1 触摸画板实验

章节导读

本章将介绍如何通过 I2C 协议驱动 TFT LCD 屏上的触摸芯片，从而实现触摸画板的功能。

1.1 电容屏工作原理

电容屏也叫电容式触摸屏，是一种利用电容触控技术来工作的四层复合玻璃屏，电容屏的内表面和夹层各涂有一层 ITO 导电层，最外层是只有 0.0015 毫米厚的矽土玻璃保护层。电容屏要实现多点触控，靠的就是增加互电容的电极，简单地说，就是将屏幕分块，在每一个区域里设置一组互电容模块都是独立工作，所以电容屏就可以独立检测到各区域的触控情况，进行处理后，简单地实现多点触控。

电容屏是利用人体的电流感应进行工作的。当触摸电容屏时，由于人体电场，用户手指和工作面形成一个耦合电容，因为工作面上接有高频信号，于是手指吸收走一个很小的电流，这个电流分别从屏的四个角上的电极中流出，且理论上流经四个电极的电流与手指头到四角的距离成比例，控制器通过对四个电流比例的精密计算，得出位置。可以达到 99%的精确度，具备小于 3ms 的响应速度。

1.2 触摸芯片简介

本次实验使用的 TFT LCD 屏上的触摸芯片为 GT911，其在通信过程中需要使用到的引脚如下表 1 所示。

表 1 触摸芯片引脚说明表

模块引脚序号	名称	说明
16	PEN	电容触摸屏中断信号（CT_INT）
18	MOSI	电容触摸屏 IIC_SDA 信号（CT_SDA）
22	CS	电容触摸屏复位信号（CT_RST）
19	CLK	电容触摸屏 IIC_SCL 信号（CT_SCL）

上表中的 PEN 脚，也就是触摸屏对应的 INT 口，其具有上升沿或者下降沿中断触发功能，并且当其为输入状态的时候，主控端必须为悬浮态，取消内部上下拉功能；主机通过输出高、低电平控 RST 口为高或低。GT911 与主机的通

信采用的是标准的 I2C 通信，由 SCL 和 SDA 与主 CPU 进行通讯，在系统中 GT9147 始终作为从设备，所有通讯都是由主 CPU 发起，建议通讯速度为 400Kbps 或以下。

GT911 的 I2C 从设备地址由 7 位设备地址加上 1 位读写控制位，高 7 位为地址，bit 0 为读写控制位。GT911 一共有两个从设备地址可以选择，如下表 2 所示。

表 2 设备地址选择表

7 位地址	8 位写地址	8 位读地址
0x5D	0xBA	0xBB
0X14	0x28	0x29

1.2.1 触摸芯片关键寄存器

GT911 的寄存器有很多，这里简单的介绍一下 GT911 关键的一些寄存器。更多更详细的寄存器介绍可以查看具体的器件手册。

1.2.1.1 产品 ID 寄存器（0x8140~0x8143）

一共由 4 个寄存器组成，用于保存产品 ID，对于 GT911，这四个寄存器读出来就是：9，1，1，Null 四个字符（ASCII 码格式）。因此，我们可以通过这四个寄存器的值，来判断驱动 IC 的型号从而执行不同的初始化。

1.2.1.2 配置寄存器组（0x8047~0x8100）

一共有 184 个寄存器，一般厂家都会提供给我们，我们只需要将对应的值写入对应的寄存器中就可以了，需要写入的值如下所示：

```
{
        'B',0x20,0x3 ,0xe0,0x1 ,0xa ,0xd ,0x0 ,0x1 ,0xa ,
        0x28,0xf ,0x50,0x32,0x3 ,0x5 ,0x0 ,0x0 ,0x0 ,0x0 ,
        0x22,0x22,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x89,0x29,0xb ,
        0x26,0x24,0xea,0x4 ,0x0 ,0x0 ,0x0 ,0x2 ,0x3 ,0x1c,
        0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x3 ,0x64,0x32,0x0 ,0x0 ,
        0x0 ,0x14,0x37,0x94,0xc5,0x2 ,0x7 ,0x0 ,0x0 ,0x4 ,
        0xe8,0x13,0x0 ,0x97,0x1e,0x0 ,0x61,0x30,0x0 ,0x3e,
        0x4c,0x0 ,0x28,0x78,0x0 ,0x28,0x0 ,0x0 ,0x0 ,0x0 ,
        0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,
        0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,
        0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,
        0x0 ,0x0 ,0x16,0x14,0x12,0x10,0xe ,0xc ,0xa ,0x8 ,
        0x6 ,0x4 ,0x2 ,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
        0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
        0xff,0xff,0x22,0x21,0x20,0x1f,0x1e,0x1d,0x1c,0x18,
```

```
        0x16,0x12,0x10,0xf ,0xa ,0x8 ,0x6 ,0x4 ,0x2 ,0x0 ,
        0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
        0xff,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,0x0 ,
        0x0 ,0x0 ,0x0 ,0x0 ,
};
```

1.2.1.3 状态寄存器（0x814E）

对于状态寄存器的每一位的介绍如下表 3 所示：

表 3 状态寄存器介绍表

Bit	说明
7	Buffer status，1 表示坐标（或按键）已经准备好，主控可以读取；0 表示未就绪，数据无效。当主控读取完坐标后，必须通过 I2C 将此标志（或整个字节）写为 0
6	Large detect，大型检测
5~4	保留
3~0	Number of touch points，屏上的坐标点个数

