个人资料

xubin341719

访问： 537044次
积分： 5973
等级： BLOG 6
排名： 第1561名

原创： 69篇 转载： 3篇
译文： 0篇 评论： 402条

文章搜索

文章分类

S5PXX(三星) (30)
Ubuntun (2)
linux (15)
android (19)
camera (1)
ALSA (0)
WINCE (0)
协议 (15)
嵌入式入门 (10)
s3c2440 (5)

文章存档

2014年10月 (3)
2014年08月 (7)
2014年07月 (4)
2013年06月 (5)
2013年05月 (3)
2013年03月 (1)
2013年01月 (2)
2012年12月 (3)
2012年09月 (1)
2012年08月 (5)
2012年07月 (9)

Android LCD(三)：Samsung LCD接口篇

分类： S5PXX(三星) linux android 嵌入式入门 2013-06-26 14:27 8317人阅读 评论(3) 收藏 举报

关键词：**android LCD**控制器 **Framebuffer PWM**

平台信息:

内核：**linux2.6/linux3.0**

系统：**android/android4.0**

平台：**samsung exynos 4210**、**exynos 4412** 、**exynos 5250**

作者：**xubin341719(**欢迎转载，请注明作者**)**

欢迎指正错误，共同学习、共同进步！！

下载链接：**LCD**规格书（**404**份），之前工作用用到的 、 **LCD**规格书**00** 、 **LCD**规格书**01** 、 LCD测试图片，彩条灰阶等

参考：**S5PV210**显示驱动分析与移植（**android**）

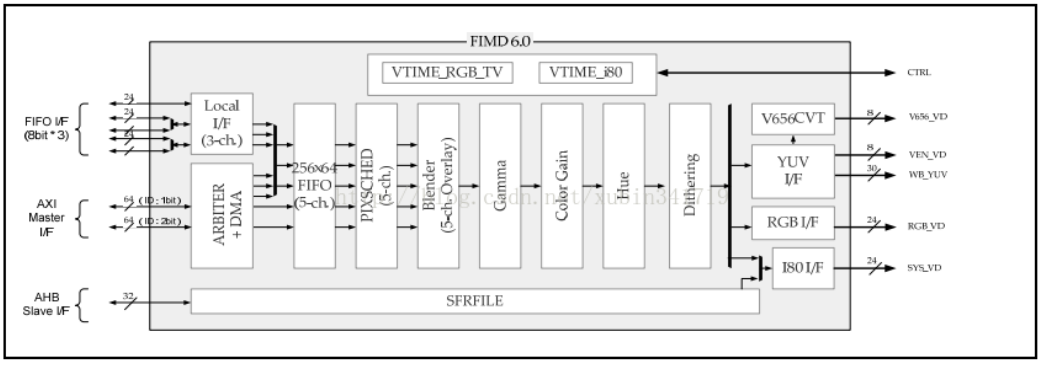
这篇文章中转载的成分比较多，不过大部分内容是从芯片手册上翻译过来。Framebuffer部分是黄冈老师--《嵌入式Linux之我行》这一系列博客中的，嵌入式Linux之我行这系列博客写的非常精，我刚学习Linux时经常拜读他的博客。这部分内容比较固定，三星的芯片跟新了好多代，不过这部分变化不大，技术是一个积累的过程，感谢那些前辈给我们整理比较好的学习资料，有比较好的技术继承。

这篇从LCD控制器、接口信号硬件接口 寄存器、Framebuffer 、接口函数的实现及寄存器的操作来讲解，同事补充两个知点：如何阅读LCD、PWM概述；

一、 **LCD**控制器

功能模块的实现其实是芯片里面集成了一个相应的控制器，比如IIC有IIC控制器，UART有UART控制器等，像其他功能模块一样LCD也有一个控制器，来实现图形信息的处理。LCD控制器可以通过编程支持不同LCD屏的要求，例如行和列像素数，数据总线宽度，接口时序和刷新频率等。LCD控制器的主要作用，是将定位在系统存储器中的显示缓冲区中的LCD图像数据传送到外部LCD驱动器，并产生必要的控制信号，例如RGB_VSYNC,RGB_HSYNC, RGB_VCLK等。

如下图所示，在Exynos4412规格书中截图，LCD控制器的构成。



2012年06月 (11)
2012年04月 (1)
2012年03月 (1)
2012年02月 (2)
2011年12月 (12)
2011年11月 (2)

阅读排行

android 电容屏（一）：
(28792)
android 电池（二）：an
(24779)
android camera(二)：摄
(24660)
android 电容屏（三）：
(22132)
android camera(四)：ca
(18475)
android HDMI(一)：HDI
(18184)
android camera(一)：ca
(17204)
Android 4.0 虚拟按键、
(16460)
android camera(三)：ca
(15784)
android 电容屏（二）：
(15588)

评论排行

android 电容屏（三）：(66)
android camera(四)：ca (45)
android HDMI(一)：HDI (21)
android camera(二)：摄 (20)
android 电池（三）：an (19)
android 电池（二）：an (16)
android电池（四）：电 (13)
蓝牙核心技术概述（四） (12)
android camera(三)：ca (12)
Android LCD(一)：LCD (11)

推荐文章

最新评论

Android BlueDroid（三）：Blue
xuexingyang: 说好的下一节呢。。。。都在等呢。。。。楼主加油啊～～～

蓝牙核心技术概述（三）：蓝牙
as453866908: 博主写的非常棒！ 帮我解决了很多疑惑，但是我这里用蓝牙有个问题就是蓝牙复位后我设置了使能Simple...

Android BlueDroid（一）：Blue
auxor: 我想在把Android手机变成蓝牙耳机设备，这个在技术上是否可行？我理解只要在手机上实现了Heads...

Android BlueDroid（一）：Blue
auxor: 我想在把Android手机变成蓝牙耳机设备，这个在技术上是否可行？我理解只要在手机上实现了Heads...

