();

//Volume

Serial.println (" ");

node.readHoldingRegisters (6, 1);

Serial.print ("[6] Volume music ");

vl = node.getResponseBuffer(0);

Serial.println (vl);

node.clearResponseBuffer ();

//Control

Serial.println (" ");

node.readHoldingRegisters (7, 1);

Serial.print ("[7] Control music ");

control = node.getResponseBuffer(0);

Serial.println (control);

node.clearResponseBuffer ();

//floder

Serial.println (" ");

node.readHoldingRegisters (8, 1);

Serial.print ("[8] floder music ");

Floder = node.getResponseBuffer (0);

Serial.println (Floder);

node.clearResponseBuffer ();

//Song

Serial.println (" ");

node.readHoldingRegisters (9, 1);

Serial.print ("[9] Song ");

song = node.getResponseBuffer(0);

Serial.println (song);

node.clearResponseBuffer ();

delay (100);

}