由上表可以看出，如果有数据的话，最高位就会是 1，最低四位代表有效触点的个数，范围是：0~5，0 表示没有触摸，5 表示有 5 点触摸。

1.2.1.4 坐标数据寄存器

坐标寄存器用于存储触点信息，每个触点通过 6 个寄存器存储信息。坐标寄存器的起始地址为 0x8150，以点 1 的坐标数据寄存器组为例，得到寄存器的说明如下表 4 所示。0x8150 ~ 0x8151 用于存储触点 1 的 X 坐标值的低字节和高字节，0x8152 ~ 0x8153 用于存储触点 1Y 坐标值的低字节和高字节，0x8154 ~ 0x8155 用于存储触点 1 的宽和高。

表 4 触点 1 的坐标数据寄存器的说明表

寄存器地址	说明
0x8150	触点 1 x 坐标低八位
0x8151	触点 1 x 坐标的高八位
0x8152	触点 1 y 坐标低八位
0x8153	触点 1 y 坐标高八位
0x8154	触点 1 触摸尺寸低八位
0x8155	触点 1 触摸尺寸高八位

1.3 设计实例

本章实验将在 LCD 屏上绘制出一个画板，画板主要分为三部分：画笔颜色选取按钮、清除按钮和绘制区域。画笔颜色选取和清除功能将通过触摸的 LCD

屏上对应位置进行控制。同时，TFT 显示的图像可以通过 HDMI 显示到 HDMI 显示设备上。

1.4 PL 端逻辑系统搭建

本次实验可以基于 RGB_TFT 工程搭建，将 RGB_TFT 的 TD 工程另存为新工程，随后打开工程进行配置。这里我们需要使能 I2C0 和 I2C1，将 I2C0 的引脚路由设置为 PS IO 50...51（触摸屏的 IIC 配置引脚为 PS MIO50...51），将 I2C1 的引脚路由设置为 PL IO（HDMI 转换芯片 SIi9022 的 IIC 引脚位于 PL 侧）。同时，使能 GPIO PS，用于 TP_INT 信号（TP_INT 信号连接在 PS MIO0）；使能 GPIO PL，设置位宽为 1，用于触摸屏的复位信号，如图 1-1 所示:

