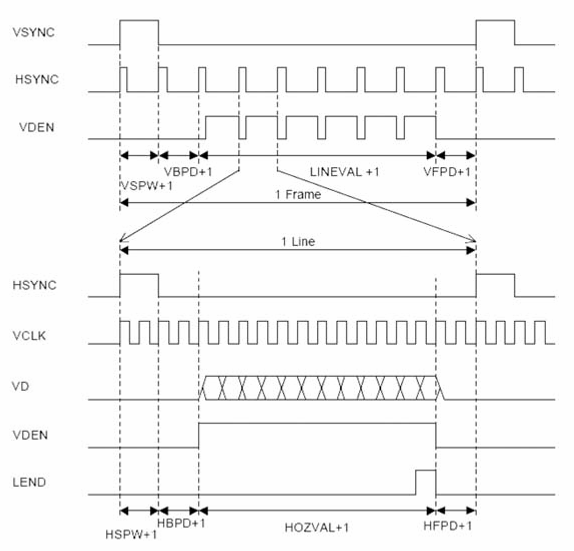
LCD设备驱动

1. LCD硬件原理

a、利益液晶制成的显示器称为LCD，嵌入式系统中主要用TFT屏。

b、TFT屏工作时序图：



VCLK，VSYNC，HSYNC都是像素时钟信号。

VSYNC:帧同步信号

HSYNC:行同步信号

VDEN:数据有效标志位

VD:图像数据信号

每发一个行同步脉冲，表示开始传输一行新的数据，每发一个帧同步脉冲，意味着新的一屏数据信号开始发送。

对于数据的传输可以理解为，一帧（即一整屏幕），一行，一点。数据的传输是一个震荡周期传输一个点，一个行周期传一行，然后是一个帧脉冲开始传一屏。

C、回扫时间，行切换和帧切换的回扫时间。

左右边界为行切换需要的回扫时间，上下边界为帧切换需要的回扫时间，水平同步和垂直同步为行和帧同步本身需要的时间。Xres,yres指屏幕的水平和垂直分辨率。

上边界

左边界

下边界

右边界

水平同步

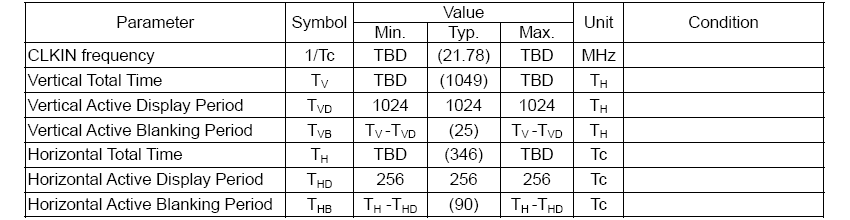
垂直同步

yres

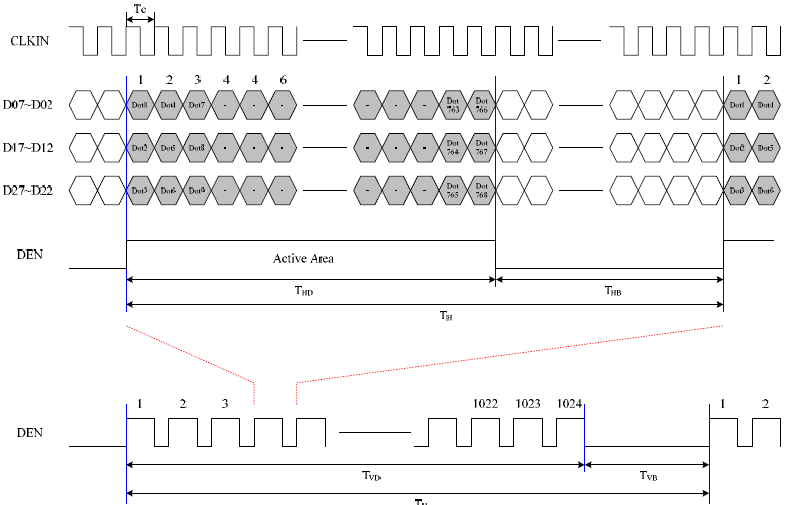
xres

d、以我们TFT-LCD中需要了解到的硬件参数描述为例。

分辨率：768\*1024（256\*3=768）



Spec中的时序图。



了解硬件参数需要根据具体不同的硬件设置参数。

1. 帧缓冲（framebuffer）

帧缓冲是linux系统为显示设备提供的借口，将显示缓冲区抽象。帧缓冲设备为标准的字符设备，因此我们可以参加一般字符设备驱动的架构来理解。上层应用程序只需要往帧缓冲设备的显示缓冲区中写入与显示点对应的区域的颜色值，对应的颜色就会显示在屏幕上。

所以对于LCD驱动的理解，主要部分就是对帧缓冲设备的操作。

1. 帧缓冲设备驱动架构：

Pxafb.c

Fbmem.c

注册帧缓冲设备驱动，register\_framebuffer.

