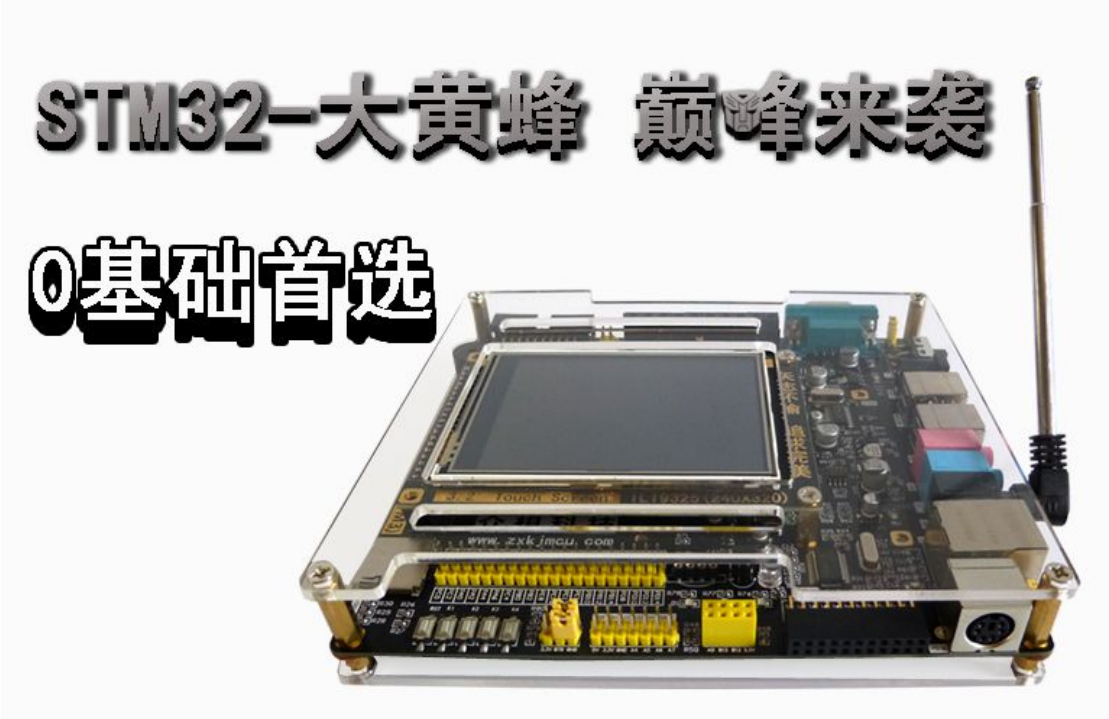
学 ARM 从 STM32 开始

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

4.33 STM32 触摸屏工作原理及程序设计

4.33.1 概述

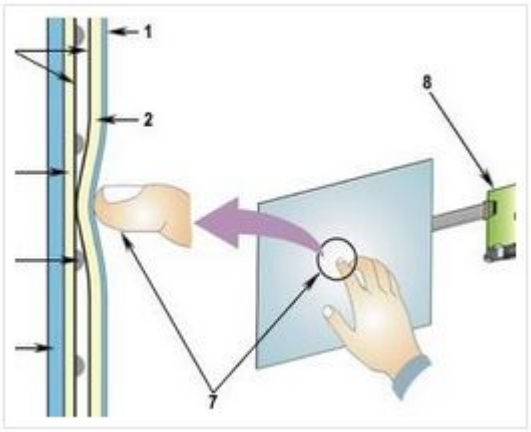
4.33.1.1 触摸屏原理

触摸屏技术主要就是快速准确地处理随机触摸点坐标的技术，目前成熟的触摸屏技术有 4 种，即四线电阻式、电容式、红线式、表面声波式，在这里我们着重分析电容式和四线电阻式原理，简单描述如下：

