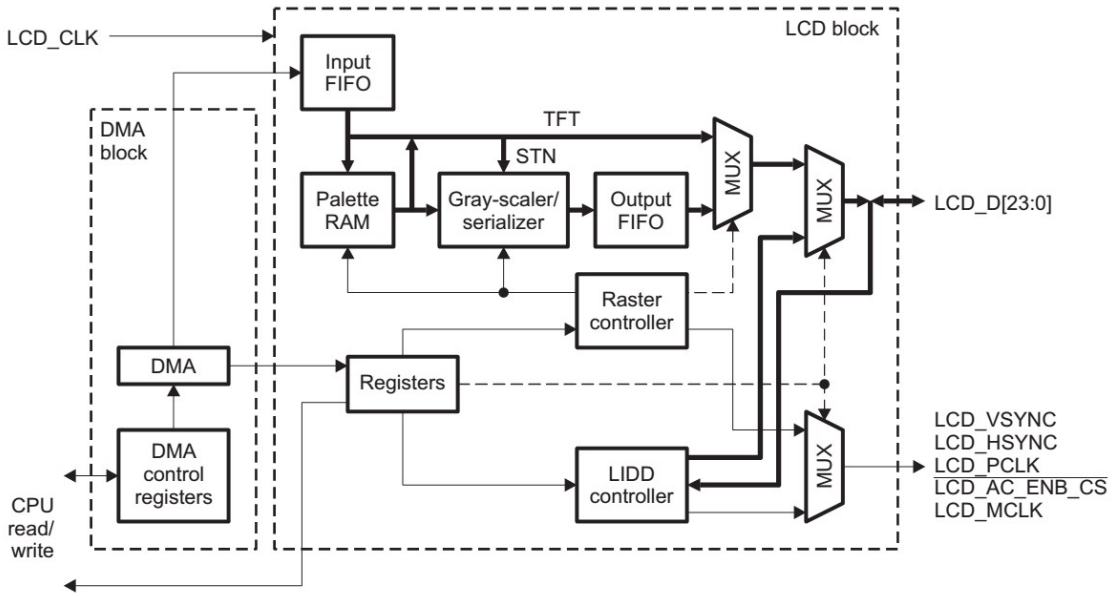
lcd 控制器总体上，由下面这些组件组成：

Figure 13-1. LCD Controller



1.内置 DMA

2.Raster 控制器

3.LIDD 控制器

Raster 控制器和 LIDD 控制器，是二选一的，同一时间只能使能一个。

raster 控制器对应的是同步 LCD 接口（现在流行的 lcd 接口，包含 hsync,vsync 等控制信号）。

LCD Interface Display Driver (LIDD) 控制器对应的是异步的 LCD 接口(以前老式的 lcd 接口，包含 we,oe 之类的控制线)。

因为现在的屏幕大多是同步 LCD 接口的，所以后面的篇幅重点解释 raster 的细节，LIDD 的细节予以忽略。

然后由 raster/LIDD 再去把数据发给外接的 lcd。

所以总的数据流程是：

frame buffer -> DMA -> raster/lidd -> lcd

支持 lcd 的种类包括：

被动式矩阵 lcd　　STN, DSTN, and C-DSTN

主动式矩阵 lcd　　TFT

OLED lcd 主动/被动矩阵 OLED 屏

lcd 控制器的时钟输入有这几路：

Table 13-2. LCD Controller Clock Signals

Clock Signal	Max Freq	Reference / Source	Comments
l3_clk Master Interface Clock	200 MHz	CORE_CLKOUTM4	pd_per_lcd_l3_gclk From PRCM
l4_clk Slave Interface Clock	100 MHz	CORE_CLKOUTM4 (divided within LCDC)	pd_per_lcd_l3_gclk From PRCM
lcd_clk Functional Clock	200 MHz	Display PLL CLKOUT	pd_per_lcd_gclk From PRCM

