**中国海洋大学计算机科学与技术学院**

实验报告

**姓名：惠欣宇 年级：2021级 专业：计算机科学与技术**

**科目：物联网系统设计与开发 题目：校验码生成**

**实验时间：2024年 4月 12日 实验教师:郭忠文**

**目录**

[一、实验目的及要求 2](#_Toc164302036)

[1. 实验目的 2](#_Toc164302037)

[2. 实验要求 2](#_Toc164302038)

[二、实验内容及步骤 2](#_Toc164302039)

[1. 实验内容 2](#_Toc164302040)

[2. 实验步骤 2](#_Toc164302041)

[三、报告要求 3](#_Toc164302042)

[1. 说明校验码的主要用途 3](#_Toc164302043)

[2. CheckSum（校验和） 3](#_Toc164302044)

[2.1 主要功能 3](#_Toc164302045)

[2.2 使用方法 4](#_Toc164302046)

[2.3 计算结果 6](#_Toc164302047)

[3. CRC16（循环冗余校验） 8](#_Toc164302048)

[3.1 主要功能 8](#_Toc164302049)

[3.2 使用方法 10](#_Toc164302050)

[3.3 计算结果 11](#_Toc164302051)

[4. 实验遇到的主要问题和解决方法 13](#_Toc164302052)

[4.1 问题及解决方法1 13](#_Toc164302053)

[4.2 问题及解决方法2 15](#_Toc164302054)

[四、实验总结 15](#_Toc164302055)

一、实验目的及要求

1. 实验目的

通过调试运行通信协议中常用校验码的计算程序，学会典型校验码生成软件的使用，为后续复杂通信协议的运用和传感数据的校验打下基础。

2. 实验要求

掌握调用校验码生成程序进行校验码生成的方法，保留调试后的程序，供后续实验使用，实验完成后撰写实验报告。

二、实验内容及步骤

1. 实验内容

针对给出的字符串“@~ABCDEFG$#%”，在本实验系统提供的 C 语言代码基础上进行修改，编写 C 语言程序，分别生成“校验和”及“CRC16”校验码，并在计算机屏幕上显示。

2. 实验步骤

（1）针对给出的字符串“@~ABCDEFG$#%”，对计算机桌面上“琏雾实验系统\实验 3.1\CheckSum”文件夹中的 main.c 进行调用（程序的开发运行环境为 Visual Studio 2013），编写 C 语言程序，调用 getCheckSum 函数，实现“校验和”校验码的计算，并在计算机屏幕上显示，函数调用说明见本文件夹中的“校验和函数调用说明.txt”；

（2）针对给出的字符串“@~ABCDEFG$#%”，对计算机桌面上“琏雾实验系统\实验 3.1\ CRC16”文件夹中的 main.c 进行修改，编写 C 语言程序，调用 CRC16\_MODBUS 函数，实现 CRC16 循环冗余校验码的计算，并在计算机屏幕上显示，函数调用说明见本文件夹中的“循环冗余码函数调用说明.txt”。

三、报告要求

1. 说明校验码的主要用途

校验码主要是为了解决计算机各部件进行数据传输和交换，确保传送过程的正确无误，一是为了**提高硬件电路的可靠性**，二是**提高代码的校验能力**。通信协议中的校验码是用于**确保数据传输的准确性和完整性**的一种机制。这里根据上学期计算机网络课程对该方面的知识的学习并结合本学期物联网课上内容来说明校验码的主要用途。

在数据通信中，校验码可以确保数据完整性、提高通信系统可靠性和防止数据出错、提高数据通信的安全性和可靠性。大部分的校验码都是通过在数据传输中附加校验码（例如奇偶校验码、CRC循环冗余校验码、海明校验码等等），接收方能够有效地检测和识别由于噪声、干扰或硬件故障导致的数据损坏或错误，从而保证数据传输及交付的完整性和正确性。校验码能够及时发现传输中的错误，同时与纠错码相互配合来维护通信系统的稳定运行并降低错误率。这里的纠错码就是一种修复机制，熟知的纠错码包括：线性分组码、卷积码、交织编码等等。

