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：实验四：串口通信汇编程序设计

一. 实验目的

（1）熟悉定时中断计时的工作及编程方法。

（2）理解串行通信的基本概念。

（3）掌握 UART 构件基本应用方法，理解 UART 构件的通信过程。

（4）理解 UART 构件的中断控制过程。

（5）进一步深入理解 MCU 的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

参照“Exam7_1”工程。实现接受上位机发送的字符串，并在字符串末端添加”From MCU”几个字符返回给上位机。上位机测试程序可以是任何串口调试软件。

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\ Exam7_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“Exam7_3”工程，利用该程序框架实现：通过串口调试工具或“..\06-Other\ C#2013 串口测试程序”，发送字符串“open”或者“close”来控制开发板上的 LED 灯，MCU 的 UART 接收到字符串“open”时打开 LED 灯，接收到字符串“close”时关闭 LED 灯。

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。

（3）进阶实验★

上位机通过串口调试软件发送指令如下：

- 1=》红灯亮
- 2=》蓝灯亮
- 3=》绿灯亮
- 4=》青灯亮
- 5=》紫灯亮
- 6=》黄灯亮
- 7=》白灯亮
- 0=》所有灯灭

以上指令都是数字。

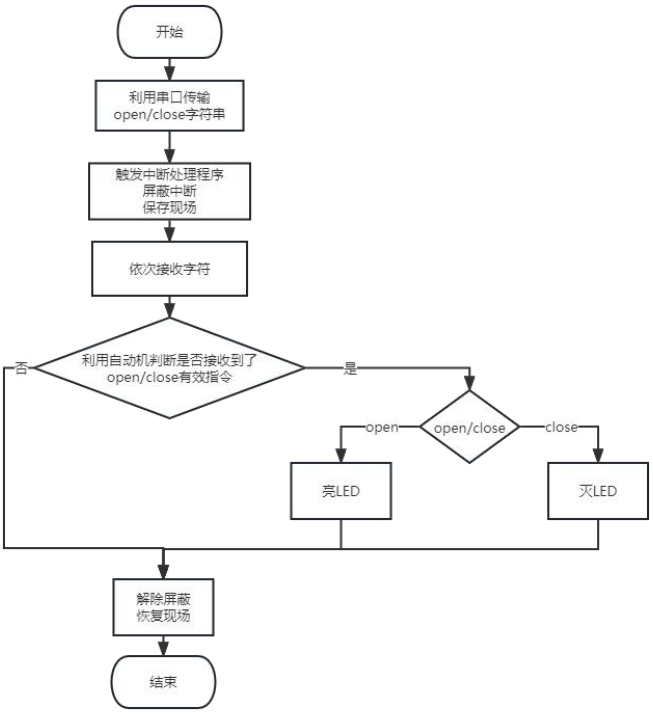
请在实验报告中给出 MCU 端程序 main.s 和 isr.s 流程图及程序语句。