Android BlueDroid（一）：Blue
auxor: 看到了你写的Bluedroid概述文章，特来请教几个问题。我想在把Android手机变成蓝牙耳机设备...

android 电容屏（三）：驱动调
sunnyliang1: 楼主很棒！感谢！

Android bluetooth介绍（三）：
wi100sh: 多谢分享～

android camera(一)：camera模
u010423298: 楼主，求推荐相关

（下面这部分来自网络翻译，规格书中的描述）

主要由VSFR,VDMA, VPRCS , VTIME和视频时钟产生器几个模块组成：

（1）、VSFR由121个可编程控制器组，一套gamma LUT寄存器组（包括64个寄存器），一套i80命令寄存器组（包括12个寄存器）和5块256*32调色板存储器组成，主要用于对lcd控制器进行配置。

（2）、VDMA是LCD专用的DMA传输通道，可以自动从系统总线上获取视频数据传送到VPRCS，无需CPU干涉。

（3）、VPRCS收到数据后组成特定的格式（如16bpp或24bpp），然后通过数据接口（RGB_VD, VEN_VD, V656_VD or SYS_VD）传送到外部LCD屏上。

（4）、VTIME模块由可编程逻辑组成，负责不同lcd驱动器的接口时序控制需求。VTIME模块产生 RGB_VSYNC, RGB_HSYNC, RGB_VCLK, RGB_VDEN,VEN_VSYNC等信号。

主要特性：

（1）、支持4种接口类型：RGB/i80/ITU 601(656)/YTU444
（2）、支持单色、4级灰度、16级灰度、256色的调色板显示模式
（3）、支持64K和16M色非调色板显示模式
（4）、支持多种规格和分辨率的LCD
（5）、虚拟屏幕最大可达16MB
（6）、5个256*32位调色板内存
（7）、支持透明叠加

二、接口信号

FIMD显示控制器全部信号定义如下所示

Signal	I/O	Description	LCD Type
LCD_HSYNC	O	水平同步信号	RGB I/F
LCD_VSYNC	O	垂直同步信号	
LCD_VDEN	O	数据使能	
LCD_VCLK	O	视频时钟	
LCD_VD[23:0]	O	LCD像素数据输出	
SYS_OE	O	输出使能	
VSYNC_LDI	O	Indirect i80接口，垂直同步信号	i80 I/F
SYS_CS0	O	Indirect i80接口，片选LCD0	
SYS_CS1	O	Indirect i80接口，片选LCD1	
SYS_RS	O	Indirect i80接口，寄存器选择信号	
SYS_WE	O	Indirect i80接口，写使能信号	
SYS_VD[23:0]	IO	Indirect i80接口，视频数据输入输出	
SYS_OE	O	Indirect i80接口，输出使能信号	

VEN_HSYNC	O	601接口水平同步信号	ITU 601/656 I/F
VEN_VSYNC	O	601接口垂直同步信号	
VEN_HREF	O	601接口数据使能	
V601_CLK	O	601接口数据时钟	
VEN_DATA[7:0]	O	601接口YUV422格式数据输出	
V656_DATA[7:0]	O	656接口YUV422格式数据输出	
V656_CLK	O	656接口数据时钟	
VEN_FIELD	O	601接口域信号	

1、其中主要的**RGB**接口信号：

（**1**）、**LCD_HSYNC:**行同步信号，表示一行数据的开始，LCD控制器在整个水平线（整行）数据移入LCD驱动器后，插入一个LCD_HSYNC信号；

（**2**）、**LCD_VSYNC:** 帧同步信号，表示一帧数据的开始，LCD控制器在一个完整帧显示完成后立即插入一个LCD_VSYNC信号，开始新一帧的显示；VSYNC信号出现的频率表示一秒钟内能显示多少帧图像，称为“显示器的频率”

（**3**）、**LCD_VCLK**：像素时钟信号，表示正在传输一个像素的数据；

（**4**）、**LCD_VDEN:**数据使能信号；

（**5**）、 **LCD_VD[23:0]:** LCD像素数据输出端口

2、**RGB**信号的时序

下图是LCDRGB接口工作时序图：

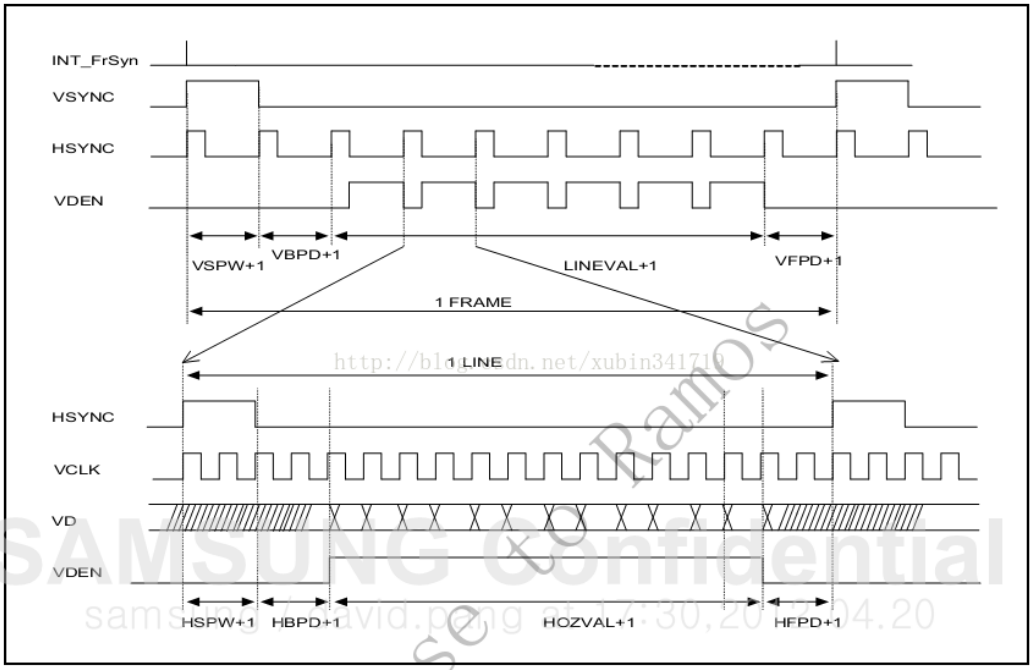


Figure 41-39 LCD RGB Interface Timing

（1）、上面时序图上各时钟延时参数的含义如下：这些配置可以在LCD规格书中查取

VBPD(vertical back porch)：表示在一帧图像开始时，垂直同步信号以后的无效的行数

VFBD(vertical front porch)：表示在一帧图像结束后，垂直同步信号以前的无效的行数VSPW(vertical sync pulse width)：表示垂直同步脉冲的宽度，用行数计算

HBPD(horizontal back porch)：表示从水平同步信号开始到一行的有效数据开始之间的VCLK的个

数HFPD(horizontal front porth)：表示一行的有效数据结束到下一个水平同步信号开始之间的VCLK的个数

HSPW(horizontal sync pulse width)：表示水平同步信号的宽度，用VCLK计算

（2）、帧的传输过程

VSYNC信号有效时，表示一帧数据的开始， 信号宽度为（VSPW +1）个HSYNC信号周期，即（VSPW +1）个无效行；

VSYNC信号脉冲之后，总共还要经过（VBPD+ 1）个HSYNC信号周期，有效的行数据才出现； 所以，在VSYNC信号有效之后，还要经过（VSPW +1 + VBPD + 1）个无效的行；

