Airoha IoT SDK for RTOS LCM Porting Guide

Version: 1.2

Release date: 30 March 2018

Document Revision History

Revision	Date	Description
1.0	30 June 2016	Initial release
1.1	4 November 2016	• Added external ISINK backlight driver porting guide for MT2533.
1.2	30 March 2018	• Modified the code flow and example code to align with the latest SDK version.

Table of contents

1. **Overview** **1**
2. **Creating a New LCM Driver** **2**
   - 2.1. Create a DBI LCM driver 2
   - 2.2. Create a DSI LCM driver 5
3. **Backlight** **8**
   - 3.1. ISINK backlight 8
   - 3.2. Display PWM backlight 8
   - 3.3. LCM brightness 8
4. **Display** **10**
   - 4.1. DBI interface 10
   - 4.2. DSI interface 12

Lists of tables and figures

Table 1. The LCM functions 3

Figure 1. Example drivers 2
Figure 2. The source and header files for backlight APIs 8
Figure 3. The files and location of display APIs 10

1. Overview

Airoha IoT development platform for RTOS enables to add an LCD module (LCM) to the LinkIt HDK. MT2523 and MT2533 chipsets support three different backlight types — ISINK, Display PWM and LCM brightness, and two display output interfaces – Display Bus Interface (DBI) and Display Serial Interface (DSI). The official LCM daughterboard ST7789H2 uses ISINK for backlight and DBI for display output interface. RM69032 daughterboard uses LCM brightness and DSI for display output interface.

This guide provides detailed description on LCM porting, including the LCM driver creation and backlight control. The modifications of the board support package (BSP) layer of backlight and display are described in section 3, “Backlight” and in section 4, “Display”, respectively.

2. Creating a New LCM Driver

This section describes how to customize an existing LCM driver with DBI and DSI interfaces for a new LCM device.

The example LCM drivers are shown in Figure 1.

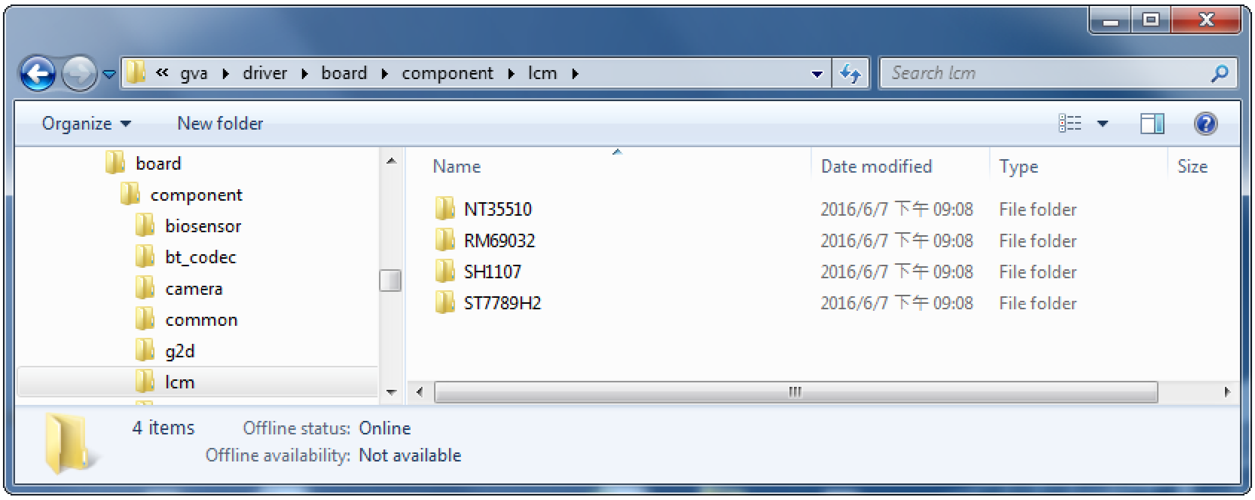

Figure 1. Example drivers

2.1. Create a DBI LCM driver

To create a DBI interface LCM driver based on ST7789H2:

1) Create a copy of the LCM driver (ST7789H2 folder) and name to a desired LCM name, such as `MyDbiLCM`.