根据lcd屏幕特点，初始化lcd控制器

申请帧缓冲设备显示缓冲区空间

应用程序

LCD控制器

fb\_read

fb\_write

fb\_mmap

fb\_ioctl

Pxafb.c

File\_operations

1. 理解帧缓冲设备需要了解的几个重要结构体，也就是在注册驱动时需要用到的几个结构体

fb\_info结构体（简称FBI）

FBI是帧缓冲设备中最关键的一个结构体，包括了帧缓冲设备属性和操作的完整性

struct fb\_info {

int node;

int flags;

struct fb\_var\_screeninfo var; /\* Current var \*///可变参数

struct fb\_fix\_screeninfo fix; /\* Current fix \*///固定参数

struct fb\_monspecs monspecs; /\* Current Monitor specs \*/

struct work\_struct queue; /\* Framebuffer event queue \*/

struct fb\_pixmap pixmap; /\* Image hardware mapper \*/

struct fb\_pixmap sprite; /\* Cursor hardware mapper \*/

struct fb\_cmap cmap; /\* Current cmap \*/

struct list\_head modelist; /\* mode list \*/

struct fb\_videomode \*mode; /\* current mode \*/

#ifdef CONFIG\_FB\_BACKLIGHT

/\* assigned backlight device \*/

/\* set before framebuffer registration,

remove after unregister \*/

struct backlight\_device \*bl\_dev;

/\* Backlight level curve \*/

struct mutex bl\_curve\_mutex;

u8 bl\_curve[FB\_BACKLIGHT\_LEVELS];

#endif

struct fb\_ops \*fbops;

struct device \*device; /\* This is the parent \*/

struct device \*dev; /\* This is this fb device \*/

int class\_flag; /\* private sysfs flags \*/

#ifdef CONFIG\_FB\_TILEBLITTING

struct fb\_tile\_ops \*tileops; /\* Tile Blitting \*/

#endif

char \_\_iomem \*screen\_base; /\* Virtual address \*/

unsigned long screen\_size; /\* Amount of ioremapped VRAM or 0 \*/

void \*pseudo\_palette; /\* Fake palette of 16 colors \*/

#define FBINFO\_STATE\_RUNNING 0

#define FBINFO\_STATE\_SUSPENDED 1

u32 state; /\* Hardware state i.e suspend \*/

void \*fbcon\_par; /\* fbcon use-only private area \*/

/\* From here on everything is device dependent \*/

void \*par;

};

其中，fb\_ops，是指向硬件底层操作的函数指针。他们最终适合LCD控制器硬件打交道

static struct fb\_ops pxafb\_ops = {

.owner = THIS\_MODULE,

.fb\_check\_var = pxafb\_check\_var, //用于检查可以修改的屏幕参数并调整到合适的值

.fb\_set\_par = pxafb\_set\_par, //是的用户设置的屏幕参数在硬件上有效。

.fb\_setcolreg = pxafb\_setcolreg,

.fb\_fillrect = cfb\_fillrect,//定义在cfbfillrect.c

.fb\_copyarea = cfb\_copyarea,//定义在cfbcopyarea.c

.fb\_imageblit = cfb\_imageblit,//定义在cfbimgblt.c

.fb\_blank = pxafb\_blank,

.fb\_mmap = pxafb\_mmap,

#ifdef CONFIG\_FB\_PXA\_MINILCD

.fb\_ioctl = pxafb\_minilcd\_ioctl,

#endif

};

fb\_var\_screeninfo结构体记录用户可修改的显示控制器参数，如分辨率，每个点的比特数。

fb\_fix\_screeninfo结构体记录用户不能修改的显示控制器参数，如屏幕的缓冲区物理地址，长度。

Fb\_cmap结构体，记录设备无关的颜色表信息。用户可以通过FBIOGETCMAP，FBIOSETCMAP读取和设定颜色表值。

C、文件操作函数集

File\_operation（在fbmem.c中实现，驱动程序一般不需要修改）



帧缓冲设备驱动的模块加载卸载

模块加载的工作应该有：

1. 申请FBI结构体的内存空间，初始化FBI结构中的固定和可变屏幕参数，
2. 根据具体的LCD特点完成硬件LCD控制器硬件的初始化、
3. 申请帧缓冲设备的显示缓冲区空间
4. 注册帧缓冲设备、

以我们的driver code中的实现为例：

int \_\_init pxafb\_probe(struct platform\_device \*dev)

{

struct pxafb\_info \*fbi;//其中包含fb\_info结构体。

struct pxafb\_mach\_info \*inf;

……

fbi = pxafb\_init\_fbinfo(&dev->dev);//申请FBI，初始化。

……

/\* Initialize video memory \*/

ret = pxafb\_map\_video\_memory(fbi);//显示缓冲区的申请

……

/\*

\* This makes sure that our colour bitfield

\* descriptors are correctly initialised.