随即发出（**LINEVAL + 1**）行的有效数据；

最后是（**VFPD + 1**）个无效的行；

（3）、行中像素数据的传输过程

HSYNC信号有效时，表示一行数据的开始，信号宽度为（HSPW+ 1）个VCLK信号周期，即（HSPW +1）个无效像素；

HSYNC信号脉冲之后，还要经过（HBPD +1）个VCLK信号周期，有效的像素数据才出现；

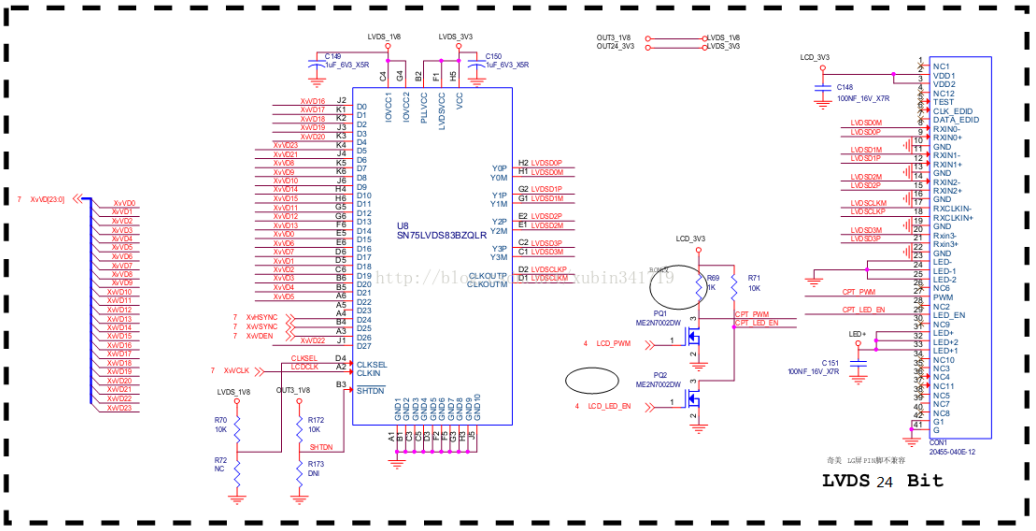
随后发出（HOZVAL+ 1）个像素的有效数据；

最后是（HFPD +1）个无效的像素；

（4）、将VSYNC、HSYNC、VCLK等信号的时间参数设置好之后，并将帧内存的地址告诉LCD控制器，它即可自动地发起DMA传输从帧内存中得到图像数据，最终在上述信号的控制下出现在数据总线VD[23:0]上。用户只需要把要显示的图像数据写入帧内存中。

其实现实的图像有像素点主城行、行组成场、场组成动画、动画叠加也就是3D的出现，也就是我们所说的“点动成线、线动成面、面动成体”。

三、**LCD**的硬件接口



1、**16M**（**24BPP**）色的显示模式

用24位的数据来表示一个像素的颜色，每种颜色使用8位。 LCD控制器从内存中获得某个像素的24为颜色值后，直接通过VD[23:0]数据线发送给LCD；在内存中，使用4个字节（32位）来表示一个像素，其中的3个字节从高到低分别表示红、绿、蓝，剩余的1个字节无效；

2、**64K**（**16BPP**）色的显示模式

用16位的数据来表示一个像素的颜色；格式又分为两种： 5：6：5 ——使用5位来表示红色，6位表示绿色，5位表示蓝色 ； 5：5：5：1——分别使用5位来表示红、绿、蓝，最后一位表示透明度；

3、16BPP

4、serialRGB

不同的BPP接线方式如下所示：

Table 41-5 Timing Reference Code (XY Definition)

–	Parallel RGB			Serial RGB	
	24 BPP (888)	18 BPP (666)	16 BPP (565)	24 BPP (888)	18 BPP (666)
VD[23]	R[7]	R[5]	R[4]	D[7]	D[5]
VD[22]	R[6]	R[4]	R[3]	D[6]	D[4]
VD[21]	R[5]	R[3]	R[2]	D[5]	D[3]
VD[20]	R[4]	R[2]	R[1]	D[4]	D[2]
VD[19]	R[3]	R[1]	R[0]	D[3]	D[1]
VD[18]	R[2]	R[0]	–	D[2]	D[0]
VD[17]	R[1]	–	–	D[1]	–
VD[16]	R[0]	–	–	D[0]	–
VD[15]	G[7]	G[5]	G[5]	–	–
VD[14]	G[6]	G[4]	G[4]	–	–
VD[13]	G[5]	G[3]	G[3]	–	–
VD[12]	G[4]	G[2]	G[2]	–	–
VD[11]	G[3]	G[1]	G[1]	–	–
VD[10]	G[2]	G[0]	G[0]	–	–
VD[9]	G[1]	–	–	–	–
VD[8]	G[0]	–	–	–	–
VD[7]	B[7]	B[5]	B[4]	–	–
VD[6]	B[6]	B[4]	B[3]	–	–
VD[5]	B[5]	B[3]	B[2]	–	–
VD[4]	B[4]	B[2]	B[1]	–	–
VD[3]	B[3]	B[1]	B[0]	–	–
VD[2]	B[2]	B[0]	–	–	–
VD[1]	B[1]	–	–	–	–
VD[0]	B[0]	–	–	–	–

四、寄存器

主要寄存器如下：

VIDCON0:配置视频输出格式，显示使能

VIDCON1:RGB 接口控制信号

VIDCON2: 输出数据格式控制

VIDCON3: 图像增强控制

I80IFCONx:i80接口控制信号

ITUIFCON: ITU接口控制信号

VIDTCONx:配置视频输出时序及显示大小

WINCONx:每个窗口特性设置

VIDOSDxA,B: 窗口位置设置

VIDOSDxC,D:OSD大小设置

五、**Framebuffer**驱动部分

这部分是：分析的比较好，我刚学linux的时候就拿个mini2440的板子对着他的博客练习)。其实这部分也是博主从S3c2440上分析的，三星芯片更新了这么多代，这块的原理还是不变的。就像一些协议一样，这么多年基本上不会变化，唯一出现的结果就是出来新的接口替代。LCD这块就是：TTL、LVDS、EDP、MIPI、HDMI等等.....速度更快，接线、PCB走线更简单，这就是集成化的好处。

1、简介

帧缓冲是Linux为显示设备提供的一个接口，它把一些显示设备描述成一个缓冲区，允许应用程序通过FrameBuffer定义好的接口访问这些图形设备，从而不用去关心具体的硬件细节。对于帧缓冲设备而言，只要在显示缓冲区与显示点对应的区域写入颜色值，对应的颜色就会自动的在屏幕上显示。下面来看一下在不同色位模式下缓冲区与显示点的对应关系：

