文章编号:1674-7070(2012)03-0258-08

嵌入式系统 CRC 循环冗余校验算法设计研究

彭伟[1]

摘要

介绍了 CRC 循环冗余校验基本原理及生成多项式表示,分别研究了嵌入式系统 CRC-8-Dallas/Maxim 与 CRC-16-IBM 生成多项式及其硬件描述. 以 DS18B20 器件的 ROM ID/Scratchpad 数据校验及 Modbus 总线网络数据帧校验为例,通过对生成多项式及硬件描述的分析研究得出了基本比特型校验算法设计,在数学推导的基础上得出了其改进的比特型校验算法及单字节、半字节查表校验算法. 为获得更高的校验速度,提出了一种基于块及多表的校验算法,比较了几种校验算法的 ROM 空间占用与校验处理速度. 所设计的 CRC 校验程序为嵌入式系统数据的可靠传输提供了重要保证.

关键词

嵌入式; 循环冗余校验; 差错控制; 校验; 生成多项式; 算法

中图分类号 TP368.1/TN919.3

文献标志码 A

收稿日期 2011-11-02

资助项目 湖北省教育科学"十一五"规划项目(2009B-349)

作者简介

彭伟,男,工学硕士,副教授,研究方向为嵌入式系统设计、网络技术以及算法设计. pw95aaa@foxmail.com

1 武汉城市职业学院 电子信息工程学院,武汉,430064

0 引言

嵌入式系统微控制器与外部数据采集器件之间、嵌入式系统内多个微控制器之间以及微控制器与上位主机之间通过各种接口传输数据时,被噪声影响的问题不可避免. 为提高数据传输的可靠性,需要对传输过程进行差错控制[1-2]. 循环冗余校验(Cyclic Redundancy Check,CRC)由于编、解码方法简单,检错与纠错能力强而被广泛应用. 本文将以 Dallas/Maxim 的传感器 DS18B20 及 Modbus 总线网络数据帧 CRC 校验为例,分析研究典型的 CRC-8、CRC-16 校验生成多项式及硬件描述,设计对应的基本比特型校验算法,通过数学推导,进一步得出改进的比特型校验算法、单字节及半字节查表校验算法. 为获得更高的校验处理速度,提出一种基于块及多表的校验算法,并比较各种算法的码表所占用 ROM 空间及校验处理速度,所设计的 CRC 校验算法将为嵌入式系统数据传输提供重要保证.

1 CRC 基本原理及生成多项式表示

1.1 基本原理

CRC 循环冗余校验码是一种线性分组码[3],在不增加过多冗余位的情况下就能具备较强的检错与纠错能力. 假设待传送的 k 位信息码为 $M=(M_{k-1},M_{k-2},\cdots,M_1,M_0)$,将其看成多项式的系数,在其后面添加 r 个 0,有 $M(x)\cdot x^r=M_{k-1}\cdot x^{r+k-1}+M_{k-2}\cdot x^{r+k-2}+\cdots+M_1\cdot x^{r+1}+M_0\cdot 2^r$,将该多项式作为被除式,选择一个 r 次生成多项式 $G(x)$ 做除法,可得商式 $Q(x)$ 和最高为 $r-1$ 次的余式 $R(x)$,即:

$$\frac{M(x)\cdot x^r}{G(x)}=Q(x)+\frac{R(x)}{G(x)}, \tag{1}$$

$$M(x)\cdot x^r=Q(x)\ G(x)+R(x). \tag{2}$$

CRC 校验码计算使用模 2 除法,加、减通过按位异或运算实现,无进位和借位. 例如: $R(x)+R(x)$ 与 $R(x)-R(x)$ 结果均为全 0. 在式(2)两边同加 $R(x)$,有

$$M(x)\cdot x^r+R(x)=Q(x)\ G(x). \tag{3}$$

显然,式(3)左端代数式为 $G(x)$ 的整数倍,它正好是左移 r 位的原始信息多项式后面附加余式的结果. 该多项式的各项系数为 $(M_{k-1}\sim M_0,R_{r-1}\sim R_0)$,其中高 k 位为信息码,低 r 位为所附加的 CRC 校验码(监督码),发送端输出的正是该多项式的所有系数. 如果该序

列到达接收端后能被同一生成多项式 $G(x)$ 整除则表示数据传输正确,否则表示数据传输出现错误.

图 1 演示了二进制序列 CRC 校验码的计算过程. 被除数"1101011"对应的多项式为 $B(x) = x^6 + x^5 + x^3 + x + 1$,除数"10011"对应的生成多项式为 $G(x) = x^4 + x + 1$(即表 1 中的 CRC-4-ITU),所得模除余数为 0101.