2) Replace all occurrences of ST7789H2 in the copied LCM source files with the new name (`MyDbiLCM`).

3) Modify the LCM settings under `lcm_config_para_t` structure for DBI interface, as shown below:

```
#define MAIN_LCD_CMD_ADDR          LCD_SERIAL0_A0_LOW_ADDR
#define MAIN_LCD_DATA_ADDR         LCD_SERIAL0_A0_HIGH_ADDR

static lcm_config_para_t MyDbiLCM_para =
{
    .type = LCM_INTERFACE_TYPE_DBI,
    .backlight_type = BACKLIGHT_TYPE_ISINK,
    .main_command_address = LCD_SERIAL0_A0_LOW_ADDR,
    .main_data_address = LCD_SERIAL0_A0_HIGH_ADDR,
    .main_lcd_output = LCM_16BIT_16_BPP_RGB565_1,
    .output_pixel_width = 16,
#ifdef MTK_TE_ENABLE
    .te_enable = true,
#endif
};
```

a) Modify the macro definitions for MAIN_LCD_CMD_ADDR and MAIN_LCD_DATA_ADDR.

b) `backlight_type` can be defined as one of the options BACKLIGHT_TYPE_ISINK/BACKLIGHT_TYPE_DISPLAY_PWM/ BACKLIGHT_TYPE_LCM_BRIGHTNESS.

c) `main_command_address` should be `LCD_SERIAL0_A0_HIGH_ADDR`, if the LCM transfers commands while A0 is on high, otherwise define it as `LCD_SERIAL0_A0_LOW_ADDR`.

d) `main_data_address` should be `LCD_SERIAL0_A0_HIGH_ADDR`, if the LCM transfers data while A0 is on high, otherwise define it as `LCD_SERIAL0_A0_LOW_ADDR`.

e) `main_lcd_output` (in pixels) is defined as `LCM_16BIT_16_BPP_RGB565_1`.

   i) If 2-data lane is enabled and the output color format is RGB565, `main_lcd_output` should be `LCM_16BIT_16_BPP_RGB565_1`.

   ii) If the 2-data lane is disabled and the output color format is RGB565, the `main_lcd_output` should be `LCM_8BIT_16_BPP_RGB565_1`.

   iii) More information on the output color format can be found at `<sdk_root>\driver\board\component\common\bsp_lcd.h`.

f) `te_enable` should be `true` to enable tearing-free calculation.

4) Initialize the LCM and configure the settings. The functions (prototype names) are described in Table 1.

Table 1. The LCM functions

Function	Description
`Init()`	The initial sequence of the LCM driver IC.
`Init_lcd_interface()`	Setups the interface timing.
`BlockWrite()`	Sends the region setting to the LCM and start data transfer.
`EnterSleepMode()`	Sends display off command to the LCM.
`ExitSleepMode()`	Sends display on command to the LCM.
`EnterIdleMode()`	Sends enable idle mode command to the LCM.
`ExitIdleMode()`	Sends disable idle mode command to the LCM.
`ClearScreen()`	Fills the screen with the same color.
`ClearScreenBW()`	Fills the screen - the upper part with white and lower part with black.
`IOCTRL()`	Returns the parameter request by LCM_IOCTRL_ID_ENUM.
`CheckID()`	Returns the ID check result, if the ID can be read, return true.

a) Implement the initial sequence provided by the LCM driver IC vendor in the `LCD_Init_MyDbiLCM()` function. Replace the initial sequence in the example code with the initial sequence provided by the LCM driver IC vendor.

```
void LCD_Init_MyDbiLCM(uint16_t bkground)
{
    hal_display_lcd_toggle_reset(10, 120); /* toogle reset pin */
    /* Implment the initial code here */
    ...
    /* Clear all screen to the same color */
    LCD_ClearAll_MyDbiLCM(bkground);
}
```

