# 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 | 柯骅 | 学号 | 2027405033 |
|---|---|---|---|---|---|---|---|
| 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 | 2023.05.30 | | |

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。

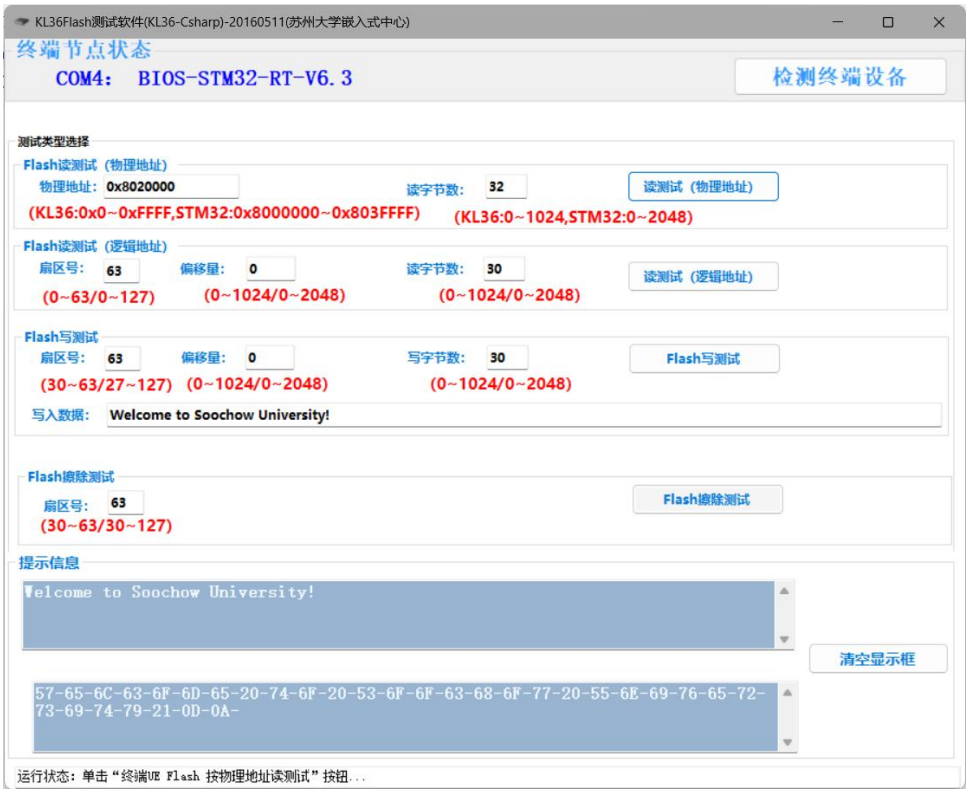
## 四、实验结果

**（1）用适当文字、图表描述实验过程。**

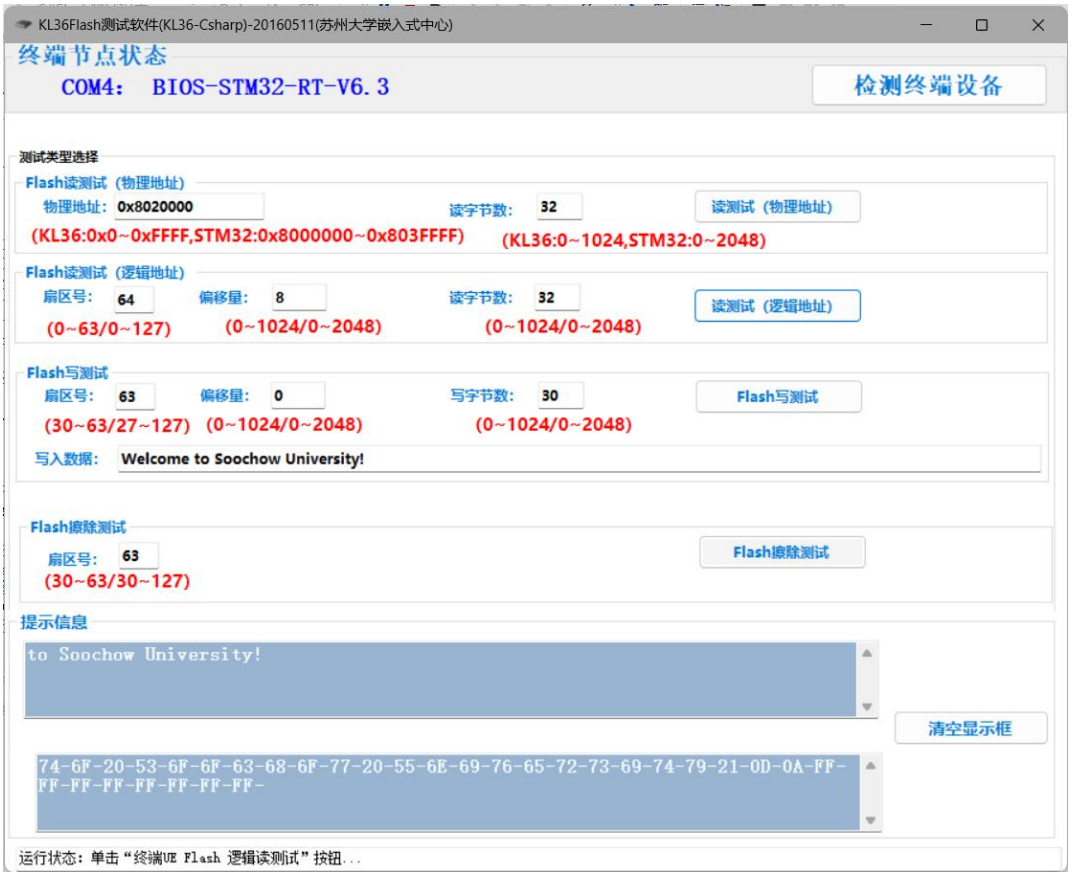
验证性实验：


设计性实验：

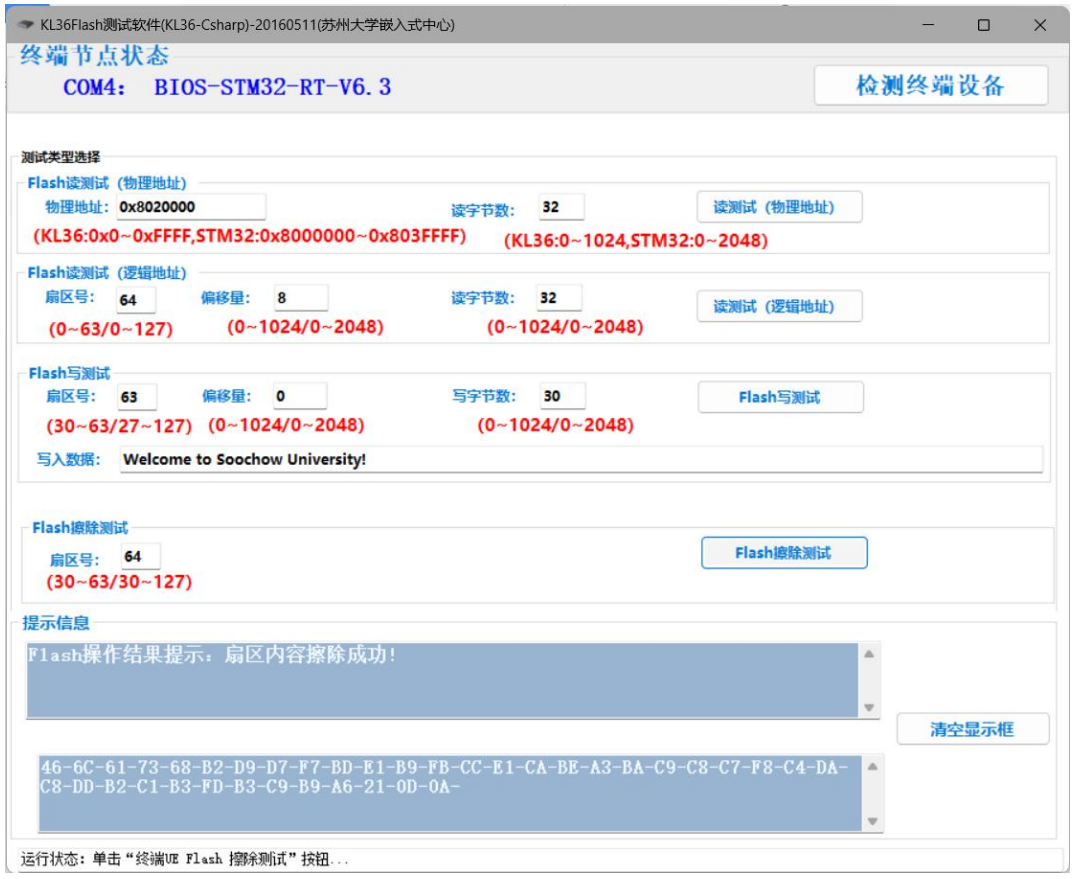
读取 0x8020000 地址（64 扇区）长度为 32 字节的数据：



读取 64 扇区，偏移量 8 的 32 字节数据：

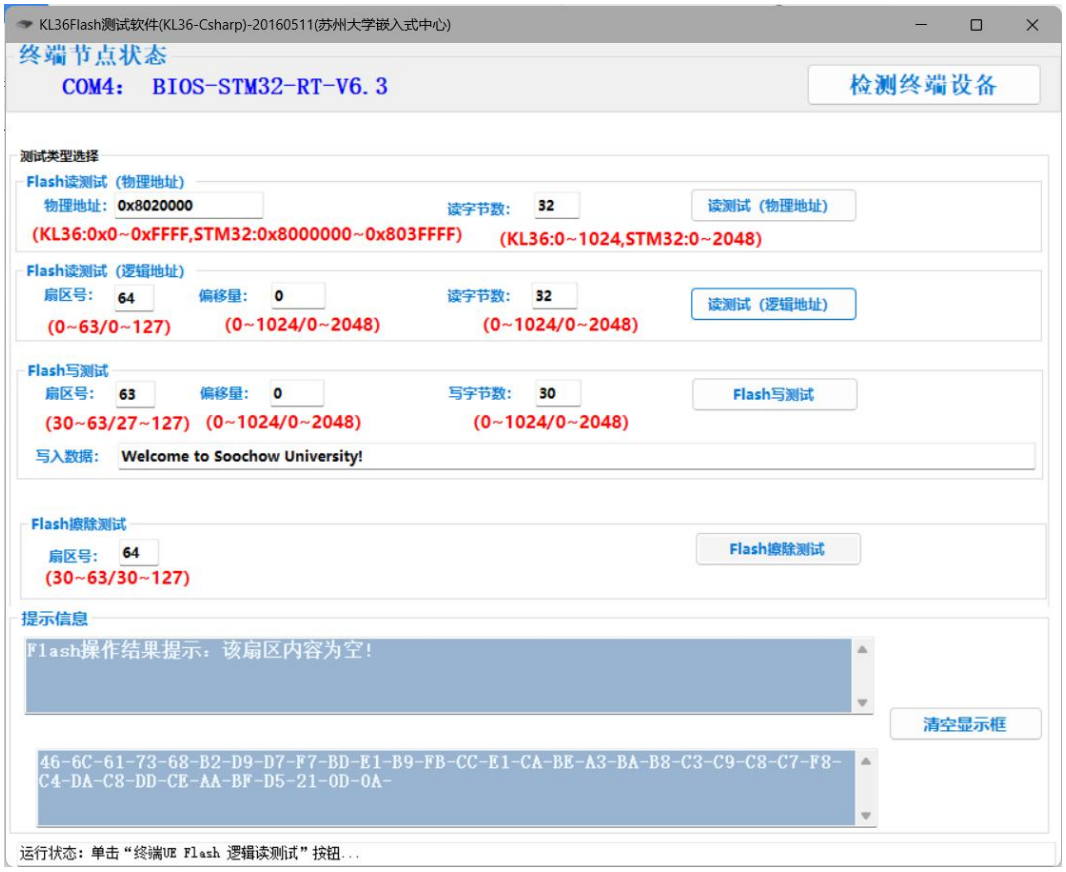



擦除 64 扇区的数据：

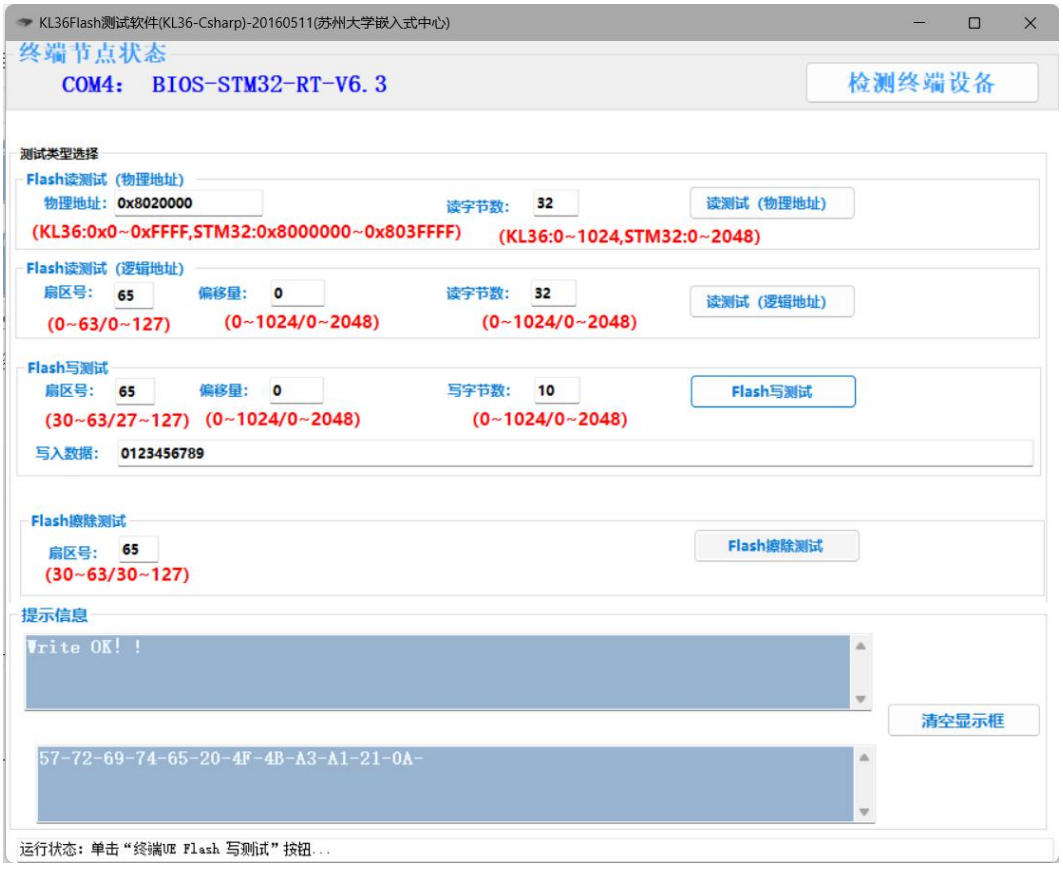



教务处制

读取 64 扇区，偏移量 0 的 32 字节数据：




向 65 扇区，偏移量 0 的位置写入 10 字节的数据：0123456789




教务处制

读取 65 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

```
//=========================================================================
//程序名称：UARTA_Handler（UARTA 接收中断处理程序）
//触发条件：UART_User 串口收到一个字节触发
//程序功能：收到一个字节，直接返回该字节
//=========================================================================