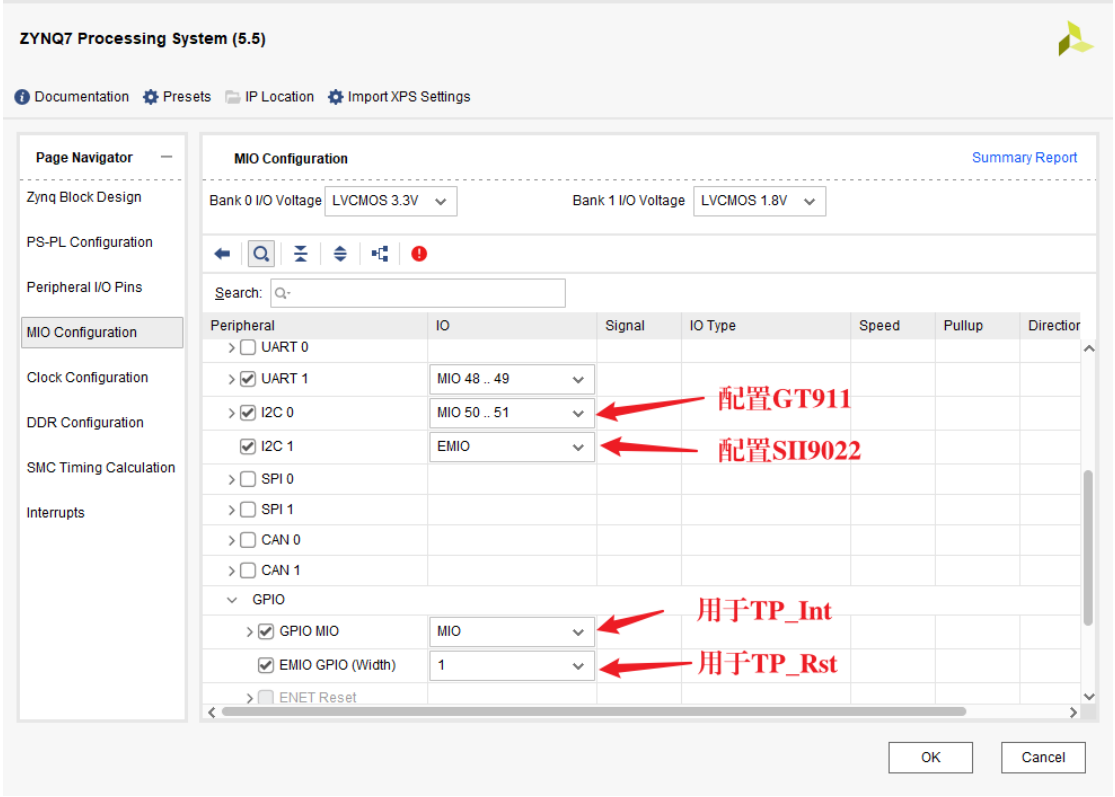

图 1-1 使能外设接口

配置完成后，将相关引脚导出并重命名方便区分，如图 1-2 所示：

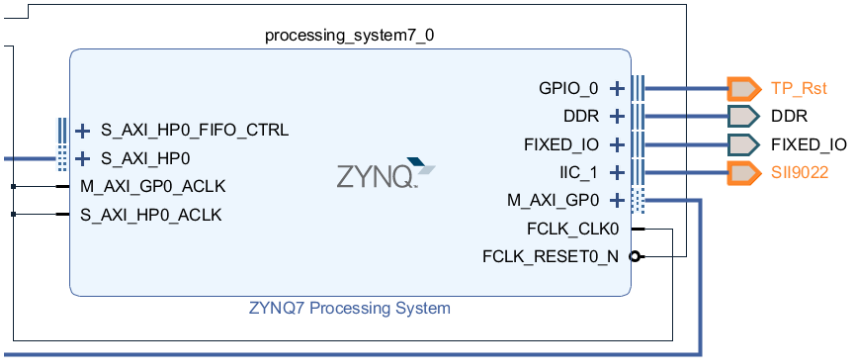

图 1-2 导出引脚

接着验证设计，确认无误后，生成输出，如图 1-3 所示：

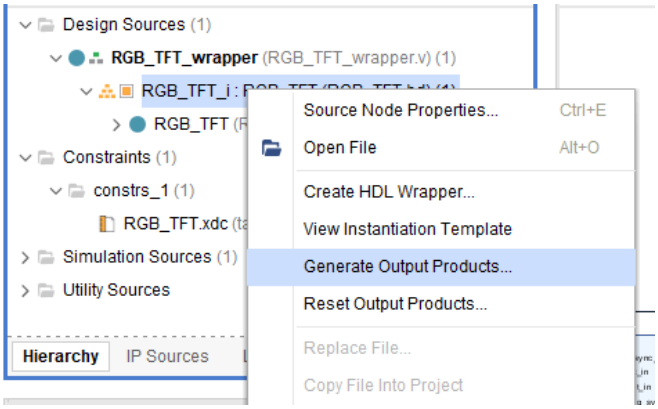

图 1-3 生成输出

生成完成后，分配引脚并生成 bit 流，将 bit 流与硬件规范一起导出。其中新增的管脚分配如表 5：

表 5 新增引脚分配表

信号名	FPGA 引脚
SII9022_scl_io	G22
SII9022_sda_io	H22
TP_Rst_tri_io	R15

分配管脚时注意要设置这三个引脚为上拉，如图 1-4 所示：

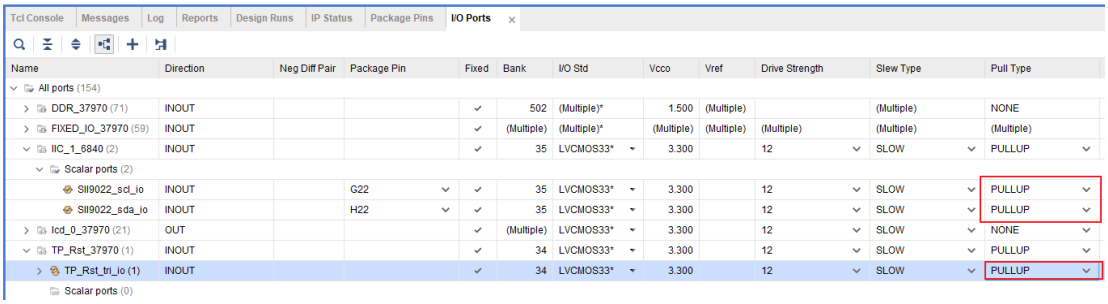

图 1-4 设置上拉

1.5 PS 端应用程序设计

导出硬件规范文件后，运行 vitis，首先还是新建一个硬件工程和一个应用工程。本次设计需要将 PS_GPIO、PS_IIC、SII9022、Touch 库添加到工程中，用户可以从例程中拷贝。其中，SII9022 库用于初始化 HDMI 转换芯片 SII9022，Touch 库用于初始化以及扫描触摸屏。