1、电容技术触摸屏其结构最为简单。它是在紧贴显示屏前的双夹层玻璃中涂有一层透明的氧化、金属导体层，四角引出四个电极受控于控制器。通过引线，夹层导体中有高频电流流动，由于人体电场的存在，触摸点手指与屏幕内涂层构成一个微小的耦合电容，而高频电流对于通过小电容是很容易的这样，对称四电极上的高频电流通过触点小电容被分流。这个被破坏了对称的变化量由控制器侦测到。由于流入四电极的电流与手指触点到四角的距离成反比，故可计算出触点坐标值。电容技术触摸屏灵敏度极高，能感知轻微快速的触碰(响应时间最快为 3ms)，所以它不怕污染和带手套触摸等，但它怕外界强电场干扰。

2、电阻触摸屏的工作原理主要是通过压力感应原理来实现对屏幕内容的操作和控制的，这种触摸屏屏体部分是一块与显示器表面非常配合的多层复合薄膜，其中第一层为玻璃或有机玻璃底层，第二层为隔层，第三层为多元树脂表层，表面还涂有一层透明的导电层，上面再盖有一层外表面经硬化处理、光滑防刮的塑料层。在多元脂表层表面的传导层及玻璃层感应器是被

许多微小的隔层所分隔电流通过表层，轻触表层压下时，接触到底层，控制器同时从四个角读出相称的电流及计算手指位置的距离。这种触摸屏利用两层高透明的导电层组成触摸屏，两层之间距离仅为 2.5 微米。当手指触摸屏幕时，平常相互绝缘的两层导电层就在触摸点位置有了一个接触，因其中一面导电层接通 Y 轴方向的 5V 均匀电压场，使得侦测层的电压由零变为非零，控制器侦测到这个接通后，进行 A/D 转换，并将得到的电压值与 5V 相比，即可得触摸点的 Y 轴坐标，同理得出 X 轴的坐标，这就是电阻技术触摸屏共同的最基本原理。



触摸屏包含上下叠合的两个透明层，四线和八线触摸屏由两层具有相同表面电阻的透明阻性材料组成，五线和七线触摸屏由一个阻性层和一个导电层组成，通常还要用一种弹性材料来将两层隔开。当触摸屏表面受到的压力(如通过笔尖或手指进行按压)足够大时，顶层与底层之间会产生接触。所有的电阻式触摸屏都采用分压器原理来产生代表 X 坐标和 Y 坐标的电压。分压器是通过将两个电阻进行串联来实现的。上面的电阻(R1)连接正参考电压(VREF)，下面的电阻(R2)接地。两个电阻连接点处的电压测量值与下面那个电阻的阻值成正比。

为了在电阻式触摸屏上的特定方向测量一个坐标，需要对一个阻性层进行偏置：将它的一边接 VREF，另一边接地。同时，将未偏置的那一层连接到一个 ADC 的高阻抗输入端。当触摸屏上的压力足够大，使两层之间发生接

触时，电阻性表面被分隔为两个电阻。它们的阻值与触摸点到偏置边缘的距离成正比。触摸点与接地边之间的电阻相当于分压器中下面的那个电阻。因此，在未偏置层上测得的电压与触摸点到接地边之间的距离成正比。

触摸屏赋予了多媒体系统崭新的面貌，极富吸引力。多媒体技术层出不穷，为现代人提供更好的交流需求。

4.33.1.2　STM32 实验板用触摸屏规格

由于触摸屏操作方便，人机交互很好，在大黄蜂实验板上我们标配了一款电阻触摸屏，具体参考“图 4.33.2　外扩的触摸屏实物”，主要有三种规格：2.4 寸、3.2 寸、4.3 寸。

