« 博客园首页

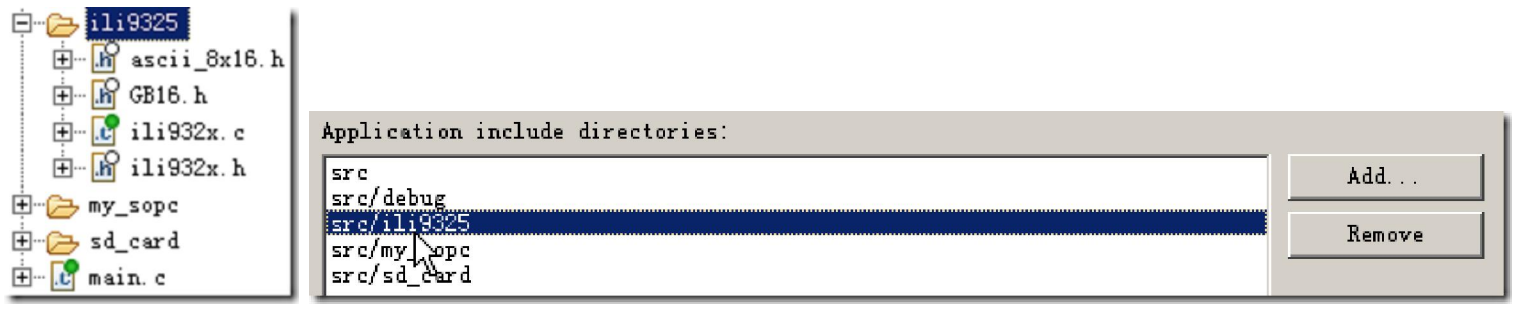
作者：.COM 缺氧® 来源：博客园 发布时间：2010-12-27 21:23 阅读：474 次 原文链接 [收藏]

准备资料

[整理].ILI9325 TFT驱动中文资料

编写驱动程序

步骤1 将ili9325的文件夹加入APP路径



步骤2 编写驱动文件

时间比较紧张，我就贴出来代码，挑重点的讲几句。

代码2.1 ili932x.h

```
01 #ifndef ILI932X_H_
02 #define ILI932X_H_
03 //
04 #include "my_types.h"
05 #include "my_regs.h"
06 //
07 #define White          0xFFFF
08 #define Black          0x0000
09 #define Blue           0x001F
10 #define Blue2          0x051F
11 #define Red            0xF800
12 #define Magenta        0xF81F
13 #define Green          0x07E0
14 #define Cyan           0x7FFF
15 #define Yellow         0xFFE0
16 //
17 #define ID_AM    110
18 //
19 #define DB_o_EN  ili_DB->DIRECTION=0xFF
20 #define DB_i_EN  ili_DB->DIRECTION=0x00
21 //
22 void ili_WrDB_2x8b(u8 DH, u8 DL);
23 void ili_WrCmd(u8 DH, u8 DL);
24 void ili_WrData(u8 DH, u8 DL);
25 void ili_WrReg(u8 cmd, u16 data);
26 void ili_WrDB_16b(u16 data);
27 //
28 void ili_DelayMs(u32 n);
29 void ili_Initial(void);
30 void ili_SetCursor(u8 x, u16 y);
31 void ili_SetDispArea(u16 x0, u16 y0, u8 xLength, u16 yLength, u16 xOffset, u16 yOffset);
32 void ili_ClearScreen(u32 bColor);
```

```
33 //
34 void ili_PlotPoint(u8 x, u16 y, u16 color);
35 void ili_PlotPixel(u8 x, u16 y, u16 color);
36 void ili_PlotBigPoint(u8 x, u16 y, u16 color);
37 //
38 void ili_PutAscii_8x16(u16 x, u16 y, uc8 c, u32 fColor, u32 bColor);
39 void ili_PutGb_16x16(u16 x, u16 y, uc8 c[2], u32 fColor, u32 bColor);
40 void ili_PutString(u16 x, u16 y, uc8 *s, u32 fColor, u32 bColor);
41 //
42 void ili_DispColorBar(void);
43 //
44 #endif /* ILI932X_H_ */
```

