AN1508 ATK-7’TFTLCD V2 电容触摸屏模块使用说明

本应用文档（AN508，对应对应**战舰 V3 / 精英 STM32F103 开发板的 触摸屏实验**）将教大家如何在**战舰 V3 / 精英 STM32F103 开发板**上使用 ATK-7’ TFTLCD V2 电容触摸屏模块。

对于 ALIENTEK MiniSTM32F103 和探索者 STM32F407 开发板，同样也可以参考本文档进行学习，大同小异。

本文档分为如下几部分：

1， ATK-7’ TFTLCD V2 电容触摸屏模块简介
2， 硬件连接
3， 软件实现
4， 验证

1、ATK-7’ TFTLCD V2 电容触摸屏模块简介

ATK-7’ TFTLCD V2 模块是 ALIENTEK 推出的一款高性能 7 寸电容触摸屏模块。该模块屏幕分辨率为 800*480，16 位真彩显示，采用 NT35510 驱动，该芯片直接自带 GRAM，无需外加驱动器，因而任何单片机，都可以轻易驱动。

模块采用电容触摸屏，最多支持 5 点同时触摸，具有非常好的操控效果。模块硬件接口与 ALIENTEK 其他液晶模块(2.4’/2.8’/3.5’/7’等)接口完全一致，因而原有产品，硬件上不需要任何变动，只需要稍微修改一下软件，就可以使用我们的 ATK-7’ TFTLCD 电容触摸屏模块。

1.1 模块引脚说明

ATK-7’ TFTLCD V2 电容触摸屏模块通过 2*17 的排针（2.54mm 间距）同外部连接，模块可以与 ALIENTEK 的 STM32 开发板直接对接，我们提供相应的例程，用户可以在 ALIENTEK STM32 开发板上直接测试。ATK-7’ TFTLCD V2 电容触摸屏模块外观如图 1.1.1 所示：

图 1.1.1-1 ATK-7’ TFTLCD V2 电容触摸屏模块正面图

图 1.1.1-2 ATK-7’ TFTLCD V2 电容触摸屏模块背面图

ATK-7’ TFTLCD V2 电容触摸屏模块通过 34（2*17）个引脚同外部连接，对外接口原理图如图 1.1.2 所示:

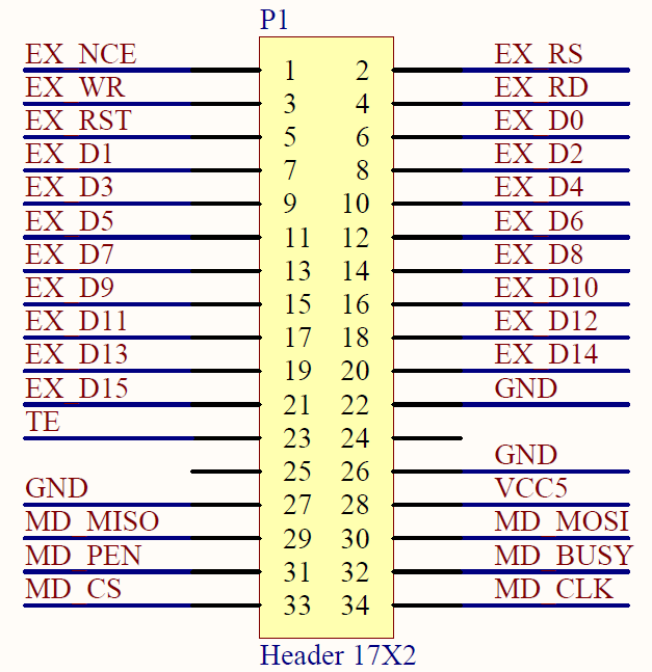

图 1.1.2 模块对外接口原理图

对应引脚功能详细描述如表 1.1.1 所示：

序号	名称	说明
1	EX_NCE	LCD 控制器片选信号（低电平有效）
2	EX_RS	命令/数据控制信号（0，命令；1，数据；）
3	EX_WR	写使能信号（低电平有效）
4	EX_RD	读使能信号（低电平有效）
5	EX_RST	复位信号（低电平有效）

6~21	EX_D0~D15	双向数据总线
22,26,27	GND	地线
23	TE	撕裂效应信号
24,25	NC	未用到
28	VCC5	5V 电源输入引脚
29	MD_MISO	NC，电容触摸屏未用到
30	MD_MOSI	电容触摸屏 IIC_SDA 信号(CT_SDA)
31	MD_PEN	电容触摸屏中断信号(CT_INT)
32	MD_BUSY	NC，电容触摸屏未用到
33	MD_CS	电容触摸屏复位信号(CT_RST)
34	MD_CLK	电容触摸屏 IIC_SCL 信号(CT_SCL)

表 2.1.1 ATK-7’ TFTLCD V2 模块引脚说明

表 1.1.1 ATK-7’ TFTLCD V2 模块引脚说明

从上表可以看出，在 16 位模式下：LCD 控制器总共需要 21 个 IO 口驱动（不包括 TE 信号），电容触摸屏需要 4 个 IO 口驱动，这样整个模块需要 25 个 IO 口驱动。而在 8 位/9 位/12 位模式下，则可以少用一些 IO，不过驱动速度就会相应的下降。

TE 是撕裂效应信号，是 LCD 控制器反馈给单片机的信号，用于指示 LCD 控制器的显示状态。在非显示周期内，TE 信号为高。因此，本信号使单片机通过观察非显示周期发送数据，以避免撕裂。不过这个信号我们一般用不到，一般情况下可以不需要理会此信号。

特别注意：模块需要双电源供电：5V 和 3.3V，都必须接上，才可以正常工作，5V 电源用于背光供电，3.3V 用于除背光外的其他电源部分供电。

1.2 模块接口时序

ATK-7’ TFTLCD V2 模块采用 SSD1963 作为驱动器，支持 8 位/9 位/12 位/16 位 8080 总时线接口，总线写时序如图 1.2.1 所示：

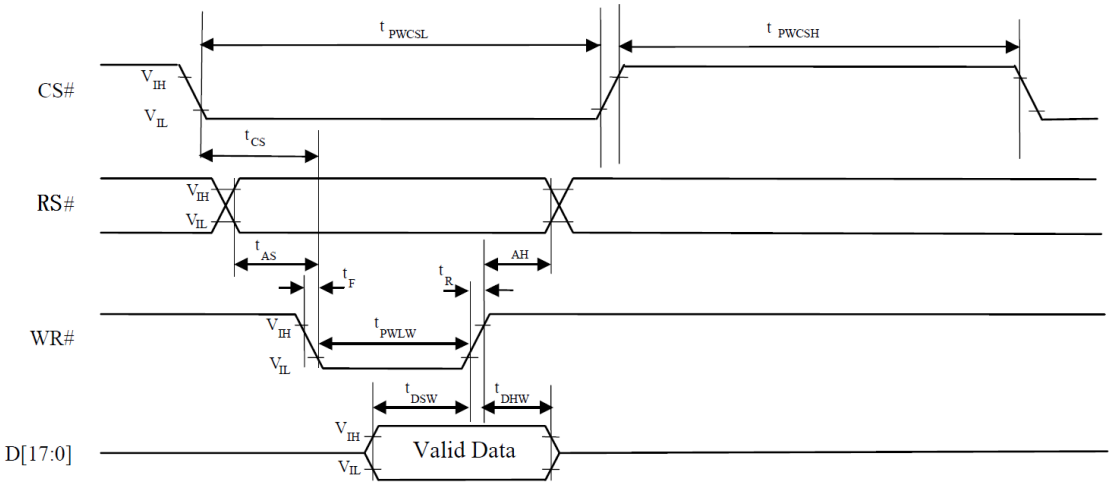

图 1.2.1 总线写时序

图中，当 RS 为 0 的时候，表示写入的是寄存器地址（0~7），RS 为 1 的时候，表示写入的是数据（寄存器值/GRAM 数据）。

总线读时序如图 1.2.2 所示：

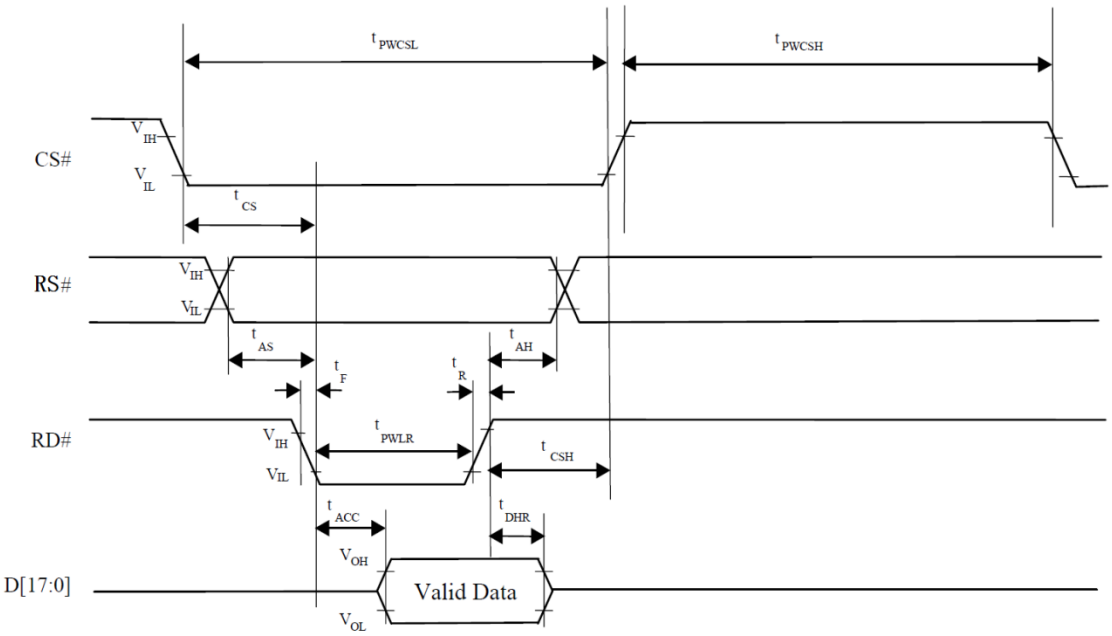

图 1.2.2 总线读时序

图 1.2.1 和图 1.2.2 中各时间参数见表 1.2.1 所示：

Symbol	Parameter		Min	Typ	Max	Unit
f_{MCLK}	System Clock Frequency*		1	-	110	MHz
t_{MCLK}	System Clock Period*		1/ f_{MCLK}	-	-	ns
t_{PWCSL}	Control Pulse High Width	Write Read	13 30	1.5* t_{MCLK} 3.5* t_{MCLK}	-	ns
t_{PWCSH}	Control Pulse Low Width	Write (next write cycle) Write (next read cycle) Read	13 80 80	1.5* t_{MCLK} 9* t_{MCLK} 9* t_{MCLK}	-	ns
t_{AS}	Address Setup Time		1	-	-	ns
t_{AH}	Address Hold Time		2	-	-	ns
t_{DSW}	Write Data Setup Time		4	-	-	ns
t_{DHW}	Write Data Hold Time		1	-	-	ns
t_{PWLW}	Write Low Time		12	-	-	ns
t_{DHR}	Read Data Hold Time		1	-	-	ns
t_{ACC}	Access Time		32	-	-	ns
t_{PWLR}	Read Low Time		36	-	-	ns
t_R	Rise Time		-	-	0.5	ns
t_F	Fall Time		-	-	0.5	ns
t_{CS}	Chip select setup time		2	-	-	ns
t_{CSH}	Chip select hold time to read signal		3	-	-	ns

