苏州大学实验报告

院、系	计算机学院	年级专业	计算机科学与技术	姓名	张昊	学号	1927405160
课程名称	微型计算机技术					成绩	
指导教师	姚望舒	同组实验者	无		实验日期	2022 年 6 月 10 日	

实验名称：实验三：存储器实验

一. 实验目的

（1）掌握 Flash 存储器在线编程的基本概念。

（2）掌握 Flash 存储器的在线编程擦除和写入编程的基本方法。

（3）进一步深入理解 MCU 和 C#串口通信的编程方法。

二. 实验准备

（1）硬件部分。PC 机或笔记本电脑一台、开发套件一套。

（2）软件部分。根据电子资源“..\02-Doc”文件夹下的电子版快速指南，下载合适的电子资源。

（3）软件环境。按照电子版快速指南中“安装软件开发环境”一节，进行有关软件工具的安装。

三. 实验参考样例

教材“Exam6_1”工程为参考样例

四. 实验过程或要求

（1）验证性实验

① 下载开发环境 AHL-GEC-IDE。根据电子资源下“..\05-Tool\AHL-GEC-IDE 下载地址.txt”文件指引，下载由苏州大学-Arm 嵌入式与物联网技术培训中心（简称 SD-Arm）开发的金葫芦集成开发环境（AHL-GEC-IDE）到“..\05-Tool”文件夹。该集成开发环境兼容一些常规开发环境工程格式。

② 建立自己的工作文件夹。按照“分门别类，各有归处”之原则，建立自己的工作文件夹。并考虑随后内容安排，建立其下级子文件夹。

③ 拷贝模板工程并重命名。所有工程可通过拷贝模板工程建立。例如，“\04-Soft\ Exam4_1”工程到自己的工作文件夹，可以改为自己确定的工程名，建议尾端增加日期字样，避免混乱。

④ 导入工程。在假设您已经下载 AHL-GEC-IDE，并放入“..\05-Tool”文件夹，且按安装电子档快速指南正确安装了有关工具，则可以开始运行“..\05-Tool\AHL-GEC-IDE\AHL-GEC-IDE.exe”文件，这一步打开了集成开发环境 AHL-GEC-IDE。接着单击“ ”→“ ”→导入你拷贝到自己文件夹并重新命名的工程。导入工程后，左侧为工程树形目录，右边为文件内容编辑区，初始显示 main.s 文件的内容。

⑤ 编译工程。在打开工程，并显示文件内容前提下，可编译工程。单击“ ”→“ ”，则开始编译。

⑥ 下载并运行。

步骤一，硬件连接。用 TTL-USB 线（Micro 口）连接 GEC 底板上的“MicroUSB”串口与电脑的 USB 口。

步骤二，软件连接。单击“ ”→“ ”，将进入更新窗体界面。点击“ ”查找到目标 GEC，则提示“成功连接……”。

步骤三，下载机器码。点击“ ”按钮导入被编译工程目录下 Debug 中的.hex 文件（看准生成时间，确认是自己现在编译的程序），然后单击“ ”按钮，等待程序自动更新完成。

此时程序自动运行了。若遇到问题可参阅开发套件纸质版导引“常见错误及解决方法”一节，也可参阅电子资源“..\02-Doc”文件夹中的快速指南对应内容进行解决。

⑦ 观察运行结果与程序的对应。

第一个程序运行结果（PC 机界面显示情况）见图 4-7。为了表明程序已经开始运行了，在每个样例程序进入主循环之前，使用 printf 语句输出一段话，程序写入后立即执行，就会显示在开发环境下载界面的中的右下角文本框中，提示程序的基本功能。

