51 单片机综合学习

12864 液晶原理分析 1

辛勤学习了好几天，终于对 12864 液晶有了些初步了解～没有视频教程学起来真有些累，基本上　　程序写入顺序都是根据程序自我变动，然后逆向反推出原理……

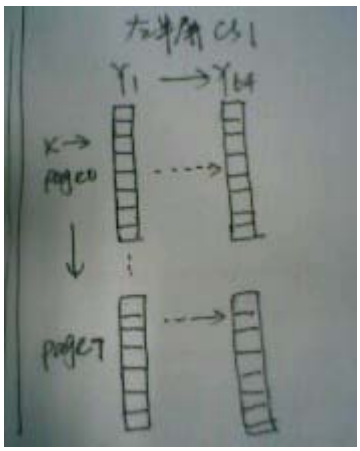
芯片：**YM12864R** P-1 控制芯片:**ST7920A**　**带中文字库**

初步小结：

1、 控制芯片不同，寄存器定义会不同
2、 显示方式有并行和串行，程序不同
3、 含字库芯片显示字符时不必对字符取模了
4、 对芯片的结构地址一定要理解清楚
5、 显示汉字时液晶芯片写入数据的顺序（即显示的顺序）要清楚
6、 显示图片时液晶芯片写入数据的顺序（即显示的顺序）要清楚
7、 显示汉字时的二级单元（一级为八位数据写入单元）要清楚
8、 显示图片时的二级单元（一级为八位数据写入单元）要清楚

12864 点阵液晶显示模块（LCM）就是由 128*64 个液晶显示点组成的一个 128 列*64 行的阵列。每个显示点对应一位二进制数，1 表示亮，0 表示灭。存储这些点阵信息的 RAM 称为显示数据存储器。要显示某个图形或汉字就是将相应的点阵信息写入到相应的存储单元中。图形或汉字的点阵信息由自己设计，问题的关键就是显示点在液晶屏上的位置（行和列）与其在存储器中的地址之间的关系。由于多数液晶显示模块的驱动电路是由一片行驱动器和两片列驱动器构成，所以 12864 液晶屏实际上是由左右两块独立的 64*64 液晶屏拼接而成，每半屏有一个 512*8 bits 显示数据 RAM。左右半屏驱动电路及存储器分别由片选信号 CS1 和 CS2 选择。显示点在 64*64 液晶屏上的位置由行号（line,0~63）与列号（column,0~63）确定。512*8 bits RAM 中某个存储单元的地址由页地址（Xpage,0~7）和列地址（Yaddress,0~63）确定。每个存储单元存储 8 个液晶点的显示信息。

为了使液晶点位置信息与存储地址的对应关系更直观关，将 64*64 液晶屏从上至下 8 等分为 8 个显示块，每块包括 8 行*64 列个点阵。每列中的 8 行点阵信息构成一个 8bits 二进制数，存储在一个存储单元中。（注意：二进制的高低有效位顺序与行号对应关系因不同商家而不同）存放一个显示块的 RAM 区称为存储页。即 64*64 液晶屏的点阵信息存储在 8 个存储页中，每页 64 个字节，每个字节存储一列(8 行)点阵信息。因此存储单元地址包括页地址（Xpage,0~7）和列地址（Yaddress,0~63）。例如点亮 128*64 的屏中（20，30）位置上的液晶点，因列地址 30 小于 64，该点在左半屏第 29 列，所以 CS1 有效；行地址 20 除以 8 取整得 2，取余得 4，该点在 RAM 中页地址为 2，在字节中的序号为 4；所以将二进制数据 00010000（也可能是 00001000，高低顺序取决于制造商）写入 Xpage=2，Yaddress=29 的存储单元中即点亮（20，30）上的液晶点。


芯片的结构一定要清楚！

点阵 LCD 的显示原理

在数字电路中，所有的数据都是以 0 和 1 保存的，对 LCD 控制器进行不同的数据操作，可以得到不同的结果。对于显示英文操作，由于英文字母种类很少，只需要 8 位（一字节）即可。而对于中文，常用却有 6000 以上，于是我们的 DOS 前辈想了一个办法，就是将 ASCII 表的高 128 个很少用到的数值以两个为一组来表示汉字，即汉字的内码。而剩下的低 128 位则留给英文字符使用，即英文的内码。

