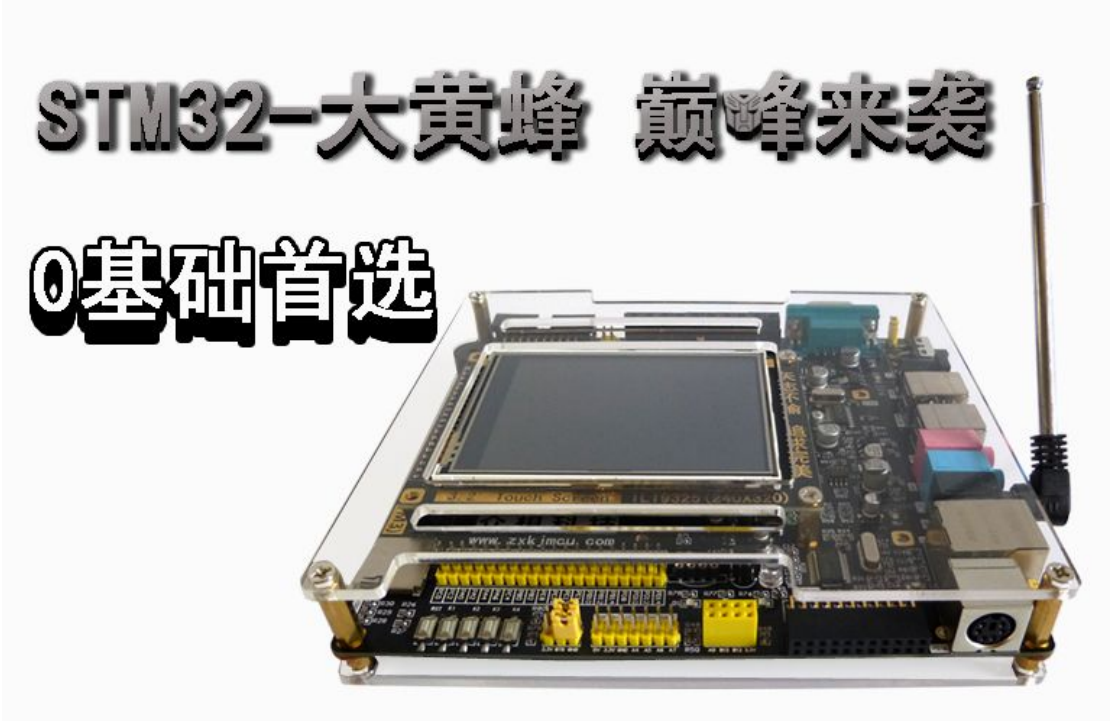
学 ARM 从 STM32 开始

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

4.28 STM32 SPI 读写 SST25VF016B(硬件接口 SPI)实验

4.28.1 概述

前面的文章已经详细的介绍了 SST25VF016B 的工作方式，详细请参考《4.27 STM32 外设篇-SST25VF016B 工作原理(SPI 接口 FLASH)》。以下主要介绍具体的操作过程。FLASH 芯片在大黄蜂实验板上的位置如下图蓝色框图 2 所示。

