(1条消息)ModBus-RTU详解 - leolian - CSDN博客

Modbus 一个工业上常用的通讯协议、一种通讯约定。Modbus协议包括RTU、ASCII、TCP。其中MODBUS-RTU最常用，比较简单，在单片机上很容易实现。虽然RTU比较简单，但是看协议资料、手册说得太专业了，起初很多内容都很难理解。

所谓的协议是什么？就是互相之间的约定嘛，如果不让别人知道那就是暗号。现在就来定义一个新的最简单协议。例如，

协议： “A” --“LED灭”
“B” --“报警”
“C” --“LED亮”
单片机接收到“A”控制一个LED灭，单片机接收到“B”控制报警，单片机接收到“A”控制一个LED亮。那么当收到对应的信息就执行相应的动作，这就是协议，很简单吧。

先来简单分析一条MODBUS-RTU报文，例如：01 06 00 01 00 17 98 04

01	06	00 01	00 17	98 04
从机地址	功能号	数据地址	数据	CRC校验

这一串数据的意思是：把数据 0x0017(十进制23) 写入 1号从机地址 0x0001数据地址。

先弄明白下面的东西。

1、报文
一个报文就是一帧数据，一个数据帧就一个报文： 指的是一串完整的指令数据，就像上面的一串数据。

2、CRC校验
意义：例如上面的 98 04 是它前面的数据（01 06 00 01 00 17）通过一算法（见附录2，很简单的）计算出来的结果，其实就像是计算累加和那样。（累加和：就是010600010017加起来的值，然后它的算法就是加法）。
作用：在数据传输过程中可能数据会发生错误，CRC检验检测接收的数据是否正确。比如主机发出01 06 00 01 00 17 98 04，那么从机接收到后要根据01 06 00 01 00 17 再计算CRC校验值，从机判断自己计算出来的CRC校验是否与接收的CRC校验（98 04主机计算的）相等，如果不相等那么说明数据传输有错误这些数据不能要。

3、功能号
意义：modbus 定义。见附录1。
作用：指示具体的操作。

MODBUS-RTU
一、一个报文分析
先声明下我们的目的，我们是要两个设备通讯，用的是MODBUS协议。上面简单介绍了：“报文”“CRC校验”“功能号”。

在单片机中拿出一部分内存（RAM）进行两个设备通讯，例如：

```
INT8U   OX[20];          // 定义8位的数组变量。   输出线圈     功能码：0x01,0x05,0x0f   地址：0x
INT8U   IX[20];          // 定义8位的数组变量。   输入线圈     功能码：0x02             地址：1x
INT16U  HoldDataReg[30]; // 定义16位的数组变量。  保持寄存器   功能码：0x03,0x06,0x10   地址：4x
INT16U  InDataReg[30];   // 定义16位的数组变量。  输入寄存器   功能码：0x04             地址：3x
```

数组后面的注释，说明
OX[20] 代表是输出线圈，用功能码 0x01，0x05，0x0F 访问， 开头地址是 0（这个后续说明）

IX[20] 代表是输入线圈，用功能码 0x02 访问， 开头地址是 1（这个后续说明）
另外两个一样的道理。
注意：所谓的“线圈”“寄存器”就是“位变量”“16位变量”，不要被迷惑。之所以称“线圈”我觉得应该是对于应用的设备，MODBUS协议是专门针对485总线设备（例PLC）开发的。

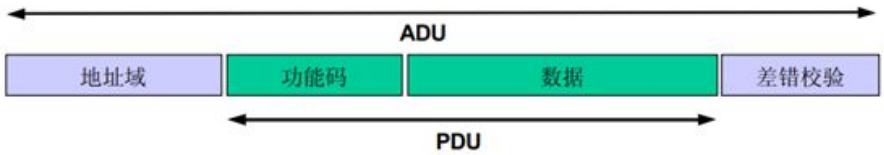
1、主机对从机写数据操作
如果单片机接收到一个报文那么就对报文进行解析执行相应的处理，如上面报文：
01 06 00 01 00 17 98 04
从机地址 功能号 数据地址 数据 CRC校验

假如本机地址是 1 ，那么单片机接收到这串数据根据数据计算CRC校验判断数据是否正确，如果判断数据无误，则结果是：
HoldDataReg[1] = 0x0017;
MODBUS主机就完成了一次对从机数据的写操作，实现了通讯。

2、主机对从机读数据操作
主机进行读HoldDataReg[1] 操作，则报文是：
01 03 00 01 00 01 D5 CA
从机地址 功能号 数据地址 读取数据个数 CRC校验
那么单片机接收到这串数据根据数据计算CRC校验判断数据是否正确，如果判断数据无误，则结果是：返回信息给主机，返回的信息也是有格式的：
返回内容：
01 03 02 0017 F8 4A
从机地址 功能号 数据字节个数 两个字节数据 CRC校验
MODBUS主机就完成了一次对从机数据的读操作，实现了通讯。