文件添加完成后，修改 main.c 为如下代码：

```
#include <math.h>
#include "COMMON.h"
#include "Create_Features.h"

//越界判断
#define OutXBd(x)
((x<BdX1(Board))?BdX1(Board):((x>BdX2(Board))?BdX2(Board):x))
#define OutYBd(y)
((y<BdY1(Board))?BdY1(Board):((y>BdY2(Board))?BdY2(Board):y))

void LCD_Draw_BLine(u16 x1,u16 y1,u16 x2,u16 y2,u32 color);
void Init_Homepage();
void Touch_Scan();

uint32_t Palette_Color = LCD_RED;

//主函数
int main(void)
{
  Init_Homepage();  //初始化外设和绘制界面

  while(1) {
    Touch_Scan(); //触摸扫描处理
  }

    return 0;
}

//画粗实线  x1,y1:起点坐标  x2,y2:终点坐标  color:颜色
void LCD_Draw_BLine(u16 x1,u16 y1,u16 x2,u16 y2,u32 color)
{
  u16 i,j;
  u8 LWidth = 3;  //线条宽度

  for(i=0;i<2*LWidth;i++) {
    for(j=0;j<2*LWidth;j++) {
      if((pow(j-LWidth,2) + pow(i-LWidth,2) - pow(LWidth,2)) < 0) {
```

```
            LCD_DrawLine(OutXBd(x1-LWidth+j),OutYBd(y1-LWidth+i),
                OutXBd(x2-LWidth+j),OutYBd(y2-LWidth+i), color);
        }
      }
    }
}

//初始化外设和主界面
void Init_Homepage()
{
  LCD_Init();
  SII9022_Init();
  GT911_Init();

  Draw_Normal_Button(Clear);
  Draw_Button_Effect(Palette_Red);
  Draw_Normal_Button(Palette_Green);
  Draw_Normal_Button(Palette_Blue);
  Draw_Normal_Button(Palette_Yellow);
  Draw_Normal_Button(Palette_Cyan);
  Draw_Normal_Button(Palette_Purple);
  Draw_Normal_Button(Palette_White);
  Draw_Normal_Button(Palette_Black);
  Draw_Box(Board,LCD_BLACK,-1);

  LCD_Refresh();//刷新 Data Cache，更新图像
}

//触摸扫描与处理
void Touch_Scan()
{
  u8 i = 0;
  u16 TpX_Last[5],TpY_Last[5];

  GT911_Scan(&Touch_LCD);
  //绘制线条
  for(i=0;i<Touch_LCD.Touch_Num;i++) {
    if(Touch_LCD.Touched & (1<<i)) {
      if(Touch_LCD.Touched != Touch_LCD.Touched_Last) {
        TpX_Last[i] = Touch_LCD.Tp_X[i];
        TpY_Last[i] = Touch_LCD.Tp_Y[i];
      }
      if((Touch_LCD.Tp_X[i] > BdX1(Board))&&(Touch_LCD.Tp_X[i] <
BdX2(Board))&&
        (Touch_LCD.Tp_Y[i] > BdY1(Board))&&(Touch_LCD.Tp_Y[i] <
BdY2(Board))) {
```

```
        LCD_Draw_BLine(TpX_Last[i],
TpY_Last[i],Touch_LCD.Tp_X[i],Touch_LCD.Tp_Y[i],Palette_Color);
        LCD_Refresh();//刷新 Data Cache，更新图像
      }
      TpX_Last[i] = Touch_LCD.Tp_X[i];
      TpY_Last[i] = Touch_LCD.Tp_Y[i];
    }
  }

  //调色盘、清屏(调色盘按下有特效)
  if((!Touch_LCD.Touched_Last)&&(Touch_LCD.Touched)) {
    //Clear 清屏按键
    if(Judge_TpXY(Touch_LCD,Clear.Box)) {
      Fill_Box(Board,LCD_WHITE,0);
      Draw_Normal_Button(Clear);
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    //判断按键坐标，绘制按键特效
    if(Judge_TpXY(Touch_LCD,Palette_Red.Box)) {
      Draw_Button_Effect(Palette_Red);
      Palette_Color = LCD_RED;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Judge_TpXY(Touch_LCD,Palette_Green.Box)) {
      Draw_Button_Effect(Palette_Green);
      Palette_Color = LCD_GREEN;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Judge_TpXY(Touch_LCD,Palette_Blue.Box)) {
      Draw_Button_Effect(Palette_Blue);
      Palette_Color = LCD_BLUE;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Judge_TpXY(Touch_LCD,Palette_Yellow.Box)) {
      Draw_Button_Effect(Palette_Yellow);
      Palette_Color = LCD_YELLOW;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Judge_TpXY(Touch_LCD,Palette_Cyan.Box)) {
      Draw_Button_Effect(Palette_Cyan);
      Palette_Color = LCD_CYAN;
      LCD_Refresh();//刷新 Data Cache，更新图像
```

```
    }

    if(Judge_TpXY(Touch_LCD,Palette_Purple.Box)) {
      Draw_Button_Effect(Palette_Purple);
      Palette_Color = LCD_PURPLE;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Judge_TpXY(Touch_LCD,Palette_White.Box)) {
      Draw_Button_Effect(Palette_White);
      Palette_Color = LCD_WHITE;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Judge_TpXY(Touch_LCD,Palette_Black.Box)) {
      Draw_Button_Effect(Palette_Black);
      Palette_Color = LCD_BLACK;
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    //清除其它按键特效
    if(Palette_Color != LCD_RED) {
      Draw_Normal_Button(Palette_Red);
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Palette_Color != LCD_GREEN) {
      Draw_Normal_Button(Palette_Green);
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Palette_Color != LCD_BLUE) {
      Draw_Normal_Button(Palette_Blue);
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Palette_Color != LCD_YELLOW) {
      Draw_Normal_Button(Palette_Yellow);
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Palette_Color != LCD_CYAN) {
      Draw_Normal_Button(Palette_Cyan);
      LCD_Refresh();//刷新 Data Cache，更新图像
    }

    if(Palette_Color != LCD_PURPLE) {
```

```
        Draw_Normal_Button(Palette_Purple);
        LCD_Refresh();//刷新 Data Cache，更新图像
      }

      if(Palette_Color != LCD_WHITE) {
        Draw_Normal_Button(Palette_White);
        LCD_Refresh();//刷新 Data Cache，更新图像
      }

      if(Palette_Color != LCD_BLACK) {
        Draw_Normal_Button(Palette_Black);
        LCD_Refresh();//刷新 Data Cache，更新图像
      }
    }
}
```

主函数中主要执行 Init_Homepage 和 Touch_Scan 两个函数，这里我们逐个介绍这两个函数。

1.5.1 Init_Homepage 函数

函数用于初始化外设和主界面，该函数所包含的各个函数功能如下：

表 6 函数功能

函数名	功能
LCD_Init	LCD 显示初始化
SII9022_Init	初始化 SII9022HDMI 芯片配置
GT911_Init	初始化 GT911 触摸芯片
Draw_Normal_Button	普通按键绘制
Draw_Button_Effect	特殊按键绘制
Draw_Box	方框绘制
LCD_Refresh	刷新 Dcache，更新图像数据

其中，大部分函数在前面课程中有过讲解，这里主要给大家讲一下 GT911_Init 函数。该函数用于初始化 GT911 触摸芯片，函数开始工作时，首先会初始化 IIC 和 GPIO 控制器，随后通过 GPIO 设置触摸芯片的设备地址。如下所示：

```
IIC_Init(400000); //初始化 I2C,频率为 100K
PS_GPIO_Init();   //初始化 PS 端 GPIO

PS_GPIO_SetMode(TP_RST_PIN, OUTPUT, 1);    //INT 引脚设置为输出 1
PS_GPIO_SetMode(TP_INT_PIN, OUTPUT, 0);    //RST 输出为 0，复位
usleep(10000);    //延时 10ms
PS_GPIO_SetPort(TP_INT_PIN, 1);    //RST 输出为 1，释放复位
usleep(10000);    //延时 10ms
PS_GPIO_SetPort(TP_RST_PIN, 0);    //INT 输出为 0
```

```
usleep(100000);    //延时 100ms
```