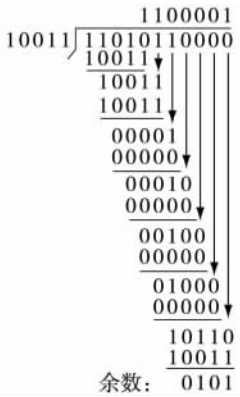

图 1 CRC 校验码计算示例

Fig. 1 Example for CRC checkcode calculation

1.2 $G(x)$ 的十六进制表示

表 1 给出了几种常见的 CRC 校验码生成多项式,其任意一个 n 阶生成多项式的最高次项 x^n 的系数恒为 1,它"界定"出 $n-1$ 次余式 $R(x)$ 的最大项数(即 CRC 校验码位数)为 n,除此以外它在"去高位"以后的 $n-1$ 次多项式的 2 种十六进制典型表示中不再有任何意义:1) 去高位正序表示,其最高位为 x^{n-1} 的系数,最低位为 x^0 的系数;2) 去高位逆序表示,其最低位为 x^{n-1} 的系数,最高位为 x^0 的系数.

表 1 几种常见的 CRC 校验生成多项式

Table 1 Several common CRC generating polynomial

CRC 名称	$G(x)$ 及其去高位正序与逆序表示
CRC-4-ITU	$x^4 + x + 1$(ITU-T), 0x3/0xC
CRC-8-Dallas/Maxim	$x^8 + x^5 + x^4 + 1$(1-Wire bus), 0x31/0x8C
CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$(Modbus), 0x8005/0xA001
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$, 0x04C11DB7/0xEDB88320

考察图 1 所示模除示例. 对于所选择的表 1 中的生成多项式 CRC-4-ITU,$G(x) = x^4 + x + 1$,除数"10011"的去高位"正序"与"逆序"表示分别为 0x3(0011)和 0xC(1100). 模除过程中除数固定为 5 位,每次模减的被减数也为 5 位,记为 a. 例如示例中前 3 次模减时 a 分别为 11010、10011 和 00001. a 与 5 位的除数 10011 模减时,a 高位为 1、0 分别对应执行 $a \hat{} 10011$、$a \hat{} 00000$,注意前者并非是在 $a \geqslant 10011$ 时商置 1 才执行. 两者异或(即模减)所得余数高位均为 0 且被丢弃,余数实际有效位最多为 4 位,它由被减数 a 及除数的低 4 位异或得到,且 4 位余数的最高位正好是下一被减数 a 的最高位. 模除过程中影响最终结果的是生成多项式的"去高位表示"(即 0011 或 1100). CRC 校验硬件电路或软件算法所使用的生成多项式十六进制典型表示均去掉了最高次项,去高位表示不影响模除结果.

实际应用中常见的是"去高位逆序表示",具体使用何种十六进制表示取决于器件的校验硬件生成电路设计或系统规约. 限于篇幅,这里略去对 Koopman 提出的"去低位正序表示"或称"互反多项式去高位逆序表示"的讨论.

2 CRC-8-Dallas/Maxim 校验算法设计

2.1 器件数据编码格式及校验硬件描述

Dallas/Maxim 公司 1-Wire 总线器件 DS18B20 的 ROM ID 和 Scratchpad 校验均使用表 1 中的 CRC-8-Dallas/Maxim 生成多项式. 器件 64 位全球唯一 ROM ID 数据编码格式如表 2 所示,其中高字节为 CRC 校验码,生成多项式 $G(x) = x^8 + x^5 + x^4 + 1$.

表 2 ROM ID 编码格式

Table 2 ROM ID coding format bit

CRC	Serial Code	Family Code
8	48	8

以表 2 格式逆序输出的某 ROM ID:28 98 AA 4C 00 00 00 72 为例,最前面的 0x28 为家族代码,最后面的 0x72 为校验码,它由器件内部硬件电路移入前 56 位后自动生成,校验码初值固定为 0x00. 器件内 9 B 的 Scratchpad 包括环境温度、报警温度、配置信息等,最后面的字节同样为 CRC 校验码字节,该器件 ROM ID 及 Scratchpad 的校验码均由图 2 所示硬件生成电路生成.

图 2 所示 CRC 校验码生成多项式及硬件描述包含有 8 级移位寄存器及 3 个异或门,通过带反馈的线性移位寄存器实现与生成多项式的"除法"运

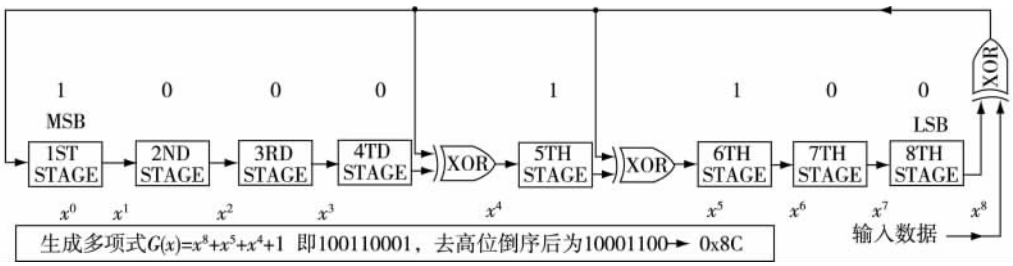

图 2　CRC-8-Dallas/Maxim 生成多项式与硬件描述

Fig. 2　CRC-8-Dallas/Maxim generating polynomial and hardware description

算. 最右边异或门的当前输出对应于图 1 所示模除示例中当前所置商位. 图 1 中每次商置 1 或 0 时，分别模减“10011”或“00000”(实际上是模减“0011”或“0000”)，这相当于图 2 所示硬件描述中通过反馈异或“10001100”或“00000000”，二者差别仅在于图 1 所示模除示例为正序计算，图 2 所示硬件生成电路为逆序处理.

