C

ASM51

VB

C51

循环冗余校验码（CRC）计算

——C、VB、C51、ASM51编程实例

杜树春 编著

Cyclic Redundancy Check
Code Calculation

usingC 、VB 、C51 and ASM51 Programming Language

清華大學出版社

Cyclic Redundancy Check Code Calculation

using C、VB、C51 and ASM51 Programming Language

循环冗余校验码（CRC）计算

——C、VB、C51、ASM51编程实例

杜树春 编著

清华大学出版社

北京

内 容 简 介

本书是一本用多种不同计算机语言编程计算循环冗余校验码的程序集。全书共分4章，第1章是概述，介绍循环冗余校验码的概念。第2章是8位循环冗余校验码的计算程序，所用计算机语言包括C语言、VB语言、C51语言和ASM51汇编语言。第3章是16位循环冗余校验码的计算程序，所用计算机语言同第2章。第4章是32位循环冗余校验码的计算程序，所用计算机语言包括C语言和C51语言。

本书适合以下人员阅读或参考。一是学习C语言或VB语言或51系列单片机课程的大、中专及高等职业学校、中等职业学校的在校学生；二是使用这些语言的广大工程技术人员；三是这些语言编程的初学者；四是广大程序设计爱好者。

图书在版编目(CIP)数据

循环冗余校验码(CRC)计算：C、VB、C51、ASM51编程实例/杜树春编著. —北京：清华大学出版社，2017

ISBN 978-7-302-47650-4

Ⅰ. ①循…　Ⅱ. ①杜…　Ⅲ. ①程序设计　Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2017)第156515号

责任编辑：文　怡
封面设计：李召霞
责任校对：徐俊伟
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：12.25　　**字　　数**：241千字
版　　次：2017年9月第1版　　**印　　次**：2017年9月第1次印刷
印　　数：1～2000
定　　价：39.00元

产品编号：075545-01

前言

自从人类进入信息社会后，有线的或无线的通信网络便遍布全球。在通信系统中，由于存在干扰或信道传输特性不好，容易对信道造成不良的影响。通过对所传输的数字信息进行特殊的处理（如差错控制编码），即可实现对传输信息中错误的自动检验，称为检错。检错对数字通信系统的有效传输起着重要的作用。

检错的手段有三种，一是奇偶校验，二是累加和校验，三是循环冗余校验。循环冗余校验是这三种校验中最好的一种。

同一组数据采用不同的生成多项式，将得出不同的循环冗余校验码。一般说来，一组数据的 CRC 值由四个因素决定，一是生成多项式，二是该生成多项式正序还是反序，三是余数初始值是什么（通常有 00H 和 0FFH，0000H 或 0FFFFH，00000000H 或 0FFFFFFFFH 的选择），四是结果异或值是 00000000H 还是 0FFFFFFFFH（结果异或值只在 CRC32 码中考虑）。当生成多项式固定，正反序固定，余数初始值固定，结果异或值也固定时，同一组数的 CRC 值便是唯一的。

本书是一本用不同的计算机语言编程计算循环冗余校验码的程序集。常用的循环冗余校验码有 8 位的、16 位的和 32 位的三种，每种之下又有不同的生成多项式、正序反序、余数初始值取值和结果异或值的区分。所用计算机语言有四种：分别是 C 语言、VB 语言、C51 语言和 ASM51 汇编语言，前两种在个人电脑或笔记本电脑上运行，后两种在 51 系列单片机上运行。

本书共分 4 章，第 1 章是概述，介绍循环冗余校验码的概念。第 2 章是 8 位循环冗余校验码的计算程序，生成多项式①为 $x^8+x^5+x^4+1$，包括正序和反序；生成多项式②为 x^8+x^2+x+1，包括正序。所用计算机语言包括 C 语言、VB 语言、C51 语言和 ASM51 汇编语言。第 3 章是 16 位循环冗余校验码的计算程序，生成多项式①为 $x^{15}+x^{13}+1$；生成多项式②为 $x^{16}+x^{12}+x^5+1$。所用计算机语言包括 C 语言、VB

语言、C51 语言和 ASM51 汇编语言。第 4 章是 32 位循环冗余校验码的计算程序，生成多项式为 $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$，包括正序和反序，所用计算机语言包括 C 语言和 C51 语言。

本书最后有三个附录，分别是介绍 Visual C++ 6.0 上机操作的附录 A，介绍单片机开发软件 Keil C51 使用方法的附录 B 和介绍 Visual Basic 6.0 上机操作的附录 C。

