

第 32 章 LCD12864 液晶显示实验

上一章我们学习了如何在 LCD1602 显示字符数据，我们知道 LCD1602 是字符型显示器，它不能显示汉字图形等。对于需要显示汉字或图形的项目中，LCD1602 不能实现，因此本章就来介绍一个可以实现字符、汉字、图形等显示的液晶屏——LCD12864。LCD12864 分为带字库和不带字库两种，开发板配置的是不带字库 LCD12864。我们开发板上集成了一个 LCD12864 液晶显示器接口，它可兼容带字库和不带字库两种屏，将配置的 LCD12864 液晶对应插入即可。本章要实现的功能是：系统运行时，LCD12864 上显示汉字字符信息。学习本章可以参考前面的实验章节内容。若结合视频学习效果更佳。本章分为如下几部分内容：

- 32.1 LCD12864 介绍
- 32.2 硬件设计
- 32.3 软件设计
- 32.4 实验现象

32.1 LCD12864 介绍

32.1.1 LCD12864 简介

LCD12864 液晶屏结构上与 LCD1602 一样，只是在行列数与现实像素上区别很大。LCD12864，以下简称 12864，注意区分 LCD1602 和 LCD12864。12864 是 64 行 128 列，当然也有可能会设计成 64 列 128 行，这里的行列不像 1602 那样，1602 是按照八行四列标准英文字符格式，以一行十六个字符，两列字符命名，而 12864 是以 128 列像素，64 行像素，也就是有 128×64 个像素点组成。就好比是 128 列 64 行的点阵。需要一行一列的去显示像素点。

常用的 12864 分为带字库和不带字库两种。以下分别介绍：

1、带字库 LCD12864



上图左侧为带字库 12864 显示字符和汉字，右侧为图形模式显示。注意：这里图形模式虽然显示了字符和汉字，但是并不是使用字库里的，显示的方法也是多样化的。

对于带字库 LCD12864，最常见的标志就是在屏幕背后，会有存放字库的芯片。如下图所示：



2、不带字库 LCD12864



这种不带字库的 LCD12864 背面没有芯片，一般为转接板，将 12864 显示屏转接到开发板。这种显示屏的操作就像带字库的 12864 操作图片形式一样。所有的显示都需要取模，取模方法和点阵取模一样。我们开发板所配置的是不带字库的 LCD12864，从外形上看其体积比带字库 LCD12864 要小很多，因此我们称之为 MiniLCD12864。

32.1.2 MiniLCD12864 介绍

开发板上配置的 MiniLCD12864 是不带字库的，要想显示汉字或其他字符需通过取字模方式来实现。要让其显示首先还得初始化，这个和 LCD1602 原理类似，即通过对内部一些特殊寄存器设置实现特定功能。我们配置的 MiniLCD12864 内部驱动芯片是 ST7565P，其数据手册以及转换板原理图在实验例程“\5--实验程序\基础实验例程\实验 30 LCD12864 液晶”内可以查看。

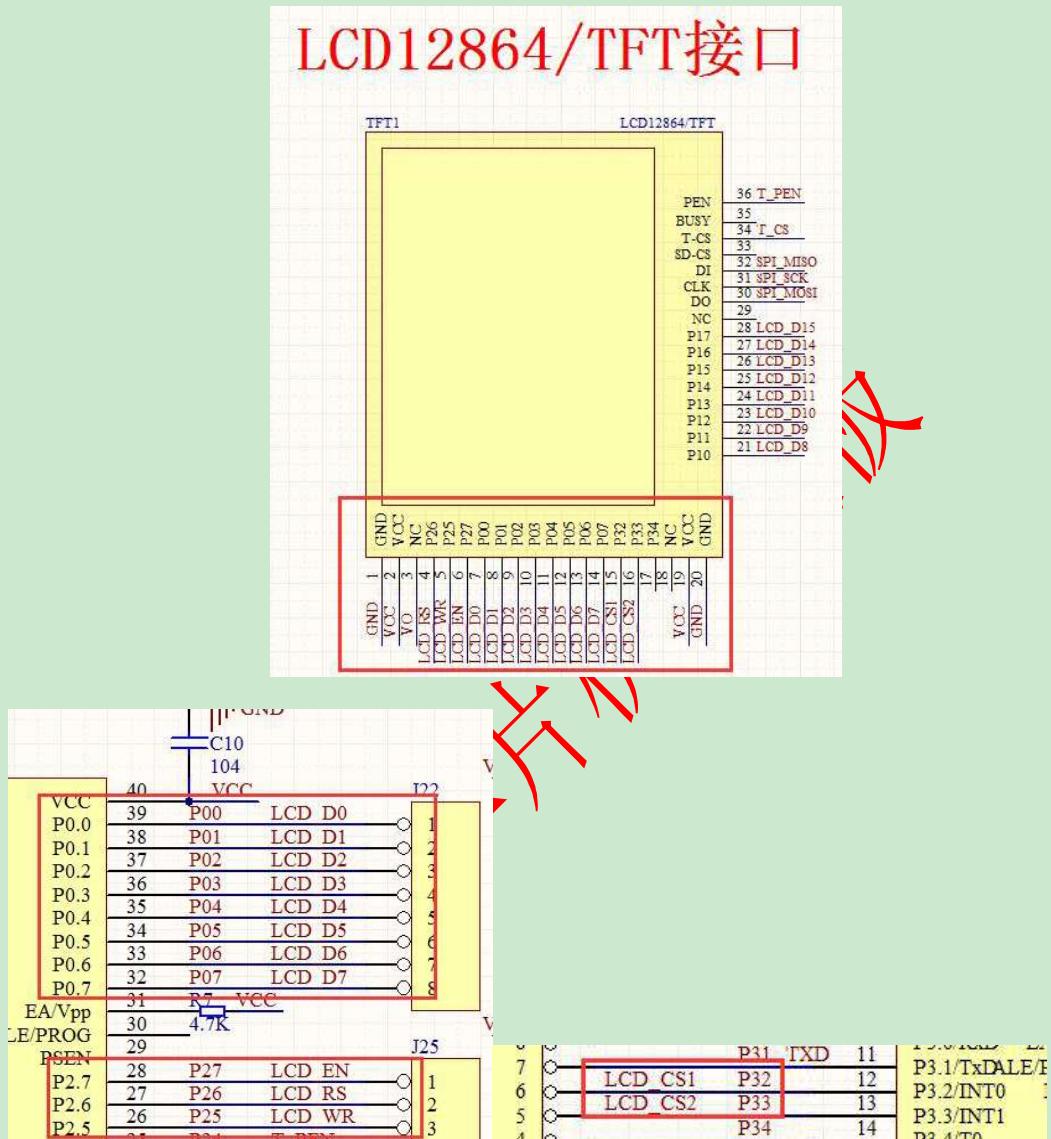
ST7565P 数据手册是液晶屏内核手册，里面有驱动方法和指令和程序有关的数据。通常液晶屏的数据手册都是英文的，但是其内容并不复杂，我们要查看的通常都是某几个寄存器内容，没有必要把手册整篇都看完。而且寄存器的介绍都是一些较容易看懂的单词，如果不明白可以借助翻译软件理解。在这里要强调一下，既然打算进入嵌入式行业，一定要习惯看英文资料，以后进入高级单片机或者 ARM 的学习会遇到很多英文资料的查阅。关于 ST7565P 的介绍大家可以查阅手册，这里就不列举了。

32.2 硬件设计

本实验使用到硬件资源如下：

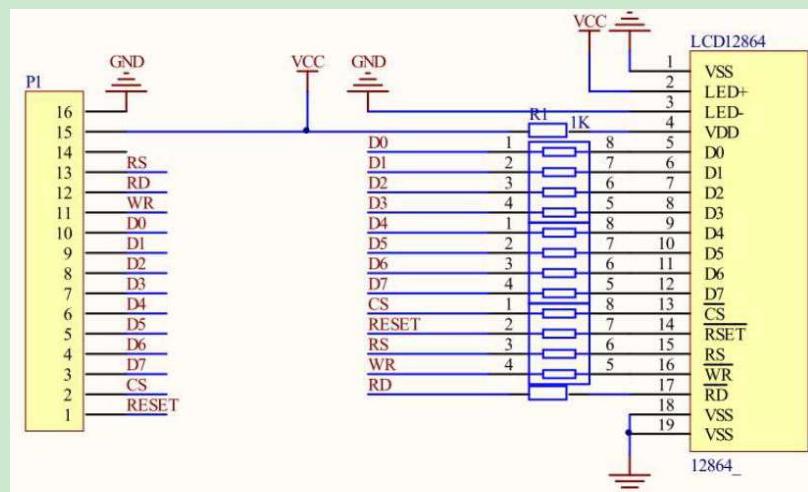
(1) LCD12864 液晶

开发板上集成了一个 LCD12864 液晶接口，下面我们来看下开发板上 LCD12864 液晶接口电路，如下图所示：



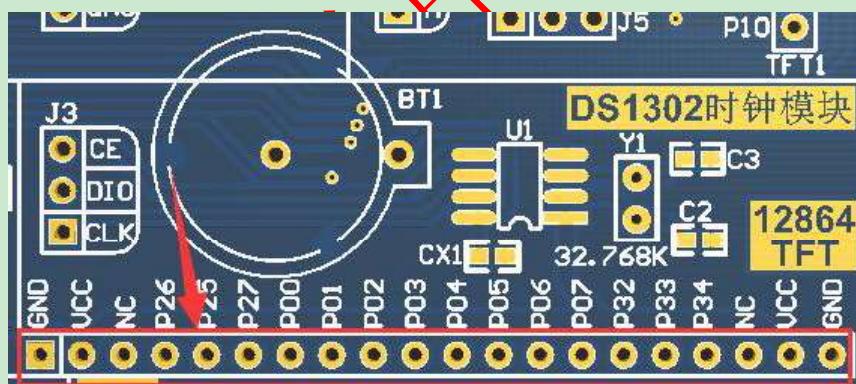
从上图中可以看出，该接口是 LCD12864 与 TFT 液晶彩屏共用的，LCD12864 占用下面 20 个管脚，我们配置的 LCD12864 是 16 脚，而带字库的 LCD12864 通常是 20 脚。这样可以兼容带字库的 LCD12864 和我们配置的不带字库的 LCD12864。至于 TFT 后面章节会介绍。该电路并不是独立的，LCD12864 的 8 位数据口 LCD_D0-LCD_D7 与单片机的 P00-P07 管脚连接，LCD12864 的 RS、RW、E 脚与单片机的 P26、P25、P27 管脚连接，LCD_CS1、LCD_CS2 与单片机的 P32 和 P33 管脚连接，所以当使用 LCD12864 时，其他设备就不要占用这些管脚，即使占用也只能分时复用。

LCD12864 液晶屏转接板原理图如下所示：



上图中的 LCD12864 是液晶屏 LCD12864 的排线接口，焊接在转接板的顶层，而图中的电阻 R1 及排阻都焊接在转接板的底层，P1 插针接口是 LCD12864 液晶屏的引出控制口，直接与开发板上的接口连接。

做本章实验时只需要将配置的 LCD12864 液晶从开发板的 LCD12864/TFT 接口左端处，按照丝印位置对应插入，无需额外接线，如下所示（具体可参考实验现象接线图）：



32.3 软件设计

本章所要实现的功能是：系统运行时，LCD12864 上显示汉字字符信息。

我们直接复制上一章创建的工程，在此模板基础上进行程序开发。为了能够与开发攻略教程对应，将复制过来的模板文件夹重新命名为“实验 30：LCD12864 液晶”。打开工程文件夹，发现里面有 3 个工程实验文件夹，分别是：刷屏、显

示文字、显示图片。刷屏实验是最基础的，对于初学者可以先看下这个实验了解 MiniLCD12864 是如何初始化的，然后显示文字实验，该实验是在刷屏基础上实现了文字显示函数。最后显示图片实验是在显示文字基础上实现了图片显示函数。这里我们以显示文字实验为例介绍 MiniLCD12864 的驱动方法。

打开该工程，里面多了 3 个文件：st7565.c 源文件、st7565.h 头文件和 charcode.h 头文件。前两个文件保存的是 MiniLCD12864 的驱动程序，最后一个头文件保存的是显示汉字及其他字符的字模数据，这些数据是通过取模软件实现，与 LED 点阵一样。至于如何创建多文件工程，在前面章节已经介绍过，这里就不再重复。下面我们就来分析下 MiniLCD12864 的驱动。

32.3.1 显示文字

(1) 设置 IO

在程序里设置要使用的 IO，例程里使用如下定义：

```
//--定时使用的 IO 口--//
#define DATA_PORT P0
sbit LCD12864_CS = P3^2;
sbit LCD12864_RSET = P3^3;
sbit LCD12864_RS = P2^6;
sbit LCD12864_RW = P2^7;
sbit LCD12864_RD = P2^5;
```

这样定义的好处是，以后修改 IO 时更方便，移植性更好。可以看到数据口是 P0 口连接，其它五个是控制 IO。

(2) 驱动函数声明

首先必须知道需要用到什么函数，定义函数如下：

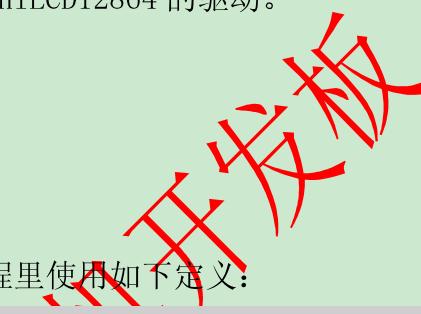
```
//--定义全局函数--//
void LcdSt7565_WriteCmd(cmd);
void LcdSt7565_WriteData(dat);
void Lcd12864_Init();
void Lcd12864_ClearScreen(void);
uchar Lcd12864_Write16CnCHAR(uchar x, uchar y, uchar *cn);
```

这些函数分别用来写指令，写数据，初始化，清屏和输出字符。

(3) 函数定义

1、写指令函数；

```
void LcdSt7565_WriteCmd(cmd)
```



```

{
    LCD12864_CS = 0; //chip select, 打开片选
    LCD12864_RD = 1; //disable read, 读失能
    LCD12864_RS = 0; //select command, 选择命令
    LCD12864_RW = 0; //select write, 选择写模式
    _nop_();
    _nop_();
    DATA_PORT = cmd; //put command, 放置命令
    _nop_();
    _nop_();
    LCD12864_RW = 1; //command writing , 写入命令
}

```

2、写数据函数；

```

void LcdSt7565_WriteData(dat)
{
    LCD12864_CS = 0; //chip select, 打开片选
    LCD12864_RD = 1; //disable read, 读失能
    LCD12864_RS = 1; //select data, 选择数据
    LCD12864_RW = 0; //select write, 选择写模式
    _nop_();
    _nop_();
    DATA_PORT = dat; //put data, 放置数据
    _nop_();
    _nop_();
    LCD12864_RW = 1; //data writing, 写数据
}

```

写指令和数据使用的是同一个时序图，在下图中 A0 是数据指令控制线，在程序里使用的是 RS 宏定义；/CS1 是片选线在程序中是 CS 宏定义，CS2 是复位线，是 RSET 宏定义；图中读写控制线是 WR/RD，程序中是 RW 宏定义；图中的 D[0:7]数据总线，在程序中是 DATA_PORT 宏定义。

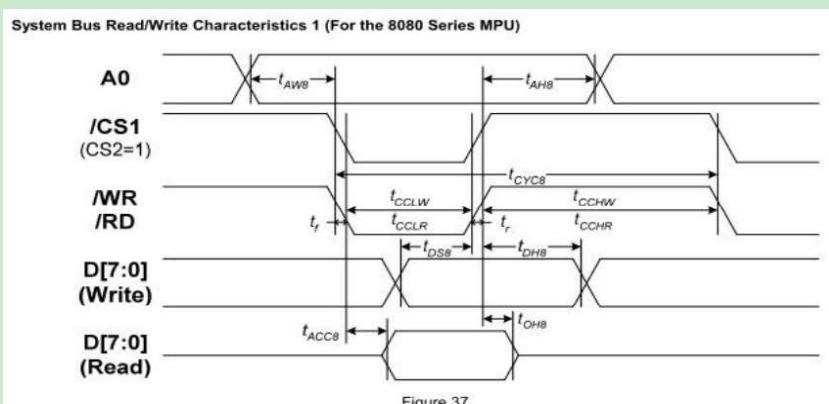


Table 24

Item	Signal	Symbol	Condition	Rating		Units
				Min.	Max.	
Address hold time	A0	tAH8		0	—	
Address setup time		tAW8		0	—	
System cycle time		tCYC8		240	—	

61 總計 71 | ⏪ ⏴ ⏵ ⏶ ⏷ ⏹ ⏺

3、初始化函数；

```

/*
*****
*****          : LCD12864_Init
* 函数功能      : 初始化 12864
* 输入          : 无
* 输出          : 无
* 说    明      : LCD12864 的命令指令可以查看例程文件夹下的
《ST7565p 数据手册》
*
*           * 的第 51 页的位置。
*****
*****/
void Lcd12864_Init()
{
    uchar i;
    LCD12864_RSET = 0;
    for (i=0; i<100; i++);
    LCD12864_CS = 0;
    LCD12864_RSET = 1;

    //-----Star Initial Sequence-----//
    //-----程序初始化设置，具体命令可以看文件夹下---//

    //--软件初始化--//
    LcdSt7565_WriteCmd(0xE2); //reset
    for (i=0; i<100; i++); //延时一下

    //--表格第 8 个命令， 0xA0 段（左右）方向选择正常方向（0xA1 为
反方向）--//
    LcdSt7565_WriteCmd(0xA1); //ADC select segment direction

    //--表格第 15 个命令， 0xC8 普通(上下)方向选择选择反向， 0xC0
为正常方向--//
    LcdSt7565_WriteCmd(0xC8); //Common direction

    //--表格第 9 个命令， 0xA6 为设置字体为黑色， 背景为白色---//
    //--0xA7 为设置字体为白色， 背景为黑色---//
    LcdSt7565_WriteCmd(0xA6); //reverse display
}

```

```

//--表格第 10 个命令, 0xA4 像素正常显示, 0xA5 像素全开--//
LcdSt7565_WriteCmd(0xA4); //normal display

//--表格第 11 个命令, 0xA3 偏压为 1/7, 0xA2 偏压为 1/9--//
LcdSt7565_WriteCmd(0xA2); //bias set 1/9

//--表格第 19 个命令, 这个是个双字节的命令, 0xF800 选择增压为
4X;--//
//--0xF801, 选择增压为 5X, 其实效果差不多--//
LcdSt7565_WriteCmd(0xF8); //Boost ratio set
LcdSt7565_WriteCmd(0x01); //x4

//--表格第 18 个命令, 这个是个双字节命令, 高字节为 0X81, 低字
节可以--//
//--选择从 0x00 到 0X3F。用来设置背景光对比度。---/
LcdSt7565_WriteCmd(0x81); //V0 a set
LcdSt7565_WriteCmd(0x23);

//--表格第 17 个命令, 选择调节电阻率--//
LcdSt7565_WriteCmd(0x25); //Ra/Rb set

//--表格第 16 个命令, 电源设置。--//
LcdSt7565_WriteCmd(0x2F);
for (i=0; i<100; i++);

//--表格第 2 个命令, 设置显示开始位置--//
LcdSt7565_WriteCmd(0x40); //start line

//--表格第 1 个命令, 开启显示--//
LcdSt7565_WriteCmd(0xAF); // display on
for (i=0; i<100; i++);

}

```

初始化函数是对时序线的初始化和初始指令的写入。这些指令的意义都在程序有标注，详细的解释可以查看“ST7565p 数据手册.pdf”。

4、清屏函数；

```

void Lcd12864_ClearScreen(void)
{
    uchar i, j;
    for(i=0; i<8; i++)
    {
        //--表格第 3 个命令, 设置 Y 的坐标--//

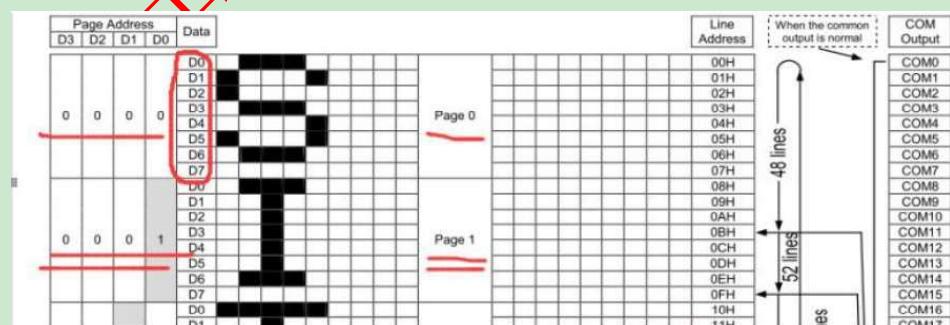
```

```

//--Y 轴有 64 个，一个坐标 8 位，也就是有 8 个坐标--//
//所以一般我们使用的也就是从 0xB0 到 0xB7, 就够了--//
LcdSt7565_WriteCmd(0xB0+i);
//--表格第 4 个命令，设置 X 坐标--//
//--当你的段初始化为 0xA1 时，X 坐标从 0x10, 0x04 到
0x18, 0x04, 一共 128 位--//
//--当你的段初始化为 0xA0 时，X 坐标从 0x10, 0x00 到
0x18, 0x00, 一共 128 位--//
//--在写入数据之后 X 坐标的坐标是会自动加 1 的，我们初
始化使用 0xA0 所以--//
//--我们的 X 坐标从 0x10, 0x00 开始---//
LcdSt7565_WriteCmd(0x10);
LcdSt7565_WriteCmd(0x04);
//--X 轴有 128 位，就一共刷 128 次，X 坐标会自动加 1,
所以我们不用再设置坐标--//
for(j=0; j<128; j++)
{
    LcdSt7565_WriteData(0x00); //如果设置背景为白色时，清屏
选择 0xFF
}
}
}

```

在清屏函数中，指令 `0xB0+i` 是“~~页~~”地址设置，从下图中可以看出页地址使用低四位配置。一共有 8 页，每页 8 行，共 64 行。从上图中还可以看出，数据是纵向的，这里可以认为，在取模的时候需要纵向取模。可以知道设置了页地址以后，加上数据的每一位，就能控制到每一行，也就等同于设置了行地址。



从上图得到的是行，还需要设置列，清屏函数中使用 `0x10, 0x04` 写入初始列。

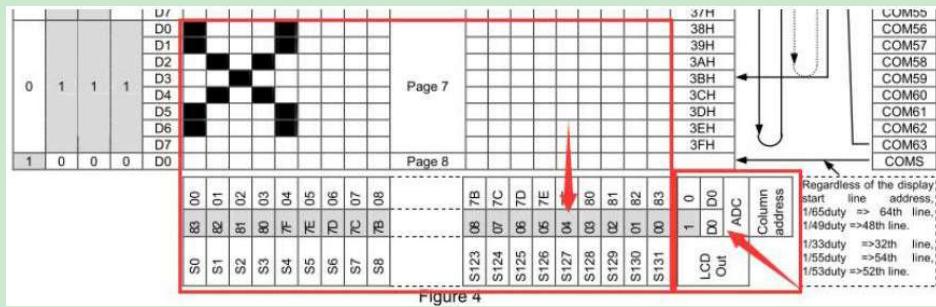


图 32.1

Column Address Set

This command specifies the column address of the display data RAM shown in Figure 4. The column address is split into two sections (the higher 4 bits and the lower 4 bits) when it is set (fundamentally, set continuously). Each time the display data RAM is accessed, the column address automatically increments (+1), making it possible for the MPU to continuously read from/write to the display data. The column address increment is topped at 83H. This does not change the page address continuously. See the function explanation in "The Column Address Circuit," for details.

