


xmj



博客园 首页 新随笔 联系 管理 订阅 

随笔 - 545 文章 - 0 评论 - 6 0

公告

昵称: xmj
园龄: 10年
粉丝: 12
关注: 13
[+加关注](#)

< 2019年11月 >

日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)

随笔分类

C++语言(3)
C语言
Go语言(14)
linux(3)
mysql
redis
thrift
win32&MFC(2)
安全
调试
分布式(2)
逆向分析
书籍(2)
数据库(2)
网络(6)

随笔档案

2018年9月(1)
2018年5月(5)

基于STM32的uCGUI移植和优化

基于STM32的uCGUI移植和优化

首先开始这个说明之前,要简要说明下具体的环境:
编译工具:MDK4.20
开发板:安富莱v2版开发板
调试器:JLink v8盗版

移植篇









相信大家有移植经验的都知道,移植确实是一件非常墨迹的事情,怎么说呢,代码都是别人的,风格也是别人的,文件结构,定义之类都是别人的,看别人的东西是种进步,但是,也是一个痛苦的过程,因为有时候资料确实很少,而且有时候还是E文的,专业名词一大堆,我们根本没有办法想象工作量是多么的巨大.

不过事情都是这样,你不懂他的时候他就像是巨山,但是一旦你理解他的时候,你才会感觉到原来他是那么的简单(从我的经验上来看,至少应该是这样的).

好吧,闲话少说,我们就来开始我们的移植之旅把.

首先,我们需要准备的东西有uCGUI3.90,这个版本是大家现在用的比较多的,效率也比较高,别人都是这么评论的,至于其他版本的,我没有接触很多,所以不能过多评论.

uCGUI有三个文件夹,一个是tool,这个文件夹是用来使用一些uCgui的上位机程序,基本都是字体和模板查看之类的.在sample文件夹下面是已经别人帮你写好了很多有用的东西,像跟操作系统有关的GUI_X或者一些模板(后面我们会用到的自己定义的Demo),或者是gui配置.后面再——详细叙说这个文件夹的功能.在Start文件夹里面,这是我们最主要的文件夹.里面就包含了uCGUI的源代码,uCGUI的作者把源代码放进vc里面进行编译了(当然,这是用标准C语言写的程序,所以我们可以放在任何C语言平台下编译而不会担心兼容性问题,这个uCGUI在这方面做的算是完美了),所以,我们可以在vc平台下写界面,然后再把代码拷进我们的下位机编译器进行编译,这样子效率就会非常高了.(像51那时候写界面就是疯狂的一次一次的烧,真是纠结..).

	AntiAlias	2011/7/23 17:22	文件夹
	ConvertColor	2011/7/23 17:22	文件夹
	ConvertMono	2011/7/23 17:22	文件夹
	Core	2011/7/23 17:22	文件夹
	Font	2011/7/23 17:22	文件夹
	JPEG	2011/7/23 17:22	文件夹
	LCDDriver	2011/7/23 17:22	文件夹
	MemDev	2011/7/23 17:22	文件夹
	MultiLayer	2011/7/23 17:22	文件夹
	Widget	2011/7/23 17:22	文件夹
	WM	2011/7/23 17:22	文件夹

然后这里放的就是uCGUI的源代码了,在GUI文件夹下面.

https://www.cnblogs.com/xumaojun/p/8541672.html

1/14

2018年3月(537)
2017年12月(1)
2016年4月(1)

最新评论

1. Re:在Visual Studio Code中配置GO开发环境
mark
--立志做一个好的程序员
2. Re:基于STM32的uCGUI移植和优化
MARK
--tianqi911
3. Re:WebSocket的C++服务器端实现
服务器数据发送功能，如何实现呢
--证道树
4. Re:TCC（Tiny C Compiler）介绍
tcc不支持C11呀
--楓羽37
5. Re:TCC（Tiny C Compiler）介绍
真好 这才是最简纯c开发环境 网上推荐一大堆IDE
--楓羽37

阅读排行榜

1. 哈希(Hash)与加密(Encrypt)的基本原理、区别及工程应用(9303)
2. 在Visual Studio Code中配置GO开发环境(8768)
3. WebSocket的C++服务器端实现(7865)
4. 解析Monte-Carlo算法(基本原理,理论基础,应用实践)(7321)
5. AES加密算法(C++实现,附源码)(5805)

