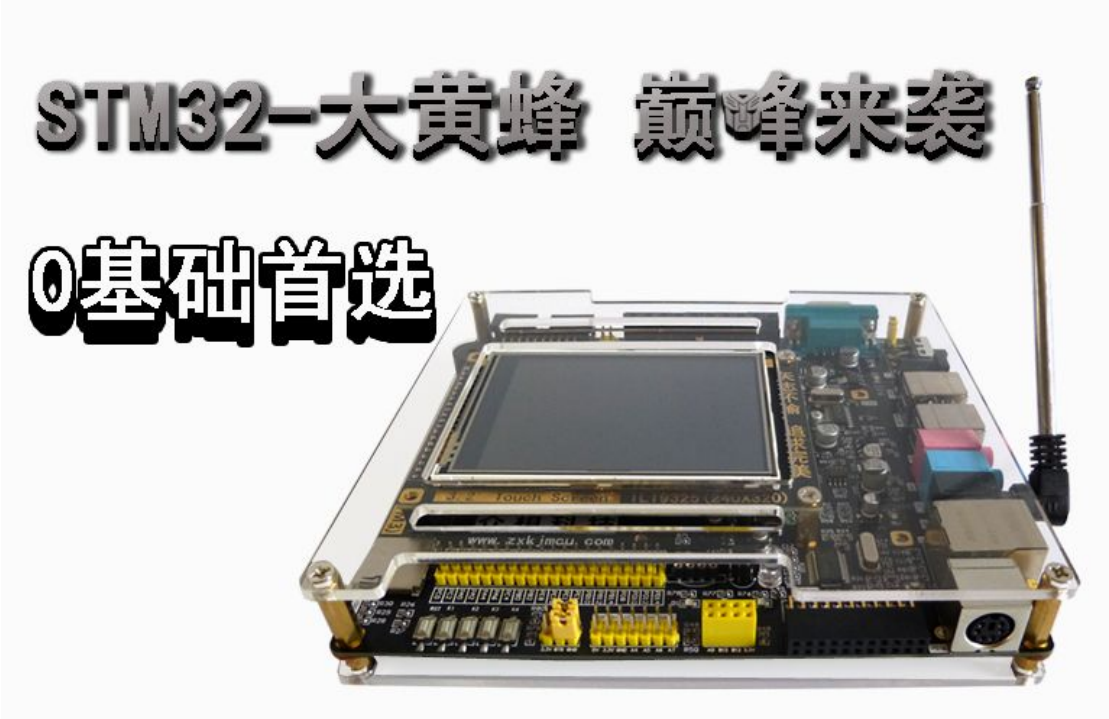
学 ARM 从 STM32 开始

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

4.19 STM32 FLASH 模拟 EEPROM 实验

4.19.1 概述

4.19.1.1 FLASH（嵌入式闪存）特点

高性能的闪存模块有以下的主要特性：

● 高达 512K 字节闪存存储器结构：闪存存储器有主存储块和信息块组成：

— 主存储块容量：

小容量产品主存储块最大为 4K×64 位，每个存储块划分为 32 个 1K 字节的页。

中容量产品主存储块最大为 16K×64 位，每个存储块划分为 128 个 1K 字节的页。

大容量产品主存储块最大为 64K×64 位，每个存储块划分为 256 个 2K 字节的页。

互联型产品主存储块最大为 32K×64 位，每个存储块划分为 128 个 2K 字节的页。

— 信息块容量：互联型产品有 2360×64 位。其它产品有 258×64 位。

闪存存储器接口的特性为：

● 带预取缓冲器的读接口(每字为 2×64 位)

● 选择字节加载器

● 闪存编程/擦除操作

● 访问/写保护 表 2 闪存模块的

表 1 闪存模块的组织(互联型产品)

