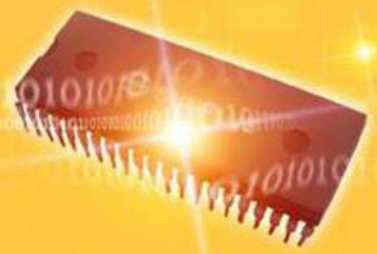
# 嵌入式系统工程师



# LCD原理及应用

# 大纲

- LCD的原理
- 常见LCD连接方式
- RGB接口LCD控制时序
- S5PV210 LCD寄存器简介
- Frame buffer机制

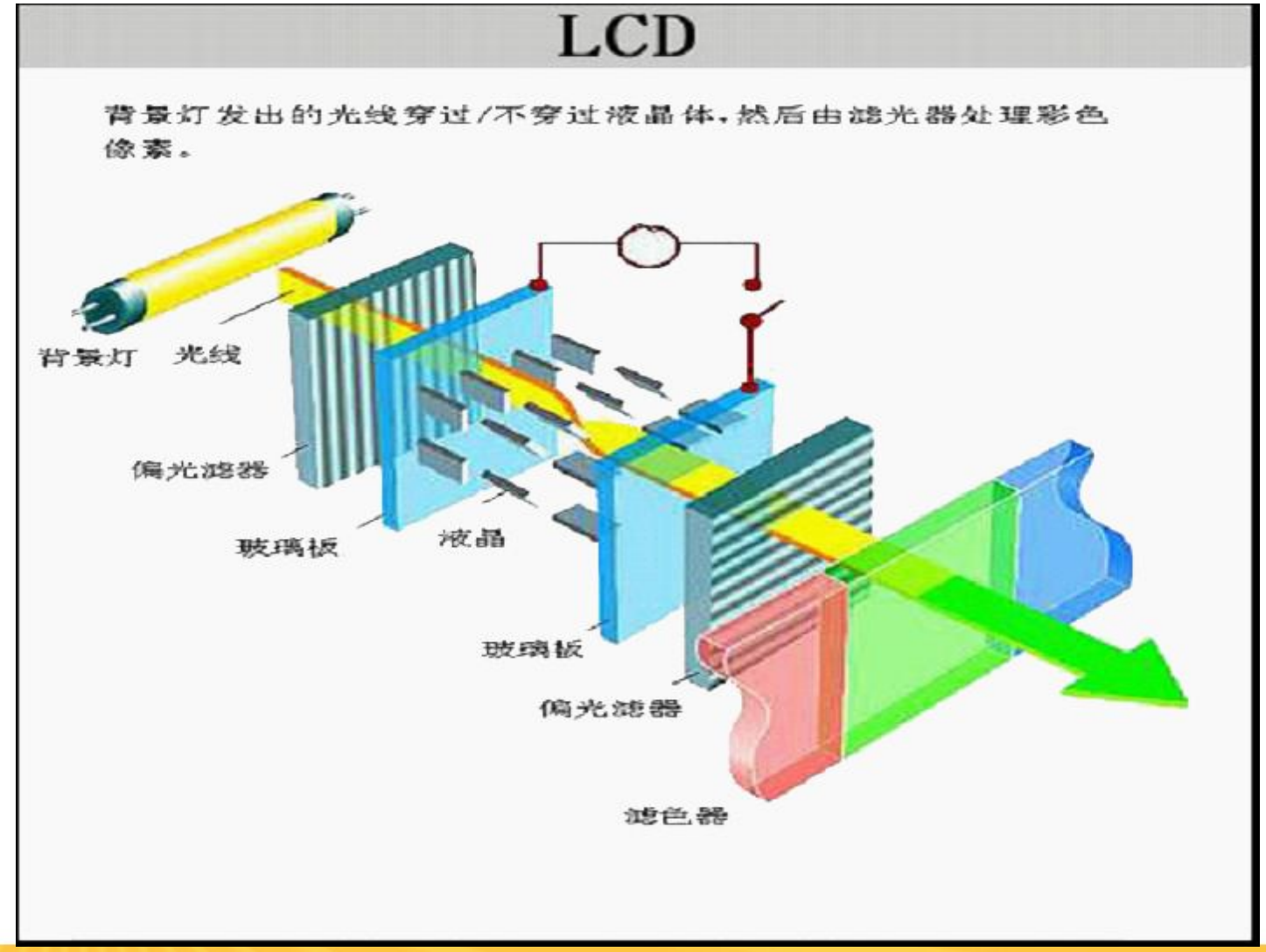
# 大纲

- LCD的原理
- 常见LCD连接方式
- RGB接口LCD控制时序
- S5PV210 LCD寄存器简介
- Frame buffer机制

# LCD原理

- 液晶面板主要是由两块无钠玻璃夹着一个由**偏光板、液晶层和彩色滤光片**构成的夹层所组成
- 液晶是一种规则性排列的有机化合物，它是一种介于固体和液体之间的物质，液晶本身并不能够发光，而是通过控制光线的通过比例来显示图像，因此需要一个光源，液晶只是光线传输所经过的介质
- 偏光板、彩色滤光片决定了有多少光可以通过以及生成何种颜色的光线。

