MT6573 LCM Customer Document

Version: 1.0
Release date: 2011-08-17

Document Revision History

Revision	Date	Author	Description
1.0	2011-07-01	Zaikuo Wang	Initial Draft.

Table of Contents

Document Revision History **2**
Table of Contents **3**
1 Introduction **5**
1.1 Software/Hardware Environment 5
1.1.1 Software Environment 5
1.1.2 Hardware Environment 5
1.2 Functionality 5
1.3 Hardware background 5
1.3.1 MT6573 LCM Design Note 7
2 Design **9**
2.1 Architecture 9
2.2 Procedure & Flow 10
3 Interface **12**
3.1 Enumerations 12
3.1.1 LCM_TYPE 12
3.1.2 LCM_CTRL 12
3.1.3 LCM_POLARITY 12
3.1.4 LCM_CLOCK_PHASE 12
3.1.5 LCM_COLOR_ORDER 13
3.1.6 LCM_DRIVING_CURRENT 13
3.1.7 LCM_DBI_CLOCK_FREQ 13
3.1.8 LCM_DBI_DATA_WIDTH 13
3.1.9 LCM_DBI_CPU_WRITE_BITS 14
3.1.10 LCM_DBI_FORMAT 14
3.1.11 LCM_DBI_TRANS_SEQ 14
3.1.12 LCM_DBI_PADDING 14
3.1.13 LCM_DBI_TE_MODE 15
3.1.14 LCM_DBI_TE_VS_WIDTH_CNT_DIV 15
3.1.15 LCM_DPI_FORMAT 15
3.2 Data Structures 18
3.2.1 LCM_DBI_DATA_FORMAT 18
3.2.2 LCM_DBI_SERIAL_PARAMS 18
3.2.3 LCM_DBI_PARALLEL_PARAMS 20
3.2.4 LCM_DBI_PARAMS 22
3.2.5 LCM_DPI_PARAMS 23
3.2.6 LCM_PARAMS 27
3.2.7 LCM_UTIL_FUNCS 27
3.2.8 LCM_DRIVER 28
3.3 Global Variables 30
3.3.1 LCM_GetDriver 30

4 Customization 31
4.1 Adaptive LCM Support 31
At first you should add you new lcm driver as listed in the previous section, and then add the adaptive support: 31
4.2 Case Study – DBI Interface LCM Driver Porting 32
4.3 Case Study – DPI Interface LCM Driver Porting 37
5 Build 41
5.1 Source Code Structure & File Description 43
5.2 Build Option 44

Lists of Tables and Figures

Table 3-2 f_{out} Frequency Table 25
Table 5-1 LCM Driver Source File 43

Figure 1-1 DBI LCM HW Block Diagram 5
Figure 1-2 DPI LCM HW Block Diagram 6
Figure 2-1 Display Driver in Android SW Stack 9
Figure 2-2 LCM Driver Related Interfaces 10
Figure 2-3 Call flow of LCM driver(LCM Driver specified) 11
Figure 2-4 Call flow of LCM driver(Adaptive LCM Driver) 11
Figure 3-1 DBI Serial Interface Output Signal Timing Diagram 20
Figure 3-2 DPI Pixel Clock Generation Block Diagram 25
Figure 3-3 DPI Timing Parameters 26
Figure 3-4 Intermediated Buffers Between LCD And DPI 27
Figure 5-1 MTK Framebuffer Driver Software Architecture 44

1 Introduction

1.1 Software/Hardware Environment

1.1.1 Software Environment

Driver discussed in this document is designed based on U-Boot, Linux Kernel 2.6.35 and Android 2.3. However, the lcm driver don't have dependency with linux kernel/android version, you don't need to care the exact version of your platform when porting lcm driver.

1.1.2 Hardware Environment

Driver discussed in this document can run on the devices with MT6573.

1.2 Functionality

The LCM driver is used to configure LCM. The main functionality is
(1) Initialize LCM
(2) Fully/Partially update the LCD screen
(3) Suspend/Resume LCM
(4) Reading id(For LCM auto detect);

1.3 Hardware background

MT6573 supports three types of LCM interface
(1) DBI (Display Bus Interface or CPU Interface) LCM
(2) DPI (Display Pixel Interface or RGB Interface) LCM
(3) DSI (MIPI Display Serial Interface) LCM
As shown in Figure 1-1, DBI type LCM equips with its own frame buffer memory and refreshes the LCD panel periodically by itself. MT6573 host processor updates the LCM frame buffer content through the 8080 series type DBI interface. The update can be performed only when any frame buffer content is modified on Host side.

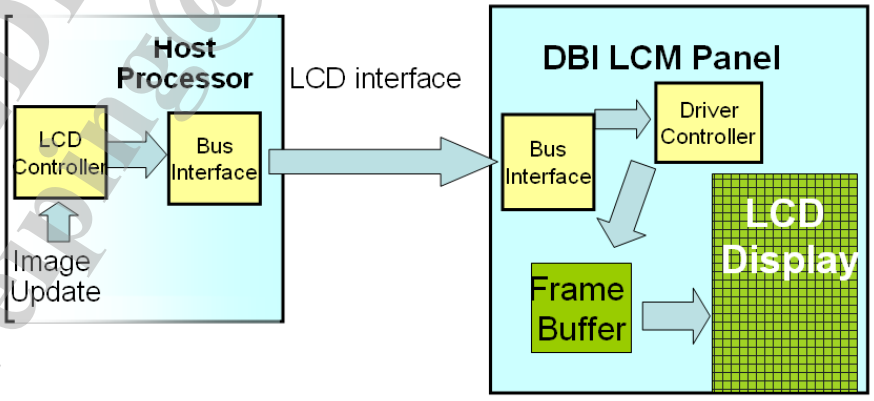


Figure 1-1 DBI LCM HW Block Diagram

As shown in Figure 1-2, DPI LCM does not equip with its own frame buffer memory and need MT6573 Host processor periodically transfer frame buffer data to LCM in order to refresh the LCD panel.

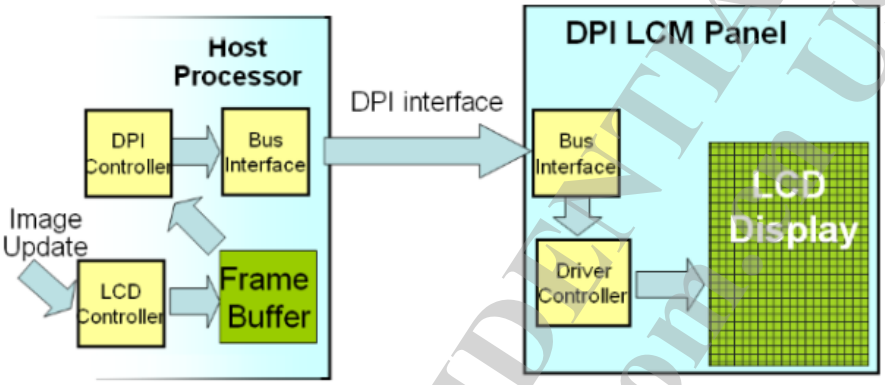


Figure 1-2 DPI LCM HW Block Diagram

DSI Controller support two mode: Command mode and Video mode.
As shown in Figure 1-3, DSI command mode LCM equip with its own frame buffer memory and refreshes the LCD panel periodically by itself like DBI interface LCM. And DSI command mode need MT6573 Host processor LCD controller direct couple frame buffer data to DSI controller when any frame buffer content is modified on Host side.

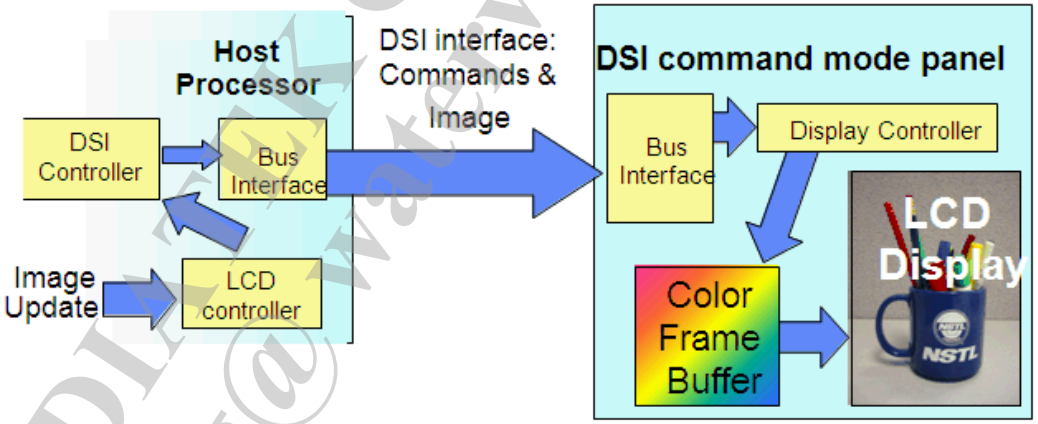


Figure 1-3 DSI Command mode HW Block Diagram

As shown in Figure 1-4, DSI video mode LCM does not equip with its own frame buffer memory like DPI interface LCM. It need MT6573 Host processor DPI controller periodically direct couple frame buffer data to DSI controller.

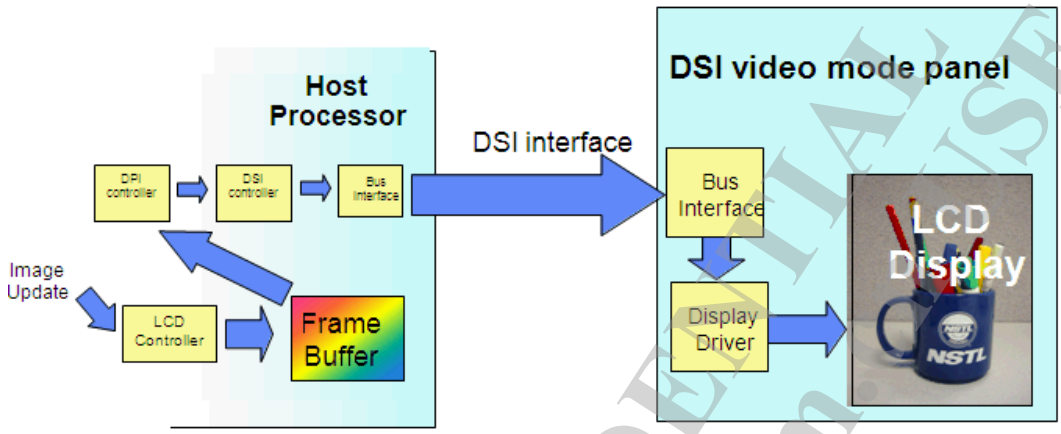
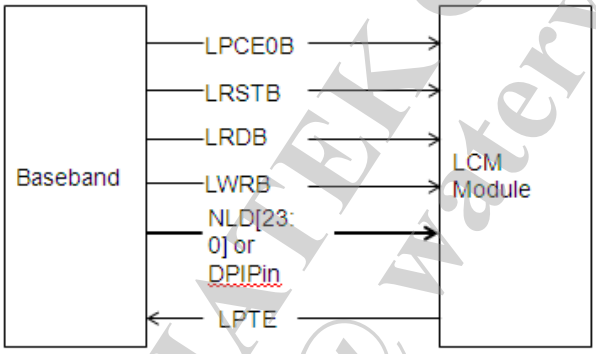


Figure 1-4 DSI Video mode HW Block Diagram

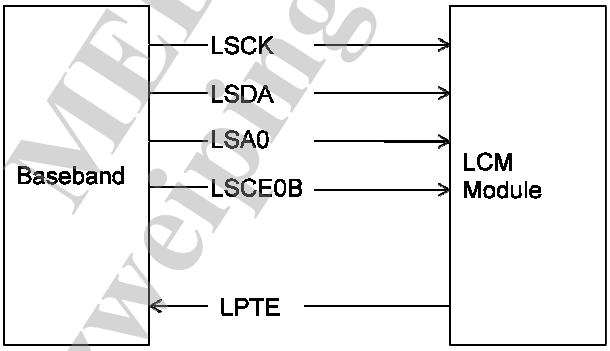
1.3.1 MT6573 LCM Design Note

For each type of LCM, the pin connection between MT6573 baseband and LCM module are shown as the following:

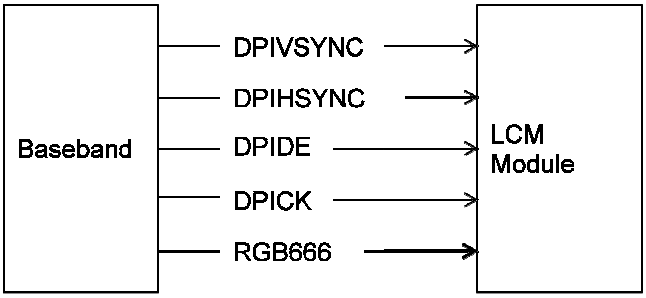
DBI parallel interface LCM (24bits in this example)



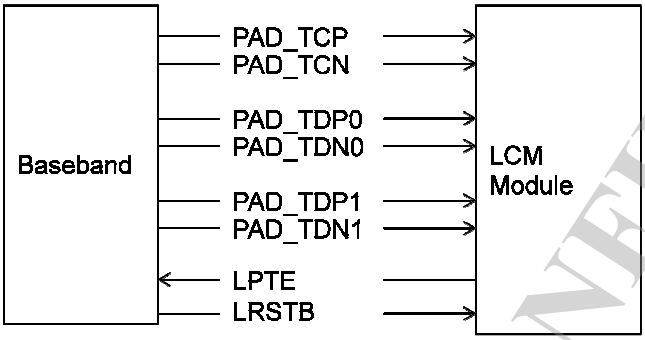
DBI serial interface LCM



DPI interface LCM (RGB666 in this example)



DSI interface LCM



For more details of each pin connection, please refer to MT6573 design notice document.

2 Design

2.1 Architecture

As shown in Figure 2-1, Android adopts Linux framebuffer driver as its backend display driver which is used directly by Surface Manager, aka SurfaceFlinger.

Figure 2-1 Display Driver in Android SW Stack

As shown in Figure 2-2, the `LCM_DRIVER` is the common abstract interface for all LCM drivers. Display driver controls the LCM through this interface which includes initialize, suspend/resume LCM …etc. The LCM driver can access some utility functions through `LCM_UTILS_FUNCS` which is necessary for LCM driver to control the LCM. It includes reset LCM, GPIO access functions, small delay function and LCM register read write function … etc. Besides, sometimes the same LCM driver may be used in different projects with different board configurations, e.g. GPIO pin definitions. In this case, the LCM driver should use the constants defined in the customized header file, e.g. `cust_gpio_usage.h`, instead of hard code.

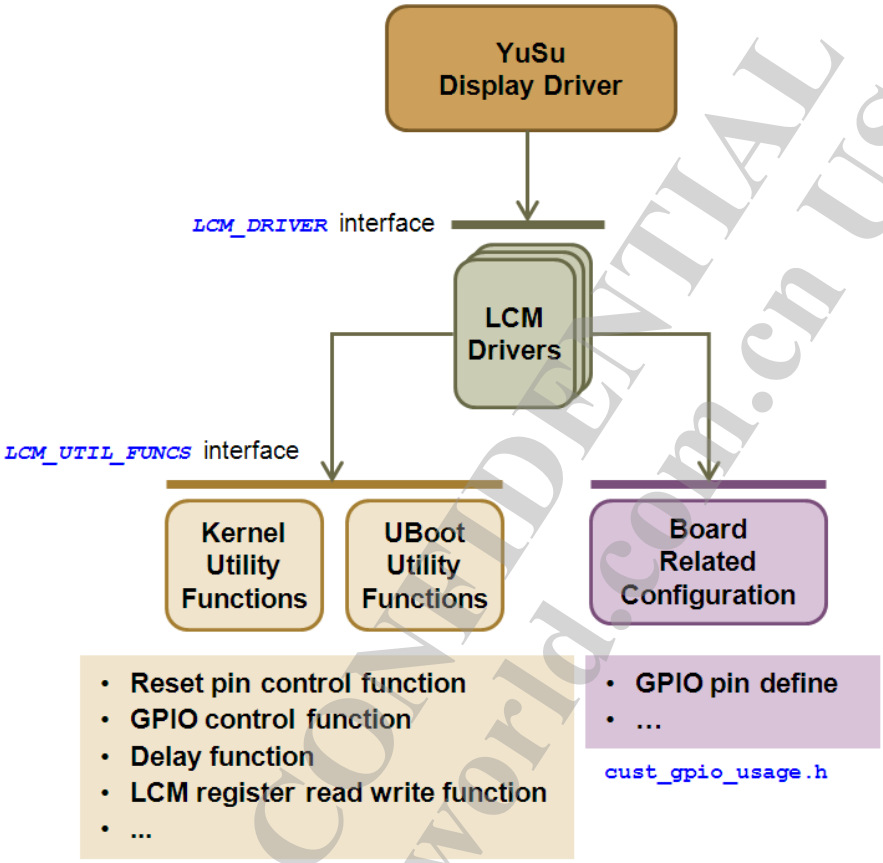


Figure 2-2 LCM Driver Related Interfaces

2.2 Procedure & Flow

The LCM driver calling procedure is shown as Figure 2-3:

(1) Display Driver get the array pointer `lcm_driver_list[]`which stores all the lcm driver pointer built in;
(2) Display Driver get the value `lcm_count`, which indicate how many lcm drivers we have;
(3) If `lcm_count` is equal to 1, we will directly use the first member in lcm_driver_list[] as LCM driver; If the LCM support reading id, we will try to read it. If reading id failed, we know that the LCM isn't connected to the hw, then TE signal will not be enabled, to avoid hang system;
(4) If `lcm_count` is larger than 1, we will traverse all the LCM drivers, and try to detect whether the id is correct (if a LCM driver can't support reading id, we will skip it. Actually if the LCM connected to the hw can't support reading id, adaptive LCM driver shouldn't be used);
(5) By now, the LCM driver has been found;
(6) Display driver set the `LCM_UTIL_FUNCS` interface to LCM driver
(7) Display driver get the `LCM_PARAMS` from LCM driver to initialize related MT6573 HW modules
(8) Display driver initializes the LCM by calling LCM `init` function
(9) Display driver updates LCD screen by calling LCM `update` function
(10) Display driver power off LCM by calling LCM `suspend` function
(11) Display driver power on LCM by calling LCM `resume` function

(12) Display driver set backlight level and PWM by calling `LCM_setbacklight` and `LCM_SetPWM` functions when using CABC

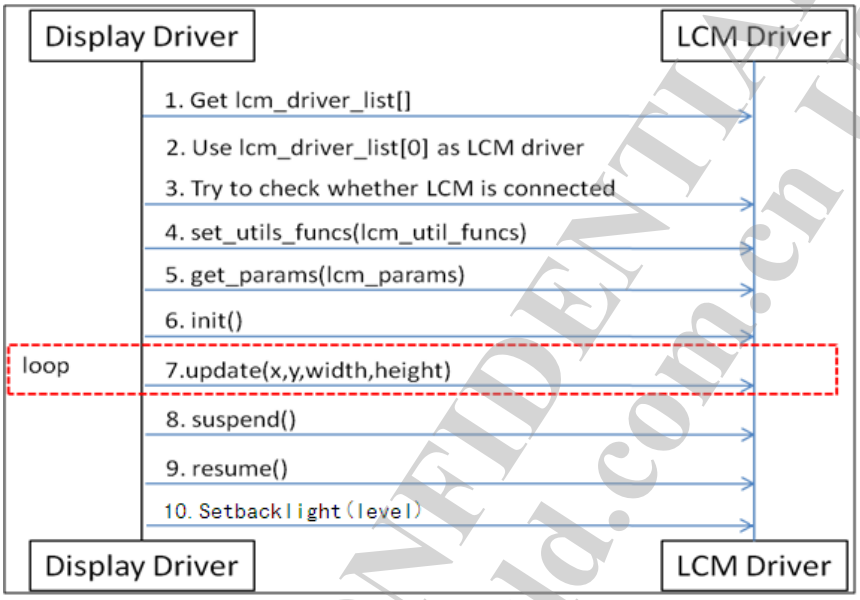


Figure 2-3 Call flow of LCM driver(LCM Driver specified)

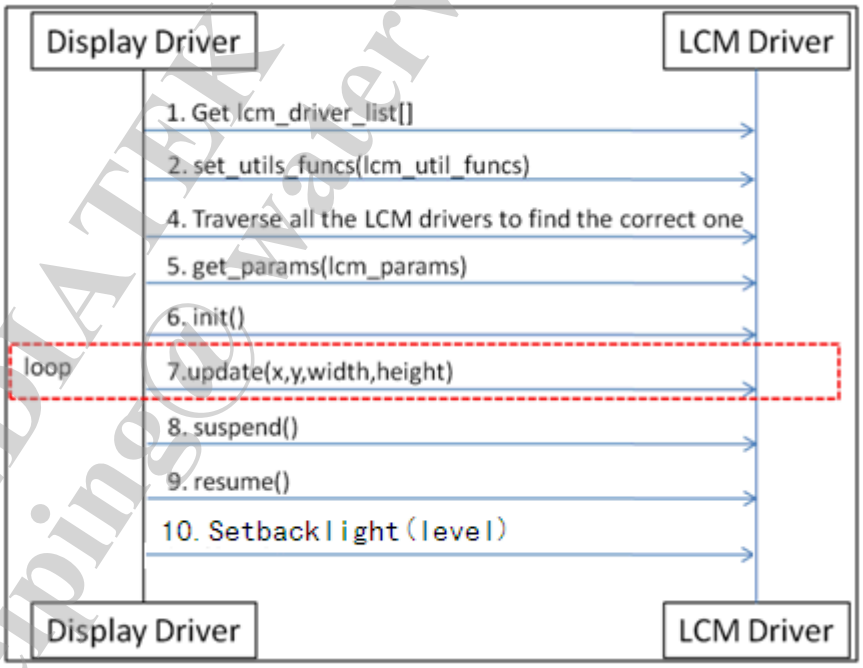


Figure 2-4 Call flow of LCM driver(Adaptive LCM Driver)

3 Interface

This chapter describes the data structure and interface used by LCM driver.

