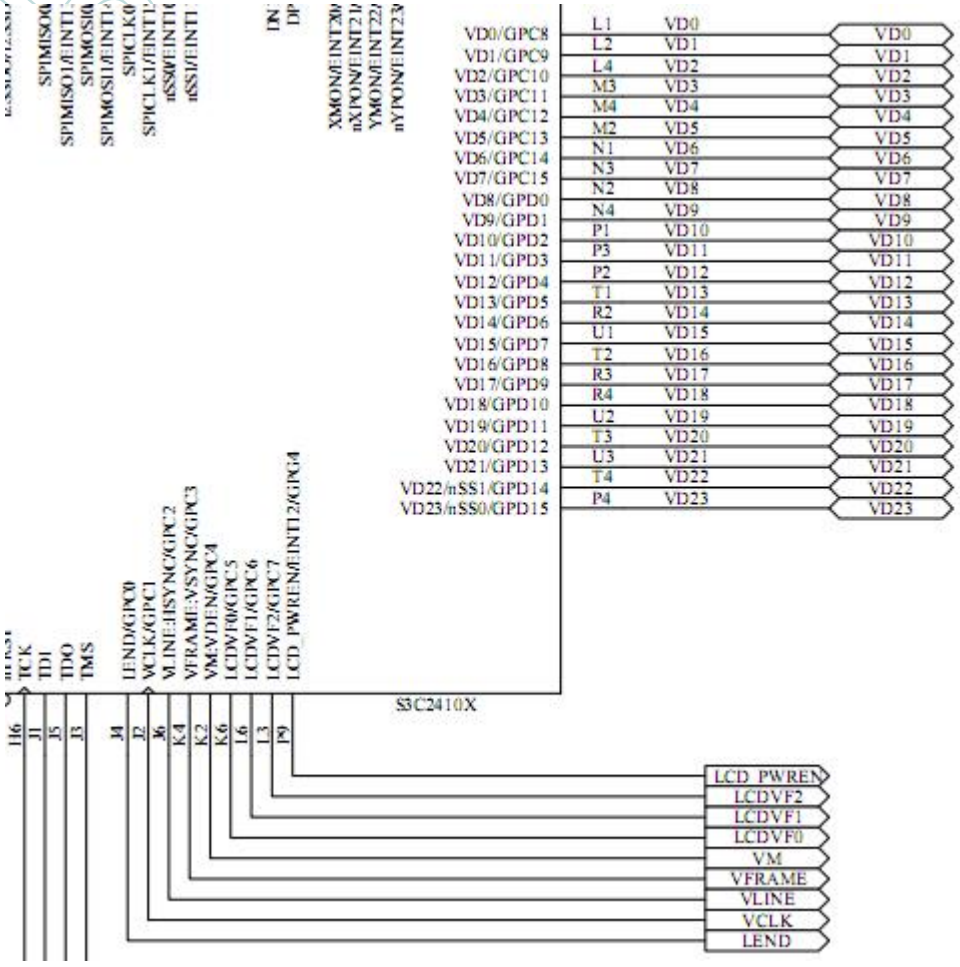
Linux lcd 驱动分析

华清远见 深圳中心 1001 期 学员 冯来林 2010-5-10

、

以下是从基础开始对 lcd 驱动进行分析。

一：CPU 上相关的 GPIO 介绍



图一 核心板上 LCD 的接口

根据上面 core 板上的信息，GPC8-GPC15 和 GPD0-GPD15 可用于连接

VD[0:23]（为 lcd 的 24 位数据线）。而 GPC0-7 可用于配置 LCD 屏的时序。GPG4 可用于 LCD_PWEREN。看数据手册可知，GPCCON 与 GPDCON 的每两位配置一个 GPCX 或 GPDX。GPCCON = 0xaaaaaaaa;GPDCON = 0xaaaaaaaa 可把 GPC 与 GPD 这两组的所有引脚用于 LCD 的功能。(注：具体的初始化即 GPIO 介绍可参考后面的 GPIO 一节。)

外部接口信号介绍：

VFRAME/VSYNC/STV ： 帧同步信号（STN）/ 垂直同步信号（TFT）/ SEC TFT 信号

VLINE/HSYNC/CPV： 行同步脉冲信号（STN）/ 水平同步信号（TFT）/ SEC TFT 信号

VCLK/LCD_HCLK ： 像素时钟信号（STN/TFT）/ SEC TFT 信号

VD[23:0] ： LCD 像素数据输出端口(STN/TFT/SEC TFT)

VM/VDEN/TP ： LCD 驱动器交流信号（STN）/ 数据使能信号（TFT）/ SEC TFT 信号

LEND/STH ： 行结束信号(TFT)/SEC TFT 信号

LCD_PWREN ： LCD 屏电源控制信号

LCDVF0 ： SEC TFT 信号 OE（SEC 表示 Samsung Electronics Company）

LCDVF1 ： SEC TFT 信号 REV

LCDVF2 ： SEC TFT 信号 REVB

注：上述信号的设置将会在后面的操作控制寄存器中讲到

LCD 的接口原理图（在 dev 中有，这里没有截出来）：

观察后可知 S3C2410 板子有两个 74LVCH162245 芯片，是为了增强驱动能力（具体可查 74LVCH162245 的数据手册），LCD_CON 是 LCD 屏的接口。Lcd 控制器将数据和控制时序传输给 74LVCH162245 再传给 LCD 屏的驱动器。