利用 printf 语句将程序运行的结果直接输出到 PC 机屏幕上，使得嵌入式软件开发的输出调试

变得十分便利，调试嵌入式软件与调试 PC 机软件几乎一样方便，改变了传统交叉调试模式。实验步骤和结果

（2）设计性实验

复制样例程序“Exam6_1”工程，利用该程序框架实现：在集成开发环境 AHL-GEC-IDE 中菜单栏中单击“ 工具 ”→“ 存储器操作 ”或者通过 “..\06-Other\ C# Flash 测试程序” 发送擦除、写入、读取命令及其参数，参数能够设置扇区号（0-63）、写入/读取扇区内部偏移地址（要求为 0,4，8,12，.....）；写入/读取字节数目（要求为 4，8,12,.....）和数据。

请在实验报告中给出 MCU 端程序 main.s 和 isr.s 流程图及程序语句。

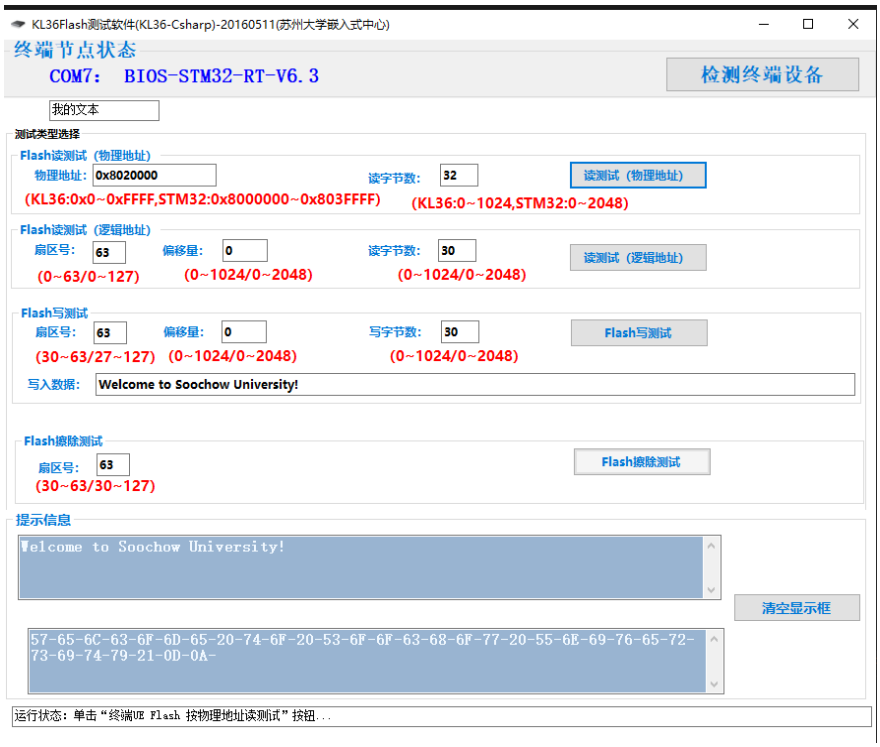
四、实验结果

（1）用适当文字、图表描述实验过程。

验证性实验：（Exam6_1）

设计性实验：（使用 C# Flash 测试程序）

读取 0x8020000 地址（64 扇区）长度为 32 字节的数据：



读取 64 扇区，偏移量 4 的 32 字节数据：

擦除 64 扇区的数据：

读取 64 扇区，偏移量 0 的 32 字节数据：

向 64 扇区，偏移量 0 的位置写入 10 字节的数据：0123456789

读取 64 扇区，偏移量 0 的 10 字节数据：

（2）完整给出 GEC 方串口通信程序的执行流程、Flash 读取写入的逻辑操作以及 PC 方的 Flash 测试程序的执行流程。

```
//设计性实验中对 flash 操作的代码片段（isr.s）
USART2_IRQHandler:
//屏蔽中断，并且保存现场
      cpsid  i          //关可屏蔽中断
      push {r0-r7,lr}      //r0-r7,lr进栈保护
     //uint_8 flag
     sub sp,#4          //通过移动sp指针获取地址
     mov r7,sp          //将获取到的地址赋给r7
//接收字节
      mov r1,r7          //r1=r7 作为接收一个字节的地址
      mov r0,#UARTA        //r0指明串口号
      bl  uart_re1       //调用接收一个字节子函数，接收到的存到r0
      ldr r1,=0x20003000  //接收到数据部分的存入地址gcRecvBuf
      bl  emuart_frame  //调用帧解析函数,r0=数据部分长度
      cmp r0,#0          //若成功解析帧格式，跳转协议解析
      bne UARTA_IRQHandler_success_revc
      //否则直接退出
      bl UARTA_IRQHandler_finish    //break
```

```
UARTA_IRQHandler_success_revc:
    //握手协议: 11, 'a', 'u', 'a', 'r', 't', '?'
    ldr r0,=0x20003000
    ldrb r0,[r0]    //读取数据部分第一位->r0
    cmp r0,#11      //判断握手协议
    bne UARTA_IRQHandler_next_r
    ldr r0,=handshake_check_str//string1
    ldr r1,=0x20003000
    add r1,#1       //string2
    mov r2,#6       //字符串长度
    bl cmp_string
    cmp r0,#0
    bne UARTA_IRQHandler_next_r//不是握手协议
    //与上位机握手，确立通信关系
    mov r0,#UARTA
    ldr r1,=handshake_send_str
    bl  uart_send_string
    bl UARTA_IRQHandler_finish    //break
UARTA_IRQHandler_next_r:
    //读逻辑地址: "r"+扇区号1B+偏移量(2B)+读字节数1B
    ldr r0,=0x20003000
    ldrb r0,[r0]    //读取数据部分第一位->r0
    cmp r0,#'r'     //读取（按逻辑地址）
    bne UARTA_IRQHandler_next_a
    //flash_read_logic(flash_ContentDetail,gcRecvBuf[1],
    //                 (gcRecvBuf[2]<<8)|gcRecvBuf[3],gcRecvBuf[4]);
    ldr r0,=flash_ContentDetail
    ldr r1,=0x20003000
    add r1,#1
    ldrb r1,[r1]    //r1=gcRecvBuf[1]
    ldr r2,=0x20003000
    add r2,#2
    ldrb r2,[r2]
    lsl r2,r2,#8    //r2=gcRecvBuf[2]<<8
    ldr r3,=0x20003000
    add r3,#3
    ldrb r3,[r3]    //gcRecvBuf[3]
    orr r2,r2,r3    //r2=(gcRecvBuf[2]<<8)|gcRecvBuf[3]
    ldr r3,=0x20003000
    add r3,#4
    ldrb r3,[r3]    //r3=gcRecvBuf[4]
    bl flash_read_logic
    //判空
    ldr r0,=flash_ContentDetail
    ldr r1,=0x20003000
```

```
      add r1,#4
      ldrb r1,[r1]     //r1=gcRecvBuf[4]
      bl data_isempty
      cmp r0,#0
      beq UARTA_IRQHandler_next_r_empty  //为空跳转
      //不为空，发送数据
      mov r0,#UARTA
      ldr r1,=0x20003000