3.1 Enumerations

3.1.1 LCM_TYPE

This enumeration defines the LCM type.

```
typedef enum
{
    LCM_TYPE_DBI = 0,
    LCM_TYPE_DPI,
    LCM_TYPE_DSI
} LCM_TYPE;
```

3.1.2 LCM_CTRL

This enumeration defines which interface is used to access LCM registers.

```
typedef enum
{
    LCM_CTRL_NONE = 0,
    LCM_CTRL_SERIAL_DBI,
    LCM_CTRL_PARALLEL_DBI,
    LCM_CTRL_GPIO
} LCM_CTRL;
```

3.1.3 LCM_POLARITY

This enumeration defines the polarity of output signal.

```
typedef enum
{
    LCM_POLARITY_RISING  = 0,
    LCM_POLARITY_FALLING = 1
} LCM_POLARITY;
```

3.1.4 LCM_CLOCK_PHASE

This enumeration defines the phase of output clock signal.

```
typedef enum
{
```

```
    LCM_CLOCK_PHASE_0  = 0,
    LCM_CLOCK_PHASE_90 = 1
} LCM_CLOCK_PHASE;
```

3.1.5 LCM_COLOR_ORDER

This enumeration defines the RGB order of output color format.

```
typedef enum
{
    LCM_COLOR_ORDER_RGB = 0,
    LCM_COLOR_ORDER_BGR = 1
} LCM_COLOR_ORDER;
```

3.1.6 LCM_DRIVING_CURRENT

This enumeration defines the driving current of output signal.

```
typedef enum
{
   LCM_DRIVING_CURRENT_8MA = 0,
   LCM_DRIVING_CURRENT_4MA,
   LCM_DRIVING_CURRENT_2MA,
   LCM_DRIVING_CURRENT_SLEW_CNTL
} LCM_DRIVING_CURRENT;
```

3.1.7 LCM_DBI_CLOCK_FREQ(Because MT6573 LCD parallel interface clock is designed to 122.88MHz and not programmable, you can ignore this setting in LCM driver)

This enumeration defines the DBI clock frequency.

```
typedef enum
{
    LCM_DBI_CLOCK_FREQ_104M = 0,
    LCM_DBI_CLOCK_FREQ_52M,
    LCM_DBI_CLOCK_FREQ_26M,
    LCM_DBI_CLOCK_FREQ_13M,
    LCM_DBI_CLOCK_FREQ_7M
} LCM_DBI_CLOCK_FREQ;
```

3.1.8 LCM_DBI_DATA_WIDTH

This enumeration defines the DBI physical data width per transaction.
Valid values are listed as the following:
Serial DBI : 8bits, 9bits
Parallel DBI : 8bits, 9bits, 16bits, 18bits, 24bits

```
typedef enum
{
    LCM_DBI_DATA_WIDTH_8BITS  = 0,
    LCM_DBI_DATA_WIDTH_9BITS  = 1,
    LCM_DBI_DATA_WIDTH_16BITS = 2,
    LCM_DBI_DATA_WIDTH_18BITS = 3,
    LCM_DBI_DATA_WIDTH_24BITS = 4
} LCM_DBI_DATA_WIDTH;
```

3.1.9 LCM_DBI_CPU_WRITE_BITS

This enumeration defines the logical data width for each CPU write. If the logical data width is larger than the physical data bus width, the logical transaction will be split as multiple physical transactions.

```
typedef enum
{
    LCM_DBI_CPU_WRITE_8_BITS  = 8,
    LCM_DBI_CPU_WRITE_16_BITS = 16,
    LCM_DBI_CPU_WRITE_32_BITS = 32,
} LCM_DBI_CPU_WRITE_BITS;
```

3.1.10 LCM_DBI_FORMAT

This enumeration defines the DBI output color format. Details please refer to 3.1.22.

```
typedef enum
{
    LCM_DBI_FORMAT_RGB332 = 0,
    LCM_DBI_FORMAT_RGB444 = 1,
    LCM_DBI_FORMAT_RGB565 = 2,
    LCM_DBI_FORMAT_RGB666 = 3,
    LCM_DBI_FORMAT_RGB888 = 4
} LCM_DBI_FORMAT;
```

3.1.11 LCM_DBI_TRANS_SEQ

This enumeration defines the DBI transfer sequence. Details please refer to 3.1.22.

```
typedef enum
{
    LCM_DBI_TRANS_SEQ_MSB_FIRST = 0,
    LCM_DBI_TRANS_SEQ_LSB_FIRST = 1
} LCM_DBI_TRANS_SEQ;
```

3.1.12 LCM_DBI_PADDING

This enumeration defines the DBI padding bits position. Details please refer to 3.1.22.

```
typedef enum
{
    LCM_DBI_PADDING_ON_LSB = 0,
    LCM_DBI_PADDING_ON_MSB = 1
} LCM_DBI_PADDING;
```

3.1.13 LCM_DBI_TE_MODE

This enumeration defines the DBI tearing control mode.
The details please refer to MT6573 functional specification – LCD controller, LCD_TECON register.

```
typedef enum
{
    LCM_DBI_TE_MODE_DISABLED       = 0,
    LCM_DBI_TE_MODE_VSYNC_ONLY     = 1,
    LCM_DBI_TE_MODE_VSYNC_OR_HSYNC = 2,
} LCM_DBI_TE_MODE;
```

3.1.14 LCM_DBI_TE_VS_WIDTH_CNT_DIV

This enumeration defines the clock divisor for vertical TE detection.
(used in `VSYNC_OR_HSYNC` mode).
The details please refer to MT6573 functional specification – LCD controller, LCD_TECON register.

```
typedef enum
{
    LCM_DBI_TE_VS_WIDTH_CNT_DIV_8  = 0,
    LCM_DBI_TE_VS_WIDTH_CNT_DIV_16 = 1,
    LCM_DBI_TE_VS_WIDTH_CNT_DIV_32 = 2,
    LCM_DBI_TE_VS_WIDTH_CNT_DIV_64 = 3,
} LCM_DBI_TE_VS_WIDTH_CNT_DIV;
```

3.1.15 LCM_DPI_FORMAT

This enumeration defines the DPI output color format.

```
typedef enum
{
    LCM_DPI_FORMAT_RGB565 = 0,
    LCM_DPI_FORMAT_RGB666 = 1,
    LCM_DPI_FORMAT_RGB888 = 2
} LCM_DPI_FORMAT;
```

3.1.16 LCM_SERIAL_CLOCK_FREQ

This enumeration defines the DBI serial interface base clock frequency.

```
typedef enum
{
    LCM_SERIAL_CLOCK_FREQ_104M = 0,
    LCM_SERIAL_CLOCK_FREQ_26M = 1,
    LCM_SERIAL_CLOCK_FREQ_52M = 2
} LCM_SERIAL_CLOCK_FREQ;
```

3.1.17 LCM_SERIAL_CLOCK_DIV

This enumeration defines the DBI serial interface clock divisor.
The details please refer to MT6573 functional specification – LCD controller, LCD_SCNF register.

```
typedef enum
{
    LCM_SERIAL_CLOCK_DIV_2 = 0,
    LCM_SERIAL_CLOCK_DIV_4 = 1,
    LCM_SERIAL_CLOCK_DIV_8 = 2,
    LCM_SERIAL_CLOCK_DIV_16 = 3
} LCM_SERIAL_CLOCK_DIV;
```

3.1.18 LCM_DSI_MODE_CON

This enumeration defines the DSI interface mode.

```
typedef enum
{
    CMD_MODE = 0,
    SYNC_PULSE_VDO_MODE = 1,
    SYNC_EVENT_VDO_MODE = 2,
    BURST_VDO_MODE = 3
} LCM_DSI_MODE_CON;
```

3.1.19 LCM_LANE_NUM

This enumeration defines the DSI interface lane number

```
typedef enum
{
    LCM_ONE_LANE = 1,
    LCM_TWO_LANE = 2,
} LCM_LANE_NUM;
```

3.1.20 LCM_PS_TYPE

This enumeration defines the DSI interface pixel stream type.
The details please refer to MT6573 functional specification – DSI controller, DSI_PSCON register

```
typedef enum
{
    LCM_PACKET_PS_16BIT_RGB565 = 0,
```

```
    LCM_LOOSELY_PS_18BIT_RGB666 = 1,
    LCM_PACKET_PS_24BIT_RGB888 = 2,
    LCM_PACKET_PS_24BIT_RGB666 = 3
} LCM_LANE_NUM;
```

3.1.21 Data Structures

3.1.22 LCM_DBI_DATA_FORMAT

This data structure defines the DBI output data format.

```
typedef struct
{
    LCM_COLOR_ORDER color_order;
    LCM_DBI_TRANS_SEQ trans_seq;
    LCM_DBI_PADDING padding;
    LCM_DBI_FORMAT format;
    LCM_DBI_DATA_WIDTH width;
} LCM_DBI_DATA_FORMAT;
```

Refer to MT6573 functional specification – LCD Controller, LCD_WROICON register.

Variable	LCD_WROICON Bits	Description
color_order	0	0: BGR sequence 1: RGB sequence
trans_seq	1	0: MSB first 1: LSB first
padding	2	0: padding bits on LSBs 1: padding bits on MSBs
format	3..5	000 for RGB332 001 for RGB444 010 for RGB565 011 for RGB666 100 for RGB888
width	6..7/25	000 for 8-bit interface 001 for 16-bit interface 010 for 9-bit interface 011 for 18-bit interface Bit25:0x1→24bit interface output

3.1.23 LCM_DBI_SERIAL_PARAMS

This data structure defines the DBI serial interface specific parameters.

```
typedef struct
{
    LCM_POLARITY cs_polarity;
    LCM_POLARITY clk_polarity;
    LCM_CLOCK_PHASE clk_phase;
```

```
    unsigned int is_non_dbi_mode;
    LCM_SERIAL_CLOCK_FREQ clock_base;
    LCM_SERIAL_CLOCK_DIV clock_div;
} LCM_DBI_SERIAL_PARAMS;
```

Figure 3-1 shows some sample DBI serial output signal timing diagram.

Variable	Abbrev.	Description
cs_polarity	CPO	Chip select polarity
clk_polarity	SPO	Clock polarity
clk_Phase	SPH	Clock phase
is_non_dbi_mode	NON_DBI	0: DBI type C serial interface, 8 or 9 bits per transaction 1: non-DBI type C serial interface, 8/9/16/18/24/32 bits per transaction
Clock_base	IF_CLK	Serial interface base clock freq. 00: 104MHz 01: 26MHz 10: 52MHz 11: reserved
Clock_div	DIV	00: LSCK freq = clock_base/2 01: LSCK freq = clock_base/4 10: LSCK freq = clock_base/8 11: LSCK freq = clock_base/16