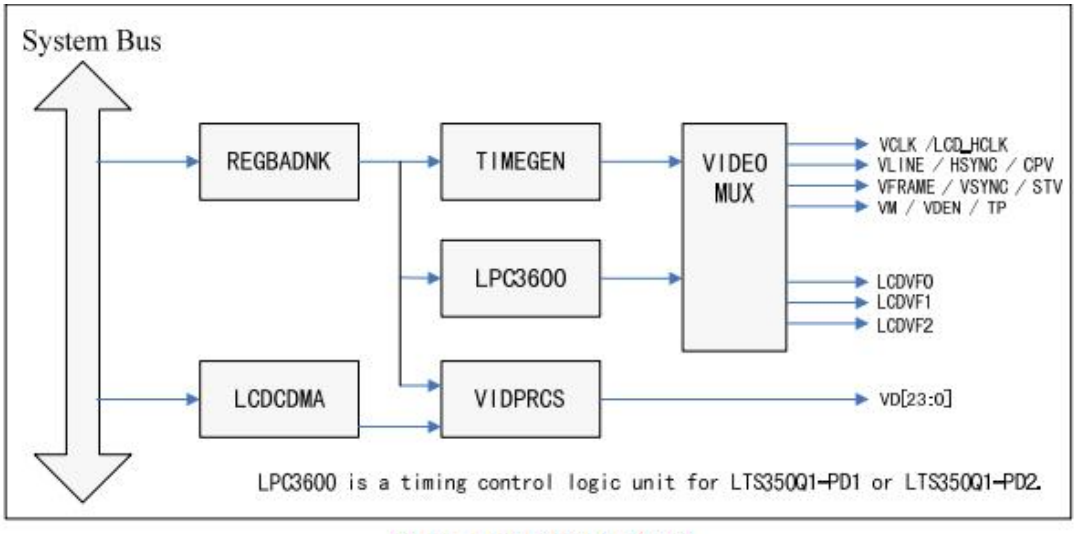


图 15-1 LCD 控制器框图

LCD 控制器框图：

S3C2410X 中 LCD 控制器用为传送视频数据和产生需要的控制信号的，如 VFRAME，VLINE,VCLK，VM,等。除控制信号外，S3C2410X 中的 LCD 控制器还有传送视频数据的端口，如图中 VD[23:0]所示。LCD 控制器由 REGBANK，LCDCDMA，VIDPRCS，TIMEGEN，and LPC3600（如图 15-1 LCD 控制器框图） 组成。 REGBANK 有 17 个可编程寄存器组和用来配置 LCD 控制器的 256*16 的调色板存储器。LCDCDMA 是一个专用 DMA，自动传送帖数据到 LCD 驱动器。利用这个专用的 DMA，视频数据可以在没有 CPU 的参与下自动显示。VIDPRCS 从 LCDCDMA 接收视频数据，然后将其转换成适合的数据格式通过数据端口 VD[23:0]发送到 LCD 驱动器上，例如 4/8 位单扫描或 4 位双扫描模式。TIMEGEN 由可编程逻辑组成，支持各种常见 LCD 驱动器的定时与速率界面的不同要求。TIMEGEN 模块产生 FRAME，VLINE，VCLK，VM 等信号。

二：阅读数据手册

1、LCD 屏的参数：

型号:WX3500F-M15#04

大小：3.5 英寸（1 英寸=3.5cm）

类型：TFT LCD

屏的分辨率：320*240(一帧有效的分辨率 240 行，每行有 320 个像素)

显示模式（位数）：可以支持到 24bpp（有待验证,bpp:bit per pixel 即每个像素用多少位来表示其颜色，如 16BPP 为 64K 即 65536 色。平时说的 130 万像素就是指一个像素点由 130 万阶组成。）

2、 S3C2410 内置 LCD 控制器分析

2、1 LCD 控制器概述：

S3C2410X 中的 LCD 控制器由传送逻辑构成，这种逻辑是把位于系统内存显示缓冲区中 LCD 视频数据传到外部的 LCD 驱动器，并提供必要的控制信号，如 VFRAME、VLINE、VCLK、VM 等。而这些控制信号参数是在 LCD 控制寄存器中设置的。

当把 VSYNC、HSYNC、VCLOCK 等信号的时间参数设置好后，并将帧内存（frame memory）的地址告诉 LCD 控制器，LCD 控制器即可自动地发起 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。LCD 控制器从内存中获得某个像素的 24 位颜色值后，直接通过 VD[23:0]数据线发送给 LCD。只需要把要显示的图像数据写入帧内存中。

2、2　控制信号简介：

不同的 LCD 厂商对于控制信号有不同的叫法，S3C2410 芯片手册也给出了一个信号的多个名称如下：

EXTERNAL INTERFACE SIGNAL

VFRAME/VSYNC/STV:Frame synchronous signal (STN)/vertical synchronous signal(TFT)/SEC TFT sign

VLINE/HSYNC/CPV : Line synchronous pulse signal (STN)/horizontal sync signal (TFT)/SEC TFT sign

VCLK/LCD_HCLK : Pixel clock signal (STN/TFT)/SEC TFT signal

VD[23:0] : LCD pixel data output ports (STN/TFT/SEC TFT)

VM/VDEN/TP : AC bias signal for the LCD driver (STN)/data enable signal (TFT)/SEC TFT sign

LEND/STH : Line end signal (TFT)/SEC TFT signal

LCD_PWREN : LCD panel power enable control signal

LCDVF0 : SEC TFT Signal OE

LCDVF1 : SEC TFT Signal REV

LCDVF2 : SEC TFT Signal REVB

The 33 output ports in total includes 24 data bits and 9 control bits

外部接口信号

VFRAME/VSYNC/STV : 帧同步信号（STN）/ 垂直同步信号（TFT）/ SEC TFT 信号

VLINE/HSYNC/CPV：行同步脉冲信号（STN）/ 水平同步信号（TFT）/ SEC TFT 信号

VCLK/LCD_HCLK : 像素时钟信号（STN/TFT）/ SEC TFT 信号

VD[23:0] : LCD 像素数据输出端口(STN/TFT/SEC TFT)

VM/VDEN/TP：LCD 驱动器交流信号（STN）/ 数据使能信号（TFT）/ SEC TFT 信号

LEND/STH : 行结束信号(TFT)/SEC TFT 信号

LCD_PWREN ： LCD 屏电源控制信号

LCDVF0 ： SEC TFT 信号 OE

LCDVF1 ： SEC TFT 信号 REV

LCDVF2 ： SEC TFT 信号 REVB

这里我做一个简单的介绍：

VFRAME：LCD 控制器和 LCD 驱动器之间的帧同步信号。该信号告诉 LCD 屏的新的一帧开始了。LCD 控制器在一个完整帧显示完成后立即插入一个 VFRAME 信号，开始新一帧的显示；即“是跳到最上边的时候了”。

VLINE：LCD 控制器和 LCD 驱动器之间的行同步脉冲信号，该信号用于 LCD 驱动器将水平线（行）移位寄存器的内容传送给 LCD 屏显示。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 VLINE 信号；即“是跳到最左边的时候了”。

VCLK：LCD 控制器和 LCD 驱动器之间的像素时钟信号，由 LCD 控制器送出的数据在 VCLK 的上升沿处送出，在 VCLK 的下降沿处被 LCD 驱动器采样；

VM：LCD 驱动器的 AC 信号。VM 信号被 LCD 驱动器用于改变行和列的电压极性，从而控制像素点的显示或熄灭。VM 信号可以与每个帧同步，也可以与可变数量的 VLINE 信号同步。

3） 数据线：也就是我们说的 RGB 信号线，S3C2410 芯片手册上都有详细的说明，由于篇幅关系，在此不一一摘录，不过需要与硬件工程是配合的是他采用了哪种接线方法，24 位 16 位或其它。对于 16 位 TFT 屏又有两种方式，在写驱动前你要清楚是 5：6：5 还是 5：5：5：1，这些与驱动的编写都有关系

