NXP Semiconductors
Application Note

Document Number: AN12302
Rev. 0, 12/2018

i.MX RT eLCDIF RGB Mode Use Case

1. Introduction

The enhanced Liquid Crystal Display Interface (eLCDIF) is a general display controller. The block supports Red-Green-Blue (RGB) mode interface (DOTCLK) and the programmable functionalities, including:

- system bus master to source frame buffer data for display refresh;
- 8/16/18/24-bit LCD data bus for various color format;
- programmable timing and parameters for DOTCLK LCD interfaces.

The demo code used as an example in this document is the *sd_jpeg* project in the released SDK. The hardware environment is MIMXRT1050-EVKB board.

Contents

2. LCD display RGB interface mode

The i.MX RT eLCDIF controller supports the RGB (DOTCLK) LCD interface. It is a general parallel LCD data/control bus.

2.1. RGB mode introduction

LCD (TFT) display can drive three segments (1 pixel) per clock with variable electric field strength. A light source (backlight) is needed to drive light through the aligned crystal field. The RGB mode:

- Supports many colors.
- Always supports 1 pixel per clock (three segments of Red, Green, and Blue).
- Contains the color levels dependin on the number of data lines on the LCD panel and number of LCD controller data output signals. It may be 24 lines -24 bits per pixel (bpp), or 18 bpp, 16 bpp, 15 bpp, 8 bpp.
- Supports the parallel data interface. 1 clock requires 24 bits (or other formats) for 1 pixel.

Figure 1 shows the data for 1 pixel.

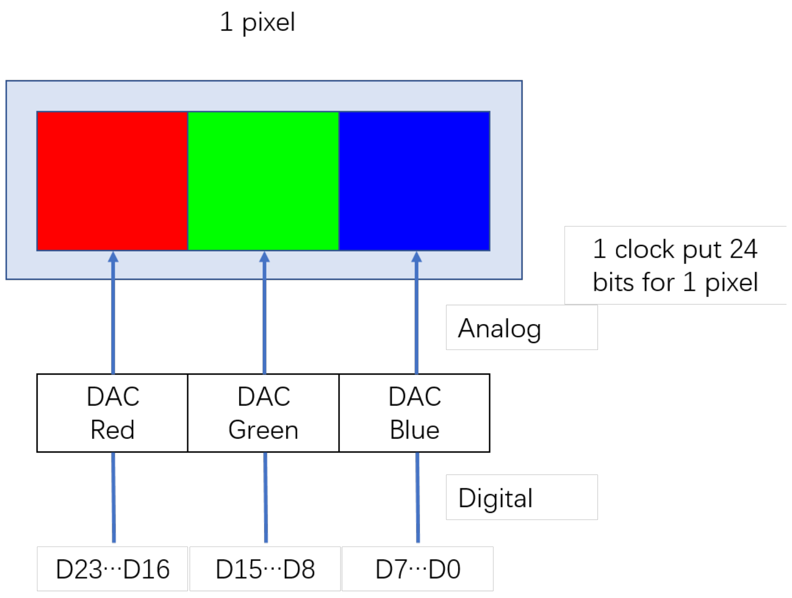


Figure 1. Data for 1 pixel

For the RGB mode interface, the LCD backlight is not controlled through the LCD controller bus. Different backlight technology is used on different panel size: large panel with fluorescent light and smaller panel with LED. Usually, the backlight is powered by a constant current source and a digital signal is used for ON/OFF control.

2.2. LCD RGB bus timing parameters

For LCD RGB mode bus, on every pixel clock edge (raising or falling) and within the LCD active area, the controller fetches one pixel data from its FIFO, converts it to the pixel panel format (coded in the RGB888 or other formats), puts it in the signal generator unit, and drives out to the RGB interface. The pixel data is then displayed on the screen.

Some timing parameters should be considered for a size of LCD display. The parameters should be programmed to match the display specifications.

Figure 2 illustrates the programmable timings and resolutions. *Table 1* shows an example of timing parameters for a 480 × 272 LCD display.