\*/

pxafb\_check\_var(&fbi->fb.var, &fbi->fb);

pxafb\_set\_par(&fbi->fb);

platform\_set\_drvdata(dev, fbi);

ret = register\_framebuffer(&fbi->fb);//注册framebuffer

if (ret < 0) {

dev\_err(&dev->dev, "Failed to register framebuffer device: %d\n", ret);

goto failed;

}

……

#ifdef CONFIG\_FB\_PXA\_MINILCD//如果有mini lcd则注册minilcd

pxafb\_minilcd\_register(&fbi->fb);

#endif

set\_ctrlr\_state(fbi, C\_ENABLE);// now enable the LCD controller

……

}

卸载则是释放FBI，关闭LCD，释放显示缓冲区以及注销设备。

1. 帧缓冲设备显示缓冲区的申请。

显示缓冲区的大小= max\_res（最大x解析度） \* max\_res \*max\_ BPP(bit per pixel)

嵌入式中，是直接在RAM空间分配一段显示缓冲区。

static int \_\_init pxafb\_map\_video\_memory(struct pxafb\_info \*fbi)

{

u\_long palette\_mem\_size;

#if 0

fbi->map\_size = PAGE\_ALIGN(fbi->fb.fix.smem\_len + PAGE\_SIZE);

#else

/\* we reserve 3 pages after the framebuffer for occasionally out

\* of boundary access

\*/

fbi->map\_size = PAGE\_ALIGN(fbi->fb.fix.smem\_len + 4 \* PAGE\_SIZE);//显示缓冲区大小

#endif

fbi->map\_cpu = dma\_alloc\_writecombine(fbi->dev, fbi->map\_size,

&fbi->map\_dma, GFP\_KERNEL);//分配内存

if (fbi->map\_cpu) {

/\* prevent initial garbage on screen \*/

memset(fbi->map\_cpu, 0, fbi->map\_size);

fbi->fb.screen\_base = fbi->map\_cpu + PAGE\_SIZE;

fbi->screen\_dma = fbi->map\_dma + PAGE\_SIZE;

fbi->fb.fix.smem\_start = fbi->screen\_dma;//帧缓冲设备显示缓冲区的首地址

fbi->palette\_size = fbi->fb.var.bits\_per\_pixel == 8 ? 256 : 16;

palette\_mem\_size = fbi->palette\_size \* sizeof(u16);

pr\_debug("pxafb: palette\_mem\_size = 0x%08lx\n", palette\_mem\_size);

fbi->palette\_cpu = (u16 \*)(fbi->map\_cpu + PAGE\_SIZE - palette\_mem\_size);

fbi->palette\_dma = fbi->map\_dma + PAGE\_SIZE - palette\_mem\_size;

}

return fbi->map\_cpu ? 0 : -ENOMEM;

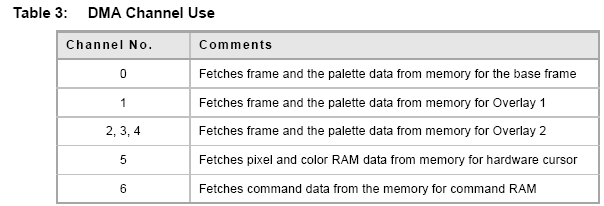
}

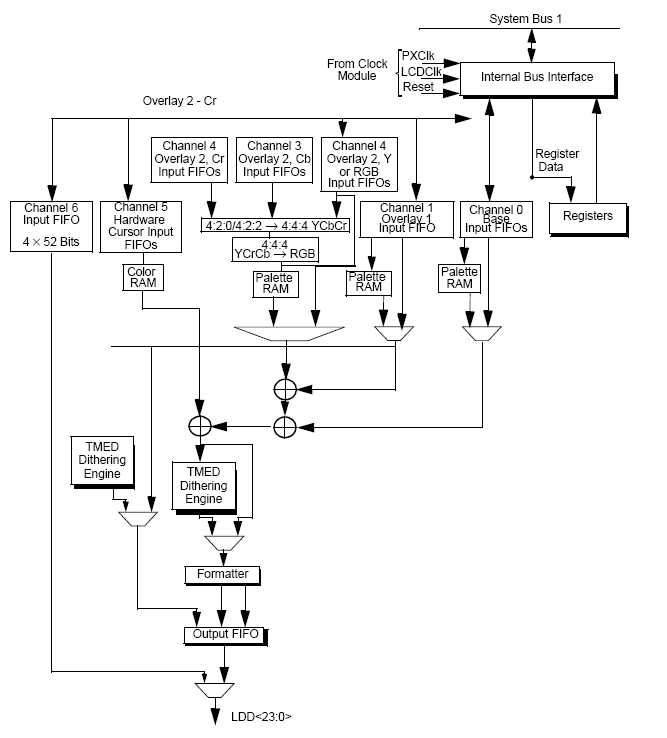
1. 帧缓冲设备的用户空间访问
2. 读写dev/fb，相当于读写屏幕缓冲区
3. 映射操作：通过mmap()映射操作将屏幕缓冲区的物理地址映射到用户空间的一段虚拟地址，之后用户就可以通过读写这段虚拟地址访问屏幕的缓冲区，在屏幕上绘图。
4. I/O控制：
5. LCD controller （参考文档PXA3xx\_DM\_Vol\_III）

