

# MINI LCD12864 使用教程

一. 资料介绍.....	2
1.1 资料简介.....	2
二. 原理图.....	2
2.1 转接板原理图和底板原理图.....	2
2.2 实际安装.....	4
三. 驱动程序.....	5
3.1 设置 IO 口.....	5
3.2 驱动函数声明.....	5
3.2 函数定义.....	5
四. 显示文字.....	13
4.1 取模软件如何制作简单的字库.....	13
4.2 12864 怎么显示字.....	18
五. 显示图像.....	20
5.1 如何将图像取模.....	20
5.2 12864 如何显示图像.....	21

普中科技开发板

# 一. 资料介绍

## 1.1 资料简介

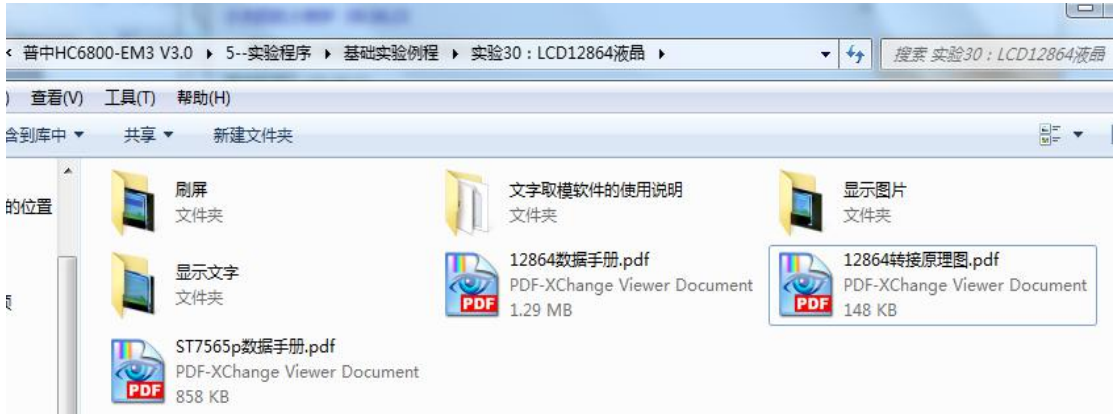


图 1. 1

在光盘资料的实验例程里有 LCD12864 的实验例程和资料，这里的资料是对应的 Mini LCD12864。有相应的 12864 数据手册，是介绍 12864 液晶屏的硬件规格；12864 转接板原理图是液晶屏底板的原理图；ST7565p 数据手册是液晶屏内核手册，里面有驱动方法和指令等等和程序有关的数据。

实验例程，刷屏、显示图片、显示文件，是三个基础实验。

文字取模软件的使用说明，里面包含取模软件和使用方法。可以对文字和图像进行取模。

# 二. 原理图

## 2.1 转接板原理图和底板原理图

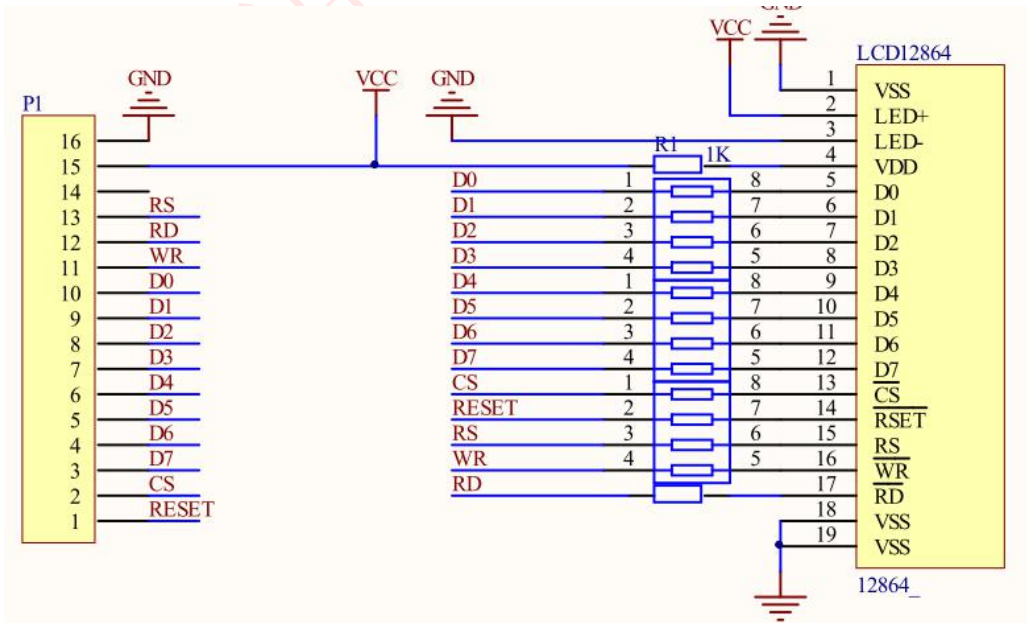


图 2. 1

在原理图上左侧是转接板排针，右侧是 mini LCD12864 液晶屏的排线接口。

排线已经焊接到转接板上，已被液晶屏盖住。

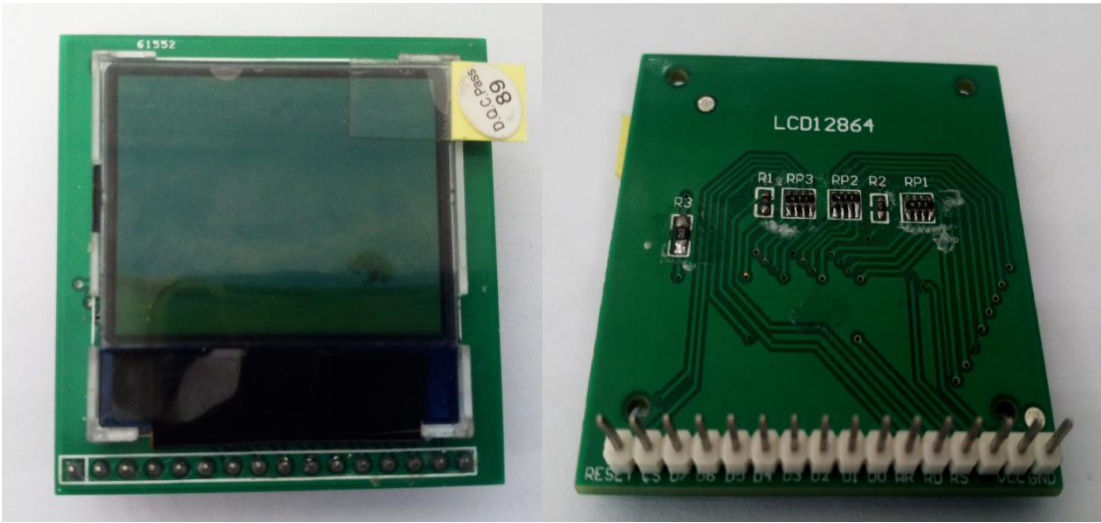


图 2.2

从实物图的背面可以看到，引脚从左至右是以 GND VCC 开始（注意上图是背面拍照，所以看上去 GND 在右侧，实际上正面看，GND 是在左侧的）。对比原理图会发现 GND VCC 是排针的 16 15 两个引脚，因此能够依次找到每个引脚在原理图上的对应，从而知道液晶屏实际排线与排针之间真实的连线。