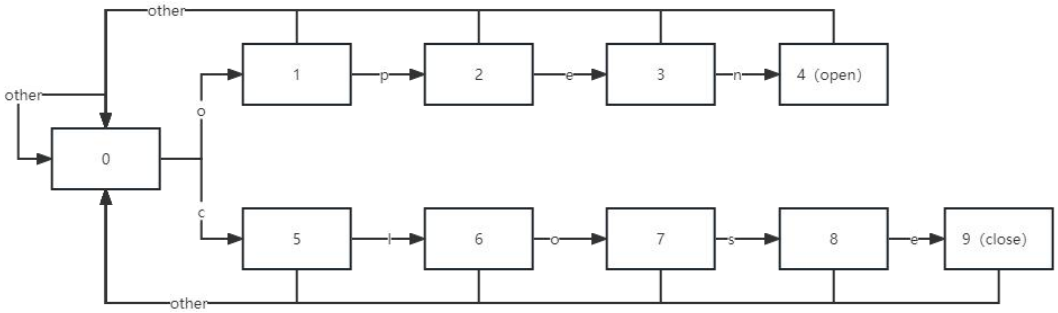
四、实验结果

（1）用适当文字、图表描述实验过程。

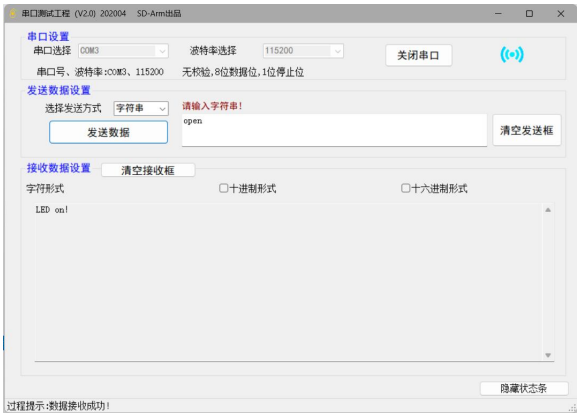
设计性实验：

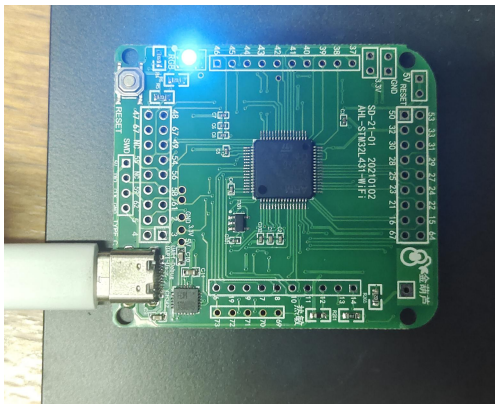
采用自动机来接受处理串口发送到设备的指令，当设备接收到字符串时，进入中断处理部分，其中利用自动机来判断是否完整地接收到了 open 或 close，并做出相应的打开或熄灭 LED 灯，流程图与 DFA 如下：