	E R/W	A0 /RD /WR	D7	D6	D5	D4	D3	D2	D1	D0	A7	A6	A5	A4	A3	A2	A1	A0	Column address
High bits →	0	1	0	0	0	1	A7	A6	A5	A4	0	0	0	0	0	0	0	0	0
Low bits →						0	A3	A2	A1	A0	0	0	0	0	0	0	0	1	1
											0	0	0	0	0	0	1	0	2
											1	0	0	0	0	0	1	0	130
											1	0	0	0	0	0	1	1	131

图 32.2

在图 32.1 的右侧有设置列的增长方向，在初始化函数中设置的 0xA1，反方向，表示设置列地址应该从右至左开始。图中箭头指向 04，这里是表示程序中设置的指令 0x10, 0x04，意思是说从这一列向左到 0x83，也就是倒数第 S127 列到第 S0 列，共 128 列。在图 32.2 中可以看到设置列的指令使用方法，是先把列数据的高四位放在 0x1x 的低位中发送，紧接着把列数据的低四位放到 0x0x 的低位中发送。在 12864 内部会自动组合成右侧的 A7-A0 的列地址，一共可操作 132 列。但显示只能有 128 列，故需要设置从第几列开始显示。程序中设置的是右侧第四列开始，就去掉了四列。这四列是显示不出来的。

函数最后是输出数据 0x00，一共输出 128 次，可以看到这里的列是在自动增长，这个增长方向是有之前的初始化函数设置的。然后就是循环 8 次，每次修改页地址到下一页，列地址从新开始。这样一次循环就完成一次清屏。

5、写字符函数；

```
uchar Lcd12864_Write16CnCHAR(uchar x, uchar y, uchar *cn)
{
    uchar j, x1, x2, wordNum;

    //--Y 的坐标只能从 0 到 7，大于则直接返回--//
```

```

if(y > 7)
{
    return 0;
}

//--X 的坐标只能从 0 到 128， 大于则直接返回--//
if(x > 128)
{
    return 0;
}
y += 0xB0; //求取 Y 坐标的值
//--设置 Y 坐标--//
LcdSt7565_WriteCmd(y);
while (*cn != '\0') //在 C 语言中字符串结束以 ‘\0’ 结尾
{

    //--设置 Y 坐标--//
    LcdSt7565_WriteCmd(y);

    x1 = (x >> 4) & 0x0F; //由于 X 坐标要两句命令， 分高低 4 位，  

所以这里先取出高 4 位
    x2 = x & 0x0F; //去低四位
    //--设置 X 坐标--//
    LcdSt7565_WriteCmd(0x10 + x1); //高 4 位
    LcdSt7565_WriteCmd(0x04 + x2); //低 4 位

    for (wordNum=0; wordNum<50; wordNum++)
    {
        //--查询要写的字在字库中的位置--//
        if ((CN16CHAR[wordNum].Index[0] == *cn)
            && (CN16CHAR[wordNum].Index[1] == *(cn+1)))
        {
            for (j=0; j<32; j++) //写一个字
            {
                if (j == 16) //由于 16X16 用到两个 Y 坐标，当大于  

等于 16 时，切换坐标
                {
                    //--设置 Y 坐标--//
                    LcdSt7565_WriteCmd(y + 1);

                    //--设置 X 坐标--//
                    LcdSt7565_WriteCmd(0x10 + x1); //高 4 位
                    LcdSt7565_WriteCmd(0x04 + x2); //低 4 位
                }
            }
        }
    }
}

```

```
LcdSt7565_WriteData(CN16CHAR[wordNum].Msk[j]);  
}  
x += 16;  
}//if 查到字结束  
} //for 查字结束  
cn += 2;  
} //while 结束  
return 1;  
}
```

该函数的目的是在一个大小 16*16 的像素窗口显示一个字，有可能是中文，也可以是字符（这里字符要做成 16*16 的）。这个函数有三个参数，一个是 X 坐标，即页地址，一个是 Y 坐标，即列地址，最后一个是一个指针，是字的索引。函数一开始判断页地址和列地址时候超出范围，超出范围就退出该函数，否则向下执行。

首先求取这个字要从哪一页。

然后是进入写数据循环，这里判断~~*cn != '\0'~~，表示该函数可以连续写入字（字串的结尾一般是' \0'）。在函数结尾有~~cn += 2;~~；这里字以两个地址增长，也就是说如果不是汉字是字符，就应该用字符加空格做索引。

然后在此写入页地址，计算从哪一列开始写，在写完一个字以后，重新写入页地址，且列地址以加 16 列变化，表示下一个连续的字写在这个字的后面。

然后有一个~~for (wordNum=0; wordNum<50; wordNum++)~~，这个循环的目的是从字库文件里找字，这个~~wordNum~~ 限制在 50 以内，表示字库里的字不能超过 50 个，如果要超过，就需要修改这里。循环 50 次来查找这个字。如果没找到就不显示，继续查找下一个字。

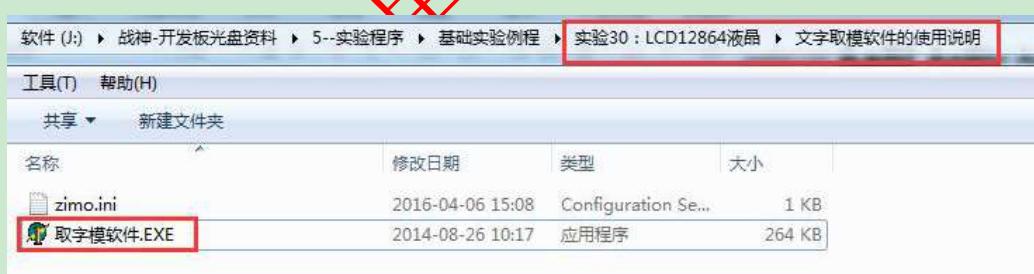
如何判断是否有这个字。在字库文件里有一个汉字字模结构体，该结构体前两个位字的索引，对应的是这个汉字的字符串，在查找时对比汉字的两个字节是否相等。如果都相等，表示这个汉字在字库里。如果是使用的字符，那么第一个对比的是字符第二个是空格，如果两个相等，表示字库里有这个字符。在结构体重第二个元素是一个 32 字节的字符串，是存放字模的，当找到相应字的索引时，就可以调用该字的字模来显示这个字。

如果查到字库有这个字，就会进入循环写数据中。一个字像素点是 16×16 个，一个字节占 8 个，一共 32 个字节数据，也就是要写 32 次。在写的过程中是先写上一页 16 个数据，当数据超过 16 个时，跳转到下一页开始写 16 个数据，两页上下合并为 16×16 个点，32 个数据。写完一个字，就进入到查找下一个字，直到字符串结束写字循环。

(4) 取模软件使用

取模软件的使用方法在前面学习 LED 点阵实验的时候就已经介绍过，这里我们再次介绍下。

1. 打开取字模软件



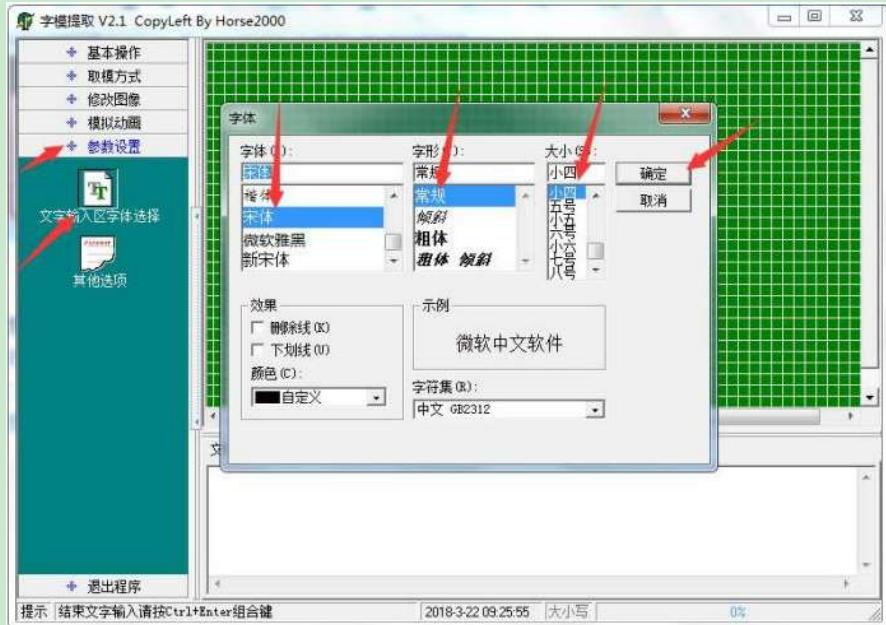
打开界面如下：



在上图中，左侧是各种设置区，这里可以设置取模方向、取模方式等等必要参数。右上侧是显示预览区，会显示这个字在显示屏上需要点亮的点和区域大小。右下侧是数据输入和输出区，在“文字输入区”输入需要取的字或者字符，按 Ctrl+Enter 键，能够把文字的预览显示在预览区。当点击生成方式的时候能够在点阵生成区显示字模。这个操作接下来会详细介绍。

1) 设置字体

首先在设置区选“参数设置”→文字输入区字体选择，这里不用选择“基本操作”→新建图像，在输入数字以后按 Ctrl+Enter 就能产生新的图像。打开字体选择以后弹出如下窗口。

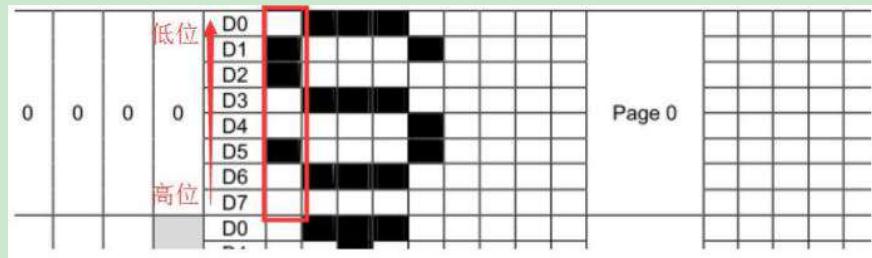


在窗口里设置字体如图所示，在这种字体下生成的字模大小为 16*16。如果字体不对，生成的大小就不一定是 16*16 的，后面会介绍如何修改修改字体生成其它大小的字模。注意字模的大小和输出字的函数有关。这里设置完毕后，点击“确定”。

2) 设置“其它选项”



在“参数设置”->其它选项，设置“纵向取模”，原因是 12864 中数据是列方向向下排列。如下图所示，纵向为一个数据，这里确定为纵向取模。



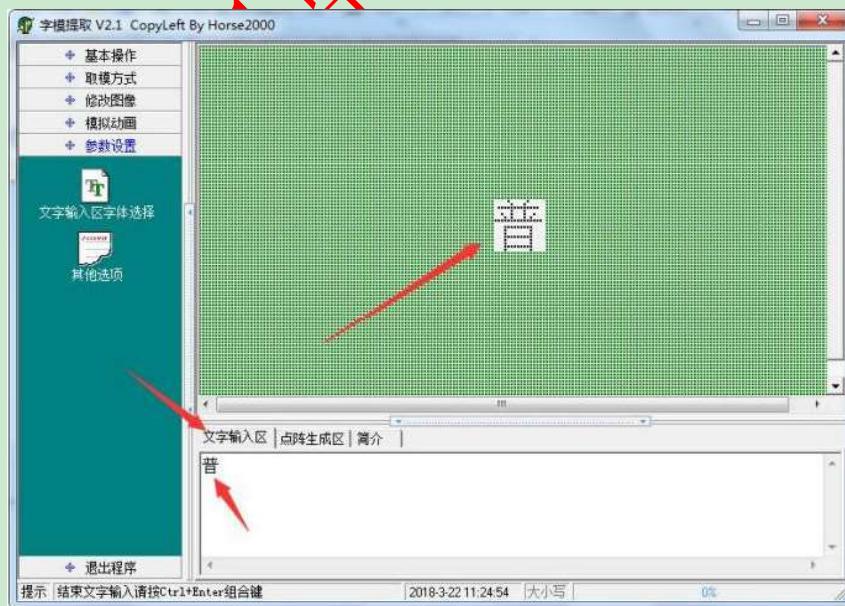
“字节倒序”的确定是看数据高低位，如果不倒序，则从上到下是由高到低，从手册上可以看出，一个字节中数据是由下到上是由高位到低位，故取模的时候要设置选中倒序。当然有时候在程序中找到数据上下颠倒的寄存器设置，也可以不设置倒序，这里需要开发人员自己去思考。

保留最后的逗号，这个可选，也可以不选，需要看你在程序里如何存字库，一般连续存放，最好选上，如果没选，就在程序中输入一个也行。如果只有一个字，就可以不选择。

“A51 的位数”这里可以不用管，我们程序是 C51 格式的，不需要生成 A51 格式。A51 格式适用于汇编编程，是否需要加 0，得看编译方式和显示器，这里不做讲解。

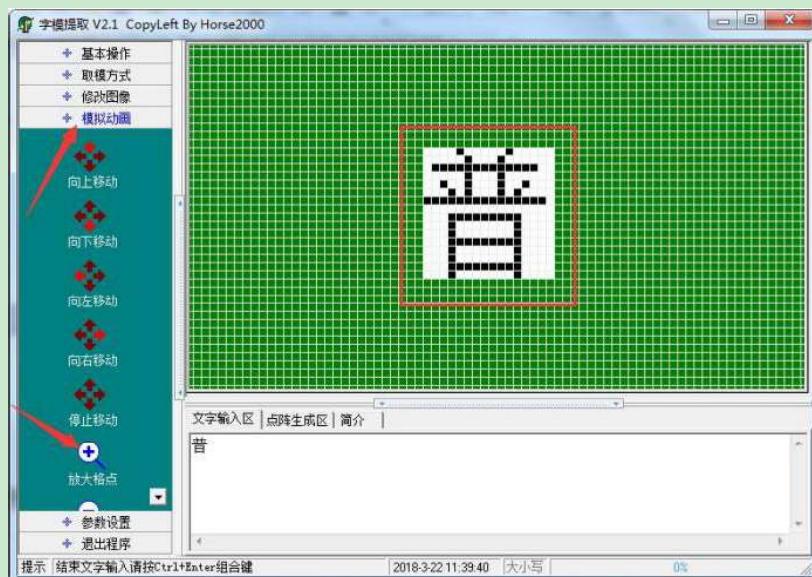
设置完毕后，点击“确定”完成设置。

3) 输入文字



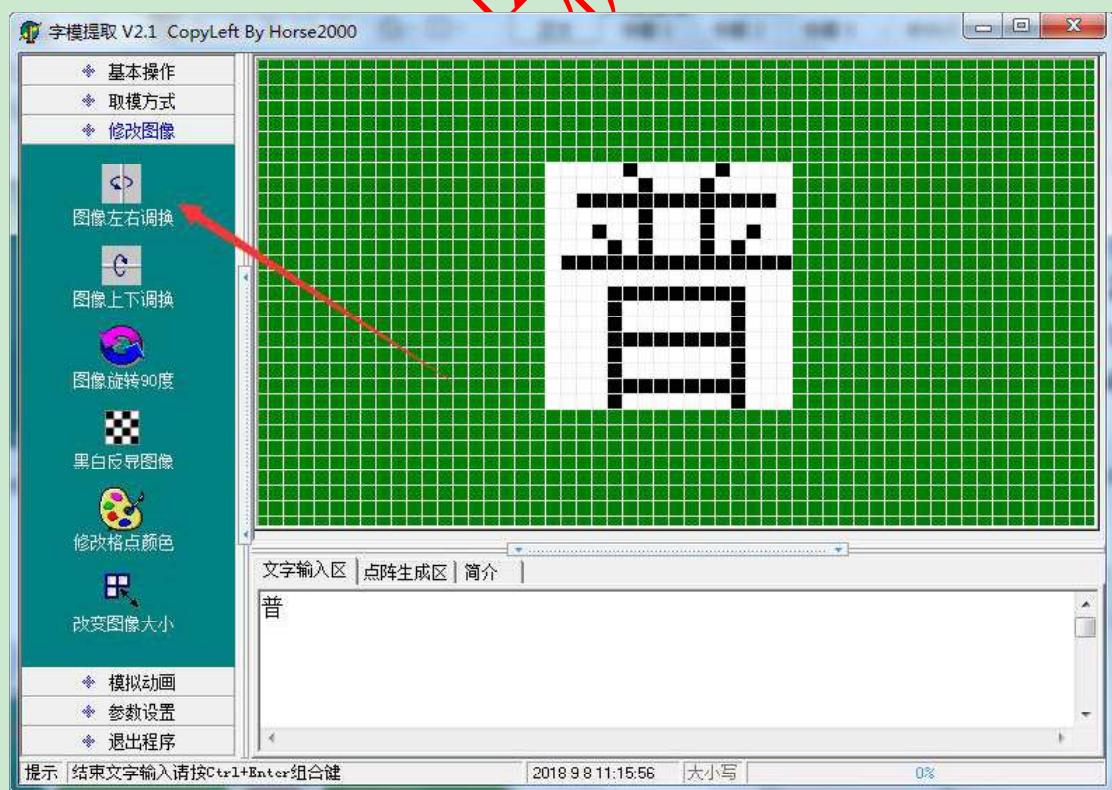
在文字输入区输入汉字，如上图所示，输入需要的汉字，然后按 Ctrl+Enter 键，在预览区会显示这个字的样式，如果不喜欢，可以进行自主修改，建议在学会使用以后再进行。这时候显示的字很小，需要放大。在“模拟动画”设置项里

有“放大格点”，点击可放大预览区。

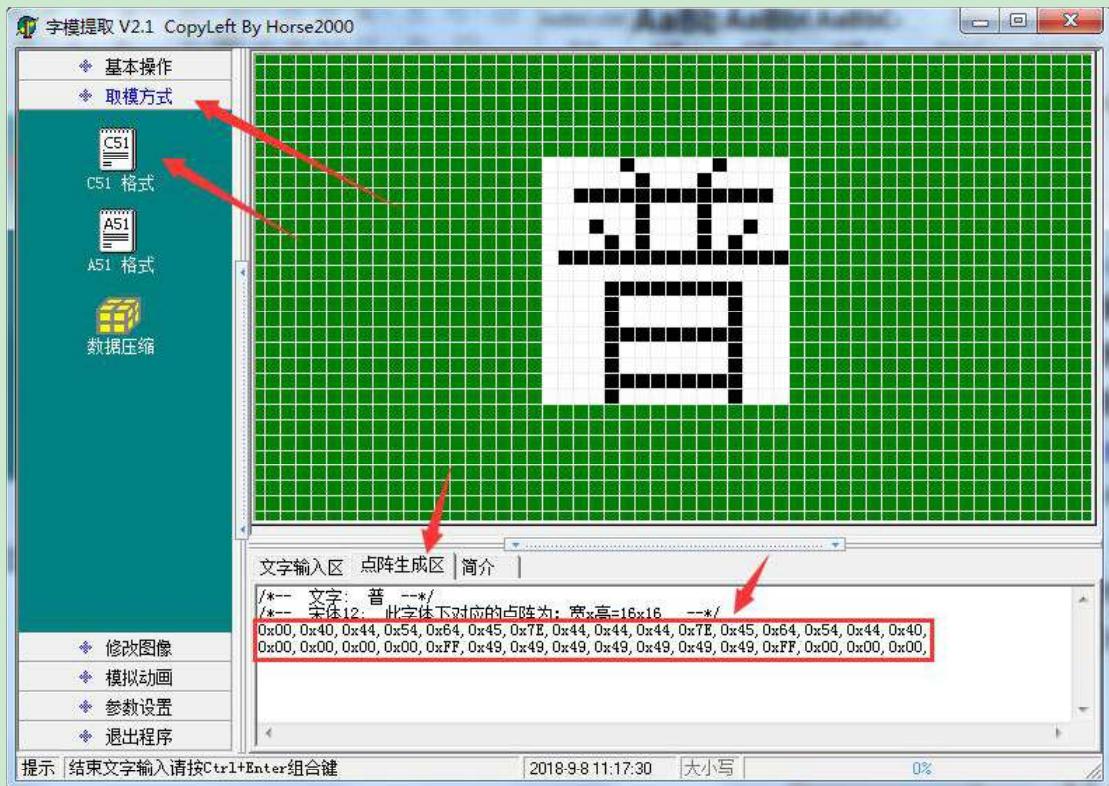


4) 左右调换字模

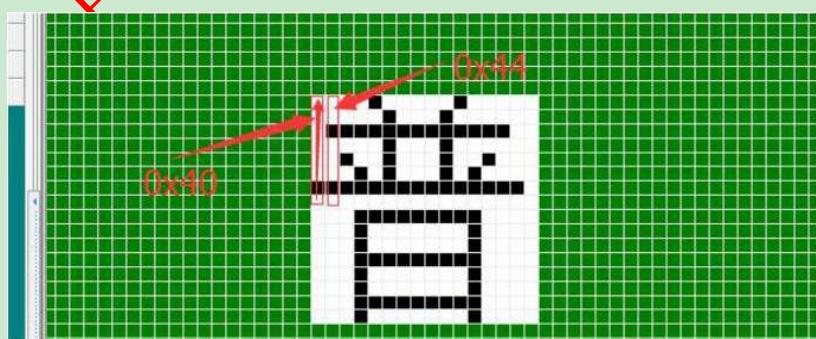
在“修改图像”区选择图像左右调换，文字就会相应的左右调换，如果不左右调换则显示在屏幕上的文字就会左右反过来显示，这个是根据我们程序来设置的。



5) 生成字模



在“设置区”->取模方式里点击“C51 格式”，就会在“输出区”->点阵生成区里有输出对应字的字模。输出区里有这个字和这个字的像素大小，这里是 16*16 的。还有两行是字模，字模一共 32 个字节。第一个字节是 0x40，从下图可以看出第一个字节的八个点从下到上依次是 0100 0000，也就是 0x40，第二个字节从上到下是 0100 0100，也就是 0x44。由此可以看出取模方向是纵向的是从左至右一个字节一个字节的生成的，因此在程序里设计的是页设置以后，写 16 列，在换页写十六列，和这里是对应的。如果两个不对应，就得修改成对应的。



这时候就做出了一个字的字模。现在就是做一个字模文件，一般是做成头文件直接使用。

5) 制作字模文件

这里可以按照例程写头文件，做一个宏定义#define CHAR_CODE，这里用宏定义是为了在使用的时候，如果不使用文字，可以不定义这个，就不会被编译进来，是程序易于裁剪和编辑。

从上图可以看到在结构体数组中复制了之前生成的字模，然后在前面添加了索引。这里索引是用双引号加输入的字，做成一个字符串，占用两个字节，在结构体中是放到索引中的。从而只做了一个简单的字库。

如果需要多个，可以重复上述步骤。一般一个数组中使用同一个大小的字做字库。可以看看例程里。

~~注意：例程中查询字数最多 50 个，字库里字数可能不止 50 个，可以设计更多，如果有足够大的空间，可存放更多个字，这里查询的数量就应该变大，或者以找到改字为准。这个需要用户自己去思考。~~

(5) 12864 怎么显示字

如何显示字，~~需要在函数中对输出字的函数进行调用。~~

1) 添加头文件支持

```
main.c ST7565.H CHARCODE.H
01 #ifndef __ST7565_H
02 #define __ST7565_H
03
04 #include<reg51.h>
05 #include<intrins.h>
06
07 //---包含字库头文件
08 #define CHAR_CODE
09
10
11 //---重定义关键词---//
12 #ifndef uchar
13 #define uchar unsigned char
14 #endif
15
```

这里添加宏定义，即可包含字库头文件。也会将 ST7565.c 中有关于输出字的函数进行编译。

```

1 main.c ST7565.H CHARCODE.H st7565.c
01 #include<reg51.h>
02 #include"st7565.h"
03 //---声明一个全局变量---/
04 void Delay10ms(unsigned int c);
05
06 *****
07 * 函数名      : main
08 * 函数功能    : 主函数
09 * 输入        : 无
10 * 输出        : 无
11 *****
12
13