# LCD原理


LCD
背景灯发出的光线穿过/不穿过液晶体，然后由滤光器处理彩色像素。
背景灯
光线
偏光滤器
玻璃板
液晶
玻璃板
偏光滤器
滤色器

# LCD原理

- **TN型液晶屏：**光线90度旋转
  - 只有明暗两种情形（或称黑白），屏不易做很大
- **STN型液晶屏：**光线180~270度旋转
  - 加上彩色滤光片即可显示彩色图像，改变了TN型屏的一些缺点
- **TFT型液晶屏（薄膜晶体管液晶屏）：**
  - 每个液晶像素点都是由集成在像素点后面的薄膜晶体管来驱动
  - 速度高、亮度高、对比度高、分辨率高，显示效果出色

# LCD原理

- 对于一幅图片或者一块LCD一般的描述参数：

  **分辨率、色深、尺寸、PPI等**

  - **分辨率：**一幅图像被称为一帧（frame），每帧有若干行、列的像素数组成，常见的分辨率如下：

    320*240(QVGA)　640*480(VGA)　800*480(WVGA)

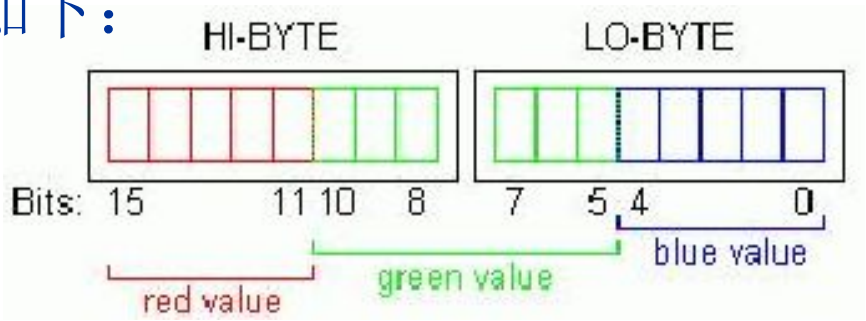
    800*600(SVGA)　1280*800(WXGA)　……

  - **色深(色位)：**每个像素的颜色使用若干位的二进制数据来表示，常见的色深如下：

    RGB565(65K)

    RGB888(16M)

    红、绿、蓝3基色的分量



# LCD原理

- 对于一幅图片或者一块LCD一般的描述参数：

  **分辨率、色深、尺寸、PPI等**

  - **尺寸**：一般液晶屏使用对角线的长度表示屏幕的大小

    单位为英寸，常见的屏幕尺寸为:

    2.0寸、2.5寸、3.0寸、5寸、7寸、17寸、19存…

  - **PPI(pixels per inch)**：在图像中，每英寸所包含的像素数，也是描述一个图片信息的重要属性，图像ppi值越高，画面的细节就越丰富

    常见的PPI值有:72ppi，180ppi和300ppi

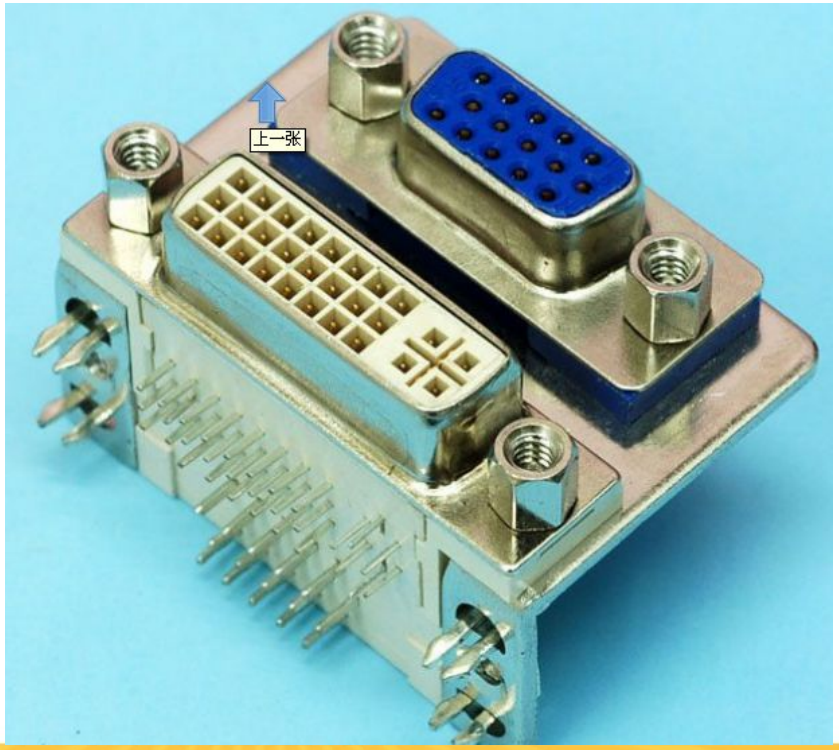
# 大纲

- LCD的原理
- 常见LCD连接方式
- RGB接口LCD控制时序
- S5PV210 LCD寄存器简介
- Frame buffer机制

# 常见LCD连接方式

- PC机领域显示器常见接口有：

  VGA(模拟接口)、DVI(数字接口)、HDMI(数字高清接口)




