8 触摸画板实验

8.1. 背景介绍

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。

8.1.1. 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。

8.1.2. 触摸芯片简介

本次实验使用的 TFT LCD 屏上的触摸芯片为 GT1151，其在通信过程中需要使用到的引脚如下表 8- 1 所示。

表 8- 1 触摸芯片引脚说明表

模块引脚序号	名称	说明
30	MOSI	电容触摸屏 IIC_SDA 信号（CT_SDA）
31	PEN	电容触摸屏中断信号（CT_INT）
33	CS	电容触摸屏复位信号（CT_RST）
34	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。GT1151 与主机的通信采用的是标准的 I2C 通信，最高可以支持至 400Kbps。当主机采用 200Kbps 以上的通信速率的时候，需要特别注意 I2C 的外部上拉电阻阻值，以保证 SCL、SDA 边沿足够陡峭。GT1151 在通信中始终作为从设备，其 I2C 的设备地址由 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。GT1151 一共

有两个从设备地址可以选择，如下表 8- 2 所示。

表 8- 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

8.1.3. 触摸芯片关键寄存器

GT1151 的寄存器有很多，这里简单的介绍一下 GT1151 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。

8.1.3.1. 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 GT1151，这四个寄存器读出来就是：1，1，5，8 四个字符的（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。

8.1.3.2. 配置寄存器组（0x8047~0x8100）

一共有 186 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：

```
{ 0X60, 0XE0, 0X01, 0X20, 0X03, 0X05, 0X35,
0X00, 0X02, 0X08, 0X1E, 0X08, 0X50, 0X3C, 0X0F, 0X05, 0X00, 0X00, 0XFF,
0X67, 0X50, 0X00, 0X00, 0X18, 0X1A, 0X1E, 0X14, 0X89, 0X28, 0X0A, 0X30,
0X2E, 0XBB, 0X0A, 0X03, 0X00, 0X00, 0X02, 0X33, 0X1D, 0X00, 0X00, 0X00,
0X00, 0X00, 0X00, 0X00, 0X32, 0X00, 0X00, 0X2A, 0X1C, 0X5A, 0X94, 0XC5,
0X02, 0X07, 0X00, 0X00, 0X00, 0XB5, 0X1F, 0X00, 0X90, 0X28, 0X00, 0X77,
0X32, 0X00, 0X62, 0X3F, 0X00, 0X52, 0X50, 0X00, 0X52, 0X00, 0X00, 0X00,
0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00,
0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X0F, 0X0F, 0X03, 0X06,
0X10, 0X42, 0XF8, 0X0F, 0X14, 0X00, 0X00, 0X00, 0X00, 0X1A, 0X18, 0X16,
0X14, 0X12, 0X10, 0X0E, 0X0C, 0X0A, 0X08, 0X00, 0X00, 0X00, 0X00, 0X00,
0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00,
0X00, 0X00, 0X00, 0X29, 0X28, 0X24, 0X22, 0X20, 0X1F, 0X1E, 0X1D, 0X0E,
0X0C, 0X0A, 0X08, 0X06, 0X05, 0X04, 0X02, 0X00, 0XFF, 0X00, 0X00, 0X00,
0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0XFF, 0XFF, 0XFF, 0XFF,
0XFF, 0XFF, 0XFF, 0XFF, 0XFF, 0XFF, 0XFF, 0XFF, 0XFF, 0, 0};
```

8.1.3.3. 状态寄存器（0x814E）

对于状态寄存器的每一位的介绍如下表 8- 3 所示：

表 8- 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	Proximity Valid，接近感应有效置 1
4	HaveKey，1 表示有按键，0 表示无按键（已经松键）
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。

8.1.3.4. 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 8- 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 8- 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

8.1.4. I2C 通讯协议简介

I2C 总线（I2C bus，Inter-IC bus）是一个双向的两线连续总线，提供集成电路（ICs）之间的通信线路。Philips 公司推出的 I2C 总线采用一条数据线（SDA），加一条时钟线（SCL）来完成数据的传输及外围器件的扩展。如下图 8.1 所示主控芯片引出两条线 SCL，SDA 线，在一条 I2C 总线上可以接很多 I2C 设备，比如 EEPROM、RTC 以及任何拥有 I2C 接口的器件。数据可以从主设备传到从设备上，从设备也能传数据到主设备上，即双向传输。

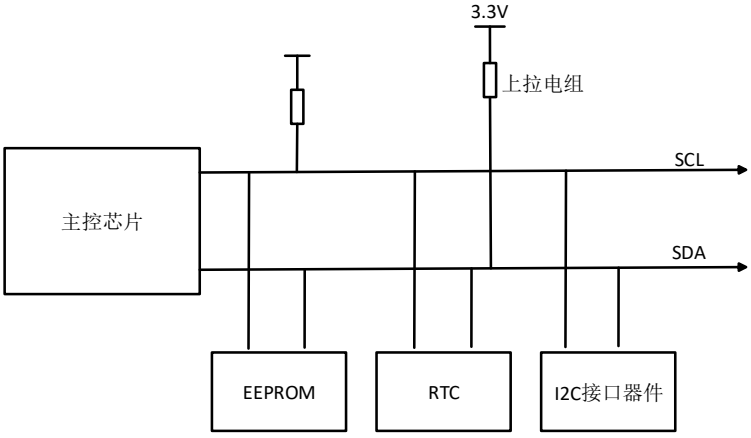

图 8. 1 I2C 物理总线拓扑结构图

通常我们为了方便把 I2C 设备分为主设备和从设备，基本上谁控制时钟线（控制 SCL 的电平高低变换）谁就是主设备。I2C 的主要特点如下所示：

1. I2C 主设备功能：主要产生时钟，起始信号和停止信号。
2. I2C 从设备功能：可编程的 I2C 地址检测，停止位检测。
3. I2C 的一个优点是它支持多主控，其中任何一个能够进行发送和接收的设备都可以成为主总线。一个主控能够控制信号的传输和时钟频率。当然，在任何时间点上只能有一个主控。
4. 支持不同速率的通讯速度，标准速度（最高速度 100Khz），快速（400Khz）。
5. SCL 和 SDA 都需要接上拉电阻以保证数据的稳定性，减少干扰。
6. I2C 是半双工，而不是全双工，同一时间只可以单向通信。