2. CheckSum（校验和）

2.1 主要功能

在CheckSum的main.c文件中主要包含**getCheckSum**函数以及**main**函数。

**main**函数其实就是一个交互式的菜单，让用户选择校验的格式。如果选择普通字符串校验（flag == 1），程序会提示用户输入一个字符串，并调用 getCheckSum 函数来生成校验和。如果选择16进制字符串校验（flag == 2），程序会提示用户输入一个16进制字节串，并调用 getCheckSum 函数来生成校验和。在生成校验和后，程序会打印出结果。这个程序会一直执行下去，直至选择的校验格式不在可选择范围内。

**getCheckSum**函数是本校验程序中的主要功能部分，他主要实现了本程序的校验和的计算过程。函数的参数列表为一个无符号字符指针 string 和一个整数 length，表示要计算校验和的字符串和字符串长度。在函数内部首先计算字符串中所有字符的ASCII码值的总和，然后将这个总和模65536得到一个16位中间结果。接着将这个中间结果与0xFFFF进行异或（也就是按位取反），再加1。最后将这个16位的中间结果的高8位存入校验和字段的第一个字节处，同时将低8位存入校验和字段的第二个字节处。这里可以看到我将原先代码中的checksum数组的定义注释了，后面使用malloc动态分配内存来创建这个数组。最后将指向校验和数组的指针返回给主程序。

下面是**getCheckSum**函数的具体实现：

1. unsigned char \*getCheckSum(const unsigned char \*string, int length) {
2. int sum = 0;  // 字符的ASCII码值的总和
3. int i = 0;
4. for (; i<length; i++) {  // 计算总和部分
5. sum += string[i];
6. }
7. unsigned short result = sum % 65536;  // 对字符总和模65536得到一个16位中间结果 result
8. result = result ^ 0xFFFF;  // 将中间结果与 0xFFFF 进行异或操作
9. result += 1;  // 再将中间结果加 1
10. // unsigned char checkSum[2];
11. auto \*checkSum = (unsigned char \*)malloc(2 \* sizeof(unsigned char));  // 更改校验和定义，防止返回出错
12. if (checkSum == nullptr) {  // 检查内存分配是否成功，失败则执行
13. printf("内存分配失败！\n");
14. exit(1);
15. }
16. checkSum[0] = result >> 8;  // 将16位中间结果的高 8 位存储在检验和的第一个字节
17. checkSum[1] = result;  // 将16位中间结果的低 8 位存储在校验和的第二个字节
18. return checkSum;  // 返回指向校验和数组的指针
19. }

2.2 使用方法

我们运行代码在命令行会出现如下图1界面。在界面中为我们提供了两种选择，一个是普通字符串校验（选择“1”），另一个是16进制字符串校验（选择“2”）。我们可以根据自己的需要选择一种校验格式。

文本, 信件

描述已自动生成

**图1** 运行代码后的用户交互界面

这里我们首先**选择“1”（普通字符串校验）**，因此在这里我们键入“1”并回车，此时就会出现以下图2界面。

文本

描述已自动生成

**图2** 选择校验格式为普通字符串校验

交互界面提示我们输入需要校验的字符串，以回车结束。此时我们根据用户界面交互提示我们输入测试字符串“@~ABCDEFG$#%”并按下回车，完成校验和计算，就会出现以下图3界面。

文本, 信件

描述已自动生成

**图3** 输入测试字符串后（测试数据）

根据图3我们可以看出，这个程序在执行完本次校验和计算后又开始了下一次的校验格式选择，不仅验证了校验和的计算结果，也证明这个程序可以再一次运行的情况下计算多组校验和，直至校验格式选择错误。

我们再来看校验格式**选择为“2”（16进制字符串校验）**时的情形，如下图4，此时交互界面提示我们这是16进制字符串校验，但这里我们需要先输入这个16进制字符串所包含的字节数。

