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 5 月 20 日	

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★

利用 7.3 中的例二的组帧方法完成 MCU 方程序功能，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。

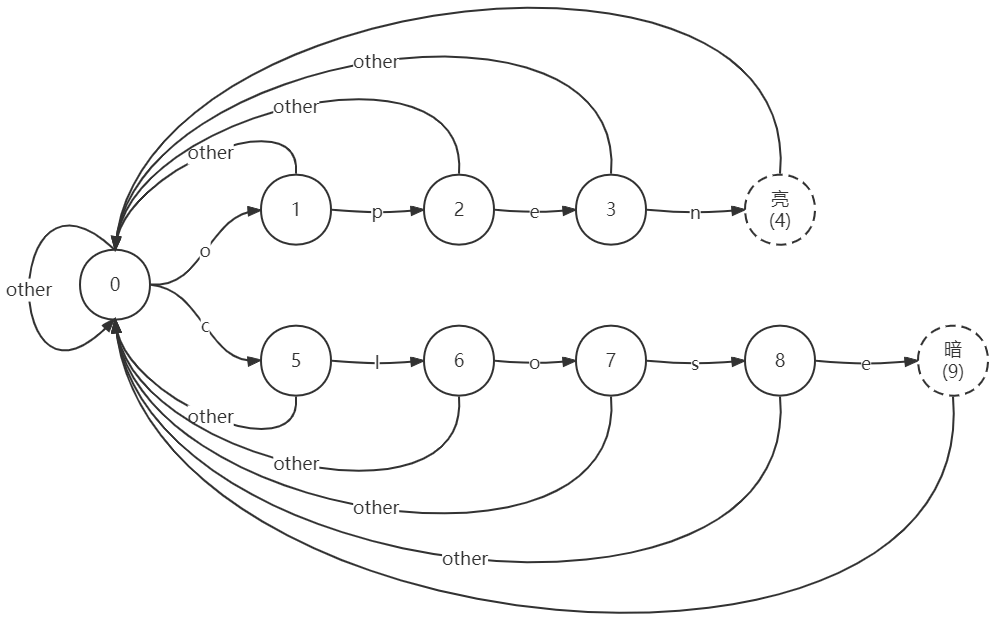
提示：组帧的双方可约定“帧头+数据长度+有效数据+帧尾”为数值帧的格式，帧头和帧尾请自行设定。

