上海交通大學

《工程实践与科技创新Ⅱ-A》课程

学生实验报告

实验名称：UART 串行通讯口实验

姓　　名：彭俊杰

学　　号：521021910404

班　　级：ICE0501-5

任课老师：朱兰娟、邵群

2023 年 6 月 4 日

目录

1 实验目的

了解 UART 串行通讯的工作原理；
掌握在 PC 端使用串口调试工具（SerialProV1.04.exe）与实验板进行 UART 通讯的方法；
掌握 UART 的堵塞式与非堵塞式数据传输编程方法；
掌握中断优先级设置方法及对程序运行的影响。

2 实验原理

2.1 硬件原理分析

2.1.1 通用异步收发器 UART

计算机与外部设备连接有两类连接接口：并行接口是指数据的各个位同时进行传送，传输速度快，但当传输距离远、位数多时，通信线路变复杂且成本提高；串行接口是指数据一位一位地顺序传送，通信线路简单，在远距离通信时可以大大降低通信成本。串行通信分为异步串行通信 ASYNC（Asynchronous Data Communication）和同步串行通信 SYNC（Synchronous Data Communication）。

UART（Universal Asynchronous Receiver/Transmitter，通用异步收发器）是设备间进行异步串行通信的关键模块。TM4C1294NCPDT 的 UART 模块功能图如下：

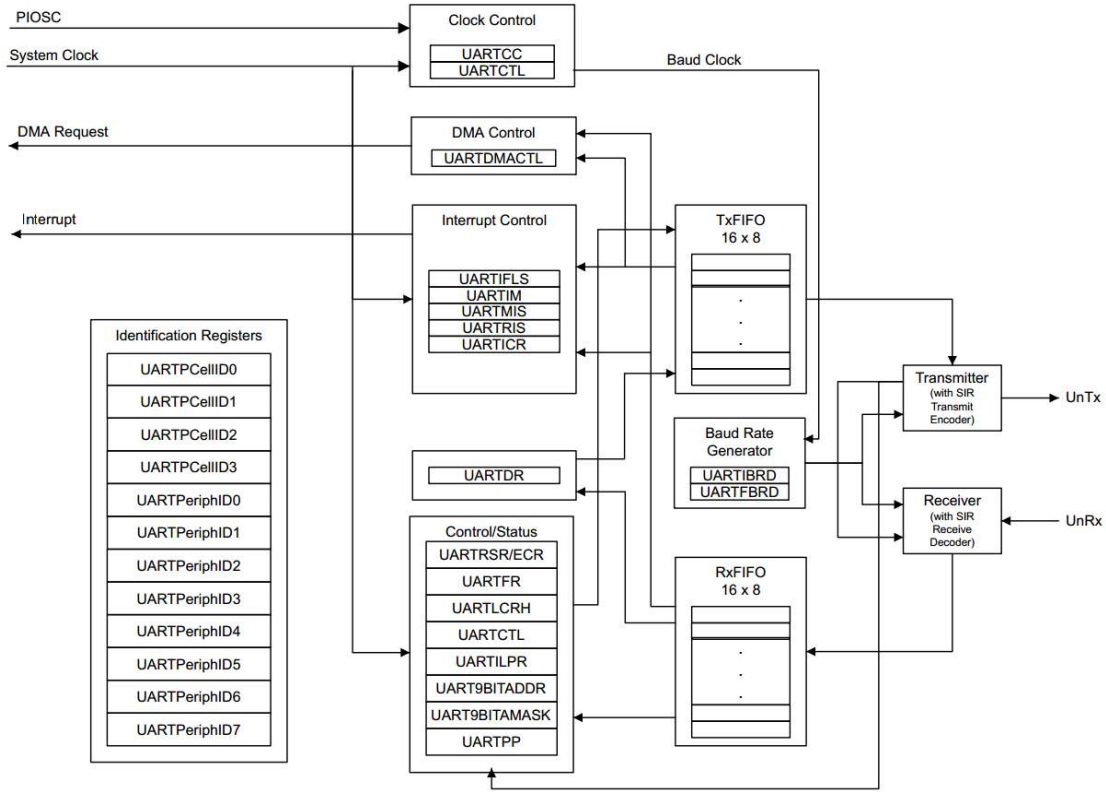

图 1 UART 模块功能框图

其中，TxFIFO 用于发送数据，RxFIFO 用于接收数据，transmitter 是发送器，Receiver 是接收器。

2.1.2 UART 常用电路接口

UART 只定义了串行通讯的逻辑层，即字符传送的格式与速率，但对于利用 UART 进行传输的物理电平接口需要另外定义，因此根据接口电平的不同与控制方式的不同，产生了 RS-232 与 RS-485 接口标准。

RS-232 是 PC 机与设备通信里应用最广泛的一种串行接口。它被定义为一种在低速率串行通讯中增加通讯距离的单端标准。由于其最大通信距离的限制，因此它常用于本地设备之间的通信。

