**12864液晶深入学习笔记\_1——基于msp430g2553**

这是我对12864的学习笔记，12864液晶功能很全面，使用起来也很方便，能够满足一般的研究和工程应用的需求。

下面我就对几个方面讲述一下我对它学习过程中的一些体会。我会尽量全面的介绍，并且会主要针对一些特殊的应用和一些我认为要特别注意的地方进行较为详细的说明。而对于那些较为固定的，常用的方面会简略一些。

其 中的不管串行模式还是并行模式，对液晶的写指令，写数据，读操作等这些基本的操作都是一些比较固定的，基础的东西。我认为只要能够看懂，并且能够在不同的 处理器上移植就够了，没有必要非要自己对着手册时序图写出来。因为有人已经写好了，而且工作稳定，我们只要在它的基础上学会应用就可以了。现在是知识爆炸 时期，知识，信息迅速膨胀，我们要学会使用已有的成果，然后在这个基础上自己再进行开发应用的研究。我们不必一定要从底层开始把别人已经做的很成熟的东西 再做一遍，这样不但效率不高，而且我们一般人的精力也不允许。

好了，不说废话了，下面就开始介绍，当然，那些基础层面的东西我也会介绍的。

下面所涉及到的程序，是针对msp430g2553的，都是我已经调通的，可以直接应用。

一，12864的介绍

1，液晶显示模块是128×64 点阵的汉字图形型液晶显示模块，可显示汉字及图形，内置国标GB2312 码简体中文字库（16X16 点阵）、128 个字符（8X16 点阵）及64X256 点阵显示RAM（GDRAM）。可与CPU 直接接口，提供两种界面来连接微处理机：8-位并行及串行两种连接方式。具有多种功能：光标显示、画面移位、睡眠模式等。

2，常用的12864液晶内部都是使用ST7920控制器。

1），ST7920提供8位元，4位元及串行三种微处理器控制方式，大陆常用的是8位元和串行控制方式。

2），ST7920可以控制显示字母，数字符号，中文字型和自定义的图画。可以用来显示图形，演示动画，绘制曲线等。

3），字符显示RAM (DDRAM)

ST7920的字符显示RAM (DDRAM)最多可以控制16字元\*4行，LCD的显示范围为16字元\*2行。

这里要注意，其实ST7920的DDRAM每一行可以控制16个汉字的，共有4行。但是LCD的每行只能显示8个字符，为了显示观察的方便，在lcd制作的过程中，是将DDRAM的其中两行拆分开成四行，然后在lcd上显示，也即是DDRAM只用到了一半。

lcd的显示字符的坐标地址如下表：

汉字显示坐标

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

从上表不难看出，其中第一行和第三行是DDRAM中的同一行拆分来的，同理2 4 行也是DDRAM中的同一行拆分而来的。

了 解了这一点就不难理解下面程序中在换行显示时，要认为人为手动地指定下一行的地址。例如：如果第一行显示完了，下面的数据我要接着显示在第二行，这样才符合人 观察的习惯，那么我就要在换第二行显示之前要手动地把显示地址切换到第二行。要不然的话，第一行显示完了，地址会自动增加，就会显示到第三行上去，这样的 话我们观察起来就不自然了。程序实例会在下面涉及到的。

4），中文字库ROM (CGROM)

内置的是GB2312码简体中文字库，共提供了8192个16\*16点的中文字型。

5），半宽字型ROM (HCGROM)

提供126个16\*8点半字宽的字母符号字型。

6），图画显示RAM (GDRAM)

提供64\*256位元的GDRAM

其中我们常用的RAM就是上面所提到的，还有一些CGRAM ，IRAM这些我们不经常使用，就不再介绍了。

上面介绍的DDRAM控制显示汉字，字符。GDRAM控制显示图画，上电后，默认DDRAM是打开的，控制液晶显示。GDRAM默认不打开，它里面的数据是随机的，如果此时打开了GDRAM的话，lcd会同时受到DDRAM和GDRAM的控制，由于GDRAM中的数据是随机的，所以会显示乱码。所以在使用GDRAM之前要先清除里面的随机数据。

清除GDRAM的函数如下：

void Clear\_GDRAM(void) //清除GDRAM中的的随机数据。因为上电后GDRAM中的数据是随机的，如果不清除而直接打开GDRAM显示时，会显示乱码

//所以在局部使用GDRAM显示图形时，要先清除随机数据。如果是全局使用GDRAM，即整个lcd屏全部设置显示数据，则可以

//不必清除，因为新数据会把随机数据给覆盖掉

{

uchar i,j,k;

wr\_lcd(comm,0x34); //打开扩展指令集 操作GDRAM是扩展指令集

i = 0x80;

for(j = 0;j < 32;j++)

