CRC 算法原理及 C 语言实现

摘 要 本文从理论上推导出 CRC 算法实现原理，给出三种分别适应不同计算机或微控制器硬件环境的 C 语言程序。读者更能根据本算法原理，用不同的语言编写出独特风格更加实用的 CRC 计算程序。

关键词 CRC 算法 C 语言

1 引言

循环冗余码 CRC 检验技术广泛应用于测控及通信领域。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。

这里将提供三种算法，它们稍有不同，一种适用于程序空间十分苛刻但 CRC 计算速度要求不高的微控制器系统，另一种适用于程序空间较大且 CRC 计算速度要求较高的计算机或微控制器系统，最后一种是适用于程序空间不太大，且 CRC 计算速度又不可以太慢的微控制器系统。

2 CRC 简介

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。

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 2^{16})后，再除以一个多项式，最后所得到的余数既是 CRC 码，如式（2-1）式所示，其中 B(X)表示 n 位的二进制序列数，G(X)为多项式，Q(X)为整数，R(X)是余数（既 CRC 码）。

$$\frac{B(X)\cdot 2^{16}}{G(X)} = Q(X) + \frac{R(X)}{G(X)} \qquad (2\text{-}1)$$

求CRC码所采用模 2 加减运算法则，既是不带进位和借位的按位加减，这种加减运算实际上就是逻辑上的异或运算，加法和减法等价，乘法和除法运算与普通代数式的乘除法运算是一样，符合同样的规律。生成CRC码的多项式如下，其中CRC-16 和CRC-CCITT 产生 16 位的CRC码，而CRC-32 则产生的是 32 位的CRC码。本文不讨论 32 位的CRC算法，有兴趣的朋友可以根据本文的思路自己去推导计算方法。

CRC-16：（美国二进制同步系统中采用） $G(X) = X^{16} + X^{15} + X^2 + 1$

CRC-CCITT：（由欧洲 CCITT 推荐） $G(X) = X^{16} + X^{12} + X^5 + 1$

CRC-32： $$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + 1$$

接收方将接收到的二进制序列数（包括信息码和 CRC 码）除以多项式，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，关于其原理这里不再多述。用软件计算 CRC 码时，接收方可以将接收到的信息码求 CRC 码，比较结果和接收到的 CRC 码是否相同。

3 按位计算 CRC

对于一个二进制序列数可以表示为式(3-1)：

$$B(X)=B_n\cdot 2^n+B_{n-1}\cdot 2^{n-1}+\cdots+B_1\cdot 2+B_0 \qquad (3\text{-}1)$$

求此二进制序列数的 CRC 码时，先乘以 2^{16} 后（既左移 16 位），再除以多项式 G(X)，所得的余数既是所要求的 CRC 码。如式(3-2)所示：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=\frac{B_n\cdot 2^{16}}{G(X)}\cdot 2^n+\frac{B_{n-1}\cdot 2^{16}}{G(X)}\cdot 2^{n-1}+\cdots+\frac{B_1\cdot 2^{16}}{G(X)}\cdot 2+\frac{B_0\cdot 2^{16}}{G(X)} \qquad (3\text{-}2)$$

可以设：

$$\frac{B_n\cdot 2^{16}}{G(X)}=Q_n(X)+\frac{R_n(X)}{G(X)} \qquad (3\text{-}3)$$

其中 $Q_n(X)$ 为整数，$R_n(X)$ 为 16 位二进制余数。将式(3-3)代入式(3-2)得：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=\{Q_n(X)+\frac{R_n(X)}{G(X)}\}\cdot 2^n+\frac{B_{n-1}\cdot 2^{16}}{G(X)}\cdot 2^{n-1}+\cdots+\frac{B_1\cdot 2^{16}}{G(X)}\cdot 2+\frac{B_0\cdot 2^{16}}{G(X)}$$

$$=Q_n(X)\cdot 2^n+\{\frac{R_n(X)\cdot 2}{G(X)}+\frac{B_{n-1}\cdot 2^{16}}{G(X)}\}\cdot 2^{n-1}+\cdots+\frac{B_1\cdot 2^{16}}{G(X)}\cdot 2+\frac{B_0\cdot 2^{16}}{G(X)} \qquad (3\text{-}4)$$

再设：

$$\frac{R_n(X)\cdot 2}{G(X)}+\frac{B_{n-1}\cdot 2^{16}}{G(X)}=Q_{n-1}(X)+\frac{R_{n-1}(X)}{G(X)} \qquad (3\text{-}5)$$

