# 《USART 串口通信》实验报告

| | | |
|---|---|---|
| 课程名称： 嵌入式系统设计 | 年级：23 级 | 上机实践成绩： |
| 指导教师：郭建 | 姓名： 张建夫 | |
| 上机实践名称： | 学号：10235101477 | 上机实践日期： |
| USART 串口通信 | | 2025/05/06 |
| 上机实践编号： | 组号： | 上机实践时间： |
| | | 14：50~16：30 |

---

## 一、 目的与要求

简单了解串口通信

了解 STM32 的 USART 外设

熟悉使用 USART 进行设备间通信的方法

## 二、 内容与实验原理

### 1. 实验内容：

了解串口通信和 STM32 的 USART 外设

学习使用 STM32 的 USART 和电脑进行通信

### 2. 实验原理：

**（1）串口基本知识：**

- 串口通信(Serial Communication)是一种设备间非常常用的串行通信方式
- 它简单便捷，大部分电子设备都支持该通信方式，电子工程师在调试设备时也经常使用该通信方式输出调试信息。

STM32芯片具有多个 USART外设用于串口通信，即通用同步异步收发器，可以灵活地与外部设备进行全双工数据交换。

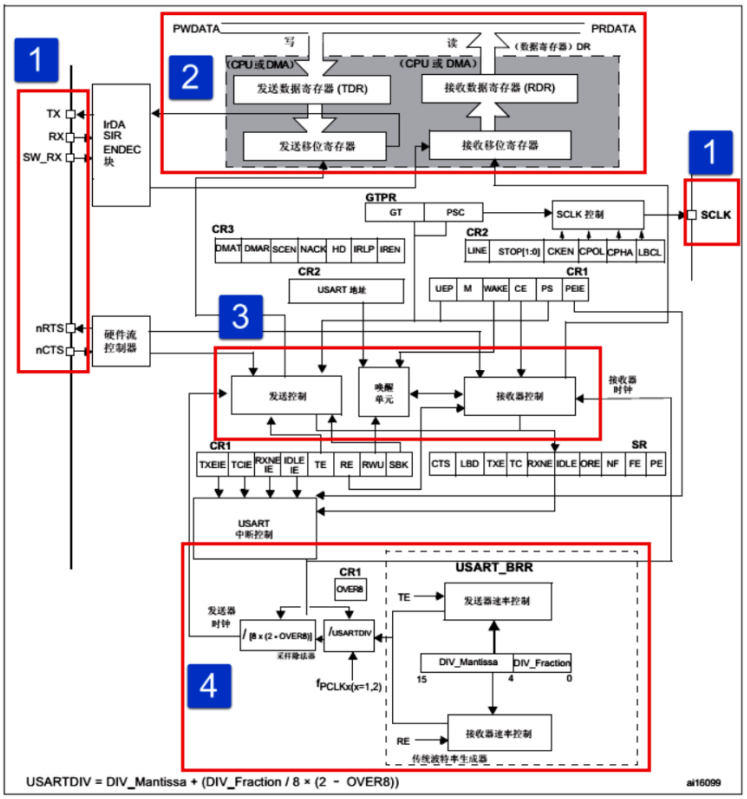
除了USART外设之外，它还有具有UART外设， UART是在USART基础上裁剪掉了同步通信功能，只有异步通信，也就是说USART不但有异步通信功能，还有同步通信功能，而UART只有异步通信功能。

区分同步和异步就是看通信时需不需要对外提供时钟输出，常用的串口通信基本都是 UART。

USART在STM32应用最多莫过于“打印”程序信息，一般在硬件设计时都会预留一个USART通信接口连接电脑，用于调试。

在调试程序时，可以把一些调试信息“打印”在电脑端的串口调试助手工具上，从而了解程序运行是否正确、指出运行出错位置等等。

（2）USART 功能框图：




注意到左上角的五个引脚，本次实验只用到了接收和发送引脚（TX 和 RX）：

**TX**：发送数据输出引脚。

**RX**：接收数据输入引脚。

**SW_RX**：数据接收引脚。只用于单线和智能卡模式，属于内部引脚，没有具体外部引脚。

**nRTS**：请求以发送(Request To Send)，n表示低电平有效。如果使能 RTS 流控制，当 USART接收器准备好接收新数据时就会将 nRTS变成低电平；当接收寄存器已满时， nRTS将被设置为高电平。该引脚只适用于硬件流控制。