# 常见LCD连接方式

- **嵌入式领域多LCD常见接口有：**

  MCU模式，RGB模式，SPI模式，VSYNC模式，MDDI模式等。

- **MCU模式：**
  - LCD自带控制器，会解码命令，产生时序信号驱动lcd
  - MCU模式下，数据可以先存到IC内部GRAM后再往屏上写，所以这种模式LCD可以直接接在MEMORY的总线上。
  - 优点是：控制简单方便，无需时钟和同步信号
  - 缺点是：要耗费GRAM，所以难以做到大屏（QVGA以上）

# 常见LCD连接方式

- RGB模式：
  - 分为模拟RGB、 ADC接口、数字RGB接口。
  - RGB接口的TFT LCD，没有内部RAM，具有HSYNC、VSYNC、ENABLE、CS、RESET、RS信号
  - 不带RAM所以资料是直接往屏上写的，不能保存，所以要往屏上不断的写资料，不这样做屏就会变白
  - 需要外部RAM，把资料存在外部RAM里，再往屏上刷，通常连接到接有RAM的CPU上 ,CPU带有LCD的控制器
  - 该模式的屏色彩鲜艳尺寸可以做大，可用于手机、PDA、平板电脑等领域

# 常见LCD连接方式

- **SPI模式：**
  采用较少，连线为CS/，SLK，SDI，SDO四根线，连线少但是软体控制比较复杂。
- **VSYNC模式：**
  该模式是在MCU模式下增加了一根VSYNC（帧同步）信号线而已，应用于运动画面更新
- **MDDI模式：**
  高通公司于2004年提出的接口MDDI（Mobile Display Digital Interface），通过减少连线可提高移动电话的可靠性并降低功耗，这将取代SPI模式而成为移动领域的高速串行接口。

