LCD12864 液晶的使用

整理：大海橡树

LCD12864 分为两种，带字库和不带字库的，个人比较喜欢不带字库的，因为显示汉字的时候可以选择自己喜欢的字体，而带字库的液晶，只能显示 GB2312 的宋体，当然了，也可以显示其他的字体，不过不是液晶本身字库中带的了，而是用图片的形式显示。本讲由于内容较多，故分两篇进行讲解，本人水平有限，难免有错误之处，还望大家批评改正！

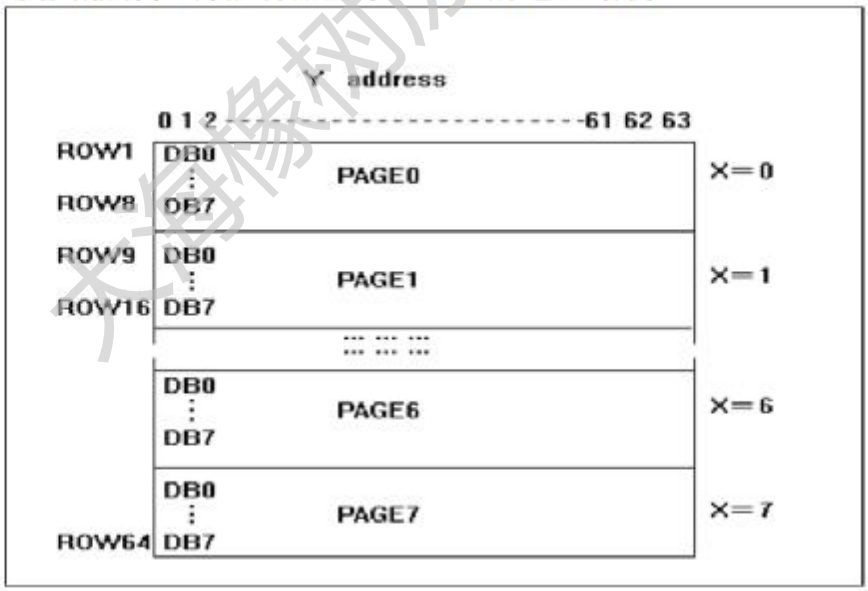
Ø 无字库型 LCD12864

首先介绍下不带字库的 LCD12864，现就以 Proteus 中的 LCD12864 为例进行讲解，Proteus 中 AMPIRE128*64，其液晶驱动器为 KS0108，我在网上搜了好就都没找到它的 datasheet，不过我们可以找到类似的芯片的手册，它的控制逻辑和 HD61202 是类似的，我们可以网上下载它的 datasheet 进行参考。

这块液晶的显示是左右和上下显示这一点一定要注意

与带字库液晶不同，此块液晶中含有两个液晶驱动器，一块驱动器控制 64*64 个点，左右显示，这就是为什么 AMPIRE128*64 引脚有 CS1 和 CS2 的原因。学习液晶主要看的它的指令系统，再次先说明一下“页”的概念，此液晶有 8 页，一页有 8 行。68/8=8；如下图所示。

四、HD61202 及其兼容控制驱动器显示 RAM 的地址结构



再介绍一下其他的几个重要指令：

指令一、行设置命令

RW	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	1	X	X	X	X	X	X

由此可见其显示的其实行为 0xC0，有规律的改变起始行号，可以实现滚屏的效果

指令二、页(page)设置指令

RW	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	1	1	1	X	X	X

起始页为 0xB8 显示的 RAM 共 64 行，分为 8 页，每页有 8 行，刚才在上面已经讲过。

指令三、列(Y address)地址设置指令

RW	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	X	X	X	X	X	X

第一列为 0x40 一直到 0x7F 共 64 列，因为此液晶有 128 列，所以有两块驱动芯片驱动。

指令四、读状态指令

RW	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	Busy	0	ON/OFF	RESET	0	0	0	0

Busy：为 1 内部忙，不能对液晶进行操作。0--工作正常。

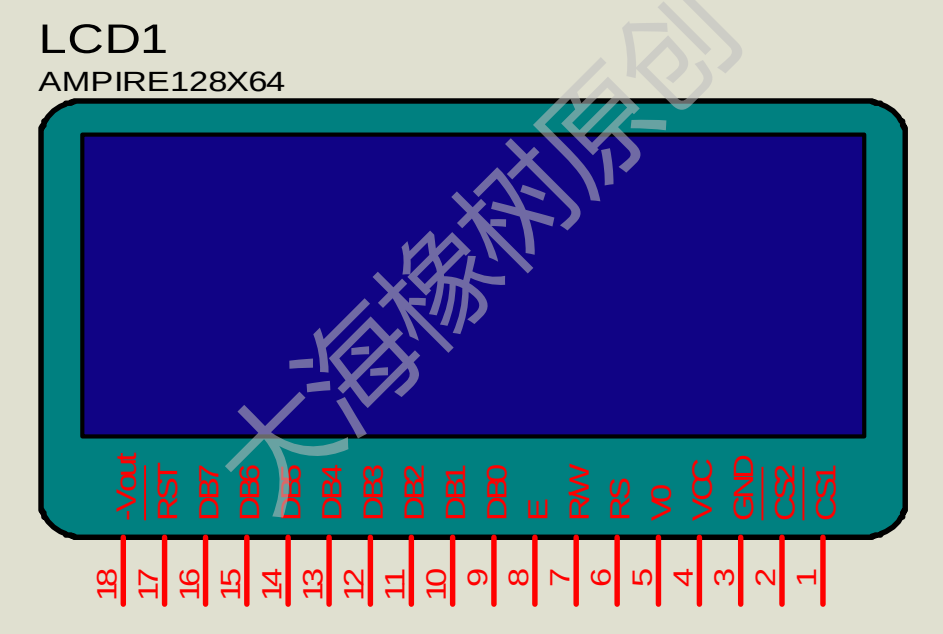
ON/OFF：1------显示关闭；0------显示打开

RESET：1-----复位状态；0------正常。

说明在 Busy 和 RESET 状态时，除读状态指令外，其他任何指令均不会对驱动器产生作用。

其他的读数据和写数据和 LCD1602 是一样的，由于篇幅有限这里就不赘述了。不懂的可以参看下液晶手册。另外要说明的就是 CS1 和 CS2 的作用(两者都是低电平有效）

引脚图：



HD61202 及其兼容控制驱动器的引脚功能如下：

引脚符号	状态	引脚名称	功　　能
CS1,CS2, CS3	输入	芯片片选端	CS1 和 CS2 低电平选通，CS3 高电平选通
E	输入	读写使能信号	在 E 下降沿，数据被锁存(写)入 HD61202 及其兼容控制驱动器；在 E 高电平期间，数据被读出
R/W	输入	读写选择信号	R/W＝1 为读选通， R/W＝0 为写选通
D/I	输入	数据、指令选择信号	D/I＝1 为数据操作， D/I＝0 为写指令或读状态
DB0~DB7	三态	数据总线	
RST	输入	复位信号	复位信号有效时，关闭液晶显示，使显示起始行为 0。RST 可跟 MPU 相连，由 MPU 控制；也可直接接 Vcc，使之不起作用。