4） 要注意一下 LCD 的电源电压，对于手持设备来说一般都为 5V 或 3.3V，或同时支持 5V 和 3.3V，如果 LCD 的需要的电源电压是 5V，那就要注意了，S3C2410 的逻辑输出电压只有 3.3V，此时一定要让你们的硬件工程师帮忙把 S3C2410 的逻辑输出电压提高到 5V，否则你可能能将屏点亮，但显示的图像要等到太阳从西边出来的那一天才能正常，呵呵，我可吃过苦头的哦！

注释： LCD 驱动器：通常 LCD 驱动器会以 CON/COG 的形式与 LCD 玻璃基板制作在一起。

LCD 控制器：许多 MCU 内部直接集成了 LCD 控制器。通过它可以很方便的控制 STN 和 TFT 屏。

2、3 TFT LCD 的工作时序分析：

图 2、1 是 TFT 屏的典型时序。其中 VSYNC 是帧同步信号，VSYNC 每发出 1 个脉冲，都意味着新的 1 屏视频资料开始发送。而 HSYNC 为行同步信号，每个 HSYNC 脉冲都表明新的 1 行视频资料开始发送。而 VDEN 则用来标明视频资料的有效，VCLK 是用来锁存视频资料的像数时钟。

并且在帧同步以及行同步的头尾都必须留有回扫时间，例如对于 VSYNC 来说前回扫时间就是（VSPW+1）＋（VBPD+1），后回扫时间就是（VFPD +1）；HSYNC 亦类同。这样的时序要求是当初 CRT 显示器由于电子枪偏转需要时间，但后来成了实际上的工业标准，乃至于后来出现的 TFT 屏为了在时序上于 CRT 兼容，也采用了这样的控制时序。