那么，得到了汉字的内码后，还仅是一组数字，那又如何在屏幕上去显示呢？这就涉及到文字的字模，字模虽然也是一组数字，但它的意义却与数字的意义有了根本的变化，它是用数字的各位信息来记载英文或汉字的形状，如英文的'A'在字模的记载方式如图 1 所示：

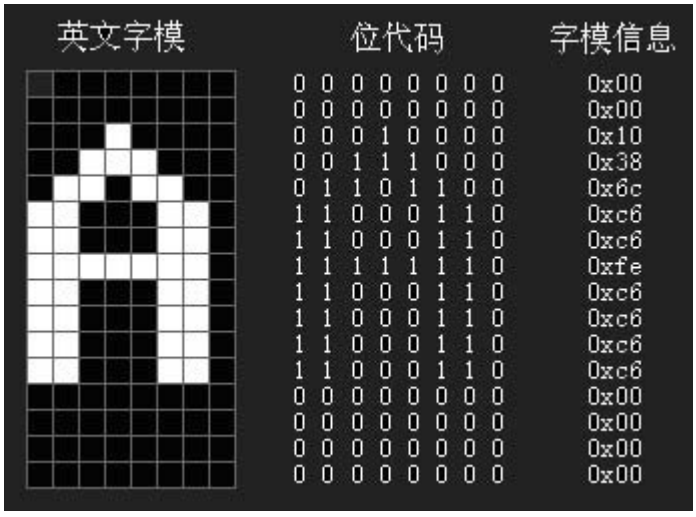


图 1 “A”字模图

而中文的“你”在字模中的记载却如图 2 所示：

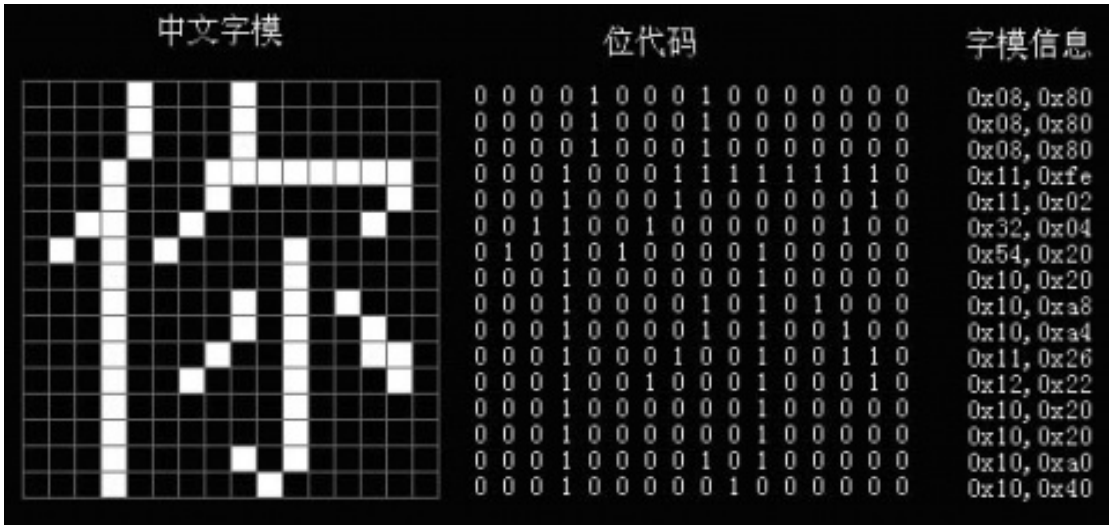


图 2 “你”字模图

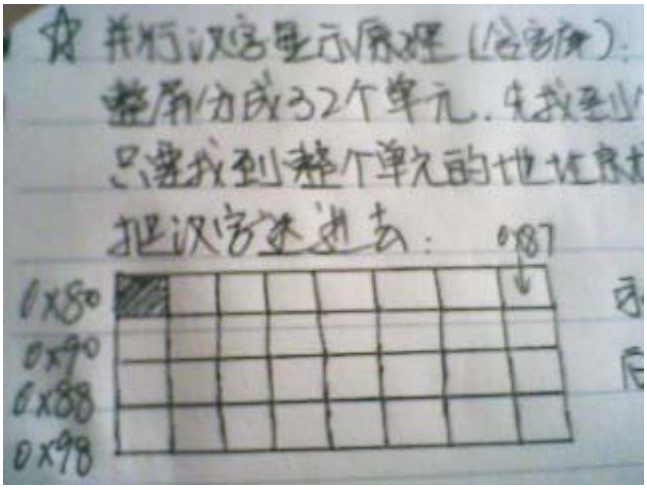


图 3

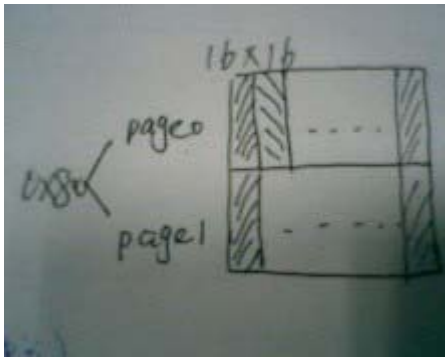


图 4

字符二级单元（图 3 中阴影部分）

汉字显示坐标

	X 坐标							
Line1	80H	81H	82H	83H	84H	85H	86H	87H
Line2	90H	91H	92H	93H	94H	95H	96H	97H
Line3	88H	89H	8AH	8BH	8CH	8DH	8EH	8FH
Line4	98H	99H	9AH	9BH	9CH	9DH	9EH	9FH

