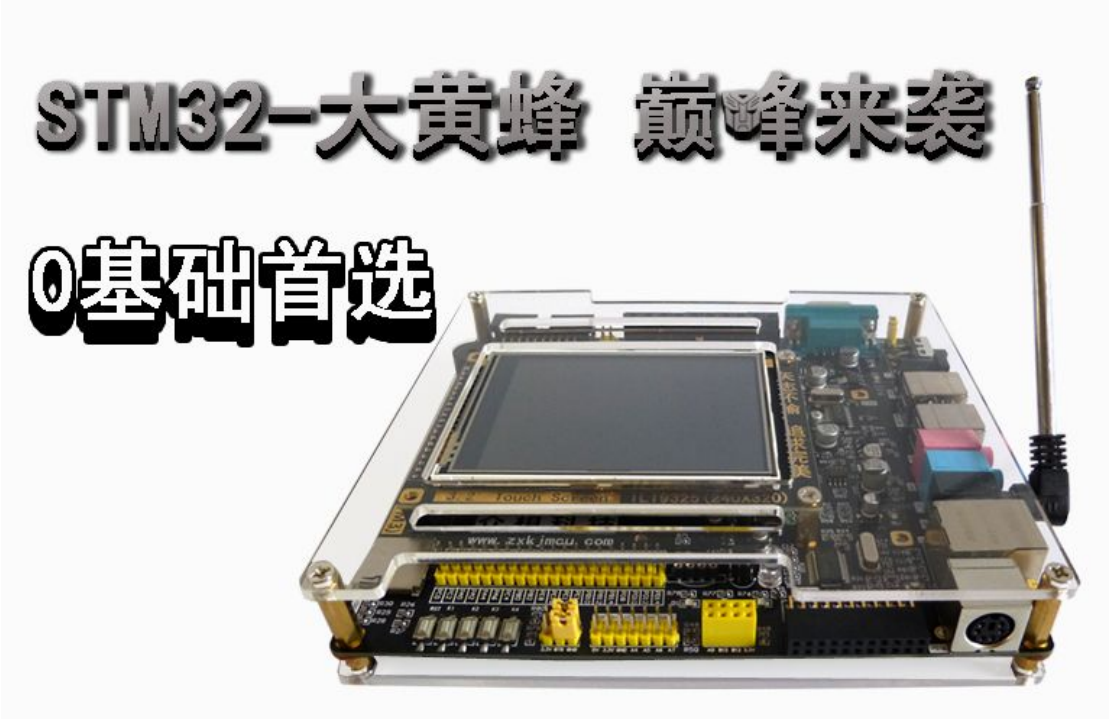
学 ARM 从 STM32 开始

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

4.31 STM32 驱动 LCD 彩色液晶屏实验

4.31.1 概述

4.31.1.1 彩色 LCD 液晶屏分辨率

关于液晶屏的图象分辨率，许多厂家的标注方法不同，象 320×234，有的液晶屏资料上标注为 960×234，这实际上是将 R、G、B 三基色乘上了 320。即 3×320=960。同样地，7" 16:9 的屏有的标为 480×234，有的标为 1440×234，它也是将 3×480=1440 而得出的。图象的分辨率指标主要是看垂直方向的线数，比如，两个分别标有 800×480 和 1440×234 的 7"液晶屏，哪个像素点多，分辨率高呢？显然应该是 800×480 的分辨率高，它是数字屏，可以支持 VGA 输入。那么是不是数字屏就分辨率高呢？也不尽然。象夏普 LM32C041，EPSON 4"、5.6"、6.5"，ALPS LFUBK9111A/LFUBK3041A 虽然是数字屏，但其分辨率也只有 320×234。