文本

描述已自动生成

**图4** 选择校验格式为16进制字符串校验

下来我们给出相应测试数据，我们使用的测试数据为 [4字节：11 12 FE FF]，由此我们首先输入字节数4，如下图5所示。

文本, 信件

描述已自动生成

**图5** 校验字符串的字节数为4（测试数据）

交互界面提示我们输入需要校验的字节，每个字节之间使用空格分开，以回车结束。此时我们根据用户界面交互提示输入测试的16进制字符串“11 12 FE FF”并按下回车，完成校验和计算，如下图6所示。

文本, 信件

描述已自动生成

**图6** 输入测试字符串后（测试数据）

这实际上是一个无限循环的程序，因此如果我们需要终止程序只能手动结束任务。另一种解决办法就是将选择校验格式部分判断不在可选范围内时退出本程序，即在选择有误时退出程序。

2.3 计算结果

* 普通字符串校验

在前面“使用方法”中我们已经对普通字符串校验做了测试，因此我们这里只给出测试的计算结果截图。如下图7所示，对于普通字符串“@~ABCDEFG$#%”，其校验和为 FC FA。

文本, 信件

描述已自动生成

**图7** 普通字符串校验计算结果1

我在这里还尝试了“校验和函数调用说明.txt”中的一个测试数据，如下图8所示，对于普通字符串“21016042F0100014001400160014”，其校验和为 FA 83，。

文本, 信件

描述已自动生成

**图8** 普通字符串校验计算结果2

* 16进制字符串校验

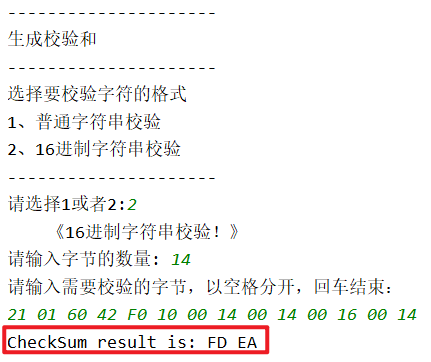
在前面“使用方法”中我们已经对16进制字符串校验做了测试，因此我们这里只给出测试的计算结果截图。如下图9所示，对于16进制字符串“11 12 FE FF”，其校验和为 FD E0。

文本

描述已自动生成

**图9** 16进制字符串校验计算结果1

我在这里还尝试了“校验和函数调用说明.txt”中的一个测试数据，这里我将原始数据按照每两位拆分为一个字节，因此一共有14个字节，按照空格分开，如下图10所示，对于16进制字符串“21 01 60 42 F0 10 00 14 00 14 00 16 00 14”，其校验和为 FD EA。



**图10** 16进制字符串校验计算结果2

3. CRC16（循环冗余校验）

3.1 主要功能

在CRC16的main.c文件中主要包含**InvertUint8**函数、**InvertUint16**函数**、CRC16\_MODBUS**函数以及**main**函数。

**main**函数其实就是一个交互式的菜单，让用户选择校验的格式。如果选择普通字符串校验（flag == 1），程序会提示用户输入一个字符串，并调用CRC16\_MODBUS函数来生成校验和。如果选择16进制字符串校验（flag == 2），程序会提示用户输入一个16进制字节串，并调用CRC16\_MODBUS函数来生成校验和。在生成校验和后，程序会打印出结果。这个程序会一直执行下去，直至选择的校验格式不在可选择范围内。

**InvertUint8**函数的作用是反转一个8位数据的位序。函数内部创建了一个临时变量tmp来存储反转后的结果。遍历源数据srcBuf的每一位，在循环中，如果当前位为1，就将对应的tmp位置设置为1，最后将反转后的结果存储在dBuf中。