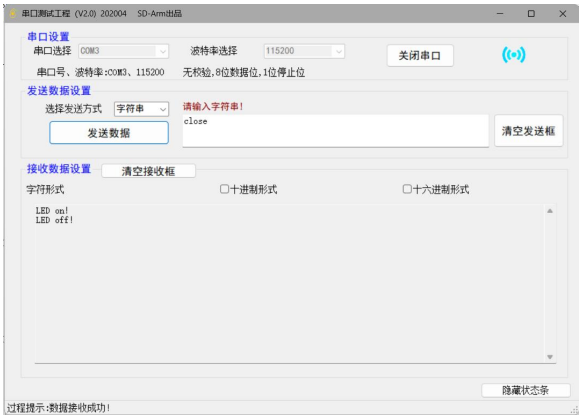


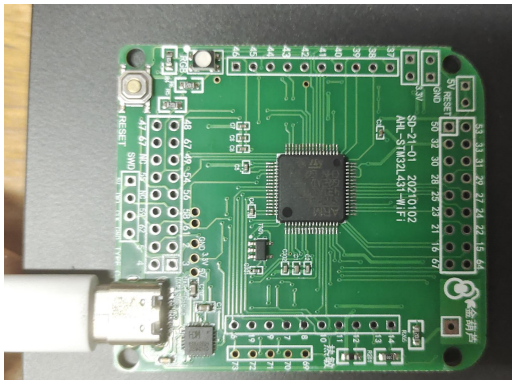


完成相关代码的编写与编译后，运行结果如下：









进阶实验：

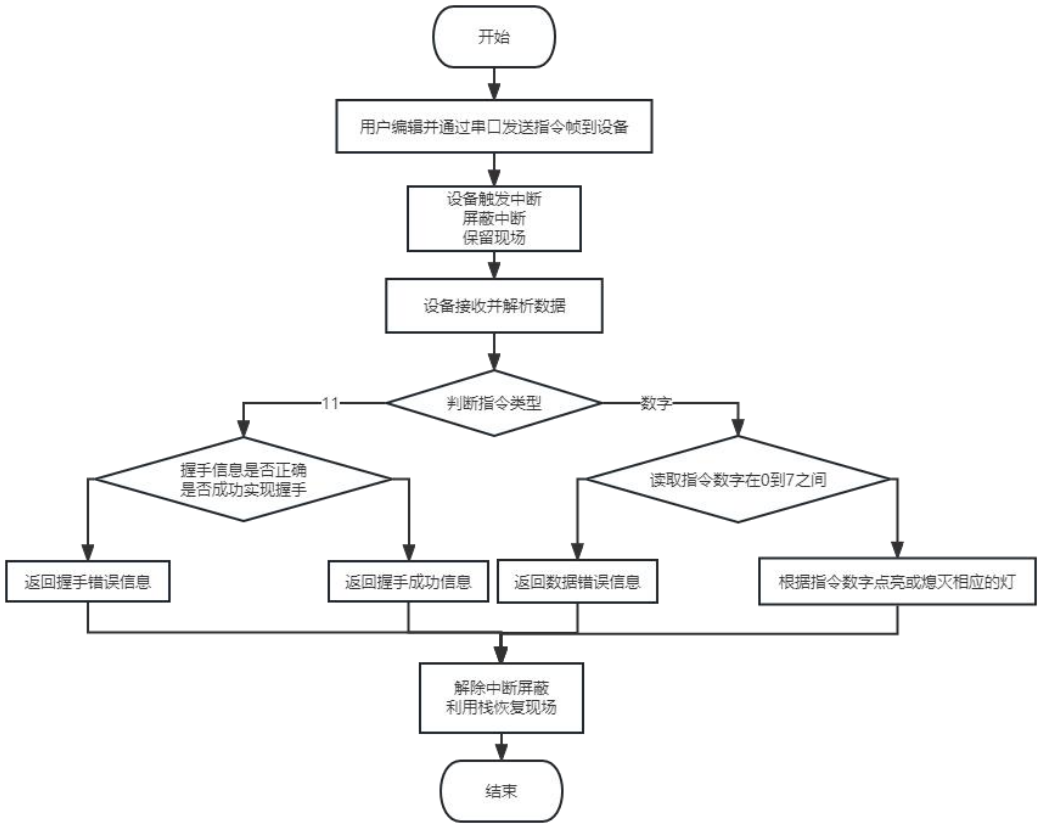
在本实验中，采用的方法与实验 3 类似，其中，握手协议的格式不变，仍然采用如下形式：

```
# 握手协议：(byte)11, (byte)'a', (byte)'u', (byte)'a', (byte)'r', (byte)'t', (byte)'?'
```

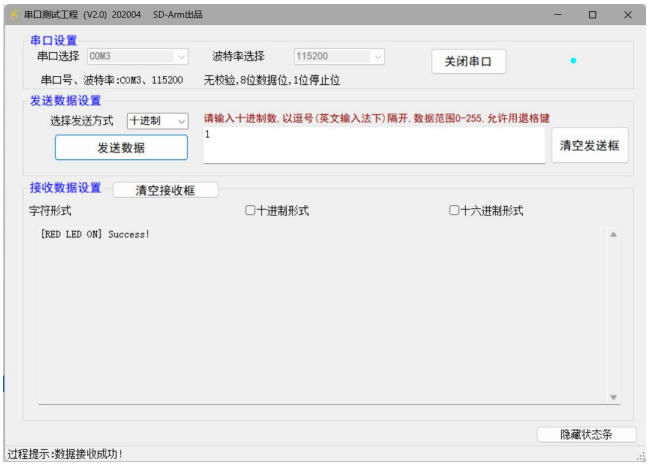
与实验 3 有区别的是：在本实验中，并不要求发送读取或修改 Flash 区域的指令，而是发送如下格式的控制 LED 灯的指令：

