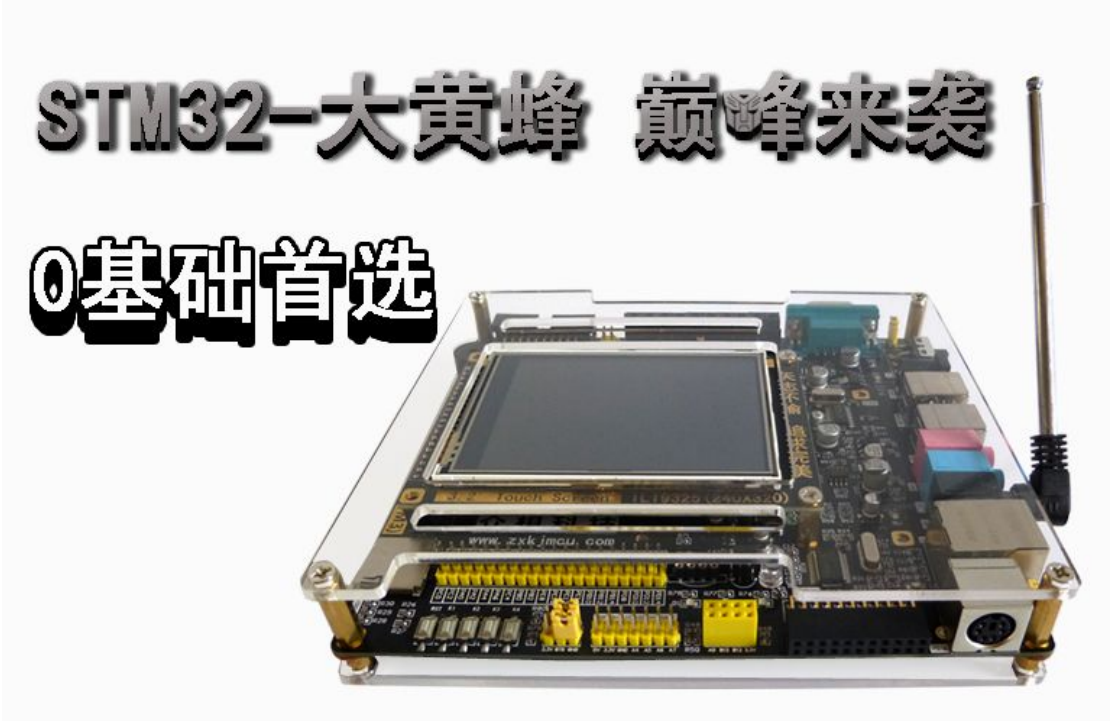
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

4.8 STM32 RS485 串口通讯实验

4.8.1 概述

4.8.1.1 RS485 串口通讯

RS-485 接口具有良好的抗噪声干扰性、长的传输距离和多站能力等。上述优点就使其成为首选的串行接口。因为工业 RS485 通讯接口组成的半双工网络，一般只需二根连线，所以工业 RS485 通讯接口均采用屏蔽双绞线传输。

4.8.1.2 EIA－485 标准简介

为扩展应用范围，EIA 于 1983 年在 EIA-422 基础上制定了 EIA-485 标准，增加了多点、双向通信能力，即允许多个发送器连接到同一条总线上。同时增加了发送器的驱动能力和冲突保护特性，扩展了总线共模范围，后命名为 TIA/EIA-485-A 标准。

由于 EIA-485 是从 EIA-422 基础上发展而来的，所以 EIA-485 许多电气规定与 EIA-422 相仿，如都采用平衡传输方式、都需要在传输线上接终接电阻、最大传输距离约为 1219 米、最大传输速率为 10Mbps 等。但是，EIA-485 可以采用二线与四线方式，采用二线制时可实现真正的多点双向通信，而采用四线连接时，与 EIA-422 一样只能实现点对多点通信，但它比 EIA-422 有改进， 无论四线还是二线连接方式总线上可接多达 32 个设备。

