LCD那些事儿

LCD工作原理

LCD即Liquid Crystal Display 液晶显示器。

液晶，得名于其物理特性：它的分子晶体，以液态存在而非固态。它的棒状结构在液晶盒内一般平行排列，但在电场作用下能改变其排列方向。

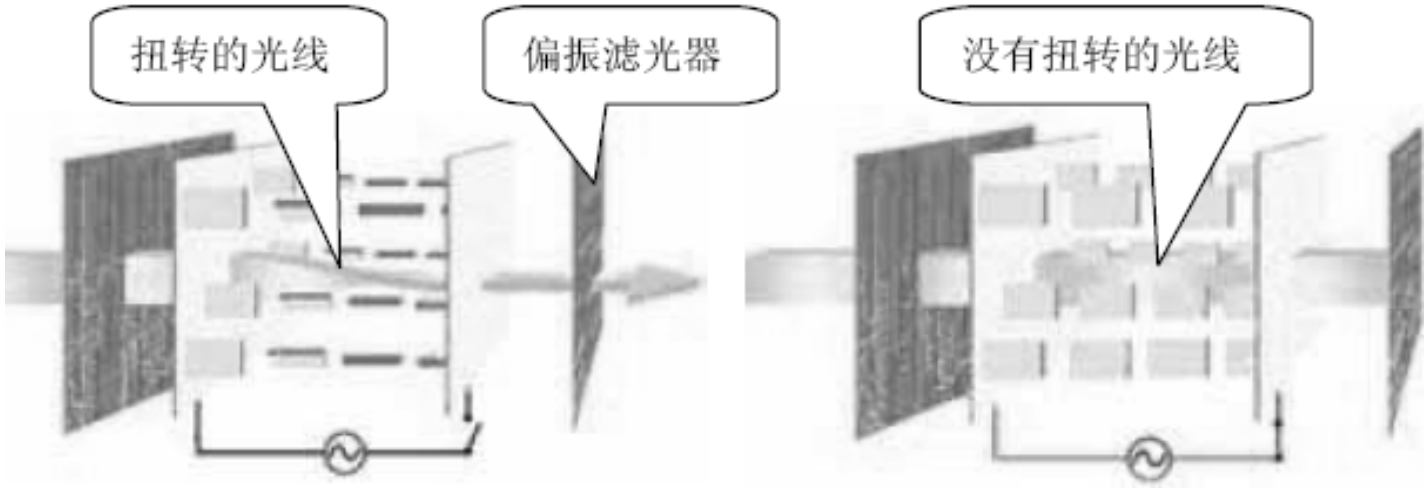


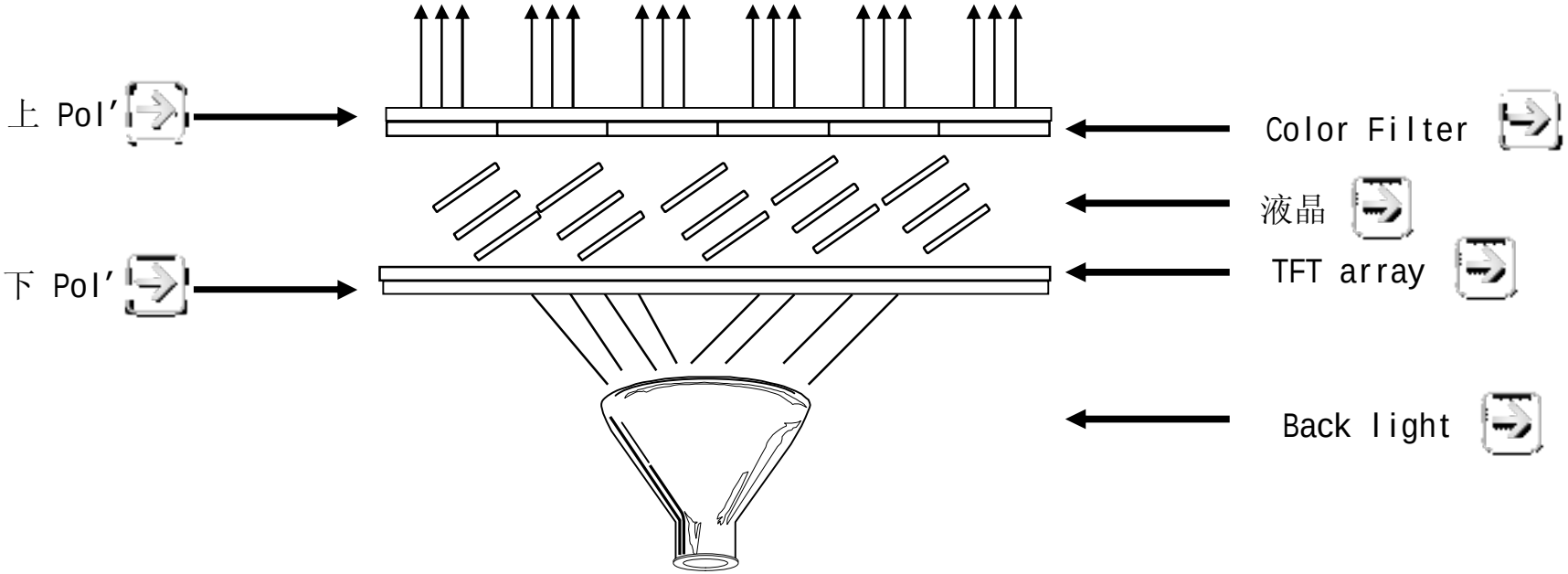
图 A　加电时　　图 B　不加电时

光线穿过与阴断示意图

“ 液晶标识装置．利用随着认可电压液晶透视图的变化，各种装置中发生的多样电子信息变化，使变为开始信息传达的电子元件。 无自身发光性