模块	名称	地址	大小（字节）
主存储器	页 0	0x0800 0000-0x0800 07FF	2K
	页 1	0x0800 0800-0x0800 0FFF	2K
	页 2	0x0800 1000-0x0800 17FF	2K
	页 3	0x0800 1800-0x0800 1FFF	2K
	... ...	... ...	... ...
	页 127	0x0803 F800-0x0803 FFFF	2K
信息快	系统存储器	0x1FFF B000-0x1FFF F7FF	18K
	选择字节	0x1FFF F800-0x1FFF F80F	6
闪存存储器 接口寄存器	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FALSH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	保留	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

4.19.1.2 FLASH 读取

闪存读取 闪存的指令和数据访问是通过 AHB 总线完成的。预取模块是用于通过 ICode 总线读取指令的。仲裁是作用在闪存接口，并且 DCode 总线上的数据访问优先。

有关闪存寄存器的详细信息，请参考《STM32F10xxx 闪存编程手册》 闪存读取 闪存的指令和数据访问是通过 AHB 总线完成的。预取模块是用于通过 ICode 总线读取指令的。仲裁是作用在闪存接口，并且 DCode 总线上的数据访问优先。

读访问可以有以下配置选项：

- 等待时间：可以随时更改的用于读取操作的等待状态的数量。
- 预取缓冲区(2 个 64 位)：在每一次复位以后被自动打开，由于每个缓冲区的大小(64 位)与闪存的带宽相同，因此只通过需一次读闪存的操作

即可更新整个缓冲区的内容。由于预取缓冲区的存在，CPU 可以工作在更高的主频。CPU 每次取指最多为 32 位的字，取一条指令时，下一条指令已经在缓冲区中等待。

● 半周期：用于功耗优化。

1. 这些选项应与闪存存储器的访问时间一起使用。等待周期体现了系统时钟(SYSCLK)频率与闪存访问时间的关系：

0 等待周期，当 0 < SYSCLK < 24MHz

1 等待周期，当 24MHz < SYSCLK ≤ 48MHz

2 等待周期，当 48MHz < SYSCLK ≤ 72MHz

2．半周期配置不能与使用了预分频器的 AHB 一起使用，时钟系统应该等于 HCLK 时钟。该特性只能用在时钟频率为 8MHz 或低于 8MHz 时，可以直接使用的内部 RC 振荡器(HSI)，或者是主振荡器(HSE)，但不能用 PLL。

3. 当 AHB 预分频系数不为 1 时，必须置预取缓冲区处于开启状态。

4. 只有在系统时钟(SYSCLK)小于 24MHz 并且没有打开 AHB 的预分频器(即 HCLK 必须等于 SYSHCLK)时，才能执行预取缓冲器的打开和关闭操作。一般而言，在初始化过程中执行预取缓冲器的打开和关闭操作，这时微控制器的时钟由 8MHz 的内部 RC 振荡器(HSI)提供。

5. 使用 DMA:DMA 在 DCode 总线上访问闪存存储器,它的优先级比 ICode 上的取指高。DMA 在每次传送完成后具有一个空余的周期。有些指令可以和 DMA 传输一起执行。