USART2_IRQHandler:
        cpsid  i                //屏蔽中断
        push {r0-r7,lr}         //保留现场
        sub sp,#4               //使 sp 指针偏移获取地址
        mov r7,sp
//接收一个字节
//（1）接收一个字节
        mov r1,r7
        mov r0,#UARTA           //r0->串口号
        bl  uart_re1            //调用接收一个字节的函数
        ldr r1,=0x20003000  //gcRecvBuf：保存接收到的数据
        bl  emuart_frame    //r0=数据部分长度，调用帧解析函数
        cmp r0,#0               //若成功解析帧格式，跳转协议解析
        bne UARTA_IRQHandler_success
```

```
        bl UARTA_IRQHandler_finish    //若没有成功接收数据，直接退出
//（2）协议分析
UARTA_IRQHandler_success:
        //判断是否为握手协议：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
        ldr r1,=0x20003000
        add r1,#1
        mov r2,#6         //length
        bl str_equal
        cmp r0,#0
        //接收到的数据不是握手协议
        bne UARTA_IRQHandler_next_r
        //接收到的数据是握手协议，则握手
        mov r0,#UARTA
        ldr r1,=handshake_send_str
        bl  uart_send_string
        bl UARTA_IRQHandler_finish    //处理完成，跳转完成函数