```

在主函数源文件这里添加 ST7565.h 的头文件。这样在主函数里就可以调用显示函数了。

2) 调用函数进行显示

```

14 void main()
15 {
16     uchar i = 128;
17     Lcd12864_Init();
18     Lcd12864_ClearScreen();
19
20     while (1)
21     {
22         for (i=0; i<8; i += 2)
23         {
24             Lcd12864_ClearScreen();
25
26             //--由于这个函数显示方向正好相反--//
27             Lcd12864_Write16CnCHAR(0, i, "司公限有技科中普");
28             Delay10ms(100);
29         }
29     }
30     while(1); //注释掉就成上下关东字幕
31
32 }
33
34

```

这里首先调用初始化函数，对 LCD12864 内部寄存器进行初始化配置。然后调用清屏函数，这里清屏是用的是函数里的设置颜色。然后进入了循环调用，这里只有四次循环。用 $i < 8$ ，是为了表示有八页，每次翻两页。然后是重新清屏，这里是为了显示下一行数据，如果没有清屏，下一行显示的时候，上一行的数据是会一直显示的。接着是输出要显示的数据，调用了输出字的函数，函数参数首先是起始列，然后是页，这里页地址按照两页在变化，然后是数据，历程中的字库仅有需要使用的几个字，如果用户只做了自己的字库，就可以显示其他字。否则只有例程中的字。然后是延时，四次循环，结束后停留在最后一行显示。

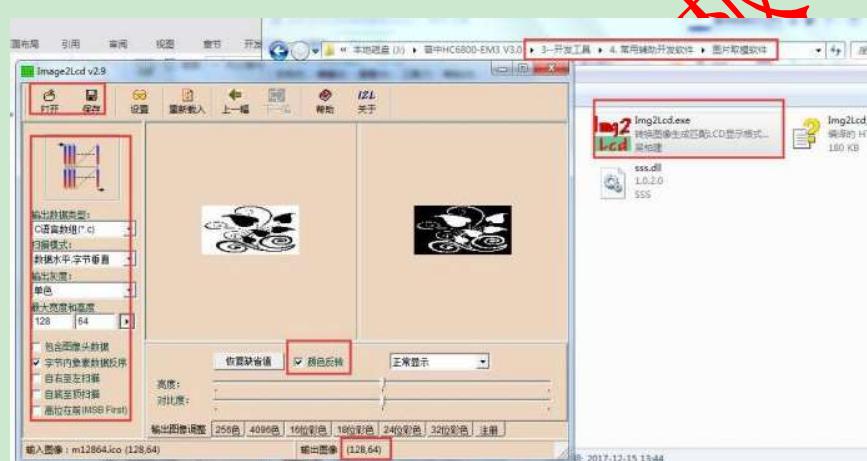
32.3.2 显示图片

(1) 如何将图像取模

Mini 12864 是灰度显示图像，也就是单色的，可以使用图像取模软件，取模方法大同小异，需要用户自己去实践。这里使用光盘中提供的图像取模软件，位置如下图，如果光盘资料不一样，应该会在其他位置，需要用户仔细搜索整个光盘资料。首先我们需要准备一张单色的图像，这里图像大小不应该超过 128*64，这里是 128 列，64 行的像素，如果不对，就需要进行裁剪，或用 PS 等软件进行像素调整。这里直接使用 128*64 的图像，不介绍如何调整图像，用户可以自己去查阅资料。

1) 打开图像

打开软件以后，电机菜单栏的“打开”，选择修改好的黑白图片。设置如图所示：



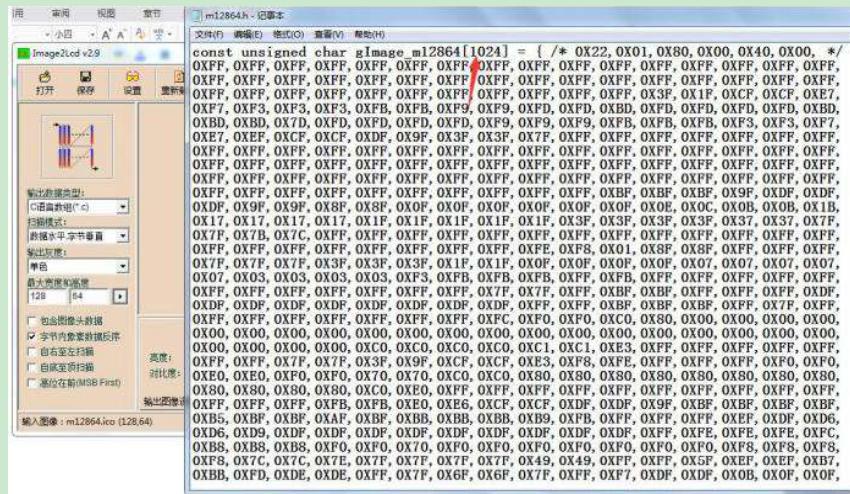
在图上数据是 C 语言类型。扫描模式是按照程序对应来，程序中设计成写完第一页以后，写第二页，故数据取模应该字节垂直，数据水平方向。输出颜色是单色的。大小设置成 12864，这里的大小修改以后，要看右下角的输出是否是 12864，如果不是，就需重新使用画图软件修改大小然后重新打开，直到图像是自己需要的大小。然后设置字节反向，是因为在页内，字节数据由下至上是高到低，这里普通情况是由上至下是由高到低，就需要反序。颜色反转选择是由于在程序里初始化 LCD12864 的时候，输出颜色有反向，对应这里就需要反转。

2) 生成图像的模

设置好以后，点击保存，设置输出 xxx.h 文件，然后会弹出图像的模，如下图。

弹出的模里，可以看到生成了一个大小是 1024 的数组，实际上就是 128*8，由于 64 行分成了 8 字节，所以是 1024 个数据。这里数据中前几个是图像头

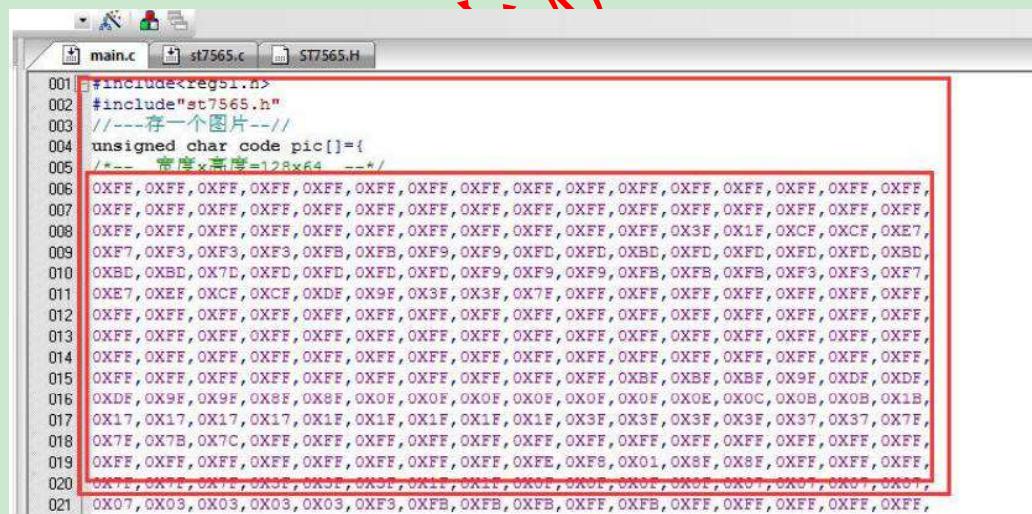
的数据，这个一般不用，复制其余的 1024 个数据到程序中替换历程中的 1024 个数据，就能显示当前这个图像。



(2) 12864 如何显示图像

1) 添加图像的模到程序中

把图像的模复制到这里覆盖程序里原来的数据，就可以显示当前的图像。这里可以看到包含有 12864 的驱动源文件和头文件，可以对 12864 进行操作。



2) 输出显示代码

The screenshot shows the Keil MDK-ARM IDE interface. On the left, the Project Explorer displays a file structure under Target 1: Source Group 1 contains STARTUP.A51, main.c, reg51.h, st7565.h, and intrins.h; main.c also contains st7565.c, reg51.h, and intrins.h. The main.c code window on the right shows the following C code:

```

145 void main(void)
146 {
147     uchar i , j;
148     uint n;
149     Lcd12864_Init();           //初始化
150     Lcd12864_ClearScreen();   //清屏
151
152     for (i=0;i<8;i++)
153     {
154         //--设置初始显示开始位置--
155         LcdSt7565_WriteCmd(0x40);
156
157         //--设置y坐标--
158         LcdSt7565_WriteCmd(0xB0+i);
159
160         //--设置x坐标--
161         LcdSt7565_WriteCmd(0x10);
162         LcdSt7565_WriteCmd(0x04);
163         for(j=0; j<128; j++)
164         {
165             LcdSt7565_WriteData(~pic[n]);
166             n=n+1;
167         }
168     }
169     while (1)
170     {
171     }
172 }

```

主函数中调用了 12864 初始化，清屏。

然后是一个八次循环，这是表示有八页，然后设置显示开始位置，这个其实可以没有，然后设置页地址，每循环一次，页地址会随着循环往下换页，然后设置起始列地址，从每一页的第一列开始写，这里设置起始列以后，直接输出 128 个数据就可以写满一页，这是因为列地址在写的时候会自动增长。这里使用的 128 次循环输出，数组里每一个数据都进行了取反输出，这也表示了为什么在图像取模的时候需要颜色反转。如果这个不取反，取模就不需要反转颜色，用户可以自己实践。程序里的 n 是用来表示当前输出的数组里第几个数，没输出一次加 1，就表示输出下一个数。

这样就完成了一次图像的输出。然后执行死循环，防止程序跑飞。显示效果如下，就不再是例程里的图像，而是自己定义的。

至此我们就介绍完，具体代码大家可以打开工程查看。

32.4 实验现象

使用 USB 线将开发板和电脑连接成功后(使用短接片将 USB 转串口模块上 J39 端子的 P31T 与 URXD 短接，J44 端子的 P30R 与 UTXD 短接，电脑能识别开发板上 CH340 驱动串口)，把编写好的程序编译后将编译产生的.hex 文件烧入到芯片内，按照如下图所示的接线方式可以看到：LCD12864 逐行显示“普中科技有限公司”。



课后作业

- (1) 在 LCD12864 上显示时钟。
(温馨提示：将本章实验与 DS1302 时钟程序组合)

第 33 章 TFTLCD 显示实验

在前面章节，我们介绍了 LCD1602 和 LCD12864 液晶显示器，它们只能用来显示字符、汉字和简单单色图片，而不能显示彩色图片。这一章我们就来介绍一种彩色液晶显示装置--TFTLCD 薄膜晶体管液晶显示器，使用它不仅可以显示更多的汉字、字符数字信息，还可以显示 16 位色真彩图片，在一些高端设备中应用广泛，通常采用 8080 总线时序通信。我们开发板上集成了一个 TFTLCD 液晶显示器接口，将配置的 TFTLCD 液晶对应插入即可。本章要实现的功能是：系统运行时，在 TFTLCD 上显示 ASCII 字符和汉字。学习本章可以参考前面的实验章节内容。本章分为如下几部分内容：

- 33.1 TFTLCD 介绍
- 33.2 硬件设计
- 33.3 软件设计
- 33.4 实验现象

33.1 TFTLCD 介绍

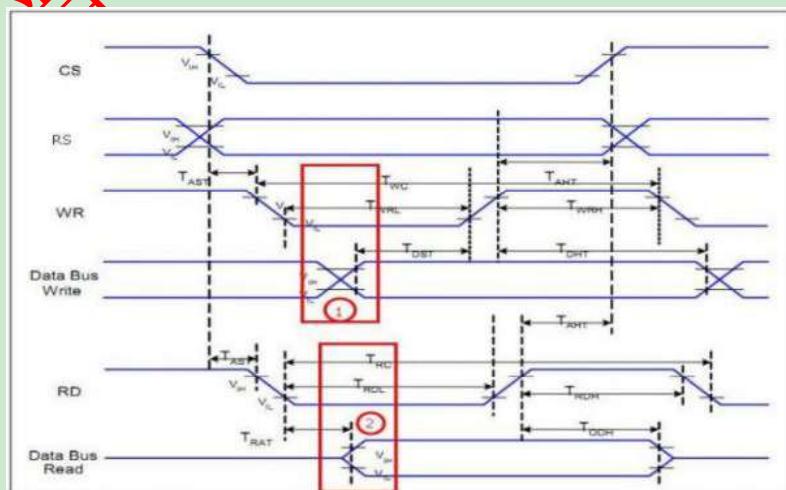
TFT-LCD 是薄膜晶体管液晶显示器英文 thin film transistor-liquid crystal display 字头的缩写。TFT 液晶为每个像素都设有一个薄膜晶体管 (TFT)，每个像素都可以通过点脉冲直接控制，因而每个节点都相对独立，并可以连续控制，不仅提高了显示屏的反应速度，同时可以精确控制显示色阶，所以 TFT 液晶的色彩更真，因此 TFT-LCD 也被叫做真彩液晶显示器。

常用的 TFT 液晶屏接口有很多种，8 位、9 位、16 位、18 位都有，这里的位数表示的是彩屏数据线的数量。常用的通信模式主要有 6800 模式和 8080 模式，对于 TFT 彩屏通常都使用 8080 并口（简称 80 并口）模式。

如果大家接触过 LCD1602 或者 LCD12864 等，那么就会发现 8080 模式的读写时序其实跟 LCD1602 或者 LCD12864 的读写时序是差不多的。8080 接口有 5 条基本的控制线和多条数据线，数据线的数量主要看液晶屏使用的是几位模式，有 8 根、9 根、16 根、18 根四种类型。它们的功能如下：

RST	0: 复位选择 1: 取消复位
CS	0: 片选选择 1: 取消片选
RS	0: 控制寄存器 1: 数据寄存器
RD	0: 读选择 1: 读取消
WR	0: 写选择 1: 写取消
DB0~DB8	数据线

接下来我们来看一下 8080 接口模式的时序，如下图：



从上图我们就可以很清晰的看得出液晶屏的读写时序：

①：在 WR 跳变为低电平之后，液晶屏开始读取总线上面的数据。如果使用 I0 口模拟写入的时候，可以先在总线上面写入数据，然后在跳变 WR，以保证当读取的时候，总线上面的数据是稳定的。

②：在 RD 跳变为低电平之后，液晶屏放置数据到总线上面。

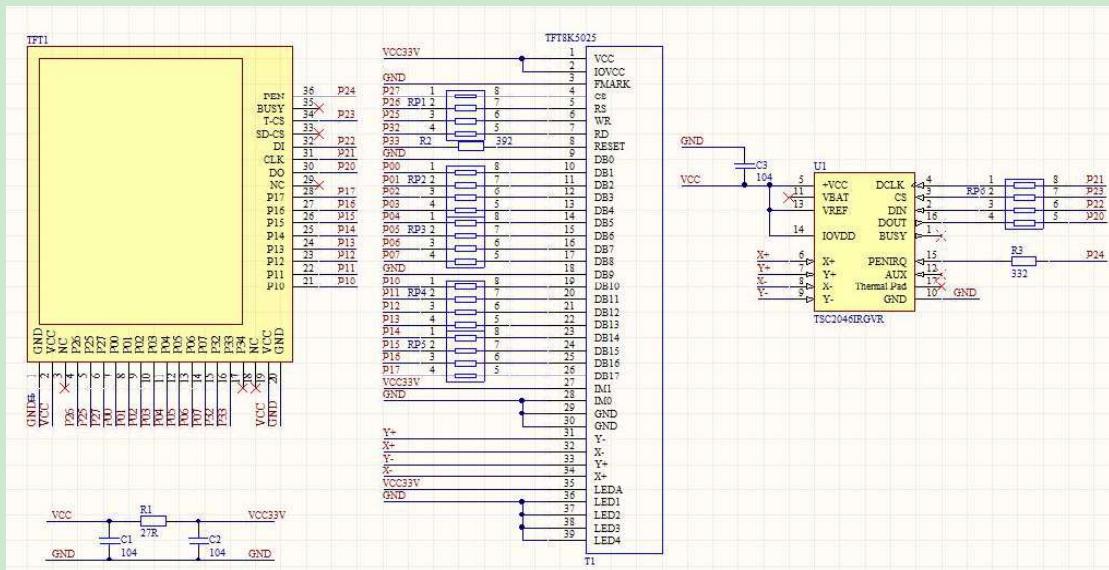
我们使用 51 单片机就是通过单片机的 I0 口模拟 8080 时序进行 TFT 彩屏控制。

下面我们来介绍下 TFTLCD 模块，我们公司推出的 TFTLCD 模块有很多种，按照屏幕大小的不同可分为 2.0、2.4、2.8、3.0、3.2、3.5、3.6、4.3、4.5、7 寸等，不同尺寸的彩屏对应的分辨率可能不同，比如说 3.5 寸的彩屏分辨率为 320*480（长*高），4.5 寸的为 480*854，当然这个具体要看对应彩屏的数据手册，彩屏数据手册和彩屏原理图在光盘的“\5—实验程序\基础实验例程\实验 31：TFT 彩屏”，由于彩屏更新速度较快，所以光盘内我们提供的是一个链接，大家需要在这个链接内下载对应的彩屏数据手册和原理图及程序。按照 TFT 彩屏驱动芯片的不同可分为海信 HX83xx、ILI93xx、R615xx、LG45xx、NT355 等等，你手上的彩屏驱动芯片具体是哪一种，需要看下彩屏板表面左上角或者背面型号，通常我们都会将彩屏的驱动芯片型号放在 TFTLCD 模块的左上角或者背面。我们的 TFTLCD 模块都自带触摸功能，可用来做输入控制。

本章我们就以 2.6 寸的 TFTLCD 模块为例来介绍（其他尺寸的彩屏和驱动芯片使用方法类似），该模块驱动芯片型号是 R61509V，分辨率为 240*320，接口为 16 位的 80 并口，自带触摸功能。该模块的外观图如下图所示：



该模块原理图如下图所示：



TFTLCD 模块采用垂直两排插针，横向是一个 20Pin 的插针，纵向是一个 16Pin 的插针，间距 2.54 公排针与开发板上 TFT/LCD12864 接口连接，从图中可以看出，此 TFTLCD 模块采用 16 位的并口方式与外部连接，之所以不采用 8 位的方式，是因为彩屏的数据量比较大，尤其在显示图片的时候，如果用 8 位数据线，就会比 16 位方式慢一倍以上，我们当然希望速度越快越好，所以选择 16 位的接口，当然不同 TFTLCD 数据位数不一样，如果彩屏是 8 位的同样也是接在 16 位的对应高 8 位或者低 8 位上，接口使用 16 位是方便兼容其他彩屏。图中还列出了触摸屏芯片的接口，关于触摸屏本章我们不多介绍，在后面的章节会有详细的介绍。该模块的 80 并口有如下一些信号线：

CS: TFTLCD 片选信号。

WR: 向 TFTLCD 写入数据控制。

RD: 从 TFTLCD 读取数据控制。

RS: 命令/数据选择 (0, 读写命令; 1, 读写数据)。

DB[15: 0]: 16 位双向数据线。

RST: TFTLCD 复位。

80 并口的通信时序前面已经介绍，所以要控制 TFTLCD 模块显示，总共需要 21 个 IO 口（除触摸功能管脚）。

知道了模块的管脚功能及通信时序，接下来我们就来介绍下如何让液晶模块显示。通常按照以下几步即可实现 TFT 液晶显示：

(1) 设置 STM32F1 与 TFTLCD 模块相连接的 IO

要让 TFTLCD 模块显示，首先得定义 TFTLCD 模块与 51 单片机相连的 IO 口，以便控制 TFTLCD。

(2) 初始化 TFTLCD 模块 (写入一系列设置值)

初始化 IO 口，接着就是对 TFTLCD 进行配置，首先就是要复位下 LCD，然后就是初始化序列，即向 LCD 控制器写入一系列的设置值(比如 RGB 格式、LCD 显示方向、伽马校准等)，这部分代码一般 LCD 厂商会提供，我们直接使用这些初始化序列即可，无需深入研究。关于这些设置值可以在你所使用的彩屏模块驱动芯片数据手册内查找到，只不过这些数据手册全是英文的，其实也不是很难，我们用到的只是几个设置值而已，不认识的可以百度翻译下。初始化完成之后，LCD 就可以正常使用了。

(3) 将要显示的内容写到 TFTLCD 模块内

这一步需要按照：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，如果我们想要显示字符或数字，就必须要多次使用这个步骤，从而达到显示字符或数字的目的，一般我们会设计一个函数来封装这些过程（实现字符或数字的显示），之后只需调用该函数，就可以实现字符或数字的显示了。

这一部分内容等到我们后面编写程序的时候大家就可以看到，其实还是比较简单的。接下来我们看下 TFTLCD 的硬件电路。

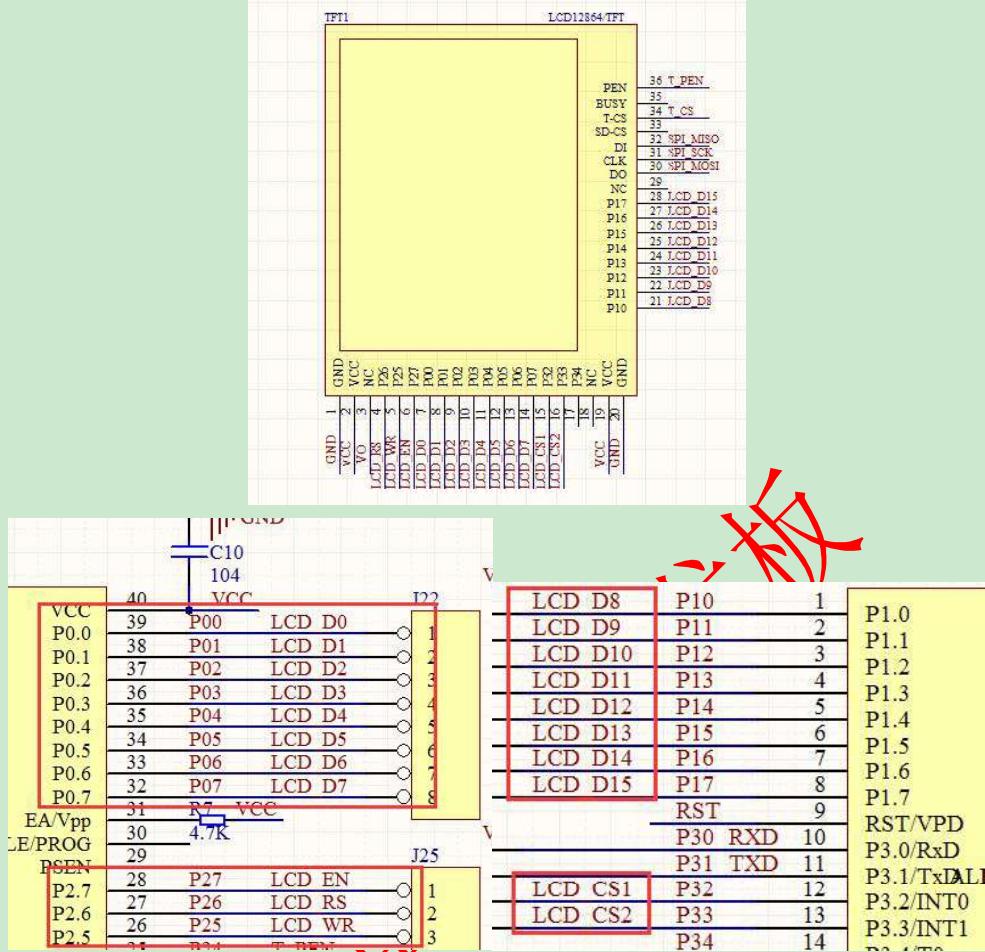
33.2 硬件设计

本实验使用到硬件资源如下：

(1) TFTLCD 液晶模块

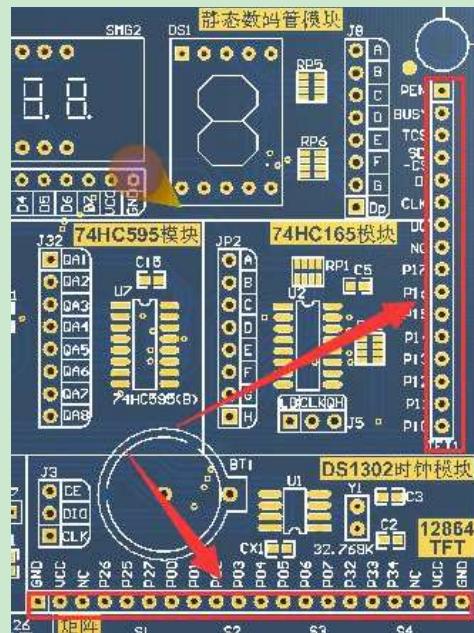
开发板上集成了一个 TFTLCD 液晶接口，下面我们来看下 51 单片机与 TFTLCD 接口的连接关系，如下图所示：