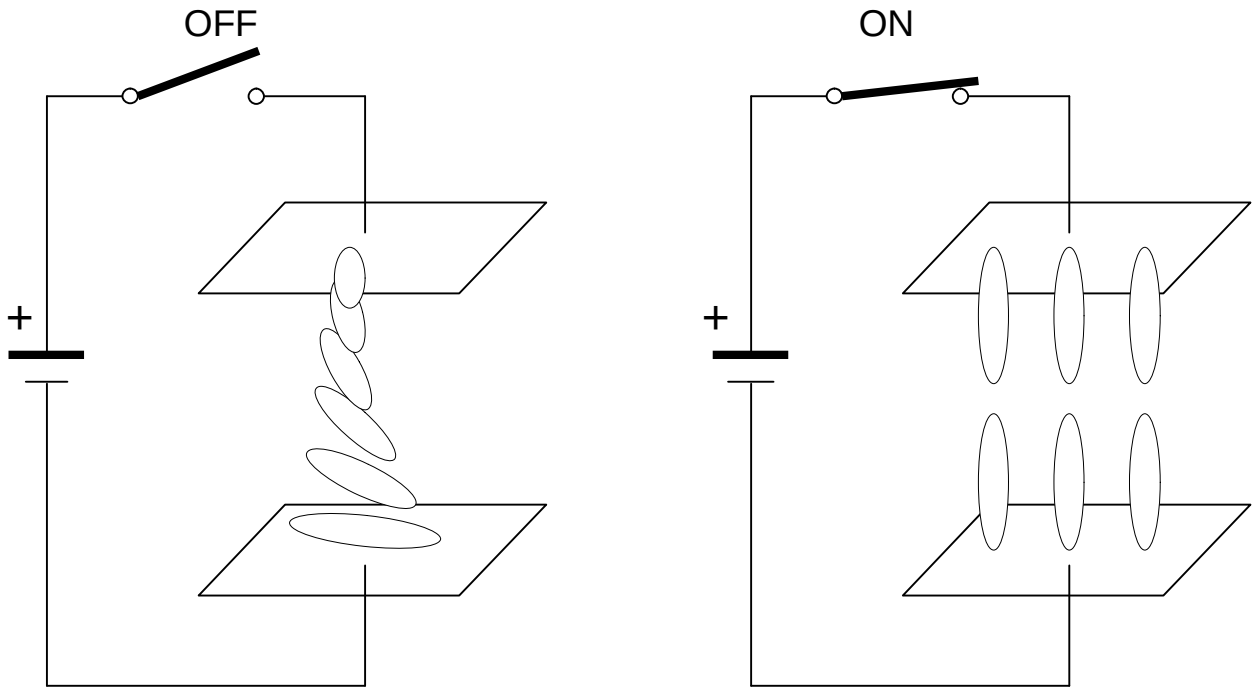
不用借光，但是消费电力小，便携式可以广泛使用的平板显示。．”

(Naver百科词典)

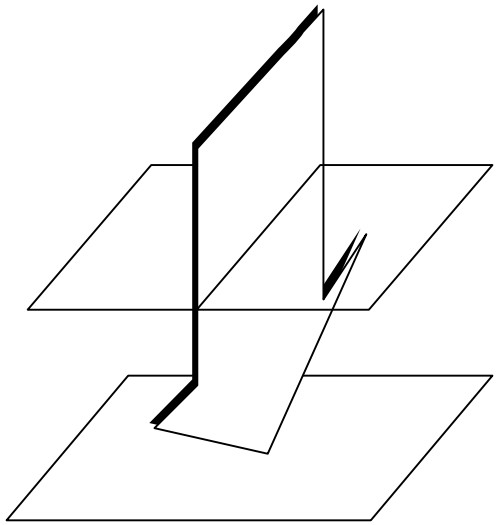


Thin Film Transistor (薄膜场效应晶体管)，是指液晶显示器上的每一液晶象素点都是由集成在其后的薄膜晶体管来驱动。从而可以做到高速度高亮度高对比度显示屏幕信息，TFT-LCD（薄膜晶体管液晶显示器）是多数液晶显示器的一种