另外一个问题是：如何区分 STN 屏（伪彩屏）和 TFT 屏（真彩屏）呢？STN 由于工艺技术比较落后，其彩色鲜艳度，色彩还原性，图象响应速度，图象观看角度等与 TFT 屏相比，都有明显的差距，两种类型的屏放在一起，很容易区分出来。早期的 STN 屏响应速度很慢，在播放动态图象时，会有明显的拖尾现象，只适合显示静态图象。但象卡西欧 CMV54NT04P，CMW72NS46P，西铁城 USC-501/504/610 和夏普 LM6Q401，LM072QCAT50，虽然也是 STN 屏，但由于采用了新的技术，提高了响应速度和色彩鲜艳度，使许多新手误把它看成了真彩屏。另外，大多数液晶屏通过其型号也能看出是 STN 还是 TFT 屏，如夏普液晶屏"LQ"字头的一般是 TFT 真彩屏，"LM"

字头的是 STN 伪彩屏。

4.31.1.2 彩色 LCD 液晶屏实验步骤

1.打开 FSMC 外设时钟

2.配置 FSMC 的 GPIO 管脚状态

3.FSMC 初始化配置

4.调用 LCD 彩色液晶屏初始化函数

5.编写 LCD 彩色液晶屏读写函数(命令/数据)

6.发送数据到 LCD 彩色液晶屏的缓存里并显示

4.31.2 实验目的

把我们组织好的数据写入彩色液晶屏中，使彩色液晶屏每间隔 1 秒刷新一遍，在红色、绿色、黄色、白色和黑色之间来回转换，通过这种方法来验证我们程序设计是否正确。

4.31.3 硬件设计

彩色液晶屏和 CPU 之间采用的是 SPI 通讯方式，从硬件电路设计上可以看出，外扩彩色液晶屏和大黄蜂实验板通过 40 位的插座产生电气连接。具体的引脚规定在《4.29 STM32 外设篇-LCD 彩色液晶屏工作原理》中已经做了详细的介绍。

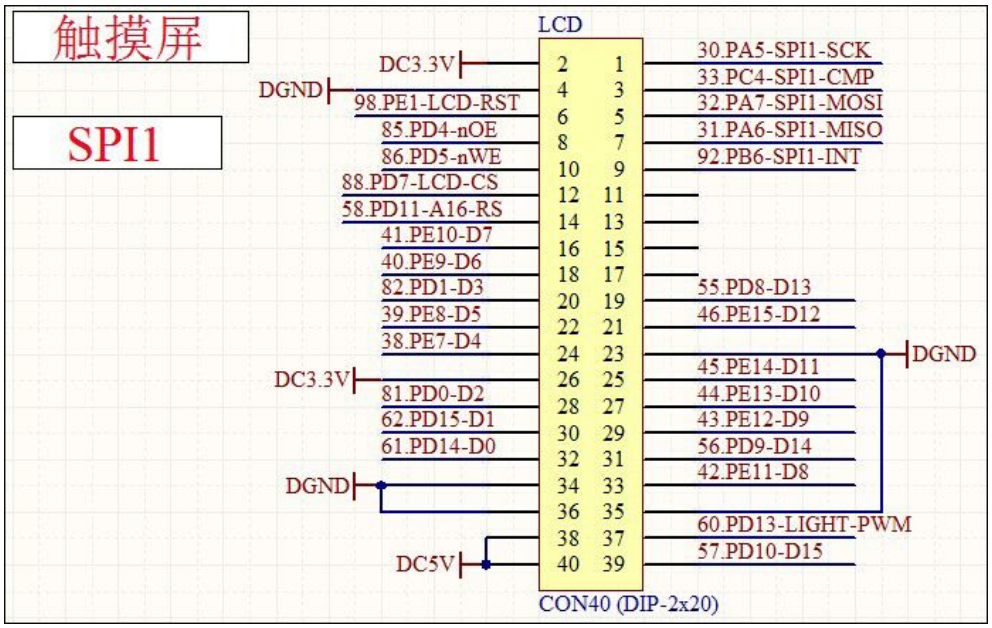


图 4.31.1 外扩触摸屏接口槽原理图

4.31.4 软件设计

4.31.4.1 软件设计说明

1、采用 SPI 通讯方式。

2、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；USART 串口通讯函数；定时器函数等。

4.31.4.2 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_fsmc.c  // FSMC 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。