L3 和 l4 时钟，都来自于 CORE_CLKOUTM4

Lcd_clk 来自于 display pll，当 display pll 不可用的时候，也可取自 CORE_CLKOUTM6 or PER_CLKOUTM2。

这些个时钟，都受 PRCM 的控制。

注：PRCM 就是 Power, Reset, and Clock Management，就是 am3354 的一个电源和时钟管理模块。

Lcd 控制器引出脚的列表如下：

Table 13-3. LCD Controller Pin List

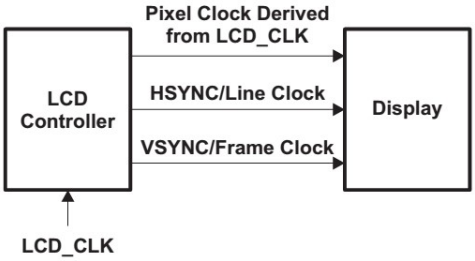
Pin	Type	Description
lcd_cp	O	Pixel Clock in Raster model Read Strobe or Read/Write Strobe in LIDD mode
lcd_pixel_i[15:0]	I	LCD Data Bus input (for LIDD mode only)
lcd_pixel_o[23:0]	O	LCD Data Bus output
lcd_lp	O	Line Clock or HSYNC in Raster mode; Write Strobe or Direction bit in LIDD mode
lcd_fp	O	Frame Clock or VSYNC in Raster mode; Address Latch Enable in LIDD mode
lcd_ac	O	AC bias or Latch Enable in Raster mode; Primary Chip Select/Primary Enable in LIDD MPU/Hitachi mode
lcd_mclk	O	N/A in Raster mode; Memory Clock/Secondary chip Select/Secondary Enable in LIDD Synchronous/Async MPU/Hitachi mode

Table 13-4. LCD External I/O Signals

Signal	Type	Description
LCD_VSYNC	OUT	Raster controller: Frame clock the LCD uses to signal the start of a new frame of pixels. Also used by TFT displays as the vertical synchronization signal. LIDD character: Register select (RS) or address latch enable (ALE) LIDD graphics: Address bit 0 (A0) or command/data select (C/D)
LCD_HSYNC	OUT	Raster controller: Line clock the LCD uses to signal the end of a line of pixels that transfers line data from the shift register to the screen and to increment the line pointer(s). Also used by TFT displays as the horizontal synchronization signal. LIDD character: not used. LIDD graphics: • 6800 mode = read or write enable • 8080 mode = not write strobe
LCD_PCLK	OUT	Raster controller: Pixel clock the LCD uses to clock the pixel data into the line shift register. In passive mode, the pixel clock transitions only when valid data is available on the data lines. In active mode, the pixel clock transitions continuously, and the ac-bias pin is used as an output enable to signal when data is available on the LCD pin. LIDD character: not used. LIDD graphics: • 6800 mode = enable strobe • 8080 mode = not read strobe
LCD_AC_BIAS_EN	OUT	Raster controller: ac-bias used to signal the LCD to switch the polarity of the power supplies to the row and column axis of the screen to counteract DC offset. Used in TFT mode as the output enable to signal when data is latched from the data pins using the pixel clock. LIDD character: Primary enable strobe LIDD graphics: Chip select 0 (CS0)
LCD_MCLK	OUT	Raster controller: not used. LIDD character: Secondary enable strobe LIDD graphics: Chip select 1 (CS1)
LCD_D[23:0]	Raster: OUT	LCD data bus, providing a 4-, 8-, 16- or 24-bit data path. Raster controller: For monochrome displays, each signal represents a pixel; for passive color displays, groupings of three signals represent one pixel (red, green, and blue). LCD_DATA[3:0] is used for monochrome displays of 2, 4, and 8 BPP; LCD_DATA[7:0] is used for color STN displays and LCD_DATA[15:0] or LCD_DATA[23:0] is used for active (TFT) mode.
	LIDD: OUT/IN	LIDD character: LCD_DATA[15:0] Read and write the command and data registers. LIDD graphics: LCD_DATA[15:0] Read and write the command and data registers.

Lcd 控制器，输出到 lcd 屏幕的时钟，一共三路。

Figure 13-3. Input and Output Clocks



Pclk 直接由 lcd_clk 分频得来。

$$LCD\_PCLK = \frac{LCD\_CLK}{CLKDIV}$$

Pclk 的作用是驱动像素数据录入到 line shift register 里面去。

Hsync 会在所有的水平线的像素数据传输完以后，再加上数据传输开始前和结束后，可编程的延迟时间（hback porch / hfront porch），去拉这个信号。

Hsync 能被配置成和 pclk 的上升沿或者下降沿同步。

Hsync 的极性也是可以配置的。

Hsync 会在所有的垂直线的像素数据传输完以后，再加上数据传输开始前和结束后，可编程的延迟时间（vback porch / vfront porch），去拉这个信号。

Hsync 能被配置成和 pclk 的上升沿或者下降沿同步。

Hsync 的极性也是可以配置的。

LCD_AC_BIAS_EN 脚（也就是 lcd_ac 脚），在 TFT 模式下，充当 OE 的作用，为了提示 lcd 数据总线上出现有效的数据。

DMA:

DMA 支持 1～2 个 framebuffer.当启用 2 个 framebuffer 的时候，就能进行乒乓操作，一边写入 framebuffer，一边从 framebuffer 拷到 raster 控制器以输出到屏幕。也可只配置一个 framebuffer。