CS1 和 CS2 的屏幕选择说明

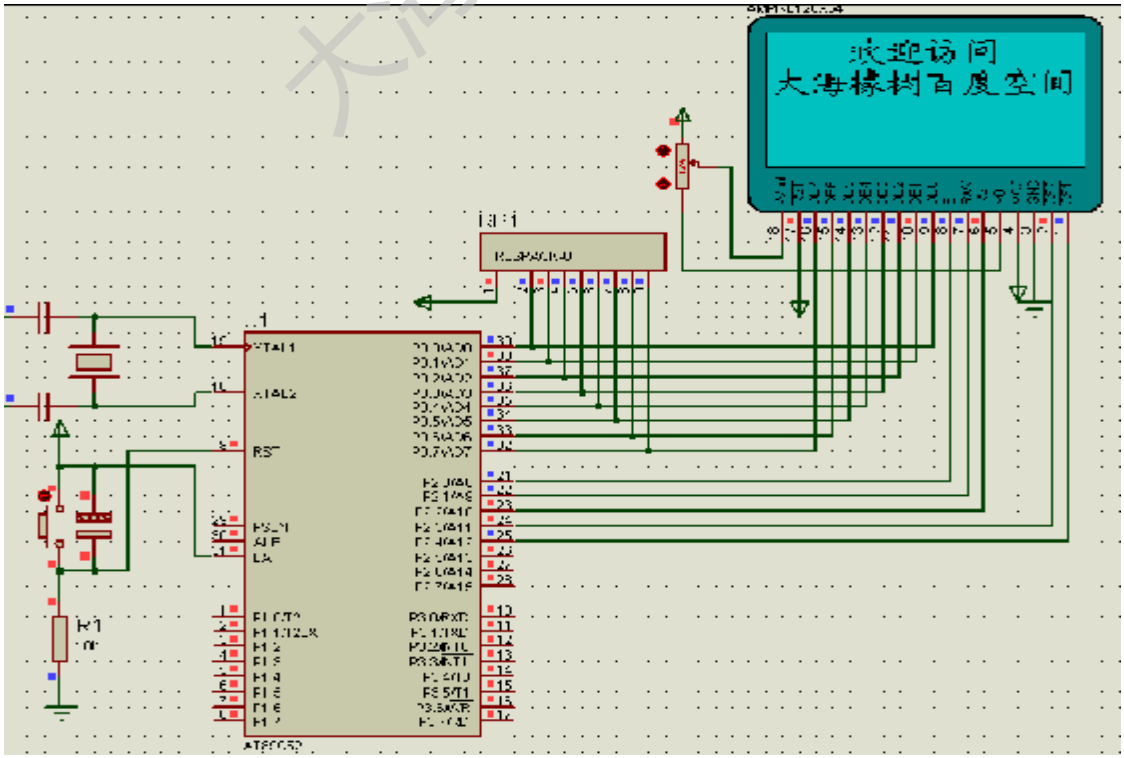
CS1	CS2	选屏
0	0	全屏
0	1	左半屏
1	0	右半屏
1	1	不选

注：CS1 和 CS2 均为低电平有效

一、显示汉字

由于这块液晶补带字库我们就要自己编写字库，编写字库的软件还是 Zimo21，LCD1602 显示自定义字符的时候用的也是它。不过在取模之前我们要进行一些设定，根据此液晶显示显示的原理，设置如下：(若不是这样，取模的数据将不是我们想要的)

在此我选择楷体小四号字体，效果如下图所示：



下面是程序代码：

```
/****http://hi.baidu.com/echoas******
程序：LCD12664 液晶显示原理
内容：显示汉字
学习板：Proteus 仿真图
液晶：AMPIRE128X64(无字库)
软件：keil uVision3
作者：大海橡树
整理日期：2010-12-02
******http://hi.baidu.com/echoas****/
#include <AT89X52.h>
#include <intrins.h>
#include"ZK.h"     //中文字库
#define uchar unsigned char
#define uint  unsigned int
#define LCD_databus P0 //LCD8 位数据口
uchar num;
sbit RS=P2^2;      //RS 为 0 命令；1 数据
sbit RW=P2^1;      //RW 为 1--写；0--读
sbit EN=P2^0;      //使能端
sbit CS1=P2^4;     //片选 1 低电平有效，控制左半屏
sbit CS2=P2^3;     //片选 1 低电平有效，控制右半屏
void delay(uint i)
{
    while(--i);
}
void Read_busy()//读“忙”函数-----数据线的最高位 DB71 则 busy
{
    P0=0x00;
    RS=0;
    RW=1;
    EN=1;
    while(P0 & 0x80);
    EN=0;
}
void write_LCD_command(uchar value)
{
   Read_busy();//每次读写都要忙判断
   RS=0;       //选择命令
   RW=0;       //读操作
   LCD_databus=value;
   EN=1;       //EN 由 1----0 锁存有效数据
       _nop_();
       _nop_();
       EN=0;
}

void write_LCD_data(uchar value)//写数据函数
{
   Read_busy();
   RS=1;           //选择数据
   RW=0;
   LCD_databus=value;
   EN=1;           //EN 由 1----0 锁存有效数据
       _nop_();
       _nop_();
       EN=0;
}

void Set_page(uchar page)//设置“页” LCD12864 共 8 页，一页是 8 行点阵点
{
   page=0xb8|page;       //页的首地址为 0xB8
   write_LCD_command(page);
}

void Set_line(uchar startline)  //设置显示的起始行
{
   startline=0xC0|startline;      //起始行地址为 0xC0
   write_LCD_command(startline); //设置从哪行开始：共 0--63;一般从 0 行开始显示
}

void Set_column(uchar column)    //设置显示的列
{
   column=column &0x3f;       //列的最大值为 64
   column= 0x40|column;       //列的首地址为 0x40
   write_LCD_command(column);//列位置
```

```
}
void SetOnOff(uchar onoff)  //显示开关函数：0x3E 是关显示，0x3F 是开显示
{
    onoff=0x3e|onoff; //0011 111x,onoff 只能为 0 或者 1
    write_LCD_command(onoff);
}
void SelectScreen(uchar screen) //选屏
{
   switch(screen)
   {
     case 0: CS1=0;CS2=0;break; //全屏
     case 1: CS1=0;CS2=1;break;//左半屏
     case 2: CS1=1;CS2=0;break;//右半屏
     default:break;
   }
}
void ClearScreen(uchar screen)  //清屏
{
    uchar i,j;
    SelectScreen(screen);
for(i=0;i<8;i++)//控制页数 0-7，共 8 页
     {
       Set_page(i);
        Set_column(0);
        for(j=0;j<64;j++) //控制列数 0-63，共 64 列
        {
         write_LCD_data(0x00); //写入 0，地址指针自加 1
        }
     }
}
void InitLCD()    //LCD 的初始化
{
    Read_busy();
    SelectScreen(0);
    SetOnOff(0); //关显示
    SelectScreen(0);
    SetOnOff(1); //开显示
    SelectScreen(0);
    ClearScreen(0);//清屏
    Set_line(0); //开始行:0
}
void Display(uchar ss,uchar page,uchar column,uchar *p)  //显示汉字
{
    uchar i;
    SelectScreen(ss);
    Set_page(page);         //写上半页
    Set_column(column); //控制列
    for(i=0;i<16;i++)       //控制 16 列的数据输出
    {
        write_LCD_data(p[i]); //汉字的上半部分
    }
    Set_page(page+1);  //写下半页
    Set_column(column);     //控制列
    for(i=0;i<16;i++)  //控制 16 列的数据输出
    {
        write_LCD_data(p[i+16]);
    //汉字的下半部分
    }
}
void main()
{
    InitLCD();              //初始化
    ClearScreen(0);         //清屏
    Set_line(0);       //显示开始行
Display(1,0,2*16,huan);//欢
Display(1,0,3*16,ying);//迎
Display(2,0,0*16,fang);//访
Display(2,0,1*16,wen);//问
Display(1,2,0*16,da);//大
Display(1,2,1*16,hai);//海
Display(1,2,2*16,xiang);//橡
Display(1,2,3*16,shu);//树
Display(2,2,0*16,bai);//百
Display(2,2,1*16,du);//度
Display(2,2,2*16,kong);//空
Display(2,2,3*16,jian);//间
While(1);
}
}
```