在表 2 中我们介绍过触摸芯片一共有两种设备地址供我们选择，分别为 0xBA/0xBB 和 0x28/0x29。主控在上电初始化时或通过 Reset 脚复位（唤醒）时，均需要设定 I2C 设备地址。设备地址的设置是通过控制 Reset 和 INT 口时序进行设定的。

设定地址为 0x28/0x29 的时序如下图 1-5 所示。

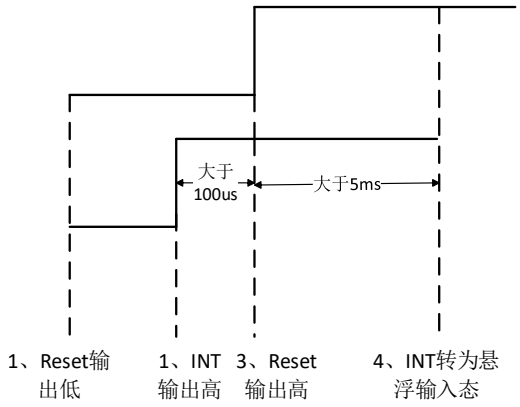

图 1-5 地址为 0x28/0x29 的时序图

设定地址为 0xBA/0xBB 的时序如下图 1-6 所示：

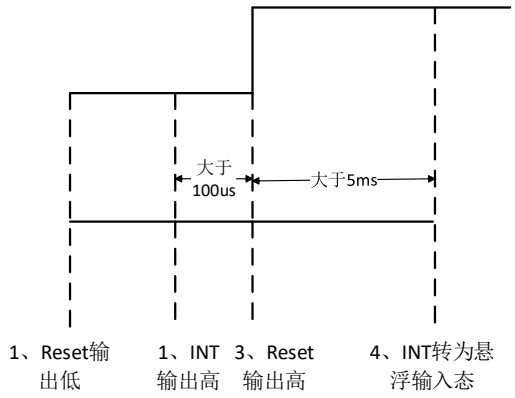

图 1-6 地址为 0xBA/0xBB 的时序图

代码中是将地址设置为 0x28/0x29 这一组。

设置完地址后，函数会读取触摸芯片 ID，正确 ID 为“911 ”（hex 码格式）。然后，会读取版本号，并更新配置信息。具体代码如下：

```
GT911_RD_Reg(GT_PID_REG,id,4);     //读取 ID
printf("ID:%s\n",id);    //打印 ID
buff[0]=0X02;
GT911_WR_Reg(GT_CTRL_REG,buff,1); //软复位 GT911
GT911_RD_Reg(GT_CFGS_REG,buff,1); //读取 GT_CFGS_REG 寄存器
printf("Previous version: %c\n",buff[0]);  //显示之前配置的版本号（A~Z）
```

```
if(buff[0] < GT911_CFG_TBL[0])    //如果之前配置版本小于预配置版本
GT911_Send_Cfg(1);    //更新配置并保存
GT911_RD_Reg(GT_CFGS_REG,buff,1);    //读取 GT_CFGS_REG 寄存器
printf("Current version: %c\n",buff[0]);    //显示当前配置的版本号
（A~Z）
usleep(10000);    //延时 10ms
buff[0]=0X00;
GT911_WR_Reg(GT_CTRL_REG,buff,1);//结束复位
```

芯片的 ID 通过 GT_PID_REG 寄存器读取，查询宏定义可以知道，正是前面我们讲解过的产品 ID 寄存器（0x8140~ 0x8143）

```
//GT911 常用寄存器
#define GT_CTRL_REG      0X8040      //GT911 控制寄存器
#define GT_CFGS_REG      0X8047      //GT911 配置起始地址寄存器
#define GT_CHECK_REG     0X80FF      //GT911 校验和寄存器
#define GT_PID_REG       0X8140      //GT911 产品 ID 寄存器
```

更新配置信息使用的函数是 GT911_Send_Cfg 函数，其需要配置的信息在 1.2.1.2 节中已经列举出来了。该函数只有一个变量 cmd，为 1 表示更新配置信息，为 0 不更新。必须保证新的版本号大于等于其内部 flash 原有版本号，才会更新配置。在写入配置信息表之前需要进行计算校验和，预设值为0。最后两个字节分别为“配置信息校验（0x8047 到 0x80FE 寄存器内部字节和的补码）和配置已更新标记（由主控写入标记 0x01）”。函数的代码实现如下：

```
/**
  ************************************************************************
*********
  * @功能描述: 发送 GT911 配置参数
  *
  * @输入参数: mode:0,参数不保存到 flash;1,参数保存到 flash
  ************************************************************************
*********
**/
uint8_t GT911_Send_Cfg(uint8_t mode)
{
    uint8_t buf[2];
    uint8_t i=0;
    buf[0]=0;
    buf[1]=mode;    //是否掉电保存
    for(i=0; i<sizeof(GT911_CFG_TBL); i++)
        buf[0]+=GT911_CFG_TBL[i]; //计算校验和
    buf[0]=(~buf[0])+1;
    GT911_WR_Reg(GT_CFGS_REG,(uint8_t*)GT911_CFG_TBL,sizeof(GT911_CFG_
TBL));//发送寄存器配置
    GT911_WR_Reg(GT_CHECK_REG,buf,2);//写入校验和,和配置更新标记
    return 0;
```

```
}
```