8 位色显示缓冲区与显示点对应关系图

RGB			RGB		
7 - 5	4 - 2	1 - 0	7 - 5	4 - 2	1 - 0
R	G	B			

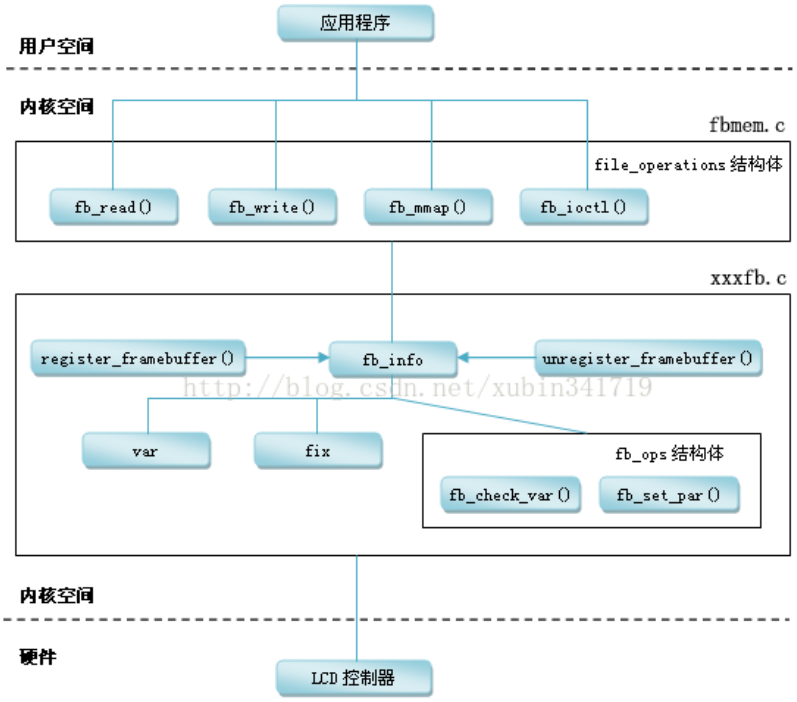
16 位色显示缓冲区与显示点对应关系图

位	15 - 11	10 - 5	4 - 0
RGB565	R	G	B
RGB555	R	G	B

2、驱动结构

帧缓冲设备为标准的字符型设备，在Linux中主设备号29，定义在/linux/major.h中的FB_MAJOR，次设备号定义帧缓冲的个数，最大允许有32个FrameBuffer，定义在/include/linux/fb.h中的FB_MAX，对应于文件系统下/dev/fb%d设备文件。

帧缓冲设备驱动在Linux子系统中的结构如下：



帧缓冲设备驱动程序结构图

我们从上面这幅图看，帧缓冲设备在Linux中也可以看做是一个完整的子系统，大体由fbmem.c和xxxfb.c（对应我们的s3cfb.c）组成。向上给应用程序提供完善的设备文件操作接口(即对FrameBuffer设备进行read、write、ioctl等操作)，接口在Linux提供的fbmem.c文件中实现；向下提供了硬件操作的接口，只是这些接口Linux并没有提供实现，因为这要根据具体的LCD控制器硬件进行设置，所以这就是我们要做的事情了(即s3cfb.c部分的实现)。

3、数据结构及接口函数

从帧缓冲设备驱动程序结构看，该驱动主要跟fb_info结构体有关，该结构体记录了帧缓冲设备的全部信息，包括设备的设置参数、状态以及对底层硬件操作的函数指针。在Linux中，每一个帧缓冲设备都必须对应一个fb_info，fb_info在/linux/fb.h中的定义如下：(只列出重要的一些)

```
struct fb_info {
    int node;
    int flags;
    struct fb_var_screeninfo var;/*LCD可变参数结构体*/
    struct fb_fix_screeninfo fix;/*LCD固定参数结构体*/
    struct fb_monspecs monspecs; /*LCD显示器标准*/
    struct work_struct queue;    /*帧缓冲事件队列*/
    struct fb_pixmap pixmap;     /*图像硬件mapper*/
    struct fb_pixmap sprite;     /*光标硬件mapper*/
    struct fb_cmap cmap;         /*当前的颜色表*/
    struct fb_videomode *mode;   /*当前的显示模式*/
#ifdef CONFIG_FB_BACKLIGHT
    struct backlight_device *bl_dev;/*对应的背光设备*/
    struct mutex bl_curve_mutex;
    u8 bl_curve[FB_BACKLIGHT_LEVELS];/*背光调整*/
#endif
#ifdef CONFIG_FB_DEFERRED_IO
    struct delayed_work deferred_work;
    struct fb_deferred_io *fbdefio;
#endif
    struct fb_ops *fbops; /*对底层硬件操作的函数指针*/
    struct device *device;
    struct device *dev;   /*fb设备*/
    int class_flag;
```

```
#ifdef CONFIG_FB_TILEBLITTING
    struct fb_tile_ops *tileops; /*图块Blitting*/
#endif
    char __iomem *screen_base;   /*虚拟基地址*/
    unsigned long screen_size;   /*LCD IO映射的虚拟内存大小*/
    void *pseudo_palette;        /*伪16色颜色表*/
#define FBINFO_STATE_RUNNING    0
#define FBINFO_STATE_SUSPENDED  1
    u32 state;  /*LCD的挂起或恢复状态*/
    void *fbcon_par;
    void *par;
};
```

其中，比较重要的成员有struct fb_var_screeninfo var、structfb_fix_screeninfo fix和struct fb_ops *fbops，他们也都是结构体。

fb_var_screeninfo结构体主要记录用户可以修改的控制器的参数，比如屏幕的分辨率和每个像素的比特数等，该结构体定义如下：

```
struct fb_var_screeninfo {
    __u32 xres;                 /*可见屏幕一行有多少个像素点*/
    __u32 yres;                 /*可见屏幕一列有多少个像素点*/
    __u32 xres_virtual;         /*虚拟屏幕一行有多少个像素点*/
    __u32 yres_virtual;         /*虚拟屏幕一列有多少个像素点*/
    __u32 xoffset;              /*虚拟到可见屏幕之间的行偏移*/
    __u32 yoffset;              /*虚拟到可见屏幕之间的列偏移*/
    __u32 bits_per_pixel;       /*每个像素的位数即BPP*/
    __u32 grayscale;            /*非0时，指的是灰度*/
    struct fb_bitfield red;     /*fb缓存的R位域*/
    struct fb_bitfield green;   /*fb缓存的G位域*/
    struct fb_bitfield blue;    /*fb缓存的B位域*/
    struct fb_bitfield transp;  /*透明度*/
    __u32 nonstd;               /* != 0 非标准像素格式*/
    __u32 activate;
    __u32 height;               /*高度*/
    __u32 width;                /*宽度*/
    __u32 accel_flags;
    /*定时：除了pixclock本身外，其他的都以像素时钟为单位*/
    __u32 pixclock;             /*像素时钟(皮秒)*/
    __u32 left_margin;          /*行切换，从同步到绘图之间的延迟*/
    __u32 right_margin;         /*行切换，从绘图到同步之间的延迟*/
    __u32 upper_margin;         /*帧切换，从同步到绘图之间的延迟*/
    __u32 lower_margin;         /*帧切换，从绘图到同步之间的延迟*/
    __u32 hsync_len;            /*水平同步的长度*/
    __u32 vsync_len;            /*垂直同步的长度*/
    __u32 sync;
    __u32 vmode;
    __u32 rotate;
    __u32 reserved[5];          /*保留*/
};
```