//不是握手协议
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
                                  //  对  应  c  程  序  中  :
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]
        ldr r2,=0x20003000
        add r2,#2
        ldrb r2,[r2]
        lsl r2,r2,#8
        ldr r3,=0x20003000
        add r3,#3
        ldrb r3,[r3]
        orr r2,r2,r3
        ldr r3,=0x20003000
        add r3,#4
        ldrb r3,[r3]
```

教务处制

```
64.         bl flash_read_logic
65.         //判空
66.         ldr r0,=flash_ContentDetail
67.         ldr r1,=0x20003000
68.         add r1,#4
69.         ldrb r1,[r1]
70.         bl is_empty
71.         cmp r0,#0
72.         beq UARTA_IRQHandler_next_r_empty  //empty
73.         //发送数据
74.         mov r0,#UARTA
75.         ldr r1,=0x20003000
76.         add r1,#4
77.         ldrb r1,[r1]
78.         ldr r2,=flash_ContentDetail
79.         bl uart_sendN
80.         bl UARTA_IRQHandler_finish  //break
81.   //empty->返回空数据
82.   UARTA_IRQHandler_next_r_empty:
83.         mov r0,#UARTA
84.         ldr r1,=flash_str1
85.         bl uart_send_string
86.         bl UARTA_IRQHandler_finish  //break
87.   //读物理地址: "a"+地址 4B+读字节数 1B
88.   UARTA_IRQHandler_next_a:
89.         ldr r0,=0x20003000
90.         ldrb r0,[r0]       //读取数据部分第一位->r0
91.         cmp r0,#'a'        //读取（按物理地址）
92.         bne UARTA_IRQHandler_next_w
93.                                  //  对  应  c  程  序  中  :
    flash_read_physical(flashRead,(gcRecvBuf[1]<<24)|(gcRecvBuf[2]<<16)|(gcRecvBuf[3]
    <<8)|gcRecvBuf[4],gcRecvBuf[5]);
94.         ldr r0,=flash_ContentDetail
95.         ldr r1,=0x20003000
96.         add r1,#1
97.         ldrb r1,[r1]
98.         lsl r1,r1,#24
99.         ldr r2,=0x20003000
100.        add r2,#2
101.        ldrb r2,[r2]
102.        lsl r2,r2,#16
103.        orr r1,r1,r2
104.        ldr r2,=0x20003000
105.        add r2,#3
106.        ldrb r2,[r2]
107.        lsl r2,r2,#8
```