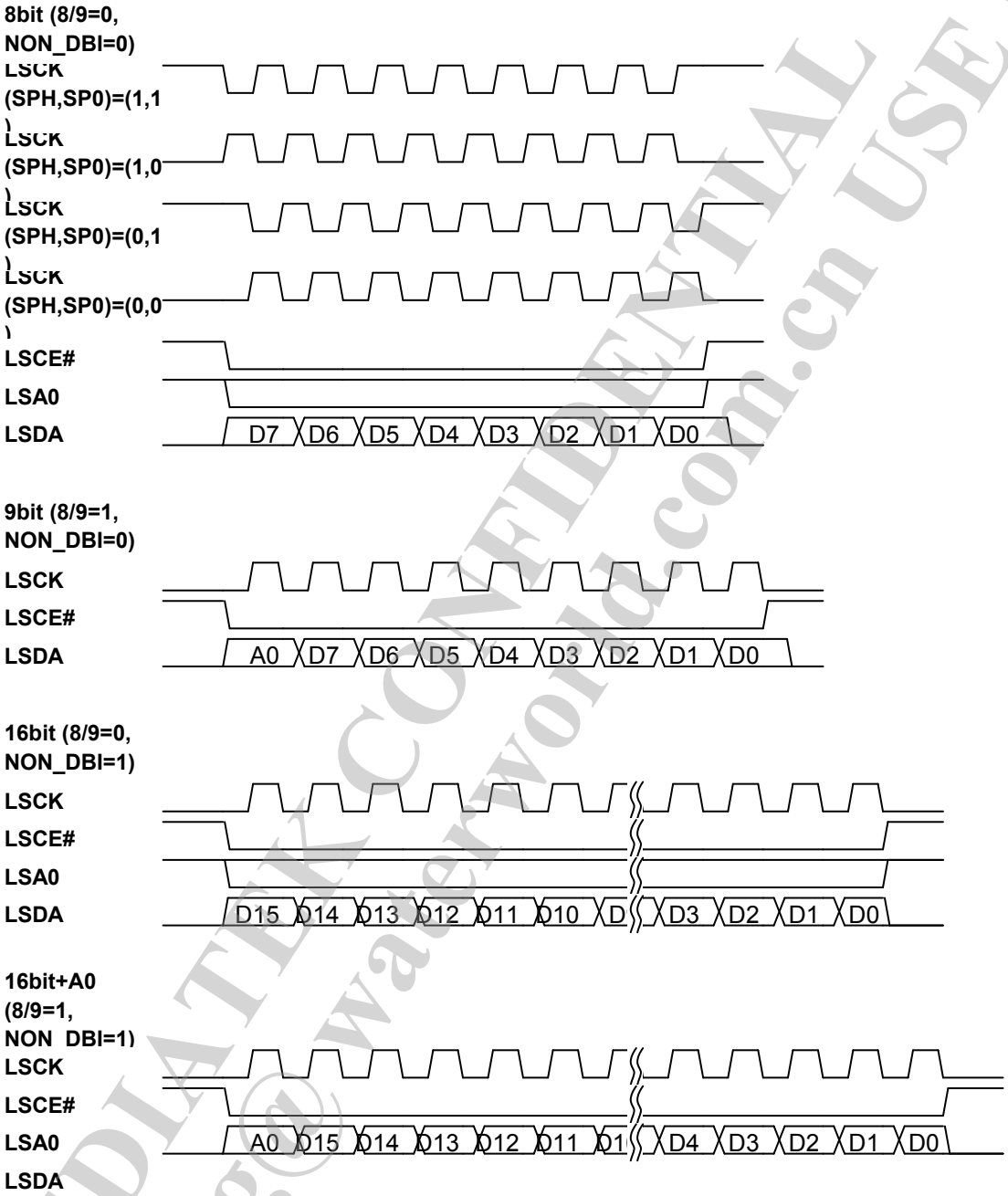


Figure 3-1 DBI Serial Interface Output Signal Timing Diagram

3.1.24 LCM_DBI_PARALLEL_PARAMS

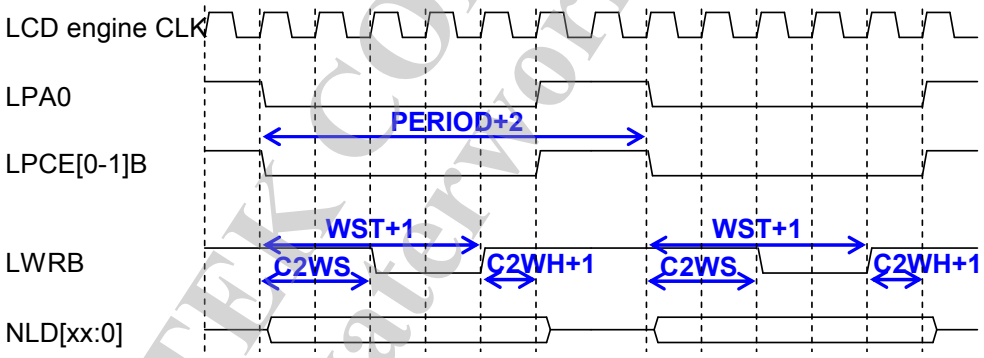
This data structure defines the DBI parallel interface specific timing parameters.

```
typedef struct
{
    /* timing parameters */
```

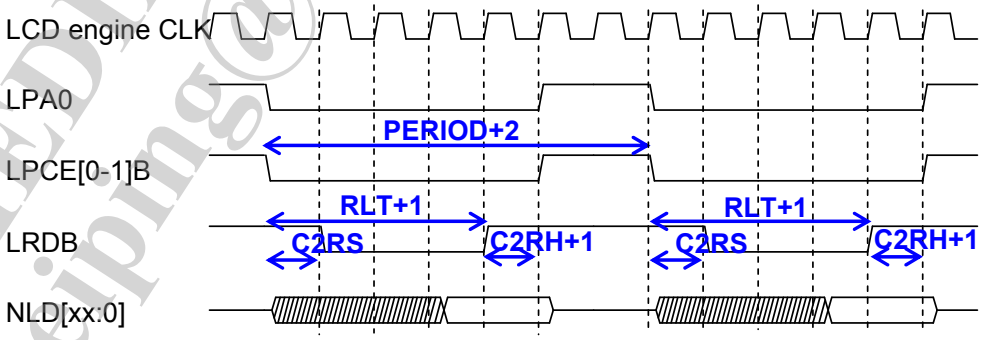
```
    unsigned int write_setup;
    unsigned int write_hold;
    unsigned int write_wait;
    unsigned int read_setup;
    unsigned int read_latency;
    unsigned int wait_period;
} LCM_DBI_PARALLEL_PARAMS;
```

Variable	Abbrev.	Description
write_setup	C2WS	Chip Select (LPCE#) to Write Strobe (LWR#) Setup Time
write_hold	C2WH	Chip Select (LPCE#) to Write Strobe (LWR#) Hold Time
write_wait	WST	Write Wait State Time
read_setup	C2RS	Chip Select (LPCE#) to Read Strobe (LRD#) Setup Time
read_latency	RLT	Read Latency Time
wait_period	WP	Waiting period between two consecutive transfers

Parallel Interface Write timing
C2WS=2, WST=3, C2WH=0, LCD_WROICON.PERIOD=0, **C2WS must <= WST**



Parallel Inteface Read timing
C2RS=1, RLT=3, C2RH=0, LCD_WROICON.PERIOD=0, **C2RS must <= RLT**



NLD is sampled at the rising edge of LRDB.

3.1.25 LCM_DBI_PARAMS

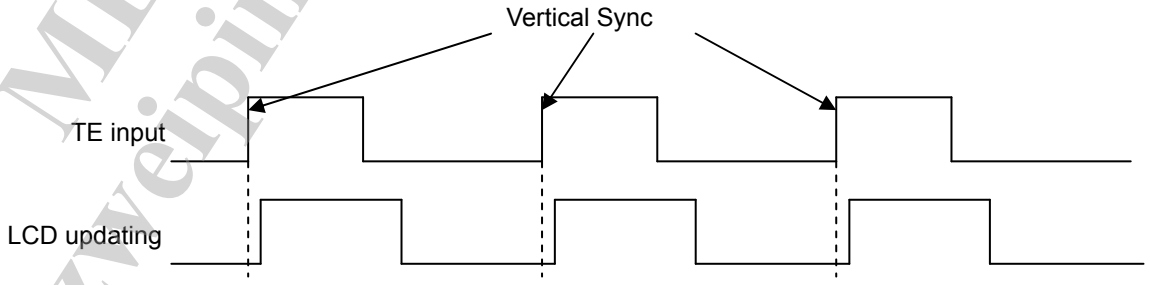
This data structure defines the DBI interface common parameters.

```
typedef struct
{
    /* common parameters for serial & parallel interface */
    unsigned int port;
    LCM_DBI_CLOCK_FREQ     clock_freq;
    LCM_DBI_DATA_WIDTH     data_width;
    LCM_DBI_DATA_FORMAT    data_format;
    LCM_DBI_CPU_WRITE_BITS cpu_write_bits;
    LCM_DRIVING_CURRENT    io_driving_current;

    /* tearing control */
    LCM_DBI_TE_MODE              te_mode;
    LCM_POLARITY                 te_edge_polarity;
    unsigned int                 te_hs_delay_cnt;
    unsigned int                 te_vs_width_cnt;
    LCM_DBI_TE_VS_WIDTH_CNT_DIV  te_vs_width_cnt_div;

    /* particular parameters for serial & parallel interface */
    union {
        LCM_DBI_SERIAL_PARAMS serial;
        LCM_DBI_PARALLEL_PARAMS parallel;
    };
} LCM_DBI_PARAMS;
```

MT6573 support 2 types of tearing control modes:

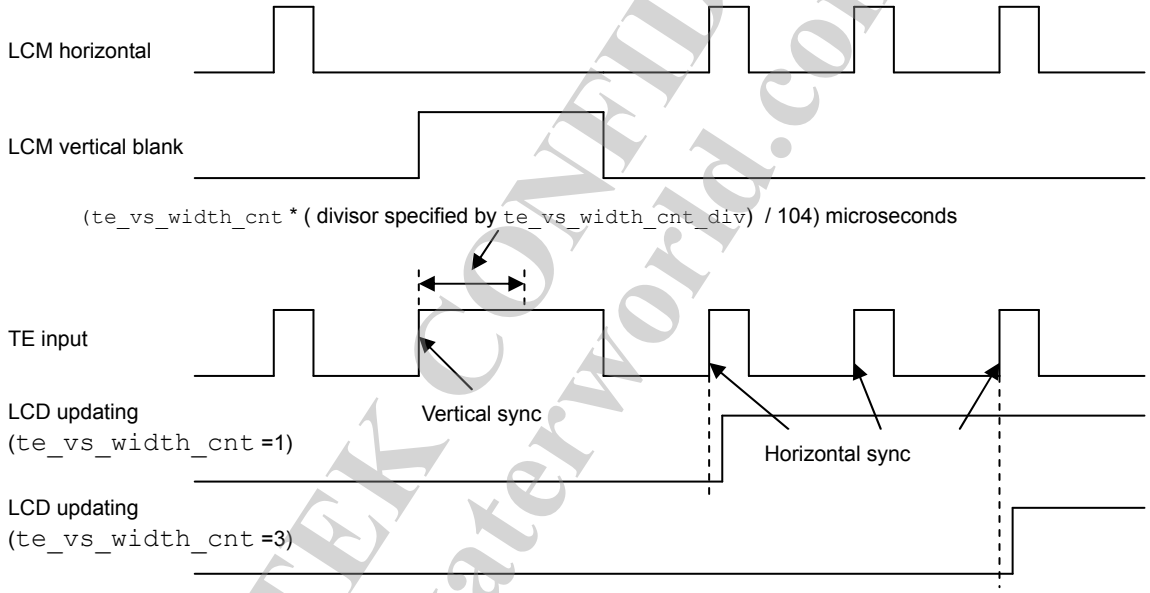
3.1.25.1 Vertical Synchronization Mode

In Vertical Synchronization mode, the LCD controller assumes TE input is only a vertical synchronization signal. In this mode, LCD controller starts to update LCM after each rising edge of the TE input (or falling edge, depends on `te_edge_polarity`). In this mode, `te_hs_delay_cnt`, `te_vs_width_cnt` and `te_vs_width_cnt_div` are not used.



3.1.25.2 Vertical and Horizontal Synchronization Mode

In Vertical and Horizontal Synchronization mode, the LCD controller assumes TE input is an OR of vertical and horizontal synchronization signal. `te_vs_width_cnt` gives the minimum time that TE input must stay active to be detected by the LCD controller as a vertical synchronization. This time is (`te_vs_width_cnt` * (divisor specified by `te_vs_width_cnt_div`) / 122.88) microseconds. Any pulse longer than this time is considered a vertical synchronization. Any pulse shorter than this time is considered a horizontal synchronization. Once a vertical synchronization has been detected, the LCD controller counts the active edges on the TE input. When the number of active edges reaches `te_vs_width_cnt`, LCD controller starts to update LCM. If a new vertical synchronization is detected before `te_vs_width_cnt` horizontal sync. have been detected, the counter is reset and horizontal synchronization count begins once again



3.1.26 LCM_DPI_PARAMS

This data structure defines the DPI interface parameters.

```
typedef struct
{
    /*
        Pixel Clock Frequency = 26MHz * mipi_pll_clk_div1
                                      / (mipi_pll_clk_ref + 1)
                                      / (2 * mipi_pll_clk_div2)
                                      / dpi_clk_div
    */
    unsigned int mipi_pll_clk_ref;   // 0..1
    unsigned int mipi_pll_clk_div1;  // 0..63
    unsigned int mipi_pll_clk_div2;  // 0..15
    unsigned int dpi_clk_div;        // 2..32
```

```
    unsigned int dpi_clk_duty;       // (dpi_clk_div - 1) .. 31

    /* polarity parameters */
    LCM_POLARITY clk_pol;
    LCM_POLARITY de_pol;
    LCM_POLARITY vsync_pol;
    LCM_POLARITY hsync_pol;

    /* timing parameters */
    unsigned int hsync_pulse_width;
    unsigned int hsync_back_porch;
    unsigned int hsync_front_porch;
    unsigned int vsync_pulse_width;
    unsigned int vsync_back_porch;
    unsigned int vsync_front_porch;

    /* output format parameters */
    LCM_DPI_FORMAT format;
    LCM_COLOR_ORDER rgb_order;
    unsigned int is_serial_output;  // enable sequential R/G/B output

    /* intermediate buffers parameters */
    unsigned int intermediat_buffer_num; // 2..3

    /* iopad parameters */
    LCM_DRIVING_CURRENT io_driving_current;

} LCM_DPI_PARAMS;
```

As shown in code comment and Figure 3-2, the DPI pixel clock frequency can be calculated as:

$$f_{DPI\_CLK} = \frac{26\text{MHz} \times (\text{mipi\_pll\_clk\_div1}+1)}{(2 \times \text{mipi\_pll\_clk\_div2}) \times \text{dpi\_clk\_div}}$$

And Table 5-1 lists some possible f_{out} values. Find the closest one according to the typical pixel clock specified in the LCM datasheet.