4.19.2 STM32 编程和擦除闪存

闪存编程一次可以写入 16 位(半字)。

闪存擦除操作可以按页面擦除或完全擦除(全擦除)。全擦除不影响信息块。

为了确保不发生过度编程， 闪存编程和擦除控制器块是由一个固定的时钟控制的。

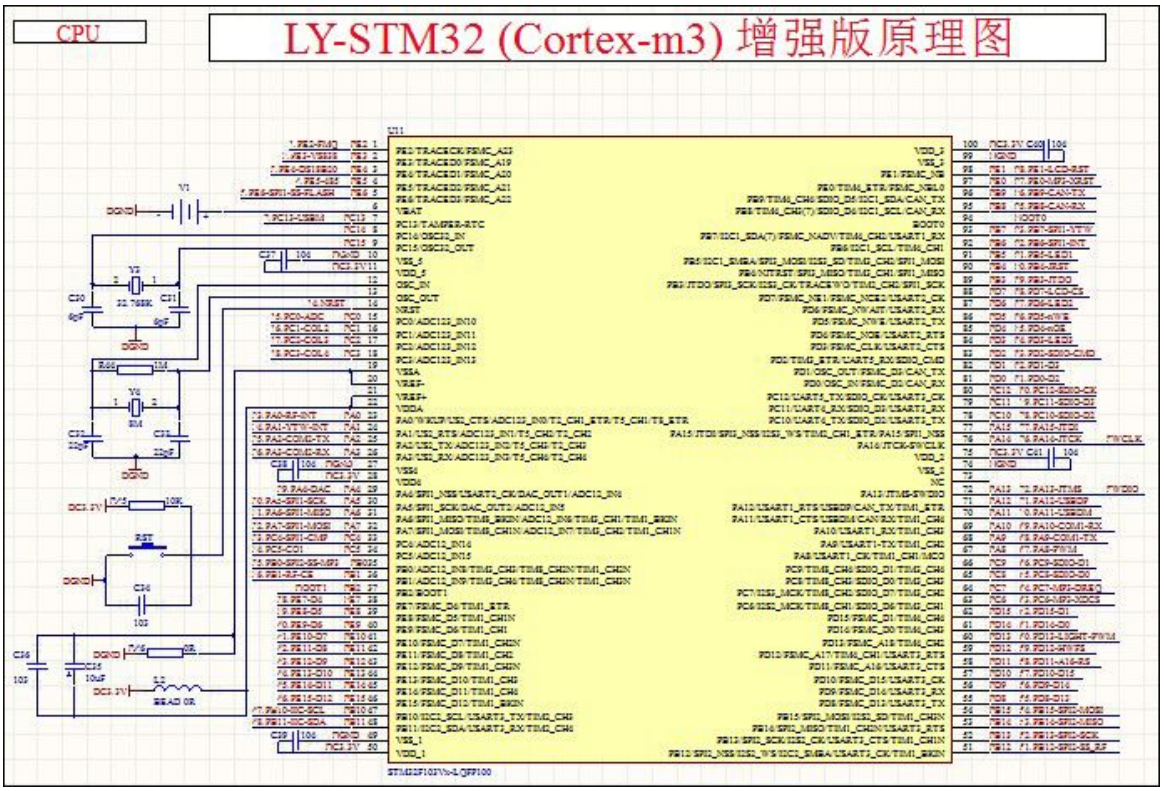
写操作(编程或擦除)结束时可以触发中断。仅当闪存控制器接口时钟开启时，此中断可以用来从 WFI 模式退出。

4.19.3 实验目的

通过下传编写好的 FLASH 读取程序，通过 prinif 重定向程序把写入 FLASH 的数据读出打印到串口精灵上，显示在显示器上。

4.19.4 硬件设计

利用实验板上的 CPU 和串口电路，可以很方便的实现这个功能。



4.19.5 软件设计

4.19.5.1 软件设计说明

这次试验是向 FLASH 中写入一组数据，然后读出来打印输出。

FLASH 读写程序设计和 prinif 重定向程序很相似，只有 3 点不同，下面对软件的设计做一次说明：

1、FLASH 解锁，因为 FLASH 不允许随便读取；

2、清标志；

3、写入数据，再读出数据。

4.19.5.2 FLASH 相关函数简介

FLASH 是很重要的，在下面我们着重介绍相关应用函数，先熟悉应用函数然后编写程序。

1、函数 FLASH_ClearFlag（清除 FLASH 待处理标志位）

表 1 描述了函数 FLASH_ClearFlag

函数名称	FLASH_ClearFlag
函数原型	void FLASH_ClearFlag(u16 FLASH_Flag)
功能描述	清除 FLASH 待处理标志位
输入参数	FLASH_FLAG：待清除的标志位 参阅 Section：FLASH_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