2.4 寸触摸屏分辨率是：240×320

3.2 寸触摸屏分辨率是：240×320

4.3 寸触摸屏分辨率是：272×480

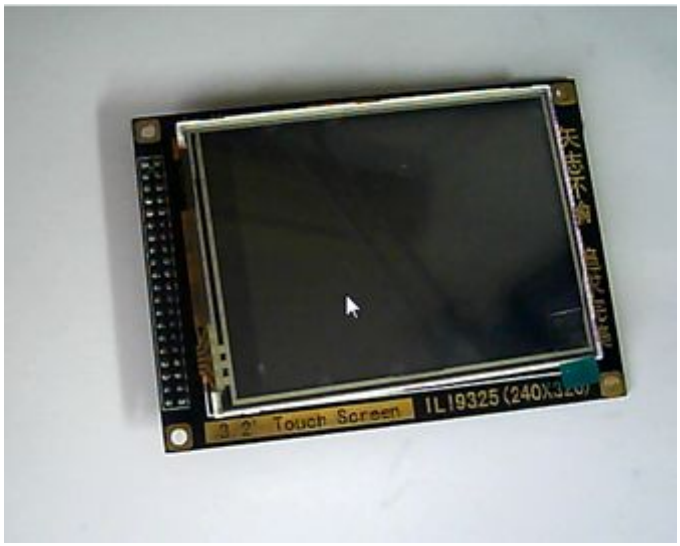


图 4.33.1　3.2 寸触摸屏

在大黄蜂试验板左侧有 40P 插座，如“图 4.33.2　3.2 寸触摸屏触控芯片位置”红色矩形区域所示，我们的触摸屏模块直接插到这个插座上就可以了。

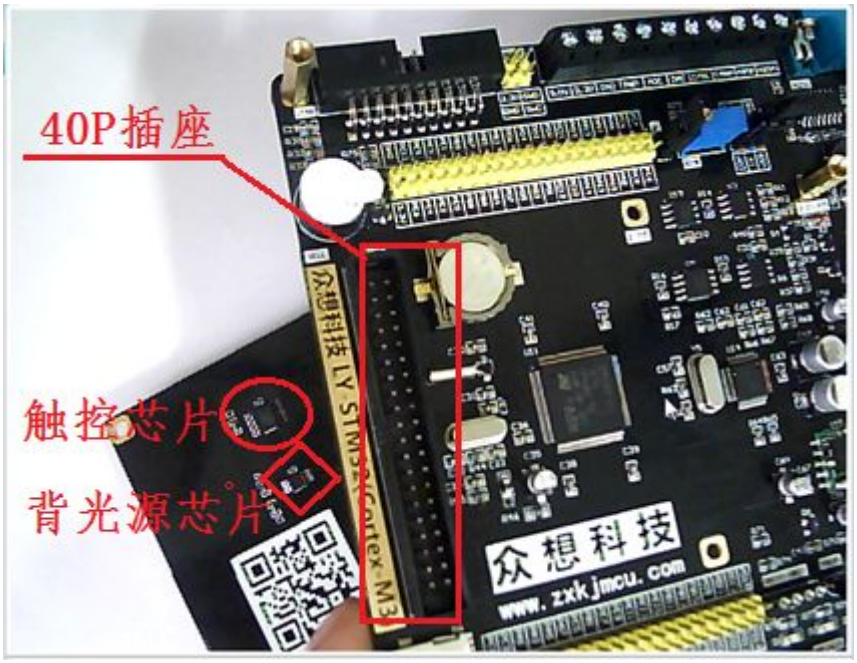


图 4.33.2 3.2 寸触摸屏触控芯片位置

4.33.2 XPT2046 触摸屏控制芯片说明

XPT2046 是一款 4 线制电阻式触摸屏控制器，内含 12 位分辨率 125KHz 转换速率逐步逼近型 A/D 转换器。XPT2046 支持从 1.5V 到 5.25V 的低电压 I/O 接口。XPT2046 能通过执行两次 A/D 转换查出被按的屏幕位置，除此之外，还可以测量加在触摸屏上的压力。内部自带 2.5V 参考电压，可以作为辅助输入、温度测量和电池监测之用，电池监测的电压范围可以从 0V 到 6V。XPT2046 片内集成有一个温度传感器。 在 2.7V 的典型工作状态 在 2.7V 的典型工作状态下，关闭参考电压，功耗可小于 0.75mW。XPT2046 采用微小的封装形式：TSSOP-16,QFN-16 和 VFBGA－48。工作温范围为-40℃～+85℃。与 ADS7846、TSC2046、AK4182A 完全兼容。