Raster 模式下，DMA 会把 framebuffer 的数据，扔到 input FIFO 里面。而 raster controller 会从 FIFO 里面取数据来刷新屏幕。所以 DMA 需要保证 FIFO 一直是满的。LIDD 模式下，DMA 会直接把数据从 framebuffer 扔到 LIDD 控制器的寄存器里面，没有 FIFO 环节。

DMA 的寄存器包含下面这些：

Table 13-5. Register Configuration for DMA Engine Programming

Register	Configuration
LCDDMA_CTRL	Configure DMA data format
LCDDMA_FB0_BASE LCDDMA_FB0_CEILING	Configure frame buffer 0
LCDDMA_FB1_BASE LCDDMA_FB1_CEILING	Configure frame buffer 1. (If only one frame buffer is used, these two registers will not be used.)

从上面可以看出，包含配置 dma 数据格式的寄存器。还有 frame buffer 0 和 1 的 base 和 ceiling 寄存器（也就是 framebuffer 的首尾地址）。

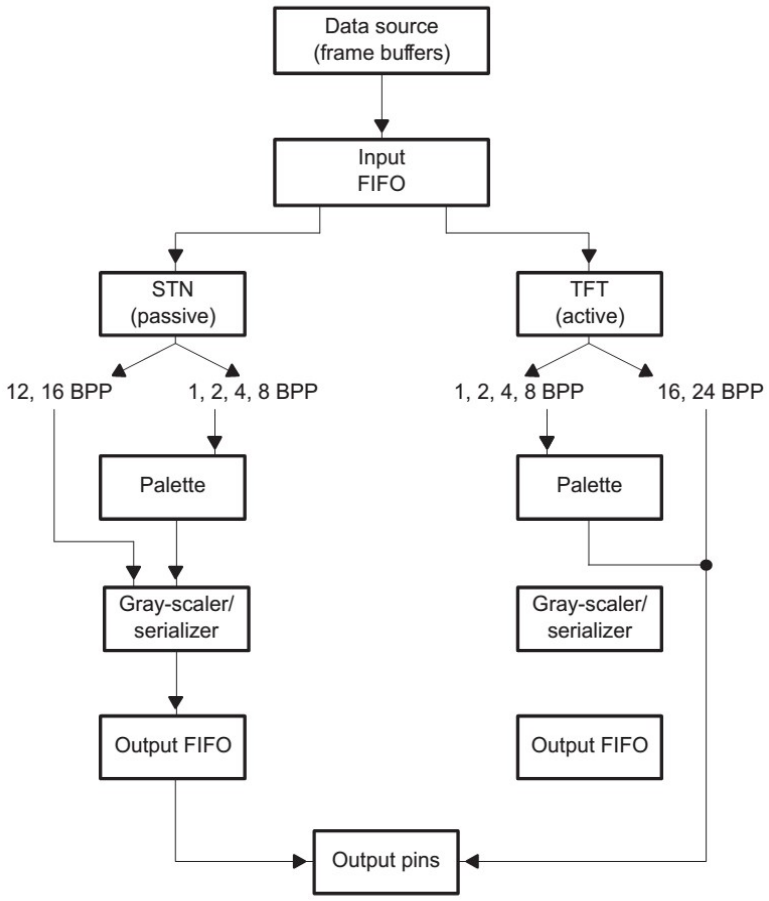
DMA 会产生一些中断，raster 模式下，具体有：

1. Output FIFO under-run。这种情况，就是说，LCD 刷新数据的速度（由 pclk 决定），大于 DMA 填充 FIFO 的速度。原因可能是：系统内存的带宽问题，或者是 pclk 设定的不对。

2. Frame synchronization lost.这种情况，就是 DMA 去读它认为的 video buffer 的第一个 word，但实际上却发觉无法识别其为第一个 word。也就是说帧的同步被破坏了。导致的原因可能是错误的 framebuffer 地址，或者错误的 BPP 值（详见后续有关 framebuffer 的解释）。
3. Palette loaded。When using palette-only or palette+data modes, the PL bit in the IRQSTATUS_RAW register will be set when the palette portion of a DMA transfer has been loaded into palette RAM.（没看懂）
4. AC bias transition。 If the ACB_I bit in the RASTER_TIMING_2 register is programmed with a nonzero value, an internal counter will be loaded with this value and starts to decrement each time LCD_AC_BIAS_EN (AC-bias signal) switches its state. When the counter reaches zero, the ACB bit in the IRQSTATUS_RAW register is set, which will deliver an interrupt signal to the system interrupt controller (if the interrupt is enabled.) The counter reloads the value in field ACB_I, but does not start to decrement until the ACB bit is cleared by writing 1 to this bit in the IRQSTATUS register.（没看懂）
5. Frame transfer completed.一帧传输完毕。