* System Clock denotes external input clock (PLL-bypass) or internal generated clock (PLL-enabled)

表 2.3.1 SSD1963 参数表

一般我们设置 fMCLK 的频率为 100Mhz，因此写周期长度最小为 3 个 tMCLK，即写一个像素，最快需要 3 个 tMCLK，即：30ns，对于本模块，屏幕分辨率是 800*480，因此可以计算得出：频率的刷新率，最快是 86.8 帧。模块的读取速度相对较慢：读周期需要 12.5 个 tMCLK，即：125ns。

1.3 模块驱动说明

ATK-7’ TFTLCD V2 模块采用 SSD1963 作为 LCD 驱动芯片，该芯片自带 LCD GRAM，无需外加独立驱动器，并且，在指令上，基本兼容 ILI9341，使用非常方便。模块采用 16 位 8080 并口与外部连接（也支持 8 位/9 位/12 位/16 位并口模式，通过程序设置即可），在 8080 并口模式下，LCD 驱动需要用到的信号线如下：

EX_NCE：LCD 片选信号。

EX_WR：向 LCD 写入数据。

EX_RD：从 LCD 读取数据。

EX_D[15：0]：16 位双向数据线。

EX_RST：硬复位 LCD。

EX_RS：命令/数据标志（0，读写命令；1，读写数据）。

为了简单起见，这里我们并没有用到 TE 信号，该信号的详细介绍，请大家参考 SSD1963 数据的相关内容。

SSD1963 自带 LCD GRAM（480*864*3 字节），并且最高支持 24 位颜色深度（1600 万色），我们的模块引出 16 位数据线，通过设置（指令：0XF0），可以支持 8 位/9 位/12 位/16 位总线宽度，可以支持 16 位/18 位/24 位颜色深度。总线宽度与颜色深度对应表，如表 1.3.1 所示：

Interface	Cycle	D[15]	D[14]	D[13]	D[12]	D[11]	D[10]	D[9]	D[8]	D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
16 bits (565 format)	1st	R5	R4	R3	R2	R1	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1
16 bits	1st	R7	R6	R5	R4	R3	R2	R1	R0	G7	G6	G5	G4	G3	G2	G1	G0
	2nd	B7	B6	B5	B4	B3	B2	B1	B0	R7	R6	R5	R4	R3	R2	R1	R0
	3rd	G7	G6	G5	G4	G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0
12 bits	1st					R7	R6	R5	R4	R3	R2	R1	R0	G7	G6	G5	G4
	2nd					G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0
9 bits	1st								R5	R4	R3	R2	R1	R0	G5	G4	G3
	2nd								G2	G1	G0	B5	B4	B3	B2	B1	B0
8 bits	1st									R7	R6	R5	R4	R3	R2	R1	R0
	2nd									G7	G6	G5	G4	G3	G2	G1	G0
	3rd									B7	B6	B5	B4	B3	B2	B1	B0

表 1.3.1 总线宽度与像素颜色深度对应表

从上表可以看出，除了 16 位（RGB565）格式，一个点的颜色值可以在一个周期内完成，其他都需要 2~3 个周期才可以完成。虽然 18 位/24 位颜色深度可以得到更好的颜色还原效果，不过代价就是显示速度变慢。所以，我们一般使用 16 位颜色深度（65K 色），RGB565 格式，这样，可以达到最快的显示速度。

特别注意 SSD1963 所有的指令都是 8 位的（高 8 位无效），且参数除了读写 GRAM 的时候可能是 8 位/9 位/12 位/16 位（根据 0XF0 的设置而定），其他操作参数，都是 8 位的。本例程，我们采用 16 位 RGB565 模式驱动，以得到最快的速度。

关于 SSD1963 的指令介绍，请大家参考：《ATK-7' TFTLCD V2 模块用户手册》这个 pdf 文档，里面有详细的介绍。

一般 TFTLCD 模块的使用流程如图 1.3.1：

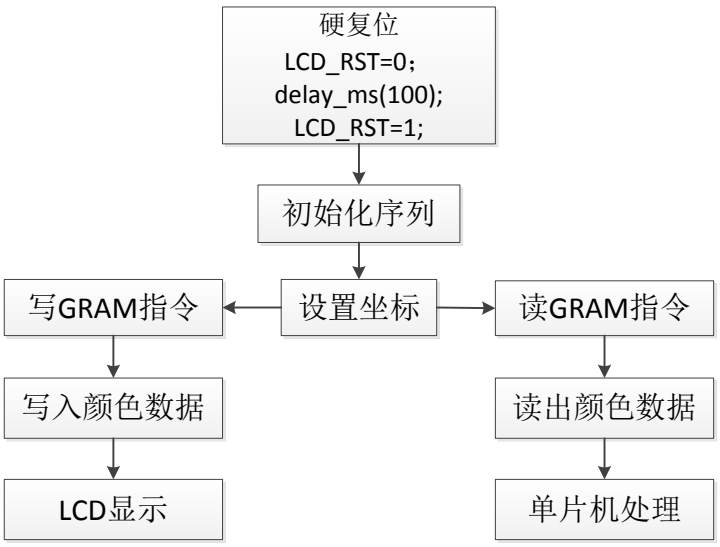

图 1.3.1 TFTLCD 使用流程

任何 LCD，使用流程都可以简单的用以上流程图表示。其中硬复位和初始化序列，只需要执行一次即可。而画点流程就是：设置坐标→写 GRAM 指令→写入颜色数据，然后在 LCD 上面，我们就可以看到对应的点显示我们写入的颜色了。读点流程为：设置坐标→读 GRAM 指令→读取颜色数据，这样就可以获取到对应点的颜色数据了。

以上只是最简单的操作，也是最常用的操作，有了这些操作，一般就可以正常使用 TFTLCD 了。接下来，我们看看要利用 STM32 驱动模块并显示字符/数字的操作步骤：通过以上介绍，我们可以得出 TFTLCD 显示字符/数字需要的相关设置步骤如下：

1）设置 STM32 与 TFTLCD 模块相连接的 IO。

这一步，先将我们与 TFTLCD 模块相连的 IO 口进行初始化，以便驱动 LCD。这里需要根据连接电路以及 TFTLCD 模块的设置来确定。

2）初始化 TFTLCD 模块。

即图 1.3.1 的初始化序列，这里我们例程里面没有硬复位 LCD 的操作，因我们 STM32 开发板的 LCD 接口，将 TFTLCD 的 RST 同 STM32 的 RESET 连接在一起了，只要按下开发板的 RESET 键，就会对 LCD 进行硬复位，所以这步直接由 MCU 的硬复位替代了。

初始化序列，这部分代码就是设置 SSD1963 相关参数，结合 TFTLCD 液晶屏的参数，对 SSD1963 进行设置，具体的，请看源代码。在初始化之后，LCD 模块就可以正常使用了。

3）通过函数将字符和数字显示到 TFTLCD 模块上。

这一步则通过图 1.3.1 左侧的流程，即：设置坐标→写 GRAM 指令→写 GRAM 来实现，但是这个步骤，只是一个点的处理，我们要显示字符/数字，就必须要多次使用这个步骤，从而达到显示字符/数字的目标，所以需要设计一个函数来实现数字/字符的显示，之后调用该函数，就可以实现数字/字符的显示了。

1.4 电容触摸屏接口说明

1.4.1 电容式触摸屏简介

现在几乎所有智能手机，包括平板电脑都是采用电容屏作为触摸屏，电容屏是利用人体感应进行触点检测控制，不需要直接接触或只需要轻微接触，通过检测感应电流来定位触摸坐标。

ATK-7’ TFTLCD V2 模块自带的触摸屏采用的是电容式触摸屏，下面简单介绍下电容式触

摸屏的原理。

电容式触摸屏主要分为两种：

1、 表面电容式电容触摸屏。

表面电容式触摸屏技术是利用 ITO(铟锡氧化物，是一种透明的导电材料)导电膜，通过电场感应方式感测屏幕表面的触摸行为进行。但是表面电容式触摸屏有一些局限性，它只能识别一个手指或者一次触摸。

2、 投射式电容触摸屏。

投射电容式触摸屏是传感器利用触摸屏电极发射出静电场线。一般用于投射电容传感技术的电容类型有两种：自我电容和交互电容。

自我电容又称绝对电容，是最广为采用的一种方法，自我电容通常是指扫描电极与地构成的电容。在玻璃表面有用 ITO 制成的横向与纵向的扫描电极，这些电极和地之间就构成一个电容的两极。当用手或触摸笔触摸的时候就会并联一个电容到电路中去，从而使在该条扫描线上的总体的电容量有所改变。在扫描的时候，控制 IC 依次扫描纵向和横向电极，并根据扫描前后的电容变化来确定触摸点坐标位置。笔记本电脑触摸输入板就是采用的这种方式，笔记本电脑的输入板采用 X*Y 的传感电极阵列形成一个传感格子，当手指靠近触摸输入板时，在手指和传感电极之间产生一个小量电荷。采用特定的运算法则处理来自行、列传感器的信号来确定手指的位置。

交互电容又叫做跨越电容，它是在玻璃表面的横向和纵向的 ITO 电极的交叉处形成电容。交互电容的扫描方式就是扫描每个交叉处的电容变化，来判定触摸点的位置。当触摸的时候就会影响到相邻电极的耦合，从而改变交叉处的电容量，交互电容的扫面方法可以侦测到每个交叉点的电容值和触摸后电容变化，因而它需要的扫描时间与自我电容的扫描方式相比要长一些，需要扫描检测 X*Y 根电极。目前智能手机/平板电脑等的触摸屏，都是采用交互电容技术。

ALIENTEK 所选择的电容触摸屏，也是采用的是投射式电容屏（交互电容类型），所以后面仅以投射式电容屏作为介绍。

透射式电容触摸屏采用纵横两列电极组成感应矩阵，来感应触摸。以两个交叉的电极矩阵，即： X 轴电极和 Y 轴电极，来检测每一格感应单元的电容变化，如图 1.4.1.1 所示：

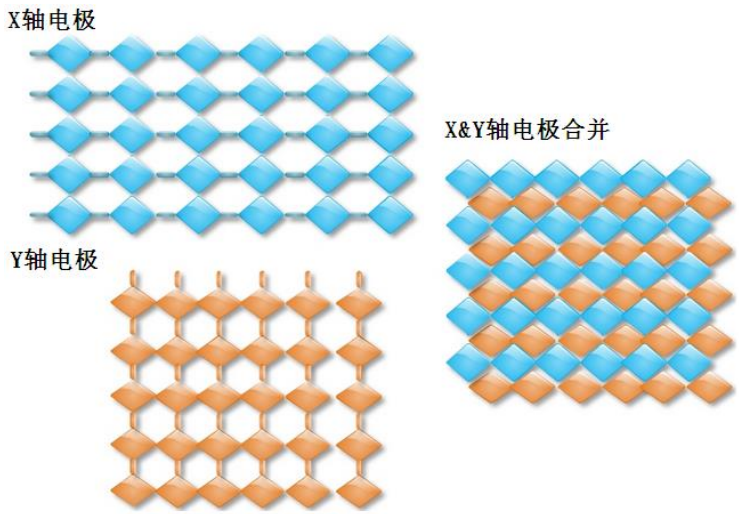

图 1.4.1.1 投射式电容屏电极矩阵示意图

示意图中的电极，实际是透明的，这里是为了方便大家理解。图中，X、Y 轴的透明电极电容屏的精度、分辨率与 X、Y 轴的通道数有关，通道数越多，精度越高。以上就是电容触摸屏的基本原理，接下来看看电容触摸屏的优缺点：