下面是**InvertUint8**函数的具体实现：

1. void InvertUint8(unsigned char \*dBuf, const unsigned char \*srcBuf) {
2. int i;
3. unsigned char tmp[4];
4. tmp[0] = 0;
5. for (i = 0; i < 8; i++) {  // 遍历8位数据
6. if (srcBuf[0] & (1 << i))  // 如果srcBuf的当前位为1，则将对应的tmp位设置为1
7. tmp[0] |= 1 << (7 - i);
8. }
9. dBuf[0] = tmp[0];
10. }

**InvertUint16**函数的作用是反转一个16位数据的位序。它也使用一个临时变量tmp来存储反转后的结果。遍历源数据srcBuf的每一位，在循环中，如果某一位为1，就将对应的tmp位置设置为1，最后将反转后的结果存储在dBuf中。

下面是**InvertUint16**函数的具体实现：

1. void InvertUint16(unsigned short \*dBuf, const unsigned short \*srcBuf) {
2. int i;
3. unsigned short tmp[4];
4. tmp[0] = 0;
5. for (i = 0; i < 16; i++)  {  // 遍历16位数据
6. if (srcBuf[0] & (1 << i))  // 如果srcBuf的当前位为1，则将对应的tmp位设置为1
7. tmp[0] |= 1 << (15 - i);
8. }
9. dBuf[0] = tmp[0];
10. }

最后是本程序的核心部分**CRC16\_MODBUS**函数，CRC16\_MODBUS函数实现了CRC16\_MODBUS校验算法的操作过程。首先它初始化了两个16位寄存器：wCRCin初始为0xFFFF，wCPoly初始为0x8005，代表CRC的多项式。然后遍历数据缓冲区中的每个字节，并对每个字节进行8位的位序反转。反转完成后，函数将反转后的字节左移8位，并与wCRCin进行异或运算，这一步是为了模拟多项式除法来更新CRC的值。每次异或后，都需要去检查wCRCin的最高位，如果为1，则进行进一步的异或运算。遍历完所有数据字节后，函数执行16位的位序反转，这是为了确保CRC符合MODBUS协议的要求。最后，像CheckSum部分一样，将这个16位的中间结果的高8位存入校验和字段的第一个字节处，同时将低8位存入校验和字段的第二个字节处。这里我也将原先代码中的result数组的定义注释了，后面使用malloc动态分配内存来创建这个数组。最后将指向校验和数组的指针返回给主程序。

下面是**CRC16\_MODBUS**函数的具体实现：

1. unsigned char\* CRC16\_MODBUS(unsigned char \*puchMsg, unsigned int usDataLen) {
2. unsigned short wCRCin = 0xFFFF;  // 初始寄存器使用全1
3. unsigned short wCPoly = 0x8005;  // 多项式
4. unsigned char wChar = 0;
5. while (usDataLen--) {  // 遍历数据长度
6. wChar = \*(puchMsg++);
7. InvertUint8(&wChar, &wChar);  // 反转8位数据的位序
8. wCRCin ^= (wChar << 8);  // 左移八位扩充至十六位
9. int i = 0;
10. for (; i < 8; i++) {  // 循环8次，对CRC进行计算
11. if (wCRCin & 0x8000)
12. wCRCin = (wCRCin << 1) ^ wCPoly;  // 左移一位并进行异或运算
13. else
14. wCRCin = wCRCin << 1;  // 左移一位
15. }
16. }
17. InvertUint16(&wCRCin, &wCRCin);  // 反转16位数据的位序
18. // unsigned char result[2];
19. auto \*result = (unsigned char \*)malloc(2 \* sizeof(unsigned char));  // 分配内存
20. if (result == nullptr) {  // 内存分配失败提示
21. printf("内存分配失败！\n");
22. exit(1);
23. }
24. result[1] = wCRCin >> 8;  // 高8位
25. result[0] = wCRCin;       // 低8位
26. return result;
27. }

3.2 使用方法

我们运行代码在命令行会出现如下图11界面。在界面中为我们提供了两种选择，一个是普通字符串校验（选择“1”），另一个是16进制字符串校验（选择“2”）。我们可以根据自己的需要选择一种校验格式。