数据传输的流程图：

Figure 13-4. Logical Data Path for Raster Controller



那么针对最常见的情形，也就是 TFT 屏，色深 24BPP.可以看到，数据就是经由 framebuffer,FIFO，然后跳过调色板和灰阶，直接输出到 LCD 屏幕的管脚上去了。

所有有关调色板和灰阶的内容，后续就不再提及了。

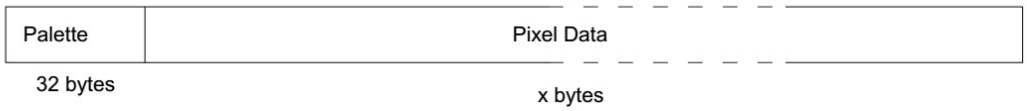
==========

Framebuffer:

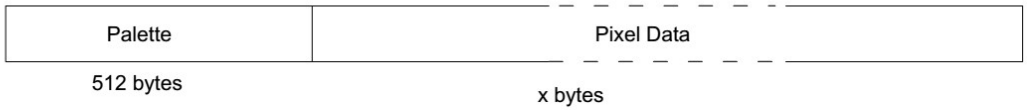
Framebuffer 就是一块连续的内存。在 framebuffer 里面，保存了两件东西：像素的数据，以及调色板索引表。如下图：

Figure 13-5. Frame Buffer Structure

1, 2, 4, 12, 16, 24 BPP Modes



8 BPP Mode



调色板部分，和比较老的屏幕有关，在 12，16，24bit 色深的模式下，已无需使用调色板的数据了。但调色板的预留位置予以保留。前 32 个字节仍然预留，值固定为 0x4000h。pixel data 部分，如果是 12，16，24bit 色深的模式下，就是真实的 RGB 数据。

那么对于比较老的屏幕，也就是色深在 8bit 及其以下的屏幕。以 8bit 为例。8bit 色深，就是有 256 中颜色。那么调色板就有 256 个色彩入口，每个入口(entry)代表一种颜色。这个调色板信息存放在 512bytes 的调色板区域，每个入口占用 **2 个字节**，所以 256x2=512 字节。而后续的 pixel data，其实只是对调色板色彩入口的索引值。至于 1,2,4bit 的屏，原理也是一样的。

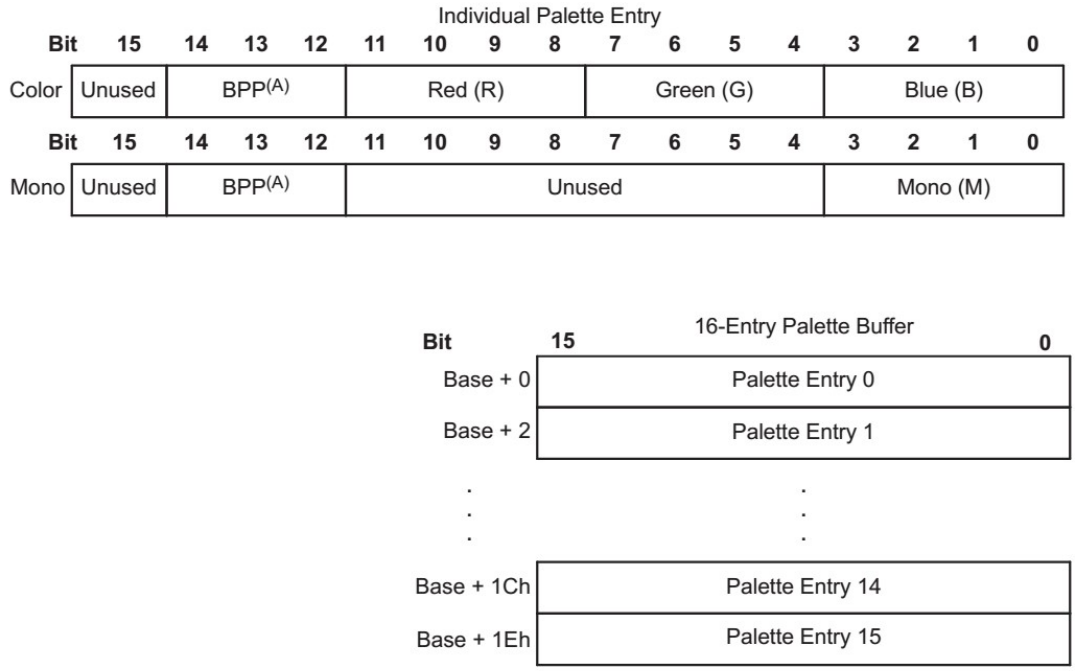
Framebuffer 大小的计算，可以参考下表：

Table 13-9. Frame Buffer Size According to BPP

BPP	Frame Buffer Size
1	32 + (Lines × Columns)/8
2	32 + (Lines × Columns)/4
4	32 + (Lines × Columns)/2
8	512 + (Lines × Columns)
12/16	32 + 2 × (Lines × Columns)