一个汉字的二级单元是一个 16*16 的区域，因些 128*64 液晶可以显示 4 行 8 列共 32 个汉字（如图 3）。而它的一个二级单元如图 4（在无字库时，对汉字的取模有横向跟纵向两种，要注意），对于并行含有字库芯片的显示，只要设定好这个二级单元的地址（如 0X80+i，这样设定 i 的范围为 0～31，这里注意第一行会直接跳到第三行；或者根据自己需要如第二行 0X90+i，i 范围为 0～7；第三行 0X88+i，i 范围为 0～7；），然后直接把汉字写入就 OK 了～（串行无字符库的后面再做分析）

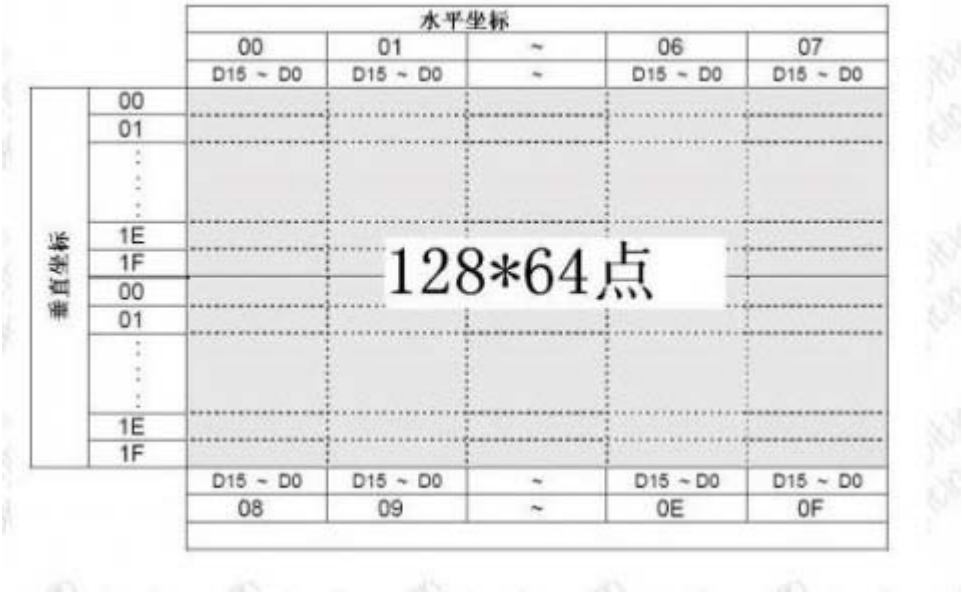


图 5：垂直坐标：上半屏 00～1F，总共为 32　　水平坐标：上半屏水平坐标分别为 0X80+（00～07）

下半屏 00～1F，总共为 32　　下半屏水平坐标分别为 **0X88+（00～07）**

图片显示芯片结构分块与汉字显示不一样

图象显示过程是这样的：首先设置垂直地址，再设水平地址（连续写入两个字节的资料来完成垂直与水平的坐标地址，然后在每个地址里写入 16 位数据）。　　垂直地址范围 AC5...AC0

