# AM335x 在 Uboot 增加自定义的命令控制 LCD 功能

--Eggsy Pang

一．时钟域的配置 & LCD 时钟源的选择

二．LCD 控制寄存器的配置（时序、LCD 工作模式、DMA）

三． 一副图片如何生成一个十六进制的数组

四．在 Uboot 里增加 LCD 显示功能

五．通过添加 Uboot 的自定义命令来控制 LCD

本文的内容如下：

一. 简单介绍一下 AM335x 的时钟域的概念，然后讲解如何配置 LCDC 的时钟
二. 讲解 LCDC 的寄存器的内容和如何根据所选的 LCD 屏的特性进行时序上的配置
三. 介绍用 tool 将任意图片生成十六进制的数据数组
四. 在新版本的 processor SDK v3.0 中，在 uboot 阶段如何实现增加 LCD logo 的显示
五. 基于第四点内容，增加 Uboot 的命令，效果是在 Uboot 进入命令模式，使用自定义的命令控制 LCD 屏的开、关和复位

如果对 AM335x 的 LCD 时序以及 LCD 配置流程清楚的读者，可以跳过一二三章的内容，直接阅读四五章 LCD 在 Linux 的 Uboot 阶段的配置流程。

## 一．时钟域的配置 & LCD 时钟源的选择

首先介绍一下时钟域的概念。时钟域的作用是为了更好地管理各个外设模块，让它们能够独立的控制又能够互相的沟通与合作。说的有点正经，简单来说就是，时钟域就相当于由很多开关组成，时钟的总开关，模块唤醒开关，睡眠开关等等，比如不用的模块就把对应的开关关闭，但又不影响其他模块的正常工作，比如需要唤醒某个模块，那就开启唤醒开关，这样做是为了达到低功耗、灵活使能、独立控制的效果。

举个例子吧，比如 LCD 控制器和 DDR 这两个模块，DDR 存放着 LCD 要显示的图片数据，但是在某个时刻我们不需要 LCD 显示，为了低功耗，我们会把 LCD 的功能关掉，但是 DDR 模块依然需要正常工作，保留数据，这时候时钟域就发挥作用了，仅仅把 LCD 的时钟关闭，DDR 的时钟依然存在。上面所说的模块时钟叫做 Function Clock，主要是为模块正常工作提供时钟源。那 DDR 和 LCD 怎么沟通呢，也就是说如何把 DDR 保存的图片数据传给 LCD 控制器呢？无非就是 DDR 往相应的 buffer 填数据，LCD 控制器从 buffer 里取数据，很明显，这个过程需要 clock 来进行，这个

clock 也是时钟域提供的，这种 clock 就叫做 interface clock，主要功能是为模块内部沟通提供时钟源。

那么如何配置 LCD 的时钟域呢？也就是说怎么找对应的时钟开关呢？看下图：

（摘取 AM335x Technical Reference Manual）

**Figure 10-1. L3 Topology**

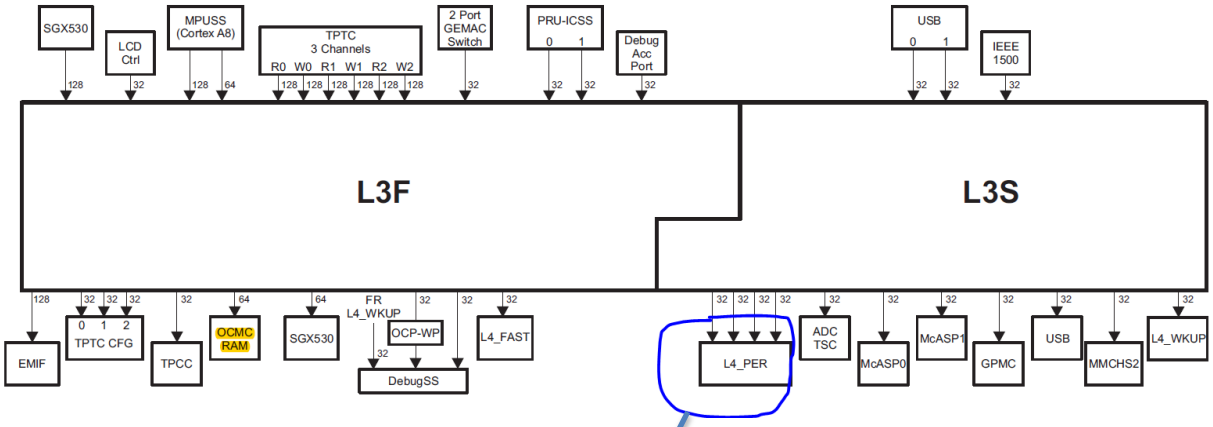
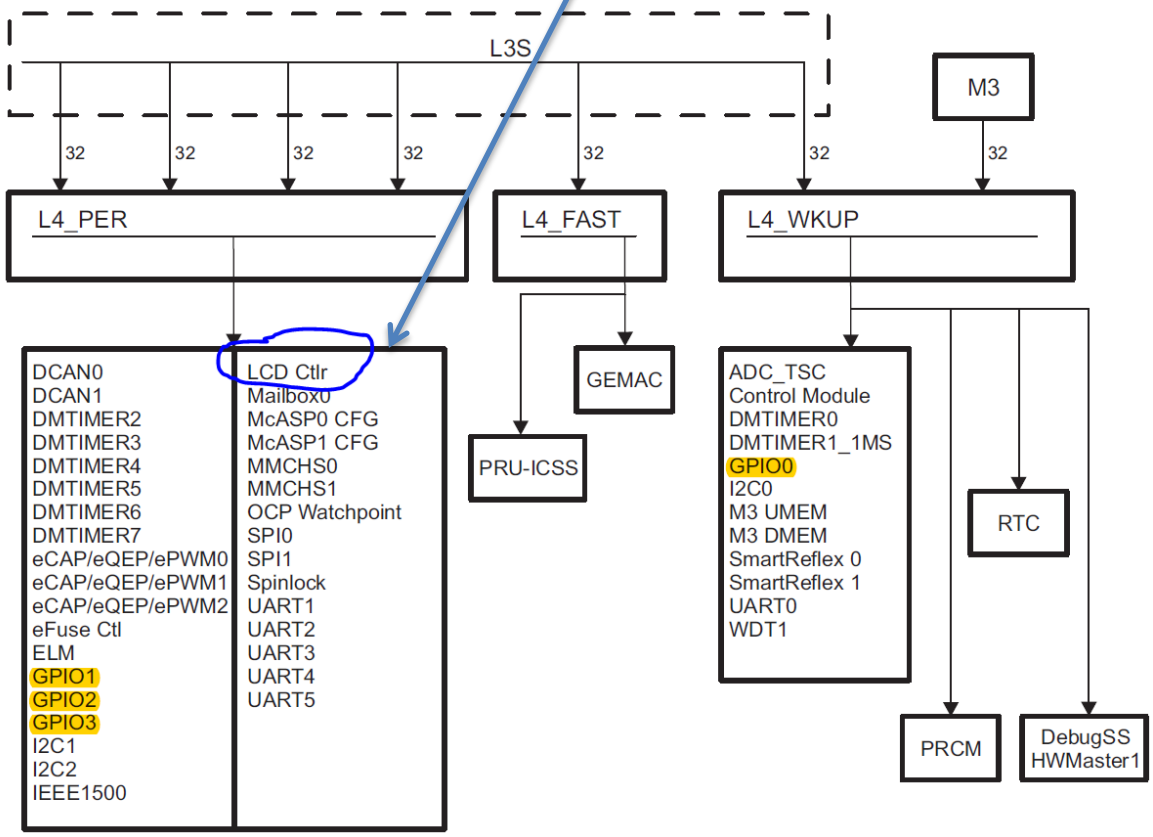