电子资料包的内容，仍是以书中章节为单位。在每一章(指第 2 章～第 4 章)下，都有一个章文件夹，每章文件夹下面有节文件夹，节文件夹内有四个文件夹，分别是 C 语言程序夹、VB 语言程序夹、C51 语言程序夹和 ASM51 汇编语言程序夹。打开任意一个程序文件夹，里面是例子文件夹。如果是 C 语言程序夹下的例子文件夹，打开后，多个文件中必有扩展名是.dsw 的文件，在 Visual C++ 6.0 软件已安装在电脑中的前提下，双击具有.dsw 扩展名的文件就可进入 Visual C++ 6.0 软件集成环境，也就是 Visual C++ 6.0 的编辑、编译、连接、调试、运行环境。如果是 VB 语言程序夹下的例子文件夹，打开后，多个文件中必有扩展名是.vbp 的文件，在 Visual Basic 6.0 软件已安装在电脑中的前提下，双击具有.vbp 扩展名的文件就可进入 Visual Basic 6.0 软件集成环境，也就是 Visual Basic 6.0 的编辑、编译、连接、调试、运行环境。如果是 C51 语言程序夹或 ASM51 汇编语言程序夹下的例子文件夹，打开后，多个文件中必有扩展名是.Uv2 的文件，在单片机开发软件 Keil C51 已安装在电脑中的前提下，双击具有.Uv2 扩展名的文件，就可进入 C51 语言和 ASM51 汇编语言软件集成环境。

本书适合以下人员阅读或参考。一是学习 C 语言、VB 语言或 51 系列单片机课程的大、中专及高等职业学校、中等职业学校的在校学生；二是使用这些语言的广大工程技术人员；三是这些语言编程的初学者；四是广大程序设计爱好者。

通过这本 CRC 程序集的出版，希望达到如下目的：当广大编程人员遇到 CRC 计算的应用问题时，能从本书中查到符合自己所需位数、所需生成多项式和合适程序语言的程序模块，从而加快编程的工作进度。

由于编著者水平有限且时间仓促，书中难免存在缺点和错误，恳请读者批评指正。我的电子邮箱为 dushuchun@263.net。

本书电子资料包请扫描二维码下载。

编著者

2017 年 4 月

目录

第1章

概　述

1.1　循环冗余校验简介

在远距离数据通信中，为确保高效而无差错地传输数据，必须对数据检错，即差错控制。有多种检错的方法，如奇偶校验、算术累加和校验及循环冗余校验。而循环冗余校验是其中最好的一种。

循环冗余校验(Cyclic Redundancy Check，CRC)是目前运用非常广泛的一种数据校验方式。其特点是：检错能力极强，开销小，易于用编码器及检测电路实现。从检错能力来看，其不能发现错误的几率在0.0047%以下；从性能和开销上考虑，其远远优于奇偶校验及算术累加和校验等方式。因而，在计算机网络、磁盘存储和数据通信等领域，CRC无处不在。

Modbus通信协议下有两种通信模式：一种是ASCII(美国标准信息交换代码)通信模式，另一种是RTU(远程终端单元)通信模式，后者就采用了CRC校验方法。著名的通信协议X.25的FCS(帧检错序列)采用的是CRC-CCITT；磁盘驱动器的读写

采用的是CRC16；ARJ、LHA等压缩工具软件采用的是CRC32；一些半导体厂商生产的单总线芯片用CRC8校验其芯片固有编号，如芯片DS18B20的8字节的序列号，最后1个字节是前面7个字节的CRC码，这是为了保证序列号的唯一性与正确性；此外通用的图像存储格式GIF、TIFF等都用CRC作为检错手段。

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r位监督码(即CRC码)，并附在信息后面，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。

CRC校验可以简单地描述为：例如我们要发送一些数据(信息字段)，为了避免一些干扰并在接收端判断接收的是否是真实的数据，这时就要加上校验数据(即CRC校验码)，以判断接收的数据是否正确。在发送端，根据要传送的k位二进制码序列，以一定的规则(CRC校验有不同的规则，规则一词，在差错控制理论中称为“生成多项式”)产生一个校验用的r位校验码(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(即与发送时生成CRC校验码相同的规则)进行检验，校验采用计算机的模2除法，即除数和被除数(即生成多项式)做异或运算，进行异或运算时除数和被除数最高位对齐，进行按位异或运算，若最终的数据能被除尽，则传输正确；否则，传输错误。