针对 32 字节的调色板的 framebuffer，这 32 字节具体是这么组成的：

Figure 13-6. 16-Entry Palette/Buffer Format (1, 2, 4, 12, 16 BPP)



A. Bits-per-pixels (BPP) is only contained within the first palette entry (palette entry 0).

注意，BPP 只在第一个 entry 里面有。一共是 16 个 entry.刚刚说了一个 entry 是 2 字节。所以一共 16x2=32 字节。

进一步细化，这里面的 BPP 信息，格式如下：

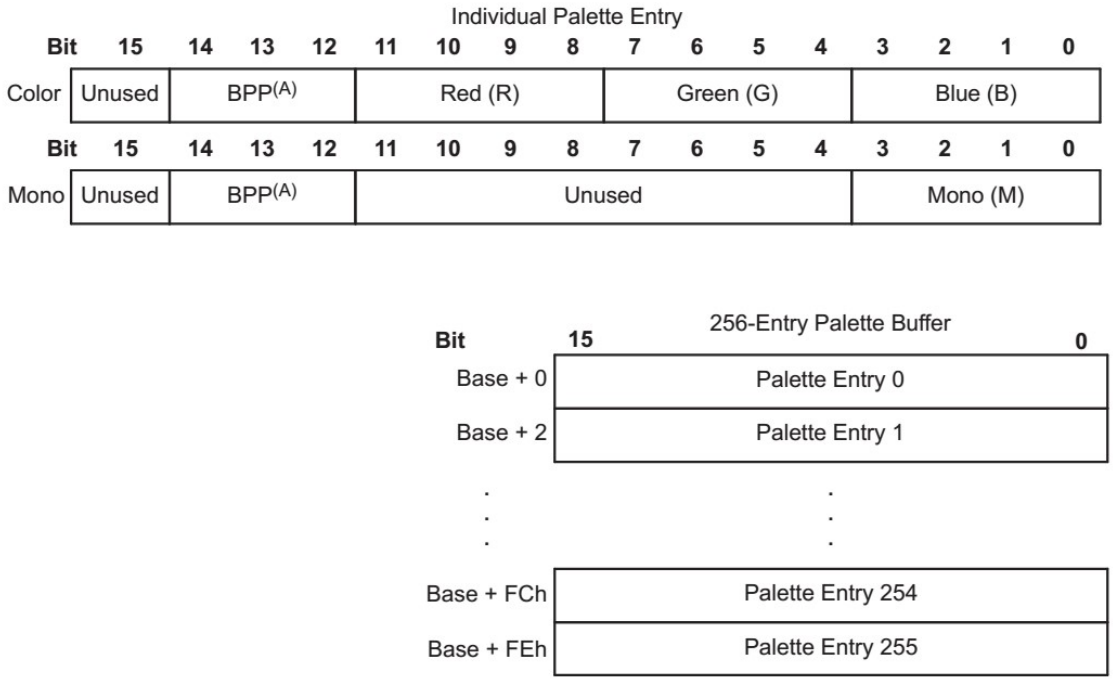
Table 13-8. Bits-Per-Pixel Encoding for Palette Entry 0 Buffer

Bit	Name	Value	Description(1) (2) (3)
14-12	BPP		Bits-per-pixel.
		000	1 BPP
		001	2 BPP
		010	4 BPP
		011	8 BPP
		1xx	12 BPP in passive mode (TFT_STN = 0 and STN_565 = 0 in RASTER_CTRL)
			16 BPP in passive mode (TFT_STN = 0 and STN_565 = 1 in RASTER_CTRL)
			16 BPP in active mode (LCDTFT = 1 and TFT24 = 0 in RASTER_CTRL)
			24 BPP in active mode (LCDTFT = 1 and TFT24 = 1 in RASTER_CTRL)

也就是说，根据这个 BPP 信息，就知道 entry 有几个了，因为 BPP 提示了色深是几位的。

针对 512 字节的调色板的 framebuffer，这 32 字节具体是这么组成的：

Figure 13-7. 256-Entry Palette/Buffer Format (8 BPP)

