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 21 日	

实验名称：实验五：理解中断与定时器

一. 实验目的

（1）熟悉定时中断计时的工作及编程方法。

（2）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“Exam8_1”工程。该程序通过 SYSTICK 定时器，每一秒将红灯亮暗状态反转一次并输出相应的时间和提示信息。

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，利用该程序框架实现：通过集成开发环境 AHL-GEC-IDE 中的“串口-串口工具”，发送当前系统时间如：“10:55:12”来设置开发板上的初始计时时间。

（3）进阶实验★

复制样例程序，利用该程序框架实现：通过 C#程序界面显示系统当前时间，通过按钮发送当前系统时间给 MCU 来设置开发板上的初始计时时间，MCU 将计时时间发送给 C#进行显示。C#界面设计如图 8-4 所示。不断电运行一段时间后，请给出 MCU 和 PC 系统时间的误差是多少？

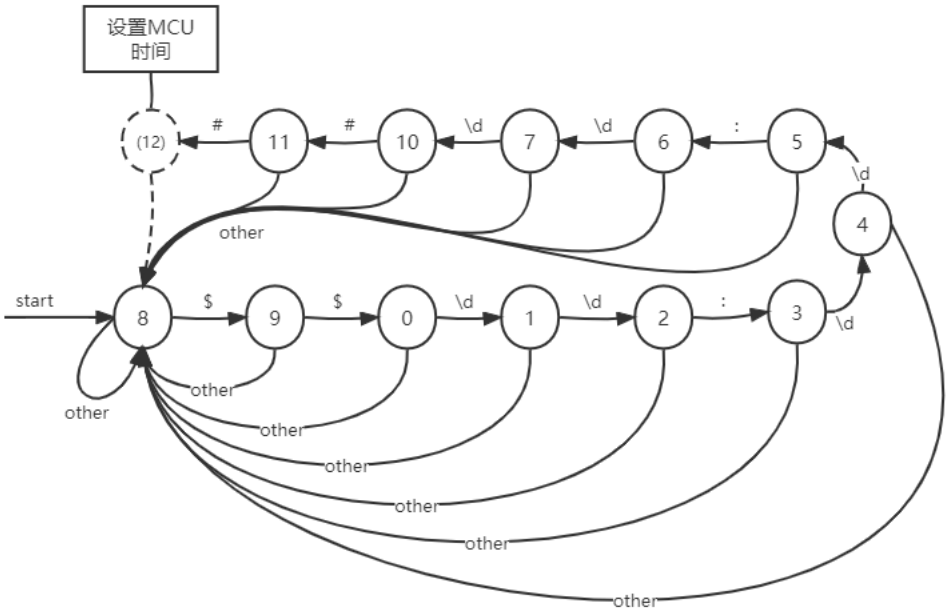
请在实验报告中给出 MCU 端程序 main.c 和 isr.c 流程图及程序语句和 C#方主要程序段。