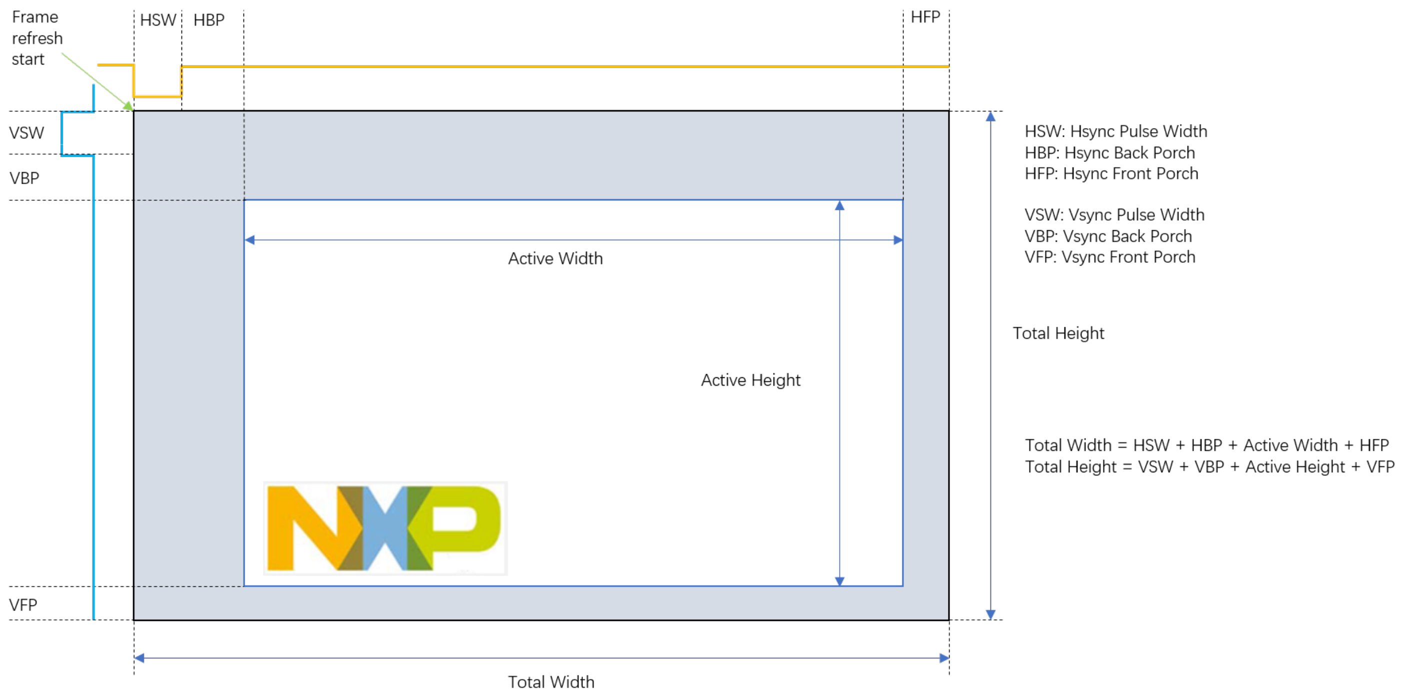


Figure 2. Programmable timings and resolution

Table 1. Timing parameters for a 480*272 LCD display

Item		Symbol	Min.	Type	Max.	Unit	Remarks
DCLK frequency		Fclk	5	9	12	MHz	—
DCLK period		Fclk	83	110	200	Ms	—
Hsync	Period time	Th	490	531	605	DCLK	—
	Display period	Thdisp	—	480	—	DCLK	—
	Back porch	Thbp	8	43	—	DCLK	By H_BLANKING setting
	Front porch	Thfp	2	8	—	DCLK	—
	Pulse width	Thw	1	—	—	DCLK	—
Vsync	Period time	Tv	275	288	335	H	—
	Display period	Tvdisp	—	272	—	H	—
	Back porch	Tvbp	2	12	—	H	By V_BLANKING setting
	Front porch	Tvfp	1	4	—	H	
	Pulse width	Tvw	1	10	—	H	

Figure 3 illustrates the process of timing parameters for a full frame.

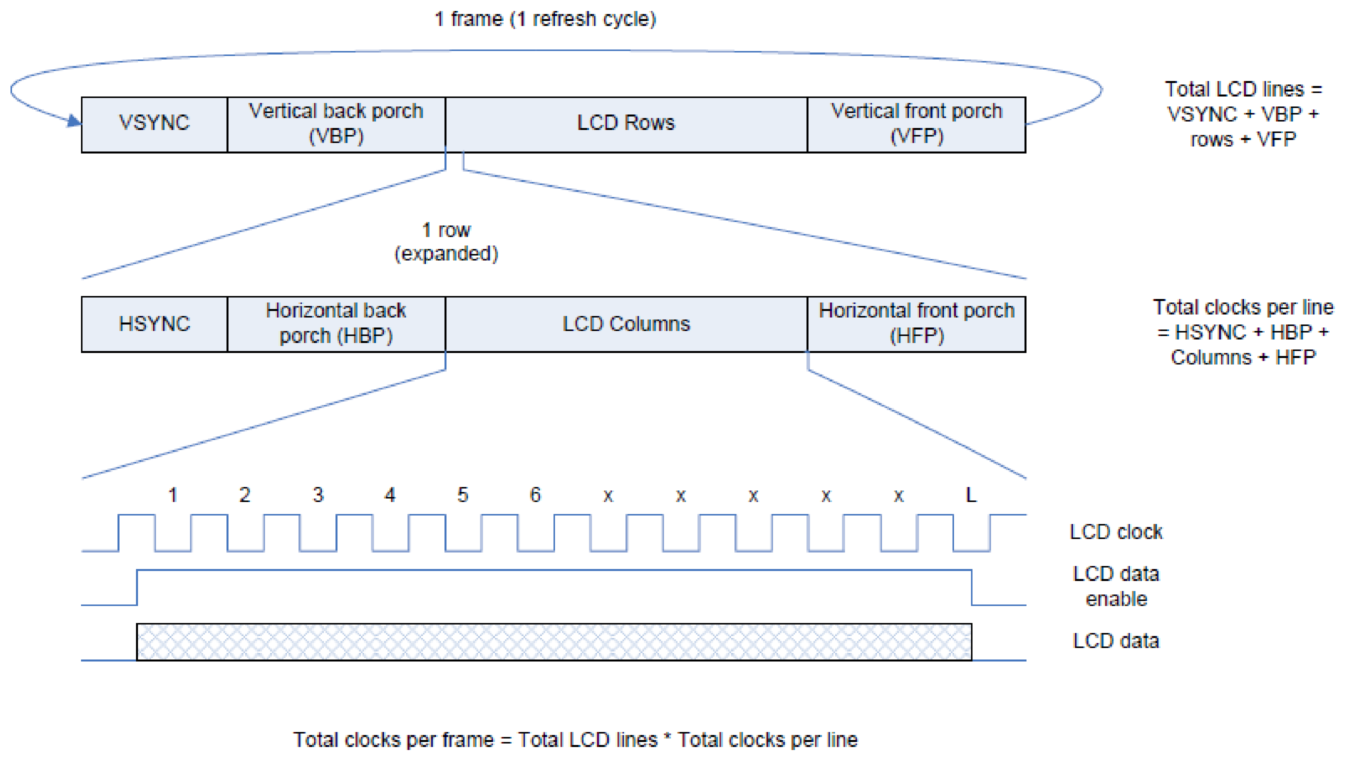

1 frame (1 refresh cycle)
VSYNC
Vertical back porch (VBP)
LCD Rows
Vertical front porch (VFP)
Total LCD lines = VSYNC + VBP + rows + VFP
1 row (expanded)
HSYNC
Horizontal back porch (HBP)
LCD Columns
Horizontal front porch (HFP)
Total clocks per line = HSYNC + HBP + Columns + HFP
1 2 3 4 5 6 x x x x x L
LCD clock
LCD data enable
LCD data
Total clocks per frame = Total LCD lines * Total clocks per line

Figure 3. Frame and line timing parameters

2.3. LCD display system

A basic embedded display system consists of an MCU, a framebuffer, a display controller, and a display glass.

