AN1612 ATK-7’ TFTLCD 电容触摸屏模块使用说明

本应用文档(AN1612)将教大家如何在 ALIENTEK 阿波罗 STM32F429 开发板上使用 ATK-7’ TFTLCD 电容触摸屏模块。(本文档仅针对 V1 版本的 7 寸屏模块，即：CPLD+FT5206/FT5426 方案；非 V2 版本的 7 寸屏模块，V2 版本采用的是：SSD1963+FT5206/5426 方案)。

本文档分为如下几部分：

1， ATK-7’ TFTLCD 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 电容触摸屏模块简介

ATK-7’ TFTLCD 电容触摸屏模块，是 ALIENTEK 生产的一款高性能 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，模块自带 LCD 控制器，拥有多达 8MB 的显存，能提供 8 页的显存，并支持任意点颜色读取。模块采用电容触摸屏，支持 5 点同时触摸，具有非常好的操控效果。同时，模块还提供了镜像翻转、背光控制等功能，方便用户使用。

ATK-7’ TFTLCD 模块采用单 5V 供电，工作电流为 130mA~350mA，功耗很低，非常适合各类型产品使用。

1.1 模块引脚说明

ATK-7’ TFTLCD 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 电容触摸屏模块外观如图 1.1.1 所示：

图 1.1.1-1 ATK-7’ TFTLCD 电容触摸屏模块正面图

图 1.1.1-2 ATK-7’ TFTLCD 电容触摸屏模块背面图

模块通过 34（2*17）个引脚同外部连接，各引脚的详细描述如表 1.1.1 所示：

序号	名称	说明
1	NCE	LCD 控制器片选信号（低电平有效）
2	RS	命令/数据控制信号（0，命令；1，数据；）
3	WR	写使能信号（低电平有效）
4	RD	读使能信号（低电平有效）
5	RST	复位信号（低电平有效）
6~21	D0~D15	双向数据总线
22,26,27	GND	地线
23~25	NC	未用到
28	VCC	5V 电源输入引脚
29	MISO	NC，电容触摸屏未用到
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)
32	BUSY	NC，电容触摸屏未用到
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)

表 1.1.1 ATK-7’ TFTLCD 模块引脚说明

从上表可以看出，LCD 控制器总共需要 21 个 IO 口驱动，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。

1.2 LCD 控制器接口时序

ATK-7’ TFTLCD 模块自带的 LCD 控制器采用 16 位 8080 总线接口，总线写时序如图 1.2.1 所示：

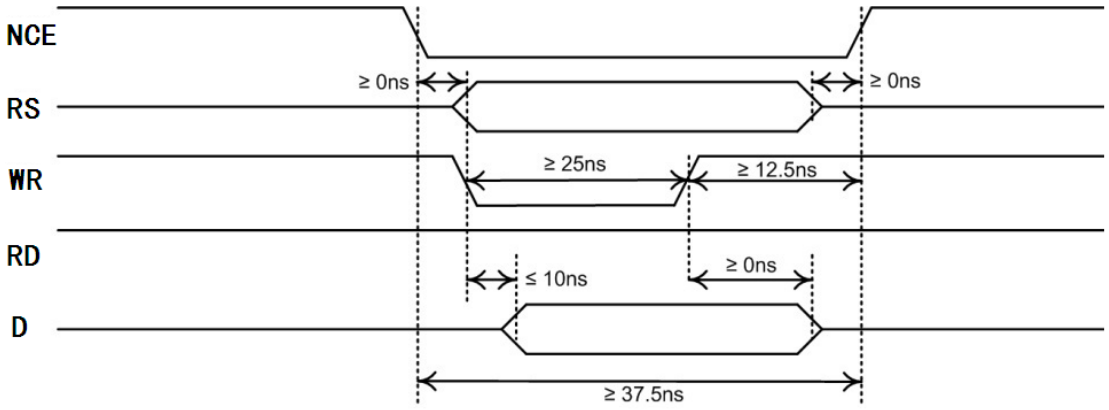

图 1.2.1 总线写时序

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：

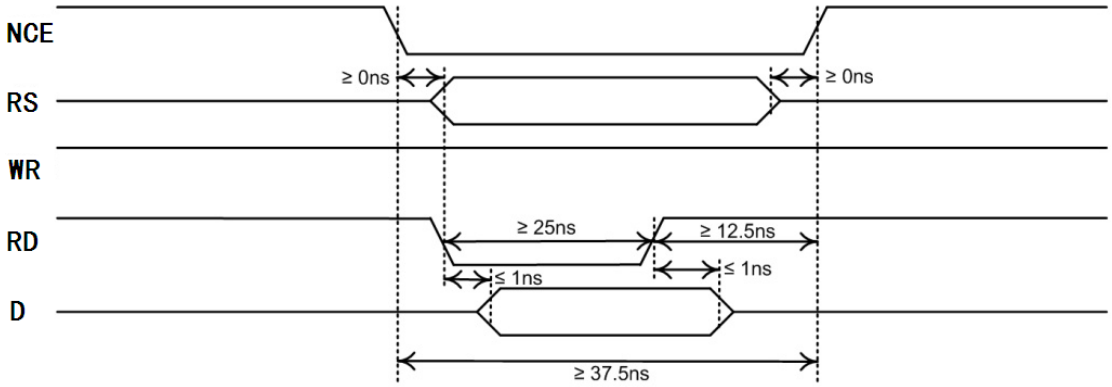

图 1.2.2 总线读时序

ATK-7’ TFTLCD 模块自带的 LCD 控制器可读的寄存器只有 2 个，当 RS 为 0 的时候，表示读取的是状态寄存器(STATE)，当 RS 为 1 的时候，表示读取的是像素数据(DATA)，读期间的地址寄存器(ADDR)将被忽略。

1.3 LCD 控制器寄存器说明

ATK-7’ TFTLCD 模块自带的 LCD 控制器各个寄存器的地址和功能简介如表 1.3.1 所示：

RS	操作	位宽	地址	名称	功能简介	复位值
0	写	16	—	ADDR	设置地址寄存器的值	0x0000
0	读	16	—	STATE	读状态寄存器	0x0000
1	读	16	—	DATA	读像素数据	0x0000
1	写	16	0x00	CUR_Y	设置屏幕的 Y 坐标	0x0000
1	写	16	0x01	CUR_X	设置屏幕的 X 坐标	0x0000
1	写	16	0x02	PIXELS	写入像素数据	0x0000
1	写	16	0x03	END_X	设置 X 方向自动返回的坐标，以及页拷贝时 X 方向的结束坐标	0x031f
1	写	16	0x04	保留		
1	写	16	0x05	PREF	设置当前显示页、当前操作页，背光等	0x0000

1	写	8	0x06	保留		
1	写	8	0x07	MIRROR	控制镜像翻转	0x0001

表 1.3.1 ATK-7’ TFTLCD 模块自带 LCD 驱动器寄存器地址和功能简介

1.3.1 CUR_X 寄存器(0x01)和 CUR_Y 寄存器(0x00)

寄存器 CUR_X 和 CUR_Y 用于设置待操作像素点的坐标，TFTLCD 屏幕上坐标的排列如图 1.3.1.1 所示：

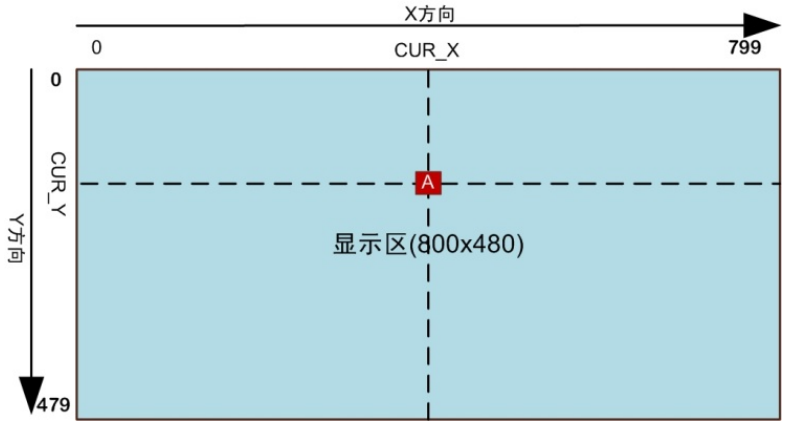

图 1.3.1.1 坐标排列

当 CUR_Y 和 CUR_X 的值确定后，像素点 A 的位置便被唯一的确定了，随后的写入的像素数据会被准确的放置在 A 点。

1.3.2 PIXELS 寄存器(0x02)

寄存器 PIXELS 对应着 16 位的颜色数据，如果当前显示页与当前操作页相同，那么写入 PIXELS 的数据会被立即呈现在由 CUR_X 和 CUR_Y 选中的当前激活点上，如果当前显示页与当前操作页不相同，那么写入 PIXELS 的数据不会被立即呈现出来。

ATK-7’ TFTLCD 模块的颜色格式为 RGB565，具体的颜色与每个位对应关系如表 1.3.2.1 所示：

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

图 1.3.2.1 颜色与位对应关系

1.3.3 END_X 寄存器(0x03)

为了提高像素数据连续读写的效率，当设置好 CUR_X 和 CUR_Y 后，每读取/写入一个像素，当前激活点的 X 坐标就会自动加一，当激活点的 X 坐标等于 END_X 后，便会自动返回 CUR_X 同时 Y 坐标自动加一。如图 1.3.3.1 所示：