注意第19~20行，定义两个宏来操纵8位DB双向总线的方向。

代码2.2 ili_932x.c

```
001 #include "ili932x.h"
002 #include "unistd.h"       // usleep()
003 #include "ascii_8x16.h"   // ascii码字库
004 #include "GB16.h"         // 汉字字库
005
006
007 // 8位总线模式，通过两次写操作写入高8位和低8位
008 void ili_WrDB_2x8b(u8 DH, u8 DL)
009 {
010   ili_DB->DATA=DH;
011   ili_nWR=0;
012     ili_nWR=1;
013     ili_DB->DATA=DL;
014   ili_nWR=0;
015     ili_nWR=1;
016 }
017
018
019 // 写命令
020 void ili_WrCmd(u8 DH, u8 DL)
021 {
022   ili_RS=0;
023   ili_WrDB_2x8b(DH, DL);
024 }
025
026
027 // 写数据
028 void ili_WrData(u8 DH, u8 DL)
029 {
030   ili_RS=1;
031   ili_WrDB_2x8b(DH, DL);
032 }
033
034
035 // 向DB总线写数据
036 void ili_WrDB_16b(u16 data)
037 {
038   ili_WrData(data>>8, data);
039 }
040
```

```
041
042 // 写寄存器
043 void ili_WrReg(u8 cmd, u16 data)
044 {
045   ili_WrCmd(0x00, cmd);
046   ili_WrDB_16b(data);
047 }
048
049
050 // 延时ms
051 void ili_DelayMs(u32 n)
052 {
053   usleep(n*1000);
054 }
055
056
057 // ILI93525初始化
058 void ili_Initial(void)
059 {
060   // 硬件复位
061   ili_nRST=0;
062   ili_DelayMs(1);
063   ili_nRST=1;
064   // 打开片选，输出使能
065   ili_nCS=0;
066   DB_o_EN;
067   //
068   ili_WrReg(0xE3, 0x3008);
069   ili_WrReg(0xE7, 0x0012);
070   ili_WrReg(0xEF, 0x1231); // Set the internal timing
071   ili_WrReg(0x01, 0x0000); // Set SS and SM bit
072   ili_WrReg(0x02, 0x0700); // Set 1 line inversion
073   // 屏幕旋转控制
074 #if   ID_AM==000
075   ili_WrReg(0x03, 0x1000); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=00, AM=0
076 #elif ID_AM==001
077   ili_WrReg(0x03, 0x1008); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=00, AM=0
078 #elif ID_AM==010
079   ili_WrReg(0x03, 0x1010); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=01, AM=0
080 #elif ID_AM==011
081   ili_WrReg(0x03, 0x1018); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=01, AM=1
082 #elif ID_AM==100
083   ili_WrReg(0x03, 0x1020); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=10, AM=0
084 #elif ID_AM==101
085   ili_WrReg(0x03, 0x1028); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=10, AM=1
086 #elif ID_AM==110
087   ili_WrReg(0x03, 0x1030); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=11, AM=0
088 #elif ID_AM==111
089   ili_WrReg(0x03, 0x1038); // TRI=0, DFM=x, BGR=0, HWM=0, ORG=0, I/D[1:0]=11, AM=1
090 #endif
091   ili_WrReg(0x04, 0x0000); // Resize register
092   ili_WrReg(0x08, 0x0404); // Set the back porch and front porch
093   ili_WrReg(0x09, 0x0000); // Set non-display area refresh cycle ISC[3:0]
094   ili_WrReg(0x0A, 0x0000); // FMARK function
095   ili_WrReg(0x0C, 0x0000); // RGB interface setting
096   ili_WrReg(0x0D, 0x0000); // Frame marker Position
```

```
  ili_WrReg(0x0F, 0x0000); // RGB interface polarity
  // Power on sequence VGHVGL
  ili_WrReg(0x10, 0x0000); // SAP, BT[3:0], AP, DSTB, SLP, STB
  ili_WrReg(0x11, 0x0007); // DC1[2:0], DC0[2:0], VC[2:0]
  ili_WrReg(0x12, 0x0000); // VREG1OUT voltage
  ili_WrReg(0x13, 0x1300); // VDV[4:0] for VCOM amplitude
  ili_WrReg(0x07, 0x0001);
  ili_DelayMs(50);         // Dis-charge capacitor power voltage
  ili_WrReg(0x10, 0x1590); // SAP, BT[3:0], AP, DSTB, SLP, STB
  ili_WrReg(0x11, 0x0227); // DC1[2:0], DC0[2:0], VC[2:0]
