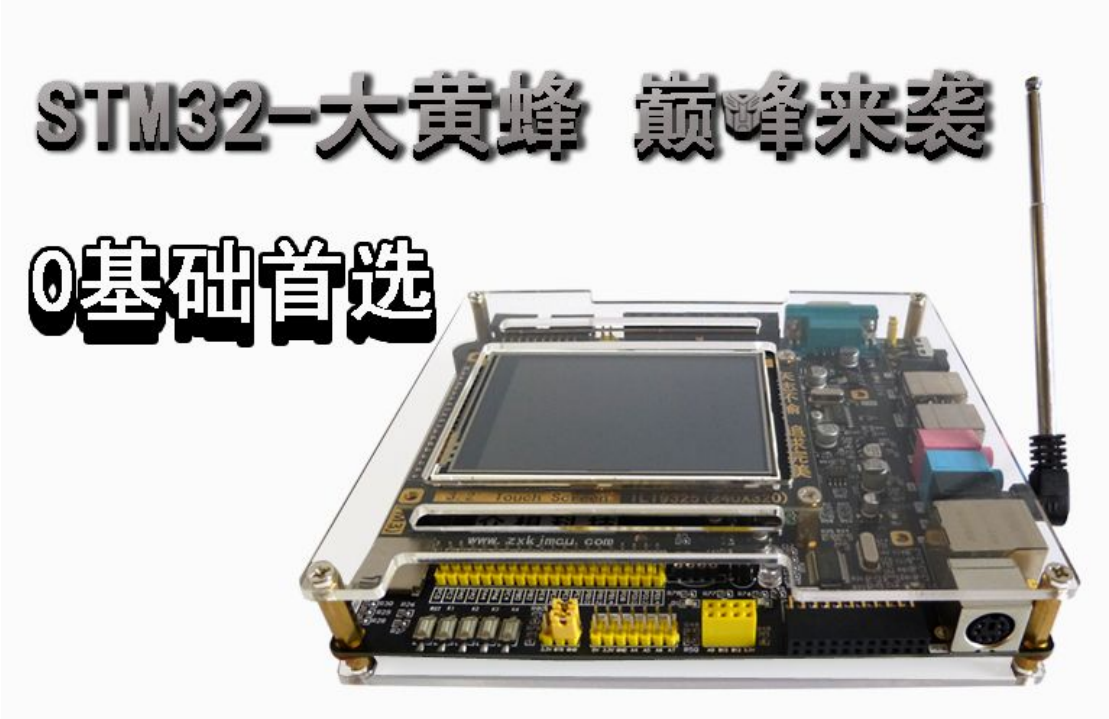
学 ARM 从 STM32 开始

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

4.7 STM32 USART 串口通信实验

4.7.1 概述

4.7.1.1 USART 串行通信概念

USART 串口，就是通用的、既支持同步也支持异步的接收、发送电子“模块”。在芯片内部，与 SPI、I2C 一起构成单片机的总线“枝干”，对于串口，就是指串行通信的接口。串口通信指串口按位（bit）发送和接收字节。尽管比按字节（byte）的并行通信慢，但是串口可以在使用一根线发送数据的同时用另一根线接收数据。

4.7.1.2 EIA－232、EIA-422 和 EIA－485 标准简介

最被人们熟悉的串行通信技术标准是 EIA－232、EIA-422 和 EIA－485，也就是以前所称的 RS-232、RS-422 和 RS-485。由于 EIA 提出的建议标准都是以“RS”作为前缀，所以在工业通信领域，仍然习惯将上述标准以 RS 作前缀称谓。

EIA－232、EIA-422 和 EIA－485 都是串行数据接口标准，最初都是由电子工业协会（EIA）制订并发布的，EIA-232 在 1962 年发布，后来陆续有不少改进版本，其中最常用的是 EIA-232-C 版。

1、EIA-232

目前 EIA-232 是 PC 机与通信工业中应用最广泛的一种串行接口。EIA-232 被定义为一种在低速率串行通信中增加通信距离的单端标准。EIA-232 采取不平衡传输方式，即所谓单端通信。标准规定，EIA－232 的传