**Figure 10-2. L4 Topology**



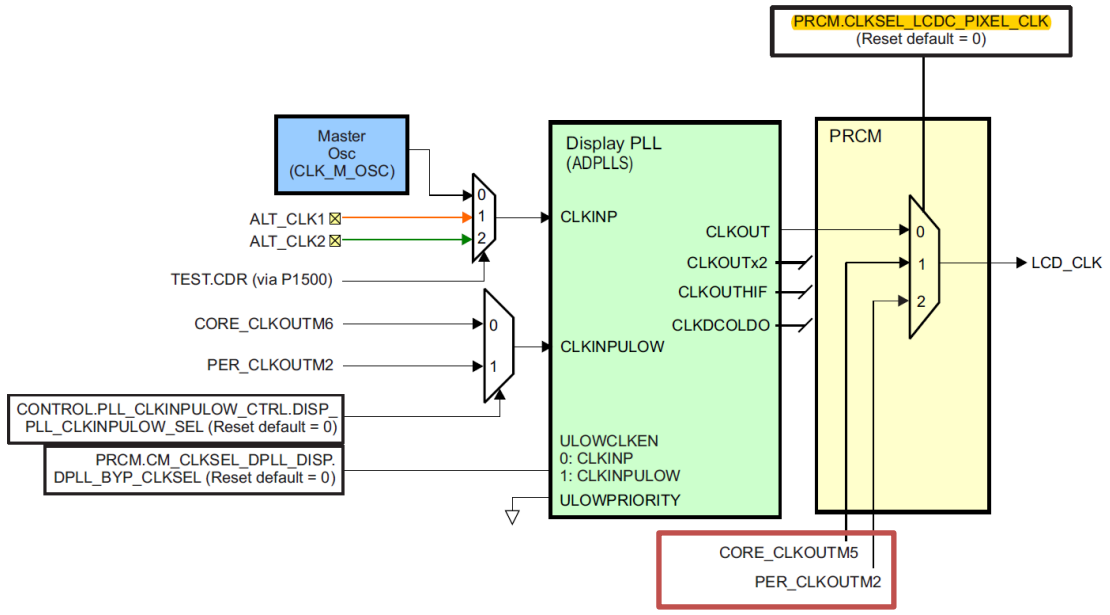
很清晰的看到，蓝色圈圈，也即 LCD 控制器所在地方，它挂在 L3s 的 L4 PER 上，换句话说，L3 的时钟域包含了 L3s 的时钟域，L3s 的时钟域又包含 L4 PER(又名 L4LS)的时钟域，L4 PER 的时钟域包含了 LCD 控制器的时钟域，所以我们要使能 LCD 控制器的模块和它的时钟，首先要使能 L3 的 clock 和模块，再使能 L3s 的模块和 clock，然后使能 L4 PER 的 clock 和模块，最后为 LCD 控制器选择 clock。就像一层层地往下打开开关，一层一层的往下使能，是不是很像打开一个多重文件夹一样，那么如何使能呢？很简单，就是在相应的寄存器的某个位上写“1”。这些寄存器大部分位于 PRCM 的 CM_PER Registers 上，Technical Reference Manual 的第八章有详细的介绍，个人觉得这一章节内容相当重要。下面通过 LCD 时钟初始化程序进行剖析：

```
void LCDModuleClkConfig(void)
{
 // HWREG（y）=x 表示往地址为 y 的寄存器写的值是 x
 //软件唤醒 L3 的 clock
   HWREG(SOC_PRCM_REGS + CM_PER_L3_CLKSTCTRL) |= CM_PER_L3_CLKSTCTRL_CLKTRCTRL_SW_WKUP;

 //软件唤醒 L3s 的 clock
   HWREG(SOC_PRCM_REGS + CM_PER_L3S_CLKSTCTRL) |=  CM_PER_L3S_CLKSTCTRL_CLKTRCTRL_SW_WKUP;

 //使能 L3 模块
   HWREG(SOC_PRCM_REGS + CM_PER_L3_CLKCTRL) |= CM_PER_L3_CLKCTRL_MODULEMODE_ENABLE;
 //软件唤醒 L4LS 的 clock
   HWREG(SOC_PRCM_REGS + CM_PER_L4LS_CLKSTCTRL) |= CM_PER_L4LS_CLKSTCTRL_CLKTRCTRL_SW_WKUP;
 //使能 L4LS 模块
   HWREG(SOC_PRCM_REGS + CM_PER_L4LS_CLKCTRL) |= CM_PER_L4LS_CLKCTRL_MODULEMODE_ENABLE;

 //此处选择 DISP PLL CLKOUTM2 作为 LCDC PIXEL 的时钟源，LCDC PIXEL 的时钟源有三种可选，接下来
 //会详细说明
   HWREG(SOC_CM_DPLL_REGS + CM_DPLL_CLKSEL_LCDC_PIXEL_CLK) =
CM_DPLL_CLKSEL_LCDC_PIXEL_CLK_CLKSEL_SEL1;
 //使能 LCDC 模块
   HWREG(SOC_PRCM_REGS + CM_PER_LCDC_CLKCTRL) |= CM_PER_LCDC_CLKCTRL_MODULEMODE_ENABLE;

 //前面的 L3s L3 L4 L4LS LCDC 相关的时钟域唤醒后，while 等待这些时钟全部打开
   while(!(HWREG(SOC_PRCM_REGS + CM_PER_L3S_CLKSTCTRL) &
         CM_PER_L3S_CLKSTCTRL_CLKACTIVITY_L3S_GCLK));

   while(!(HWREG(SOC_PRCM_REGS + CM_PER_L3_CLKSTCTRL) &
         CM_PER_L3_CLKSTCTRL_CLKACTIVITY_L3_GCLK));

   while(!(HWREG(SOC_PRCM_REGS + CM_PER_OCPWP_L3_CLKSTCTRL) &
        (CM_PER_OCPWP_L3_CLKSTCTRL_CLKACTIVITY_OCPWP_L3_GCLK |
         CM_PER_OCPWP_L3_CLKSTCTRL_CLKACTIVITY_OCPWP_L4_GCLK)));

    while(!(HWREG(SOC_PRCM_REGS + CM_PER_L4LS_CLKSTCTRL) &
        (CM_PER_L4LS_CLKSTCTRL_CLKACTIVITY_L4LS_GCLK |
         CM_PER_L4LS_CLKSTCTRL_CLKACTIVITY_LCDC_GCLK)));
}
```