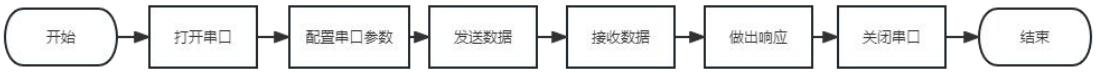
```
        orr r1,r1,r2
        ldr r2,=0x20003000
        add r2,#4
        ldrb r2,[r2]
        orr r1,r1,r2
        ldr r2,=0x20003000
        add r2,#5
        ldrb r2,[r2]
        bl flash_read_physical
        ldr r0,=flash_ContentDetail
        ldr r1,=0x20003000
        add r1,#5
        ldrb r1,[r1]
        bl is_empty
        cmp r0,#0
        beq UARTA_IRQHandler_next_a_empty  //empty
        //发送数据
        mov r0,#UARTA
        ldr r1,=0x20003000
        add r1,#5
        ldrb r1,[r1]
        ldr r2,=flash_ContentDetail
        bl uart_sendN
        bl UARTA_IRQHandler_finish  //break
//empty->返回空数据
UARTA_IRQHandler_next_a_empty:
        mov r0,#UARTA
        ldr r1,=flash_str1
        bl uart_send_string
        bl UARTA_IRQHandler_finish  //break
//写入："w"+扇区号 1B+偏移量(2B)+写入字节数 1B+写入数据 nB
UARTA_IRQHandler_next_w:
        ldr r0,=0x20003000
        ldrb r0,[r0]
        cmp r0,#'w'
        bne UARTA_IRQHandler_next_e
        //擦除，对应 c 程序中：flash_erase(gcRecvBuf[1]);
        ldr r0,=0x20003000
        add r0,#1
        ldrb r0,[r0]
        bl flash_erase
        cmp r0,#0
        bne UARTA_IRQHandler_next_w_failed
                                    //  对  应  c  程  序  中  :
flash_write(gcRecvBuf[1],  (gcRecvBuf[2]<<8)|gcRecvBuf[3],gcRecvBuf[4],&gcRecvBuf[
5]);
```

```
        ldr r0,=0x20003000
        add r0,#1
        ldrb r0,[r0]
        ldr r1,=0x20003000
        add r1,#2
        ldrb r1,[r1]
        lsl r1,r1,#8
        ldr r2,=0x20003000
        add r2,#3
        ldrb r2,[r2]
        orr r1,r1,r2
        ldr r2,=0x20003000
        add r2,#4
        ldrb r2,[r2]
        ldr r3,=0x20003000
        add r3,#5
        bl flash_write
        cmp r0,#0
        bne UARTA_IRQHandler_next_w_failed  //写入失败，r0!=0
        //写入成功，r0==0
        mov r0,#UARTA
        ldr r1,=flash_str5
        bl uart_send_string
        bl UARTA_IRQHandler_finish  //break
//写入失败，r0!=0
UARTA_IRQHandler_next_w_failed:
        mov r0,#UARTA
        ldr r1,=flash_str6
        bl uart_send_string
        bl UARTA_IRQHandler_finish  //break
//擦除："e"+扇区号 1B
UARTA_IRQHandler_next_e:
        ldr r0,=0x20003000
        ldrb r0,[r0]
        cmp r0,#'e'
        bne UARTA_IRQHandler_next_default
        ldr r0,=0x20003000
        add r0,#1
        ldrb r0,[r0]
        bl flash_erase
        cmp r0,#0
        bne UARTA_IRQHandler_next_e_failed  //擦除失败，r0!=0
        //擦除成功，r0==0
        mov r0,#UARTA
        ldr r1,=flash_str3
        bl uart_send_string
```

```
198.        bl UARTA_IRQHandler_finish  //break
199. //擦除失败
200. UARTA_IRQHandler_next_e_failed:
201.        mov r0,#UARTA
202.        ldr r1,=flash_str4
203.        bl uart_send_string
204.        bl UARTA_IRQHandler_finish  //break
205. UARTA_IRQHandler_next_default:  //default
206.        ldr r1,=unknown_info_str
207.        mov r0,#UARTA     //r0 指明串口号
208.        bl  uart_send_string
209. //解除屏蔽，并且恢复现场
210. UARTA_IRQHandler_finish:
211.        cpsie   i          //解除屏蔽中断
212.        add r7,#4          //还原 r7
213.        mov sp,r7          //还原 sp
214.        pop {r0-r7,pc}        //r0-r7,pc 出栈，还原 r7 的值；pc←lr,即返回中断前程序继续执
    行
215.
216. //=============================================================================
217. //函数名称：is_empty
218. //函数参数：r0：需要判断的数组位置
219. //          r1：r0.length
220. //函数返回：r0==0->数据为空
221. //          r0==1->数据不为空
222. //函数功能：判断 r0 指向的数组是否为空
223. //=============================================================================
224. is_empty:
225.     push {r1-r7,lr}
226.     mov r2,#0
227.     mov r3,#0
228. is_empty_loop:
229.     cmp r2,r1
230.     bge is_empty_exit
231.     ldrb r4,[r0,r2]
232.     add r2,#1
233.     cmp r4,#0xff
234.     beq is_empty_loop
235.     mov r3,#1
236. is_empty_exit:
237.     mov r0,r3
238.     pop {r1-r7,pc}
239.
240. //=============================================================================
241. //函数名称：str_equal
242. //函数参数：r0:str1
```

```
243. //          r1:str2,
244. //          r2:str.length
245. //函数返回：r7==0->str1==str2
246. //          r7==1->str1!=str2
247. //函数功能：比较 r0 与 r1 的字符串是否相同
248. //================================================================
249. str_equal:
250.     push {r1-r7,lr}
251.     mov r3,#0
252.     mov r7,#0
253. str_equal_loop:
254.     cmp r3,r2
255.     bge str_equal_exit
256.     ldrb r4,[r0,r3]
257.     ldrb r5,[r1,r3]
258.     add r3,#1
259.     cmp r4,r5
260.     beq str_equal_loop
261.     mov r7,#1
262. str_equal_exit:
263.     mov r0,r7
264.     pop {r1-r7,pc}
```