LCD显示器的基本原理就是通过给不同的液晶单元供电，控制其光线的通过与否，从而达到显示的目的。

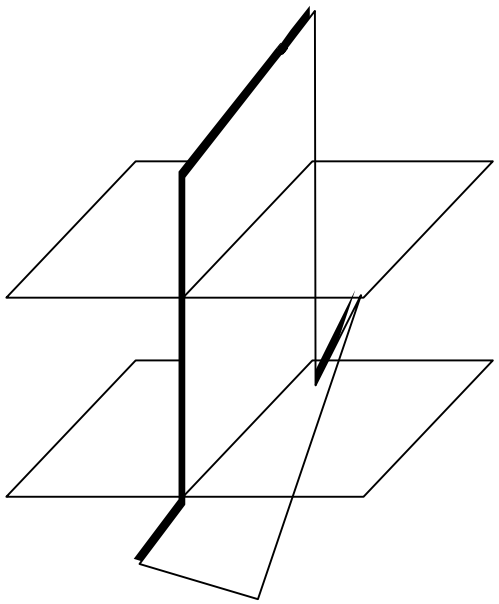


<电压未认可时>　　<电压认可时>



遮光

<方向不一致>



光通过

<方向一致>

LCD的驱动控制归于对每个液晶单元的通断电的控制，每个液晶单元都对应着一个电极，对其通电，便可使光线通过，也有为了省电的需要不通电时光线通过，通电时光线不通过。

液晶显示是一种被动的显示，它不能发光，只能使用周围环境的光，它显示图案或字符只需要很小的能量。

LCD的显示方式：
反射型LCD、透射型LCD、透反射型LCD。

LCD的显示方式还分为正性和负性。
正性：白底黑字，在反射和透反射型显示最佳。
负性：黑底白字，用于透射型，加上背光源，字体清晰，易于阅读。

反射型LCD：

底偏光片后面加了一块反射板，它一般在户外和光线良好的办公室使用，而一般微控制器上使用的LCD为反射式，需要外界提供光源，靠反射光来工作。

透射型LCD：

底偏光片是透射偏光片，它需要连续使用背光源，一般在光线差的环境使用。如笔记本电脑的LCD显示屏，屏后面有一个光源，因此外界环境可以不需要光源。

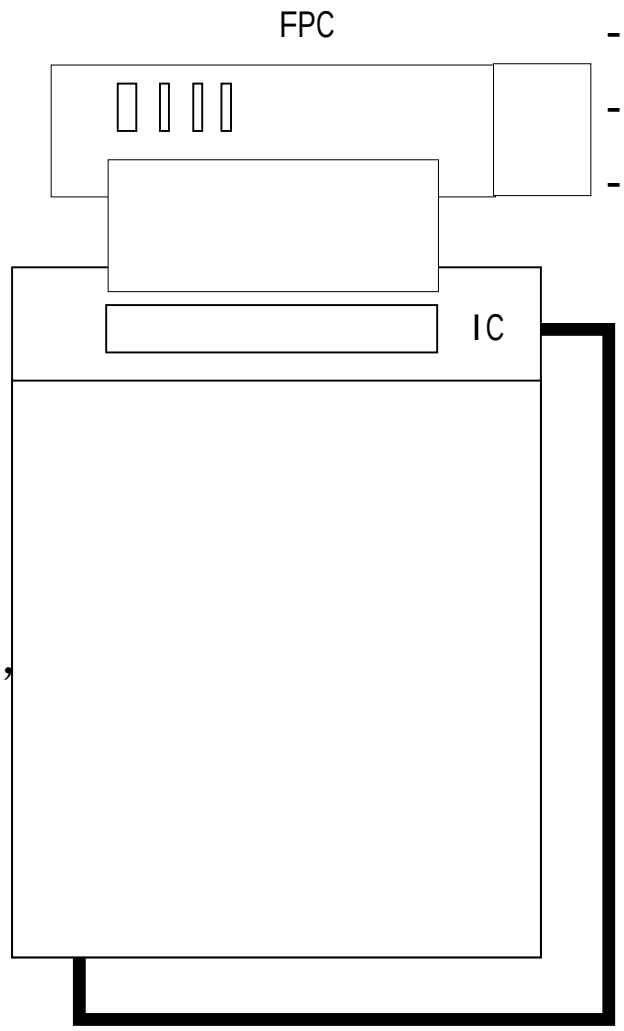
透反射型LCD：

是处于以上两者之间，底偏光片能部分反光，一般也带背光源，光线好的时候，可关掉背光源；光线差的时候可以点亮背光源使用LCD。

LCD-Module 构造：

外装部品 (驱动IC用)
- 为了驱动IC的运转的部品.
- 构成要素
  -> 电源安全化电容
  -> 电源电压生成电容
  -> Panel电压生成电容

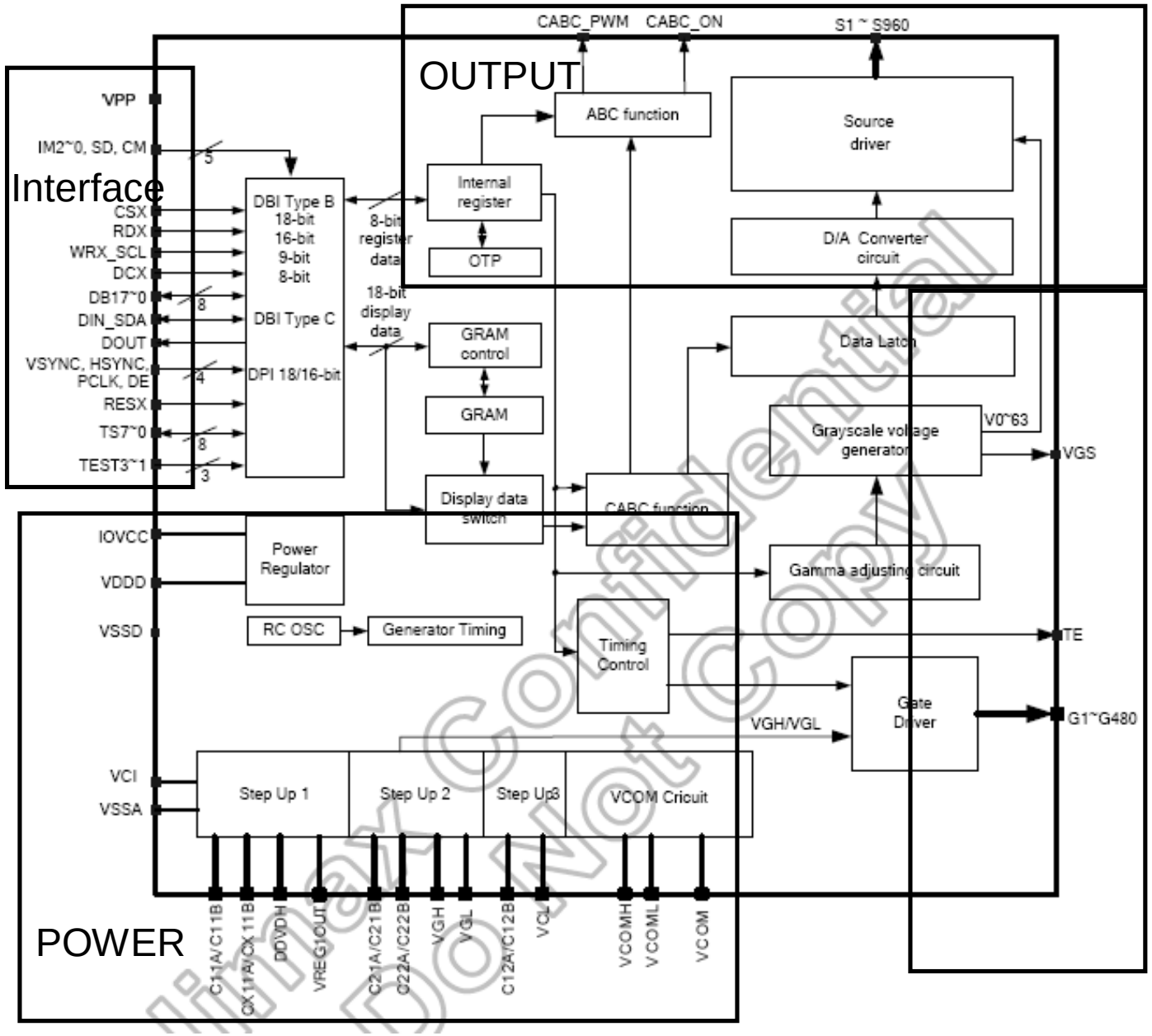
驱动 IC
- 是驱动回路的核心要素.
- 由于使用半导体集成回路工程，可构成小的大小的回路.
- 作用 (相似体信号生成)
  -> Panel驱动数据生成
  -> Panel驱动信号生成
  -> Panel驱动电压生成