送距离要求可达 50 英尺（约 15 米），最高速率为 20kbps。

2、EIA-422

由于 EIA-232 存在传输距离有限等不足，于是 EIA-422 诞生了。EIA-422 标准全称是“平衡电压数字接口电路的电气特性”，它定义了一种平衡通信接口，将传输速率提高到 10Mbps，传输距离延长到 4000 英尺（约 1219 米），并允许在一条平衡总线上连接最多 10 个接收器。当然，EIA－422 也有缺陷：因为其平衡双绞线的长度与传输速率成反比，所以在 100kbps 速率以内，传输距离才可能达到最大值，也就是说，只有在很短的距离下才能获得最高传输速率。一般在 100 米长的双绞线上所能获得的最大传输速率仅为 1Mbps。另外有一点必须指出，在 EIA-422 通信中，只有一个主设备（Master），其余为从设备（Slave），从设备之间不能进行通信，所以 EIA-422 支持的是点对多点的双向通信。

3、EIA-485

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端电阻、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

4.7.1.3 USB 接口

USB 接口的出现及其发展

USB 是英文 Universal Serial Bus 的缩写，翻译成中文的含义是“通用串行总线”。从技术上看，USB 是一种串行总线系统，它的最大特性是支持即插即用和热插拔功能。在 Windows 2000 的操作系统中，任何一款标准的 USB 设备可以在任何时间、任何状态下与计算机连接，并且能够马上开始工作。

USB 诞生于 1994 年，是由康柏、IBM、Intel 和 Microsoft 共同推出的，即在统一外设接口，如打印机、外置 Modem、扫描仪、鼠标等的接口，以便于用户进行便捷的安装和使用，逐步取代以往的串口、并口和 PS/2 接口。

发展至今，USB 共有三种标准：1996 年发布的 USB1.0，1998 年发布的 USB1.1 以及刚刚发布的最新标准 USB2.0。此三种标准最大的差别就在于数据传输速率方面，当然，在其他方面也有不同程度的改进。就目前的 USB2.0 而言，其传输速度可以达到 480Mbps，最多可以支持 127 个设备。

USB 接口和三个 EIA 标准的对比

目前在 IT 领域，USB 接口可谓春风得意。人们在市场上可以看到，每一款计算机主板都带有不少于 2 个 USB 接口，USB 打印机、USB 调制解调器、USB 鼠标、USB 音箱、USB 存储器等产品越来越多，USB 接口已经占据了串行通信技术的垄断地位。

但是，在工业领域，使用 USB 接口的产品则甚为少见。在工业领域，人们更

要求产品的可靠性和稳定性，目前，EIA 标准下的串行通信技术完全可以满足人们对工业设备传输的各种性能要求，而且，这些产品价格非常低廉。相比之下，USB 价格较高，并且其即插即用的功能在工业通信中没有优势。因为工业设备一般连接好以后很少进行重复插拔，USB 特性的优越性不能很好地被体现出来，也就得不到工业界的普遍认可。因此，在工业领域，EIA 标准依然占据统治地位。