LCD12864/TFT 接口



在前面介绍 LCD12864 时我们知道该 TFT1 接口可以兼容 LCD12864 和 TFTLCD 模块，LCD12864 占用 TFT1 下面 20 个管脚，而 TFTLCD 模块占用全部引脚。从图中可以看到，TFTLCD 模块的 16 位数据口的高八位连接在单片机的 P1 口，低八位连接在单片机的 P0 口。TFTLCD 模块的 CS、RS、RW、RD、RESET 脚分别连接在单片机的 P2.7、P2.6、P2.5、P3.2、P3.3 口。所以当使用 TFTLCD 时，其他设备就不要占用这些管脚，即使占用也只能分时复用。TFTLCD 接口上的 T_PEN、T_CS、SPI_SCK 等引脚是用于控制触摸的，这些在后面触摸实验章节会介绍到。

因此我们只需要将 TFTLCD 模块插入开发板上 TFTLCD 接口即可，如下所示（具体可参考实验现象接线图）：



33.3 软件设计

本章所要实现的功能是：系统运行时，在 TFTLCD 上显示 ASCII 字符和汉字。

我们直接复制上一章创建的工程，在此模板基础上进行程序开发。为了能够与开发攻略教程对应，将复制过来的模板文件夹重新命名为“**实验 31：TFTLCD 显示实验**”。打开工程文件夹，发现里面多了几个文件夹：APP、GUI、Output、Public、User 和一个工程文件 template.uvproj。下面先介绍下这几个文件夹的作用。

APP 文件夹：用来存放 TFTLCD 液晶和触摸的驱动程序。

GUI 文件夹：用来存放在 TFTLCD 液晶底层驱动基础上封装的应用。

Output 文件夹：用于存放编译产生的 c/汇编/链接的列表清单、调试信息、hex 文件、预览信息、封装库等文件。

Public 文件夹：用于存放 51 单片机公共文件，如串口驱动、延时等。

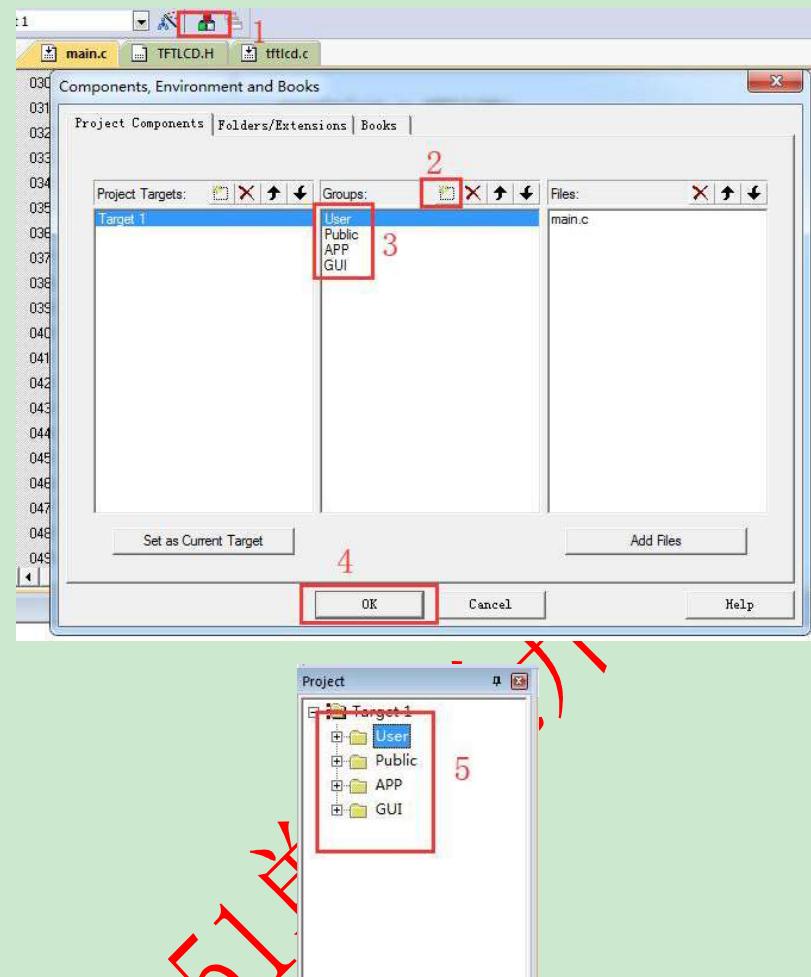
User 文件夹：用于存放 51 单片机公共文件，如串口驱动、延时等。

template.uvproj 文件：工程文件，直接使用 KEIL C51 打开。

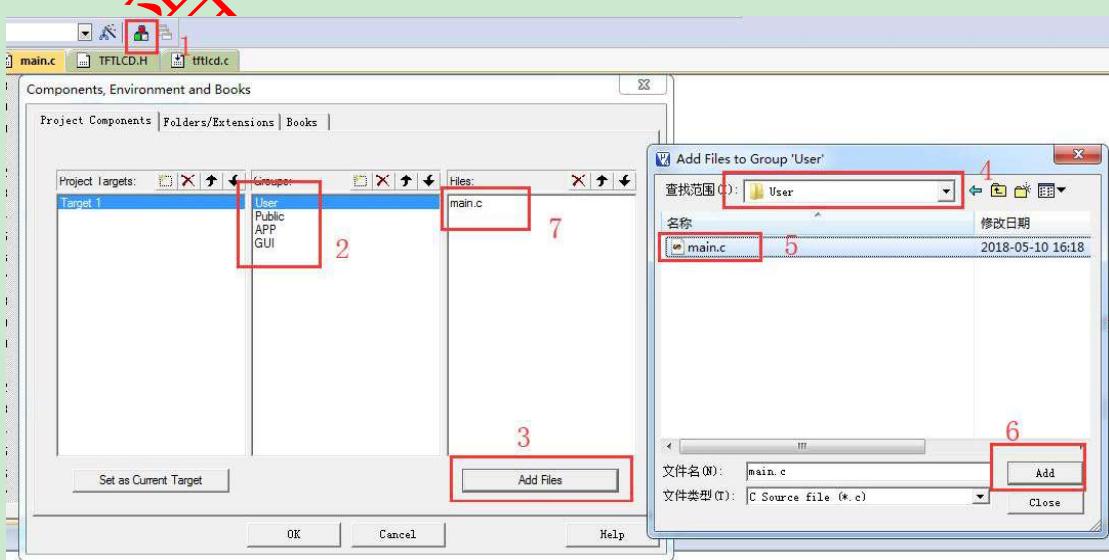
如何在 KEIL C51 内创建分组的工程呢？

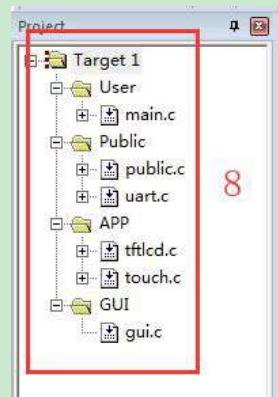
1) 首先在工程文件夹内创建上述文件夹目录，然后打开工程。选择项目组管

理快捷工具，按照创建的文件夹名新建分组，然后在工程中就会出现我们新建的几个组。具体操作步骤如下所示：



2) 将源文件添加到对应的分组内，直到所有源文件都添加到对应的工程分组中，~~操作步骤如下所示：~~





下面我们就来介绍下里面的一些重要函数，其他部分程序大家可以打开工程查看。

(1) TFTLCD 管脚定义

TFTLCD 控制管脚定义在 tftlcd.h 头文件内，如下所示：

```
//TFTLCD 彩屏数据控制端口定义
#define TFT_DATAPORTH P1
#define TFT_DATAPORTL P0

sbit TFT_CS = P2^7;
sbit TFT_RST = P3^3;
sbit TFT_RS = P2^6;
sbit TFT_WR = P2^5;
sbit TFT_RD = P3^2;
```

(2) TFTLCD 读写函数

```
//写寄存器函数
//cmd:寄存器值
void LCD_WriteCmd(u16 cmd)
{
#ifndef TFT22_ILI9225B
    TFT_WR=1;
    TFT_CS=0;
    TFT_RD=1;
    TFT_RS=0;

    TFT_DATAPORTH=cmd>>8;
    TFT_WR=0;
    TFT_WR=1;

    TFT_DATAPORTH=cmd&0x00ff;
    TFT_WR=0;
    TFT_WR=1;
}
```

```
TFT_CS=1;  
#endif  
  
#ifdef TFT24_ST7781R  
    TFT_CS=0;  
    TFT_RS=0;  
    TFT_DATAPORTH=cmd>>8;  
    TFT_DATAPORTL=cmd;  
    TFT_WR=0;  
    TFT_WR=1;  
    TFT_CS=1;  
#endif  
  
#ifdef TFT26_R61509V  
    TFT_CS=0;  
    TFT_RS=0;  
    TFT_DATAPORTH=cmd>>8;  
    TFT_DATAPORTL=cmd;  
    TFT_WR=0;  
    TFT_WR=1;  
    TFT_CS=1;  
#endif  
  
#ifdef TFT26_ILI9325D  
    TFT_CS=0;  
    TFT_RS=0;  
    TFT_DATAPORTH=cmd>>8;  
    TFT_DATAPORTL=cmd;  
    TFT_WR=0;  
    TFT_WR=1;  
    TFT_CS=1;  
#endif  
}  
  
//写数据  
//dat:要写入的值  
void LCD_WriteData(u16 dat)  
{  
#ifdef TFT22_ILI9225B  
    TFT_WR=1;  
    TFT_CS=0;  
    TFT_RD=1;  
    TFT_RS=1;
```

```
TFT_DATAPORTH=dat>>8;
TFT_WR=0;
TFT_WR=1;
TFT_DATAPORTH=dat&0x00ff;
TFT_WR=0;
TFT_WR=1;

TFT_CS=1;
#endif

#define TFT24_ST7781R
TFT_CS=0;
TFT_RS=1;
TFT_DATAPORTH=dat>>8;
TFT_DATAPORTL=dat;
TFT_WR=0;
TFT_WR=1;
TFT_CS=1;
#endif

#define TFT26_R61509V
TFT_CS=0;
TFT_RS=1;
TFT_DATAPORTH=dat>>8;
TFT_DATAPORTL=dat;
TFT_WR=0;
TFT_WR=1;
TFT_CS=1;
#endif

#define TFT26_ILI9325D
TFT_CS=0;
TFT_RS=1;
TFT_DATAPORTH=dat>>8;
TFT_DATAPORTL=dat;
TFT_WR=0;
TFT_WR=1;
TFT_CS=1;
#endif
}

void LCD_WriteData_Color(u16 color)
{
```

```
#ifdef TFT22_ILI9225B
    TFT_WR=1;
    TFT_CS=0;
    TFT_RD=1;
    TFT_RS=1;

    TFT_DATAPORTH=color>>8;
    TFT_WR=0;
    TFT_WR=1;
    TFT_DATAPORTH=color&0x00ff;
    TFT_WR=0;
    TFT_WR=1;

    TFT_CS=1;
#endif

#ifndef TFT24_ST7781R
    TFT_CS=0;
    TFT_RS=1;
    TFT_DATAPORTH=color>>8;
    TFT_DATAPORTL=color;
    TFT_WR=0;
    TFT_WR=1;
    TFT_CS=1;
#endif

#ifndef TFT26_R61509V
    TFT_CS=0;
    TFT_RS=1;
    TFT_DATAPORTH=color>>8;
    TFT_DATAPORTL=color;
    TFT_WR=0;
    TFT_WR=1;
    TFT_CS=1;
#endif

#ifndef TFT26_ILI9325D
    TFT_CS=0;
    TFT_RS=1;
    TFT_DATAPORTH=color>>8;
    TFT_DATAPORTL=color;
    TFT_WR=0;
    TFT_WR=1;
    TFT_CS=1;

```

```
#endif  
}  
  
void LCD_WriteCmdData(u16 cmd, u16 dat)  
{  
    LCD_WriteCmd(cmd);  
    LCD_WriteData(dat);  
}  
  
//读数据  
//返回值:读到的值  
u16 LCD_ReadData(void)  
{  
    u16 ram;  
  
    TFT_DATAPORTH=0xff;  
    TFT_DATAPORTL=0xff;  
  
#ifdef TFT22_ILI9225B  
    TFT_CS=0;  
    TFT_RS=1;  
    TFT_RD=0;  
    _nop_();  
    ram=TFT_DATAPORTH;  
    ram<<=8;  
  
    TFT_RD=1;  
    TFT_RD=0;  
    _nop_();  
    ram|=TFT_DATAPORTH;  
    TFT_RD=1;  
    TFT_CS=1;  
#endif  
  
#ifdef TFT24_ST7781R  
    TFT_CS=0;  
    TFT_RS=1;  
    TFT_RD=0;  
    _nop_();  
    ram=TFT_DATAPORTH;  
    ram<<=8;  
    ram|=TFT_DATAPORTL;  
    TFT_RD=1;  
    TFT_CS=1;
```

```
#endif

#define TFT26_R61509V
    TFT_CS=0;
    TFT_RS=1;
    TFT_RD=0;
    _nop_();
    ram=TFT_DATAPORTH;
    ram<<=8;
    ram|=TFT_DATAPORTL;
    TFT_RD=1;
    TFT_CS=1;
#endif

#define TFT26_ILI9325D
    TFT_CS=0;
    TFT_RS=1;
    TFT_RD=0;
    _nop_();
    ram=TFT_DATAPORTH;
    ram<<=8;
    ram|=TFT_DATAPORTL;
    TFT_RD=1;
    TFT_CS=1;
#endif

    return ram;
}
```

读写函数是根据前面介绍的 8080 时序编写，对于时序图的介绍我们不再重复。因为有些 TFTLCD 屏是 8 位数据口，为了能够兼容其他屏，这里使用了条件编译，关于条件编译的工作原理在 LCD1602 章节已经介绍过。这些宏定义可在 tftlcd.h 头文件中查看，我们以 2.6 寸的 R61509V 彩屏为例，只需将对应彩屏驱动的宏定义前面的 “//” 删除即定义 TFT26_R61509V，这样程序中就可以编译定义 TFT26_R61509V 的程序段。大家根据自己手上彩屏的驱动型号来定义程序中的宏，对应的宏定义，如下所示：

```
//定义 LCD 彩屏的驱动类型 可根据自己手上的彩屏背面型号来选择打开哪种驱动
//#define TFT22_ILI9225B
//#define TFT22_ILI9340
//#define TFT24_ST7781R
#define TFT26_R61509V
```

```
//#define TFT26_ILI9325D
```