评论排行榜

1. TCC（Tiny C Compiler）介绍(2)
2. 在Visual Studio Code中配置GO开发环境(1)
3. WebSocket的C++服务器端实现(1)
4. 基于STM32的uCGUI移植和优化(1)
5. 在Chrome浏览器中保存的密码有多安全？(1)

推荐排行榜

1. 哈希(Hash)与加密(Encrypt)的基本原理、区别及工程应用(2)
2. 漫谈WebQQ 协议(1)

目 录	内 容
Config	配置文件
GUI/AntiAlias	抗锯齿支持 *
GUI/ConvertMono	用于 B/W（黑白两色）及灰度显示的色彩转换程序
GUI/ConvertColor	用于彩色显示的色彩转换的程序
GUI/Core	μC/GUI 内核文件
GUI/Font	字体文件
GUI/LCDDriver	LCD 驱动
GUI/Mendev	存储器件支持 *
GUI/Touch	触摸屏支持 *
GUI/Widget	视窗控件库 *
GUI/WM	视窗管理器 *

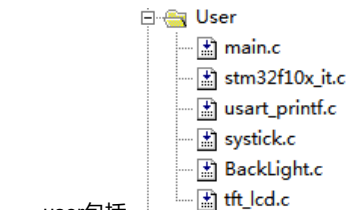
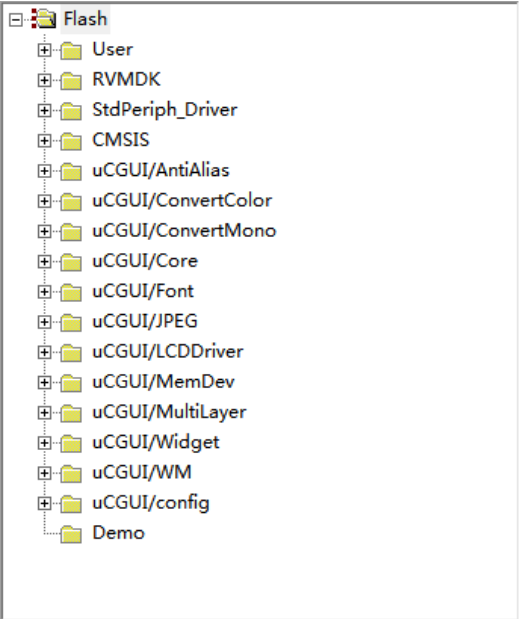
这则是每个文件夹的功能(参考uCGUI中文手册,ucgui.com翻译).

大概看一下就可以了,这个跟我们移植的关系不大,关键是带*的可以不包含进去(待会配置会讲到.),然后其他的都要包含进去.

接着我们要把我们的文件包含进我们已经搭建好的工程,这里说明下我们的工程要求.

一般来说,我们要画一个图形,最基本的就是从点开始,从点到线,从点到面...,所以在已经建好的工程里面你要能点亮你的屏幕,能点出最基本的点,能填充出最基本矩阵(这是uCGUI最包含的函数),反正我移植的时候涉及到的包括三个函数,LCD_Init();LCD_Draw_Point(x,y,color),LCD_Fillcircuit(x1,x2,y1,y2).这三个函数是必须的,后面也会说明如何把这三个函数进行填充.

当我们把文件复制进去的时候,再加上我们一开始已经创建好的工程的时候,文件结构差不多就是这个样子了,截图如下



user包括 ,main函数就是我们初始化和函数调用,绘图用的文件,另外那几个文件相信大家明白了把,tft_lcd.c就是你在,没有移植uCGUI的情况下,纯液晶屏驱动,这里建议把液晶屏的

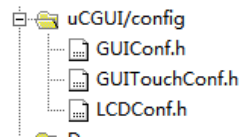
3. Golang源码探索(三) GC的实现原理(1)
4. TCC (Tiny C Compiler) 介绍(1)
5. 解析Monte-Carlo算法(基本原理,理论基础,应用实践)(1)

Copyright © 2019 xmj
Powered by .NET Core 3.0.0 on Linux

API和最底层驱动(API就是画圆啊,画椭圆啊,清除屏幕之类的,底层驱动就是驱动液晶屏的管脚运作,fsmc初始化,时钟配置之类的),不过我这里也是集成在一起了,比较懒,大家别学。

其他文件夹我都包含进去了,在没包含进去的时候,编译是可以通过的,但是,那么多文件包含进去,有些配置还是没有设定好的,所以会有错误,蛮编译一下,没事的。

这里我们需要修改的文件只有这几个:



,这是ucgui开放给我们的用户层的文件,在ucgui中,lcdDrive文件夹要自己加进去,GUI_X.c也是,另外三个文件都是包含了,在GUIConf.h中

```

1 #ifndef GUICONF_H
2 #define GUICONF_H
3
4 #define GUI_OS (0) // 这里指的是对操作系统的支持,因为我们这里只有单纯的
移植uCGUI,
5 // 所以,要把这个关闭,不然后面会有很多东西编译进去
6 // 不然到时候编译的时候会发生很多你无法修改的错误
7 #define GUI_SUPPORT_TOUCH (0) // 这里则是对触摸屏的支持,触摸屏我是能做,但是没有
用,
8 //所以省去麻烦,把触摸屏关掉,相信移植好之后,要支持触摸
屏大家都会有门路了
9 #define GUI_SUPPORT_UNICODE (1) // UNICODE编码支持,如果大家只是单纯的想用英文显示,
而不移植中文字库进去,
10 //这个是可以关掉的,因为UNICODE是向下支持的,所以开不
开无所谓
11 // 反正都是能够正常显示的
12 #define GUI_DEFAULT_FONT &GUI_Font6*8 // 这里是设定默认字体的,我们可以在要写什
么字的时候把该字号的字体.
13 //c包含进我们的主函数里面,所以这里不用
改
14 #define GUI_ALLOC_SIZE 5000 // 这里讲的是动态内存机制
15 // 这里rgb接口模式的可能会有用到,uCGUI就是在我们的ram开辟一
块空间,
16 //然后uCGUI把运算好的每个点都放进我们主控ram里面的空间
17 // 所以,这里就相当于把写进液晶gram里面的操作变成了写进主控
ram里面,
18 //那么大家可能就会问了,干嘛这么多次一举,直接写进去不就可以了
19
20 /*原理: 一般来说,在大的屏幕上面(4.0以上吧,印象中),都是没有控制器,(像我的液晶屏就是spfd5420,当然,
21 不同的屏幕的液晶主控都是不一样的,但是寄存器操作都是差不多的,
22 所以有些初始化配置还是能互用的。)所以呢,这时候我们要用到的就只有RGB接口了,
23 RGB要求我们要不断的刷新屏幕,刷新率越高,效果就越好,因为一般这种用来做动态的,uCGUI就是属于静
态类型的
24 像如果我们要用stm32主控做视频应用的时候,就是动态的,我们需要不断的刷新屏幕,但是当我们主控一
边运算,
25 一边往液晶接口送数据的时候,会有明显刷屏的感觉(运算->画点->运算->画点....,这个运算
->运算.....画点->画点->画点...是不一样的,因为对屏幕一直画点,填充,而中间不用插入运算,
26 刷一个屏幕时间时间倍速差别是非常巨大的,后面大家也会见识到这种差别。),所以,用GUI申请的空间里
面
27 边运算,边填充,填充完再一次性运出去(这里可以通过DMA控制FSMC总线,不断的从外置SRAM往GRAM自动
搬运数据,
28 这是不用主控去插手的,所以,主控大部分时间是负责运算,其他时间可以空闲出来,
29 让DMA自己去忙活),同理,因为dma跟cpu的分工,所以,这里同样的把画点,画点,运算,运算不完全的分开
了,
30 屏幕刷新速度非常可观(DMA的速度相比大家还是非常了解的,它就是为速度而生的。),*/
31
32 // #define GUI_ALLOC_SIZE 1024*1024
/* Size of dynamic memory ... For WM and memory devices*/

```

```

33      /*****
34      *
35      *          Configuration of available packages
36      */
37
38 #define GUI_WINSUPPORT      1    // 这个是窗口支持,一般开始开着的
39 #define GUI_SUPPORT_MEMDEV  1    // 内存控制,开
40 #define GUI_SUPPORT_AA      0    // 抗锯齿,为了性能着想,还是关了比较好
41
42 #endif /* Avoid multiple inclusion */

```



GUITouchConf.h是有关于触摸屏配置的,这里我们就略过了.

```

1 #ifndef LCDCONF_H
2 #define LCDCONF_H
3
4 /*****
5 *
6 *          General configuration of LCD
7 *
8 *****/
9 */
10
11 /*
12 * 这个定义的是你x轴的长度,像我这里的屏幕长为400个像素
13 */
14 #define LCD_XSIZE      (400)
15
16 /*
17 * 这里这是屏幕的宽
18 */
19 #define LCD_YSIZE      (240)    /* Y轴长度 */
20
21 /*
22 * 这里是屏幕的颜色有多少个位
23 */
24 #define LCD_BITSPERPIXEL (16) /* 定义数据长度为16bit*/
25
26 /*
27 * 控制器类型,如果你里面有包含这些判断变量,这个最好改成你认识的
28 */
29 #define LCD_CONTROLLER 9325    /* 定义控制器类型 */
30
31
32 #endif /* LCDCONF_H */