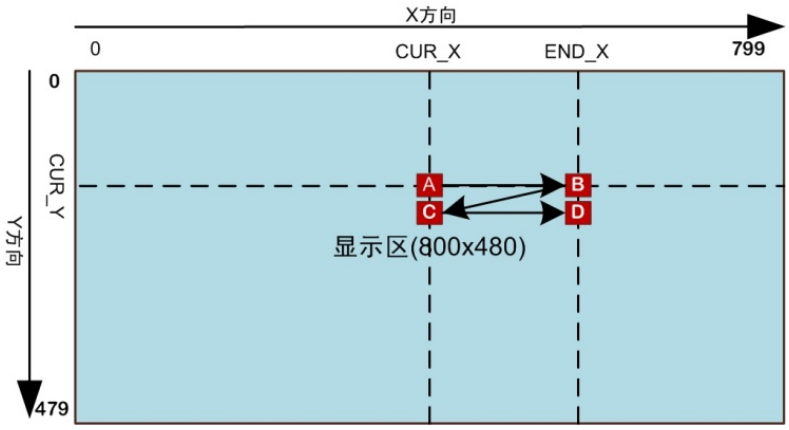

图 1.3.3.1 X 坐标自动返回示意图

以写数据为例，假设 CUR_X、CUR_Y、END_X 分别为 400、200、500，A 点、B 点、C 点、D 点的坐标分别为（400，200）、（500，200）、（400，201）、（500，201）。设置好 CUR_X、CUR_Y 后，第一个像素写到了 A 点，第 100 个像素写到 B 点，第 101 个像素写到 C 点，第 200 个像素写到 D 点，依此类推。

借助 END_X 寄存器，可以简化 MCU 批量数据读写的流程，假设 MCU 需要以（100，200）为起始坐标写入一个 10×20 的矩形，那么只需要将 CUR_X 设为 100，CUR_Y 设为 200，END_X 设为 210，然后进行 200 次的像素点读/写操作即可，期间不需要再进行坐标设置操作，所有的坐标都会被自动推算。

1.3.4 PREF 寄存器(0x05)

PREF 寄存器用于设置当前显示页、当前操作页和 TFT 背光，各个位的具体含义如表 1.3.4.1 所示：

位	名称	功能简介	复位值
b5~b0	BK_PWM	背光控制	0
b8~b6	保留	——	0
b11~b9	CUR_PAGE	当前显示的页	0
b14~b12	OPT_PAGE	当前操作的页	0
b15	保留	——	0

表 1.3.4.1 HREF 寄存器各位定义

其中，BK_PWM 用于设置背光信号的占空比，从而调节 TFT 背光的亮度，取值范围为 0~63，0 代表背光关闭，63 代表背光最亮。上电复位后 BK_PWM 的值默认为 0，也就是背光关闭，在 MCU 对 BK_PWM 赋以非零值后，背光才能点亮。

当前显示页由 CUR_PAGE 指定，表示屏幕上实际显示的显存分页，当前操作页由 OPT_PAGE 指定，表示当前读写操作的显存分页。如果 CUR_PAGE 与 OPT_PAGE 指向同一显存分页，那么写显存操作的结果会被立即呈现在屏幕上，如果 CUR_PAGE 与 OPT_PAGE 指向不同的显存分页，那么对 OPT_PAGE 的任何操作都不会影响屏幕上的显示内容，只有在 CUR_PAGE 切换到 OPT_PAGE 后，OPT_PAGE 中数据才会被显示出来。

1.3.5 MIRROR 寄存器(0X07)

MIRROR 寄存器用于实现图像的水平和垂直镜像翻转，该寄存器各位的具体含义如表 1.3.5.1 所示。

位	名称	功能简介	复位值
b15~b2	保留	——	0
b1	UD	控制垂直镜像翻转	0
b0	LR	控制水平镜像翻转	1

表 1.3.5.1 MIRROR 寄存器各位定义

UD 位用于控制显示画面的垂直翻转，LR 位用于控制显示画面的水平翻转，操作 UD 位和 LR 位会影响 TFT 上的像素点位置与显存中数据地址的映射关系，但不会改变显存中的数据，不同的 UD 和 LR 值所对应的显示效果如图 1.3.5.1 所示。

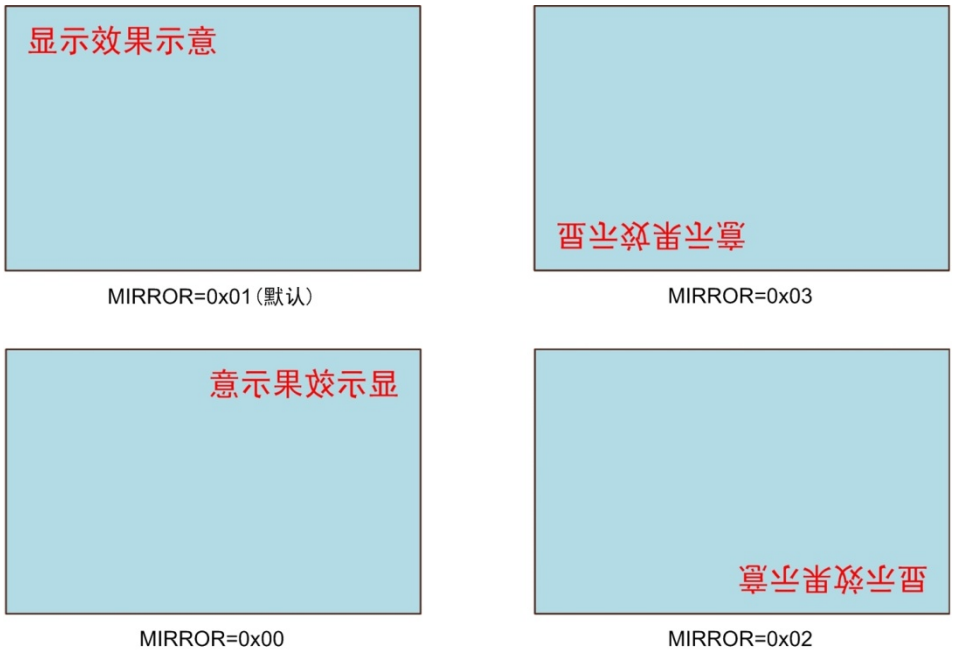

图 1.3.5.1 显示效果示意图

1.3.6 STATE/ DATA 寄存器

这两个寄存器相互配合，用于完成像素数据的读操作。STATE 寄存器的位定义如表 1.3.6.1 所示，读取该寄存器会自动启动像素点的读操作，当 MCU 查询到 STATE 的 DATA_OK 位（b0 位）为 1 后，表示像素数据有效，然后 MCU 读 DATA 寄存器即可获得对应点的像素数据，与写像素数据的操作相同，读像素数据的像素点位置也是由当前的 CUR_X 和 CUR_Y 定义的。当 MCU 读取 DATA 寄存器后，DATA_OK 位会被自动清零。需要注意的是，读 STATE 寄器时，b15~b1 位是随机值，因此在判断 DATA_OK 时，需要屏蔽掉这些位。

位	名称	功能简介	复位值
b15~b1	保留		0
b0	DATA_OK	数据有标志	0

表 1.3.6.1 STATE 寄存器各位定义

1.4 LCD 控制器使用说明

经过前面的介绍，我们对 ATK-7’ TFTLCD 模块自带的 LCD 控制器有了个大概的了解，接下来，我们介绍一下该驱动器的具体使用方法。

1.4.1 初始化

ATK-7’ TFTLCD 模块自带的 LCD 控制器初始化非常简单，分为如下 3 个步骤：

1， 复位。

通过拉低模块 RST 脚，实现对 LCD 控制器的复位，延时 100ms 左右，再拉高 RST 脚，完成对 LCD 的复位。

2， 等待控制器准备好。

在复位后，我们通过直接读 LCD 控制的数据，直到读到的数据最低位变为 1，说明 LCD 驱动器准备好了，可以开始后续的的操作。

3， 设置显示相关寄存器。

这里，我们要设置的包括：MIRROR 寄存器、PREF 寄存器、ENDX 寄存器等三个寄存器。通过 MIRROR 寄存器设置屏幕的显示方向，这里我们默认设置为 0X01。然后，通过 PREF 寄存器设置当前操作页、当前显示页以及背光控制等参数，这里我们默认设置操作页和显示页均为第 0 页。然后设置背光为 63，设置背光到最亮。最后，通过 ENDX 寄存器设置 X 坐标的结束位置，由于 ATK-7’ TFTLCD 模块使用的 LCD 分辨率为 800*480，所以默认设置 ENDX 的值为 799，满足全屏显示的需要。

经过以上三步设置，LCD 驱动器的初始化就完成了。

1.4.2 画点

LCD 驱动最重要的就是画点了，这里我们简单介绍一下 ATK-7’ TFTLCD 模块的画点实现。用模块自带的 LCD 驱动器实现画点也是非常简单，我们只需通过操作三个寄存器，即可实现任意点的画点。

首先，将我们要画的点的坐标写入 CUR_X 和 CUR_Y 这两个寄存器。然后，我们在 PIXELS 寄存器里面写入该点的颜色值，即完成画点操作了。如果设置操作页和显示页相同的话，我们就可以立马在 LCD 上看到这个画出来的点。

1.4.3 读点

LCD 驱动另一个重要的功能就是读取点的颜色，方便做其他处理。ATK-7’ TFTLCD 模块自带的 LCD 驱动器可以实现任意点的读取，方法类似画点操作，不过稍有区别。

首先，我们同样是通过 CUR_X 和 CUR_Y 两个寄存器，设置要读取点的坐标。然后，我们读 STATE 寄存器（RS=0），等待 STATE 寄存器的最低位变为 1，之后，我们读取 DATA 寄存器（RS=1），就可以读到指定点的颜色。 这样我们就实现一个点颜色的读取。

1.5 电容触摸屏接口说明

1.5.1 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 V1 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：

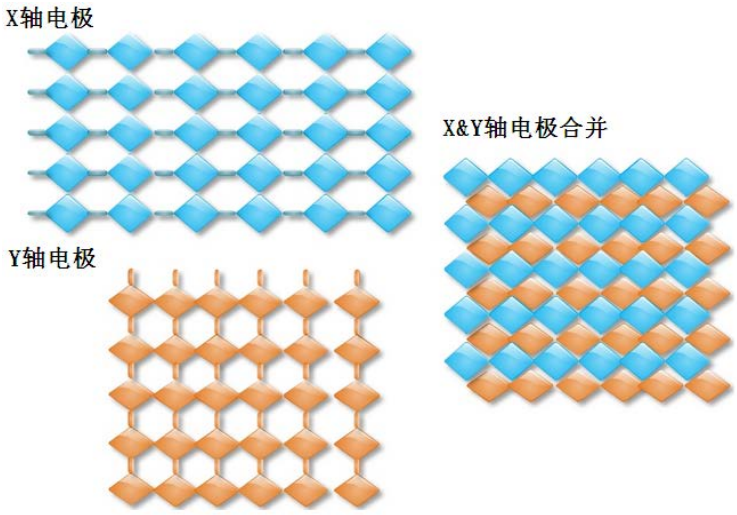

图 1.5.1.1 投射式电容屏电极矩阵示意图

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 V1 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。