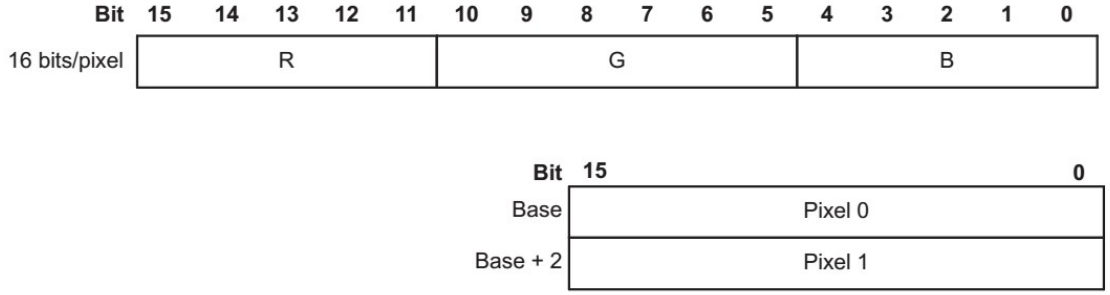


A. Bits-per-pixels (BPP) is only contained within the first palette entry (palette entry 0).

那么具体的 pixel 数据，格式如下：

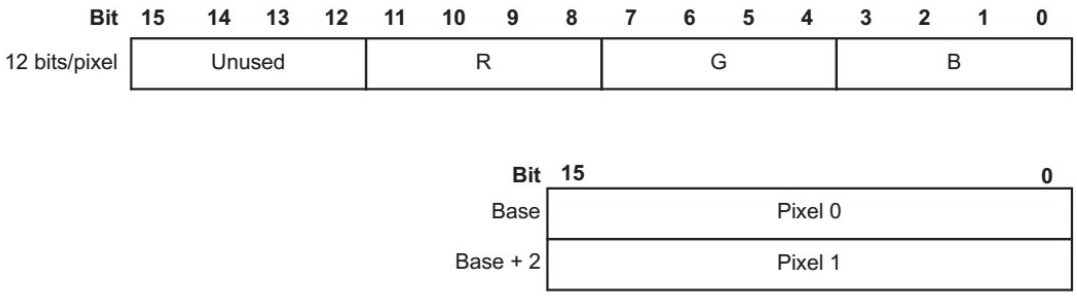
16bit:

Figure 13-8. 16-BPP Data Memory Organization (TFT Mode Only)—Little Endian



12bit:

Figure 13-9. 12-BPP Data Memory Organization—Little Endian



Unused [15-12] bits are filled with zeroes in TFT mode.

8bit 及以下:

Figure 13-10. 8-BPP Data Memory Organization

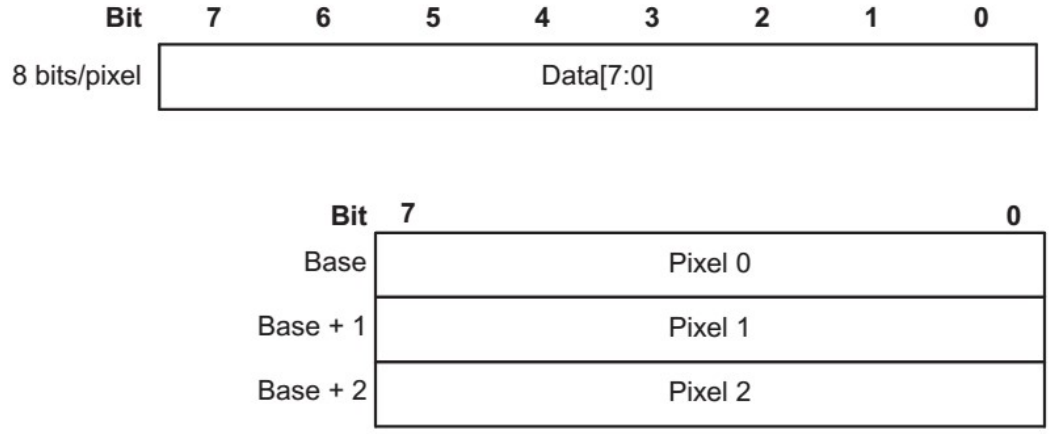


Figure 13-11. 4-BPP Data Memory Organization

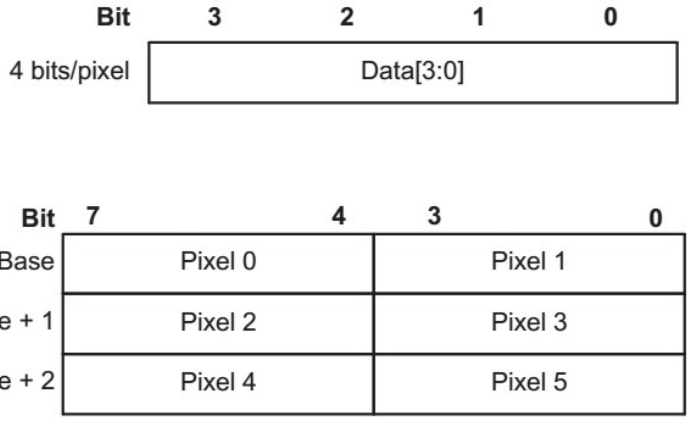


Figure 13-12. 2-BPP Data Memory Organization

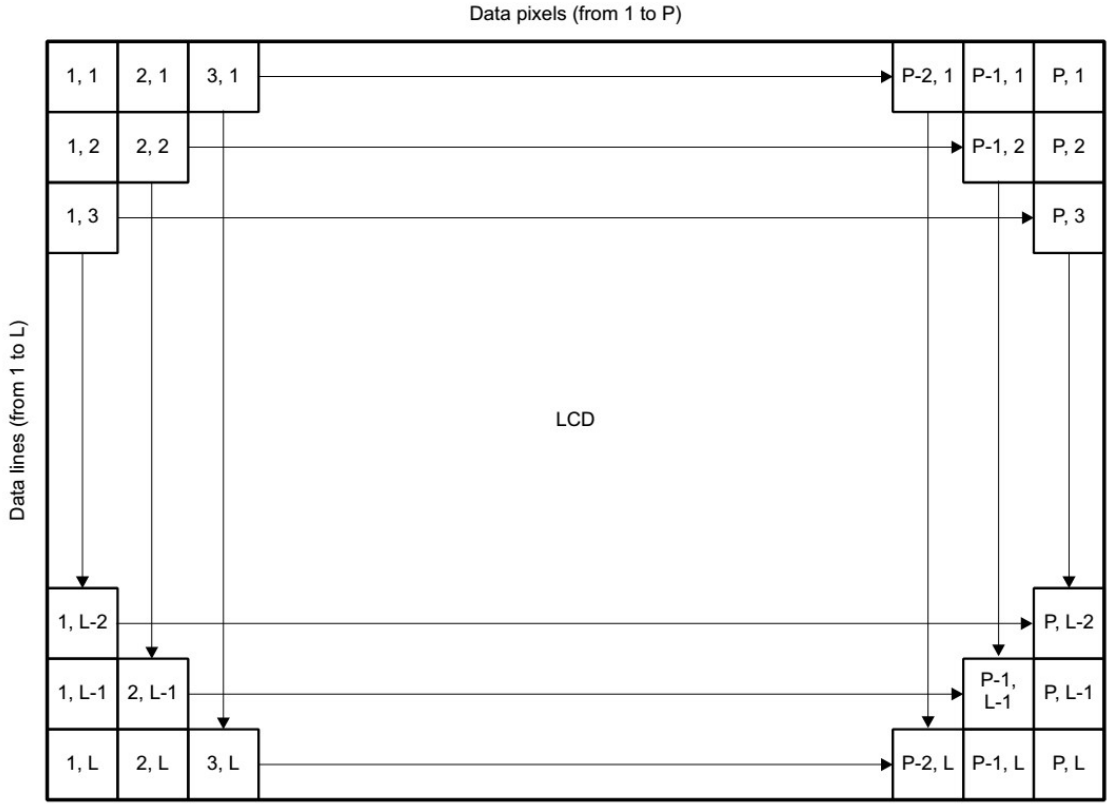
Bit	7 6	5 4	3 2	1 0
Base	Pixel 0	Pixel 1	Pixel 2	Pixel 3
Base + 1	Pixel 4	Pixel 5	Pixel 6	Pixel 7
Base + 2	Pixel 8	Pixel 9	Pixel 10	Pixel 11

Figure 13-13. 1-BPP Data Memory Organization

Bit	7	6	5	4	3	2	1	0
Base	P0	P1	P2	P3	P4	P5	P6	P7
Base + 1	P8	P9	P10	P11	P12	P13	P14	P15

像素编号的方式：

Figure 13-17. Raster Mode Display Format



==========

中断部分解释：

Lcdc 的中断的来源，有以下几个：

- DMA End of Frame 0

• DMA End of Frame 1

• Palette Loaded

• FIFO Underflow

• AC Bias Count

• Sync Lost

• Recurrent Frame Done

• LIDD or Raster Frame Done

其中一些常用的中断解释如下：

DMA End of Frame 0 和 1 的中断，当 DMA 传输完介于 cfg_fb0_base/cfg_bf0_ceil 或者 cfg_fb1_base/cfg_fb1_ceil 之间的 framebuffer 的内容，就会触发。