生成CRC码的多项式$g(x)$又叫生成多项式(Generation Polynomial)，生成多项式有多种(见表1-1)。生成多项式不同，产生的CRC码也不同。这就类似作除法时被除数相同，除数不同，所求得商和余数不同。生成多项式如果是(4+1)位的，则产生4位的CRC4码；如果是(8+1)位的，则产生8位的CRC8码；如果是(12+1)位的，则产生12位的CRC12码；如果是(16+1)位的，则产生16位的CRC16码；如果是(32+1)位的，则产生32位的CRC32码。

表1-1 标准CRC码生成多项式

名　称	生成多项式	简记式	标准引用
CRC8	$x^8+x^5+x^3+1$	0x129	
CRC8	x^8+x^2+x+1	0x107	
CRC8	$x^8+x^6+x^4+x^3+x^2+x$	0x15E	
CRC12	$x^{12}+x^{11}+x^3+x^2+x+1$	0x180F	
CRC16	$x^{16}+x^{15}+x^2+1$	0x18005	IBM SDLC
CRC16-CCITT	$x^{16}+x^{12}+x^5+1$	0x11021	ISO HDLC, ITU X.25

续表

名　称	生成多项式	简记式	标准引用
CRC16-REV	$x^{15}+x^{13}+1$	0xA001	
CRC32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	0x04C11DB7	ZIP, RAR, IEEE 802

CRC 校验码的产生的大致步骤是：把要发送数据(通常是由多个字节组成的数组)以二进制的格式排列起来，拿生成多项式所对应的二进制数去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。

从表 1-1 可以看出 CRC 码有 8 位到 32 位的，常用的是 8 位(一个字节)的 CRC8 码、16 位(两个字节)的 CRC16 码和 32 位(四个字节)的 CRC32 码。以下主要介绍 CRC8 码、CRC16 码和 CRC32 码的生成方法。

1.2 循环冗余校验的基本过程

CRC 校验的基本过程：

采用 CRC 校验时，通信的发送方和接收方用同一个生成多项式 $g(x)$，多项式正序反序，初始值也相同，并且 $g(x)$的首位和最后一位的系数必须为 1。

CRC 检验的处理方法是：发送方用发送数据的二进制多项式 $t(x)$除以 $g(x)$，得到余数 $y(x)$作为 CRC 校验码。校验时，以计算的校正结果是否为 0 为依据，判断数据帧是否出错。设生成多项式是 r 阶的(最高位是 $x\hat{}r$)，具体步骤描述如下。

发送方：

(1) 在发送的 m 位数据的二进制多项式 $t(x)$后添加 r 个 0，扩展到 $m+r$ 位，以容纳 r 位的校验码，追加 0 后的二进制多项式为 $T(x)$。

(2) 用 $T(x)$除以生成多项式 $g(x)$，得到 r 位的余数 $y(x)$，它就是 CRC 校验码。

(3) 把 $y(x)$追加到 $t(x)$后面，此时的数据 $s(x)$就是包含了 CRC 校验码的待发送字符串；由于 $s(x)=t(x)y(x)$，因此 $s(x)$肯定能被 $g(x)$除尽。

接收方：

(1) 接收数据 $n(x)$，这个 $n(x)$就是包含了 CRC 校验码的 $m+r$ 位数据；

(2) 计算 $n(x)$除以 $g(x)$，如果余数为 0 则表示传输过程没有错误，否则表示有错误。从 $n(x)$去掉尾部的 r 位数据，得到的就是原始数据。

1.3 使用循环冗余校验码的一个例子

Modbus 通信协议是工业控制领域实现单个电子控制器互联的通信协议，实际上它已成为一种通用的工业标准。硬件采用 RS232 串行口或 RS485 串行口，通信方式为主从式半双工通信，主机呼叫从机地址，从机应答通信。数据帧共 10 位，1 个起始位，8 个数据位，1 个停止位，无校验。波特率：9600 或 19200。图 1-1 是采用 Modbus 通信协议的总站、分站通信系统结构示意图。以下介绍 Modbus 通信协议的通信格式。

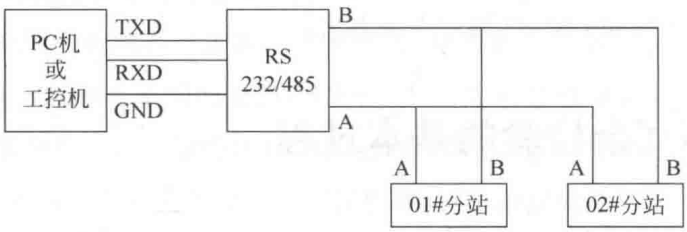

图 1-1 采用 Modbus 通信协议的总站、分站通信系统结构示意图

1.3.1 Modbus 通信协议通信格式说明

1. 功能 03H：读寄存器值

主机发送：

1	2	3	4	5	6	7	8
ADR	03H	起始寄存器高位	起始寄存器低位	寄存器数高位	寄存器数低位	CRC 低位	CRC 高位

第 1 字节 ADR : 从机地址码(= 001～254)
第 2 字节 03H : 读寄存器值(功能码)
第 3、4 字节 : 要读的寄存器开始地址
第 5、6 字节 : 要读的寄存器数量
第 7、8 字节 : 从字节 ADR 起的数据校验和