而fb_fix_screeninfo结构体又主要记录用户不可以修改的控制器的参数，比如屏幕缓冲区的物理地址和长度等，该结构体的定义如下：

```
struct fb_fix_screeninfo {
    char id[16];                /*字符串形式的标示符 */
    unsigned long smem_start;   /*fb缓存的开始位置 */
```

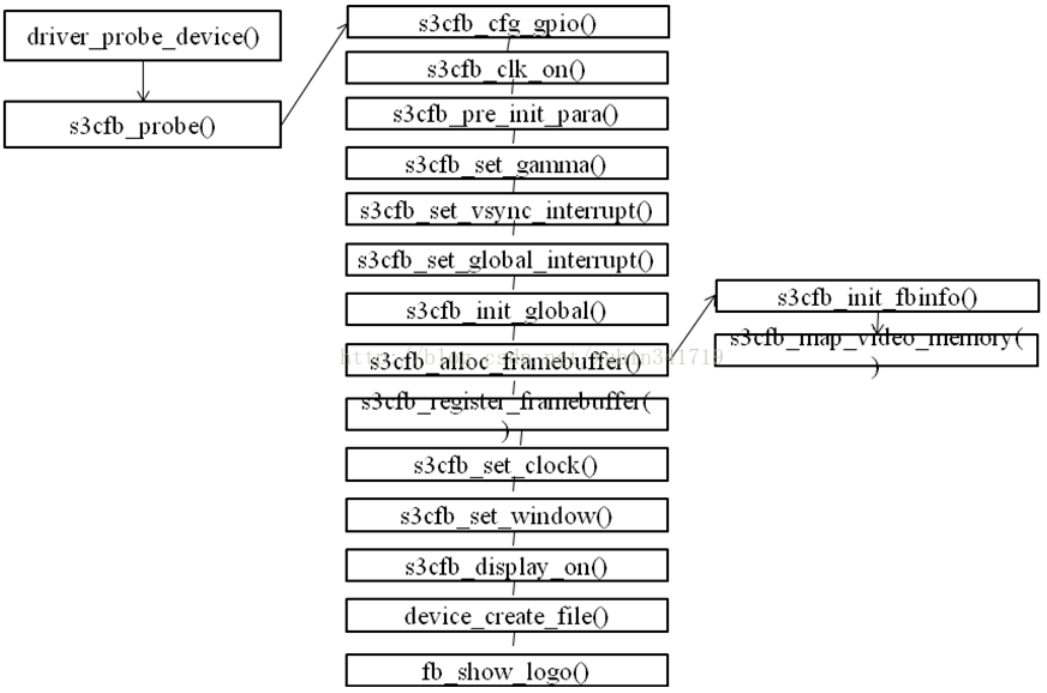
```
    __u32 smem_len;             /*fb缓存的长度 */
    __u32 type;                 /*看FB_TYPE_* */
    __u32 type_aux;             /*分界*/
    __u32 visual;               /*看FB_VISUAL_* */
    __u16 xpanstep;             /*如果没有硬件panning就赋值为0 */
    __u16 ypanstep;             /*如果没有硬件panning就赋值为0 */
    __u16 ywrapstep;            /*如果没有硬件ywrap就赋值为0 */
    __u32 line_length;          /*一行的字节数 */
    unsigned long mmio_start;   /*内存映射IO的开始位置*/
    __u32 mmio_len;             /*内存映射IO的长度*/
    __u32 accel;
    __u16 reserved[3];          /*保留*/
};
```

fb_ops结构体是对底层硬件操作的函数指针，该结构体中定义了对硬件的操作有:(这里只列出了常用的操作)

```
struct fb_ops {
    struct module *owner;
    //检查可变参数并进行设置
    int (*fb_check_var)(struct fb_var_screeninfo *var, struct fb_info *info);
    //根据设置的值进行更新，使之有效
    int (*fb_set_par)(struct fb_info *info);
    //设置颜色寄存器
    int (*fb_setcolreg)(unsigned regno, unsigned red, unsigned green,
            unsigned blue, unsigned transp, struct fb_info *info);
    //显示空白
    int (*fb_blank)(int blank, struct fb_info *info);
    //矩形填充
    void (*fb_fillrect) (struct fb_info *info, const struct fb_fillrect *rect);
    //复制数据
    void (*fb_copyarea) (struct fb_info *info, const struct fb_copyarea *region);
    //图形填充
    void (*fb_imageblit) (struct fb_info *info, const struct fb_image *image);
};
```