4.31.4.3 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。

4.31.4.4 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 "stm32_fsmc.h" //自定义的 stm32_fsmc 专属函数
#include "lcd_ILI9325.h" //自定义的 LCD-ILI9325 专属函数

extern u8 dt;       //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_fsmc.h”和“lcd_ILI9325.h”是我们自定义的，为了是在大程序设计中方便移植和功能分类而独立新建。

4.31.4.5 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
}
```

4.31.5 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，这点很重要。

4.31.6 GPIO 引脚时钟使能

```
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,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_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,ENABLE);//功能复用 FSMC 时钟
使能
    }
```

本节实验用到了 PA 端口、PD 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 FSMC 功能服用，FSMC 时钟是 AHB 产生，这点要注意。

4.31.7 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);
     ILI_9325_CLEAR(RED);
     delay_ms(1000);
     ILI_9325_CLEAR(GREEN);
     delay_ms(1000);
     ILI_9325_CLEAR(BLUE);
     delay_ms(1000);
     ILI_9325_CLEAR(BLACK);
     delay_ms(1000);
     ILI_9325_CLEAR(WHITE);
  }
```

4.31.8 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);
   }
}
```

4.31.9 stm32_fsmc.h 文件里的内容是

函数 stm32_fsmc.h 在这里是为了声明 FSMC 模式形式，方便其他程序调用。它是一个通用的 FSMC 声明函数。stm32_fsmc.h 的内容如下：

```
#ifndef _STM32_FSMC_H
#define _STM32_FSMC_H

#include "pbdata.h"
void FSMC_Configuration(void);//声明 FSMC 初始化函数
#endif
```

4.31.10 stm32_fsmc.c 文件里的内容是

自定义函数 stm32_fsmc.c 函数中，它是配置 FSMC 功能的核心程序，基本的时序都体现在这个程序中。这段程序编写起来很复杂，需要查找很多资料，想在网络发达，我们在网上找到了一份比较详细的关于 FSMC 功能配置的说明，请参考附件《FSMC 配置》文档。在这里函数 stm32_fsmc.c 其内容如下：

```
#include "pbdata.h"

void FSMC_Configuration(void)
{
  FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
  FSMC_NORSRAMTimingInitTypeDef  p;

  p.FSMC_AddressSetupTime = 0x02;//地址建立时间
  p.FSMC_AddressHoldTime = 0x00;//地址保持时间
  p.FSMC_DataSetupTime = 0x05;//数据建立时间
  p.FSMC_BusTurnAroundDuration = 0x00;//总线恢复时间
  p.FSMC_CLKDivision = 0x00;//时钟分频
  p.FSMC_DataLatency = 0x00;//数据保持时间
  p.FSMC_AccessMode = FSMC_AccessMode_B;

  //NOR FLASH 的 BANK1
  FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM1;
  //数据线与地址线不复用
  FSMC_NORSRAMInitStructure.FSMC_DataAddressMux =
FSMC_DataAddressMux_Disable;
  //存储器类型 NOR FLASH
  FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_NOR;
  //数据宽度为 16 位
  FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth =
FSMC_MemoryDataWidth_16b;
  //使用异步写模式，禁止突发模式
  FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =
FSMC_BurstAccessMode_Disable;
  //本成员的配置只在突发模式下有效，等待信号极性为低
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity =
FSMC_WaitSignalPolarity_Low;
  //禁止非对齐突发模式
  FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
  //本成员配置仅在突发模式下有效。NWAIT 信号在什么时期产生
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive =
```

```
FSMC_WaitSignalActive_BeforeWaitState;
      //本成员的配置只在突发模式下有效，禁用 NWAIT 信号
      FSMC_NORSRAMInitStructure.FSMC_WriteOperation =
FSMC_WriteOperation_Enable;
      //禁止突发写操作
      FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
      //写使能
      FSMC_NORSRAMInitStructure.FSMC_ExtendedMode =
FSMC_ExtendedMode_Disable;
      //禁止扩展模式，扩展模式可以使用独立的读、写模式
      FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
      //配置读写时序
      FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &p;
      //配置写时序
      FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &p;