FIFO Underflow 的中断，前面已经提到，DMA 是要把 framebuffer 扔到 raster 的 fifo 里面。所以 fifo 必须是满的。如果 dma 没及时填充 fifo，就会造成 fifo 不满，从而导致这个 underflow 中断。

Sync Lost 的中断：当 DMA 把一个 framebuffer 扔到 raster 的 FIFO 里面的时候，它会设置一个帧开始标志和帧结束标志。用来指示一个完整的帧的头尾。那么 raster 的 lcd_clk 模块在从 FIFO 中取数据的时候，会检查，如果新的一帧的头部，没找到帧开始的标志，就说明出现了同步丢失的问题，会产生这个中断。

Recurrent Frame Done 的中断，当这个完整的帧被送到了外部 lcd 屏幕的引脚后，会触发。

==========

电源管理：

从 Figure 13-1 来看，如果要关闭时钟，似乎只要关闭 lcd_clk 就可以了。后面再补充。

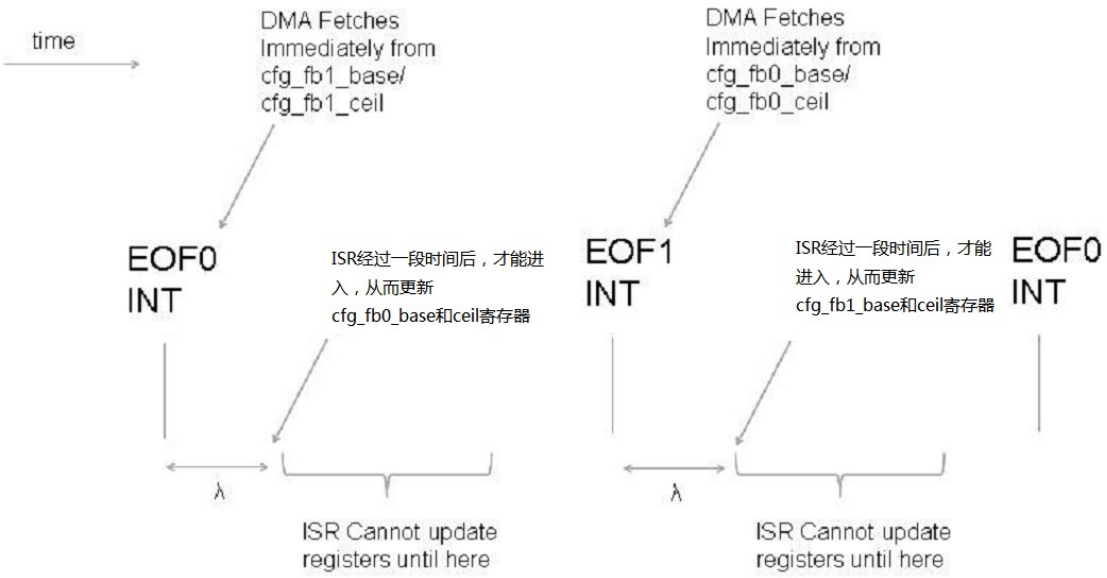
==========

Programming Model：

DMA End of Frame 中断的细节解释：

当 DMA 把 framebuffer0 的最后一个 word 扔到 fifo 里面去后，会触发一个 EOF0 的中断。但此时 DMA 不会停止，会立即乒乓操作，从 framebuffer1 去取数据扔到 fifo.当 framebuffer1 的数据扔完，产生 eof1 中断，然后 dma 又立即回到 framebuffer0 去取数据。如此往复。

在继续看里面的细节：



从图中可以看出，DMA 在 eof 中断后，是不会停止工作的，马上换 buffer 操作。而真正处理 eof0 或者 eof1 的 isr，在一个特定的延迟时间以后（从中断到进入 isr 需要时间），能开始更新当前不在使用的那个 framebuffer 的 base 和 ceiling。

==========

Disable and Software Reset Sequence：

流程如下：

• Set cfg_lcden='0'.

• Wait for the Done interrupt.（因为要传输完最后一帧）

• Set the software reset bits high (cfg_main_rst='1' or [cfg_dma_rst='1' and cfg_core_rst='1']) for several

cycles.（设置 1，持续一些时钟周期）

• Set the resets back low.（再设置回 0）

• Set cfg_lcden='1'.