# 大纲

- LCD的原理
- 常见LCD连接方式
- RGB接口LCD控制时序
- S5PV210 LCD寄存器简介
- Frame buffer机制

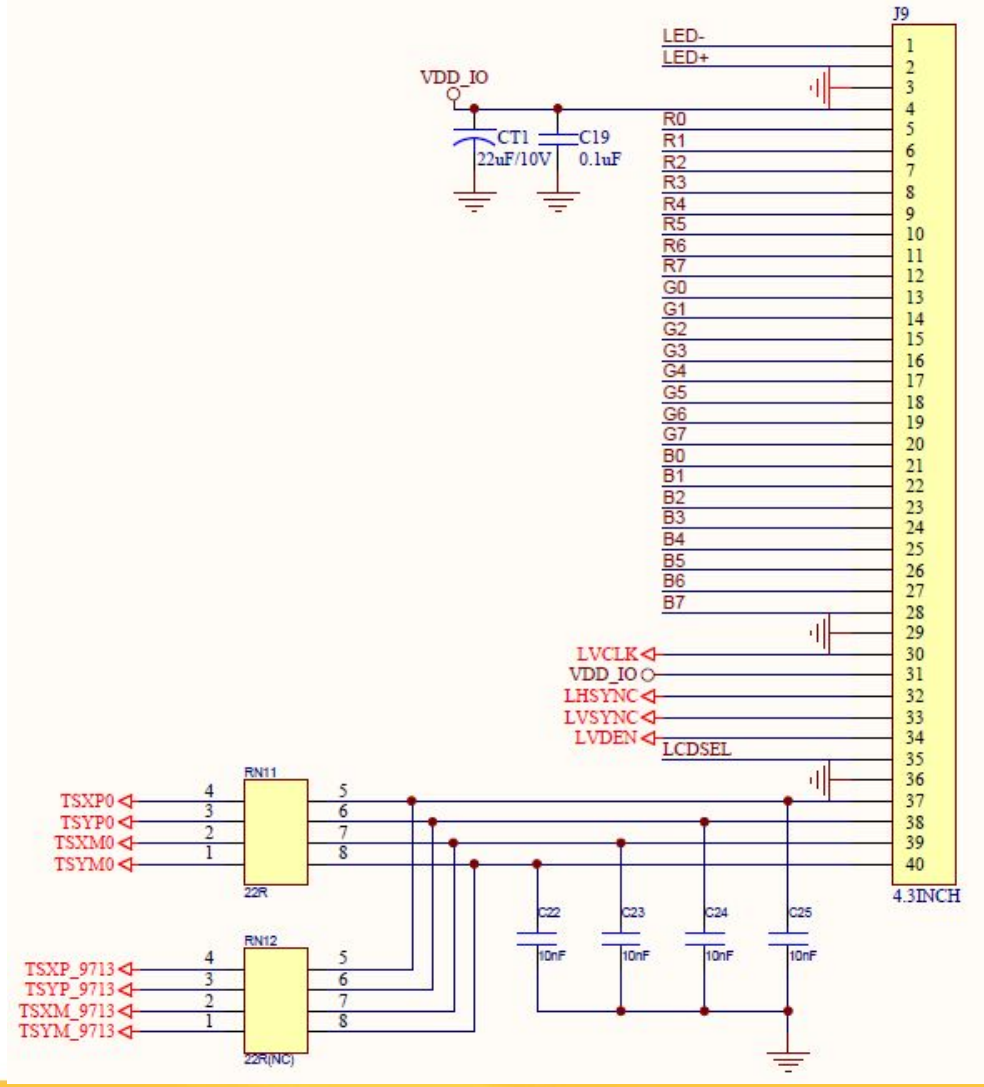
# S5PV210 LCD控制器

➢S5PV210 LCD控制器具有一般LCD控制器功能，产生各种信号、传输显示数据到LCD驱动器

| 信号名称 | 描述 |
| --- | --- |
| VSYNC | 帧（垂直）同步信号 |
| HSYNC | 行（水平)同步信号 |
| VCLK | 像素时钟信号 |
| VD[23:0] | 数据信号 |
| VM(VDEN) | 数据有效信号 |
| PWREN | 电源开关信号 |