(3) TFTLCD 初始化函数

要让 TFTLCD 显示，还需要初始化序列，即 TFT 彩屏厂家提供的 TFTLCD 寄存器设置值。将这些数值写入到 TFTLCD 内对应的命令寄存器中即可，所以还需要编写 TFTLCD 写命令和写数据等函数。TFTLCD 初始化代码如下：

```
void TFTLCD_Init(void)
{
    TFT_RST=1;
    delay_ms(100);

    TFT_RST=0;
    delay_ms(100);

    TFT_RST=1;
    delay_ms(100);

#endif TFT22_ILI9225B
    LCD_WriteCmd(0X0000);
    tftlcd_data. id=LCD_ReadData();
#endif

#endif TFT24_ST7781R
    LCD_WriteCmd(0X0000);
    tftlcd_data. id=LCD_ReadData();
#endif

#endif TFT26_R61509V
    LCD_WriteCmd(0X0000);
    tftlcd_data. id=LCD_ReadData();
#endif

#endif TFT26_ILI9325D
    LCD_WriteCmd(0X0000);
    tftlcd_data. id=LCD_ReadData();
#endif

    printf(" LCD ID:%x\r\n", tftlcd_data. id); //打印 LCD ID

#endif TFT22_ILI9225B
    LCD_WriteCmdData(0x0001, 0x011C); // set SS and NL bit
```

```
LCD_WriteCmdData(0x0002, 0x0100); // set 1 line inversion
LCD_WriteCmdData(0x0003, 0x1030); // set GRAM write direction and
BGR=1.
LCD_WriteCmdData(0x0008, 0x0808); // set BP and FP
LCD_WriteCmdData(0x000C, 0x0000); // RGB interface setting
R0Ch=0x0110 for RGB 18Bit and R0Ch=0111for RGB16Bit
LCD_WriteCmdData(0x000F, 0x0B01); // Set frame rate
LCD_WriteCmdData(0x0020, 0x0000); // Set GRAM Address
LCD_WriteCmdData(0x0021, 0x0000); // Set GRAM Address
//*****Power On sequence *****/
delay_ms(50); // Delay 50ms
LCD_WriteCmdData(0x0010, 0x0A00); // Set SAP, DSTB, STB
LCD_WriteCmdData(0x0011, 0x1038); // Set APON, PON, AON, VCI1EN, VC
delay_ms(50); // Delay 50ms
LCD_WriteCmdData(0x0012, 0x2121); // Internal reference voltage=
Vci;
LCD_WriteCmdData(0x0013, 0x007A); // Set GVDD
LCD_WriteCmdData(0x0014, 0x595c); // Set VCOMH/VCOML voltage
//----- Set GRAM area -----//
LCD_WriteCmdData(0x0030, 0x0000);
LCD_WriteCmdData(0x0031, 0x00DB);
LCD_WriteCmdData(0x0032, 0x0000);
LCD_WriteCmdData(0x0033, 0x0000);
LCD_WriteCmdData(0x0034, 0x00DB);
LCD_WriteCmdData(0x0035, 0x0000);
LCD_WriteCmdData(0x0036, 0x00AF);
LCD_WriteCmdData(0x0037, 0x0000);
LCD_WriteCmdData(0x0038, 0x00DB);
LCD_WriteCmdData(0x0039, 0x0000);

// ----- Adjust the Gamma Curve -----//
LCD_WriteCmdData(0x0050, 0x0000);
LCD_WriteCmdData(0x0051, 0x0704);
LCD_WriteCmdData(0x0052, 0x0C08);
LCD_WriteCmdData(0x0053, 0x0502);
LCD_WriteCmdData(0x0054, 0x080C);
LCD_WriteCmdData(0x0055, 0x0407);
LCD_WriteCmdData(0x0056, 0x0000);
LCD_WriteCmdData(0x0057, 0x0205);
LCD_WriteCmdData(0x0058, 0x0000);
LCD_WriteCmdData(0x0059, 0x0000);

delay_ms(50); // Delay 50ms
LCD_WriteCmdData(0x0007, 0x1017);
```

```
#endif

#ifndef TFT24_ST7781R
//ST7781R_HSD2.4
//-----Display Control
Setting-----//
    LCD_WriteCmdData(0x0001, 0x0100); //Output Direct
    LCD_WriteCmdData(0x0002, 0x0700); //Line Inversion
    LCD_WriteCmdData(0x0003, 0x5030); //Entry Mode (262K, BGR)
    LCD_WriteCmdData(0x0004, 0x0000); //Resize Control
    LCD_WriteCmdData(0x0008, 0x0302); //Display Control2 (Porch
Setting)
    LCD_WriteCmdData(0x0009, 0x0000); //Display Control3
    LCD_WriteCmdData(0x000A, 0x0000); //Display Control4
    LCD_WriteCmdData(0x000C, 0x0000); // RGB Display Interface
Control1
    LCD_WriteCmdData(0x000D, 0x0000); //Frame Marker Position
    LCD_WriteCmdData(0x000F, 0x0000); // RGB Display Interface
Control2
    //-----End Display Control
setting-----//
    delay_ms(100);
    //----- Power Control Registers
Initial -----//
    LCD_WriteCmdData(0x0010, 0x10E0); //Power Control1
    //-----End Power Control Registers
Initial -----//
    delay_ms(100); //Delay 100ms
    //-----Gamma Cluster
Setting-----//
    LCD_WriteCmdData(0x0030, 0x0000);
    LCD_WriteCmdData(0x0031, 0x0406);
    LCD_WriteCmdData(0x0032, 0x0302);
    LCD_WriteCmdData(0x0035, 0x0006);
    LCD_WriteCmdData(0x0036, 0x0700);
    LCD_WriteCmdData(0x0037, 0x0000);
    LCD_WriteCmdData(0x0038, 0x0406);
    LCD_WriteCmdData(0x0039, 0x0302);
    LCD_WriteCmdData(0x003c, 0x0006);
    LCD_WriteCmdData(0x003d, 0x0700);
    //-----End Gamma
Setting-----//
    //-----Display Windows 240 X
```

```
320-----//  
    LCD_WriteCmdData(0x0020, 0x0000); // Horizontal Address Start  
Position  
    LCD_WriteCmdData(0x0021, 0x0000); // Vertical Address Start  
Position  
    LCD_WriteCmdData(0x0050, 0x0000); // Horizontal Address Start  
Position  
    LCD_WriteCmdData(0x0051, 0x00ef); // Horizontal Address End  
Position  
    LCD_WriteCmdData(0x0052, 0x0000); // Vertical Address Start  
Position  
    LCD_WriteCmdData(0x0053, 0x013f); // Vertical Address End  
Position  
    //-----End Display Windows 240 X  
320-----//  
    //-----Frame Rate  
Setting-----//  
    LCD_WriteCmdData(0x0060, 0xA700); //Gate scan control  
    LCD_WriteCmdData(0x0061, 0x0001); //Non-display Area setting  
    LCD_WriteCmdData(0x006A, 0x0000); //Vertical Scroll Control  
    LCD_WriteCmdData(0x0090, 0x0030); //RTNI setting  
    LCD_WriteCmdData(0x0095, 0x021E); //Panel Interface Control 4  
    //-----END Frame Rate  
setting-----//  
    //-----Partial Image Display  
Initial-----//  
    LCD_WriteCmdData(0x0080, 0x0000); // Partial Image 1 Display  
Position  
    LCD_WriteCmdData(0x0081, 0x0000); // Partial Image 1 Area (Start  
Line)  
    LCD_WriteCmdData(0x0082, 0x0000); // Partial Image 1 Area (End  
Line)  
    LCD_WriteCmdData(0x0083, 0x0000); // Partial Image 2 Display  
Position  
    LCD_WriteCmdData(0x0084, 0x0000); // Partial Image 2 Area (Start  
Line)  
    LCD_WriteCmdData(0x0085, 0x0000); // Partial Image 2 Area (End  
Line)  
    //-----END Partial Image Display  
Initial -----//  
    //----- Power Supply Startup 1  
Setting-----//  
    LCD_WriteCmdData(0x00FF, 0x0001); //CMD 2 Enable  
    LCD_WriteCmdData(0x00B0, 0x310E); //Power Control12
```

```
LCD_WriteCmdData(0x0OFF, 0x0000); // CMD 2 Disable
//----- End Power Supply Startup 2
Setting-----//
delay_ms(100); //Delay 100ms
LCD_WriteCmdData(0x0007, 0x0133); //Display Control1
delay_ms(50); //Delay 50ms

LCD_WriteCmd(0x0022);

#endif

#define TFT26_R61509V
LCD_WriteCmd(0x0000);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0000);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0000);LCD_WriteData(0x0000);
delay_ms(100);

LCD_WriteCmd(0x0400);LCD_WriteData(0x6200);
LCD_WriteCmd(0x0008);LCD_WriteData(0x0808);

LCD_WriteCmd(0x0300);LCD_WriteData(0x0C00);
LCD_WriteCmd(0x0301);LCD_WriteData(0x5A0B);
LCD_WriteCmd(0x0302);LCD_WriteData(0x0906);
LCD_WriteCmd(0x0303);LCD_WriteData(0x1017);
LCD_WriteCmd(0x0304);LCD_WriteData(0x2300);
LCD_WriteCmd(0x0305);LCD_WriteData(0x1700);
LCD_WriteCmd(0x0306);LCD_WriteData(0x6309);
LCD_WriteCmd(0x0307);LCD_WriteData(0x0C09);
LCD_WriteCmd(0x0308);LCD_WriteData(0x010C);
LCD_WriteCmd(0x0309);LCD_WriteData(0x2232);

LCD_WriteCmd(0x0010);LCD_WriteData(0x0016);
LCD_WriteCmd(0x0011);LCD_WriteData(0x0101);
LCD_WriteCmd(0x0012);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0013);LCD_WriteData(0x0001);

LCD_WriteCmd(0x0100);LCD_WriteData(0x0330);
LCD_WriteCmd(0x0101);LCD_WriteData(0x0336);
LCD_WriteCmd(0x0103);LCD_WriteData(0x1000);

LCD_WriteCmd(0x0280);LCD_WriteData(0x6100);
LCD_WriteCmd(0x0102);LCD_WriteData(0xBBB4);
delay_ms(100);

LCD_WriteCmd(0x0001);LCD_WriteData(0x0100);
```

```
LCD_WriteCmd(0x0002);LCD_WriteData(0x0100);
LCD_WriteCmd(0x0003);LCD_WriteData(0x1030);
LCD_WriteCmd(0x0009);LCD_WriteData(0x0001);
LCD_WriteCmd(0x000C);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0090);LCD_WriteData(0x0800);
LCD_WriteCmd(0x000F);LCD_WriteData(0x0000);

LCD_WriteCmd(0x0210);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0211);LCD_WriteData(0x00EF);
LCD_WriteCmd(0x0212);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0213);LCD_WriteData(0x018F); //;400

LCD_WriteCmd(0x0500);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0501);LCD_WriteData(0x0000);
LCD_WriteCmd(0x0502);LCD_WriteData(0x005F);

LCD_WriteCmd(0x0401);LCD_WriteData(0x0001);
LCD_WriteCmd(0x0404);LCD_WriteData(0x0000);
delay_ms(100);

LCD_WriteCmd(0x0007);LCD_WriteData(0x0100);
delay_ms(100);

LCD_WriteCmd(0x0202);
#endif

#ifndef TFT26_ILI9325D
//***** Start Initial Sequence *****/
LCD_WriteCmdData(0x0001, 0x0100); // set SS and SM bit
LCD_WriteCmdData(0x0002, 0x0200); // set 1 line inversion
LCD_WriteCmdData(0x0003, 0x1030); // set GRAM write direction and
BGR=1.
LCD_WriteCmdData(0x0004, 0x0000); // Resize register
LCD_WriteCmdData(0x0008, 0x0202); // set theback porch and front
porch
LCD_WriteCmdData(0x0009, 0x0000); // set non-display area refresh
cycle ISC[3:0]
LCD_WriteCmdData(0x000A, 0x0000); // FMARK function
LCD_WriteCmdData(0x000C, 0x0000); // RGB interface setting
LCD_WriteCmdData(0x000D, 0x0000); // Frame marker Position
LCD_WriteCmdData(0x000F, 0x0000); // RGB int erface polarity
//*****Power On sequence *****/
LCD_WriteCmdData(0x0010, 0x0000); // SAP, BT[3:0], AP, DSTB, SLP,
```

```
STB
    LCD_WriteCmdData(0x0011, 0x0007); // DC1[2:0], DC0[2:0], VC[2:0]
    LCD_WriteCmdData(0x0012, 0x0000); // VREG1OUT voltage
    LCD_WriteCmdData(0x0013, 0x0000); // VDV[4:0] for VCOM amplitude
    delay_ms(10); // Dis-charge capacitor
power voltage
    LCD_WriteCmdData(0x0010, 0x1690); // SAP, BT[3:0], AP, DSTB, SLP,
STB
    LCD_WriteCmdData(0x0011, 0x0227); // R11h=0x 0221 at VCI=3.3V,
DC1[2:0], DC0[2:0], VC[2:0]
    delay_ms(10); // Delay 50ms
    LCD_WriteCmdData(0x0012, 0x008D); // External reference voltage=
Vci;
    delay_ms(10); // Delay 50ms
    LCD_WriteCmdData(0x0013, 0x1200); // VDV[4:0] for VCOM amplitude
    LCD_WriteCmdData(0x0029, 0x0005); // VCM[5:0] for VCOMH
    LCD_WriteCmdData(0x002B, 0x000C); // Frame Rate = 91Hz
    delay_ms(10); // Delay 50ms
    LCD_WriteCmdData(0x0020, 0x0000); // GRAM horizontal Address
    LCD_WriteCmdData(0x0021, 0x0000); // GRAM Vertical Address
// ----- Adjust the Gamma Curve -----
//a-Si TFT LCD Single Chip Driver
//240RGBx320 Resolution and 262K color ILI9325D
//The information contained herein is the exclusive property of
ILI Technology Corp. and shall not be distributed, reproduced, or
disclosed in
    //whole or in part without prior written permission of ILI
Technology Corp.
    //Page 24 of 26 V0.14
    LCD_WriteCmdData(0x0030, 0x0000);
    LCD_WriteCmdData(0x0031, 0x0303);
    LCD_WriteCmdData(0x0032, 0x0103);
    LCD_WriteCmdData(0x0035, 0x0103);
    LCD_WriteCmdData(0x0036, 0x0004);
    LCD_WriteCmdData(0x0037, 0x0406);
    LCD_WriteCmdData(0x0038, 0x0404);
    LCD_WriteCmdData(0x0039, 0x0707);
    LCD_WriteCmdData(0x003C, 0x0301);
    LCD_WriteCmdData(0x003D, 0x0004);
//----- Set GRAM area -----
    LCD_WriteCmdData(0x0050, 0x0000); // Horizontal GRAM Start
Address
    LCD_WriteCmdData(0x0051, 0x00EF); // Horizontal GRAM End Address
    LCD_WriteCmdData(0x0052, 0x0000); // Vertical GRAM Start Address
```

```

LCD_WriteCmdData(0x0053, 0x013F); // Vertical GRAM Start Address
LCD_WriteCmdData(0x0060, 0xA700); // Gate Scan Line
LCD_WriteCmdData(0x0061, 0x0001); // NDL, VLE, REV
LCD_WriteCmdData(0x006A, 0x0000); // set scrolling line
//----- Partial Display Control -----
LCD_WriteCmdData(0x0080, 0x0000);
LCD_WriteCmdData(0x0081, 0x0000);
LCD_WriteCmdData(0x0082, 0x0000);
LCD_WriteCmdData(0x0083, 0x0000);
LCD_WriteCmdData(0x0084, 0x0000);
LCD_WriteCmdData(0x0085, 0x0000);
//----- Panel Control -----
LCD_WriteCmdData(0x0090, 0x0010);
LCD_WriteCmdData(0x0092, 0x0600);
LCD_WriteCmdData(0x0007, 0x0133); // 262K color and display ON

LCD_WriteCmd(0x0022);

#endif

LCD_Display_Dir(0); //0: 竖屏 1: 横屏 默认竖屏
LCD_Clear(BACK_COLOR);
}

```

该函数中的 `LCD_WriteCmd`、`LCD_WriteData` 和 `LCD_WriteData_Color` 为写命令、写数据函数，其实现过程很简单，在前面已经列出。

下面我们再来介绍下 `tftlcd.h` 中另一个重要的结构体：

```

typedef struct
{
    u16 width; //LCD 宽度
    u16 height; //LCD 高度
    u16 id; //LCD ID
    u8 dir; //LCD 显示方向
} _tftlcd_data;
extern _tftlcd_data tftlcd_data; //管理 TFTLCD 重要参数

```

该结构体用于保存一些 TFTLCD 重要参数信息，比如 LCD 的尺寸、LCD ID（驱动 IC 型号）、LCD 显示方向，当然你也可以在我们这个结构体内添加其他 LCD 相关参数变量。通过对此结构体的管理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。所以大家今后要多学习使用结构体管理不同属性。

初始化函数内我们还使用了条件编译语句来兼容多种彩屏驱动程序。比如：

```
#ifdef TFT22_ILI9225B  
    .... 对应 TFT22_ILI9225B 驱动的操作代码  
#endif  
  
#ifdef TFT22_ILI9340  
    .... 对应 TFT22_ILI9340 驱动的操作代码  
#endif  
  
#ifdef TFT26_R61509V  
    .... 对应 TFT26_R61509V 驱动的操作代码  
#endif  
  
#ifdef TFT26_ILI9325D  
    .... 对应 TFT26_ILI9325D 驱动的操作代码  
#endif
```

本章所介绍的 TFTLCD 驱动 IC 是 TFT26_R61509V，所以要对这部分代码操作，只需在 tftlcd.h 文件开始处开启这个宏定义，其他彩屏的宏定义就关闭它。如果您手上使用的是其他驱动类型屏，~~对应型号~~ 打开即可，如下：（后面如果又增加了屏，使用同样方法来兼容）

// 定义 LCD 彩屏的驱动类型 可根据自己手上的彩屏背面型号来选择打开哪种驱动

```
//#define TFT22_ILI9225B  
  
//#define TFT22_ILI9340  
  
//#define TFT24_ST7781R  
  
#define TFT26_R61509V  
  
//#define TFT26_ILI9325D
```

在初始化函数最后部分还调用了 LCD_Display_Dir 和 LCD_Clear 函数，LCD_Display_Dir 函数用于切换 TFTLCD 显示方向（横屏和竖屏），默认我们设置为竖屏，代码如下：

```
// 设置 LCD 显示方向  
// dir: 0, 竖屏; 1, 横屏  
void LCD_Display_Dir(u8 dir)  
{  
    if(dir==0) // 默认竖屏方向
```

```
{  
#ifdef TFT22_ILI9225B  
    LCD_WriteCmdData(0x0001, 0x011C); //Output Direct  
    LCD_WriteCmdData(0x0003, 0x1030); //设置彩屏显示方向的寄存器  
#endif  
  
#ifdef TFT24_ST7781R  
    LCD_WriteCmdData(0x0001, 0x0100); //Output Direct  
    LCD_WriteCmd(0x0003); //设置彩屏显示方向的寄存器  
    LCD_WriteData(0x5030);  
#endif  
  
#ifdef TFT26_R61509V  
    LCD_WriteCmdData(0x0001, 0x0100); //Output Direct  
    LCD_WriteCmdData(0x0003, 0x1030); //设置彩屏显示方向的寄存器  
#endif  
  
#ifdef TFT26_ILI9325D  
    LCD_WriteCmdData(0x0001, 0x0100); //Output Direct  
    LCD_WriteCmdData(0x0003, 0x1030); //设置彩屏显示方向的寄存器  
#endif  
  
    tftlcd_data.height=HEIGHT;  
    tftlcd_data.width=WIDTH;  
  
    tftlcd_data.dir=0;  
}  
else  
{  
#ifdef TFT22_ILI9225B  
    LCD_WriteCmdData(0x0001, 0x031C); //Output Direct  
    LCD_WriteCmdData(0x0003, 0x1038); //设置彩屏显示方向的寄存器  
#endif  
#endif  
  
#ifdef TFT24_ST7781R  
    LCD_WriteCmdData(0x0001, 0x0000); //Output Direct  
    LCD_WriteCmd(0x0003); //设置彩屏显示方向的寄存器  
    LCD_WriteData(0x5038);  
#endif
```

```

#define TFT26_R61509V
    LCD_WriteCmdData(0x0001, 0x0000); //Output Direct
    LCD_WriteCmdData(0x0003, 0x1038); //设置彩屏显示方向的寄
存器
#endif

#define TFT26_ILI9325D
    LCD_WriteCmdData(0x0001, 0x0000); //Output Direct
    LCD_WriteCmdData(0x0003, 0x1038); //设置彩屏显示方向的寄
存器
#endif

    tftlcd_data.height=WIDTH;
    tftlcd_data.width=HEIGHT;

    tftlcd_data.dir=1;
}
}

```

横屏和竖屏的控制只需要对 TFTLCD 显示方向寄存器进行操作（寄存器可通过彩屏数据手册查找），然后再把他们的 X 和 Y 轴像素调换即可。

LCD_Clear 函数用于清屏，函数入口参数用于选择清屏的颜色，常用的颜色都在 tftlcd.h 文件内进行了宏定义，直接调用宏即可。清屏函数实现过程很简单，其实就是设置 TFTLCD 显示窗口，然后写入

tftlcd_data.width*tftlcd_data.height 个颜色数据即可。设置窗口函数代码如下：

```

//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.
void LCD_Set_Window(u16 sx, u16 sy, u16 width, u16 height)
{
#define TFT22_ILI9225B
    if(tftlcd_data.dir==0)
    {
        LCD_WriteCmdData(0x0037, sx);
        LCD_WriteCmdData(0x0036, width);
        LCD_WriteCmdData(0x0039, sy);
        LCD_WriteCmdData(0x0038, height);

        LCD_WriteCmdData(0x0020, sx);
        LCD_WriteCmdData(0x0021, sy);
    }
}

```

```
        LCD_WriteCmd(0x0022) ;
    }
else
{
    LCD_WriteCmdData(0x0039, sx) ;
    LCD_WriteCmdData(0x0038, width) ;
    LCD_WriteCmdData(0x0037, sy) ;
    LCD_WriteCmdData(0x0036, height) ;

    LCD_WriteCmdData(0x0021, sx) ;
    LCD_WriteCmdData(0x0020, sy) ;

    LCD_WriteCmd(0x0022) ;
}
#endif

#ifndef TFT24_ST7781R
if(tftlcd_data.dir==0)
{
    LCD_WriteCmdData(0x0050, sx); // Horizontal Address Start
Position
    LCD_WriteCmdData(0x0051, width); // Horizontal Address End
Position
    LCD_WriteCmdData(0x0052, sy); // Vertical Address Start
Position
    LCD_WriteCmdData(0x0053, height); // Vertical Address End
Position

    LCD_WriteCmdData(0x0020, sx); // Horizontal Address Start
Position
    LCD_WriteCmdData(0x0021, sy); // Vertical Address Start
Position
    LCD_WriteCmd(0x0022);
}
else
{
    LCD_WriteCmdData(0x0052, sx); // Horizontal Address Start
Position
    LCD_WriteCmdData(0x0053, width); // Horizontal Address End
Position
    LCD_WriteCmdData(0x0050, sy); // Vertical Address Start
Position
    LCD_WriteCmdData(0x0051, height); // Vertical Address End
}
```

Position

```
    LCD_WriteCmdData(0x0021, sx); // Horizontal Address Start
Position
    LCD_WriteCmdData(0x0020, sy); // Vertical Address Start
Position
    LCD_WriteCmd(0x0022);
}
#endif

#ifndef TFT26_R61509V
if(tftlcd_data.dir==0)
{
    LCD_WriteCmd(0x0210);    LCD_WriteData(sx);
    LCD_WriteCmd(0x0211);    LCD_WriteData(width);
    LCD_WriteCmd(0x0212);    LCD_WriteData(sy);
    LCD_WriteCmd(0x0213);    LCD_WriteData(height);

    LCD_WriteCmd(0x0200);    LCD_WriteData(sx);
    LCD_WriteCmd(0x0201);    LCD_WriteData(sy);

    LCD_WriteCmd(0x0202);
}
else
{
    LCD_WriteCmd(0x0212);    LCD_WriteData(sx);
    LCD_WriteCmd(0x0213);    LCD_WriteData(width);
    LCD_WriteCmd(0x0210);    LCD_WriteData(sy);
    LCD_WriteCmd(0x0211);    LCD_WriteData(height);

    LCD_WriteCmd(0x0201);    LCD_WriteData(sx);
    LCD_WriteCmd(0x0200);    LCD_WriteData(sy);

    LCD_WriteCmd(0x0202);
}
#endif

#ifndef TFT26_ILI9325D
if(tftlcd_data.dir==0)
{
    LCD_WriteCmdData(0x0050, sx); // Horizontal GRAM Start
Address
    LCD_WriteCmdData(0x0051, width); // Horizontal GRAM End
Address
```

```

LCD_WriteCmdData(0x0052, sy); // Vertical GRAM Start Address
LCD_WriteCmdData(0x0053, height); // Vertical GRAM Start
Address

LCD_WriteCmdData(0x0020, sx); // GRAM horizontal Address
LCD_WriteCmdData(0x0021, sy); // GRAM Vertical Address
LCD_WriteCmd(0x0022);
}

else
{
    LCD_WriteCmdData(0x0052, sx); // Horizontal GRAM Start
Address
    LCD_WriteCmdData(0x0053, width); // Horizontal GRAM End
Address
    LCD_WriteCmdData(0x0050, sy); // Vertical GRAM Start Address
    LCD_WriteCmdData(0x0051, height); // Vertical GRAM Start
Address

    LCD_WriteCmdData(0x0021, sx); // GRAM horizontal Address
    LCD_WriteCmdData(0x0020, sy); // GRAM Vertical Address
    LCD_WriteCmd(0x0022);
}
#endif

}

```

其实就是在 TFTLCD 的 X 方向和 Y 方向寄存器内写入窗口值。我们讲解的 R61509V 的 X 方向和 Y 方向命令（寄存器）是 0x0210, 0x0211 和 0x0212, 0x0213，写完窗口值还需要将这些值写入到 TFTLCD 的 GRAM 内，命令是 0x0202。这些命令也是通过彩屏数据手册查找，这部分也是彩屏厂家提供的，可以不作了解。在初始化函数内使用了 printf 函数打印 TFTLCD 的 ID，所以在主函数内需要对串口初始化，否则将导致程序死在 printf 里面，如果不想用 printf，那么请注释掉它。

(4) TFTLCD 显示函数

上述几个函数完成后，我们就可以使用 TFTLCD 了，要在 TFTLCD 上显示内容，我们需要编写对应的显示函数，比如要显示一个点，那么就编写一个点的函数。TFTLCD 显示函数如下：

```

//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 xState, u16 yState, u16 xEnd, u16 yEnd, u16 color)
{
    u16 temp=0;

    if((xState > xEnd) || (yState > yEnd))
    {
        return;
    }
    LCD_Set_Window(xState, yState, xEnd, yEnd);
    xState = xEnd - xState + 1;
    yState = yEnd - yState + 1;

    while(xState--)
    {
        temp = yState;
        while (temp--)
        {
            LCD_WriteData_Color(color);
        }
    }
}

//在指定区域内填充指定颜色块
//(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_Color_Fill(u16 sx, u16 sy, u16 ex, u16 ey, u16 *color)
{
    u16 height, width;
    u16 i, j;
    width=ex-sx+1;           //得到填充的宽度
    height=ey-sy+1;           //高度
    LCD_Set_Window(sx, sy, ex, ey);
    for(i=0;i<height;i++)
    {
        for(j=0;j<width;j++)
        {
            LCD_WriteData_Color(color[i*width+j]);
        }
    }
}

//画点
//x, y:坐标
//FRONT_COLOR:此点的颜色
```

```
void LCD_DrawPoint(u16 x, u16 y)
{
    LCD_Set_Window(x, y, x, y); //设置点的位置
    LCD_WriteData_Color(FRONT_COLOR);
}

//快速画点
//x, y:坐标
//color:颜色
void LCD_DrawFRONT_COLOR(u16 x, u16 y, u16 color)
{
    LCD_Set_Window(x, y, x, y);
    LCD_WriteData_Color(color);
}

//画一个大点
//2*2 的点
void LCD_DrawBigPoint(u16 x, u16 y, u16 color)
{
    LCD_DrawFRONT_COLOR(x, y, color); //中心点
    LCD_DrawFRONT_COLOR(x+1, y, color);
    LCD_DrawFRONT_COLOR(x, y+1, color);
    LCD_DrawFRONT_COLOR(x+1, y+1, color);
}

//画线
//x1, y1:起点坐标
//x2, y2:终点坐标
void LCD_DrawLine(u16 x1, u16 y1, u16 x2, u16 y2)
{
    u16 t=0;
    int xerr=0, yerr=0, delta_x, delta_y, distance;
    int incx, incy, uRow, uCol;
    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
{
    LCD_DrawPoint(uRow, uCol); //画点
    xerr+=delta_x ;
    yerr+=delta_y ;
    if(xerr>distance)
    {
        xerr-=distance;
        uRow+=incx;
    }
    if(yerr>distance)
    {
        yerr-=distance;
        uCol+=incy;
    }
}
}

void LCD_DrawLine_Color(u16 x1, u16 y1, u16 x2, u16 y2, u16 color)
{
    u16 t;
    int xerr=0, yerr=0, delta_x, delta_y, distance;
    int incx, incy, uRow, uCol;
    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
    {
        LCD_DrawFRONT_COLOR(uRow, uCol, color); //画点
        xerr+=delta_x ;
        yerr+=delta_y ;
        if(xerr>distance)
        {
            xerr-=distance;
```

```
        uRow+=incx;
    }
    if(yerr>distance)
    {
        yerr-=distance;
        uCol+=incy;
    }
}

// 画一个十字的标记
// x: 标记的 X 坐标
// y: 标记的 Y 坐标
// color: 标记的颜色
void LCD_DrowSign(u16 x, u16 y, u16 color)
{
    u8 i;

    /* 画点 */
    LCD_Set_Window(x-1, y-1, x+1, y+1);
    for(i=0; i<9; i++)
    {
        LCD_WriteData_Color(color);
    }

    /* 画竖 */
    LCD_Set_Window(x-4, y, x+4, y);
    for(i=0; i<9; i++)
    {
        LCD_WriteData_Color(color);
    }

    /* 画横 */
    LCD_Set_Window(x, y-4, x, y+4);
    for(i=0; i<9; i++)
    {
        LCD_WriteData_Color(color);
    }
}

//画矩形
//(x1, y1), (x2, y2):矩形的对角坐标
void LCD_DrawRectangle(u16 x1, u16 y1, u16 x2, u16 y2)
```

```

{
    LCD_DrawLine(x1, y1, x2, y1);
    LCD_DrawLine(x1, y1, x1, y2);
    LCD_DrawLine(x1, y2, x2, y2);
    LCD_DrawLine(x2, y1, x2, y2);
}

//在指定位置画一个指定大小的圆
//(x, y) :中心点
//r      :半径
void LCD_Draw_Circle(u16 x0, u16 y0, u8 r)
{
    int a, b;
    int di;
    a=0;b=r;
    di=3-(r<<1);           //判断下个点位置的标志
    while(a<=b)
    {
        LCD_DrawPoint(x0+a, y0-b); //5
        LCD_DrawPoint(x0+b, y0-a); //0
        LCD_DrawPoint(x0+b, y0+a); //4
        LCD_DrawPoint(x0+a, y0+b); //6
        LCD_DrawPoint(x0-a, y0+b); //1
        LCD_DrawPoint(x0-b, y0+a); //2
        LCD_DrawPoint(x0-a, y0-b); //3
        LCD_DrawPoint(x0-b, y0-a); //7
        a++;
        //使用 Bresenham 算法画圆
        if(di<0) di +=4*a+6;
        else
        {
            di+=10+4*(a-b);
            b--;
        }
    }
}

//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);           //得到字体一个字符对应点阵集所占的字节数
num=num-' ';//得到偏移后的值 (ASCII 字库是从空格开始取模, 所以-' '就是对应字符的字库)
for(t=0;t<csize;t++)
{
    if(size==12)temp=ascii_1206[num][t];           //调用 1206 字体
    else if(size==16)temp=ascii_1608[num][t]; //调用 1608 字体
    else if(size==24)temp=ascii_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_DrawFRONT_COLOR(x, y, FRONT_COLOR);
        else if(mode==0)LCD_DrawFRONT_COLOR(x, y, BACK_COLOR);
        temp<<=1;
        y++;
        if(y>=tftlcd_data.height)return;           //超区域了
        if((y-y0)==size)
        {
            y=y0;
            x++;
            if(x>=tftlcd_data.width)return; //超区域了
            break;
        }
    }
}
}

//m^n 函数
//返回值:m^n 次方.
u32 LCD_Pow(u8 m, u8 n)
{
    u32 result=1;
    while(n--)result*=m;
    return result;
}

//显示数字, 高位为 0, 则不显示
//x, y :起点坐标
//len :数字的位数
//size:字体大小
//color:颜色
//num:数值(0~4294967295);
void LCD_ShowNum(u16 x, u16 y, u32 num, u8 len, u8 size)