8.1.4.1. I2C 时序简介

I2C 协议整体时序图如图 8. 2 所示。

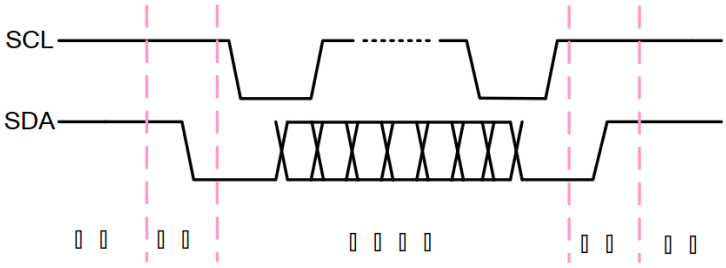

图 8. 2 I2C 总线时序图

对上述整体的时序分析如下所示：

1. 空闲状态：SCL 为高电平，SDA 为高电平。
2. 起始位：SCL 为高电平时，SDA 出现下降沿，产生一个起始位。
3. 结束位：SCL 为高电平时，SDA 出现上升沿，产生一个停止位。
4. 读写状态：主要包括数据的串行输出输入和数据接收方对数据发送方的

响应信号。

传输数据的过程中，SDA 线上的数据必须在时钟的高电平周期保持稳定，数据线的高或者低电平状态，只有在 SCL 时钟信号为低电平的时候才能改变。SCL 为高电平的时候表示有效数据，此时，若 SDA 为高电平表示为“1”，低电平表示“0”；SCL 为低电平时，表示无效数据，此时 SDA 会进行电平转换，为下次数据传输做准备。

I2C 通信过程中，还有一个重要的信号：响应信号。主/从设备在发送完一次数据后，需要一个 ACK 响应信号。每个通信周期传输 8 个数据在第九个通信周期，发送端将 SDA 总线拉高，然后释放总线的控制权。如果接收端在第九个周期将 SDA 拉低，就是发出了 ACK 信号，如果在第 9 个周期 SDA 一直是高电平则代表没有发出 ACK 信号，表示从机和主机没有进行正常的通信。

8.1.4.2. I2C 写时序

I2C 的写时序可以分为单字节的写时序和连续的写时序（页写时序）。

1. 单字节的写时序

如下图 8.3 代表的是 1 字节地址段器件单字节写时序。图 8.4 代表的是 2 字节地址段器件单字节写时序。

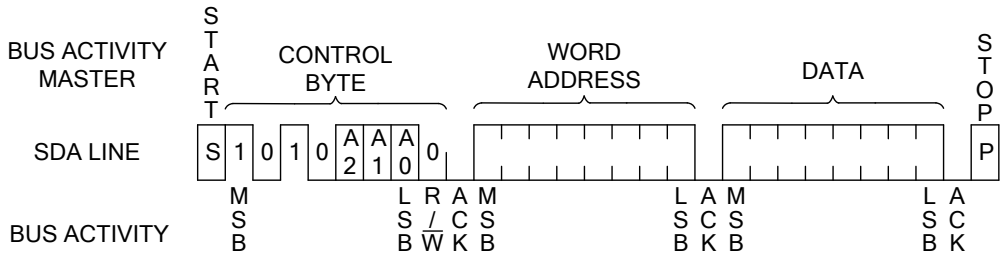

图 8.3 1 字节地址段器件单字节写时序

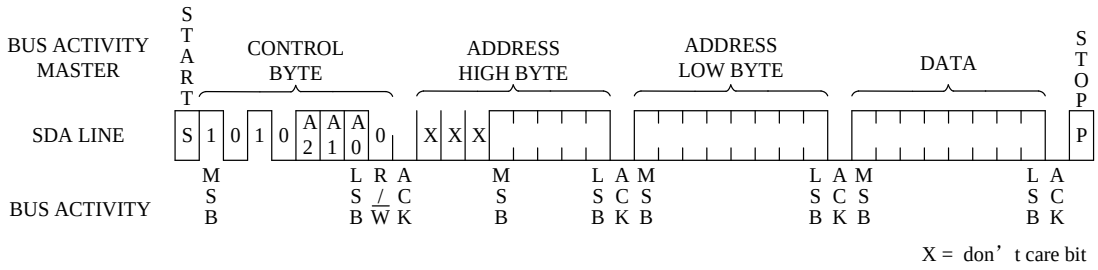

图 8.4 2 字节地址段器件单字节写时序

对上面的 1 字节地址段器件单字节写时序图分析如下：

(1) 主机设置 SDA 为输出；

(2) 主机发起起始信号；

(3) 主机传输器件地址字节，其中最低位为 0，表明为写操作；

(4) 主机设置 SDA 为三态门输入，读取从机应答信号；

(5) 读取应答信号成功，主机设置 SDA 为输出，传输 1 字节地址数据（2 字节地址此操作执行两次，先传输高位的地址，再传输低位地址）；

(6) 主机设置 SDA 为三态门输入，读取从机应答信号；

(7) 设置 SDA 为三态门输入，读取从机应答信号，对于两字节地址段器件，接着步骤 9；对于 1 字节地址段器件，直接跳转到步骤 11；

(8) 读取应答信号成功，主机设置 SDA 为输出，传输待写入的数据（对于两字节地址段器件）；

(9) 设置 SDA 为三态门输入，读取从机应答信号（两字节地址段器件）；

(10) 读取应答信号成功，主机产生 STOP 位，终止传输。

2. 连续的写时序

连续写（也称页写，需要注意的是 I2C 连续写时序仅部分器件支持）是主机连续写多个字节数据到从机，这个和单字节写操作类似，连续多字节写操作也是分为 1 字节地址段器件和 2 字节地址段器件的写操作，其时序图分别如图 8.5 和图 8.6 所示。