四、实验结果

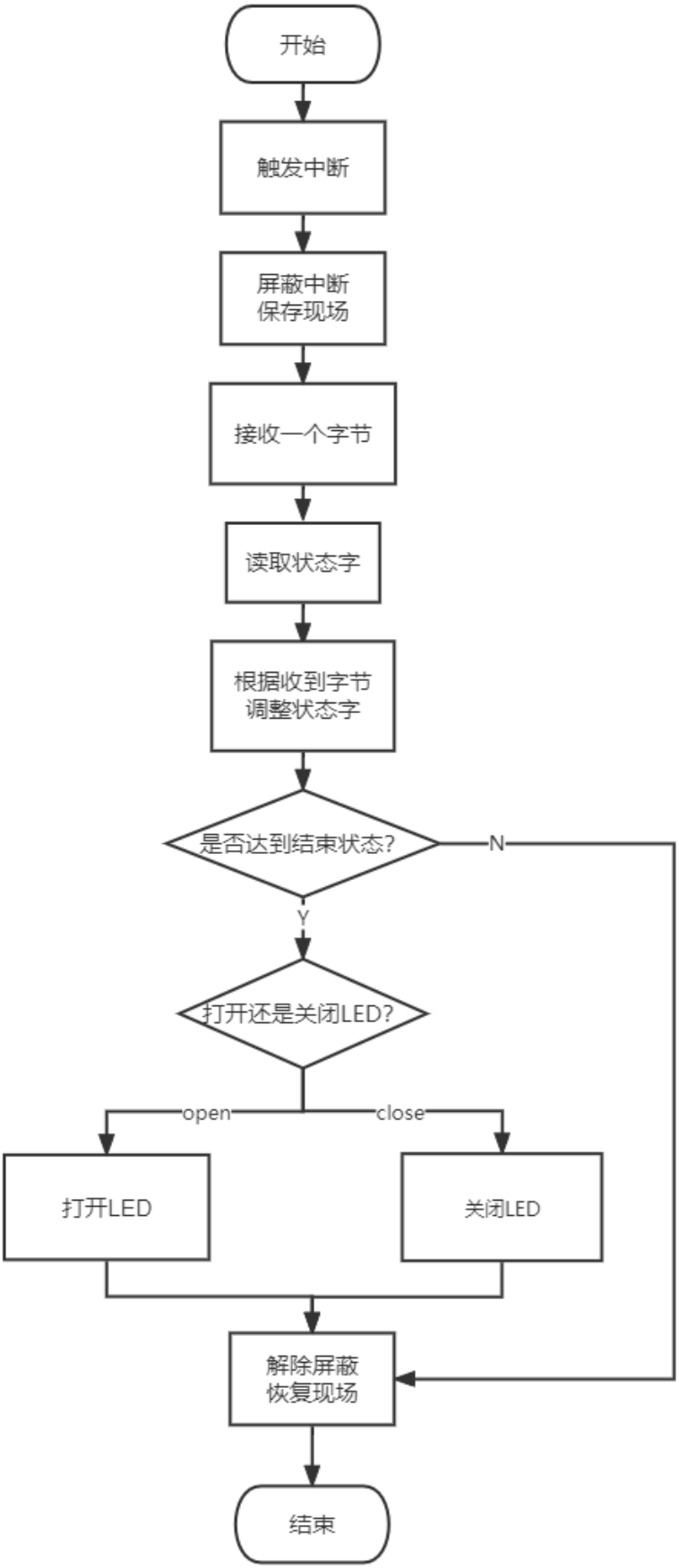
（1）用适当文字、图表描述实验过程。

设计性实验：

在 MCU 端使用一个自动机来实现字符串“open”或者“close”的接收，维护一个全局状态 recv_state，每次接收一个字符，中断处理利用自动机，根据收到的字符和当前状态来判断是否完整收到了字符串“open”或者“close”，相应地调用控制 LED 灯亮灭的函数。自动机如下图所示（虚线状态不会停留）：



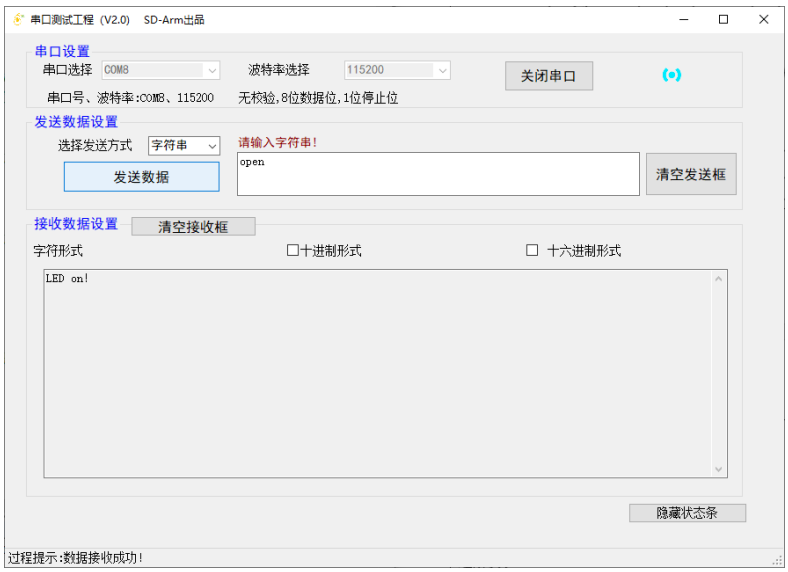
上述流程图：

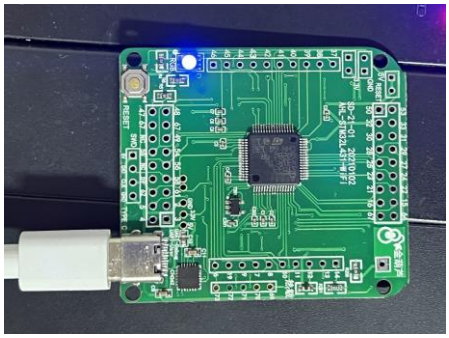

开始
触发中断
屏蔽中断
保存现场
接收一个字节
读取状态字
根据收到字节
调整状态字
是否达到结束状态?
N
Y
打开还是关闭LED?
open
close
打开LED
关闭LED
解除屏蔽
恢复现场
结束