1.5.2 Touch_Scan 函数

函数用于扫描触摸屏，获取触摸点相关数据，并基于触摸点，实现相关特效。这里我们主要讲解触摸点获取功能，特效的实现，用户可以自行参看例程源码。

触摸点的获取由 GT911_Scan 函数实现，其实现步骤如下：

(1)读取当前触摸情况，也就是读取 0x814E 寄存器中的值，当其中的 buffer status 位为 1 时（0x80）表示坐标（或按键）已经准备好，主控可以读取；0 表示未就绪，数据无效，清除所有触摸点按下有效标志

(2)判断几个触摸点按下，0x814E 的 Bit3~0 表示有效触点的个数。

(3)~(0xFF << (sta & 0x0F))将点的个数转换为触摸点按下有效标志。

(4)分别判断触摸点 1-5 是否被按下，被按下则读取对应触摸点坐标数据，为了方便操作，将触摸点寄存器拟合在一个数组中。

(5)使用完后清空 0x814E 寄存器所有标志，返回读取到的坐标值。

函数的实现代码如下所示：

```
/**
  ************************************************************************
*********
  * @功能描述：扫描触摸屏(轮询方式)
  * @输入值：Touch_Data 结构体变量
  ************************************************************************
*********
**/
void GT911_Scan(Touch_Data *Touch_LCD)
{
    uint8_t i;
    uint8_t State = 0;
    uint8_t Data_XY[4]={0};
    uint8_t Zero = 0;

    Touch_LCD->Touched_Last = Touch_LCD->Touched;//保存上次的触摸状态
    GT911_RD_Reg(GT_GSTID_REG,&State,1);     //读取触摸状态寄存器
    //最高位为 1 表示数据有效
    if(State & 0X80) {
        Touch_LCD->Touch_Num = State & 0X0F;         //获取触点数量，数
量为 0 表示无按键

        //将触摸点换算到对应 bit，为 1 表示触摸，为 0 表示未触摸
        Touch_LCD->Touched = 0x1F >> (5 - Touch_LCD->Touch_Num);
```

```
        //依次获取触摸点坐标
        for(i=0;i<5;i++) {//for(i=0;i<Touch_LCD->Touch_Num;i++) {
            GT911_RD_Reg(GT911_TPX_TBL[i],Data_XY,4);  //读取 XY 坐标值
            if(((((uint16_t)Data_XY[1]<<8)+Data_XY[0]) <= 800) &&
                    ((((uint16_t)Data_XY[3]<<8)+Data_XY[2]) <= 480))
            Touch_LCD->Tp_X[i] = ((uint16_t)Data_XY[1]<<8)+Data_XY[0];
            Touch_LCD->Tp_Y[i] = ((uint16_t)Data_XY[3]<<8)+Data_XY[2];
        }
        GT911_WR_Reg(GT_GSTID_REG, &Zero, 1);    //写 0 清寄存器来开启下
一次检测
    }
}
```