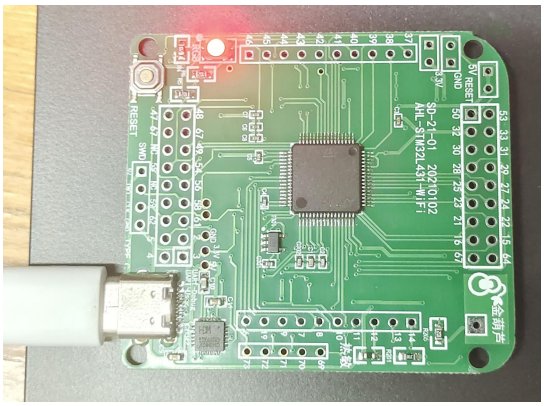
```
# 指令格式：(byte)0/(byte)1/(byte)2/(byte)3/(byte)4/(byte)5/(byte)6/(byte)7
```

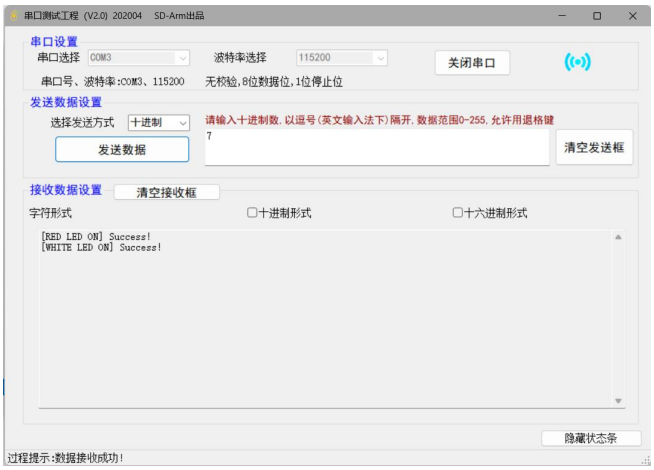
所以只需要在实验 3 的基础上将接收处理指令的部分修改为以上格式即可，流程图如下：