图 2、2 是 WXCAT35-TG3#001F（一款 3.5 寸 TFT 真彩 LCD 屏，分辨率为 240*320）的时序要求

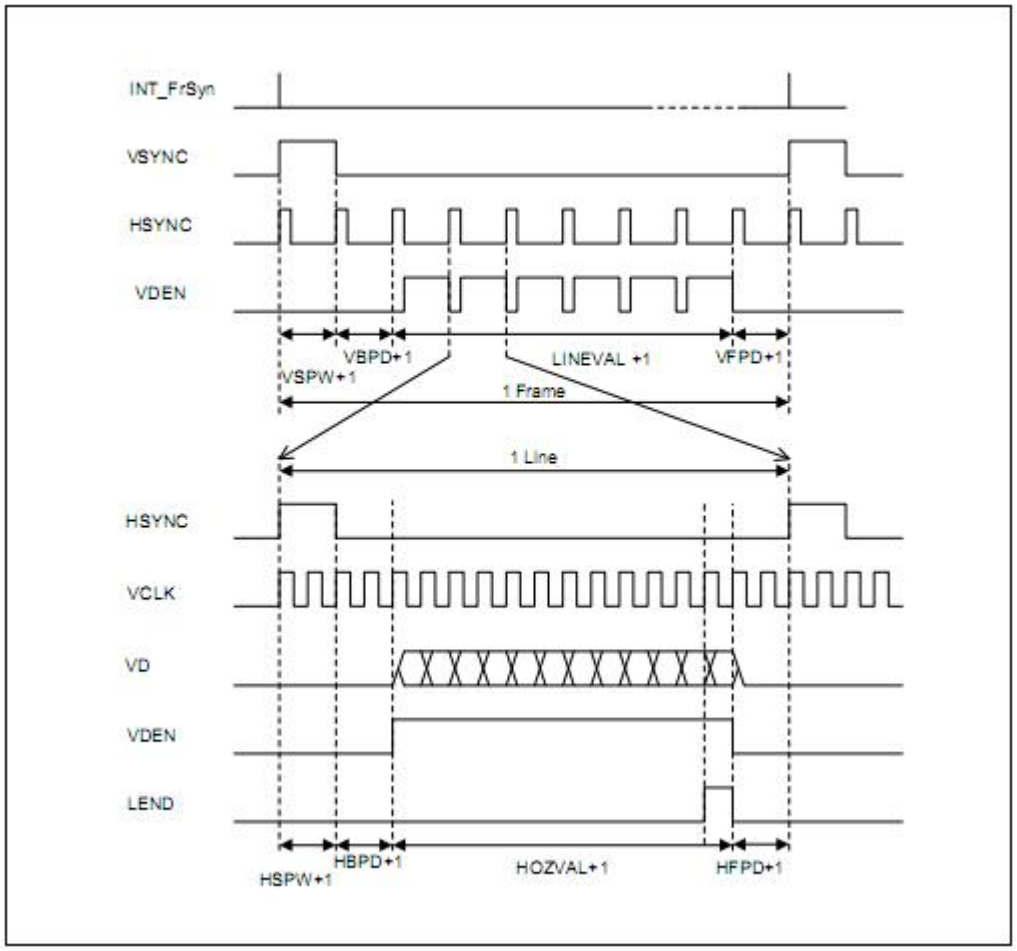


图 2.1

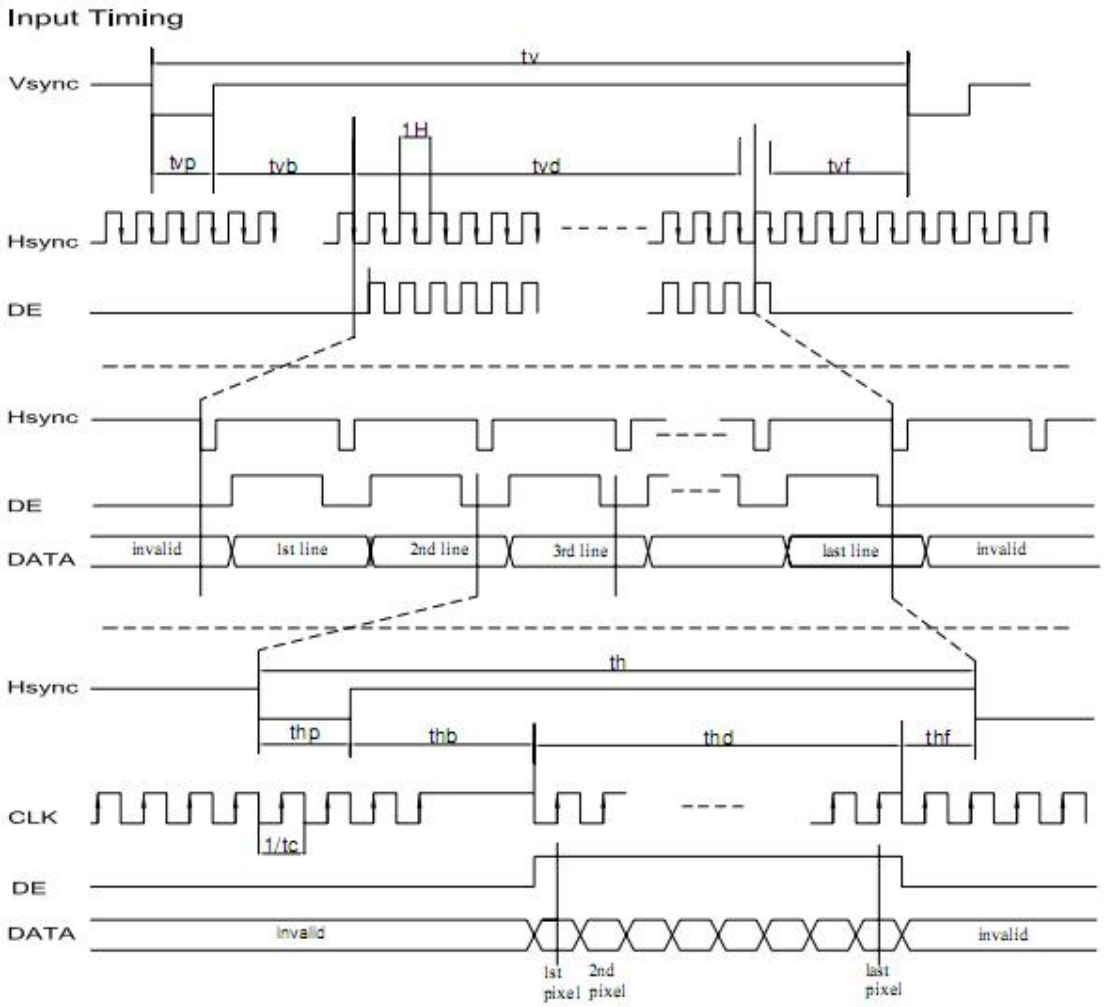

Input Timing
Vsync
tv
tvp
tvb
1H
tvd
tvf
Hsync
DE
Hsync
DE
DATA
invalid
1st line
2nd line
3rd line
last line
invalid
Hsync
th
thp
thb
thd
thf
CLK
1/tc
DE
DATA
invalid
invalid
1st pixel
2nd pixel
last pixel

图 2.2

Signal	Item	Symbol	Min	Typ	Max	Unit
Dclk	Frequency	Dclk	-	6.4	-	MHZ
	Dclk-Period	Tosc	-	156	-	ns
Data	Setup Time	TSU	12	-	-	ns
	Hold Time	THD	12	-	-	ns
Hsync	Period	Th	-	408	-	DCLK
	Pulse Width	Thp	-	30	-	DCLK
	Back-Porch	Thb	-	38	-	DCLK
	Display Period	Thd	-	320	-	DCLK
	Front-Porch	Thf	-	20	-	DCLK
Vsync	Period	Tv	-	270	-	TH
	Pulse Width	Tvp	-	3	-	TH
	Back-Porch	Tvb	-	15	-	TH
	Display Period	Tvd	-	240	-	TH
	Front-Porch	Tvf	-	12	-	TH

图 2.3

结合上面三幅图，我们不难得出：

VSPW+1=tvp=3 -> VSPW=2

VBPD+1=15 -> VBPD=14

LINVAL+1=240-> LINVAL=239

VFPD+1=12 -> VFPD=11

HSPW+1=30 -> HSPW=29

HBPD+1=38 -> HBPW=37

HOZVAL+1=320-> HOZVAL=319

HFPD+1=20 -> HFPD=19

以上各参数，除了 LINVAL 和 HOZVAL 直接和屏的分辨率有关，其它的参数在实际操作过程中应以上面的为参考，不应偏差太多。

以操作 WX3500F-M15#04 为例说明时序

VCLK：像素时钟，作为时序图的基准信号，它的频率计算如下：

VCLK(Hz) = HCLK/[(CLKVAL+1)x2]，VCLK 的频率取决于寄存器 LCDCON1 中 CLKVAL 的值，数据手册中有如下说明：

LCDCON1	Bit	描述	初始值
LINECNT (只读)	[27:18]	行计数器状态位，值由 LINEVAL 递减至 0	0000000000
CLKVAL	[17:8]	Determine the rates of VCLK and CLKVAL[9:0]. **STN**: VCLK = HCLK / (CLKVAL×2) (CLKVAL≧2) **TFT**: VCLK = HCLK / [(CLKVAL+1) × 2] (CLKVAL≧0)	0000000000

VSYNC：此信号有效时，表示一帧数据的开始。它的频率有下面的公式：

Frame Rate = 1/ [{ (VSPW+1) + (VBPD+1) + (LINEVAL + 1) + (VFPD+1) } * {(HSPW+1) + (HBPD +1) + (HFPD+1) + (HOZVAL + 1) }*{ 2 *(CLKVAL+1) / (HCLK) }]

VSPW：表示 VSYNC 信号的脉冲宽度为（VSPW+1）个 HSYNC 信号周期，即(VSPW+1)行，这（VSPW+1）行无效。

VBPD：表示 VSYNC 信号脉冲之后，还要经过（VBPD+1）个 HSYNC 信号周期，即（VBPD+1）行，这（VBPD+1）行无效。

综上所述：在 VSYNC 信号有效之后，共要经过（VSPW+1+VBPD+1）个无效的行。

LINEVAL：随后连续发出（LINEVAL+1）行有效数据。这里设为 239

VFBD:最后是（VFBD+1）个无效的行，到此，完整的一帧结束。接着下一帧开始。

现在深入到一行中像素数据的传输过程，

（1）HSYNC 信号有效时表示一行数据的开始

（2）HSPW 表示 HSYNC 信号的脉冲宽度为（HSYNC+1）个 VCLK 信号周期，即

（HSPW+1）个像素，这（HSPW+1）个像素无效。

（3）HBPD 表示 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。

（4）所以在 HSYNC 信号有效之后，总共还要经过（HSPW+1+HBPD+1）个无效的像素，第一个有效地像素才会出现。随后即连续发出（HOZVAL+1）个像素的有效数据。

（5）最后是（HFPD+1）个无效的像素，接着就是下一行的数据，即下一个 HSYNC 信号。

以上的 VSPW、VBPD、LINEVAL、VFPD、HBPD、HOZVAL、HFPD、CLKVAL 等的值根据 LCD 屏手册在 LCDCONX 中设置。

2.4 寄存器操作：

注意：寄存器中的值由 LCD 屏决定的，设置 LCd 控制器的值去适应 LCd 屏。

对于 TFT LCD 一般情况下只需要设置 LCDCON1-LCDCON5(用于选择 LCd 类型、设置各类控制信号的时间特性等)和 LCDSADDR1-LCDSADDR3（设置帧内存地址）LCD 屏参数：