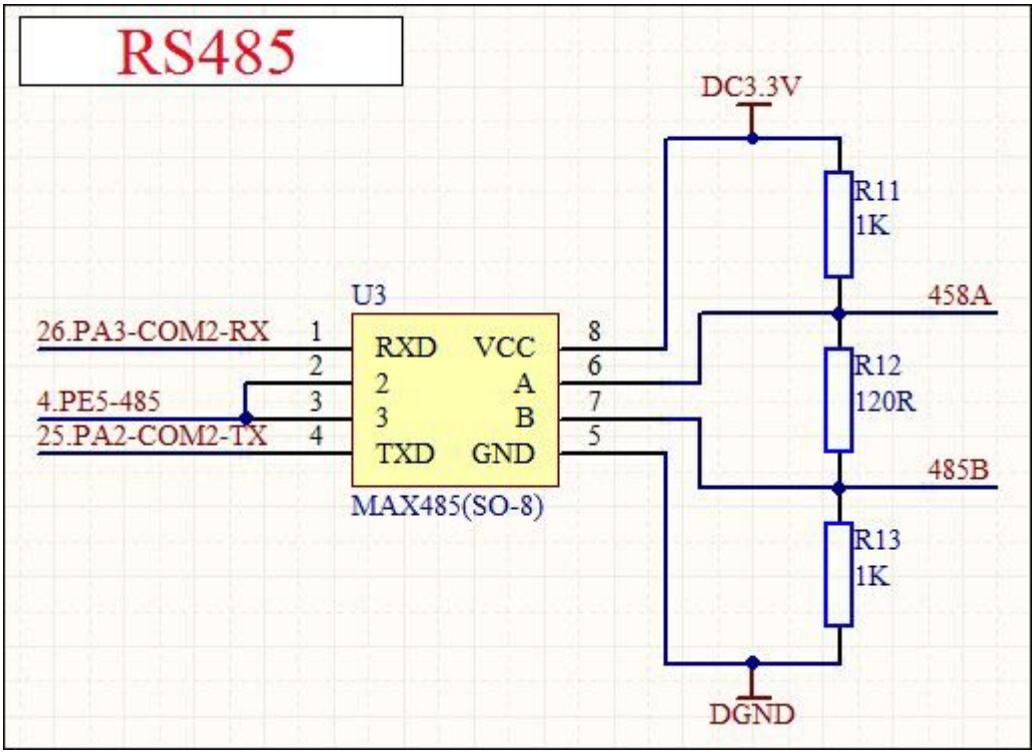
EIA-232、EIA-422 与 EIA-485 标准的优点

4.8.2 RS485 通讯实验目的

通过下传编写好的串口通讯程序，验证串口通讯的正确性，掌握串口通讯软件的设计方法。

4.8.3 硬件设计

在这里使用的串口通讯芯片是常规芯片 MXA485，RS485 串口通讯电路是一个很成熟的电路，电路大家都熟悉了（参考原理图纸）。从图中可以看出芯片 1#管脚是数据接收端，4#管脚是数据发送端；2#、3#管脚是发送/接收状态转换控制端。



图一 RS485 通讯电路

4.8.4 软件设计

RS485 通讯程序中我们用到中断、端口复用功能等函数。我们采用 PA 端口进行串口通讯；PA2 端口接 TX，PA3 端口接 RX，状态转换控制线连接 PE5 端口。在这里我们也是全部使用库函数编写程序。

4.8.4.1 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c  // 中断控制字（优先级设置）库函数
stm32f10x_exti.c // 外部中断库处理函数
stm32f10x_usart.c  // 串口通讯函数
```

都是串口通讯，引用的库函数和 RS232 都是一样的。其中 stm32f10x_gpio.h 头文件包含了 GPIO 端口的定义。stm32f10x_rcc.h 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 stm32f10x_gpio.c 和 stm32f10x_rcc.c 加到工程中；Misc.c 库函数主要包含了中断优先级的设置，stm32f10x_exti.c 库函数主要包含了外部中断设置参数，tm32f10x_usart.c 库函数主要包含串行通讯设置，这些函数也要添加到函数库中。

4.8.4.2 自定义头文件