输入连接(INTERFACE)
- 是Set和Modul的连接部分.
- 由20~ 40 pin构成.
- 构成要素
  -> 数据信号 (4~18个)
  -> 电子协议(传送方式)
  -> 电源及 GND

Light Guide Plate
-位于LC Panel 下层，以Panel方向，起供给均匀的平面光的作用
-LED 内部 /外部

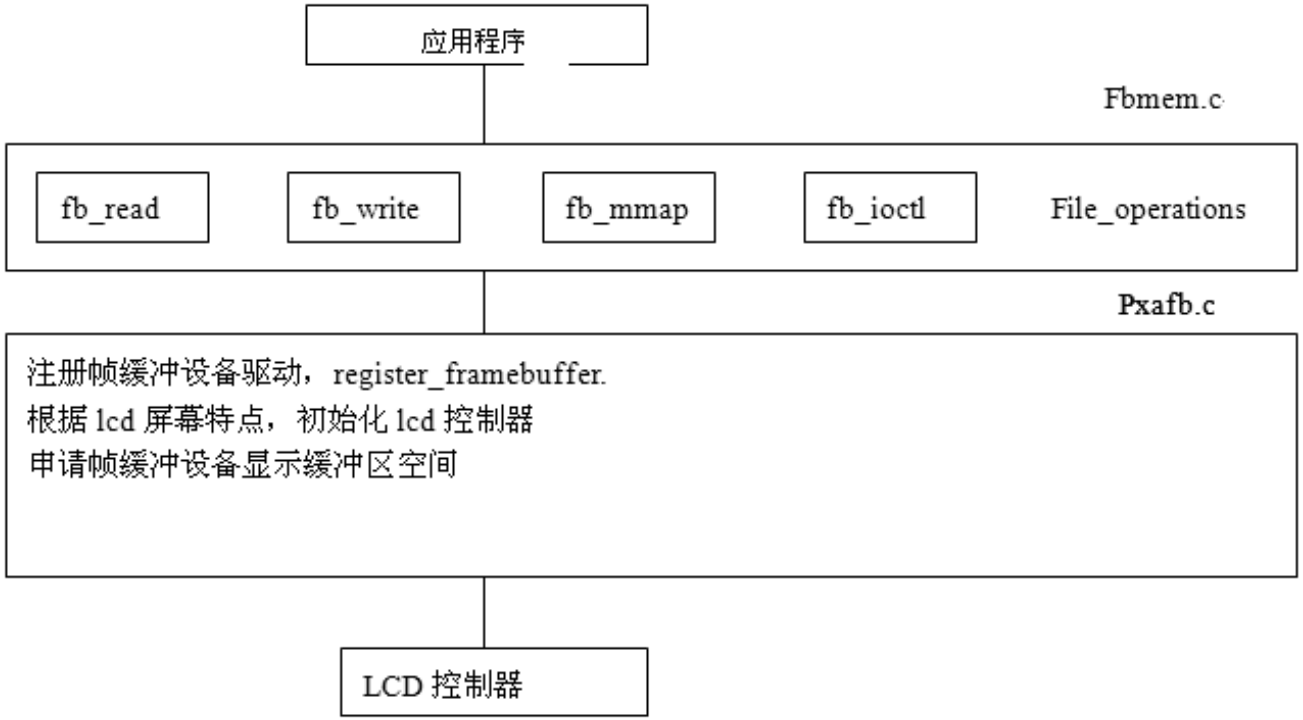
IC BLOCK Diagram(HX8357B)



LCD驱动架构

帧缓冲（framebuffer）
帧缓冲是linux系统为显示设备提供的接口，将显示缓冲区抽象。帧缓冲设备为标准的字符设备，因此我们可以参照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驱动架构的理解，主要部分就是对帧缓冲设备的操作。