图 1　C#界面设计

四、实验结果

（1）用适当文字、图表描述实验过程。

设计性实验和进阶实验使用同一套汇编程序。

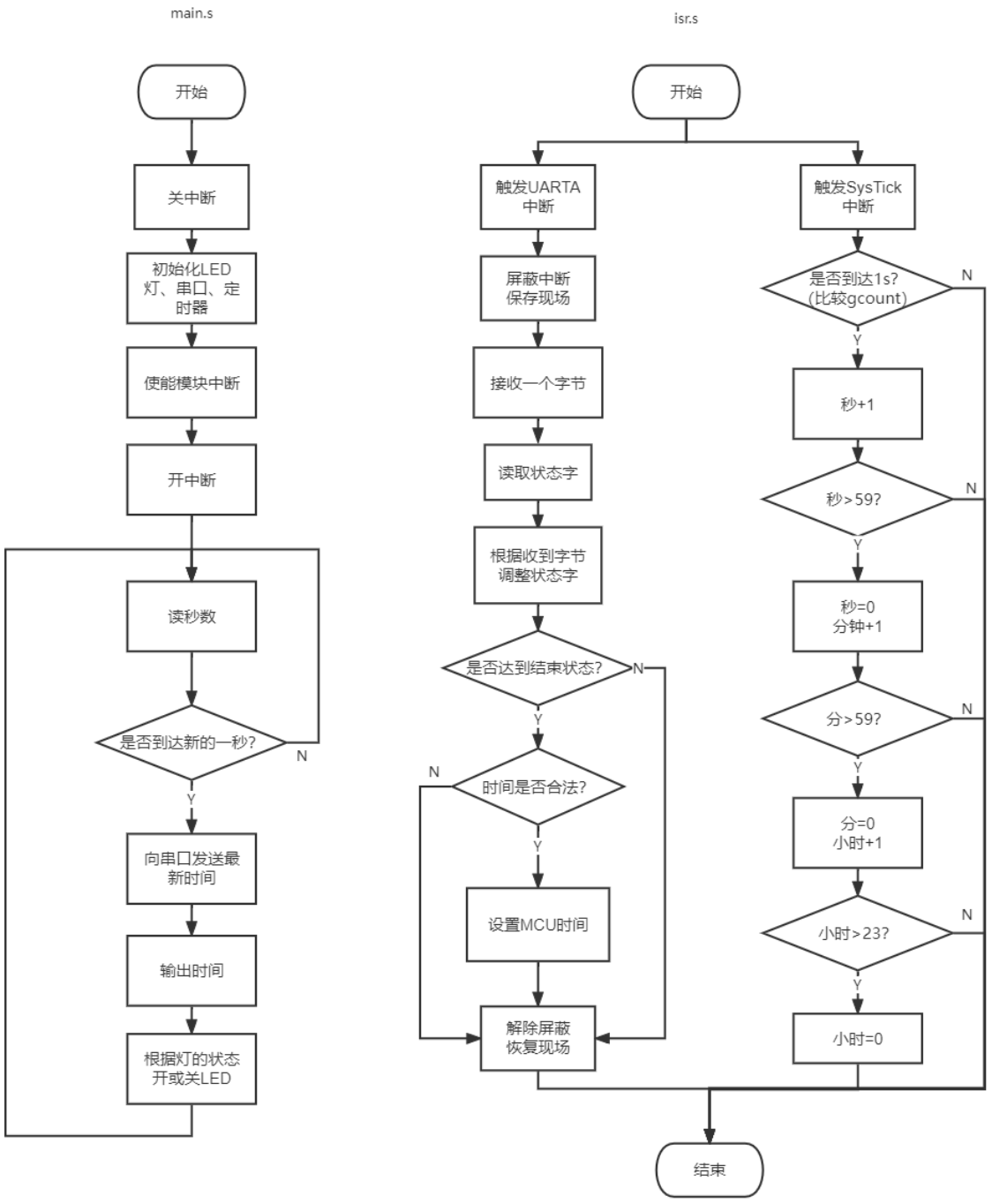


约定 PC 机向 MCU 发送一个形如 **$$HH:mm:ss##** 的字符串来设置 MCU 的时间，其中时间以两位数显示，**$$**为帧头，**##**为帧尾。在 MCU 端使用一个自动机来接收该字符串，维护一个全局状态

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该时间字符串，并实时保存接收到的新的时间。若成功接收 PC 机发来的时间，MCU 先判断时间是否合法，若合法则修改 MCU 的时间。自动机如上页图所示（虚线状态不会停留，实际实现中并不存在该状态，只是为了更好地解释）。

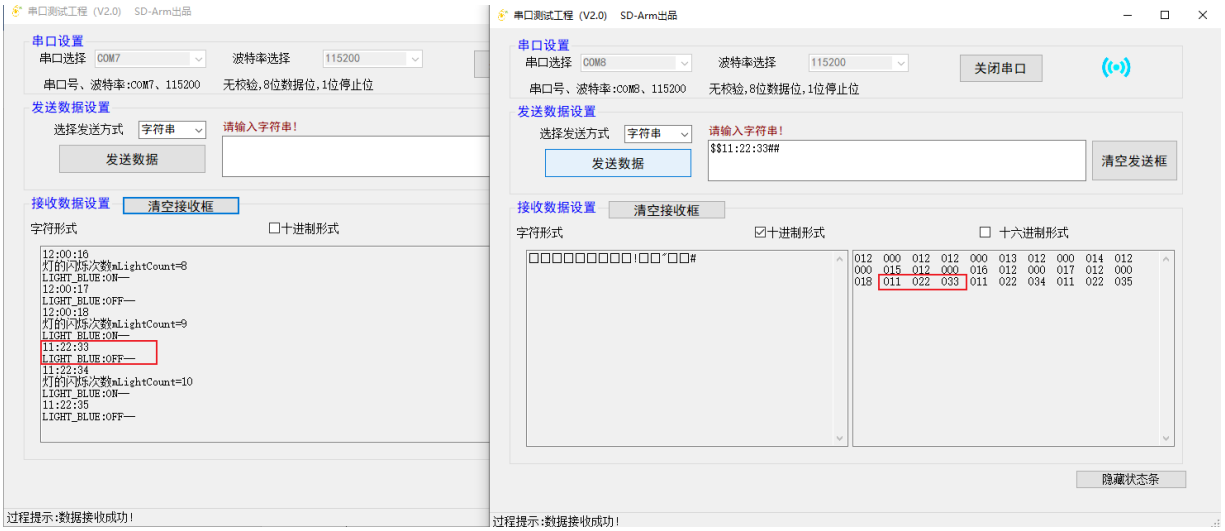
样例程序中 main 函数会在秒数更新时闪烁 LED 灯，为满足进阶实验的要求，并同时更好地调试程序，本实验中与此同时向串口依次发送三个字节，分别表示 MCU 时间中的小时、分钟、秒数（均为数字）。