实验结果：

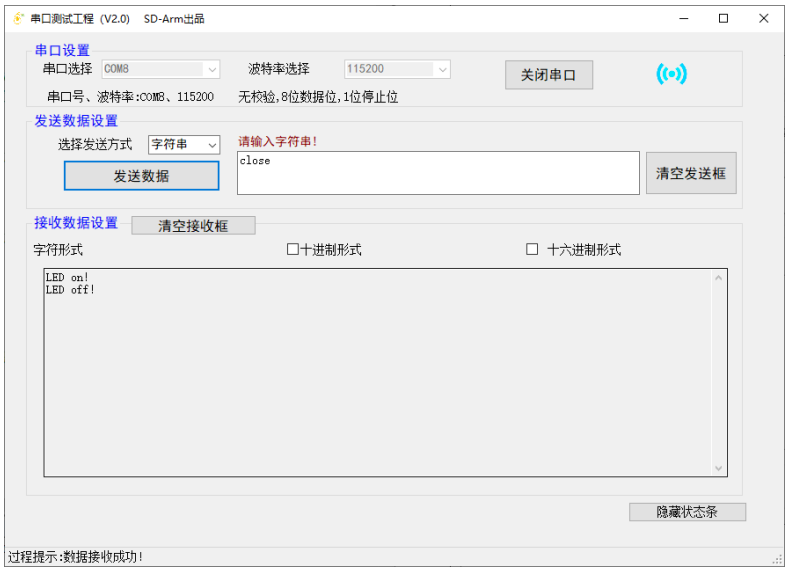
使用串口调试工具进行测试：

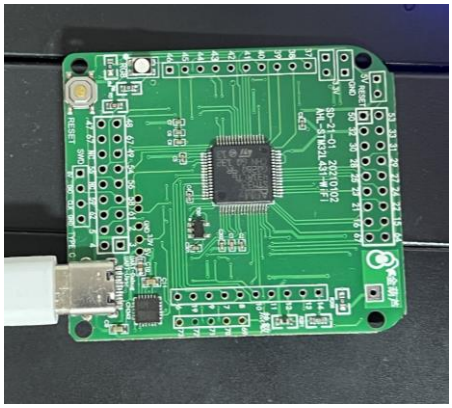
发送 open 打开 LED 灯：





发送 close 关闭 LED 灯：





进阶实验：

使用自己编写的 C#程序与 MCU 连接并发送指令。

参考实验三使用的 C# Flash 程序，并做修改。

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）。

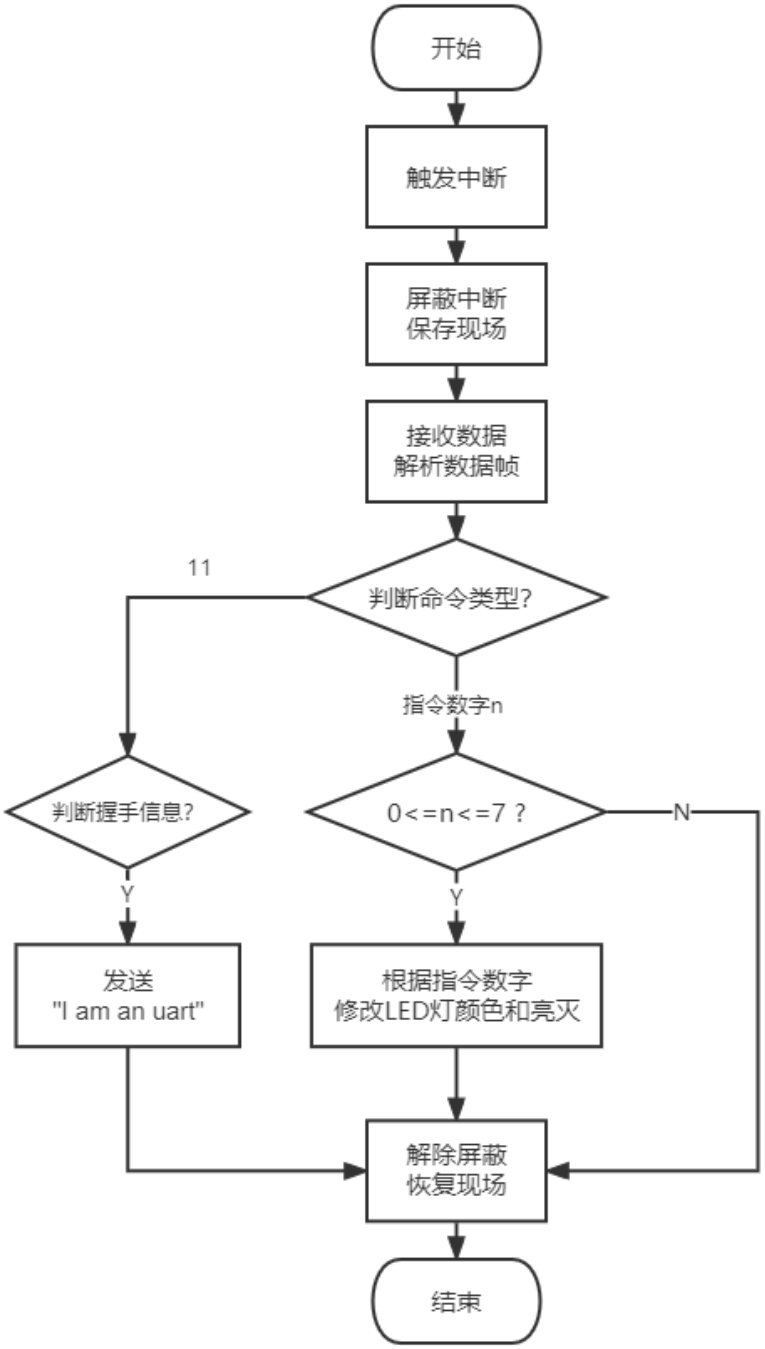
协议的命令如下：

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"	

LED 灯控制命令		