1.5.2 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 V1 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 1.5.2.1 所示：

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

表 1.5.2.1 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 1.5.2.2 所示：

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

表 1.5.2.2 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 1.5.2.3 所示：

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

表 1.5.2.3 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 1.5.2.4 所示：

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

表 1.5.2.4 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 V1 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 1.5.2.5 所示：

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

表 1.5.2.5 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 1.5.2.6 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 1.5.2.6 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。

注意：实际上我们使用的触摸 IC 可能是 FT5206 也可能是 FT5426，他们的使用完全一模一样，代码上面也是兼容的，所以，我们不单独介绍 FT5206 了。

2、硬件连接

本实验功能简介：本实验用于测试 ATK-7’ TFTLCD 模块，总共包括三大项测试：

1，电容触摸屏测试—通过按 KEY0 按键进入此项测试。进入测试后，可以在屏幕上实现触摸画线，最多支持 5 点触摸，通过按屏幕右上角的"RST"可以实现清屏。

2，图片显示测试—通过按 KEY1 按键进入此项测试。此项测试需要一个 SD 卡，并且在 SD 卡根目录放一个 PICTURE 文件夹，里面放一些图片文件(bmp/jpeg/gif 等)，然后程序检测到图片后，就开始在 LCD 模块上 PICTURE 文件夹里面的图片。通过 KEY0/KEY1 可以切换下一张/上一张图片，通过按 KEY_UP 按键，可以暂停/继续自动播放(DS1 用于指示是否处于暂停状态)。

3，液晶自测试—通过按 KEY_UP 按键进入此项测试。此项测试又分为 4 个测试小项:速

度测试/镜像测试/缓存测试/背光测试。速度测试类似 ucGUI 的测试效果，测试结果将显示在 LCD 上(像素/秒);镜像测试，展示液晶的 4 个显示效果:正常/上下调转/左右调转/上下左右都调转;缓存测试，用于测试 LCD 模块的 8 页显存，每页显示一种颜色;背光测试，用于测试模块的背光控制功能，背光将从暗到亮变化。

所要用到的硬件资源如下：

1， 指示灯 DS0 和 DS1

2， KEY0/KEY1/KEY_UP 等三个按键

3， 串口 1

4， ATK-7’ TFTLCD 电容触摸屏模块

ATK-7’ TFTLCD 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 阿波罗 STM32F429 开发板的 MCU 屏接口上（插底板上的 MCU 屏接口），不过由于 7 寸屏比较大，建议大家采用延长线连接，方便测试使用。

在硬件上，ATK-7’ TFTLCD 模块与阿波罗 STM32F429 开发板的 IO 口对应关系如下：

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;

NCE 对应 PD7 即 FSMC_NE1;

RS 对应 PD13 即 FSMC_A18;

WR 对应 PD5 即 FSMC_NWE;

RD 对应 PD4 即 FSMC_NOE;

D[15:0]则直接连接在 FSMC_D15~FSMC_D0;

MOSI(CT_SDA)连接 PI3;

CLK(CT_SCL)连接 PF11;

PEN(CT_INT)连接 PH7;

CS(CT_RST)连接 PI8;

最后提醒大家：延长线如果自己做，在没有转接板的情况下，需要在一端做跳线，绝对不能一对一的直接压线，否则压出来的是反的!! 所以，最好采用 ALIENTEK 提供的转接板，这样就可以直接压线了。

3、软件实现

本实验，我们在阿波罗 STM32F429 开发板的图片显示实验基础上进行修改，我们去掉原工程的 lcd.c 和其他一些未用到的.c 文件，同时加入 timer.c、blcd.c、ctiic.c 和 ft5206.c 等三个.c 文件，具体步骤我们就不一一细说了，最终的工程如图 3.1 所示：