```

```
{  
    u8 t, temp;  
    u8 enshow=0;  
    for(t=0;t<len;t++)  
    {  
        temp=(num/LCD_Pow(10, len-t-1))%10;  
        if(enshow==0&&t<(len-1))  
        {  
            if(temp==0)  
            {  
                LCD_ShowChar(x+(size/2)*t, y, ' ', size, 0);  
                continue;  
            } else enshow=1;  
        }  
        LCD_ShowChar(x+(size/2)*t, y, temp+' 0', size, 0);  
    }  
}  
  
//显示数字,高位为 0,还是显示  
//x, y:起点坐标  
//num:数值(0~999999999);  
//len:长度(即要显示的位数)  
//size:字体大小  
//mode:  
//[7]:0,不填充;1,填充 0.  
//[6:1]:保留  
//[0]:0,非叠加显示;1,叠加显示.  
void LCD_ShowxNum(u16 x, u16 y, u32 num, u8 len, u8 size, u8 mode)  
{  
    u8 t, temp;  
    u8 enshow=0;  
    for(t=0;t<len;t++)  
    {  
        temp=(num/LCD_Pow(10, len-t-1))%10;  
        if(enshow==0&&t<(len-1))  
        {  
            if(temp==0)  
            {  
                if(mode&0X80)LCD_ShowChar(x+(size/2)*t, y, ' 0', size, mode&0X01);  
                else LCD_ShowChar(x+(size/2)*t, y, ' ', size, mode&0X01);  
                continue;  
            }  
        }  
    }  
}
```

```

    }else ensshow=1;

}

LCD_ShowChar(x+(size/2)*t, y, temp+'0', size, mode&0X01);
}

}

//显示字符串
//x, y:起点坐标
//width, height:区域大小
//size:字体大小
//*p:字符串起始地址
void LCD_ShowString(u16 x, u16 y, u16 width, u16 height, u8 size, u8 *p)
{
    u8 x0=x;
    width+=x;
    height+=y;
    while((*p<='~')&&(*p>=' '))//判断是不是非法字符!
    {
        if(x>=width){x=x0;y+=size;}
        if(y>=height)break;//退出
        LCD_ShowChar(x, y, *p, size, 0);
        x+=size/2;
        p++;
    }
}

/***** */
***** */

*函数名: LCD_ShowFontHZ
*输入: x: 汉字显示的 X 坐标
*      * y: 汉字显示的 Y 坐标
*      * cn: 要显示的汉字
*      * wordColor: 文字的颜色
*      * backColor: 背景颜色
*输出:
*功能: 写二号楷体汉字
***** */
***** */

#endif
void LCD_ShowFontHZ(u16 x, u16 y, u8 *cn)
{
    u8 i, j, wordNum;
    u16 color;
    while (*cn != '\0')

```

```

{
    LCD_Set_Window(x, y, x+31, y+28);
    for (wordNum=0; wordNum<20; wordNum++)
    { //wordNum 扫描字库的字数
        if ((CnChar32x29[wordNum].Index[0]==*cn)
            &&(CnChar32x29[wordNum].Index[1]==*(cn+1)))
        {
            for(i=0; i<116; i++)
            { //MSK 的位数
                color=CnChar32x29[wordNum].Msk[i];
                for(j=0; j<8; j++)
                {
                    if((color&0x80)==0x80)
                    {
                        LCD_WriteData_Color(FRONT_COLOR);

                    }
                    else
                    {
                        LCD_WriteData_Color(BACK_COLOR);
                    }
                    color<<=1;
                } //for(j=0; j<8; j++) 结束
            }
        }
    } //for (wordNum=0; wordNum<20; wordNum++) 结束
    cn += 2;
    x += 32;
}
#endif

#if 0
void LCD_ShowFontHZ(u16 x, u16 y, u8 *cn)
{
    u8 i, j, wordNum;
    u16 color;
    u16 x0=x;
    u16 y0=y;
    while (*cn != '\0')
    {
        for (wordNum=0; wordNum<20; wordNum++)
        { //wordNum 扫描字库的字数

```

```

if ((CnChar32x29[wordNum].Index[0]==*cn)
    && (CnChar32x29[wordNum].Index[1]==*(cn+1)))
{
    for(i=0; i<116; i++)
    { //MSK 的位数
        color=CnChar32x29[wordNum].Msk[i];
        for(j=0; j<8; j++)
        {
            if((color&0x80)==0x80)
            {
                LCD_DrawFRONT_COLOR(x, y, FRONT_COLOR);
            }
            else
            {
                LCD_DrawFRONT_COLOR(x, y, BACK_COLOR);
            }
            color<<=1;
            x++;
            if((x-x0)==32)
            {
                x=x0;
                y++;
                if((y-y0)==29)
                {
                    y=y0;
                }
            }
        } //for(j=0; j<8; j++) 结束
    }
}

} //for (wordNum=0; wordNum<20; wordNum++) 结束
cn += 2;
x += 32;
x0=x;
}
}

#endif

void LCD_ShowPicture(u16 x, u16 y, u16 wide, u16 high, u8 *pic)
{
    u16 temp = 0;
    long tmp=0, num=0;
    LCD_Set_Window(x, y, x+wide-1, y+high-1);
}

```

```

num = wide * high*2 ;
do
{
    temp = pic[tmp + 1];
    temp = temp << 8;
    temp = temp | pic[tmp];
    LCD_WriteData_Color(temp); //逐点显示
    tmp += 2;
}
while(tmp < num);
}

```

我们给大家提供很多 TFTLCD 操作的 API 函数，在 tftlcd.h 文件内可查询到，如下：

```

void LCD_WriteCmd(u16 cmd);
void LCD_WriteData(u16 dat);
void LCD_WriteCmdData(u16 cmd, u16 dat);
void LCD_WriteData_Color(u16 color);

void TFTLCD_Init(void);           //初始化
void LCD_Set_Window(u16 sx, u16 sy, u16 width, u16 height);
void LCD_Clear(u16 Color);
    //清屏
void LCD_Fill(u16 xState, u16 yState, u16 xEnd, u16 yEnd, u16 color);
void LCD_Color_Fill(u16 sx, u16 sy, u16 ex, u16 ey, u16 *color);
void LCD_DrawPoint(u16 x, u16 y); //画点
void LCD_DrawFRONT_COLOR(u16 x, u16 y, u16 color);
void LCD_DrawBigPoint(u16 x, u16 y, u16 color);
u16 LCD_ReadPoint(u16 x, u16 y);
void LCD_DrawLine(u16 x1, u16 y1, u16 x2, u16 y2);
void LCD_DrawLine_Color(u16 x1, u16 y1, u16 x2, u16 y2, u16 color);
void LCD_DrowSign(u16 x, u16 y, u16 color);
void LCD_DrawRectangle(u16 x1, u16 y1, u16 x2, u16 y2);
void LCD_Draw_Circle(u16 x0, u16 y0, u8 r);
void LCD_ShowChar(u16 x, u16 y, u8 num, u8 size, u8 mode);
void LCD_ShowNum(u16 x, u16 y, u32 num, u8 len, u8 size);
void LCD_ShowxNum(u16 x, u16 y, u32 num, u8 len, u8 size, u8 mode);
void LCD_ShowString(u16 x, u16 y, u16 width, u16 height, u8 size, u8 *p);
void LCD_ShowFontHZ(u16 x, u16 y, u8 *cn);

void LCD_ShowPicture(u16 x, u16 y, u16 wide, u16 high, u8 *pic);

```

我们提供的这么多函数，其实很多函数都是调用画点函数 LCD_DrawPoint 完成的，这些函数的使用方法也很简单，我们就以显示字符函数 LCD_ShowChar

为例进行介绍（其他函数大家可参考对应的源代码）：

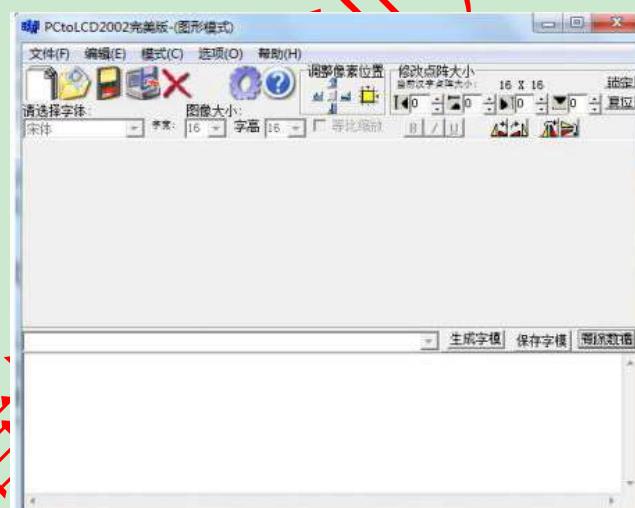
```
//在指定位置显示一个字符
//x, y:起始坐标
//num:要显示的字符：“--->”“~”
//size:字体大小 12/16/24
//mode:叠加方式(1)还是非叠加方式(0)
void LCD_ShowChar(u16 x, u16 y, u8 num, u8 size, u8 mode)
{
    u8 temp, t1, t;
    u16 y0=y;
    u8 csize=(size/8+((size%8)?1:0))*(size/2);           //得到字体一个字符对应点阵集所占的字节数
    num=num-' '; //得到偏移后的值（ASCII 字库是从空格开始取模，所以' '就是对应字符的字库）
    for(t=0;t<csize;t++)
    {
        if(size==12)temp=ascii_1206[num][t]; //调用 1206 字体
        else if(size==16)temp=ascii_1608[num][t]; //调用 1608 字体
        else if(size==24)temp=ascii_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_DrawFRONT_COLOR(x, y, FRONT_COLOR);
            else if(mode==0)LCD_DrawFRONT_COLOR(x, y, BACK_COLOR);
            temp<<=1;
            y++;
            if(y>=tftlcd_data.height) return; //超区域了
            if((y-y0)==size)
            {
                y=y0;
                x++;
                if(x>=tftlcd_data.width) return; //超区域了
                break;
            }
        }
    }
}
```

函数入口参数 x 和 y 用来设置显示的起始位置， num 是要显示的字符， size 用来选择显示字体大小，本实验支持 12/16/24 号的 ASCII 字符显示。 mode 用来设置是否支持叠加显示，为 0 表示不使用叠加，为 1 表示使用叠加显示。叠加方式显示多用于在显示的图片上显示字符，非叠加方式一般用于普通

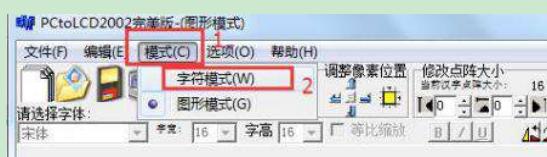
的显示。函数内我们用到了三个字符集点阵数据数组 ascii_1206、ascii_1608 和 ascii_2412。

下面我们重点介绍下如何让字符显示在 TFTLCD 模块上。要显示字符，我们先要有字符的点阵数据，ASCII 常用的字符集总共有 95 个，从空格符开始，分为：!"#\$%&' ()*+, -0123456789:;<=>?@ABCDEFHJKLMNOPQRSTUVWXYZ[\]`_`abc defghijklmnopqrstuvwxyz{|}~.

我们先要得到这个字符集的点阵数据，这里我们介绍一个款很好的字符提取软件：PCtoLCD2002 完美版，在我们光盘的“\3--开发工具\4. 常用辅助开发软件\PCtoLCD2002 完美版”位置。该软件可以提供各种字符，包括汉字（字体和大小都可以自己设置）阵提取，且取模方式可以设置好几种，常用的取模方式，该软件都支持。该软件还支持图形模式，也就是用户可以自己定义图片的大小，然后画图，根据所画的图形再生成点阵数据，这功能在制作图标或图片的时候很有用。双击这个软件，弹出界面如下：



这里我们介绍下如何取汉字和字符字模数据，点击“模式”，选择“字符模式”，操作如下：



然后点击“选项”，会弹出一个字模选项对话框，将其设置为“阴码+逐列式+双向+C51 格式”，操作如下：

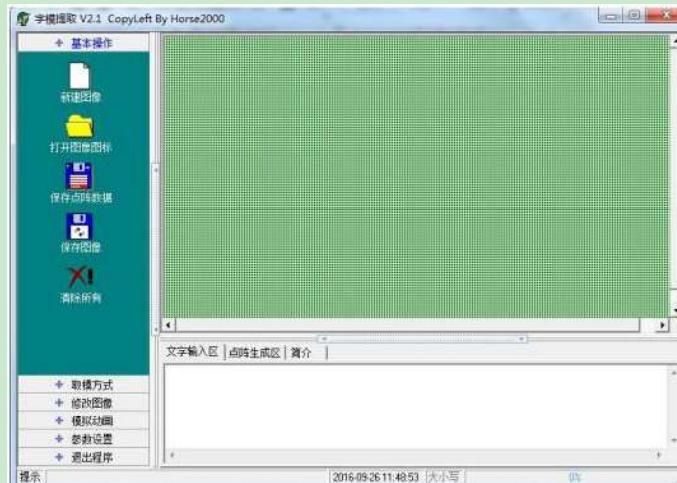


上图设置的取模方式，在右上角的取模说明里面有，即：从第一列开始向下每取 8 个点作为一个字节，如果最后不足 8 个点就补满 8 位。取模顺序是从高到低，即第一个点作为最高位。如*-----取为 10000000。其实就是按如下图的这种方式：

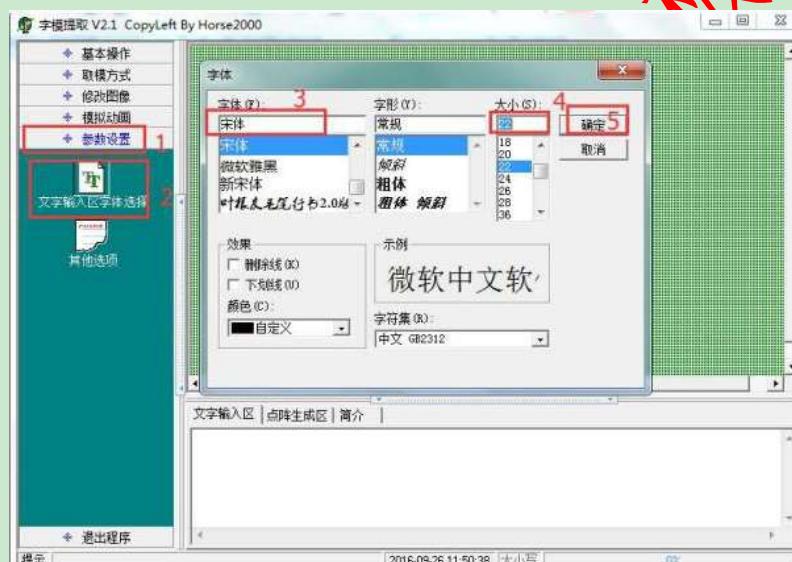


从上到下，从左到右，高位在前。我们按这样的取模方式，然后把 ASCII 字符集按 12*6 大小、16*8 和 24*12 大小取模出来（对应汉字大小为 12*12、16*16 和 24*24，字符只有汉字的一半大！），保存在 font.h 里面，每个 12*6 的字符占用 12 个字节，每个 16*8 的字符占用 16 个字节，每个 24*12 的字符占用 36 个字节。具体见 font.h 部分代码。

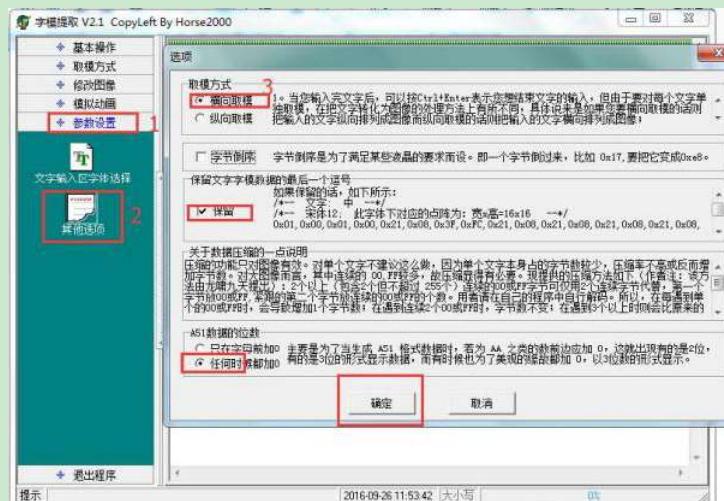
本实验我们还给大家提供了一个汉字显示函数，这里再给大家介绍另一款取模软件，放在光盘“\3-开发工具\4. 常用辅助开发软件\文字取模软件”目录内，此软件用于汉字取模极其方便，下面打开这个软件，界面如下：



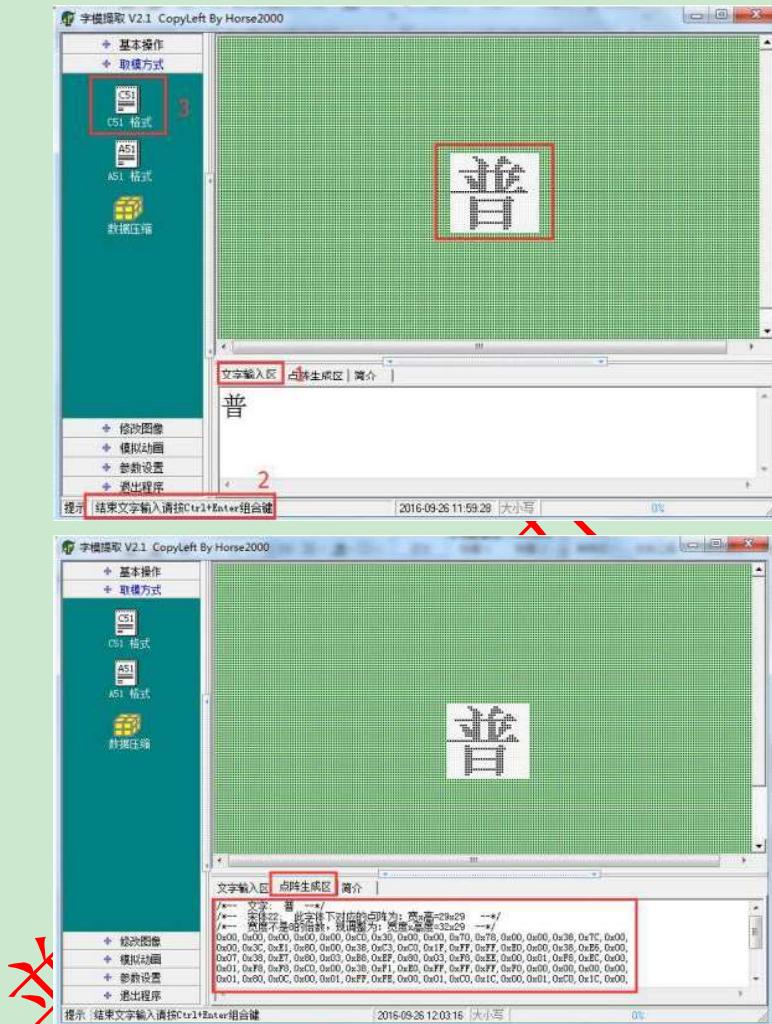
首先我们选择“参数设置”设定好字体类型及大小，本实验所取汉字大小为宋体 22 号，具体操作步骤如下：



设置好字体大小后，我们再来设置下取模的方式，点击“其他选项”，选择“横向取模”，操作步骤如下：



设置好取模方式后，接下来就是取汉字字模数据了，我们直接在“文字输入区”输入所要显示的汉字，比如输入“普”，然后按下“Ctrl+Enter 组合键”就会在窗口显示输入的汉字，点击“取模方式”选择“C51 格式”，就会在“点阵生成区”显示汉字“普”的字模数据，操作步骤如下：



这样我们直接将其保存在 font.h 的 CnChar32x29 数组内，我们在定义显示汉字时还加了汉字内锁码，所以在保存这些汉字字模数据时，前面加上我们所输入的汉字“普”，如下：

```

321 "普",0x00,0x00,0x00,0x00,0x00,0xE0,0x38,0x00,0x00,0x78,0x3C,0x00,0x00,0x3C,0x78,0x00,
322 0x00,0x3C,0x70,0xC0,0x00,0x1C,0xE0,0xF0,0xFF,0xF0,0x00,0x1C,0xE3,0x00,
323 0xE,0x1C,0xE7,0xC0,0x07,0x9C,0xE7,0x80,0x03,0xDC,0xEF,0x00,0x03,0xDC,0xEE,0x00,
324 0x01,0xDC,0xFC,0x60,0x00,0x9C,0xF9,0xF0,0x7E,0xFF,0xFF,0xF8,0x00,0x00,0x00,0x00,
325 0x01,0xC0,0xOE,0x00,0x01,0xFF,0xFF,0x00,0x01,0xC0,0xOE,0x00,0x01,0xC0,0xOE,0x00,
326 0x01,0xC0,0xOE,0x00,0x01,0xFF,0x00,0x01,0xC0,0xOE,0x00,0x01,0xC0,0xOE,0x00,
327 0x01,0xC0,0xOE,0x00,0x01,0xFF,0xFE,0x00,0x01,0xC0,0xOE,0x00,0x01,0xC0,0xOE,0x00,
328 0x00,0x00,0x00,
329

```

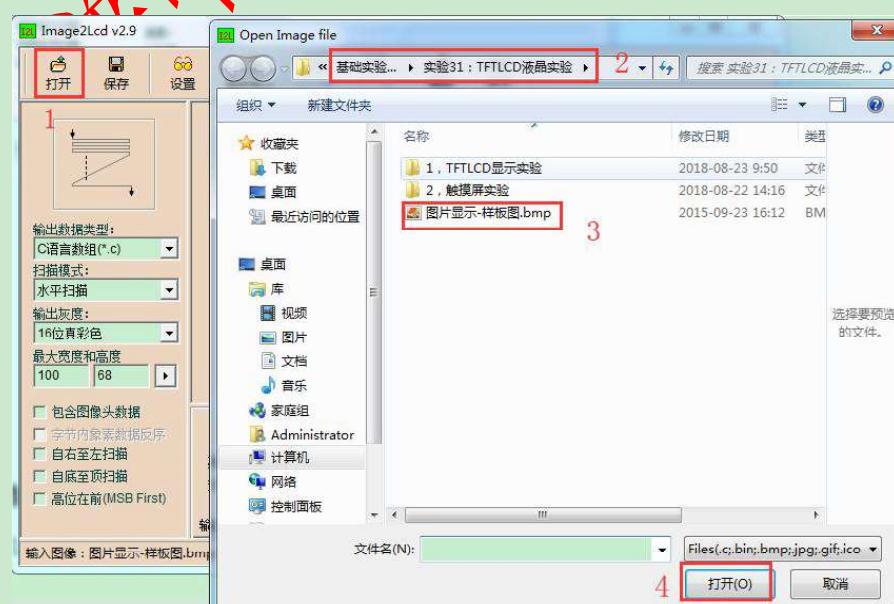
如果要显示其他的汉字，取模方法和上述一样，这些显示函数的具体实现过程，大家可以参考工程源代码，这里就不多介绍。