**nCTS**：清除以发送(Clear To Send)，n表示低电平有效。如果使能 CTS流控制，发送器在发送下一帧数据之前会检测 nCTS引脚，如果为低电平，表示可以发送数据，如果为高电平则在发送完当前数据帧之后停止发送。该引脚只适用于硬件流控制。

**SCLK**：发送器时钟输出引脚。这个引脚仅适用于同步模式。

具体的引脚和 USART 的引脚对应关系如下图所示：

| | APB2(最高 90MHz) | | APB1(最高 45MHz) | | | | | |
|---|---|---|---|---|---|---|---|---|
| | USART1 | USART6 | USART2 | USART3 | UART4 | UART5 | UART7 | UART8 |
| TX | PA9/PB6 | PC6/PG14 | PA2/PD5 | PB10/PD8 /PC10 | PA0/PC10 | PC12 | PF7/PE8 | PE1 |
| RX | PA10/PB7 | PC7/PG9 | PA3/PD6 | PB11/PD9 /PC11 | PA1/PC11 | PD2 | PF6/PE7 | PE0 |
| SCLK | PA8 | PG7/PC8 | PA4/PD7 | PB12/PD10 /PC12 | | | | |
| nCTS | PA11 | PG13/PG15 | PA0/PD3 | PB13/PD11 | | | | |
| nRTS | PA12 | PG8/PG12 | PA1/PD4 | PB14/PD12 | | | | |

由于 ppt 上的内容太多了，全解释的话篇幅过长而且实验报告的核心不在于复述 ppt，

此处简介 stm32F4 系列的 USART 和 UART：

1. STM32F42xxx 系统控制器有四个 USART 和四个 UART，其中 USART1 和 USART6 的时钟来源于 APB2 总线时钟，其最大频率为 90MHz，其他六个的时钟来源于 APB1 总线时钟，其最大频率为 45MHz。

2. USART 有同步功能，支持 DMA（Direct Memory Access）传输，本次实验并不使用其同步功能。

3. USART 数据寄存器(USART_DR)的第九位数据是否有效取决于 USART 控制寄存器 1(USART_CR1)的 M 位设置，当 M 位为 0 时表示 8 位数据字长，当 M 位为 1 表示 9 位数据字长。

4. 使用 USART 之前需要将 USART_CR1 寄存器的 UE 位置 1 使能 USART。

**（3）stm32F4 中的串行通信流程：**

第一步：

当 USART_CR1 寄存器的发送使能位 TE 置 1 时，启动数据发送，发送移位寄存器的数据会在 TX 引脚输出，一个字符帧发送需要三个部分：起始位+数据帧+停止位。

第二步（发送方）：

发送器开始会先发送一个空闲帧(一个数据帧长度的高电平)，接下来就可以往 USART_DR 寄存器写入要发送的数据。然后等待 USART 状态寄存器(USART_SR)的 TC 位为 1，表示数据传输完成，如果此时 USART_CR1 寄存器的 TCIE 位置 1，将产生中断。

第三步（接收方）：

如果将 USART_CR1 寄存器的 RE 位置 1，使能 USART 接收，使得接收器在 RX 线开始搜索起始位。在确定到起始位后就根据 RX 线电平状态把数据存放在接收移位寄存器内。接收完成后就把接收移位寄存器数据移到 RDR 内，并把 USART_SR 寄存器的 RXNE 位置 1，同时如果 USART_CR2 寄存器的 RXNEIE 置 1 的话可以产生中断。

另外一个知识点是 USART 的发送器和接收器使用相同的波特率，波特率等价于比特率，波特率越大，传输速率越快。

## 三、 使用环境

调用 dxdiag 工具：

Operating System: Windows 11 家庭中文版 64-bit (10.0, Build 22621) (22621.ni_release.220506-1250)
Language: Chinese (Simplified) (Regional Setting: Chinese (Simplified))
System Manufacturer: HP
System Model: HP Pavilion Aero Laptop 13-be2xxx
BIOS: F.13 (type: UEFI)
Processor: AMD Ryzen 5 7535U with Radeon Graphics (12 CPUs), ~2.9GHz

Memory: 16384MB RAM
Available OS Memory: 15574MB RAM
Page File: 27604MB used, 5685MB available
Windows Dir: C:\WINDOWS
DirectX Version: DirectX 12
DX Setup Parameters: Not found
User DPI Setting: 144 DPI (150 percent)
System DPI Setting: 192 DPI (200 percent)
DWM DPI Scaling: UnKnown
Miracast: Available, with HDCP
Microsoft Graphics Hybrid: Not Supported