FLASH_FLAG 为能够被函数 FLASH_ClearFlag 清除的标志位。它们列举于下表：

附表 1 FLASH_FLAG 值

FLASH_FLAG	描述
FLASH_FLAG_BSY	FLASH 忙标志位
FLASH_FLAG_EOP	FLASH 操作结束标志位
FLASH_FLAG_PGERR	FLASH 编写错误标志位
FLASH_FLAG_WRPRTERR	FLASH 页面写保护错误标志位

2、函数 FLASH_ErasePage（要擦除页的起始地址）

表 2 描述了函数 FLASH_ErasePage

函数名称	FLASH_ErasePage

函数原型	FLASH_Status FLASH_ErasePage(u32 Page_Address)
功能描述	擦除一个 FLASH 页面
输入参数	无
输出参数	无
返回值	擦除操作状态
先决条件	无
被调用函数	无

3、函数 FLASH_ProgramWord（要擦除页的起始地

表 3 描述了函数 FLASH_ProgramWord

函数名称	FLASH_ProgramWord
函数原型	FLASH_Status FLASH_ProgramWord(u32 Address, u32 Data)
功能描述	在指定地址编写一个字
输入参数 1	Address：待编写的地址
输入参数 2	Data：待写入的数据
输出参数	无
返回值	编写操作状态
先决条件	无
被调用函数	无

4.19.5.3 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
stm32f10x_flash.c  // FLASH 读写函数
```

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，tm32f10x_flash.c 库函数主要包含 FLASH 读

写应用设置，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。

4.19.5.4 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。

4.19.5.5 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"
#include "stm32f10x_flash.h"
#include "sttdio.h"

extern u8 dt;//定义变量

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。

4.19.5.6 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

u8 dt=0;
```

```
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
}
```

4.19.6 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。

4.19.7 GPIO 引脚时钟使能

```
    SystemInit();//72m
    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。

4.19.8 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)

{
 }
```

4.19.9 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。

```
#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);
   }
}
```

4.19.10 main.c 文件里的内容是

大家都知道 prinif 重定向是把需要显示的数据打印到显示器上。在这个试验中主程序向 FLASH 中写入一组数据再读出来，然后通过 prinif 重定向打印到串口来验证写入数据的正确性。

大家在使用 FLASH 之前，一定要准确的计算一下程序所占用的空间，一定在 FLASH 中留出做够程序使用的空间，在程序空间之后才可以存储数据。这样不会无缘无故的出错，能很好的提高了编程和调试的效率。

```
#include "pbdata.h"

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

#define FLASH_ADR 0x0807F800//定义变量，把真实的地址“0x0807F800” 付给字符串
“ FLASH_ADR”
int fputc(int ch,FILE *f)
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
   u32 data=12345678;//编写要写入的数据

   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();
   NVIC_Configuration();

    FLASH_Unlock();//解锁
    FLASH_ClearFlag(FLASH_FLAG_BSY|FLASH_FLAG_EOP|
                    FLASH_FLAG_PGERR|FLASH_FLAG_WRPRTERR);//清除标志位

    FLASH_ErasePage(FLASH_ADR);//要擦除页的起始地址
    FLASH_ProgramWord(FLASH_ADR,data);//写数据
    //FLASH_ProgramWord(FLASH_ADR+4,data);
    FLASH_Lock();//锁定 FLASH

    data=0;//清除接收区，做好存储读出来数据的准备
    data=(*(__IO uint32_t*)(FLASH_ADR));//读 FLASH 可以直接读取，不需要解锁。
    printf("输出 data=%d\r\n",data); //打印输出到串口
     while(1);//等待
```

```
}

    void RCC_Configuration(void)
    {
        SystemInit();//72m
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    }

    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;
        GPIO_Init(GPIOA,&GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX
        GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOA,&GPIO_InitStructure);
    }

    void NVIC_Configuration(void)
    {
        NVIC_InitTypeDef NVIC_InitStructure;

        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

        NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        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;
```