对上述 LCDModuleClkConfig 函数的红色部分进行说明

我们的 AM335x 对 LCD_CLK 的设置，是比较灵活的。因为 LCDC 的 clock 有三种时钟源可以选择，如下图所示：

**Figure 8-13. Display PLL Structure**




首先看第一幅图的右半边部分 PRCM，可以看到可选 LCDC_CLK 的三种时钟源分别是：

1. PER_CLKOUTM2

基于 Starterware 的 LCD 显示例程，就是使用的 PER_CLKOUTM2 作为时钟源，该时钟为 192MHz（其实 PER_CLKOUTM2 时钟也是可以灵活配置的，但是不推荐，因为它是属于时钟数比较核心的时钟，如果这个一改变，其他的外设也要跟着变，影响比较大，所以对于该路时钟老老实实用 192MHz 就好，这方面描述可以参考 TRM 的 Table 8-24. Per PLL Typical Frequencies(MHz)，所以这给很多人的直观印象就是我们的 LCD_CLK 只能从 192MHz 分频，事实上还是可以有更多选择的。

2. CORE_CLKOUTM5

这路时钟是 250MHz 的（类似于 PER_CLKOUTM2，也可更改配置但不推荐，参考 TRM 的 Table 8-22. Core PLL Typical Frequencies (MHz)

3. 从 Display PLL 倍频后获取

这时请看第一幅图的左边 Display PLL 部分，一般 Display PLL 会选择外部晶振(比如 24MHz 晶振)作为它的时钟源（Resource Clock），然后进行相关的配置，实现倍频，倍频公式：

LCD_CLK= Resource Clock * DPLL_Mult / (N +1) /M2

其中 DPLL_Mult、N 的值由 CM_CLKSEL_DPLL_DISP 寄存器决定，如下图所示。M2 默认为 1。

**Table 8-110. CM_CLKSEL_DPLL_DISP Register Field Descriptions**

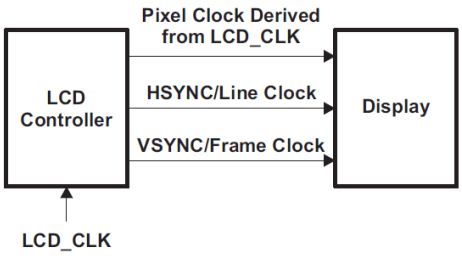
| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 31-24 | RESERVED | Rreturns0s | 0h | |
| 23 | DPLL_BYP_CLKSEL | R/W | 0h | Select CLKINP or CLKINPULOW as bypass clock<br>0h (R/W) = Selects CLKINP Clock as BYPASS Clock<br>1h (R/W) = Selects CLKINPULOW as Bypass Clock |
| 22-19 | RESERVED | Rreturns0s | 0h | |
| 18-8 | DPLL_MULT | R/W | 0h | DPLL multiplier factor (2 to 2047).<br>This register is automatically cleared to 0 when the DPLL_EN field in the *CLKMODE_DPLL* register is set to select MN Bypass mode. (equal to input M of DPLL<br>M=2 to<br>2047 => DPLL multiplies by M).<br>0h (R/W) = 0 : Reserved<br>1h (R/W) = 1 : Reserved |
| 7 | RESERVED | Rreturns0s | 0h | |
| 6-0 | DPLL_DIV | R/W | 0h | DPLL divider factor (0 to 127) (equal to input N of DPLL actual division factor is N+1). |

所以 Clock 的配置选择是不是很灵活呢。那么如何设置去选取这三个不同的时钟源呢？很简单，在 CM_DPLL 寄存器（0x44E0_0500)偏移量为 34h 的 CLKSEL_LCDC_PIXEL_CLK 寄存器中，1-0 位就是对这个时钟的选择配置。可参考上述程序 LCDModuleClkConfig 函数的红色部分。