      //初始化 FSMC
      FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);
      //使能 FSMC BANK1_SRAM 模式
      FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);
    }
```

4.31.11 lcd_ILI9325.h 文件里的内容是

函数 lcd_ILI9325.h 在这里是针对于 ILI9325 芯片自定义的一份专用于这款芯片的头文件。在 lcd_ILI9325.h 中我们要定义几个地址，具体说明在程序注释中。lcd_ILI9325.h 的内容如下：

```
    #ifndef _LCD_ILI9325_H
    #define _LCD_ILI9325_H
    #include "pbdata.h"


    //6000 0000
    //A16  16 位通讯

    #define Bank1_LCD_D (u32)0x60020000 //命令位是“1”，所以在 16 位通讯命令
格式时地址是 2000
    #define Bank1_LCD_C (u32)0x60000000 //数据位是“0”，所以在 16 位通讯数据
格式时地址是 0
```

```
#define RGB565(r, g, b) ((r >> 3) << 11 | (g >> 2) << 5 | (b >> 3))

#define RED   RGB565(255,0,0)      //红色
#define GREEN RGB565(0,255,0)      //绿色
#define BLUE  RGB565(0,0,255)      //蓝色
#define BLACK RGB565(0,0,0)        //黑色
#define WHITE RGB565(255,255,255)  //白色

void ILI9325_Init(void);
void LCD_WR_REG(u16 index);
void LCD_WR_Data(u16 val);
void LCD_WR_CMD(u16 index,u16 val);

void ILI_9325_Draw_Point(u8 x,u16 y,u16 color);
void ILI_9325_CLEAR(u16 color);

#endif
```

4.31.12 lcd_ILI9325.c 文件里的内容是

函数 lcd_ILI9325.c 在这里是针对于 ILI9325 芯片自定义的一份专用于这款芯片的库函数。他是读写 ILI9325 芯片的核心程序，基本的时序都体现在这个程序中。每一款新的触摸屏生产出来后，生产厂家都会为这款新产品配好了初始化子函数，我们在做开发程序的时候，直接使用就可以。在 lcd_ILI9325.c 中就包含了厂家针对于这款产品开发的初始化子函数。

lcd_ILI9325.c 的内容如下：

```
#include "pbdata.h"

void LCD_WR_REG(u16 index)  //定义子函数，写命令子函数
{
   *(__IO u16 *)(Bank1_LCD_C)=index;//把我们定义的数据读写首地址转化为指针，指向 LCD 真实的扩展地址
}

void LCD_WR_Data(u16 val)  //定义子函数，写数据子函数
```

```
    {
      *(__IO u16 *)(Bank1_LCD_D)=val;//把我们定义的命令读写首地址转化为指针，
指向 LCD 真实的扩展地址
    }

    void LCD_WR_CMD(u16 index,u16 val)  //定义范围子函数，命令和数据一起传送
子函数
    {
      *(__IO u16 *)(Bank1_LCD_C)=index;
      *(__IO u16 *)(Bank1_LCD_D)=val;
    }

    //厂家配置好的初始化函数（生产厂家为这款触摸屏定制的函数）
    void ILI9325_Init(void)
    {
       GPIO_ResetBits(GPIOE, GPIO_Pin_1);      //硬件复位
       delay(0xAFFf);
       GPIO_SetBits(GPIOE, GPIO_Pin_1 );
      delay(0xAFFf);

      LCD_WR_CMD(0x0001, 0x0100); // set SS and SM bit
      LCD_WR_CMD(0x0002, 0x0700); // set 1 line inversion
      LCD_WR_CMD(0x0003, 0x1030); // set GRAM write direction and BGR=1.
      LCD_WR_CMD(0x0004, 0x0000); // Resize register
      LCD_WR_CMD(0x0008, 0x0207); // set the back porch and front porch
      LCD_WR_CMD(0x0009, 0x0000); // set non-display area refresh cycle
ISC[3:0]
      LCD_WR_CMD(0x000A, 0x0000); // FMARK function
      LCD_WR_CMD(0x000C, 0x0000); // RGB interface setting
      LCD_WR_CMD(0x000D, 0x0000); // Frame marker Position
      LCD_WR_CMD(0x000F, 0x0000); // RGB interface polarity