RS232 标准定义逻辑“1”信号相对于地为-3V~ -15V，而逻辑“0”相对于地为+3V~+15V。所以，当一个微控制器中的 UART 相连于 PC 时，它需要一个 RS232 驱动器来转换电平。最高传输率 20Kb/s，最大传输距离约为 15m，具有单端传输（点对点）的特点，共模抑制比低。

RS-485 是一种常用于远距离和多机通信的串行接口。半双工工作方式，易于多点互连或联网成分布式系统。RS-485 只是定义电压和阻抗，与 TTL 电平兼容，最高传输速率 10Mb/s，传输距离可达 1200m，具有双端传输的特点，共模抑制比高。需要 RS485 驱动器来将单端信号转换为双端信号。

2.1.3 UART 调试工具的使用

安装好驱动后，S800 实验板通过 MICRO-USB 线与 PC 机连接，正常情况下打开电脑的设备管 理器，会至少看到以下两个设备，一是 ICDI 在线调试工具，二是虚拟串口（COM 口），虚拟串口可以用来当作 MCU 的 UART 通讯。

发送器对 TxFIFO 中的数据执行“并→串”转换，按字符帧格式输出串行比特流，按照“先进先出”的原则将 TxFIFO 中的数据一个个发送出去，直到 TxFIFO 全空为止。

接收器在检测到一个有效的起始脉冲后，对接收到的比特流执行“串→并”转换和 错误检测，并写入 RxFIFO，当硬件逻辑接收到数据，就会往 RxFIFO 里填充接收到的数据，直到 RxFIFO 满。

UART 发送的字符帧格式如下图所示：

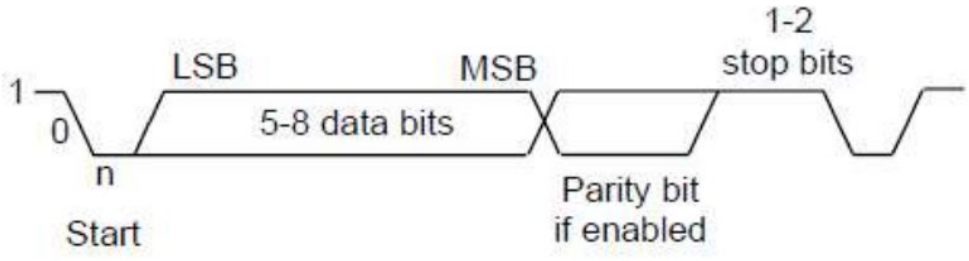

图 2　UART 字符帧格式图

要想和 S800 之间进行通讯，我们还需要串口调试工具，这里我们使用到的是 SerialProV1.04。

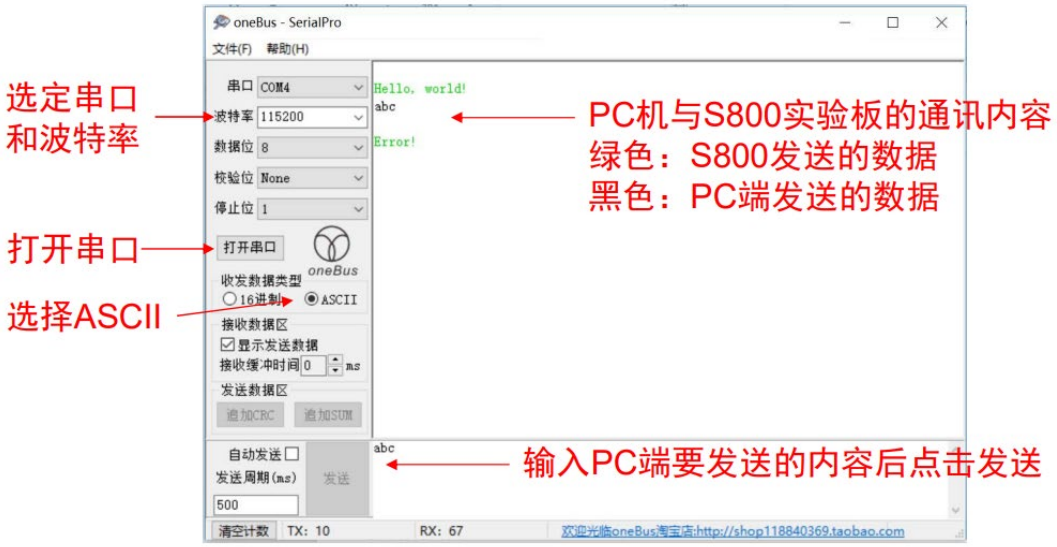

图 3 SerialProV1.04 使用图

2.2 软件算法分析（含程序流程图和源代码）

2.2.1 UART 中断流程

以例程 3-2 为例， UART 中断流程图以及 UART_Handler 代码如下：

图 4 UART 中断流程图

```
1.  void UART0_Handler(void)
2.  {
3.      int32_t uart0_int_status;
4.  //  char uart_receive_char;
5.
6.    uart0_int_status = UARTIntStatus(UART0_BASE, true);          // Ge
    t the interrrupt status.
```

```
7.    UARTIntClear(UART0_BASE, uart0_int_status);
       //Clear the asserted interrupts
8.
9.      if (uart0_int_status & (UART_INT_RX | UART_INT_RT))      //接收或接
   收超时
10.     {
11.         uart_receive_status = 1;
12.     }
13. }
```