其中 $Q_{n-1}(X)$ 为整数，$R_{n-1}(X)$ 为 16 位二进制余数 ，将式(3-5)代入式(3-4)，如上类推，最后得到：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=Q_n(X)\cdot 2^n+Q_{n-1}(X)\cdot 2^{n-1}+Q_{n-2}(X)\cdot 2^{n-2}+\cdots+Q_0(X)+\frac{R_0(X)}{G(X)} \qquad (3\text{-}6)$$

根据 CRC 的定义，很显然，十六位二进制数 $R_0(X)$ 既是我们要求的 CRC 码。

式(3-5)是编程计算 CRC 的关键，它说明计算本位后的 CRC 码等于上一位 CRC 码乘以 2 后除以多项式，所得的余数再加上本位值除以多项式所得的余数。由此不难理解下面求 CRC 码的 C 语言程序。*ptr 指向发送缓冲区的首字节，len 是要发送的总字节数，0x1021 与多项式有关。

```
unsigned int cal_crc(unsigned char *ptr, unsigned char len) {
  unsigned char i;
  unsigned int crc=0;

  while(len--!=0) {
    for(i=0x80; i!=0; i/=2) {
      if((crc&0x8000)!=0) {crc*=2; crc^=0x1021;}     /* 余式 CRC 乘以 2 再求 CRC   */
        else crc*=2;
      if((*ptr&i)!=0) crc^=0x1021;                   /* 再加上本位的 CRC */
    }
    ptr++;
  }
  return(crc);
}
```

按位计算 CRC 虽然代码简单，所占用的内存比较少，但其最大的缺点就是一位一位地计算会占用很多的处理器处理时间，尤其在高速通讯的场合，这个缺点更是不可容忍。因此下面再介绍一种按字节查表快速计算 CRC 的方法。

4 按字节计算 CRC

不难理解，对于一个二进制序列数可以按字节表示为式(4-1)，其中 $B_n(X)$ 为一个字节(共 8 位)。

$$B(X)=B_n(X)\cdot 2^{8n}+B_{n-1}(X)\cdot 2^{8(n-1)}+\cdots+B_1(X)\cdot 2^8+B_0(X) \tag{4-1}$$

求此二进制序列数的 CRC 码时，先乘以 2^{16} 后（既左移 16 位），再除以多项式 G(X)，所得的余数既是所要求的 CRC 码。如式(4-2)所示：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=\frac{B_n(X)\cdot 2^{16}}{G(X)}\cdot 2^{8n}+\frac{B_{n-1}(X)\cdot 2^{16}}{G(X)}\cdot 2^{8(n-1)}+\cdots+\frac{B_0(X)\cdot 2^{16}}{G(X)} \tag{4-2}$$

可以设：

$$\frac{B_n(X)\cdot 2^{16}}{G(X)}=Q_n(X)+\frac{R_n(X)}{G(X)} \tag{4-3}$$

其中 $Q_n(X)$ 为整数，$R_n(X)$ 为 16 位二进制余数。将式(4-3)代入式(4-2)得：

$$\begin{aligned}\frac{B(X)\cdot 2^{16}}{G(X)}&=[Q_n(X)+\frac{R_n(X)}{G(X)}]\cdot 2^{8n}+\frac{B_{n-1}(X)\cdot 2^{16}}{G(X)}\cdot 2^{8(n-1)}+\cdots+\frac{B_0(X)\cdot 2^{16}}{G(X)}\\&=Q_n(X)\cdot 2^{8n}+\{\frac{R_n(X)\cdot 2^8}{G(X)}+\frac{B_{n-1}(X)\cdot 2^{16}}{G(X)}\}\cdot 2^{8(n-1)}+\cdots+\frac{B_0\cdot 2^{16}}{G(X)}\end{aligned} \tag{4-4}$$

因为：

$$\begin{aligned}R_n(X)\cdot 2^8&=[R_{nH8}(X)\cdot 2^8+R_{nL8}(X)]\cdot 2^8\\&=R_{nH8}(X)\cdot 2^{16}+R_{nL8}(X)\cdot 2^8\end{aligned} \tag{4-5}$$

其中 $R_{nH8}(X)$ 是 $R_n(X)$ 的高八位，$R_{nL8}(X)$ 是 $R_n(X)$ 的低八位。将式（4-5）代入式（4-4），经整理后得：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=Q_n(X)\cdot 2^{8n}+\{\frac{R_{nL8}(X)\cdot 2^8}{G(X)}+\frac{[R_{nH8}(X)+B_{n-1}(X)]\cdot 2^{16}}{G(X)}\}\cdot 2^{8(n-1)}+\cdots+\frac{B_0\cdot 2^{16}}{G(X)} \tag{4-6}$$