主机读取器件数据后，将包括 CRC 字节在内的各字节逐一除以生成多项式，最终结果为 0x00 则表示传输正确，否则表示出错. 为加快通信速度，器件未给每一字节分别附加 CRC 校验码字节，而是采用了累计模加(异或)的方法[4]，将每一个字节的 CRC 校验码字节与下一个字节异或，再用生成多项式模除得到新的中间余数，如此下去，直到所有数据字节均被处理完毕.

2.2　CRC-8-Dallas/Maxim 校验算法设计

参考图 2 所示校验硬件描述，每一参与校码字节需要经过 8 次位右移，在当前余数最低位异于参与校验数据的最低位时，最右边的异或门输出“商值”1，反馈输出“10001100”(0x8C)，它们分别是 $x^0 \sim x^7$ 的系数；当输出“商值”0 时，反馈输出的 0x00 在硬件电路中不影响余数寄存器值，在软件实现时可以省略. 通过上述分析，可有 CRC8 校验算法具体实现如下：

```
void CRC8(unsigned char d) {
for(unsigned char i=0;i < 8;i++) {
if((uCRC8 ^ d) & 0x01)
uCRC8=(uCRC8>> 1) ^ 0x8C;
else uCRC8>>=1;
d>>=1;}}
```

其中校验码变量 uCRC8 为全局字节变量，d 为当前参与校验的数据字节.

假设主机读取的 8 B ROM ID 保存于字节数组 romcode [8]，低字节在前，高字节(CRC)在后，为判断读取的 ROM ID 是否可靠，有：

```
unsigned char i,uCRC8=0x00;
for(i=0;i < 8;i++) CRC8(romcode [i]);
if(uCRC8==0x00) {/* 校验通过* /};
else            {/* 校验失败* /};
```

上述循环可以缩减 1 次，在循环外比较 uCRC8 是否等于校验码字节 romcode [7] 同样可判断数据是否通过校验. 9 B Scratchpad 的数据校验与之类似，代码此略.

3　CRC-16-IBM 校验算法应用

3.1　Modbus 数据帧格式及其校验多项式

Modbus 总线协议是一种在工业制造领域中被广泛应用的网络协议，网络中的控制器可配置为 ASCII 或 RTU 通信模式，通信以帧为单位传送数据. 以 RTU 模式为例，其帧格式如表 3 所示，最后 2 B 为 16 位的 CRC 校验码. 所选择的 CRC-16-IBM 是 IBM 的二进制同步通信协议使用的生成多项式，它被 Modbus 总线系统所采用，Dallas/Maxim iButtons 系列器件的 RAM 数据校验使用的也是该多项式. Modbus 总线系统各节点向主机发送数据帧时，最后 2 B 的 CRC 校验码既可通过专用硬件电路生成，也可以通过软件计算得到.

表 3　RTU 通信帧格式

Table 3　RTU communication frame format

从机地址	功能码	数据区	校验码
1B	1B	0 ~ 252 B	2 B(CRC-16-IBM)

图 3 给出了 Modbus 总线系统中 CRC 校验生成多项式 $G(x)=x^{16}+x^{15}+x^2+1$ 及其硬件描述，它与图 2 所示硬件描述非常相似，主要差别仅在于 CRC-16-IBM 的“去高位逆序表示”为 1010000000000001 (0xA001)，且余数寄存器为 16 级(2 B).