二、MODBUS报文模型

以上了解到了MODBUS的一帧报文是如何通讯的，其实每个报文的格式都基本一样的。



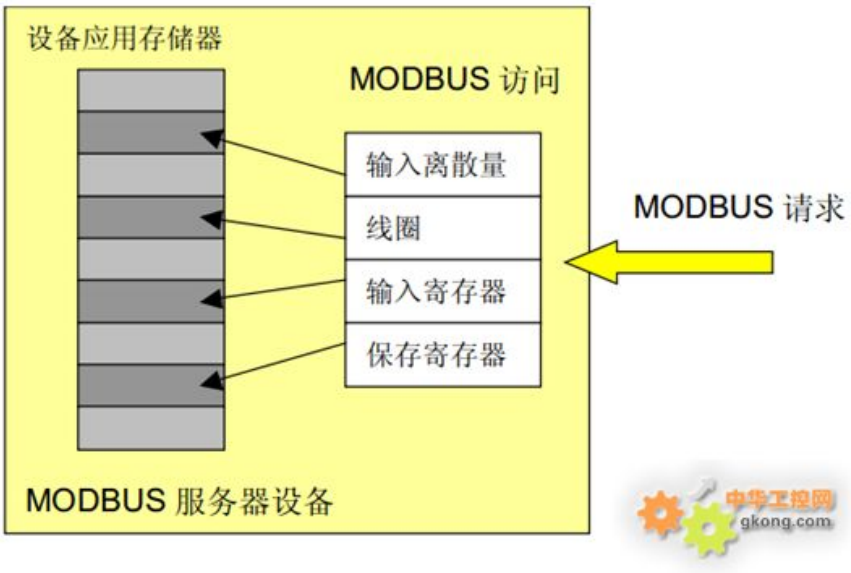
通用 MODBUS 帧

这里两个缩略词以前不知道，但是现在要明白指的是什么，“ADU”“PDU”
ADU： 应用数据单元
PDU： 协议数据单元

三、MODBUS数据模型

MODBUS 以一系列具有不同特征表格上的数据模型为基础。四个基本表格为：

基本表格	对象类型	访问类型	内容
离散量输入	单个比特	只读	I/O 系统提供这种类型数据
线圈	单个比特	读写	通过应用程序改变这种类型数据
输入寄存器	16-比特字	只读	I/O 系统提供这种类型数据
保持寄存器	16-比特字	读写	通过应用程序改变这种类型数据



四、MODBUS事务处理

下列状态图描述了在服务器侧MODBUS事务处理的一般处理过程。

五、MODBUS请求与响应

看MODBUS协议手册，中文第 10 页开始，英文第 24 页开始。手册非常详细举例说明了MODBUS协议各个功能号的请求与响应。

modbus协议在单片机上实现过程

MODBUS 任务处理函数

```
/*******************************************************************************
* Function Name  : ModbusHandle
* Input          :
* Return         :
* Description    : ModBus 处理函数
*******************************************************************************/
void ModbusHandle(void)
{
  if(RcvBuf[0]!=LOCALADDR){  // 比较地址，是否是本机地址
     UartClearBuffer();      // 不是本机地址，清空接收缓冲区
     return;
  }
  switch(RcvBuf[1]){
     case  0x01:{     ReadCoilState('O');    // 读取输出线圈状态
     }break;
     case  0x02:{     ReadCoilState('I');    // 读取输入线圈状态
     }break;
     case  0x03:{     ReadRegiState('H');    // 读取保持寄存器
     }break;
     case  0x04:{     ReadRegiState('I');    // 读取输入寄存器
     }break;
     case  0x05:{     SetSingleCoil();       // 设置单个线圈
     }break;
     case  0x06:{     SetSingleRegVal('H');  // 设置单个寄存器
     }break;
     case  0x0f:{     SetMultCoil();         // 设置多个线圈
     }break;
     case  0x10:{     SetMultRegVal('H');    // 预置多个寄存器
     }break;
     default:{
       RcvBuf[1] |= 0x80;
       RcvBuf[2] =  1;
       UartSend(RcvBuf, 3);                 // 返回错误码(不支持的功能号)
     }break;
  }
  UartClearBuffer();                        // 报文处理完成，清空接收缓冲区
}
```

函数中，RcvBuf 为串口接收缓冲区，如果接收了一个报文则，RcvBuf[0] 为从机地址，RcvBuf[0] 为MODBUS功能号。根据功能号做出响应，而具体的操作根据功能号在各自的函数中执行，相当于解析接收到的数据。

附录1：MODBUS-RTU功能码

最常用功能码：

下面“线圈”“寄存器”其实分别直的就是“位变量”“16位变量”

01 (0x01)	读线圈
02 (0x02)	读离散量输入
03 (0x03)	读保持寄存器
04(0x04)	读输入寄存器
05 (0x05)	写单个线圈
06 (0x06)	写单个寄存器
15 (0x0F)	写多个线圈
16 (0x10)	写多个寄存器

01 Read Coil Status
02 Read Input Status
03 Read Holding Registers
04 Read Input Registers
05 Force Single Coil
06 Preset Single Register
07 Read Exception Status
11 (0B Hex) Fetch Comm Event Ctr
12 (0C Hex) Fetch Comm Event Log
15 (0F Hex) Force Multiple Coils
16 (10 Hex) Preset Multiple Regs
17 (11 Hex) Report Slave ID
20 (14Hex) Read General Reference
21 (15Hex) Write General Reference
22 (16Hex) Mask Write 4X Register
23 (17Hex) Read/Write 4X Registers
24 (18Hex) Read FIFO Queue

附录2：CRC Generation

CRC Generation Function

```
unsigned short CRC16(puchMsg, usDataLen)

unsigned char *puchMsg ;                    /* message to calculate CRC upon       */
unsigned short usDataLen ;                  /* quantity of bytes in message        */

{
    unsigned char uchCRCHi = 0xFF ;         /* high byte of CRC initialized        */
    unsigned char uchCRCLo = 0xFF ;         /* low byte of CRC initialized         */
    unsigned uIndex ;                       /* will index into CRC lookup table    */

    while (usDataLen--)                     /* pass through message buffer         */
        {
        uIndex = uchCRCHi ^ *puchMsgg++ ;   /* calculate the CRC                   */
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
        uchCRCLo = auchCRCLo[uIndex] ;
        }

    return (uchCRCHi << 8 | uchCRCLo) ;
}
```