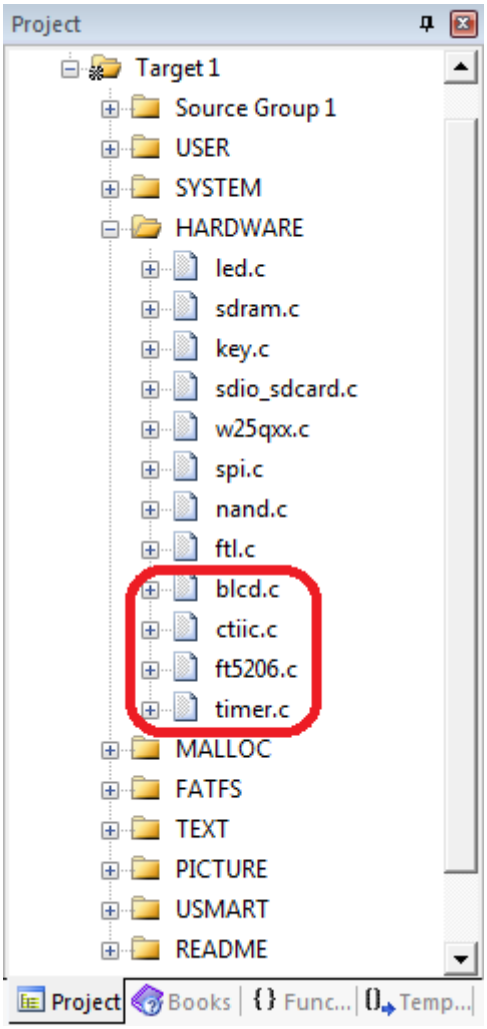

图 3.1 电容触摸屏实验工程截图

这里，我们主要看三个.c 文件里面的代码，首先是 blcd.c，该文件是 ATK-7’TFTLCD 模块 LCD 驱动器的驱动代码，blcd.c 里面的代码如下：

```
//LCD 的画笔颜色和背景色
u16 POINT_COLOR=0x0000;     //画笔颜色
u16 BACK_COLOR=0xFFFF;    //背景色
//管理 LCD 重要参数
//默认为竖屏
_lcd_dev lcddev;
//写寄存器函数
//regval:寄存器值
void LCD_WR_REG(vu16 regval)
{
    regval=regval;        //使用-O2 优化的时候,必须插入的延时
    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(vu16 data)
```

```
{
    data=data;                  //使用-O2 优化的时候,必须插入的延时
    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(u8 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(u8 LCD_Reg)
{
    LCD->LCD_REG = LCD_Reg;         //写入要写的寄存器序号
    return LCD->LCD_RAM;
}
//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
}
//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 t=0;
    LCD_SetCursor(x,y);
    while(t<0X1FFF)
    {
        if(LCD->LCD_REG&0x0001)break;
        t++;
    }
    return LCD->LCD_RAM;
}
//LCD 背光设置
//pwm:背光等级,0~63.越大越亮.
void LCD_BackLightSet(u8 pwm)
{
    lcddev.sysreg&=~0X003F;                    //清除之前的设置
    lcddev.sysreg|=pwm&0X3F;                   //设置新的值
    LCD_WriteReg(LCD_PREF,lcddev.sysreg); //写 LCD_PREF 寄存器
}
//扫描方向上,X 的终点坐标.
void LCD_EndXSet(u16 x)
{
    LCD_WriteReg(LCD_END_X,x);                 //设置 X 终坐标
}
//LCD 开启显示
void LCD_DisplayOn(void)
{
    LCD_WriteReg(LCD_PREF,lcddev.sysreg); //恢复 LCD_PREF 寄存器
}
//LCD 关闭显示
void LCD_DisplayOff(void)
{
    LCD_WriteReg(LCD_PREF,0);//关闭 TFT,相当于把背光关掉，无背光，无显示
}
//设置当前显示层
//layer:当前显示层
void LCD_SetDisplayLayer(u16 layer)
{
    lcddev.sysreg&=~0X0E00;                    //清除之前的设置
    lcddev.sysreg|=(layer&0X07)<<9;            //设置新的值
    LCD_WriteReg(LCD_PREF,lcddev.sysreg); //写 LCD_PREF 寄存器
}
//设置当前操作层
//layer:当前显示层
void LCD_SetOperateLayer(u16 layer)
{
```

```
        lcddev.sysreg&=~0X7000;                         //清除之前的设置
        lcddev.sysreg|=(layer&0X07)<<12;            //设置新的值
        LCD_WriteReg(LCD_PREF,lcddev.sysreg); //写 LCD_PREF 寄存器
}
//设置光标位置
//Xpos:横坐标
//Ypos:纵坐标
void LCD_SetCursor(u16 Xpos, u16 Ypos)
{
        LCD_WriteReg(lcddev.setycmd,Ypos); //设置 Y 坐标
        LCD_WriteReg(lcddev.setxcmd,Xpos); //设置 X 坐标
}
//设置 LCD 的自动扫描方向
//注意:我们的驱动器,只支持左右扫描设置,不支持上下扫描设置
void LCD_Scan_Dir(u8 dir)
{
        switch(dir)
        {
                case L2R_U2D:     //从左到右,从上到下
                        LCD_WriteReg(LCD_MIRROR,1);     //写 LCD_PREF 寄存器
                        break;
                case L2R_D2U:     //从左到右,从下到上
                        LCD_WriteReg(LCD_MIRROR,3);     //写 LCD_PREF 寄存器
                        break;
                case R2L_U2D:     //从右到左,从上到下
                        LCD_WriteReg(LCD_MIRROR,0);     //写 LCD_PREF 寄存器
                        break;
                case R2L_D2U:     //从右到左,从下到上
                        LCD_WriteReg(LCD_MIRROR,2);     //写 LCD_PREF 寄存器
                        break;
                default:          //其他,默认从左到右,从上到下
                        LCD_WriteReg(LCD_MIRROR,1);     //写 LCD_PREF 寄存器
                        break;
        }
}
//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;
}
```

```
//快速画点
//x,y:坐标
//color:颜色
void LCD_Fast_DrawPoint(u16 x,u16 y,u16 color)
{
	LCD->LCD_REG=lcddev.setycmd;
	LCD->LCD_RAM=y;
	LCD->LCD_REG=lcddev.setxcmd;
	LCD->LCD_RAM=x;
	LCD->LCD_REG=lcddev.wramcmd;
	LCD->LCD_RAM=color;
}
//设置 LCD 显示方向（7 寸屏,不能简单的修改为横屏显示）
//dir:0,竖屏；1,横屏
void LCD_Display_Dir(u8 dir)
{
}
//设置窗口,并自动设置画点坐标到窗口左上角(sx,sy).
//sx,sy:窗口起始坐标(左上角)
//width,height:窗口宽度和高度,必须大于 0!!
//窗体大小:width*height.
//68042,横屏时不支持窗口设置!!
void LCD_Set_Window(u16 sx,u16 sy,u16 width,u16 height)
{
	LCD_EndXSet(sx+width-1);
	LCD_SetCursor(sx,sy); //设置光标位置
}
//初始化 lcd
//该初始化函数可以初始化各种 ILI93XX 液晶,但是其他函数是基于 ILI9320 的!!!
//在其他型号的驱动芯片上没有测试!
void LCD_Init(void)
{
	u16 i=0;
	RCC->AHB1ENR|=1<<1;		//使能 PORTB 时钟
	RCC->AHB1ENR|=3<<3;		//使能 PD,PE
	RCC->AHB3ENR|=1<<0;		//使能 FMC 时钟
	GPIO_Set(GPIOB,PIN5,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_50M,
		GPIO_PUPD_PU);		//PB5 推挽输出,控制背光
	……//省略部分代码
	GPIO_AF_Set(GPIOE,15,12);		//PE15,AF12
	//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 NE1 ，也就对应 BTCR[0],[1]。
	FMC_Bank1->BTCR[0]=0X00000000;
```

```
    FMC_Bank1->BTCR[1]=0X00000000;
    FMC_Bank1E->BWTR[0]=0X00000000;
    //操作 BCR 寄存器    使用异步模式
    FMC_Bank1->BTCR[0]|=1<<12;      //存储器写使能
    FMC_Bank1->BTCR[0]|=1<<14;      //读写使用不同的时序
    FMC_Bank1->BTCR[0]|=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
    FMC_Bank1->BTCR[1]|=0<<28;      //模式 A
    FMC_Bank1->BTCR[1]|=8<<0;       //地址建立时间(ADDSET)为 8 个 HCLK(40ns)
    //因为液晶驱动 IC 的读数据的时候，速度不能太快,尤其是个别奇葩芯片。
    FMC_Bank1->BTCR[1]|=8<<8;       //数据保存时间(DATAST)为 8 个 HCLK(40ns)
    //写时序控制寄存器
    FMC_Bank1E->BWTR[0]|=0<<28;     //模式 A
    FMC_Bank1E->BWTR[0]|=8<<0;      //地址建立时间(ADDSET)为 8 个 HCLK=40ns
    FMC_Bank1E->BWTR[0]|=8<<8;      //数据保存时间(DATAST)为 40ns
    //使能 BANK1,区域 1
    FMC_Bank1->BTCR[0]|=1<<0;       //使能 BANK1，区域 1
    //LCD_RST=0;//阿波罗开发板的 RESET 和 LCD 的 RST 共用，并未用 IO 控制复位
    delay_ms(100);
    //LCD_RST=1;
    while(i<0X1FFF)
    {
        if(LCD_RD_DATA()&0x0001)break;//等待控制器准备好
        i++;
    }
    lcddev.setxcmd=LCD_CUR_X;   //设置写 X 坐标指令
    lcddev.setycmd=LCD_CUR_Y;   //设置写 Y 坐标指令
    lcddev.wramcmd=LCD_PIXELS; //设置写入 GRAM 的指令
    lcddev.width=800;           //设置宽度
    lcddev.height=480;          //设置高度
    LCD_Scan_Dir(L2R_U2D);      //设置默认扫描方向.
    LCD_SetDisplayLayer(0);     //显示层为 0
    LCD_SetOperateLayer(0);     //操作层也为 0
    LCD_EndXSet(799);           //x 终点坐标为 800
    LCD_BackLightSet(63);       //背光设置为最亮
    LCD_Clear(WHITE);           //清屏
}
//清屏函数
//color:要清屏的填充色
void LCD_Clear(u16 color)
{
    u32 index=0;
    u32 totalpoint=lcddev.width;
```

```
        totalpoint*=lcddev.height;     //得到总点数
        LCD_SetCursor(0x00,0x0000);    //设置光标位置
        LCD_WriteRAM_Prepare();        //开始写入 GRAM
        for(index=0;index<totalpoint;index++) LCD->LCD_RAM=color;
    }
    //在指定区域内填充单个颜色
    //(sx,sy),(ex,ey):填充矩形对角坐标,区域大小为:(ex-sx+1)*(ey-sy+1)
    //color:要填充的颜色
    void LCD_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 color)
    {
        u16 i,j;
        u16 xlen=0;
        xlen=ex-sx+1;
        for(i=sy;i<=ey;i++)
        {
            LCD_SetCursor(sx,i);                    //设置光标位置
            LCD_WriteRAM_Prepare();                 //开始写入 GRAM
            for(j=0;j<xlen;j++)LCD->LCD_RAM=color;  //送入 LCD
        }
    }
    ……//省略部分代码
```

以上代码，我们没有全部贴出，后续部分代码，同 lcd.c 里面的代码一模一样，blcd.c 里面的函数名和用法同 lcd.c 里面的一模一样，并且 blcd.c 所调用的头文件，也是 lcd.h，所以大家如果要修改某个实验支持 ATK-7’ TFTLCD 模块，只需要直接拿 blcd.c 替换掉原工程的 lcd.c 即可。

首先，我们看看 LCD_Init 函数，该函数同样是采用 FSMC（关于 FSMC 驱动 LCD 的介绍，请参考《STM32F429 开发指南》第十七章）来驱动 LCD，这里初始化的步骤，就是按我们在 1.4.1 接所介绍的步骤进行初始化，该函数执行完后，我们便可以对 LCD 进行正常的读写操作了。

其次，LCD_DrawPoint 函数，该函数用于画点，方法同我们在 1.4.2 接介绍的一模一样。同样，LCD_ReadPoint 用于读点，方法在 1.4.3 节已由介绍。

这两个函数都比较简单，效率也比较高，当然由于我们的 LCD 驱动器支持地址自增方式，所以在填充颜色的时候，有更高效的办法，如在 LCD_Clear 函数里面，我们就只需要设置 1 次坐标，然后填充 lcddev.width*lcddev.height 个数据就可以实现整屏数据填充了，大大提高了画点速度。

其他函数，我们就不一一介绍了，这里提醒大家，LCD_Display_Dir 是一个空函数，我们没有实现，因为 ATK-7’TFTLCD 模块不是很好做竖屏显示（如果一定要做竖屏，只能用坐标变换的办法，不过坐标自增就变成 Y 轴自增了），所以固定为横屏显示。

blcd.c 所使用的头文件也是 lcd.h，同我们 ALIENTEK 开发板标准例程保持最大限度的一致性，方便大家使用。

接下来，我们看看 ctiic.c 里面的内容，ctiic.c 用于实现和 FT5206 的 IIC 通信，该文件代码如下：

```
    //控制 I2C 速度的延时
    void CT_Delay(void)
```

```
{
    delay_us(2);
}
//电容触摸芯片 IIC 接口初始化
void CT_IIC_Init(void)
{
    RCC->AHB1ENR|=1<<7;                 //使能 PORTH 时钟
    RCC->AHB1ENR|=1<<8;                 //使能 PORTI 时钟
    GPIO_Set(GPIOH,PIN6,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_100M,
            GPIO_PUPD_PU);      //PH6 设置为推挽输出
    GPIO_Set(GPIOI,PIN3,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_100M,
            GPIO_PUPD_PU);      //PI3 设置为推挽输出
    CT_IIC_SDA=1;
    CT_IIC_SCL=1;
}
//产生 IIC 起始信号
void CT_IIC_Start(void)
{
    CT_SDA_OUT();       //sda 线输出
    CT_IIC_SDA=1;
    CT_IIC_SCL=1;
    delay_us(30);
    CT_IIC_SDA=0;//START:when CLK is high,DATA change form high to low
    CT_Delay();
    CT_IIC_SCL=0;//钳住 I2C 总线，准备发送或接收数据
}
//产生 IIC 停止信号
void CT_IIC_Stop(void)
{
    CT_SDA_OUT();//sda 线输出
    CT_IIC_SCL=1;
    delay_us(30);
    CT_IIC_SDA=0;//STOP:when CLK is high DATA change form low to high
    CT_Delay();
    CT_IIC_SDA=1;//发送 I2C 总线结束信号
}
//等待应答信号到来
//返回值：1，接收应答失败
//        0，接收应答成功
u8 CT_IIC_Wait_Ack(void)
{
    u8 ucErrTime=0;
    CT_SDA_IN();        //SDA 设置为输入
    CT_IIC_SDA=1;
```

```
        CT_IIC_SCL=1;
        CT_Delay();
        while(CT_READ_SDA)
        {
            ucErrTime++;
            if(ucErrTime>250)
            {
                CT_IIC_Stop();
                return 1;
            }
            CT_Delay();
        }
        CT_IIC_SCL=0;//时钟输出 0
        return 0;
}
//产生 ACK 应答
void CT_IIC_Ack(void)
{
        CT_IIC_SCL=0;
        CT_SDA_OUT();
        CT_Delay();
        CT_IIC_SDA=0;
        CT_Delay();
        CT_IIC_SCL=1;
        CT_Delay();
        CT_IIC_SCL=0;
}
//不产生 ACK 应答
void CT_IIC_NAck(void)
{
        CT_IIC_SCL=0;
        CT_SDA_OUT();
        CT_Delay();
        CT_IIC_SDA=1;
        CT_Delay();
        CT_IIC_SCL=1;
        CT_Delay();
        CT_IIC_SCL=0;
}
//IIC 发送一个字节
//返回从机有无应答
//1，有应答
//0，无应答
void CT_IIC_Send_Byte(u8 txd)
```

```
{
    u8 t;
    CT_SDA_OUT();
    CT_IIC_SCL=0;//拉低时钟开始数据传输
    CT_Delay();
    for(t=0;t<8;t++)
    {
        CT_IIC_SDA=(txd&0x80)>>7;
        txd<<=1;
        CT_IIC_SCL=1;
        CT_Delay();
        CT_IIC_SCL=0;
        CT_Delay();
    }
}
//读 1 个字节，ack=1 时，发送 ACK，ack=0，发送 nACK
u8 CT_IIC_Read_Byte(unsigned char ack)
{
    u8 i,receive=0;
    CT_SDA_IN();//SDA 设置为输入
    delay_us(30);
    for(i=0;i<8;i++ )
    {
        CT_IIC_SCL=0;
        CT_Delay();
        CT_IIC_SCL=1;
        receive<<=1;
        if(CT_READ_SDA)receive++;
    }
    if (!ack)CT_IIC_NAck();//发送 nACK
    else CT_IIC_Ack(); //发送 ACK
    return receive;
}
```