上述流程中流程图如下：



教务处制

设计性实验结果：打开两个串口工具，分别连接至 MCU 的两个串口（COM7 和 COM8），向 COM8 串口发送字符串：**$$11:22:33##**，观察接收到的数据（COM8 需打开十进制输出框），可见 MCU 成功设置了时间：



进阶实验：使用自己编写的 C#程序连接到串口并发送上述指令，C#程序基于串口通信工具源码开发。首先点击打开串口连接 MCU，发送时间。不断电运行 35 分钟时间后，可以看到 MCU 和 PC 系统时间的误差约为 22 秒。

教务处制

（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
//省略函数定义区域
.section .text
//=========================================================================
//程序名称：UARTA_Handler（UARTA接收中断处理程序）
//触发条件：收到一个字节触发
//程序功能：接收PC机下发的时间
//=========================================================================
USART2_IRQHandler:
    //（1）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
    //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
    ldr r1,=received_data
    str r0,[r1]        //保存接收到的数据
    //分状态处理
    ldr r3,=recv_state
    ldr r2,[r3]
    cmp r2,#0         //State 0: num->1, other->8
    beq USART2_IRQHandler_state0
    cmp r2,#1         //State 1: num->2, other->8
    beq USART2_IRQHandler_state1
    cmp r2,#2         //State 2: ':'->3, other->8
    beq USART2_IRQHandler_state2
    cmp r2,#3         //State 3: num->4, other->8
    beq USART2_IRQHandler_state3
    cmp r2,#4         //State 4: num->5, other->8
    beq USART2_IRQHandler_state4
    cmp r2,#5         //State 5: ':'->6, other->8
    beq USART2_IRQHandler_state5
    cmp r2,#6         //State 6: num->7, other->8
    beq USART2_IRQHandler_state6
    cmp r2,#7         //State 7: num->10, other->8
    beq USART2_IRQHandler_state7
```

```
    cmp r2,#8        //State 8:[Start] $->9, other->8
    beq USART2_IRQHandler_state8
    cmp r2,#9        //State 9: $->0, other->8
    beq USART2_IRQHandler_state9
    cmp r2,#10        //State 10: #->11, other->8
    beq USART2_IRQHandler_state10
    cmp r2,#11        //State 11: #->set_new_time->8, other->8
    beq USART2_IRQHandler_state11
    bl  USART2_IRQHandler_exit
//Hour
USART2_IRQHandler_state0:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state0_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state0_other
    //收到字符0-9
    sub r0,r0,#'0'    //转为数字0-9
    ldr r1,=new_hour  //存入new_hour
    str r0,[r1]
    mov r0,#1         //转1
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state0_other:
    mov r0,#8         //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state1:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state1_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state1_other
    //收到字符0-9
    sub r0,r0,#'0'    //转为数字0-9
    ldr r3,=new_hour  //new_hour=new_hour*10+r0
    ldr r1,[r3]
```

```
    mov r2,#10
    mul r1,r2,r1
    add r0,r0,r1
    str r0,[r3]
    mov r0,#2          //转2
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state1_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state2:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#':'
    bne USART2_IRQHandler_state2_other
    //收到冒号
    mov r0,#3          //转3
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state2_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
//Minute
USART2_IRQHandler_state3:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state3_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state3_other
    //收到字符0-9
    sub r0,r0,#'0'     //转为数字0-9
    ldr r1,=new_minute//存入new_minute
    str r0,[r1]
    mov r0,#4          //转4
```


教务处制