      add r1,#4
      ldrb r1,[r1]     //r1=gcRecvBuf[4]
      ldr r2,=flash_ContentDetail
      bl uart_sendN
      bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_r_empty:
      //为空，发送空信息
      mov r0,#UARTA
      ldr r1,=flash_str1
      bl uart_send_string
      bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_a:
      //读物理地址："a"+地址4B+读字节数1B
      ldr r0,=0x20003000
      ldrb r0,[r0]     //读取数据部分第一位->r0
      cmp r0,#'a'      //读取（按物理地址）
      bne UARTA_IRQHandler_next_w
   //flash_read_physical(flashRead,
   // (gcRecvBuf[1]<<24)|(gcRecvBuf[2]<<16)|(gcRecvBuf[3]<<8)|gcRecvBuf[4],
   // gcRecvBuf[5]);
      ldr r0,=flash_ContentDetail
      ldr r1,=0x20003000
      add r1,#1
      ldrb r1,[r1]
      lsl r1,r1,#24       //r1=gcRecvBuf[1]<<24
      ldr r2,=0x20003000
      add r2,#2
      ldrb r2,[r2]
      lsl r2,r2,#16       //gcRecvBuf[2]<<16
      orr r1,r1,r2     //r1=(gcRecvBuf[1]<<24)|(gcRecvBuf[2]<<16)
      ldr r2,=0x20003000
      add r2,#3
      ldrb r2,[r2]
      lsl r2,r2,#8     //gcRecvBuf[3]<<8
      orr r1,r1,r2
    //r1=(gcRecvBuf[1]<<24)|(gcRecvBuf[2]<<16)|(gcRecvBuf[3]<<8)
      ldr r2,=0x20003000
```

```
      add r2,#4
      ldrb r2,[r2]     //gcRecvBuf[4]
      orr r1,r1,r2
  //r1=(gcRecvBuf[1]<<24)|(gcRecvBuf[2]<<16)|(gcRecvBuf[3]<<8)|gcRecvBuf[4]
      ldr r2,=0x20003000
      add r2,#5
      ldrb r2,[r2]     //r2=gcRecvBuf[5]
      bl flash_read_physical
      //判空
      ldr r0,=flash_ContentDetail
      ldr r1,=0x20003000
      add r1,#5
      ldrb r1,[r1]     //r1=gcRecvBuf[5]
      bl data_isempty
      cmp r0,#0
      beq UARTA_IRQHandler_next_a_empty  //为空跳转
      //不为空，发送数据
      mov r0,#UARTA
      ldr r1,=0x20003000
      add r1,#5
      ldrb r1,[r1]     //r1=gcRecvBuf[5]
      ldr r2,=flash_ContentDetail
      bl uart_sendN
      bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_a_empty:
      //为空，发送空信息
      mov r0,#UARTA
      ldr r1,=flash_str1
      bl uart_send_string
      bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_w:
      //写入："w"+扇区号1B+偏移量(2B)+写入字节数1B+写入数据nB
      ldr r0,=0x20003000
      ldrb r0,[r0]     //读取数据部分第一位->r0
      cmp r0,#'w'      //写入
      bne UARTA_IRQHandler_next_e
      //写入前先擦除：flash_erase(gcRecvBuf[1]);
      ldr r0,=0x20003000
      add r0,#1
      ldrb r0,[r0]     //r0=gcRecvBuf[1]
      bl flash_erase
      cmp r0,#0
      bne UARTA_IRQHandler_next_w_failed  //不为0,跳转失败
      //flash_write(gcRecvBuf[1], (gcRecvBuf[2]<<8)|gcRecvBuf[3],
      //            gcRecvBuf[4], &gcRecvBuf[5]);
```

```
        ldr r0,=0x20003000
        add r0,#1
        ldrb r0,[r0]     //r0=gcRecvBuf[1]
        ldr r1,=0x20003000
        add r1,#2
        ldrb r1,[r1]
        lsl r1,r1,#8     //r1=gcRecvBuf[2]<<8
        ldr r2,=0x20003000
        add r2,#3
        ldrb r2,[r2]     //gcRecvBuf[3]
        orr r1,r1,r2     //r1=(gcRecvBuf[2]<<8)|gcRecvBuf[3]