4.7.1.4 USART 功能概述

USART 接口通过三个引脚与其它设备连接在一起。任何 USART 双向通信至少需要两个脚：接收数据输入(RX)和发送数据输出（TX）。

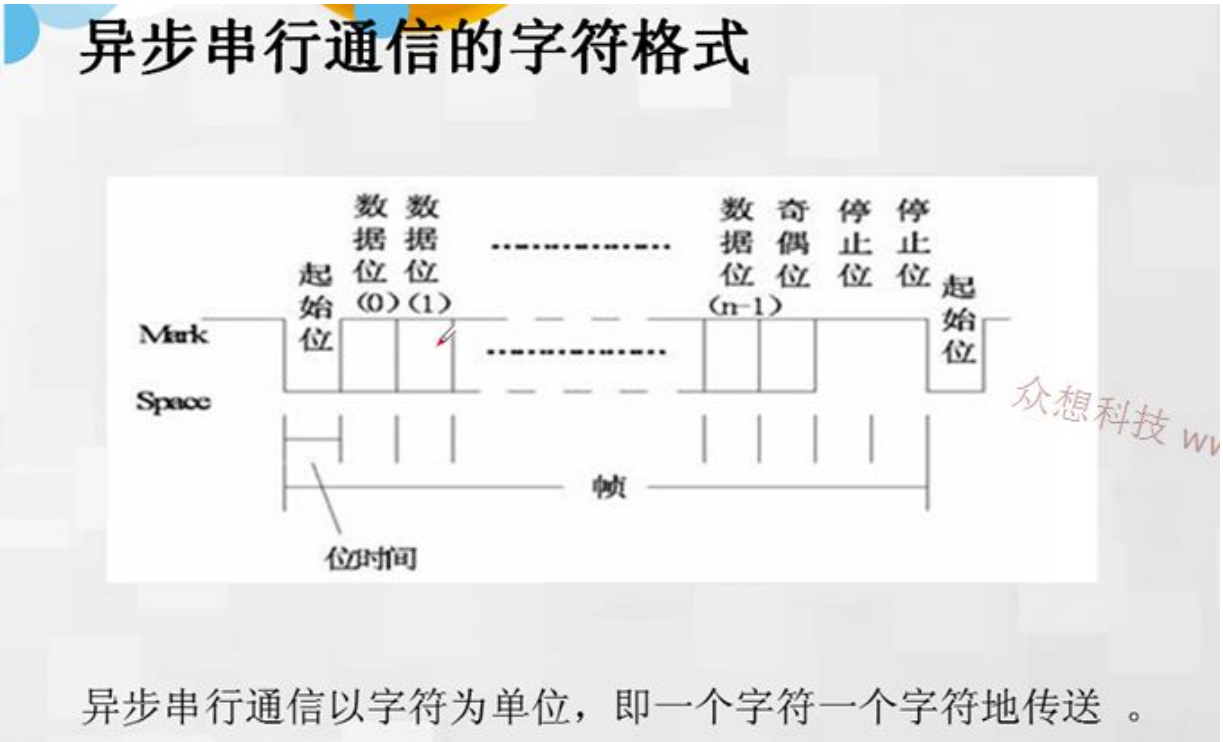
RX：接收数据串行输入。通过采样技术来区别数据和噪音，从而恢复数据。

TX：发送数据串行输出。当发送器被禁止时，输出引脚回复到它的 I/O 端口配置。当发送器被激活，并且不发送数据时，TX 引脚处于高电平。在单线和智能卡模式里，此 I/O 口被同事用于数据的发送和接收。

- 总线在发送或者接收前应处于空闲状态
- 一个起始位
- 一个数据字（8 位或者 9 位），最低有效位在前
- 1 或者 2 个停止位，由此表示数据位的结束
- 使用分数波特率发生器-12 位整数和 4 位小数的表示方法
- 一个状态寄存器（USART_SR）
- 一个波特率寄存器（USART-BRR），12 位整数和 4 位小数
- 一个智能卡模式下的保护时间寄存器（USART-GTPR）

- IrDA_RDI:IrDA 模式下的数据输入
- IrDA_TDO:IrDA 模式下的数据输出
- nCTS:清除发送，若是高电平，在当前数据传输结束时阻断下一次的数据发送。
- nRTS:发送请求，若是低电平，表明 USART 准备好接收数据

4.7.1.5 异步串行通信格式



4.7.1.6 串行通信速率

串行通信传输速率用于说明传输的快慢。在串行通信中，数据是按位进行传送的，因此传输速率用每秒钟传送格式位的数目来表示，称之波特率（band rate)。每秒传送一个格式位就是 1 波特。常用的波特率有：4800、9600、19200、115200 波特。

4.7.1.7 USART 模式配置

STM32 内部集成了 5 个串行通信接口，每个串行通信接口支持的方式详细见

表 1.从图中可以看出“USART1”、“USART2”、“USART3”全部支持表格中列出的功能；“USART4”不支持“硬件流控制”，不支持“同步”功能，、不支持“智能卡”功能；“USART5”不支持“硬件流控制”，不支持“同步”功能，、不支持“智能卡”功能。

表 1 USART 模式配置说明

USART 模式	USART1	USART2	USART3	USART4	USART5
异步模式	×	×	×	×	×
硬件流控制	×	×	×	NA	NA
多缓存通讯	×	×	×	×	×
多处理器通讯	×	×	×	×	×
同步	×	×	×	NA	NA
智能卡	×	×	×	NA	NA
半双工	×	×	×	×	×
IrDA	×	×	×	×	×
LIN	×	×	×	×	×

4.7.1.8 USART 库函数说明

表 2 USART 库函数

序号	USART 库函数
1	USART_ReceiveData
2	USART_SendBreak
3	USART_SetPrescaler
4	USART_CardCmd
5	USART_CardNackCmd
6	USART_HalfDuplexCmd
7	USART_IrDAConfig
8	USART_IrDACmd
9	USART_GetFlagStatus
10	USART_ClearFlag
11	USART_GetITStatus
12	USART_ClearITPendingBit