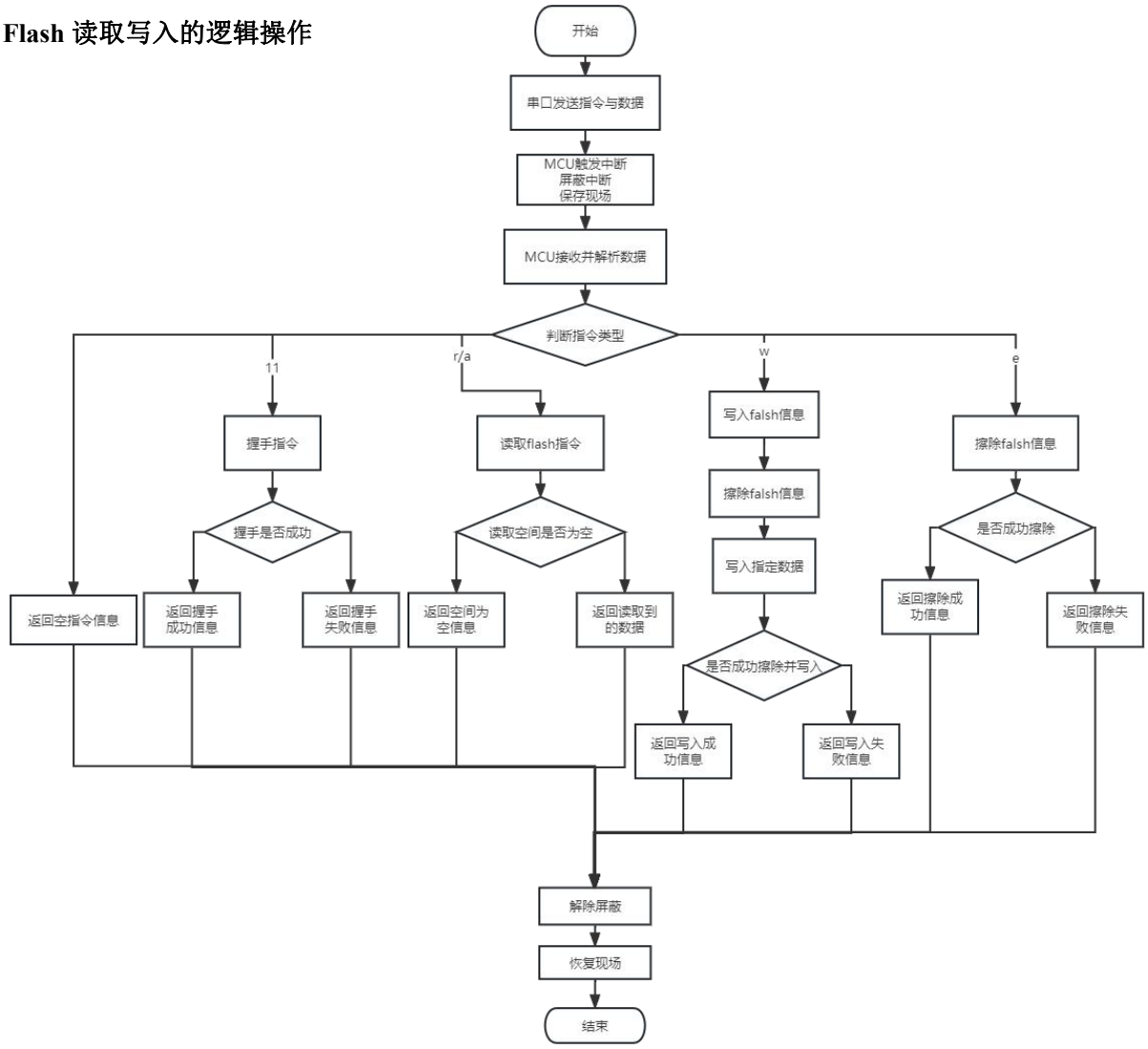
**GEC 方串口通信程序的执行流程:**



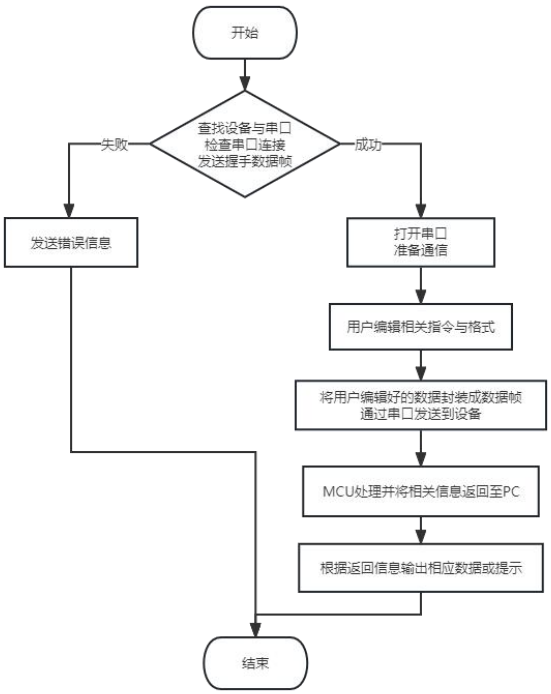

说明：

- 打开串口：首先，需要通过相应的串口库或 API 函数打开串口连接。这将建立与串口设备的物理连接，并为后续的通信做好准备。
- 配置串口参数：在打开串口之后，需要设置串口的参数，如波特率（Baud Rate）、数据位数、校验位、停止位等。这些参数需要与通信的设备或接收方相匹配，以确保正确的数据传输。
- 发送数据：一旦串口配置完成，可以使用相应的函数或方法将数据发送到串口。发送的数据可以是文本、二进制数据或其他格式，取决于具体的应用需求。
- 接收数据：在发送数据后，可以使用相应的函数或方法从串口接收数据。接收到的数据可以进行处理、解析或显示，以满足特定的应用需求。
- 关闭串口：当通信完成或不再需要串口连接时，可以使用相应的函数或方法关闭串口。这将释放串口资源并断开与串口设备的物理连接。