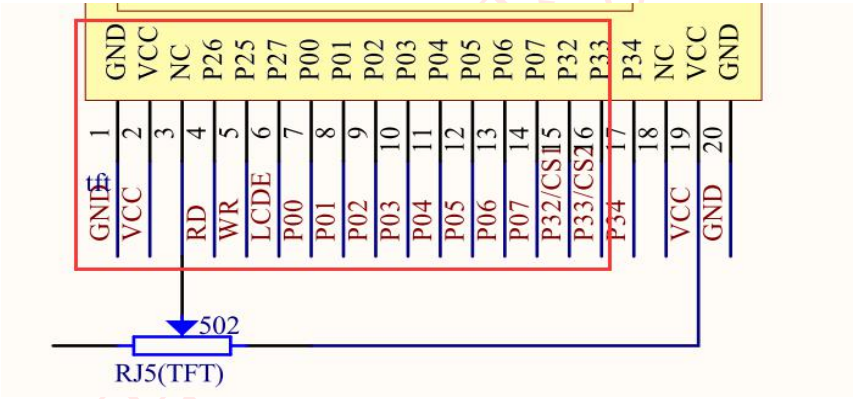


图 2.3

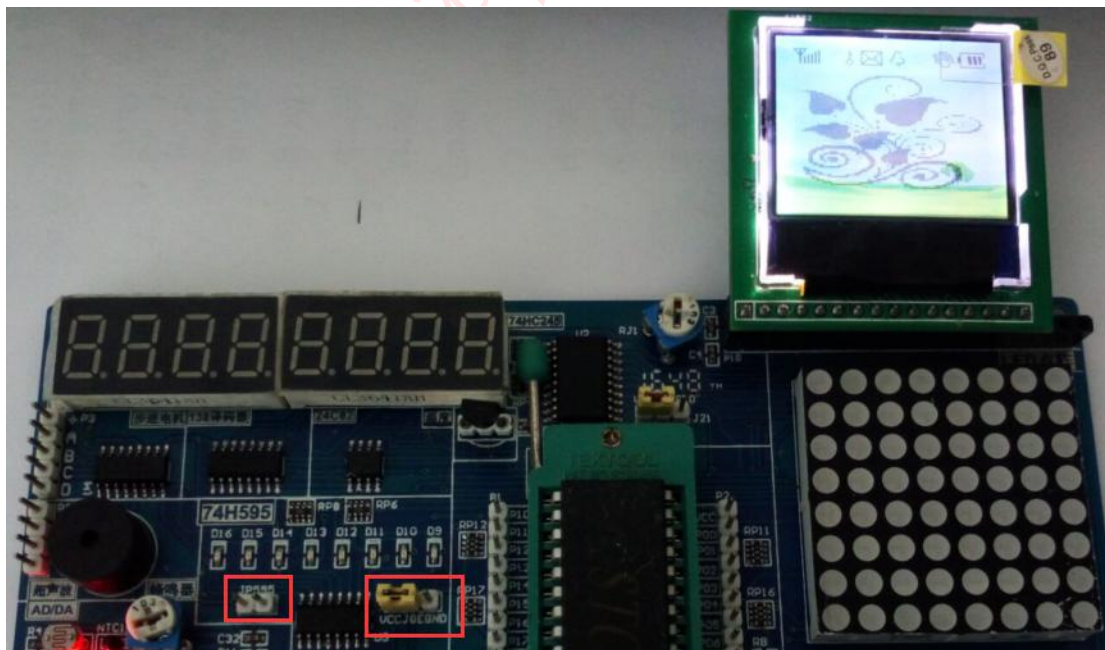
上图是开发板 HC6800 EM3 V3.0 上 12864 的接口，设计的是 20 针脚的排母，适用于带字库的普通 LCD12864 液晶。建议使用其他液晶时，看准原理图上 IO 和电源引脚是否对应，避免造成电源短路。图上框的 16 脚是 Mini LCD12864 的接线座。在 12864 的转接板原理图上可以看到第三脚是悬空的，所以这里的电位器不适合用于 Mini LCD12864 的背光调节。

## 2.2 实际安装



图 2.4

上图是 Mini LCD12864 在开发板 HC6800 EM3 V3.0 上的安装。安装时排针左侧第一个针与排母左侧第一个孔对齐。



在 HC6800-ES V2.0 上插法，注意跳线帽如何插，这里显示的是图像取模以后的显示结果。

## 三. 驱动程序

### 3.1 设置 IO 口

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。

### 3.2 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。
```

### 3.2 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。

## TIMING CHARACTERISTICS

System Bus Read/Write Characteristics 1 (For the 8080 Series MPU)

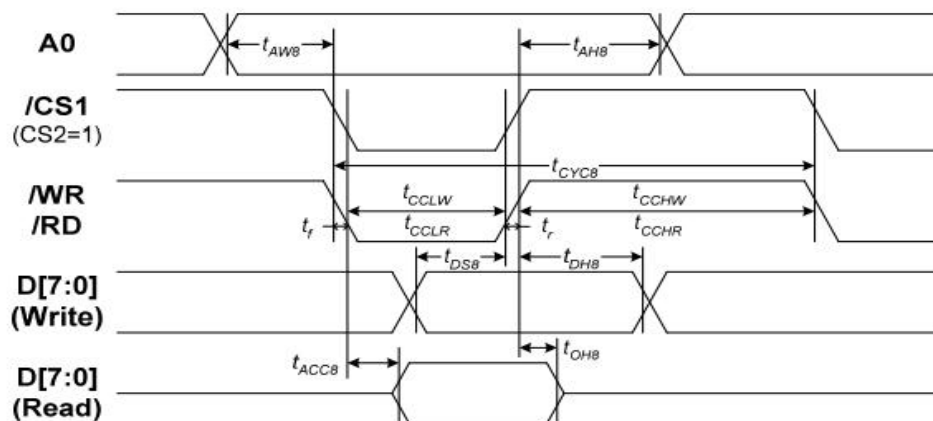


Figure 37

Table 24

(VDD = 3.3V, Ta = -30 to 85°C)

Item	Signal	Symbol	Condition	Rating		Units
				Min.	Max.	
Address hold time	A0	t <sub>AH8</sub>		0	—	
Address setup time		t <sub>AW8</sub>		0	—	
System cycle time		t <sub>CYC8</sub>		240	—	



图 3.1

3、初始化函数；

```

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，详细的解释在  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。