b) Configure the output timing and mode settings for DBI interface in `LCD_Init_Interface_MyDbiLCM()`, as shown in the example code below.

```
  void LCD_Init_Interface_MyDbiLCM(void)
  {
      hal_display_lcd_interface_mode_t mode_settings;
      hal_display_lcd_interface_timing_t timing_settings;

      mode_settings.port_number = HAL_DISPLAY_LCD_INTERFACE_SERIAL_0;
      mode_settings.three_wire_mode = 1;
      mode_settings.cs_stay_low_mode = 0;
      mode_settings.driving_current = HAL_DISPLAY_LCD_DRIVING_CURRENT_16MA;
      mode_settings.single_a0_mode = 0;
      mode_settings.read_from_SDI = 0;
      mode_settings.width = HAL_DISPLAY_LCD_INTERFACE_WIDTH_8;
      mode_settings.hw_cs = 1;
      mode_settings.power_domain = HAL_DISPLAY_LCD_POWER_DOMAIN_1V8;
      mode_settings.start_byte_mode = 0;

      hal_display_lcd_set_interface_mode(mode_settings);

      timing_settings.port_number = HAL_DISPLAY_LCD_INTERFACE_SERIAL_0;
      timing_settings.csh = 0;
      timing_settings.css = 0;
      timing_settings.wr_low= 0;
      timing_settings.wr_high= 0;
      timing_settings.rd_low= 7;
      timing_settings.rd_high= 7;
      timing_settings.clock_freq= HAL_DISPLAY_LCD_INTERFACE_CLOCK_124MHZ;

      hal_display_lcd_set_interface_timing(timing_settings);
}
```

The configuration settings in `mode_settings` and the timing duration settings in `timing_settings` are described in the Airoha IoT SDK v4.1 or later API Reference Manual.

An example calculation of the timing parameters is described as follows.

The minimum of `css/csh/wr_low/wr_high` of `MyDbiLCM` is 6ns and the minimum of `rd_low` and `rd_high` is 60ns. The input clock is set to `HAL_DISPLAY_LCD_INTERFACE_CLOCK_124MHZ`, thus the input cycle duration is 1/124MHz ≈ 8ns. The minimum time of `css/csh/wr_low/wr_high` is 6ns and it's less than (8ns * 1). The timing counter in `hal_display_lcd_interface_timing_t` starts from 1. So the parameter of `css/csh/wr_low/wr_high` should be set to 1-1=0.

The minimum time of `rd_low` and `rd_high` is 60ns < (8ns * 8), so the `rd_low` and `rd_high` should be set to 8-1=7.

c) Implement the `LCD_BlockWrite_MyDbiLCM()` function, as in the example code shown below.

```
  LCD_BlockWrite_MyDbiLCM
  {
      LCD_CtrlWrite_MyDbiLCM(0x2A);
      LCD_DataWrite_MyDbiLCM((startx&0xFF00)>>8);
      LCD_DataWrite_MyDbiLCM(startx&0xFF);
      LCD_DataWrite_MyDbiLCM((endx&0xFF00)>>8);
      LCD_DataWrite_MyDbiLCM(endx&0xFF);
      LCD_CtrlWrite_MyDbiLCM(0x2B);
      LCD_DataWrite_MyDbiLCM((starty&0xFF00)>>8);
      LCD_DataWrite_MyDbiLCM(starty&0xFF);
      LCD_DataWrite_MyDbiLCM((endy&0xFF00)>>8);
      LCD_DataWrite_MyDbiLCM(endy&0xFF);

      LCD_CtrlWrite_MyDbiLCM(0x2C);
```



```
#ifdef MTK_TE_ENABLE
    if(MyDbiLCM_para.te_enable) {
        hal_display_lcd_start_dma(1);}
    else
#endif
    {
        hal_display_lcd_start_dma(0);
    }
}
```

Replace the region settings for (0x2A, 0x2B) using (LCD_CtrlWrite_MyDbiLCM(0x2A), LCD_CtrlWrite_MyDbiLCM(0x2B)) and memory write (0x2C) using (LCD_CtrlWrite_MyDbiLCM(0x2C)) commands provided by the LCM driver, if necessary. The rest of the settings can also be configured based on the LCM datasheet.