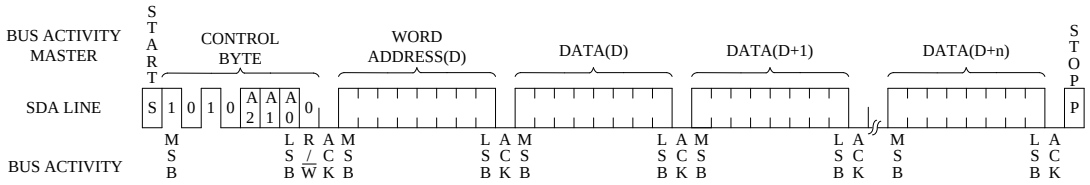

图 8.5 1 字节地址段器件连续多字节写时序

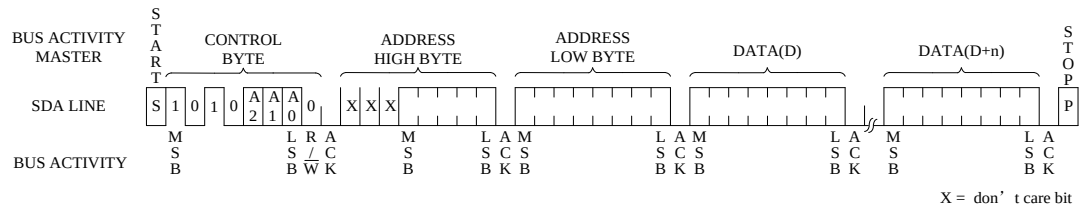

图 8.6 2 字节地址段器件连续多字节写时序

对上述时序分析如下所示：

(1) 主机设置 SDA 为输出；

(2) 主机发起起始信号；

(3) 主机传输器件地址字节，其中最低位为 0，表明为写操作；

(4) 主机设置 SDA 为三态门输入，读取从机应答信号；

(5) 读取应答信号成功，主机设置 SDA 为输出，传输 1 字节地址数据；

(6) 主机设置 SDA 为三态门输入，读取从机应答信号；

(7) 读取应答信号成功后，主机设置 SDA 为输出，对于两字节地址段器件，

传输低字节地址数据，对于 1 字节地址段器件，传输待写入的第 1 个数据；

(8) 设置 SDA 为三态门输入，读取从机应答信号，对于两字节地址段器件，接着步骤 9；对于 1 字节地址段器件，直接跳转到步骤 11；

(9) 读取应答信号成功后，主机设置 SDA 为输出，传输待写入的第 1 个数据（两字节地址段器件）；

(10) 设置 SDA 为三态门输入，读取从机应答信号（两字节地址段器件）；

(11) 读取应答信号成功后，主机设置 SDA 为输出，传输待写入的下一个数据；

(12) 设置 SDA 为三态门输入，读取从机应答信号；n 个数据被写完，转到步骤 13，若数据未被写完，转到步骤 11；

(13) 读取应答信号成功后，主机产生 STOP 位，终止传输。

8.1.4.3. I2C 读时序

I2C 的读时序可以分为单字节的读时序和连续的读时序（页读时序）。

1. 单字节的读时序

I2C 的单字节的读时序的时序图如图 8.7 和图 8.8 所示，分别是 1 字节地址段器件单字节数据读操作时序和 2 字节地址段器件单字节数据读操作时序。

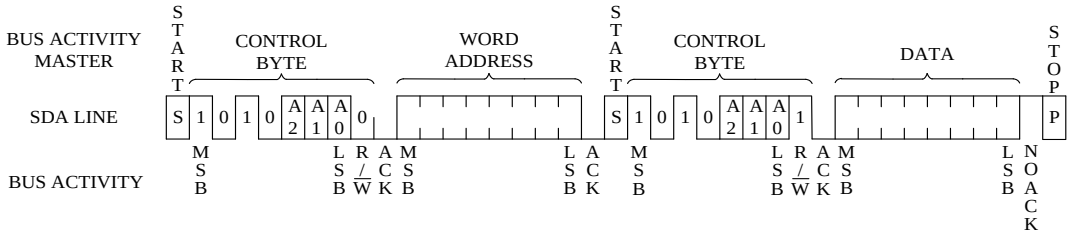

图 8.7 1 字节地址段器件单字节数据读操作时序

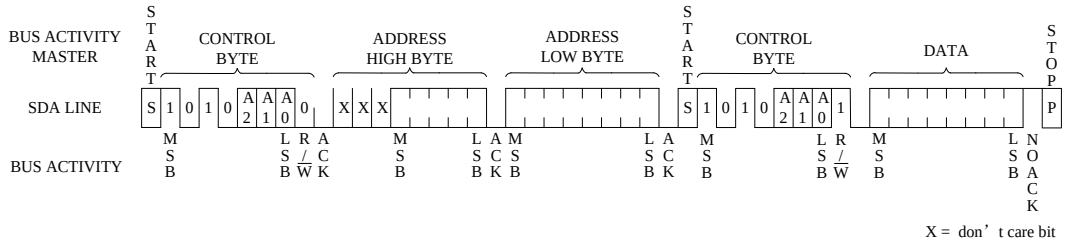

图 8.8 2 字节地址段器件单字节数据读操作时序

对于上述时序图分析如下：

(1) 主机设置 SDA 为输出；

(2) 主机发起起始信号；

(3) 主机传输器件地址字节，其中最低位为 0，表明为写操作；

(4) 主机设置 SDA 为三态门输入，读取从机应答信号；

(5) 读取应答信号成功，主机设置 SDA 输出，传输 1 字节地址数据；

(6) 主机设置 SDA 为三态门输入，读取从机应答信号；

(7) 读取应答信号成功，主机设置 SDA 输出，对于两字节地址段器件，传输低字节地址数据；对于 1 字节地址段器件，无此步骤，直接跳转到步骤 8；

(8) 主机发起起始信号；

(9) 主机传输器件地址字节，其中最低位为 1，表明为读操作；

(10) 设置 SDA 为三态门输入，读取从机应答信号；

(11) 读取应答信号成功，主机设置 SDA 为三态门输入，读取 SDA 总线上的一个字节的数据；

(12) 产生无应答信号（高电平）（无需设置为输出高电平，因为总线会被自动拉高）；

(13) 主机产生 STOP 位，终止传输。

2. 连续的读时序（页读时序）

连续读是主机连续从从机读取多个字节数据，这个和单字节读操作类似，连续多字节读操作也是分为 1 字节地址段器件和 2 字节地址段器件的读操作，图 8.9 和图 8.10 分别为 1 字节地址段器件和 2 字节地址段器件连续多字节读时序图。