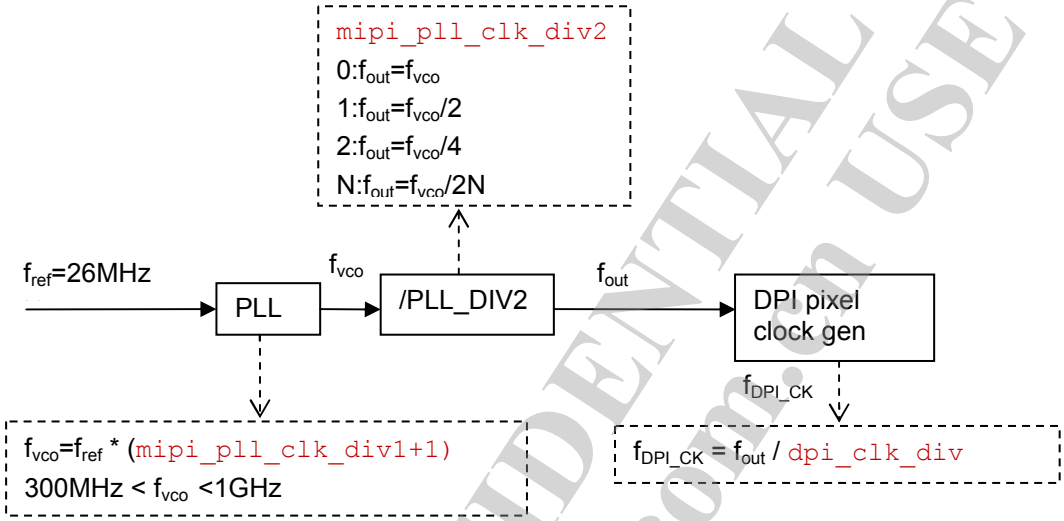


Figure 3-2 DPI Pixel Clock Generation Block Diagram

ref clock frequency			26															
MIPITX_CON3.RG_PLL_PODIV			0															
post divider			1															
			div2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
div1 (dec)	div1 (hex)	fvco (MHz)	divide facto	1	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
20	14	546		546.0	273.0	136.5	91.0	68.3	54.6	45.5	39.0	34.1	30.3	27.3	24.8	22.8	21.0	19.5	18.2
21	15	572		572.0	286.0	143.0	95.3	71.5	57.2	47.7	40.9	35.8	31.8	28.6	26.0	23.8	22.0	20.4	19.1
22	16	598		598.0	299.0	149.5	99.7	74.8	59.8	49.8	42.7	37.4	33.2	29.9	27.2	24.9	23.0	21.4	19.9
23	17	624		624.0	312.0	156.0	104.0	78.0	62.4	52.0	44.6	39.0	34.7	31.2	28.4	26.0	24.0	22.3	20.8
24	18	650		650.0	325.0	162.5	108.3	81.3	65.0	54.2	46.4	40.6	36.1	32.5	29.5	27.1	25.0	23.2	21.7
25	19	676		676.0	338.0	169.0	112.7	84.5	67.6	56.3	48.3	42.3	37.6	33.8	30.7	28.2	26.0	24.1	22.5
26	1A	702		702.0	351.0	175.5	117.0	87.8	70.2	58.5	50.1	43.9	39.0	35.1	31.9	29.3	27.0	25.1	23.4
27	1B	728		728.0	364.0	182.0	121.3	91.0	72.8	60.7	52.0	45.5	40.4	36.4	33.1	30.3	28.0	26.0	24.3
28	1C	754		754.0	377.0	188.5	125.7	94.3	75.4	62.8	53.9	47.1	41.9	37.7	34.3	31.4	29.0	26.9	25.1
29	1D	780		780.0	390.0	195.0	130.0	97.5	78.0	65.0	55.7	48.8	43.3	39.0	35.5	32.5	30.0	27.9	26.0
30	1E	806		806.0	403.0	201.5	134.3	100.8	80.6	67.2	57.6	50.4	44.8	40.3	36.6	33.6	31.0	28.8	26.9
31	1F	832		832.0	416.0	208.0	138.7	104.0	83.2	69.3	59.4	52.0	46.2	41.6	37.8	34.7	32.0	29.7	27.7
32	20	858		858.0	429.0	214.5	143.0	107.3	85.8	71.5	61.3	53.6	47.7	42.9	39.0	35.8	33.0	30.6	28.6
33	21	884		884.0	442.0	221.0	147.3	110.5	88.4	73.7	63.1	55.3	49.1	44.2	40.2	36.8	34.0	31.6	29.5
34	22	910		910.0	455.0	227.5	151.7	113.8	91.0	75.8	65.0	56.9	50.6	45.5	41.4	37.9	35.0	32.5	30.3
35	23	936		936.0	468.0	234.0	156.0	117.0	93.6	78.0	66.9	58.5	52.0	46.8	42.5	39.0	36.0	33.4	31.2
36	24	962		962.0	481.0	240.5	160.3	120.3	96.2	80.2	68.7	60.1	53.4	48.1	43.7	40.1	37.0	34.4	32.1
37	25	988		988.0	494.0	247.0	164.7	123.5	98.8	82.3	70.6	61.8	54.9	49.4	44.9	41.2	38.0	35.3	32.9
38	26	1014		1014.0	507.0	253.5	169.0	126.8	101.4	84.5	72.4	63.4	56.3	50.7	46.1	42.3	39.0	36.2	33.8
39	27	1040		1040.0	520.0	260.0	173.3	130.0	104.0	86.7	74.3	65.0	57.8	52.0	47.3	43.3	40.0	37.1	34.7

Table 3-1 f_{out} Frequency Table

Figure 3-3 shows the blanking timing diagram:

Variable	Abbrev.	Description
hsync_pulse_width	Hsync	Horizontal sync pulse width in pixel clock
hsync_back_porch	HBP	Horizontal back porch width in pixel clock
hsyc_front_porch	HFP	Horizontal front porch width in pixel clock
vsync_pulse_width	VSync	Vertical sync pulse width in horizontal sync. pulse
vsync_back_porch	VBP	Vertical back porch width in horizontal sync. Pulse
vsyc_front_porch	VFP	Vertical front porch width in horizontal sync. pulse

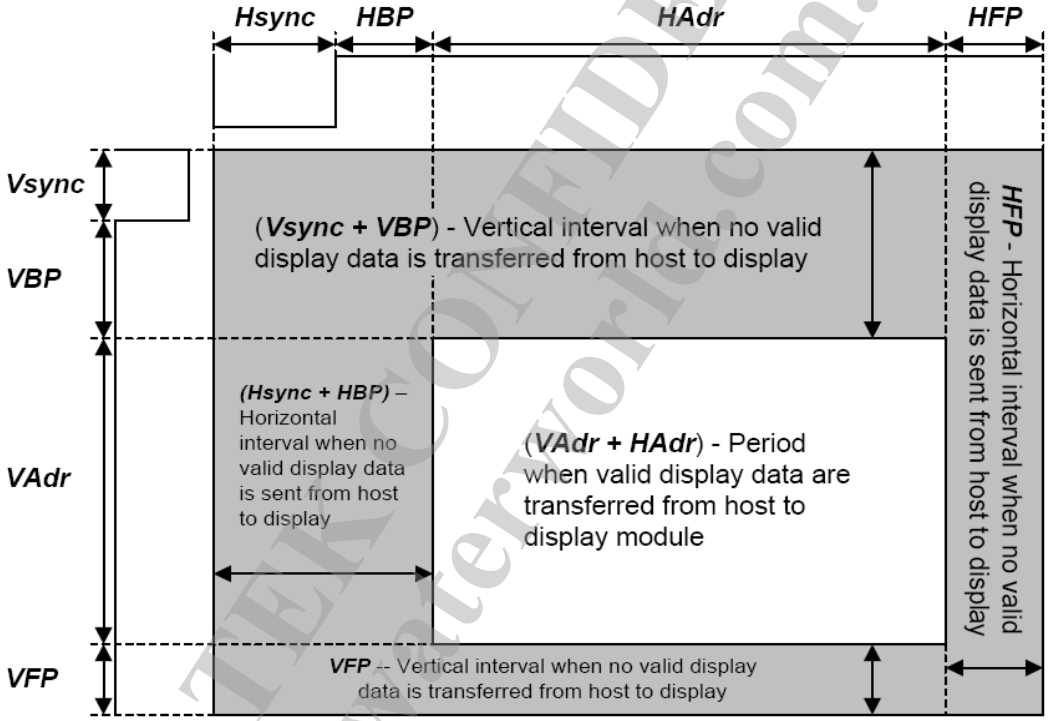


Figure 3-3 DPI Timing Parameters

Figure 3-4 shows the intermediated buffers between LCD and DPI controller. LCD controller is responsible to overlay multiple surfaces onto the intermediated buffers. And then DPI controller read the overlaid buffer and output to LCM. The intermediated buffer number can be configured by `intermediat_buffer_num`.

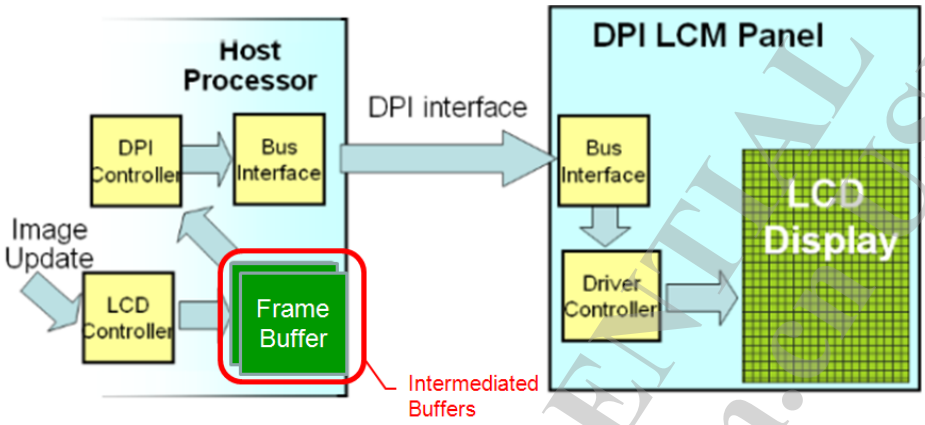


Figure 3-4 Intermediated Buffers Between LCD And DPI

3.1.27 LCM_PARAMS

This data structure defines the LCM common and particular parameters.

```
typedef struct
{
    LCM_TYPE type;
    LCM_CTRL ctrl;  // specify how to access LCM registers

    /* common parameters */
    unsigned int width;
    unsigned int height;
    unsigned int io_select_mode;
    /* particular parameters */
    LCM_DBI_PARAMS dbi;
    LCM_DPI_PARAMS dpi;
    LCM_DSI_PARAMS dsi;
} LCM_PARAMS;
```

3.1.28 LCM_UTIL_FUNCS

This data structure defines the common utility functions for LCM driver.

```
typedef struct
{
    void (*set_reset_pin)(unsigned int value);
    int  (*set_gpio_out)(unsigned int gpio, unsigned int value);

    void (*udelay)(unsigned int us);
    void (*mdelay)(unsigned int ms);

    void (*send_cmd)(unsigned int cmd);
    void (*send_data)(unsigned int data);
```

```
    unsigned int (*read_data)(void);

    void (*wait_transfer_done)(void);

} LCM_UTIL_FUNCS;
```

Function Name	Description
set_reset_pin	Out value to the LCM reset pin
set_gpio_out	Output value to the specified GPIO pin
udealy	Delay several microseconds
mdealy	Delay several milliseconds
send_cmd	Write command to the LCM
send_data	Write data to the LCM
read_data	Read data from the LCM
wait_transfer_done	Wait till data transfer to LCM is done

3.1.29 LCM_DRIVER

This data structure defines the LCM driver interface.
Note: The member of “name” and “check_status” are added for adaptive lcm driver

```
typedef struct
{
    const char* name;
    void (*set_util_funcs)(const LCM_UTIL_FUNCS *util);
    void (*get_params)(LCM_PARAMS *params);

    void (*init)(void);
    void (*suspend)(void);
    void (*resume)(void);

    void (*update)(unsigned int x, unsigned int y,
                   unsigned int width, unsigned int height);
    unsigned int (*compare_id)(void);
    void (*set_backlight)(unsigned int level);
} LCM_DRIVER;
```

Function Name	Description
name	The name of LCM, such as hx8369, this name must be unique.
set_util_funcs	Set LCM utility function interface to LCM driver
get_params	Return LCM parameters for display driver to initialize related HW components
init	Initialize the LCM
suspend	Suspend the LCM
resume	Resume the LCM

Function Name	Description
update	Send the block update commands to the LCM
set_backlight	Set back light level when using CABC to control back light
compare_id	Try to read LCM id and check whether it's valid

3.2 Global Variables

3.2.1 LCM_GetDriver

Get LCM driver interface for display driver to control LCM.
The compile option will automatically generated if adaptive lcm driver is enabled, please see customization section for details.

```
#if !defined(ADAPTIVE_LCM_DRIVER)
LCM_DRIVER* lcm_driver_list[] =
{
    &ta7601_lcm_drv
};

unsigned int lcm_count = sizeof(lcm_driver_list)/sizeof(LCM_DRIVER*);
#endif
```

Variable Name	Description
lcm_driver_list	An array stores the lcm_drv pointers.
lcm_count	Indicate how many LCM drivers we have.