4.33.2.1 XPT2046 主要特性

- 工作电压范围为 1.5V～5.25V
- 支持 1.5V～5.25V 的数字 I/O

- 内建 2.5V 参考电压源
- 电源电压测量（0V~6V）
- 内建结温测量功能
- 触摸压力测量
- 采用 3 线制 SPI 通信接口
- 具有自动省电功能

4.33.2.2 XPT2046 引脚功能说明

附表 1 引脚功能描述

QFN 引脚号	TSSOP 引脚号	VFBGA 引脚号	名称	说明
1	13	A5	BUSY	忙时信号线。当 CS 为高电平时为高阻态
2	14	A4	DIN	串行数据输入端。当 CS 为低电平时，数据在 DCLK 上升沿锁存进来
3	15	A3	CS	片选信号。控制转换时序和使能串行输入输出寄存器，高电平时 ADC 掉电。
4	16	A2	DCLK	外部时钟信号输入
5	1	B1 和 C1	VCC	电源输入端
6	2	D1	XP	XP 位置输入端
7	3	E1	YP	YP 位置输入端
8	4	G2	XN	XN 位置输入端
9	5	G3	YN	YN 位置输入端
10	6	G4 和 G5	GND	地
11	7	G6	VBAT	电源监视输入端
12	8	E7	AUX	ADC 辅助输入通道
13	9	D7	VREF	参考电压输入/输出
14	10	C7	IOVDD	数字电源输入端
15	11	B7	PENIRQ	笔接触中断引脚
16	12	A6	DOUT	串行数据输出端。数据在 DCLK 的下降沿移出，当 CS 为高电平时为高阻态。

4.33.2.3 XPT2046 数字接口

XPT2046 数据接口是串行接口，其典型工作时序如图 4.33.3 所示，图中展示的信号来自带有基本串行接口的单片机或数据信号处理器。处理器和

转换器之间的通信需要 8 个时钟周期，可采用 SPI 同步串行接口。一次完整的转换需要 24 个串行同步时钟（DCLK）来完成。

前 8 个时钟用来通过 DIN 引脚输入控制字节。当转换器获取有关下一次转换的足够信息后，将进入采样模式。3 个多时钟周期后，控制字节设置完成，转换器进入转换状态。接着的 12 个时钟周期将完成真正的模数转换，第 13 个时钟将输出转换结果的最后一位。剩下的 3 个多时钟周期将用来完成被转换器忽略的最后字节（DOUT 置低）。

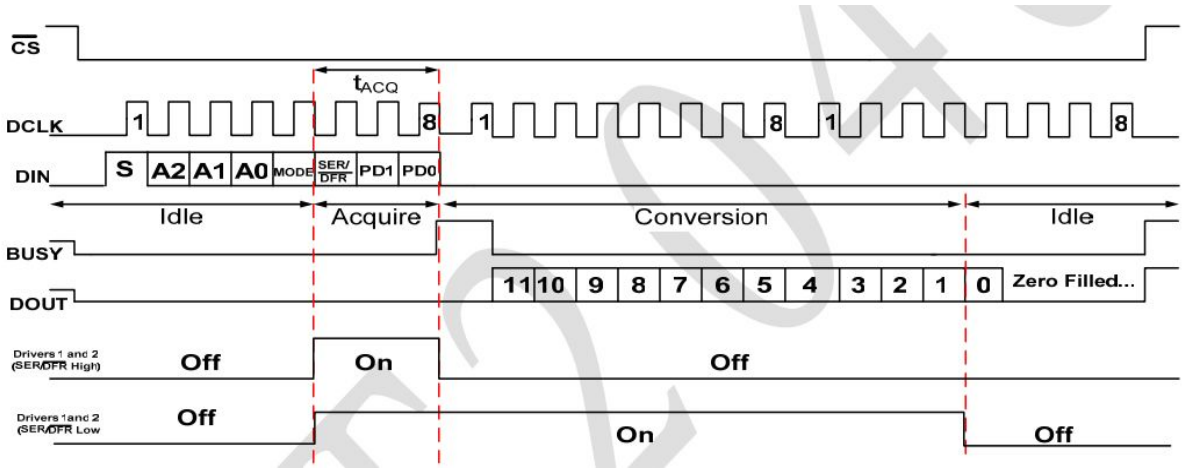


图 4.33.3　时序

控制字节由 DIN 输入的控制字如附表 2 所示，它用来启动转换，寻址，设置 ADC 分辨率，配置和对 XPT2046 进行掉电控制。附表 3 给出控制字的各控制位的详细说明。