        ldr r2,=0x20003000
        add r2,#4
        ldrb r2,[r2]     //r2=gcRecvBuf[4]
        ldr r3,=0x20003000
        add r3,#5        //r3=&gcRecvBuf[5]
        bl flash_write
        cmp r0,#0
        bne UARTA_IRQHandler_next_w_failed  //不为0,跳转失败
        //为0，写入成功
        mov r0,#UARTA
        ldr r1,=flash_str5
        bl uart_send_string
        bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_w_failed:
        //写入失败
        mov r0,#UARTA
        ldr r1,=flash_str6
        bl uart_send_string
        bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_e:
        //擦除："e"+扇区号1B
        ldr r0,=0x20003000
        ldrb r0,[r0]     //读取数据部分第一位->r0
        cmp r0,#'e'      //擦除
        bne UARTA_IRQHandler_next_default
        //flash_erase(gcRecvBuf[1]);
        ldr r0,=0x20003000
        add r0,#1
        ldrb r0,[r0]     //r0=gcRecvBuf[1]
        bl flash_erase
        cmp r0,#0
        bne UARTA_IRQHandler_next_e_failed  //不为0,跳转失败
        //为0，擦除成功
        mov r0,#UARTA
```

```
      ldr r1,=flash_str3
      bl uart_send_string
      bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_e_failed:
      //擦除失败
      mov r0,#UARTA
      ldr r1,=flash_str4
      bl uart_send_string
      bl UARTA_IRQHandler_finish  //break
UARTA_IRQHandler_next_default:  //default
      ldr r1,=unknown_info_str
      mov r0,#UARTA       //r0指明串口号
      bl  uart_send_string
//解除屏蔽，并且恢复现场
UARTA_IRQHandler_finish:
      cpsie   i         //解除屏蔽中断
      add r7,#4         //还原r7
      mov sp,r7         //还原sp
      pop {r0-r7,pc}    //r0-r7,pc出栈，还原r7的值；pc←lr

//=========================内部函数=====================================
//======================================================================
//函数名称：cmp_string
//函数参数：r0-string1，r1-string2，r2-字符串长度
//函数返回：0-相同，1-不同
//函数功能：比较两字符串是否相同
//======================================================================
cmp_string:
    push {r1-r7,lr}
    mov r3,#0  //计数变量
    mov r7,#0  //result, 0=eq, 1=neq
cmp_string_loop:
    cmp r3,r2  //若r3>=r2，结束循环
    bge cmp_string_exit
    ldrb r4,[r0,r3]    //r4=r0[r3]
    ldrb r5,[r1,r3]    //r4=r1[r3]
    add r3,#1  //r3++
    cmp r4,r5  //若相等，继续循环
    beq cmp_string_loop
    mov r7,#1  //不相等，r6=1，结束循环
cmp_string_exit:
    mov r0,r7
    pop {r1-r7,pc}

//======================================================================
```

```
//函数名称：data_isempty
//函数参数：r0-data，r1-长度
//函数返回：0-empty，1-not
//函数功能：检查读出的data数组是否为空
//=========================================================================
data_isempty:
    push {r1-r7,lr}
    mov r2,#0  //计数变量
    mov r3,#0  //0-empty，1-not
data_isempty_loop:
    cmp r2,r1  //若r2>=r1，结束循环
    bge data_isempty_exit
    ldrb r4,[r0,r2] //r4=r0[r2]
    add r2,#1      //r2++
    cmp r4,#0xff  //若相等，继续循环
    beq data_isempty_loop
    //若不等，则有数据，不为空，结束循环
    mov r3,#1
data_isempty_exit:
    mov r0,r3
    pop {r1-r7,pc}
```