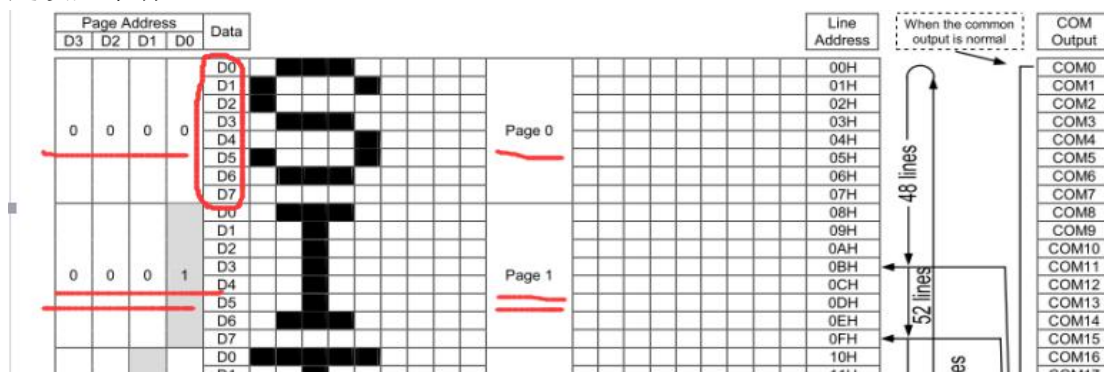


图 3.2

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。

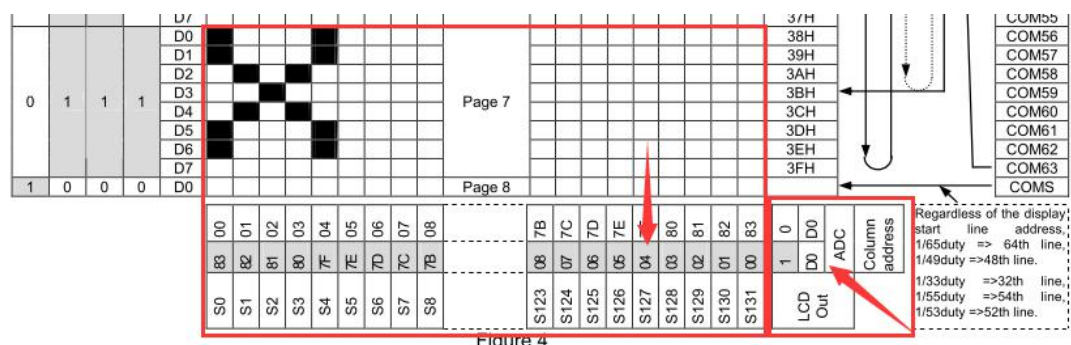


图 3.3

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

图 3.4

在图 3.3 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 3.4 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。

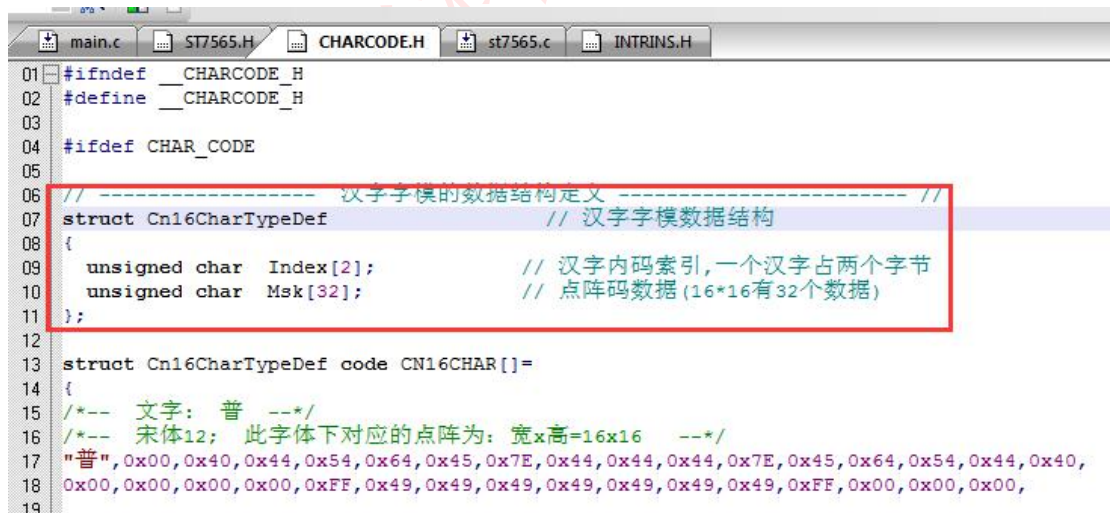


图 3.5

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，一个字节站 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，直到字串结束写字循环。



## 四. 显示文字

### 4.1 取模软件如何制作简单的字库

#### 4.1.1、打开取字模软件



图 4.1

如上图所示，找到实验例程下的 12864 例程，其中有取字模软件，有的光盘资料整理方式不一样，有可能在工具文件夹或其他位置。找到取字模软件双击运行。打开界面如下：