  ili_DelayMs(50);
  ili_WrReg(0x12, 0x001E); // Internal reference voltage= Vci;
  ili_DelayMs(50);
  ili_WrReg(0x13, 0x1500); // Set VDV[4:0] for VCOM amplitude
  ili_WrReg(0x29, 0x0026); // Set VCM[5:0] for VCOMH
  ili_WrReg(0x2B, 0x000F); // Set Frame Rate
  ili_DelayMs(50);
  ili_WrReg(0x20, 0x0000);
  ili_WrReg(0x21, 0x013f);
  // GRAM起始位置
#if   ID_AM==000
  ili_WrReg(0x20, 0x00EF);
  ili_WrReg(0x21, 0x013F);
#elif ID_AM==001
  ili_WrReg(0x20, 0x00EF);
  ili_WrReg(0x21, 0x013F);
#elif ID_AM==010
  ili_WrReg(0x20, 0x0000);
  ili_WrReg(0x21, 0x013F);
#elif ID_AM==011
  ili_WrReg(0x20, 0x0000);
  ili_WrReg(0x21, 0x013F);
#elif ID_AM==100
  ili_WrReg(0x20, 0x00EF);
  ili_WrReg(0x21, 0x0000);
#elif ID_AM==101
  ili_WrReg(0x20, 0x00EF);
  ili_WrReg(0x21, 0x0000);
#elif ID_AM==110
  ili_WrReg(0x20, 0x0000);
  ili_WrReg(0x21, 0x0000);
#elif ID_AM==111
  ili_WrReg(0x20, 0x0000);
  ili_WrReg(0x21, 0x0000);
#endif
  // Adjust the Gamma Curve
  ili_WrReg(0x30, 0x0007);
  ili_WrReg(0x31, 0x0007);
  ili_WrReg(0x32, 0x0107);
  ili_WrReg(0x35, 0x0206);
  ili_WrReg(0x36, 0x0406);
  ili_WrReg(0x37, 0x0101);
  ili_WrReg(0x38, 0x0101);
  ili_WrReg(0x39, 0x0207);
  ili_WrReg(0x3C, 0x0504);
  ili_WrReg(0x3D, 0x0806);
```

```
153   // Set GRAM area
154   ili_WrReg(0x50, 0x0000); // Horizontal GRAM Start Address
155   ili_WrReg(0x51, 0x00EF); // Horizontal GRAM End Address
156   ili_WrReg(0x52, 0x0000); // Vertical GRAM Start Address
157   ili_WrReg(0x53, 0x013F); // Vertical GRAM Start Address
158   ili_WrReg(0x60, 0x2700); // Gate Scan Line
159   ili_WrReg(0x61, 0x0001); // NDL,VLE, REV
160   ili_WrReg(0x6A, 0x0000); // Set scrolling line
161   // Partial Display Control
162   ili_WrReg(0x80, 0x0000);
163   ili_WrReg(0x81, 0x0000);
164   ili_WrReg(0x82, 0x0000);
165   ili_WrReg(0x83, 0x0000);
166   ili_WrReg(0x84, 0x0000);
167   ili_WrReg(0x85, 0x0000);
168   // Panel Control
169   ili_WrReg(0x90, 0x0010);
170   ili_WrReg(0x92, 0x0600);
171   ili_WrReg(0x93, 0x0003);
172   ili_WrReg(0x95, 0x0110);
173   ili_WrReg(0x97, 0x0000);
174   ili_WrReg(0x98, 0x0000);
175   ili_WrReg(0x07, 0x0173); // 262K color and display ON
176   // 关闭片选
177   ili_nCS=1;
178 }
179
180
181 // 设定Cursor
182 void ili_SetCursor(u8 x, u16 y)
183 {
184   ili_WrReg(0x20, x);
185   ili_WrReg(0x21, y);
186 }
187
188
189 // 设定显示区域
190 void ili_SetDispArea(u16 x0, u16 y0, u8 xLength, u16 yLength, u16 xOffset, u16 yOffset)
191 {
192 #if ID_AM==000
193   ili_SetCursor(x0+xLength-1+xOffset, y0+yLength-1+yOffset);
194 #elif ID_AM==001
195   ili_SetCursor(x0+xLength-1+xOffset, y0+yLength-1+yOffset);
196 #elif ID_AM==010
197   ili_SetCursor(x0+xOffset, y0+yLength-1+yOffset);
198 #elif ID_AM==011
199   ili_SetCursor(x0+xOffset, y0+yLength-1+yOffset);
200 #elif ID_AM==100
201   ili_SetCursor(x0+xLength-1+xOffset, y0+yOffset);
202 #elif ID_AM==101
203   ili_SetCursor(x0+xLength-1+xOffset, y0+yOffset);
204 #elif ID_AM==110
205   ili_SetCursor(x0+xOffset, y0+yOffset);
206 #elif ID_AM==111
207   ili_SetCursor(x0+xOffset, y0+yOffset);
208 #endif
```

```
  ili_WrReg(0x50, x0+xOffset);              // 水平 GRAM起始位置
  ili_WrReg(0x51, x0+xLength-1+xOffset);    // 水平GRAM终止位置