- The MCU computes the image/GUI to be displayed in the framebuffer. The more often the framebuffer is updated based on the computing capability, the more fluent the display is.
- The framebuffer is a memory space used to store pixel data of the image/GUI. This memory space is usually called as the framebuffer. The required size of the framebuffer depends on the LCD resolution and color depth of the display. Use double framebuffers to avoid breaking the current displaying data.
- The display controller continuously refreshes the LCD panel and transfers the framebuffer content to the display glass 60 times per second (60 Hz). The display controller can be embedded either in the display module or in the MCU. The i.MX RT MCUs include a display controller named eLCDIF.

- The display glass is driven by the display controller and displays the image/GUI. A display glass is characterized by:
  - Display size

    The display size is defined by the number of pixels of the display with horizontal (pixels number) multiplying vertical (lines number).
  - Color depth

    The color depth defines the number of colors in which a pixel can be drawn. It is represented in bits per pixel (bpp). For a color depth of 24 bpp (which can also be represented by RGB888), a pixel can be represented in 16777216 colors.
  - Refresh rate (in Hz)

    The refresh rate is the number of times that the display panel is refreshed every second. A display shall be refreshed 60 times per second (60 Hz), as a lower refresh rate results in bad visual effects.

Figure 4 shows the basic LCD RGB mode system.

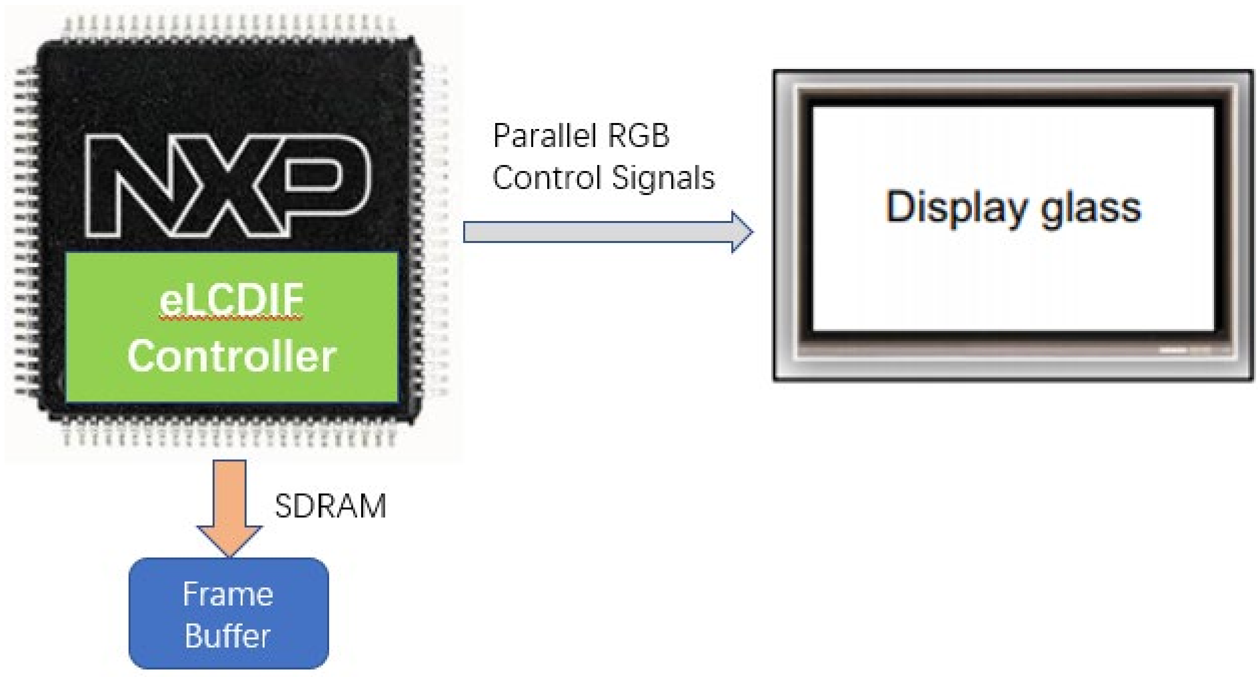


Figure 4. Basic LCD RGB mode system

For LCD display system, the framebuffer is the key point. It influences the performance of the system and visual effects.

- Memory space allocated for the framebuffer is usually shared with other system devices (CPU core, DMA, network, etc.)
- The data in framebuffer is organized as an array of bits, bytes, half-words, or words, depending on the selected color depth and color bit organization

- Buffer size is computed using (columns×rows×size of (color depth))
    - For example

800×600 display @16bpp (half-word RGB565) = 800 columns×600 rows×2 bytes/pixel = 960000 bytes

-
    - 24-bit data is stored in a 32-bit field (tossing the high byte for each pixel).

For the display compatibility, the most important step is to determine the framebuffer memory size and its location. The required framebuffer size is doubled for double framebuffer configuration. It is common to use a double buffer configuration where one buffer is used to store the current image while the other to prepare the next image.

Table 2 shows the framebuffer size for standard screen resolutions with different pixel color formats.