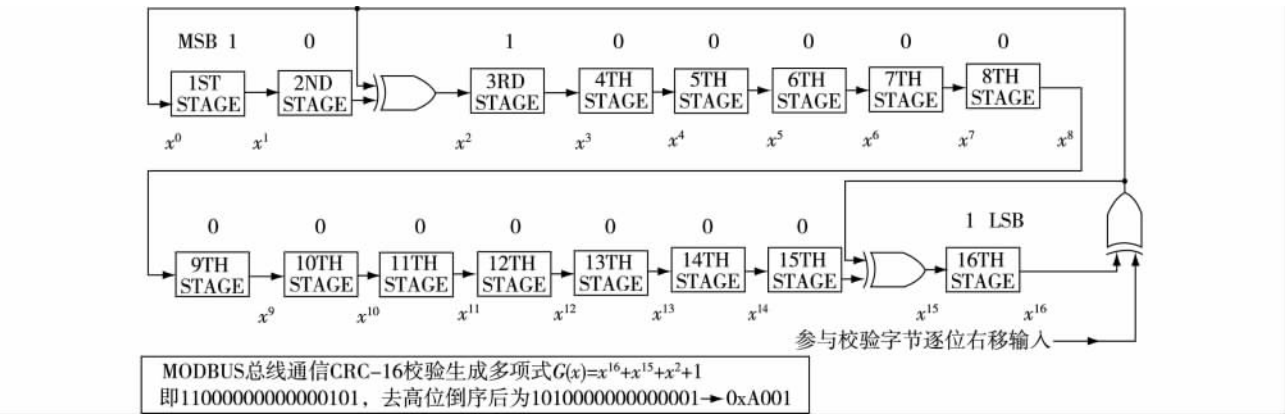

图3 MODBUS CRC-16 生成多项式与硬件描述

Fig. 3 MODBUS CRC-16 generating polynomial and hardware description

3.2 CRC-16-IBM 校验算法设计

图3所示校验硬件描述与图2非常相似,故二者的算法设计也很相似. CRC-16-IBM 校验算法 CRC16 具体实现如下,其中校验变量 uCRC16 为 unsigned int 类型(2 B):

```
void CRC16( unsigned char d) {
for( unsigned char i = 0; i < 8; i + + ) {
if( ( uCRC16 ^ d) & 0x0001)
uCRC16 = ( uCRC16≫ 1) ^ 0xA001;
else uCRC16≫ = 1;
d≫ = 1;}}
```

Modbus 总线系统中主机与从机的 CRC 校验码初值 uCRC16 在软件实现时要统一约定,例如可设为 0x0000 或 0xFFFF.

4 比特型校验算法的改进

4.1 比特型校验算法的数学推导

已经实现的基本比特型校验函数 CRC8 与 CRC16 均参照硬件生成电路描述设计. 如果将待校验二进制序列(例如保存于数组 romcode 中的 ROM ID)表示为

$B(x) = B_n \cdot 2^n + B_{n-1} \cdot 2^{n-1} + \cdots + B_1 \cdot 2 + B_0$,

其中 $B_n \sim B_0$ 共计为 $n+1$ 个比特位. 求取 CRC-8 校验码时有

$$\frac{B(x)\cdot 2^8}{G(x)} = \frac{B_n\cdot 2^8}{G(x)}\cdot 2^n + \frac{B_{n-1}\cdot 2^8}{G(x)}\cdot 2^{n-1} + \cdots + \frac{B_1\cdot 2^8}{G(x)}\cdot 2 + \frac{B_0\cdot 2^8}{G(x)}.$$

上式右边每一除式项均可得出一个商式及余式. 现将第1个除式项的商单列,其余式项则与第2个除式项合并,有

$$\frac{B(x)\cdot 2^8}{G(x)} = Q_n(x)\cdot 2^n + \left[\frac{R_n(x)\cdot 2}{G(x)} + \frac{B_{n-1}\cdot 2^8}{G(x)}\right]\cdot 2^{n-1} + \cdots + \frac{B_1\cdot 2^8}{G(x)}\cdot 2 + \frac{B_0\cdot 2^8}{G(x)}.$$

显然,方括号中的组合项将得出又一个商式和余式,依此递推下去,有

$$\frac{B(x)\cdot 2^8}{G(x)} = Q_n(x)\cdot 2^n + Q_{n-1}(x)\cdot 2^{n-1} + \cdots + Q_1(x)\cdot 2 + Q_0(x) + \frac{R_0(x)}{G(x)}.$$

最终可得唯一余式 $R_0(x)$,注意它是整个比特序列的最终余式,而不是仅由最后面的 B_0 位单独计算的余式. 递推过程中的关键步骤为方括号中的组合除式项,由该组合项可知:参与校验的二进制序列中,当前位参与校验后的校验码等于前一位参与校验所得校验码左移1位(乘以2)再加上当前位左移8位(乘以2^8),然后除以生成多项式. 其中 $R_n(x)\cdot 2$(即 $R_n(x)\ll 1$)的系数可记为 $R_7\langle R_6,\cdots,R_0,0\rangle$,另一项 $B_{n-1}\cdot 2^8$(即 $B_{n-1}\ll 8$)可记为 $B_{n-1}\langle 00000000\rangle$,二者合并的结果为 $(R_7 \hat{} B_{n-1})\langle R_6,\cdots,R_0,0\rangle$. 如果 $R_7 \hat{} B_{n-1} = 1$ 则需进一步求取 $\langle 1,R_6,\cdots,R_0,0\rangle$ 的余数,即 $\langle R_6,\cdots,R_0,0\rangle \hat{} G(x)$ 的十六进制表示,否则余数即 $\langle R_6,\cdots,R_0,0\rangle$.

参照上述推算过程,在软件实现时可以先对二者左移出的高位执行异或,再根据异或值决定是否对左移1位的余数异或生成多项式. 上述推导结果与 CRC-8-Dallas/Maxim 的软、硬件实现一致. 由于 DS18B20 内部硬件为逆序处理,最低位对应于最高次项,故校验函数 CRC8 应根据二者即将右移(≫)出的最低位执行异或比较. CRC-16 逐位校验的数学推导与之类似,此略.