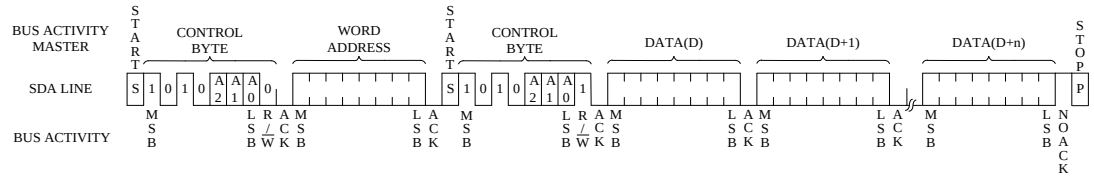

图 8.9 1 字节地址段器件多字节数据读操作时序图

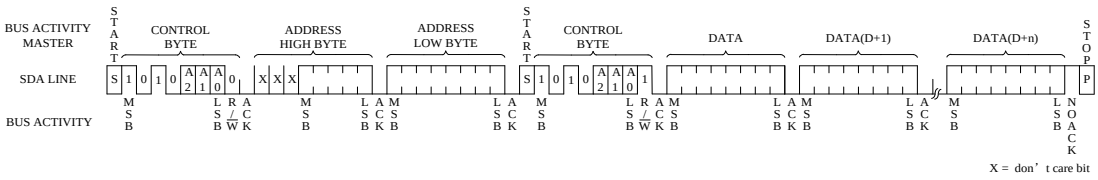

图 8.10 2 字节地址段器件连续多字节读操作时序图

对上述时序图分析如下：

(1) 主机设置 SDA 为输出

(2) 主机发起起始信号

(3) 主机传输器件地址字节，其中最低位为 0，表明为写操作。

(4) 主机设置 SDA 为三态门输入，读取从机应答信号。

(5) 读取应答信号成功，主机设置 SDA 输出，传输 1 字节地址数据

(6) 主机设置 SDA 为三态门输入，读取从机应答信号。

(7) 读取应答信号成功，主机设置 SDA 输出，对于两字节地址段器件，传输低字节地址数据；对于 1 字节地址段器件，无此步骤；直接跳转到步骤 9；

(8) 主机发起起始信号；

(9) 主机传输器件地址字节，其中最低位为 1，表明为读操作；

(10) 设置 SDA 为三态门输入，读取从机应答信号；

(11) 设置 SDA 为三态门输入，读取 SDA 总线上的第 1 个字节的数据；

(12) 主机设置 SDA 输出，发送一位应答信号；

(13) 设置 SDA 为三态门输入，读取 SDA 总线上的下一个字节的数据；若 n 个字节数据读完成，跳转到步骤 14，若数据未读完，跳转到步骤 12；

主机设置 SDA 输出，产生无应答信号（高电平）（无需设置为输出高电平，因为总线会被自动拉高）；

8.1.5. I2C 寄存器映射

SEAL Cortex-M3 中有一个 I2C 主模块，在下表 8- 5 中显示 I2C 的寄存器映射。寄存器地址=基地址+基址偏移，基址为 0x4000_6000。

表 8- 5 I2C 寄存器映射表

Name	Base offset	Type	Width	Reset value	Description
CONTROL_REG	0x00	RW	32	0x0	[0]ctrl_tx_enable; [1]ctrl_tx_irq_enable; [3:2] ctrl_tx_buff_wlevel; [4]ctrl_start; [5] ctrl_stop; [6] ctrl_auto_stop; [7] ctrl_err_irq_enable; [8] ctrl_rx_enable; [9] ctrl_rx_irq_enable; [11:10] ctrl_rx_buff_wlevel; [12] ctrl_ack_set:0:ACK;1:NACK; [13] ctrl_tc_irq_enable; [14] ctrl_stop_irq_enable; [15] ctrl_ms_mode:1:Master mode0:Slave mode [19:16] ctrl_num_bytes; [30:20] reserved; [31] ctrl_sw_rst: 1: All I2C registers will be reset to initial value;0: Reset released
FREQ_DIV	0x04	RW	24	0x0	[1:0] speed_mode:10:High;01:fast;00:standard; [2] Used to hold sclk in slave mode; [15:3] clock divide factor; [17:16] SDA total delay count selection: 00: total count is sda_del_cnt * 4;