附表 2 制字的控制位命令

位 7（MSB）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（LSB）
S	A2	A1	A0	MODE	SER/DFR	PD1	PD0

附表 3 控制字节各位描述

位	名称	功能描述
7（MSB）	S	开始位。为 1 表示一个新的控制字到来，为 0 则忽略 PIN 引脚数据

6-4	A2-A0	通道选择位。参见表 1 和表 2
3	MODE	12 位/8 位转换分辨率选择位。为 1 选择 8 位为转换分辨率，为 0 选择 12 位分辨率
2	SER/DFR	单端输入方式/查分输入方式选择位。为 1 是单端输入方式，为 0 是差分输入方式
1-0	PD1-PD0	低功率模式选择位。若为 11，器件总处于供电状态；若为 00，器件在变换之间处于低功率模式

起始位——第一位，即 S 位。控制字的首位必须是 1，即 S＝1。在 XPT2046 的 DIN 引脚检测到起始位前，所有的输入将被忽略。

地址——接下来的 3 位（A2、A1 和 A0）选择多路选择器的现行通道，触摸屏驱动和参考源输入。

附表 4 差分模式输入配置（SER/DER=0）

A2	A1	A0	+REF	-REF	YN	XP	YP	Y-位置	X-位置	Z1-位置	Z2-位置	驱动
0	0	1	YP	XN		+IN		测量				YP, XN
0	1	1	YP	XN		+IN				测量		YP, XN
1	0	0	YP	XN	+IN						测量	YP, XN
1	0	1	YP	XN			+IN		测量			YP, XN

MODE——模式选择位，用于设置 ADC 的分辨率。MODE＝0，下一次的转换将是 12 位模式；MODE＝1，下一次的转换将是 8 位模式。

SER/DFR——位控制参考源模式，选择单端模式（SER/DFR＝1），或者差分模式（SER/DFR＝0）。在 X 坐标、Y 坐标和触摸压力测量中，为达到最佳性能，首选差分工作模式。参考电压来自开关驱动器的电压。在单端模式下，转换器的参考电压固定为 VREF 相对于 GND 引脚的电压。

PD0 和 PD1——ADC 的内部参考电压可以单独关闭或者打开，但是，在转换前，需要额外的时间让内部参考电压稳定到最终稳定值；如果内部参考源处于掉电状态，还要确保有足够的唤醒时间。ADC 要求是即时使用，无唤醒时间的。另外还得注意，当 BUSY 是高电平的时候，内部参考源禁止进入掉电模式。XPT2046 的通道改变后，如果要关闭参考源，则要重新对 XPT2046 写

入命令。

附表 5 掉电和内部参考电压选择

PD1	PD0	PENIRQ	功能说明
0	0	使能	在两次 A/D 转换之间掉电，下次转换一开始，芯片立即进入完全上电状态，而无需额外的延时。在这种模式下，YN 开关一直处于 ON 状态。
0	1	禁止	参考电压关闭，ADC 打开
1	0	使能	参考电压打开，ADC 关闭
1	1	禁止	芯片处于上电状态，参考电压和 ADC 总是打开

4.33.2.4 XPT2046 工作时序

详细工作时序请参考《XPT2046 用户手册》。

4.33.3 实验目的

通过这个实验我们要熟悉 3.2 寸触摸屏的基本程序设计结构，通过程序设计要熟练掌握触摸屏的使用方法和原理。再有就是要熟练掌握触摸屏时序图，因为所有程序的编写都要严格按照硬件工作的时序来设计，只有这样硬件电路才能正常工作。