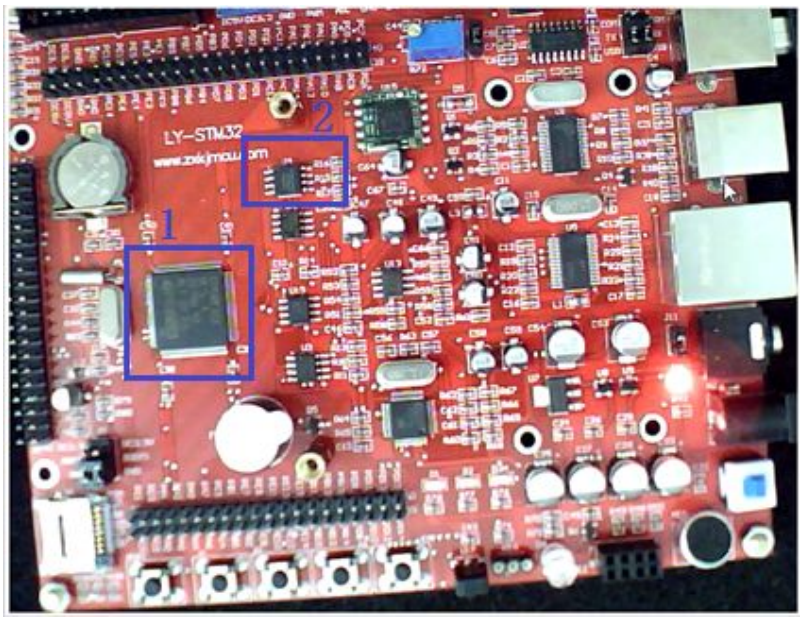


图 4.28.1 大黄蜂实验板上图外扩的 FLASH 芯片

4.28.2 实验目的

把我们组织好的数据写入 FLASH 中，再读回来。通过 printf 重定向打印输出到显示器显示出来。我们组织的内容是："LY-STM32 主讲人：刘洋 视频教程下载地址 www.zxkjmcu.com" 。

4.28.3 硬件设计

外扩的FLASH芯片采用的是SPI 工作方式，从硬件电路设计上可以看出，SST25VF016B 芯片 1 脚是片选信号、2 脚是 MISO 信号、4 脚是 MOSI 信号、6 脚是时钟信号。硬件设计见图 4.28.2 外扩的 FLASH 芯片原理图。

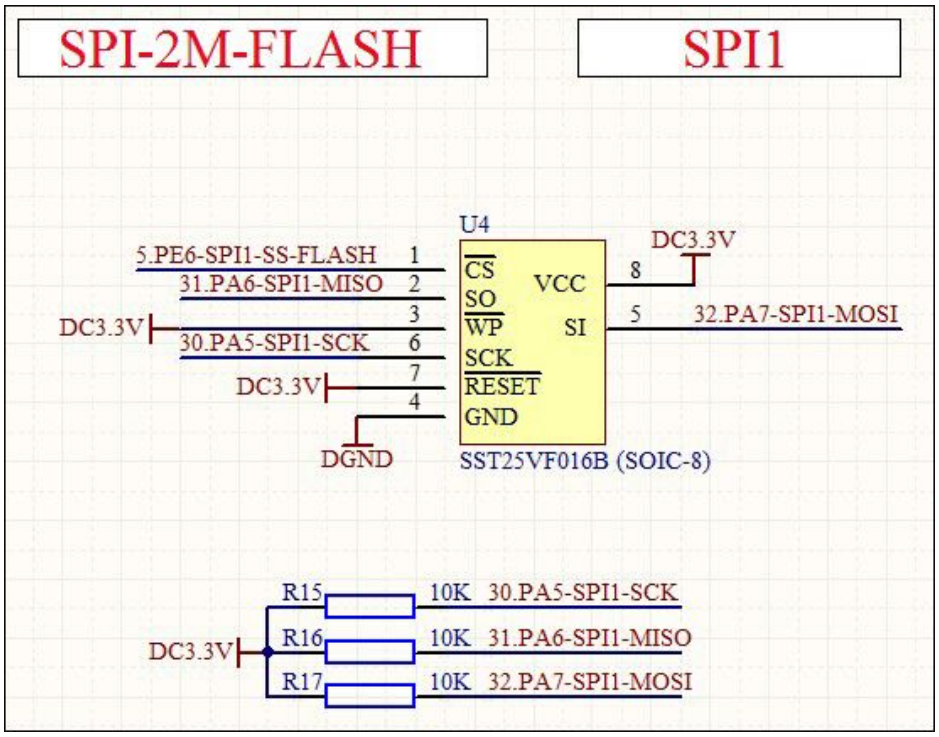


图 4.28.2 外扩的 FLASH 芯片原理图

4.28.4 软件设计

4.28.4.1 软件设计说明

利用实验板上集成的 SST25VF016B 芯片电路，采用 SPI 通讯方式，读写 SST25VF016B 芯片。然后送到串口打印输出到计算机显示出来。这套程序严格按照 SPI 的工作时序编写。在这个程序中我们要禁止触摸屏和以太网的 SPI 功能，避免冲突。