## 二．LCD 控制寄存器的配置（时序、LCD 工作模式、DMA）

获取 LCDC_CLK 后，要和 LCD 屏沟通，还需要配置 piexl、HSYNC 和 VSYNC 时钟，这三者时钟都带来自于 LCDC_CLK 这个时钟源。如右图所示。

**Figure 13-3. Input and Output Clocks**




那么在哪里进行配置呢？

答案是在 LCD 控制器的寄存器（地址是 0x4830_E000）上进行配置，如右下图所示，除可配置时钟外还可以配置 DMA、中断等。

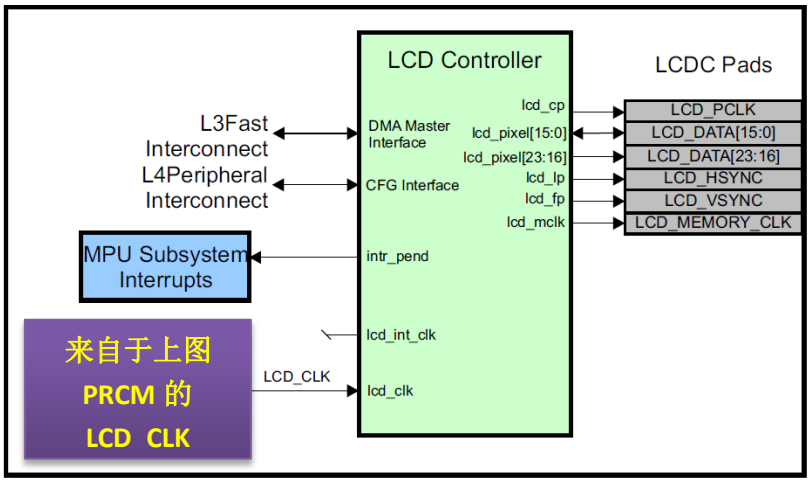



**Figure 13-2. LCD Controller Integration**

配置流程：

1. 首先查找所需要配置的 LCD 屏的 datasheet
2. 从中获取重要参数：piexl 时钟的大小，行宽，列宽，Hsw Hbp hfp vsw Vbp Vsw

对应的配置接口在lcd/rasterDisplay.c下的RasterHparamConfig()和RasterVparamConfig()函数

摘取某一个 LCD 屏的 datesheet 的内容作为配置例子，如下图所示：

| Item Sy | mbol | Min. | Typ. | Max. | Unit | Note |
|---|---|---|---|---|---|---|
| DCLK cycle | time | 25 | - | - | ns | |
| DCLK frequency | fclk | - | 30 | 40 | MHz | pxl_clk：像素时钟 |
| DCLK pulse duty | Tcwh | 40 | 50 | 60 | % | |
| VSD setup time | Tvst | 8 | - | - | ns | |
| VSD hold time | Tvhd | 8 | - | - | ns | |
| HSD setup time | Thst | 8 | - | - | ns | |
| HSD hold time | Thhd | 8 | - | - | ns | |
| Data setup time | Tdsu | 8 | - | - | ns | |
| Data hold time | Tdhd | 8 | - | - | ns | |
| DE setup time | Tesu | 8 | - | - | ns | |
| DE hold time | Tehd | 8 | - | - | ns | |
| Horizontal display area | thd | - | 800 | - | Tcph | width：行宽 |
| HSD period time | th | - | 928 | - | Tcph | |
| HSD pulse width | thpw | 1 | 48 | - | Tcph | hsw * = 48 -1 |
| HSD back porch | thb | - | 40 | - | Tcph | hbp * = 40 -1 |
| HSD front porch | thfp | - | 40 | - | Tcph | hfp * = 40 -1 |
| Vertical display area | tvd | - | 480 | - | th | height：列宽（高度） |
| VSD period time | tv | - | 525 | - | th | |
| VSD pulse width | tvpw | - | 3 | - | th | vsw * = 3 -1 |
| VSD back porch | tvb | - | 29 | - | th | vbp = 29 |
| VSD front porch | tvfp | - | 13 | - | th | vfp = 13 |

下面简单介绍一下这些参数代表什么意思。

首先先介绍 TFT-LCD 系统的构成。它主要由三部分构成：TFT-LCD 控制器、TFT-LCD 驱动器和 TFT-LCD 屏。TFT-LCD 控制器就集成在我们的 AM335x 中的，而购买的 LCD 屏一般都包含剩下的 TFT-LCD 驱动器和 TFT-LCD 屏这两个部分。

我们都知道，一般的 TFT-LCD 的显示刷新频率是 60Hz，也就是每秒钟显示 60 帧画面。而我们视频播放帧率是 23~30fps，我们一般不会利用 LCD 控制器每秒送 60 帧图像的 RGB 信息给 LCD，而是每秒钟送 20~30 帧的 RGB 信息给 LCD，让其显示。其实，我们是通过 LCD 控制器把 RGB 数据送给 LCD 驱动器，LCD 驱动器就把它放到缓存中，然后以 60fps 的速度送给 LCD 屏显示。可能有人会问，视频帧率有多大，LCD 就显示多快，不行吗？不行，由于液晶分子有一种特性，就是不能够一直固定在某一个电压不变，不然时间久了，液晶分子就遭到了破坏。所以要以一定的频率（通常是 60Hz）不停的刷新 LCD 屏，然后在帧消隐的时间里，变换显示电压极性，达到保护液晶的目的。

