AN1103 内部 FLASH 图片显示

本应用文档（AN1103）将向大家展示,如何将直接存放在 STM32 内部 FLASH 的图片数据（解码后的数据）显示到 LCD 上.以及介绍图片数据生成工具 image2lcd V2.9 的一些用法。本实验以 ALIENTEK MiniSTM32 开发板为实验对象。

本文档分为如下几部分：

1， 图片显示原理。

2， Image2lcd 简介。

3， 软件实现。

一， 图片显示原理。

在 LCD 上显示图片无非就是画点。原理就这么简单。画点需要 2 个要素：1，坐标。2，颜色。知道了这两个要素，画图就简单了，一副图片在 LCD 上显示出来，我们只需要在正确的位置写入正确的颜色即可。就像你用铅笔在本子上画图一样，本子就是 LCD，铅笔就是你的画点函数。

图片显示另外一个重要的特点就是他的数据量很大，比如画一幅 320*240 的图像，以 16 位色计算，那么光颜色的数据量就有：320*240*2=153600 字节。这其中还不包括设置坐标的过程，如果加上坐标设置，数据量就是颜色数据量的 5 倍（每次坐标设置需要发送 5 次命令/数据）以上。所以尽量优化画点过程，才能使你的图片显示得流畅。

单纯的画点，显然无法做太多优化，因为坐标设置是必须的。幸好，我们 ALIENTEK 所使用的液晶都是支持开窗显示以及坐标自增显示的。这样，我们只需要设置一次窗口，然后设置一次坐标，就可以不停的往 LCD 写颜色数据，而不需要再做地址设置了。这样可以使得速度比单纯的画点显示要快至少 5 倍以上。

开窗也有几个条件：1，窗大小。2，GRAM 自增方向（就是扫描方向）。

开窗的概念：如图 1 所示：

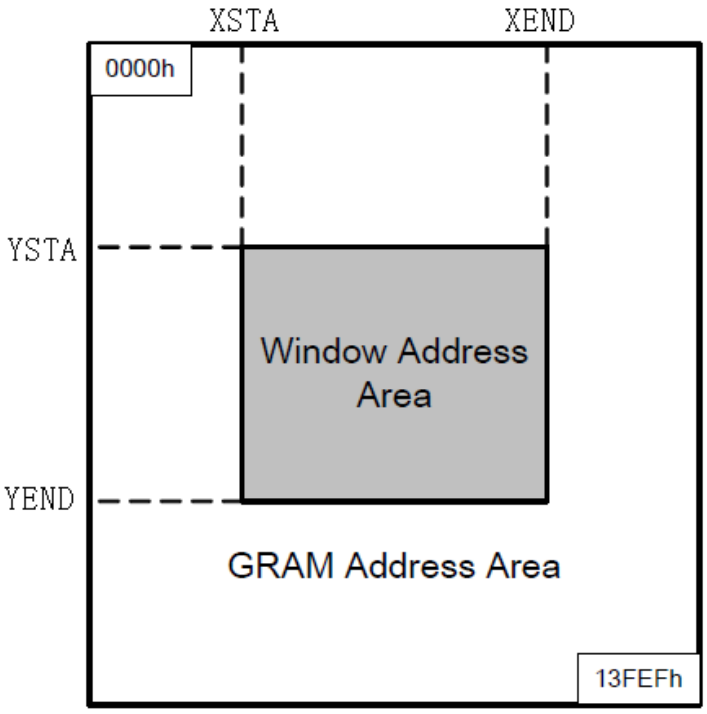

图 1

我们本来的液晶是分辨率是 240*320.对图 1 的 0X00~0XEF（x 坐标），0X00~0X13F（y 坐标）。图一中我们开辟了一个灰色区域的窗口，它的范围为 XSTA~XEND，YSTA~YEND。这样我们开辟窗口以后，再往 LCD 写数据，它就只会在这个窗口范围内地址按照设定的方向自增。比如从左到右，从上到下的扫描方式，规律为：从(XSTA,YSTA)开始,x 坐标递增，每次遇到 XEND 地址则 y 坐标增 1，同时 x 坐标重设为 XSTA，直到 y 坐标递增到大于 YEND，此时坐标又变为 (XSTA,YSTA)。