4 Customization

The following shows the steps to add a new LCM driver:

(1) Create LCM driver folder `$LCM` in alps/mediatek/custom/common/kernel/lcm/
(2) Create LCM driver source file `$LCM.c` in alps/mediatek/custom/common/kernel/lcm/`$LCM`
(3) Implement LCM driver and export `lcm_driver_list/lcm_count` variables.

```
LCM_DRIVER hx8369_lcm_drv =
{
    .name           = "hx8369",
    .set_util_funcs = lcm_set_util_funcs,
    .get_params     = lcm_get_params,
    .init           = lcm_init,
    .suspend        = lcm_suspend,
    .resume         = lcm_resume,
    .update         = lcm_update,
    .set_backlight  = lcm_setbacklight,
    .compare_id     = lcm_compare_id,
    .check_status   = lcm_check_status
};
```

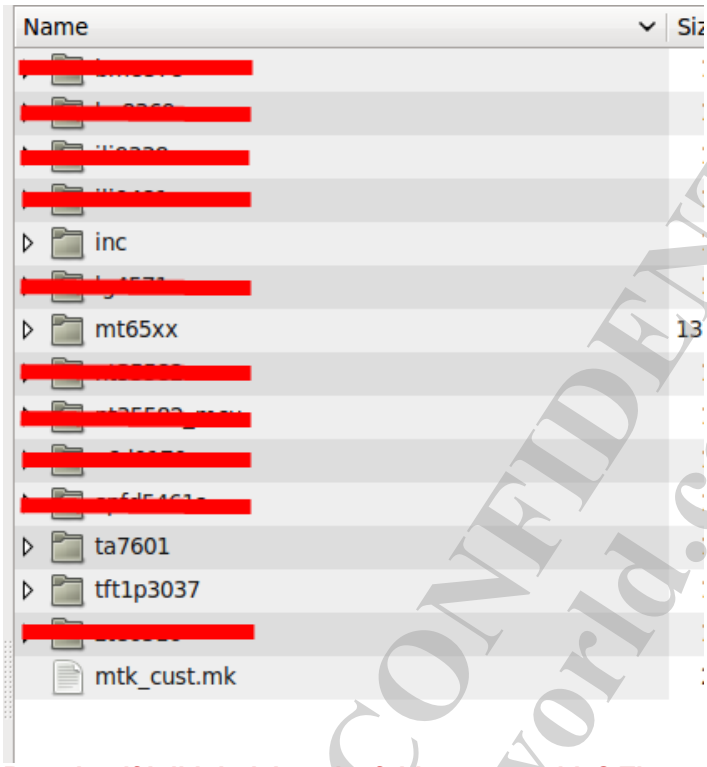
4.1 Adaptive LCM Support

At first you should add you new lcm driver as listed in the previous section, and then add the adaptive support:

1. Modify the adaptive lcm driver common file:
   alps/mediatek/custom/common/kernel/lcm/mt65xx_lcm_list.c
2. Add the new LCM driver global variables as shown in the figure below:

```
LCM_DRIVER* lcm_driver_list[] =
{
#if defined(HX8369)
    &hx8369_lcm_drv,
#endif
```

3. **[VERY IMPORTANT]** Please delete the unused LCM driver folder in the customization folder, such as shown in the figure below:



4. **But what if I didn't delete the folders un-usable? The code size will increase, because the other LCM drivers will be compiled into codebase too; And if your project doesn't define GPIO usage for serial interface, there will be build error.**
5. Modify the project makefile alps/mediate/config/(project)/ProjetConfig.mk
   Modify CUSTOM_KERNEL_LCM=mt65xx
   Modify CUSTOM_UBOOT_LCM=mt65xx

4.2 Case Study – DBI Interface LCM Driver Porting

In this chapter, we'll go through a real case study of DBI LCM driver porting.

LCM specifications:

- LCM Drive IC: hx8369
- Interface: 24-bit 80 system bus interface
- LCD size: 480*800

1. Create LCM driver folder and LCM driver source file
   alps/mediatek/custom/common/kernel/lcm/hx8369/hx8369.c

2. Modify the project makefile alps/mtk/make/$(project).mak
   Add CUSTOM_KERNEL_LCM= hx8369
   Add CUSTOM_UBOOT_LCM= hx8369

3. Fill the LCM parameters

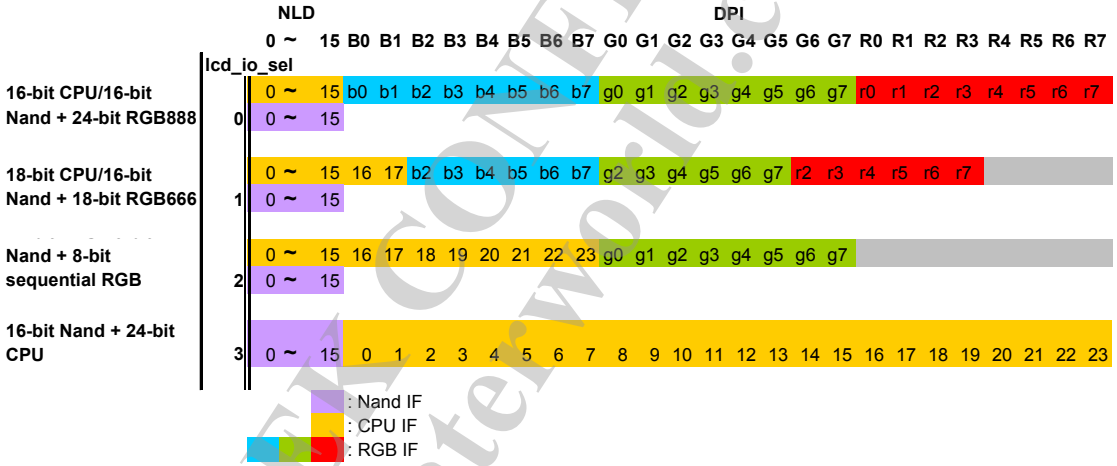
A. Configure the basic information according to the HW connection, LCM type and LCM size:

```
#define FRAME_WIDTH  (480)
#define FRAME_HEIGHT (800)

static void lcm_get_params(LCM_PARAMS *params)
{
    memset(params, 0, sizeof(LCM_PARAMS));

    params->type   = LCM_TYPE_DBI;
    params->ctrl   = LCM_CTRL_PARALLEL_DBI;
    params->width  = FRAME_WIDTH;
    params->height = FRAME_HEIGHT;
    params->io_select_mode = 3;
```

Note:

1. Please select correct DBI/DPI data IO selection, or LCM panel may not light up.
2. The width/height setting must be identical to LCM panel size arrangement

MT6573 LCD IO selection is as shown in following table. LCD data pin can share between DPI and NAND data pin



For example, we use 24bits DBI interface LCM, and connect LCM data pin to MT6573 baseband Pin DPIR[7:0], DPIG[7:0], DPIB[7:0]. So we must set io_select_mode to be 3

B. Configure data format according to the RGB data pin assignment of the LCM datasheet

```
params->dbi.clock_freq                = LCM_DBI_CLOCK_FREQ_104M;
params->dbi.data_width                = LCM_DBI_DATA_WIDTH_24BITS;
params->dbi.data_format.color_order   = LCM_COLOR_ORDER_RGB;
params->dbi.data_format.trans_seq     = LCM_DBI_TRANS_SEQ_MSB_FIRST;
params->dbi.data_format.padding       = LCM_DBI_PADDING_ON_MSB;
params->dbi.data_format.format        = LCM_DBI_FORMAT_RGB888;
params->dbi.data_format.width         = LCM_DBI_DATA_WIDTH_24BITS;
params->dbi.cpu_write_bits            = LCM_DBI_CPU_WRITE_32_BITS;
params->dbi.io_driving_current        = LCM_DRIVING_CURRENT_8MA;
```

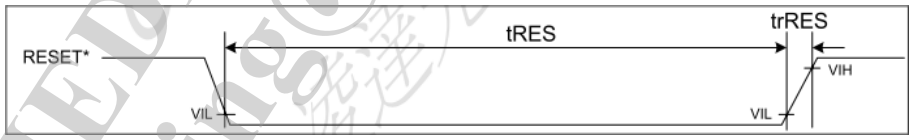
C. Configure LCM waveform timing according to the requirement specified in the LCM datasheet

```
params->dbi.parallel.write_setup     = 1;
params->dbi.parallel.write_hold      = 1;
params->dbi.parallel.write_wait      = 3;
params->dbi.parallel.read_setup      = 3;
params->dbi.parallel.read_latency    = 20;
params->dbi.parallel.wait_period     = 0;
```

Note: MT6573 LCD controller clock frequency is 122.88MHz, so clock cycle time is 1/122.88MHz = 8.13ns

4. Implement LCM init function
   According the init process specified in LCM datasheet, pull down/up the reset pin, delay and set LCM init register settings.

```
static LCM_UTIL_FUNCS lcm_util = {0};

#define SET_RESET_PIN(v)    (lcm_util.set_reset_pin((v)))

#define UDELAY(n) (lcm_util.udelay(n))
#define MDELAY(n) (lcm_util.mdelay(n))


// ---------------------------------------------------------------
//  Local Functions
// ---------------------------------------------------------------

static __inline void send_ctrl_cmd(unsigned int cmd)
{
    lcm_util.send_cmd(cmd);
}

static __inline void send_data_cmd(unsigned int data)
{
    lcm_util.send_data(data&0xff);
}

static __inline unsigned int read_data_cmd(void)
{
    return 0xFF&lcm_util.read_data();
}

static __inline void set_lcm_register(unsigned int regIndex,
        unsigned int regData)
{
    send_ctrl_cmd(regIndex);
    send_data_cmd(regData);
}
static void lcm_init(void)
{
    SET_RESET_PIN(0);
    MDELAY(25);
    SET_RESET_PIN(1);
    MDELAY(50);

    init_lcm_registers();

    //Set TE register
    send_ctrl_cmd(0x35);
    send_data_cmd(0x00);

    send_ctrl_cmd(0X0044);  // Set TE
```



Item	Symbol	Unit	Min.	Typ.	Max.
Reset low-level width	tRES	ms	1	–	–
Reset rise time	trRES	µs	–	–	10