六、**Framebuffer**设备注册

S3cfb.c中的s3cfb_probe设备探测，是驱动注册的主要函数，



```
/*定义一个结构体用来维护驱动程序中各函数中用到的变量
 先别看结构体要定义这些成员，到各函数使用的地方就明白了*/

static int __devinit s3cfb_probe(struct platform_device *pdev)
{
        struct s3c_platform_fb *pdata;/*LCD屏配置信息结构体*/
        struct s3cfb_global *fbdev;/*驱动程序全局变量结构体*/
        struct resource *res; /*用来保存从LCD平台设备中获取的LCD资源*/
        int i, j, ret = 0;

        printk("%s\n",__func__);
        fbdev = kzalloc(sizeof(struct s3cfb_global), GFP_KERNEL);
        if (!fbdev) {
                dev_err(&pdev->dev, "failed to allocate for "
                        "global fb structure\n");
                ret = -ENOMEM;
                goto err_global;
        }
        fbdev->dev = &pdev->dev;

        fbdev->regulator = regulator_get(&pdev->dev, "pd");
        if (!fbdev->regulator) {
                dev_err(fbdev->dev, "failed to get regulator\n");
                ret = -EINVAL;
                goto err_regulator;
        }
        ret = regulator_enable(fbdev->regulator);
        if (ret < 0) {
                dev_err(fbdev->dev, "failed to enable regulator\n");
                ret = -EINVAL;
                goto err_regulator;
        }

        /*获取LCD参数信息*/
        pdata = to_fb_plat(&pdev->dev);
        if (!pdata) {
                dev_err(fbdev->dev, "failed to get platform data\n");
                ret = -EINVAL;
                goto err_pdata;
        }

        fbdev->lcd = (struct s3cfb_lcd *)pdata->lcd;

        /*配置GPIO端口*/
        if (pdata->cfg_gpio)
                pdata->cfg_gpio(pdev);

        /*设置时钟参数*/
        if (pdata->clk_on)
                pdata->clk_on(pdev, &fbdev->clock);
```

```
        /*获取LCD平台设备所使用的IO端口资源，注意这个IORESOURCE_MEM标志和LCD平台设备定义中的一致*/
        res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
        if (!res) {
                dev_err(fbdev->dev, "failed to get io memory region\n");
                ret = -EINVAL;
                goto err_io;
        }

        /*申请LCD IO端口所占用的IO空间(注意理解IO空间和内存空间的区别),request_mem_region定义在
ioport.h中*/
        res = request_mem_region(res->start,
                                 res->end - res->start + 1, pdev->name);
        if (!res) {
                dev_err(fbdev->dev, "failed to request io memory region\n");
                ret = -EINVAL;
                goto err_io;
        }

        /*将LCD的IO端口占用的这段IO空间映射到内存的虚拟地址，ioremap定义在io.h中
                 注意：IO空间要映射后才能使用，以后对虚拟地址的操作就是对IO空间的操作*/
        fbdev->regs = ioremap(res->start, res->end - res->start + 1);
        if (!fbdev->regs) {
                dev_err(fbdev->dev, "failed to remap io region\n");
                ret = -EINVAL;
                goto err_mem;
        }
#ifdef CONFIG_FB_S3C_LTE480WV
        /*设置寄存器初始状态*/
        s3cfb_pre_init_para(fbdev);
#endif

    /*设置gamma 值*/
        s3cfb_set_gamma(fbdev);
        /*设置VSYNC中断*/
        s3cfb_set_vsync_interrupt(fbdev, 1);
        /*设置全局中断*/
        s3cfb_set_global_interrupt(fbdev, 1);
        /*fb设备参数信息初始化*/
        s3cfb_init_global(fbdev);

        /*为framebuffer分配空间，进行内存映射，填充fb_info*/
        if (s3cfb_alloc_framebuffer(fbdev)) {
                ret = -ENOMEM;
                goto err_alloc;
        }

        /*注册fb设备到系统中*/
        if (s3cfb_register_framebuffer(fbdev)) {
                ret = -EINVAL;
                goto err_register;
        }

        s3cfb_set_clock(fbdev);
```

```
        s3cfb_set_window(fbdev, pdata->default_win, 1);

        s3cfb_display_on(fbdev);

        fbdev->irq = platform_get_irq(pdev, 0);
        if (request_irq(fbdev->irq, s3cfb_irq_frame, IRQF_SHARED,
                        pdev->name, fbdev)) {
                dev_err(fbdev->dev, "request_irq failed\n");
                ret = -EINVAL;
                goto err_irq;
        }

#ifdef CONFIG_FB_S3C_LCD_INIT
        if (pdata->backlight_on)
                pdata->backlight_on(pdev);

        if (!bootloaderfb && pdata->reset_lcd)
                pdata->reset_lcd(pdev);

        if (pdata->lcd_on)
                pdata->lcd_on(pdev);
#endif

#ifdef CONFIG_HAS_EARLYSUSPEND
        fbdev->early_suspend.suspend = s3cfb_early_suspend;
        fbdev->early_suspend.resume = s3cfb_late_resume;
        fbdev->early_suspend.level = EARLY_SUSPEND_LEVEL_DISABLE_FB;
        register_early_suspend(&fbdev->early_suspend);
#endif

        /*对设备文件系统的支持，创建fb设备文件*/
        ret = device_create_file(&(pdev->dev), &dev_attr_win_power);
        if (ret < 0)
                dev_err(fbdev->dev, "failed to add sysfs entries\n");

        dev_info(fbdev->dev, "registered successfully\n");

        /*显示开机logo*/
#if !defined(CONFIG_FRAMEBUFFER_CONSOLE) && defined(CONFIG_LOGO)
        if (fb_prepare_logo( fbdev->fb[pdata->default_win], FB_ROTATE_UR)) {
                printk("Start display and show logo\n");
                /* Start display and show logo on boot */
                fb_set_cmap(&fbdev->fb[pdata->default_win]->cmap, fbdev->fb[pdata->default_win]);
                fb_show_logo(fbdev->fb[pdata->default_win], FB_ROTATE_UR);
        }
#endif

        return 0;
}
```

七、如何阅读**LCD**规格书

首先我们调试LCD的时候要获得的一些参数，没必要把整个规格书通读一遍，我刚开始调试屏的时候拿到一个规格书不知道从何入手，也不知那些参数有用，比较模糊，其实只提取一些有用的信息就可以，下面这些对初学者

也许有点用处。

1、GeneralSpecification

尺寸、分辨率、位数、色彩、像素时钟频率、接口类型

（**1**）、尺寸：

Module	**10.1"(10.1") WUXGA 16:10 Color TFT-LCD with LED Backlight design**

（2）、分辨率：1920 1200；

Pixels H x V		1,920x3(RGB) x 1,200

（3）、接口：双通道LVDS；

Electrical Interface		2 channel LVDS

（4）、色彩：16.7M，这里可以确认数据位数8bitRGB三色：3*8=24，2的24次方=16.7M

6bitRGB 三色：3*6=18，2的18次方=262 144;

所以当看到色彩是1.7M是，说明LCD是24bit的，如果是262 144说明LCD是18bit的。

Support Color		16.7M colors (RGB 8-bits)

2、Timing Characteristics

Parameter		Symbol	Min.	Typ.	Max.	Unit
Frame Rate		---	--	60	---	Hz
Clock frequency		1/ T_{Clock}	64	76.36	85	MHz
Vertical Section	Period	T_V	1210	1212	1240	T_{Line}
	Active	T_{VD}	1200			
	Blanking	T_{VB}	10	12	40	
Horizontal Section	Period	T_H	1034	1050	1140	T_{Clock} (Note 2)
	Active	T_{HD}	960			
	Blanking	T_{HB}	74	90	180	

（1）、Frame rate :是60HZ，也就是帧率；

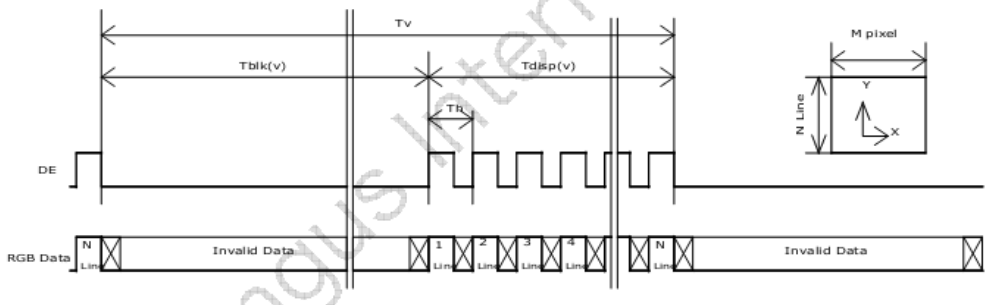
（2）、clock frequency:像素时钟，这里面有最大值、中间值和最小值，这个屏默认值为：76.36MHz;