以上库函数大家知道，知道在什么地方能方便的找到就可以，不需要背下来，最主要的是做到活学活用。

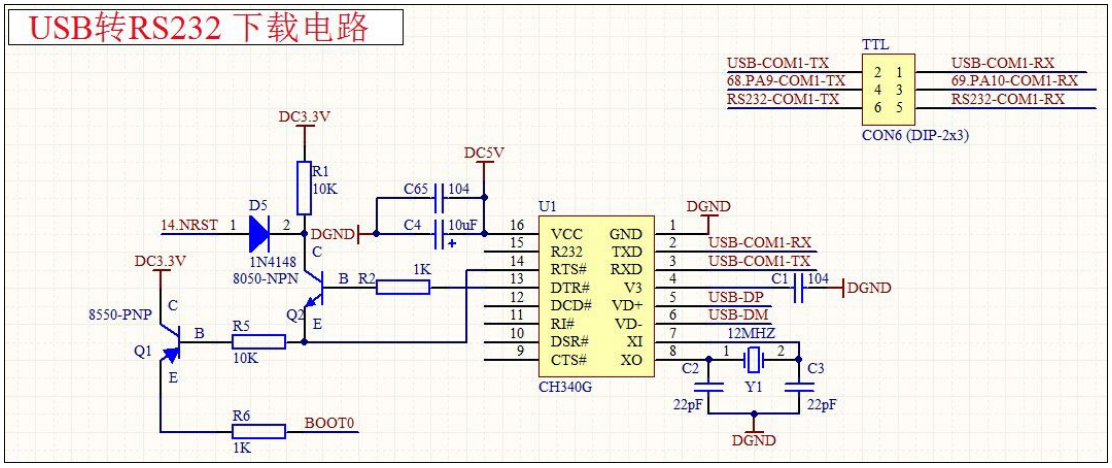
4.7.2 实验目的

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。

4.7.3 硬件设计

在这里使用的串口通讯芯片是 MXA3232ESE，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）。在这里就不再重复说明。



图一　串口通讯应用电路

4.7.4 软件设计

在这个通讯程序中我们用到中断、端口功能等函数。我们采用 PA9、PA10 端口进行串口通讯，并且我们还要使用到管脚复用功能。这些功能函数都要初始化。

4.7.4.1 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_tim.c  // 定时器库处理函数
stm32f10x_usart.c  // 串口通讯函数
```

本节实验及以后的实验我们都是用到库文件，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_tim.c 库函数主要包含定时器设置，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。以上库文件包含了本次实验所有要用到的函数使用功能。

4.7.4.2 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。

4.7.4.3 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。

4.7.4.4 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 */
 }
}
void delay(u32 nCount)
{
   for(;nCount!=0;nCount--);
}
/*******************************************************************************
* 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
}
```

4.7.5 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。

4.7.6 GPIO 引脚时钟使能

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 时钟使
能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能
```

本节实验用到了 PA 端口，所以要把 PA 端口的时钟打开；串口 1 时钟源是通过 APB2 预分频器得到的，串口 1 时钟初始化；因为要与外部芯片通讯，

所以要打开功能复用时钟。

4.7.7 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)
```

4.7.8 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中仅串口 1 子函数非空，进入中断处理函数后，只有串口 1 有参数输出。还有就是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)
{

}

void USART1_IRQHandler(void)
{
   if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
   {
      USART_SendData(USART1,USART_ReceiveData(USART1));
      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
   }
}
```

4.7.9 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。在这个试验中主程序大部分是初始化函数和几个子函数，主要执行过程就使等待

USART 中断的产生,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();//中断优先级初始化

   while(1);//循环语句
}

void RCC_Configuration(void)
{
   SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);//端口 PA 时钟使能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 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;
    //LED
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//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_10;//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 = USART1_IRQn;//USART1 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;
        NVIC_Init(&NVIC_InitStructure);
    }
    void USART_Configuration(void)
    {
         USART_InitTypeDef  USART_InitStructure;//串口结构体
        USART_InitStructure.USART_BaudRate=9600;//比特率
        USART_InitStructure.USART_WordLength=USART_WordLength_8b;//数据位
        USART_InitStructure.USART_StopBits=USART_StopBits_1;//1 位起始位
        USART_InitStructure.USART_Parity=USART_Parity_No;//停止位
        USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowContr
ol_None;
        USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

        USART_Init(USART1,&USART_InitStructure);
        USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);//打开接收中断
        USART_Cmd(USART1,ENABLE);//打开整个串口 1 外设
        USART_ClearFlag(USART1,USART_FLAG_TC);//清除串口 1 中断标志位
    }
```