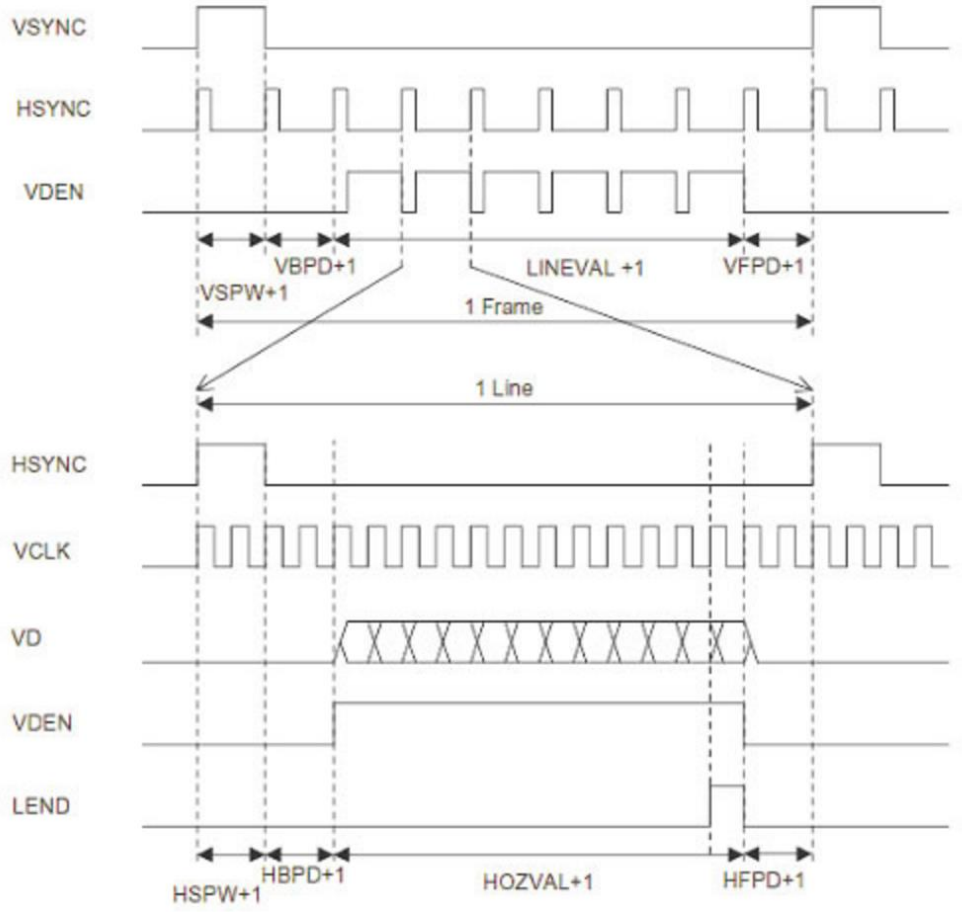
对于 PAL 制式的 CRT 显示器，帧频是 25Hz，场频是 50Hz，这时，视频帧率和显示器刷新率是相同的。而我们在嵌入式当中使用 TFT-LCD 显示器，视频帧率一般是不等于显示器帧率的。所以，LCD 控制器时序和 LCD 驱动器时序（LCD 显示扫描时序）虽然表示方法大体相同，但有实质上的区别，**LCD 控制器时序**控制着**视频帧率**；**LCD 驱动器时序**控制着**显示器刷新率**。这是大家在学习 LCD 的时候最容易混淆的地方。

好，如果还对 LCD 存在疑惑之处，欢迎一起讨论。下面对上述的重要参数进行见解。见下图。其中 VSYNC 是帧同步信号，VSYNC 每发出 1 个脉冲，都意味着新的 1 屏视频数据开始发送。而 HSYNC 为线同步信号，每个 HSYNC 脉冲都表明新的 1 扫描线视频数据开始发送。而 VDEN 则用来标明视频数据有效，VCLK 是像数时钟，一个 VCLK 会带来一位 RGB。并且在帧同步以及线同步的头尾都必须留有消隐时间，例如对于 VSYNC 来说消隐时间是（VSPW+1）+（VBPD+1）+（VFPD+1）；HSYNC 亦类同。对于 LCD 的时序分析，如下图所示：

其中：

VSPW：Vertical sync pulse width，也就是 VSYNC 处于高电平时的线（HSYNC）的数目
VFPD：Vertical front porch delay
VBPD：Vertical back porch delay

HSPW：Hertical sync pulse width，也就是 HSYNC 处于高电平时的 VCLK（像素 CLOCK）的数目
HFPD：Hertical front porch delay
HBPD：Hertical back porch delay




ok，接着上述的配置流程

3. 最后一个步骤就是对 AM335x 的 LCD 控制器的寄存器进行相关操作，让 LCDC 能够配合 LCD 屏的特性进行时序上的沟通，通过程序来分析其流程：

```
static void SetUpLCD(void)
{
  /* 使能相关时钟域，在前面已经分析该函数 */
  LCDModuleClkConfig();
  /* 相应的引脚分配，可以使用 TI 的工具 Pin mux 进行配置，快、准、可视化 */
  LCDPinMuxSetup();
 /*前面只是使能相关时钟域，还剩下最后一个时钟开关需要打开，该开关控制在 LCDC 寄存器 */
  RasterClocksEnable(SOC_LCDC_0_REGS);
  /* 配置 pclk 等上述所说的重要参数前，需要先关闭 LCDC 的功能 */
  RasterDisable(SOC_LCDC_0_REGS);

  /* 配置像素时钟 pclk =30M，写到第二参数，第三个参数是 LCDC_CLK,这里选择 100M */
  RasterClkConfig(SOC_LCDC_0_REGS, 30000000, 100000000);

/* 配置 LCD 控制器的 DMA，此处选择 FIFO 的阈值是 8 个字节，DMA 双通道乒乓模式，详细请看 13 章内容 */
  RasterDMAConfig(SOC_LCDC_0_REGS, RASTER_DOUBLE_FRAME_BUFFER,
            RASTER_BURST_SIZE_16, RASTER_FIFO_THRESHOLD_8,
            RASTER_BIG_ENDIAN_DISABLE);

  /* 配置 LCD 的模式：TFT or STN , 彩色 0r 黑白等 */
  RasterModeConfig(SOC_LCDC_0_REGS, RASTER_DISPLAY_MODE_TFT_UNPACKED,
             RASTER_PALETTE_DATA, RASTER_COLOR, RASTER_RIGHT_ALIGNED);

   /* 配置 LCDC 的 CLOCK 工作极性，也即帧信号或像素时钟是高电平有效还是低电平有效，同步信号是上升沿有效还是下降沿有效*/
  RasterTiming2Configure(SOC_LCDC_0_REGS, RASTER_FRAME_CLOCK_LOW |
                         RASTER_LINE_CLOCK_LOW  | RASTER_PIXEL_CLOCK_HIGH|
                         RASTER_SYNC_EDGE_RISING|RASTER_SYNC_CTRL_ACTIVE|
                         RASTER_AC_BIAS_HIGH    , 0, 255);

  /* 配置 LCD 屏关于水平方向的参数，如上图摘取某 LCD datasheet 上有写：800 个水平像素点，HSPW =48 hfp=40，hbp=40 */
  RasterHparamConfig(SOC_LCDC_0_REGS, 800, 48, 40, 40);

  /* 配置 LCD 屏关于垂直方向的参数，如上图摘取某 LCD datasheet 上有写：480 个垂直像素点，VSPW =3，Vfp=13，Vbp=29 */
  RasterVparamConfig(SOC_LCDC_0_REGS, 480, 3, 13, 29);

  RasterFIFODMADelayConfig(SOC_LCDC_0_REGS, 128);
```