UARTIntStatus（）对中断状态进行读取，如若有中断，uart0_int_status 置 1，在此基础上，如果由外部信号输入，则读取外部信息，这部分依靠 UARTgetchar()在 main 函数部分实现。

2.2.2 UART 初始化过程

UART 初始化过程大体分为以下四个部分：

1、 使能 UART 模块
2、 使能提供 UART 接口信号的 GPIO 端口
3、 将 GPIO 引脚设置为 UART 复用功能
4、 配置并使能 UART 通讯（包括通讯格式和 FIFO 深度等）

```
1.  void S800_UART_Init(void)
2.  {
3.      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
4.    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);                    /
   /Enable PortA
5.      while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));         //Wai
   t for the GPIO moduleA ready
6.
7.      GPIOPinConfigure(GPIO_PA0_U0RX);
                  // Set GPIO A0 and A1 as UART pins.
8.    GPIOPinConfigure(GPIO_PA1_U0TX);
9.
10.   GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
11.
12.     // Configure the UART for 115,200, 8-N-1 operation.
13.   UARTConfigSetExpClk(UART0_BASE, ui32SysClock,115200,(UART_CONFIG_WL
   EN_8 | UART_CONFIG_STOP_ONE |UART_CONFIG_PAR_NONE));
14.
15.     UARTFIFOLevelSet(UART0_BASE,UART_FIFO_TX2_8,UART_FIFO_RX4_8);//se
   t FIFO Level
16.
```

```
17.   UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //Enable UART
    0 RX,TX interrupt
18.     IntEnable(INT_UART0);
19.
20.     UARTStringPut("\r\nHello, world!\r\n");
21. }
```

2.2.3 UART 的数据发送模式

UART 的数据发送模式有两种，分别为阻塞发送和非阻塞发送，具体代码如下所示：

阻塞发送，查询式数据发送，若 TxFIFO 没有空位，则等待直到发送成功。

```
1.  void UARTStringPut(const char *cMessage)
2.  {
3.      while(*cMessage!='\0')
4.          UARTCharPut(UART0_BASE,*(cMessage++));
5.  }
```

非阻塞发送，若 TxFIFO 有空位则发送，否则不发送直接返回。

```
1.  void UARTStringPutNonBlocking(const char *cMessage)
2.  {
3.      while(*cMessage!='\0')
4.          UARTCharPutNonBlocking(UART0_BASE,*(cMessage++));
5.  }
```

3 实验内容和步骤

3.1 实验要求 1

3.1.1 任务要求

Task1 完成 UART ECHO，即通过 PC 端串口调试窗口向实验板发送字符串，实验板收到后将收到的字符改为大写后返回。

3.1.2 任务实现

根据 ASCII 码大小写字符相差 32 的特性，判断字符的 ASCII 码，如果在 97（字符“a”）到 122（字符“z”）之间，减去 32 即可得到对应的大写字符。也可以通过 ctype.h 头文件，

调用函数 toupper()实现。

前者代码如下：

```
1.  //task1: uart echo
2.          if (UARTCharsAvail(UART0_BASE))
3.          {
4.            //Read the character from the UART and write it back to the UART.
5.              uart_receive_char = UARTCharGet(UART0_BASE);
6.
7.              //A=65
8.              if (uart_receive_char>=65 && uart_receive_char<=90)
9.              {
10.                 UARTCharPut(UART0_BASE, uart_receive_char);
11.             }
12.
13.             //a=97，小写变大写
14.             else if(uart_receive_char>=97 && uart_receive_char<=122)
15.             {
16.                 UARTCharPut(UART0_BASE, uart_receive_char-32);
17.             }
18.
19.             else
20.             {
21.                 UARTCharPut(UART0_BASE, uart_receive_char);
22.
23.             }
24.         }
```

效果图如下：

图 5　3-1 效果图

3.2 实验要求 2

3.2.1 任务要求

修改程序结构，将 Task1 移至中断处理程序中，说明两种结构各自的优势。

3.2.2 任务实现

将任务 1 代码移到中断服务程序内，代码如下所示：

```
1.  void UART0_Handler(void)
2.  {
3.      int32_t uart0_int_status;
4.  //  char uart_receive_char;
5.
6.    uart0_int_status = UARTIntStatus(UART0_BASE, true);
    // Get the interrrupt status.
7.    UARTIntClear(UART0_BASE, uart0_int_status);
       //Clear the asserted interrupts
8.
9.
10.     //Task1: uart echo
11.     if (uart0_int_status & (UART_INT_RX | UART_INT_RT))     //接收或接
    收超时
12.     {
13.         GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,GPIO_PIN_1 );
    //turn on PN1
14.
15.         while(UARTCharsAvail(UART0_BASE))
                    // Loop while there are characters in the receive FIFO
    .
16.         {
17.             //Read the next character from the UART and write it back
     to the UART.
18.             uart_receive_char = UARTCharGetNonBlocking(UART0_BASE);
19.             UARTCharPutNonBlocking(UART0_BASE,uart_receive_char);
20.         }
21.         UARTStringPut("\r\n");
22.         GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,0 );  //turn off PN1
23.
24.         uart_receive_status = 0;
25.     }
26. }
```

