**原创  CRC循环冗余码原理** [**收藏**](JavaScript:d=document;t=d.selection?(d.selection.type!='None'?d.selection.createRange().text:''):(d.getSelection?d.getSelection():'');void(saveit=window.open('http://wz.csdn.net/storeit.aspx?t='+escape(d.title)+'&u='+escape(d.location.href)+'&c='+escape(t),'saveit','scrollbars=no,width=590,height=300,left=75,top=20,status=no,resizable=yes'));saveit.focus();)

一。

　　在远距离数据通信中，为确保高效而无差错地传送数据，必须对数据进行校验即差错控制。循环冗余校验CRC(Cyclic   Redundancy   Check)是对一个传送数据块进行校验，是一种高效的差错控制方法。     
      
  1　循环冗余校验码原理     
      
  　　CRC校验采用多项式编码方法，如一个8位二进制数(B7B6B5B4B3B2B1B0)可以用7阶二进制码多项式B7X7+B6X6+B5X5+B4X4+B3X3+B2X2+B1X1+B0X0表示。     
  　　例如11000001可表示为     
  　　1X7+1X6+0X5+0X4+0X3+0X2+0X1+0X0     
  　　一般说，n位二进制数可用(n-1)阶多项式表示。它把要发送的数据位串看成是系数只能为“1”或“0”的多项式。一个n位的数据块可以看成是从Xn-1到X0的n项多项式的系数序列，位于数据块左边的最高位是Xn-1项的系数，次高位是Xn-2项的系数，依此类推，位于数据块右边的最低位是X0项的系数，这个多项式的阶数为n-1。     
  　　多项式乘除法运算过程与普通代数多项式的乘除法相同。多项式的加减法运算以2为模，加减时不进、错位，如同逻辑异或运算。     
  　　采用CRC校验时，发送方和接收方事先约定一个生成多项式G(X)，并且G(X)的最高项和最低项的系数必须为1。设m位数据块的多项式为M(X)，生成多项式G(X)的阶数必需比M(X)的阶数低。CRC校验码的检错原理是：发送方先为数据块生成CRC校验码，使这个CRC校验码的多项式能被G(X)除尽，实际发送此CRC校验码；接收方用收到的CRC校验码除以G(X)，如果能除尽，表明传输正确，否则，表示有传输错误，请求重发。     
  　　生成数据块的CRC校验码的方法是：     
  　　(1)   设G(X)为r阶，在数据块末尾添加r个0，使数据块为m+r位，则相应的多项式为XrM(X)；     
  　　(2)   以2为模，用对应于G(X)的位串去除对应于XrM(X)的位串，求得余数位串；     
  　　(3)   以2为模，从对应于XrM(X)的位串中减去余数位串，结果就是为数据块生成的带足够校验信息的CRC校验码位串。     
  　　例如，设要发送的数据为1101011011，G(X)=X4+X+1，则首先在发送数据块的末尾加4个0，得到11010110110000，然后用G(X)的位串10011去除，再用11010110110000减去余数位串1110，得到的即为CRC位串11010110111110，将对应多项式称为T(X)，显然，T(X)能被G(X)除尽。这样，一旦接收到的CRC位串不能被同样的G(X)的位串除尽，那么一定有传输错误。     
  　　当使用CRC校验码进行差错控制时，除了为G(X)的整数倍的差错多项式不能被检测外，其它差错均能被查出。CRC校验码的差错控制效果取决于G(X)的阶数，阶数越高，效果越好。目前，常用的有两种生成多项式G(X)的方法，分别是：     
  　　CRC-16　　X16+X15+X2+1     
  　　CCITT　　X16+X12+X5+1     
  　　CRC校验码实际上是一种线性码，将任意CRC校验码循环移位后仍然是一个CRC校验码。由于它有良好的结构，检错能力强，易于实现硬件编、译码，因此在数据通信系统中得到广泛的应用。