PC 机发送	Length=1	数据=指令数字 0~7，其中 1 表示红灯亮，2 表示蓝灯亮，3 表示绿灯亮，4 表示青灯亮，5 表示紫灯亮，6 表示黄灯亮，7 表示白灯亮，0 表示所有灯灭。
MCU 响应	根据指令改变 LED 灯的颜色和亮灭，返回操作是否成功	

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。

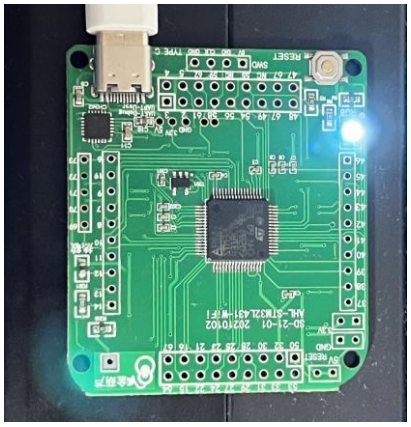
上述流程图:



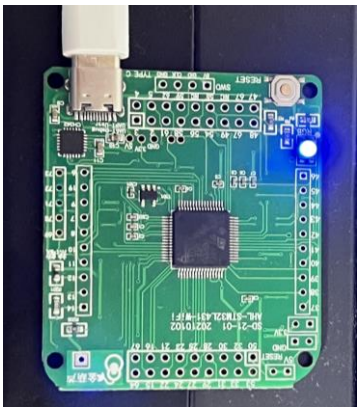
实验结果：

将开发板与 PC 通过串口连接后，MCU 上电，LED 灯默认为白色，如下页左图所示;

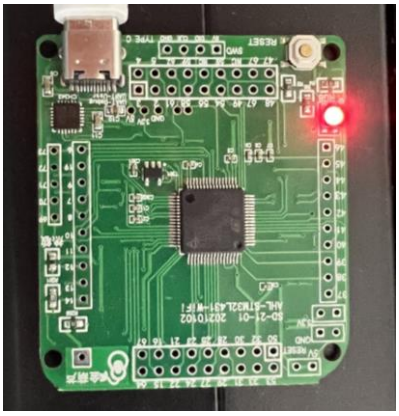
运行 C#程序，界面如右图：



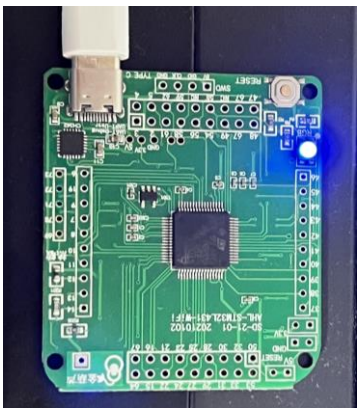
点击“连接终端设备”按钮，与 MCU 握手，握手成功后 MCU 的蓝色 LED 灯亮起：