所以，只要我们预先知道图片数据的生成格式，以及图片尺寸，那么我们就可以采用开窗方式来画图，从而提高效率。

二， Image2lcd 简介。

Image2Lcd 是一款非常好的图像工具软件，它能把各种来源的图片转换成特定的数据格式以用来匹配单片机系统所需要的显示数据格式。Image2Lcd 支持的输入图像格式包括：BMP, WBMP, JPG, GIF, WMF, EMF, ICO, 等等。Image2Lcd 的输出数据类型包括定制的二进制类型、C 语言数组类型和标准的 BMP 格式、WBMP 格式。Image2Lcd 能可视调节输入图象的数据扫描方式、灰度(颜色数)、图像数据排列方式、亮度、对比度、等等。对于包含了图像头数据保存的图像数据文件，Image2Lcd 能重新打开作为输入图像。

因为 image2lcd 能生成带图像数据头的数据文件，使得我们处理起来方便很多，这里我们仅以 16 位真彩色为例进行说明。

在该软件的帮助文件查到对“4096 色/16 位真彩色/18 位真彩色/24 位真彩色/32 位真彩色”图片，其生成的图像数据头的结构为：

```
typedef struct _HEADCOLOR
{
    unsigned char scan;
    unsigned char gray;
    unsigned short w;
    unsigned short h;
    unsigned char is565;
    unsigned char rgb;
}HEADCOLOR;
```

各个成员的功能描述如下：

scan: 扫描模式

Bit7: 0:自左至右扫描，1:自右至左扫描。

Bit6: 0:自顶至底扫描，1:自底至顶扫描。

Bit5: 0:字节内象素数据从高位到低位排列，1:字节内象素数据从低位到高位排列。

Bit4: 0:WORD 类型高低位字节顺序与 PC 相同，1:WORD 类型高低位字节顺序与 PC 相反。

Bit3~2: 保留。

Bit1~0: [00]水平扫描，[01]垂直扫描，[10]数据水平,字节垂直，[11]数据垂直,字节水平。

gray: 灰度值

灰度值，1:单色，2:四灰，4:十六灰，8:256 色，12:4096 色，16:16 位彩色，24:24 位彩色，32:32 位彩色。

w: 图像的宽度。

h: 图像的高度。

is565: 在 4096 色模式下为 0 表示使用[16bits(WORD)]格式，此时图像数据中每个 WORD 表示一个象素；为 1 表示使用[12bits(连续字节流)]格式，此时连续排列的每 12Bits 代表一个象素。在 16 位彩色模式下为 0 表示 R G B 颜色分量所占用的位数都为 5Bits，为 1 表示 R G B 颜色分量所占用的位数分别为 5Bits,6Bits,5Bits。

在 18 位彩色模式下为 0 表示"6Bits in Low Byte"，为 1 表示"6Bits in High Byte"。

在 24 位彩色和 32 位彩色模式下 is565 无效。

rgb: 描述 R G B 颜色分量的排列顺序，rgb 中每 2Bits 表示一种颜色分量，[00]表示空白，[01]表示 Red，[10]表示 Green，[11]表示 Blue。

在 HEADCOLOR 中，scan，w，h 这三个参数对我们的图片显示尤为有用。直到了 w 和 h，就可以直到开窗的大小。而 scan 的最高两位，则代表了图片数据生成时的扫描方向，也就