4.2 改进后的比特型校验算法

求取 CRC-8 与 CRC-16 校验码时，所有参与校验的数据均以字节为单位，由此可考虑将二进制序列表示为 $B(x) = B_n(x) \cdot 2^{8n} + B_{n-1}(x) \cdot 2^{8(n-1)} + \cdots + B_1(x) \cdot 2^8 + B_0(x)$，其中 $B_n \sim B_0$ 共计为 $n+1$ 个字节，它们与原始二进制序列中由高位到低位每 8 位构成的分组一一对应. 以 CRC-8 为例，求取校验码时有

$$\frac{B(x) \cdot 2^8}{G(x)} = \frac{B_n(x) \cdot 2^8}{G(x)} \cdot 2^{8n} + \frac{B_{n-1}(x) \cdot 2^8}{G(x)} \cdot 2^{8(n-1)} + \cdots + \frac{B_0(x) \cdot 2^8}{G(x)} = Q_n(x) \cdot 2^{8n} + \left[\frac{R_n(x) \cdot 2^8}{G(x)} + \frac{B_{n-1}(x) \cdot 2^8}{G(x)}\right] \cdot 2^{8(n-1)} + \cdots + \frac{B_0(x) \cdot 2^8}{G(x)}.$$

进一步递推下去，最终同样可得整个字节序列的最终余式 $R_0(x)$. 由考察式中方括号内 2 项除式可知：当前字节 B_{n-1} 参与校验后的校验码等于此前校验码异或当前字节本身再求取的 CRC 校验码，即 $[R_n(x) + B_{n-1}(x)) \cdot 2^8] / G(x)$. 由该除式可知：对于原算法中循环内的 uCRC8 与 d，可以首先在循环外执行 uCRC8 ^= d，将当前字节 d 异或到前一字节参与校验后的校验码 uCRC8，然后进入循环对 uCRC8 执行模除操作(8 次移位及异或)，处理过程与图 1 所示模除示例对应，每次仅根据前一余数的高位即可决定当前是否需要将生成多项式的 16 进制表示异或到 uCRC8，具体实现时注意逆序处理. 改进后的比特型校验函数如下：

```
void CRC8(unsigned char d) {
uCRC8 ^= d; //先异或待校验字节 d
for(unsigned char i=0; i < 8; i++) {
unsigned char m = uCRC8 & 0x01;
uCRC8>>=1; if(m) uCRC8^=0x8C;}}
```

循环内语句还可进一步简化为：

```
unsigned char m = -(uCRC8 & 0x01);
uCRC8 = (uCRC8>>1) ^(0x8C & m);
```

简化后 if 分支语句被消除. 注意其中“m = ”后面是负数符号“-”，而非按位取反运算符“~”，由于 m 后面括号中的表达式值只可能为 0x01 或 0x00，且 -0x01 = 0xFF，-0x00 = 0x00，故而，当 uCRC8 低位为 1 时，uCRC8 = (uCRC8>>1) ^0x8C，反之则异或 0x00，相当于仅仅将 uCRC8 右移 1 位.

求取 $B(x)$ 的 CRC-16 校验码时，类似于前面的推导过程，有

$$\frac{B(x) \cdot 2^{16}}{G(x)} = \frac{B_n(x) \cdot 2^{16}}{G(x)} \cdot 2^{8n} + \frac{B_{n-1}(x) \cdot 2^{16}}{G(x)} \cdot 2^{8(n-1)} + \cdots + \frac{B_0(x) \cdot 2^{16}}{G(x)} = Q_n(x) \cdot 2^{8n} + \left[\frac{R_n(x) \cdot 2^8}{G(x)} + \frac{B_{n-1}(x) \cdot 2^{16}}{G(x)}\right] \cdot 2^{8(n-1)} + \cdots + \frac{B_0(x) \cdot 2^{16}}{G(x)}.$$

继续递推下去，同样可得整个字节序列最终的 16 位余式 $R_0(x)$. 方括号内组合多项式中 16 位的 R_n 左移 8 位与 8 位的 B_{n-1} 左移 16 位后异或，它相当于将当前字节 B_{n-1} 异或到 16 位余数的 R_n 的高字节(即高 8 次项)，所求取的余数即当前字节 B_{n-1} 参与校验后的余数. 由于图 3 及校验函数 CRC16 为逆序处理，低字节对应于高 8 次项，故 CRC16 应将待校验字节异或到 16 位余数的低字节，这相当于直接求取 $R_n + B_{n-1}$ 的余数. 改进后的比特型校验函数 CRC16 具体实现如下：

```
void CRC16(unsigned char d) {
uCRC16 ^= d;
for(unsigned char i=0; i < 8; i++) {
unsigned int m = -(uCRC16 & 0x0001);
uCRC16 = (uCRC16>>1) ^
(0xA001 & m);}}
```

经运行测试，“基本比特型”及“改进比特型”校验算法求取的校验码完全相同.

5 CRC 查表校验算法设计

5.1 单字节查表校验算法

“基本比特型”及“改进比特型”校验函数处理 1 B 均需要 8 次移位，不能很好地适应实时性要求较高的场合. 现以 CRC-8 为例，推导其改进型比特算法时已知当前字节 B_{n-1} 参与校验后的校验码等于前一校验码异或当前字节本身后再求取的校验码. 基于该推导结论，可以考虑提前构造 0x00 ~ 0xFF 的 256 B CRC 校验码码表[5-6]. 此后字节序列中每一字节参与校验时，可将其与前一 CRC 校验码异或，再以异或值为索引查表得到当前 CRC 校验码. 由于省去了对 $G(x)$ 的除法运算，查表校验速度将明显高于比特型校验算法. 现将所有 CRC-8-Dallas/Maxim 的校验码保存于数组 T8，仍以 ROM ID 校验为例，有

```
const unsigned char T8[256] = {
0x00,0x5E,0xBC,0xE2,0x61,0x3F,…,
0x0A,0x54,0xD7,0x89,0x6B,0x35};
unsigned char uCRC8 = 0x00;
for(unsigned char i=0; i < 8; i++)
uCRC8 = T8[uCRC8 ^ romcode[i]];
if(uCRC8 == 0x00) {/* 校验通过* /};
else              {/* 校验失败* /};
```

其中 uCRC8 ^ romcode [i] 与 $R_n + B_{n-1}$ 相对应,同时它也对应于改进后的比特型校验函数 CRC8 在循环外首先将 d 异或到 uCRC8.