二。

CRC的全称为Cyclic Redundancy Check，中文名称为循环冗余校验。它是一类重要的线性分组码，编码和解码方法简单，检错和纠错能力强，在通信领域广泛地用于实现差错控制。  
实际上，除数据通信外，CRC在其它很多领域也是大有用武之地的。例如我们读软盘上的文件，以及解压一个ZIP文件时，偶尔会碰到“Bad CRC”错误，由此它在数据存储方面的应  
用可略见一斑。

差错控制理论是在代数理论基础上建立起来的。这里我们着眼于介绍CRC的算法与实现，对原理只能捎带说明一下。若需要进一步了解线性码、分组码、循环码、纠错编码等方面的原  
理，可以阅读有关资料。

利用CRC进行检错的过程可简单描述为：在发送端根据要传送的k位二进制码序列，以一定的规则产生一个校验用的r位监督码(CRC码)，附在原始信息后边，构成一个新的二进制码序  
列数共k+r位，然后发送出去。在接收端，根据信息码和CRC码之间所遵循的规则进行检验，以确定传送中是否出错。这个规则，在差错控制理论中称为“生成多项式”。

**1 代数学的一般性算法**

在代数编码理论中，将一个码组表示为一个多项式，码组中各码元当作多项式的系数。例如 1100101 表示为  
1·x6+1·x5+0·x4+0·x3+1·x2+0·x+1，即 x6+x5+x2+1。

设编码前的原始信息多项式为P(x)，P(x)的最高幂次加1等于k；生成多项式为G(x)，G(x)的最高幂次等于r；CRC多项式为R(x)；编码后的带CRC的信息多项式为T(x)。

发送方编码方法：将P(x)乘以xr(即对应的二进制码序列左移r位)，再除以G(x)，所得余式即为R(x)。用公式表示为  
T(x)=xrP(x)+R(x)

接收方解码方法：将T(x)除以G(x)，如果余数为0，则说明传输中无错误发生，否则说明传输有误。

举例来说，设信息码为1100，生成多项式为1011，即P(x)=x3+x2，G(x)=x3+x+1，计算CRC的过程为

xrP(x) x3(x3+x2) x6+x5 x  
 -------- = ---------- = -------- = (x3+x2+x) + --------  
 G(x) x3+x+1 x3+x+1 x3+x+1

即 R(x)=x。注意到G(x)最高幂次r=3，得出CRC为010。

如果用竖式除法，计算过程为

1110  
 -------   
 1011 /1100000 (1100左移3位)  
 1011  
 ----  
 1110  
 1011  
 -----  
 1010  
 1011  
 -----  
 0010  
 0000  
 ----  
 010

因此，T(x)=(x6+x5)+(x)=x6+x5+x, 即 1100000+010=1100010

如果传输无误，

T(x) x6+x5+x  
 ------ = --------- = x3+x2+x,  
 G(x) x3+x+1

无余式。回头看一下上面的竖式除法，如果被除数是1100010，显然在商第三个1时，就能除尽。

三。

CRC校验

    crc算法已经有成熟和比较经典的现成代码可供我们利用。CRC计算可以靠专用的硬件来实现,但是对于低成本的微控制器系统,在没有硬件支持下实现CRC检验,关键的问题就是如何通过软件来完成CRC计算,也就是CRC算法的问题。CRC校验的基本思想是利用线性编码理论,在发送端根据要传送的k位二进制码序列,以一定的规则产生一个校验用的监督码（既CRC码）r位,并附在信息后边,构成一个新的二进制码序列数共(k+r)位,最后发送出去。在接收端,则根据信息码和CRC码之间所遵循的规则进行检验,以确定传送中是否出错。

1.生成多项式。  
16位的CRC码产生的规则是先将要发送的二进制序列数左移16位（既乘以   
）后,再除以一个多项式,最后所得到的余数既是CRC码。任意一个由二进制位串组成的代码都可以和一个系数仅为‘0’和‘1’取值的多项式一一对应。例如：代码1010111对应的多项式为x6+x4+x2+x+1，而多项式为x5+x3+x2+x+1对应的代码101111。