此部分代码实现与 FT5206 的 IIC 通信的底层操作，包括初始化，收发数据和应答等，这样外部可以调用这些底层函数，实现同 FT5206 的 IIC 通信。

接着，我们看看 ft5206.c 里面的内容：

```
ft_ctp_dev tp_dev=
{
    FT5206_Init,
    FT5206_Scan,
    0,
    0,
    0,
    0,
```

```
};

//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->AHB1ENR|=1<<7;          //使能 PORTH 时钟
        RCC->AHB1ENR|=1<<8;          //使能 PORTI 时钟
        GPIO_Set(GPIOH,PIN7,GPIO_MODE_IN,0,0,GPIO_PUPD_PU);  //PH7 上拉输入
        GPIO_Set(GPIOI,PIN8,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_100M,
                GPIO_PUPD_PU);      //PI8 设置为推挽输出
        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，不能小于 12，最大 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)||temp[1]==0X01||temp[1]==0X02)//5206/5426
        {
                printf("CTP ID:%x\r\n",((u16)temp[0]<<8)+temp[1]);
                return 0;
        }
        return 1;
}
//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,res=0,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)<6))
        {
            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
                    tp_dev.y[i]=((u16)(buf[0]&0X0F)<<8)+buf[1];
                    tp_dev.x[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
                }
            }
            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;
}
```

此部分代码，就是我们电容触摸屏驱动部分的核心代码了，其中 FT5206_Init 用于初始化电容触摸屏驱动芯片 FT5206，初始化步骤同 1.5.2 节介绍的一样，

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。

接下来打开 ft5206.h 文件，此文件里面的代码如下：

```
#define CT_MAX_TOUCH  5        //电容触摸屏最大支持的点数
#define TP_PRES_DOWN  0x80     //触屏被按下
#define TP_CATH_PRES  0x40     //有按键按下了
//电容触摸屏控制器
typedef struct
{
    u8    (*init)(void);       //初始化触摸屏控制器
    u8    (*scan)(u8);         //扫描触摸屏
    u16 x[CT_MAX_TOUCH];       //触摸 X 坐标
    u16 y[CT_MAX_TOUCH];       //触摸 Y 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:电容触摸屏按下的点数
}ft_ctp_dev;

extern ft_ctp_dev tp_dev;
//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                 PIout(8)     //FT5206 复位引脚
#define FT_INT                 PHin(7)      //FT5206 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);
```

该文件里面，我们重点看看 ft_ctp_dev 结构体，该结构体带 2 个函数参数：init 和 scan。

分别用于 FT5206 的初始化和触摸屏数据扫描（读取）。然后 x、y 分别用于保存触摸点的 x、y 坐标。最后 sta 用于保存触摸状态，标记有效触摸点。

最后我们修改 test.c 里面的内容如下：

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//01，电容触摸屏测试部分
//电容触摸测试 gui
void ctouch_paint_gui(void)
{
    LCD_Clear(WHITE);//清屏
    POINT_COLOR=BLUE;//设置字体为蓝色
    LCD_ShowString(lcddev.width-24,0,lcddev.width,16,16,"RST");//显示清屏区域
    POINT_COLOR=RED;//设置画笔蓝色
}
//画水平线
//x0,y0:坐标
//len:线长度
//color:颜色
void gui_draw_hline(u16 x0,u16 y0,u16 len,u16 color)
{
    if(len==0)return;
    LCD_Fill(x0,y0,x0+len-1,y0,color);
}
//画实心圆
//x0,y0:坐标
//r:半径
//color:颜色
void gui_fill_circle(u16 x0,u16 y0,u16 r,u16 color)
{
    u32 i;
    u32 imax = ((u32)r*707)/1000+1;
    u32 sqmax = (u32)r*(u32)r+(u32)r/2;
    u32 x=r;
    gui_draw_hline(x0-r,y0,2*r,color);
    for (i=1;i<=imax;i++)
    {
        if ((i*i+x*x)>sqmax)// draw lines from outside
        {
            if (x>imax)
            {
                gui_draw_hline (x0-i+1,y0+x,2*(i-1),color);
                gui_draw_hline (x0-i+1,y0-x,2*(i-1),color);
            }
            x--;
        }
```

```
            // draw lines from inside (center)
            gui_draw_hline(x0-x,y0+i,2*x,color);
            gui_draw_hline(x0-x,y0-i,2*x,color);
        }
    }
    //两个数之差的绝对值
    //x1,x2：需取差值的两个数
    //返回值：|x1-x2|
    u16 my_abs(u16 x1,u16 x2)
    {
        if(x1>x2)return x1-x2;
        else return x2-x1;
    }
    //画一条粗线
    //(x1,y1),(x2,y2):线条的起始坐标
    //size：线条的粗细程度
    //color：线条的颜色
    void lcd_draw_bline(u16 x1, u16 y1, u16 x2, u16 y2,u8 size,u16 color)
    {
        u16 t;
        int xerr=0,yerr=0,delta_x,delta_y,distance;
        int incx,incy,uRow,uCol;
        if(x1<size|| x2<size||y1<size|| y2<size)return;
        delta_x=x2-x1; //计算坐标增量
        delta_y=y2-y1;
        uRow=x1;
        uCol=y1;
        if(delta_x>0)incx=1; //设置单步方向
        else if(delta_x==0)incx=0;//垂直线
        else {incx=-1;delta_x=-delta_x;}
        if(delta_y>0)incy=1;
        else if(delta_y==0)incy=0;//水平线
        else{incy=-1;delta_y=-delta_y;}
        if( delta_x>delta_y)distance=delta_x; //选取基本增量坐标轴
        else distance=delta_y;
        for(t=0;t<=distance+1;t++ )//画线输出
        {
            gui_fill_circle(uRow,uCol,size,color);//画点
            xerr+=delta_x ;
            yerr+=delta_y ;
            if(xerr>distance)
            {
                xerr-=distance;
                uRow+=incx;
```

```
            }
            if(yerr>distance)
            {
                yerr-=distance;
                uCol+=incy;
            }
        }
    }
    //5 个触控点的颜色
    const u16 POINT_COLOR_TBL[5]={RED,GREEN,BLUE,BROWN,GRED};
    //01,电容触摸屏测试
    //测试电容触摸屏，最大支持 5 点触控。
    void ctouch_paint_test(void)
    {
        u8 i=0,t=0;
        u16 lastpos[5][2];              //最后一次的数据
        u8  ctout[5];                   //5 个触摸点的释放计时器
        LCD_Clear(WHITE);
        POINT_COLOR=RED;        //设置字体为红色
        Show_Str(60,50,lcddev.width,16,"测试 1：电容触摸屏测试",16,0);
        Show_Str(60,70,lcddev.width,16,"最大同时触摸点数：5 点",16,0);
        Show_Str(60,90,lcddev.width,16,"清屏：点击右上角的的‘RST’\
                                    可以清除整个屏幕",16,0);
        while(tp_dev.init())            //初始化电容触摸屏
        {
            Show_Str(60,110,lcddev.width,16,"电容触摸屏初始化失败！",16,0);
            delay_ms(200);
            Show_Str(60,110,lcddev.width,16,"          请检查!!!        ",16,0);
            delay_ms(200);
        };
        Show_Str(60,110,lcddev.width,16,"电容触摸屏初始化成功！",16,0);
        delay_ms(1500);
        delay_ms(1500);
        ctouch_paint_gui();
        for(i=0;i<5;i++)
        {
            lastpos[i][0]=0XFFFF;   //全部设置为非法值
            lastpos[i][1]=0XFFFF;
            ctout[i]=0;                     //计时器清零
        }
        while(1)
        {
            tp_dev.scan(0);
            if(tp_dev.sta&TP_PRES_DOWN)       //触摸屏被按下
```

```
{
    for(t=0;t<5;t++)
    {
        if(tp_dev.sta&(1<<t))
        {
            if(tp_dev.x[t]<lcddev.width&&tp_dev.y[t]<lcddev.height)
            {
                if(tp_dev.x[t]>(lcddev.width-24)&&tp_dev.y[t]<16)\
                  ctouch_paint_gui();//清除
                else
                {
                    if(lastpos[t][0]==0XFFFF)//属于第一次按下
                    {
                        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],3,POINT_COLOR_TBL[t]);//画线
                    lastpos[t][0]=tp_dev.x[t];
                    lastpos[t][1]=tp_dev.y[t];
                }
            }
            ctout[t]=0;
        }
    }
    tp_dev.sta=0;
}else
{
    delay_ms(5);  //没有按键按下的时候
    for(t=0;t<5;t++)
    {
        ctout[t]++;
        if(ctout[t]>10)//判定此点以松开
        {
            lastpos[t][0]=0XFFFF;
            ctout[t]=0;
        }
    }
}
i++;
if(i==20)
{
    i=0;
    LED0=!LED0;
```

```
            }
        }
    }
    ////////////////////////////////////////////////////////////////////////////////////////////////
    //02,图片显示测试部分