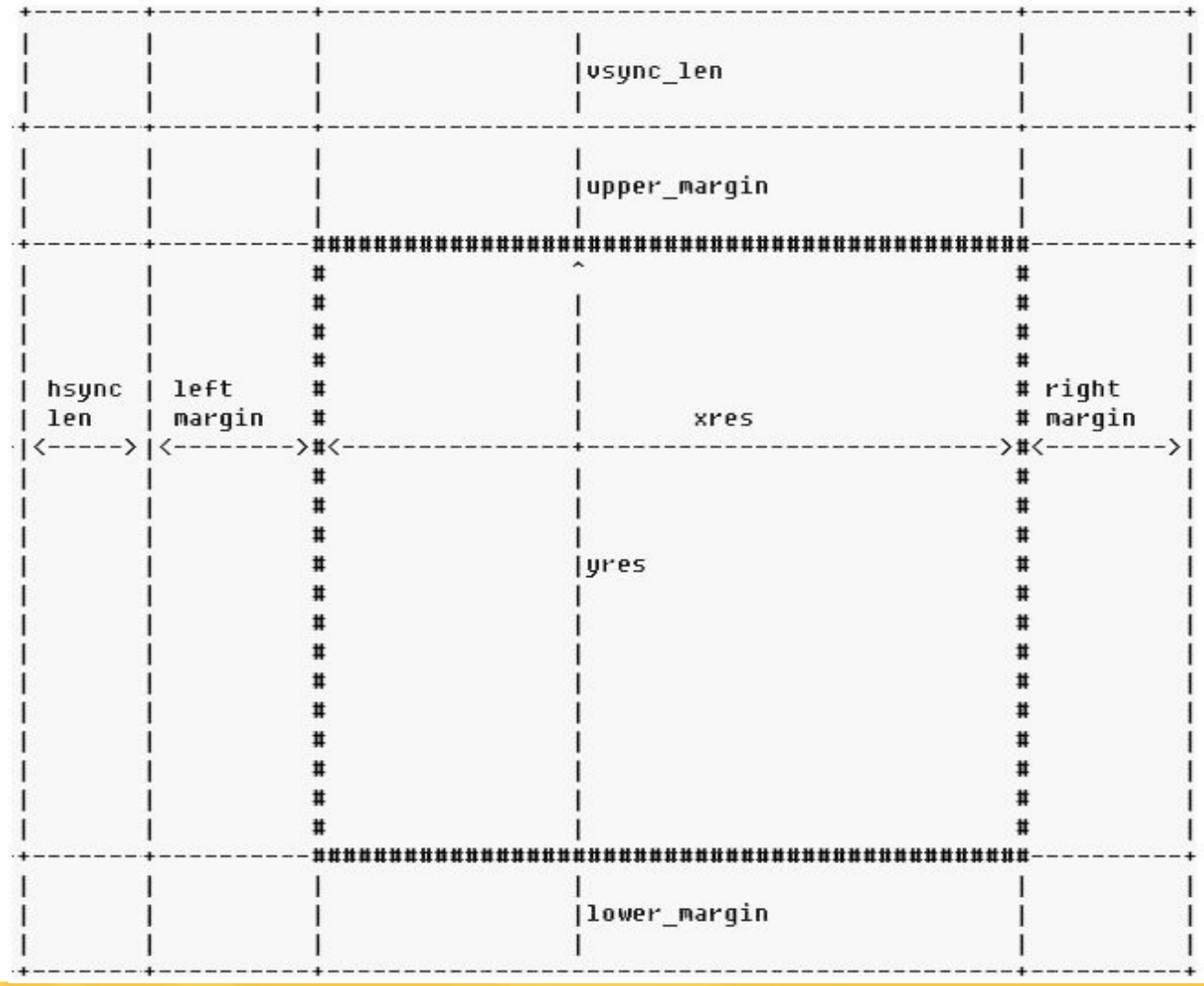
# S5PV210 LCD控制器

- 右图为S5PV210实验仪的LCD接口(详细参看a8底板电路图):

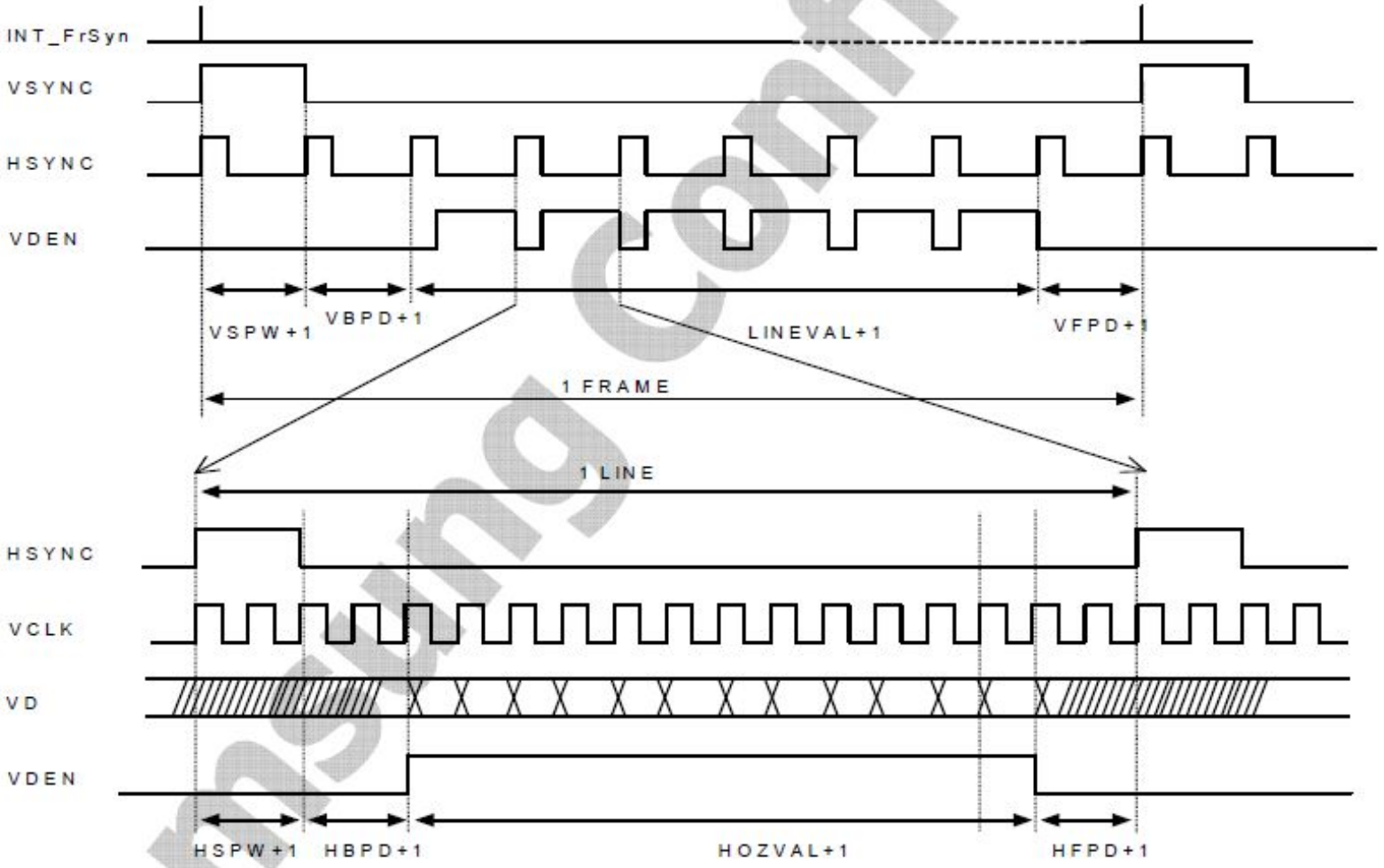


# Frame buffer使用

- LCD屏的显示区域及相关参数：



# S5PV210 TFT 时序



# S5PV210 TFT 时序

- VSYNC信号有效时，代表一桢数据的开始
  - VSPW表示VSYNC信号的脉冲宽度为（VSPW+1），本周期内数据无效
  - VSYNC信号脉冲之后，还需经过(VBPD+1)个行时钟周期，行信号才有效
- HSYNC信号有效时，表示一行数据的开始
  - 同帧扫信号，在经过(HSPW+1)、(HBPD+1)像素时钟后，数据时钟才有效
- 随后发出HOZAL+1个像素的有效数据
  - 行结束时：发出 (HFPD+1)个结束时钟，表示一行结束
  - 帧结束时：发出(VFPD+1)个结束时钟，代表一桢结束