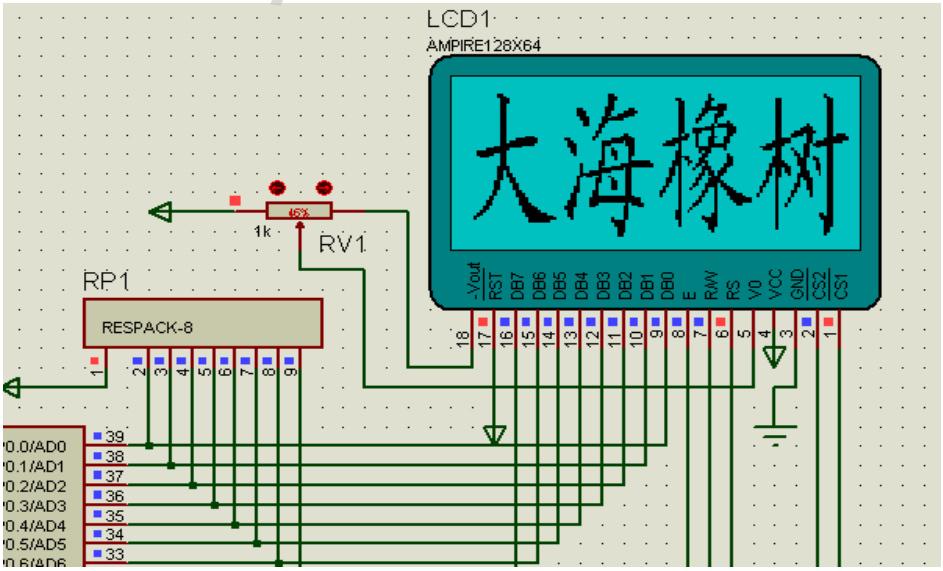
上面说了，我们可以通过改变页的地址的变化实现 LCD 滚屏的效果，大家可以试一下，很简单！

如果单片机容量够的话，可以做个电子书，呵呵！显示汉字部分就介绍到这里，那么显示汉字回了，显示一张图片就简单了！

二、显示图片

显示图片和显示汉字的原理是一样的，只不过是大小的问题！现在我们就以一张 128*64 分辨率的图片进行取模。

取模图片：



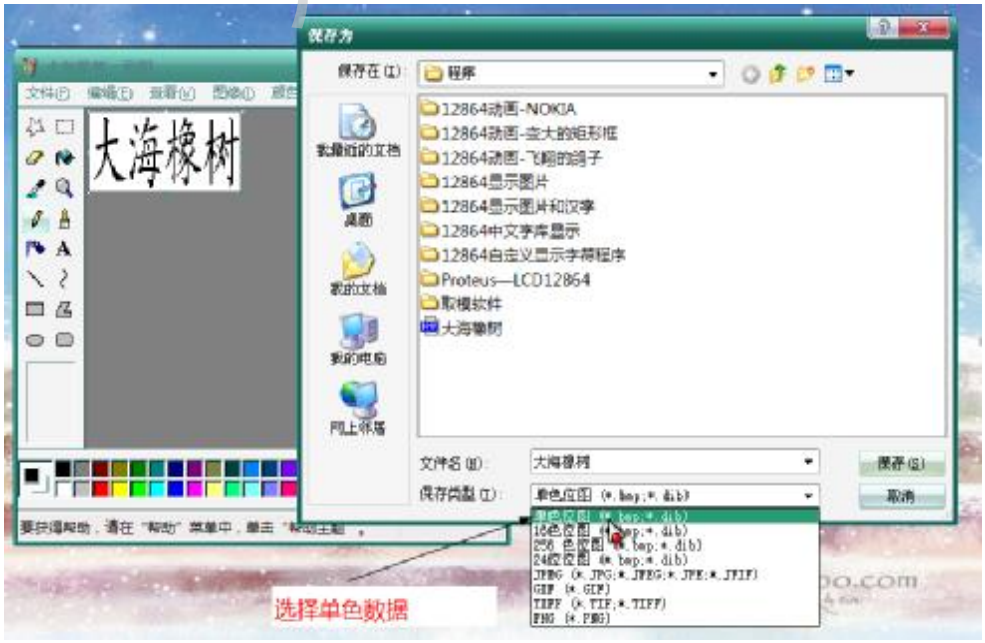
现在我们主要看一下显示图片的函数：

void display_BMP(uchar a[][64])//显示图形的左边部分

```
{
        uchar i,j;
        for(j=0;j<8;j++)
        {
        SelectScreen(1);
        Set_page(j);
        Set_column(0);
        for(i=0;i<64;i++)   //显示左屏
        {
            write_LCD_data(a[2*j][i]);//每隔一行取一次数组中的数据
        }
        SelectScreen(2);
        Set_page(j);
        Set_column(0);
        for(i=0;i<64;i++)   //显示右屏
        {
            write_LCD_data(a[2*j+1][i]);//每隔一行取一次数组中的数据
        }
        }
}
```

取出图片的数据是二维数组形式！注意，液晶是先显示左半屏，再显示右半屏，但是取数组中的元素是逐个往下取出的，所以是对于一个半屏而言，每隔一行取数组中的元素，而隔去的那一行是另一个半屏要取的数据！这一点一定要注意。

还有一点要补充的就是，这个取模软件只能识别 BMP 和 ICO 格式的单色图片，如果我们选择的图片不是 128*64 大小的话，可以通过软件将其改成 128*64 大小的，如果我们直接用软件取模的话，显示的会是乱码，解决办法是我们可以通过 windows 自带的画图工具，什么也不做修改，将大小设置好的图片另存为 BMP 单色文件就行了，然后再用取模软件取模就 OK 了，O(∩_∩)O~！算了，给大家截个图



如果大家不想这么麻烦的话就网上直接搜 128*64 大小的图片也可以！

我当时因为不知道原因，显示的是乱码，结果找到原来是这样子的！还以为是程序的原因，浪费了我不少时间，结果发现原来是这个原因!希望大家引以借鉴！呵呵！共同学习、进步。

对于其他部分的函数，和显示汉字的一样，这里就不做介绍了，大家看下代码就清楚了!