Table 2. Framebuffer size fir different screen resolution

Screen resolution	Number of pixels	Framebuffer size (Kbyte)			
		8 bpp	16 bpp	24 bpp	32 bpp
QVGA (320 × 240)	76800	75	150	225	300
Custom (480 × 272)	130560	128	255	383	510
HVGA (480 × 320)	153600	150	300	450	600
VGA (640 × 480)	307200	300	600	900	1200
WVGA (800 × 480)	384000	375	750	1125	1500
SVGA (800 × 600)	480000	469	938	1407	1875
XGA (1024 × 768)	786432	768	1536	2304	3072
HD (1280 × 720)	921600	900	1800	2700	3600

NOTE

The i.MX RT eLCDIF controller can support 1280 × 800 HD LCD display working at 60 fps

The framebuffer usually stores data in RGB color space format.

- Red, green, and blue components are bit fields of a pixel's color value.
    - RGB332 -> 8 bbp (Red 3, Green 3, Blue 2), organized as a byte in memory as (RRR GGG BB)
    - RGB555 -> 16 bpp (Red 5, Green 5, Blue 5), organized as a half-word in memory as (U RRRRR GGGGG BBBBB)
    - RGB565 -> 16 bpp (Red 5, Green 6, Blue 5), organized as a half-word in memory as (RRRRR GGGGGG BBBBB)
    - RGB888 -> 24 bpp (Red 8, Green 8, Blue 8), organized as a word (32-bit) in memory as (UUUUUUUU RRRRRRRR GGGGGGGG BBBBBBBB) (U = unused)

3. i.MX RT eLCDIF controller

i.MX RT includes enhanced Liquid Crystal Display Interface (eLCDIF) controller. The block supports RGB (DOTCLK) mode interface and the programmable functionalities include:

- system bus master to source framebuffer data for display refresh;
- 8/16/18/24-bit LCD data bus for different color formats;
- programmable timing and parameters for various LCD resolutions.

3.1. Basic features and hardware connection

The eLCDIF controller uses three clock domains, as shown in *Figure 5*.

- AHB clock domain

  The AHB clock domain is used to transfer data from the memories to the FIFO layer.
- APB clock domain

  The APB clock domain is used to access the configuration and status registers.
- Pixel clock domain

  The pixel clock domain is used to generate the LCD interface signals. The configurations of pixel clock output follow the panel requirements through the PLL.

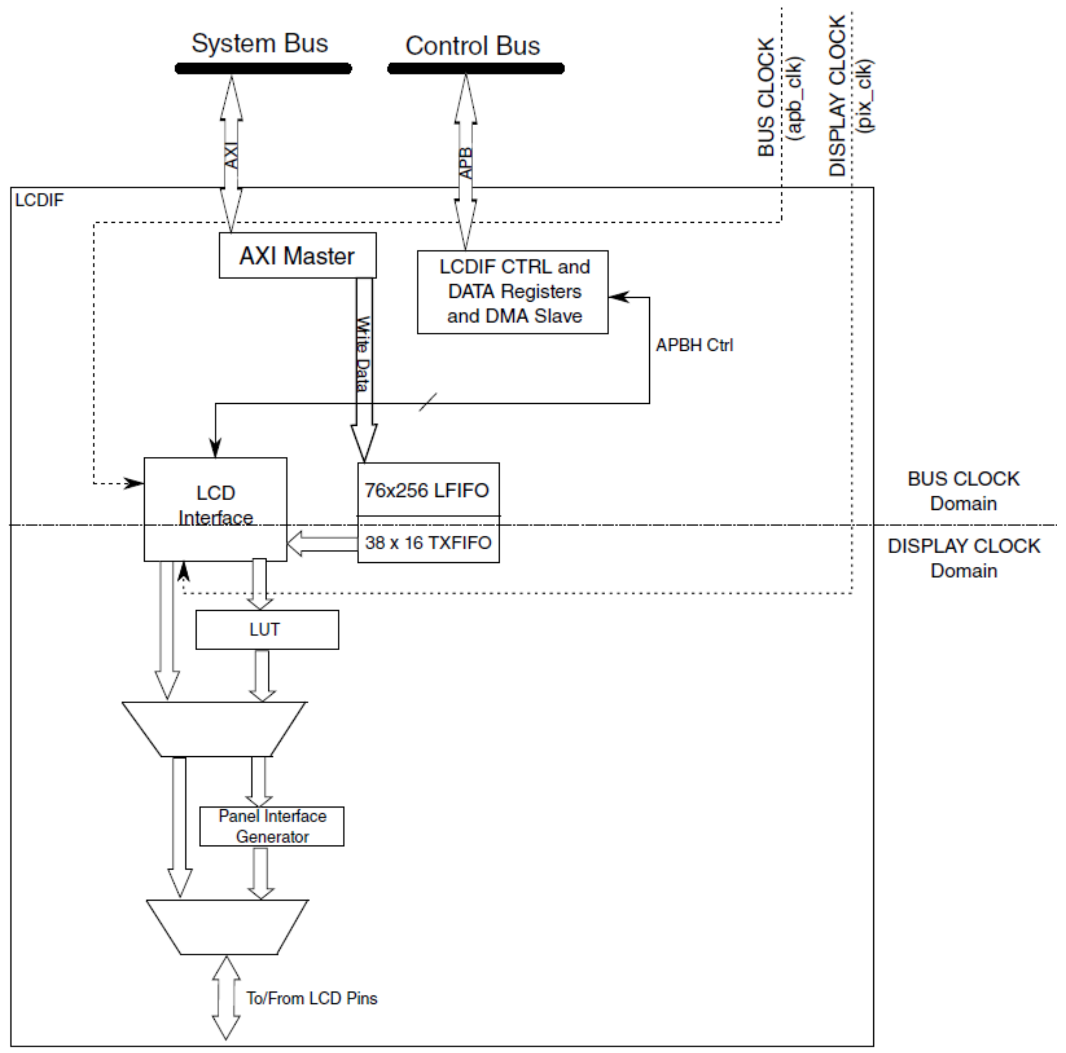


Figure 5. Top-level block of eLCDIF controller

Based on the timings and hardware interface flexibility, the eLCDIF controller can drive several monitors with different resolutions and signal polarities.