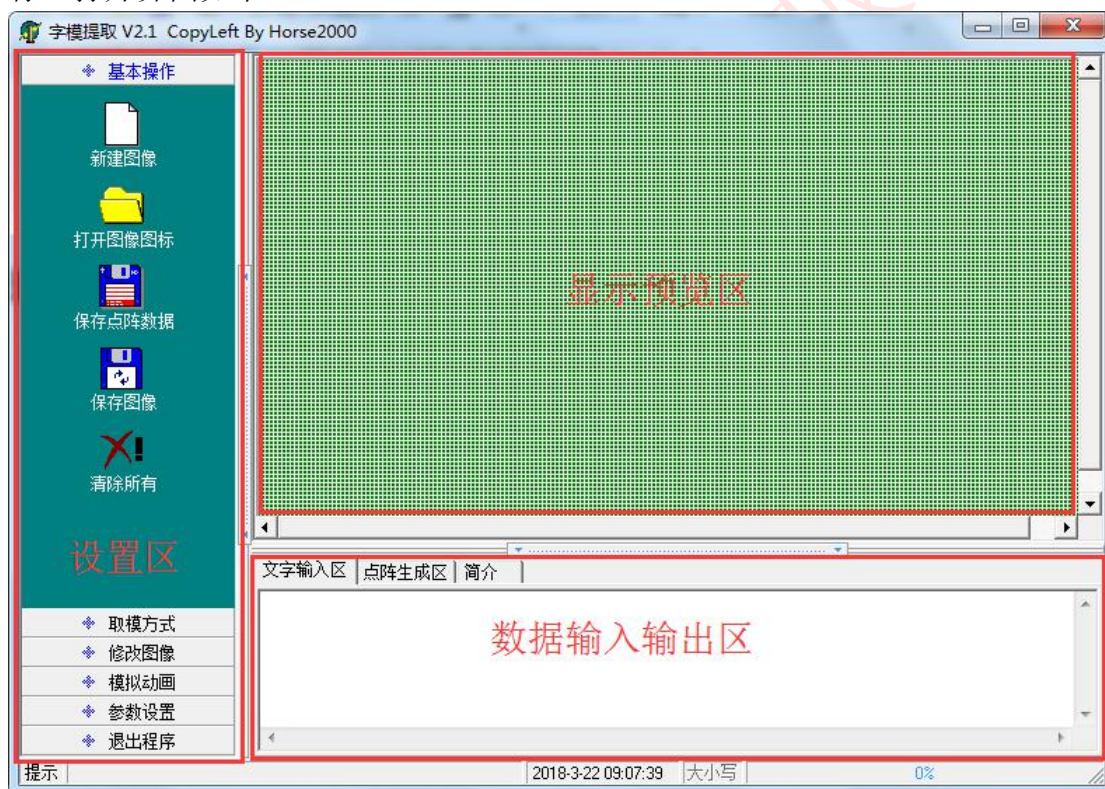


图 4.2

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。

#### 4.1.2、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。

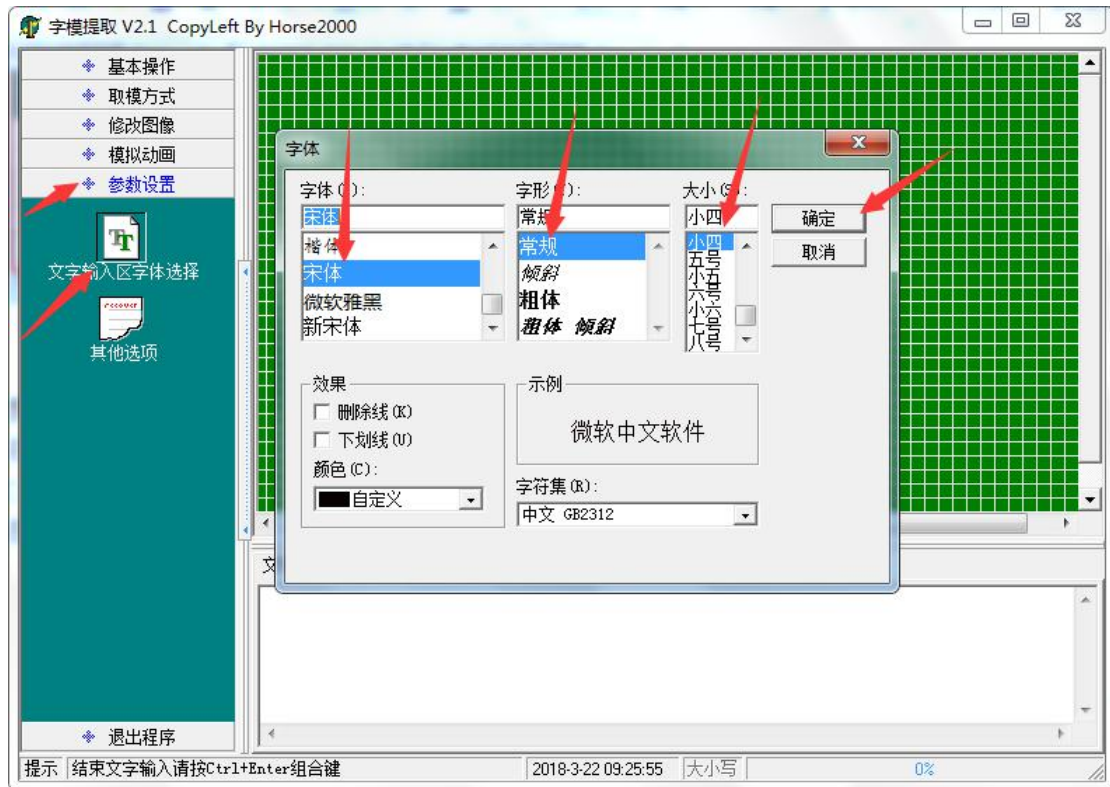


图 4.3

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”。

#### 4.1.2、设置“其它选项”

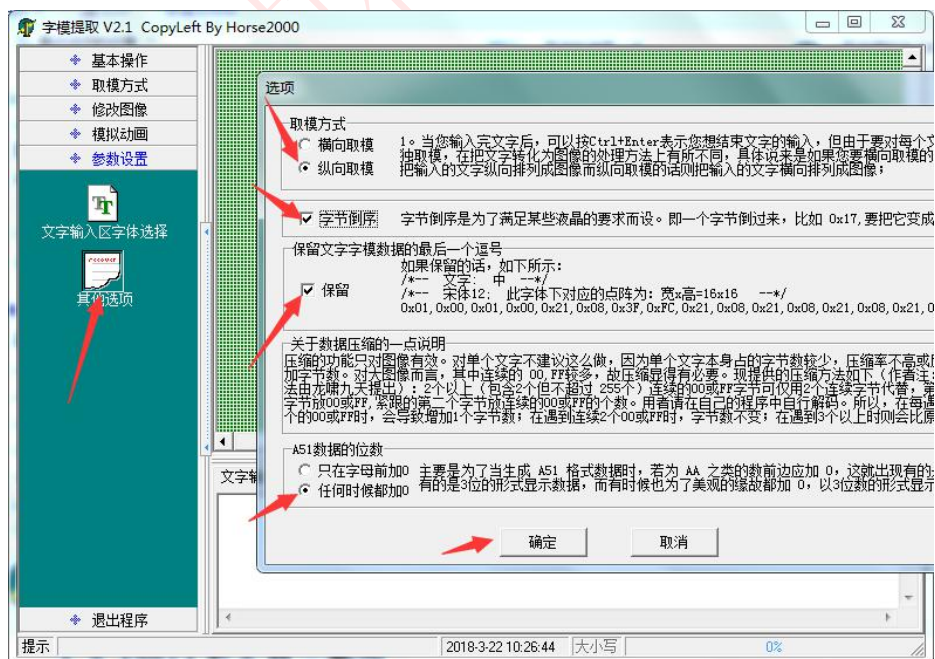


图 4.4

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。

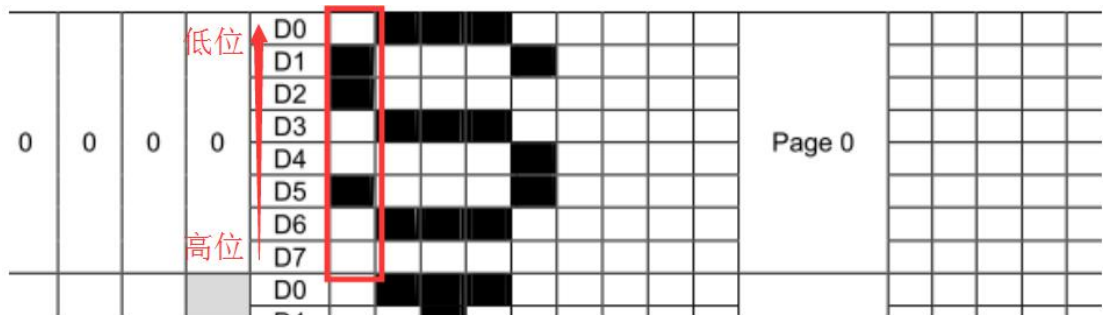