# 时序表参数值

| Item | Symbol | Values | | | Unit | Remark |
|---|---|---|---|---|---|---|
| | | Min. | Typ. | Max. | | |
| Horizontal Display Area | thd | - | 800 | - | DCLK | |
| DCLK Frequency | fclk | 26.4 | 33.3 | 46.8 | MHz | |
| One Horizontal Line | th | 862 | 1056 | 1200 | DCLK | |
| HS pulse width | thpw | 1 | - | 40 | DCLK | |
| HS Blanking | thb | 46 | 46 | 46 | DCLK | |
| HS Front Porch | thfp | 16 | 210 | 354 | DCLK | |
| Vertical Display Area | tvd | - | 480 | - | TH | |
| VS period time | tv | 510 | 525 | 650 | TH | |
| VS pulse width | tvpw | 1 | - | 20 | TH | |
| VS Blanking | tvb | 23 | 23 | 23 | TH | |
| VS Front Porch | tvfp | 7 | 22 | 147 | TH | |

# 大纲

- LCD的原理
- 常见LCD连接方式
- RGB接口LCD控制时序
- S5PV210 LCD寄存器简介
- Frame buffer机制

# LCD控制器专用寄存器

- S5PV210片上集成了LCD控制器，用来产生控制时序，驱动LCD屏工作
- 用户只需要通过寄存器来配置LCD控制器，而不必关心具体时序的产生
- linux内核一般都会带有LCD控制器的相关驱动代码，用户可以不必关心具体寄存器的配置情况，有时只是对时序参数的适当修改

# LCD控制器专用寄存器

- VIDCONX（视频主控制寄存器）主要用来配置视频输出格式与显示使能与否，该组寄存器分以下四个寄存器：
  - VIDCON0:配置视频输出格式，如显示模式、色深等等
  - VIDCON1:控制时序控制信号的状态、极性等
  - VIDCON2:设置RGB输出顺序、YUV等相关参数
  - VIDCON3:设置Gamma、HUE相关参数

# LCD控制器专用寄存器

- VIDTCONX（视频时序控制寄存器）主要配置视频输出时序及显示区域，该寄存器分为以下四个寄存器：
  - VIDTCON0:设置垂直方向各信号的参数，如VBPD、VFPD、VSPW等
  - VIDTCON1:设置水平方向各信号参数，如HBPD、HFPD、HSPW
  - VIDTCON2:设置水平和垂直方向显示区域大小
  - VIDTCON3:相关控制信号的使能配置

# 大纲

- LCD的原理
- 常见LCD连接方式
- RGB接口LCD控制时序
- S5PV210 LCD寄存器设置
- Frame buffer机制
  - Frame buffer概念、使用、驱动框架
  - Platform机制讲解
  - S5PV210下frame buffer实例分析