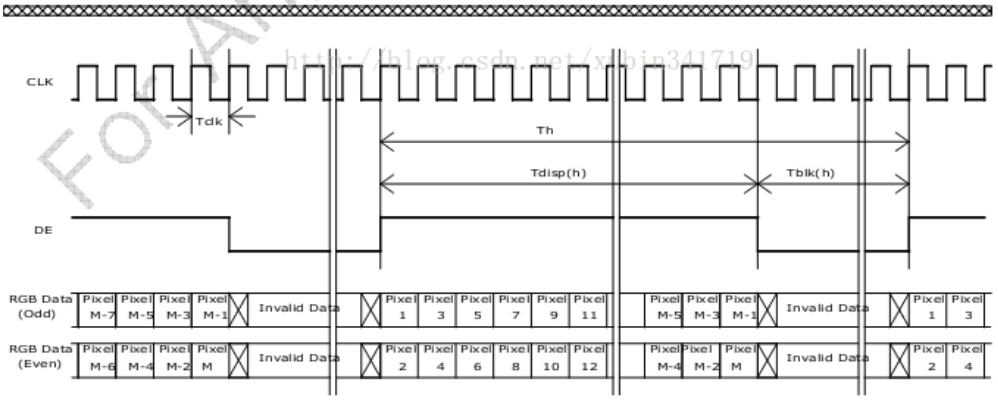
（3）、Vertical Seciton：VSWidth +Back Porc+Front Porch，前间距、后间距。这个我们再RGB信号哪里详细解释，这个我们前面有说过；

（4）、Horizontal Section：HS Width +Back Porc+Front Porch，这个跟VS的Porch相同。

3、LCD Timing diagram信号时序图，如下所示

有些读者会问，为什么没有行、场、数据等信号。其实这个是LVDS信号的时序，这个根据屏厂的习惯，有的画的是LVDS输入的信号时序，有的是TTL（RGB）的时序。





上面我们以一个例子说明，做驱动的（软件方面）要知道的一些参数，如果是硬件方面的问题，可以再对一下接口。其实一个LCD规格书要了解的也就这么多，调试软件就够用：

（1）、**General Specification**中可得到，尺寸、分辨率、位数、色彩、像素时钟频率、接口类型；

（2）、Timing Characteristics中可以得到一些具体的参数；

（3）、LCD Timing diagram信号时序图，可以看到一些信号的时序、极性等；

八、**PWM**概述

1、先解释两个名词：

PWM：脉冲宽度调制(PWM)，是英文“Pulse WidthModulation”的缩写，简称脉宽调制。

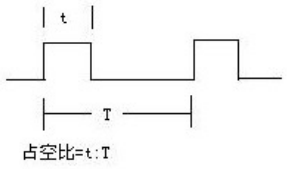
占空比：占空比(DutyRation)在电信领域中有如下含义：

在一串理想的脉冲周期序列中（如方波），正脉冲的持续时间与脉冲总周期的比值。例如：（假设脉冲为3V）

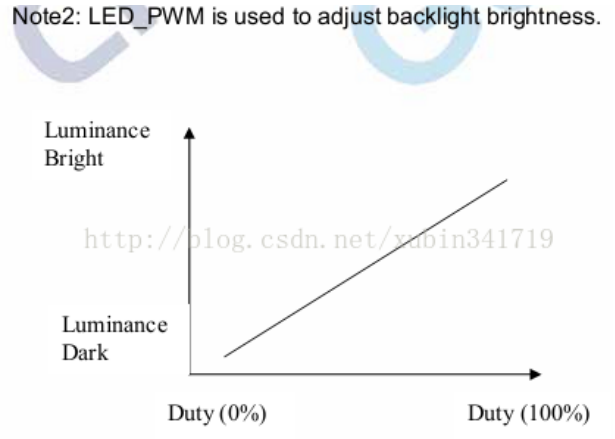
脉冲宽度 1μs，信号周期4μs的脉冲序列占空比为0.25，平均电压为：3*0.25=0.75V；