```



配置层的东西我们都已经搞定了,接下来我们要修改的是uCGUI开放给我们的用户层的东西,GUI_X.c可以直接拷进去,这个是用户层和系统层的关联文件,一些demo也会用到这个文件的时间函数或者延迟函数,所以这个文件拷进去放着就可以了.



```

1 #include "GUI.h"
2 #include "GUI_X.h"
3
4 /*****
5 *
6 *      Global data
7 */
8 volatile int OS_TimeMS;
9
10 /*****
11 *
12 *      Timing:
13 *          GUI_X_GetTime()
14 *          GUI_X_Delay(int)
15
16 Some timing dependent routines require a GetTime
17 and delay function. Default time unit (tick), normally is
18 1 ms.
19 译:一些需要时间的相关函数需要用到gettime和延迟.
20 默认时间单位为1ms.
21 */
22
23 int GUI_X_GetTime(void) {
24     return 0;
25 }
26
27 void GUI_X_Delay(int ms)
28 {
29 }
30
31 /*****
32 *
33 *      GUI_X_Init()
34 *
35 * Note:
36 *      GUI_X_Init() is called from GUI_Init is a possibility to init
37 *      some hardware which needs to be up and running before the GUI.
38 *      If not required, leave this routine blank.
39 *
40 *      译:GUI_X_Init()是在gui_init()调用前,gui启动或者运行前准备.
41 *      如果不是必须的,可以把这个函数留空白.
42 */
43
44 void GUI_X_Init(void)
45 {
46 }
47
48
49
50
51 /*****
52 *
53 *      GUI_X_ExecIdle
54 *
55 * Note:
56 *      Called if WM is in idle state
57 *      译:视窗管理器空闲时候调用
58 */
59
60 void GUI_X_ExecIdle(void) {}
61
62 /*****
63 *
64 *      Logging: OS dependent
65
66 Note:
67 Logging is used in higher debug levels only. The typical target
68 build does not use logging and does therefor not require any of
69 the logging routines below. For a release build without logging
70 the routines below may be eliminated to save some space.
71 (If the linker is not function aware and eliminates unreferenced
72 functions automatically)

```

```

73 译:系统日志层应用程序
74
75 */
76
77 void GUI_X_Log      (const char *s) {}
78 void GUI_X_Warn     (const char *s) {}
79 void GUI_X_ErrorOut(const char *s) {}

```



在uCGUI和底层驱动接口文件时LCDDriver.c,大家打开文件夹可以看到这几个文件:lcdwin.c,lcdnull.c,lcdDummy.c,这三个文件你随便修改哪个都行,一开始我是直接修改lcdnull.c的,不过一开始没经验,一直不能正常调用我的uCGUI函数,所以,我的队友告诉我,要修改另外一个(lcdwin.c),这次才反应过来,理论上,修改lcdnull.c也是可以的,就是一些细节性的东西可以多注意点.下面我讲解下怎么修改lcdwin.c这个文件.

源文件我先贴出来:

这个文件可能有点大,500多行,都是密密麻麻的代码,不过,我们要想使我们的uCgui能够使用,要修改的函数其实也就每几句.

我先给大家讲解下这里面函数的关系:

```

#if LCD_BITSPERPIXEL <= 8
#define PIXELINDEX U8
#else
#define PIXELINDEX WORD
#endif

```

这里我们定义的是16位的像素点(在前面那三个配置文件里面),定义为word.

```

#define SETPIXEL(x, y, c) LCDSIM_SetPixelIndex(x, y, c, LCD_DISPLAY_INDEX)
#endif
#define XORPIXEL(x, y)    _XorPixel(x, y)

```

这里定义的是显示点的函数,我们需要把所有LCDSIM_SetPixelIndex(x, y, c, LCD_DISPLAY_INDEX),都换成你的画点函数,比如我这里使用的是

LCD_L0_SetPixelIndex(x轴坐标,y轴坐标,该点颜色);然后呢,就是修改LCD_L0_FillRect(int x0, int y0, int x1, int y1);这个是填充矩形函数,

你可以直接把函数里面的东西清空,然后写上自己在底层硬件驱动的api,我的函数优化得体无完肤了,基本没有办法再拷贝过来,所以待会讲这函数的时候,直接跳过去.