**Flash 读取写入的逻辑操作**

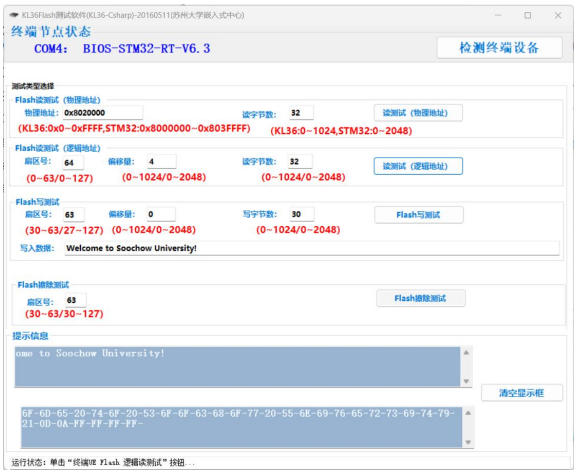



**PC 方的 Flash 测试程序的执行流程。**

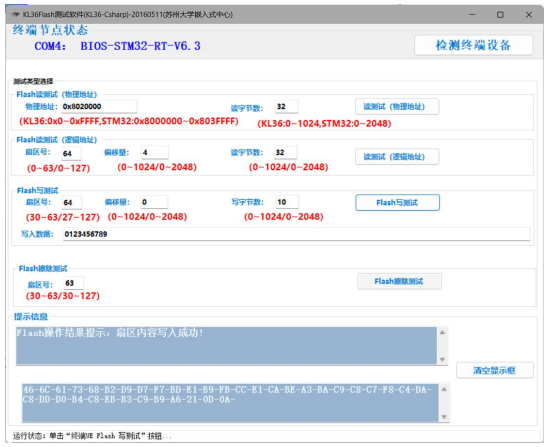



教务处制

运行结果：

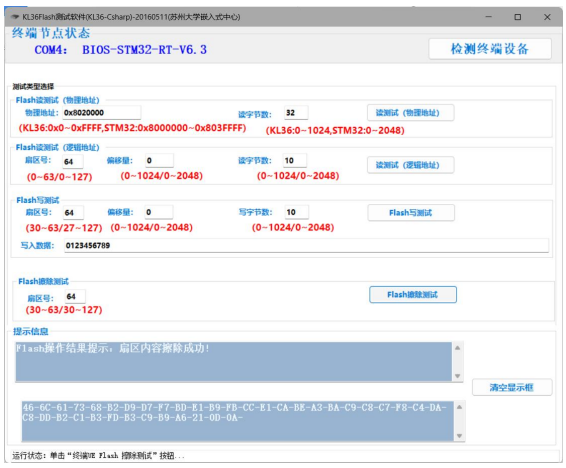



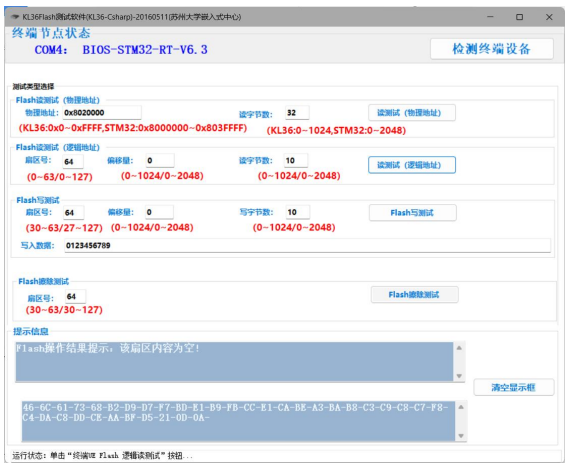

使用物理地址（上左）和逻辑地址（上右）读取到的数据如图，读取正确。





写入数据到 64 扇区（上左）并读取（上右），成功写入并读取




擦除 64 扇区（上左）并再次读取 64 扇区数据（上右），读取失败，符合预期。

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？

- **数据备份**：在进行 Flash 编程之前，务必先备份原始的 Flash 数据。这是因为编程过程中可能发生错误或擦除数据，导致数据丢失。通过备份原始数据，可以在出现问题时进行恢复。
- **正确的编程序列**：Flash 编程通常需要按照特定的序列进行操作，如先擦除扇区、然后写入数据、最后进行验证。确保按照正确的顺序执行编程操作，以避免数据写入错误或无效。
- **擦除操作**：在写入新数据之前，通常需要先擦除 Flash 芯片或特定的扇区。擦除操作是一个不可逆的过程，可能需要较长的时间。确保在适当的时机执行擦除操作，以及正确地选择擦除范围（如扇区或整个芯片）。
- **错误检测和处理**：Flash 编程过程中可能会出现错误，如通信故障、编程失败等。在编程程序中加入错误检测和处理机制，以及错误提示和日志记录，有助于及时发现问题并采取相应的措施。