## 四、主要实验内容和结果展示

本次实验使用的是之前提供的模板，需要将 bsp_key 的头文件和源文件均变为 bsp_usart 的文件内容。除此之外，本次实验较为简单（虽然中间因为一些头大的驱动问题和电脑的硬件问题卡了一段时间），于是想结合之前的 led 灯控制和中断控制做一些有意思的项目，而这个学期选了一门通识课讲到使用灯光表示摩斯码来传递信息，便做了一个小项目：用户通过串口输入一个字符串，板载 led 就将该字符串使用摩斯码表示出来（红色表示点，白色表示横线，灯灭表示当前字符结束）。

### 1. 示例实验：

按照 ppt 上的教程，先要在头文件（bsp_usart.h）对串口的变量进行映射（宏定义）：

```
#define USARTx                          USART1
#define USARTx_CLK                      RCC_APB2Periph_USART1
#define USARTx_CLOCKCMD                 RCC_APB2PeriphClockCmd
#define USARTx_BAUDRATE                 115200  //串口波特率

#define USARTx_RX_GPIO_PORT             GPIOA
#define USARTx_RX_GPIO_CLK              RCC_AHB1Periph_GPIOA
#define USARTx_RX_PIN                   GPIO_Pin_10
#define USARTx_RX_AF                    GPIO_AF_USART1
#define USARTx_RX_SOURCE                GPIO_PinSource10

#define USARTx_TX_GPIO_PORT             GPIOA
#define USARTx_TX_GPIO_CLK              RCC_AHB1Periph_GPIOA
#define USARTx_TX_PIN                   GPIO_Pin_9
#define USARTx_TX_AF                    GPIO_AF_USART1
#define USARTx_TX_SOURCE                GPIO_PinSource9
```

本次使用的串口为 USART1，其中根据前一部分的引脚对应关系表，USART1 的发送引脚对应的 GPIO 引脚为 9，端口为 A， 波特率设为 115200。

该处我在理解代码的时候产生了一个疑惑，可以从图上看出，串口使用的时钟是 APB 总线上的时钟，而接收端和发送端的接口时钟使用的是 AHB 总线上的时钟，这两者不会有冲突吗？后面我查阅了相关资料，发现 TX/RX 的信号发出依靠 AHB1 上的 GPIO 来完成（二者挂载在 AHB1 总线上），而控制逻辑和数据流处理都在 USART 模块本身，二者并不冲突，在实际实现中，USART 是用 APB 时钟驱动 + 与 AHB 上的 GPIO 配合实现物理通信。

接着编写串口初始化函数：

```
void usartx_Config(void)
{
    // 定义初始化结构体变量
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    // 本实验使用串口1，以及PA9 PA10 两个GPIO口
    // 因此开启串口1 PA9 PA10 这组GPIO的的时钟
    RCC_AHB1PeriphClockCmd(USARTx_RX_GPIO_CLK|USARTx_TX_GPIO_CLK,ENABLE);
    USARTx_CLOCKCMD(USARTx_CLK, ENABLE);

    // IO口初始化
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

    // 配置Tx引脚为复用功能
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin =  USARTx_TX_PIN;
    GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStructure);

    // 配置Rx引脚为复用功能
    GPIO_InitStructure.GPIO_Pin =  USARTx_RX_PIN;
    GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStructure);

    /*USART1通信需要使用的PA9和PA10上电后默认不是其发送和接收
    引脚，需要进行端口复用设置后才是，这里进行I/O口的端口复用*/
    GPIO_PinAFConfig(USARTx_RX_GPIO_PORT,USARTx_RX_SOURCE,USARTx_RX_AF);
    GPIO_PinAFConfig(USARTx_TX_GPIO_PORT,USARTx_TX_SOURCE,USARTx_TX_AF);
```

此处 GPIO 初始化加上了 USART 初始化特色：分别为 Tx 和 Rx 引脚设置复用功能，并且要对这两个引脚进行端口复用（对应底部的两行代码）

接下来是 USART 的初始化：

```
    // USART初始化
    USART_InitStructure.USART_BaudRate = USARTx_BAUDRATE;                    // 波特率
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;      // 8位字长
    USART_InitStructure.USART_StopBits = USART_StopBits_1;                   // 一个停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;                      // 无奇偶校验
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;      // 收发全双工模式

    USART_Init(USARTx, &USART_InitStructure);                    // 用以上参数调用库函数初始化

    USART_Cmd(USARTx, ENABLE);                                               // 开启串口
```