    //得到 path 路径下,目标文件的总个数
    //path:路径
    //返回值:总有效文件数
    u16 pic_get_tnum(u8 *path)
    {
        u8 res;
        u16 rval=0;
        DIR tdir;              //临时目录
        FILINFO *tfileinfo;    //临时文件信息
        tfileinfo=(FILINFO*)mymalloc(SRAMIN,sizeof(FILINFO));//申请内存
        res=f_opendir(&tdir,(const TCHAR*)path);      //打开目录
        if(res==FR_OK&&tfileinfo)
        {
            while(1)//查询总的有效文件数
            {
                res=f_readdir(&tdir,tfileinfo);                 //读取目录下的一个文件
                if(res!=FR_OK||tfileinfo->fname[0]==0)break;//错误了/到末尾了,退出
                res=f_typetell((u8*)tfileinfo->fname);
                if((res&0XF0)==0X50)//取高四位,看看是不是图片文件
                {
                    rval++;//有效文件数增加 1
                }
            }
        }
        myfree(SRAMIN,tfileinfo);//释放内存
        return rval;
    }
    //02,图片显示测试
    //循环显示 SD 卡，PICTURE 文件夹下面的图片文件。
    void picture_display_test(void)
    {
        u8 res;
        DIR picdir;             //图片目录
        FILINFO *picfileinfo;   //文件信息
        u8 *pname;              //带路径的文件名
        u16 totpicnum;          //图片文件总数
        u16 curindex;           //图片当前索引
        u8 key;                 //键值
```

```
u8 pause=0;                 //暂停标记
u8 t;
u32 temp;
u32 *picoffsettbl;   //图片索引表
LCD_Clear(WHITE);
Show_Str(60,50,lcddev.width,16,"测试 2：图片显示测试",16,0);
Show_Str(60,70,lcddev.width,16,"KEY0:下一张图片",16,0);
Show_Str(60,90,lcddev.width,16,"KEY1:上一张图片",16,0);
Show_Str(60,110,lcddev.width,16,"KEY_UP:暂停/继续 自动播放",16,0);
while(f_opendir(&picdir,"0:/PICTURE"))//打开图片文件夹
{
    Show_Str(60,130,lcddev.width,16,"PICTURE 文件夹错误!",16,0);
    delay_ms(200);
    LCD_Fill(60,130,lcddev.width,130+16,WHITE);//清除显示
    delay_ms(200);
}
totpicnum=pic_get_tnum("0:/PICTURE"); //得到总有效文件数
while(totpicnum==NULL)//图片文件为 0
{
    Show_Str(60,130,lcddev.width,16,"没有图片文件!",16,0);
    delay_ms(200);
    LCD_Fill(60,130,lcddev.width,130+16,WHITE);//清除显示
    delay_ms(200);
}
picfileinfo=(FILINFO*)mymalloc(SRAMIN,sizeof(FILINFO)); //申请内存
pname=mymalloc(SRAMIN,_MAX_LFN*2+1); //为带路径的文件名分配内存
picoffsettbl=mymalloc(SRAMIN,4*totpicnum);//申请 4*totpicnum 个字节的内存
while(picfileinfo==NULL||pname==NULL||picoffsettbl==NULL)//内存分配出错
{
    Show_Str(60,130,lcddev.width,16,"内存分配失败!",16,0);
    delay_ms(200);
    LCD_Fill(60,130,lcddev.width,130+16,WHITE);//清除显示
    delay_ms(200);
}
//记录索引
res=f_opendir(&picdir,"0:/PICTURE"); //打开目录
if(res==FR_OK)
{
    curindex=0;//当前索引为 0
    while(1)//全部查询一遍
    {
        temp=picdir.dptr;                                   //记录当前 dptr 偏移
        res=f_readdir(&picdir,picfileinfo);                 //读取目录下的一个文件
        if(res!=FR_OK||picfileinfo->fname[0]==0)break; //错误了/到末尾了,退出
```

```
                res=f_typetell((u8*)picfileinfo->fname);
                if((res&0XF0)==0X50)//取高四位,看看是不是图片文件
                {
                    picoffsettbl[curindex]=temp;//记录索引
                    curindex++;
                }
            }
        }
        delay_ms(1200);
        Show_Str(60,130,lcddev.width,16,"开始显示...",16,0);
        delay_ms(1800);
        piclib_init();                                          //初始化画图
        curindex=0;                                             //从 0 开始显示
        res=f_opendir(&picdir,(const TCHAR*)"0:/PICTURE");      //打开目录
        while(res==FR_OK)//打开成功
        {
            dir_sdi(&picdir,picoffsettbl[curindex]);            //改变当前目录索引
            res=f_readdir(&picdir,picfileinfo);                 //读取目录下的一个文件
            if(res!=FR_OK||picfileinfo->fname[0]==0)break; //错误了/到末尾了,退出
            strcpy((char*)pname,"0:/PICTURE/");                 //复制路径(目录)
            strcat((char*)pname,(const char*)picfileinfo->fname);//将文件名接在后面
            LCD_Clear(BLACK);
            ai_load_picfile(pname,0,0,lcddev.width,lcddev.height,1);//显示图片
            Show_Str(2,2,lcddev.width,16,pname,16,1);           //显示图片名字
            t=0;
            while(1)
            {
                key=KEY_Scan(0);                //扫描按键
                if(t>250&&(pause==0))key=KEY0_PRES;//非暂停，2.5 秒，模拟按下 KEY0
                if(key==KEY1_PRES)              //上一张
                {
                    if(curindex)curindex--;
                    else curindex=totpicnum-1;
                    break;
                }else if(key==KEY0_PRES)        //下一张
                {
                    curindex++;
                    if(curindex>=totpicnum)curindex=0;//到末尾的时候,自动从头开始
                    break;
                }else if(key==WKUP_PRES)
                {
                    pause=!pause;
                    LED1=!pause;        //暂停的时候 LED1 亮.
                }
```

```
                t++;
                if((t%20)==0)LED0=!LED0;
                delay_ms(10);
            }
            res=0;
        }
        myfree(SRAMIN,picfileinfo);             //释放内存
        myfree(SRAMIN,pname);                   //释放内存
        myfree(SRAMIN,picoffsettbl);    //释放内存
}
////////////////////////////////////////////////////////////////////////////////////////////////////
//03,液晶屏自测试
//得到一个随机数
//(min,max)期望的随机数范围
//返回值:符合期望的随机数值
u16 speed_test_get_rval(u16 min,u16 max)
{
        u16 t=0Xffff;
        while((t<min)||(t>max))t=rand();
        return t;
}
const u16 SPEED_COLOR_TBL[10]={RED,GREEN,BLUE,BROWN,GRED,BRRED,
CYAN,YELLOW,GRAY,MAGENTA};
//得到速度测试一次填充的相关信息
//*x,*y,*width,*height,*color:获取到的填充坐标/尺寸/颜色等信息
void speed_test_get_fill_parameter(u16 *x,u16 *y,u16 *width,u16 *height,u16 *color)
{
        *x=speed_test_get_rval(0,700);
        *y=speed_test_get_rval(0,380);
        *width=speed_test_get_rval(80,800-*x);
        *height=speed_test_get_rval(80,480-*y);
        *color=SPEED_COLOR_TBL[speed_test_get_rval(0,9)];
}
u16 speed_run_time;             //速度测试测试时间长度(单位为 10ms)
//3-1 速度测试
void tftlcd_speed_test(void)
{
        u16 x,y,width,height,color;
        u32 pixelcnt=0;
        u8 *str;
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
        Show_Str(60,50,lcddev.width,16,"测试 3-1：速度测试",16,0);
        Show_Str(60,70,lcddev.width,16,"通过在 LCD 各个位置随机的填充不同尺寸的单色
```

```
                                        矩形,来测试速度",16,0);
    Show_Str(60,90,lcddev.width,16,"测试时长约为 5 秒,测试结果将显示在 LCD
                                        上",16,0);
    delay_ms(1500);delay_ms(1500);
    LCD_Clear(RED);
    str=mymalloc(SRAMIN,60); //分配 60 个字节的内存
    srand(TIM3->CNT);
    TIM3->CR1&=~(1<<0);       //关闭定时器 3
    TIM3->CNT=0;              //清零
    speed_run_time=0;         //计时器清零
    while(1)
    {
        speed_test_get_fill_parameter(&x,&y,&width,&height,&color);//得到各参数
        pixelcnt+=width*height;
        TIM3->CR1|=1<<0;          //开启定时器 3
        LCD_Fill(x,y,x+width-1,y+height-1,color);
        TIM3->CR1&=~(1<<0);           //关闭定时器 3
        if(speed_run_time>300)break;//大于 3 秒钟
    }
    sprintf((char*)str,"LCD 速度测试结果:%d 像素/秒",(pixelcnt*100)/speed_run_time);
    Show_Str(270,230,lcddev.width,16,str,16,0);
    myfree(SRAMIN,str);       //释放内存
    delay_ms(1800);delay_ms(1800);delay_ms(1400);
}
//3-2 镜像测试
void tftlcd_mirror_test(void)
{
    LCD_Clear(WHITE);
    Show_Str(60,50,lcddev.width,16,"测试 3-2：镜像测试",16,0);
    Show_Str(60,70,lcddev.width,16,"测试步骤:默认显示/上下调转/左右调转/上下左右
                                        都调转",16,0);
    delay_ms(1500);delay_ms(1500);
    LCD_Clear(WHITE);
    POINT_COLOR=RED;
    LCD_DrawRectangle(30,30,180,130);
    LCD_Fill(lcddev.width-30-150,lcddev.height-30-100,lcddev.width-30,lcddev.height-30,
            BLUE);
    Draw_Circle(90,lcddev.height-90,60);
    gui_fill_circle(lcddev.width-90,90,60,GRED);
    POINT_COLOR=BLUE;
    Show_Str(140,150,lcddev.width,16,"ATK-7' TFTLCD 模块镜像功能测试",16,0);
    LCD_Scan_Dir(L2R_U2D);//默认
    delay_ms(1200);delay_ms(1300);
    LCD_Scan_Dir(L2R_D2U);//上下调转
```