					01: total count is sda_del_cnt * 2; 10: total count is sda_del_cnt; 11: total count is 0; [19:18] Reserved, must be kept cleared. [23:20] SDA delay count
OWN_ADDR	0x08	RW	11	0x0	[0] own_address_mode: 0: 7-bit address 1: 10-bit address [7:1] own_address_7bit: [1:7] 7 bit address [10:8] own_address_10bit: [1:10] 10 bit address
SLV_ADDR	0x0C	RW	16	0x0	[0] slv_address_mode: 0: 7-bit address 1: 10-bit address [7:1] slv_address_7bit: [1:7] 7 bit address [10:8] slv_address_10bit: [1:10] 10 bit address [14:11] Reserved, must be kept cleared [15] slv_address_RW: 0: Transmit and write 1: Receive and read
TX_DATA	0x10	RW	8	0x0	[7:0] TX Data buff
RX_DATA	0x14	RW	8	0x0	[7:0] RX Data buff
REG_STATUS1	0x18	RO	24	0x0	[0]Start bit status. This bit will turn to “1” when start signal is detected. Note that sw reset must be released first. [1]Address 7-bit sent/receive matched [2]Address 10-bit sent/receive matched [3] 1: Transmit FIFO is full. 0: Transmit FIFO is not full. [4] 1: Transmit FIFO is empty. 0: Transmit FIFO is not empty [8:5] This register shows the absolute value of TX FIFO’s write pointer minus read pointer. |tx_wptr-tx_rptr| [9] 1: Receive FIFO is full. 0: Receive FIFO is not full [10] 1: Receive FIFO is empty. 0: Receive FIFO is not empty [14:11] This register shows the absolute value of RX FIFO’s write pointer minus read pointer. |rx_wptr-rx_rptr| [15] Byte transfer finished/ Stop bit detect [16] Transmit completed flag, which means the transmitted data byte number has reached the number defined in ctrl_num_bytes section in control register [17] Data_received flag [19:18] Reserved, must be kept cleared [23:20] This register shows the number of data transmitted or received
REG_STATUS2	0x1C	RO	10	0x0	[0]Hardware general call received flag [1]High speed mode flag

					[2] Arbitration lost flag [3] Acknowledge bit failure flag [4] Bus busy flag, which means the I2C bus is not idle. [5] Bus error interrupt is detected [6] TX interrupt is raised [7] RX interrupt is raised [8] TC interrupt is raised [9] Stop bit detected interrupt is raised.

8.2. 实验介绍

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。

8.3. 建立 HQFPGA 工程

将 TFT LCD 屏实验的工程复制至存放本次实验的文件夹之下，并且对其进行修改，修改方式参考实验二中 2.3 节中的内容。

8.4. 物理管脚约束

在设计管理器界面，点击设计文件中的“ac208.upc”文件，对于 TFT LCD 屏的触摸引脚进行分配，根据表 8-1 中的内容可以得知，如果需要使用到触摸功能，需要添加的引脚为 I2C 通信使用到的引脚 SCL 和 SDA，触摸中断 INT 和 RST 引脚，添加如下语句：

```
#----IIC 接口接触摸屏-----------
#IIC_SCL lcd_gpio[27]
phycst.pin.set {GPIO[15]} J1
#IIC_SDA lcd_gpio[23]
phycst.pin.set {GPIO[16]} N1

#----TFT_LCD 触摸屏 INT 接口和复位接口-----------
#---------复位接口 lcd_gpio[26]--------------
phycst.pin.set {GPIO[28]} J2
#---------中断接口 lcd_gpio[24]--------------
phycst.pin.set {GPIO[29]} J3
```

8.5. 编译设计

保存完修改后的顶层文件和物理约束文件之后，点击全部运行，生成本次实验需要的 FPGA 侧的 bin 文件，操作如图 8.11 所示。

图 8. 11 编译运行整个文件

8.6. 建立 MDK 工程

将 TFT LCD 屏显示实验的 MDK 工程复制至本次工程的文件下，并对其进行修改，操作方式参考 2.8 节中的内容。

8.7. 软件设计

本次实验需要添加的库文件有三个，用户可以在我们给的例程中进行复制添加，添加方式参考 3.6.1 节中的内容。第一个需要添加的就是官方给的 I2C 驱动库“CM3DS_i2c”（\...工程源码\LCD_Touch\CM3_Softerware\CMSIS\Device\StdPeriph_Driver）。第二个需要添加的就是 I2C 应用库，我们使用了两种方法实现，一种是 IO 模拟 I2C 通信，对应的库文件为“io_i2c”（\...工程源码\LCD_Touch\CM3_Softerware\HARDWARE）；另外一种就是使用硬件 I2C，对应的库文件为“i2c”。最后一个需要添加的就是触摸芯片的应用库“touch”。

8.7.1. Touch 库

8.7.1.1. Touch_IIC_Init

触摸 I2C 初始化，我们 Touch 库中包含两种 I2C 实现的方式，可以在“Touch.c”进行选取，代码如下所示，需要使用哪一个种方式，就将其后面的数字置 1。

```
#define LCD_TOUCH_HAL 1 //选择 CM3 自带的 I2C
#define LCD_TOUCH_IO 0//选择 IO 模拟
```

初始化的时候，分为两种方式进行初始化：

IO 模拟 I2C 初始化，主要进行的操作就是关闭 SCL 和 SDA 引脚的中断和复用功能。然后将这两个引脚都设置为高电平，是 I2C 处于空闲状态，使用的时候调用如下代码：

```
io_i2c_init();//IO_IIC 初始化
```

硬件 I2C 初始化，将 I2C 通信的 SCL 和 SDA 引脚进行复用，配置 I2C 为主机模式（Control 寄存器的第[15]位为 1），标准速率模式（FREQ_DIV 寄存器的[1:0]位设置为 00），以及 I2C 作为主机的地址 CM3_I2C_OWN_ADDR

（OWN_ADDR 寄存器中写入对应的地址，本次实验写入的是 0x0B）,代码如下所示：

```
IIC_Init(CM3DS_MPS2_I2C, I2C_Mode_Master, I2C_Speed_Standard, CM3_I2C_OWN_ADDR);
```