是我们开窗后地址自增的方向。直到了这几个参数，我们就可以很方便的解析各种大小，各种扫描方式的图片数据了。

下面我们以图 2 为例，按从左到右，从上到下的扫描方式，生成 16 位真彩（RGB:565）格式的图像数据。

图 2

该图片的尺寸为 200*168。我们用 image2lcd V2.9 打开此图片，设置如图 3 所示：

图 3

图 3 中，我们设置如上图。我们要生成的的图像数据为 16 位真彩，所以在 2 处选择 16 位真彩色，然后扫描方式为水平扫描，在 3 处选中包含头像数据头选项**（注意：不能选择“高位在前（MSB First）”这个选项！！！）**。在 5 处选中 16 位彩色选项卡，然后在颜色位数（4 处）选择 RGB565（因为我们的液晶刚好也是 RGB565 格式）。然后点击保存，命名为 image1，可以得到图像数组如下：

```
const unsigned char gImage_image1[67208] = { 0X00,0X10,0XC8,0X00,0XA8,0X00,0X01,0X1B,
    0X6B,0X6E,0XD1,0X9F,0XF5,0XB7,0XD3,0XAF,0XF3,0XAF,0X0F,0X8F,0XCE,0X86,0XF3,0XAF,
                                    ……
    0X73,0XB7,0XF6,0XCF,0XF9,0XD7,0X98,0XCF,0X71,0XAE,0XD6,0XDF,0XFA,0XE7,0XF8,0XCF,
```

```
0XF6,0XC7,0X10,0X9F,0X53,0XB7,0XD5,0XC7,0XF6,0XCF,0X74,0XBF,0XD1,0XA6,0XF7,0XD7,
};
```

其中红色数字为图像头数据，一共是 8 个字节，刚好是 HEADCOLOR 的大小。紧随其后的就是按设定的方向顺序存放的图像数据（颜色数据）。这样我们只需要在软件上对这个数组（gImage_image1）的数据进行解析，就可以还原图像了。

三， 软件实现。

在第二节的介绍中，我们得到了一个数组（gImage_image1），而从第一节的介绍，我们需要一个开窗函数，以及一个扫描方向设置函数，这里提供这两个函数的代码如下：

```
//设置 LCD 的自动扫描方向
//0~7：代表 8 个方向(具体定义见 lcd.h)
//9320/9325/9328/4531/1505/b505/8989 等 IC 已经实际测试
void LCD_Scan_Dir(u8 dir)
{
    u16 regval=0;
    u8 dirreg=0;
#if USE_HORIZONTAL//使用横屏
    switch(dir)//方向转换
    {
        case 0:dir=6;break;
        case 1:dir=7;break;
        case 2:dir=4;break;
        case 3:dir=5;break;
        case 4:dir=1;break;
        case 5:dir=0;break;
        case 6:dir=3;break;
        case 7:dir=2;break;
    }
#endif
    if(DeviceCode==0x8989)//8989 IC
    {
        dirreg=0X11;
        regval=0X6040;//65K
    }else//其他驱动 IC
    {
        dirreg=0X03;
        regval=1<<12;
    }
    switch(dir)
    {
        case L2R_U2D://从左到右,从上到下
            regval|=(1<<5)|(1<<4)|(0<<3);
            break;
        case L2R_D2U://从左到右,从下到上
            regval|=(0<<5)|(1<<4)|(0<<3);
            break;
        case R2L_U2D://从右到左,从上到下
            regval|=(1<<5)|(0<<4)|(0<<3);
            break;
```

```
            case R2L_D2U://从右到左,从下到上
                regval|=(0<<5)|(0<<4)|(0<<3);
                break;
            case U2D_L2R://从上到下,从左到右
                regval|=(1<<5)|(1<<4)|(1<<3);
                break;
            case U2D_R2L://从上到下,从右到左
                regval|=(1<<5)|(0<<4)|(1<<3);
                break;
            case D2U_L2R://从下到上,从左到右
                regval|=(0<<5)|(1<<4)|(1<<3);
                break;
            case D2U_R2L://从下到上,从右到左
                regval|=(0<<5)|(0<<4)|(1<<3);
                break;
        }
        LCD_WriteReg(dirreg,regval);
}
//设置窗口
//sx,sy,ex,ey 窗口坐标
//窗口大小:(ex-sx+1)*(ey-ex+1)
//注意,确保 ex>=sx;ey>=sy!!!!
//9320/9325/9328/4531/1505/b505/8989 等 IC 已经实际测试
void LCD_Set_Window(u16 sx,u16 sy,u16 ex,u16 ey)
{
        u8 hsareg,heareg,vsareg,veareg;
        u16 hsaval,heaval,vsaval,veaval;
#if USE_HORIZONTAL   //使用横屏
        //窗口值
        hsaval=sy;
        heaval=ey;
        vsaval=319-ex;
        veaval=319-sx;
#else                        //竖屏
        //窗口值
        hsaval=sx;
        heaval=ex;
        vsaval=sy;
        veaval=ey;
#endif
        if(DeviceCode==0X8989)//8989 IC
        {
            hsareg=0X44;heareg=0X44;//水平方向窗口寄存器 (1289 的由一个寄存器控制)
            hsaval|=(heaval<<8);    //得到寄存器值.
```

```
        heaval=hsaval;
        vsareg=0X45;veareg=0X46;//垂直方向窗口寄存器
    }else   //其他驱动 IC
    {
        hsareg=0X50;heareg=0X51;//水平方向窗口寄存器
        vsareg=0X52;veareg=0X53;//垂直方向窗口寄存器
    }
    //设置寄存器值
    LCD_WriteReg(hsareg,hsaval);
    LCD_WriteReg(heareg,heaval);
    LCD_WriteReg(vsareg,vsaval);
    LCD_WriteReg(veareg,veaval);
}
```