  ili_WrReg(0x52, y0+yOffset);              // 垂直GRAM起始位置
  ili_WrReg(0x53, y0+yLength-1+yOffset);    // 垂直GRAM终止位置
  ili_WrCmd(0x00, 0x22);
}


// 清屏
void ili_ClearScreen(u32 bColor)
{
  u32 i;

  ili_nCS=0;
  DB_o_EN;

  ili_SetDispArea(0, 0, 240, 320, 0, 0);
  for (i=0; i<76800; i++) ili_WrDB_16b(bColor);

  ili_nCS=1;
}


// 画点（单次操作）
void ili_PlotPoint(u8 x, u16 y, u16 color)
{
  ili_nCS=0;
  DB_o_EN;

  ili_SetCursor(x, y);
  ili_WrCmd(0x00, 0x22);
  ili_WrDB_16b(color);

  ili_nCS=1;
}


// 画点（连续操作的一部分）
void ili_PlotPixel(u8 x, u16 y, u16 color)
{
  ili_SetCursor(x, y);
  ili_WrCmd(0x00, 0x22);
  ili_WrDB_16b(color);
}


// 画一个大点
void ili_PlotBigPoint(u8 x, u16 y, u16 color)
{
  u8 i, j;
  ili_nCS=0;
  DB_o_EN;
  for(i=0; i<3; i++)
  {
    for(j=0; j<3; j++) ili_PlotPixel(x+i, y+j, color);
  }
```

```
  ili_nCS=1;
}


// 打印ASCII码（8x16）
void ili_PutAscii_8x16(u16 x, u16 y, uc8 c, u32 fColor, u32 bColor)
{
  u32 i, j;
  u8 temp;

  ili_nCS=0;
  DB_o_EN;

  ili_SetDispArea(x, y, 8, 16, 0, 0);
  for(i=0; i<16; i++)
  {
    temp = ascii_8x16_tab[c*16+i];
    for(j=0; j<8; j++)
    {
      if((temp&0x80) == 0x80)
        ili_WrDB_16b(fColor);
      else
        ili_WrDB_16b(bColor);
      temp <<= 1;
    }
  }

  ili_nCS=1;
}


// 打印汉字（16x16）
void ili_PutGB16(u16 x, u16 y, uc8 c[2], u32 fColor, u32 bColor)
{
  u32 i, j, k;
  u16 temp;

  ili_nCS=0;
  DB_o_EN;

  ili_SetDispArea(x, y, 16, 16, 0, 0);
  for(k=0; k<64; k++) // 64表示自建汉字库中的个数，循环查询内码
  {
    if ( (GB16[k].Index[0]==c[0])
      && (GB16[k].Index[1]==c[1]) )
    {
      for(i=0; i<32; i++)
      {
        temp = GB16[k].Msk[i];
        for(j=0; j<8; j++)
        {
          if((temp&0x80)==0x80)
            ili_WrDB_16b(fColor);
          else
            ili_WrDB_16b(bColor);
          temp <<= 1;
```

```
321         }
322       }
323     }
324   }
325
326   ili_nCS=1;
327 }
328
329
330 // 打印字符串
331 void ili_PutString(u16 x, u16 y, uc8 *s, u32 fColor, u32 bColor)
332 {
333   u8 l=0;
334   while(*s != '\0')
335   {
336     if(*s < 0x80)
337     {
338       ili_PutAscii_8x16(x+l*8, y, *s, fColor, bColor);
339       s++;
340       l++;
341     }
342     else
343     {
344       ili_PutGB16(x+l*8, y, (u8*)s, fColor, bColor);
345       s+=2;
346       l+=2;
347     }
348   }
349 }
350
351
352 // 彩条测试
353 void ili_DispColorBar(void)
354 {
355   u16 V, H;
356
357   ili_nCS=0;
358   DB_o_EN;
359
360   ili_SetDispArea(0, 0, 240, 320, 0, 0);
361   for(H=0; H<240; H++)
362   {
363     for(V=0; V<40; V++) ili_WrDB_16b(White);
364   }
365   for(H=0;H<240;H++)
366   {
367     for(V=40; V<80; V++) ili_WrDB_16b(Black);
368   }
369   for(H=0;H<240;H++)
370   {
371     for(V=80; V<120; V++) ili_WrDB_16b(Blue);
372   }
373   for(H=0;H<240;H++)
374   {
375     for(V=120; V<160; V++) ili_WrDB_16b(Red);
376   }
```

```
377   for(H=0;H<240;H++)
378   {
379     for(V=160; V<200; V++) ili_WrDB_16b(Magenta);
380   }
381   for(H=0;H<240;H++)
382   {
383     for(V=200; V<240; V++) ili_WrDB_16b(Green);
384   }
385   for(H=0;H<240;H++)
386   {
387     for(V=240; V<280; V++) ili_WrDB_16b(Cyan);
388   }
389   for(H=0;H<240;H++)
390   {
391     for(V=280;V<320;V++) ili_WrDB_16b(Yellow);
392   }
393
394   ili_nCS=1;
395 }
```