此处首先设置了波特率为宏定义里的 115200，设置了字长（数据位+校验位）为 8，一个停止位，不使用奇偶校验（ppt 里面的注释有问题，应该是无奇偶校验，而不是偶校验，偶校验的宏应为 USART_Parity_Even），不使用硬件流控制，使用收发全双工模式（使能接收和发送），最后初始化 USART 并将其使能。

接下来，由于为了成功实现串口通信（为了键盘上的输入能够到达开发板，我们要“重定向”io 流），需要“重写”C 库函数 fgetc 和 fputc，原因是 C 库里面的其他输入输出函数的实现都是调用这两个函数实现的，下面是“重写”代码：

```
//重定向c库函数printf到串口，重定向后可使用printf函数
int fputc(int ch, FILE *f)
{
        USART_SendData(USARTx, (uint8_t) ch);
        while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
        return (ch);
}

//重定向c库函数scanf到串口，重写向后可使用scanf、getchar等函数
int fgetc(FILE *f)
{
        while (USART_GetFlagStatus(USARTx, USART_FLAG_RXNE) == RESET);
        return (int)USART_ReceiveData(USARTx);
}
```

该处代码的实现采用了忙等的方式，不断等待串口输入/输出数据，直到 TXE 标志位（发送寄存器为空）或 RXNE 标志位（读数据寄存器非空）被设置为 reset。

最后，就是 main 函数的实现：

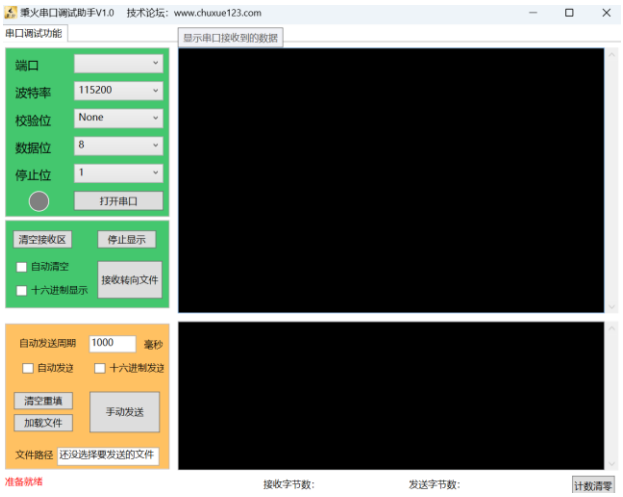
```
    char ch;

    /* LED 端口初始化 */
    LED_GPIO_Config();

  /*初始化串口*/
  usartx_Config();

    Show_message();

    while(1)
    {
        ch = getchar();

        if(ch == '1'){
            LED_RED;
        }
        else{
            LED_RGBOFF;
            printf("输入错误，请重输\n");
        }
    }
```