这两个函数已经添加到 ILI93xx.c 的源码中，具体请看本应用文档的对应扩展实验（ALIENTEK MINISTM32 扩展实验 13 内部 FLASH 图片显示实验）。更新后的 ILI93xx.c 版本为 V1.6。同时该实验的 USMART 部分也有了更新，最新版本的 USMART 为 V2.6。

扩展实验 13 的源码是在标准实验 10 的基础上修改而来的，加入了 usmart 组建以及新建了 IMAG2LCD 的组。见图 4：

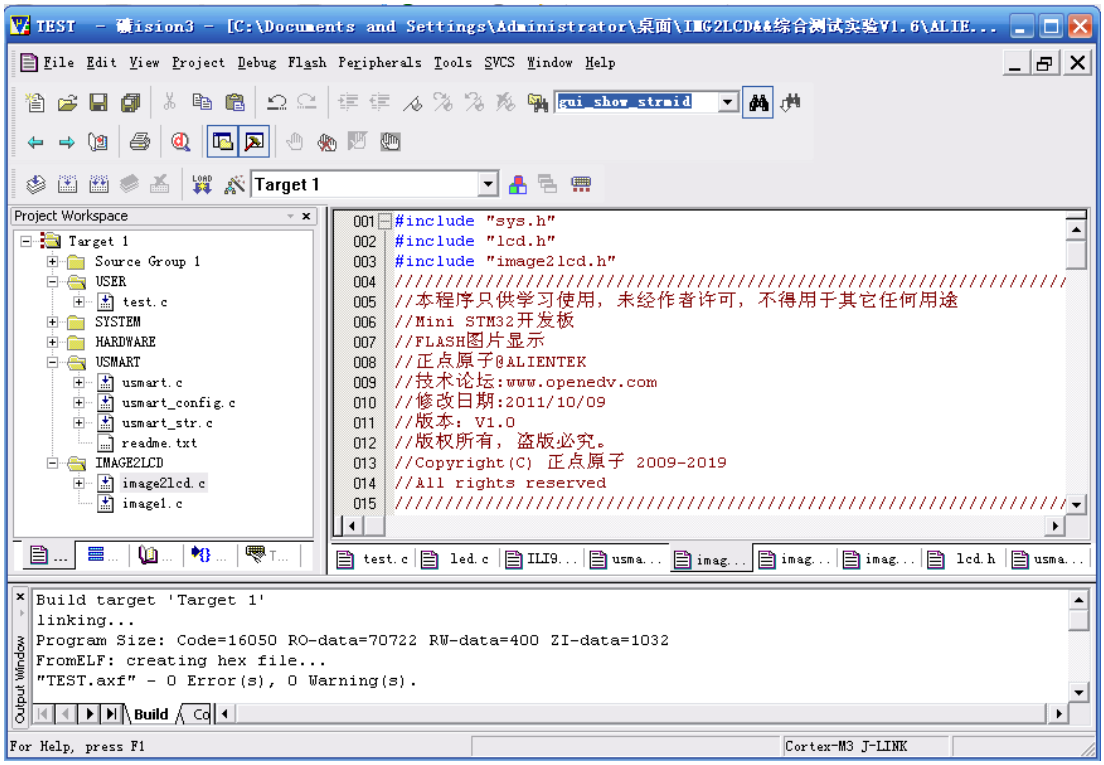

图 4

有了 LCD_Set_Window 和 LCD_Scan_Dir 这两个函数，做起来就方便多了（通过画点的方式也可以实现），根据第一节的原理，编写出的 flash->lcd 函数部分即 image2lcd.c 的内容如下：

```
//从 8 位数据获得 16 位颜色
//mode:0,低位在前,高位在后.
//       1,高位在前,低位在后.
```

```
//str:数据
u16 image_getcolor(u8 mode,u8 *str)
{
	u16 color;
	if(mode)
	{
		color=((u16)*str++)<<8;
		color|=*str;
	}else
	{
		color=*str++;
		color|=((u16)*str)<<8;
	}
	return color;
}
//在液晶上画图
//xsta,ysta,xend,yend:画图区域
//scan:见 image2lcd V2.9 的说明.
//*p:图像数据
void image_show(u16 xsta,u16 ysta,u16 xend,u16 yend,u8 scan,u8 *p)
{
	u32 i;
	u32 len=0;
	LCD_Set_Window(xsta,ysta,xend,yend);
	if((scan&0x03)==0)//水平扫描
	{
		switch(scan>>6)//设置扫描方式
		{
			case 0:
				LCD_Scan_Dir(L2R_U2D);//从左到右,从上到下
				LCD_SetCursor(xsta,ysta);//设置光标位置
				break;
			case 1:
				LCD_Scan_Dir(L2R_D2U);//从左到右,从下到上
				LCD_SetCursor(xsta,yend);//设置光标位置
				break;
			case 2:
				LCD_Scan_Dir(R2L_U2D);//从右到左,从上到下
				LCD_SetCursor(xend,ysta);//设置光标位置
				break;
			case 3:
				LCD_Scan_Dir(R2L_D2U);//从右到左,从下到上
				LCD_SetCursor(xend,yend);//设置光标位置
				break;
```

```
            }
        }else   //垂直扫描
        {
            switch(scan>>6)//设置扫描方式
            {
                case 0:
                    LCD_Scan_Dir(U2D_L2R);//从上到下,从左到右