8.7.1.2. GT9147_ID_SET

设置触摸芯片的设备地址。在 7.1.2 节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 8. 12 所示。

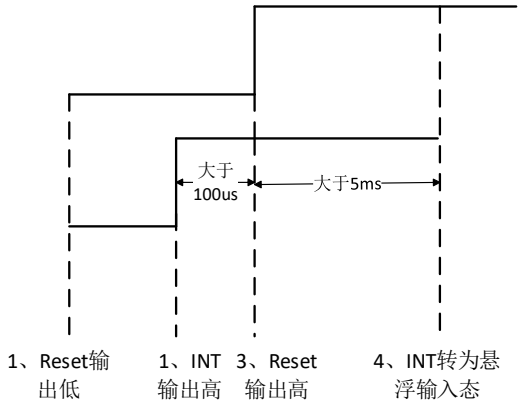

图 8. 12 地址为 0x28/0x29 的时序图

设定地址为 0xBA/0xBB 的时序如下图 8. 13 所示：

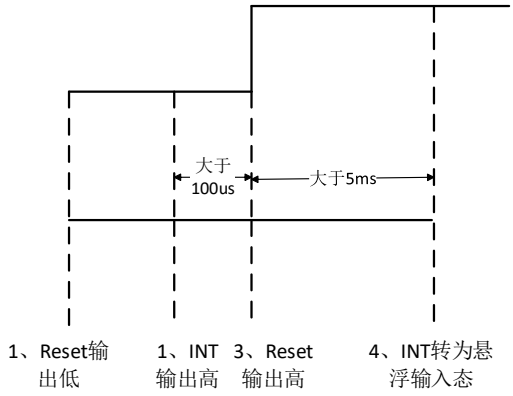

图 8. 13 地址为 0xBA/0xBB 的时序图

本次实验是将地址设置为 0x28/0x29 这一组，代码如下所示：

```
void GT9147_ID_SET()
{
    //设置 Int 和 Rst 信号为输出，用以设置 GT9147 的 I2C ID
    CM3DS_MPS2_GPIO0->OUTENABLESET |= ( 1 << Touch_RST);
```

```
    CM3DS_MPS2_GPIO0->OUTENABLESET |= ( 1 << Touch_INT);
    //RST 和 INT 都为 0:RST:GPIO[28]；INT：GPIO[29]
    GPIO_WriteBit(CM3DS_MPS2_GPIO0,Touch_RST,0);
    GPIO_WriteBit(CM3DS_MPS2_GPIO0,Touch_INT,0);
    delay_us(300);
    //INT 拉高为 1
    GPIO_WriteBit(CM3DS_MPS2_GPIO0,Touch_INT,1);
    delay_us(300);
    //RST 设为 1，INT 为 1，完成将 ID 设置为 0x28 的功能
    GPIO_WriteBit(CM3DS_MPS2_GPIO0,Touch_RST,1);
    GPIO_WriteBit(CM3DS_MPS2_GPIO0,Touch_INT,1);
    delay_us(8000);
    //设置 Int 信号为输入，RST 继续为输出
    CM3DS_MPS2_GPIO0->OUTENABLECLR = (1 << Touch_INT);//设置 INT 为输入
}
```

8.7.1.3. GT9147_Init

触摸芯片初始化。调用 GT9147_ID_SET 函数，设置触摸芯片的器件地址，读取触摸芯片的产品 ID，返回得到的 ID 应该为“1158（HEX 码格式）”,然后是读取版本号，如果版本号比较低的话，就更新配置信息。函数代码如下所示：

```
void GT9147_Init(void) {
    uint8_t temp = 0;
    unsigned char ID[4] = {0};
    GT9147_ID_SET();
    delay_us(10000);
    //读取产品 ID 号
    GT9147_RdNByte(GT9147_DevAdr0, GT9147_ProductIDReg, ID, 4);
    /********************写入配置信息********************/
    //读取版本号，第一个字节为版本号
    GT9147_RdNByte(GT9147_DevAdr0, GT9147_ConfigMsgReg, &temp, 1);
    if (temp < 0x60) //版本号较低，则更新配置信息
            {
        GT9147_SendConfigMSG(0x01);
    }
}
```

在该函数中主要使用到了 GT9147_RdNByte 函数和 GT9147_WrNByte 函数，函数的参数说明如下表 8- 6 所示：

表 8- 6 GT9147_RdNByte/GT9147_WrNByte 函数变量说明表

变量名称	变量意义
Daddr	器件地址
Waddr	寄存器首地址

a	接收数据缓冲区首地址
n	读数据的长度

根据 8.1.3 的内容可以得到，读 ID 和版本号的寄存器分别为 0x8140~0x8143。通过 GT9147_RdNByte 即可读取对应的信息。

更新配置信息使用的函数是 GT9147_SendConfigMSG 函数，其需要配置的信息在 8.1.3.2 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时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 之字节和的补码）和配置已更新标记（由主控写入标记 0x01）”。最后利用 GT9147_WrNByte 将配置信息写入。通过 GT9147_ID_SET 函数将触摸芯片的器件地址为 0x28，配置信息寄存器的起始地址为 0x8047，需要写入的数据的长度为 186。函数实现的代码如下所示：

```
void GT9147_SendConfigMSG(uint8_t cmd) {
  uint8_t i;
  GT9147_ConfigMSGTBL[184] = 0;
  for (i = 0; i < 184; i++) {
    //计算校验和(0x8047 到 0x80FE 之间字节和)
    GT9147_ConfigMSGTBL[184] += GT9147_ConfigMSGTBL;
  }
  //配置信息校验(0x8047 到 0x80FE 之间字节和的补码)
  GT9147_ConfigMSGTBL[184] = ~GT9147_ConfigMSGTBL[184] + 1;
  //配置已更新标记，写入 0x01 则保存更新配置
  GT9147_ConfigMSGTBL[185] = cmd;
  //写入配置信息（0x8047 到 0x8100）
  GT9147_WrNByte(GT9147_DevAdr0, GT9147_ConfigMsgReg, GT9147_ConfigMSGTBL,186);
}
```