电容触摸屏的优点：手感好、无需校准、支持多点触摸、透光性好。

电容触摸屏的缺点：成本高、精度不高、抗干扰能力较差。

这里特别提醒大家电容触摸屏对工作环境的要求是比较高的，在潮湿、多尘、高低温环境下面，都是不适合使用电容屏的。

电容触摸屏一般都需要一个驱动 IC 来检测电容触摸，且一般是通过 IIC 接口输出触摸数据的。ATK-7’ TFTLCD V2 模块的电容触摸屏，采用的是 15*10 的驱动结构（10 个感应通道，15 个驱动通道），采用的是 FT5206 做为驱动 IC。

1.4.2 FT5206 简介

FT5206 是敦泰电子（FocalTech）生产的一颗电容触摸屏驱动 IC，最多支持 448 个通道。支持 SPI/IIC 接口，在 ATK-7’ TFTLCD V2 电容触摸屏上，FT5206 只用了 150 个通道，采用 IIC 接口。IIC 接口模式下，该驱动 IC 与 STM32 的连接仅需要 4 根线：SDA、SCL、RST 和 INT，SDA 和 SCL 是 IIC 通信用的，RST 是复位脚（低电平有效），INT 是中断输出信号，关于 IIC 我们就不详细介绍了，请参考开发板 IIC 实验。

FT5206 的器件地址为 0X38（不含最低位，换算成读写命令则是读：0X70，写：0X71），接下来，介绍一下 FT5206 的几个重要的寄存器。

1， 工作模式寄存器(0X00)

该寄存器用于设置 FT5206 的工作模式。该寄存器各位描述如表 1.4.2.1 所示：

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0X00	0	MODE[2:0]			0	0	0	0

表 1.4.2.1 0X00 寄存器各位描述

MODE[2：0]用于控制 FT5206 的工作模式，一般设置为：000b，表示正常工作模式。

2， 中断状态控制寄存器(0XA4)

该寄存器用于设置 FT5206 的中断状态。该寄存器各位描述如表 1.4.2.2 所示：

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0XA4	0	0	0	0	0	0	0	M

表 1.4.2.2 0XA4 寄存器各位描述

该寄存器只有最低位有效，M=0 的时候，表示查询模式；M=1 的时候，表示触发模式。一般设置为查询模式。

3， 有效触摸门限控制寄存器(0X80)

该寄存器用于设置 FT5206 的有效触摸门限值。该寄存器各位描述如表 1.4.2.3 所示：

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0X80	T7	T7	T5	T4	T3	T2	T1	T0

表 1.4.2.3 0X80 寄存器各位描述

该寄存器 8 位数据都有效，用于设置 FT5206 有效触摸的门限值，计算公式为：

有效触摸门限值=T[7:0]*4

T[7:0]所设置的值越小，触摸越灵敏，默认状态下 T[7:0]=70。

4， 激活周期控制寄存器(0X88)

该寄存器用于设置 FT5206 的激活周期。该寄存器各位描述如表 1.4.2.4 所示：

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0X88	0	0	0	0	P3	P2	P1	P0

表 1.4.2.4 0X88 寄存器各位描述

该寄存器只有低 4 位有效，用于设置 FT5206 的激活周期。P[3:0]的设置范围为：3~14，不过建议一般不要小于 12。

5， 库版本寄存器(0XA1 和 0XA2)

这里由 2 个寄存器：0XA1 和 0XA2 组成，用于读取 FT5206 的驱动库版本，0XA1 用于读取版本的高字节，0XA2 用于读取版本的低字节。ATK-7’TFTLCD V2 模块所用的 FT5206 库版本为：0X3003。

6， 触摸状态寄存器(0X02)

该寄存器用于读取 FT5206 的触摸状态。该寄存器各位描述如表 1.4.2.5 所示：

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0X02	0	0	0	0	TD3	TD2	TD1	TD0

表 1.4.2.5 0X02 寄存器各位描述

该寄存器只有低 4 位有效，TD[3:0]的取值范围是：1~5，表示有多少个有效触摸点。我们可以根据这个寄存器的值来判断有效触摸点的个数，然后通过 0X03/0X09/0X0F/0X15 和 0X1B 等寄存器来读取触摸坐标数据。

7， 触摸数据寄存器(0X03~0X1E)

这里总共包括 20 个寄存器，他们是：0X03~0X06、0X09~0X0C、0X0F~0X12、0X15~0X18、0X1B~0X1E。每 4 个寄存器为 1 组，表示一个触摸点的坐标数据，比如 0X03~0X06，则表示触摸点 1 的坐标数据，其他的以此类推。这里，我们仅介绍 0X03~0X06 寄存器，如表 1.4.2.6 所示：

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0X03	Event FLAG		0	0	X[11:8]			
0X04	X[7:0]							
0X05	Touch ID		0	0	Y[11:8]			
0X06	Y[7:0]							

表 1.4.2.6 0X03~0X06 寄存器各位描述

这里的 Event FLAG 用于表示触摸状态：00，按下；01，松开；10，持续触摸；11，保留。一般我们只需要判断该状态为 10 即可，即持续触摸状态，就可以稳定的读取触摸坐标数据了。而 Touch ID，我们一般用不到，这里就不做介绍了。最后，是 X 和 Y 的坐标数据，这些数据以 12 位的形式输出，如表 2.5.1.6 所示。

其他的 0X09~0X0C、0X0F~0X12、0X15~0X18、0X1B~0X1E 等寄存器，则分别用于读取第 2~5 个触摸点的坐标数据。

FT5206 的寄存器就介绍到这里，详细介绍，请参考：FTS_AN_CTPM_Standard.pdf 这个文档。

FT5206 只需要经过简单的初始化就可以正常使用了，初始化流程：硬复位→延时 20ms→结束硬复位→延时 50ms→设置工作模式、触摸有效阀值、激活周期等参数→完成初始化。此时 FT5206 即可正常使用了。

然后，我们不停的查询 0X02 寄存器，判断是否有有效触点，如果有，则读取坐标数据寄存器，得到触点坐标。

2、硬件连接

本章实验功能简介：开机的时候先初始化 LCD，读取 LCD ID，随后，根据 LCD ID 判断是电阻触摸屏还是电容触摸屏，如果是电阻触摸屏，则先读取 24C02 的数据判断触摸屏是否已经校准过，如果没有校准，则执行校准程序，校准过后再进入电阻触摸屏测试程序，如果已经校准了，就直接进入电阻触摸屏测试程序。

然后根据 LCD 的 ID 判断是电阻屏还是电容屏，如果是 SSD1963，则执行 FT5206 的电容触摸屏初始化代码，初始化电容触摸屏，随后进入电容触摸屏测试程序（电容触摸屏无需

校准！！）。

电阻触摸屏测试程序和电容触摸屏测试程序基本一样，只是电容触摸屏支持最多 5 点同时触摸，电阻触摸屏只支持一点触摸，其他一模一样。测试界面的右上角会有一个清空的操作区域（RST），点击这个地方就会将输入全部清除，恢复白板状态。使用电阻触摸屏的时候，可以通过按 KEY0 来实现强制触摸屏校准，只要按下 KEY0 就会进入强制校准程序。

所要用到的硬件资源如下：

1） 指示灯 DS0

2） KEY0 按键

3） ATK-7’ TFTLCD V2 模块（带电容式触摸屏）

4） 24C02

ATK-7’ TFTLCD V2 模块的接口同 ALIENTEK 的 2.4’/2.8’/3.5’/4.3’TFTLCD 模块接口一模一样，所以可以直接插在 ALIENTEK STM32 开发板上（还是靠右插哦！）。在硬件上，ATK-7’TFTLCD 模块与战舰 V3 / 精英 STM32F103 开发板的 IO 口对应关系如下：

EX_RST 对应开发板的复位引脚 RESET，通过开发板复位键复位 LCD 控制器;

EX_NCE 对应 PG12 即 FSMC_NE4;

EX_RS 对应 PG0 即 FSMC_A10;

EX _WR 对应 PD5 即 FSMC_NWE;

EX _RD 对应 PD4 即 FSMC_NOE;

EX _D[15:0]则直接连接在 FSMC_D15~FSMC_D0;

MD_MOSI(CT_SDA)连接 PF9;

MD_CLK(CT_SCL)连接 PB1;

MD_PEN(CT_INT)连接 PF10;

MD_CS(CT_RST)连接 PF11;

3、软件实现

本实验（注：这里仅以战舰 V3 代码为例进行介绍，精英板对应代码与之完全一样）对应战舰 V3 开发板的实验：实验 13 TFTLCD 显示实验。

3.1 LCD 驱动代码

本实验就是战舰 V3/精英 STM32 开发板的 触摸屏实验，由于代码量比较多，这里，我们仅挑一些重点进行介绍。

本实验，我们用到 FSMC 驱动 LCD，通过前面的介绍，我们知道 TFTLCD 的 RS 接在 FSMC 的 A10 上面，CS 接在 FSMC_NE4 上，并且是 16 位数据总线。即我们使用的是 FSMC 存储器 1 的第 4 区，我们定义如下 LCD 操作结构体（在 lcd.h 里面定义）：

```
//LCD 操作结构体
typedef struct
{
    vu16 LCD_REG;
    vu16 LCD_RAM;
} LCD_TypeDef;
//使用 NOR/SRAM 的 Bank1.sector4,地址位 HADDR[27,26]=11 A10 作为数据命令区分
```

```
//线，注意设置时 STM32 内部会右移一位对其!
#define LCD_BASE          ((u32)(0x6C000000 | 0x000007FE))
#define LCD               ((LCD_TypeDef *) LCD_BASE)
```

其中 LCD_BASE，必须根据我们外部电路的连接来确定，我们使用 Bank1.sector4 就是从地址 0X6C000000 开始，而 0X000007FE，则是 A10 的偏移量。我们将这个地址强制转换为 LCD_TypeDef 结构体地址，那么可以得到 LCD->LCD_REG 的地址就是 0X6C00,07FE，对应 A10 的状态为 0(即 RS=0)，而 LCD-> LCD_RAM 的地址就是 0X6C00,0800（结构体地址自增），对应 A10 的状态为 1（即 RS=1）。

所以，有了这个定义，当我们要往 LCD 写命令/数据的时候，可以这样写：

```
LCD->LCD_REG=CMD;   //写命令
LCD->LCD_RAM=DATA;  //写数据
```

而读的时候反过来操作就可以了，如下所示：

```
CMD= LCD->LCD_REG;   //读 LCD 寄存器
DATA = LCD->LCD_RAM; //读 LCD 数据
```