                    LCD_SetCursor(xsta,ysta);//设置光标位置
                    break;
                case 1:
                    LCD_Scan_Dir(D2U_L2R);//从下到上从,左到右
                    LCD_SetCursor(xsta,yend);//设置光标位置
                    break;
                case 2:
                    LCD_Scan_Dir(U2D_R2L);//从上到下,从右到左
                    LCD_SetCursor(xend,ysta);//设置光标位置
                    break;
                case 3:
                    LCD_Scan_Dir(D2U_R2L);//从下到上,从右到左
                    LCD_SetCursor(xend,yend);//设置光标位置
                    break;
            }
        }
        LCD_WriteRAM_Prepare();                 //开始写入 GRAM
        len=(xend-xsta+1)*(yend-ysta+1);  //写入的数据长度
        for(i=0;i<len;i++)
        {
            LCD_WR_DATA(image_getcolor(scan&(1<<4),p));
            p+=2;
        }
    #if USE_HORIZONTAL   //使用横屏
        LCD_Set_Window(0,0,319,239);
    #else
        LCD_Set_Window(0,0,239,319);
    #endif
    }

    //在指定的位置显示一个图片
    //此函数可以显示 image2lcd 软件生成的任意 16 位真彩色图片.
    //限制:1,尺寸不能超过屏幕的区域.
    //      2,生成数据时不能勾选:高位在前(MSB First)
    //      3,必须包含图片信息头数据
    //x,y:指定位置
    //imgx:图片数据(必须包含图片信息头,"4096 色/16 位真彩色/18 位真彩色/24 位真彩色/32 位
```

```
真彩色”的图像数据头)
//注意:针对 STM32,不能选择 image2lcd 的"高位在前(MSB First)"选项,否则 imginfo 的数据将不正确!!
void image_display(u16 x,u16 y,u8 * imgx)
{
    HEADCOLOR *imginfo;
    u8 ifosize=sizeof(HEADCOLOR);//得到 HEADCOLOR 结构体的大小
    imginfo=(HEADCOLOR*)imgx;
    image_show(x,y,x+imginfo->w-1,y+imginfo->h-1,imginfo->scan,imgx+ifosize);
}
```