标准CRC生成多项式如下表：

   名称        生成多项式              简记式\*   标准引用   
   CRC-4       x4+x+1                  3         ITU G.704   
   CRC-8       x8+x5+x4+1              0x31                      
   CRC-8       x8+x2+x1+1              0x07                      
   CRC-8       x8+x6+x4+x3+x2+x1       0x5E   
   CRC-12      x12+x11+x3+x+1          80F  
   CRC-16      x16+x15+x2+1            8005      IBM SDLC  
   CRC16-CCITT x16+x12+x5+1            1021      ISO HDLC, ITU X.25, V.34/V.41/V.42, PPP-FCS   
   CRC-32      x32+x26+x23+...+x2+x+1 04C11DB7 ZIP, RAR, IEEE 802 LAN/FDDI, IEEE 1394, PPP-FCS   
   CRC-32c     x32+x28+x27+...+x8+x6+1 1EDC6F41 SCTP //叶子：这里不知道问什么省略了，有些迷惑哦。要是生成多项式要是都省了，那还怎么校验？我猜想可能是中间的全为一吧。  
                                
   生成多项式的最高位固定的1，故在简记式中忽略最高位1了，如0x1021实际是0x11021。  
I、基本算法（人工笔算）：  
   以CRC16-CCITT为例进行说明，CRC校验码为16位，生成多项式17位。假如数据流为4字节：BYTE[3]、BYTE[2]、BYTE[1]、BYTE[0]；  
数据流左移16位，相当于扩大256×256倍，再除以生成多项式0x11021，做不借位的除法运算（相当于按位异或），所得的余数就是CRC校验码。  
发送时的数据流为6字节：BYTE[3]、BYTE[2]、BYTE[1]、BYTE[0]、CRC[1]、CRC[0]；

II、计算机算法1（比特型算法）：  
1)将扩大后的数据流（6字节）高16位（BYTE[3]、BYTE[2]）放入一个长度为16的寄存器；  
2)如果寄存器的首位为1，将寄存器左移1位(寄存器的最低位从下一个字节获得)，再与生成多项式的简记式异或；  
    否则仅将寄存器左移1位(寄存器的最低位从下一个字节获得)；  
3)重复第2步，直到数据流（6字节）全部移入寄存器；  
4)寄存器中的值则为CRC校验码CRC[1]、CRC[0]。

III、计算机算法2（字节型算法）：256^n表示256的n次方  
    把按字节排列的数据流表示成数学多项式，设数据流为BYTE[n]BYTE[n－1]BYTE[n－2]、、、BYTE[1]BYTE[0]，表示成数学表达式为BYTE[n]×256^n+BYTE[n-1]×256^(n-1)

+...+BYTE[1]\*256+BYTE[0]，在这里+表示为异或运算。设生成多项式为G17（17bit），CRC码为CRC16。  
    则，CRC16＝(BYTE[n]×256^n+BYTE[n-1]×256^(n-1)+...+BYTE[1]×256+BYTE[0])×256^2/G17，即数据流左移16位，再除以生成多项式G17。  
    先变换BYTE[n-1]、BYTE[n-1]扩大后的形式，  
    CRC16＝BYTE[n]×256^n×256^2/G17+BYTE[n-1]×256^(n-1)×256^2/G17+...+BYTE[1]×256×256^2/G17+BYTE[0]×256^2/G17  
         ＝(Z[n]+Y[n]/G17)×256^n+BYTE[n-1]×256^(n-1)×256^2/G17+...+BYTE[1]×256×256^2/G17+BYTE[0]×256^2/G17  
         ＝Z[n]×256^n+{Y[n]×256/G17+BYTE[n-1]×256^2/G17}×256^(n-1)+...+BYTE[1]×256×256^2/G17+BYTE[0]×256^2/G17  
         ＝Z[n]×256^n+{(YH8[n]×256+YHL[n])×256/G17+BYTE[n-1]×256^2/G17}×256^(n-1)+...+BYTE[1]×256×256^2/G17+BYTE[0]×256^2/G17  
         ＝Z[n]×256^n+{YHL[n]×256/G17+(YH8[n]+BYTE[n-1])×256^2/G17}×256^(n-1)+...+BYTE[1]×256×256^2/G17+BYTE[0]×256^2/G17  
    这样就推导出，BYTE[n-1]字节的CRC校验码为{YHL[n]×256/G17+(YH8[n]+BYTE[n-1])×256^2/G17}，即上一字节CRC校验码Y[n]的高8位（YH8[n]）与本字节BYTE[n-1]异或，