      LCD_WR_CMD(0x0010, 0x0000); // SAP, BT[3:0], AP, DSTB, SLP, STB
      LCD_WR_CMD(0x0011, 0x0007); // DC1[2:0], DC0[2:0], VC[2:0]
      LCD_WR_CMD(0x0012, 0x0000); // VREG1OUT voltage
      LCD_WR_CMD(0x0013, 0x0000); // VDV[4:0] for VCOM amplitude
      LCD_WR_CMD(0x0007, 0x0001);
      delay(12000); // Dis-charge capacitor power voltage
      LCD_WR_CMD(0x0010, 0x1490); // SAP, BT[3:0], AP, DSTB, SLP, STB
      LCD_WR_CMD(0x0011, 0x0227); // DC1[2:0], DC0[2:0], VC[2:0]
      delay(15500); // Delay 50ms
      LCD_WR_CMD(0x0012, 0x001C); // Internal reference voltage= Vci;
      delay(15000); // Delay 50ms
      LCD_WR_CMD(0x0013, 0x1A00); // Set VDV[4:0] for VCOM amplitude
```

```
    LCD_WR_CMD(0x0029, 0x0025); // Set VCM[5:0] for VCOMH
    LCD_WR_CMD(0x002B, 0x000C); // Set Frame Rate
    delay(15000); // Delay 50ms
    LCD_WR_CMD(0x0020, 0x0000); // GRAM horizontal Address
    LCD_WR_CMD(0x0021, 0x0000); // GRAM Vertical Address
    // ----------- Adjust the Gamma Curve ----------//
    LCD_WR_CMD(0x0030, 0x0000);
    LCD_WR_CMD(0x0031, 0x0506);
    LCD_WR_CMD(0x0032, 0x0104);
    LCD_WR_CMD(0x0035, 0x0207);
    LCD_WR_CMD(0x0036, 0x000F);
    LCD_WR_CMD(0x0037, 0x0306);
    LCD_WR_CMD(0x0038, 0x0102);
    LCD_WR_CMD(0x0039, 0x0707);
    LCD_WR_CMD(0x003C, 0x0702);
    LCD_WR_CMD(0x003D, 0x1604);
    //------------------ Set GRAM area ---------------//
    LCD_WR_CMD(0x0050, 0x0000); // Horizontal GRAM Start Address
    LCD_WR_CMD(0x0051, 0x00EF); // Horizontal GRAM End Address
    LCD_WR_CMD(0x0052, 0x0000); // Vertical GRAM Start Address
    LCD_WR_CMD(0x0053, 0x013F); // Vertical GRAM Start Address
    LCD_WR_CMD(0x0060, 0xA700); // Gate Scan Line
    LCD_WR_CMD(0x0061, 0x0001); // NDL,VLE, REV

    LCD_WR_CMD(0x006A, 0x0000); // set scrolling line
    //-------------- Partial Display Control ---------//
    LCD_WR_CMD(0x0080, 0x0000);
    LCD_WR_CMD(0x0081, 0x0000);
    LCD_WR_CMD(0x0082, 0x0000);
    LCD_WR_CMD(0x0083, 0x0000);
    LCD_WR_CMD(0x0084, 0x0000);
    LCD_WR_CMD(0x0085, 0x0000);
    //-------------- Panel Control -------------------//
    LCD_WR_CMD(0x0090, 0x0010);
    LCD_WR_CMD(0x0092, 0x0600);
    LCD_WR_CMD(0x0007, 0x0133); // 262K color and display ON
}

//240*320  向触摸屏送数据，一个点一个点都要扫描到

void ILI_9325_Draw_Point(u8 x,u16 y,u16 color)
{
    LCD_WR_CMD(0x50,x);  //x 起始
    LCD_WR_CMD(0x51,x);  //x 结束
```