帧缓冲设备驱动架构：



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 *///
        struct fb_fix_screeninfo fix;   /* Current fix *///
        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 the 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。他们最终是和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, //使得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。

LCD驱动调试添加一个新的DPI-LCD驱动

LCD驱动原理

LCM_DRIVER hx8357b_lcm_drv =
{
.name = “hx8357b”,
.set_util_funcs = lcm_set_util_funcs,//跟到这个函数的定义会发现是一个结构体LCM_UTIL_FUNCS其中定义了读写、GPIO设置等基本函数。
.get_params = lcm_get_params,//时序设置，也有相应的结构体对于操作函数的定义。LCD_DRV
.init =lcm_init,
.suspend =lcm_suspend,
.resume =lcm_resume,
.update =lcm_update,
.set_backlight =lcm_setbacklight,//背光控制，HX8357B的IC可以控制背光，由于我们在实际工程中有自己的背光控制芯片，所以这个设置基本不用。
};

Disp_drv.c　　Mt6573_disp_drv.c

```
LCM_DRIVER *disp_drv_get_lcm_driver(const char *lcm_name)
{
}

static void disp_dump_lcm_parameters(LCM_PARAMS
*lcm_params)
{
}

 BOOL DISP_SelectDevice(const char* lcm_name)
 {
 }

  BOOL DISP_DetectDevice(void)
  {
  }
```

Mtkfb.c

```
static struct platform_driver mtkfb_driver =
{
    .driver = {
        .name    = MTKFB_DRIVER,
        .bus     = &platform_bus_type,
        .probe   = mtkfb_probe,
        .remove  = mtkfb_remove,
        .suspend = mtkfb_suspend,
        .resume  = mtkfb_resume,
    },
};
```

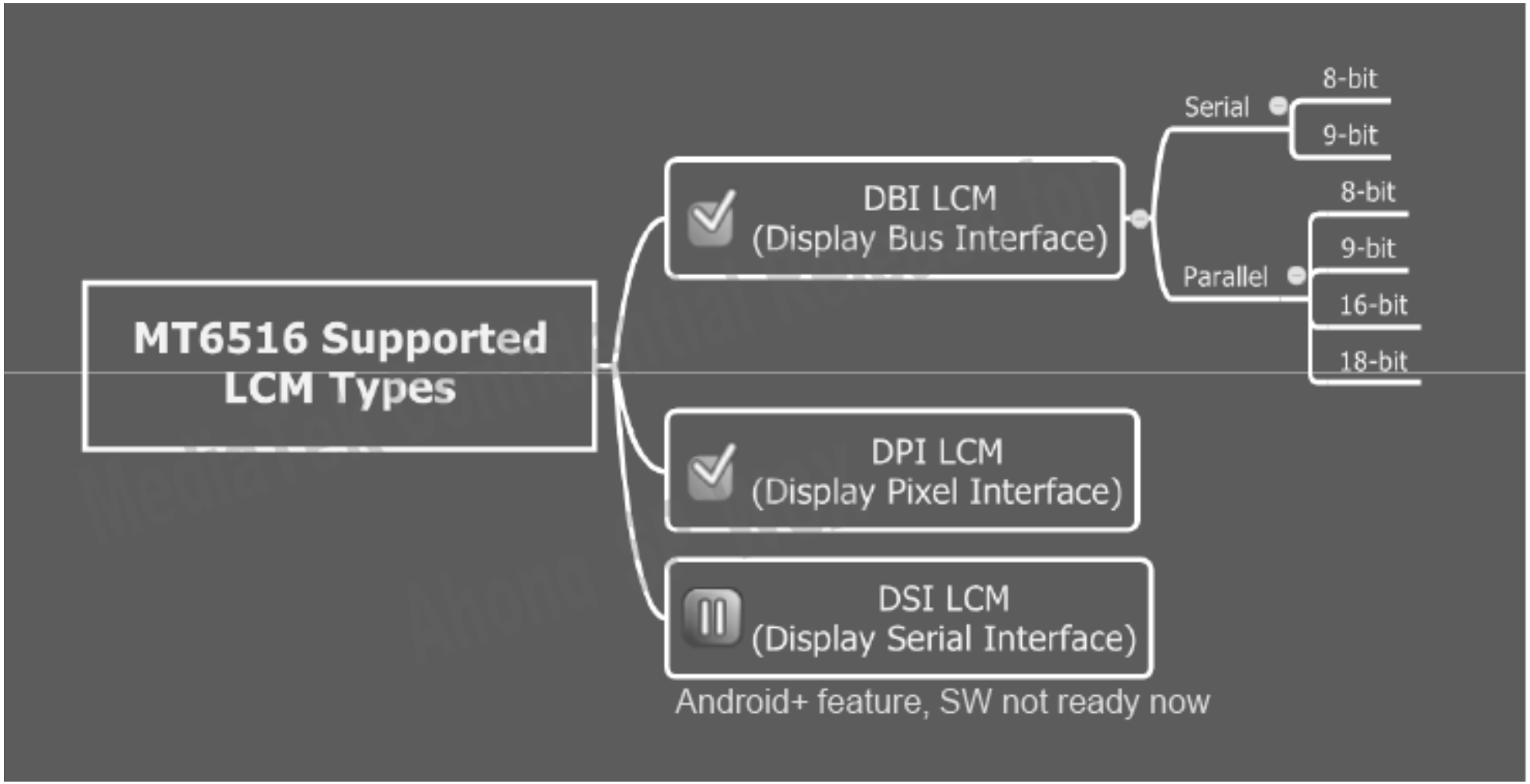
```
module_init(mtkfb_init);
module_exit(mtkfb_cleanup);

MODULE_DESCRIPTION("MEDIATEK framebuffer driver");
MODULE_AUTHOR("Zaikuo Wang <zaikuo.wang@mediatek.com>");
MODULE_LICENSE("GPL");
```