这其中，CS、WR、RD 和 IO 口方向都是由 FSMC 控制，不需要我们手动设置。我们再来看 lcd.h 里面的另一个重要结构体：

```
//LCD 重要参数集
typedef struct
{
    u16 width;          //LCD  宽度
    u16 height;         //LCD  高度
    u16 id;             //LCD ID
    u8  dir;            //横屏还是竖屏控制：0，竖屏；1，横屏。
    u16  wramcmd;       //开始写 gram 指令
    u16 setxcmd;        //设置 x 坐标指令
    u16 setycmd;        //设置 y 坐标指令
}_lcd_dev;
//LCD 参数
extern _lcd_dev lcddev; //管理 LCD 重要参数
```

该结构体用于保存一些 LCD 重要参数信息，比如 LCD 的长宽、LCD ID（驱动 IC 型号）、LCD 横竖屏状态等，这个结构体虽然占用了 14 个字节的内存，但是却可以让我们的驱动函数支持不同尺寸的 LCD，同时可以实现 LCD 横竖屏切换等重要功能，所以还是利大于弊的。有了以上了解，下面我们介绍 ILI93xx.c 里面的一些重要函数。

先看 7 个简单，但是很重要的函数：

```
//写寄存器函数
//regval:寄存器值
void LCD_WR_REG(u16 regval)
{
    LCD->LCD_REG=regval;   //写入要写的寄存器序号
```

```
}
//写 LCD 数据
//data:要写入的值
void LCD_WR_DATA(u16 data)
{
    LCD->LCD_RAM=data;
}
//读 LCD 数据
//返回值:读到的值
u16 LCD_RD_DATA(void)
{
    vu16 ram;                   //防止被优化
    ram=LCD->LCD_RAM;
    return ram;
}
//写寄存器
//LCD_Reg:寄存器地址
//LCD_RegValue:要写入的数据
void LCD_WriteReg(u16 LCD_Reg, u16 LCD_RegValue)
{
    LCD->LCD_REG = LCD_Reg;         //写入要写的寄存器序号
    LCD->LCD_RAM = LCD_RegValue;  //写入数据
}
//读寄存器
//LCD_Reg:寄存器地址
//返回值:读到的数据
u16 LCD_ReadReg(u16 LCD_Reg)
{
    LCD_WR_REG(LCD_Reg);            //写入要读的寄存器序号
    delay_us(5);
    return LCD_RD_DATA();           //返回读到的值
}
//开始写 GRAM
void LCD_WriteRAM_Prepare(void)
{
    LCD->LCD_REG=lcddev.wramcmd;
}
//LCD 写 GRAM
//RGB_Code:颜色值
void LCD_WriteRAM(u16 RGB_Code)
{
    LCD->LCD_RAM = RGB_Code;//写十六位 GRAM
}
```

因为 FSMC 自动控制了 WR/RD/CS 等这些信号，所以这 7 个函数实现起来都非常简单，

我们就不多说。这些函数实现功能见函数前面的备注，通过这几个简单函数的组合，我们就可以对 LCD 进行各种操作了。

第八个要介绍的函数是坐标设置函数，该函数代码如下：

```
//设置光标位置
//Xpos:横坐标
//Ypos:纵坐标
void LCD_SetCursor(u16 Xpos, u16 Ypos)
{
    if(lcddev.id==0X9341||lcddev.id==0X5310)
    {
        LCD_WR_REG(lcddev.setxcmd);
        LCD_WR_DATA(Xpos>>8);LCD_WR_DATA(Xpos&0XFF);
        LCD_WR_REG(lcddev.setycmd);
        LCD_WR_DATA(Ypos>>8);LCD_WR_DATA(Ypos&0XFF);
    }else if(lcddev.id==0X6804)
    {
        if(lcddev.dir==1)Xpos=lcddev.width-1-Xpos;//横屏时处理
        LCD_WR_REG(lcddev.setxcmd);
        LCD_WR_DATA(Xpos>>8);LCD_WR_DATA(Xpos&0XFF);
        LCD_WR_REG(lcddev.setycmd);
        LCD_WR_DATA(Ypos>>8);LCD_WR_DATA(Ypos&0XFF);
    }else if(lcddev.id==0X1963)
    {
        if(lcddev.dir==0)//x 坐标需要变换
        {
            Xpos=lcddev.width-1-Xpos;
            LCD_WR_REG(lcddev.setxcmd);
            LCD_WR_DATA(0);LCD_WR_DATA(0);
            LCD_WR_DATA(Xpos>>8);LCD_WR_DATA(Xpos&0XFF);
        }else
        {
            LCD_WR_REG(lcddev.setxcmd);
            LCD_WR_DATA(Xpos>>8);LCD_WR_DATA(Xpos&0XFF);
            LCD_WR_DATA((lcddev.width-1)>>8);
            LCD_WR_DATA((lcddev.width-1)&0XFF);
        }
        LCD_WR_REG(lcddev.setycmd);
        LCD_WR_DATA(Ypos>>8);LCD_WR_DATA(Ypos&0XFF);
        LCD_WR_DATA((lcddev.height-1)>>8);
        LCD_WR_DATA((lcddev.height-1)&0XFF);
    }else if(lcddev.id==0X5510)
    {
        LCD_WR_REG(lcddev.setxcmd);LCD_WR_DATA(Xpos>>8);
        LCD_WR_REG(lcddev.setxcmd+1);LCD_WR_DATA(Xpos&0XFF);
```

```
            LCD_WR_REG(lcddev.setycmd);LCD_WR_DATA(Ypos>>8);
            LCD_WR_REG(lcddev.setycmd+1);LCD_WR_DATA(Ypos&0XFF);

        }else
        {
            if(lcddev.dir==1)Xpos=lcddev.width-1-Xpos;//横屏其实就是调转 x,y 坐标
            LCD_WriteReg(lcddev.setxcmd, Xpos);
            LCD_WriteReg(lcddev.setycmd, Ypos);
        }
    }
```

该函数非常重要，该函数实现了将 LCD 的当前操作点设置到指定坐标(x,y)，有了该函数，我们就可以在液晶上任意作图了。这里面的 lcddev.setxcmd、lcddev.setycmd、lcddev.width、lcddev.height 等指令/参数都是在 LCD_Display_Dir 函数里面初始化的，该函数根据 lcddev.id 的不同，执行不同的设置，由于篇幅所限，这里就不列出来介绍了，请大家参考本例程源码。另外，因为 9341/5310/6804/1963/5510 等的设置同其他屏有些不太一样，所以进行了区别对待。

接下来我们介绍第九个函数：画点函数。该函数实现代码如下：

```
//画点
//x,y:坐标
//POINT_COLOR:此点的颜色
void LCD_DrawPoint(u16 x,u16 y)
{
    LCD_SetCursor(x,y);          //设置光标位置
    LCD_WriteRAM_Prepare(); //开始写入 GRAM
    LCD->LCD_RAM=POINT_COLOR;
}
```

该函数实现比较简单，就是先设置坐标，然后往坐标写颜色。其中 POINT_COLOR 是我们定义的一个全局变量，用于存放画笔颜色，顺带介绍一下另外一个全局变量：BACK_COLOR，该变量代表 LCD 的背景色。LCD_DrawPoint 函数虽然简单，但是至关重要，其他几乎所有上层函数，都是通过调用这个函数实现的。

有了画点，当然还需要有读点的函数，第十个介绍的函数就是读点函数，用于读取 LCD 的 GRAM，这里说明一下，为什么 OLED 模块没做读 GRAM 的函数，而这里做了。因为 OLED 模块是单色的，所需要全部 GRAM 也就 1K 个字节，而 TFTLCD 模块为彩色的，点数也比 OLED 模块多很多，以 16 位色计算，一款 320×240 的液晶，需要 320×240×2 个字节来存储颜色值，也就是也需要 150K 字节，这对任何一款单片机来说，都不是一个小数目了。而且我们在图形叠加的时候，可以先读回原来的值，然后写入新的值，在完成叠加后，我们又恢复原来的值。这样在做一些简单菜单的时候，是很有用的。这里我们读取 TFTLCD 模块数据的函数为 LCD_ReadPoint，该函数直接返回读到的 GRAM 值。该函数使用之前要先设置读取的 GRAM 地址，通过 LCD_SetCursor 函数来实现。LCD_ReadPoint 的代码如下：

```
//读取个某点的颜色值
//x,y:坐标
//返回值:此点的颜色
u16 LCD_ReadPoint(u16 x,u16 y)
```

```
{
    u16 r=0,g=0,b=0;
    if(x>=lcddev.width||y>=lcddev.height)return 0;    //超过了范围,直接返回
    LCD_SetCursor(x,y);
    if(lcddev.id==0X9341||lcddev.id==0X6804||lcddev.id==0X5310||lcddev.id==0X1963)
    LCD_WR_REG(0X2E);//9341/6804/3510/1963 发送读 GRAM 指令
    else if(lcddev.id==0X5510)LCD_WR_REG(0X2E00);//5510 发送读 GRAM 指令
    else LCD_WR_REG(0X22);                          //其他 IC 发送读 GRAM 指令
    if(lcddev.id==0X9320)opt_delay(2);              //FOR 9320,延时 2us
    r=LCD_RD_DATA();                                //dummy Read
    if(lcddev.id==0X1963)return r;                  //1963 直接读就可以
    opt_delay(2);
    r=LCD_RD_DATA();                                //实际坐标颜色
    if(lcddev.id==0X9341||lcddev.id==0X5310||lcddev.id==0X5510)//要分 2 次读出
    {
        opt_delay(2);
        b=LCD_RD_DATA();
        g=r&0XFF;//对于 9341/5310/5510,第一次读的是 RG 值,R 在前,G 在后,各占 8 位
        g<<=8;
    }
    if(lcddev.id==0X9325||lcddev.id==0X4535||lcddev.id==0X4531||lcddev.id==0XB505||
        lcddev.id==0XC505)return r; //这几种 IC 直接返回颜色值
    else if(lcddev.id==0X9341||lcddev.id==0X5310||lcddev.id==0X5510)
        return (((r>>11)<<11)|((g>>10)<<5)|(b>>11));
        //ILI9341/NT35310/NT35510 需要公式转换一下
    else return LCD_BGR2RGB(r);    //其他 IC
}
```

在 LCD_ReadPoint 函数中，因为我们的代码不止支持一种 LCD 驱动器，所以，我们根据不同的 LCD 驱动器（(lcddev.id）型号，执行不同的操作，以实现对各个驱动器兼容，提高函数的通用性。

第十一个要介绍的是字符显示函数 LCD_ShowChar，该函数同前面 OLED 模块的字符显示函数差不多，但是这里的字符显示函数多了 1 个功能，就是可以以叠加方式显示，或者以非叠加方式显示。叠加方式显示多用于在显示的图片上再显示字符。非叠加方式一般用于普通的显示。该函数实现代码如下：

```
//在指定位置显示一个字符
//x,y:起始坐标
//num:要显示的字符:" "--->"~"
//size:字体大小 12/16/24
//mode:叠加方式(1)还是非叠加方式(0)
void LCD_ShowChar(u16 x,u16 y,u8 num,u8 size,u8 mode)
{
    u8 temp,t1,t;
    u16 y0=y;
    u8 csize=(size/8+((size%8)?1:0))*(size/2);//得到一个字符对应点阵集所占的字节数
```

```
        num=num-' ';  //ASCII 字库从空格开始取模，所以-' '即可得到对应字符字库（点阵）
        for(t=0;t<csize;t++)
        {
            if(size==12)temp=asc2_1206[num][t];         //调用 1206 字体
            else if(size==16)temp=asc2_1608[num][t];  //调用 1608 字体
            else if(size==24)temp=asc2_2412[num][t];  //调用 2412 字体
            else return;                                          //没有的字库
            for(t1=0;t1<8;t1++)
            {
                if(temp&0x80)LCD_Fast_DrawPoint(x,y,POINT_COLOR);
                else if(mode==0)LCD_Fast_DrawPoint(x,y,BACK_COLOR);
                temp<<=1;
                y++;
                if(y>=lcddev.height)return;          //超区域了
                if((y-y0)==size)
                {
                    y=y0; x++;
                    if(x>=lcddev.width)return;    //超区域了
                    break;
                }
            }
        }
    }
```