```
    LCD_WR_CMD(0x52,y);  //y 起始
    LCD_WR_CMD(0x53,y);  //y 结束

    LCD_WR_CMD(0x20,x);
    LCD_WR_CMD(0x21,y);
    LCD_WR_REG(0x22);
    LCD_WR_Data(color);
}

void ILI_9325_CLEAR(u16 color)//清屏函数
{
    u16 i=0,j=0;

    for(i=0;i<240;i++)
    {
        for(j=0;j<320;j++)
        {
            ILI_9325_Draw_Point(i,j,color);
        }
    }
}
```

4.31.13 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)
{
   RCC_Configuration();  //系统时钟初始化
   GPIO_Configuration();//端口初始化
```

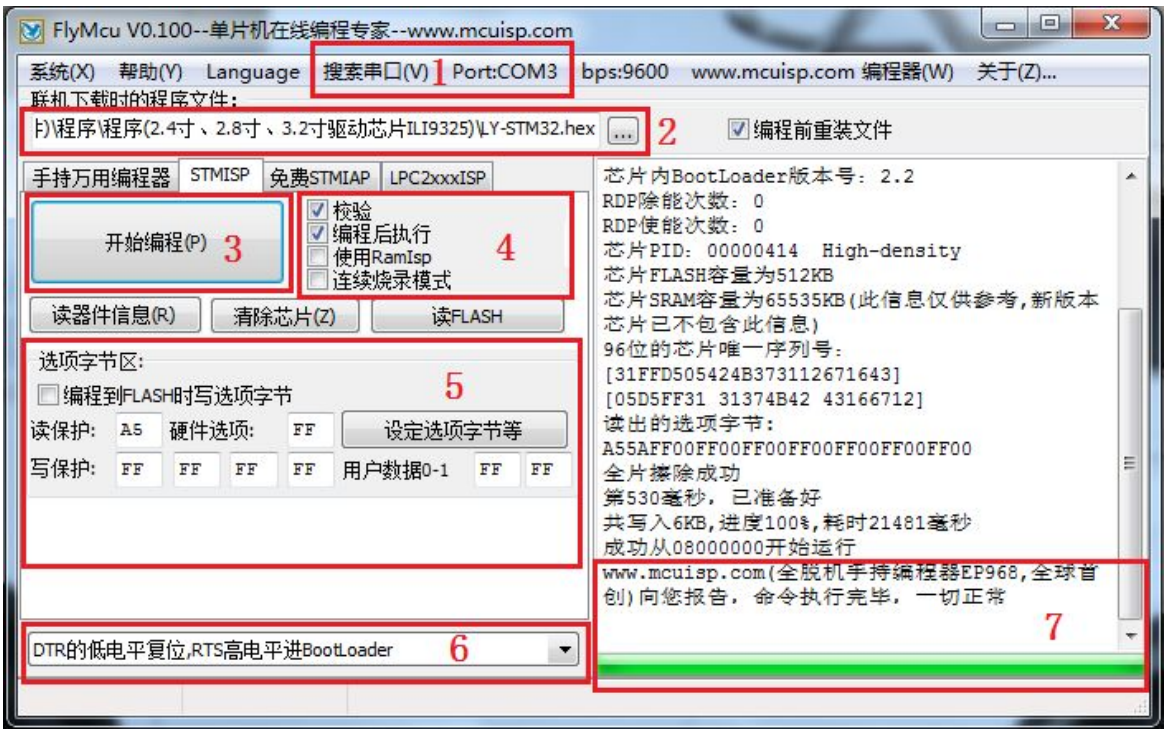
```
    USART_Configuration();
    NVIC_Configuration();
    FSMC_Configuration();
    ILI9325_Init();

    while(1)
    {
        delay_ms(1000);
        ILI_9325_CLEAR(RED);
        delay_ms(1000);
        ILI_9325_CLEAR(GREEN);
        delay_ms(1000);
        ILI_9325_CLEAR(BLUE);
        delay_ms(1000);
        ILI_9325_CLEAR(BLACK);
        delay_ms(1000);
        ILI_9325_CLEAR(WHITE);
    }
}

void RCC_Configuration(void)
{
     SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,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);
```

```
    //FSMC  管脚初始化

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD,&GPIO_InitStructure);
    GPIO_SetBits(GPIOD,GPIO_Pin_13);//打开背光

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_1;//复位
    GPIO_Init(GPIOE,&GPIO_InitStructure);

    //启用 fsmc 复用功能 复用上拉模式

    GPIO_InitStructure.GPIO_Pin =GPIO_Pin_14 //D0
                                |GPIO_Pin_15 //D1
                                |GPIO_Pin_0    //D2
                                |GPIO_Pin_1    //D3
                                |GPIO_Pin_8    //D13
                                |GPIO_Pin_9    //D14
                                |GPIO_Pin_10;//D15
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_Init(GPIOD,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin =GPIO_Pin_7    //D4
                                |GPIO_Pin_8    //D5
                                |GPIO_Pin_9    //D6
                                |GPIO_Pin_10 //D7
                                |GPIO_Pin_11 //D8
                                |GPIO_Pin_12 //D9
                                |GPIO_Pin_13 //D10
                                |GPIO_Pin_14 //D11
                                |GPIO_Pin_15;//D12
    GPIO_Init(GPIOE,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin =GPIO_Pin_11 //RS
                                |GPIO_Pin_4    //nOE
                                |GPIO_Pin_5; //nWE
    GPIO_Init(GPIOD,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin =GPIO_Pin_7; //NE1
    GPIO_Init(GPIOD,&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);
}
```