由于要操作的寄存器及设置的参数太多，且需要对照 LCD 屏手册和数据手册对应着看，来设置 S3c2410 相应寄存器的位。文字描述不清楚。这里只说几个注意的地方：

（1）在操作 LCDCON5 的几个关于极性的位时，选择是正常还是反转是根据 lcd 屏手册的时序图和 S3C2410 的时序图对比得出的。

LCDSADDR1-LCDSADDR3（帧内存地址寄存器）：

帧内存可以很大，而真正要显示的区域被称为视口（view poit），它处于帧内存之内。这三个寄存器用于确定帧内存起始地址，定位视口在帧存中的位置。

三：linux 驱动代码分析

3.1 FrameBuffer

Linux 是工作在保护模式下，所以用户态进程是无法像 DOS 那样使用显卡 BIOS 里提供的中断调用来实现直接写屏，Lin 仿显卡的功能，将显 ux 抽象出 FrameBuffer 这个设备来供用户态进程实现直接写屏。Framebuffer 机制模卡硬件结构抽象掉，可以通过 Framebuffer 的读写直接对显存进行操作。用户可以将 Framebuffer 看成是显示内存的一个映像，将其映射到进程地址空间之后，就可以直接进行读写操作，而写操作可以立即反应在屏幕上。这种操作是抽象的，统一的。用户不必关心物理显存的位置、换页机制等等具体细节。这些都是由 Framebuffer 设备驱动来完成的。在 Linux 系统下，FrameBuffer 的主要的结构如图所示。Linux 为了开发 FrameBuffer 程序的方便，使用了分层结构。fbmem.c 处于 Framebuffer 设备驱动技术的中心位置。它为上层应用程序提供系统调用，也为下一层的特定硬件驱动提供接口；那些底层硬件驱动需要用到这儿的接口

向系统内核注册它们自己。

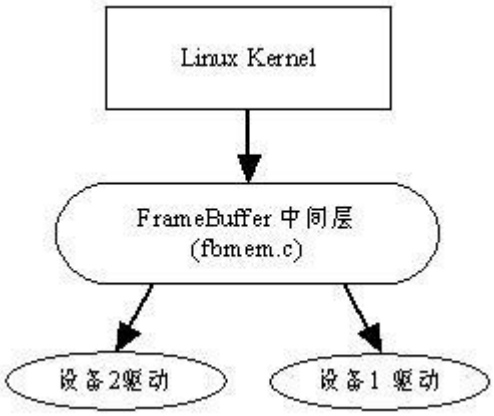


图 3.1

fbmem.c 为所有支持 FrameBuffer 的设备驱动提供了通用的接口，它就是 FrameBuffer 的驱动程序，避免重复工作。

下面将介绍 fbmem.c 主要的一些数据结构。参考宋宝华的设备驱动开发详解。

在 FrameBuffer 中，fb_info 可以说是最重要的一个结构体，它是 Linux 为帧缓冲设备定义的驱动层接口，在 include/linux/Fb.h 中定义。简称 FBI，它包括了关于帧缓冲设备属性和操作的完整描述，它不仅包含了底层函数，而且还有记录设备状态的数据。每个帧缓冲设备都必须与一个 fb_info 结构相对应 fb_info 的主要成员如下

```
struct fb_info {
 int node;
 struct fb_var_screeninfo var; /* Current var */
 struct fb_fix_screeninfo fix;  /* Current fix */
 struct fb_videomode *mode; /* current mode */

 struct fb_ops *fbops;
 struct device *device;   /* This is the parent */
 struct device *dev;   /* This is this fb device */

 char __iomem *screen_base; /* Virtual address */
 unsigned long screen_size; /* Amount of ioremapped VRAM or 0 */
 ..........
};
```

其中 node 成员域标示了特定的 FrameBuffer，实际上也就是一个 FrameBuffer 设备的次设备号。fb_var_screeninfo 结构体成员记录用户可修改的显示控制器参数，包括屏幕分辨率和每个像素点的比特数。fb_var_screeninfo 中的 xres 定义屏幕一行有多少个点，yres 定义屏幕一列有多少个点，bits_per_pixel 定义每个点用多少个字节表示。其他域见以下代码注释。

```
struct fb_var_screeninfo {
 __u32 xres;   /* visible resolution */
```

```
 __u32 yres;
 __u32 xoffset;  /* offset from virtual to visible */
 __u32 yoffset;  /* resolution */
 __u32 bits_per_pixel; /* bits/pixel */
 __u32 pixclock;  /* pixel clock in ps (pico seconds) */
 __u32 left_margin; /* time from sync to picture */
 __u32 right_margin; /* time from picture to sync */
 __u32 hsync_len;  /* length of horizontal sync */
 __u32 vsync_len;  /* length of vertical sync */
 ............
};
```

在 fb_info 结构体中，fb_fix_screeninfo 中记录用户不能修改的显示控制器的参数，如屏幕缓冲区的物理地址，长度。当对帧缓冲设备进行映射操作的时候，就是从 fb_fix_screeninfo 中取得缓冲区物理地址的。

```
struct fb_fix_screeninfo {
 char id[16];        /* identification string eg "TT Builtin" */
 unsigned long smem_start;     /* Start of frame buffer mem (physical
address) */
 __u32 smem_len;          /* Length of frame buffer mem */
 unsigned long mmio_start;     /* Start of Mem Mapped I/O(physical address)
*/
 __u32 mmio_len;       /* Length of Memory Mapped I/O  */
 ............
};
```

fb_info 还有一个很重要的变量就是 fb_ops。它是提供给底层设备驱动的一个接口。通常我们编写字符驱动的时候，要填写一个 file_operations 结构体，并使用 register_chrdev()注册之，以告诉 Linux 如何操控驱动。当我们编写一个 FrameBuffer 的时候，就要依照 Linux FrameBuffer 编程的套路，填写 fb_ops 结构体。这个 fb_ops 也就相当于通常的 file_operations 结构体。

```
struct fb_ops {
int (*fb_open)(struct fb_info *info, int user);
int (*fb_release)(struct fb_info *info, int user);
ssize_t (*fb_read)(struct file *file, char __user *buf, size_t count,
loff_t *ppos);
ssize_t (*fb_write)(struct file *file, const char __user *buf, size_t
count,
loff_t *ppos);
int (*fb_set_par)(struct fb_info *info);
int (*fb_setcolreg)(unsigned regno, unsigned red, unsigned green,
unsigned blue, unsigned transp, struct fb_info *info);
int (*fb_setcmap)(struct fb_cmap *cmap, struct fb_info *info)
int (*fb_mmap)(struct fb_info *info, struct vm_area_struct *vma);
..............
}
```