8.7.1.4. GT9147_Scan

扫描触摸屏，得到触摸点相关数据，其实现步骤如下所示：

(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：

```
const uint16_t GT9147_TPR_TBL[5] = {
  GT9147_TouchPoint1Reg,
  GT9147_TouchPoint2Reg,
  GT9147_TouchPoint3Reg,
  GT9147_TouchPoint4Reg,
  GT9147_TouchPoint5Reg
};
unsigned int GT9147_Scan(void) {
  uint8_t i, sta = 0, dat[4] = { 0 };
  static int touch_value;
  GT9147_RdNByte(GT9147_DevAdr0, GT9147_TouchStateReg, &sta, 1); //读取当前触摸情况
  if (sta & 0x80){ //坐标数据有效，0x814E 寄存器的最高位为 1 表示坐标已准备好
    //此位使用完后应清零，若未清零，下次将不能读取坐标数据
    if (sta & 0x0F){ //判断几个触摸点按下，0x814E 寄存器的低 4 位表示有效触点个数
       //~(0xFF << (sta & 0x0F))将点的个数转换为触摸点按下有效标志
      TPR_Structure.TouchSta = ~(0xFF << (sta & 0x0F));
      for (i = 0; i < 5; i++) {
        if (TPR_Structure.TouchSta & (1 << i)){ //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
          GT9147_RdNByte(GT9147_DevAdr0, GT9147_TPR_TBL[i], &dat, 4);
        }
      }
    }
    sta = 0; //使用完后清空 0x814E 寄存器所有标志
    GT9147_WrNByte(GT9147_DevAdr0, GT9147_TouchStateReg, &sta, 1);
    touch_value = (dat[1] << 24) | (dat[0] << 16) | (dat[3] << 8) | dat[2];
    return touch_value;
  } else //清除所有触摸点按下有效标志
  {
    TPR_Structure.TouchSta &= 0xE0; //低 5 位置 0
    return touch_value;
  }
}
```

8.7.2. 添加用户代码

实验功能：通过在 TFT LCD 屏上绘制画板界面，并且通过触摸的方式实现清屏和画笔颜色的选择。

实验代码：初始化 I2C、触摸芯片和 LCD 屏，将背景设置为白色，默认画笔颜色为黑色，初始化画板界面，获取触摸置，根据触摸坐标的位置实现清除、画笔颜色的选取和绘画的功能，主函数中的代码如下所示：

```
int main()
```

```
{
    uint32_t touch_value = 0;//定义变量存储坐标的参数
    uint16_t x,y; //第一次得到的坐标值
    Touch_IIC_Init();//I2C 初始化
    GT9147_Init();//GT9147 初始化
    TFTLCD_Init();//初始化 LCD 屏
    LCD_Clear(LCD_WHITE);
    delay(13000);
    POINT_COLOR=LCD_BLACK;
    BACK_COLOR=LCD_WHITE;
    Set_Display();//界面初始化
    while(1){
        touch_value = GT9147_Scan();//得到触摸值
        x = 800 - (touch_value & 0xffff);
        y = touch_value >> 16;
        Draw_color(x,y);//画笔颜色选择函数
        if((x > 20 && x < 100) && (y < 460 && y > 430)){//清除绘画区域
            LCD_Fill(121,21,779,459,LCD_WHITE);
        }
        if((x > 125 && x < 775) && (y < 455 && y > 25)){//只在绘画区域内进行绘画
            LCD_Draw_ALLCircle(x,y,DrawLine);//连续画实心圆形成线，半径可改变线的粗细
        }
    }
}
```

需要注意的是，触摸芯片返回的坐标与 TFT LCD 屏上的坐标是不一致的，其对应关系如下图 8. 14 所示。

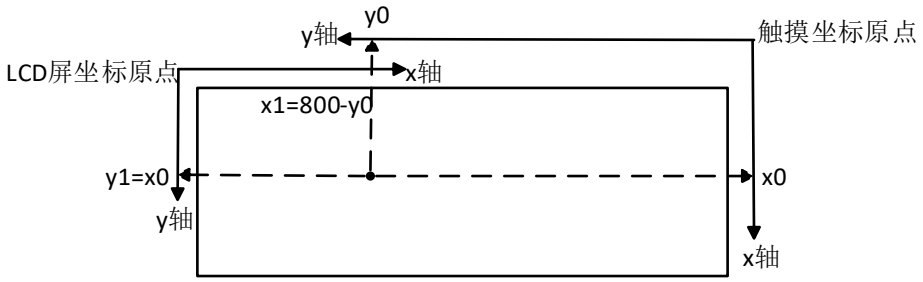

图 8. 14 触摸坐标与屏幕坐标对应关系图

关于界面绘制的函数如下所示，主要就是通过调用 TFT LCD 库中的画矩形、填充区域颜色和显示字符串这几个函数实现的。

```
//显示界面配置
void Set_Display()
{
    //定义画画的区域
    LCD_DrawRectangle(120,20,780,460);
```

```
    //定义清除按钮
    LCD_DrawRectangle(20,430,100,460);
    LCD_ShowString(24, 432, 80, 20, 24, 1, "Clean");
    //画按钮界面
    Draw_Button(BT_BLACK);
    Draw_Button(BT_BLUE);
    Draw_Button(BT_RED);
    Draw_Button(BT_YELLOW);
    Draw_Button(BT_GREEN);
    Draw_Button(BT_WHITE);
    Draw_Button(BT_GRAY);
    //填充对应按键的颜色
    Fill_Button(BT_BLACK,LCD_BLACK);
    Fill_Button(BT_BLUE,LCD_BLUE);
    Fill_Button(BT_RED,LCD_RED);
    Fill_Button(BT_YELLOW,LCD_YELLOW);
    Fill_Button(BT_GREEN,LCD_GREEN);
    Fill_Button(BT_WHITE,LCD_WHITE);
    Fill_Button(BT_GRAY,LCD_GRAY);
}
```