To drive LCD displays, the eLCDIF provides up to 29 signals using simple 3.3 V signaling. The signals include:

- Pixel clock: LCD_DOTCLK.
- Data enable: LCD_ENABLE.
- Reset: LCD_RESET.
- Synchronization signals: LCD_HSYNC and LCD_VSYNC.
- Pixel data RGB888.

Table 3. eLCDIF interface output signals

eLCDIF signal	Description
LCD_DOTCLK	The LCD_DOTCLK acts as the data valid signal for the LCD. The data is considered by the display only on the LCD_DOTCLK rising or falling edge.
LCD_HSYNC	The line synchronization signal (LCD_HSYNC) manages horizontal line scanning and acts as line display strobe.
LCD_VSYNC	The frame synchronization signal (LCD_VSYNC) manages vertical scanning and acts as a frame update strobe.
LCD_ENABLE	The LCD_ENABLE signal indicates to the LCD that the data in the RGB bus is valid and must be latched drawn.
LCD_RESET	The LCD_RESET signal is used to reset the LCD or touch panel.
Pixel RGB data	The eLCDIF interface can be configured to output more than one color depth. It can use up to 24 data lines (RGB888) as display interface bus.

The eLCDIF control signals polarity is programmable allowing the i.MX RT to drive any RGB parallel display. The control signals (Hsync, Vsync and data enable LCD_ENABLE), as well as the pixel clock, (LCD_DOTCLK) can be defined to be active high or active low through the eLCDIF register.

LCD panel data signals don’t always map directly to eLCDIF interface signals. However,

- unused LCD panel signals (usually the lower weighted bits) can be grounded for interfaces with less signals than the LCD panel;
- eLCDIF interface signals are kept unused for interfaces with more signals than the LCD panel;
- non-standard color mappings is used.

Figure 6 shows an example of 24-bit RGB888 eLCDIF interface to 18 bpp LCD panel.

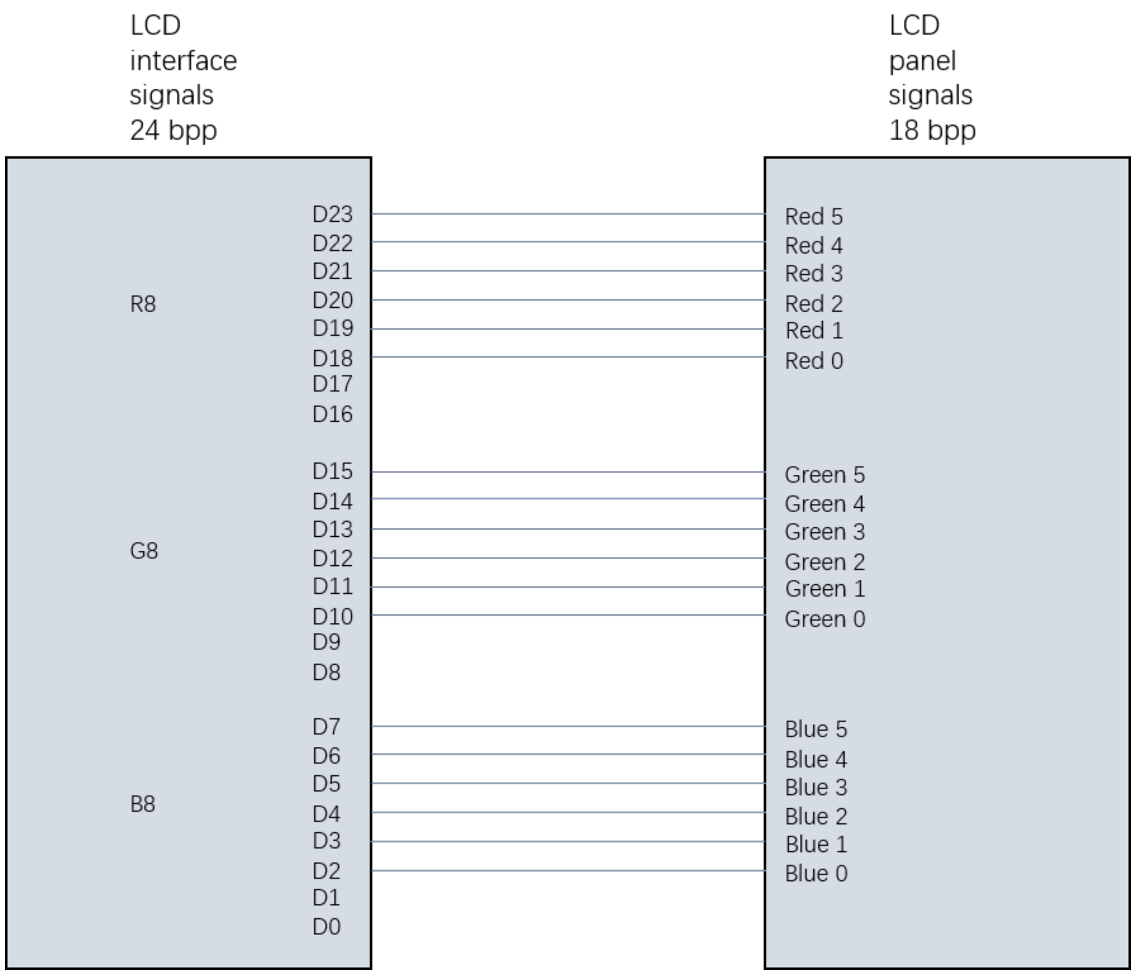


Figure 6. 24-bit RGB888 eLCDIF interface to 18bpp LCD panel

Table 4 lists the timing parameters for best performance.