水平地址范围 AC3...AC0

绘图 RAM 的地址计数器（AC）只会对水平地址(X 轴)自动加一,当水平地址=0FH 时会重新设为 00H

但并不会对垂直地址做进位自动加一，故当连续写入多笔资料时，程序需自行判断垂直地址是否需重新设定。GDRAM 的坐标地址与资料排列顺序如图 5：分上下屏写入。

```
for(i=0;i<32;i++)            // 上半屏 32 个垂直地址
{
write_com(0x80 + i);    //  垂直地址
write_com(0x80);        //  水平地址
for(j=0;j<16;j++)
 {
write_data(*adder);
adder++;
}
}
```

带中文字库的 128X64 显示模块时应注意以下几点：

①欲在某一个位置显示中文字符时，应先设定显示字符位置，即先设定显示地址，再写入中文字符编码。

②显示 ASCII 字符过程与显示中文字符过程相同。不过在显示连续字符时，只须设定一次显示地址，由模块自动对地址加 1 指向下一个字符位置，否则，显示的字符中将会有一个空 ASCII 字符位置。

③当字符编码为 2 字节时，应先写入高位字节，再写入低位字节。

④模块在接收指令前，向处理器必须先确认模块内部处于非忙状态，即读取 BF 标志时 BF 需为“0”，方可接受新的指令。如果在送出一个指令前不检查 BF 标志，则在前一个指令和这个指令中间必须延迟一段较长的时间，即等待前一个指令确定执行完成。指令执行的时间请参考指令表中的指令执行时间说明。⑤“RE”为基本指令集与扩充指令集的选择控制位。当变更“RE”后，以后的指令集将维持在最后的状态，除非再次变更“RE”位，否则使用相同指令集时，无需每次均重设“RE”位。

程 序 ———————— 并 行 （ 串 行 后 面 再 分 析）————————————————————

```
#include <stdio.h>
#include <reg52.h>
#include <intrins.h>
#include <string.h>

#define uchar unsigned char
#define uint unsigned int

uchar code LCD_data1[];
uchar code LCD_data2[];
uchar code LCD_picture1[];
uchar code LCD_picture2[];

sbit RS = P2^4;
sbit RW = P2^5;
sbit EN = P2^6;
sbit PSB = P2^1;
```

```
sbit RES = P2^3;
sbit Dataport = P0;
sbit Busyport = P0^7;
//////////////////////////////////////////////////////////////
void     delay_ms(unsigned int n)           //延时 10×n 毫秒程序
 {
         unsigned int i,j;
         for(i=0;i<n;i++)
          for(j=0;j<2000;j++);
 }
void     delay(unsigned int m)           //1US 延时程序
{
       unsigned int i,j;
       for(i=0;i<m;i++)
        for(j=0;j<10;j++);
}
//////////////////////////////////////////////////////////////
//判 LCM 忙子函数
void check_LCD_busy (void)
{
       Dataport = 0xff;
       RS = 0;
       RW = 1;
       EN = 1;
       while (Busyport);
       EN = 0;
}
//////////////////////////////////////////////////////////////
//写命令子函数
void write_com(uchar Command)
{
       check_LCD_busy();
       RW=0;
       RS=0;
       delay(1);
       P0=Command;
       EN=1;
       delay(1);
       EN=0;
}
//////////////////////////////////////////////////////////////
//写数据子函数
```

```
void write_data(uchar Data)
{
        check_LCD_busy();
        RW=0;
        RS=1;
        delay(1);
        P0=Data;
        EN=1;
        delay(1);
        EN=0;
}
//////////////////////////////////////////////////////////////
//LCM 清屏函数
void lcdClear (void)
{
        write_com(0x01);
}
//////////////////////////////////////////////////////////////
//LCM 复位函数
void reset ()
{
        RES=0;          //复位
        delay(1);         //延时
        RES=1;          //复位置高
        delay(10);
}
//////////////////////////////////////////////////////////////
//显示汉字
void dispString (uchar X, Y,uchar *msg)   //X 为哪一行，Y 为哪一列。msg
为汉字
{
        if(X==0)      X = 0x80;          //第一行，汉字显示坐标
        else if(X==1) X = 0x90;        //第二行
        else if(X==2) X = 0x88;        //第三行
        else       X = 0x98;             //第四行
        Y = X + Y;                      //Y 为 1 往右移一位
        write_com(Y);                    //写入坐标
 while (*msg)
 {
          write_data(*msg++); //显示汉字
 }
}
//////////////////////////////////////////////////////////////
//显示图象
```