该结果单独计算CRC校验码（即单字节的16位CRC校验码，对单字节可建立表格，预先生成对应的16位CRC校验码），所得的CRC校验码与上一字节CRC校验码Y[n]的低8位（YL8[n]）

乘以256（即左移8位）异或。然后依次逐个字节求出CRC，直到BYTE[0]。  
    字节型算法的一般描述为：本字节的CRC码，等于上一字节CRC码的低8位左移8位，与上一字节CRC右移8位同本字节异或后所得的CRC码异或。      
    字节型算法如下：  
    1)CRC寄存器组初始化为全"0"(0x0000)。（注意：CRC寄存器组初始化全为1时，最后CRC应取反。）  
    2)CRC寄存器组向左移8位,并保存到CRC寄存器组。  
    3)原CRC寄存器组高8位（右移8位）与数据字节进行异或运算，得出一个指向值表的索引。  
    4)索引所指的表值与CRC寄存器组做异或运算。  
    5)数据指针加1，如果数据没有全部处理完，则重复步骤2)。  
    6)得出CRC。

unsigned short GetCrc\_16(unsigned char \* pData, int nLength)  
//函数功能：计算数据流\* pData的16位CRC校验码，数据流长度为nLength  
{  
    unsigned short cRc\_16 = 0x0000;    // 初始化  
      
    while(nLength>0)  
    {  
        cRc\_16 = (cRc\_16 << 8) ^ cRctable\_16[((cRc\_16>>8) ^ \*pData) & 0xff]; //cRctable\_16表由函数mK\_cRctable生成  
        nLength--;  
        pData++;  
    }  
      
    return cRc\_16;      
}

void mK\_cRctable(unsigned short gEnpoly)  
//函数功能：生成0－255对应的16CRC校验码，其实就是计算机算法1（比特型算法）  
//gEnpoly为生成多项式  
//注意，低位先传送时，生成多项式应反转(低位与高位互换)。如CRC16-CCITT为0x1021，反转后为0x8408  
{   
unsigned short cRc\_16=0;  
unsigned short i,j,k;

for(i=0,k=0;i<256;i++,k++)  
{  
      cRc\_16 = i<<8;  
      for(j=8;j>0;j--)  
      {  
if(cRc\_16&0x8000)                 //反转时cRc\_16&0x0001  
             cRc\_16=(cRc\_16<<=1)^gEnpoly; //反转时cRc\_16=(cRc\_16>>=1)^gEnpoly  
         else  
             cRc\_16<<=1;                   //反转时cRc\_16>>=1  
      }  
      cRctable\_16[k] = cRc\_16;  
}  
}

2：CRC码集选择的原则

若设码字长度为N，信息字段为K位，校验字段为R位(N=K+R)，则对于CRC码集中的任一码字，存在且仅存在一个R次多项式g(x)，使得

V(x)=A(x)g(x)=xRm(x)+r(x);

其中:    m(x)为K次信息多项式， r(x)为R-1次校验多项式，

         g(x)称为生成多项式：

g(x)=g0+g1x+ g2x2+...+g(R-1)x(R-1)+gRxR

发送方通过指定的g(x)产生CRC码字，接收方则通过该g(x)来验证收到的CRC码字。

3、CRC校验码软件生成方法：

    借助于多项式除法，其余数为校验字段。

例如：信息字段代码为: 1011001；对应m(x)=x6+x4+x3+1

      假设生成多项式为：g(x)=x4+x3+1；则对应g(x)的代码为: 11001

      x4m(x)=x10+x8+x7+x4 对应的代码记为：10110010000；

采用多项式除法: 得余数为: 1010     (即校验字段为：1010）

发送方：发出的传输字段为: 1 0 1 1 0 0 1 1 0 10