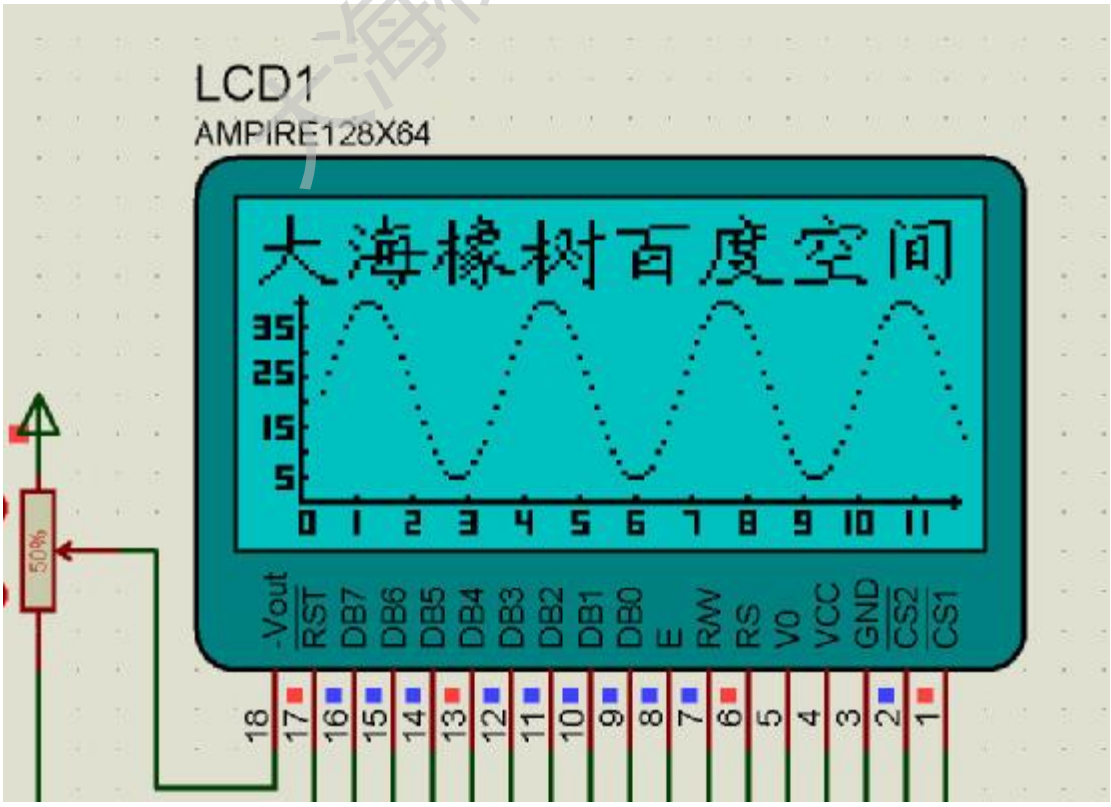
显示一张静态的图片没有问题，那么就能显示一个动画了！其实动画就是一张张静态的图片不断的刷新就可以了，根据人的视觉暂留效果就可以做出一个动画！我试着写了个动画的程序，完全可行！但是要注意图片刷新的时间！其实关于时间刷新的问题，LCD12864 大概 1S 钟可以显示 10 张图片，这对于显示动画足够了！大家在写程序的时候主要注意两个函数，一个是写数据函数 write_LCD_data 和写命令函数 write_LCD_command，不管让液晶显示什么，我们都要不断的调用这两个函数，显示的内容越多，调用这两个函数的次数就越多，如果这两个函数执行的时间过长，就会造成图片的刷新频率过低，就没办法显示动画了！所以我在写这两个函数的时候 EN 从 1 变到 0，用了 delay(2);短暂的延时，大家可以试一下！

PS：动画呢，大家可以选择一张动态的 gif 图片，然后对图片的每一帧截图；我在网上搜了个可以显示每一帧图的动态图片，名字叫做 jiftools，将一张动态图片导入，就可以显示每一帧数据，很方便呵呵！现在网上真是什么软件都有，大家一定要利用一下网上的资源，百度、谷歌是很好的老师！呵呵，话扯远了，言归正传。我的这个动画呢，是找了一张飞翔的鸽子的图片，然后一张张取模弄出来的！

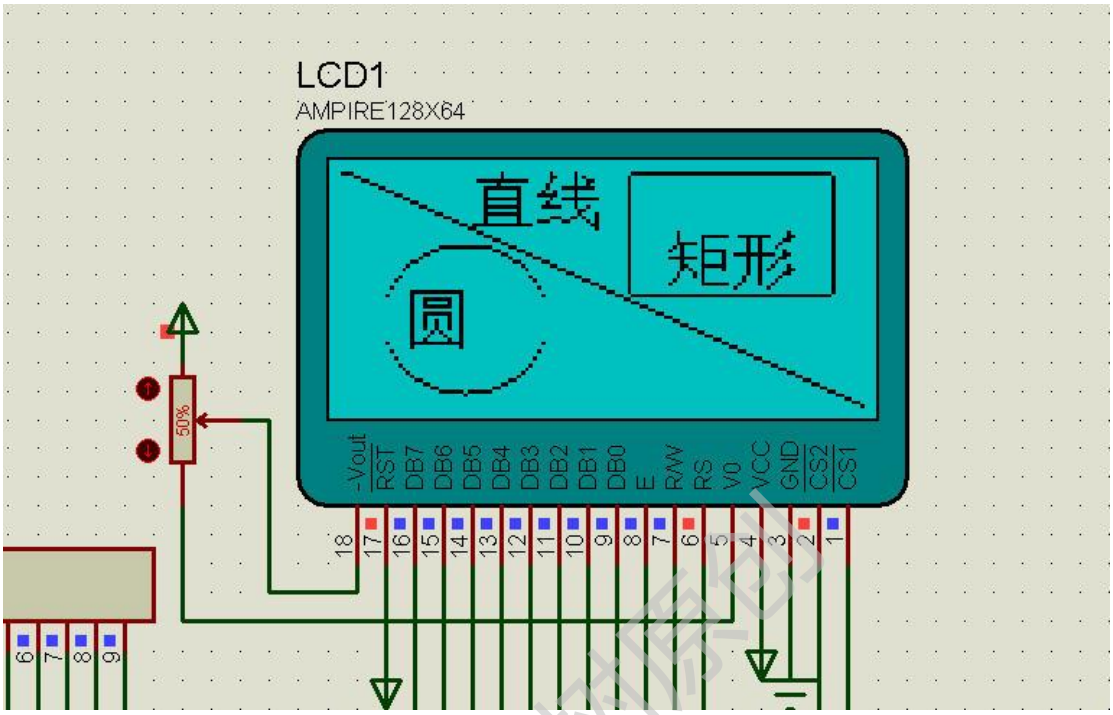
好了，到此就将 Proteus 中的 LCD12864 讲解完毕了，其实学习液晶主要是多思考，多做练习，一定要清楚液晶的工作原理和各种指令!从最开始显示一个简单的字符，再到显示汉字，显示图片、动画。一步一个脚印，你就会发现，其实液晶就这么点东西，也不是很难！

下面是我写个几个程序的仿真效果：

显示正弦曲线：

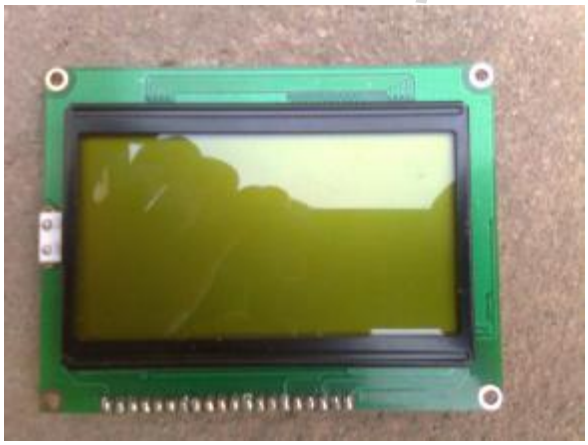


显示几何图形：



Ø 字库型 LCD12864

下面介绍下带字库的液晶，由于 Proteus 中没有，就以实物为准吧！我手头上这块液晶是 QY128*64HZ1，它的驱动器是 ST7920，想必大家很熟悉了，百度、谷歌一下它的芯片手册很多!在学习此块液晶之前,建议大家好好看看它的驱动芯片的手册!它的驱动和 LCD1602 很像，甚至，读忙、写指令和写数据函数都是一样的，就初始化不一样，因为指令系统不同嘛！下面是我手头字库液晶的实物图。