对于求字节序列 CRC-16 校验码的递推式中的组合项,设 16 位的 R_n 高、低 8 位分别为 $R_{n\mathrm{H8}}$ 与 $R_{n\mathrm{L8}}$,代入 2 个除式项,重组后的除式项为

$$\frac{R_{n\mathrm{L8}}(x)\cdot 2^8}{G(x)}+\frac{[R_{n\mathrm{H8}}(x)+B_{n-1}(x)]\cdot 2^{16}}{G(x)}.$$

该式是求取当前字节 B_{n-1} 参与校验后的 CRC-16 校验码的递推式,式中前者求取的是 $R_{n\mathrm{L8}}\ll 8$ 的余数,由于 $R_{n\mathrm{L8}}\ll 8$ 将低 8 位的 $R_{n\mathrm{L8}}$ 移到了 16 位余数的高字节位置,所得 24 位被除数的最高 8 位全为 0,所得余数即 $R_{n\mathrm{L8}}\ll 8$. 由于图 3 及校验函数 CRC16 为逆序处理,低位为高次项,高位为低次项,对应于校验变量 uCRC16 则要将其高 8 位移到低字节的位置,即 uCRC16≫8. 同样由于为逆序处理,第 2 项中 $R_{n\mathrm{H8}}$ ^ B_{n-1} 将高 8 位的 $R_{n\mathrm{H8}}$ 与当前待校验字节异或后左移 16 位求取余数,它对应于将 uCRC16 的低 8 位与待校验字节 d 异或,即(uCRC16 & 0xFF) ^d,然后通过 for 循环移位操作求取余数,而余数的求取同样可通过查校验码表得到. 基于上述分析,对于 CRC-16-IBM,可先调用比特型校验函数 CRC16 求出以 0x0000 为初值的 256 项(512 B) CRC 校验码表[7],即

```
const unsigned int T16[256] = {
0x0000,0xC0C1,0xC181,0x0140,…,
0x4100,0x81C1,0x8081,0x4040};
```

现假设主机读取的数据帧 frame[N] 的最后 2 B(16 位)为校验码,为验证读取的整个数据帧是否可靠,可有如下校验程序:

```
unsigned int uCRC16 = 0xFFFF; //约定的 CRC 初值
for(unsigned char i = 0; i < N; i++)
uCRC16 = (uCRC16≫8) ^
T16[uCRC16 & 0xFF) ^ frame[i]];
if(uCRC16 = = 0x0000) {/* 校验通过* /};
else                  {/* 校验失败* /};
```

5.2 半字节查表校验算法

比特型校验算法处理速度慢,而单字节查表法占用 ROM 空间大,对此可以考虑选择一种折中的方法,即"半字节查表校验法"[8]. 现将待校验字节序列表示为

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

其中 $B_n\sim B_0$ 均为 4 位的半字节,仍以 CRC-8 为例,求取 $B(x)$ 的余数时作如下推导:

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

$$Q_n(x)\cdot 2^{4n}+\left[\frac{R_{n\mathrm{L4}}(x)\cdot 2^4}{G(x)}+\frac{R_{n\mathrm{H4}}(x)\cdot 2^8+B_{n-1}(x)\cdot 2^8}{G(x)}\right]\cdot 2^{4(n-1)}+\cdots+\frac{B_0(x)\cdot 2^8}{G(x)}.$$

方括号内 $R_{n\mathrm{L4}}$ 与 $R_{n\mathrm{H4}}$ 分别为 R_n 的低 4 位与高 4 位. 考察方括号内计算"当前半字节"B_{n-1} 参与校验后的校验码的递推组合多项式:第 1 项的余数值即为 $R_{n\mathrm{L4}}\ll 4$,它将使前一余数的低 4 位移到高 4 位,对于逆序处理的校验函数 CRC8,它相当于将 uCRC8 的高 4 位移到低 4 位,即 uCRC8≫4;对于第 2 项,同样由于为逆序处理,它相当于将 uCRC8 的低 4 位(即 uCRC8 & 0x0F)与当前参与校验的"半字节"异或并左移 8 位求取余数,该余数可通过查半字节校验码表得到. 由于函数参数 d 为字节类型,故要将其拆解为 2 个 4 位的半字节(即 d & 0x0F、d≫4),先处理低 4 位,后处理高 4 位,通过连续 2 次移位、查表及异或得到校验码. 设计 CRC-8 的半字节查表程序时,应先基于"0x8C"及初值 0x00 构造"半字节"码表,然后通过"半字节"查表程序得到校验码,具体实现如下:

```
const unsigned char T8_HB[] =
{0x00,0x9D,0x23,0xBE,…,0xE9,0x74};
void CRC8_H(unsigned char d){
uCRC8 = (uCRC8≫4) ^
T8_HB[uCRC8 & 0x0F) ^(d & 0x0F)];
uCRC8 = (uCRC8≫4) ^
T8_HB[uCRC8 & 0x0F) ^(d≫4)];}
```