实验结果显示，当发送字符超过八个，会分为两行返回。这是因为中断条件中将接收FIFO 的触发深度设为了 4/8，也就是 8 位。每隔八位会中断一次，CPU 收取一次接收 FIFO 里的字符并将其送上发送 FIFO，最后送上回车和换行。最后一行不足八位的也能返回是因为设置了接收超时作为另一个中断条件。

实验结果上看，是否把任务 1 代码放入中断服务程序没有什么区别。但是从结构上看两种方法各有优势。如果将代码放入中断服务程序，可以保证中断时能执行到这一代码，否则在中断服务程序外可能受到其他代码的干扰；如果不将代码放入中断服务程序，前后台程序之间显得更合理，否则所有的工作都被中断服务程序完成，有喧宾夺主之感。

中断服务程序作为异常处理，应当简洁，尽量不要过多干扰系统正常运行。

3.3 实验要求 3

3.3.1 任务要求

编程实现一个虚拟 AT 指令集：

当 PC 端发来 AT+CLASS 后，实验板回以 CLASS######，其中#####为你的班级号

当 PC 端发来 AT+STUDENTCODE 后，实验板回以 CODE######，其中#####为你的学号。（要求大小写均能适应）

3.3.2 任务实现

第一步，当有 PC 端的数据输入时，检查字符串中的小写字母，转化为大写字母，将接受的字符存在 str 字符串中，详见代码 9-16 行。

第二步，利用 strncmp()函数，将接受的字符串和"AT+CLASS"、"AT+STUDENTCODE"相比较，并且返回相应的班级号或者学号；如果不能匹配，返回 UNKNOWN。

代码如下：

```
1.  while (1)
2.      {
3.          //Task1: uart echo
4.          if (uart_receive_status)
5.          {
6.              s = 0;
7.              GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,GPIO_PIN_1 );  /
    /turn on PN1
8.
9.              while(UARTCharsAvail(UART0_BASE))
                // Loop while there are characters in the receive FIFO.
10.             {
11.                 //Read the next character from the UART and write it
    back to the UART.
```

```
                uart_receive_char = UARTCharGetNonBlocking(UART0_BASE
);
                if (uart_receive_char <= 'z' && uart_receive_char >=
'a')
                    uart_receive_char -= 'a' - 'A'; //转换为大写
                str[s++] = uart_receive_char; // 接受的字符存至 str 字符
串
            }
            if (strncmp(str, "AT+CLASS", 8) == 0)
            {
                UARTStringPut("CLASSF2103203\r\n");
            }
            else if(strncmp(str, "AT+STUDENTCODE", 14) == 0)
            {
                UARTStringPut("STUDENT521021910404\r\n");
            }
            else
            {
                UARTStringPut("UNKNOWN\r\n");// 格式不符合要求
            }
            GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,0 );       //tur
n off PN1
            uart_receive_status = 0;
        }

    }//end while
```

效果图如下：

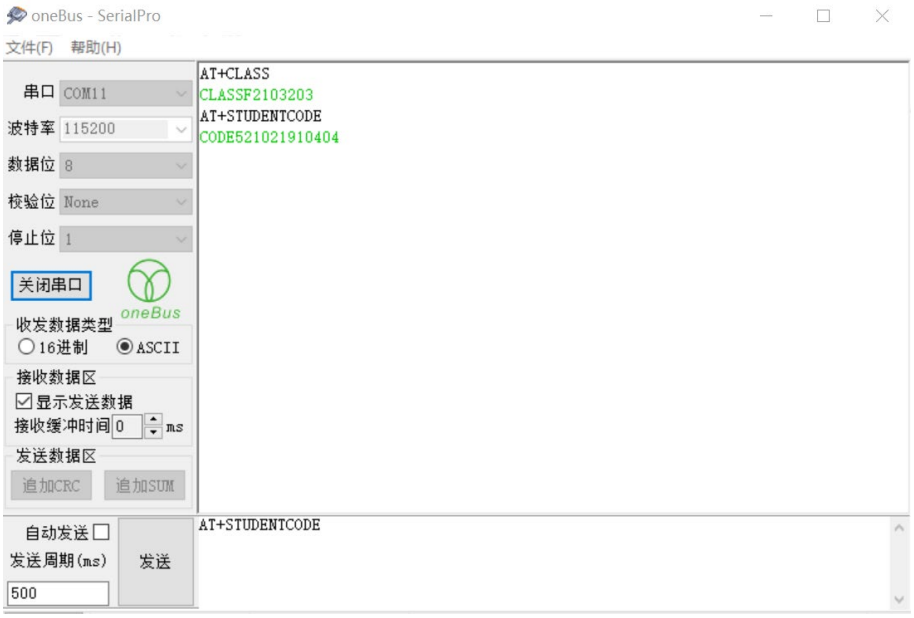

图 6　3-3 效果图

3.4 实验要求 4

3.4.1 任务要求

任务 1，主循环走马灯；
任务 2，Systick 中断中点亮 D1，长按 SW1 点亮 D2，释放熄灭 D2；
任务 3，UART0 中断中熄灭 D1 和 D2，长按 SW2 保持 UART 接收状态。