```
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state3_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state4:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state4_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state4_other
    //收到字符0-9
    sub r0,r0,#'0'     //转为数字0-9
    ldr r3,=new_minute//new_minute=new_minute*10+r0
    ldr r1,[r3]
    mov r2,#10
    mul r1,r2,r1
    add r0,r0,r1
    str r0,[r3]
    mov r0,#5          //转5
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state4_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state5:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#':'
    bne USART2_IRQHandler_state5_other
    //收到冒号
    mov r0,#6          //转6
    ldr r1,=recv_state
    str r0,[r1]
```

```
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state5_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
//Second
USART2_IRQHandler_state6:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state6_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state6_other
    //收到字符0-9
    sub r0,r0,#'0'     //转为数字0-9
    ldr r1,=new_second//new_second
    str r0,[r1]
    mov r0,#7          //转7
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state6_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state7:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'0'
    blo USART2_IRQHandler_state7_other
    cmp r0,#'9'
    bhi USART2_IRQHandler_state7_other
    //收到字符0-9
    sub r0,r0,#'0'     //转为数字0-9
    ldr r3,=new_second//new_second=new_second*10+r0
    ldr r1,[r3]
    mov r2,#10
    mul r1,r2,r1
    add r0,r0,r1
```

```
    str r0,[r3]
    mov r0,#10          //转10
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state7_other:
    mov r0,#8           //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state8:
    //重置接收到的时间
    mov r0,#0
    ldr r1,=new_hour
    str r0,[r1]
    ldr r1,=new_minute
    str r0,[r1]
    ldr r1,=new_second
    str r0,[r1]
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    cmp r0,#'$'
    bne USART2_IRQHandler_state8_other
    //收到$
    mov r0,#9           //转9
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state8_other:
    mov r0,#8           //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state9:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    //收到$
    cmp r0,#'$'
    bne USART2_IRQHandler_state9_other
    mov r0,#0           //转0
    ldr r1,=recv_state
```

```
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state9_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state10:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    //收到#
    cmp r0,#'#'
    bne USART2_IRQHandler_state10_other
    mov r0,#11         //转11
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state10_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    bl  USART2_IRQHandler_exit
USART2_IRQHandler_state11:
    //处理收到的字节
    ldr r0,=received_data
    ldr r0,[r0]
    //收到#
    cmp r0,#'#'
    bne USART2_IRQHandler_state11_other
    //set_time:
    ldr r0,=new_hour
    ldr r0,[r0]
    cmp r0,#23     //新小时>23
    bhi USART2_IRQHandler_state11_other
    ldr r1,=new_minute
    ldr r1,[r1]
    cmp r1,#59     //新分钟>59
    bhi USART2_IRQHandler_state11_other
    ldr r2,=new_second
    ldr r2,[r2]
    cmp r2,#59     //新秒数>59
    bhi USART2_IRQHandler_state11_other
```

```
    //只有正确时间格式才存入
    ldr r3,=hour
    str r0,[r3]
    ldr r3,=minute
    str r1,[r3]
    ldr r3,=second
    str r2,[r3]
    mov r0,#60
    ldr r1,=last_second
    str r0,[r1]
    ldr r1,=gcount
    mov r0,#0
    str r0,[r1]
USART2_IRQHandler_state11_other:
    mov r0,#8          //转8
    ldr r1,=recv_state
    str r0,[r1]
    //（4）解除屏蔽，并且恢复现场
USART2_IRQHandler_exit:
    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

//=====================================================================
//函数名称：SysTick_Handler
//参数说明：无
//函数返回：无
//功能概要：SysTick定时器中断服务例程
//=====================================================================
SysTick_Handler:
    push {r0,r1,r2,lr}
    ldr r0,=gcount
    ldr r1,[r0]
    cmp r1,#50
    bcs SysTick_Handler_1s
    add r1,#1
    str r1,[r0]
    b   SysTick_Handler_Exit
SysTick_Handler_1s:
    mov r1,#0
    str r1,[r0]
    b   add_sec
```

```
SysTick_Handler_Exit:
    pop {r0,r1,r2,pc}


add_sec:
    ldr r0,=second
    ldr r1,[r0]
    cmp r1,#59
    bcs sec60  //秒>=59
    add r1,#1
    str r1,[r0]//秒<59
    b   add_sec_exit
sec60:         //秒>=59
    mov r1,#0
    str r1,[r0]
    ldr r0,=minute
    ldr r1,[r0]
    cmp r1,#59
    bcs min60  //分>=59
    add r1,#1
    str r1,[r0]//分<59
    b   add_sec_exit
min60:         //分>=59
    mov r1,#0
    str r1,[r0]
    ldr r0,=hour
    ldr r1,[r0]
    cmp r1,#23
    bcs hour23  //时>=23
    add r1,#1
    str r1,[r0]//时<23
    b   add_sec_exit
hour23:
    mov r1,#0
    str r1,[r0]
add_sec_exit:
    b   SysTick_Handler_Exit