在这节程序设计中，用到了外部中断函数；prinif 重定向打印输出函

数； USART 串口通讯函数；定时器函数等。

4.28.4.2 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_spi.c  // SPI 通讯函数
```

以上库文件包含了本次实验所有要用到的函数功能。

4.28.4.3 自定义头文件

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

我们已经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。

4.28.4.4 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_spi.h"
#include "stdio.h"
#include <string.h>

#include "stm32_spi.h"   //自定义的 stm32_spi 专属函数
#include "SST25VF016B.h" //自定义的 SST25VF016B 专属函数

extern u8 dt;       //定义中间变量
extern u8 TxBuffer2[4096]; //定义中间变量

void RCC_HSE_Configuration(void);  //定义函数
void delay(u32 nCount);         //定义函数
```

```
void delay_us(u32 nus);         //定义函数
void delay_ms(u16 nms);         //定义函数

#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。“stm32_spi.h”和“SST25VF016B.h”是我们自定义的，为了是在大程序中方便移植和功能分类独立新建。

4.28.4.5 pbdata.c 文件里的内容是

下面是 pbdata.c 文件详细内容，在文件开始还是引用“pbdata.h”文件。

```
    #include "pbdata.h"

    u8 dt=0;    //中间变量赋值
    u8 TxBuffer2[4096];    //中间变量赋值

    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)
{
    //注意 delay_ms 函数输入范围是 1-1863
```

```
        //所以最大延时为 1.8 秒
        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
    }
```

4.28.5 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。

4.28.6 GPIO 引脚时钟使能

```
        SystemInit();//72m
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
        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
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,ENABLE);//功能复用 SPI 时钟
使能
    }
```

本节实验用到了 PA 端口、PB 端口、PC 端口、PE 端口，所以要打开这些端口的使能；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；重要的是一定要打开 SPI 功能服用，SPI 才能正常工作。

4.28.7 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，每次进入 while(1)循环语句就是延时 1 秒就退出循环。

```
while(1)
   {
      delay_ms(1000);
   }
```

4.28.8 stm32f10x_it.c 文件里的内容

在中断处理 stm32f10x_it.c 文件里中有串口 1 子函数。

```
#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);
   }
}
```

4.28.9 stm32_spi.h 文件里的内容是

函数 stm32_spi.h 在这里是为了声明 SPI 模式形式,方便其他程序调用。它是一个通用的 SPI 声明函数。stm32_spi.h 的内容如下：

```
#ifndef _STM32_SPI_H
#define _STM32_SPI_H
```

```
void SPI_Configuration(void);//声明 SPI 初始化函数
u8 SPI_SendByte(u8 byte);//声明 SPI 读写函数

#endif
```

4.28.10 stm32_spi.c 文件里的内容是

自定义函数 stm32_spi.c 函数中，他是读写 SST25VF016B 芯片的核心程序，基本的时序都体现在这个程序中。其内容如下：

```
#include "pbdata.h"   //引用头文件，这是必须的

void SPI_Configuration(void)   //SPI 初始化函数
{
   SPI_InitTypeDef   SPI_InitStructure;
   GPIO_InitTypeDef     GPIO_InitStructure;

   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
   GPIO_Init(GPIOA,&GPIO_InitStructure);

   //enc28j60 以太网初始化，
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_7;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;  //推挽输出
   GPIO_Init(GPIOB,&GPIO_InitStructure);

   //触摸屏初始化
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_4;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
   GPIO_Init(GPIOC,&GPIO_InitStructure);

   //flash 片选
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_6;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
   GPIO_Init(GPIOE,&GPIO_InitStructure);
```

```
    SPI_InitStructure.SPI_Direction=SPI_Direction_2Lines_FullDuplex;//SPI
设置为双线双向全双工
    SPI_InitStructure.SPI_Mode=SPI_Mode_Master;//设置为主 SPI
    SPI_InitStructure.SPI_DataSize=SPI_DataSize_8b;//SPI 发送接收 8 位帧结
构
    SPI_InitStructure.SPI_CPOL=SPI_CPOL_High;//设置时钟悬空高
    SPI_InitStructure.SPI_CPHA=SPI_CPHA_2Edge;//数据捕获于第二个时钟沿
    SPI_InitStructure.SPI_NSS=SPI_NSS_Soft;//内部 NSS 信号有 SSI 位控制
    SPI_InitStructure.SPI_BaudRatePrescaler=SPI_BaudRatePrescaler_8;//波
特率预分频值为 8
    SPI_InitStructure.SPI_FirstBit=SPI_FirstBit_MSB;//数据传输从 MSB 位开
始
    SPI_InitStructure.SPI_CRCPolynomial = 7;//SPI_CRCPolynomial 定义了用于
CRC 值计算的多项式
    SPI_Init(SPI1,&SPI_InitStructure);//参数传送