修改任务优先级，使 Systick 优先级大于或等于 UART0 优先级，或者分组设置为不抢占时，长按 USR_SW1 和 USR_SW2 分别会出现怎样的现象，观察并解释任务 1~3 的执行情况。

3.4.2 任务实现

在例程代码中，systick 中断优先级低于 UART0 中断。简单交换两者优先级就可以使得 systick 中断优先级高于 UART0 中断。

代码如下：

```
1.  //setup interrupt priority
2.  ui32IntPriorityMask = IntPriorityMaskGet();
3.
4.  IntPriorityGroupingSet(7);                //Set all priority to pre-
    emtption priority
5.  IntPrioritySet(INT_UART0,0xe0);              //Set INT_UART0 to lowest
     priority
6.  IntPrioritySet(FAULT_SYSTICK,0x3);       //Set INT_SYSTICK to highest
    priority
7.
8.  ui32IntPriorityGroup = IntPriorityGroupingGet();
9.  ui32IntPriorityUart0 = IntPriorityGet(INT_UART0);
10. ui32IntPrioritySystick = IntPriorityGet(FAULT_SYSTICK);
```

实验现象：不按任何按键时，D1 亮，走马灯正常运行，从 PC 端发送信息可以正常回应；长按 USR_SW1 时，D1 和 D2 亮，走马灯暂停，从 PC 端发送信息无回应，松开按键后 D2 灭，实验板发回了信息，但如果 PC 发送的信息超过十六位，实验板只能回应前十六位；长按 USR_SW2 时，D1 亮，当 PC 端发送信息后，实验板能正常回应，走马灯暂停。

解释：
长按 USR_SW1 时，程序陷入 systick 中断服务程序中，由于 systick 中断优先级高于 UART0 中断，因此 pc 发来数据使接收 fifo 达到设定深度或超时时不会触发 UART0 中断进行接收，更不会再发送回来，因此实验板无响应信息，task3 不会执行。同时由于程序在 systick 中断服务程序内，因此不会触发 task1 和下一个 systick 中断，故跑马灯暂停。松开按键后，程序在 systick 定时中断的间隔中可以进入 UART0 中断，接收到接收 fifo 内的字符并发回。

由于实验时，一般是在 PC 端发送完全部字符后再松开的按键，因此，十六位之后的字符因为接收 fifo 已满而丢失，所以实验板只能回应前十六位。

长按 USR_SW2 时，由于 systick 中断优先级高，程序每隔一个 systick 定时就进入一次 systick 中断，之后回到 UART0 中断，task2 能够正常运行，由于关闭 D1 的语句在 UART0 中断服务程序的 while 循环外，回到 UART0 中断直接回到 while 内而不会执行关闭操作，故 D1 表现出连续发光。其余时间程序都在 UART0 中断服务程序内，因此 task1 不可能执行，所以跑马灯暂停。由于程序大部分时间都在 UART0 接收状态，且收发函数也写在 UART0 中断服务程序 while 内，因此实验板能发送回字符。

3.5 实验要求 5

3.5.1 任务要求

请根据上位机的命令作应答。命令格式形如“MAY±01”。其中：

MAY 为月份，（JAN, FEB, …, DEC）均为三位。

+表示加运算符，-表示减运算符，占 1 位。加减均对月份进行模 12 运算。

01 表示增加或减少量，均为 2 位。范围 00~11。

如：MAY+01 应该回之以 JUN

MAY-06 应该回之以 NOV

3.5.2 任务实现

第一步，设置字符 uart_receive_char 保存 PC 端发出的字符。uart_receive_string[7]字符串保存上位机的命令，uart_receive_month[3]保存三位月份字符串；定义*month[]为月份的字符串数组，final_month 为计算出的应该显示的月份，uart_receive_number 为命令中的数字。

第二步，while(UARTCharsAvail(UART0_BASE))循环实现存放 PC 端发出的字符，将小写字母字符大写。其中，将“+”“-”改为空格是为了方便使用 sscanf()函数分割月份字符串 uart_receive_month[3]和命令中的数字 uart_receive_number。