点击红色按钮，C#程序发送指令 1，MCU 的红色 LED 灯亮起：



点击蓝色按钮，C#程序发送指令 2，MCU 的蓝色 LED 灯亮起：

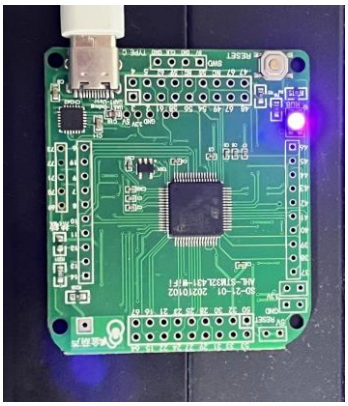


点击绿色按钮，C#程序发送指令 3，MCU 的绿色 LED 灯亮起：

点击青色按钮，C#程序发送指令 4，MCU 的青色 LED 灯亮起：

点击黄色按钮，C#程序发送指令 6，MCU 的黄色 LED 灯亮起：

点击紫色按钮，C#程序发送指令 5，MCU 的紫色 LED 灯亮起：



点击白色按钮，C#程序发送指令 7，MCU 的红色、蓝色、绿色 LED 灯亮起：

点击关闭 LED 按钮，C#程序发送指令 0，MCU 的所有 LED 灯关闭：

（2）完整给出代码片段

```
//设计性实验中的代码片段
//isr.s
USART2_IRQHandler:
//（1）屏蔽中断，并且保存现场
    cpsid  i          //关可屏蔽中断
    push {r7,lr}      //r7,lr进栈保护（r7后续申请空间用，lr中为进入中断前pc的值）
    //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
//（2）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
    //处理接受到的信息
    ldr r3,=recv_state
    ldr r2,[r3]
    cmp r2,#0         //State 0: o->1, c->5, other->0
    beq state0
    cmp r2,#1         //State 1: open, p->2, other->0
    beq state1
```

```
    cmp r2,#2        //State 2: open, e->3, other->0
    beq state2
    cmp r2,#3        //State 3: open, n->openLED->0
    beq state3
    cmp r2,#5        //State 5: close, l->6, other->0
    beq state5
    cmp r2,#6        //State 5: close, o->7, other->0
    beq state6
    cmp r2,#7        //State 6: close, s->8, other->0
    beq state7
    cmp r2,#8        //State 7: close, e->closeLED->0
    beq state8
state0:
    cmp r0,#'o'       //o->1
    bne state0_c
    mov r2,#1
    str r2,[r3]
    bl USART2_IRQHandler_exit
state0_c:
    cmp r0,#'c'       //c->5
    bne state0_other
    mov r2,#5
    str r2,[r3]
    bl USART2_IRQHandler_exit
state0_other:
    mov r2,#0     //other->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state1:
    cmp r0,#'p'      //p->2
    bne state1_other
    mov r2,#2
    str r2,[r3]
    bl USART2_IRQHandler_exit
state1_other:
    mov r2,#0        //other->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state2:
    cmp r0,#'e'      //e->3
    bne state2_other
    mov r2,#3
    str r2,[r3]
    bl USART2_IRQHandler_exit
state2_other:
```

```
    mov r2,#0        //other->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state3:
    cmp r0,#'n'      //n->openLED
    bne state3_other
    //LED ON
    bl led_on_caller
state3_other:
    mov r2,#0        //all->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state5:
    cmp r0,#'l'      //l->6
    bne state5_other
    mov r2,#6
    str r2,[r3]
    bl USART2_IRQHandler_exit
state5_other:
    mov r2,#0        //other->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state6:
    cmp r0,#'o'      //o->7
    bne state6_other
    mov r2,#7
    str r2,[r3]
    bl USART2_IRQHandler_exit
state6_other:
    mov r2,#0        //other->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state7:
    cmp r0,#'s'      //s->8
    bne state7_other
    mov r2,#8
    str r2,[r3]
    bl USART2_IRQHandler_exit
state7_other:
    mov r2,#0        //other->0
    str r2,[r3]
    bl USART2_IRQHandler_exit
state8:
    cmp r0,#'e'      //e->closeLED
    bne state8_other
```

```
    //LED OFF
    bl led_off_caller
state8_other:
    mov r2,#0         //all->0
    str r2,[r3]
USART2_IRQHandler_exit:
//（4）解除屏蔽，并且恢复现场
    cpsie   i         //解除屏蔽中断
    add r7,#4         //还原r7
    mov sp,r7         //还原sp
    pop {r7,pc}       //r7,pc出栈，还原r7的值；pc←lr,即返回中断前程序继续执行

led_on_caller:  //亮灯
    push {r0-r7,lr}
    ldr r0,=LIGHT_BLUE  //亮灯
    ldr r1,=LIGHT_ON
    bl  gpio_set
    mov r0,#UARTA     //Light on string