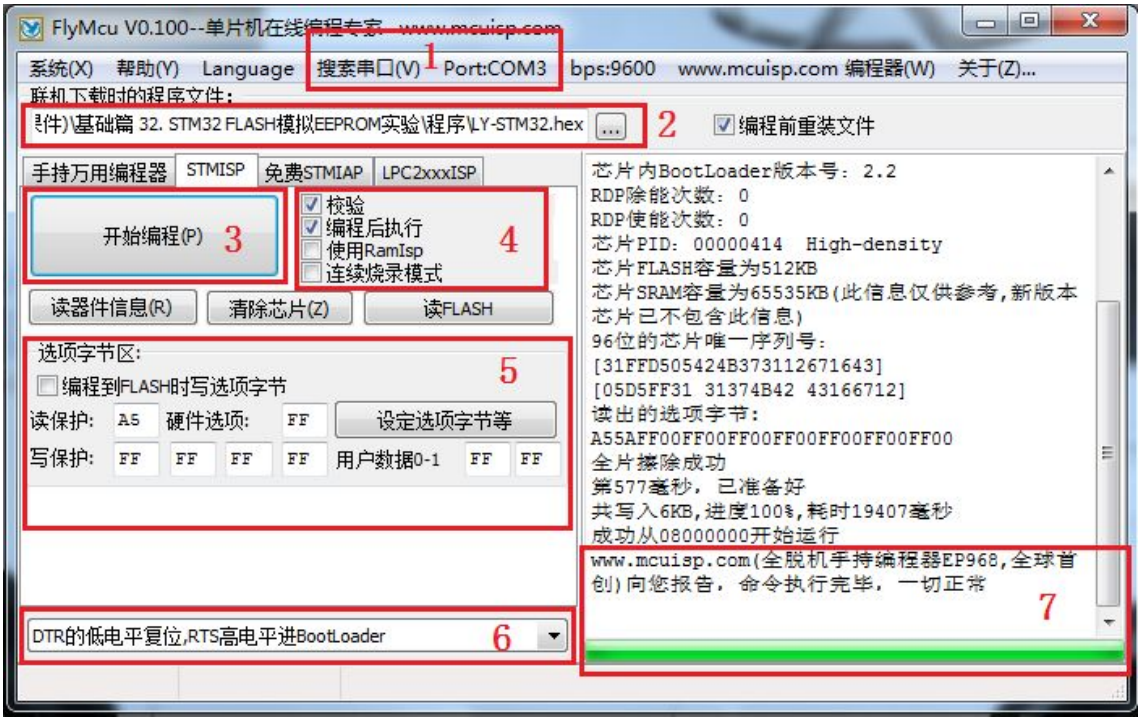
```
        USART_InitStructure.USART_Parity=USART_Parity_No;
        USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowContr
ol_None;
        USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

        USART_Init(USART1,&USART_InitStructure);
        USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
        USART_Cmd(USART1,ENABLE);
        USART_ClearFlag(USART1,USART_FLAG_TC);
    }
```

我们知道就在程序中有“\r\n”，它的意思是回车换行的意思，在程序语句尾部加上“\r\n”，在把数据打印到屏幕上时，会自动回车换上打印屏幕的。

在 main(void)程序体中代码比较多，大部分都是前期课程的延伸，新的知识点不多，请大家注意 FLASH 的设置和读写顺序。

4.19.11 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 COM2。（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。