第三步，计算应该显示的月份，UARTStringPut(month[final_month-1])输出*month[]字符串数组中对应的月份。

第四步，收尾工作，还原 uart_receive_string[7]、uart_receive_status 等。

代码如下：

```
1.  void UART0_Handler(void)
2.  {
3.      int32_t uart0_int_status;
4.      char uart_receive_char;
5.      char uart_receive_string[7];
6.      char uart_receive_month[3];
7.      char *month[] = {"JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG",
    "SEP","OCT","NOV","DEC"};
8.      int i=0, final_month=-99, uart_receive_number=-99;
9.
```

```
  uart0_int_status = UARTIntStatus(UART0_BASE, true);           // Get the interrrupt status.
  UARTIntClear(UART0_BASE, uart0_int_status);
   //Clear the asserted interrupts


    //Task1: uart echo
    if (uart0_int_status & (UART_INT_RX | UART_INT_RT))     //接收或接收超时
    {
        GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,GPIO_PIN_1 );  //turn on PN1

        while(UARTCharsAvail(UART0_BASE))       // Loop while there are characters in the receive FIFO.
        {
            //Read the next character from the UART and write it back to the UART.
            uart_receive_char = UARTCharGetNonBlocking(UART0_BASE);
            uart_receive_string[i] = toupper(uart_receive_char);//将前三个月份字母大写
            i++;
            if (i == 3){ //“+””-”
                uart_receive_string[i]=' ';
                i++;
            }
        }

        sscanf(uart_receive_string, "%s %d", uart_receive_month, &uart_receive_number);

        for (i=0;i<12;i++){
            if (strcmp(uart_receive_month, month[i]) == 0){
                final_month = uart_receive_number + i + 1;
            }
        }

        if (final_month < 1 && final_month != -99){
            final_month = final_month + 12;
        }

        if (final_month > 12){
            final_month = final_month - 12;
        }
```

```
46.
47.         if (final_month == -99 || uart_receive_number == -99){
48.             UARTStringPut("\r\nERROR\r\n");
49.         }
50.         else{
51.             UARTStringPut("\r\n");
52.             UARTStringPut(month[final_month-1]);
53.             UARTStringPut("\r\n");
54.         }
55.
56.         strcpy(uart_receive_string, "");
57.         i=0;
58.         final_month = -99;
59.
60.         GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,0 );       //turn of
    f PN1
61.         uart_receive_status = 0;
62.     }
63. }
```

其中，将 final_month、uart_receive_number 初始化为-99，目的是将变量初始化为一个在正常程序执行中不太可能出现的值。-99 既不是有效月份，也不是有效数字的值。这样初始化后，可以更容易地检查这些变量是否已经被正确地初始化。如果变量 final_month 或 uart_receive_number 的值在中断处理程序中仍然保持为-99，则表示在解析输入字符串时发生了错误。

效果图如下：

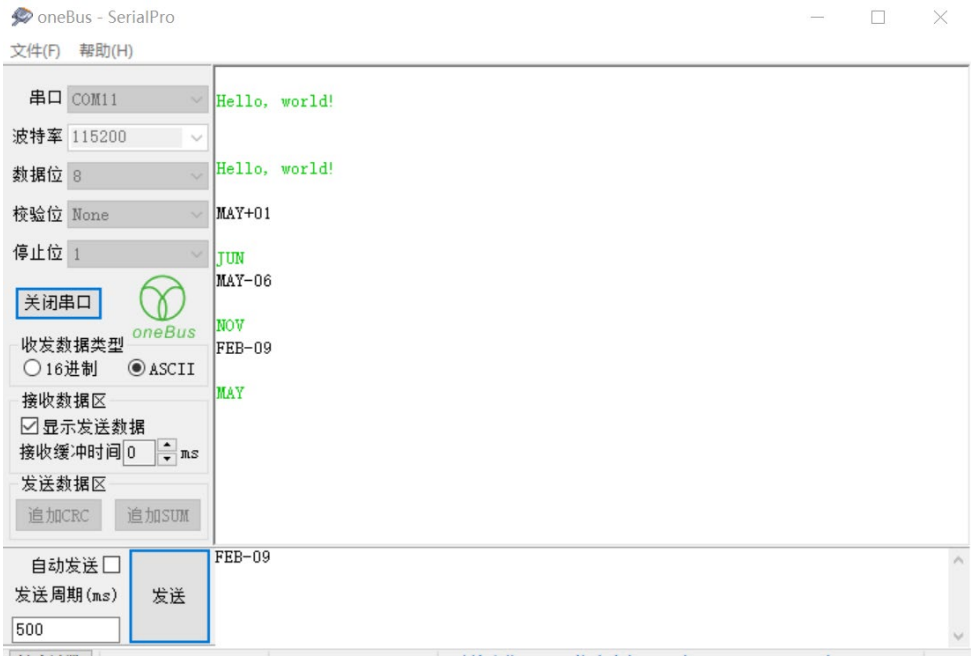

图 7　3-5 效果图

3.6 实验要求 6

3.6.1 任务要求

编程实现模拟时钟程序。底板 Reset 后从 12:00:00 开始自动进行 1 秒计时，并在数码管上以“HH-MM-SS”形式动态显示当前时间。程序运行中可以接收以下三个命令：