LCD那些事儿

- LCD Interface Introduce
- The DBI & DPI Interface
- Frequently Asked Questions

•LCD Interface Introduce

MT6516、MT6573



一般来说，LCD的数据接口有多少位，LCD屏幕显示的色彩的位数就为2^n

但是一块IC有支持色彩位数的最大值。硬件接口的位数只是决定了在最大值的范围内所采用的色彩位数。

2^6=64
2^{18}=262,144
2^{24}=16.7M

16-bit data extend to 18-bit

Eg：ILI9481

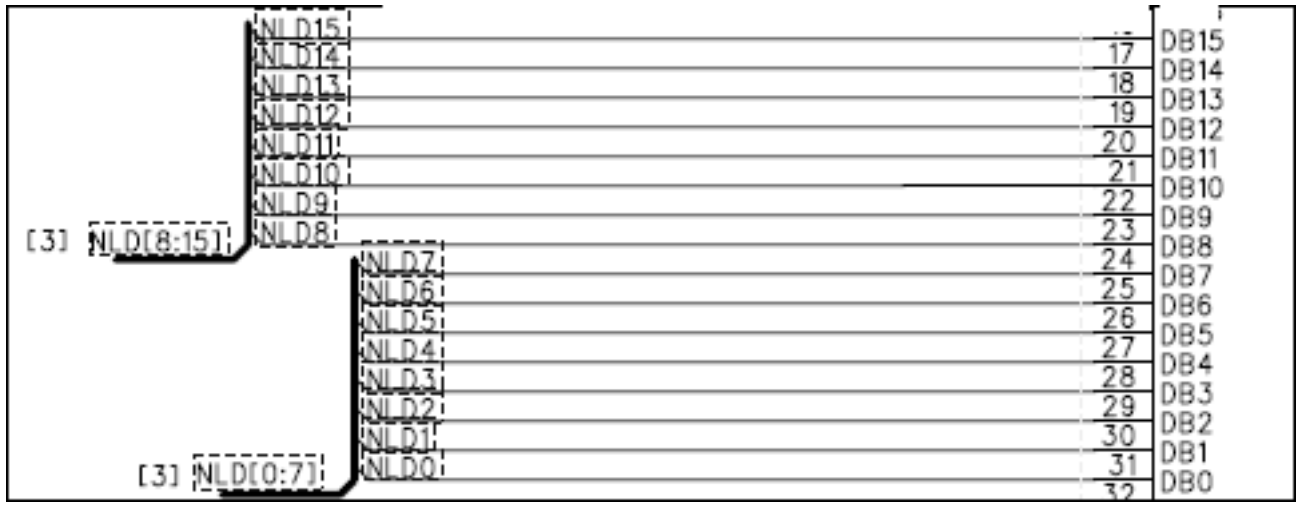
		Frame Memory Data (18bpp)																	
Set_pixel_format	EPF[1:0]	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
18bpp	*	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]
16bpp	2'h0	R[4]	R[3]	R[2]	R[1]	R[0]	0	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[4]	B[3]	B[2]	B[1]	B[0]	0
	2'h1	R[4]	R[3]	R[2]	R[1]	R[0]	1	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[4]	B[3]	B[2]	B[1]	B[0]	1
	2'h2	R[4]	R[3]	R[2]	R[1]	R[0]	R[4]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[4]	B[3]	B[2]	B[1]	B[0]	B[4]