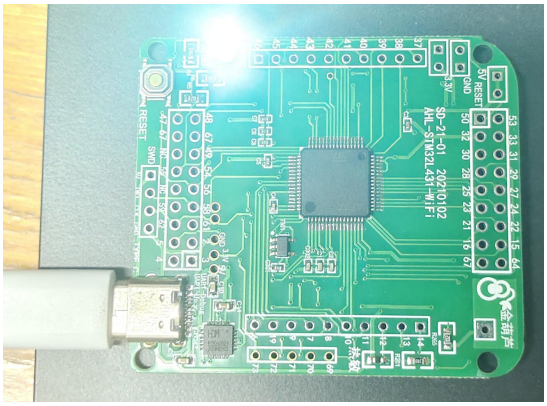


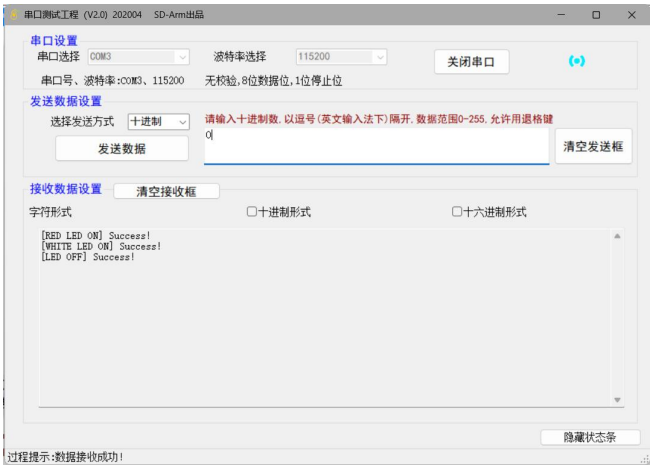
完整相关代码后，实验结果如下：

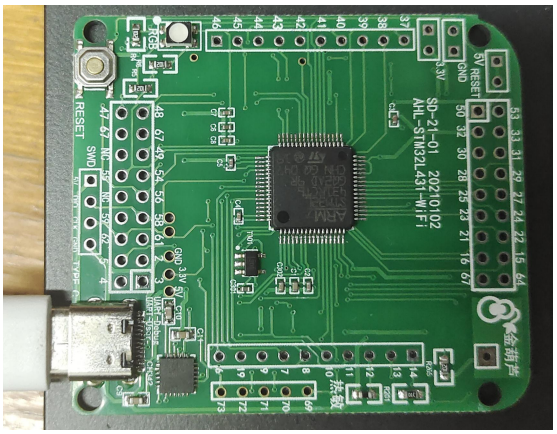












（2）完整给出定时器操作的代码片段

//设计性实验中对定时器操作的代码片段

//isr.s

```
//=====================================================================
//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 {r7,lr}      //保留现场
    sub sp,#4
    mov r7,sp         //r7 存储由 sp 指针偏移获取到的数据
//接收一个字节
    mov r1,r7
    mov r0,#UARTA
    bl  uart_re1      //调用函数接收一个字节
    ldr r3,=recv_state//分析接收到的数据
    ldr r2,[r3]
    //根据 DFA 进行状态转换
    cmp r2,#0
    beq state0
    cmp r2,#1
    beq state1
    cmp r2,#2
    beq state2
    cmp r2,#3
    beq state3
    cmp r2,#5
    beq state5
    cmp r2,#6
    beq state6
```

```
    cmp r2,#7
    beq state7
    cmp r2,#8
    beq state8
state0:
//0--o-->1
//0--c-->5
//0--other-->0
    cmp r0,#'o'
    bne state0_c
    mov r2,#1
    str r2,[r3]
    bl USART2_IRQHandler_exit
state0_c:
    cmp r0,#'c'
    bne state0_other
    mov r2,#5
    str r2,[r3]
    bl USART2_IRQHandler_exit
state0_other:
    mov r2,#0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state1:
//1--p-->2
//1--other-->0
    cmp r0,#'p'
    bne state1_other
    mov r2,#2
    str r2,[r3]
    bl USART2_IRQHandler_exit
state1_other:
    mov r2,#0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state2:
//2--e-->3
//2--other-->0
    cmp r0,#'e'
    bne state2_other
    mov r2,#3
    str r2,[r3]
    bl USART2_IRQHandler_exit
state2_other:
    mov r2,#0
    str r2,[r3]
```

```
76.      bl USART2_IRQHandler_exit
77.  state3:
78.  //3--n-->openLED-->0
79.  //3--other-->0
80.      cmp r0,#'n'
81.      bne state3_other
82.      bl led_on//open LED
83.  state3_other:
84.      mov r2,#0
85.      str r2,[r3]
86.      bl USART2_IRQHandler_exit
87.  state5:
88.  //5--l-->6
89.  //5--other-->0
90.      cmp r0,#'l'
91.      bne state5_other
92.      mov r2,#6
93.      str r2,[r3]
94.      bl USART2_IRQHandler_exit
95.  state5_other:
96.      mov r2,#0
97.      str r2,[r3]
98.      bl USART2_IRQHandler_exit
99.  state6:
100. //6--o-->7
101. //6--other-->0
102.     cmp r0,#'o'
103.     bne state6_other
104.     mov r2,#7
105.     str r2,[r3]
106.     bl USART2_IRQHandler_exit
107. state6_other:
108.     mov r2,#0
109.     str r2,[r3]
110.     bl USART2_IRQHandler_exit
111. state7:
112. //7--s-->8
113. //7--other-->0
114.     cmp r0,#'s'
115.     bne state7_other
116.     mov r2,#8
117.     str r2,[r3]
118.     bl USART2_IRQHandler_exit
119. state7_other:
120.     mov r2,#0
121.     str r2,[r3]
```

```
    bl USART2_IRQHandler_exit
state8:
//8--e-->closeLED-->0
//8--other-->0
    cmp r0,#'e'
    bne state8_other
    bl led_off//close LED
state8_other:
    mov r2,#0
    str r2,[r3]
USART2_IRQHandler_exit:
    //解除屏蔽中断
    cpsie i
    //还原现场
    add r7,#4
    mov sp,r7
    pop {r7,pc}

led_on:
//open LED
    push {r0-r7,lr}
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_ON
    bl  gpio_set
    mov r0,#UARTA
    ldr r1,=light_on_string
    bl uart_send_string
    pop {r0-r7,pc}

led_off:
//close LED
    push {r0-r7,lr}
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    mov r0,#UARTA
    ldr r1,=light_off_string
    bl uart_send_string
    pop {r0-r7,pc}
```