在 LCD_ShowChar 函数里面，我们采用快速画点函数 LCD_Fast_DrawPoint 来画点显示字符，该函数同 LCD_DrawPoint 一样，只是带了颜色参数，且减少了函数调用的时间，详见本例程源码。该代码中我们用到了三个字符集点阵数据数组 asc2_2412、asc2_1206 和 asc2_1608，这几个字符集的点阵数据的提取方式，参见开发板例程的 OLED 实验。

最后，我们再介绍一下 TFTLCD 模块的初始化函数 LCD_Init，该函数先初始化 STM32 与 TFTLCD 连接的 IO 口，并配置 FSMC 控制器，然后读取 LCD 控制器的型号，根据控制 IC 的型号执行不同的初始化代码，其简化代码如下：

```
//初始化 lcd
void LCD_Init(void)
{
    RCC->AHBENR|=1<<8;              //使能 FSMC 时钟
    RCC->APB2ENR|=1<<3;           //使能 PORTB 时钟
    RCC->APB2ENR|=1<<5;           //使能 PORTD 时钟
    RCC->APB2ENR|=1<<6;           //使能 PORTE 时钟
    RCC->APB2ENR|=1<<8;           //使能 PORTG 时钟
    GPIOB->CRL&=0XFFFFFFF0;   //PB0 推挽输出 背光
    GPIOB->CRL|=0X00000003;
    GPIOD->CRH&=0X00FFF000;   //PORTD 复用推挽输出
```

```
GPIOD->CRH|=0XBB000BBB;
GPIOD->CRL&=0XFF00FF00;
GPIOD->CRL|=0X00BB00BB;
//PORTE 复用推挽输出
GPIOE->CRH&=0X00000000;
GPIOE->CRH|=0XBBBBBBBB;
GPIOE->CRL&=0X0FFFFFFF;
GPIOE->CRL|=0XB0000000;
GPIOG->CRH&=0XFFF0FFFF;  //PORTG12 复用推挽输出
GPIOG->CRH|=0X000B0000;
GPIOG->CRL&=0XFFFFFFF0;  //PG0->RS
GPIOG->CRL|=0X0000000B;
//寄存器清零
//bank1 有 NE1~4,每一个有一个 BCR+TCR，所以总共八个寄存器。
//这里我们使用 NE4 ，也就对应 BTCR[6],[7]。
FSMC_Bank1->BTCR[6]=0X00000000;
FSMC_Bank1->BTCR[7]=0X00000000;
FSMC_Bank1E->BWTR[6]=0X00000000;
//操作 BCR 寄存器      使用异步模式
FSMC_Bank1->BTCR[6]|=1<<12;           //存储器写使能
FSMC_Bank1->BTCR[6]|=1<<14;           //读写使用不同的时序
FSMC_Bank1->BTCR[6]|=1<<4;            //存储器数据宽度为 16bit
//操作 BTR 寄存器
//读时序控制寄存器
FSMC_Bank1->BTCR[7]|=0<<28;           //模式 A
FSMC_Bank1->BTCR[7]|=1<<0;
//地址建立时间（ADDSET）为 2 个 HCLK 1/36M=27ns(实际>200ns)
//因为液晶驱动 IC 的读数据的时候，速度不能太快，尤其对 1289 这个 IC。
FSMC_Bank1->BTCR[7]|=0XF<<8;  //数据保存时间为 16 个 HCLK
//写时序控制寄存器
FSMC_Bank1E->BWTR[6]|=0<<28;  //模式 A
FSMC_Bank1E->BWTR[6]|=0<<0;   //地址建立时间（ADDSET）为 1 个 HCLK
//4 个 HCLK（72M）因液晶驱动 IC 的写信号脉宽，最少也得 50ns。72M/4=24M=55ns
FSMC_Bank1E->BWTR[6]|=3<<8;   //数据保存时间为 4 个 HCLK
//使能 BANK1,区域 4
FSMC_Bank1->BTCR[6]|=1<<0;       //使能 BANK1，区域 4
```

```
delay_ms(50);                       // delay 50 ms
lcddev.id=LCD_ReadReg(0x0000);      //读 ID（9320/9325/9328/4531/4535 等 IC）
if(lcddev.id<0XFF||lcddev.id==0XFFFF||lcddev.id==0X9300)//读到 ID 不正确,新增
//lcddev.id==0X9300 判断，因为 9341 在未被复位的情况下会被读成 9300
{
    //尝试 9341 ID 的读取
    LCD_WR_REG(0XD3);
    lcddev.id=LCD_RD_DATA();        //dummy read
    lcddev.id=LCD_RD_DATA();        //读到 0X00
    lcddev.id=LCD_RD_DATA();        //读取 93
    lcddev.id<<=8;
    lcddev.id|=LCD_RD_DATA();       //读取 41
    if(lcddev.id!=0X9341)           //非 9341,尝试是不是 6804
    {
        LCD_WR_REG(0XBF);
        lcddev.id=LCD_RD_DATA();    //dummy read
        lcddev.id=LCD_RD_DATA();    //读回 0X01
        lcddev.id=LCD_RD_DATA();    //读回 0XD0
        lcddev.id=LCD_RD_DATA();    //这里读回 0X68
        lcddev.id<<=8;
        lcddev.id|=LCD_RD_DATA();   //这里读回 0X04
        if(lcddev.id!=0X6804)       //也不是 6804,尝试看看是不是 NT35310
        {
            LCD_WR_REG(0XD4);
            lcddev.id=LCD_RD_DATA();//dummy read
            lcddev.id=LCD_RD_DATA();//读回 0X01
            lcddev.id=LCD_RD_DATA();//读回 0X53
            lcddev.id<<=8;
            lcddev.id|=LCD_RD_DATA();//这里读回 0X10
            if(lcddev.id!=0X5310)   //不是 NT35310,尝试看是不是 NT35510
            {
                LCD_WR_REG(0XDA00);
                lcddev.id=LCD_RD_DATA();        //读回 0X00
                LCD_WR_REG(0XDB00);
                lcddev.id=LCD_RD_DATA();        //读回 0X80
                lcddev.id<<=8;
```

```
                        LCD_WR_REG(0XDC00);
                        lcddev.id|=LCD_RD_DATA();           //读回 0X00
                        if(lcddev.id==0x8000)lcddev.id=0x5510;//NT35510 读回的 ID 是
                                                //8000H,为方便区分,我们强制设置为 5510
                        if(lcddev.id!=0X5510)//不是 NT5510,尝试看看是不是 SSD1963
                        {
                            LCD_WR_REG(0XA1);
                            lcddev.id=LCD_RD_DATA();
                            lcddev.id=LCD_RD_DATA();        //读回 0X57
                            lcddev.id<<=8;
                            lcddev.id|=LCD_RD_DATA();       //读回 0X61
                            if(lcddev.id==0X5761)lcddev.id=0X1963;
                            //SSD1963 ID 是 5761H,为方便区分,我们强制设置为 1963
                        }
                    }
                }
            }
        }
        printf(" LCD ID:%x\r\n",lcddev.id);     //打印 LCD ID
        if(lcddev.id==0X9341)                   //9341 初始化
        {
            ……//9341 初始化代码
        } else if(lcddev.id==0X6804)            //6804 初始化
        {
            ……//6804 初始化代码
        } else if(lcddev.id==0X5310)            //5310 初始化
        {
            ……//5310 初始化代码
        } else if(lcddev.id==0X5510)            //5510 初始化
        {
            ……//5510 初始化代码
        }else if(lcddev.id==0x9325)//9325
        {
            ……//9325 初始化代码
        }else if(lcddev.id==0x9328)             //ILI9328     OK
        {
```

```
        ……//9328 初始化代码
    }else if(lcddev.id==0x9320||lcddev.id==0x9300)//未测试.
    {
        ……//9300 初始化代码
    }else if(lcddev.id==0X9331)
    {
        ……//9331 初始化代码
    }else if(lcddev.id==0x5408)
    {
        ……//5408 初始化代码
    }
    else if(lcddev.id==0x1505)//OK
    {
        ……//1505 初始化代码
    }else if(lcddev.id==0xB505)
    {
        ……//B505 初始化代码
    }else if(lcddev.id==0xC505)
    {
        ……//C505 初始化代码
    }else if(lcddev.id==0x8989)
    {
        ……//8989 初始化代码
    }else if(lcddev.id==0x4531)
    {
        ……//4531 初始化代码
    }else if(lcddev.id==0x4535)
    {
        ……//4535 初始化代码
    }else if(lcddev.id==0X1963)
    {
        LCD_WR_REG(0xE2); //Set PLL with OSC = 10MHz ,Multiplier N= 29,
                          //250MHz<VCO<800MHz=OSC*(N+1),VCO = 300MHz
        LCD_WR_DATA(0x1D);      //参数 1
        LCD_WR_DATA(0x02);      //参数 2 Divider M = 2, PLL = 300/(M+1) = 100MHz
        LCD_WR_DATA(0x04);      //参数 3 Validate M and N values
```

```
delay_us(100);
LCD_WR_REG(0xE0);       // Start PLL command
LCD_WR_DATA(0x01);      // enable PLL
delay_ms(10);
LCD_WR_REG(0xE0);       // Start PLL command again
LCD_WR_DATA(0x03);      // now, use PLL output as system clock
delay_ms(12);
LCD_WR_REG(0x01);       //软复位
delay_ms(10);
LCD_WR_REG(0xE6);       //设置像素频率,33Mhz
LCD_WR_DATA(0x2F);
LCD_WR_DATA(0xFF);
LCD_WR_DATA(0xFF);
LCD_WR_REG(0xB0);       //设置 LCD 模式
LCD_WR_DATA(0x20);      //24 位模式
LCD_WR_DATA(0x00);      //TFT  模式
LCD_WR_DATA((SSD_HOR_RESOLUTION-1)>>8);//设置 LCD 水平像素
LCD_WR_DATA(SSD_HOR_RESOLUTION-1);
LCD_WR_DATA((SSD_VER_RESOLUTION-1)>>8);//设置 LCD 垂直像素
LCD_WR_DATA(SSD_VER_RESOLUTION-1);
LCD_WR_DATA(0x00);      //RGB 序列
LCD_WR_REG(0xB4);       //Set horizontal period
LCD_WR_DATA((SSD_HT-1)>>8);
LCD_WR_DATA(SSD_HT-1);
LCD_WR_DATA(SSD_HPS>>8);
LCD_WR_DATA(SSD_HPS);
LCD_WR_DATA(SSD_HOR_PULSE_WIDTH-1);
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x00);
LCD_WR_REG(0xB6);       //Set vertical period
LCD_WR_DATA((SSD_VT-1)>>8);
LCD_WR_DATA(SSD_VT-1);
LCD_WR_DATA(SSD_VPS>>8);
LCD_WR_DATA(SSD_VPS);
LCD_WR_DATA(SSD_VER_FRONT_PORCH-1);
```

```
            LCD_WR_DATA(0x00);
            LCD_WR_DATA(0x00);

            LCD_WR_REG(0xF0);       //设置 SSD1963 与 CPU 接口为 16bit
            LCD_WR_DATA(0x03);      //16-bit(565 format) data for 16bpp
            LCD_WR_REG(0x29);       //开启显示
            //设置 PWM 输出  背光通过占空比可调
            LCD_WR_REG(0xD0);       //设置自动白平衡 DBC
            LCD_WR_DATA(0x00);      //disable
            LCD_WR_REG(0xBE);       //配置 PWM 输出
            LCD_WR_DATA(0x05);      //1 设置 PWM 频率
            LCD_WR_DATA(0xFE);      //2 设置 PWM 占空比
            LCD_WR_DATA(0x01);      //3 设置 C
            LCD_WR_DATA(0x00);      //4 设置 D
            LCD_WR_DATA(0x00);      //5 设置 E
            LCD_WR_DATA(0x00);      //6 设置 F
            LCD_WR_REG(0xB8);       //设置 GPIO 配置
            LCD_WR_DATA(0x03);      //2 个 IO 口设置成输出
            LCD_WR_DATA(0x01);      //GPIO 使用正常的 IO 功能
            LCD_WR_REG(0xBA);
            LCD_WR_DATA(0X01);      //GPIO[1:0]=01,控制 LCD 方向
            LCD_SSD_BackLightSet(100);//背光设置为最亮
        }
        LCD_Display_Dir(0);             //默认为竖屏显示
        LCD_LED=1;                      //点亮背光
        LCD_Clear(WHITE);
    }
```