画笔颜色选择函数，主要是根据触摸的位置，选择对应的颜色，代码如下所示：

```
void Draw_color(u16 x,u16 y)
{
    //选择黑色
    if((x > 20 && x < 100) && (y < 60 && y > 20)){
        POINT_COLOR=LCD_BLACK;
    }
    //选择蓝色
    if((x > 20 && x < 100) && (y < 120 && y > 80)){
        POINT_COLOR=LCD_BLUE;
    }
    //选择红色
    if((x > 20 && x < 100) && (y < 180 && y > 140)){
        POINT_COLOR=LCD_RED;
    }
    //选择黄色
    if((x > 20 && x < 100) && (y < 240 && y > 200)){
        POINT_COLOR=LCD_YELLOW;
    }
    //选择绿色
    if((x > 20 && x < 100) && (y < 300 && y > 260)){
```

```
        POINT_COLOR=LCD_GREEN;
    }
    //选择白色
    if((x > 20 && x < 100) && (y < 360 && y > 320)){
        POINT_COLOR=LCD_WHITE;
    }
    //选择灰色
    if((x > 20 && x < 100) && (y < 420 && y > 380)){
        POINT_COLOR=LCD_GRAY;
    }
}
```

上述只是代码的一部分，完整的代码内容参看工程中的源码。

8.8. 板级验证

8.8.1. 实验所需硬件

(1) AC208 开发板
(2) FPGA 下载器：XIST USB Cable
(3) CM3 仿真器：DAP Link
(4) 电源线一根
(5) TFT LCD 屏一块

8.8.2. 硬件连接

JTAG 和 DAP Link 的硬件连接参考实验一。

在连接 TFT LCD 屏的时候，由于开发板显示接口有 2*18 共 36 个接口座，而 TFT LCD 屏只有 34 个接口，所以在连接的时候，请保持靠下插接的方式，空出主板上显示扩展接口的 1、2 两个脚，即对应的开发板后面显示的 GND 和 3V3 不要接。需要空出的接口如下图 8. 15 所示。

图 8. 15 TFT LCD 屏连接时需要空出的引脚

TFT LCD 屏正确的连接方式如下图 8. 16 所示。

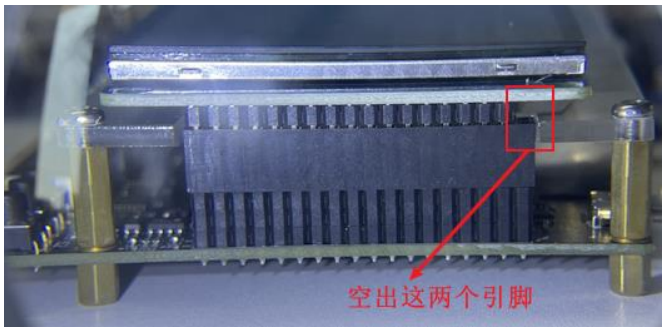

图 8. 16 TFT LCD 显示屏正确连接示意图

8.8.3. 下载文件至目标板

下载文件的方式参考实验一。

8.8.4. 功能演示

将程序下载之后，屏幕显示绘制效果示意如下所示。

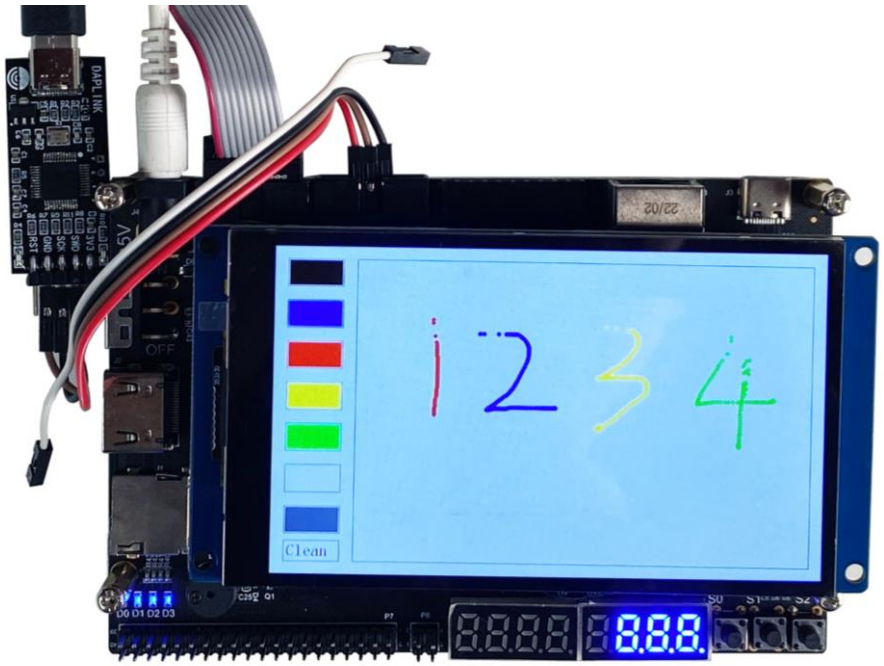

图 8. 17 画笔颜色选取绘制效果示意图

8.9. 思考与总结

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，通用显示扩展接口的 1 脚和 2 脚不要接，

也就是开发板后面的 GND 和 3V3 这两个脚不要接。

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。