4.33.4 大黄蜂实验板触摸屏硬件设计

3.2 寸触摸屏属于实验板外扩设备，属于大黄蜂开发板标准配备的显示工具，在使用的时候直接把触摸屏插接到大黄蜂开发板标准的 40 位插座上即可。触摸屏采用标准的 SPI 通讯方式。下面简单介绍 LCD 接口功能。

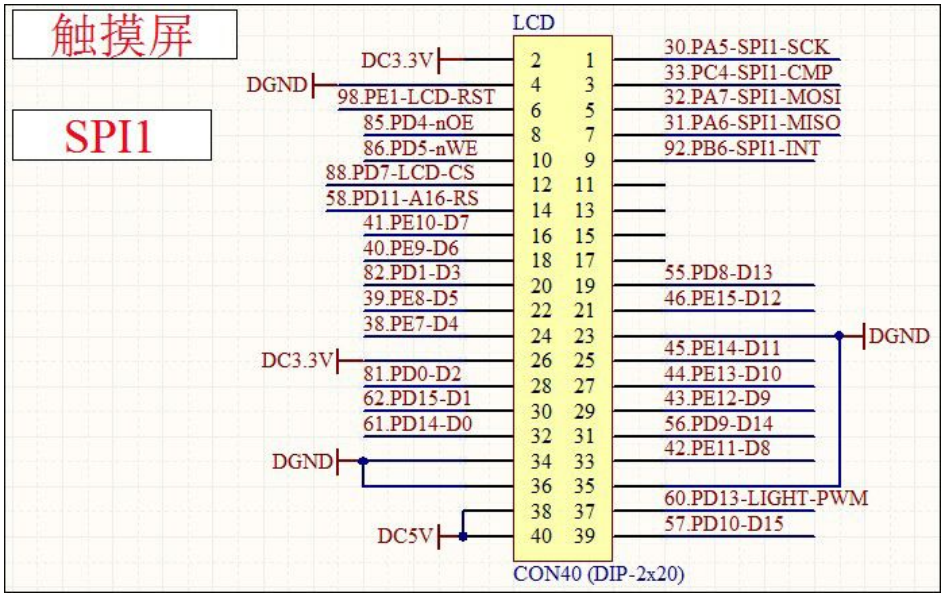


图 4.33.4 触摸屏扩展槽原理图

PD7-LCD-CS：LCD 片选信号；

PE1-LCD-RST：LCD 复位信号；

PD5-nWE：写使能，连接 LCD 的 RW 脚；

PD4-nOE：输出使能连接 LCD 的 RD 脚；

PD11-A16-RS：命令/数据标志（0，读写命令；1，读写数据）；

D[15:0]：16 位双向数据线；

PD13-LIGHT-PWM：LCD 背光控制；

4.33.5 软件设计

4.33.5.1 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，要使触摸屏正常工作，在程序设计中要关闭以太网和 FLASH，因为他们公用 SPI 接口，必须以多选一的方式，所以要保证只有一个占用 SPI 端口，在这个程序中我们不使用串口

通讯。

3、在这个实验程序中，我们为了程序简单化，就采用查询的方式。

4.33.5.2 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
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。

4.33.5.3 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。

4.33.5.4 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 "stm32_spi.h" //自定义的 stm32_spi.h 专属函数
#include "stm32_touch.h"  //自定义的 stm32_touch.h 专属函数
#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

//定义变量
extern u8 dt;
```

```
//定义函数
void RCC_HSE_Configuration(void);
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。“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。

“stm32_touch.h”函数中存放我们触摸操作时的程序处理时序，它是我们为触摸操作新建的专属函数。

4.33.5.5 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;


    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
    }
```

4.33.6 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，这点很重要。

4.33.7 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_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_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,ENABLE);
    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、PB 端口、PC 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。

4.33.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，。

```
while(1)
   {
      if(PEN==0)
      {
          count=0;
          do
          {
            buffer[0][count]=TPReadX();
            buffer[1][count]=TPReadY();
            count++;
          }while((PEN==0)&&count<10);

          if(count==10)
          {
              tp_x=0;
              tp_y=0;
              for(count=0;count<10;count++)
              {
                  tp_x=tp_x+buffer[0][count];
                  tp_y=tp_y+buffer[1][count];
              }
```