```
        delay_ms(1200);delay_ms(1300);
        LCD_Scan_Dir(R2L_U2D);//左右调转
        delay_ms(1200);delay_ms(1300);
        LCD_Scan_Dir(R2L_D2U);//上下左右都调转
        delay_ms(1200);delay_ms(1300);
        LCD_Scan_Dir(L2R_U2D);//恢复默认设置
    }
    //3-3 多缓存测试
    void tftlcd_multi_gram_test(void)
    {
        u8 i;
        u8 *str;
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
        Show_Str(60,50,lcddev.width,16,"测试 3-3：多缓存测试",16,0);     //
        Show_Str(60,70,lcddev.width,16,"ATK-7' TFTLCD 模块拥有多达 8 页 LCD 缓存,本测
                                        试将测试全部 8 页缓存",16,0);
        delay_ms(1500);delay_ms(1500);
        str=mymalloc(SRAMIN,60);              //分配 60 个字节的内存
        for(i=0;i<8;i++)
        {
            LCD_SetOperateLayer(i);           //设置当前操作缓存
            LCD_SetDisplayLayer(i);           //设置当前显示缓存
            LCD_Clear(SPEED_COLOR_TBL[i]);
            sprintf((char*)str,"我是第%d 页缓存",i);
            POINT_COLOR=BLACK;
            Show_Str(360,230,lcddev.width,16,str,16,0);
            delay_ms(1200);delay_ms(1300);
        }
        myfree(SRAMIN,str);          //释放内存
        //恢复默认设置
        LCD_SetOperateLayer(0);
        LCD_SetDisplayLayer(0);
    }
    //3-4 背光测试
    void tftlcd_backlight_test(void)
    {
        u8 i;
        u8 *str;
        float bkl=0;
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
        Show_Str(60,50,lcddev.width,16,"测试 3-4：背光测试",16,0);
        Show_Str(60,70,lcddev.width,16,"ATK-7' TFTLCD 模块自带背光控制功能,只需发送
```

```
                                        相关指令即可设置背光亮度",16,0);
        delay_ms(1500);delay_ms(1500);
        str=mymalloc(SRAMIN,60);            //分配 60 个字节的内存
        for(i=0;i<8;i++)
        {
            LCD_BackLightSet(i*8+7);  //背光亮度设置
            bkl=(float)(i+1)*8/64;
            sprintf((char*)str,"当前背光亮度:%3.1f%%",bkl*100);
            POINT_COLOR=BLUE;
            Show_Str(330,230,lcddev.width,16,str,16,0);
            delay_ms(1200);delay_ms(1300);
        }
        myfree(SRAMIN,str);         //释放内存
    }
    //03,液晶自测试
    //速度测试/镜像测试/多缓存测试/背光测试,这几个循环进行测试
    void tftlcd_self_test(void)
    {
        while(1)
        {
            tftlcd_speed_test();
            tftlcd_mirror_test();
            tftlcd_multi_gram_test();
            tftlcd_backlight_test();
        }
    }
    int main(void)
    {
        u8 key;
        u8 t=0;
        Stm32_Clock_Init(360,25,2,8);     //设置时钟,180Mhz
        delay_init(180);                  //初始化延时函数
        uart_init(90,115200);             //初始化串口波特率为 115200
        usmart_dev.init(90);
        LED_Init();                       //初始化与 LED 连接的硬件接口
        SDRAM_Init();                     //初始化 SDRAM
        LCD_Init();                       //初始化 LCD
        KEY_Init();                       //初始化按键
        W25QXX_Init();                    //初始化 W25Q256
        TIM3_Int_Init(100-1,9000-1);//10Khz 的计数频率，计数 100 次为 10ms
        my_mem_init(SRAMIN);              //初始化内部内存池
        my_mem_init(SRAMEX);              //初始化外部内存池
        my_mem_init(SRAMCCM);             //初始化 CCM 内存池
        exfuns_init();                    //为 fatfs 相关变量申请内存
```

```
f_mount(fs[0],"0:",1);          //挂载 SD 卡
f_mount(fs[1],"1:",1);          //挂载 FLASH.
f_mount(fs[2],"2:",1);          //挂载 NAND FLASH.
POINT_COLOR=RED;
while(font_init())          //检查字库
{
    POINT_COLOR=RED;
    LCD_Clear(WHITE);
    LCD_ShowString(60,50,lcddev.width,16,16,"ALIENTEK STM32");
    LCD_ShowString(60,70,lcddev.width,16,16,"Font Updating...");
    while(update_font(60,90,16,0)!=0)//字体更新出错
    {
        LCD_ShowString(60,90,lcddev.width,16,16," Font Update error! ");
        delay_ms(200);
        LCD_ShowString(60,90,lcddev.width,16,16,"    Please Check....    ");
        delay_ms(200);
        LED0=!LED0;
    };
    LCD_Clear(WHITE);
}
Show_Str(60,50,lcddev.width,16,"ALIENTEK ATK-7'电容触摸屏测试实验",16,0);
Show_Str(60,70,lcddev.width,16,"请选择测试模式：",16,0);
POINT_COLOR=BLUE;
Show_Str(60,90,lcddev.width,16, "KEY0：电容触摸屏测试（支持 5 点触控）",16,0);
Show_Str(60,110,lcddev.width,16,"KEY1：图片显示测试（需要 SD 卡支持）",16,0);
Show_Str(60,130,lcddev.width,16,"KEY_UP:自测试(速度/镜像/多缓存/背光)",16,0);
POINT_COLOR=RED;
Show_Str(60,170,lcddev.width,16,"广州市星翼电子科技有限公司",16,0);
Show_Str(60,190,lcddev.width,16,"官方网站：www.alientek.com",16,0);
Show_Str(60,210,lcddev.width,16,"开源电子网论坛：www.openedv.com",16,0);
Show_Str(60,230,lcddev.width,16,"电话(传真)：020-38271790",16,0);
Show_Str(60,250,lcddev.width,16,"2014 年 4 月 1 日",16,0);
while(1)
{
    key=KEY_Scan(0);
    switch(key)
    {
        case KEY0_PRES://KEY0 按下,电容触摸测试
            ctouch_paint_test();
            break;
        case KEY1_PRES://KEY1 按下,图片显示测试
            picture_display_test();
            break;
        case WKUP_PRES://KEY_UP 按下,液晶自测试
```