上面的结构体，根据函数的名字就可以看出它的作用，这里不在一一说明。下图给出了 Linux FrameBuffer 的总体结构，作为这一部分的总结。

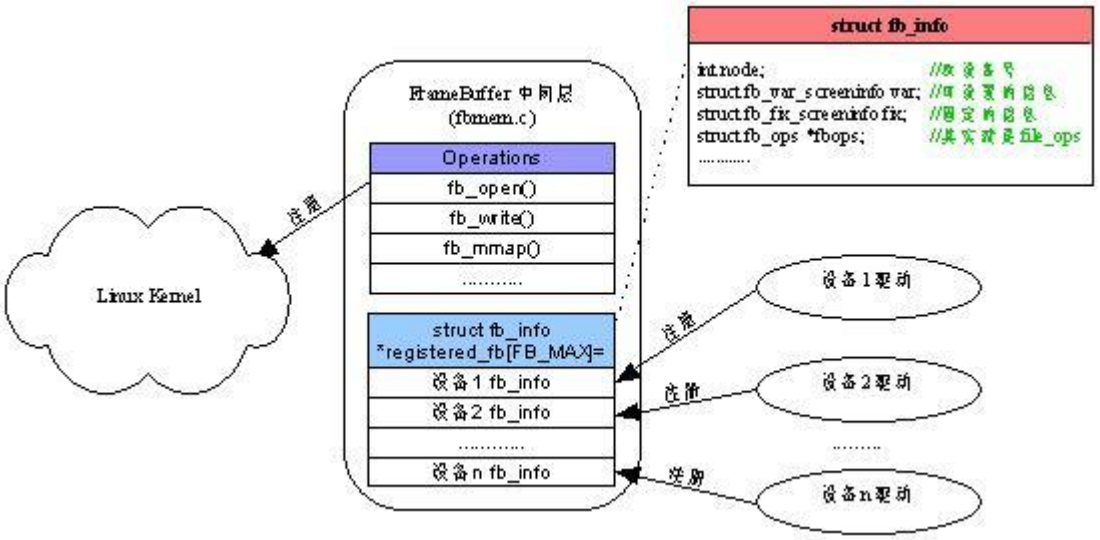


图 3.2

3.2 S3C2410 中 LCD 的数据结构

在 S3C2410 的 LCD 设备驱动中（drives/video/3sc2410fb.h），定义 s3c2410fb_info 来标识一个 LCD 设备，结构体如下：

```
struct s3c2410fb_info {
    struct fb_info       *fb;
    struct device    *dev;
    struct clk    *clk;

    struct s3c2410fb_mach_info *mach_info;

    /* raw memory addresses */
    dma_addr_t    map_dma;   /* physical */
    u_char *      map_cpu;   /* virtual */
    u_int         map_size;

    struct s3c2410fb_hw  regs;

    /* addresses of pieces placed in raw buffer */
    u_char *      screen_cpu;    /* virtual address of buffer */
    dma_addr_t    screen_dma;    /* physical address of buffer */
    unsigned int      palette_ready;

    /* keep these registers in case we need to re-write palette */
    u32       palette_buffer[256];
    u32       pseudo_pal[16];
```

```
};
```

成员变量 fb 指向我们上面所说明的 fb_info 结构体，代表了一个 FrameBuffer。dev 则表示了这个 LCD 设备。map_dma，map_cpu，map_size 这三个域向了开辟给 LCD DMA 使用的内存地址。screen_cpu，screen_dma 指向了 LCD 控制器映射的内存地址。另外 regs 标识了 LCD 控制器的寄存器。

```
/* linux/include/asm/arch-s3c2410/fb.h */
struct s3c2410fb_hw {
    unsigned long lcdcon1;
    unsigned long lcdcon2;
    unsigned long lcdcon3;
    unsigned long lcdcon4;
    unsigned long lcdcon5;
};
```

这个寄存器和硬件的寄存器一一对应，主要作为实际寄存器的映像，以便程序使用。

这个 s3c2410fb_info 中还有一个 s3c2410fb_mach_info 成员域。它存放了和体系结构相关的一些信息，如时钟、LCD 设备的 GPIO 口等等。这个结构体定义为

```
/* linux/include/asm/arch-s3c2410/fb.h */

struct s3c2410fb_mach_info {
    unsigned char fixed_syncs;  /* do not update sync/border */

    /* Screen size */
    int    width;
    int    height;

    /* Screen info */
    struct s3c2410fb_val xres;
    struct s3c2410fb_val yres;
    struct s3c2410fb_val bpp;

    /* lcd configuration registers */
    struct s3c2410fb_hw  regs;

    /* GPIOs */

    unsigned long gpcup;
    unsigned long gpcup_mask;
    unsigned long gpccon;
    unsigned long gpccon_mask;
    unsigned long gpdup;
    unsigned long gpdup_mask;
    unsigned long gpdcon;
    unsigned long gpdcon_mask;
```

```
    /* lpc3600 control register */
    unsigned long lpcsel;
};
```