图 4.5

“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。

#### 4.1.3、输入文字

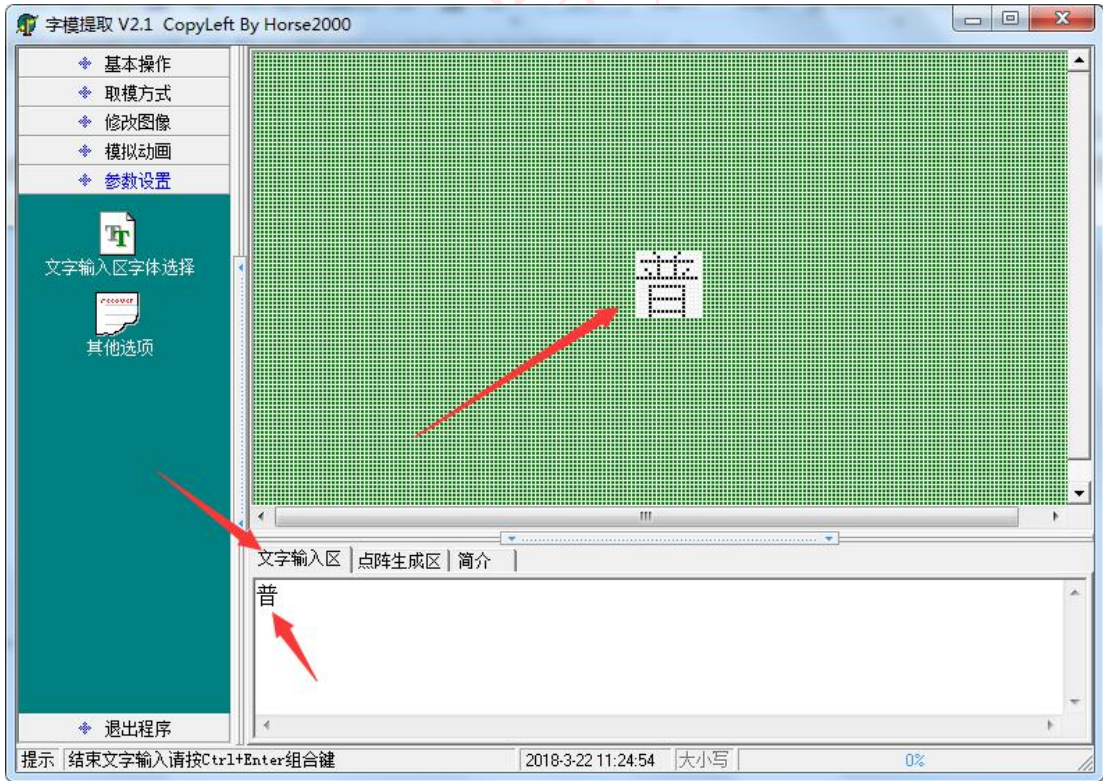


图 4.6

在文字输入区输入汉字，如图 4.6 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。

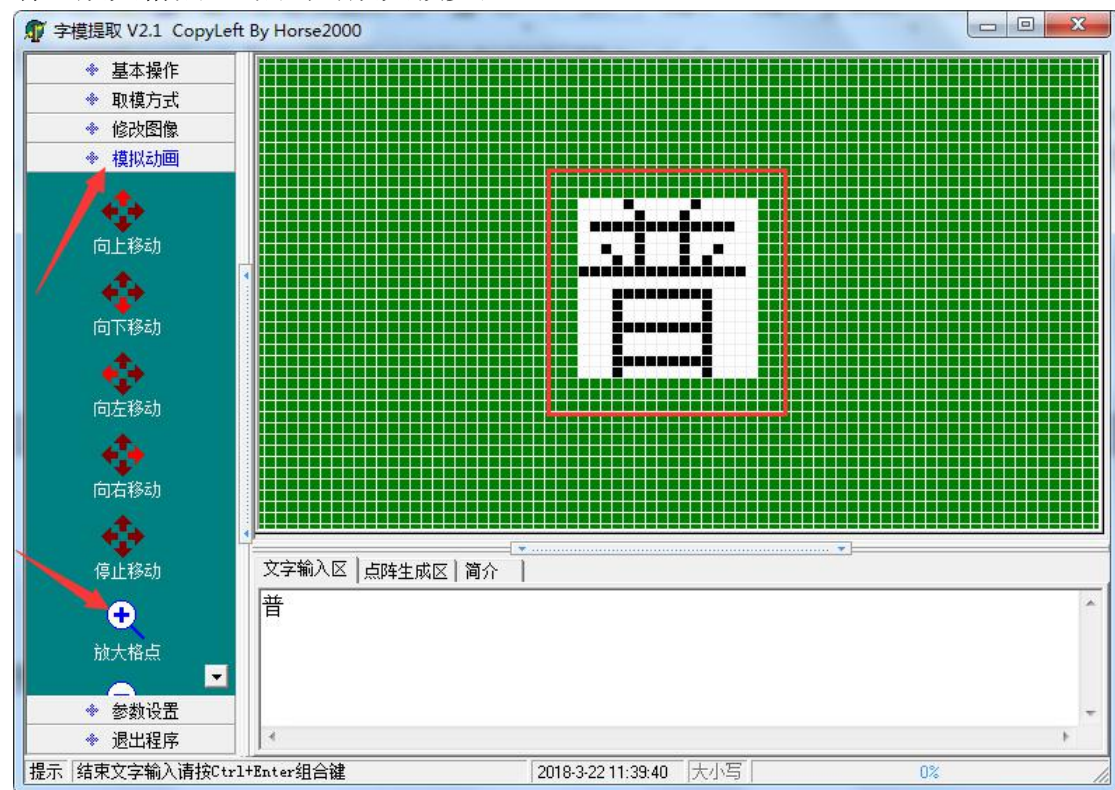


图 4.7

#### 4.1.4、生成字模

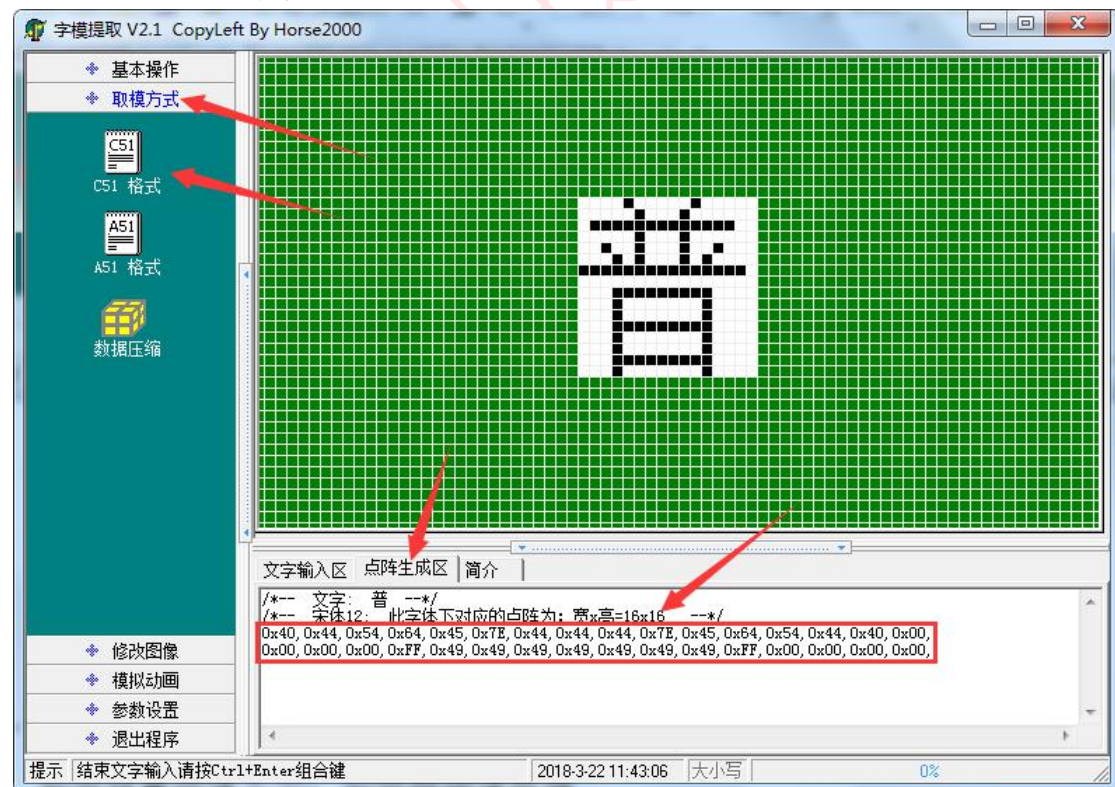


图 4.8

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，第二个自己从上到下是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。