首先对所有变量初始化，然后在串口上打印提示信息，此处使用了一个 show_message 函数将提示信息进行了封装，打印内容如下：

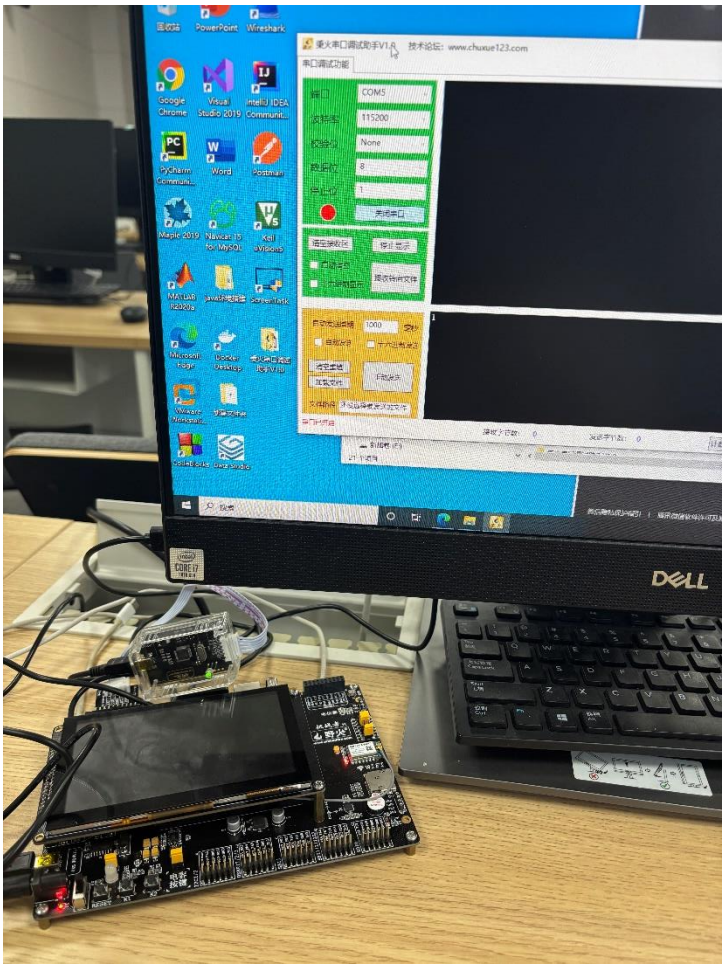
```
static void Show_message(){
    printf("\r\n 这是一个通过串口通信指令控制RGB彩灯实验 \n");
    printf("使用 USART1 参数为 %d 8-N-1 \n", USARTx_BAUDRATE);
    printf("开发板接到指令后控制RGB彩灯颜色，指令对应如下 \n");
    printf("指令   ------- 彩灯颜色 \n");
    printf("1 ------ 红 \n");
}
```

在 main 函数中，我做了一些小小的改变，相比于 ppt 上的代码：我在 else 语句中加入了 LED_RGBOFF，也就是说，如果用户输入的不是 1，那么就会关闭红灯，这样就不会在输入 1 后红灯一直亮着了。

到了这里，软件的部分算是做完，只要 ppt 仔细阅读，代码都不难理解，但是，硬件，才是接下来使我比较伤脑筋的部分，在烧录完代码，按照指令连接 usb，打开了老师提供的串口调试助手后，发现没有任何反应：



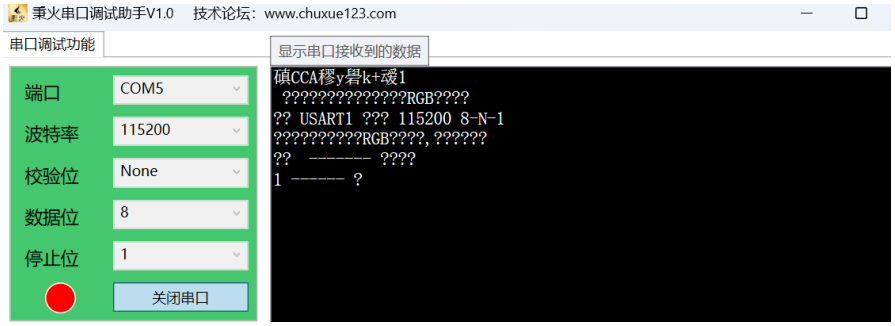

端口处无法进行识别，其他同学此处都是自动识别的，而我这个愣是没反应，在排除了软件问题后，决定换成学校电脑试一次，这次端口处有反映了，但是发送数据后没有任何的打印信息，板载的 led 也没有亮：



后面我又在自己电脑和学校电脑试了几次，打开设备管理器看了又看，发现当我插上 usb 时，我个人 pc 的设备管理器没有任何反应，学校电脑的也是（这里有个伏笔，后面解释，可以说明我用的那台学校电脑硬件也有点问题），下课后求助老师和助教，老师的诊断结果是串口没有成功接上，导致通信没有反应，但是原因不清楚，当时有一位同学（实名感谢洪宇翔同学，他让我立马找到了解决方案，还帮我解决了一个字符编码的小问题）说可以用他的电脑试一下，看是否是开发板的问题。在接上我的开发板后，他的串口上成功显示了我