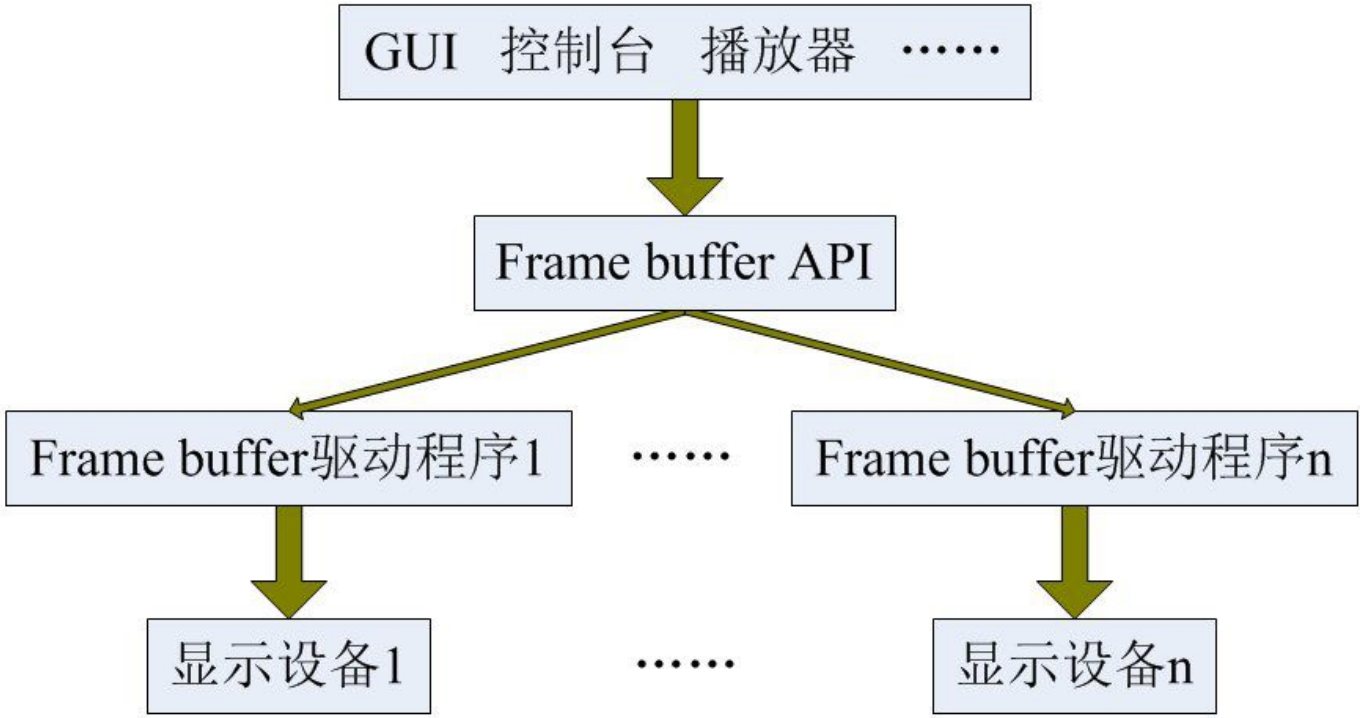
# Frame buffer概念

- 在linux系统中LCD液晶屏在程序中被映射为一个二维数组
- 以分辨率为800*480，色深为32位色，大小为7寸的液晶屏为例：
  - 程序中定义一个800*480(32bit)字节的int型数组来表达此屏幕
  - 把对屏幕的操作等效为对此数组的操作，也就是我们经常听说的framebuffer操作

# Frame buffer概念

- Frame buffer概念：帧缓冲，是Linux视频系统的核心概念
- 帧缓冲的意义：
  - Framebuffer接口允许应用程序与底层图形硬件变化无关
  - 如果应用和显示器驱动程序遵循Framebuffer接口：
    1. 应用程序不用改变就可以在不同类型的视频硬件上运行
    2. 不同类型的液晶屏或驱动器不需要考虑上层应用仅需要实现Framebuffer所要求的接口即可

# Frame buffer概念



# Frame buffer使用

- Frame buffer的使用：
  - 在软件编程中，LCD等效为一个二维数组，我们从驱动接口中仅需要得到以下几个信息即可进行应用：
    - 屏幕分辨率：即800*480或者640*480等
    - 屏幕的色深：即16位色24位色还是32位色等
    - 屏幕等效数组的起始地址
  - 嵌入式中几乎所有的GUI都是基于framebuffer完成的

# Frame buffer驱动框架

- framebuffer驱动框架—（设备号）
  - 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/include/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
  - 帧缓冲设备对应于文件系统下/dev/fb%d设备文件