该函数先对 FSMC 相关 IO 进行初始化，然后是 FSMC 的初始化，这个我们在前面都有介绍，最后根据读到的 LCD ID，对不同的驱动器执行不同的初始化代码，从上面的代码可以看出，这个初始化函数可以针对十多款不同的驱动 IC 执行初始化操作，这样大大提高了整个程序的通用性。大家在以后的学习中应该多使用这样的方式，以提高程序的通用性、兼容性。

最后，我们本例程重点是要驱动 SSD1963，所以我们留下了 SSD1963 的初始化代码，这个代码就是依照我们在《ATK-7' TFTLCD V2 模块用户手册》里面的介绍，根据 LCD 屏的参数，来设置 SSD1963 的相关寄存器，完成对模块的驱动。这里面，用到了几个屏幕参数，在 lcd.h 里面定义了，如下：

```
//LCD 分辨率设置
#define SSD_HOR_RESOLUTION          800         //LCD 水平分辨率
```

```
#define SSD_VER_RESOLUTION      480     //LCD 垂直分辨率
//LCD 驱动参数设置
#define SSD_HOR_PULSE_WIDTH     1       //水平脉宽
#define SSD_HOR_BACK_PORCH      46      //水平前廊
#define SSD_HOR_FRONT_PORCH     210     //水平后廊
#define SSD_VER_PULSE_WIDTH     1       //垂直脉宽
#define SSD_VER_BACK_PORCH      23      //垂直前廊
#define SSD_VER_FRONT_PORCH     22      //垂直前廊
//如下几个参数，自动计算
#define SSD_HT   (SSD_HOR_RESOLUTION+SSD_HOR_BACK_PORCH
                  +SSD_HOR_FRONT_PORCH)
#define SSD_HPS  (SSD_HOR_BACK_PORCH)
#define SSD_VT   (SSD_VER_RESOLUTION+SSD_VER_BACK_PORCH
                  +SSD_VER_FRONT_PORCH)
#define SSD_VPS (SSD_VER_BACK_PORCH)
```

以上参数，就是根据 ATK-7’ TFTLCD V2 模块所用 LCD 屏：AT070TN92 的数据手册来设置的。不同的 LCD 屏，一般也只需要修改这几个参数即可。

特别注意：本函数使用了 printf 来打印 LCD ID，所以，如果你在主函数里面没有初始化串口，那么将导致程序死在 printf 里面!！如果不想用 printf，那么请注释掉它。

3.2 电容触摸屏驱动代码

在 HARDWARE\TOUCH 文件夹下面有：ctiic.c、ctiic.h、ott2001a.c、ott2001a.h、gt9147.c、gt9147.h、ft5206.c 和 ft5206.h 等 8 个文件，这些文件用于驱动电容触摸屏。另外，还有 touch.c 和 touch.h，用于支持电阻触摸屏，这样，总共有 10 个文件。

首先，我们看看 touch.c 里面的代码，由于代码比较多，我们就不一一介绍了，这里我们仅介绍 TP_Init 函数，该函数根据 LCD 的 ID（即 lcddev.id）判别是电阻屏还是电容屏，然后执行不同的初始化，该函数代码如下：

```
//触摸屏初始化
//返回值:0,没有进行校准
//       1,进行过校准
u8 TP_Init(void)
{
    if(lcddev.id==0X5510)                   //4.3 寸电容触摸屏
    {
        if(GT9147_Init()==0)                //是 GT9147
        {
            tp_dev.scan=GT9147_Scan;        //扫描函数指向 GT9147 触摸屏扫描
        }else
        {
            OTT2001A_Init();
```

```
            tp_dev.scan=OTT2001A_Scan;      //扫描函数指向 OTT2001A 触摸屏扫描
        }
        tp_dev.touchtype|=0X80;             //电容屏
        tp_dev.touchtype|=lcddev.dir&0X01;  //横屏还是竖屏
        return 0;
    }else if(lcddev.id==0X1963)             //7 寸电容触摸屏
    {
        FT5206_Init();
        tp_dev.scan=FT5206_Scan;            //扫描函数指向 GT9147 触摸屏扫描
        tp_dev.touchtype|=0X80;             //电容屏
        tp_dev.touchtype|=lcddev.dir&0X01;  //横屏还是竖屏
        return 0;
    }else                                   //电阻式触摸屏
    {
        //注意,时钟使能之后,对 GPIO 的操作才有效
        //所以上拉之前,必须使能时钟.才能实现真正的上拉输出
        RCC->APB2ENR|=1<<3;                 //PB 时钟使能
        RCC->APB2ENR|=1<<7;                 //PF 时钟使能
        GPIOB->CRL&=0XFFFFF00F;             //PB1/2 设置
        GPIOB->CRL|=0X00000830;             //PB1 推挽输出,PB2 上拉输入
        GPIOB->ODR|=3<<1;                   //PB1/2 上拉
        GPIOF->CRH&=0XFFFF000F;             //PF9/10/11 设置
        GPIOF->CRH|=0X00003830;             //PF10 上拉输入,PF9/11 推挽输出
        GPIOF->ODR|=7<<9;                   //PF9,10,11  全部上拉
        TP_Read_XY(&tp_dev.x[0],&tp_dev.y[0]);//第一次读取初始化
        AT24CXX_Init();                     //初始化 24CXX
        if(TP_Get_Adjdata())return 0;       //已经校准
        else                                //未校准?
        {
            LCD_Clear(WHITE);               //清屏
            TP_Adjust();                    //屏幕校准
        }
        TP_Get_Adjdata();
    }
    return 1;
}
```

该函数比较简单，重点说一下：tp_dev.scan，这个结构体函数指针，默认是指向 TP_Scan 的，如果是电阻屏则用默认的即可，如果是电容屏，则指向新的扫描函数 GT9147_Scan 或 OTT2001A_Scan 或 FT5206_Scan（根据芯片 ID 判断到底指向那个），执行电容触摸屏的扫描函数，这两个函数在后续会介绍。

touch.c 里面的其他的函数我们这里就不多介绍了（请参考本例程源码）。接下来打开 touch.h 文件，看看该文件的代码：

```
#ifndef __TOUCH_H__
#define __TOUCH_H__
#define TP_PRES_DOWN 0x80          //触屏被按下
#define TP_CATH_PRES 0x40          //有按键按下了
#define CT_MAX_TOUCH  5            //电容屏支持的点数,固定为 5 点
//触摸屏控制器
typedef struct
{
    u8 (*init)(void);              //初始化触摸屏控制器
    u8 (*scan)(u8);                //扫描触摸屏.0,屏幕扫描;1,物理坐标;
    void (*adjust)(void);          //触摸屏校准
    u16 x[CT_MAX_TOUCH];           //当前坐标
    u16 y[CT_MAX_TOUCH];           //电容屏最多 5 组坐标,电阻屏则用 x[0],y[0]代表:此次扫
                                   //描时,触屏坐标,用 x[4],y[4]存储第一次按下时的坐标.
    u8  sta;                       //笔的状态
                                   //b7:按下 1/松开 0;
                                   //b6:0,没有按键按下;1,有按键按下.
                                   //b5:保留
                                   //b4~b0:电容屏按下点数(0,表示未按下,1 表示按下)
    ////////////////////触摸屏校准参数(电容屏不需要校准)//////////////////////
    float xfac;
    float yfac;
    short xoff;
    short yoff;
    //新增的参数,当触摸屏的左右上下完全颠倒时需要用到.
    //b0:0,竖屏(适合左右为 X 坐标,上下为 Y 坐标的 TP)
    //   1,横屏(适合左右为 Y 坐标,上下为 X 坐标的 TP)
    //b1~6:保留.
    //b7:0,电阻屏
    //   1,电容屏
    u8 touchtype;
```

```
}_m_tp_dev;
extern _m_tp_dev tp_dev;                    //触屏控制器在 touch.c 里面定义
//电阻/电容屏芯片连接引脚
#define PEN          PFin(10)     //PF10 INT
#define DOUT         PBin(2)      //PB2   MISO
#define TDIN         PFout(9)     //PF9   MOSI
#define TCLK         PBout(1)     //PB1   SCLK
#define TCS          PFout(11)    //PF11   CS
//电阻屏函数
void TP_Write_Byte(u8 num);                          //向控制芯片写入一个数据
u16 TP_Read_AD(u8 CMD);                              //读取 AD 转换值
u16 TP_Read_XOY(u8 xy);                              //带滤波的坐标读取(X/Y)
u8 TP_Read_XY(u16 *x,u16 *y);                        //双方向读取(X+Y)
u8 TP_Read_XY2(u16 *x,u16 *y);                       //带加强滤波的双方向坐标读取
void TP_Drow_Touch_Point(u16 x,u16 y,u16 color);     //画一个坐标校准点
void TP_Draw_Big_Point(u16 x,u16 y,u16 color);       //画一个大点
void TP_Save_Adjdata(void);                          //保存校准参数
u8 TP_Get_Adjdata(void);                             //读取校准参数
void TP_Adjust(void);                                //触摸屏校准
void TP_Adj_Info_Show(u16 x0,u16 y0,u16 x1,u16 y1,u16 x2,u16 y2,u16 x3,u16 y3,
                      u16 fac);//显示校准信息
//电阻屏/电容屏  共用函数
u8 TP_Scan(u8 tp);                                   //扫描
u8 TP_Init(void);                                    //初始化
#endif
```