字库型液晶显示可以分为串行方式和并行方式两种，通过引脚 PSB 进行选择，它只有一个驱动芯片， 不像 Proteus 中无字库液晶有两个驱动芯片。显示是整体显示，而不是左右屏的显示！大家一定要注意！

再看一下引脚分布：

引脚号	标识	说明
PIN1	GND	接0V
PIN2	VCC	接4.8V-5V
PIN3	V0	VCC和VEE接可调电阻，中间抽头接至V0
PIN4	RS CS	并行模式：RS=0，指令寄存器；RS=1，数据寄存器。 串行模式：片选
PIN5	R/W SID	并行模式：R/W=0，写；R/W=1，读。 串行模式：数据
PIN6	E SCK	并行模式：允许信号。串行模式：脉冲
PIN7	D0	并行模式：数据0； 串行模式：不连接
PIN8	D1	并行模式：数据1； 串行模式：不连接
PIN9	D2	并行模式：数据2； 串行模式：不连接
PIN10	D3	并行模式：数据3； 串行模式：不连接
PIN11	D4	并行模式：数据4； 串行模式：不连接
PIN12	D5	并行模式：数据5； 串行模式：不连接
PIN13	D6	并行模式：数据6； 串行模式：不连接
PIN14	D7	并行模式：数据7； 串行模式：不连接
PIN15	PSB	并行模式：PSB=1；串行模式：PSB=0
PIN16	NC	不需连接
PIN17	/RST	复位
PIN18	NC	不需连接
PIN19	LED+	背光正极，接4.8V～5V
PIN20	LED-	背光负极，接0V

1、控制口信号说明：

注：①忙标志 Bust_flag=1 说明 LCD 内部正忙，此时不能对 LCD 进行操作，忙标志的判断由 DB7 也就是数据口的最高位所决定！这和 LCD1602 一样！

②上面对 RS 和 RW 的操作需配合使能信号 EN 来操作！否则无效！

2、显示说明

（1）、字符产生 ROM(CGROM)

RS	RW	功能
0	0	单片机写指令到指令暂存器(IR)
0	1	读出忙标志(BF)及地址计数器(AC)的状态
1	0	单片机写数据到数据暂存器(DR)
1	1	单片机从数据暂存器中读出数据

里面提供了 8192(2^{13})个汉字 GB2132 宋体

（2）、显示数据 RAM（DDRAM）

内部提供 64*2 位空间，最多可控制 4 行 16 字，也就是 16 个中文字型显示，当写入显示数据 RAM 时，可分别显示 CGROM 和 CGRAM 的字型，可以用来显示三种字型：半角英文数字型、CGRAM 字型和 CGROM 的中文字型，三种字型的选择，由在 DDRAM 总写入的编码选择，在 0000H—0006H 的编码中（其代码分别为 0000、0002、0004、0006 共四个）将选择 CGRAM 的自定义字型，02H—7FH 的编码中将显示半角英文数字型的字型（也就是 ASCII 码，大小为 16*8），至于 A1 以上的编码将自动结合下一个位元组，组成两个位元组的编码，从而形成一个中文字型的编码，也就是说显示一个汉字要两个 ASCII 码显示的位置，即大小为 16*16。BIG（A140—D75F）,GB(A1A0—F7FF)。

（3）、字型产生 RAM

上面已经介绍了该种液晶提供四组可定义显示，是 16*16 大小的自定义图像空间，通过在特

定的编码位置，写入我们要显示的自定义图像即可，这个和 LCD1602 液晶的自定义显示字符的原理是一样的！这个将在下文加以详细介绍！

3、指令说明

指令表 1：（RE=0：基本指令）

指令	指令码										功能
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
清除显示	0	0	0	0	0	0	0	0	0	1	将DDRAM填满"20H"，并且设定DDRAM的地址计数器(AC)到"00H"
地址归位	0	0	0	0	0	0	0	0	1	X	设定DDRAM的地址计数器(AC)到"00H"，并且将游标移到开头原点位置；这个指令不改变DDRAM的内容
显示状态开/关	0	0	0	0	0	0	1	D	C	B	D=1：整体显示 ON C=1：游标 ON B=1：游标位置反白允许
进入点设定	0	0	0	0	0	0	0	1	I/D	S	指定在数据的读取与写入时，设定游标的移动方向及指定显示的移位
游标或显示移位控制	0	0	0	0	0	1	S/C	R/L	X	X	设定游标的移动与显示的移位控制位；这个指令不改变DDRAM的内容
功能设定	0	0	0	0	1	DL	X	RE	X	X	DL=0/1：4/8 位数据 RE=1：扩充指令操作 RE=0：基本指令操作
设定CGRAM地址	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	设定 CGRAM 地址
设定DDRAM地址	0	0	1	0	AC5	AC4	AC3	AC2	AC1	AC0	设定 DDRAM 地址（显示位址） 第一行：80H－87H 第二行：90H－97H
读取忙标志和地址	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	读取忙标志(BF)可以确认内部动作是否完成，同时可以读出地址计数器(AC)的值
写数据到 RAM	1	0	数据								将数据 D7——D0 写入到内部的 RAM（DDRAM/CGRAM/IRAM/GRAM）
读出 RAM 的值	1	1	数据								从内部 RAM 读取数据 D7——D0 (DDRAM/CGRAM/IRAM/GRAM)

指令表 2：（RE=1：扩充指令）

指令	指令码										功能
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
待命模式	0	0	0	0	0	0	0	0	0	1	进入待命模式，执行其他指令都保终止 待命模式
卷动地址开关开启	0	0	0	0	0	0	0	0	1	SR	SR=1：允许输入垂直卷动地址 SR=0：允许输入 IRAM 和 CGRAM 地址
反白选择	0	0	0	0	0	0	0	1	R1	R0	选择 2 行中的任一行作反白显示，并可反白与否。初始值 R1R0=00，第一次设反白显示，再次设定变回正常
睡眠模式	0	0	0	0	0	0	1	SL	X	X	SL=0：进入睡眠模式 SL=1：脱离睡眠模式
扩充功能设定	0	0	0	0	1	CL	X	RE	G	0	CL=0/1：4/8 位数据 RE=1：扩充指令操作 RE=0：基本指令操作 G=1/0：绘图开关
设定绘图 RAM 地址	0	0	1	0 AC6	0 AC5	0 AC4	AC3 AC3	AC2 AC2	AC1 AC1	AC0 AC0	设定绘图 RAM 先设定垂直(列)地址 AC6AC5…AC0 再设定水平(行)地址 AC3AC2AC1AC0 将以上 16 位地址连续写入即可