同时，这个硬件接口的位数也决定了LCD的数据的传输方式。例如：RGB的传输方式16-bit意味着RGB是按照565的方式传送，18-bit就是666的方式传送。

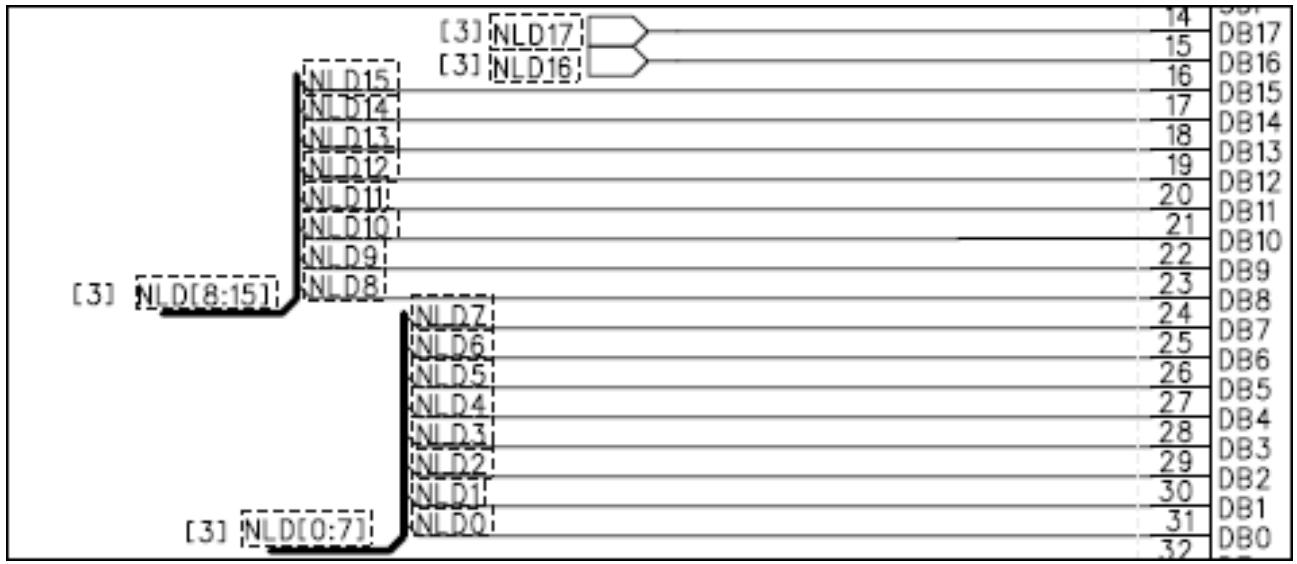
但是如何知道硬件接口的位数呢？

——看原理图

像这种直接就可以数出来它的数据位数：



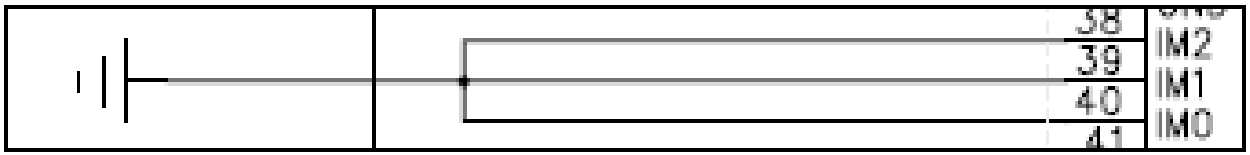
但是如果硬件做了兼容呢？



那就用另一种方法：

IM2	IM1	IM0	MPU-Interface Mode	DB Pin in use	Colors
0	0	0	DBI Type B 18-bit	DB[17:0]	262K
0	0	1	DBI Type B 9-bit	DB[8:0]	262K
0	1	0	DBI Type B 16-bit	DB[15:0]	65K/262K
0	1	1	DBI Type B 8-bit	DB[7:0]	65K/262K
1	0	0	Setting prohibited	-	-
1	0	1	DBI Type C 9-bit	DIN, DOUT	8/262K
1	1	0	Setting prohibited	-	-
1	1	1	DBI Type C 8-bit	DIN, DOUT	8/262K

同样的，看原理图中：



IM0/IM1/IM2都接地，所以判断其为DBI Type B 18-bit

不仅判断出了它对应的硬件接的位数，而且判断出它

是DBI的接口方式。最为直接有效。

•The DBI & DPI Interface

（1）DBI接口

也就是通常所讲的MCU接口，俗称80 system接口。The lcd interface between host processor and LCM device list as below,The LCM driver will repeated update panel display。MCU接口通过并行接口传输控制命令和数据，并通过往LCM模组自带的GRAM（graphic RAM）更新数据实现屏幕的刷新。

以典型的18位数据跟16位数据做说明（8位寄存器控制）

18-bit data bus DB[17:0] interface, IM[2:0] = 000

	Set_pixel_format	DFM	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Command/Parameter Write	*	*											D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
Command/Parameter Read		*											D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]

	Set_pixel_format	DFM	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
18bpp Frame Memory Write	3'h6	*	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]						
Frame Memory Read		*	r[5]	r[4]	r[3]	r[2]	r[1]	r[0]	g[5]	g[4]	g[3]	g[2]	g[1]	g[0]						

如上硬件采用18位数据线，控制命令和参数占用DB0到DB7并行传输，图像数据采用RGB666的格式并行传输。

16-bit data bus DB[15:0] interface, IM[2:0] = 010

	Set_pixel_format	DFM	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Command/Parameter Write	*	*									D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
Command/Parameter Read		*									D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]

	Set_pixel_format	DFM	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
16bpp Frame Memory Write	3'h5	*	R[4]	R[3]	R[2]	R[1]	R[0]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]					
16bpp Frame Memory Read		*	r[4]	r[3]	r[2]	r[1]	r[0]	g[5]	g[4]	g[3]	g[2]	g[1]	g[0]					