If TE pin is connected to the LinkIt 2523 HDK, the input parameter of the function hal_display_lcd_start_dma() should be set to 1, so the LCD engine will start transferring data once the sync signal from the LCM driver IC is received. If the TE pin isn't connected, the input parameter and the te_enable parameter in MyDbiLCM_para should be set to 0, to avoid LCD engine transfer failure.

d) Replace the command in the LCD_EnterSleepMode_MyDbiLCM()/LCD_ExitSleepMode_MyDbiLCM()/LCD_EnterIdleMode_MyDbiLCM()/LCD_ExitIdleMode_MyDbiLCM() to the correct command, if necessary.

e) Modify the return value of LCD_IOCTRL_MyDbiLCM() to the current LCM setting.

f) Implement the LCD_CheckID_MyDbiLCM() function with read ID function for the LCM driver IC.

5) Add the LCM driver to codebase. Modify the makefile at driver/board/mt25x3_hdk/module.mk to include the following:

```
C_FILES  = $(BOARD_SRC)/lcd/mt25x3_hdk_lcd.c
C_FILES  += $(COMPONENT_SRC)/lcm/ST7789H2/lcd.c
#Add the LCM driver source here
C_FILES  += $(BOARD_SRC)/backlight/mt25x3_hdk_backlight.c
```

6) Modify BSP backlight and display code flow. See section 3, "Backlight" to modify backlight driver and section 4, "Display" to modify the display driver.

2.2. Create a DSI LCM driver

To create a DSI interface LCM driver based on RM69032:

1) Create a copy of the LCM driver (RM69032 folder) and name to a desired LCM name, such as MyDsiLCM.

2) Replace all RM69032 in the copied LCM source files with MyDsiLCM.

3) Modify the LCM settings under lcm_config_para_t structure for DSI interface, as shown below:

```
static lcm_config_para_t MyDsiLCM_para =
{
    .type = LCM_INTERFACE_TYPE_DSI,
    .backlight_type = BACKLIGHT_TYPE_LCM_BRIGHTNESS,
    .main_command_address = LCD_SERIAL0_A0_LOW_ADDR,
    .main_data_address = LCD_SERIAL0_A0_HIGH_ADDR,
    .main_lcd_output = LCM_16BIT_24_BPP_RGB888_1,
    .output_pixel_width = 24,
#ifdef MTK_TE_ENABLE
    .te_enable = true,
```

```
#endif
};
```

a) The `backlight_type` is configured as one of the options `BACKLIGHT_TYPE_ISINK/BACKLIGHT_TYPE_DISPLAY_PWM/ BACKLIGHT_TYPE_LCM_BRIGHTNESS`.

b) `main_lcd_output` and `output_pixel_width` should be set to current LCM settings.

c) `te_enable` should be `true` to enable tearing-free calculation.

4) Initialize the LCM and configure the settings. The functions are described in Table 1.

a) Implement the initial sequence provided by the LCM driver IC vendor in the `LCD_Init_MyDsiLCM()` function. Replace the initial sequence in the example code with the initial sequence provided by LCM driver IC vendor.

```
void LCD_Init_MyDsiLCM(uint16_t bkground)
{
    hal_display_lcd_toggle_reset(10, 120); /* toogle reset pin */
    /* Implment the initial code here */
    ...
}
```

b) Configure the output timing settings for DSI interface in the `LCD_Init_Interface_MyDsiLCM()` function. MT2523 and MT2533 chipsets support manual configuration of the timing settings or auto calculation by PLL setting.