的打印信息，这一点让我确认了我的开发板和程序是没有问题的，问题只可能是我用的电脑，接下来，最振奋人心的时候到了，我让他打开他的设备管理器，看到了这个：


他的串口连接是有一个 ch340 的驱动的，而我插上去时并没有装这个驱动，随即我开始尝试这个办法。装完驱动后我成功了：




（这里屏幕上显示乱码是因为字符编码的问题，要将 Edit 菜单的 configuration 里编码格式改为如下即可：


该问题也是由洪宇翔同学帮忙解决）

但是这个成功是暂时的，当时我的仿真器（就是用于烧录程序的那个）连的是学校电脑，用于串口通信的 usb 接口连的是我个人电脑右侧的 usb 接口（我电脑有左右两个 usb 接口），回去后，我用自己电脑又试了一遍，这次失败了，后面查看设备管理器，发现只有当 usb 接上我电脑右侧的接口的时候，设备管理器才会自动刷新，结合之前那个“伏笔”，我的推测就是，1. 我的电脑左侧的 usb 和学校电脑的 usb 接口有问题。2. 我的电脑要安装 CH340 驱动才能识别串口。

由于我的左侧 usb 有问题，演示视频中采用了拓展坞将两个 usb 接到电脑上。

2. **参考 PPT， 使用 USART1 指令控制 RGB 彩灯，输入数字 1~8： “红-绿-蓝-黄-紫-青-白 - 灭”点亮相应的彩灯，其它字符错误。**

在解决硬件问题后，这个问题就显得异常简单了，只需要将示例实验里面的主函数的核心代码变为一个 switch 语句，对于符合要求的输入，展示相应的 led 灯，对于其他情况（default），输出错误信息，并让用户重新输入：

```
Show_message();

while(1)
{
    ch = getchar();

    switch(ch){
        case '1':
            LED_RED;
            break;
        case '2':
            LED_GREEN;
            break;
        case '3':
            LED_BLUE;
            break;
        case '4':
            LED_YELLOW;
            break;
        case '5':
            LED_PURPLE;
            break;
        case '6':
            LED_CYAN;
            break;
        case '7':
            LED_WHITE;
            break;
        case '8':
            LED_RGBOFF;
            break;
        default:
            printf("输入错误，请重输，且只允许1到8以内的数字\n");
            break;

    }
}
```

此处对应数字的颜色由题目决定。

演示视频见提交文件。

3. 课外项目：使用 led 表示摩斯码，字符串由串口给出：

本项目的具体需求如下：

- 用户从串口输入一个仅包含英文字符（不论大小写）的字符串，发送后，板载 led 灯应该展现这个字符串的摩斯码（红色表示点，白色表示横线，持续时间为红色的三倍，灯灭表示当前字符结束）

为了使编码较为简单，我打算使用哈希表来存储当前字符的摩斯码，其中，英文字符的摩斯码如下：

| 字符 | 电码符号 | 字符 | 电码符号 |
|---|---|---|---|
| A | • — | N | — • |
| B | — • • • | O | — — — |
| C | — • — • | P | • — — • |
| D | — • • | Q | — — • — |
| E | • | R | • — • |
| F | • • — • | S | • • • |
| G | — — • | T | — |
| H | • • • • | U | • • — |
| I | • • | V | • • • — |
| J | • — — — | W | • — — |
| K | — • — | X | — • • — |
| L | • — • • | Y | — • — — |
| M | — — | Z | — — • • |

可以从图中看出，每个英文字符由 4 个状态组成（一个状态包括点，横线，空），于是哈希表可以设置为一个二维数组（行表示字母 a~b，映射为 0~25，列则存储的是各自字母的摩斯码）：

```
int dict[26][4] = {
    // abcde
    1,2,0,0,
    2,1,1,1,
    2,1,2,1,
    2,1,1,0,
    1,0,0,0,

    // fghij
    1,1,2,1,
    2,2,1,0,
    1,1,1,1,
    1,1,0,0,
    1,2,2,2,

    // klmno
    2,1,2,0,
    1,2,1,1,
    2,2,0,0,
    2,1,0,0,
    2,2,2,0,

    // pqrst
    1,2,2,1,
    2,2,1,2,
    1,2,1,0,
    1,1,1,0,
    2,0,0,0,

    // uvwxy
    1,1,2,0,
    1,1,1,2,
    1,2,2,0,
    2,1,1,2,
    2,1,2,2,

    // z
    2,2,1,1
};
```

其中，数字 1 表示点，数字 2 表示横线，数字 0 表示空。

同时，由于是二维数组，我们能通过库函数 tolower 当前字符转化为其小写形式，再求得这个小写形式与 ascii 码中‘a’的值相减就能得到行数：

```
int row = tolower(ch[i]) - 'a';
```