    ldr r1,=light_on_string
    bl uart_send_string
    pop {r0-r7,pc}

led_off_caller:   //暗灯
    push {r0-r7,lr}
    ldr r0,=LIGHT_BLUE//暗灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    mov r0,#UARTA     //Light off string
    ldr r1,=light_off_string
    bl uart_send_string
    pop {r0-r7,pc}

//main.s
main:
//【不变】关总中断
    cpsid i
//用户外设模块初始化
//  初始化蓝灯，r0、r1、r2是gpio_init的入口参数
    ldr r0,=LIGHT_BLUE     //r0指明端口和引脚
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_ON       //r2指明引脚的初始状态为亮
    bl  gpio_init          //调用gpio初始化函数
//  初始化串口UARTA
    mov r0,#UARTA              //r0=更新串口
    ldr r1,=UART_BAUD
```

```
    bl  uart_init
//使能模块中断
    mov r0,#UARTA               //r0指明串口号
    bl  uart_enable_re_int      //调用串口中断使能函数
//【不变】开总中断
    cpsie  i
    ldr r0,=instruction
    bl printf
//======主循环部分（开头）=======================================
main_loop:                      //主循环标签（开头）
    b main_loop                 //死循环
//======主循环部分（结尾）=======================================
.end        //整个程序结束标志（结尾）
```

```
//进阶实验中的代码片段
//isr.s
.section .text
USART2_IRQHandler:
    //屏蔽中断，并且保存现场
    cpsid  i          //关可屏蔽中断
    push  {r0-r7,lr}   //r7,lr进栈保护（r7后续申请空间用，lr中为进入中断前pc的值）
    //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,#UART_User     //串口号
    bl  uart_re1       //调用接收一个字节子函数
    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 USART2_IRQHandler_success_revc
    //否则直接退出
    bl  USART2_IRQHandler_exit    //break
USART2_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 USART2_IRQHandler_next
    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 USART2_IRQHandler_next    //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
    //亮蓝灯
    bl Light_Connected
    bl USART2_IRQHandler_exit    //break
USART2_IRQHandler_next:
    ldr r0,=0x20003000
    ldrb r0,[r0]       //读取数据部分第一位（指令）->r0
    cmp r0,#7          //大于7，指令无效
    bhi USART2_IRQHandler_exit
    ldr r1,=mLightCommand
    str r0,[r1]           //指令保存至mLightCommand变量
    //调用Light_Control
    bl  Light_Control
    ldr r1,=success_command_string       //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
USART2_IRQHandler_exit:
    //解除屏蔽，并且恢复现场
    cpsie   i         //解除屏蔽中断
    add r7,#4         //还原r7
    mov sp,r7         //还原sp
    pop {r0-r7,pc}    //r7,pc出栈，还原r7的值；pc←lr,即返回中断前程序继续执行

//=========================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}

//=========================================================================
//程序名称：Light_Control
//参数说明：无
//返回值说明：无
//程序功能：根据灯指令（mLightCommand），对LED灯进行操作
//  1=》红灯亮  2=》蓝灯亮  3=》绿灯亮  4=》青灯亮(蓝,绿)
//  5=》紫灯亮(蓝,红)  6=》黄灯亮(红,绿)  7=》白灯亮(红,蓝,绿)  0=》所有灯灭
//=========================================================================
Light_Control:
    push {r0-r7,lr}
    //首先关掉所有的灯
    ldr r0,=LIGHT_RED //灭红灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_GREEN //灭绿灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_BLUE //灭蓝灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    //读取灯指令保存至r0
    ldr r0,=mLightCommand
    ldr r0,[r0]
    //判断r0是不是0
    cmp r0,#0
    bne Light_Control_case_1  //不为0，判断并亮灯
    //若为0直接结束
    ldr r1,=light_off_string      //r1指明字符串
    mov r0,#UART_User         //串口号
    bl  uart_send_string    //向原串口回发
    bl  Light_Control_exit
Light_Control_case_1:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#1