{

wr\_lcd(comm,i++);

wr\_lcd(comm,0x80);

for(k = 0;k < 16;k++)

{

wr\_lcd(dat,0x00); //写入空字符，就相当于清零

}

}

i = 0x80;

for(j = 0;j < 32;j++)

{

wr\_lcd(comm,i++);

wr\_lcd(dat,0x88);

for(k = 0;k < 16;k++)

{

wr\_lcd(dat,0x00);

}

}

wr\_lcd(comm,0x30); //回到基本指令集

}

3，12864有两种工作模式

1），并行模式和串行模式。并行模式就是常用的8位数据线，4为控制线。这种方式虽然占用的IO口较多，但是向液晶收发数据较容易实现，数据传输速度较快。所以在一些连续显示多幅图画，演示动画或对显示的实时性要求较高的场合应该考虑这种方式。其中在并行模式中，在向液晶写数据或命令前，要进行液晶忙标志判断 BF,要确定液晶显示不忙了，才能进行操作。

其中并行模式的液晶的读写数据，命令函数如下：

void Write\_Cmd(uchar cmd)

{

uchar lcdtemp = 0;

LCD\_RS\_L;

LCD\_RW\_H;

LCD\_DataIn; //数据输入单片机

do //判忙

{

LCD\_EN\_H;

\_NOP();

lcdtemp = LCD2MCU\_Data;

LCD\_EN\_L;

}

while(lcdtemp & 0x80); //判断忙标志 等待忙

LCD\_DataOut; //数据输出到lcd

LCD\_RW\_L;

MCU2LCD\_Data = cmd; //单片机向lcd输入命令

LCD\_EN\_H;

\_NOP();

LCD\_EN\_L;

}

void Write\_Data(uchar dat)

{

uchar lcdtemp = 0;

LCD\_RS\_L;

LCD\_RW\_H;

LCD\_DataIn;

do //判忙

{

LCD\_EN\_H;

\_NOP();

lcdtemp = LCD2MCU\_Data;

LCD\_EN\_L;

}

while(lcdtemp & 0x80); //等待忙

LCD\_DataOut;

LCD\_RS\_H;

LCD\_RW\_L;

MCU2LCD\_Data = dat; //单片机向lcd中输入数据

LCD\_EN\_H;

\_NOP();

LCD\_EN\_L;

}

2），串行模式只用到了两根线WR EN于单片机进行通信。这种方式可以大大减少单片机IO口的开销，适用于IO口资源有限的单片机（如msp430g2553）。但是这种方式实现起来较麻烦，数据的传输效率不高。对于一般的文字，简单图形的显示还是可以的。（有可能是因为msp430g2553的处理能力较强，我现在用串行连接方式，显示下面的几幅图画，显示效果很好，看不出有什么数据传输速度慢的问题）。

由于我用的是msp430g2553，所以我一直都是使用串行的控制模式。

//下面重点讲一下串行的时序

//SCLK:串行同步时钟线，每操作一位数据都要有一个SCLK跳变沿，而且在这里是上升沿有效。也即是说，每次SCLK由低电平变为高电平的瞬间，液晶控

//制器将SID上的数据读入或输出。

//SID:串行数据，每一次操作都由三个字节数据组成，第一个字节向控制器发送命令控制字，告诉控制器接下来是什么操作，若为写指令则发送11111000

//(0xf8)，若为若为写数据则发送11111010(0xfa)，若为读状态则发送11111100(0xfc)，若为读数据则发送11111110(0xfe)。

//第二个字节的高4位为发送指令或数据的高4位，第二个字节的低4位补0.

//第三个字节的高4位为发送指令或数据的低4位，第三个字节的低4位补0

//具体的可以观察时序进行理解

其中数据的传输的函数如下：

//12864串行连接写数据，写命令函数 按照手册上的时序进行编程  
void wr\_lcd(uchar dat\_comm,uchar content)//  
{ // 要写的数据  
 uchar a,i,j;  
 delay\_us(50);  
 a=content;  
 LCD\_SCLK0; //en=0;  
 LCD\_SID1; //wr=1  
 for(i=0;i<5;i++) //数据时序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8 前5个高电平的同步码  
 {  
 LCD\_SCLK1;  
 LCD\_SCLK0;  
 }  
 LCD\_SID0; //wr=0 写操作

LCD\_SCLK1; //en=1 来一个时钟  
 LCD\_SCLK0; //en=0  
 if(dat\_comm)  
 LCD\_SID1; //RS=1 写数据  
 else  
 LCD\_SID0; //RS=0 写指令