a) PC 端发来绝对对时命令，如 SET12:56:03 或 SET12-56-03，自动将当前时间同步到 12:56:03，并回之以当前时间；

b) PC 端发来相对对时命令，如 INC00:00:12，自动将当前时间加 12 秒，并回之以当前时间；

c) PC 端发来查询命令，GETTIME，自动回之以当前时间；

所有回复的当前时间格式统一为 TIME12:56:03，其中 TIME 为字符，后续为时间值。要求将调整后的时间同步显示在数码管上。

3.6.2 任务实现

第一步，初始化 hour、min、sec、cnt，其中 cnt 用于计数，代码 25-30 行实现 1 秒计时，使得 cnt 每秒加 1。代码 5-23 行实现 hour、min、sec 设置，代码 34-72 行实现数码管显示。因为 reset 后 systick_1s_couter 是初始值 0，会立即触发一次 systick 中断中关于 1 秒定时的语句，使运行秒数立即加 1，所以故意设置初始时间为 11：59：59。

```
1.  while (1)
2.      {
3.
4.
5.          hour = cnt / 3600;
6.          min = (cnt - 3600 * hour) / 60;
7.          sec = cnt - 3600 * hour - 60 * min;
8.
9.          if (sec >= 60){
10.             sec = sec - 60;
11.             min++;
12.         }
13.
14.         if (min >= 60){
15.             sec = sec - 60;
16.             hour++;
17.         }
18.
19.         if (hour >= 24){
20.             sec = 0;
```

```
            min = 0;
            hour = 0;
        }

        if (systick_1s_status)
        {
            systick_1s_status   = 0;

            cnt++;
        }

         //数码管

        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    seg7[hour/10]);    //write port 1
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    1);                     //write port 2
        SysCtlDelay(ui32SysClock/3000);

        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    seg7[hour%10]);    //write port 1
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    2);                     //write port 2
        SysCtlDelay(ui32SysClock/3000);

        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    64);              //write port 1
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    4);                     //write port 2
        SysCtlDelay(ui32SysClock/3000);

      result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0)
    ;                   //write port 2
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    seg7[min/10]); //write port 1
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    8);                     //write port 2
        SysCtlDelay(ui32SysClock/3000);
```

```
53.
54.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
55.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    seg7[min%10]); //write port 1
56.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    16);                    //write port 2
57.         SysCtlDelay(ui32SysClock/3000);
58.
59.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
60.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    64);              //write port 1
61.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    32);                    //write port 2
62.         SysCtlDelay(ui32SysClock/3000);
63.
64.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
65.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    seg7[sec/10]); //write port 1
66.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    64);                    //write port 2
67.         SysCtlDelay(ui32SysClock/3000);
68.
69.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    0);                     //write port 2
70.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
    seg7[sec%10]); //write port 1
71.         result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,
    128);                   //write port 2
72.         SysCtlDelay(ui32SysClock/3000);
73.
74.
75.     }//end while
```

第二步，中断服务程序中，和实验 3.5 类似，代码 18-28 行实现存放 PC 端发出的字符，将小写字母字符大写等操作。其中，收到 3 个字符后添加空格，为了方便使用 sscanf()函数分割字符串和时间。代码 30-38 行根据“:”或“-”分割时分秒，根据空格分割出指令字符串。

第三步，根据不同指令，对 cnt 做不同处理，给 output_string 赋值，返回回复。其中代码 40-50 实现“SET”，52-82 行实现“INC”，84-92 行实现“GET”指令。94-99 行同样进行收尾工作。