    SPI_Cmd(SPI1,ENABLE); //使能 SPI

    //enc28j60 禁止以太网片选，在这个实验中不使用以太网，必须禁止，以免通
讯出错。
    GPIO_SetBits(GPIOB,GPIO_Pin_7);
    //禁止触摸屏片选
    GPIO_SetBits(GPIOC,GPIO_Pin_4);
    //禁止 FLASH 片选
    GPIO_SetBits(GPIOE,GPIO_Pin_6);
  }

  u8 SPI_SendByte(u8 byte) //SPI 读写函数
  {
    while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_TXE)==RESET);//判断发送
缓冲区是否忙?
    SPI_I2S_SendData(SPI1,byte);//发送数据

    while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_RXNE)==RESET);//判断接
收缓冲区是否是空的。
    return SPI_I2S_ReceiveData(SPI1);//接收数据，并返回到接收缓冲区
  }
```

4.28.11 sst25vf016b.h 文件里的内容是

函数 sst25vf016b.h 在这里是针对于 sst25vf016b 芯片自定义的一份专

用于这款芯片的头文件。sst25vf016b.h 的内容如下：

```
    #ifndef _SST25VF016B_H
    #define _SST25VF016B_H

#define SPI_FLASH_L GPIO_ResetBits(GPIOE,GPIO_Pin_6);//针对这款芯片重新定义片选的中间变量，方便与程序移植

#define SPI_FLASH_H GPIO_SetBits(GPIOE,GPIO_Pin_6);//针对这款芯片重新定义片选的中间变量，方便与程序移植

    u8 RDSR(void);
    void EWSR(void);
    void WREN(void);
    void WRDI(void);
    void BUSY(void);
    u16 RDID(void);
    void Section_Dell(u32 addr);
    void Section_Read(u32 addr,u8 *buffer,u16 Size);
    void Section_Write(u32 addr,u8 *buffer,u16 Size);
    void Section_All_dell(void);

    #endif
```

4.28.12 sst25vf016b.c 文件里的内容是

函数 sst25vf016b.c 在这里是针对于 sst25vf016b 芯片自定义的一份专用于这款芯片的库函数。他是读写 SST25VF016B 芯片的核心程序，基本的时序都体现在这个程序中。sst25vf016b.c 的内容如下：

```
    #include "pbdata.h"

    //读取状态寄存器
    u8 RDSR(void)
    {
       u8 dt;
       SPI_FLASH_L;
       SPI_SendByte(0x05);
       dt=SPI_SendByte(0);
       SPI_FLASH_H;
```

```
    return dt;
}
//使能写状态寄存器 EWSR、写状态寄存器 WRSR

void EWSR(void)
{
    SPI_FLASH_L;
    SPI_SendByte(0x50);
    SPI_FLASH_H;
    SPI_FLASH_L;
    SPI_SendByte(0x01);
    SPI_SendByte(0);
    SPI_FLASH_H;
    BUSY();
}

//写使能
void WREN(void)
{
    SPI_FLASH_L;
    SPI_SendByte(0x06);
    SPI_FLASH_H;
}

//写禁止
void WRDI(void)
{
     SPI_FLASH_L;
    SPI_SendByte(0x04);
    SPI_FLASH_H;
    BUSY();
}

//忙检测
void BUSY(void)
{
    u8 a=1;
    while((a&0x01)==1) a=RDSR();
}
//读 ID
u16 RDID(void)
{
    u16 id=0;
```

```
    SPI_FLASH_L;
    SPI_SendByte(0x90);
    SPI_SendByte(0);
    SPI_SendByte(0);
    SPI_SendByte(0);

    id=SPI_SendByte(0);
    id=id<<8;//ID 向左移动 8 位，数据移到高 8 位上了。
    id=id+SPI_SendByte(0);//再和低 8 位数据相加，这样整个 16 位 ID 号读取完毕
    SPI_FLASH_H;

    return id;
}

//扇区擦除
void Section_Dell(u32 addr)
{
    EWSR();//使能写状态寄存器
    WREN();//写使能

    SPI_FLASH_L;
    SPI_SendByte(0x20);
    SPI_SendByte((addr&0xFFFFFF)>>16);//因为要发送 3 个字节的数据，所以要与
上 FFFFFF，向右移动 16 位是先把第 1 和第 2 个字节移掉，先发送第 3 个字节。
    SPI_SendByte((addr&0xFFFF)>>8);//接着发送第 2 个字节的数据，所以要与上
FFFF，向右移动 8 位是先把第 1 个字节移掉，发送第 2 个字节。
    SPI_SendByte(addr&0xFF);//就剩下第 1 个字节了，直接发送。
    SPI_FLASH_H;
    BUSY();//忙检测
}