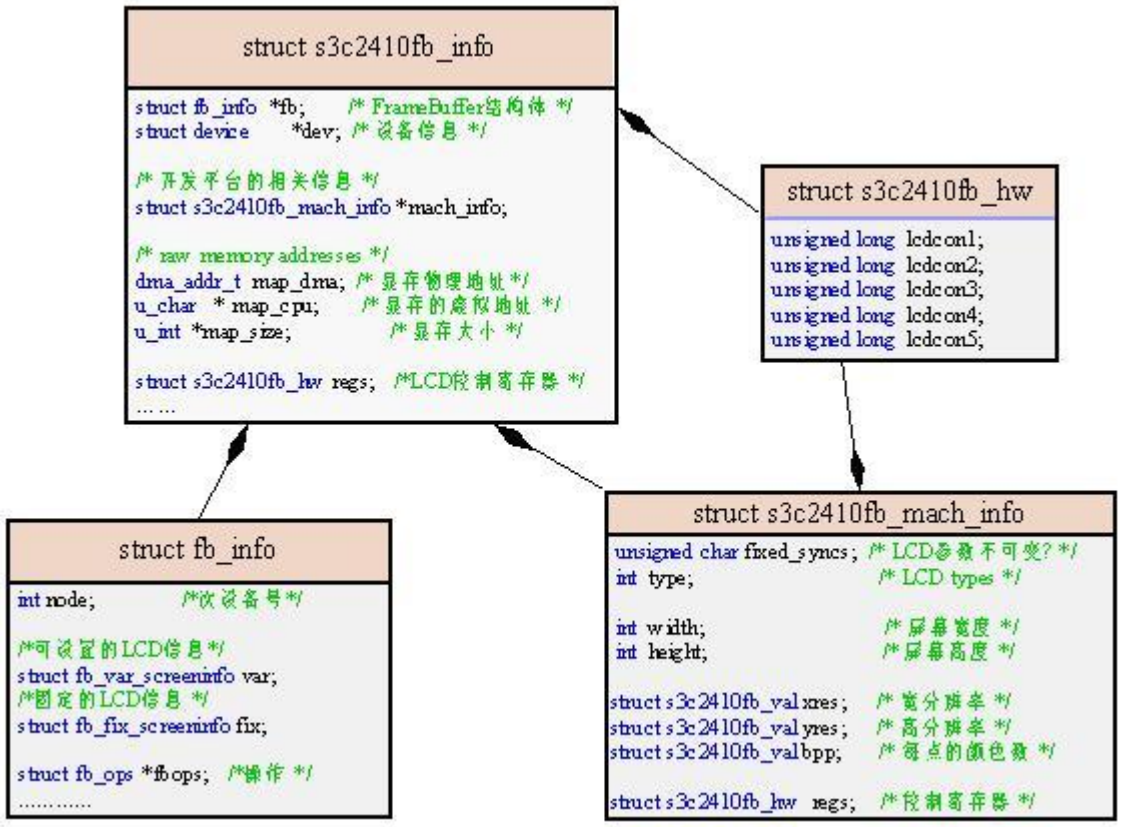


图 3.3

3.3 主要代码结构以及关键代码分析

（1）FrameBuffer 驱动的统一管理

fbmem.c 实现了 Linux FrameBuffer 的中间层，任何一个 FrameBuffer 驱动，在系统初始化时，必须向 fbmem.c 注册，即需要调用 register_framebuffer()函数，在这个过程中，设备驱动的信息将会存放入名称为 registered_fb 数组中，这个数组定义为

struct fb_info *registered_fb[FB_MAX];

int num_registered_fb;

它是类型为 fb_info 的数组，另外 num_register_fb 则存放了注册过的设备数量。

我们分析一下 register_framebuffer 的代码。

```
int register_framebuffer(struct fb_info *fb_info)
{
 int i;
 struct fb_event event;
 struct fb_videomode mode;

 if (num_registered_fb == FB_MAX) return -ENXIO; /* 超过最大数量 */
 num_registered_fb++;
```

```
for (i = 0 ; i < FB_MAX; i++)
 if (!registered_fb[i]) break;     /* 找到空余的数组空间 */
fb_info->node = i;

fb_info->dev = device_create(fb_class, fb_info->device,
    MKDEV(FB_MAJOR, i), "fb%d", i);  /* 为设备建立设备节点 */
if (IS_ERR(fb_info->dev)) {
 ............
} else{
 fb_init_device(fb_info);      /* 初始化改设备 */
}
.............
return 0;

}
```

3.4 从上面的代码可知，当 FrameBuffer 驱动进行注册的时候，它将驱动的 fb_info 结构体记录到全局数组 registered_fb 中，并动态建立设备节点，进行设备的初始化。注意，这里建立的设备节点的次设备号就是该驱动信息在 registered_fb 存放的位置,即数组下标 i 。在完成注册之后,fbmem.c 就记录了驱动的 fb_info。这样我们就有可能实现 fbmem.c 对全部 FrameBuffer 驱动的统一处理。

3.5 实现消息的分派

fbmem.c 实现了对系统全部 FrameBuffer 设备的统一管理。当用户尝试使用一个特定的 FrameBuffer 时，fbmem.c 怎么知道该调用那个特定的设备驱动呢？

我们知道，Linux 是通过主设备号和次设备号，对设备进行唯一标识。不同的 FrameBuffer 设备向 fbmem.c 注册时，程序分配给它们的主设备号是一样的，而次设备号是不一样的。于是我们就可以通过用户指明的次设备号，来觉得具体该调用哪一个 FrameBuffer 驱动。下面通过分析 fbmem.c 的 fb_open()函数来说明。(注：一般我们写 FrameBuffer 驱动不需要实现 open 函数，这里只是说明函数流程。)

```
static int fb_open(struct inode *inode, struct file *file){
 int fbidx = iminor(inode);
 struct fb_info *info;
 int res;
    /* 得到真正驱动的函数指针 */
 if (!(info = registered_fb[fbidx])) return -ENODEV;
 if (info->fbops->fb_open) {
  res = info->fbops->fb_open(info,1); //调用驱动的 open()
  if (res)  module_put(info->fbops->owner);
 }
 return res;
}
```

当用户打开一个 FrameBuffer 设备的时，将调用这里的 fb_open()函数。传进来的 inode 就是欲打开设备的设备号，包括主设备和次设备号。fb_open 函数首先通过 iminor()函数取得次设备号，然后查全局数组 registered_fb 得到设备的 fb_info 信息，而这里面存放了设备的操作函数集 fb_ops。这样，我们就可以调用具体驱动的 fb_open() 函数，实现 open 的操作。下面给出了一个 LCD 驱动的 open() 函数的调用流程图，用以说明上面的步骤。

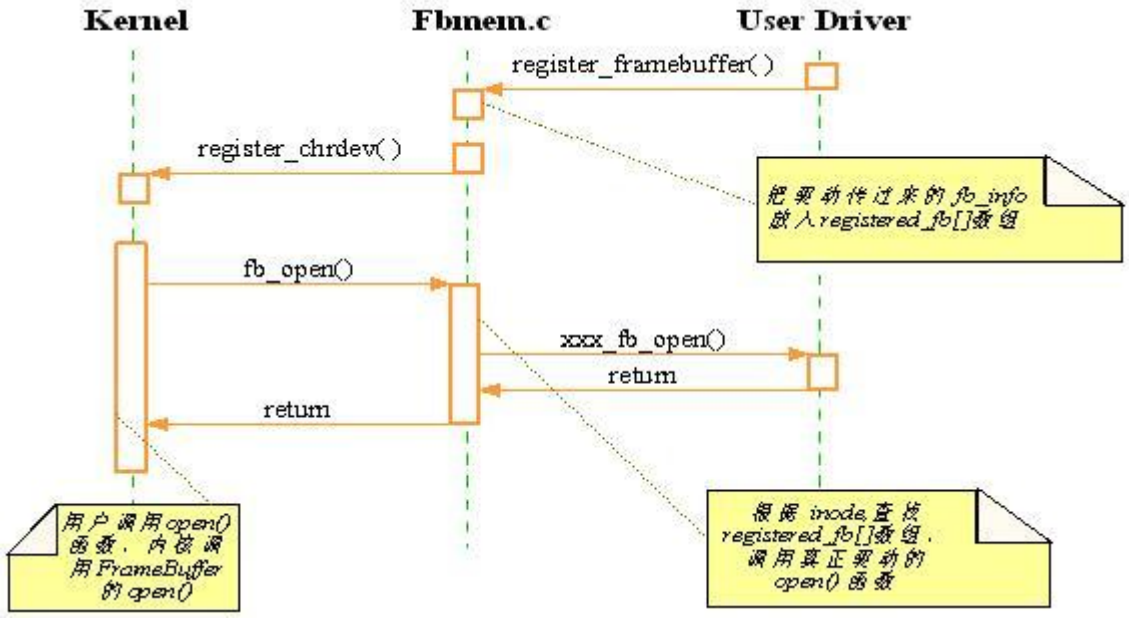


图 3.4

3.6 内核初始化时候调用 s3c2410fb_probe 函数。下面分析这个函数的做的工作。首先先动态分配 s3c2410fb_info 空间。

```
fbinfo = framebuffer_alloc(sizeof(struct s3c2410fb_info),&pdev->dev);
```