至此，我们便大致了解了主函数的工作原理，至于更加细致和完整的代码内容，用户可以参看历程源码。

接下来编译工程，此时工程有可能会出现如下报错：

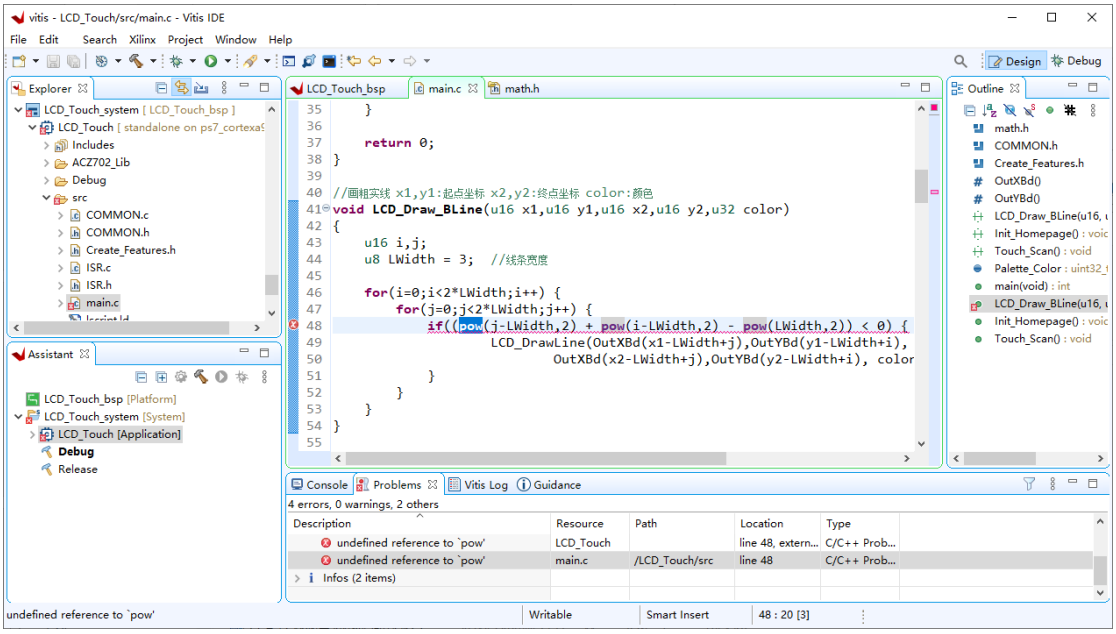

图 1-7 'pow'未定义

可以看到报错是说 pow 未定义，该函数用于求幂，此时如果我们按住 ctrl 进行跳转可以看到该函数声明于 math.h 中，如图 1-8 所示：

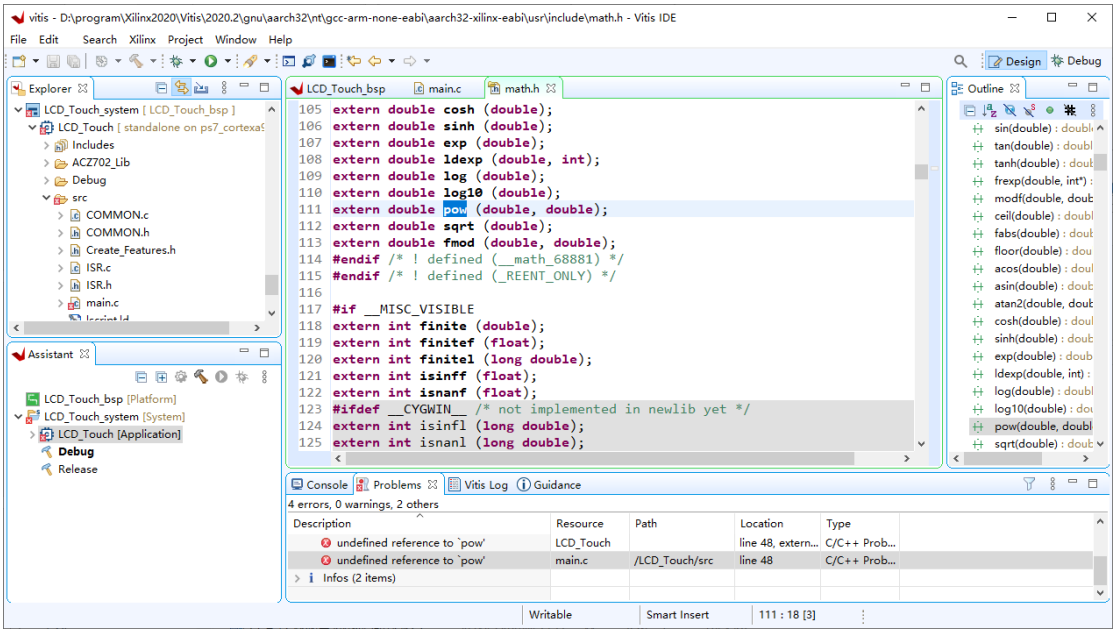

图 1-8 'pow'函数声明

而我们在 COMMON.h 中早已包含了该头文件，如图 1-9 所示：

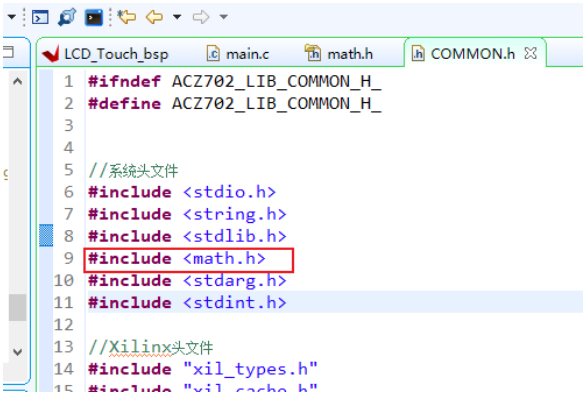

图 1-9 系统头文件包含

对于这种情况，我们需要为工程链接 math 库，右键单击应用工程，按照图 1-10 顺序，在 libraries 项中添加 m 即可。

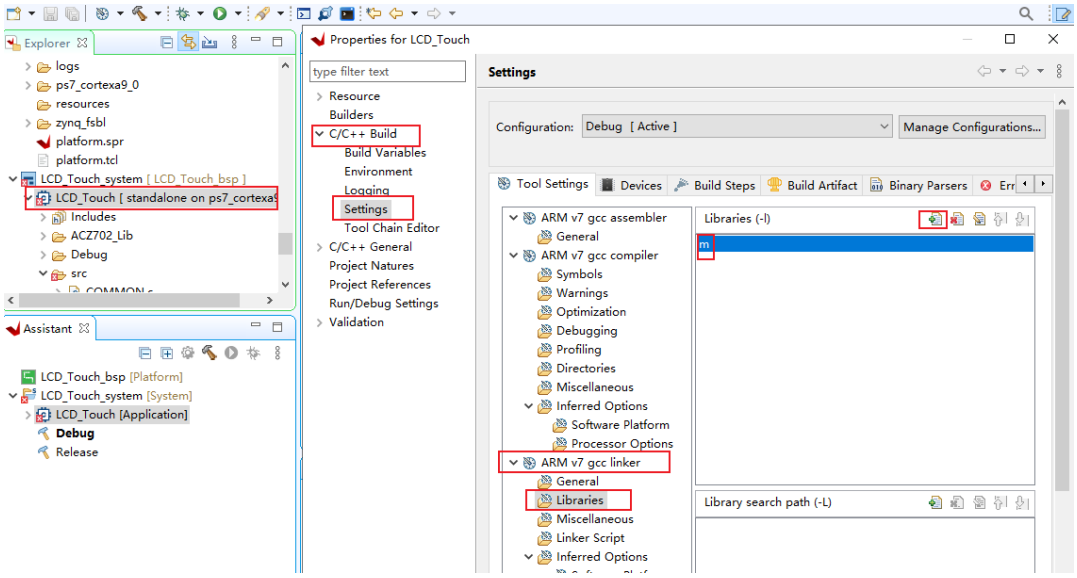

图 1-10　链接 math 库

链接完成后再次编译工程，确认工程无误后，我们可以连接硬件，准备进行烧录验证了。

1.6 板级验证

1.6.1 硬件连接

本次设计我们需要在触摸屏上实现触摸画板的功能，显示的图像还需要通过 SIi9022 输出到 HDMI 显示器上显示。因此，本次设计除了连接电源与下载器外，还需要连接 TFT 屏与 HDMI 显示设备（HDMI_1 接口），具体硬件连接如图 1-11 所示：