//扇区读
void Section_Read(u32 addr,u8 *buffer,u16 Size)
{
    u16 i=0;

    SPI_FLASH_L;
    SPI_SendByte(0x0B);
    SPI_SendByte((addr&0xFFFFFF)>>16);
    SPI_SendByte((addr&0xFFFF)>>8);
    SPI_SendByte(addr&0xFF);
    SPI_SendByte(0);

    while(i<Size)
```

```
    {
        buffer[i]=SPI_SendByte(0);
        i++;
    }
    SPI_FLASH_H;
}

//写扇区
void Section_Write(u32 addr,u8 *buffer,u16 Size)
{
    u16 i=0;

    Section_Dell(addr);//先擦除

    EWSR();//使能写状态寄存器
    WREN();//写使能

    SPI_FLASH_L;
    SPI_SendByte(0xAD);
    SPI_SendByte((addr&0xFFFFFF)>>16);
    SPI_SendByte((addr&0xFFFF)>>8);
    SPI_SendByte(addr&0xFF);
    SPI_SendByte(buffer[0]);
    SPI_SendByte(buffer[1]);
    SPI_FLASH_H;

    i=2;
    while(i<Size)
    {
        delay_us(10);
        SPI_FLASH_L;
        SPI_SendByte(0xAD);
        SPI_SendByte(buffer[i++]);
        SPI_SendByte(buffer[i++]);
        SPI_FLASH_H;
    }

    delay_us(10);
    WRDI();//退出 AAI 模式
    BUSY();
}

//全芯片擦除
void Section_All_dell(void)
```

```
    {
       EWSR();//使能写状态寄存器
       WREN();//写使能

       SPI_FLASH_L;
       SPI_SendByte(0x60);
       SPI_FLASH_H;
       BUSY();
    }
```

4.28.13 main.c 文件里的内容是

```
#include "pbdata.h"

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

int fputc(int ch,FILE *f)
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
    u16 id=0;
    u16 TxSize1=0;
    u16 i=0;

    u8 TxBuffer1[]="LY-STM32 主讲人：刘洋 视频教程下载地址 www.zxkjmcu.com";
    TxSize1=sizeof(TxBuffer1)/sizeof(TxBuffer1[0]);

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

    id=RDID();
```

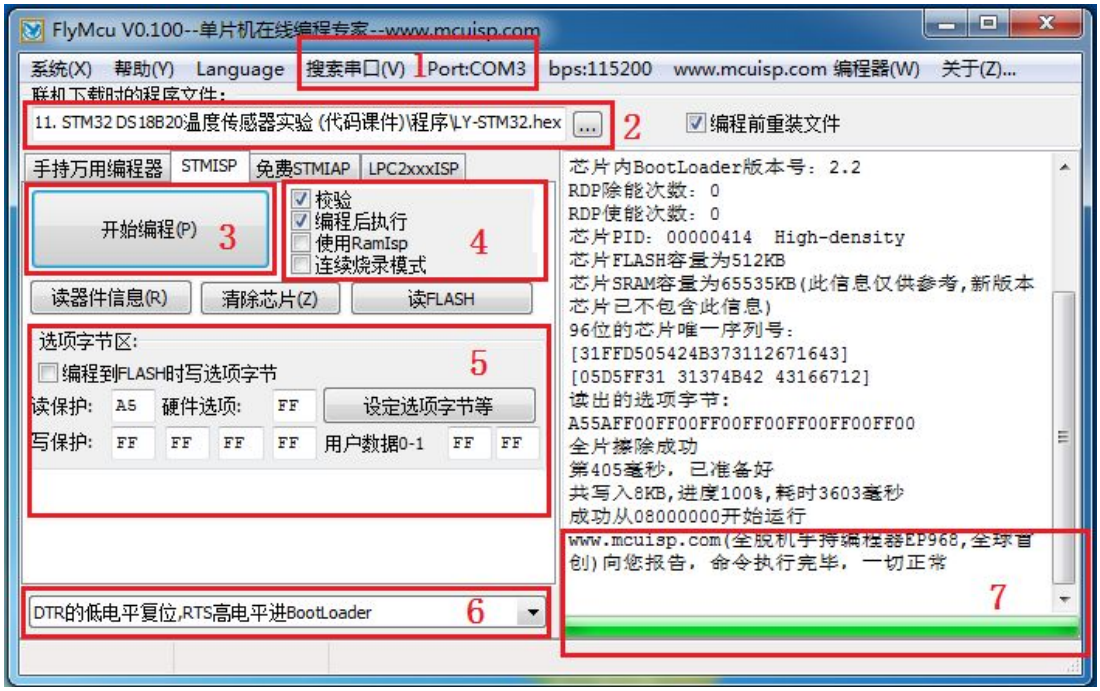
```
    printf("%X\r\n",id);