Table 4. eLCDIF timing parameters

Parameter	Symbol	Min.	Max.	Unit
LCD pixel clock frequency	tCLK (LCD)	—	75	MHz
LCD pixel clock high (falling edge capture)	tCLKH (LCD)	3	—	ns
LCD pixel clock low (rising edge capture)	tCLKL (LCD)	3	—	ns
LCD pixel clock high to data valid (falling edge capture)	td (CLKH-DV)	-1	1	ns
LCD pixel clock low to data valid (rising edge capture)	td (CLKL-DV)	-1	1	ns
LCD pixel clock high to control signal valid (falling edge capture)	td (CLKH-CTRLV)	-1	1	ns
LCD pixel clock low to control signal valid (rising edge capture)	td (CLKL-CTRLV)	-1	1	ns

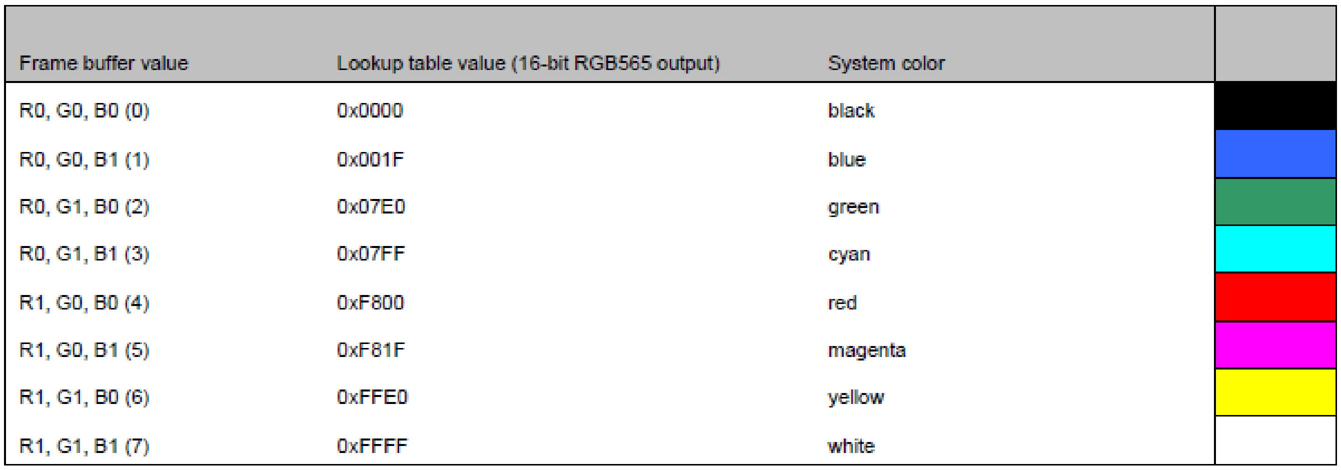
Other than the signals mentioned above, it is usual that display panel interface include other signals that are not part of the eLCDIF signals described in *Table 3*. These additional signals are required for a display module to be fully functional. The eLCDIF controller can drive only signals described in *Table 3*. The signals that are not part of the eLCDIF may be managed using GPIOs and other peripherals need specific circuits. The display panels usually embeds a backlight unit which requires an additional backlight control circuit and a GPIO. Some display panels need a serial interface, such as I^{2}C or SPI for touch panel.

The eLCDIF supports several interrupts to aid controlling and status reporting of the block:

- Underflow interrupt is asserted when the clock domain crossing FIFO (TXFIFO) becomes empty when the block is in active display portion. The software should take corrective action to prevent the event.
- Cur_frame_done interrupt occurs at the end of every frame.

In addition, the eLCDIF provides the two 256×24 bits lookup tables (LUT) to match more types of LCD cases. The framebuffer color depth and LCD panel color depth are not required to be the same. A lookup table (LUT) can be used to map a small subset to values to LCD panel color values.

Figure 7 is an example of lookup table for mapping 8-color RGB to RGB565. The example table assumes that red, green, and blue can only be turned off or turned to full intensity.



Frame buffer value	Lookup table value (16-bit RGB565 output)	System color	
R0, G0, B0 (0)	0x0000	black	⬛
R0, G0, B1 (1)	0x001F	blue	🟦
R0, G1, B0 (2)	0x07E0	green	🟩
R0, G1, B1 (3)	0x07FF	cyan	🟦
R1, G0, B0 (4)	0xF800	red	🟥
R1, G0, B1 (5)	0xF81F	magenta	🟪
R1, G1, B0 (6)	0xFFE0	yellow	🟨
R1, G1, B1 (7)	0xFFFF	white	⬜

Figure 7. 3 bpp colors (8 colors) map to 16-bit RGB565 output

3.2. NXP LCD extension board

The NXP LCD extension board is created for the i.MX RT EVK board. It can support kinds of LCD panel connecting to the i.MX RT EVK board. The EVK board exports the eLCDIF port of i.MX RT for a specific demonstration, and the port cannot be used for all of the LCD panels due to different port design from different LCD panel.

The LCD extension board can support:

- external power supply and different power switch;
- a standard port for i.MX RT EVK board;
- the switch of LCD RGB mode signals in flexibility;
- a touch panel;
- an MIPI port.