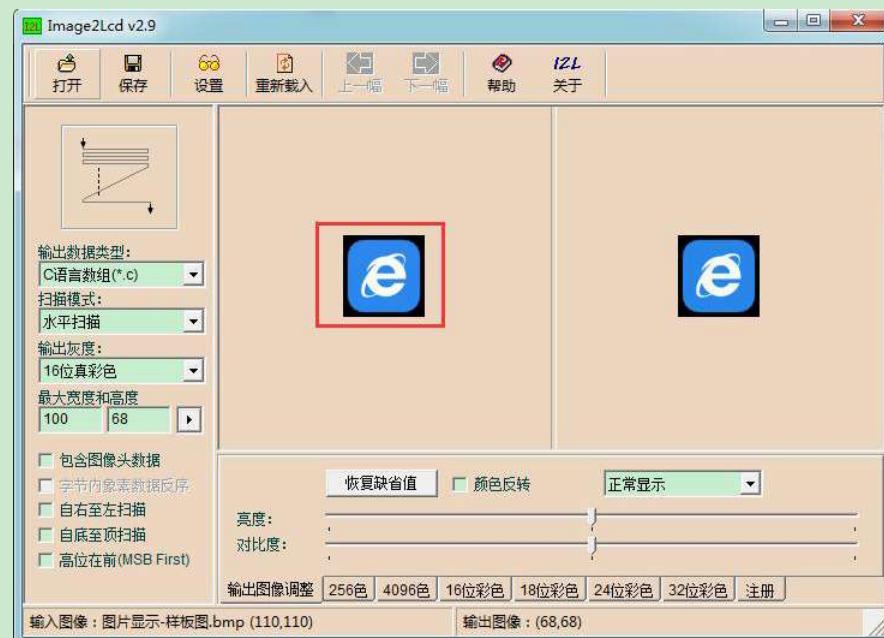
注意：有的朋友在使用取模软件取汉字时会出现这样一种情况，明明是按照教程来设置取模软件参数，但是最后取的汉字数据不是 16*16 大小的，这种情况是因为你电脑系统内缺少某种字库，你可以换台电脑测试，如果可以就把系统内字库拷贝到你电脑中去，怎么查找系统字库，这个就需要你们自己百度了。

在我们的 tftlcd.c 文件内还提供了一个图片显示函数，那么怎么来显示图片呢？这里再给大家介绍另一款取模软件，放在光盘“\3--开发工具\4. 常用辅助开发软件\图片取模软件” 目录内，此软件用于图片取模极其方便，打开后界面如下：

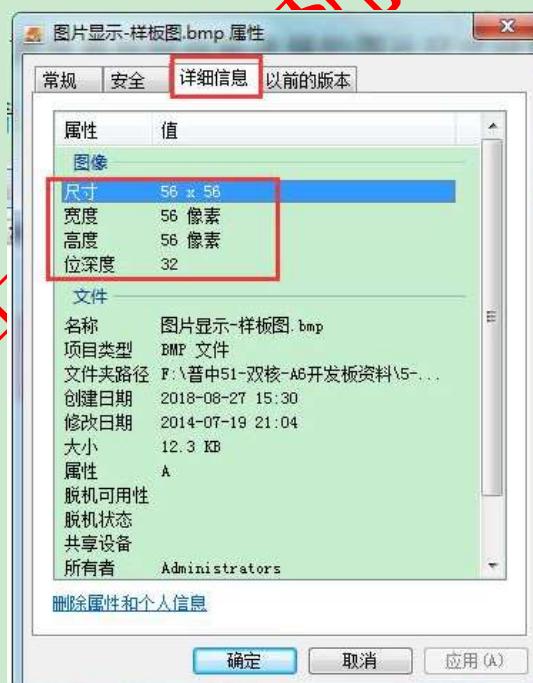


1，点击“打开”菜单按钮，选择你想要显示的图片，注意图片格式要是. BMP，如下所示：

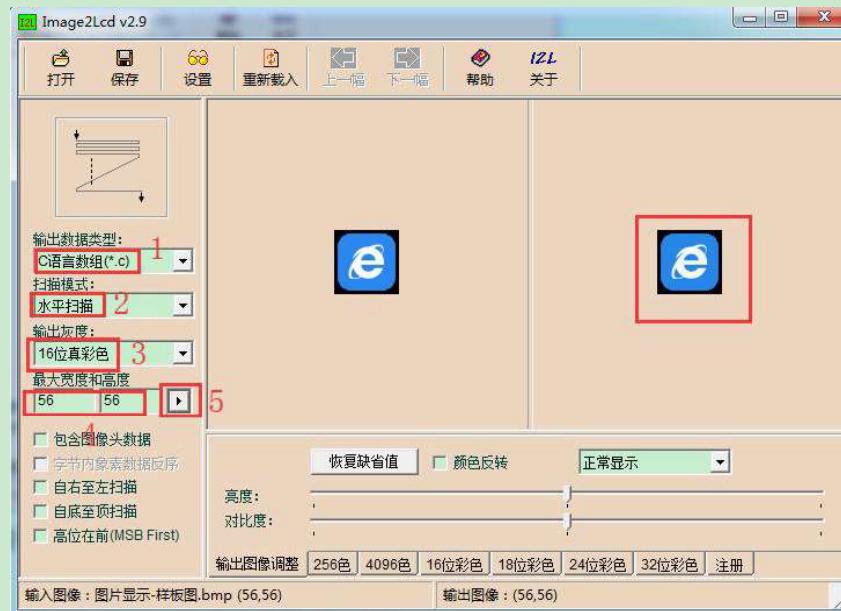




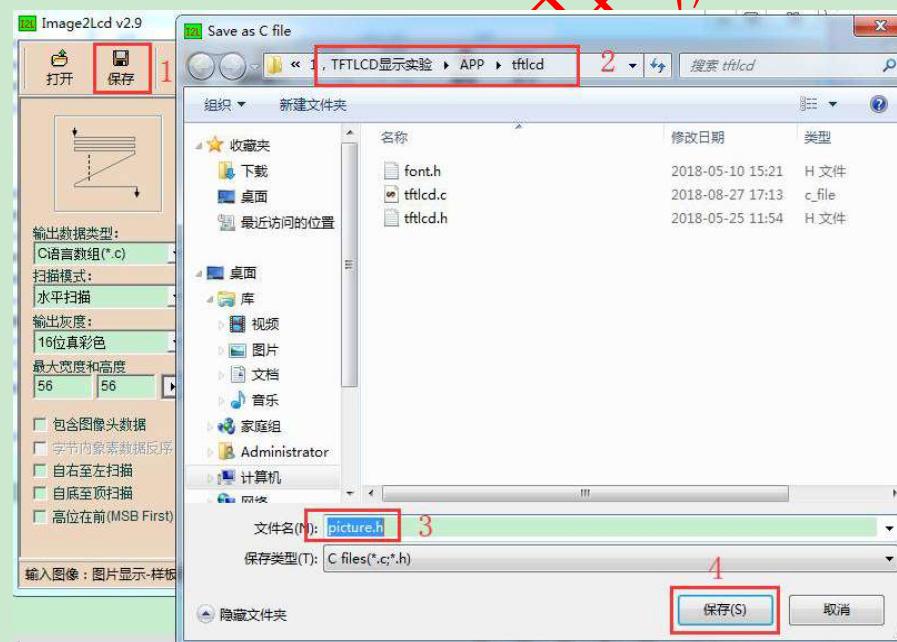
2，设置相关参数，包括输出数据类型、扫描模式、输出灰度、最大宽度和高度等，注意：最大宽度和高度要按照你所选择的图片尺寸设置。图片的实际尺寸大小可点击图片属性查看，如下：



具体设置如下：其他未标注的保持默认即可。



4，点击“保存”菜单按钮，将输出的图片数据保存为 xxx.h 格式，存储到 tftlcd 文件夹内，这里我们就命名为 picture.h，如下：



保存好后就会弹出图片取模后的数据，如下：

上述 gImage_picture 数组即为该图片的数据。由于 51 单片机内 SRAM 存储空间有限，我们要将 const 改为 code，即将这些数据存储到单片机内 FLASH 中。改好后保存，如下：

然后在 main.c 文件内将 picture.h 头文件包含进来并且主函数内调用显示图片函数即可，如下：

```
LCD_ShowPicture(10, 60, 56, 56, gImage_picture);
```

函数中前面两个参数 10, 60 为图片显示的起始位置，后面两个参数 56, 56 分

别是图片的宽度和高度，这两个值一定要保证和前面取模时图片大小一致，否则显示出错。最后一个参数 gImage_picture 即为存储图片数据的数组。

(5) 主函数

编写好 TFTLCD 初始化和显示函数后，接下来就可以编写主函数了，代码如下：

```
void main()
{
    UART_Init();
    TFTLCD_Init();

    FRONT_COLOR=WHITE;
    LCD_ShowFontHZ(tftlcd_data.width/2-2*24-12, 10, "普中科技");
    LCD_ShowString(tftlcd_data.width/2-7*12, 40, tftlcd_data.width, tftlcd_data.height, 24, "www. prechin. cn");
    //LCD_ShowPicture(10, 60, 56, 56, gImage_picture);

    while(1)
    {
    }
}
```

主函数实现的功能很简单，首先调用之前编写好的硬件初始化函数，包括串口初始化等，然后调用我们前面编写的 TFTLCD_Init 函数，用来初始化 TFTLCD，~~默认设置为竖屏显示，背景颜色为黑色~~。设置显示颜色为白色后，调用字符串及汉字显示函数进行显示，最后进入 while 循环。

33.4 实验现象

使用 USB 线将开发板和电脑连接成功后（使用短接片将 USB 转串口模块上 J39 端子的 P31T 与 URXD 短接，J44 端子的 P30R 与 UTXD 短接，电脑能识别开发板上 CH340 驱动串口），把编写好的程序编译后将编译产生的.hex 文件烧入到芯片内，将 TFTLCD 模块插上开发板上的彩屏接口，可以看到 TFTLCD 上即可显示汉字和字符信息。



课后作业

(1) 在 TFTLCD 上显示时钟。

(温馨提示：将本章实验与 D81302 时钟程序组合)

第 34 章 触摸屏实验

在前面章节我们介绍了使用 TFTLCD 模块显示字符和汉字，利用 TFTLCD 模块，51 单片机系统就有了高级信息输出的功能，如果我们还希望有一个友好的用户输入的设备，触摸屏就是非常好的选择，现如今大多电子产品是将触摸屏配合液晶显示器组成人机交互系统，比如手机、平板等。因此我们很有必要学习下触摸屏的控制。这一章我们来学习下如何使用 51 单片机来驱动触摸屏。我们开发板本身并没有触摸屏控制器，但是它支持触摸屏，可以通过外接带触摸屏的 LCD 模块（比如我们的 TFTLCD 模块）来实现触摸屏控制。本章要实现的功能是：通过 TFTLCD 模块上的触摸板（包括电阻触摸和电容触摸）实现触摸功能，最终实现一个画板的功能。学习本章可以参考彩屏相关芯片数据手册（电阻触摸屏和电容触摸屏）。本章分为如下几部分内容：

- 34.1 触摸屏介绍
- 34.2 硬件设计
- 34.3 软件设计
- 34.4 实验现象

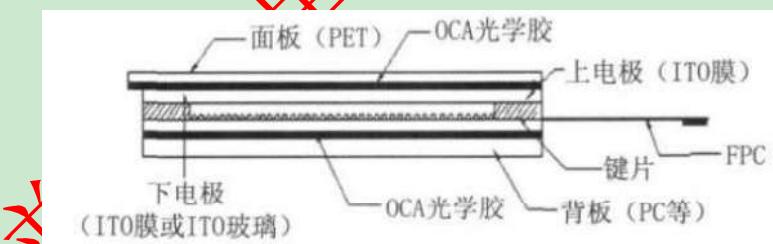
34.1 触摸屏介绍

触摸屏又称触控面板，它是一种把触摸位置转化成坐标数据的输入设备，根据触摸屏的检测原理，主要分为电阻式触摸屏和电容式触摸屏。下面我们就分别来介绍下这两种触摸屏。

34.1.1 电阻式触摸屏介绍

电阻式触摸屏是一种传感器，它将矩形区域中触摸点(X, Y)的物理位置转换为代表 X 坐标和 Y 坐标的电压。很多 LCD 模块都采用了电阻式触摸屏，比如我们 2.0/2.2/2.4/2.6/2.8/3.0/3.2/3.5 寸的 TFTLCD 模块都是采用电阻式触摸屏。使用时需要用一定的压力才会能检测到电压，即触摸。

电阻式触摸屏基本上是薄膜加上玻璃的结构，薄膜和玻璃相邻的一面上均涂有 ITO(纳米铟锡金属氧化物)涂层，ITO 具有很好的导电性和透明性。当触摸操作时，薄膜下层的 ITO 会接触到玻璃上层的 ITO，经由感应器传出相应的电信号，经过转换电路送到处理器，通过运算转化为屏幕上的 X、Y 值，而完成点选的动作，并呈现在屏幕上。电阻式触摸屏结构如下图所示：



电阻触摸屏的工作原理主要是通过压力感应原理来实现对屏幕内容的操作和控制的，这种触摸屏屏体部分是一块与显示器表面非常配合的多层复合薄膜，其中第一层为玻璃或有机玻璃底层，第二层为隔层，第三层为多元树脂表层，表面还涂有一层透明的导电层，上面再盖有一层外表面经硬化处理、光滑防刮的塑料层。在多元脂表层表面的传导层及玻璃层感应器是被许多微小的隔层所分隔电流通过表层，轻触表层压下时，接触到底层，控制器同时从四个角读出相称的电流及计算手指位置的距离。这种触摸屏利用两层高透明的导电层组成触摸屏，两层之间距离仅为 2.5 微米。当手指触摸屏幕时，平常相互绝缘的两层导电层就在触摸点位置有了一个接触，因其中一面导电层接通 Y 轴方向的电源均匀

电压场，使得侦测层的电压由零变为非零，控制器侦测到这个接通后，进行 A/D 转换，并将得到的电压值与参考电压相比，即可得触摸点的 Y 轴坐标，同理得出 X 轴的坐标，这就是所有电阻技术触摸屏共同的最基本原理。

电阻触摸屏的优点：精度高、价格便宜、抗干扰能力强、稳定性好。

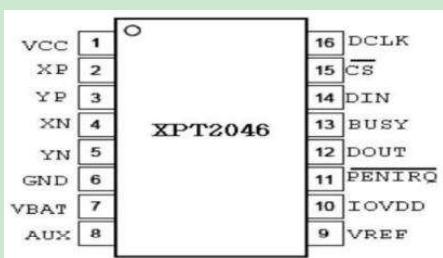
电阻触摸屏的缺点：容易被划伤、透光性不太好、不支持多点触摸。

从上面的简介，我们知道触摸屏都需要一个 AD 转换器，也就是要将电压变化读取出来，供主机求出触摸的位置。我们的 TFTLCD 模块使用的是四线电阻式触摸屏，这种触摸屏的控制芯片有很多，包括：ADS7843、ADS7846、TSC2046、XPT2046 和 AK4182 等。这几款芯片的驱动基本上是一样的，也就是说你只要写出了 XPT2046 的驱动，这个驱动对其他几个芯片也是有效的。而且封装也有一样的，而且管脚也完全兼容。所以在替换起来非常方便。

我们彩屏上面使用的触摸屏控制芯片是 XPT2046。该芯片在前面 ADC 章节介绍过，这里我们再回顾一下，XPT2046 的特点主要有：

- 1) 一款 4 导线制触摸屏控制器，采用 SPI 模式进行通信。
- 2) 内含 12 位分辨率 125KHz 转换速率逐步逼近型 A/D 转换器。
- 3) 支持从 1.5V 到 5.25V 的低电压 I/O 接口。
- 4) 只需执行两次 A/D 转换即可查出被按的屏幕位置。
- 5) 可以测量加在触摸屏上的压力
- 6) 芯片内部自带温度检测、电池电压（0-6V）监测等等

XPT2046 SOP 封装有 16 个引脚，如下图所示：



其引脚说明如下图所示：

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

芯片详细资料大家可以参考数据手册，在前面 ADC 模数转换实验章节也介绍过，可以返回看下。

34.1.2 电容式触摸屏介绍

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。

我们有的 3.5/4.3/4.5/7 寸 TFTLCD 模块上使用的触摸屏是电容式触摸屏，不过这些高端屏都是配置在 STM32 开发板上，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。

投射式电容触摸屏却具有多指触控的功能。这两种电容式触摸屏都具有透光率高、反应速度快、寿命长等优点，缺点是：随着温度、湿度的变化，电容值会发生变化，导致工作稳定性差，时常会有漂移现象，需要经常校对屏幕，且不可佩戴普通手套进行触摸定位。

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。

我们使用的电容屏也是采用投射式电容屏（交互电容类型），所以后面仅以投射式电容屏作为介绍。

投射式电容触摸屏（交互电容类型）内部由驱动电极与接收电极组成，驱动电极发出低电压高频信号投射到接收电极形成稳定的电流，当人体接触到电容屏时，由于人体接地，手指与电容屏就形成一个等效电容，而高频信号可以通过这一等效电容流入地线，这样，接收端所接收的电荷量减小，而当手指越靠近发射端时，电荷减小越明显，最后根据接收端所接收的电流强度来确定所触碰的点。以上就是电容触摸屏的基本原理，更多详细的资料大家可以百度了解下。

电容触摸屏通常也需要一个驱动 IC 来检测电容触摸，且一般是通过 I2C 接口输出触摸数据的。我们 4.5 寸电容触摸屏使用的驱动 IC 是 FT5336，不清楚自己彩屏驱动芯片及驱动触摸 IC 型号的可以看下彩屏表面及背面的型号。FT5336 支持最多 5 点触摸，这里我们以 FT5536 为例进行介绍，其他的驱动 IC 参考着学习。

FT5336 是台湾敦泰电子股份有限公司生产的一颗电容触摸屏驱动 IC，最多

支持 13*24 (314) 个通道。支持 SPI/IIC 接口，我们使用的就是 IIC 接口。

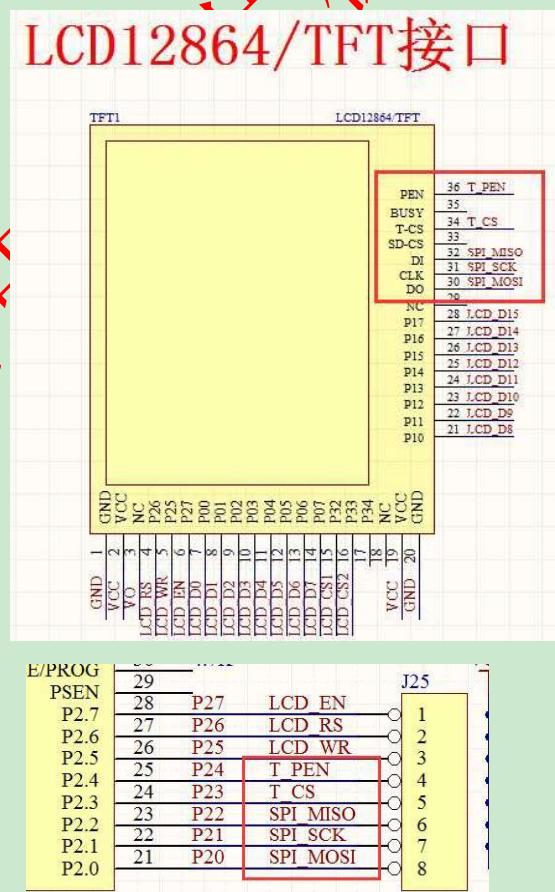
IIC 接口模式下，该驱动 IC 与 51 单片机的连接仅需要 4 根线：SDA、SCL、RST 和 INT，SDA 和 SCL 是 IIC 通信用的，RST 是复位脚（低电平有效），INT 是中断输出信号，关于 IIC 我们就不详细介绍了，在前面 I2C 实验时已经讲过。关于 FT5336 或者其他电容触摸驱动 IC 详细内容，大家可以在彩屏数据手册内查看，不过对于 51 开发板资料中我们未提供，仅仅顺带介绍下，让大家对电容屏有一个认识。这里不多说，我们主要还是看代码，从代码中体会它们的使用。

34.2 硬件设计

本实验使用到硬件资源如下：

(1) TFTLCD 液晶模块（带电阻触摸屏）

TFTLCD 液晶接口在前面章节已经介绍过，下面我们来看下 51 单片机与 TFTLCD 触摸屏的连接关系，如下图所示：



从电路图中可以看到，SPI_MOSI、SPI_MISO、SPI_SCK、T_CS 和 T_PEN 分别连接在 51 单片机的 P20、P22、P21、P23 和 P24 上。而 SPI_MOSI、SPI_MISO、SPI_SCK、T_CS 和 T_PEN 引脚之前我们介绍 TFTLCD 显示实验的时候就给大家介绍，如果是电阻触摸屏的话，它是直接连接在 XPT2046 触摸芯片 SPI 接口及笔中断引脚上的。如果是电容触摸屏的话，只需要 4 根线即可分别是 T_PEN(INT)、T_CS(RST)、SPI_CLK(SCL) 和 SPI_MOSI(SDA)。其中：INT、_RST、SCL 和 SDA 分别是 FT5336 的中断输出信号、复位信号，IIC 的 SCL 和 SDA 信号。这里我们用查询的方式读取 FT5336 的数据，没有用到中断信号（INT），所以同 51 单片机的连接，只需要 3 根线即可，不过我们还是预留 INT 那根，方便扩展的其他驱动 IC 做 IIC 地址设定，所以保持 4 根线连接。因为 51 单片机开发板我们未配置电容屏，所以上述关于电容屏的硬件介绍大家可以不用管它。

34.3 软件设计

本章所要实现的功能是：通过 TFTLCD 模块上的触摸板（电阻触摸）实现触摸功能，最终实现一个画板的功能。

电阻触摸屏采用的是 SPI 通信（使用 IO 口模拟 SPI 时序），电容触摸屏采用的是 IIC 通信，这些通信方式在前面都介绍。下面我们打开 “\5--实验程序\基础实验例程\实验 31：TFTLCD 液晶实验\2，触摸屏实验” 工程，在 APP 工程组中可以看到添加了 touch.c 文件（里面包含了电阻屏驱动程序），同时还要包含对应的头文件路径。