```
            tp_x=tp_x/10;
            tp_y=tp_y/10;

           //tp_x=240-tp_x/17; //2.4 寸
            tp_x=tp_x/17; //3.2 寸
            tp_y=320-tp_y/13;

            ILI_9325_Draw_Point(tp_x,tp_y,RED);
        }
    }
  }
```

4.33.9 stm32_spi.h 文件里的内容

```
#ifndef _STM32_SPI_H
#define _STM32_SPI_H

void SPI1_Configuration(void);
u8 SPI1_SendByte(u8 byte);

#endif
```

4.33.10 stm32_spi.c 文件里的内容

```
#include "pbdata.h"

void SPI1_Configuration(void)
{
   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 禁止以太网片选
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_7;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
   GPIO_Init(GPIOB,&GPIO_InitStructure);
```

```
    //禁止触摸屏片选
    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_InitStructure.SPI_Mode=SPI_Mode_Master;
    SPI_InitStructure.SPI_DataSize=SPI_DataSize_8b;

    SPI_InitStructure.SPI_CPOL=SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA=SPI_CPHA_1Edge;
    SPI_InitStructure.SPI_NSS=SPI_NSS_Soft;
    SPI_InitStructure.SPI_BaudRatePrescaler=SPI_BaudRatePrescaler_64;


    SPI_InitStructure.SPI_FirstBit=SPI_FirstBit_MSB;
    SPI_InitStructure.SPI_CRCPolynomial = 7;

    SPI_Init(SPI1,&SPI_InitStructure);

    SPI_Cmd(SPI1,ENABLE); //使能 SPI

    //禁止以太网片选
    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 SPI1_SendByte(u8 byte)
{
    while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_TXE)==RESET);
    SPI_I2S_SendData(SPI1,byte);
```

```
    while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_RXNE)==RESET);
    return SPI_I2S_ReceiveData(SPI1);
}
```

4.33.11 stm32_touch.h 文件里的内容

函数 stm32_touch.h 在这里是针对于触控芯片自定义的一份专用于这款芯片的头文件。在 stm32_touch.h 中我们要定义几个地址，具体说明在程序注释中。stm32_touch.h 的内容如下：

```
#ifndef _STM32_TOUCH_H
#define _STM32_TOUCH_H
#include "pbdata.h"

#define TP_CS()   GPIO_ResetBits(GPIOC,GPIO_Pin_4)//片选选中中间变量，方便移植
#define TP_DCS()  GPIO_SetBits(GPIOC,GPIO_Pin_4)  //片选释放中间变量，方便移植，高电平释放。
#define PEN GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_6)//中断信号

void Touch_GPIO(void);//声明触摸到来的函数
u16 TPReadX(void);//声明触摸屏 X 轴函数
u16 TPReadY(void);//声明触摸屏 Y 轴函数
#endif
```

4.33.12 stm32_touch.c 文件里的内容

```
#include "pbdata.h"
//查询方式
void Touch_GPIO(void)//触摸发生函数
{
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_Init(GPIOB,&GPIO_InitStructure);
}
```

```
u16 TPReadX(void)//读取 X 轴函数
{
   u16 x=0;//中间变量

   TP_CS();//片选拉低
   delay_us(5);//延时
   SPI1_SendByte(0xD0);//12 位分辨率，查分方式，发送命令字
   delay_us(5);
   x=SPI1_SendByte(0x00);//读回来 X 轴数据，要读回来两个 8 位数据
   x<<=8;//左移 8 位
   x+=SPI1_SendByte(0x00);//和上一回读回来的数据相加到一起。
   delay_us(5);
   TP_DCS();//片选拉高
   x>>=3;//因为我们要 12 位分辨率，所以我们要把这个 16 位数据右移 3 位，正好符合我们 12 位分辨率的精度数据。
   return x;
}

u16 TPReadY(void)//读取 Y 轴函数
{
   u16 y=0;