文本, 信件

描述已自动生成

**图11** 运行代码后的用户交互界面

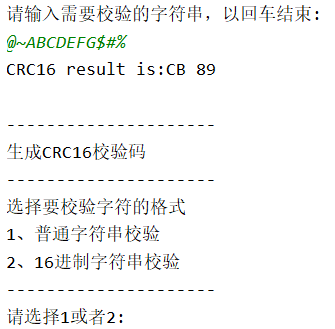
这里我们首先**选择“1”（普通字符串校验）**，因此在这里我们键入“1”并回车，此时就会出现以下图12界面。

文本

描述已自动生成

**图12** 选择校验格式为普通字符串校验

交互界面提示我们输入需要校验的字符串，以回车结束。此时我们根据用户界面交互提示我们输入测试字符串“@~ABCDEFG$#%”并按下回车，完成校验和计算，就会出现以下图13界面。



**图13** 输入测试字符串后（测试数据）

根据图13我们可以看出，这个程序在执行完本次校验和计算后又开始了下一次的校验格式选择，不仅验证了校验和的计算结果，也证明这个程序可以再一次运行的情况下计算多组校验和，直至校验格式选择错误。

我们再来看校验格式**选择为“2”（16进制字符串校验）**时的情形，如下图14，此时交互界面提示我们这是16进制字符串校验，但这里我们需要先输入这个16进制字符串所包含的字节数。

文本

描述已自动生成

**图14** 选择校验格式为16进制字符串校验

下来我们给出相应测试数据，我们使用的测试数据为 [4字节：11 12 FE FF]，由此我们首先输入字节数4，如下图15所示。

文本, 信件

描述已自动生成

**图15** 校验字符串的字节数为4（测试数据）

交互界面提示我们输入需要校验的字节，每个字节之间使用空格分开，以回车结束。此时我们根据用户界面交互提示输入测试的16进制字符串“11 12 FE FF”并按下回车，完成校验和计算，如下图16所示。

文本

描述已自动生成

**图16** 输入测试字符串后（测试数据）

这实际上是一个无限循环的程序，因此如果我们需要终止程序只能手动结束任务。另一种解决办法就是将选择校验格式部分判断不在可选范围内时退出本程序，即在选择有误时退出程序。

3.3 计算结果

* 普通字符串校验

在前面“使用方法”中我们已经对普通字符串校验做了测试，因此我们这里只给出测试的计算结果截图。如下图17所示，对于普通字符串“@~ABCDEFG$#%”，其CRC16校验码为 CB 89。

文本, 信件

描述已自动生成

**图17** 普通字符串校验计算结果1

我在这里还尝试了“校验和函数调用说明.txt”中的一个测试数据，如下图18所示。

文本, 信件

描述已自动生成

**图18** 普通字符串校验计算结果2

* 16进制字符串校验

在前面“使用方法”中我们已经对16进制字符串校验做了测试，因此我们这里只给出测试的计算结果截图。如下图19所示，对于16进制字符串“11 12 FE FF”，其CRC16校验码为 A5 3D。

文本, 表格

中度可信度描述已自动生成

**图19** 16进制字符串校验计算结果1

我在这里还尝试了“校验和函数调用说明.txt”中的一个测试数据，这里我将原始数据按照每两位拆分为一个字节，因此一共有14个字节，按照空格分开，如下图20所示，对于16进制字符串“21 01 60 42 F0 10 00 14 00 14 00 16 00 14”，其CRC16校验码为 1B 92。

图形用户界面, 文本

描述已自动生成

**图20** 16进制字符串校验计算结果2

4. 实验遇到的主要问题和解决方法

4.1 问题及解决方法1

* **问题**

我的实验使用的IDE是JetBrains公司的Clion，在Clion中运行本次实验的初始版本代码在输入完成需要进行计算的字符串之后会直接退出程序，而不会像上述内容中进行计算并输出结果然后转入下一次输入，如下图21所示。