最后最重要的要修改的就是LCD_L0_Init(void);这函数里面可以直接填充你的初始化函数,好了,剩下基本就不需要修改了,其他的修改就留给优化了.

下面我给大家详细讲解下每个函数的用途



```

1  /*****
2  *
3  *      LCD_L0_SetPixelIndex
4  */
5  void LCD_L0_SetPixelIndex(int x, int y, int PixelIndex) {
6      /* 设定指定点颜色值 */
7  }
8

```

```

9  /*****
10 *
11 *      LCD_L0_GetPixelIndex
12 */
13 unsigned int LCD_L0_GetPixelIndex(int x, int y) {
14     /* 取得某点像素颜色值,并返回 */
15 }
16
17 /*****
18 *
19 *      LCD_L0_XorPixel
20 */
21 void LCD_L0_XorPixel(int x, int y) {
22     /* 反转像素函数 */
23 }
24
25 /*****
26 *
27 *      LCD_L0_DrawHLine
28 */
29 void LCD_L0_DrawHLine(int x0, int y, int x1) {
30     /* 绘制水平线函数 */
31 }
32
33 /*****
34 *
35 *      LCD_L0_DrawVLine
36 */
37 void LCD_L0_DrawVLine(int x, int y0, int y1) {
38     /* 绘制垂直线函数 */
39 }
40
41 /*****
42 *
43 *      LCD_L0_FillRect
44 */
45 void LCD_L0_FillRect(int x0, int y0, int x1, int y1) {
46     /* 填充矩阵函数 */
47 }
48
49 /*****
50 *
51 *      LCD_L0_DrawBitmap
52 */
53 void LCD_L0_DrawBitmap(int x0, int y0,
54                         int xsize, int ysize,
55                         int BitsPerPixel,
56                         int BytesPerLine,
57                         const U8 GUI_UNI_PTR * pData, int Diff,
58                         const LCD_PIXELINDEX* pTrans)
59 {
60     /* 画位图函数 */
61 }
62
63 /*****
64 *
65 *      LCD_L0_SetOrg
66 */
67 void LCD_L0_SetOrg(int x, int y) {
68     /* 该函数保留 */
69 }
70
71 /*****
72 *
73 *      LCD_On / LCD_Off
74 */
75 void LCD_On (void) {
76     /* 开启LCD */
77 }
78 void LCD_Off(void) {
79     /* 关闭LCD */
80 }

```



```
81
82 /*****
83 *
84 *      LCD_L0_Init
85 */
86 int LCD_L0_Init(void) {
87     /* LCD初始化 */
88 }
89
90 /*****
91 *
92 *      LCD_L0_SetLUTEntry
93 */
94 void LCD_L0_SetLUTEntry(U8 Pos, LCD_COLOR Color) {
95     /* 修改 LCD 控制器的 LUT 的中的单个条目 */
96 }
97
98 #else
99
100 void LCDNull_c(void);
101 void LCDNull_c(void) {} /* 请保持这个函数为空 */
102
103 #endif
```

移植讲到这边差不多就结束了,因为是凭着自己的感觉写的,所以可能很多东西讲的不是很清楚,有需要我精讲的可以联系我,我会陆续修改.接下来我们开始我们的优化之旅

补充:

优化篇

这里的优化会介绍两种模式,GPIO口模拟程序,FSMC模块模式.

FSMC模式只有在STM32大容量模式的主控才有的,优化起来效果确实是很好的,因为当你根据液晶屏ic配置好fsmc的时候,液晶屏的寄存器和GRAM就会被映射到系统的4G内存空间的某一块区域,我们往指定地址写数据或读数据,FSMC就会自动帮你把数据送到液晶屏控制器上面了,所以这里面我们省去了对GPIO管脚和接口的操作(...这个确实很强大,因为你可以花更多的时间来进行运算,每个点都是通过指令来产生的,产生指令也是需要时间的,同样,模拟IO口时序也是需要时间的,把这部分时间剩下来,是非常可观的),至于一些人不理解FSMC其实也不是很打紧啦,你就把GRAM和液晶屏寄存器想象成一个完整的外置SRAM就可以了,因为他们的时序都是同一个原理的,而FSMC就是会自动帮你模拟各种IO口操作,我把它简单的理解成(你写的io口操作程序,他用硬件来实现了).