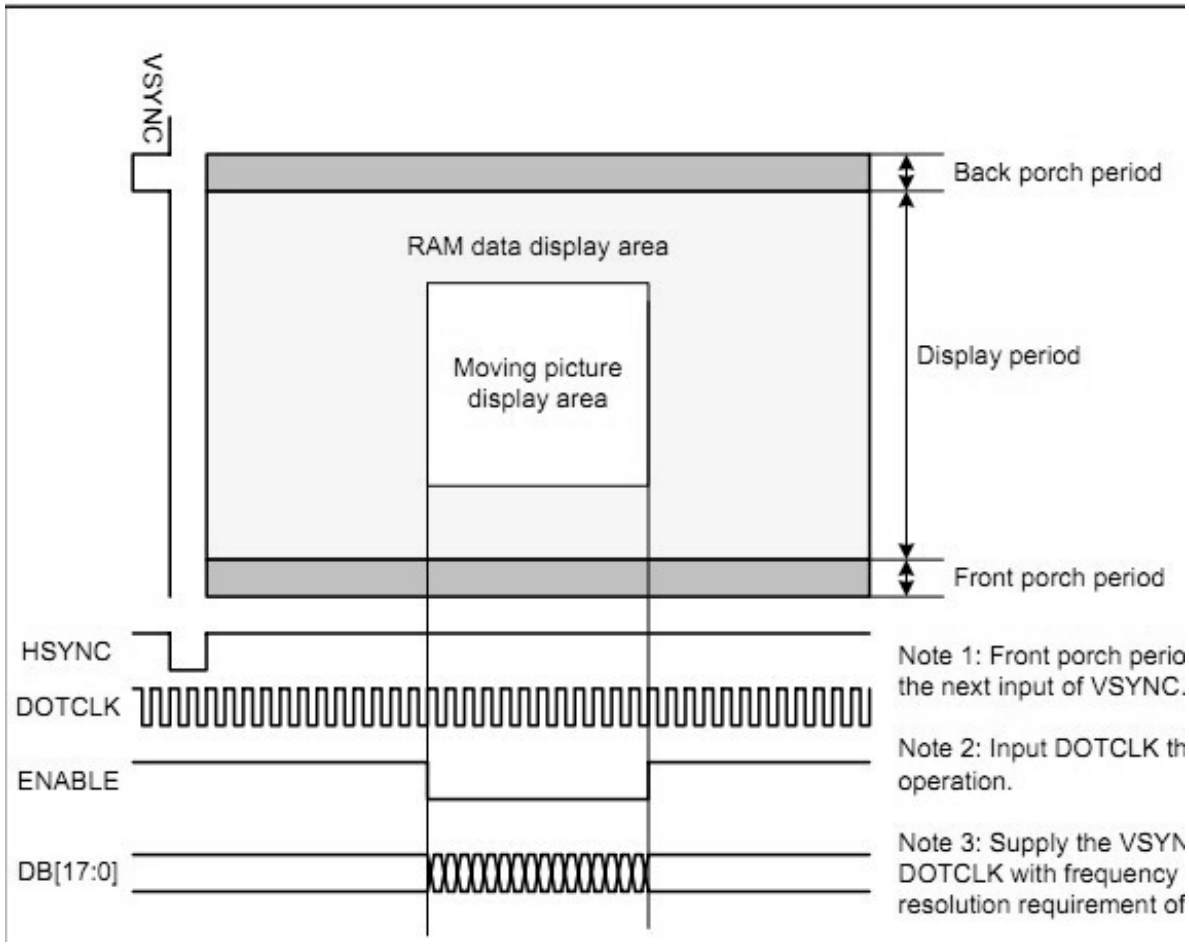
如上硬件采用16位数据线，控制命令和参数占用DB0到DB7并行传输，图像数据采用RGB565的格式并行传输。

（2）DPI接口

也就是通常所说的RGB接口，采用普通的同步、时钟、信号线来传输特定数据，采用SPI等控制线完成命令控制。某种程度上，DPI与DBI的最大差别是，DPI的数据线和控制线分离，而DBI是复用的。

它的信号时序图如下：

VSYNC:帧同步信号
HSYNC:行同步信号



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。

Frequently Asked Questions：

现象：开机第一个LOGO花屏
原因：custom_MemoryDevice.h文件中配置的MCP不正确。
解决办法：检查mediatek/custom/$(project)/peloader/inc/custom_MemoryDevice.h中CS0和CS1要对应硬件的实际连接，结果发现CS1被NC掉了，估在代码中CS1相关设置应注释掉。

现象：开机前短暂花屏或白屏
原因：因为在UBOOT时屏加载LOGO还没有完成时背光就亮起了。
解决办法：在MT65XX_BOARD.C中加载LOGO和打开背光的函数之间加延时。

问题：**往LCD的串行总线发送初始化数据后，LCD模块没有任何反应，无法正确完成初始化**

分析：通常在嵌入式领域中使用的中小型LCD模组都会需要在上电后进行初始化设置，而后才能正常工作。常见的通讯接口有CPU接口和串行总线接口，而串行总线接口又以SPI接口居多。

导致无法初始化的原因通常是两方面：

没有遵循正确的上电RESET流程

SPI通讯控制不符合LCD模组驱动芯片SPEC的要求

造成后者的原因也是多种多样的（w19-2 change lcd）

首先，虽然都是SPI接口，但是，不同的LCD模组，在控制信号的要求上往往都会有细小的不同，有时候，CPU的SPI接口甚至都无法产生LCD模组所需要的特定波形时序。有些LCD模组可能还会有特定的使能信号线控制SPI接口的工作与否。

其次，多数LCD驱动芯片其实都是具有读取寄存器和ID号的功能的，但是很多模组在封装的时候往往没有把芯片SPI接口的SDO信号线引出来，导致无法通过读取寄存器和ID号的方式判断SPI总线的通讯协议是否正确。增大了调试的难度。

解决办法：首先当然是要保证上电顺寻，RESET，使能信号等的正确，而后，如果CPU所提供的SPI接口无法配置到完全和LCD模组要求的时序波形相同，可以采用GPIO口模拟SPI信号的方式来初始化LCD，毕竟初始化地工作量并不多，也不需要经常做，所以通过GPIO模拟，对CPU占用率几乎没有影响。

最后，如果能够将SDO口引出，尽量引出，能给前期调试带来很大方便，也有利于将来自动判断LCD类型，根据不通模组自动加载不同驱动。

LCD的调试中，延时特别重要，一定要确定延长的时间足够，特别是更改电压寄存器后面的延时。

1. 初始化前需要一个延时（大概为10ms），使Reset稳定；
2. 如果出现花屏现象，很大的可能是总线速度问题；
3. 如果屏幕闪动比较明显，可以通过调整电压来稳定，一般调节的电压为VRL、VRH、VDV和VCM；这些电压也可以用来调节亮暗（对比度）；
4. 调节对比度时，也可以通过调节Gamma值来实现，要调节的对象为：PRP、PRN、VRP、VRN等；
5. 注意数据是8位、16位时，写命令和数据的函数注意要变化；
6. 如果调试时发现LCD的亮度有问题，首先检查（考虑）提供给LCD的电流是否一致，再考虑调节电压。
7. 开机花屏问题，最简单的处理方式就是在INIT结束的地方增加一个刷黑屏的功能。
8. 如果随机出现白屏问题，一个可能是静电问题，把LCD拿到头发上擦几下，如果很容易出现白屏那肯定就是静电问题了。

谢谢大家！