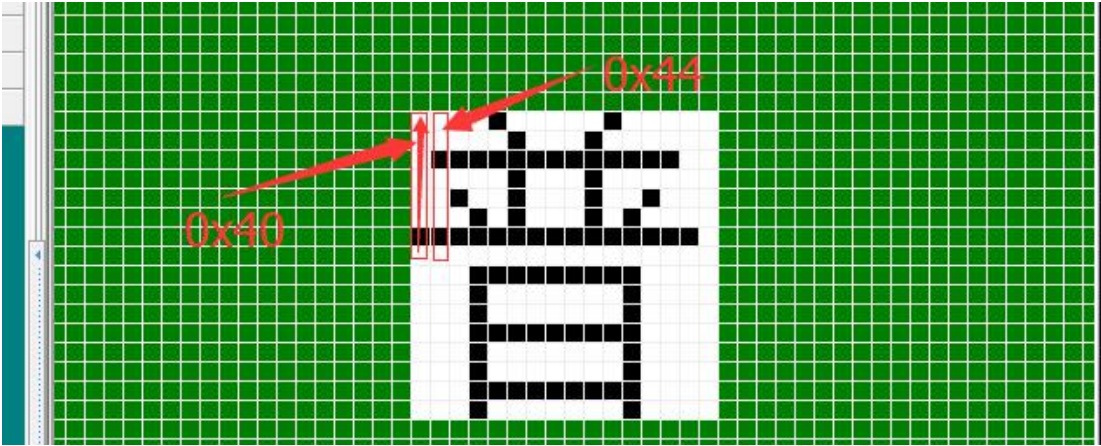


图 4.9

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。  
4.1.5、制作字模文件

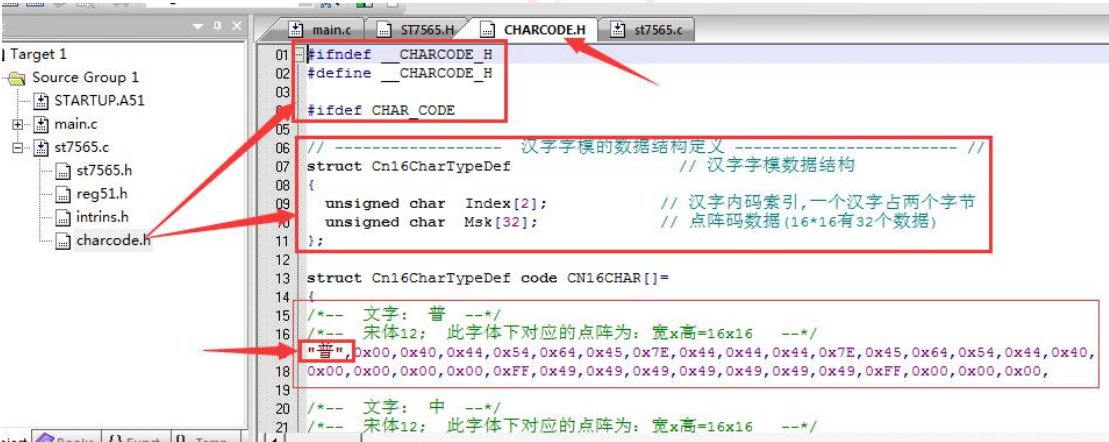


图 4.10

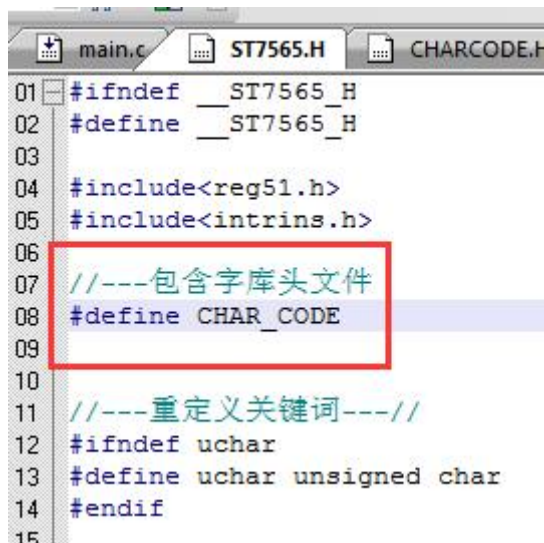
这里可以按照例程写头文件，做一个宏定义#ifdef CHAR\_CODE，这里用宏定义是为了在使用的时候，如果不使用文字，可以不定义这个，就不会被编译进来，是程序易于裁剪和编辑。  
从图 4.10 上可以看到在结构体数组中复制了之前生成的字模，然后在前面添加了索引。这里索引是用双引号加输入的字，做成一个字符串，占用两个字节，在结构体中是放到索引中的。从而只做了一个简单的字库。  
如果需要多个，可以重复上述步骤。一般一个数组中使用同一个大小的字做字库。可以看看例程里。