具体的指令上面都说明了，大家好好看看就行了！显示基本的字符和汉字用的是基本指令，显示图片用的是扩充指令！扩充指令中有个绘图开关 G，详细的操作在下文介绍。

由于串行方式比较慢，在此采用并行方式。

一、显示 ASCII 码

显示 ASCII 码和 LCD1602 是一样的，只需将字符对应的 ASCII 的数据送入液晶进行显示即可。具体的参看 LCD1602 部分

二、显示字库中的汉字

一个汉字的大小是 16*16，占两个 ASCII 字符的位置，将相应的汉字的编码分两次送入液晶

显示即可。也可以用数组的方式，编译器编译时自动将转换成对应的编码。

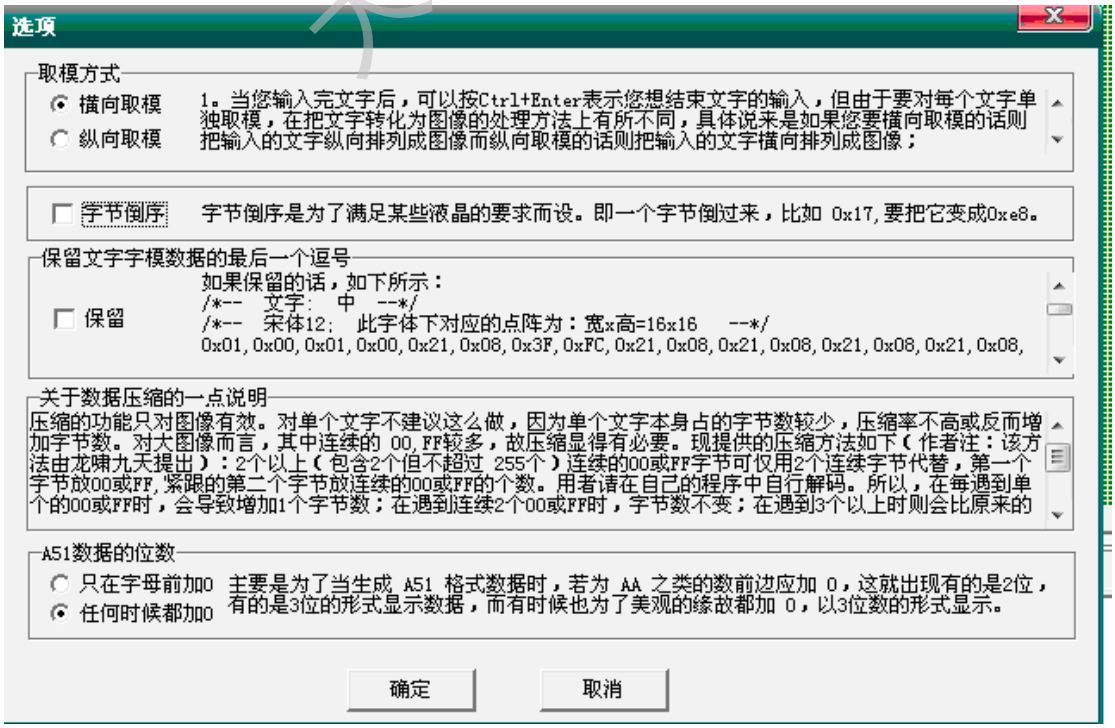
三、显示四个自定义字符

刚才说了，此块液晶可以自定义显示四个 16*16 字符，对应的编码为 0000、0002、0004 和 0006，这个到底是什么意思呢？其实就是说自定义显示字符在 RAM 区的编码，我们通过向自定义字符地址中送入自定义的数据，然后调用自定义编码就可以将自定义字符显示在液晶上面了！

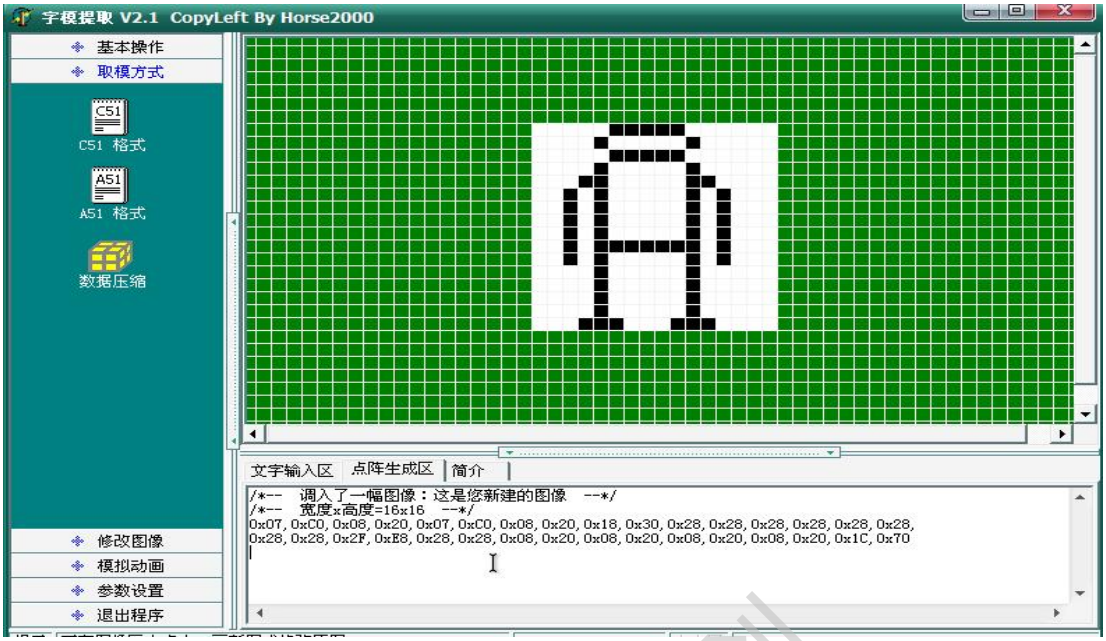
上面说了三种显示，代码就集中在一起写了，这个程序是显示 ASCII 码、字库汉字和自定义显示：

先看效果图：

取模设定和 Proteus 中的 128*64 不一样：取模软件是一样的！设定如下图所示



上面的自定义图形自己一个个点吧，呵呵，没什么捷径！



下面是代码：

==========START==========

/**********http://hi.baidu.com/echoas**********

程序：LCD12864 自定义显示字符(最多显示 4 个)

内容：ASCII 码显示、自带字库显示(8192 个)、自定义显示字符(4 个)。

软件：keil uVision3

液晶：QYLCD12864HZ1(带字库)