An example code for manual configuration where the DSI timing is up to 300Mbps is shown below.

```
void LCD_Init_Interface_MyDsiLCM(void)
{
    hal_display_dsi_dphy_timing_struct_ttiming;

    timing.da_hs_trail = 0x05;
    timing.da_hs_zero = 0x08;
    timing.da_hs_prep = 0x02;
    timing.lpx = 0x03;
    timing.da_hs_exit = 0x0C;
    timing.ta_get = 0x10;
    timing.ta_sure = 0x02;
    timing.ta_go = 0x0C;
    timing.clk_hs_trail = 0x03;
    timing.clk_hs_zero = 0x0C;
    timing.clk_hs_post = 0x09;
    timing.clk_hs_prep = 0x01;

    hal_display_dsi_set_dphy_timing(&timing);
}
```

An example code for auto calculation by PLL setting where the DSI timing is up to 300Mbps is shown below.

```
void LCD_Init_Interface_MyDsiLCM(void)
{
    hal_display_dsi_set_clock(150, false);
}
```

c) Implement the `LCD_BlockWrite_MyDsiLCM()` function, as in the example code shown below:

```
void LCD_BlockWrite_MyDsiLCM(uint16_t startx,uint16_t starty,uint16_t endx,uint16_t endy)
{
    uint32_t data_array[16];

    unsigned int x0 = startx;
    unsigned int y0 = starty;
    unsigned int x1 = endx;
    unsigned int y1 = endy;

           …
    hal_display_dsi_set_transfer_mode(HAL_DISPLAY_DSI_TRANSFER_MODE_HS);

    hal_display_lcd_start_dma(0);

#ifdef MTK_TE_ENABLE
    if(MyDsiLCM_para.te_enable) {
        if( HAL_DISPLAY_DSI_STATUS_OK !=
hal_display_dsi_start_bta_transfer(0x2C3909)) {
            MyDsiLCM_para.te_enable = false;
        }
    }
    else
#endif
    {
        data_array[0] = 0x2C3909;
        hal_display_dsi_set_command_queue(data_array, 1, 0);
    }
}
```

Replace the coordinate calculation with current LCM settings.

d) Replace the command in `LCD_EnterSleepMode_MyDsiLCM()/LCD_ExitSleepMode_MyDsiLCM()/LCD_EnterIdleMode_MyDsiLCM()/LCD_ExitIdleMode_MyDsiLCM()` to the correct command, if necessary.

e) Modify the return value of the function `LCD_IOCTRL_MyDsiLCM()` to the current LCM setting.

f) Implement the `LCD_CheckID_MyDsiLCM()` function using read ID function for the LCM driver IC.

5) Add the LCM driver to codebase. Modify `driver/board/mt25x3_hdk/module.mk` to include the following:

```
C_FILES  = $(BOARD_SRC)/lcd/mt25x3_hdk_lcd.c
C_FILES  += $(COMPONENT_SRC)/lcm/ST7789H2/lcd.c
#Add the LCM driver source here
C_FILES  += $(BOARD_SRC)/backlight/mt25x3_hdk_backlight.c
```

6) Modify BSP backlight and display code flow. See section 3, “Backlight” to modify backlight driver and section 4, “Display” to modify the display driver.

3. Backlight

The display driver provides APIs to customize the backlight settings. The source and header files for the backlight API are shown in Figure 2.