   TP_CS();
   delay_us(5);
   SPI1_SendByte(0x90);//12 位分辨率，查分方式，发送命令字
   delay_us(5);
   y=SPI1_SendByte(0x00);
   y<<=8;
   y+=SPI1_SendByte(0x00);
   delay_us(5);
   TP_DCS();
   y>>=3;
   return y;
}
```

4.33.13 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 count=0;
   u16 buffer[2][10]={{0},{0}};//定义二维数组
   u32 tp_x=0,tp_y=0;//定义 32 位的中间临时变量

   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();
   NVIC_Configuration();
   FSMC_Configuration();
   SPI1_Configuration();//SPI1 初始化
   Touch_GPIO();//触摸函数初始化
   ILI9325_Init();

   delay_ms(1000);
   ILI_9325_CLEAR(BLUE);//背景色设置

   while(1)
   {
      if(PEN==0)//说明有触摸笔按下了
      {
         count=0;//中间变量付初值
         do
         {
            buffer[0][count]=TPReadX();//数据存放在数组里
            buffer[1][count]=TPReadY();//数据存放在数组里
            count++;
         }while((PEN==0)&&count<10);//

         if(count==10)
         {
            tp_x=0;//中间变量付初值
            tp_y=0;//中间变量付初值
            for(count=0;count<10;count++)
```

```
                {
                    tp_x=tp_x+buffer[0][count];
                    tp_y=tp_y+buffer[1][count];
                }

                tp_x=tp_x/10;//求 X 轴平均值
                tp_y=tp_y/10;//求 Y 轴平均值

               //tp_x=240-tp_x/17; //2.4 寸触摸屏
                tp_x=tp_x/17; //3.2 寸触摸屏
                tp_y=320-tp_y/13;

                ILI_9325_Draw_Point(tp_x,tp_y,RED);
            }
        }
    }
}

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_GPIOD,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);
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,ENABLE);
}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

   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_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);
    }
```

4.33.14 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\20.触摸屏实验\程序）

4.33.15 实验效果图

4.33.15.1 读写扩展闪存实

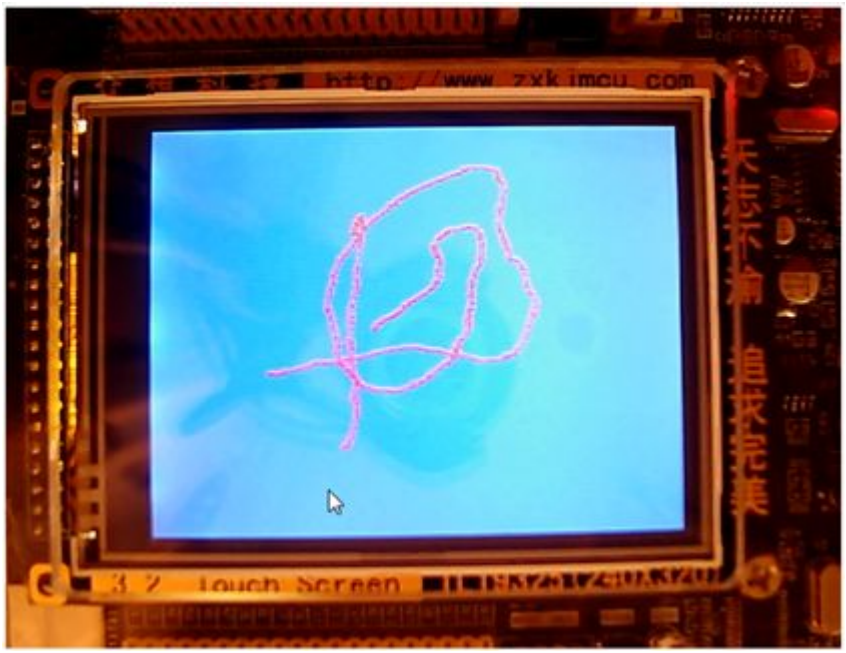


图 4.33.15.1 触摸屏触摸笔操作实验效果图