驱动加载时，需要对LCD控制器进行初始化，如果需要对硬件配置有所修改可通过LCD控制器相应寄存器修改。

其中针对多planes（base，overlay1，overlay2）的方面稍微做些介绍。

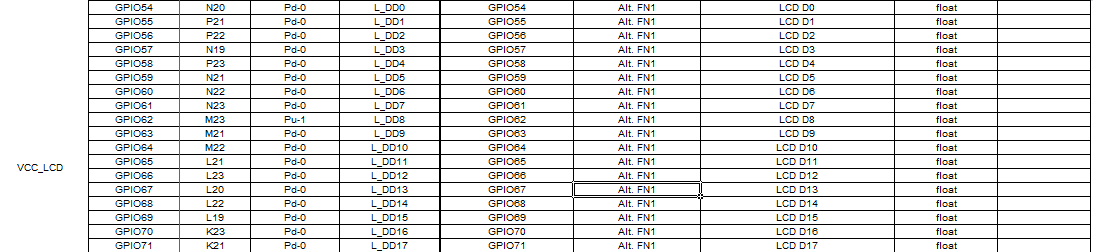
1. The LCD controller has a hardware cursor and three image planes: base, Overlay 1, and Overlay2 ，对应的DMA通道如下：





b、Depending on the type of panel used, the LCD controller is programmed to use 8-, 16-, or 18-bit

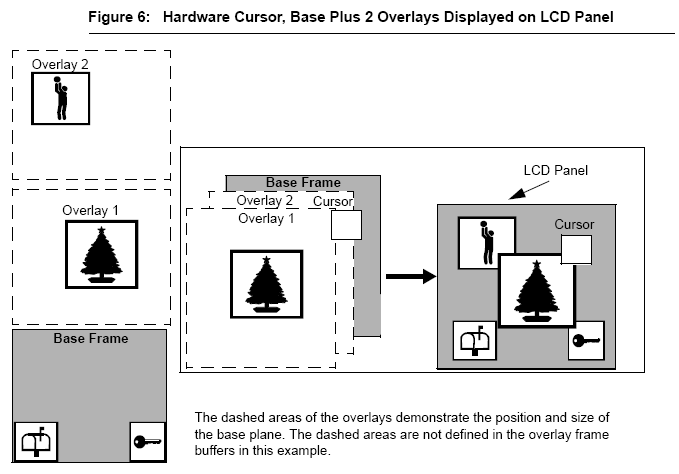
output-data pins。以下是cpu 对应的gpio pin。

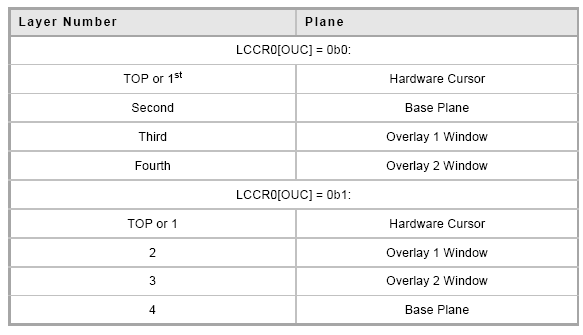


8,16,18bit的不同会使得每个时钟传递的pixel值不同。例如：8bit pin时每个clock传输2 2/3个pixel，16-bit的时候pixel/pixel\_clock

。

C、base overlay1,overlay2的显示层次并非一定，可以通过controller控制LCCR0[OUC] = 0b0:





1. LCD Controller Register，参考spec文件

参考文件：

1. 设备驱动开发详解
2. TFT-LCD Tentative Specification N080X1-T01 ver 0.0.pdf

3、PXA3xx\_DM\_Vol\_III.pdf

4、EEENOTE\_EV\_GPIO-201002042.xls

5、基础视讯