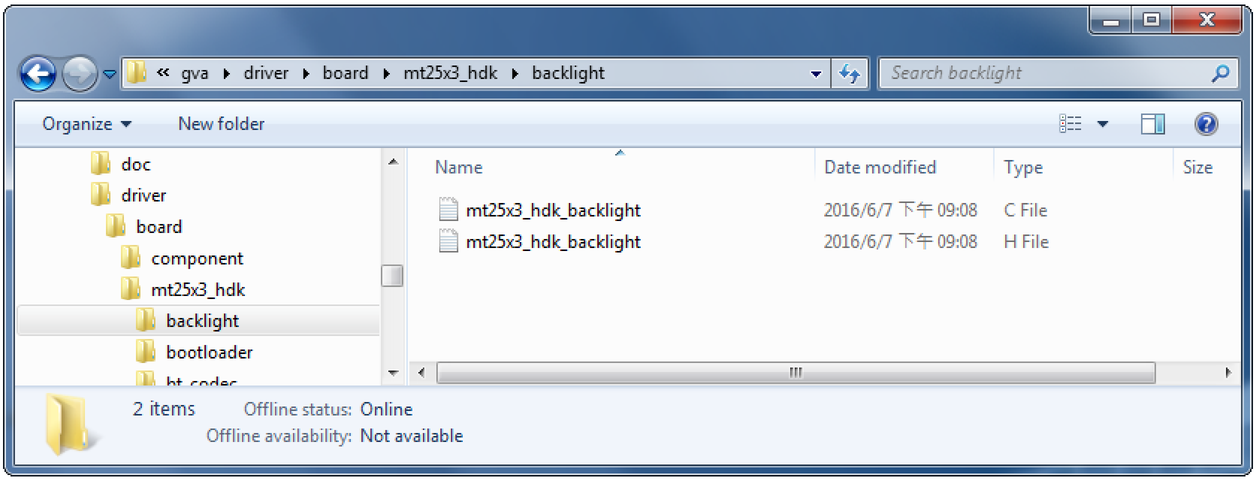

Figure 2. The source and header files for backlight APIs

3.1. ISINK backlight

ISINK backlight APIs end with postfix “`isink`” and can be used with default settings. MT2523 ISINK backlight uses the `HAL_ISINK` APIs to control the backlight output. For more details about the ISINK API usage, refer to the Airoha IoT SDK v4.1 or later API Reference Manual.

MT2533 uses external ISINK IC for ISINK backlight control. Implement the `BSP_Backlight_init_external_isink()` and `BSP_Backlight_deinit_external_isink()` functions for external ISINK backlight control.

3.2. Display PWM backlight

Display PWM backlight uses the `HAL_DISPLAY_PWM` APIs to control the backlight output. For more details about the Display PWM API usage, refer to the Airoha IoT SDK v4.1 API Reference Manual. Display PWM backlight APIs end with postfix “`display_pwm`” and the initial backlight duty can be modified.
Call the function `BSP_Backlight_init_display_pwm(void)` to initialize the display PWM settings.

```
hal_display_pwm_initialize(HAL_DISPLAY_PWM_CLOCK_26MHZ);
hal_display_pwm_set_duty(80);
```

The default duty setting is customizable.

3.3. LCM brightness

LCM brightness controls the backlight by the LCM driver IC. These APIs send data to the LCM driver IC with `hal_display_dsi_set_command_queue()`. The brightness settings can be configured, as shown below.

1) Initialize the backlight brightness settings with the function `BSP_Backlight_init_lcm_brightness(void)`. This API initializes the LCM brightness; the default brightness is 100%.

```
data_array[0] = 0x00023902;
```

```
data_array[1] = 0x51 | (0xFF << 8);
hal_display_dsi_set_command_queue(data_array, 2, 1);
```

2) De-initialize the LCM brightness by calling the function `BSP_Backlight_deinit_lcm_brightness(void)`.

```
data_array[0] = 0x00023902;
data_array[1] = 0x51 | (0x0 << 8);
hal_display_dsi_set_command_queue(data_array, 2, 1);
```

3) Call the function `BSP_Backlight_set_step_lcm_brightness(uint8_t level)` to adjust the backlight level, as shown below.

```
data_array[0] = 0x00023902;
data_array[1] = 0x51 | (level << 8);
hal_display_dsi_set_command_queue(data_array, 2, 1);
```

4. Display

The display driver provides APIs to customize the display settings. The source and header files for the display API are shown in Figure 3. For more details about display driver control flow, refer to the example project `display_drawing_image`.