相信大家都有看到别人移植好的demo程序,里面最开始以来就是一个speedDEMO,这个是测试每秒钟可以通过计算可以画多少个点,里面包含了随即填充矩阵函数.

分数越高,代表着你刷屏也就是越快,当然,跟着我的脚步走,可以快到把刷这个字去掉(不严格意义上);

在做优化的时候,我们得提前做好准备,因为优化更多代表着是结构性的破坏,可读性的疯狂下降,因为从最底层的函数开始一层一层封装上去,使得函数可以很容易的去理解调用,我们优化就是拆开这些函数调用,并把一些有关硬件层的驱动进行修改,对算法进行更新.

举个函数调用的例子,填充矩阵:

这个比较大家可看可不看,涉及到一些汇编的知识.略过对后面的理解没有问题.

```
1 void fill_Rect(int v, int h, int x, int y)
2 {
3     for (i<v)
4     {
5         for (j<h)
```



```

6      {
7          Darw_Point(a,b);
8      }
9  }
10 }
11
12 void Darw_Point(int x,int y)
13 {
14     Set_LCD_reg();
15     Set_LCD_gram();
16 }
17
18 void Set_LCD_reg()
19 {
20     /* 写入寄存器 */
21 }
22
23 void Set_LCD_gram()
24 {
25     /* 写入显存 */
26 }

```

这是我随便写的一个函数,我们差不多都是这样子调用一个写点的函数把,通过计算可得,假设我们要绘制一个100*100的矩阵,

那么就要调用10000次

Darw_Point函数,10000次Set_LCD_reg函数,10000Set_LCD_gram()函数,而每次调用一个函数的时候,要包括如下过程:

1. 把要传递的参数压入堆栈中
2. 把上一层程序的返回地址压入堆栈中
3. 对程序的运行指针进行偏移操作,指向调用程序的入口地址
4. 执行程序时候如果有需要要进行堆栈保护
5. 执行程序完毕,堆栈要进行释放,还原给调用它的函数使用
6. 弹出返回地址
7. 返回,并继续执行上层程序.

而真正的,我们执行程序的时候,嵌套了那么多层的调用,执行的只有一个寄存器赋值,所以系统更多时候是在疯狂的进行压栈和出栈活动,

如果我们把这些时间都去掉,

效率就是呈现几何倍数增长,这种是毋庸置疑的.

出了通过对函数进行拆解,算法和硬件的结合也是非常重要的,像你画个矩形,一个通过点来填充,通过直线来填充,甚至,

为什么你没有想到通过矩阵来填充呢?有时候想法

就是这样,这在很多IC上是可以很轻松的实现的,前提是你得通读datasheet.

具体的我通过函数来为大家进行一一讲解:

这是优化后的填充点函数:

```

/*****已经最优优化*****/
*
*      LCD_L0_SetPixelIndex
*
* Purpose:
*      Writes 1 pixel into the display.
*/
void LCD_L0_SetPixelIndex(int x, int y, int ColorIndex)
{
    /* 填充x轴坐标和y轴坐标 */
    *(__IO uint16_t *) (Bank4_LCD_C) = 0x0200;

```

```

*(__IO uint16_t *) (Bank4_LCD_D) = y;

*(__IO uint16_t *) (Bank4_LCD_C) = 0x0201;
*(__IO uint16_t *) (Bank4_LCD_D) = 399 - x;
/* 写显存前准备 */
*(__IO uint16_t *) (Bank4_LCD_C) = 0x0202;

/* 写入数据 */
*(__IO uint16_t *) (Bank4_LCD_D) = ColorIndex;
}

```

与优化前的进行对比:这里的数据全部改成对地址进行操作(BANK4_LCD_D和C都是被映射了的内存地址).模拟IO口可能会有差别

大家在进行写函数的时候,对管脚进行配置都是直接调用库函数的setbit或者resetbit来进行的,我们可以直接查询库函数,对寄存器进行操作:

```

void GPIO_SetBits ( GPIO_TypeDef * GPIOx,
uint16_t GPIO_Pin
)

Sets the selected data port bits.

参数:
GPIOx,: where x can be (A..G) to select the GPIO peripheral.
GPIO_Pin,: specifies the port bits to be written. This parameter can be any combination
of GPIO_Pin_x where x can be (0..15).

返回值:
None

在文件stm32f10x_gpio.c第358行定义。

参考 GPIO_TypeDef::BSRR、IS_GPIO_ALL_PERIPH及IS_GPIO_PIN.

{
/* Check the parameters */
assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
assert_param(IS_GPIO_PIN(GPIO_Pin));

GPIOx->BSRR = GPIO_Pin;
}