    bne Light_Control_case_2          //不等于1转判断2
    //等于1的情况:RED
    ldr r0,=LIGHT_RED
    ldr r1,=LIGHT_ON
```

```
    bl  gpio_set
    ldr r1,=light_red_on_string      //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
    b Light_Control_exit          //break
Light_Control_case_2:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#2
    bne Light_Control_case_3          //不等于2转判断3
    //等于2的情况:BLUE
    ldr r0,=LIGHT_BLUE
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_blue_on_string     //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
    b Light_Control_exit          //break
Light_Control_case_3:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#3
    bne Light_Control_case_4          //不等于3转判断4
    //等于5的情况:GREEN
    ldr r0,=LIGHT_GREEN
    ldr r1,=LIGHT_ON
    bl  gpio_set
    ldr r1,=light_green_on_string    //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
    b Light_Control_exit          //break
Light_Control_case_4:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#4
    bne Light_Control_case_5          //不等于4转判断5
    //等于4的情况: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     //r1指明字符串
    mov r0,#UART_User        //串口号
```

```
    bl  uart_send_string    //向原串口回发
    b Light_Control_exit          //break
Light_Control_case_5:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#5
    bne Light_Control_case_6          //不等于5转判断6
    //等于5的情况: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       //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
    b Light_Control_exit          //break
Light_Control_case_6:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#6
    bne Light_Control_case_7          //不等于6转判断7
    //等于6的情况: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       //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
    b Light_Control_exit          //break
Light_Control_case_7:
    ldr r0,=mLightCommand
    ldr r0,[r0]
    cmp r0,#7
    bne Light_Control_exit          //不等于1转判断2
    //等于7的情况:RED+BLUE+GREEN
    ldr r0,=LIGHT_RED
    ldr r1,=LIGHT_ON
    bl  gpio_set
    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_white_on_string    //r1指明字符串
    mov r0,#UART_User        //串口号
    bl  uart_send_string    //向原串口回发
Light_Control_exit:
    pop  {r0-r7,pc}

//=========================================================================
//程序名称：Light_Connected
//参数说明：无
//返回值说明：无
//程序功能：点亮蓝灯，连接成功后调用
//=========================================================================
Light_Connected:
    push {r0-r7,lr}
    //首先关掉所有的灯
    ldr r0,=LIGHT_RED //灭红灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_GREEN //灭绿灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_BLUE //灭蓝灯
    ldr r1,=LIGHT_OFF
    bl  gpio_set
    ldr r0,=LIGHT_BLUE //亮蓝灯
    ldr r1,=LIGHT_ON
    bl  gpio_set
Light_Connected_exit:
    pop  {r0-r7,pc}

//main.s
//（0）数据段与代码段的定义
.section .data
print_command_string:
    .string "Command #%d: %d\r\n"    //打印指令信息
success_command_string:
    .string "Command executed!\r\n"      //串口返回的信息（成功执行）
unknown_command_string:
    .string "Unknown command!\r\n"       //串口返回的信息（未知指令）
.global print_command_string
.global success_command_string
```

```
.global unknown_command_string
//灯亮状态输出字符串
light_red_on_string:
    .string "[RED LED ON] "
.global light_red_on_string