对于 CRC-16-IBM,同样可以先基于"0xA001"及初值 0x0000 得到半字节码表:

```
const unsigned int T16_HB[] =
{0x0000,0xCC01,0xD801,…,0x4400};
```

校验函数 void CRC16_H(unsigned char d) 的设计与 CRC8_H 完全相同,仅需要分别用 uCRC16、T16_HB 替换 uCRC8、T8_HB. 限于篇幅,这里略去其数学推导. 由于为逆序处理,生成半字节校验码表时应取"0 ~ F"左移 4 位,即"00、10、20、…、F0"的 CRC-8 或 CRC-16 校验码,这样才能保证前一参与校验的半字节是"0 ~ F"中的某一个,否则前一参与校验的半字节将恒为"0".

5.3 一种基于块与多表的校验算法

为适应系统对校验处理速度的更高要求，下面提出一种基于块与多表的校验算法，拟一次读入多字节进行校验处理. 以 CRC-16 为例，假设待校验字节数为偶数(为奇数时可用约定字节补齐)，设为 $2N$，每次读入一个数据块(暂设为 2 B)参与校验. 类似于此前的推导，可将二进制序列表示为

$$B(x) = B_n(x) \cdot 2^{16n} + B_{n-1}(x) \cdot 2^{16(n-1)} + \cdots + B_1(x) \cdot 2^{16} + B_0(x),$$

其中 $B_n \sim B_0$ 表示共有 $n+1$ 块，求取校验码时有

$$\frac{B(x) \cdot 2^{16}}{G(x)} = \frac{B_n(x) \cdot 2^{16}}{G(x)} \cdot 2^{16n} + \frac{B_{n-1}(x) \cdot 2^{16}}{G(x)} \cdot 2^{16(n-1)} + \cdots + \frac{B_0(x) \cdot 2^{16}}{G(x)} = Q_n(x) \cdot 2^{16n} + \left[\frac{R_n(x) \cdot 2^{16}}{G(x)} + \frac{B_{n-1}(x) \cdot 2^{16}}{G(x)}\right] \cdot 2^{16(n-1)} + \cdots + \frac{B_0(x) \cdot 2^{16}}{G(x)}.$$

递推式中方括号内的代数式表示当前 16 位的块参与校验后的校验码为前一数据块参与校验后的校验码与当前数据块异或后再求取的校验码. 而对于异或后所得的 16 位的块序列，如果直接通过查表法获取余数，需要构造高达 $2^{16} \cdot 2 = 128$ kB 的校验码表，其 ROM 占用不是多数系统所能提供的.

现假设当前参与校验的 16 位的块为 $< R_{15} \sim R_0 >$，求取余数时可将其拆分为 $< R_{15} \sim R_8$，00000000 > 与 $< R_7 \sim R_0 >$，最终余数可由前 16 位的余数异或后 8 位的余数得到. 求取后 8 位 $< R_7 \sim R_0 >$ 的余数时，可直接使用单字节查表算法中 512 B 的校验码表 T16，这里将表名重记为 CRC16_8. 对于 16 位序列 $< R_{15} \sim R_8$，00000000 >，求取其校验码时，可用此前任意一种 CRC16 校验算法先得出前一字节 $< R_{15} \sim R_8 >$ 的余数，再接着使 <00000000 >(即 0x00)参与校验，如此即可为任意序列 $< R_{15} \sim R_8$，00000000 > 构造出校验码表 CRC16_16，由于该序列后 8 位恒为 0，故该码表同样为 512 B(256 项). 2 个校验码表占用 ROM 空间共计为 1 kB. 基于上述分析推导，可有如下基于块与多表(这里为双字节与双表)的校验算法:

```
const unsigned int CRC16_8 [256] = {
0x0000,0xC0C1,…,0x8081,0x4040};
const unsigned int CRC16_16 [256] = {
0x0000,0x9001,…,0x6040,0xF041};
void CRC16_tt( unsigned char * d) {
uCRC16 ^= * ( unsigned int* ) d;
uCRC16 = CRC16_16 [uCRC16 & 0xFF]
^ CRC16_8 [uCRC16≫ 8];}
```

CRC16_tt 内第 1 行代码将指针类型转换为 unsigned int* ，使 2 B 的数据块(即* d、* (d + 1))首先异或到 uCRC16. 由于为逆序处理，接下来使用 CRC16_16 查表的是 uCRC16 的低 8 位，使用 CRC16_8 查表的是 uCRC16 的高 8 位. $2N$ B 的数据仅通过 N 次函数调用即可完成校验. 类似地，可设计以更大数据块为单位的多表 CRC 校验算法.