再设：

$$\frac{R_{nL8}(X)\cdot 2^8}{G(X)}+\frac{[B_{nH8}(X)+B_{n-1}(X)]\cdot 2^{16}}{G(X)}=Q_{n-1}(X)+\frac{R_{n-1}(X)}{G(X)} \tag{4-7}$$

其中 $Q_{n-1}(X)$ 为整数，$R_{n-1}(X)$ 为 16 位二进制余数。将式(4-7)代入式(4-6)，如上类推，最后得：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=Q_n(X)\cdot 2^{8n}+Q_{n-1}(X)\cdot 2^{8(n-1)}+\cdots+Q_0(X)+\frac{R_0(X)}{G(X)} \tag{4-8}$$

很显然，十六位二进制数 $R_0(X)$ 既是我们要求的 CRC 码。

式(4-7)是编写按字节计算 CRC 程序的关键，它说明计算本字节后的 CRC 码等于上一字节余式 CRC 码的低 8 位左移 8 位后，再加上上一字节 CRC 右移 8 位（也既取高 8 位）和本字节之和后所求得的 CRC 码，如果我们把 8 位二进制序列数的 CRC 全部计算出来，放如一个表里，采用查表法，可以大大提高计算速度。由此不难理解下面按字节求 CRC 码的 C 语言程序。*ptr 指向发送缓冲区的首字节，len 是要发送的总字节数，CRC 余式表是按 0x11021 多项式求出的。

```
unsigned int cal_crc(unsigned char *ptr,    unsigned char len) {
  unsigned int crc;
  unsigned char da;
  unsigned int crc_ta[256]={                    /* CRC 余式表  */
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x 1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
  };

  crc=0;
  while(len--!=0) {
    da=(uchar) (crc/256);       /*  以 8 位二进制数的形式暂存 CRC 的高 8 位  */
    crc<<=8;                    /*  左移 8 位，相当于 CRC 的低 8 位乘以 2^8   */
```

```
        crc^=crc_ta[da^*ptr];     /* 高 8 位和当前字节相加后再查表求 CRC ,再加上以前的 CRC */
        ptr++;
    }
    return(crc);
}
```

很显然，按字节求 CRC 时，由于采用了查表法，大大提高了计算速度。但对于广泛运用的 8 位微处理器，代码空间有限，对于要求 256 个 CRC 余式表（共 512 字节的内存）已经显得捉襟见肘了，但 CRC 的计算速度又不可以太慢，因此再介绍下面一种按半字节求 CRC 的算法。

5　按半字节计算 CRC

同样道理，对于一个二进制序列数可以按字节表示为式(5-1)，其中 $B_n(X)$ 为半个字节(共 4 位)。

$$B(X)=B_n(X)\cdot 2^{4n}+B_{n-1}(X)\cdot 2^{4(n-1)}+\cdots+B_1(X)\cdot 2^4+B_0(X) \qquad (5\text{-}1)$$

求此二进制序列数的 CRC 码时，先乘以 2^{16} 后（既左移 16 位），再除以多项式 G(X)，所得的余数既是所要求的 CRC 码。如式(4-2)所示：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=\frac{B_n(X)\cdot 2^{16}}{G(X)}\cdot 2^{4n}+\frac{B_{n-1}(X)\cdot 2^{16}}{G(X)}\cdot 2^{4(n-1)}+\cdots+\frac{B_0(X)\cdot 2^{16}}{G(X)} \qquad (5\text{-}2)$$

可以设：
$$\frac{B_n(X)\cdot 2^{16}}{G(X)}=Q_n(X)+\frac{R_n(X)}{G(X)} \qquad (5\text{-}3)$$

其中 $Q_n(X)$ 为整数，$R_n(X)$ 为 16 位二进制余数。将式(5-3)代入式(5-2)得：

$$\begin{aligned}\frac{B(X)\cdot 2^{16}}{G(X)}&=[Q_n(X)+\frac{R_n(X)}{G(X)}]\cdot 2^{4n}+\frac{B_{n-1}(X)\cdot 2^{16}}{G(X)}\cdot 2^{4(n-1)}+\cdots+\frac{B_0(X)\cdot 2^{16}}{G(X)}\\&=Q_n(X)\cdot 2^{4n}+\{\frac{R_n(X)\cdot 2^4}{G(X)}+\frac{B_{n-1}(X)\cdot 2^{16}}{G(X)}\}\cdot 2^{4(n-1)}+\cdots+\frac{B_0\cdot 2^{16}}{G(X)}\end{aligned} \qquad (5\text{-}4)$$