注意几个地方：

1. 初始化函数内的void ili_Initial(void)的硬件复位，nRST一定要拉低足够长时间再拉高，此处取1ms，否则会出现白屏现象。

```
1 ili_nRST=0;
2 ili_DelayMs(1);
3 ili_nRST=1;
```

2. 为了减少DB双向总线的方向切换次数及打开关闭nCS片选的此处，每次操作中只设定一次DB方向，且只打开关闭片选一次。比方在初始化函数内的void ili_Initial(void)内。

```
1 // 打开片选，输出使能
2 ili_nCS=0;
3 DB_o_EN;
```

```
1 // 关闭片选
2 ili_nCS=1;
```

因此，特别需要注意，写一组或一个寄存器前后，或写一个或一组数据到DB总线前后，只做一次设定DB方向和打开关闭片选动作。此外，千万不要将nCS永远拉低，浪费功率，最好用的时候打开，不用的时候关断。

3. 连续重复某个动作的时候，也执行第2条。比方说第246~266行，ili_PlotPixel()函数的引用。

步骤3 测试ILI9325驱动

代码3.1 main.c

```
01 #include <stdio.h>                    // printf()
02 #include <unistd.h>                   // usleep()
03 #include "my_types.h"                 // 数据类型
04 #include "debug.h"                    // debug
05 #include "sd_card.h"                  // sd卡
06 #include "ili932x.h"                  // ili9325
07
08
09 //#define ENABLE_APP_DEBUG // turn on debug message
10 #ifdef ENABLE_APP_DEBUG
11     #define APP_DEBUG(x)    DEBUG(x)
12 #else
13     #define APP_DEBUG(x)
14 #endif
15
```

```
16
17  int main(void)
18  {
19    ili_Initial();            // 初始化ILI9325
20    ili_DispColorBar();       // 彩条测试
21    while(1);
22    return 0;
23  }
```

测试效果如下（50￥的摄像头拍的，凑活看吧）。

源码下载

lcd_at_nios_nii_part.zip

目录

1 [原创][连载].基于SOPC的简易数码相框 - Quartus II部分（硬件部分）

2 [原创][连载].基于SOPC的简易数码相框 - Nios II SBTE部分（软件部分）- 配置工作

3 [原创][连载].基于SOPC的简易数码相框 - Nios II SBTE部分（软件部分）- SD卡（SPI模式）驱动

4 [原创][连载].基于SOPC的简易数码相框 - Nios II SBTE部分（软件部分）- TFT-LCD（控制器为ILI9325）驱动

5 [原创][连载].基于SOPC的简易数码相框 - Nios II SBTE部分（软件部分）- 从SD卡内读取图片文件，然后显示在TFT-LCD上

6 [原创][连载].基于SOPC的简易数码相框 - Nios II SBTE部分（软件部分）- 优化工作

7 [原创][连载].基于SOPC的简易数码相框 - Nios II SBTE部分（软件部分）- ADS7843触摸屏驱动测试

》点击查看原文...