```
pbdata.h
pbdata.c
```

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.8.4.3 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，会提示重复调用错误。

4.8.4.4 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
{
   RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
   RCC_HSEConfig(RCC_HSE_ON);    /*设置外部高速晶振（HSE） HSE 晶振打开(ON)*/

   if(RCC_WaitForHSEStartUp() == SUCCESS) {  /*等待 HSE 起振,  SUCCESS：HSE 晶振稳定且就绪*/

   RCC_HCLKConfig(RCC_SYSCLK_Div1);/*设置 AHB 时钟(HCLK)RCC_SYSCLK_Div1——AHB 时钟 = 系统时*/
   RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟(PCLK2)RCC_HCLK_Div1——APB2 时钟 = HCLK*/
   RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟(PCLK1)RCC_HCLK_Div2——APB1 时钟 = HCLK / 2*/

   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);/*设置 PLL 时钟源及倍频系数*/
   RCC_PLLCmd(ENABLE);    /*使能 PLL */
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) ; /*检查指定的 RCC 标志位(PLL 准备好标志)设置与否*/

   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);  /*设置系统时钟（SYSCLK） */
   while(RCC_GetSYSCLKSource() != 0x08);      /*0x08：PLL 作为系统时钟 */
 }
}
/*******************************************************************************
* 名    称：delay_us(u32 nus)
* 功    能：微秒延时函数
* 入口参数：u32  nus
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_us(u32 nus)
```

```
{
    u32 temp;
   SysTick->LOAD = 9*nus;
   SysTick->VAL=0X00;//清空计数器
   SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
   do
   {
    temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}

/************************************************************************
* 名    称：delay_ms(u16 nms)
* 功    能：毫秒延时函数
* 入口参数：u16 nms
* 出口参数：无
* 说    明：
* 调用方法：无
************************************************************************/
void delay_ms(u16 nms)
{
   u32 temp;
    SysTick->LOAD = 9000*nms;
   SysTick->VAL=0X00;//清空计数器
   SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
   do
   {
    temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

4.8.5 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.8.6 GPIO 引脚时钟使能

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
```

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);//设置串口 2 时钟使能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能
```

无论编写任何程序都要打开相关的时钟设置，在这个实验中用到使能 AHB 外设时钟有 3 各部分，分别是使能端口时钟、使能串口 2 时钟、使能端口复用时钟。所有的外设时钟全部挂接在 AHB 后的两个预分频器上，初学者一般都感觉在设置外设时钟时不知道怎样填写使能或使能外设时钟函数中的“参数 1”，下表我列出了一些“参数 1”的固定书写格式，方便大家查询。同理当大家熟练使用 C 语言后，通过别的途径也可以查询到下表中“参数 1”的固定书写格式。这些都存在于 stm32f10x_rcc.c 库函数中。

表 1 预分频器 APB1、APB2 控制的功能模块（部分）

序号	函数参数 1	长度
1	RCC_APB1Periph_TIM2	((uint32_t)0x00000001)
2	RCC_APB1Periph_TIM3	((uint32_t)0x00000002)
3	RCC_APB1Periph_TIM4	((uint32_t)0x00000004)
4	RCC_APB1Periph_TIM5	((uint32_t)0x00000008)
5	RCC_APB1Periph_TIM6	((uint32_t)0x00000010)
6	RCC_APB1Periph_TIM7	((uint32_t)0x00000020)
7	RCC_APB1Periph_TIM12	((uint32_t)0x00000040)
8	RCC_APB1Periph_TIM13	((uint32_t)0x00000080)
9	RCC_APB1Periph_TIM14	((uint32_t)0x00000100)
10	RCC_APB1Periph_WWDG	((uint32_t)0x00000800)
11	RCC_APB1Periph_SPI2	((uint32_t)0x00004000)
12	RCC_APB1Periph_SPI3	((uint32_t)0x00008000)
13	RCC_APB1Periph_USART2	((uint32_t)0x00020000)
14	RCC_APB1Periph_USART3	((uint32_t)0x00040000)
15	RCC_APB1Periph_UART4	((uint32_t)0x00080000)
16	RCC_APB1Periph_UART5	((uint32_t)0x00100000)
17	RCC_APB1Periph_I2C1	((uint32_t)0x00200000)
18	RCC_APB1Periph_I2C2	((uint32_t)0x00400000)
19	RCC_APB1Periph_USB	((uint32_t)0x00800000)
20	RCC_APB1Periph_CAN1	((uint32_t)0x02000000)
21	RCC_APB1Periph_CAN2	((uint32_t)0x04000000)
22	RCC_APB1Periph_BKP	((uint32_t)0x08000000)
23	RCC_APB1Periph_PWR	((uint32_t)0x10000000)
24	RCC_APB1Periph_DAC	((uint32_t)0x20000000)

25	RCC_APB1Periph_CEC	((uint32_t)0x40000000)

4.8.7 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待串口通讯中断的到来。