    for(i=0;i<TxSize1;i++)
    {
        TxBuffer2[i]=TxBuffer1[i];
    }

    Section_Write(0,TxBuffer2,4096);//写入 flash
    memset(TxBuffer2,0,sizeof(TxBuffer2));

    //Section_All_dell();

    delay_ms(100);
    Section_Read(0,TxBuffer2,4096);//读取 flash
    printf("%s\r\n",TxBuffer2);

    while(1)
    {
        delay_ms(1000);
    }
}

void RCC_Configuration(void)
{
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,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_HardwareFlowControl_
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);
}
```

4.28.14 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\2,外设篇\14. SPI 读写 SST25VF016B(硬件接口 SPI)\程序）

4.28.15 实验效果图

4.28.15.1 读写扩展闪存实验效果

在这个实验中我们打开众想科技多功能监控软件，选择 1 号区域串口通讯功能，在 2 号区域设置通讯格式，这时打开串口，在 3 号区域是可以观察到从扩展闪存芯片中读回来的数据。

从图 4.28.15.1 读写 FLASH 芯片实验效果图中可以看出，读回来的数据都是“LY-STM32 主讲人：刘洋 视频教程下载地址 www.zxkjmcu.com ”，说明我们的程序设计正确，写入扩展的闪存成功，读回来的数据也很正常。

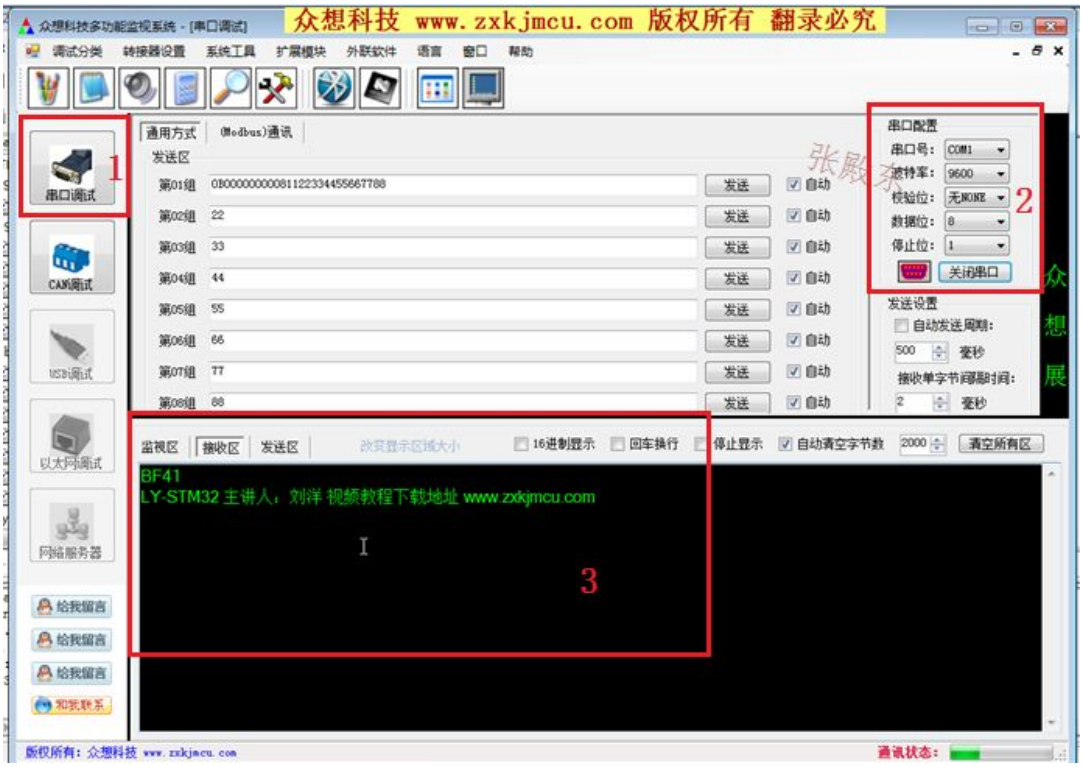


图 4.28.15.1 读写 FLASH 芯片实验效果图

4.28.15.2 全片擦除实验效果

全片擦除实验，使用 Section_All_dell()函数，实现全片擦除。