程序说明：

1、参考 C# Flash 测试程序工程文件，PC 机和 MCU 遵循如下协议：

PC 向串口发送的数据帧格式为：

帧头（2B）		数据长度（2B）	数据	帧校验（2B）	帧尾（2B）	
0xa5	0x06	Length	……	CRC	0xb6	0x07

MCU 接收到数据帧后直接通过串口发回数据（不封装成帧）。

协议的命令如下：

握手命令		
PC 机发送	Length=7	数据={11，'a'，'u'，'a'，'r'，'t'，'?'}
MCU 响应	"I am an uart"	
Flash 读取命令（按逻辑地址）		
PC 机发送	Length=5	数据="r"+扇区号 1B+偏移量 2B+读字节数 1B
MCU 响应	读取的数据，若为空则提示	
Flash 读取命令（按物理地址）		
PC 机发送	Length=6	数据="a"+地址 4B+读字节数 1B
MCU 响应	读取的数据，若为空则提示	
Flash 擦除命令		
PC 机发送	Length=2	数据="e"+扇区号 1B
MCU 响应	返回提示信息	
Flash 写入命令		
PC 机发送	Length=5+n	数据="w"+扇区号 1B+偏移量 2B+读字节数 1B+写入数据 nB
MCU 响应	返回提示信息	