LCD\_SCLK1; //来一个时钟  
 LCD\_SCLK0;

  LCD\_SID0;  //控制字的最后一位为0

  LCD\_SCLK1;  //来一个时钟  
  LCD\_SCLK0;

   for(j=0;j<2;j++)//\*\*\*\*\*\*\*\*\*\*\*\*一共2\*4次循环写写一字节数据，第一次大循环写高4位，第二次大循环写低4位  
  {  
    for(i=0;i<4;i++)  
    {  
      if(a&0x80)  
           LCD\_SID1;  
        else  
           LCD\_SID0;  
      a=a<<1;  
      LCD\_SCLK1;  
      LCD\_SCLK0;  
    }  
      LCD\_SID0;  
   for(i=0;i<4;i++)  //时钟下面来4个时钟脉冲  
    {  
       LCD\_SCLK1;  
       LCD\_SCLK0;  
    }  
  }  
}

void Draw\_TX(uchar Yaddr,uchar Xaddr,const uchar \* dp)

{

uchar j;

uchar k=0;

// wr\_lcd(comm,0x01); //清屏,只能清除DDRAM

wr\_lcd(comm,0x34); //使用扩展指令集，关闭绘图显示 打开扩展指令集

for(j=0;j<16;j++)

{

wr\_lcd(comm,Yaddr++); //Y地址

wr\_lcd(comm,Xaddr); //X地址

wr\_lcd(dat,dp[k++]); //写入数据

wr\_lcd(dat,dp[k++]);

}

wr\_lcd(comm,0x36); //打开绘图显示

// wr\_lcd(comm,0x30); //回到基本指令集模式

}

5，向液晶全屏写图像的函数如下：

void Draw\_PM(const uchar \*ptr) //整屏显示图形

{

uchar i,j,k;

wr\_lcd(comm,0x34); //打开扩展指令集

i = 0x80;

for(j = 0;j < 32;j++)

{

wr\_lcd(comm,i++);

wr\_lcd(comm,0x80);

for(k = 0;k < 16;k++)

{

wr\_lcd(dat,\*ptr++); //先写入32\*16个数据

}

}

i = 0x80;

for(j = 0;j < 32;j++)

{

wr\_lcd(comm,i++);

wr\_lcd(comm,0x88);

for(k = 0;k < 16;k++)

{

wr\_lcd(dat,\*ptr++); //再写入32\*16个数据

}

}

wr\_lcd(comm,0x36); //打开绘图显示

wr\_lcd(comm,0x30); //回到基本指令集

}

6，下面我就贴上一个对液晶功能测试的函数，其中用到了液晶的很多功能，可以有实现现象观察得到，注释的也较详细，代码如下：

#include "msp430g2553.h"

#include "ser\_12864.h"

void main( void )

{

uint i;

uchar laba[]= //16\*16大小图形数据

{ 0x00,0x00,0x00,0xC0,0x01,0x48,0x02,0x44,0x04,0x52,0xF8,0x49,0x88,0x49,0x88,0x49,

0x88,0x49,0x88,0x49,0xF8,0x49,0x04,0x52,0x02,0x44,0x01,0x48,0x00,0xC0,0x00,0x00};

WDTCTL = WDTPW + WDTHOLD; //关狗

BCSCTL1 = CALBC1\_12MHZ; //设定cpu时钟DCO频率为12MHz

DCOCTL = CALDCO\_12MHZ;

P2DIR |=BIT5+BIT4; //液晶的两条线

init\_lcd(); //初始化液晶

//下面是显示液晶字符表中的字符

wr\_lcd(comm,0x80); //写第一行的显示地址 写命令0x80

for(i = 0; i < 16; i++) //每一行可以显示16个字符

wr\_lcd(dat,0x00 + i); //显示0x00~0x0f对应的字符 写数据

wr\_lcd(comm,0x90); //写第二行的显示地址 因为12864的1 3行和2 4行分别由控制器ST7920的两行切割来的，为了看起来方便

//这里要手动把地址切换到第二行。要不然的话，显示完了第一行，液晶会自动切换显示到第三行上去

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x10 + i); //显示0x10~0x1f对应的字符

wr\_lcd(comm,0x88); //写第三行的显示地址

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x20 + i); //显示0x20~0x2f对应的字符

wr\_lcd(comm,0x98); //写第四行的显示地址

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x30 + i); //显示0x30~0x3f对应的字符