通过这三个函数即实现了对 image2lcd 软件生成的任意扫描方向、任意尺寸（不大于240*320）的图像数据的解码。

用户直接通过调用 image_display 函数既可以实现对存在 flash 里面的图像数据进行显示了。X，y 为图像要显示的位置，imgx，则为 flash 图像数据的首地址。

在 main 函数里面的实现代码如下：

```
//Mini STM32 开发板扩展实验 13
//内部 FLASH 图片显示 实验
//正点原子@ALIENTEK
//技术论坛:www.openedv.com
int main(void)
{
    u8 i=0;
    HEADCOLOR *imginfo;
    u16 x=0,y=0;
    u16 x0,y0;
    imginfo=(HEADCOLOR*)gImage_image1;     //得到文件信息

    Stm32_Clock_Init(9);//系统时钟设置
    delay_init(72);       //延时初始化
    uart_init(72,9600); //串口 1 初始化
    usmart_dev.init(72);//初始化 USMART
    LED_Init();
    LCD_Init();
    POINT_COLOR=RED;
    LCD_ShowString(30,50,"Mini STM32 ^_^");
    LCD_ShowString(30,70,"FLASH PICTURE TEST");
    LCD_ShowString(30,90,"ATOM@ALIENTEK");
    LCD_ShowString(30,110,"2011/10/09");
    delay_ms(1500);//等待 1.5 秒
    srand(imginfo->h*imginfo->w);
    while(1)
    {
        if(i==0)
        {
```

```
                LCD_Clear(0X0000);//黑屏
                if(imginfo->w>=240||imginfo->h>=320)
                {
                    POINT_COLOR=RED;
                    LCD_ShowString(10,70,"The Picture is too large");
                    continue;
                }
                x0=x;y0=y;
                while((x+imginfo->w)>240||x==x0)//超过屏幕尺寸了
                {
                    x=rand();//获得随机的 x 值
                }
                while((y+imginfo->h)>320||y==y0)//超过屏幕尺寸了
                {
                    y=rand();//获得随机的 y 值
                }
                image_display(x,y,(u8*)gImage_image1);//在指定地址显示图片
            }
            i++;
            if(i>10)i=0;
            LED0=!LED0;
            delay_ms(200);
        }
}
```

在 main 函数里面，我们取得 x，y 的随机坐标，实现一个图片在 LCD 屏幕上到处移动的效果。每 2 秒钟更新一次位置。大家可以下载该实验代码到 ALIENTEK MiniSTM32 开发板上感受下效果（速度还是很流畅的！）。

想简单一点的话，只需要调用 image_display(x,y,(u8*)gImage_image1);函数即可。gImage_image1 即为我们用 image2lcd 生成的图像数据。

正点原子@ALIENTEK

2011-10-9