- **电源稳定性**：在进行 Flash 在线编程时，保持电源的稳定性非常重要。电源的波动或中断可能导致编程失败或数据损坏。使用稳定的电源供应，并在可能的情况下考虑使用备用电源或电源管理策略。
- **数据校验**：在编程完成后，进行数据校验是很重要的一步。通过读取已编程的数据，并与原始数据进行比较，可以验证编程的正确性。
- **编程保护**：Flash 在线编程可能涉及到对敏感数据或固件的更新，因此需要采取必要的编程保护措施。这可能包括设置访问密码、写入保护位、禁用编程接口等，以确保只有授权的人员可以进行编程操作。

（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？

- 可以使用 flash_protect()扇区保护函数来保护扇区
- 分离存储区域：将需要变动的参数与其他数据进行分离存储，可以使用不同的 Flash 扇区或地址范围来存储它们。这样，在更新参数时，只需要操作特定的存储区域，而不影响其他数据的稳定性。
- 使用固定数据结构：在 Flash 区中存储参数时，可以定义固定的数据结构。这可以包括参数标识符、校验和或其他元数据，以确保正确读取和更新参数。通过严格定义数据结构，可以减少错误操作对其他数据的影响。
- 数据校验：在更新参数后，对其他数据进行校验，以确保其完整性和正确性。使用合适的校验算法（如校验和、CRC）对其他数据进行验证，以捕获任何潜在的错误。

（3）如何获取当前程序占用 Flash 的大小？

可以查看使用 AHL-GEC-IDE(4.45)编译好的工程文件中/Debug/(name).map 文件，其中包含了程序占用的 Flash 大小：

```
Memory Configuration

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

教务处制