```
Section_All_dell();

delay_ms(100);
Section_Read(0,TxBuffer2,4096);//读取 flash
printf("%s\r\n",TxBuffer2);
```

从图 4.28.15.2 全片擦除 FLASH 芯片实验效果图中可以看出，在区域 3 接收区中读回来的数据全部为 4096 个“1”，只有末尾是“0D0A”。“0D0A”是什么意思呢？“0D0A”是“\r\n”十六进制显示。说明扩展闪存全片都被擦除，原来的数据都已经不复存在了。从这个实验中我们也知道清除就是把内存全部写成“1”。

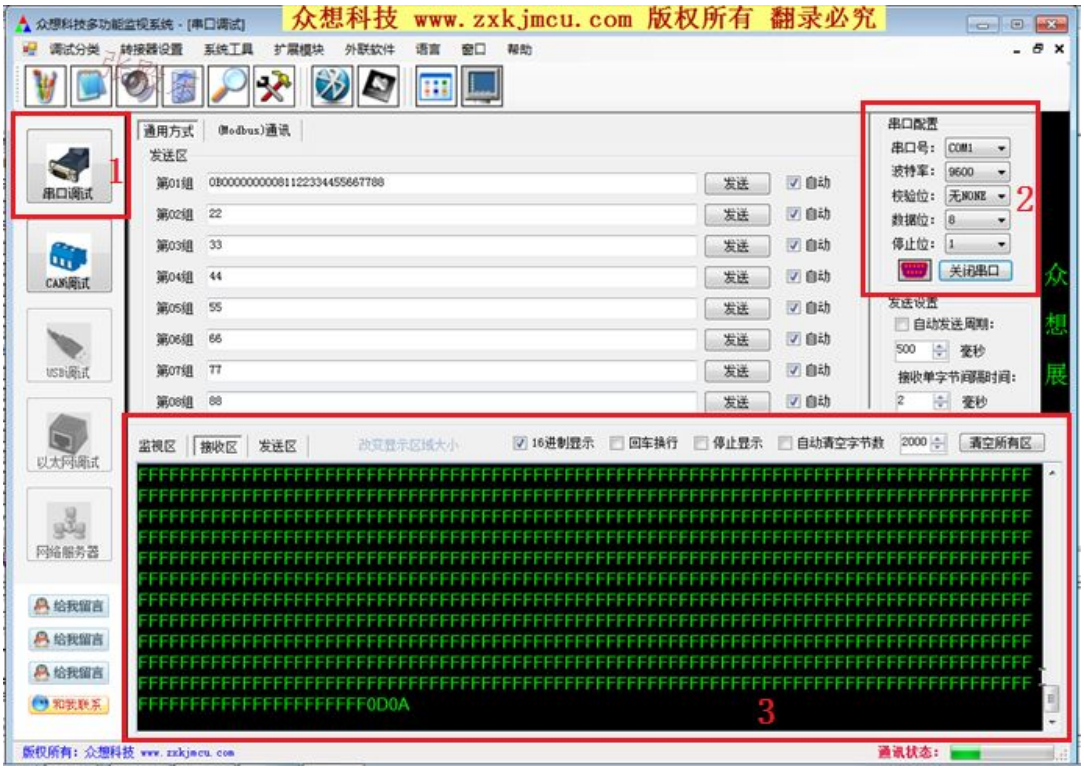


图 4.28.15.2 全片擦除 FLASH 芯片实验效果图