(1) 首先我们看下 touch.c 文件代码，如下：

```
#include "touch.h"  
#include "tftlcd.h"  
#include "uart.h"  
  
XPT_XY xpt_xy;      // 定义一个全局变量保存 X、Y 的值  
  
void TOUCH_SPI_Start(void)
```

```
{  
    TOUCH_CLK = 0;  
    TOUCH_CS = 1;  
    TOUCH_DIN = 1;  
    TOUCH_CLK = 1;  
    TOUCH_CS = 0;  
}  
  
void TOUCH_SPI_Write(u8 dat)  
{  
    u8 i;  
    TOUCH_CLK = 0;  
    for(i=0; i<8; i++)  
    {  
        TOUCH_DIN = dat >>7;      //放置最高位  
        dat <<= 1;  
        TOUCH_CLK = 0;            //上升沿放置数据  
  
        TOUCH_CLK = 1;  
  
    }  
}  
  
u16 TOUCH_SPI_Read(void)  
{  
    u16 i, dat=0;  
    TOUCH_CLK = 0;  
    for(i=0; i<12; i++)      //接收 12 位数据  
    {  
        dat <<= 1;  
  
        TOUCH_CLK = 1;  
        TOUCH_CLK = 0;  
  
        dat |= TOUCH_DOUT;  
  
    }  
    return dat;  
}  
  
#define XY_READ_TIMS 10          //读取的次数  
  
u16 TOUCH_XPT_ReadData(u8 cmd)  
{
```

```
u8 i, j;
u16 readValue[XY_READ_TIMS];
long endValue;

TOUCH_CLK = 0;      //先拉低时间
TOUCH_CS  = 0;      //选中芯片

for(i=0; i<XY_READ_TIMS; i++)          //读取 XY_READ_TIMS 次结果
{
    TOUCH_SPI_Write(cmd);   //发送转换命令
    //delay_10us();
    for(j=6; j>0; j--);   //延时等待转换结果
    TOUCH_CLK = 1;         //发送一个时钟周期, 清除 BUSY
    _nop_();
    _nop_();
    TOUCH_CLK = 0;
    _nop_();
    _nop_();

    readValue[i] = TOUCH_SPI_Read();
}
TOUCH_CS = 1; //释放片选

//---软件滤波---//
//---先大到小排序, 除去最高值, 除去最低值, 求其平均值---//
for(i=0; i<XY_READ_TIMS - 1; i++) //从大到小排序
{
    for(j= i+1; j<XY_READ_TIMS; j++)
    {
        if(readValue[i] < readValue[j])
        {
            endValue = readValue[i];
            readValue[i] = readValue[j];
            readValue[j] = endValue;
        }
    }
}
// if((readValue[2] - readValue[3]) > 5)
// {
//     return 0;
// }
endValue = 0;
for(i=2; i<XY_READ_TIMS-2; i++)
{
```

```
    endValue += readValue[i];
}
endValue = endValue/ (XY_READ_TIMS - 4); //求平均值

return endValue;
}

u8 TOUCH_XPT_ReadXY(void)
{
    u16 x1, x2, x, y1, y2, y;

    TOUCH_SPI_Start();
    //---分别读两次 X 值和 Y 值，交叉着读可以提高一些读取精度---//
    x1 = TOUCH_XPT_ReadData(XPT_CMD_X);
    y1 = TOUCH_XPT_ReadData(XPT_CMD_Y);
    x2 = TOUCH_XPT_ReadData(XPT_CMD_X);
    y2 = TOUCH_XPT_ReadData(XPT_CMD_Y);

    //---求取 X 值的差值---//
    if (x1 > x2)
    {
        x = x1 - x2;
    }
    else
    {
        x = x2 - x1;
    }

    //---求取 Y 值的差值---//
    if (y1 > y2)
    {
        y = y1 - y2;
    }
    else
    {
        y = y2 - y1;
    }

    //---判断差值是否大于 50，大于就返回 0，表示读取失败---//
    if((x > 50) || (y > 50))
    {
```

```
        return 0;
    }

//---求取两次读取值的平均数作为读取到的 XY 值---//
xpt_xy.x = (x1 + x2) / 2;
xpt_xy.y = (y1 + y2) / 2;

xpt_xy.x &= 0xFFFF0; //去掉低四位
xpt_xy.y &= 0xFFFF0;

//---确定 XY 值的范围，用在触摸屏大于 TFT 时---//
if((xpt_xy.x < 100) || (xpt_xy.y > 3800))
{
    return 0;
}

return 1; // 返回 1，表示读取成功
}

//返回 1：触摸按下
//0：无触摸
u8 TOUCH_Scan(void)
{
    u8 res=0;
    u32 temp;

    if(TOUCH_XPT_ReadXY())
    {
        //如果触摸跟显示发生偏移，可以根据显示 AD 值---//
        //---调整下面公式里面的数值---//
        if(tftlcd_data.dir==0)
        {
#endif TFT22_ILI9225B
            xpt_xy.lcdx = xpt_xy.x;
            xpt_xy.lcdx = (xpt_xy.lcdx - 250) * 180 / 3700;
            xpt_xy.lcdy = xpt_xy.y;
            xpt_xy.lcdy = (xpt_xy.lcdy - 350) * 250 / 3500;
#endif
#endif TFT24_ST7781R
            xpt_xy.lcdx=xpt_xy.x;
            xpt_xy.lcdx=((xpt_xy.lcdx - 336)*240)/3328;
            xpt_xy.lcdy=xpt_xy.y;
            xpt_xy.lcdy =((xpt_xy.lcdy - 240) *320)/3412;
    }
}
```

```
#endif

#define TFT26_R61509V
    xpt_xy.lcdx=xpt_xy.x;
    xpt_xy.lcdx=((xpt_xy.lcdx - 240)*260)/3850;
    xpt_xy.lcdy=xpt_xy.y;
    xpt_xy.lcdy =((xpt_xy.lcdy - 200)*420)/3950;
#endif

#define TFT26_ILI9325D
    xpt_xy.lcdx=xpt_xy.x;
    xpt_xy.lcdx=((xpt_xy.lcdx - 600)*297)/3550;
    xpt_xy.lcdy=xpt_xy.y;
    xpt_xy.lcdy =((xpt_xy.lcdy - 250)*362)/3850;
#endif

}

else
{
#endif TFT22_ILI9225B
    xpt_xy.lcdx = 4096-xpt_xy.y;
    xpt_xy.lcdx = (xpt_xy.lcdx - 350) * 250 / 3500;
    xpt_xy.lcdy = xpt_xy.x;
    xpt_xy.lcdy = (xpt_xy.lcdy - 250) * 180 / 3700;
#endif

#endif TFT24_ST7781R
    xpt_xy.lcdx=xpt_xy.y;
    xpt_xy.lcdx =((xpt_xy.lcdx - 240) *320)/3412;
    xpt_xy.lcdy=4096-xpt_xy.x;
    xpt_xy.lcdy =((xpt_xy.lcdy - 336)*240)/3328;
#endif

#endif TFT26_R61509V
    xpt_xy.lcdx=xpt_xy.y;
    xpt_xy.lcdx=((xpt_xy.lcdx - 200)*420)/3950;
    xpt_xy.lcdy=4096 - xpt_xy.x;
    xpt_xy.lcdy =((xpt_xy.lcdy - 240)*260)/3850;
#endif

#endif TFT26_ILI9325D
    xpt_xy.lcdx=xpt_xy.y;
    xpt_xy.lcdx=((xpt_xy.lcdx - 250)*362)/3850;
    xpt_xy.lcdy=4096 - xpt_xy.x;
```

```
xpt_xy.lcdy =((xpt_xy.lcdy - 600)*297)/3550;  
#endif  
  
}  
  
  
//      if(tftlcd_data.dir==0)  
//      {  
//          xpt_xy.lcdx=xpt_xy.lcdx;  
//          xpt_xy.lcdy=xpt_xy.lcdy;  
//      }  
//      else  
//      {  
//          temp=xpt_xy.lcdx;  
//          xpt_xy.lcdx=xpt_xy.lcdy;  
//          xpt_xy.lcdy=tftlcd_data.height-temp;  
//      }  
  
//      printf("ad_x=%d    ad_y=%d\r\n", xpt_xy.x, xpt_xy.y);  
//      printf("lcdx=%d    lcdy=%d\r\n", xpt_xy.lcdx, xpt_xy.lcdy);  
res=1;  
xpt_xy.sta=1;  
}  
else  
    xpt_xy.sta=0;  
return res;  
}
```

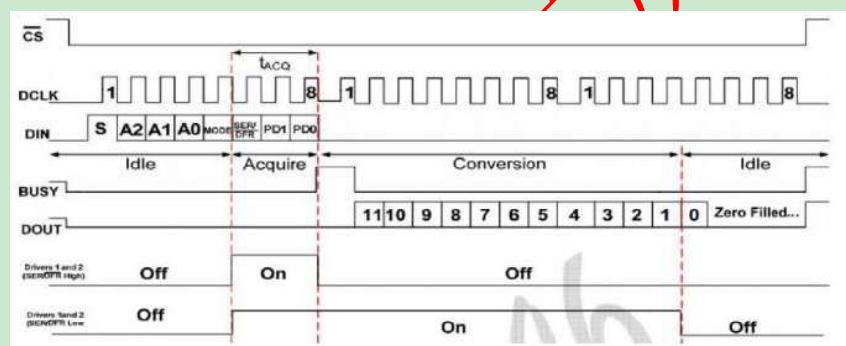
TOUCH_SPI_Write、TOUCH_SPI_Read 函数是 51 单片机 IO 口模拟 SPI 时序的读写函数程序，TOUCH_SPI_Start 函数是 IO 口初始化程序，TOUCH_XPT_ReadXY 是一个比较关键的函数，用来读取触摸屏 X\Y 轴的物理坐标，因为只有获取物理坐标后我们才能转换对应 LCD 的坐标，这部分后面会介绍。

TOUCH_XPT_ReadXY 内首先通过 TOUCH_ReadData 函数获取触摸屏物理坐标值，然后通过相应的程序滤波，保证数据的准确性，防止出现飞点等误差。比较常用的应用程序滤波的方法其实就是多次数据的读取，然后把最大最小值除去，算出平均值。这种方法读取的次数越多，得到的数据就越准确。不过为了更好的滤波，还使用了另外一种方式进行滤波，也就是当读取到两次数据之后，然后检查两个数据之间的差值，如果超过理想的误差，那么丢弃数据。这种方法也是处理飞点现象的常用方法。

该函数里面用到了很多的宏，比如 TOUCH_X_CMD、TOUCH_Y_CMD、TOUCH_MAX 等，这些都在 touch.h 文件内定义了。TOUCH_X_CMD=0xD0 和 TOUCH_Y_CMD=0X90 是 XPT2046 AD 芯片读取 X 和 Y 轴的命令，可通过《 XPT2046》 芯片数据手册查找到，如下图：

A2	A1	A0	V _{BAT}	AUX _{IN}	TEMP	YN	XP	YP	Y-位置	X-位置	Z ₁ -位置	Z ₂ -位置	X-驱动	Y-驱动
0	0	0			+IN (TEMP0)								off	off
0	0	1				+IN			测量				Off	On
0	1	0	+IN				+IN						Off	Off
0	1	1				+IN				测量			XN, On	YP, On
1	0	0			+IN						测量		XN, On	YP, On
1	0	1					+IN		测量				On	Off
1	1	0		+IN	+IN (TEMP1)								Off	Off
1	1	1											Off	Off

这里要特别说明下，要读取 XPT2046 的数据，根据其时序图可知，时序图如下：



XPT2046 完成一个完整的转换需要 24 个串行时钟，也就是需要 3 个字节的 SPI 时钟。对照上图，XPT2046 前 8 个串行时钟，是接收 1 个字节的转换命令，接收到转换命令了之后，然后使用 1 个串行时钟的时间来完成数据转换（当然在编写程序的时候，为了得到精确的数据，你可以适当的延时一下），然后返回 12 个字节长度（12 个字节长度也计时 12 个串行时钟）的转换结果。然后最后 4 个串行时钟返回 4 个无效数据。这一过程在 TOUCH_Read_AD 函数内实现。

下面我们来讲解如何将触摸屏物理坐标转换为 LCD 坐标。

我们使用 XPT2046 读取到了触摸屏的触摸位置之后，想要在 LCD 屏相对应的位置上进行操作，我们还要将它转换成 LCD 屏的坐标值。比如说，我们在 LCD 屏 (0, 0) 坐标位置按下，而读取到的物理坐标值(也就是 AD 值)为 (100, 200)，那么我们想要在 LCD 屏 (0, 0) 位置进行处理，将要将物理坐标 (100, 200) 转

换成 LCD 屏坐标。

那如何转换呢？我们知道，XPT2046 的分辨率为 12 位，也就是说我们读取 X 轴的物理坐标值（这里我们假设为：Px）和 Y 轴的物理坐标值（这里我们假设为：Py）的值肯定是在 0~4096 之间。但是我们 LCD 彩屏 X 轴和 Y 轴的像素坐标确是 240X400。（对应 TFT26_R61509V 彩屏的像素，不管多少，我们明白原理就行，为了更好的表示，在这里我们 LCD 彩屏 X 轴像素坐标我们假设为：Lcdx，LCD 彩屏 Y 轴像素坐标我们假设为：Lcdy）那么我们假设当 (Px, Py) = (0, 0) 时，正好 LCD 彩屏像素坐标的起始坐标 (0, 0)，当 (Px, Py) = (4096, 4096) 时，正好 LCD 彩屏像素坐标的终止坐标 (239, 399)。那么我们不难看出触摸屏的物理坐标跟 LCD 彩屏像素坐标的对应关系为：

$$\text{Factor}_x = \text{Lcdx} / \text{Px};$$

$$\text{Factor}_y = \text{Lcdy} / \text{Py};$$

我们就可以求出 Factor_x 和 Factor_y，然后每次读取到 Px 和 Py 之后就可以讲它很轻松的转换为 Lcdx 和 Lcdy。这是一个很简单的数学关系。

不过呢，事情没有那么理想化，我们在 LCD 像素坐标为 (0, 0) 读取的触摸屏物理坐标值不一定是 (0, 0)，在 LCD 像素坐标为最大时，也不一定读取到的是触摸屏的物理坐标最大值。所以我们要进行一些数据校正，这也是为什么电阻屏幕校正的原因。

什么意思呢？那我们在来解一个数学问题：

我们都知道每个触摸屏物理坐标值都能一一对应一个 LCD 彩屏上面的像素坐标值，也就是它们是成比例关系的。现在我们知道 LCD 彩屏的 X 轴像素坐标最小值为 Lcdx₁，我们能显示的 LCD 彩屏的 X 轴像素坐标最大值为 Lcdx₂。而我们在 LCD 彩屏像素坐标 X 轴最小值处读取的触摸屏 X 轴物理坐标为 Px₁，在 LCD 彩屏 X 轴像素坐标最大值处读取的触摸屏 X 轴的物理坐标为 Px₂。那么现在我们知道有一个触摸屏物理坐标值在 Px₁ 到 Px₂ 之间的坐标值为 Px，那么和它对应的 Lcdx 的值是多少呢？

我们可以这么解：

$$\text{Factor}_x = (\text{Lcdx}_2 - \text{Lcdx}_1) / (\text{Px}_2 - \text{Px}_1);$$

$$\text{Lcdx} = (\text{Px} - \text{Px}_1) * \text{Factor}_x;$$

那么就求得出 Lcdx 是多少了。

现在我们把它分解出来：

```
Lcdx = Px * Factorx - Px1 * Factorx;
```

然后将 Px1*Factorx 替换成一个变量 Offsetx。那么我们现在就可以得到 Lcdx 和 Px 之间的对应关系式。而关于 Y 轴也是同理，所以它们从物理坐标到像素坐标的转换关系式：

```
Lcdx = Px * Factorx - Offsetx;
```

```
Lcdy = Py * Factory - Offsety;
```

在程序中对应的如下所示：

```
xpt_xy.lcdx=xpt_xy.x;  
xpt_xy.lcdx=((xpt_xy.lcdx - 240)*260)/3850;  
xpt_xy.lcdy=xpt_xy.y;  
xpt_xy.lcdy =((xpt_xy.lcdy - 200)*420)/3950;
```

~~xpt_xy.x 和 xpt_xy.y 是读取 XPT2046 的 X 和 Y 方向的 AD 值，而 xpt_xy.lcdx 和 xpt_xy.lcdy 是对应 TFTLCD 触摸屏实际的 XY 轴坐标值。如果大家不理解没有关系，这些都是固定公式，直接按照我们例程套用即可。如果大家发现手上的触摸偏差很大，可以通过修改 (xpt_xy.lcdx - 240)*260 中的 240 值，和 (xpt_xy.lcdy - 200)*420 中的 200 值。里面的 260 和 420 是 TFTLCD 分辨率，不过我们只用到了 240*400。~~

~~在该文件中还有一个函数是非常重要的，即触摸屏扫描函数。该函数检测是否有触摸，并在该函数中获取触摸屏的物理坐标和 LCD 坐标。~~

(3) 接下来看下 main.c 文件代码，如下：

```
#include "public.h"  
#include "uart.h"  
#include "tftlcd.h"  
#include "touch.h"  
#include "gui.h"
```

```
u8 Touch_RST=0;
```

```
//触摸测试
```

```
void TOUCH_Test(void)
{
    static u16 penColor = BLUE;

    TOUCH_Scan();
    if(xpt_xy.sta)
    {
        if(xpt_xy.lcdx>tftlcd_data.width)
            xpt_xy.lcdx=tftlcd_data.width-1;
        if(xpt_xy.lcdy>tftlcd_data.height)
            xpt_xy.lcdy=tftlcd_data.height-1;

        if((xpt_xy.lcdx>=(tftlcd_data.width-3*12))&&(xpt_xy.lcdy<24))
            Touch_RST=1;
        if(xpt_xy.lcdy > tftlcd_data.height - 20)
        {
            if(xpt_xy.lcdx>100)
            {
                penColor = YELLOW;
            }
            else if(xpt_xy.lcdx>80)
            {
                penColor = CYAN;
            }
            else if(xpt_xy.lcdx>60)
            {
                penColor = GREEN;
            }
            else if(xpt_xy.lcdx>40)
            {
                penColor = MAGENTA;
            }
            else if(xpt_xy.lcdx>20)
            {
                penColor = RED;
            }
        }
        else if(xpt_xy.lcdx>0)
        {
            penColor = BLUE;
        }
    }
}
```

```
    else
    {
        LCD_Fill(xpt_xy.lcdx-2, xpt_xy.lcdy-2, xpt_xy.lcdx+2, xpt_xy.lcdy+2,
penColor);
    }

}

//GUI 测试
void GUI_Test(void)
{
    FRONT_COLOR=WHITE;
    LCD_ShowFontHZ(tftlcd_data.width/2-2*24-12, 0, "普中科技");
    LCD_ShowString(tftlcd_data.width/2-7*12, 30, tftlcd_data.width, t
ftlcd_data.height, 24, "www. prechin. cn");
    LCD_ShowString(tftlcd_data.width/2-7*8, 55, tftlcd_data.width, tf
t lcd_data.height, 24, "GUI Test");

    FRONT_COLOR=RED;
    gui_draw_bigpoint(10, 55, GREEN);
    gui_draw_bline(10, 80, 120, 80, 10, GREEN);
    gui_draw_rectangle(10, 95, 30, 30, GREEN);
    gui_draw_arcrectangle(50, 95, 30, 30, 5, 1, BLUE, GREEN);
    gui_fill_rectangle(90, 95, 30, 30, GREEN);
    //gui_fill_circle(90, 120, 20, GREEN);
    gui_fill_ellipse(30, 150, 20, 10, GREEN);

    //更多的 GUI 等待大家来发掘和编写

    delay_ms(2000);
    LCD_Clear(BACK_COLOR);
}

void main()
{
    u16 color=0;

    UART_Init();
    TFTLCD_Init();

Start:
```

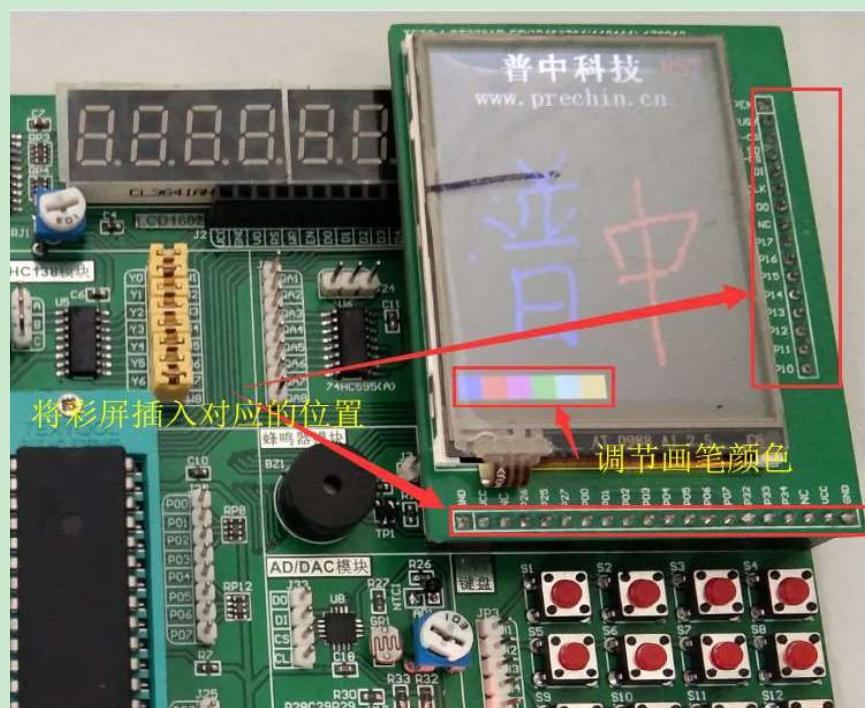
```
//GUI_Test();  
  
FRONT_COLOR=WHITE;  
LCD_ShowFontHZ(tftlcd_data.width/2-2*24-12, 0, "普中科技");  
LCD_ShowString(tftlcd_data.width/2-7*12, 30, tftlcd_data.width, tftlcd_data.height, 24, "www. prechin. cn");  
FRONT_COLOR=RED;  
LCD_ShowString(tftlcd_data.width-3*12, 0, tftlcd_data.width, tftlcd_data.height, 24, "RST");  
LCD_Fill(0, tftlcd_data.height - 20, 20, tftlcd_data.height, BLUE);  
LCD_Fill(20, tftlcd_data.height - 20, 40, tftlcd_data.height, RED);  
LCD_Fill(40, tftlcd_data.height - 20, 60, tftlcd_data.height, MAGENTA);  
LCD_Fill(60, tftlcd_data.height - 20, 80, tftlcd_data.height, GREEN);  
LCD_Fill(80, tftlcd_data.height - 20, 100, tftlcd_data.height, CYAN);  
LCD_Fill(100, tftlcd_data.height - 20, 120, tftlcd_data.height, YELLOW);  
  
while(1)  
{  
    if(Touch_RST)  
    {  
        Touch_RST=0;  
        LCD_Clear(BACK_COLOR);  
        goto Start;  
    }  
  
    TOUCH_Test();  
}  
}
```

主函数实现的功能很简单，首先调用之前编写好的硬件初始化函数，包括串口初始化等。然后调用我们前面编写的 TFTLCD 汉字字符显示等函数完成信息显示，接着进入 while 循环，不断检测 Touch_RST 变量是否置 1，如果为 1 进行 TFTLCD 清屏，然后重新回到 Start 入口处运行程序。如果没有置 1，则执行 TOUCH_Test() 函数实现触摸功能。通过触摸不同色块选择颜色，就可以画出不同

颜色的图画。

34.4 实验现象

使用 USB 线将开发板和电脑连接成功后(使用短接片将 USB 转串口模块上 J39 端子的 P31T 与 URXD 短接, J44 端子的 P30R 与 UTXD 短接, 电脑能识别开发板上 CH340 驱动串口), 把编写好的程序编译后将编译产生的. hex 文件烧入到芯片内, 将 TFTLCD 模块插上开发板上的彩屏接口, 可以看到 TFTLCD 上显示提示信息, 可通过触摸实现绘画功能。



课后作业

- (1) 在触摸屏上设计 2 个触摸按钮, 控制开发板上蜂鸣器的开关。 (温馨提示: 使用 LCD API 函数画 2 个按键, 通过判断这两个按键的位置坐标实现控制)