//main.s

```
main:
//（1）======启动部分（开头）主循环前的初始化工作=======
//（1.1）声明 main 函数使用的局部变量
//（1.2）【不变】关总中断
    cpsid i
//（1.3）给主函数使用的局部变量赋初值
```

```
//（1.4）给全局变量赋初值
//（1.5）用户外设模块初始化
//  使用 gpio_init 函数进行初始化 LED 灯
    ldr r0,=LIGHT_BLUE
    mov r1,#GPIO_OUTPUT
    mov r2,#LIGHT_ON
    bl  gpio_init
//  使用 uart_init 函数初始化串口 UARTA
    mov r0,#UARTA
    ldr r1,=UART_BAUD
    bl  uart_init
//  调用 uart_enable_re_int 函数使能模块中断
    mov r0,#UARTA
    bl  uart_enable_re_int
//（1.7）【不变】开总中断
    cpsie  i
    ldr r0,=instruction
    bl printf
main_loop:
    b main_loop
.end
```

//进阶实验：

//isr.s

```
//=====================================================================
//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
    push  {r0-r7,lr}    //保留现场
    sub sp,#4
    mov r7,sp          //r7 存储由 sp 指针偏移获取到的数据
    //接收一个字节
    mov r1,r7
    mov r0,#UART_User
    bl  uart_re1          //调用函数接收一个字节
    ldr r1,=0x20003000    //gcRecvBuf：其指向数据部分的存储位置
    bl  emuart_frame      //帧解析函数
    cmp r0,#0             //是否成功解析了帧
    bne USART2_IRQHandler_success_revc//协议解析
    bl  USART2_IRQHandler_exit   //exit
USART2_IRQHandler_success_revc:
                                          //  握  手  协议: (byte)11, (byte)'a', (byte)'u', (byte)'a', (byte)'r', (byte)'t', (byte)'?'
```