上面主要列举一下配置 LCDC 寄存器常用的函数接口，你可以理解为是 API，其实这些 API 里面很简单，仅仅是对 LCDC 寄存器的不同偏移量的地方进行相关的赋值，若想知道详细内容，从源代码进行学习（待会我会发布 LCD 驱动的源代码），结合 TRM 的第十三章一起分析，效果更佳。

完成上述步骤，基本上已经完成了 90%的配置流程，接下来就是，把你想显示的图片数据往 LCDC 的 DMA 上扔，然后使能 LCD 控制器，就可以显示了。那么怎么往 DMA 扔数据呢？

操作如下：

```
/* DMA 工作在乒乓球模式，双通道传送，DMA0 和 DMA 1 会轮流传送 image 的数据给
LCD 显示，每个 DMA 通道每次传送 8 个字节，image 是由图片数据组成的数组*/
   RasterDMAFBConfig(SOC_LCDC_0_REGS, (unsigned int)image,
                (unsigned int)image4 + sizeof(image) - 2, 0);  //DMA0

   RasterDMAFBConfig(SOC_LCDC_0_REGS, (unsigned int)image,
                (unsigned int)image + sizeof(image) - 2, 1);  //DMA1

/* 使能 LCDC 功能 */
   RasterEnable(SOC_LCDC_0_REGS);
```

## 三． 一副图片如何生成一个十六进制的数组

这里有个小插曲：简单介绍一下如何通过任意一副照片转变成一个图片数据数组 image[]。步骤如下：

1. 下载 AM335X_StarterWare_02_00_00_07_Setup.exe，链接为：

   http://software-dl.ti.com/dsps/dsps_public_sw/am_bu/starterware/latest/index_FDS.html

   下载后并安装在自定义目录中。StarterWare 里面有很多 Tool，还包含了大量的裸机代码。

2. 若安装的是 window 版本，则找到安装后的目录，把/tools/bmpToRaster 拷贝到 linux 系统下，若是 Linux 版本，则跳过该步骤。
3. 在 BitmapReader.h 头文件里屏蔽 #define COMPRESS
4. 在终端进入 bmpToRaster 目录下，执行 make，成功后将会生成 a.out
5. 选择一个 24 位的位图，格式是 bmp（若其他格式，请先通过图片工具进行转化），图片的 size=800*480（这里只是举个例子，根据你要驱动的屏幕的大小进行裁剪）
6. 将**图片名称.bmp** 拷贝到 a.out 目录下
7. 终端运行命令：./a.out 800 480 ./**图片名称.bmp** ./image.h 24 BGR BGR ？ RGB

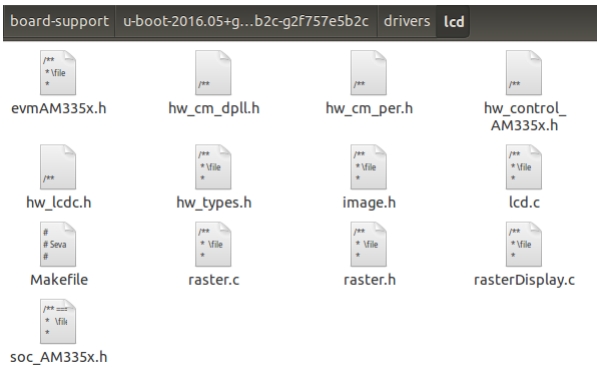
命令参数解析：800 480 和 24 BGR 的意思是生成适合 LCD 屏 800*480 的图片数据，并且数据格式是 888RGB 格式，该图片数据以数组形式存在 image.h 文件里。**图片名称.bmp 和** image.h，这两个文件的名称可自定义。生成 a.out 文件后，以后若还需将照片转化成数组形式，只需要执行第五个、第六个和第七个步骤即可。

## 四．在 Uboot 里增加 LCD 显示功能

Ok，接下来是重点了，讨论如何在 Uboot 里面初始化并使能 LCD 控制器，并让 LCD 屏显示 logo。而且有趣的是，可以进入 Uboot 的命令控制状态，输入自定义的命令，控制 LCD 显示或者关闭。

步骤如下：

1. 下载我提供的 LCD 的 drive——文件夹名“lcd”，将其放到 u-boot/drivers 文件夹里，lcd 文件夹里面内容如下图所示。比较重要的是文件 rasterDisplay.c。 其他一些.h 的头文件里主要是一些宏定义，描述相关的寄存器地址，以下表格有详细说明。