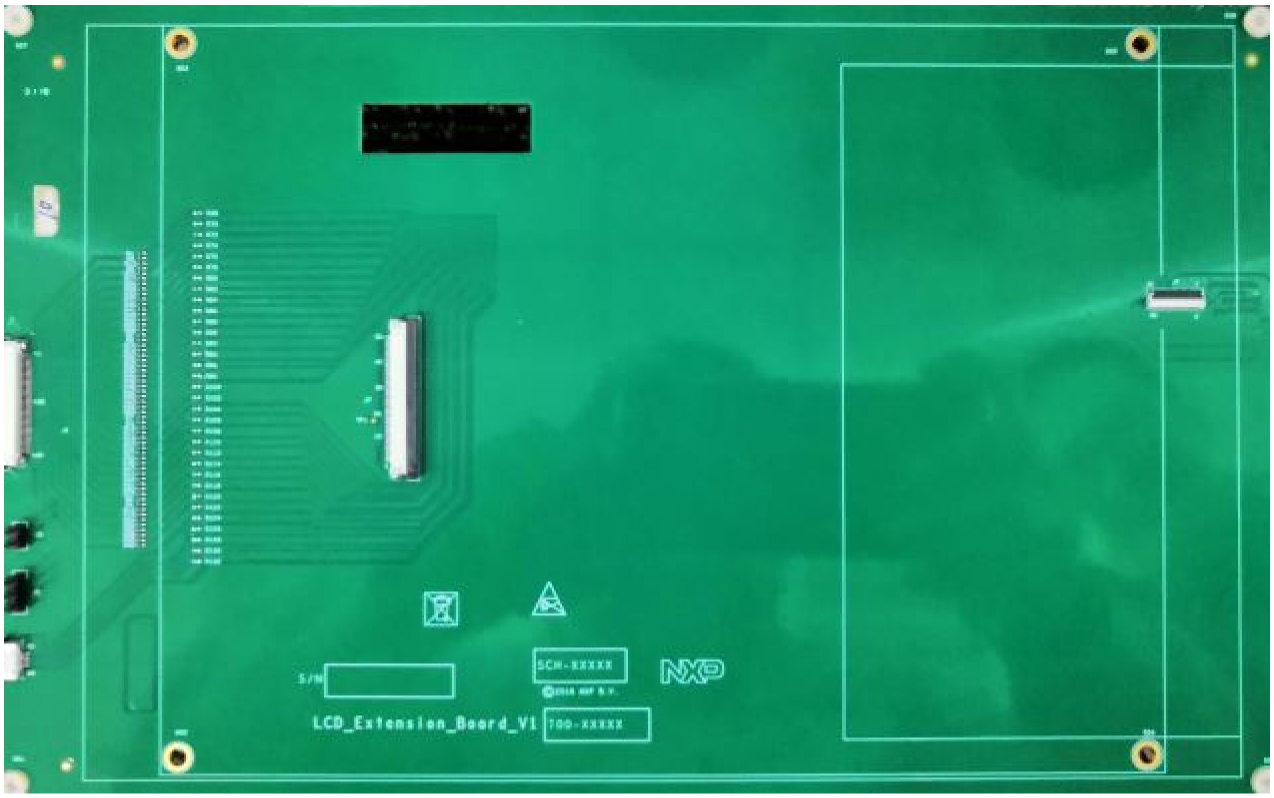


Figure 8. NXP LCD extension board

4. eLCDIF demo and performance tuning

There is a project named `sd_jpeg` in the i.MX RT SDK package. The project can demonstrate the eLCDIF controller functionalities and performance tuning. Please download the SDK package, based on your developing environment, from NXP official website.

4.1. sd_jpeg project

The `sd_jpeg` demo can be found in the `boards\evkbimxrt1050\demo_apps\sd_jpeg` directory for i.MX RT1050 SDK. The demo application reads the JPEG pictures from the SD card, decodes them, and shows them in the LCD panel one by one. For detailed steps to run the demo, please refer to the `readme.txt` in the directory.

To understand the eLCDIF demo and modify the source code to work with a 1280 × 800 HD LCD panel, the following sections describes the **i.MX RT1050EVKB + NXP LCD extension board + 1280×800 LCD panel** environment. And the toolchain uses the IAR as an example.

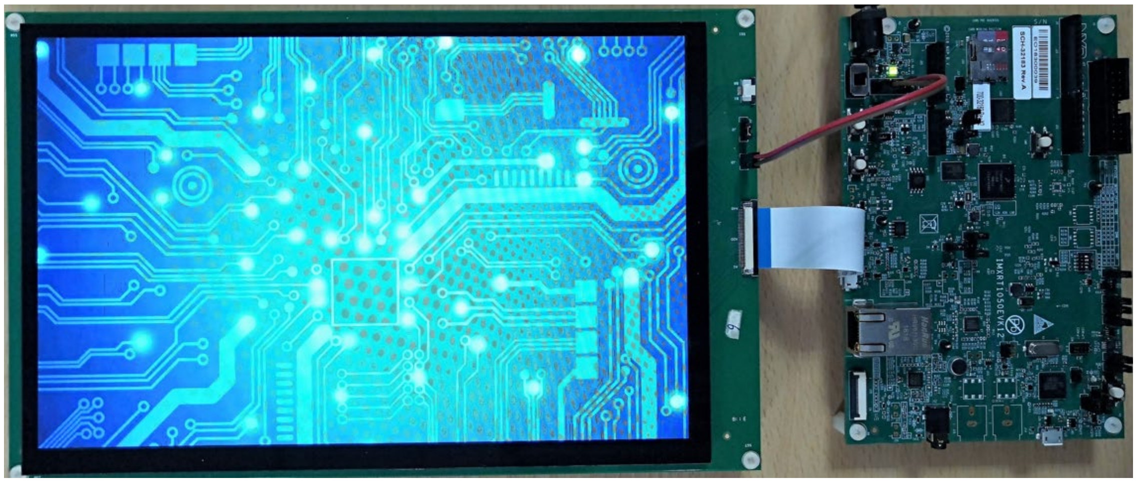


Figure 9. i.MX RT1500EVKB + NXP LCD extension board + 1280 × 800 LCD panel

The original `sd_jpeg` code is written for a 480×272 LCD panel. To modify the code for a 1280 × 800 LCD panel, please perform the following changes.

- Configure the 1280 × 800 LCD timing parameters in the `sd_jpeg.c`.