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。

## 4.2 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。

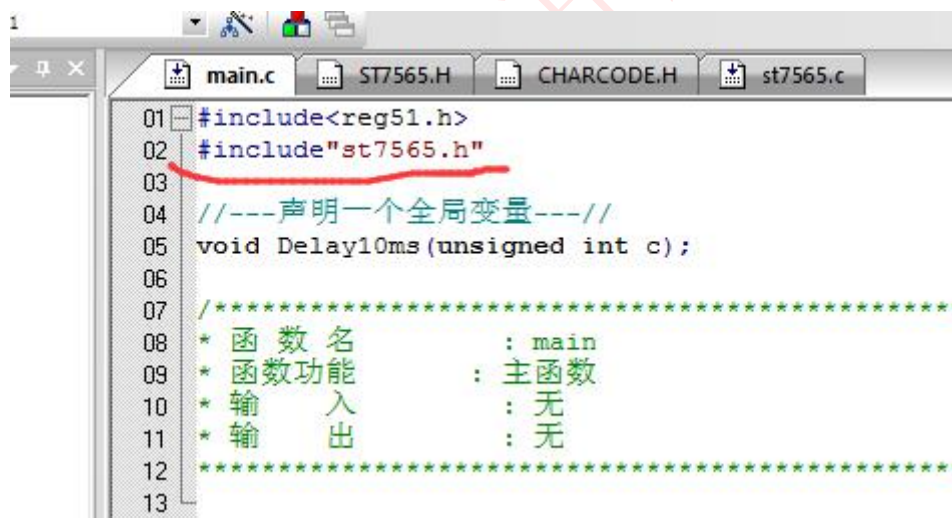
### 4.2.1 添加头文件支持



```
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
```

图 4.11

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。

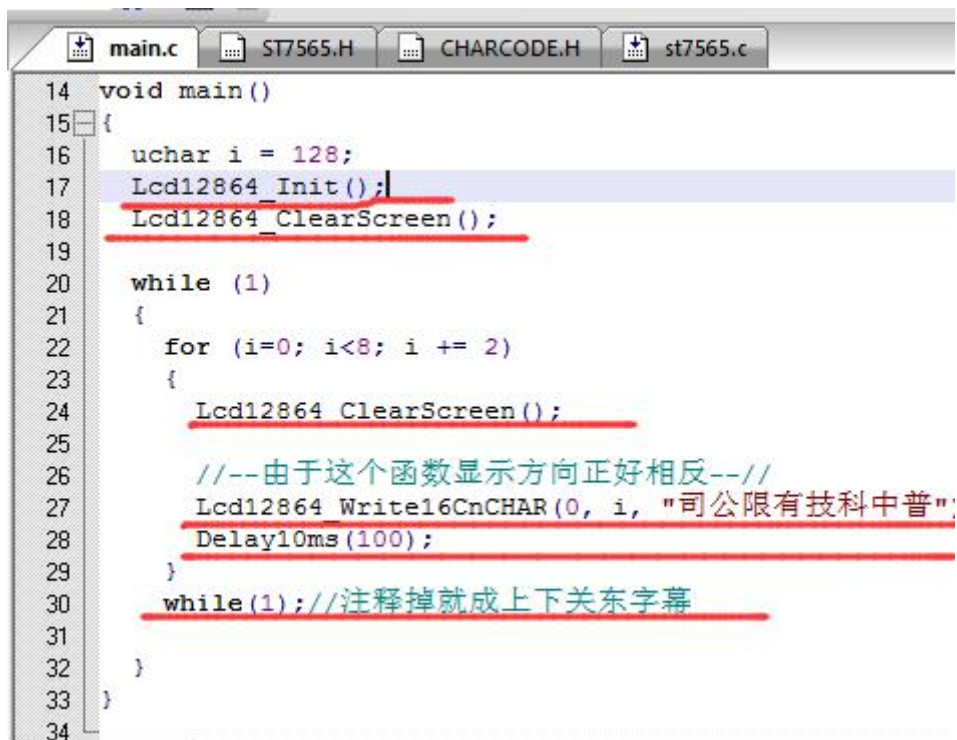


```
01 #include<reg51.h>
02 #include"st7565.h"
03
04 //---声明一个全局变量---//
05 void Delay10ms(unsigned int c);
06
07 /*****
08 * 函数名      : main
09 * 函数功能    : 主函数
10 * 输 入      : 无
11 * 输 出      : 无
12 *****/
13
```