教务处制

```
    ldr r0,=0x20003000
    ldrb r0,[r0]
    cmp r0,#11
    bne USART2_IRQHandler_next
    ldr r0,=handshake_check_str
    ldr r1,=0x20003000
    add r1,#1
    mov r2,#6
    bl str_equal
    cmp r0,#0  //判断是不是握手协议
    //不是握手协议
    bne USART2_IRQHandler_next
    //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 Light_Connected
    bl USART2_IRQHandler_exit    //exit
USART2_IRQHandler_next:
    ldr r0,=0x20003000
    ldrb r0,[r0]        //r0:读取的数据的首位
    cmp r0,#7
    bhi USART2_IRQHandler_exit  //接收到的数据>7,无效指令，exit
    ldr r1,=mLightCommand
    str r0,[r1]
    bl  Light_Control  //Light_Control 函数
    ldr r1,=success_command_string
    mov r0,#UART_User
    bl  uart_send_string    //向串口返回数据
USART2_IRQHandler_exit:
//exit,解除屏蔽，恢复现场
    cpsie   i
    add r7,#4
    mov sp,r7
    pop {r0-r7,pc}

//=============================================================================
//函数名称：str_equal
//函数参数：r0:str1
//         r1:str2
//         r2:str.length
//函数返回：r7==0->str1==str2
//         r7==1->str1!=str2
//函数功能：比较 r0 与 r1 的字符串是否相同
//=============================================================================
str_equal:
```

```
    push {r1-r7,lr}
    mov r3,#0
    mov r7,#0
str_equal_loop:
    cmp r3,r2
    bge str_equal_exit
    ldrb r4,[r0,r3]
    ldrb r5,[r1,r3]
    add r3,#1
    cmp r4,r5
    beq str_equal_loop
    mov r7,#1
str_equal_exit:
    mov r0,r7
    pop {r1-r7,pc}


//=============================================================================
//程序名称：Light_Control
//程序功能：根据接收到的指令，对 LED 灯进行相应颜色的 open/close
//          0->所有灯灭
//          1->红灯亮（RED）
//          2->蓝灯亮（BLUE）
//          3->绿灯亮（GREEN）
//          4->青灯亮（BLUE+GREEN）
//          5->紫灯亮（RED+BLUE）
//          6->黄灯亮（RED+GREEN）
//          7->白灯亮（RED+BLUE+GREEN）
//=============================================================================
Light_Control:
    push {r0-r7,lr}
    //默认熄灭 LED 灯
    ldr r0,=LIGHT_RED
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_GREEN
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    //读取灯指令保存至 r0
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#0
    bne Light_Control_case_1
```

```
    // 0->所有灯灭
    ldr r1,=light_off_string
    mov r0,#UART_User
    bl  uart_send_string    //回发信息
    bl  Light_Control_exit
Light_Control_case_1:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#1
    bne Light_Control_case_2
    //1->红灯亮（RED）
    ldr r0,=LIGHT_RED
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_red_on_string
    mov r0,#UART_User
    bl  uart_send_string
    b Light_Control_exit            //exit
Light_Control_case_2:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#2
    bne Light_Control_case_3
    //2->蓝灯亮（BLUE）
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_blue_on_string
    mov r0,#UART_User
    bl  uart_send_string
    b Light_Control_exit            //exit
Light_Control_case_3:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#3
    bne Light_Control_case_4
    // 3->绿灯亮（GREEN）
    ldr r0,=LIGHT_GREEN
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_green_on_string
    mov r0,#UART_User
    bl  uart_send_string
    b Light_Control_exit            //exit
Light_Control_case_4:
    ldr r0,=mLightCommand
```

```
    ldr r0,[r0]
    cmp r0,#4
    bne Light_Control_case_5
    // 4->青灯亮（BLUE+GREEN）
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0,=LIGHT_GREEN
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_cyan_on_string
    mov r0,#UART_User
    bl  uart_send_string
    b Light_Control_exit            //exit
Light_Control_case_5:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#5
    bne Light_Control_case_6
    // 5->紫灯亮（RED+BLUE）
    ldr r0,=LIGHT_RED
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_purple_on_string
    mov r0,#UART_User
    bl  uart_send_string
    b Light_Control_exit            //exit
Light_Control_case_6:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#6
    bne Light_Control_case_7
    //6->黄灯亮（RED+GREEN）
    ldr r0,=LIGHT_RED
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r0,=LIGHT_GREEN
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_yellow_on_string
    mov r0,#UART_User
    bl  uart_send_string
    b Light_Control_exit            //exit
```

```
206. Light_Control_case_7:
207.     ldr r0,=mLightCommand
208.     ldr r0,[r0]
209.     cmp r0,#7
210.     bne Light_Control_exit
211.     // 7->白灯亮（RED+BLUE+GREEN）
212.     ldr r0,=LIGHT_RED
213.     ldr r1,=LIGHT_ON
214.     bl  gpio_set
215.     ldr r0,=LIGHT_BLUE
216.     ldr r1,=LIGHT_ON
217.     bl  gpio_set
218.     ldr r0,=LIGHT_GREEN
219.     ldr r1,=LIGHT_ON
220.     bl  gpio_set
221.     ldr r1,=light_white_on_string
222.     mov r0,#UART_User
223.     bl  uart_send_string
224. Light_Control_exit:
225. //exit，恢复现场
226.     pop  {r0-r7,pc}
227.
228.
229. //=========================================================================
230. //程序名称：Light_Connected
231. //程序功能：当成功连接后，使 LED 亮蓝灯，表示连接成功
232. //=========================================================================
233. Light_Connected:
234.     //灭灯
235.     push {r0-r7,lr}
236.     ldr r0,=LIGHT_RED
237.     ldr r1,=LIGHT_OFF
238.     bl  gpio_set
239.     ldr r0,=LIGHT_GREEN
240.     ldr r1,=LIGHT_OFF
241.     bl  gpio_set
242.     ldr r0,=LIGHT_BLUE
243.     ldr r1,=LIGHT_OFF
244.     bl  gpio_set
245.     //亮蓝灯
246.     ldr r0,=LIGHT_BLUE
247.     ldr r1,=LIGHT_ON
248.     bl  gpio_set
249. Light_Connected_exit:
250.     pop  {r0-r7,pc}
```