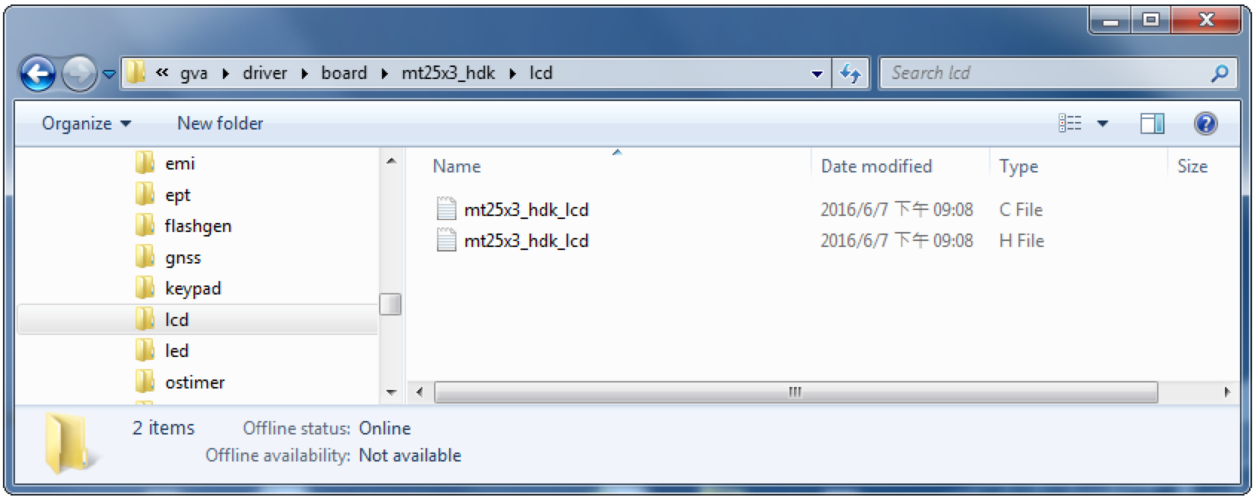

Figure 3. The files and location of display APIs

4.1. DBI interface

DBI interface uses HAL_DISPLAY_LCD APIs to control the LCD engine to output data to the LCM. Here are the APIs for application layer to control the display:

1) Call the function `bsp_lcd_init()` to configure the LCD display engine with `MainLCD` function pointer. Then initialize the LCM driver features. For example, if the LCM supports TE signal, add the following to function `bsp_lcd_init()`:

```
bsp_lcd_status_t bsp_lcd_init(uint16_t bgcolor)
{
    /* Create Semaphore for LCD engine lock and watch dog timer */
#ifdef FREERTOS_ENABLE
    if(lcd_sema == NULL) {
        vSemaphoreCreateBinary(lcd_sema);
        if(lcd_sema != NULL) {
            log_hal_info("lcd_sema create complete\n");
        } else {
            log_hal_info("Can't creare lcd_sema create\n");
            return BSP_LCD_STATUS_ERROR;
        }
    }
    lcd_watchdog_timer = xTimerCreate("LCD_WTD", (3 * 1000 /
portTICK_PERIOD_MS), pdFALSE, NULL,
(TimerCallbackFunction_t)bsp_lcd_watchdog);
#endif

    /* Turn on LCD engine and retore interface setting */
    bsp_lcd_power_on();
    MainLCD = &MyDbiLCM; /* Assign to the dedicated LCM driver */
```

```
    /* Initialize the LCD engine features based on LCM driver */
    ...
    /* For example, TE is enabled */
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__FRAME_RATE, &frame_rate);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__BACK_PORCH, &back_porch);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__FRONT_PORCH, &front_porch);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_WIDTH, &width);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_HEIGHT, &height);

    hal_display_lcd_init_te(frame_rate, back_porch, front_porch, width,
height, lcm_setting.main_lcd_output);

    /* Register a callback to unlock the display driver when the frame
data is transferred */
    hal_display_lcd_register_callback((hal_display_lcd_callback_t)
bsp_lcd_callback);

    /* Backup LCD engine setting and turn off LCD engine for power saving
*/
    bsp_lcd_power_off();
}
```

Additional function calls should be inserted before the function `bsp_lcd_power_off()` to ensure the settings are applied to the engine.

2) Call the function `bsp_lcd_config_roi()` to apply ROI settings and `bsp_lcd_config_layer()` to apply layer settings, then call `bsp_lcd_update_screen()` to start frame data transmission.

   a) Call the LCM driver `BlockWrite()` function to update the screen.

   b) When the frame data is transferred, an interrupt is issued and `bsp_lcd_callback()` is called to back up the LCD engine settings and turn off the power.