```
while(1)
```

4.8.8 stm32f10x_it.c 文件里的内容

4.8.8.1 中断处理程序

在中断处理子程序中仅串口 2 子函数非空，运行时仅它会有参数输出。中断处理子函数都包含在 stm32f10x_it.h 文件中，初学者如果不明白各种中断处理子函数的具体书写方式，可以进入这个头文件中查找，找到后复制到中断处理 stm32f10x_it.c 文件中使用。

```
#include "stm32f10x_it.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
#include "pbdata.h"

void NMI_Handler(void)//空子程序
{
}

void USART1_IRQHandler(void) //串口 1 中断处理空子程序，没有被使用
{
}

void USART2_IRQHandler(void) //串口 2 中断处理函数
{
   u8 temp=0;//中间临时变量 temp 付初值 0
   if(USART_GetITStatus(USART2,USART_IT_RXNE)!=RESET)//检查串口 2 接收中断发生与否，接收中断到来，接收数据。
   {
        temp=USART_ReceiveData(USART2);//返回串口 2 最近接收到数据，把接收到的数据放在临时变量 temp 中

      GPIO_SetBits(GPIOE,GPIO_Pin_5);//把 RS485 发送控制线置高
      delay_ms(1);
```

```
            USART_SendData(USART2,temp);//通过串口 2 发送单个数据
            while(USART_GetFlagStatus(USART2,USART_FLAG_TXE)==RESET);//等待
RS485 发送完毕，发送数据寄存器空标志位
            delay_ms(2);

            GPIO_ResetBits(GPIOE,GPIO_Pin_5);//把 RS485 发送控制线清零，转为接
收状态
        }
    }
```

在中断处理程序中，各个中断位的判断很重要，有些不好理解，下文中对通讯中断标志位的应用做了详细的说明，相信初学者会比较容易的掌握。

4.8.8.2 函数 USART_GetITStatus

表 2 USART_GetITStatus 说明

函数名	USART_GetITStatus
函数原型	ITStatus USART_ClearFlag(USART_TypeDef*USARTx,u16 USART_IT)
功能描述	检查指定的 USART 中断发生与否
输入参数 1	USARTx：x 可以是 1,2 或者 3，来选择 USART 外设
输入参数 2	USART_IT：待检查 USART 中断源 参阅 Section: USART_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	USART_IT 的新状态
先决条件	无
被调用函数	无

USART_IT

附表 2 给出了所有可能被函数 USART_GetITStatus 检查的中断标志位列表

附表 2 USART_IT 值

USART_IT	描述
USART_IT_PE	奇偶错误中断
USART_IT_TXE	发送中断
USART_IT_TC	发送完成中断
USART_IT_RXNE	接收中断
USART_IT_IDLE	空闲总线中断
USART_IT_LBD	LIN 中断探测中断
USART_IT_CTS	CTS 中断
USART_IT_ORE	溢出错误中断
USART_IT_NE	噪音错误中断
USART_IT_FE	帧错误中断

在这个程序中我们先要要检查接收中断（USART_IT_RXNE）发生与否，如果有中断产生，先接收数据；接受完数据后把控制线 PE 置高，转为发送状态，

再把接收到的数据发送回去，发送完毕，把控制线 PE 清零，转为接收状态。

4.8.9 main.c 文件里的内容是

大家都知道串行通讯是按位进行发送和接收字节的通讯方式。RS485 通讯也是这样的，在这个试验中主程序 while(1)循环等待，等待串口中断的产生后转入通讯处理程序。下面是主程序的内容。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);

int main(void)
{
   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();//串口初始化
   NVIC_Configuration();

   GPIO_ResetBits(GPIOE,GPIO_Pin_5);
   while(1);
}

void RCC_Configuration(void)
{
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);//端口 PA 时钟使能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);//设置串口 2 时钟使能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能，通过 APB2 预分频器打开功能复用 IO 时钟(所有的)
}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5;//RS_485 发送/接收控制线
```

```
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOE,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_2;//TX
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;//推挽输出模式
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_3;//RX
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;//浮空输入模式
    GPIO_Init(GPIOA,&GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);//设置优先级分组

    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn; //USART2 全局中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //先占优先级
设置

    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; //从优先级设置
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //使能
    NVIC_Init(&NVIC_InitStructure);
}

void USART_Configuration(void)
{
    USART_InitTypeDef  USART_InitStructure;

    USART_InitStructure.USART_BaudRate=9600;
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;
    USART_InitStructure.USART_StopBits=USART_StopBits_1;
    USART_InitStructure.USART_Parity=USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowContr
ol_None;
    USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

    USART_Init(USART2,&USART_InitStructure);
    USART_ITConfig(USART2,USART_IT_RXNE,ENABLE);//使能串口 2 中断
    USART_Cmd(USART2,ENABLE);//使能串口 2 外设
```

```
    USART_ClearFlag(USART2,USART_FLAG_TC);清楚串口 2 待处理标志位
}
```