//main.s
//全局变量声明
.global gcount  //50次计数单元
.global hour    //时
```

```
.global minute  //分
.global second  //秒
.global data_format
.global new_hour    //新时
.global new_minute  //新分
.global new_second  //新秒
.global received_data
.global recv_state
.global last_second
//省略部分字段定义
.align 4
mLightCount:
    .word 0
gcount:      //50次计数单元
    .word 0
hour:        //时
    .word 12
minute:      //分
    .word 0
second:      //秒
    .word 0
last_second:  //上一次记录的秒
    .word 0
received_data:  //串口接受到的数据
    .word 0
recv_state:        //接收状态
    .word 8
new_hour:        //新的时
    .word 12
new_minute:      //新的分
    .word 0
new_second:      //新的秒
    .word 0
//……
//主函数
main:
//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
//  systick定时器初始化
    mov r0,#20
    bl systick_init//每20毫秒中断一次
//  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
//开总中断
    cpsie  i
main_loop:                     //主循环标签（开头）
        ldr r0,=second
        ldr r0,[r0]              //读取秒数
        ldr r1,=last_second
        ldr r1,[r1]              //读取上一次记录的秒
        cmp r0,r1                //判断是否是新的一秒
        beq main_loop            //若未到新的一秒，继续循环
        ldr r1,=last_second
        str r0,[r1]              //否则，更新记录的秒
        //当秒数更新时，向串口发送最新时间（进阶实验）
        ldr r2,=hour
        ldr r1,[r2]
        mov r0,#UARTA
        bl  uart_send1
        ldr r2,=minute
        ldr r1,[r2]
        mov r0,#UARTA
        bl  uart_send1
        ldr r2,=second
        ldr r1,[r2]
        mov r0,#UARTA
        bl  uart_send1
        //输出时间
        ldr r2,[r2]
        ldr r1,=hour
        ldr r1,[r1]
        cmp r1,#10             //读取小时数，判断位数
        bcs hour_two_bits
        ldr r0,=time_show0      //若为一位，补足一个0
        bl  printf
```

```
hour_two_bits:
        ldr r0,=time_data_format
        ldr r1,=hour
        ldr r1,[r1]
        bl  printf              //输出小时数
        ldr r0,=time_show1
        bl  printf              //输出:
        ldr r1,=minute
        ldr r1,[r1]
        cmp r1,#10
        bcs minute_two_bits     //读取分钟数判断位数
        ldr r0,=time_show0      //若为一位，补足一个0
        bl  printf
minute_two_bits:
        ldr r0,=time_data_format
        ldr r1,=minute
        ldr r1,[r1]
        bl  printf           //输出分钟数
        ldr r0,=time_show1
        bl  printf              //输出:
        ldr r1,=second
        ldr r1,[r1]
        cmp r1,#10
        bcs second_two_bits     //读取秒数判断位数
        ldr r0,=time_show0
        bl  printf            //若为一位，补足一个0
second_two_bits:
        ldr r0,=time_data_format
        ldr r1,=second
        ldr r1,[r1]
        bl  printf          //输出分钟数
        ldr r0,=time_show2
        bl  printf            //输出换行符
//（2.3.2）如灯状态标志mFlag为'L'，灯的闪烁次数+1并显示，改变灯状态及标志
        //判断灯的状态标志
        ldr r2,=mFlag
        ldr r6,[r2]
        cmp r6,#'L'
        bne main_light_off  //mFlag不等于'L'转
        //mFlag等于'L'情况
        ldr r3,=mLightCount     //灯的闪烁次数mLightCount+1
        ldr r1,[r3]
        add r1,#1
```