// delay\_ms(1000); //延迟1s，观察效果

//调试时，可以在此处设置断点，单步执行下面的指令，观察结果

//在显示时DDAM和GDRAM是同时显示的，也就是它们的显示结果是叠加在一起的

//1.设定DDRAM地址命令

wr\_lcd(comm,0x90); //设定DDRAM地址，因为此时DDRAM地址已经溢出

//2.显示状态命令

wr\_lcd(comm,0x08); //整体显示关，游标关，游标位置关

wr\_lcd(comm,0x0c); //整体显示开，游标关，游标位置关

wr\_lcd(comm,0x0e); //整体显示开，游标开，游标位置关

wr\_lcd(comm,0x0f); //整体显示开，游标开，游标位置开 游标闪烁

//3.位址归位

wr\_lcd(comm,0x02); //位址归位，游标回到原点

wr\_lcd(comm,0x84); //将DDRAM地址设为0x88，游标在此闪烁

//4.点设定指令

//（以下四个命令是控制写入字符以后光标及整屏显示的移动）

wr\_lcd(comm,0x07); //光标右移整体显示左移

wr\_lcd(comm,0x20); //写入两个空格

wr\_lcd(dat,0x20);

wr\_lcd(comm,0x05); //光标左移整体显示右移

wr\_lcd(dat,0x20); //写入两个空格

wr\_lcd(dat,0x20);

wr\_lcd(comm,0x06); //光标右移整体显示不移动

wr\_lcd(dat,0x20); //写入两个空格

wr\_lcd(dat,0x20);

wr\_lcd(comm,0x04); //光标左移整体显示不移动

wr\_lcd(dat,0x20); //写入两个空格

wr\_lcd(dat,0x20);

//5.游标和显示移位控制

//（以下四个命令无需写入显示数据，直接控制光标和整屏显示的移动，上面执行的命令是写入空格实现光标移动和整屏的移动）

wr\_lcd(comm,0x10); //光标左移

wr\_lcd(comm,0x14); //光标右移

wr\_lcd(comm,0x18); //整体显示左移，光标跟随

wr\_lcd(comm,0x1c); //整体显示右移，光标跟随

wr\_lcd(comm,0x0c); //关闭光标

//6.进入扩展功能模式命令

wr\_lcd(comm,0x34); //打开扩展功能模式，绘图显示关闭

//7.反白命令

wr\_lcd(comm,0x04); //同时反白1、3行

wr\_lcd(comm,0x04); //再次反白1、3行，相当于关闭1、3行反白

wr\_lcd(comm,0x05); //同时反白2、4行

wr\_lcd(comm,0x05); //再次反白2、4行，相当于关闭2、4行反白

//8.睡眠模式命令

wr\_lcd(comm,0x08); //进入睡眠模式 液晶的控制器ST7920关闭，降低功耗。但是背光还是亮的

wr\_lcd(comm,0x0c); //退出睡眠模式 继续显示

//9.待命模式命令

wr\_lcd(comm,0x01); //进入待命模式 也是不显示内容的，液晶已经准备好接受数据或命令了

//10.打开GDRAM显示 //这样打开GDRAM显示，液晶的GDRAM和DDRAM会同时控制液晶的显示

wr\_lcd(comm,0x36); //打开扩展功能模式，打开绘图显示 由于上电后GDRAM中的数据是随机的，所以如果在显示之前不清除的话，会显示乱码

Draw\_TX(0x80,0x84,laba); //显示16\*16大小图形 显示上面定义的图像，是一个小喇叭

Clear\_GDRAM(); //清除上电复位后GDRAM中的随机数值，此时GDRAM显示空字符，但是DDRAM中还是有数据，所以此时会显示DDRAM中的数据

Draw\_TX(0x80,0x84,laba); //重新显示设置16\*16大小图形

//11.关闭GDRAM显示

wr\_lcd(comm,0x34); //打开扩展功能模式，关闭绘图显示 显示DDRAM中的数据

//12.设定基本指令集

wr\_lcd(comm,0x30); //回到基本指令集

//13.清除显示命令

wr\_lcd(comm,0x01); //清屏 只能清除DDRAM 此时液晶就什么都不显示了

//下面是显示液晶字符表中的字符

wr\_lcd(comm,0x80); //写第一行的显示地址

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x40 + i); //显示0x40~0x4f对应的字符

wr\_lcd(comm,0x90); //写第二行的显示地址

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x50 + i); //显示0x50~0x5f对应的字符

wr\_lcd(comm,0x88); //写第三行的显示地址

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x60 + i); //显示0x60~0x6f对应的字符

wr\_lcd(comm,0x98); //写第二行的显示地址

for(i = 0; i < 16; i++)

wr\_lcd(dat,0x70 + i); //显示0x70~0x7f对应的字符

LPM4;

}