在 main(void)程序体中代码比较多，USART_Init 函数理解起来有些困难，下面对函数 USART_Init 做一下比较详细的说明。

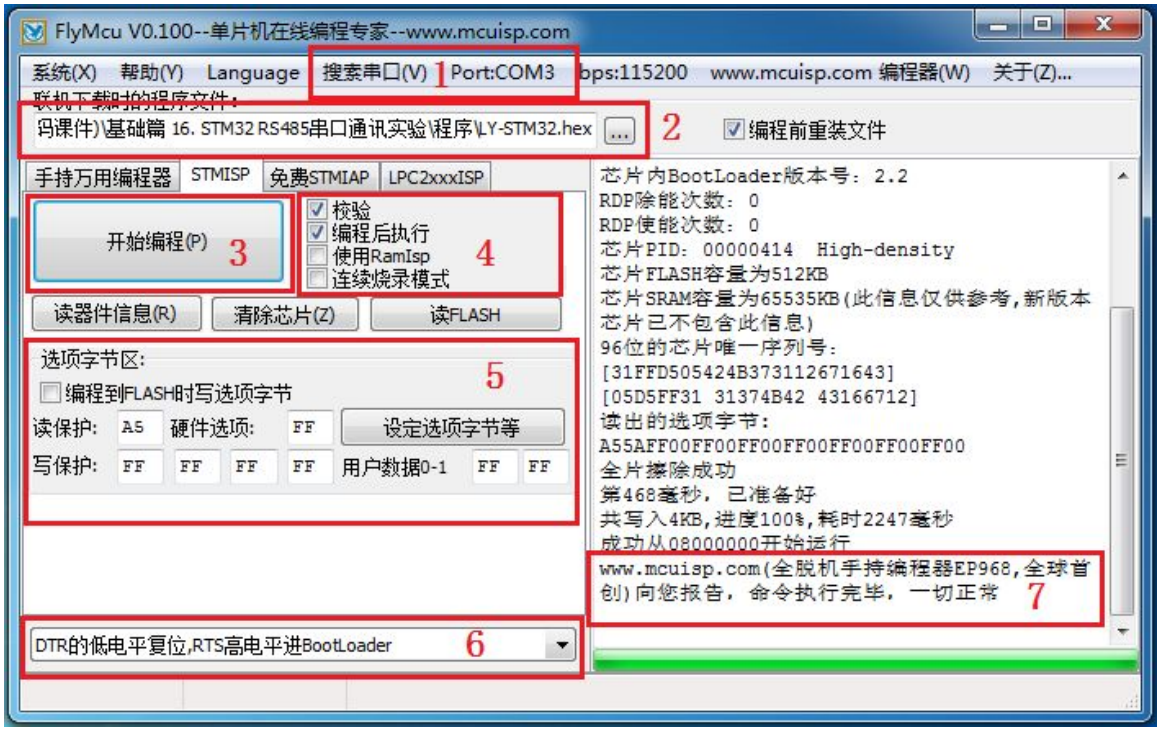
1、函数 USART_Init

表 1 USART_Init 说明

函数名	USART_Init
函数原型	Void USART_Init(USART_TypeDef*USARTx,USART_InitTypeDef*USART_InitStruct)
功能描述	根据 USART_InitStruct 中制定的参数初始化外设 USARTx 寄存器
输入参数 1	USARTx：x 可以是 1,2 或者 3，来选择 USART 外设
输入参数 2	USART_InitStruct：指向结构 USART_TypeDef 的指针，包含了外设 USART 的配置信息。参阅 Section: USART_TypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