light_blue_on_string:
    .string "[BLUE LED ON] "
.global light_blue_on_string
light_green_on_string:
    .string "[GREEN LED ON] "
.global light_green_on_string
light_cyan_on_string:
    .string "[CYAN LED ON] "
.global light_cyan_on_string
light_purple_on_string:
    .string "[PURPLE LED ON] "
.global light_purple_on_string
light_yellow_on_string:
    .string "[YELLOW LED ON] "
.global light_yellow_on_string
light_white_on_string:
    .string "[WHITE LED ON] "
.global light_white_on_string
light_off_string:
    .string "[LED OFF] "
.global light_off_string

//握手字符串
handshake_check_str:
    .string "auart?"
.global handshake_check_str
handshake_send_str:
    .string "I am an uart"
.global handshake_send_str

// (0.1.2) 定义变量
.align 4              //.word格式四字节对齐
mLightCommand:        //定义灯控制命令
    .word 0
.global mLightCommand

.equ MainLoopNUM,10000000  //主循环次数设定值（常量）

// (0.2) 定义代码存储text段开始，实际代码存储在Flash中
.section   .text
```

```
.syntax unified                 //指示下方指令为ARM和thumb通用格式
.thumb                          //Thumb指令集
.type main function               //声明main为函数类型
.global main                       //将main定义成全局函数，便于芯片初始化之后调
用
.align 2                          //指令和数据采用2字节对齐，兼容Thumb指令集
//主函数，一般情况下可以认为程序从此开始运行（实际上有启动过程，参见书稿）
main:
//（1.2）【不变】关总中断
    cpsid i
//（1.5）用户外设模块初始化
//  初始化LED灯，r0、r1、r2是gpio_init的入口参数
    ldr r0,=LIGHT_BLUE     //r0指明端口和引脚
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_ON       //r2指明引脚的初始状态为亮
    bl  gpio_init          //调用gpio初始化函数
    ldr r0,=LIGHT_RED     //r0指明端口和引脚
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_ON       //r2指明引脚的初始状态为亮
    bl  gpio_init          //调用gpio初始化函数
    ldr r0,=LIGHT_GREEN     //r0指明端口和引脚
    mov r1,#GPIO_OUTPUT    //r1指明引脚方向为输出
    mov r2,#LIGHT_ON       //r2指明引脚的初始状态为亮
    bl  gpio_init          //调用gpio初始化函数
//  初始化串口UARTA
    mov r0,#UARTA              //r0=更新串口
    ldr r1,=UART_BAUD
    bl  uart_init
//（1.6）使能模块中断
    mov r0,#UARTA              //r0指明串口号
    bl  uart_enable_re_int     //调用串口中断使能函数
//（1.7）【不变】开总中断
    cpsie  i
    ldr r0,=instruction
    bl printf
main_loop:                     //主循环标签（开头）
    b main_loop                //死循环
.end        //整个程序结束标志（结尾）

//C#发送指令程序
private void sendData(byte cmd)
{
    byte[] SendByteArray = new byte[1];          // 定义发送缓冲区
    try
    {
```

```
            SendByteArray[0] = cmd;
            sci.SCISendFrameData(ref SendByteArray);
            Thread.Sleep(800);
            if (sci.SCIReceiveData(ref recvData))
            {
                this.Txt_recv2.Text += Encoding.Default.GetString(recvData);
            }
        }
        catch
        {
            this.Txt_recv2.Text += "操作失败！\n";
        }
        this.Txt_recv2.SelectionStart = Txt_recv2.Text.Length;
        this.Txt_recv2.ScrollToCaret();
        this.Txt_recv2.Refresh();
    }
```

五. 实践性问答题

（1）波特率 9600bps 和 115200bps 的区别是什么?

波特率是每秒传送的数据位数。9600bit/s 的波特率比起 115200bps 单位时间传送的数据位数少，因此数据传送更慢，但相对传输距离更长。

（2）有什么最简单的方法知道 GEC 串口的 TX 发送了信号?

TX 的功能是发送数据到 PC 机。从 PC 机发送一个字节时，会触发中断，在 MCU 通过 Rx 接收到数据之后，立刻通过 TX 将当前接收到的内容回发。我们可以观察发送的接收到的内容是否一致来判断 TX 发送是否正常和正确。

（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。

不需要。因为 MCU 的引脚的输入输出一般都是使用 TTL 电平，而转换芯片只在内部起到电平转换的作用，因此进行 MCU 的串口通信接口编程的时候只针对 MCU 的发送和接收引脚，与电平转换芯片无关。

（4）不用其他工具，如何测试发送一个字符的真实时间?

可以发送一个较长的字符串，并记录其长度以及发送首个字符和末尾字符的系统时间，用时间差除以字符串长度即可。