遍历这一行的摩斯码即可完成该字符的摩斯码展示：

```
            for(int j=0;j < 4;j++){
                switch(dict[row][j]){
                    case 1:
                        LED_RED;
                        Delay(10000);
                        break;
                    case 2:
                        LED_WHITE;
                        Delay(30000);
                        break;
                    default:
                        break;
                }

            }
            LED_RGBOFF;
            Delay(20000);
```

（此处使用的 Delay 函数是第二个实验中模板的延迟函数，通过无效循环来延迟时间，此处不再贴代码，可以在提交的代码文件中查看。）

可以看到各个灯随摩斯码的变化状态，以及每次内循环结束后灯灭的长度，点，横线和灯灭的时间间隔之比为 1：3：2.

最后，为了防止用户输入非法字符而导致程序崩溃，要判断一下映射出来的行数是否合法：

```
int row = tolower(ch[i]) - 'a';
if(row < 0 || row > 25){
    printf("你输入的： %c 不合法，无法转译成摩斯码", row);
    continue;
}
```

最后展现一下整个程序：

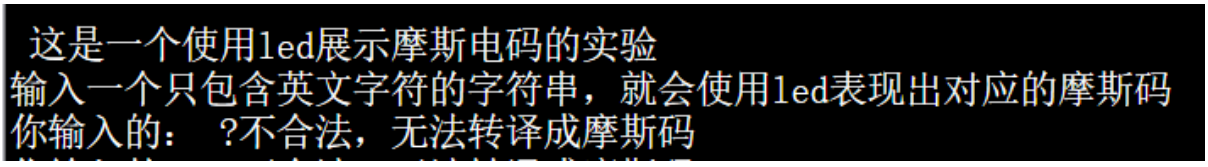
```c
int main(void)
{
    char ch[2000];

    /* LED 端口初始化 */
    LED_GPIO_Config();

  /*初始化串口*/
  usartx_Config();

    Show_message();

    while(1)
    {
        scanf("%s", ch);
        for(int i=0;i < strlen(ch); i++){
            int row = tolower(ch[i]) - 'a';
            if(row < 0 || row > 25){
                printf("你输入的： %c 不合法，无法转译成摩斯码", row);
                continue;
            }
            for(int j=0;j < 4;j++){
                switch(dict[row][j]){
                    case 1:
                        LED_RED;
                        Delay(10000);
                        break;
                    case 2:
                        LED_WHITE;
                        Delay(30000);
                        break;
                    default:
                        break;
                }

            }
            LED_RGBOFF;
            Delay(20000);
        }

    }
}
```

这个版本是初始程序，后面调试的时候发现了如下问题：

1. ch 变量给的缓冲区太大（2k 字节），导致将代码烧到板子上时板子没有任何反应，于是将 ch 的字节数变为 20：

```c
int main(void)
{
    char ch[20];

    /* LED 端口初始化 */
    LED_GPIO_Config();
```

2. 在测试非法字符输入的时候，发现提示信息无法打印用户的字符：




用户输入的字符会出现乱码，于是修改了提示信息的代码：

```c
if(row < 0 || row > 25){
    printf("你输入的： %c 不合法，无法转译成摩斯码\n", row + 'a');
    continue;
}
```

3. 调试时延时时间太长，摩斯码 led 不明显，且元字符（横线和点）中间无空格，导致看不出来摩斯码：

   我将所有延时时间缩短了 10 倍，并在元字符之间加上了较短的间隔：

```
for(int j=0;j < 4;j++){
    switch(dict[row][j]){
        case 1:
            LED_RED;
            Delay(1000);
            break;
        case 2:
            LED_WHITE;
            Delay(3000);
            break;
        default:
            break;
    }
LED_RGBOFF;
Delay(500);
}
LED_RGBOFF;
Delay(2000);
```

最终的程序见提交的代码文件。

演示视频见提交文件。

## 五、实验总结

本次实验我收获颇多，通过USART串口通信这一个功能，使开发板的潜在能力大大增强，尤其是其开始能够接收用户输入了，而不只是按个按键那么简单。此外，在本次实验中，我发现自己对硬件的认识仍然不够深入，还需要不断地学习，来积累一些硬件相关的解决办法，不然以后遇到了同类问题无法下手。本次实验也很好的激发了我对嵌入式开发探索的兴趣，后面有机会找一些有意思的嵌入式项目做一些实际的开发。