从机回送：

1	2	3	4、5	6、7		$M-1$、M	$M+1$	$M+2$
ADR	3H	字总数	寄存器数据 1	寄存器数据 2	…	寄存器数据 M	CRC 低位	CRC 高位

第 1 字节　ADR　：　从机地址码(= 001～254)
第 2 字节　03H　：　读寄存器值(功能码)
第 3 字节　字总数　：　从 4～M(包括 4 及 M)的字总数
第 4～M 字节　：　寄存器数据
第 $M+1$、$M+2$ 字节　：　从字节 ADR 起的数据校验和

当从机接收错误时，从机回送：

1	2	3	4	5
ADR	83H	错误信息码	CRC 低位	CRC 高位

第 1 字节　ADR　：　从机地址码(= 001～254)
第 2 字节　83H　：　读寄存器值出错
第 3 字节　错误信息码　：　见错误信息码表
第 4、5 字节　：　从字节 ADR 起的数据校验和

2. 功能 04H：读寄存器值

主机发送：

1	2	3	4	5	6	7	8
ADR	04H	起始寄存器高位	起始寄存器低位	寄存器数高位	寄存器数低位	CRC 低位	CRC 高位

从机回送：

1	2	3	4、5	6、7		$M-1$、M	$M+1$	$M+2$
ADR	4H	字总数	寄存器数据 1	寄存器数据 2	…	寄存器数据 M	CRC 低位	CRC 高位

当从机接收错误时，从机回送：

1	2	3	4	5
ADR	84H	错误信息码	CRC 低位	CRC 高位

第 1 字节　ADR　　：　从机地址码(= 001～254)
第 2 字节　84H　　：　读寄存器值出错
第 3 字节　错误信息码　：　见错误信息码表
第 4、5 字节　　：　从字节 ADR 起的数据校验和

3. 功能 06H：写单个寄存器值

主机发送：

1	2	3	4	5	6	7	8
ADR	06H	起始寄存器高位	起始寄存器低位	数据高字节	数据低字节	CRC 低位	CRC 高位

从机回送：

1	2	3	4	5	6	7	8
ADR	06H	起始寄存器高位	起始寄存器低位	数据高字节	数据低字节	CRC 低位	CRC 高位

当从机接收错误时，从机回送：

1	2	3	4	5
ADR	86H	错误信息码	CRC 低位	CRC 高位

第 1 字节　ADR　　：　从机地址码(= 001～254)
第 2 字节　86H　　：　读寄存器值出错
第 3 字节　错误信息码　：　见错误信息码表
第 4、5 字节　　：　从字节 ADR 起的数据校验和

1.3.2 循环冗余码 CRC16 码的用法

根据上面的 Modbus 通信协议，主机发送 8 个字节，先把前 6 个字节的 CRC 校验码算出，依照低位在前的原则，将其添到后两个字节中，然后发送出去。从机收到 8 个字节后，也是先把前 6 个字节的 CRC 校验码(仍用同一生成多项式)算出，与后两个字节相比，若相同，说明 CRC 校验通过，可执行下一步。若不同，则 CRC 校验未通过，从机应回送错误信息码，要求主机重发。

第2章

CRC8码的计算

2.1 用手工计算 CRC8 码

常用标准 CRC8 码生成多项式如表 2-1 所示。

表 2-1 常用标准 CRC8 码生成多项式

编号	名称	生成多项式	简记式 *	标准引用
1	CRC8	$x^8+x^5+x^4+1$	0x131	
2	CRC8	$x^8+x^5+x^3+1$	0x129	
3	CRC8	x^8+x^2+x+1	0x107	
4	CRC8	$x^8+x^6+x^4+x^3+x^2+x$	0x15E	

在上述 4 个常用标准 CRC8 码生成多项式中，本章着重分析编号为 1 和 3 的两个生成多项式。

CRC8 最终生成的 CRC 校验码为 1 字节，假如其生成多项式 $g(x)=x^8+x^5+x^4+1$，相当于 $g(x)=1\cdot x^8+0\cdot x^7+0\cdot x^6+1\cdot x^5+1\cdot x^4+0\cdot x^3+0\cdot x^2+0\cdot$