5. Implement LCM update function
   Send the block update commands to LCM

```
static void lcm_update(unsigned int x, unsigned int y,
        unsigned int width, unsigned int height)
{
    unsigned short x0, y0, x1, y1;
    unsigned short h_X_start,l_X_start,h_X_end,l_X_end,h_Y

    x0 = (unsigned short)x;
    y0 = (unsigned short)y;
    x1 = (unsigned short)x+width-1;
    y1 = (unsigned short)y+height-1;

    h_X_start=((x0&0xFF00)>>8);
    l_X_start=(x0&0x00FF);
    h_X_end=((x1&0xFF00)>>8);
    l_X_end=(x1&0x00FF);

    h_Y_start=((y0&0xFF00)>>8);
    l_Y_start=(y0&0x00FF);
    h_Y_end=((y1&0xFF00)>>8);
    l_Y_end=(y1&0x00FF);

    send_ctrl_cmd(0x2A);
    send_data_cmd(h_X_start);
    send_data_cmd(l_X_start);
    send_data_cmd(h_X_end);
    send_data_cmd(l_X_end);

    send_ctrl_cmd(0x2B);
    send_data_cmd(h_Y_start);
    send_data_cmd(l_Y_start);
    send_data_cmd(h_Y_end);
    send_data_cmd(l_Y_end);

    send_ctrl_cmd(0x29);

    send_ctrl_cmd(0x2C);
} ? end lcm_update ?
```

6. Implement LCM suspend/resume functions
   Send suspend/resume commands to LCM

```
static void lcm_suspend(void)
{
    send_ctrl_cmd(0x10);
    MDELAY(120);
}

static void lcm_resume(void)
{
    send_ctrl_cmd(0x11);
    MDELAY(120);
}
```

7. Rebuild uboot and kernel
   In the root directory:

```
./mk $(project) gen_cust
./mk $(project) remake uboot/kernel
./mk $(project) bootimage
```

4.3 Case Study – DPI Interface LCM Driver Porting

In this chapter, we'll go through a real case study of DPI LCM driver porting.

LCM specifications:

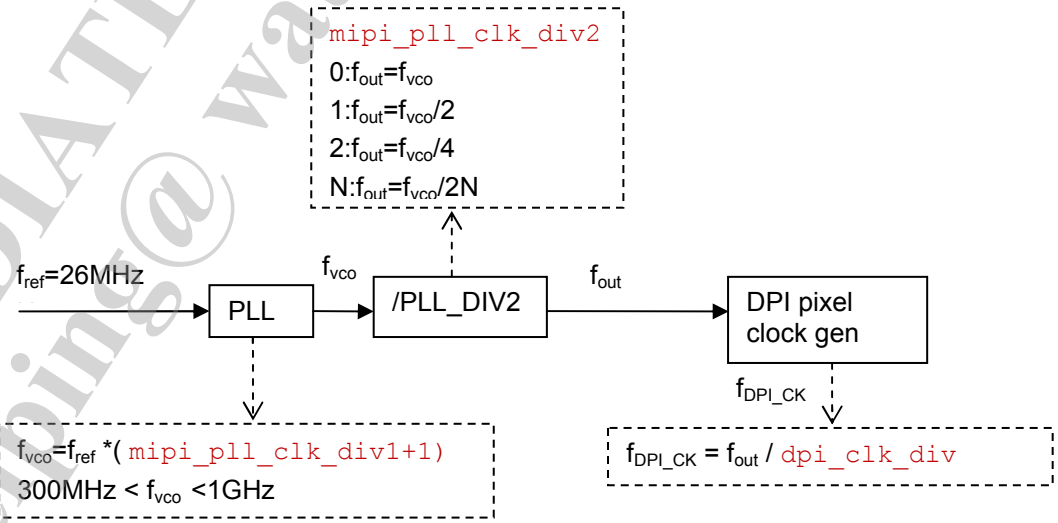
- LCM Drive IC: ili9481
- Interface: 18-bit RGB parallel data interface
- LCD size: 320*480

Basically, the DPI LCM driver porting is almost the same as DBI LCM driver porting besides step 3. and step 5.

3. Fill the LCM parameters
   According LCM datasheet, the pixel clock frequency is 8—9.6 MHz, try to lookup the Table 3-1 and find a settings closed to 9.1MHz x 2 = 18.2MHz, i.e., mipi_pll_clk_div1=20, mipi_pll_clk_div2=15. The 18.2MHz f_{out} at least to be divided by 2 for the final DPI clock frequency.

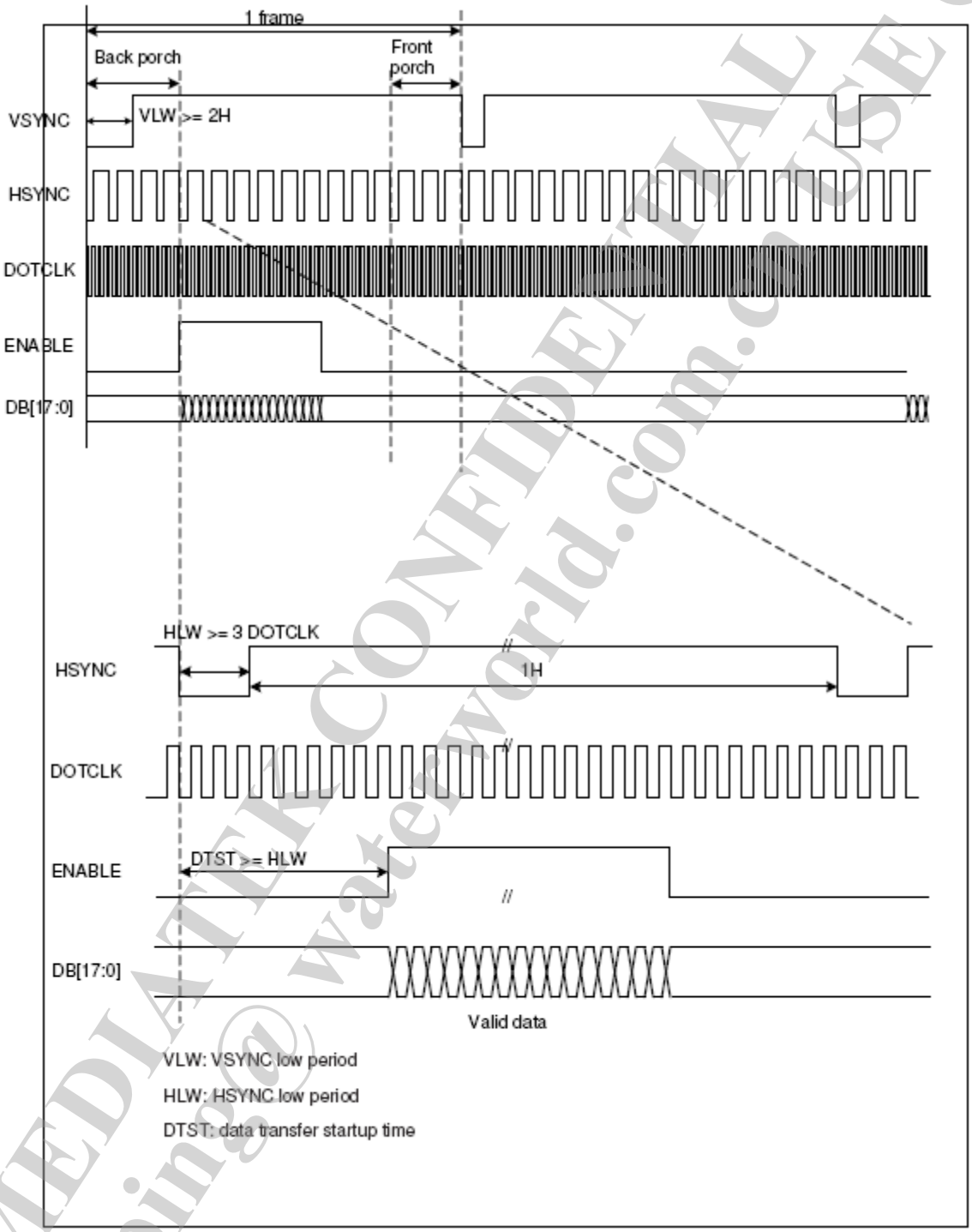
```
// CLK
params->dpi.mipi_pll_clk_ref  = 1;
params->dpi.mipi_pll_clk_div1 = 20;
params->dpi.mipi_pll_clk_div2 = 15;
params->dpi.dpi_clk_div       = 2;
params->dpi.dpi_clk_duty      = 1;
```

Vertical Frequency(*)				50	60	Hz
Horizontal Frequency(*)			-	-	-	KHz
PCLK Frequency(*)			-	8	9.6	MHz



Fill the DPI control signal polarity according to LCM datasheet

```
params->dpi.clk_pol           = LCM_POLARITY_FALLING;
params->dpi.de_pol            = LCM_POLARITY_RISING;
params->dpi.vsync_pol         = LCM_POLARITY_FALLING;
params->dpi.hsync_pol         = LCM_POLARITY_FALLING;
```



Fill the blanking timing according to the typical values specified in LCM datasheet

```
// Hsync Vsync
params->dpi.hsync_pulse_width = 2;
params->dpi.hsync_back_porch  = 3;
params->dpi.hsync_front_porch = 3;
params->dpi.vsync_pulse_width = 2;
params->dpi.vsync_back_porch  = 2;
params->dpi.vsync_front_porch = 4;
```

5. Implement LCM update function
   DPI type LCM need not implement this function.

4.4 LCM Orientation

In this chapter, we'll introduce some methods of LCM orientation configuration.
And we assume that LCM resolution is WVGA, DriverIC scan mode is from left-top to right-bottom
Please note: LCM driver width/height setting in lcm_drv.c must be identical to LCM panel size physical arrangement

1. UI Portrait mode
   a. LCM Physical scan mode is Portrait
      For example, LCM physical width is 480, and height is 800. The UI layout, image data on framebuffer and what we want to see on LCM panel are as shown in figure 4.1:

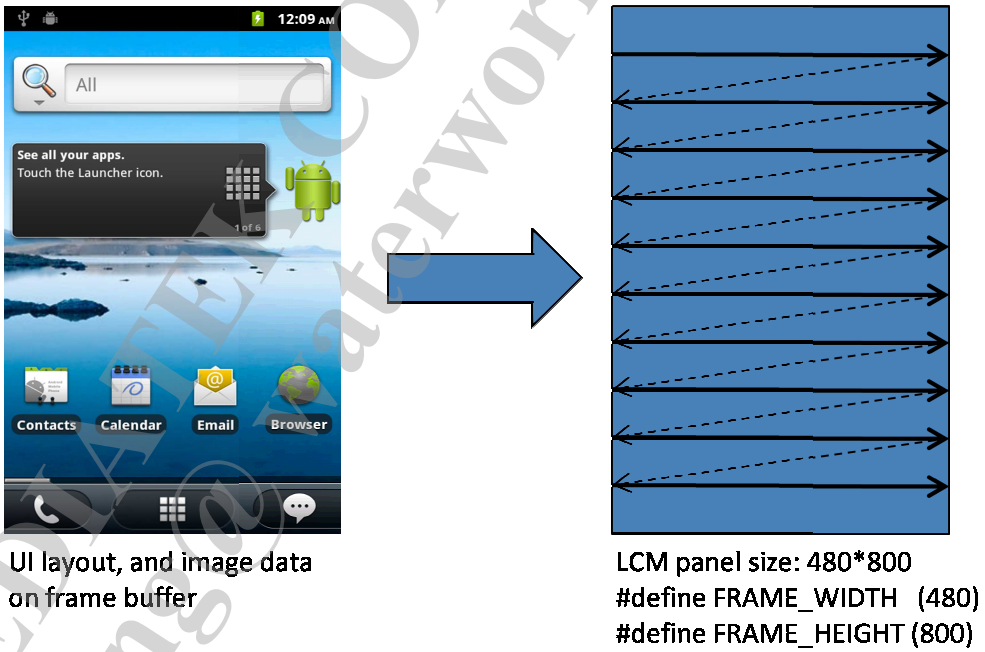


Figure 4.1 UI portrait mode and LCM rotation 0

In this case, we set some macros value in alps/mediate/config/$PROJECT/projectconfig.mk as following figure:

```
LCM_WIDTH = 480
LCM_HEIGHT = 800
BOOT_LOGO = wvga
MTK_LCM_PHYSICAL_ROTATION = 0
```

b. LCM Physical scan mode is landscape

For example, LCM physical width is 800, and height is 480. The UI layout, image data on framebuffer and what we want to see on LCM panel are as shown in figure 4.2:

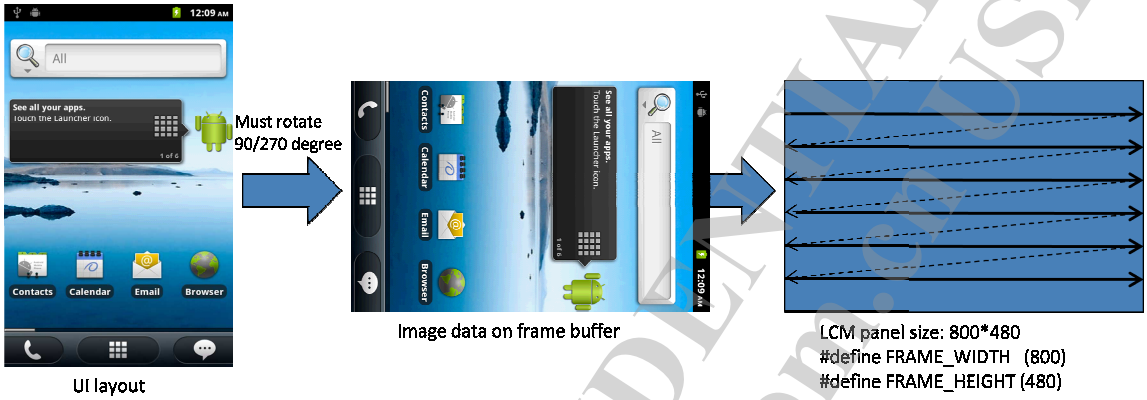


Figure 4.2 UI portrait mode and LCM rotation 90/270

In this case, we set some macros value in alps/mediate/config/$PROJECT/projectconfig.mk as following figure:

```
LCM_WIDTH = 800
LCM_HEIGHT = 480
BOOT_LOGO = wvga

MTK_LCM_PHYSICAL_ROTATION = 90
```

Or:

```
MTK_LCM_PHYSICAL_ROTATION = 270
```

2. UI Landscape mode

a. LCM Physical scan mode is Portrait

For example, LCM physical width is 480, and height is 800. The UI layout, image data on framebuffer and what we want to see on LCM panel are as shown in figure 4.3:

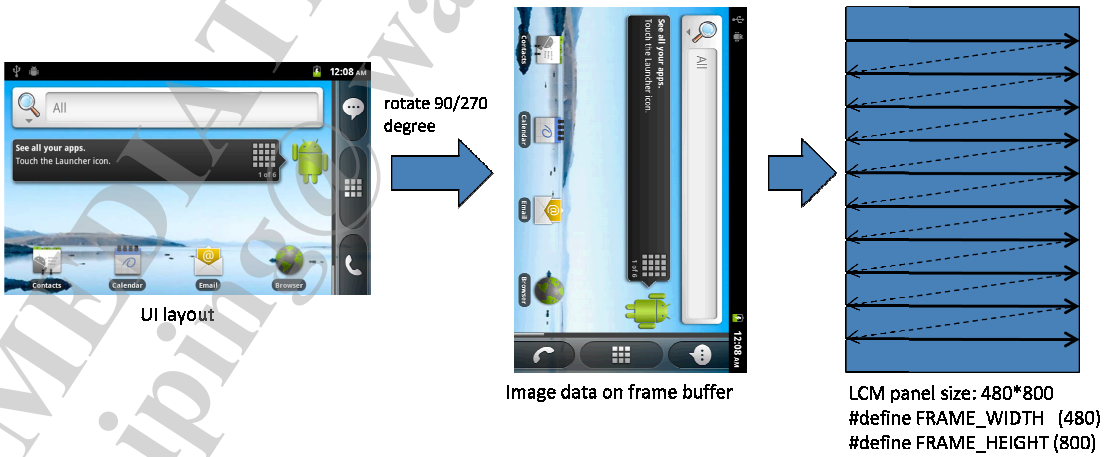


Figure 4.3 UI landscape mode and LCM rotation 90/270

In this case, we set some macros value in alps/mediate/config/$PROJECT/projectconfig.mk as following figure:

```
LCM_WIDTH = 480
LCM_HEIGHT = 800
BOOT_LOGO = wvganl
MTK_LCM_PHYSICAL_ROTATION = 90
```

Or:

```
MTK_LCM_PHYSICAL_ROTATION = 270
```

Note: if we config UI mode to landscape, we must set BOOT_LOGO to wvganl, or the boot logo would be shown error;
if there is no “wvganl” folder at mediate/custom/common/uboot/logo in your codebase, please add one “wvganl” folder including four landscape mode logos

b. LCM Physical scan mode is landscape
For example, LCM physical width is 800, and height is 480. The UI layout, image data on framebuffer and what we want to see on LCM panel are as shown in figure 4.4:

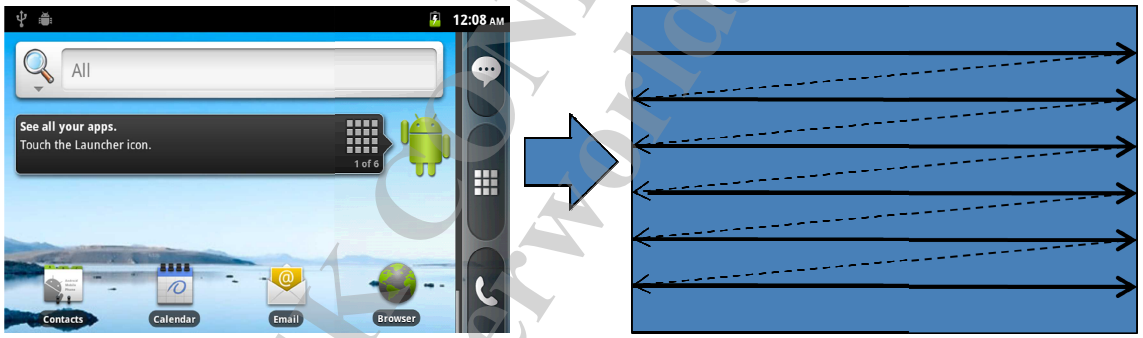


Figure 4.4 UI portrait mode and LCM rotation 90/270

In this case, we set some macros value in alps/mediate/config/$PROJECT/projectconfig.mk as following figure:

```
LCM_WIDTH = 800
LCM_HEIGHT = 480
BOOT_LOGO = wvganl
MTK_LCM_PHYSICAL_ROTATION = 0
```

3. LCM layout rotate 180 degree on PCB:
For example, LCM physical width is 480, and height is 800. Because LCM read the top-left corner pixel of frame buffer to bottom-right of panel, so we must rotate framebuffer data 180 degree. In MT6573, we use LCD controller HW module to do this work, and performance has not any drop

In this case, we set some macros value in alps/mediate/config/$PROJECT/projectconfig.mk as following figure:

```
LCM_WIDTH = 480
LCM_HEIGHT = 800
BOOT_LOGO = wvga
MTK_LCM_PHYSICAL_ROTATION = 180
```

5 Build

5.1 Source Code Structure & File Description

The LCM driver related source files are listed as Table 5-1

File	Description
alps/mediatek/custom/common/kernel/lcm/inc	
lcm_drv.h	Defines the LCM driver interface and related data structures
alps/mediatek/custom/common/kernel/lcm/$LCM	
lcm_drv.c	LCM driver
Alps/mediate/config/$PROJECT/projectconfig.mk	
Projectconfig.mk	Select LCM in project make file
alps/mediate/source/kernel/drivers/video	
mtkfb.c	MTK Linux kernel framebuffer driver
disp_drv.c	Display common library
disp_drv_dbi.c	DBI display library used for DBI type LCM
disp_drv_dpi.c	DPI display library used for DPI type LCM
disp_drv_dsi.c	DSI display library used for DSI type LCM
alps/mediate/platform/mt6573/kernel/drivers/video	
Lcd_drv.c	Kernel LCD controller driver
dpi_drv.c	Kernel DPI controller driver
dsi_drv.c	Kernel DSI controller driver
alps/mediate/platform/mt6573/uboot	
mt6573_disp_drv.c	Uboot display driver
mt6573_disp_drv_dbi.c	UBoot DBI display driver library
mt6573_disp_drv_dpi.c	Uboot DPI display driver library
mt6573_disp_drv_dsi.c	Uboot DSI display driver library

Table 5-1 LCM Driver Source File

The MTK framebuffer driver software architecture is shown as Figure 5-1

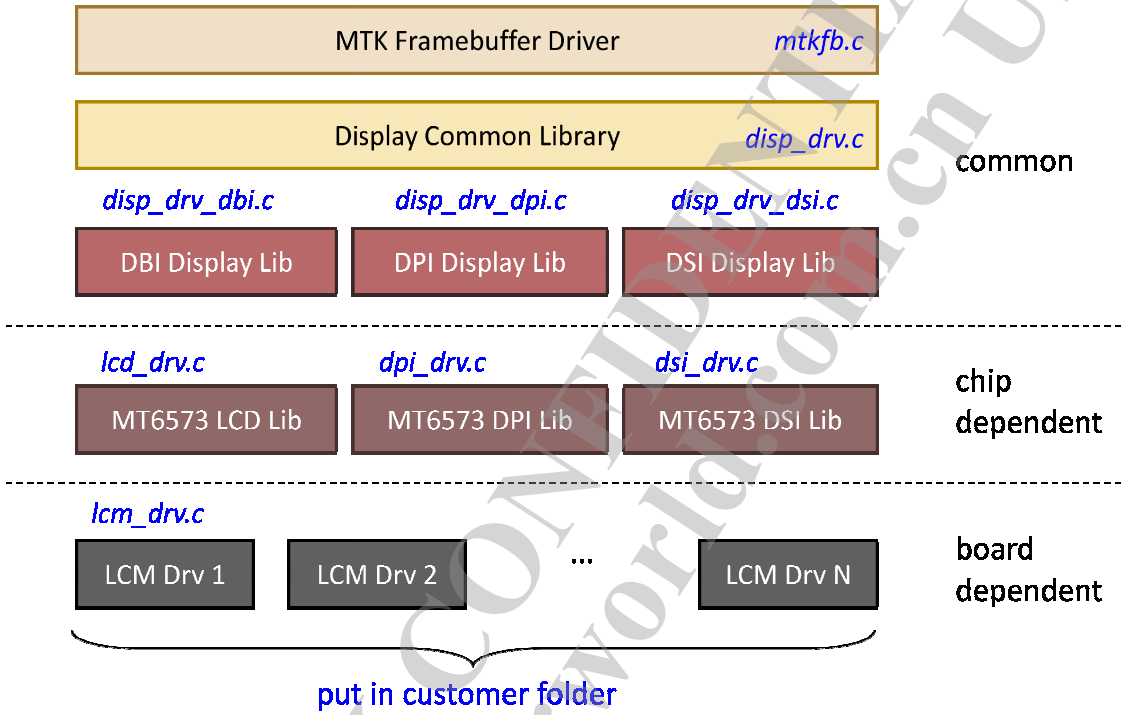


Figure 5-1 MTK Framebuffer Driver Software Architecture

5.2 Build Option

Select the LCM driver for current project by modifying the following variables in the project make file alps/mediate/config/$PROJECT/projectconfig.mk

(1) CUSTOM_KERNEL_LCM = $LCM
(2) CUSTOM_UBOOT_LCM = $LCM

Note that uboot reuses the same LCM driver code by creating the following soft link:
alps/mediatek/custom/common/uboot/lcm → alps/mediatek/custom/common/kernel/lcm