文本

描述已自动生成

**图21** 问题1（Clion）

随后我也在其他IDE上进行了测试，选用的是Visual Studio 2022，在这个里面可以输出第一次的计算结果，但仍然会出现错误，也不会像上面的内容一样，在计算一次后进入下一次等待输入，如下图22所示。

图形用户界面, 应用程序

描述已自动生成

**图22** 问题1（Visual Studio 2022）

* **解决方法**

最后经过调试代码发现在CheckSum的getCheckSum函数中，在定义数组checksum时这里使用的是“unsigned char result[2];”来声明数组。但是这种声明方式在函数结束时，该数组会被销毁，无法在函数外部访问，导致了上面的错误。因此我们需要一个在函数结束后仍能保持结果的值，这里我使用malloc动态分配内存使得在函数结束后仍然存在，可以在函数外部使用这段内存，从而避免了在函数结束时销毁结果，使得调用该函数的代码可以获取和使用计算出的CRC校验码。如下图23所示。

图片包含 文本

描述已自动生成

**图23** 问题1解决方法（CheckSum）

在CRC16的CRC16\_MODBUS函数中也是同理进行修改，如下图24所示。

图形用户界面, 应用程序, Word

描述已自动生成

**图24** 问题1解决方法（CRC16\_MODBUS）

4.2 问题及解决方法2

* **问题**

实验过程中在进行Debug时，我发现在Debug模式下程序中的printf函数的输出内容不会出现在调试器中，没有任何回显，如下图25所示。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**图25** 问题2

* **解决方法**

最后经过上网查询发现只需要在main函数中添加一个图26红框部分的代码即可，这其实就是用于设置标准输出流stdout的缓冲模式。第二个参数设置为nullptr，实际上是禁用了stdout的缓冲，使得printf函数的输出立即刷新到控制台，而不是等到缓冲区满或遇到换行符时才刷新。这样可以确保在调试或交互式的场景下，可以即时看到输出，而不需要等待缓冲区刷新，如下图26所示。

图片包含 图形用户界面

描述已自动生成

**图26** 问题2 解决方法

四、实验总结

本次实验的题目为校验码的生成，主要包括校验和（CheckSum）和CRC16（循环冗余校验）。通过调试运行通信协议中常用校验码的计算程序，学会典型校验码生成程序的使用，同时也掌握了调用校验码生成程序进行校验码生成的方法，为后续复杂通信协议的运用和传感数据的校验打下基础。

通过本次实验，我对校验码在确保数据传输准确性和系统稳定性中的作用有了更深刻的理解。校验码通过在原始数据中添加特定的信息，提供了一种检测和校验数据完整性的手段，这样可以防止数据在传输或存储过程中的更改。不仅增强了通信系统的可靠性，还能作为一种安全措施，提高数据通信的安全性和可靠性。有了校验码，我们可以及时发现并定位数据中的错误，有助于对数据的修复和恢复。

在实验中主要用到两种主要的校验码：校验和（Checksum）与循环冗余校验（CRC16）。校验和是一种简单的错误检测码，适用于错误率较低的场合；而CRC16则是一种更为复杂和强大的错误检测方法，适用于对数据传输质量要求更高的场景。通过编写和调试相应的C语言程序，我不仅掌握了这两种校验码的生成方法，还提高了编程和问题解决技能。

在实验中遇到的问题也让我学习到了一些实用的技巧，比如使用动态内存分配来保持函数计算结果，设置标准输出流的缓冲模式来确保调试过程中的输出可见性。这些技巧对于我未来的编程实践和问题解决都非常有帮助。

尽管校验码会消耗额外的带宽与存储空间，但它可以确保数据完整，介于此这种资源消耗是可以接受的。校验码还可以与纠错码结合使用，以进一步提高数据传输的准确性。校验码只是物联网系统中的一小部分内容，但也为后续课程学习打下一定基础。