因为：
$$\begin{aligned}R_n(X)\cdot 2^4&=[R_{nH4}(X)\cdot 2^{12}+R_{nL12}(X)]\cdot 2^4\\&=R_{nH4}(X)\cdot 2^{16}+R_{nL12}(X)\cdot 2^4\end{aligned} \qquad (5\text{-}5)$$

其中 $R_{nH4}(X)$ 是 $R_n(X)$ 的高 4 位，$R_{nL12}(X)$ 是 $R_n(X)$ 的低 12 位。将式（5-5）代入式（5-4），经整理后得：

$$\frac{B(X)\cdot 2^{16}}{G(X)}=Q_n(X)\cdot 2^{4n}+\{\frac{R_{nL12}(X)\cdot 2^4}{G(X)}+\frac{[R_{nH4}(X)+B_{n-1}(X)]\cdot 2^{16}}{G(X)}\}\cdot 2^{4(n-1)}+\cdots+\frac{B_0\cdot 2^{16}}{G(X)} \qquad (5\text{-}6)$$

再设：
$$\frac{R_{nL12}(X)\cdot 2^4}{G(X)}+\frac{[B_{nH4}(X)+B_{n-1}(X)]\cdot 2^{16}}{G(X)}=Q_{n-1}(X)+\frac{R_{n-1}(X)}{G(X)} \qquad (5\text{-}7)$$

其中 $Q_{n-1}(X)$ 为整数，$R_{n-1}(X)$ 为 16 位二进制余数。将式(5-7)代入式(5-6)，如上类推，最后得：

$$\frac{B(X)\cdot 2^{16}}{G(X)} = Q_n(X)\cdot 2^{4n} + Q_{n-1}(X)\cdot 2^{4(n-1)} + \cdots + Q_0(X) + \frac{R_0(X)}{G(X)} \qquad (5\text{-}8)$$

很显然，十六位二进制数 $R_0(X)$ 既是我们要求的 CRC 码。

式(5-7)是编写按字节计算 CRC 程序的关键，它说明计算本字节后的 CRC 码等于上一字节 CRC 码的低 12 位左移 4 位后，再加上上一字节余式 CRC 右移 4 位（也既取高 4 位）和本字节之和后所求得的 CRC 码，如果我们把 4 位二进制序列数的 CRC 全部计算出来，放在一个表里，采用查表法，每个字节算两次（半字节算一次），可以在速度和内存空间取得均衡。由此不难理解下面按半字节求 CRC 码的 C 语言程序。*ptr 指向发送缓冲区的首字节，len 是要发送的总字节数，CRC 余式表是按 0x11021 多项式求出的。

```
unsigned    cal_crc(unsigned char *ptr,    unsigned char len) {
   unsigned int crc;
   unsigned char da;
   unsigned int crc_ta[16]={                    /* CRC 余式表  */
      0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,
      0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,
   }

   crc=0;
   while(len--!=0) {
      da=((uchar)(crc/256))/16;          /*  暂存 CRC 的高四位  */
      crc<<=4;                           /* CRC 右移 4 位，相当于取 CRC 的低 12 位）*/
      crc^=crc_ta[da^(*ptr/16)];         /* CRC 的高 4 位和本字节的前半字节相加后查表计算 CRC，
                                            然后加上上一次 CRC 的余数  */
      da=((uchar)(crc/256))/16;          /*  暂存 CRC 的高 4 位  */
      crc<<=4;                           /* CRC 右移 4 位， 相当于 CRC 的低 12 位）  */
      crc^=crc_ta[da^(*ptr&0x0f)];       /*  CRC  的高  4  位和本字节的后半字节相加后查表计算
CRC，
                                            然后再加上上一次 CRC 的余数  */
      ptr++;
   }
   return(crc);
}
```

5　结束语

以上介绍的三种求 CRC 的程序，按位求法速度较慢，但占用最小的内存空间；按字节查表求 CRC 的方法速度较快，但占用较大的内存；按半字节查表求 CRC 的方法是前两者的均衡，即不会占用太多的内存，同时速度又不至于太慢，比较适合 8 位小内存的单片机的应用场合。以上所给的 C 程序可以根据各微处理器编译器的特点作相应的改变，比如把 CRC 余式表放到程序存储区内等。