```

库函数大家可以字节查询源代码,最后修改后demo代码如下:

```

#ifdef HARDWARE_PLATFORM_ALI
//0~15 作为数据线
#define LCD_DATA_BUS GPIO_Pin_All
//片选信号
// #define LCD_CS_LOW GPIO_ResetBits(GPIOC, GPIO_Pin_9)
// #define LCD_CS_HIGH GPIO_SetBits(GPIOC, GPIO_Pin_9)
// Ver.HXH
#define LCD_CS_LOW GPIOC->BRR = GPIO_Pin_9;
#define LCD_CS_HIGH GPIOC->BSRR = GPIO_Pin_9;
//数据/命令

```

```
//      #define LCD_RS_LOW GPIO_ResetBits(GPIOC, GPIO_Pin_8)
//      #define LCD_RS_HIGH GPIO_SetBits(GPIOC, GPIO_Pin_8)
//Ver.HXH
#define LCD_RS_LOW      GPIOC->BRR = GPIO_Pin_8;
#define LCD_RS_HIGH GPIOC->BSRR = GPIO_Pin_8;
//End.HXH
//写数据
//      #define LCD_WR_LOW GPIO_ResetBits(GPIOC, GPIO_Pin_7)
//      #define LCD_WR_HIGH GPIO_SetBits(GPIOC, GPIO_Pin_7)
//Ver.HXH
#define LCD_WR_LOW      GPIOC->BRR = GPIO_Pin_7;
#define LCD_WR_HIGH GPIOC->BSRR = GPIO_Pin_7;
//End.HXH
//读数据
//      #define LCD_RD_LOW GPIO_ResetBits(GPIOC, GPIO_Pin_6)
//      #define LCD_RD_HIGH GPIO_SetBits(GPIOC, GPIO_Pin_6)
//Ver.HXH
#define LCD_RD_LOW      GPIOC->BRR = GPIO_Pin_6;
#define LCD_RD_HIGH GPIOC->BSRR = GPIO_Pin_6;
//End.HXH
//PB0~15,作为数据线
#define DATAOUT(x) GPIOB->ODR=x; //数据输出
#define DATAIN      GPIOB->IDR;  //数据输入
#endif
```



把对管脚置高置底进行的操作完全跟改成寄存器操作,当然,你也可以改成对寄存器指针进行操作,不过效率是一样的,因为define的效果是复制,所以,通过观察源代码:

```
00127 #define GPIO_Pin_0      ((uint16_t)0x0001)
00128 #define GPIO_Pin_1      ((uint16_t)0x0002)
00129 #define GPIO_Pin_2      ((uint16_t)0x0004)
00130 #define GPIO_Pin_3      ((uint16_t)0x0008)
00131 #define GPIO_Pin_4      ((uint16_t)0x0010)
```

所以,当程序在编译的时候,也是把地址进行简单的拷贝,所以这部分功夫是可以完全剩下来的。

接下来是关于填充矩阵的函数操作:

```

*****
*
*      LCD_L0_FillRect
*/
void LCD_L0_FillRect(int x0, int y0, int x1, int y1)
{
    /* 最后修改.2011.7.23 */
    for (; y0 <= y1; y0++)
    {
        int x;
        /* 填充x轴坐标和y轴坐标 */
        *(__IO uint16_t *) (Bank4_LCD_C) = 0x0200;
        *(__IO uint16_t *) (Bank4_LCD_D) = y0;

        *(__IO uint16_t *) (Bank4_LCD_C) = 0x0201;
        *(__IO uint16_t *) (Bank4_LCD_D) = 399 - x0;

        /* 写显存前 准备 */
        *(__IO uint16_t *) (Bank4_LCD_C) = 0x0202;

        /* 开始写入显存 */
        x=x0;
        for (;x0 <= x1; x0++)
        {
            *(__IO uint16_t *) (Bank4_LCD_D) = LCD_COLORINDEX;
```



```

    }
    x0=x;
}
// /* 最后修改.2011.7.26 */
// u32 n;
// /* 设定窗口 */

// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0210;
// *(__IO uint16_t *) (Bank4_LCD_D) = y0;
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0211;
// *(__IO uint16_t *) (Bank4_LCD_D) = y1;
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0212;
// *(__IO uint16_t *) (Bank4_LCD_D) = 399-x1;
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0213;
// *(__IO uint16_t *) (Bank4_LCD_D) = 399-x0;

// /* 设定开始位置 */
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0200;
// *(__IO uint16_t *) (Bank4_LCD_D) = y0;

// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0201;
// *(__IO uint16_t *) (Bank4_LCD_D) = 399-x0;

// /* 进行填充 */
// n = (u32) (y1-y0+1)*(x1-x0+1);
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0202;
// while (n--)
// {
//     *(__IO uint16_t *) (Bank4_LCD_D) = LCD_COLORINDEX;
// }

// /* 恢复窗口 */
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0210;
// *(__IO uint16_t *) (Bank4_LCD_D) = 0x0000;
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0211;
// *(__IO uint16_t *) (Bank4_LCD_D) = 0x00EF;
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0212;
// *(__IO uint16_t *) (Bank4_LCD_D) = 0x0000;
// *(__IO uint16_t *) (Bank4_LCD_C) = 0x0213;
// *(__IO uint16_t *) (Bank4_LCD_D) = 0x018f;
}