把域 mach_info 指向 mach-smdk2410.c 中的 smdk2410_lcd_cfg 。

```
info->mach_info = pdev->dev.platform_data;
```

设置 fb_info 域的 fix，var，fops 字段。

```
fbinfo->fix.type  =  FB_TYPE_PACKED_PIXELS;
fbinfo->fix.type_aux      = 0;
fbinfo->fix.xpanstep      = 0;

fbinfo->var.nonstd      = 0;
fbinfo->var.activate    = FB_ACTIVATE_NOW;
fbinfo->var.height      = mach_info->height;
fbinfo->var.width      = mach_info->width;

fbinfo->fbops       = &s3c2410fb_ops;
......
```

该函数调用 s3c2410fb_map_video_memory()申请 DMA 内存，即显存。

```
fbi->map_size = PAGE_ALIGN(fbi->fb->fix.smem_len + PAGE_SIZE);
fbi->map_cpu  = dma_alloc_writecombine(fbi->dev, fbi->map_size,
          &fbi->map_dma, GFP_KERNEL);

fbi->map_size = fbi->fb->fix.smem_len;
.....
```

设置控制寄存器，设置硬件寄存器。

```
memcpy(&info->regs, &mach_info->regs,sizeof(info->regs));
info->regs.lcdcon1 &= ~S3C2410_LCDCON1_ENVID;
.........
```

调用函数 s3c2410fb_init_registers()，把初始值写入寄存器。

```
writel(fbi->regs.lcdcon1, S3C2410_LCDCON1);
   writel(fbi->regs.lcdcon2, S3C2410_LCDCON2);
```

（3）当用户调用 mmap()映射内存的时候，Fbmem.c 把刚才设置好的显存区域映射给用户。

```
  start = info->fix.smem_start;
  len = PAGE_ALIGN( (start & ~PAGE_MASK) + info->fix.smem_len);
  io_remap_pfn_range(vma, vma->vm_start, off >> PAGE_SHIFT,
vma->vm_end - vma->vm_start,vma->vm_page_prot);
  ......
```

这样就完成了驱动初始化到用户调用的整个过程。

4、加驱动到 uboot 中：

由于不清楚 uboot 中的编译规则。直接把 lcd_test.c 中的 Void lcd_test（void）函数加到了 uboot 第二阶段的第一个函数中。即 lib_arm/board.c 中的 start_armboot 函数。通过调用 lcd_test（）而调用一系列的函数（一些初始化），如下.

```
void lcd_test(void)函数说明;
{
    Lcd_Port_Init();                    // 设置 LCD 引脚
    Tft_Lcd_Init();                     // 初始化 LCD 控制器
  Lcd_PowerEnable(0, 0);                // 设置 LCD_PWREN 有效，它用于打
开 LCD 的电源
Lcd_EnvidOnOff(1);                      // 使能 LCD 控制器输出信号
ClearScr(0x0);                          // 清屏
 Paint_Bmp(0,0,320,240,gImage_logo);     //显示图片
}
```

Logo.c 文件说明：gImage_logo[153600]数组由工具转换而来，

修改的地方：
加 lcd_test.c 和 logo.c 到 lib_arm 目录下
改 lib_arm 目录下的 makefile：COBJS = armlinux.o board.o cache.o div0.o lcd_test.o logo.o
改 lcd_test 的头文件#include <common.h>
#include <s3c2410.h>
把初始化 GPIO 口与操作的寄存器定义直接放在 lcd_test.c 中

5、加驱动到内核中

对于不同系列的内核版本，修改的不一样，到 linux2.6.26 版本后做了更好的封装，这里以 linux2.6.26.4 为例进行讲解，目标板是 fs2410.
在 /arch/arm/mach-s3c2410/mach-smdk2410.c 里添加头文件：#include<asm/arch/fb.h>和以下代码：

```
static struct s3c2410fb_display smdk2410_display __initdata =
{
        .type = S3C2410_LCDCON1_TFT,

        .width=     320,
        .height=    240,

        .xres=      320,
        .yres=      240,
        .bpp=       16,

        .pixclock = 270000,
        .left_margin= 20,
        .right_margin= 38,
        .hsync_len= 30,
        .upper_margin= 12,
        .lower_margin= 15,
        .vsync_len= 3,

        .lcdcon5 = S3C2410_LCDCON5_FRM565 |
                        S3C2410_LCDCON5_INVVCLK|
                        S3C2410_LCDCON5_INVVLINE |
                        S3C2410_LCDCON5_INVVFRAME |
                        S3C2410_LCDCON5_PWREN |
                        S3C2410_LCDCON5_HWSWP,

}

static struct s3c2410fb_mach_info smdk2410_lcdcfg __initdata =
{
    .displays= &smdk2410_display,
```

```
    .num_displays = 1,
    .default_display = 0,
    .lpcsel=    0x0,
    .gpccon=    0xaaaaaaaa,
    .gpccon_mask=   0xffffffff,
    .gpcup=     0xffffffff,
    .gpcup_mask=    0xffffffff,
    .gpdcon=    0xaaaaaaaa,
    .gpdcon_mask=   0x0,
    .gpdup=     0xffffffff,
    .gpdup_mask=    0xffffffff,
};
```

在 smdk2410_init 函数里添加：

```
static void __init smdk2410_init(void)
{
    platform_add_devices(smdk2410_devices,
ARRAY_SIZE(smdk2410_devices));
    s3c24xx_fb_set_platdata(&smdk2410_lcdcfg);
smdk_machine_init();
}//这里要注意顺序：必须要先 add 再 set
```

c. 重新配置内核，然后就可以编译了

在内核的顶层目录下运行 make menuconfig 后配置设置支持的图像支持，具体配置界面如图所示：

```
Device Drivers  --->
  Graphics support  --->
    <*> Support for frame buffer devices  --->
      <*>   S3C2410 LCD framebuffer support
```

配置完毕后，运行 make menuconfig 进行编译，将产生的 arch/arm/boot/uImage 拷贝到/tftpboot 目录中，后启动开发板，会看到小企鹅出现。