2、在 MCU 端的下位机程序，使用 05_UserBoard\emuart.h 中定义的 emuart_frame 函数进行数据帧的解析；PC 机的 C#程序使用 EMUART 类进行数据帧的封装。

3、问题指正：在使用逻辑地址写入 Flash 时，会出现数据丢失的问题，如下图所示：

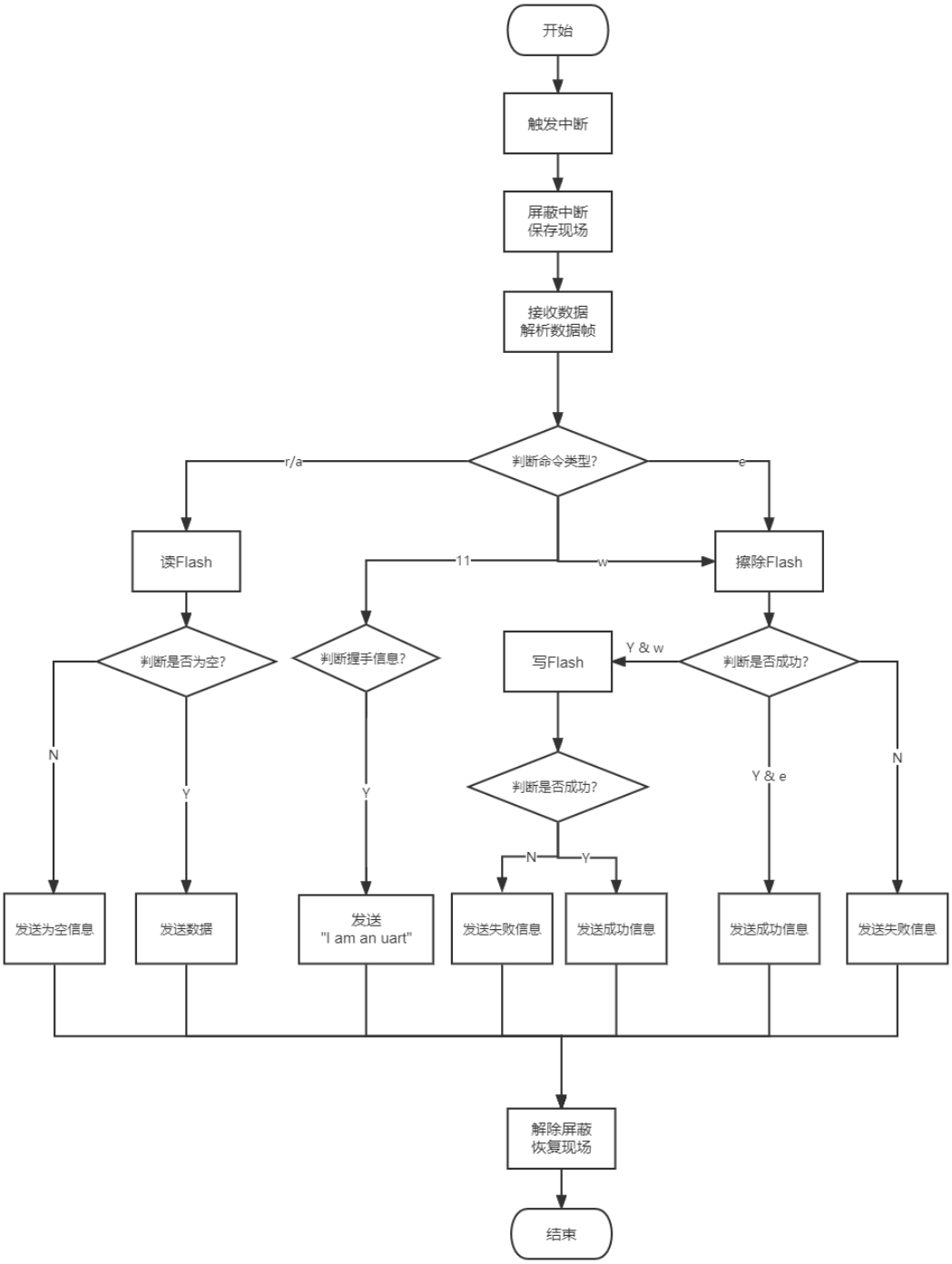
经查，此 bug 主要问题在于 03_MCU\MCU_drivers\flash.c 文件第 84 行对 flash_write 函数的实现中，双字的高、低位字（uint32_t）误用了半字（uint16_t）来存储：

```
uint8_t flash_write(uint16_t sect,uint16_t offset,uint16_t N,uint8_t *buf)
{
    uint32_t addr;     //双字写入绝对地址
    uint32_t i;     //计数，每8字节加1
    uint16_t temp_l_16,temp_h_16;
    uint16_t temp_l_32,temp_h_32;     //temp_l双字中低位字，temp_h双字中高位字
//  uint_64 temp;     //每次写入的双字
```