$x^1+1\cdot x^0$，即对应的二进制数为 0x131(100110001B)。

CRC8 校验算法：

1. CRC8 正序校验法或常规算法

例题 1* 已知信息码为 0x01，生成多项式 $g(x)=x^8+x^5+x^4+1$，求对应的 CRC8 码。

解：第一步：0x01 右边补 8 个 0，变为 1 0 0 0 0 0 0 0 0，多项式为 1 0 0 1 1 0 0 0 1B=0x131。

第二步：100000000 异或 100110001 得余数 1 1 0 0 0 1，因余数小于生成多项式，那么，余数即为 CRC 码，110001 的十六进制数为 0x31。其计算竖式为

```
  100000000
^ 100110001
  ---------
    110001
```

现在我们再用普通的算术除法，计算上面的式子。100000000/100110001 = 0x100/0x131，因为被除数小于除数，商为 0，余数就是被除数 0x100。可见，求 CRC8 码所用的除法(模 2 除法)，并不是普通算术除法，两者的计算结果也大不相同。

例题 2 已知，信息字段代码为 00000001 00000010，对应 $m(x)=x^8+x$；生成多项式 $g(x)=x^8+x^5+x^4+1$，对应 $g(x)$的二进制代码为 100110001。求包括对应的 CRC8 码在内的待发送的数据。

解：现在我们将要对 2 字节数据 0x01，0x02 生成 CRC8 校验码，并最终将生成的 1 字节 CRC 校验码跟在 0x01，0x02 的后面，即 0x01，0x02，＃＃(＃＃即 8 位 CRC 码)，最终生成的 3 字节数据就是待发送的数据。

先计算 $x^8m(x)=x^{16}+x^9$，对应的二进制数为 100000010 00000000。可以看到运算所得结果其实就是将信息字段代码的数左移 8 位。因为最终要将生成的 8 位 CRC8 校验码附在信息字段的后面，所以要将信息字段的数左移 8 位。最后用 $x^8m(x)$得到的二进制数对生成多项式 $g(x)$进行模 2 运算，最终的余数(其二进制数的位数一定比生成多项式 $g(x)$的位数小)就是所要的 CRC8 校验码。

```
  100000010 00000000
^ 100110001
 --------------------
```

---

* 手工计算例题编号为例题 1、2、3，与后面程序计算例题编号例 2.1、2.2 区分。

```
  000110011 00000000
^  100110 001
--------------------
    010101 00100000
^    10011 0001
--------------------
      00110 00110000
^      100 110001
--------------------
        010 11110100
^        10 0110001
--------------------
          00 10010110
```

对 $x^8 m(x)$ 做模 2 运算取余得 10010110(0x96)，这个 8 位的二进制数就是 CRC8 校验码。所以，经 CRC8 校验后待发送的数据就是 0x01，0x02，0x96。

我们再用通常的算术除法，计算上面的式子。100000010 00000000/100110001＝0x10200/0x131＝66048/305＝216，余数为 168＝0xa8。这再次说明，求 CRC8 码所用的除法(模 2 除法)，并不是普通算术除法，两者的计算结果也大不相同。

【说明】 “模 2 除法”与“算术除法”类似，但它既不向上位借位，也不比较除数和被除数的相同位数值的大小，只要以相同位数进行相除即可。模 2 加法运算为：1＋1＝0，0＋1＝1，0＋0＝0，无进位，也无借位；模 2 减法运算为：1－1＝0，0－1＝1，1－0＝1，0－0＝0，也无进位，无借位。相当于二进制中的逻辑异或运算。也就是比较后，两者对应位相同则结果为“0”，不同则结果为“1”。这里举一个“模 2 除法”的例子，如 100101 除以 1110，结果得到商为 11，余数为 1，如图 2-1 所示。此例的算术除法结果为 100101B/1110B＝25H/0EH＝37/14，商为 2，余数为 9。两者显然不同。

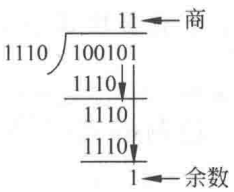

图 2-1 “模 2 除法”的一个例子

2. 逆序(或反序)CRC8 校验法

DS18B20 是一种由达拉斯(DALLAS)公司生产的温度传感器芯片。在 DS18B20 中，有两处用到 CRC8 码，一处是 DS18B20 的 8 字节的序列号，最后一字节是前面 7