上述代码，我们重点看看_m_tp_dev 结构体，改结构体用于管理和记录触摸屏（包括电阻触摸屏与电容触摸屏）相关信息。通过结构体，在使用的时候，我们一般直接调用 tp_dev 的相关成员函数/变量屏即可达到需要的效果，这种设计简化了接口，且方便管理和维护，大家可以效仿一下。

ctiic.c 和 ctiic.h 是电容触摸屏的 IIC 接口部分代码，与开发板 IIC 实验的 myiic.c 和 myiic.h 基本一样，这里就不单独介绍了，记得把 ctiic.c 加入 HARDWARE 组下。接下来看看：ft5206.c，在该文件输入如下代码：

```
//向 FT5206 写入一次数据
//reg:起始寄存器地址
//buf:数据缓缓存区
//len:写数据长度
//返回值:0,成功;1,失败.
```

```
u8 FT5206_WR_Reg(u16 reg,u8 *buf,u8 len)
{
    u8 i;
    u8 ret=0;
    CT_IIC_Start();
    CT_IIC_Send_Byte(FT_CMD_WR);   //发送写命令
    CT_IIC_Wait_Ack();
    CT_IIC_Send_Byte(reg&0XFF);    //发送低 8 位地址
    CT_IIC_Wait_Ack();
    for(i=0;i<len;i++)
    {
        CT_IIC_Send_Byte(buf[i]);  //发数据
        ret=CT_IIC_Wait_Ack();
        if(ret)break;
    }
    CT_IIC_Stop();                 //产生一个停止条件
    return ret;
}
//从 FT5206 读出一次数据
//reg:起始寄存器地址
//buf:数据缓缓存区
//len:读数据长度
void FT5206_RD_Reg(u16 reg,u8 *buf,u8 len)
{
    u8 i;
    CT_IIC_Start();
    CT_IIC_Send_Byte(FT_CMD_WR);   //发送写命令
    CT_IIC_Wait_Ack();
    CT_IIC_Send_Byte(reg&0XFF);    //发送低 8 位地址
    CT_IIC_Wait_Ack();
    CT_IIC_Start();
    CT_IIC_Send_Byte(FT_CMD_RD);   //发送读命令
    CT_IIC_Wait_Ack();
    for(i=0;i<len;i++) buf[i]=CT_IIC_Read_Byte(i==(len-1)?0:1); //发数据
    CT_IIC_Stop();//产生一个停止条件
}
```

```
//初始化 FT5206 触摸屏
//返回值:0,初始化成功;1,初始化失败
u8 FT5206_Init(void)
{
    u8 temp[2];
    RCC->APB2ENR|=1<<7;          //先使能外设 IO PORTF 时钟
    GPIOF->CRH&=0XFFFF00FF;
    GPIOF->CRH|=0X00003800;      //PF10 输入 PF11 推挽输出
    GPIOF->ODR|=3<<10;           //PF10/11 上拉/输出 1
    CT_IIC_Init();               //初始化电容屏的 I2C 总线
    FT_RST=0;                    //复位
    delay_ms(20);
    FT_RST=1;                    //释放复位
    delay_ms(50);
    temp[0]=0;
    FT5206_WR_Reg(FT_DEVIDE_MODE,temp,1);     //进入正常操作模式
    FT5206_WR_Reg(FT_ID_G_MODE,temp,1);       //查询模式
    temp[0]=22;                               //触摸有效值，22，越小越灵敏
    FT5206_WR_Reg(FT_ID_G_THGROUP,temp,1);    //设置触摸有效值
    temp[0]=12;                               //激活周期，12~14
    FT5206_WR_Reg(FT_ID_G_PERIODACTIVE,temp,1);
    //读取版本号，参考值：0x3003
    FT5206_RD_Reg(FT_ID_G_LIB_VERSION,&temp[0],2);
    if(temp[0]==0X30&&temp[1]==0X03)//版本:0X3003
    {
        printf("CTP ID:%x\r\n",((u16)temp[0]<<8)+temp[1]);
        return 0;
    }
    return 1;
}
const u16 FT5206_TPX_TBL[5]={FT_TP1_REG,FT_TP2_REG,FT_TP3_REG,
                             FT_TP4_REG,FT_TP5_REG};
//扫描触摸屏(采用查询方式)
//mode:0,正常扫描.
//返回值:当前触屏状态.
//0,触屏无触摸;1,触屏有触摸
```

```
u8 FT5206_Scan(u8 mode)
{
    u8 buf[4];
    u8 i=0; u8 res=0; u8 temp;
    static u8 t=0;//控制查询间隔,从而降低 CPU 占用率
    t++;
    if((t%10)==0||t<10)//空闲时,每 10 次 CTP_Scan 才检测 1 次,从而节省 CPU 使用率
    {
        FT5206_RD_Reg(FT_REG_NUM_FINGER,&mode,1);//读取触摸点的状态
        if((mode&0XF)&&((mode&0XF)<10))
        {
            temp=0XFF<<(mode&0XF);//将点数转换为 1 的位数,匹配 tp_dev.sta 定义
            tp_dev.sta=(~temp)|TP_PRES_DOWN|TP_CATH_PRES;
            for(i=0;i<5;i++)
            {
                if(tp_dev.sta&(1<<i))    //触摸有效?
                {
                    FT5206_RD_Reg(FT5206_TPX_TBL[i],buf,4);  //读取 XY 坐标值
                    if(tp_dev.touchtype&0X01)//横屏
                    {
                        tp_dev.y[i]=((u16)(buf[0]&0X0F)<<8)+buf[1];
                        tp_dev.x[i]=((u16)(buf[2]&0X0F)<<8)+buf[3];
                    }else
                    {
                        tp_dev.x[i]=480-(((u16)(buf[0]&0X0F)<<8)+buf[1]);
                        tp_dev.y[i]=((u16)(buf[2]&0X0F)<<8)+buf[3];
                    }
                    if((buf[0]&0XF0)!=0X80)tp_dev.x[i]=tp_dev.y[i]=0;
                    //必须是 contact 事件，才认为有效
                    //printf("x[%d]:%d,y[%d]:%d\r\n",i,tp_dev.x[i],i,tp_dev.y[i]);
                }
            }
            res=1;
            if(tp_dev.x[0]==0 && tp_dev.y[0]==0)mode=0;  //数据全 0,则忽略此次数据
            t=0;        //触发一次,则会最少连续监测 10 次,从而提高命中率
        }
```

```
        }
        if((mode&0X1F)==0)//无触摸点按下
        {
            if(tp_dev.sta&TP_PRES_DOWN) //之前是被按下的
            {
                tp_dev.sta&=~(1<<7);        //标记按键松开
            }else                           //之前就没有被按下
            {
                tp_dev.x[0]=0xffff;
                tp_dev.y[0]=0xffff;
                tp_dev.sta&=0XE0;           //清除点有效标记
            }
        }
        if(t>240)t=10;                      //重新从 10 开始计数
        return res;
    }
```

此部分总共 4 个函数，其中 FT5206_WR_Reg 和 FT5206_RD_Reg 分别用于读写 FT5206 芯片。这里，我们重点介绍下 FT5206_Scan 函数，FT5206_Scan 函数用于扫描电容触摸屏是否有按键按下，由于我们不是用的中断方式来读取 FT5206 的数据的，而是采用查询的方式，所以这里使用了一个静态变量来提高效率，当无触摸的时候，尽量减少对 CPU 的占用，当有触摸的时候，又保证能迅速检测到。至于对 FT5206 数据的读取，则完全是我们在上面介绍的方法，先读取触摸状态寄存器（FT_REG_NUM_FINGER，0X02），判断是不是有有效数据，如果有，则读取，否则直接忽略，继续后面的处理。

其他的函数我们这里就不多介绍了，保存 ft5206.c 文件，并把该文件加入到 HARDWARE 组下。接下来打开 ft5206.h 文件，在该文件里面输入如下代码：

```
#ifndef __OTT2001A_H
#define __OTT2001A_H
#include "sys.h"
//与电容触摸屏连接的芯片引脚(未包含 IIC 引脚)
//IO 操作函数
#define FT_RST                  PFout(11)   //FT5206 复位引脚
#define FT_INT                  PFin(10)    //FT5206 中断引脚
//I2C 读写命令
#define FT_CMD_WR               0X70        //写命令
#define FT_CMD_RD               0X71        //读命令
//FT5206 部分寄存器定义
#define FT_DEVIDE_MODE          0x00        //FT5206 模式控制寄存器
```

```
#define FT_REG_NUM_FINGER       0x02        //触摸状态寄存器
#define FT_TP1_REG              0X03        //第一个触摸点数据地址
#define FT_TP2_REG              0X09        //第二个触摸点数据地址
#define FT_TP3_REG              0X0F        //第三个触摸点数据地址
#define FT_TP4_REG              0X15        //第四个触摸点数据地址
#define FT_TP5_REG              0X1B        //第五个触摸点数据地址
#define FT_ID_G_LIB_VERSION     0xA1        //版本
#define FT_ID_G_MODE            0xA4        //FT5206 中断模式控制寄存器
#define FT_ID_G_THGROUP         0x80        //触摸有效值设置寄存器
#define FT_ID_G_PERIODACTIVE    0x88        //激活状态周期设置寄存器
u8 FT5206_WR_Reg(u16 reg,u8 *buf,u8 len);
void FT5206_RD_Reg(u16 reg,u8 *buf,u8 len);
u8 FT5206_Init(void);
u8 FT5206_Scan(u8 mode);
#endif
```

这段代码比较简单，主要是一些寄存器的定义和几个函数声明。另外，因为 FT5206 的 IIC 从机地址是 0X38，转换为读写操作后：0X70 表示写；0X71 表示读。

最后我们打开 test.c，修改部分代码，这里就不全部贴出来了，仅介绍三个重要的函数：

```
 //5 个触控点的颜色
const u16 POINT_COLOR_TBL[CT_MAX_TOUCH]={RED,GREEN,BLUE,BROWN,
                                         GRED};
//电阻触摸屏测试函数
void rtp_test(void)
{
    u8 key; u8 i=0;
    while(1)
    {
        key=KEY_Scan(0);
        tp_dev.scan(0);
        if(tp_dev.sta&TP_PRES_DOWN)          //触摸屏被按下
        {
            if(tp_dev.x[0]<lcddev.width&&tp_dev.y[0]<lcddev.height)
            {
                if(tp_dev.x[0]>(lcddev.width-24)&&tp_dev.y[0]<16)
                    Load_Drow_Dialog();//清除
                else TP_Draw_Big_Point(tp_dev.x[0],tp_dev.y[0],RED);//画图
```

```
                }
            }else delay_ms(10);      //没有按键按下的时候
            if(key==KEY0_PRES)  //KEY0 按下,则执行校准程序
            {
                LCD_Clear(WHITE);//清屏
                TP_Adjust();        //屏幕校准
                Load_Drow_Dialog();
            }
            i++;
            if(i%20==0)LED0=!LED0;
        }
    }
    //电容触摸屏测试函数
    void ctp_test(void)
    {
        u8 t=0; u8 i=0;
        u16 lastpos[5][2];          //记录最后一次的数据
        while(1)
        {
            tp_dev.scan(0);
            for(t=0;t<CT_MAX_TOUCH;t++)
            {
                if((tp_dev.sta)&(1<<t))
                {
                    if(tp_dev.x[t]<lcddev.width&&tp_dev.y[t]<lcddev.height)
                    {
                        if(lastpos[t][0]==0XFFFF)
                        {
                            lastpos[t][0] = tp_dev.x[t];
                            lastpos[t][1] = tp_dev.y[t];
                        }
                        lcd_draw_bline(lastpos[t][0],lastpos[t][1],tp_dev.x[t],tp_dev.y[t],2,
                                       POINT_COLOR_TBL[t]);//画线
                        lastpos[t][0]=tp_dev.x[t];
                        lastpos[t][1]=tp_dev.y[t];
                        if(tp_dev.x[t]>(lcddev.width-24)&&tp_dev.y[t]<16)
```

```
                        {
                            Load_Drow_Dialog();//清除
                        }
                    }
                }else lastpos[t][0]=0XFFFF;
            }
            delay_ms(5);i++;
            if(i%20==0)LED0=!LED0;
        }
    }
    int main(void)
    {
        Stm32_Clock_Init(9);            //系统时钟设置
        uart_init(72,115200);           //串口初始化为 115200
        delay_init(72);                 //延时初始化
        LED_Init();                     //初始化与 LED 连接的硬件接口
        LCD_Init();                     //初始化 LCD
        KEY_Init();                     //按键初始化
        tp_dev.init();                  //触摸屏初始化
        POINT_COLOR=RED;                //设置字体为红色
        LCD_ShowString(30,50,200,16,16," WarShip STM32");
        LCD_ShowString(30,70,200,16,16,"TOUCH TEST");
        LCD_ShowString(30,90,200,16,16,"ATOM@ALIENTEK");
        LCD_ShowString(30,110,200,16,16,"2015/1/15");
        if((tp_dev.touchtype&0X80)==0X00)
            LCD_ShowString(30,130,200,16,16,"Press KEY0 to Adjust");//电阻屏才显示
        delay_ms(1500);
        Load_Drow_Dialog();
        if(tp_dev.touchtype&0X80)ctp_test();        //电容屏测试
        else rtp_test();                            //电阻屏测试
    }
```