6 码表 ROM 空间及校验处理速度比较

以 CRC16 为例，表 4 列出了几种校验算法的校验码表对 ROM 空间的占用情况. 通过在主频配置为 4 MHz 的 PIC18F458 微处理器上对 1 kB 数据进行校验处理测试，得出了各算法的校验处理时间. 由表 4 显示数据可知: 2 种比特型算法均不需要码表 ROM 空间，且改进型算法校验速度高于基本型算法，但二者的校验处理速度均远远低于后 3 种算法. 如果码表占用 1 kB ROM 空间可以被系统接受，本文提出的基于块及多表的校验算法处理速度显然是最快的，它所需处理时间约为基本比特型算法的 9.27%，约为单字节查表算法的 71.24%.

表 4 码表 ROM 空间占用及校验处理速度比较

Table 4 Comparison of ROM occupancy and check time

序号	CRC16 校验算法	码表占用 ROM 空间/B	1 kB 数据校验耗时/ms
1	基本比特型	—	497.69
2	改进比特型	—	327.70
3	单字节查表	512	65.55
4	半字节查表	32	159.76
5	块-多表校验	1 024	46.17

7 结束语

通过研究 CRC 校验生成多项式及相关硬件描述，本文给出了基本比特型校验算法，并在此基础上通过数学推导得出了改进比特型校验算法及单字节、半字节查表校验算法，并提出一种基于块及多表的校验算法，所有相关算法程序均通过了实测验证. 在应用于具体的嵌入式系统设计时，应在校验码 ROM 空间占用与校验处理速度之间权衡选择.

在嵌入式系统中正确使用 CRC 校验算法，可大大提高系统数据传输的可靠性. 本文的研究将为嵌

入式系统设计者在开发过程基于所选择的生成多项式进行正确的校验算法设计提供重要参考.

参考文献

References

[1] 刘星华. CRC 校验在单片机系统中的软件快速实现[J]. 福建工程学院学报,2007,5(1):76-78
LIU Xinghua. CRC-16 high speed implementation in single-chip computer [J]. Journal of Fujian University of Technology,2007,5(1):76-78

[2] 杨利娟,陈多观. 循环冗余校验 CRC 的分析及硬件实现[J]. 苏州科技学院学报:自然科学版,2010,27(4):51-53
YANG Lijuan, CHEN Duoguan. Analysis and hardware implementation of cyclic redundancy check code [J]. Journal of Suzhou University of Science and Technology: Natural Science Edition,2010,27(4):51-53

[3] 潘矜矜,潘丹青. 基于单片机的 CRC 算法的 C51 实现[J]. 桂林航天工业高等专科学校学报,2009(1):25-26,29
PAN Jinjin, PAN Danqing. CRC algorithm achived by C51 based on SCM [J]. Journal of Guilin College of Aerospace Technology,2009(1):25-26,29

[4] 邹久朋,林瑶瑶,周建. CRC 校验编程和硬件快速校验探讨[J]. 单片机与嵌入式系统应用,2009(4):76-78
ZOU Jiupeng, LIN Yaoyao, ZHOU Jian. Discussion on CRC coding and hardware fast check [J]. Microcontrollers & Embedded Systems,2009(4):76-78

[5] Kounavis, M E, Berry, F L. A systematic approach to building high performance software-based CRC generators [C] // Proceedings of the 10th IEEE Symposium on Computers and Communications,2005:855-862

[6] Williams R N. A painless guide to CRC error detection algorithms [EB/OL]. [2008-10-28]. http: // www. repairfaq. org/filipg

[7] 胡惠玉. 16 位 CRC 校验原理与基于 PLC 的算法程序设计[J]. 常熟理工学院学报:自然科学版,2009,23(10):80-83
HU Huiyu. 16-bit CRC checksum principle and programming based on PLC [J]. Journal of Changshu Institute of Technology: Natural Science Edition, 2009, 23 (10): 80-83

[8] 戴志超. 基于物理模型的 CRC 算法分析与程序设计[J]. 计算机应用与软件,2009,26(11):141-143
DAI Zhichao. Physical model based CRC algorithms analysis and program design [J]. Computer Applications and Software,2009,26(11):141-143

Research on embedded system CRC algorithm design

PENG Wei[1]

1 School of Electronic and Information Engineering, Wuhan City Vocational College, Wuhan 430064

Abstract This paper introduces the basic principle of Cyclic Redundancy Check(CRC) and the form of generating polynomial, analyzes the generating polynomial and hardware description of CRC-8-Dallas/Maxim and CRC-16-IBM in embedded system. In the case study of the data checking of ROM ID/Scratchpad of DS18B20 device and the bus network data frame checking of Modbus, through analyzing and researching into the generating polynomial and hardware description, the paper gets a basic bit-to-bit check algorithm design, then obtains an advanced bit-to-bit check algorithm and a single byte and a half byte table lookup check algorithm on the basis of mathematical derivation. In order to improve check speed, this paper proposes a new method based on block and multi-table checking algorithm. Comparison in ROM space occupancy and processing speed is carried out among all the above check algorithms. The designed CRC program provides support for the reliable transmission of embedded system data.

Key words embedded system; CRC; error control; check; generating polynomial; algorithm