还需要在上层目录的Makefile中加上“obj-y += lcd/”

| | |
|---|---|
| rasterDisplay.c | 对 LCDC 初始化，其实里面内容就是我上面介绍的一些配置函数 |
| lcd.c | 初始化 LCDC 所有的引脚功能以及配置 LCDC 的时钟域 |
| raster.c / raster.c h | 其实里面的内容就是我上面所说的配置 LCDC 的“API”函数 |
| hw_control_Am335x.h | 描述 Control module 寄存器（配置引脚） |
| hw_cm_per.h | 描述时钟域的寄存器 |
| hw_cm_dpll.h | 描述与外设的时钟源选择有关的寄存器 |
| image.h | 照片生成的数据的头文件，里面是数组 image1[] |
| Makefile | 内容：obj-$(CONFIG_LCD_UBOOT) += rasterDisplay.o lcd.o raster.o 里面就这一句话，将 LCDC 初始化函数的所在的 C 文件 编译成相应的.o 文件，如果有 define CONFIG_LCD_UBOOT，则该步骤会在 make uboot 时候自动完成 |

| RasterDisplay.c 主要函数 | |
|---|---|
| int Lcd_Init(void) | //其实就是执行上面所说的蓝框的函数<br>//初始化的有时钟域、LCD 控制器的时序和 LCDC 的 DMA 模式 |
| void Lcd_reset(void) | //软件复位 LCD 控制器，LCDC 寄存器的值恢复为复位的状态 |
| void Lcd_off(void) | //关闭 LCDC 的功能 |
| void Lcd_on(void) | //打开 LCDC 的功能 |

2. 定义相关宏

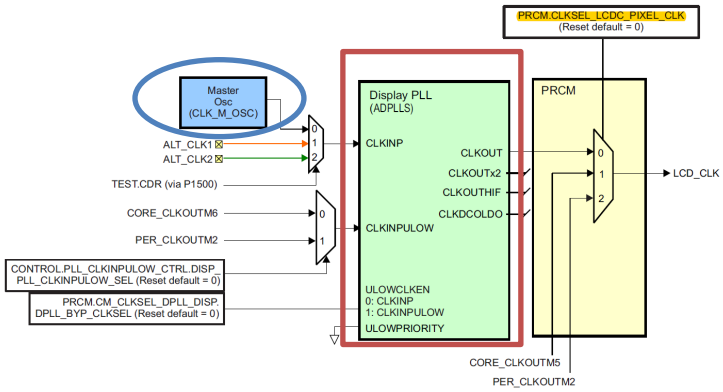
上面表格所说 makefile 的内容，需要定义 define  CONFIG_LCD_UBOOT，相关 C 文件才会被编译，那么如何增加这个宏定义呢？打开 am335x_evm.h (/u-boot /include/configs)，然后添加一句话 # define CONFIG_LCD_UBOOT，如图所示：

```
#ifndef CONFIG_SPL_BUILD
# define CONFIG_TIMESTAMP
# define CONFIG_LZO
# define CONFIG_LCD_UBOOT
#endif
```

3. 配置 Display PLL

如果还记得上面的内容，我们已经选择 Display PLL 的输出 CLKOUT 作为 LCD_CLK 的时钟源。而我提供的 lcd drive 里面还没实现 Display PLL 配置，考虑到 lcd 的时钟源的选择灵活性，所以不在 lcd drive 里面实现。接下来就是讨论如何怎么在 Uboot 里面配置 Display PLL。

Figure 8-13. Display PLL Structure




根据公式：LCD_CLK= Resource Clock * DPLL_Mult / (N +1) /M2，配置 Display PLL 也就是配置 DPLL_Mult、N 和 M2。此处 Resource Clock 使用的是 24M 晶振（蓝椭圆）作为 Display PLL 的时钟源输入。步骤分为 ABCD 四个步骤：

A. 打开文件 Clock_am33xx.c (uboot\arch\arm\cpu\armv7\am33xx)，按照下面位置进行添加（红色标明为新增加，请看紫色注释说明，帮助理解）

```
const struct dpll_params dpll_per = {
                    960, OSC-1, 5, -1, -1, -1, -1};// 找到这个位置之后进行添加

// 声明 dpll_regs 结构体 为 dpll_dis_regs，
//将与 display PLL 相关的寄存器地址 赋值给该结构体的参数：

const struct dpll_regs dpll_dis_regs = {
          .cm_clkmode_dpll      = CM_WKUP + 0x98,
          .cm_idlest_dpll       = CM_WKUP + 0x48,
          .cm_clksel_dpll       = CM_WKUP + 0x54,
          .cm_div_m2_dpll       = CM_WKUP + 0xA4,
};

// 声明 dpll_params 结构体为 dpll_dis，取名倍频结构体，对其赋值：
//DPLL_Mult =100 ； N= OSC-1； M2=1, 后面四个-1 代表参数无效
//所以根据公式 LCD_CLK=100M      （OSC=24M 为晶振频率）
const struct dpll_params dpll_dis = { 100,   OSC-1,   1,   -1, -1, -1, -1};

//返回 指向 dpll_dis（倍频结构体）的指针
const struct dpll_params *get_dpll_dis_params(void)
{
          return &dpll_dis;
}
```

B. 打开 Clock.h (uboot\arch\arm\include\asm\arch-am33xx)，添加两条语句：

```
extern const struct dpll_regs dpll_dis_regs;
const struct dpll_params *get_dpll_dis_params(void);
```

C. 打开 Clock.c (uboot\arch\arm\cpu\armv7\am33xx)，按照下面位置进行添加（红色标明为新增加，请看紫色注释说明，帮助理解）

```
static void setup_dplls(void)
{
          。。。。。
          params = get_dpll_ddr_params();
          do_setup_dpll(&dpll_ddr_regs, params);

//指向 dpll_dis 的指针赋值给 params
          params = get_dpll_dis_params();

//根据 A 步骤声明的寄存器结构体 dpll_dis_regs 和倍频结构体指针作为形参
//传参后，该函数会对寄存器进行赋值，达到倍频的效果
//执行成功后，输出 LCD_CLK=100M

          do_setup_dpll(&dpll_dis_regs, params);
}
```