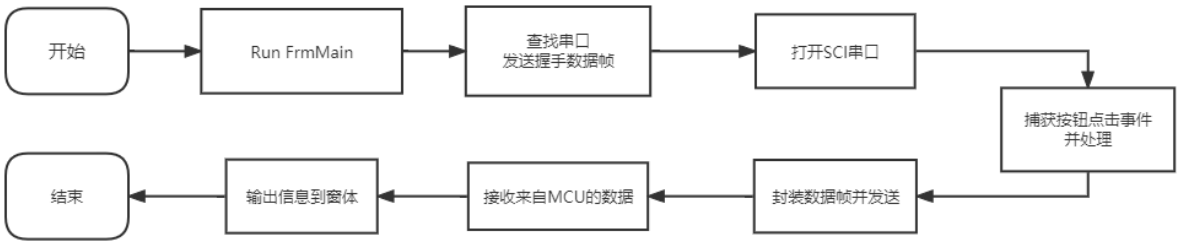
将其作如下修正即可解决（正确的实验结果见上方描述）。

```
uint32_t temp_l_32, temp_h_32;
```

MCU 串口通信与 Flash 操作的执行流程：



PC 机 C#程序执行流程：



教务处制

五. 实践性问答题

（1）Flash 在线编程的过程中有哪些注意点？

Flash 在线编程是通过运行 Flash 内部程序对 Flash 其他区域进行擦除与写入的，需要注意擦除、写入、读取命令及其参数的设置，这样才能正确地对 Flash 存储区进行操作。

Flash 的读写与一般 RAM 的读写不同，Flash 需要专门的编程过程，其基本操作有两种：擦除和写入。擦除是将存储单元的内容由二进制的 0 变成 1，而写入是将存储单元的某些位由二进制的 1 变成 0。Flash 在线编程的写入操作是以字为单位进行的。

（2）当用 Flash 区存储一些需要变动的参数时，如何保证其他数据不变？

可使用扇区保护函数 flash_protect()保护扇区。

（3）如何获取当前程序占用 Flash 的大小？

可查看编译产生的.map 文件：

Memory Configuration

Name	Origin	Length	Attributes
INTVEC	0x0800d000	0x00000800	xr
FLASH	0x0800d800	0x00032000	xr
RAM	0x20004000	0x0000c000	xrw
default	0x00000000	0xffffffff	