脉冲宽度 0μs，信号周期4μs的脉冲序列占空比为0，平均电压为：0V；

脉冲宽度 4μs，信号周期4μs的脉冲序列占空比为1，平均电压为：3V；



平均电压的变化成阶梯型变化，如果T足够小，成线性。



看下芯片片规格书中的描述：寄存器填不同值是，脉冲宽度不一样。

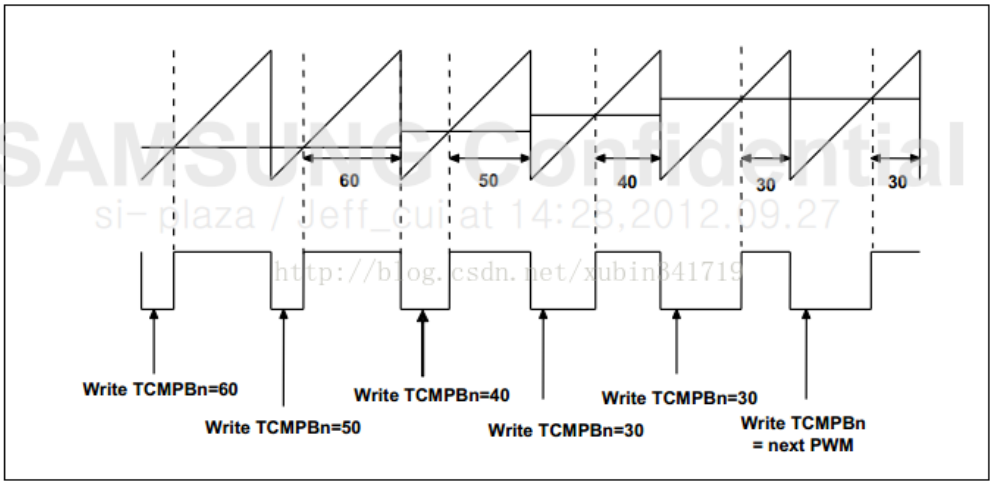


Figure 23-6 PWM

2、samusng 中的PWM控制器

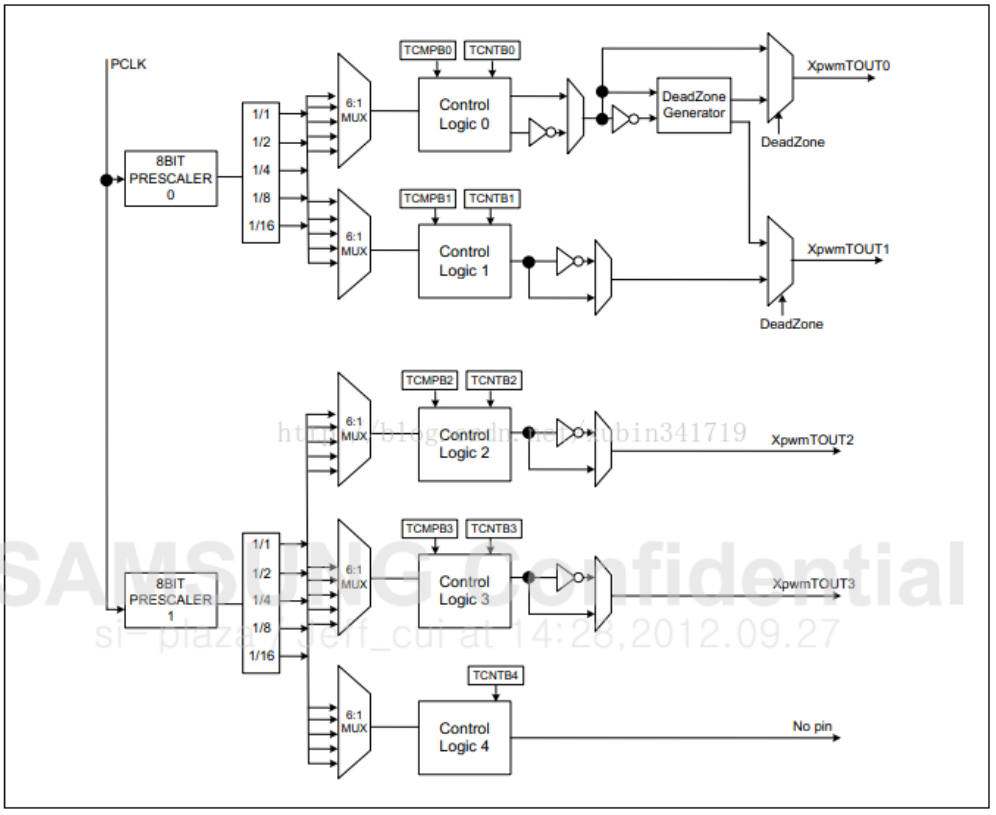


Figure 23-2 PWM TIMER Clock Tree Diagram

PWM时钟分频。跟单片机里面的有点像。死区控制器：这个是根据晶体管的特性，设置这个功能的，不过我工作中还没有用到死区控制这块。了解有这个概念。

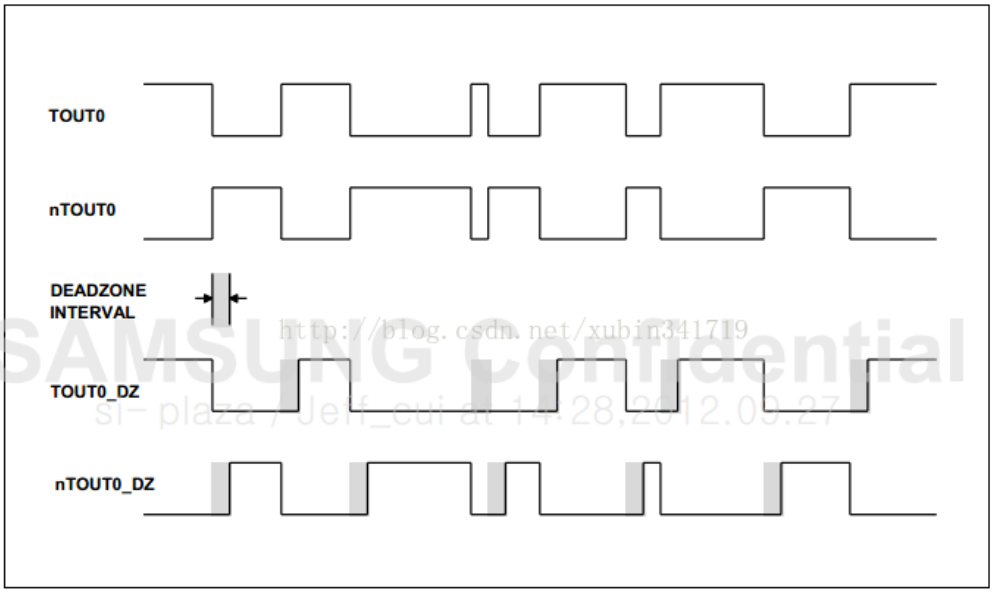


Figure 23-8 Waveform when a Dead Zone Feature is Enabled

看这些寄存器，记得用MINI2440写裸机程序的时候，直接写这些寄存器，记得上学时把s3c2440当单片机玩，有点浪费。学生时代，已经逝去的青春？？

23.5.1 Register Map Summary

- Base Address: 0x12DD_0000 (PWM)
- Base Address: 0x1316_0000 (PWM_ISP)

Register	Offset	Description	Reset Value
TCFG0	0x0000	Specifies the Timer Configuration register 0 that configures the two 8-bit prescaler and dead zone length	0x0000_0101
TCFG1	0x0004	Specifies the Timer Configuration register 1 that controls 5 MUX select bit	0x0000_0000
TCON	0x0008	Specifies the Timer Control register.	0x0000_0000
TCNTB0	0x000C	Specifies the Timer 0 Count Buffer register	0x0000_0000
TCMPB0	0x0010	Specifies the Timer 0 Compare Buffer register	0x0000_0000
TCNTO0	0x0014	Specifies the Timer 0 Count Observation register	0x0000_0000
TCNTB1	0x0018	Specifies the Timer 1 Count Buffer register	0x0000_0000
TCMPB1	0x001C	Specifies the Timer 1 Compare Buffer register	0x0000_0000
TCNTO1	0x0020	Specifies the Timer 1 Count Observation register	0x0000_0000
TCNTB2	0x0024	Specifies the Timer 2 Count Buffer register	0x0000_0000
TCMPB2	0x0028	Specifies the Timer 2 Compare Buffer register	0x0000_0000
TCNTO2	0x002C	Specifies the Timer 2 Count Observation register	0x0000_0000
TCNTB3	0x0030	Specifies the Timer 3 Count Buffer register	0x0000_0000
TCMPB3	0x0034	Specifies the Timer 3 Compare Buffer register	0x0000_0000
TCNTO3	0x0038	Specifies the Timer 3 Count Observation register	0x0000_0000
TCNTB4	0x003C	Specifies the Timer 4 Count Buffer register	0x0000_0000
TCNTO4	0x0040	Specifies the Timer 4 Count Observation register	0x0000_0000
TINT_CSTAT	0x0044	Specifies the Timer Interrupt Control and Status register	0x0000_0000

上一篇 Android LCD(二)：LCD常用接口原理篇
下一篇 Android LCD(四)：LCD驱动调试篇