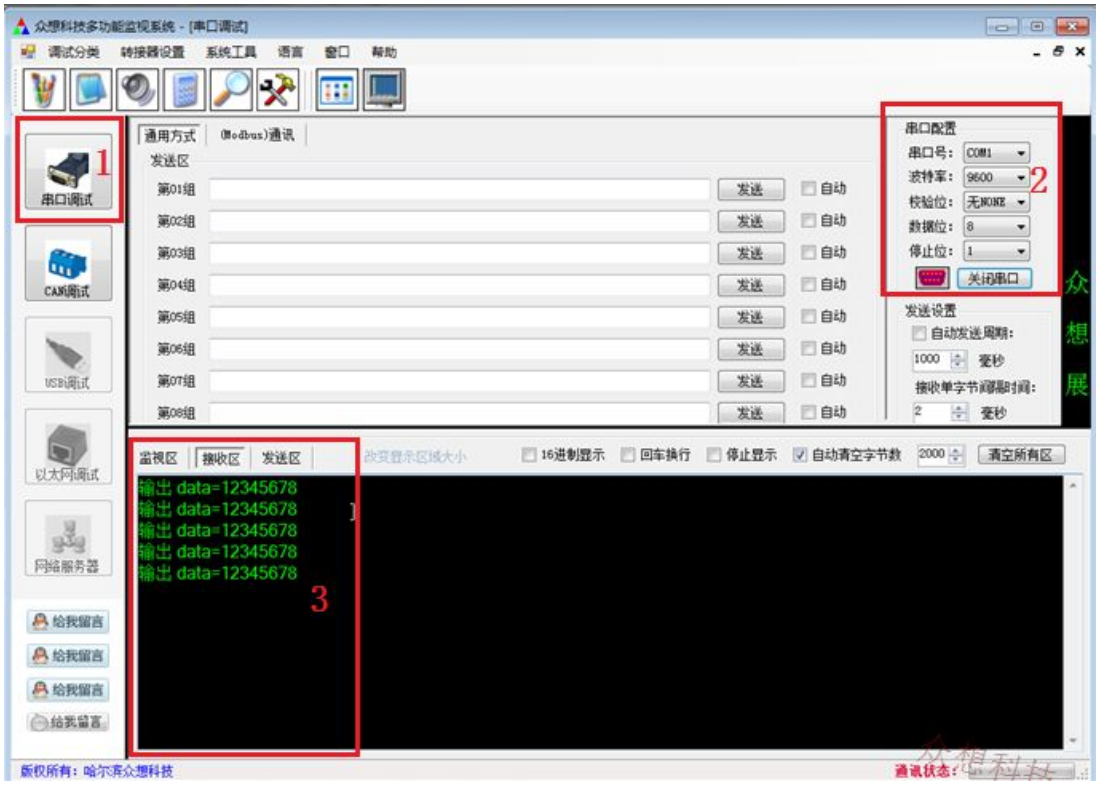


本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\1,基础篇\基础篇 32. STM32 FLASH 模拟 EEPROM 实验\程序）

4.19.12 实验效果图

初始化程序写入实验板后，就等待中断的到来。使用公司开发的多功能监视系统，在串口调试界面中的接收区就会就会接收到 FLASH 中读出的数据。请大家注意：就写入一次，也读出一次，然后打印到串口调试界面中。如果还想重复打印输出的效果，就要按复位键，是整个实验系统复位重新开始写、读 FLASH 的操作。

下面的实验画面中的数据是经过 5 次复位，5 次读出来的 FLASH 数据。


串口调试
CAN调试
USB调试
以太网调试
网络服务器
通用方式
(Modbus)通讯
发送区
串口配置
串口号: COM1
波特率: 9600
校验位: 无NONE
数据位: 8
停止位: 1
关闭串口
发送设置
自动发送周期:
1000
毫秒
接收单字节超时时间:
2
毫秒
监视区
接收区
发送区
16进制显示
回车换行
停止显示
自动清空字节数
2000
清空所有区
输出 data=12345678
输出 data=12345678
输出 data=12345678
输出 data=12345678
输出 data=12345678
1
2
3
众想
展
版权所有：哈尔滨众想科技