For example, if the LCM supports TE signal, add the following code to the function:

```
bsp_lcd_status_t bsp_lcd_update_screen(uint32_t start_x,  uint32_t
start_y, uint32_t end_x, uint32_t end_y)
{
    ...
    /* Turn on the LCD engine and restore interface setting */
    bsp_lcd_power_on();

    /* Additional feature TE is enabled. */
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_WIDTH, &width);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_WIDTH, &height);
    hal_display_lcd_calculate_te(width, height);
    hal_display_lcd_apply_setting();

    /* Start watch dog timer */
#if defined (FREERTOS_ENABLE)
    xTimerStart(lcd_watchdog_timer, 0);
#endif

    /* Update the screen. */
    MainLCD->BlockWrite(start_x, start_y, end_x, end_y);
}
```

Additional function calls should be inserted before the function `hal_display_lcd_apply_setting()` to ensure the settings are applied to the engine.

4.2. DSI interface

DSI interface uses HAL_DISPLAY_LCD and HAL_DISPLAY_DSI APIs to control the LCD and DSI engine to output data to LCM. Here are the APIs for application layer to control the display:

1) Call the function `bsp_lcd_init()` to configure the LCD display engine with `MainLCD` function pointer. Then initialize the LCM driver features. For example, if the LCM supports TE signal add the following to function `bsp_lcd_init()`:

```
bsp_lcd_status_t bsp_lcd_init(uint16_t bgcolor)
{
    /* Create a semaphore for LCD engine lock and watch dog timer. */
#ifdef FREERTOS_ENABLE
    if(lcd_sema == NULL) {
        vSemaphoreCreateBinary(lcd_sema);
        if(lcd_sema != NULL) {
            log_hal_info("lcd_sema create complete\n");
        } else {
            log_hal_info("Can't creare lcd_sema create\n");
            return BSP_LCD_STATUS_ERROR;
        }
    }
    lcd_watchdog_timer = xTimerCreate("LCD_WTD", (3 * 1000 / 
portTICK_PERIOD_MS), pdFALSE, NULL, 
(TimerCallbackFunction_t)bsp_lcd_watchdog);
#endif

    /* Turn on LCD engine and retore interface setting */
    bsp_lcd_power_on();
    MainLCD = &MyDsiLCM;
    /* Initialize the LCD and the DSI engine features based on LCM driver 
*/
    ...
    /* For example, TE is enabled */
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__FRAME_RATE, &frame_rate);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__BACK_PORCH, &back_porch);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__FRONT_PORCH, &front_porch);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_WIDTH, &width);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_HEIGHT, &height);

    /* Register a callback to unlock the display driver when the frame 
data is transferred */
    hal_display_lcd_register_callback((hal_display_lcd_callback_t)
bsp_lcd_callback);

    /* Backup LCD engine setting and turn off LCD engine for power saving 
*/
    bsp_lcd_power_off();
}
```

a) Replace the `MainLCD` function pointer to the current LCM driver table.

b) Add `hal_display_dsi_init()` to initialize the DSI hardware.

Additional features are also initialized in this function. Find out more in the API Reference Manual.

2) Call the function `bsp_lcd_config_roi()` to apply ROI settings and `bsp_lcd_config_layer()` to apply layer settings, then call `bsp_lcd_update_screen()` to start frame data transmission.

a) Call `BlockWrite()` to update the screen.

b) When the frame data is transferred, an interrupt is issued and `bsp_lcd_callback()` is called to back up the LCD engine settings and turn off the power.

For example, if the LCM supports TE signal, add the following code to the function:

```
bsp_lcd_status_t bsp_lcd_update_screen(uint32_t start_x,  uint32_t
start_y, uint32_t end_x, uint32_t end_y)
{
    ...
    /* Turn on the LCD engine and restore interface settings. */
    bsp_lcd_power_on();
    /* Additional feature TE is enabled. */
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_WIDTH, &width);
    MainLCD->IOCTRL(LCM_IOCTRL_QUERY__LCM_WIDTH, &height);
    hal_display_lcd_calculate_te(width, height);
    hal_display_lcd_apply_setting();

    /* Start watchdog timer */
#if defined (FREERTOS_ENABLE)
    xTimerStart(lcd_watchdog_timer, 0);
#endif

    /* Update the screen. */
    MainLCD->BlockWrite(start_x, start_y, end_x, end_y);
}
```