教务处制

//main.s:

```
main:
//（1）======启动部分（开头）主循环前的初始化工作======
//（1.1）声明 main 函数使用的局部变量
//（1.2）【不变】关总中断
    cpsid i
//（1.3）给主函数使用的局部变量赋初值
//（1.4）给全局变量赋初值
//（1.5）用户外设模块初始化
// 调用 gpio_init 函数初始化 LED 灯
    ldr r0,=LIGHT_BLUE
    mov r1,#GPIO_OUTPUT
    mov r2,#LIGHT_ON
    bl  gpio_init
    ldr r0,=LIGHT_RED
    mov r1,#GPIO_OUTPUT
    mov r2,#LIGHT_ON
    bl  gpio_init
    ldr r0,=LIGHT_GREEN
    mov r1,#GPIO_OUTPUT
    mov r2,#LIGHT_ON
    bl  gpio_init
// 调用 uart_init 函数初始化串口 UARTA
    mov r0,#UARTA
    ldr r1,=UART_BAUD
    bl  uart_init
//（1.6）调用 uart_enable_re_int 函数使能模块中断
    mov r0,#UARTA
    bl  uart_enable_re_int
//（1.7）【不变】开总中断
    cpsie  i
    ldr r0,=instruction
    bl printf
//（1）======启动部分（结尾）=========================================
//（2）======主循环部分（开头）=========================================
main_loop:
    b main_loop
.end
```

五. 实践性问答题

（1）波特率 9600bps 和 115200bps 的区别是什么？

- 在串口通信中，波特率是指每秒传输的比特数，通常以 bps（位每秒）为单位。波特率决定了数据传输的速度和可靠性。区分 9600bps 和 115200bps 的关键是它们的传输速度不同。
- 9600bps 表示每秒传输 9600 个位，即每秒钟可以传输 960 个字符或字节。这个速度适合于较慢的

数据传输，例如简单的文本消息或控制指令。

- 而 115200bps 表示每秒传输 115200 个位，即每秒钟可以传输 11520 个字符或字节。这个速度比 9600bps 快得多，适用于需要更高传输速度的应用，例如传输大量数据或高速传输图像和视频等。
- 总结起来，9600bps 适合于较慢的数据传输和低带宽要求的应用，而 115200bps 适合于需要更高速度和较大带宽的应用。选择合适的波特率取决于具体的应用需求和通信环境。

（2）有什么最简单的方法知道 GEC 串口的 TX 发送了信号？

要确定 GEC 串口的 TX（发送）线是否发送了信号，最简单的方法是观察发送的和接收到的内容是否一致。从 PC 机通过 GEC 串口的 TX（发送）线发送数据时，设备会触发中断，在 MCU 接收到数据之后，将会立刻通过 TX 将当前接收到的内容回发。

（3）串口通信中用电平转换芯片（RS-485 或 RS-232）进行电平转换，程序是否需要修改？说明原因。

需要修改，RS-485 是差分信号，通常使用正负电平表示逻辑值，而 RS-232 则使用正负电压表示逻辑值。因此，在使用 RS-485 或 RS-232 转换芯片时，程序需要相应地修改以适应差分信号的接收和发送。

（4）不用其他工具，如何测试发送一个字符的真实时间？

可以通过串口发送一个较长的数据，记录下发送数据的首位和末位字符的时间差，用数据的字符数除以记录下的时间差即可得到发送一个字符的真实时间，同时还可以利用统计学技巧，例如多次取平均等来减小误差。

教务处制