图 1-11　硬件连接

硬件连接时，需要注意以下两点：

1. 使用软排线连接开发板与触摸屏时，注意蓝色胶条应该朝向黑色压盖侧
2. 连接 HDMI 显示器时，HDMI 线缆应该插接 HDMI_1 接口

连接完毕后，将开发板上电，准备进行烧录。

1.6.2 功能演示

将程序烧录进开发板中，烧录时需要注意勾选配置 FPGA。烧录完成后的触摸屏以及 HDMI 显示器界面如图 1-12 所示：

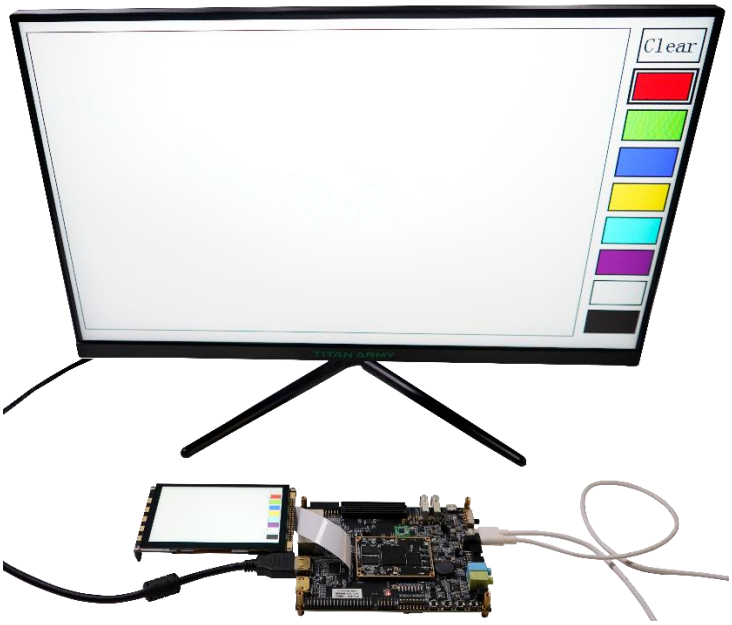

图 1-12 板级验证现象

通过点击触摸屏右侧的颜色选取按钮，可以改变画笔的颜色，如图 1-13 所示：

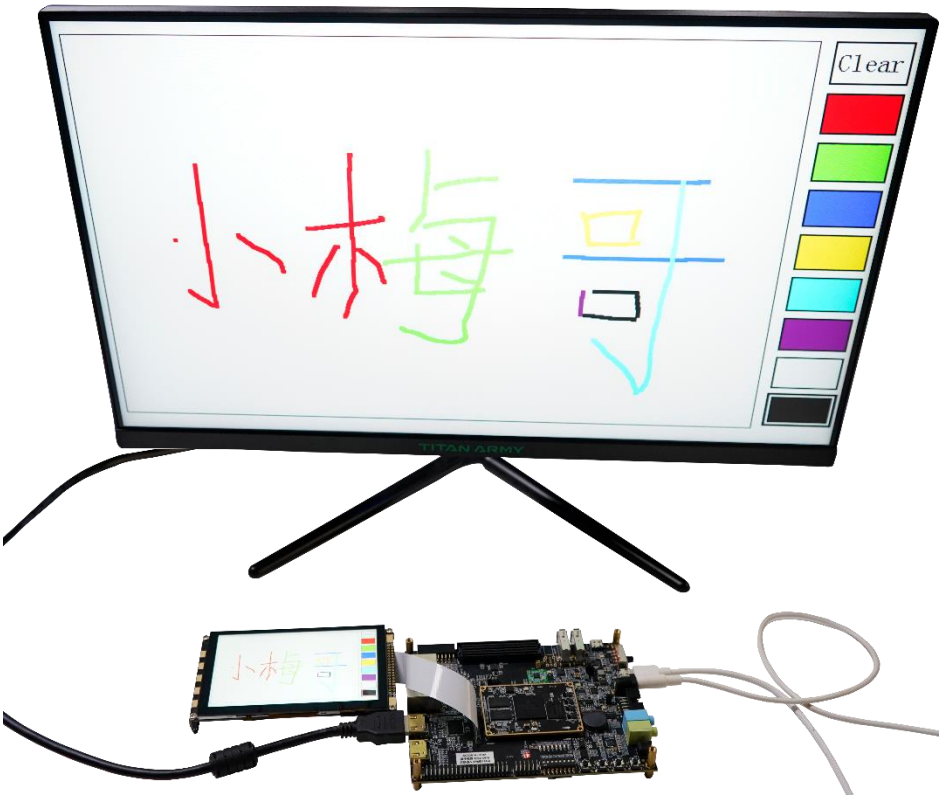

图 1-13 画笔颜色选取绘制效果示意图

可以看到，绘图笔迹清晰，颜色鲜明，HDMI 上显示稳定且无色差，说明本次设计成功。

1.7 总结

本次实验通过 I2C 驱动触摸芯片，成功获取了屏幕坐标，并且根据根据得到的坐标改变画笔的颜色和清除绘画区域的功能，实验中需要注意以下几点：

1.在连接 TFT LCD 屏的时候，注意软排线的蓝色胶条应该朝向黑色压盖；在连接 HDMI 时，注意应该插接 HDMI_1 接口。

2.在将触摸坐标和屏幕坐标进行结合的时候，一定要注意你自己定义 LCD 屏坐标的原点以及 X 轴和 Y 轴的位置是否与触摸坐标是一致的，比如本次实验，这两个坐标就不是一一对应的，需要对一方的坐标进行修改，实验中修改的触摸得到的坐标的位置，其实也可以修改 LCD 屏对应的坐标。