中断服务程序代码如下：

```
void UART0_Handler(void)
{
    int32_t uart0_int_status;
    char uart_receive_char;
    char uart_receive_string[16];
    char uart_receive_command[16];
    char output_string[16];
    int j=0, k=0, uart_receive_hour, uart_receive_min, uart_receive_sec;

  uart0_int_status = UARTIntStatus(UART0_BASE, true);           // Get the interrrupt status.
  UARTIntClear(UART0_BASE, uart0_int_status);
   //Clear the asserted interrupts


    //Task1: uart echo
    if (uart0_int_status & (UART_INT_RX | UART_INT_RT))     //接收或接收超时
    {

        while(UARTCharsAvail(UART0_BASE))
                // Loop while there are characters in the receive FIFO.
        {
            //Read the next character from the UART and write it back to the UART.
            uart_receive_char = UARTCharGetNonBlocking(UART0_BASE);
            uart_receive_string[j] = toupper(uart_receive_char);
            j++;
            if (j == 3){
                uart_receive_string[j]=' ';
                j++;
            }
        }

        if (uart_receive_string[6] == ':'){
            sscanf(uart_receive_string, "%s %d:%d:%d", uart_receive_command, &uart_receive_hour, &uart_receive_min, &uart_receive_sec);
        }
        else if (uart_receive_string[6] == '-'){
            sscanf(uart_receive_string, "%s %d-%d-%d", uart_receive_command, &uart_receive_hour, &uart_receive_min, &uart_receive_sec);
        }
```

```
36.         else{
37.             sscanf(uart_receive_string, "%3s %*s", uart_receive_comma
    nd);
38.         }
39.
40.         if (strcmp(uart_receive_command, "SET") == 0){
41.
42.             cnt = 3600 * uart_receive_hour + 60 * uart_receive_min +
    uart_receive_sec;
43.
44.             sprintf(output_string, "TIME%02d:%02d:%02d", uart_receive
    _hour, uart_receive_min, uart_receive_sec);
45.             UARTStringPut("\r\n");
46.             UARTStringPut(output_string);
47.             UARTStringPut("\r\n");
48.
49.
50.         }
51.
52.         if (strcmp(uart_receive_command, "INC") == 0){
53.
54.             h = hour;
55.             m = min;
56.             s = sec;
57.
58.             cnt = cnt + 3600 * uart_receive_hour + 60 * uart_receive_
    min + uart_receive_sec;
59.
60.             s+=uart_receive_sec;
61.         if (s >= 60){
62.             s = s - 60;
63.             m++;
64.           }
65.
66.           m+=uart_receive_min;
67.           if (m >= 60){
68.             m = m - 60;
69.             h++;
70.           }
71.
72.             h+=uart_receive_hour;
73.           if (h >= 24){
74.             h-=24;
75.           }
```

```
76.
77.             sprintf(output_string, "TIME%02d:%02d:%02d", h,m,s);
78.             UARTStringPut("\r\n");
79.             UARTStringPut(output_string);
80.             UARTStringPut("\r\n");
81.
82.         }
83.
84.         if (strcmp(uart_receive_command, "GET") == 0){
85.             sscanf(uart_receive_string, "%*s %4s", uart_receive_comma
    nd);
86.             if (strcmp(uart_receive_command, "TIME") == 0){
87.                 sprintf(output_string, "TIME%02d:%02d:%02d", hour, mi
    n, sec);
88.                 UARTStringPut("\r\n");
89.                 UARTStringPut(output_string);
90.                 UARTStringPut("\r\n");
91.             }
92.         }
93.
94.         strcpy(uart_receive_string, "");
95.         j=0;
96.
97.         GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,0 );       //turn of
    f PN1
98.
99.         uart_receive_status = 0;
100.      }
101.  }
```

4. 讨论题解答

1、 试说明实验 3-1 中，if (UARTCharsAvail(UART0_BASE))此行程序的作用。如果没有此行，会导致什么问题？

答：3-1 采用的是阻塞接收方法。题干中 if 语句用来判断接收 FIFO 内有没有字符，使只有在接收 FIFO 收到字符后才进行接收。如果没有该语句，由于阻塞接收的特性，若没有字符发来，程序将一直停在这句代码中等待接收。

2、 实验 3-2 中，采用中断方式进行 UART 接收时，为什么要使能“接收超时”中断条件？若设 RxFIFO 深度分别为 4/8 和 7/8，则当 PC 端一次性发送 17 个字符时，至少会产生几次 UART 中断请求？

答：

如果不设置超时中断，那当发来的字符数量不是预设深度的整数倍时，必定最后会剩下少量字符留在接收 FIFO 里却不能触发中断，导致信息丢失。如果设置了超时中断就可以应付这种情况。

RxFIFO 深度为 4/8 时一次可以接收 8 个字符，将产生 3 次中断；深度为 7/8 时一次可以接收 14 个字符，将产生 2 次中断。

3、实验 3-2 中，void UART0_Handler(void)为什么没有在主函数前声明？ 为什么 UART 和 GPIO 的中断处理程序需要读取并清除中断状态，而 SYSTICK 不需要？

答：因为该函数是中断服务函数。中断服务函数默认已经在程序里声明了，这里无需声明，只需要对中断服务程序重写即可。

Systick 是单源中断，硬件会自动清除中断状态，而另两个是多源中断，需要手动清除中断状态。

4、 实验 3-4 中，执行语句 IntPrioritySet(INT_UART0,0x3); 后，Uart0 的实际优先级为多少？

答：实际优先级为 b000.00011，因为已经设置为全抢占。实际优先级是最高的抢占优先级下的第三号子优先级。

5. 感想和建议

在第一、二次实验的基础上，本次实验中我对 UART 串行通讯深入的了解，掌握在 PC 端使用串口调试工具与实验板进行 UART 通讯的方法，掌握了 UART 的堵塞式与非堵塞式数据传输编程方法，掌握了中断优先级设置方法及对程序运行的影响，为后续大作业的完成夯实基础。

相比第二次实验，我优化了数码管的显示，在第二次实验中显示多位数码管不够清晰的问题得到了改善，每次实验得到的新收获都让我能在大作业完成以及后续课程的学习积累更多经验，助我进步。

最后，感谢《工程实践与科技创新Ⅱ-A》课程组准备的高质量课程和嵌入式开发条件，感谢朱兰娟老师为本门课程的精心准备与悉心讲解，深入浅出的为我们讲解嵌入式开发的诸多知识，将复杂的嵌入式知识网络按照对应模块分割开来，逐一细解，帮助我们拓宽视野，增长见识。感谢所有为我答疑解惑，帮助过我的老师、助教和同学。