# Frame buffer驱动框架

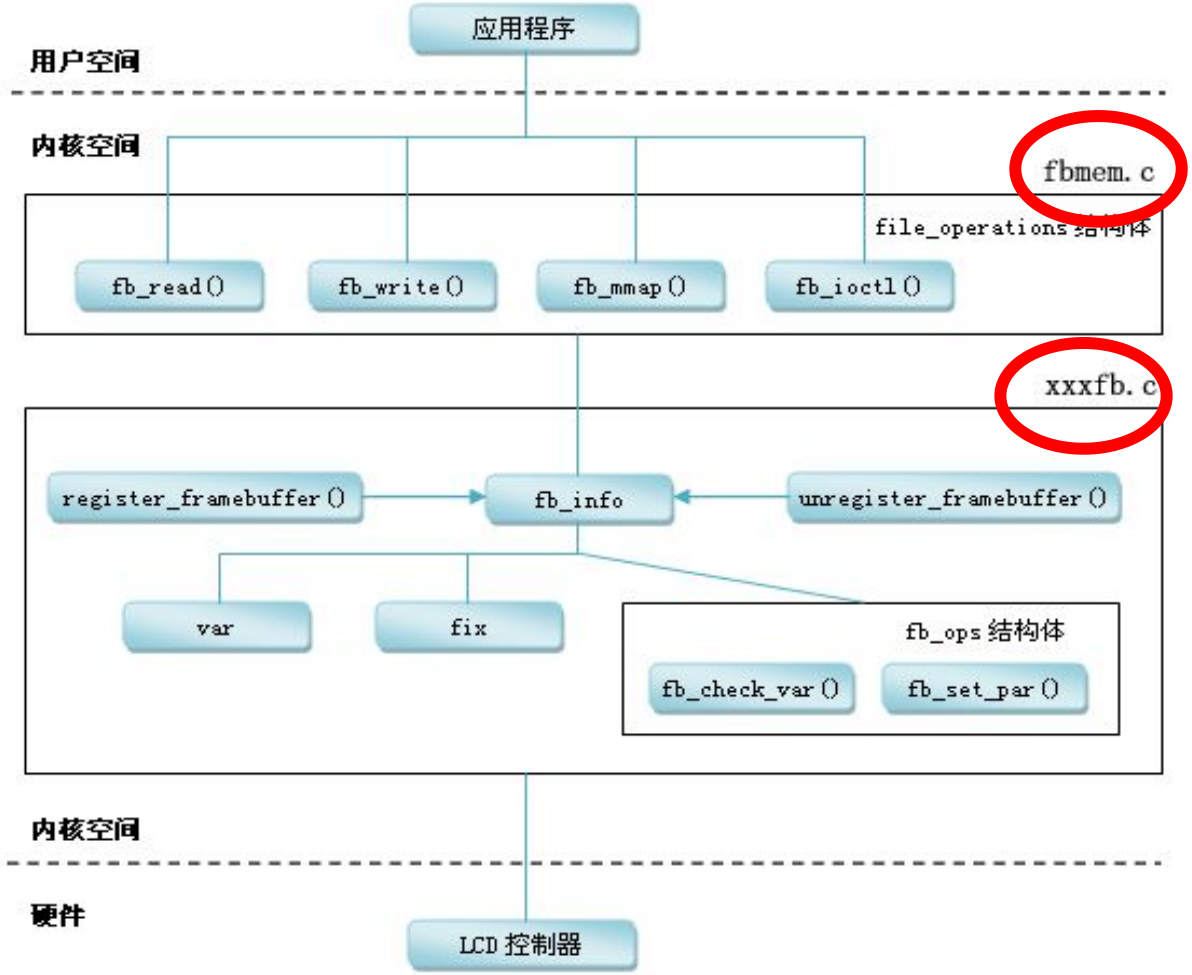
- framebuffer驱动框架—（相关文件）
  - 帧缓冲设备是一个完整的子系统，主要由fbmem.c和xxxfb.c文件组成
    - **fbmem.c文件**

      向上给应用程序提供完善的设备文件操作接口(对FrameBuffer设备进行read、write、ioctl等操作）
      向下提供了硬件操作的函数接口（没有提供实现）
    - **xxxfb.c文件**

      根据具体的LCD控制器硬件实现fbmem.c中给出的接口与具体的硬件相关

# Frame buffer驱动框架

- 帧缓冲设备驱动在Linux子系统中的结构如右图：




帧缓冲设备驱动程序结构图

# Frame buffer驱动框架

- framebuffer驱动框架—（数据结构）
  - 帧缓冲驱动的核心是fb_info结构体
    - 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。
    - 实现一个特定lcd的驱动，根本上就是实现并填充这个结构体
    - 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中定义

# Frame buffer使用

```
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
```

# Frame buffer使用

```
#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;
};
```

# Frame buffer驱动框架

- 其中，比较重要的成员有
  ```
  struct  fb_var_screeninfo var
  struct  fb_fix_screeninfo fix
  struct  fb_ops *fbops
  ```
- 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
- 而fb_fix_screeninfo结构体：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
- fb_ops结构体：对底层硬件操作的函数指针

# Frame buffer使用

```
struct fb_var_screeninfo {
    /*可见解析度*/
    __u32 xres;
    __u32 yres;
    /*虚拟解析度*/
    __u32 xres_virtual;
    __u32 yres_virtual;
    /*除pixclock本身外，其他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*/
        ......
```

# Frame buffer驱动框架

```
struct fb_ops {
    struct module *owner;
    /*打开/释放*/
    int (*fb_open)(struct fb_info *info, int user);
    int (*fb_release)(struct fb_info *info, int user);
    /*对于非线性布局的/常规内存映射无法工作的帧缓冲设备需要*/
    ssize_t (*fb_read)(struct fb_info *info, char __user *buf,
              size_t count, loff_t *ppos);
    ssize_t (*fb_write)(struct fb_info *info, const char __user *buf,
               size_t count, loff_t *ppos);
    /*检测可变参数，并调整到支持的值*/
    int (*fb_check_var)(struct fb_var_screeninfo *var, struct fb_info *info);
    /*根据info->var设置video模式*/
    int (*fb_set_par)(struct fb_info *info);
                    • • • • • •
```

# Frame buffer驱动框架

```
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*/
};
```

凌阳教育官方微信：Sunplusedu

*Tel: 400-705-9680 , BBS: www.51develop.net , QQ群：241275518*