作者：大海橡树

整理日期：2010-11-28

**********http://hi.baidu.com/echoas**********/

```
#include<AT89X52.h>
#define uchar unsigned char
#define uint  unsigned int
sbit RS=P2^5;
sbit RW=P2^6;
sbit EN=P2^7;                    //液晶的三个控制端
//sbit PSB=P2^4;//1---8 位或 4 位并口方式；0---串口方式。单片机上电高电平，选择的是并口方式
//繁体中文字“龍”
uchar table[]={0x10,0x80,0x08,0x80,0x7E,0xFC,0x24,0x80,0x18,0xFC,0xFF,0x04,0x00,0xFC,0x3E,0x80,
0x22,0xF8,0x3E,0x80,0x22,0xF8,0x3E,0x80,0x22,0xFA,0x22,0x82,0x2A,0x82,0x24,0x7E};
//一个星星、一个月亮
unsigned char code zk[]={0x08,0x20,0x1c,0x10,0x1c,0x1c,0xff,0x9e,0x7f,0x1e,0x1c,0x1f,0x3e,0x1f,
0x3e,0x1f, 0x77,0x1f,0x41,0x3f,0x00,0x7e,0x00,0xfe,0x83,0xfc,0x7f,0xf8,0x3f,0xf0,0x0f,0xc0,
};
//小人
```

```
uchar code table2[]={0x03,0xC0,0x04,0x20,0x04,0x20,0x03,0xC0,0x0C,0x30,0x14,0x28,0x14,0x28,0x14,0x28,
0x14,0x28,0x17,0xE8,0x04,0x20,0x04,0x20,0x04,0x20,0x04,0x20,0x0E,0x70,0x0E,0x70};
//喇叭
uchar code
table3[]={0x00,0x39,0x00,0x6A,0x00,0xA8,0x01,0x29,0x7E,0x2A,0xFC,0x28,0xFC,0x29,0xCC,0x2A,
0xCC,0x28,0xFC,0x29,0xFC,0x2A,0x7E,0x28,0x01,0x29,0x00,0xAA,0x00,0x68,0x00,0x38};
uchar code table4[]="LCD12864";
uchar code table5[]="自定义显示字符";
uchar code table6[]="大海橡树";

void delay(uint i)              //延时函数
{
                                while(--i);
}

void read_busy()                //读忙标志  最高位为 1 则 busy，不能进行读写操作
{
                                RS=0;
                                RW=1;
                                EN=1;       //看时序
                                while(P0 & 0x80);
                                EN=0;
}

void write_LCD_command(uchar value) //写命令函数
{
                                read_busy();      //每次读写都要进行读忙标志
                                RS=0;
                                RW=0;
                                EN=1;             //EN 从 1--0 锁存数据
                                P0=value;
                                delay(100);
                                EN=0;
}

void write_LCD_data(uchar value)        //写数据函数
{
                                read_busy();
                                RS=1;
                                RW=0;
                                EN=1;             //EN 从 1--0 锁存数据
                                P0=value;
                                delay(100);
                                EN=0;
```

```
}

void init_LCD()                    //8 位并口方式 LCD1864 初始化函数
{
                    delay(4000);              //等待时间>40ms
                    write_LCD_command(0x30);//功能设定：8 位数据、基本指令操作
                    delay(100);               //等待时间>100us
                    write_LCD_command(0x30);//功能设定：8 位数据、基本指令操作
                    delay(37);                //等待时间>37us
                    write_LCD_command(0x0C);//显示设定：整体显示、游标关、不反白
                    delay(100);               // 等待时间>100us
                    write_LCD_command(0x01);//清屏指令
                    delay(10000);             //等待时间>10ms
                    write_LCD_command(0x06);//进入点设定：地址指针加 1
}
/*==========================================
自定义函数：CGRAM 自定义显示字符对应地址及编码
地址                显示编码
0x40                0x0000
0x50                0x0002
0x60                0x0004
0x70                0x0006
==========================================*/
void CGRAM()
{
                    uchar i;
                    write_LCD_command(0x30);//基本指令操作
                    write_LCD_command(0x40);//设定 CGRAM 字符的位置
                    for(i=0;i<16;i++)
                    {
                     write_LCD_data(table[i*2]);
                     write_LCD_data(table[i*2+1]);//送显示数据到 CGRAM 区中
                    }
                    write_LCD_command(0x50);
                    for(i=0;i<16;i++)
                    {
                     write_LCD_data(zk[i*2]);
                     write_LCD_data(zk[i*2+1]);
                    }
                    write_LCD_command(0x60);
                    for(i=0;i<16;i++)
                    {
                     write_LCD_data(table2[i*2]);
                     write_LCD_data(table2[i*2+1]);
```

```
                    }
                    write_LCD_command(0x70);
                    for(i=0;i<16;i++)
                    {
                     write_LCD_data(table3[i*2]);
                     write_LCD_data(table3[i*2+1]);
                    }

}
void main()
{
                    uchar num;
                    init_LCD();
                    while(1)
                    {
                     write_LCD_command(0x80);
                     for(num=0;num<8;num++)
                     {
                            write_LCD_data(table6[num]);
                     }
                     write_LCD_command(0x84);
                     for(num=0;num<8;num++)
                     {
                            write_LCD_data(table4[num]);
                     }
                     write_LCD_command(0x90);
                     for(num=0;num<14;num++)
                     {
                            write_LCD_data(table5[num]);
                     }

                     CGRAM();
                     write_LCD_command(0x98);
                     write_LCD_data(0x00);
                     write_LCD_data(0x00);//第一个自定义显示字符编码为：0x0000

                     write_LCD_command(0x99);
                     write_LCD_data(0x00);
                     write_LCD_data(0x02);//第一个自定义显示字符编码为：0x0002

                     write_LCD_command(0x9A);
                     write_LCD_data(0x00);
                     write_LCD_data(0x04);//第一个自定义显示字符编码为：0x0004
```

```
                write_LCD_command(0x9B);
                write_LCD_data(0x00);
                write_LCD_data(0x06);//第一个自定义显示字符编码为：0x0006
                }
}
```

==========END==========

通过上面的学习，大家应该学会了怎么显示 ASCII 码、字库汉字和自定义显示了！下面介绍怎么显示图片。

四、显示图片

显示图片要用到它的扩充指令，里面有个绘图开关 G！当我们要显示图片时，要打开绘图开关 G。
还是上次的图片：取模、生成数据，显示效果如下图所示：

现在来说明一下显示图片的函数：

```
/*********************************************
函数：显示图片
说明：要先设定垂直地址再设定水平地址(连续写入两
个字节的资料，来完成垂直于水平的坐标地址)
0x80---0x87：显示上半部分
0x88---0x8F：显示下半部分
只需设定显示的第一个位置，指针会自动加 1
*********************************************/
void display_BMP(uchar *address)
{
uchar i,j;
for(i=0;i<32;i++)
{
    write_LCD_command(0x80+i);//先送垂直地址
```

```
    write_LCD_command(0x80);  //再送水平地址 ----显示图片的上半部分
    for(j=0;j<16;j++)
    {
        write_LCD_data(*address);
        address++;
     }
 }
for(i=0;i<32;i++)
{
    write_LCD_command(0x80+i);     //先送垂直地址
    write_LCD_command(0x88);       //显示图片的下半部分
    for(j=0;j<16;j++)
    {
       write_LCD_data(*address);
       address++;                  //指针地址指向下个位置
     }
}
}
```

还有一点要注意的就是显示图片和显示 ASCII 码、汉字的初始化函数不同，显示图片用的是扩展指令：

```
void init_BMP()
{
write_LCD_command(0x36);//CL=1--8 位。扩充指令(RE=1)，绘图打开(G=1)
delay(100);//适当延时
write_LCD_command(0x36);
delay(37);
write_LCD_command(0x3E); //8 位(CL=1)，扩充指令(RE=1)，绘图打开(G=1)
delay(100);
write_LCD_command(0x01);//清屏指令
delay(100);
}
```

这一点一定要注意，上面的延时函数可以不要，大家结合情况适当添加！显示图片的就这么多了，其他部分的代码和显示汉字的一样！这里就不多写了！

五、显示动画

上面也说了，不断的刷新一张张静态图片就可以显示动画了，上面的两个写函数：写数据函数和写命令函数，EN 从 1 变 0 延时应尽量短些，上面的 delay(100)；延时过长，大家可以做适当的修改！