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、USART_ClearFlag 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

USART_InitTypeDef Struct

USART_InitTypeDef 定义于文件“stm32f10x_usart.h”：

附表 1 描述了结构 USART_InitTypeDef 在同步模式和异步模式下使用的不同成员。可以看出在一部模式下使用的参数少，只用 6 个。

附表 1 USART_InitTypeDef 成员 USART 模式对比

成员	异步模式	同步模式
USART_BaudRate 波特率	X	X
USART_WordLength 数据	X	X
USART_StopBits 停止位	X	X
USART_Parity 校验位	X	X
USART_HardwareFlowControl 控制	X	X
USART_Mode	X	X
USART_Clock		X
USART_CPOL		X
USART_CPHA		X
USART_LastBit		X

2、函数 USART_ClearFlag

表 2 USART_ClearFlag 说明

函数名	USART_ClearFlag
函数原型	Void USART_ClearFlag(USART_TypeDef*USARTx,u16 USART_FLAG)
功能描述	清除 USARTx 的待处理标志位
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_FLAG：待清除 USARTx 的标志位， 参阅 Section: USART_FLAG 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

附表 2 USART_FLAG 值

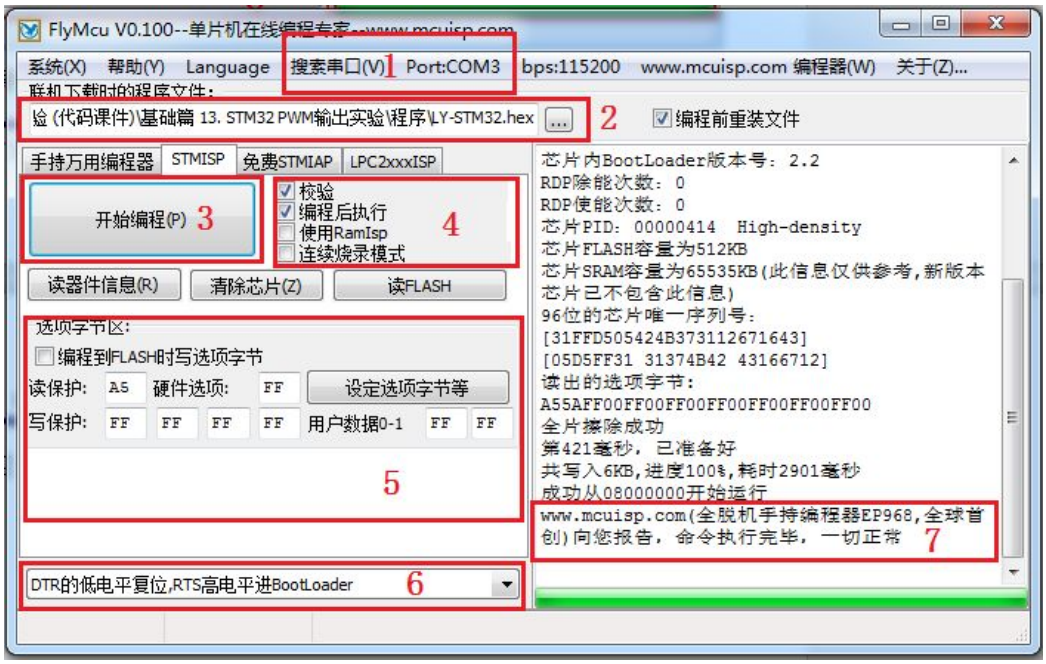
USART_FLAG	描述
USART_FLAG_CTS	CTS 标志位
USART_FLAG_LBD	LIN 中断检测标志位
USART_FLAG_TXE	发送数据寄存器空标志位
USART_FLAG_TC	发送完成标志位
USART_FLAG_RXNE	接收数据寄存器非空标志位
USART_FLAG_IDLE	空闲总线标志位
USART_FLAG_ORE	溢出错误标志位
USART_FLAG_NE	噪声错误标志位

USART_FLAG_FE	帧错误标志位
USART_FLAG_PE	奇偶错误标志位

在这个程序中我们要清除发送完成标志位（USART_FLAG_TC）。

4.7.10 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 15. STM32 RS232 串口通讯实验\程序）

4.7.11 实验效果图

串口通讯程序写入实验板后，使用公司开发的多功能监视系统，串口配置成（9600，无，8，1）在串口调试界面中的发送区输入随机数字，点击发送按钮，在接收区就可以看到发送过来的数据。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。