D. 打开 Board.c (board\ti\am335x)，按照下面位置进行添加（黄色标明为新增加，请看橙色注释说明，帮助理解）

```
#if !defined(CONFIG_SPL_BUILD)
void lcdbacklight(int gpio, char *name, int val) //该函数使能 LCD 背光引脚
{
        int ret;
        ret = gpio_request(gpio, name);
        if (ret < 0) {
                printf("%s: Unable to request %s\n", __func__, name);
                return;
        }
        ret = gpio_direction_output(gpio, 0);
        if (ret < 0) {
                printf("%s: Unable to set %s  as output\n", __func__, name);
                goto err_free_gpio;
        }
        gpio_set_value(gpio, val);
        return;
err_free_gpio:
        gpio_free(gpio);
}
void lcdbacklight_off(int gpio) //关闭 lcd 背光并且关闭 LCDC 的功能
{
    gpio_set_value(gpio,0);
    Lcd_off();                  //来自 lcd drive
    return;
}
void lcdbacklight_on(int gpio) //打开 lcd 背光并且打开 LCDC 的功能
{
#if !defined(CONFIG_SPL_BUILD)
    gpio_set_value(gpio,1);
    Lcd_on();                  //来自 lcd drive
#endif
        return;
}
void board_lcd_reset(int gpio) //关闭 lcd 背光并且软件复位 LCDC
{
    gpio_set_value(gpio,0);
    Lcd_reset();
        return;

}#endif
int board_init(void)
{ ……..
#if !defined(CONFIG_SPL_BUILD)
    lcdbacklight(7, "lcdbacklight", 1); //使能 lcd 背光，引脚是 GPIO0_7
    Lcd_Init();
#endif
……..}
```

E. 打开 Board.h (board\ti\am335x)，按照下面位置进行添加

```
int Lcd_Init(void);
void lcdbacklight(int gpio, char *name, int val);
void lcdbacklight_off(int gpio);
void lcdbacklight_on(int gpio);
void board_lcd_reset(int gpio);
```

## 五．通过添加 Uboot 的自定义命令来控制 LCD

最后一部分是讨论怎么添加 Uboot 的自定义命令，控制 LCD。步骤如下：

1. 在 u-boot/cmd 文件里，添加 lcd_cmd.c 文件，该文件会提供给大家。

细心的读者可以发现上面 D 步骤所定义的函数 void lcdbacklight_off(int gpio)、void lcdbacklight_on(int gpio))和 board_lcd_reset(int gpio)还没有被调用。

其实这些函数将会在 lcd_cmd.c 文件命令执行函数 do_lcd 中调用。

```
static int do_lcd(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    int cmd;
    /* Validate arguments */
    if ((argc < 1) || (argc > 2))
                return CMD_RET_USAGE;

    cmd = get_lcd_cmd(argv[1]); //获取命令类型
    if (cmd < 0) {
                return CMD_RET_USAGE;
    }
#ifdef CONFIG_CMDLINE
     if (cmd==0){
          printf("lcd off\n");
      lcdbacklight_off(7);
    }
    else if (cmd==1){
          printf("lcd on\n");
      lcdbacklight_on(7);
    }
   else if (cmd==2){
          printf("lcd reset\n");
         board_lcd_reset(7);
    }
          return 0;
#else
          return 1;
```

需要对应上LED_PWM引脚（goembed的是gpio3_7=113）

2. 打开 u-boot/cmd/Makefile 文件，在如下位置添加一句 obj-y += lcd_cmd.o ，确保 lcd_cmd.c 文件被编译，如图：

```
ifndef CONFIG_SPL_BUILD
# core command
obj-y += boot.o
obj-$(CONFIG_CMD_BOOTM) += bootm.o
obj-y += help.o
obj-y += lcd_cmd.o
```

3. 打开 u-bootxx/include/config/am335x_evm.h，增加红色部分，执行 boot 引导 kernel 的时候，最后把 LCDC 复位，防止在引导 kernel 时出现问题，因为 kernel 也会对 LCDC 进行初始化。

```
#define CONFIG_BOOTCOMMAND \
        "run findfdt; " \
        "run init_console; " \
        "run envboot; " \
        "run RESET_LCD; " \
        "run distro_bootcmd"

"RESET_LCD=" \
        "ULCD reset\0" \
```

验证效果：

上面的所有的步骤都完成后，就可以执行 make 指令，编译 Uboot，生成 MLO 和 U-boot.img，copy 到 SD 卡运行，或者通过其他方式加载到板子上，或者通过其他方式 bootload。加载 u-boot.img 后，可以发现此时屏幕已经亮起来并显示 logo。

进入 uboot 命令，输入 ULCD –h 命令可以看到一些命令帮助，此时先输入 ULCD off 可以看到屏幕关闭，logo 没了，输入 ULCD on 又可以看到屏幕重新显示 logo。

```
=>
=>  ULCD -h
ULCD - lcd open or close


Usage:
ULCD ULCD [on|off|reset]

=>  ULCD off
lcd off
=>  ULCD on
lcd on
=>  ULCD reset
lcd reset
=>
```

## 六．常见问题

温馨提醒：如果 Uboot 的 LCDC_CLK 选择 PER_CLKOUTM2，而不是选择 Display PLL，在 kernel 加载过程中会出现串口打印乱码、I2C 时序错误等的现象。

这是因为 kernel 配置 Per PLL 是一个逆过程，它首先根据 LCD 的 pixel clock(假设等于 30M)默认 LCDC 对时钟源 LCDC_CLK 进行的是二分频，所以 LCDC_CLK=60M，因为 Uboot 选择 PER_CLKOUTM2 作为它的时钟源，在 kernel 会默认此选择，所以 PER_CLKOUTM2=LCDC_CLK=60M，以及 kerne 将 PER_CLKOUTM2 配置为 60M，而其他一些外设默认选择时钟源为 PER_CLKOUTM2，且值=192M，但实际上 PER_CLKOUTM2=60M，所以时钟源错了，那可想而知其他的外设时钟，如 UART、I2C 等肯定会出现时序的错乱。

**Figure 8-11. Peripheral PLL Structure**