下面分别介绍一下这三个函数。

rtp_test，该函数用于电阻触摸屏的测试，该函数代码比较简单，就是扫描按键和触摸屏，如果触摸屏有按下，则在触摸屏上面划线，如果按中“RST”区域，则执行清屏。如果按键 KEY0 按下，则执行触摸屏校准。

ctp_test，该函数用于电容触摸屏的测试，由于我们采用 tp_dev.sta 来标记当前按下的触摸屏点数，所以判断是否有电容触摸屏按下，也就是判断 tp_dev.sta 的最低 5 位，如果有数

据，则划线，如果没数据则忽略，且 5 个点划线的颜色各不一样，方便区分。另外，电容触摸屏不需要校准，所以没有校准程序。

main 函数，则比较简单，初始化相关外设，然后根据触摸屏类型，去选择执行 ctp_test 还是 rtp_test。

软件部分就介绍到这里，接下来看看下载验证。

4、验证

在代码编译成功之后，我们下载代码到我们的 STM32 开发板上，ATK-7’ TFTLCD V2 电容触摸屏模块测试效果如图 4.1 和 4.2 所示：

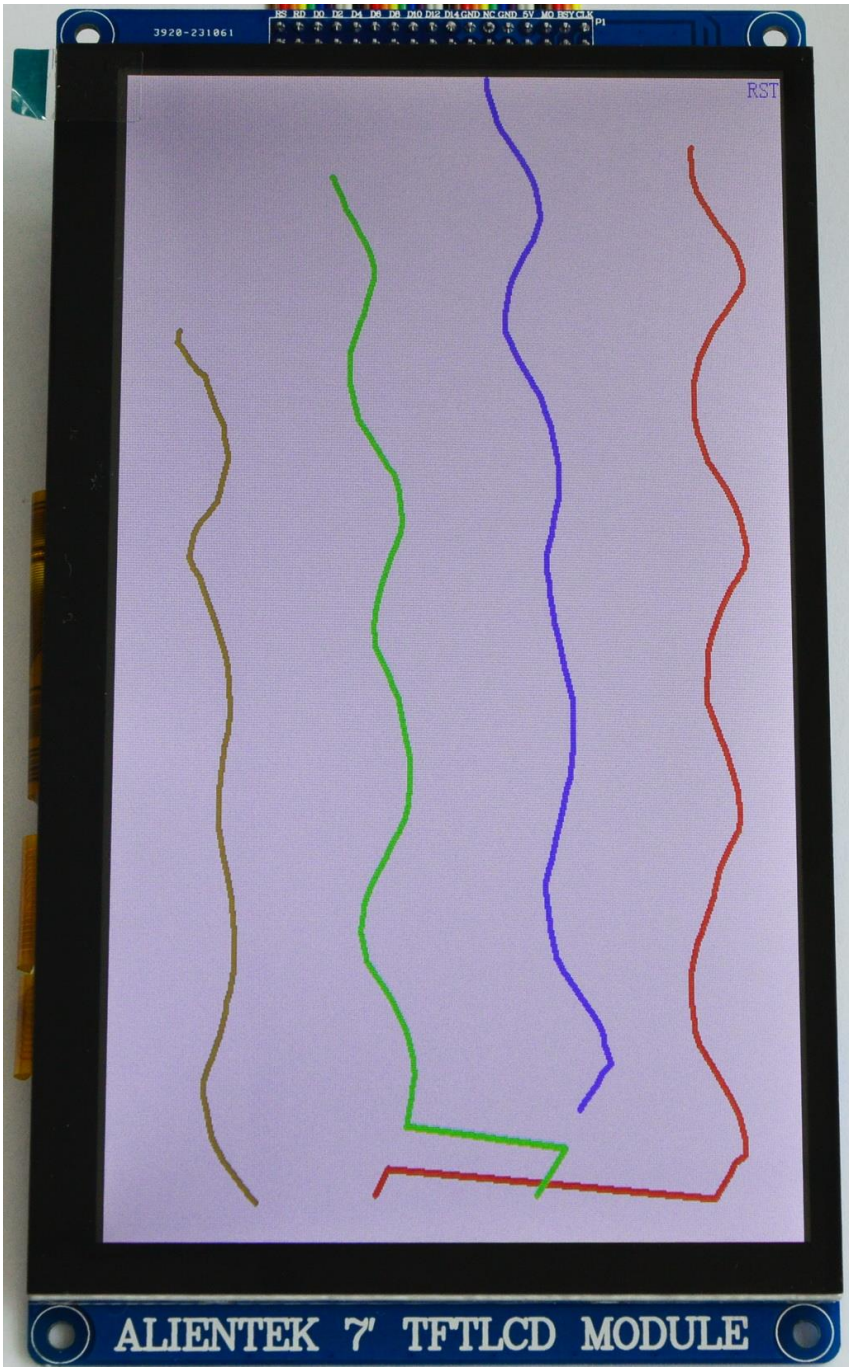

图 4.1 ATK-7’ TFTLCD V2 电容触摸屏多点触摸效果

图 4.1 是 ATK-7’ TFTLCD V2 电容触摸屏多点触摸画图的效果图，图中是 4 点同时触摸的

演示效果。模块最多支持 5 点同时触摸。

这里提醒大家，多点触摸的时候，手指尖间隔不要太近，否则触摸效果不好，甚至无法实现多点触摸。

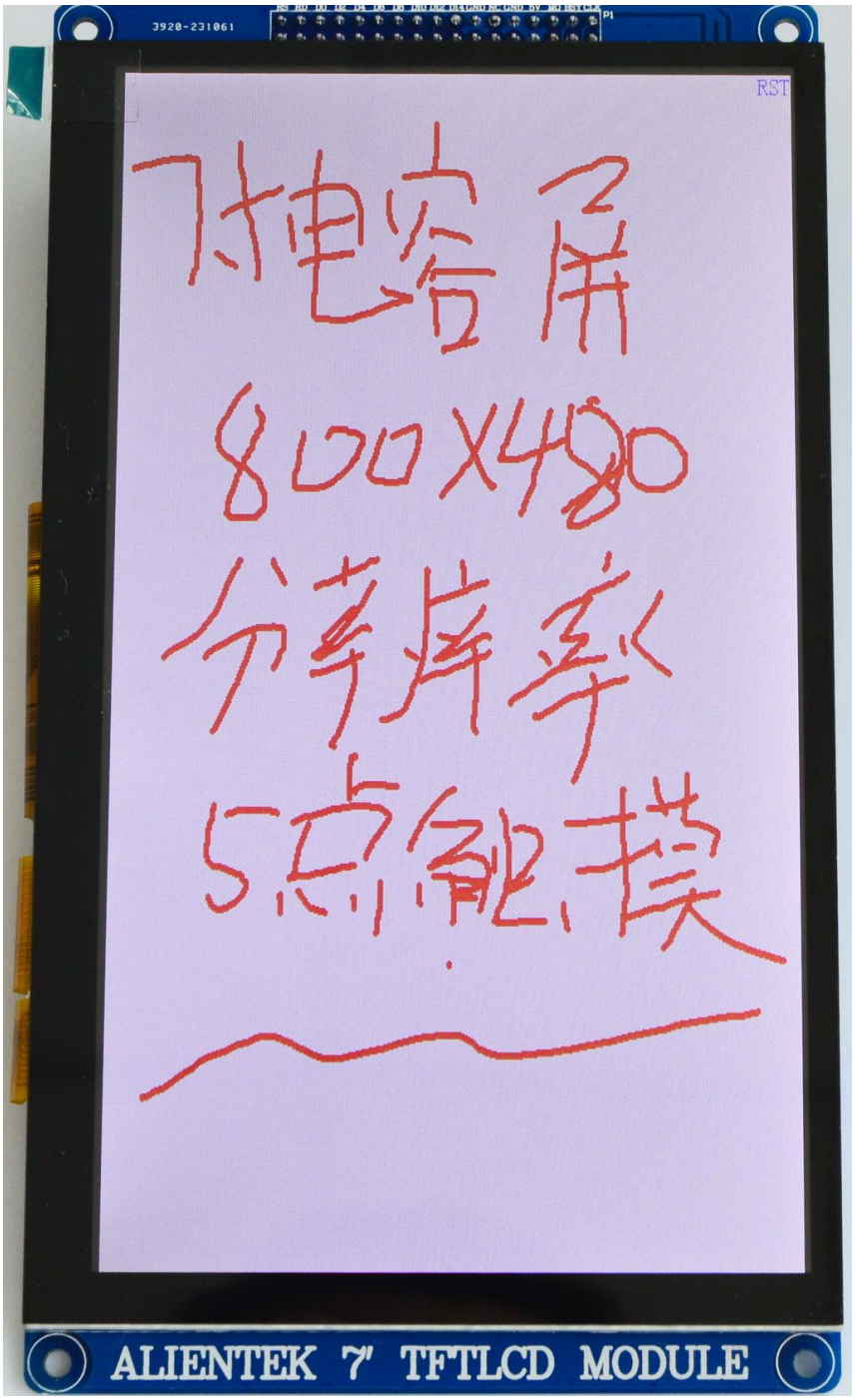

图 4.2 手写效果

图 4.2 是 ATK-7’ TFTLCD V2 电容触摸屏的手写效果。

至此，ATK-7’ TFTLCD V2 电容触摸屏模块的使用就介绍完了，希望通过本文档，大家可以快速掌握 ATK-7’ TFTLCD V2 电容触摸屏模块的使用。

正点原子@ALIENTEK
公司网址：www.alientek.com
技术论坛：www.openedv.com
电话：020-38271790
传真：020-36773971