图 4.12

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。

### 4.2.1 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         }
30         while(1); //注释掉就成上下关东字幕
31     }
32 }
33
34
```

图 4.13

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。最是四次循环，结束后停留在最后一行显示。

## 五. 显示图像

### 5.1 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。

#### 5.1.1 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：

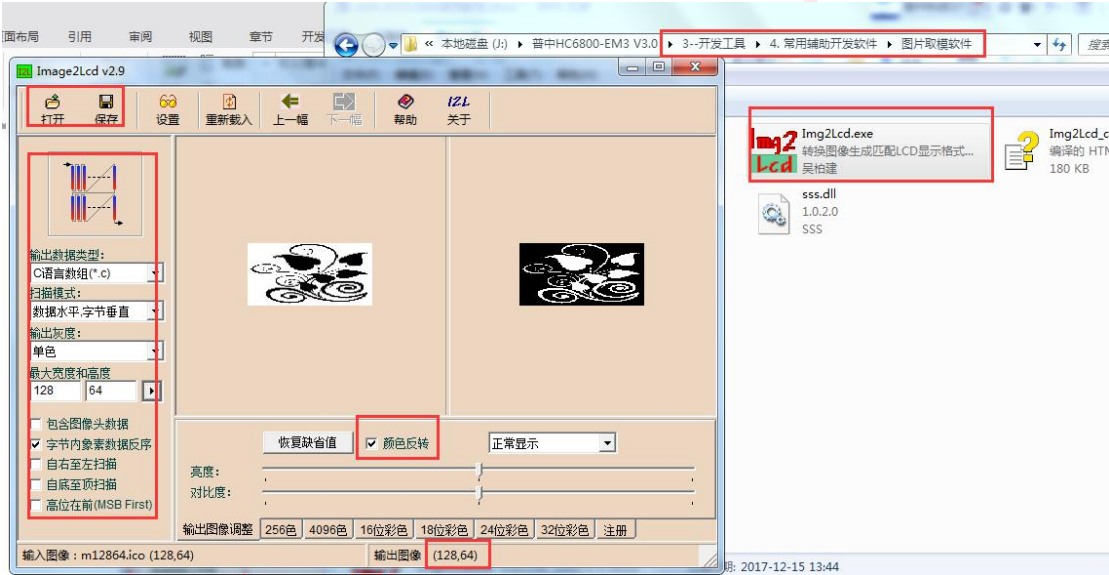


图 5.1

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。

#### 5.1.2 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。



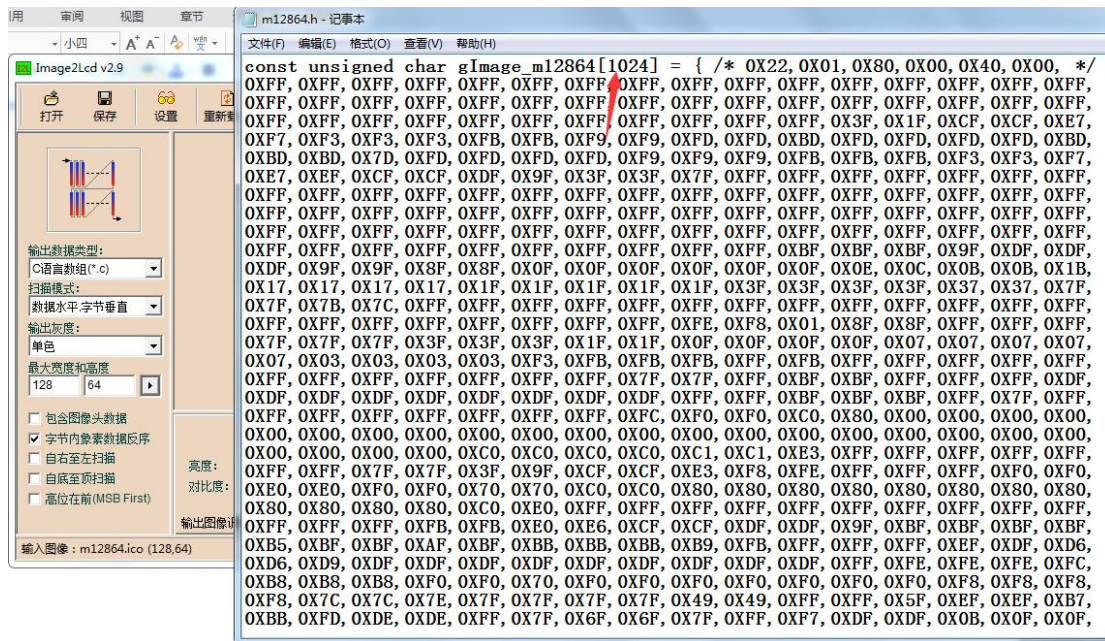


图 5.2

## 5.2 12864 如何显示图像

### 5.2.1 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。

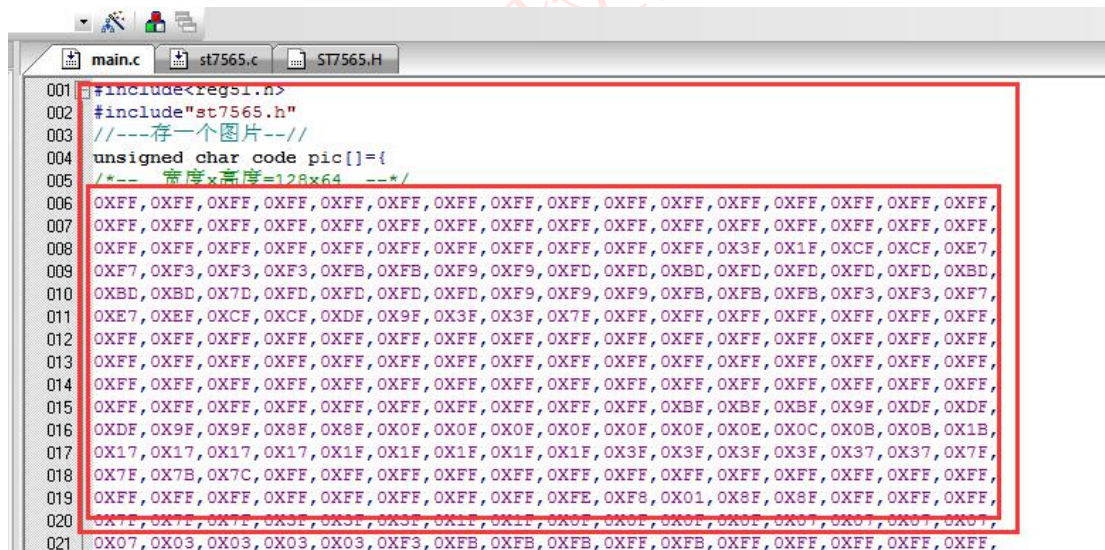


图 5.3

### 5.2.2 输出显示代码

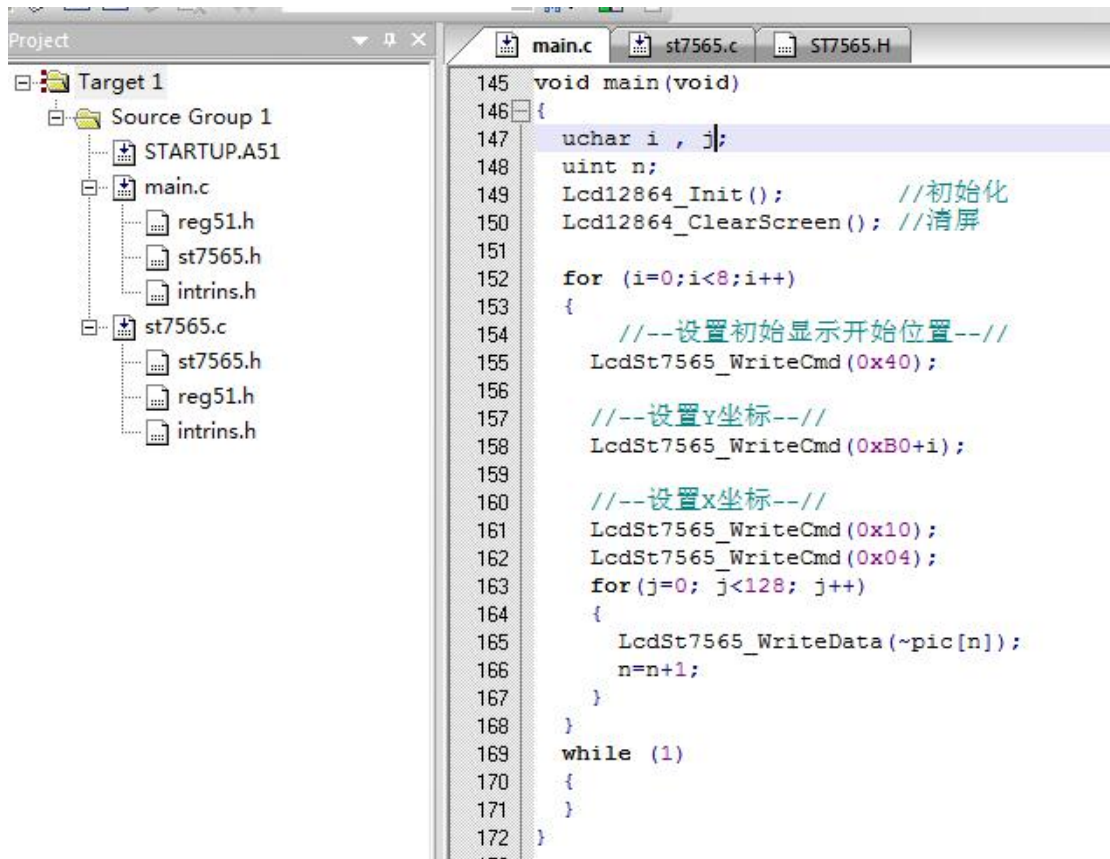


图 5.4

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了。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。