4.31.14 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\17. 点亮 LCD 彩色液晶屏\程序）

4.31.15 实验效果图

4.31.15.1 读写扩展闪存实验效果

在这个实验中我们要把彩色液晶屏和实验板连接好，送电后下载程序。程序下载完成后彩色液晶屏就会 1 秒钟刷新一次，在不停的改变颜色。

从图 4.31.15.1 试验效果能看出和我们设计的一样，说明我们的程序设计正确，驱动彩色液晶屏正常。

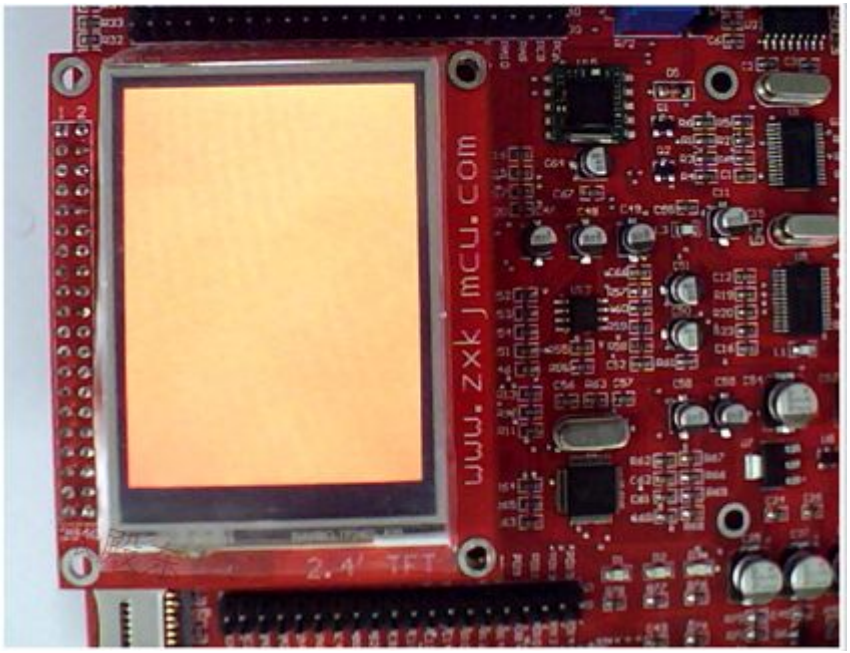


图 4.31.15.1 彩色液晶屏刷新实验效果图（一）

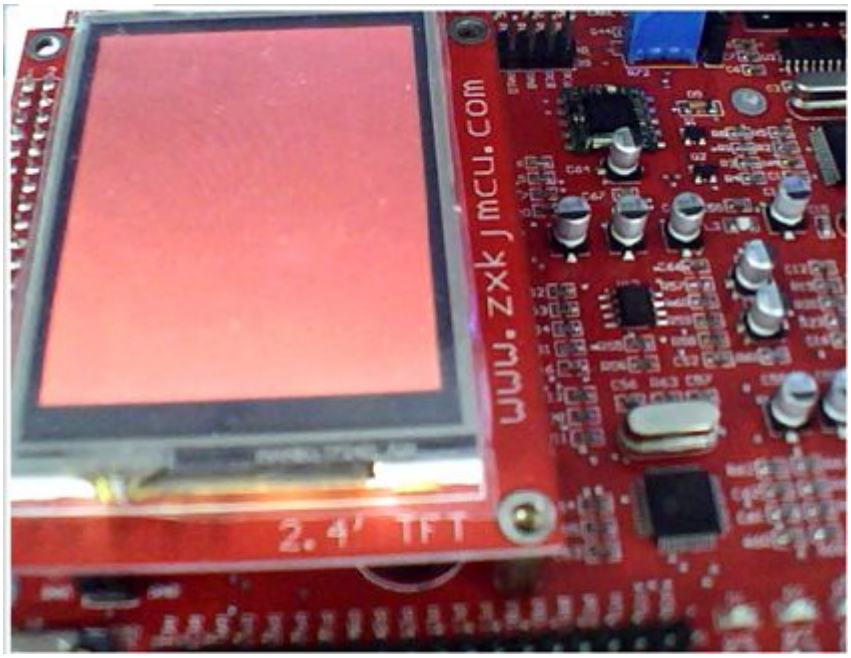


图 4.31.15.2 2.4 寸彩色液晶屏刷新实验效果图（二）

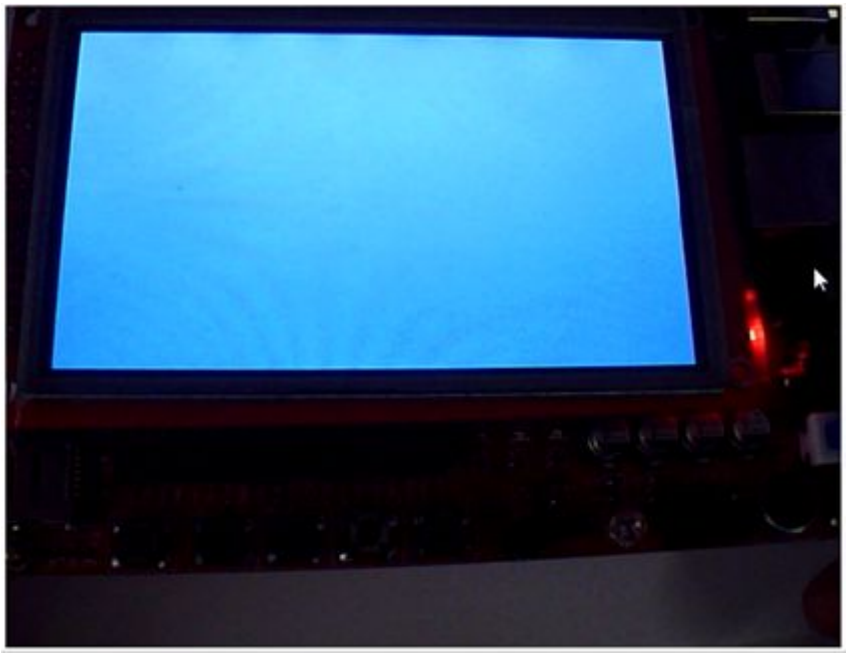


图 4.31.15.3 4.3 寸彩色液晶屏刷新实验效果图