```
                tftlcd_self_test();
                break;
            }
            t++;
            if(t==20)
            {
                t=0;
                LED0=!LED0;
                if(LED0)Show_Str(60,70,lcddev.width,16,"请选择测试模式：",16,0);
                else LCD_Fill(60,70,60+128,70+16,WHITE);
            }else delay_ms(10);
        }
    }
```

此部分代码比较多，我们挑几个重要的函数进行讲解一下。首先，是第一项测试的主函数 ctouch_paint_test，用于电容触摸屏的测试，该函数对电容触摸屏进行初始化以后，就进入死循环，轮询触摸屏数据，在得到有效数据后，就在屏幕上画线，支持 5 点同时画线，并可以通过点击屏幕右上角的“RST”实现清屏。

第二个函数是 picture_display_test，该函数是第二项测试的主函数，该函数读取 SD 卡 PICTURE 目录下的图片文件（bmp/jpeg/gif 等），并在 LCD 屏幕上显示。这里的代码同《STM32F429 开发指南》第四十八章 图片显示实验对应部分的代码基本一致，实现功能也一样。

第三个函数是 tftlcd_speed_test，该函数实现第三项测试的第一个小项，用于 LCD 速度测试，该函数随机生成矩形，并通过 LCD_Fill 函数在 ATK-7’TFTLCD 模块上面填充矩形，通过定时器统计时间，最终计算像素填充速度，最终结果将显示在 LCD 屏上。

第四个函数是 tftlcd_mirror_test，该函数实现第三项测试的第二个小项，用于测试 LCD 的镜像功能，展示不同镜像设置所得到的显示效果。

第五个函数是 tftlcd_multi_gram_test，该函数实现第三项测试的第三个小项，用于测试 ATK’TFTLCD 模块 LCD 驱动器的多页显存，在每页显存用一个不同的颜色填充，并文字提示当前显存页。

第六个函数是 tftlcd_backlight_test，该函数实现第三项测试的第四个小项，用于测试 LCD 驱动器的背光控制功能，该函数通过 LCD_BackLightSet 函数设置 LCD 的背光，并将当前 LCD 背光的亮度设置值显示在 LCD 上。通过该项测试，大家可以看到不同亮度条件下，LCD 的显示效果。

最后，是 main 函数，该函数初始化各外设，最后进入死循环，等待用户输入，选择对应的测试功能，当用户按下不同的按键（KEY0/KEY1/KEY_UP）后，进入不同的测试函数，进行对应的测试项。

另外，我们在 USMART 里面加入了很多 LCD 相关的测试函数，包括屏幕截图函数 bmp_encode 等，通过这些函数，我们可以很方便的实现对 ATK-7’TFTLCD 模块的全功能测试。

至此，软件实现部分就介绍完了，我们接下来看代码验证。

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（这里，我们通转接板+延长线的方式连接 ATK-7’TFTLCD 模块和开发板，方便测试），LCD 显示如图 4.1 所示

界面：

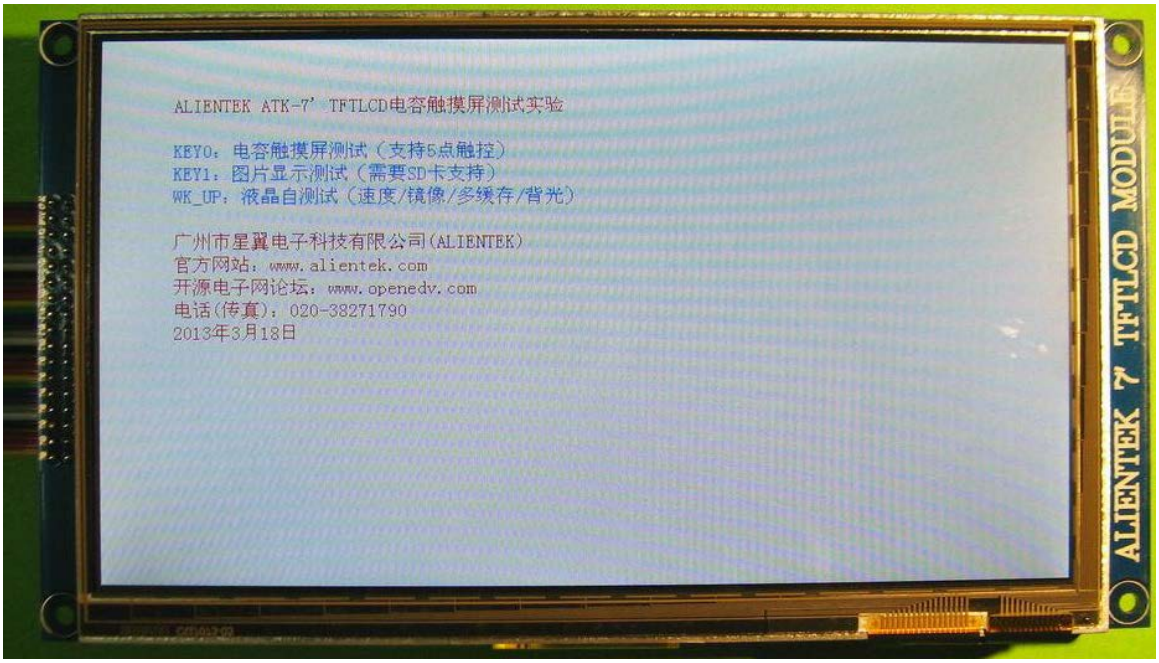

图 4.1 本测试实验界面

此时 LCD 正常显示（注意，如果是 MiniSTM32 开发板用户，则需要先插入 SD 卡更新字库），并提示通过按键选择相应测试项。我们通过 KEY0/KEY1/KEY_UP 这三个按键，即可选择不同的测试项目，进行测试。

首先看电容触摸屏测试。按 KEY0，进入该项测试，然后我们就可以在屏幕上面用手指写字了，如图 4.2,4.3 所示：

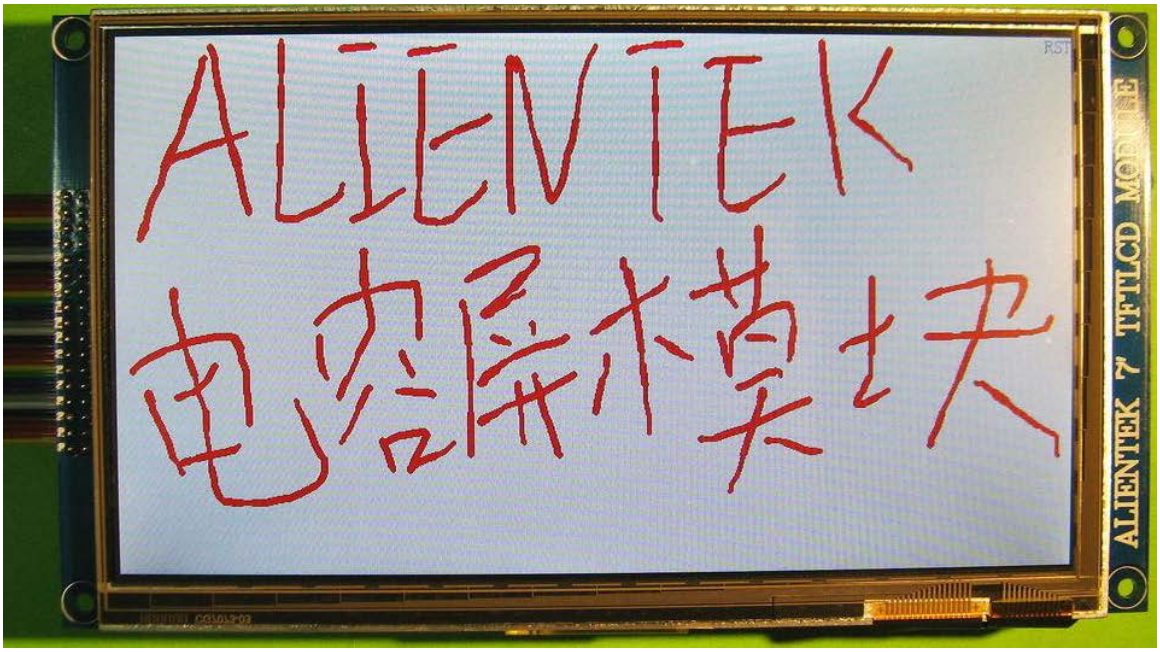

图 4.2 电容触摸输入

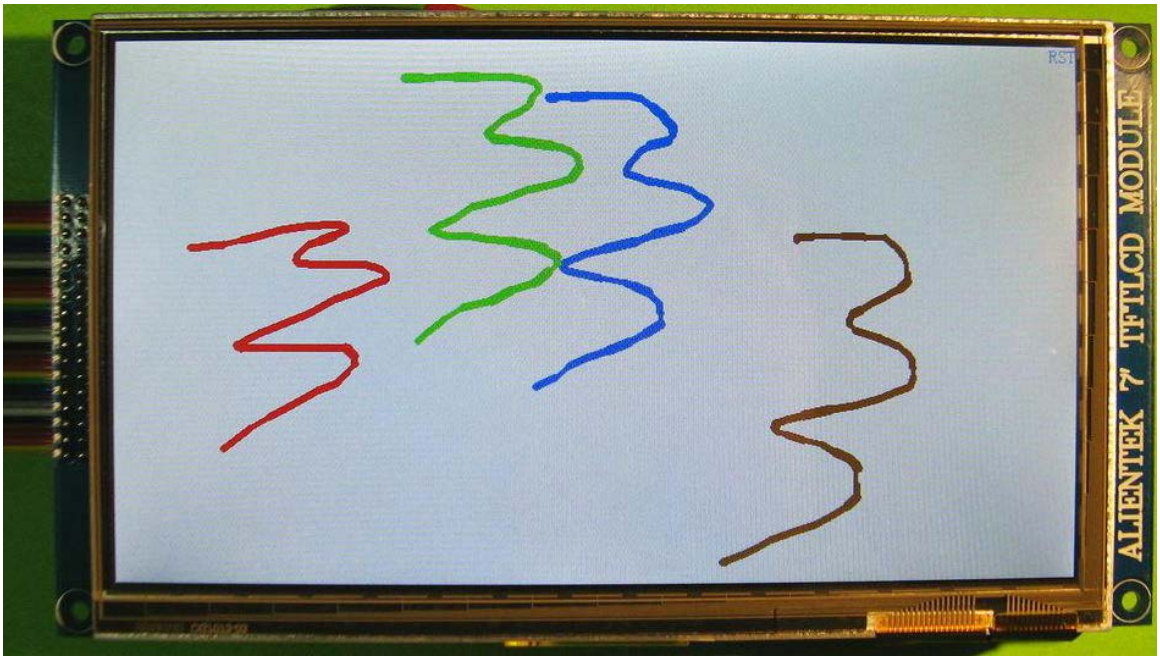

图 4.3 多点触摸（4 点）

通过按右上角的“RST”，可以实现清屏。

然后，我们复位，一下，回到默认状态，再按 KEY1，进入图片显示测试。该项测试要求必须有 SD 卡，且 SD 卡根目录存放 PICTURE 文件夹，并在 PICTURE 文件夹内放入一些图片文件（jpeg/bmp/gif）。测试效果如图 4.4 所示：

图 4.4 图片显示测试效果

我们再按一次复位，然后按 KEY_UP，进入液晶自测试功能。该项测试又分为 4 个小项：速度测试、镜像测试、多缓存测试和背光测试。分别如图 4.5~4.8 所示：

图 4.5 速度测试

图 4.6 镜像测试

图 4.7 多缓存测试

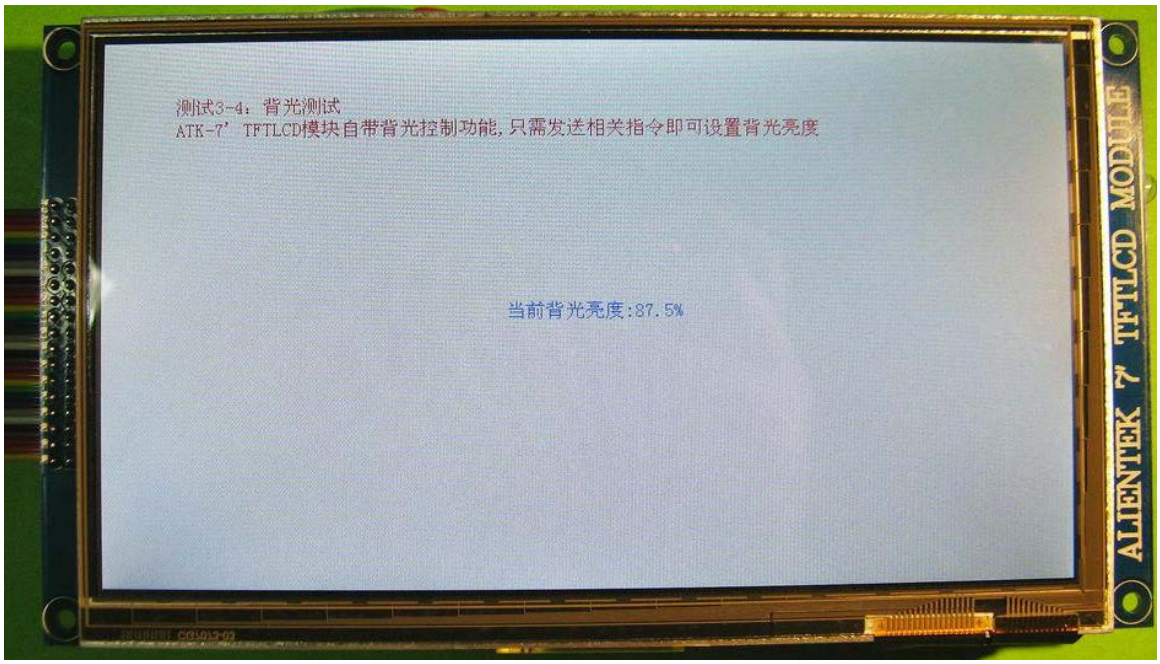

图 4.8 背光测试

最后，我们可以通过 USMART 调用相关函数，实现更全面的测试，另外，还可以通过 bmp_encode 函数，实现屏幕截图（前提是开发板要插入 SD 卡！）。 屏幕截图效果如图 4.9 所示：

图 4.9 速度测试截图

至此，关于 ATK-7’TFTLCD 电容触摸屏模块的介绍，我们就讲完了，本文档详细介绍了 ATK-7’TFTLCD 模块的使用，有助于大家快速学会 ATK-7’TFTLCD 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