```
        str r1,[r3]
        ldr r0,=light_show3   //显示“灯的闪烁次数mLightCount=”
        bl  printf
        ldr r0,=data_format    //显示灯的闪烁次数值
        ldr r2,=mLightCount
        ldr r1,[r2]
        bl  printf
        ldr r2,=mFlag          //灯的状态标志改为'A'
        mov r7,#'A'
        str r7,[r2]
        ldr r0,=LIGHT_BLUE     //亮灯
        ldr r1,=LIGHT_ON
        bl  gpio_set
        ldr r0, =light_show1    //显示灯亮提示
        bl  printf
        //mFlag等于'L'情况处理完毕，转
        b main_exit
//（2.3.3）如灯状态标志mFlag为'A'，改变灯状态及标志
main_light_off:
        ldr r2,=mFlag       //灯的状态标志改为'L'
        mov r7,#'L'
        str r7,[r2]
        ldr r0,=LIGHT_BLUE     //暗灯
        ldr r1,=LIGHT_OFF
        bl  gpio_set
        ldr r0, =light_show2    //显示灯暗提示
        bl  printf
 main_exit:
    b main_loop                 //继续循环
.end         //整个程序结束标志（结尾）
```

```
//进阶实验 C#主要程序段
/// <summary>
/// 发送时间按钮点击事件，向串口发送本机时间，格式为$$HH:mm:ss##
/// </summary>
private void send_button_Click(object sender, EventArgs e)
{
    if (!SCIPort.IsOpen)
    {
        //状态条进行提示
        program_info_text.Text += "请先打开串口!";
        return;
    }
```


教务处制

```
    string sendString = DateTime.Now.ToString("$$HH:mm:ss##");
    program_info_text.Text = "发送: " + sendString;
    //将要发送的数据进行编码,并获取编码后的数据长度
    byte[] SendByteArray = Encoding.Default.GetBytes(sendString);
    bool Flag = sci.SCISendData(SCIPort, ref SendByteArray);
    if (Flag == true)
    {
        program_info_text.Text += " 成功!";
    }
    else
    {
        program_info_text.Text += " 失败!";
    }
}
/// <summary>
/// 定时器Tick事件，用于定时更新GUI中的系统时间
/// </summary>
private void timer_sys_time_Tick(object sender, EventArgs e)
{
    DateTime dt = DateTime.Now;
    string time = dt.ToString("HH:mm:ss");
    sys_time_text.Text = time;
    sys_time_text.Refresh();
}
/// <summary>
/// 串口收到数据事件，MCU每隔1s会将MCU时间通过串口发送给PC机。
/// 格式为三个字节，分别表示小时、分钟、秒（均为数字）。
/// PC机收到数据后判断长度是否为3，并将其转换为HH:mm:ss的字符串显示在对应位置
/// </summary>
private void SCIPort_DataReceived(object sender,
                          System.IO.Ports.SerialDataReceivedEventArgs e)
{
    //如果正在执行关闭操作
    if (closing)
    {
        return;
    }
    listening = true;
    byte[] recvData = new byte[1024];
    bool Flag = sci.SCIReceiveData(SCIPort, ref recvData);
    if (Flag == true)//串口接收数据成功
    {
        int len = recvData.Length;
```

```
        if (len == 3)
        {
            mcu_time_text.Clear();
            for (int i = 0; i < len; i++)
            {
                //异步更新TextBox
                SCIUpdateRevtxtbox(mcu_time_text,
                                   recvData[i].ToString("D2"));
                if (i < len - 1)
                {
                    SCIUpdateRevtxtbox(mcu_time_text, ":");
                }
            }
        }
    }
    listening = false;  //接收数据结束
}
```

五. 实践性问答题

（1）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。

（2）请简述定时器中断的基本原理，定时器是如何实现计时的?

Arm Cortex-M4F 内核中包含了一个简单的定时器 SysTick，又称为“滴答”定时器。SysTick 定时器被捆绑在 NVIC（嵌套向量中断控制器）中，有效位数是 24 位，采用减 1 计数的方式工作，当减 1 计数到 0，可产生 SysTick 中断，中断号为 15。

实现计时的过程中，可以根据需要的定时时间，用指令对定时器设置定时常数，并用指令启动定时器开始计数，当计数到指定值时，便自动产生一个定时输出，或中断信号告知 CPU。在定时器开始工作以后，CPU 不必去管它，而可以去做其他工作。

（3）Timer 中断最小定时时间是多少?

Arm Cortex-M4F 的主频是 72M 的，按照 T=(arr+1)*(PSC+1)/TCK，其中 TCK 为时钟频率，PSC 为时钟预分频系数，arr 为自动重装载值。最小的定时时间是可以达到 0.06μs，但是指令执行需要时间，并且中断跳转也需要占用时间，实际为 0.1μs，但这个时间很不精确，波动很大，反而更大的时间 1μs 是相对可以基本保障的，所以实际经常使用的是 1μs。