  The changes define the LCD size, HSW/HBP/HFP/VSW/VBP/VFP timing parameters, and the signal polarity.

```
#define APP_IMG_WIDTH   1280
#define APP_IMG_HEIGHT  800
#define APP_HSW 10
#define APP_HBP 80
#define APP_HFP 70
#define APP_VSW 3
#define APP_VBP 10
#define APP_VFP 10
#define APP_POL_FLAGS \
    (kELCDIF_DataEnableActiveHigh | kELCDIF_VsyncActiveLow | kELCDIF_HsyncActiveLow |
kELCDIF_DriveDataOnRisingClkEdge)
```

- Configure the right reset and backlight GPIO signals in the `sd_jpeg.c`.

```
/* Display. */
#define LCD_DISP_GPIO GPIO1
#define LCD_DISP_GPIO_PIN 2
/* Back light. */
#define LCD_BL_GPIO GPIO2
#define LCD_BL_GPIO_PIN 31
```

- Initialize the RGB mode structure with timing parameters, framebuffers, color format, and data bus format values in the `sd_jpeg.c`.

```
    const elcdif_rgb_mode_config_t config = {
        .panelWidth = APP_IMG_WIDTH,
        .panelHeight = APP_IMG_HEIGHT,
        .hsw = APP_HSW,
        .hfp = APP_HFP,
        .hbp = APP_HBP,
        .vsw = APP_VSW,
        .vfp = APP_VFP,
        .vbp = APP_VBP,
        .polarityFlags = APP_POL_FLAGS,
        .bufferAddr = (uint32_t)g_frameBuffer[0],
        .pixelFormat = kELCDIF_PixelFormatRGB888,
        .dataBus = APP_LCDIF_DATA_BUS,
};
```

- Calculate and configure the pixel clock (DOTCLK) based on the 1280 × 800 LCD timing parameters in the `sd_jpeg.c`.

```
void BOARD_InitLcdifPixelClock(void)
{
    /*
     * The desired output frame rate is 60Hz. So the pixel clock frequency is:
     * (1280 + 10 + 80 + 70) * (800 + 3 + 10 + 10) * 60 = 71M.
     * Here set the LCDIF pixel clock to 70.5M.
     */



    /*
     * Initialize the Video PLL.
     * Video PLL output clock is OSC24M * (loopDivider + (denominator / numerator)) /
postDivider = 70.5MHz.
     */
    clock_video_pll_config_t config = {
        .loopDivider = 47, .postDivider = 16, .numerator = 0, .denominator = 0,
    };

    CLOCK_InitVideoPll(&config);

    /*
     * 000 derive clock from PLL2
     * 001 derive clock from PLL3 PFD3
     * 010 derive clock from PLL5
     * 011 derive clock from PLL2 PFD0
     * 100 derive clock from PLL2 PFD1
     * 101 derive clock from PLL3 PFD1
     */
    CLOCK_SetMux(kCLOCK_LcdifPreMux, 2);

    CLOCK_SetDiv(kCLOCK_LcdifPreDiv, 0);

    CLOCK_SetDiv(kCLOCK_LcdifDiv, 0);
}
```

- Configure the eLCDIF AXI master feature register to improve the performance for 1280 × 800 LCD in the `sd_jpeg.c`.

  ```
  # In the bottom of function "void APP_ELCDIF_Init(void)", add the line:

  APP_ELCDIF->CTRL2 = 0x00700000;
  ```

- Modify the IAR `icf` file for bigger framebuffer of 1280 × 800 LCD in the `MIMXRT1052xxxxx_sdram.icf`.

  For the 1280 × 800 LCD with RGB888 color format, the framebuffer can be put into the external SDRAM memory space on the i.MX RT1050 platform. The stack and heap size of image are expanded for software decoding 1280 × 800 pictures.

  ```
  --- a/boards/evkbimxrt1050/demo_apps/sd_jpeg/iar/MIMXRT1052xxxxx_sdram.icf
  +++ b/boards/evkbimxrt1050/demo_apps/sd_jpeg/iar/MIMXRT1052xxxxx_sdram.icf
  @@ -73,13 +73,13 @@ define symbol m_ncache_end             = 0x81FFFFFF;
   if (isdefinedsymbol(__stack_size__)) {
     define symbol __size_cstack__        = __stack_size__;
   } else {
  -  define symbol __size_cstack__        = 0x1000;
  +  define symbol __size_cstack__        = 0x8000;
   }

   if (isdefinedsymbol(__heap_size__)) {
     define symbol __size_heap__          = __heap_size__;
   } else {
  -  define symbol __size_heap__          = 0x20000;
  +  define symbol __size_heap__          = 0x800000;
   }
  ```

After the above modifications of source code, with the right hardware environment, the 1280 × 800 pictures in the SD card are displayed smoothly on the 1280 × 800 LCD glass.

4.2. Improve eLCDIF performance

Larger LCD panel need larger framebuffer, so for the same refresh rate (such as, 60 fps), the eLCDIF can provide faster pixel clock and better system memory bus bandwidth.

For example, a 1280 × 800 pixel display uses 24-bit color and refreshes at 60 Hz, so:

Bytes/sec for LCD = 60×1280×800×3 bytes/pixel = 184.32MBytes/sec

The performance of the eLCDIF block can be tuned from the following points:

- Change the burst length and the outstanding cycle issuing capability in the LCDIF_CTRL2 register.
  - The LCDIF_CTRL2_OUTSTANDING_REQS field controls how many requests the eLCDIF can have in flight on any given clock cycle.
  - The LCDIF_CTRL2_BURST_LEN_8 bit sets the number of 64-bit word requested for each eLCDIF system bus request to either 8 or 16 QWORDS.
- Change the *LCDIF_THRES_PANIC* value in LCDIF_THRES register and the value can be used to optimize bus throughput and power consumption.

  The LCDIF_THRES_PANIC value can be used to raise the priority of requests initiated by the eLCDIF to alter how the eLCDIF requests are arbitrated by the system bus infrastructure. The features available with the LCDIF_THRES register require supports from system clocking and dynamic priority control. To assess the system support for these features, please refer to the appropriate block documentation.

5. Conclusion

This application note describes the background knowledge of LCD display and the use case of i.MX RT eLCDIF. For more information, please refer to:

- i.MX RT1050 Processor Reference Manual
- The `readme.txt` in the SDK *`sd_jpeg`* demo

Document Number: AN12302
Rev. 0
12/2018