4.8.10 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM3。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。

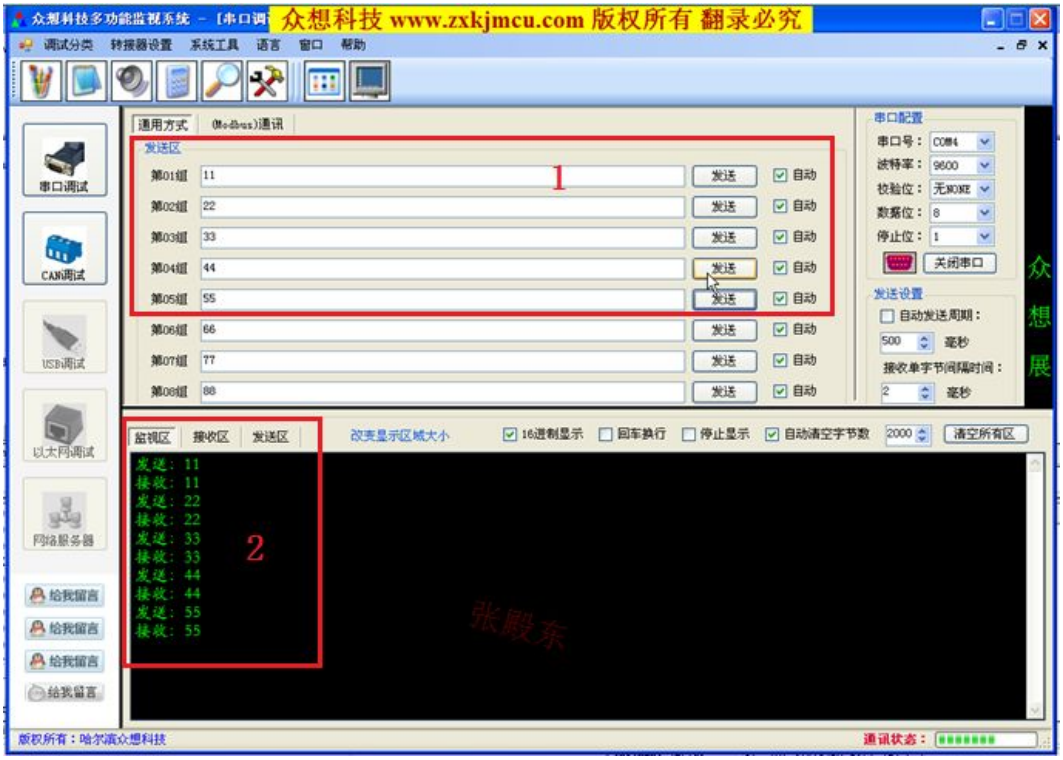


本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\1,基础篇\基础篇 16. STM32 RS485 串口通讯实验\程序）

4.8.11 实验效果图

RS485 通讯程序写入实验板后，使用公司开发的多功能监视系统，在串口调试界面中的发送区输入随机数字，点击发送按钮（参考红色框图 1 部分），如果在接收区看到有数据返回，就说明程序设计成功，RS485 通讯正常（参考红色框图 2 部分）。串口通讯涉及到函数很多，设置也比较多，细节决定成败，大家需要花费多一些的时间去理解和编程实验。只要学习者认真看书和看视频，再加上勤奋，一定会很快掌握。

具体实验效果参考下图。


众想科技 www.zxkjmcu.com 版权所有 翻录必究
串口调试
CAN调试
USB调试
以太网调试
网络服务器
通用方式
发送区
串口配置
关闭串口
监视区
接收区
发送区
清空所有区
1
2
张殿东