```

这里有关于两种填充方式,都是避开函数操作,被注释掉的是对窗口进行操作的,而没被注释掉的是对线条进行填充.

对线条进行操作的相信大家应该非常了解了,这里详细解释下对窗口进行操作的一些细节:

窗口:也就是可以进行填充的区域,液晶驱动里面,每个物理像素对应的坐标已经是固定的,但是窗口可以不固定,窗口就是可以进行填充的区域,你如果要在窗口外面

进行填充,是无法进行的,同样的,当你填充到窗口边缘的时候,会自动跳转到下一行进行填充,只要你设定的点正确,那么整个你设定的窗口区域都会被填充完毕,这段期间

你要做的知识单纯的填充数据,不需要进行设定点的操作,也不需要换行,这样子屏幕填充矩阵操作看起来效果就不会有刷屏的感觉了.

填充行:对行进行填充,只需要在换行的时候进行坐标切换,我用整个函数,慢了30万个点每秒把.

在优化的时候,我只是抛砖引玉的给大家介绍下怎么用什么样的思路进行优化,细节性的东西还是要大家好好去琢磨的.

初稿到这边就差不多结束了,后面会陆续补充,只要大家想知道都可以直接进行联系.



[好文要顶](#)[关注我](#)[收藏该文](#)

[xmj](#)
[关注 - 13](#)
[粉丝 - 12](#)
[+加关注](#)

0

推荐

0

反对

« 上一篇: [uC/OS-II源码分析 \(一\)](#)
» 下一篇: [关于Socket 多线程 的一篇好文章](#)

posted on 2018-03-10 19:44 [xmj](#) 阅读(3507) 评论(1) [编辑](#) [收藏](#)


发表评论

#1楼 2019-08-12 17:24 | tianqi911

MARK

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)， [访问 网站首页](#)。

- 【推荐】腾讯云海外1核2G云服务器低至2折，半价续费券限量免费领取！
- 【活动】京东云服务器_云主机低于1折，低价高性能产品备战双11
- 【推荐】超50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
- 【培训】马士兵老师强势回归！Java线下课程全免费，双十一大促！
- 【推荐】天翼云双十一翼降到底，云主机11.11元起，抽奖送大礼
- 【福利】个推四大热门移动开发SDK全部免费用一年，限时抢！
- 【推荐】流程自动化专家UiBot，全套体系化教程成就高薪RPA工程师

- 相关博文:
- 基于STM32的uCGUI移植和优化
 - uCOSIII uCGUI STM32 平台移植
 - stm32 移植ucGUI
 - emWin 移植 - 基于红牛开发板

· ucGUI 移植详解
» 更多推荐...

最新 IT 新闻:

· 以魔鬼鱼为灵感 科学家研制新型航天器探索金星黑暗面
· 3个搞物理的颠覆了数学常识，数学天才陶哲轩：我开始压根不相信
· 5G、云计算、物联网与边缘计算的相辅相承
· 新研究有助揭示日冕高温之谜
· 对话联想童夫尧："超算+AI+5G"将成未来超算研究新方向
» 更多新闻...

历史上的今天:

2018-03-10 用libtommath实现RSA算法
2018-03-10 【算法25】对称子字符串的最大长度
2018-03-10 【linux+C】神器 vim + 指针相关客串
2018-03-10 设计并实现同时支持多种视频格式的流媒体点播系统
2018-03-10 递归再一次让哥震惊了
2018-03-10 算法系列15天速成——第十四天 图【上】
2018-03-10 Win32环境下代码注入与API钩子的实现