还有一点要说明的是，清屏只是简单的清屏二不是清除显示的内容，我这么说吧，当你要显示很多张图片时，清屏显示后，只要 RAM 中的图片数据不变，显示还是清屏前显示的图片，要显示下一帧图片时就会变乱，解决办法是，将 0 送入显示的 RAM 区(0 写入到显示图片函数中)，也就是将一张空白图片送入到 RAM 区显示。这样就 OK 了，其它的步骤我在上面已经做了介绍！大家可以试一下！

六、显示正弦曲线

那么怎么才能显示一个正弦波形呢?我们可以这样想，如果我们可以控制 128*64 液晶上的每个像素的显示与关闭，那么就可以根据曲线的规律来显示一个正弦波形了，那么！怎么才能控制一个像素点呢？这就要用到画点函数了；

其 GDRAM 对应的显示关系如上图所示；首先我们要在液晶上打号坐标，根据 xy 坐标来确定像素点的具体位置，首先要确定列，也就是 Y 的大小，然后确定它的行，就是哪个字节的哪个位，也就 X 的大小了，点亮一个像素点就送 1，否则送 0，这个大家都应该知道吧，其他不画点的地方就送 0 就行了，可是按照我们的这个思路写下去，似乎不对，点亮的点数不止我们要求的一个！是什么原因呢？原来是对不点亮的点做填 0 操作了，造成对原来数据的破坏，解决的办法是先读出一个字节的数据，然后点亮我们要求的那个像素点，再将其余不做操作的像素点的数据送入到原理的位置，经过这样一整合，就可以只改变我们要求的那个像素点，而其余的点不发生变化!在根据正弦函数 sin(x)的对应关系就可以显示正弦曲线了！思路就是这个样子的。下面就是那个画点函数的代码：

```
/*************************************************************
画点函数说明：增加 LCD 读函数，目的是使不打点的地方数据保持
不变:方法是先读出不打点位置的数据，打完点后将读到的数据写
入原来的位置，只有这样才会显示打点的曲线。
坐标原点：屏的左上角(0,0);到右下端(127，63)；
*************************************************************/
void Draw_dots(uchar x,uchar y,uchar color)
{
                uchar ROW,xlabel,xlabel_bit;
                uchar Read_H,Read_L;            //读 LCD 中的数据
                write_LCD_command(0x34);        //扩充指令
                write_LCD_command(0x36);        //打开绘图指令
                xlabel=x>>4;                    //取 16*16 首地址
                xlabel_bit=x & 0x0F;          //计算该点在 16 位数据的第几位
                if(y<32)                        //如果是上半屏,上下半屏 y 都是 0--31
                {
                 ROW=y;
                }
                else                            //显示的是下半屏
                {
```

```
ROW=y-32;
xlabel+=8;                    //规定显示在下半屏
}
write_LCD_command(ROW+0x80); //送入垂直地址
write_LCD_command(xlabel+0x80);//再送入水平地址
ReadByte();       //读取当前 GDRAM 数据前腰进行一次空读，接下来就可以读出
```

数据了

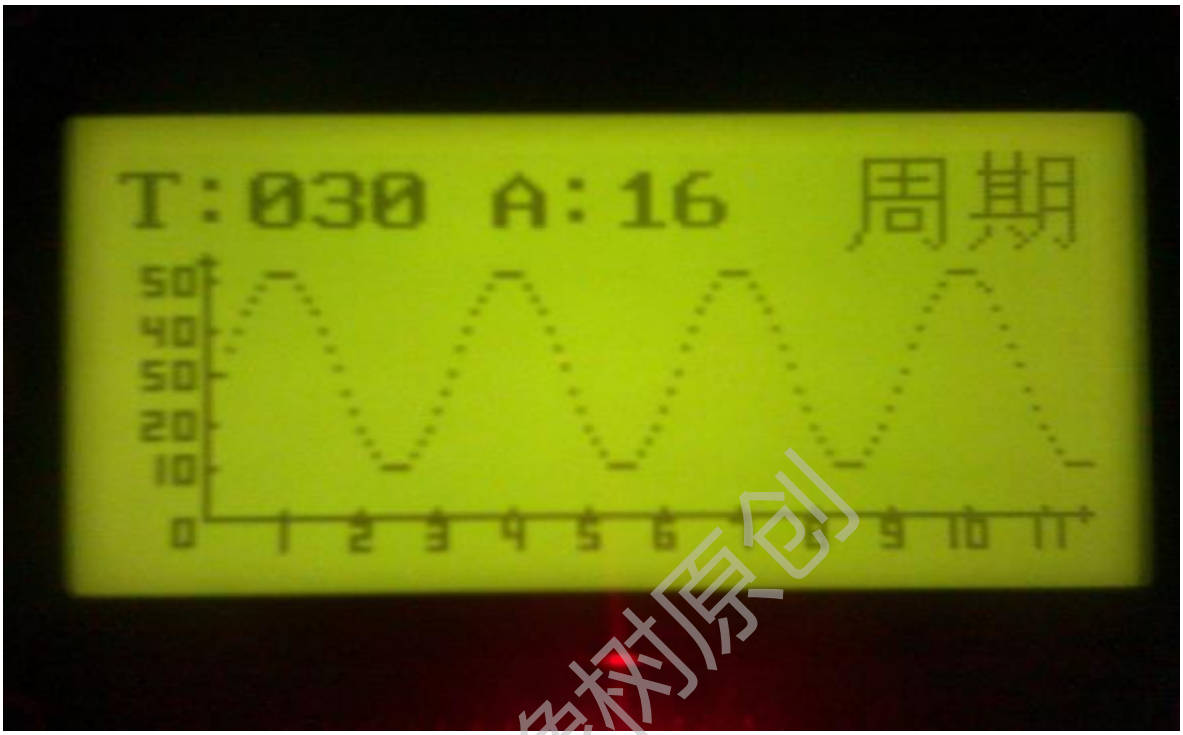
```
Read_H=ReadByte(); //读高 8 位
Read_L=ReadByte(); //读低 8 位

write_LCD_command(ROW+0x80);      //送入垂直地址
write_LCD_command(xlabel+0x80);     //再送入水平地址

if(xlabel_bit<8)
{
 switch(color)
 {
        case 0:Read_H &= (~(0x01<<(7-xlabel_bit)));//若变白
        break;
        case 1:Read_H |= (0x01<<(7-xlabel_bit));//若涂黑
        break;
        case 2:Read_H ^= (0x01<<(7-xlabel_bit));//若反转
        break;
        default:break;
 }
 write_LCD_data(Read_H);//将数据写入 GDRAM
 write_LCD_data(Read_L);//先写高位，再写低位(地址指针顺序)
 }
else
{
 switch(color)//color 设置
 {
        case 0: Read_L &= (~(0x01<<(15-xlabel_bit)));//若变白
        break;
        case 1: Read_L |= (0x01<<(15-xlabel_bit));//若涂黑
        break;
        case 2: Read_L ^= (0x01<<(15-xlabel_bit));//若反转
        break;
        default:break;
 }
 write_LCD_data(Read_H);
 write_LCD_data(Read_L);//写入数据
 }
```

write_LCD_command(0x30);//回到普通模式

}

其实写这个函数，主要还是思路，思路对了些代码就简单了！里面的正弦函数显示，可以包含 math.h 这个头文件中的正弦函数即可！

下面是我写个一个显示正弦函数的效果，通过按键可以调节幅度和频率！



下面这个是显示的温度曲线：

至此，LCD12864 全部内容就介绍到这里，里面难免有错误之处，还望大家批评改正，共同学习和交流。

在此欢迎大家访问我的百度空间

大海橡树百度空间