AN12027

Connecting TFT LCD with LCD controller of LPC MCU

Rev. 1.0 — 21 August 2017 Application note

Document information

Info	Content
Keywords	LCD controller, LCD, TFT, GUI
Abstract	This application note introduces the LCD controller on LPC microcontrollers, TFT LCD parameters and timings together with register settings of the LCD controller. The application note covers advanced topics such as multi-frame buffering and ways to avoid LCD tearing.

Revision history

Rev	Date	Description
1.0	20170821	Initial document

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

It is a general practice to use an LCD module with integrated LCD controller to interface with a microcontroller. However, this arrangement increases the overall cost of the system. Instead, we can use an LPC microcontroller with an *on-chip LCD controller*. For a LPC microcontroller with on-chip LCD controller, a bare LCD panel can be interfaced. As these microcontrollers, do not require the LCD module to have an LCD controller integrated in it, the overall cost of the system is reduced.

2. LCD controller

Some NXP LPC series of MCUs have on-chip LCD controller to interface with bare TFT and STN LCD panels. The LPC MCUs with on-chip LCD controller are shown in Fig 1.

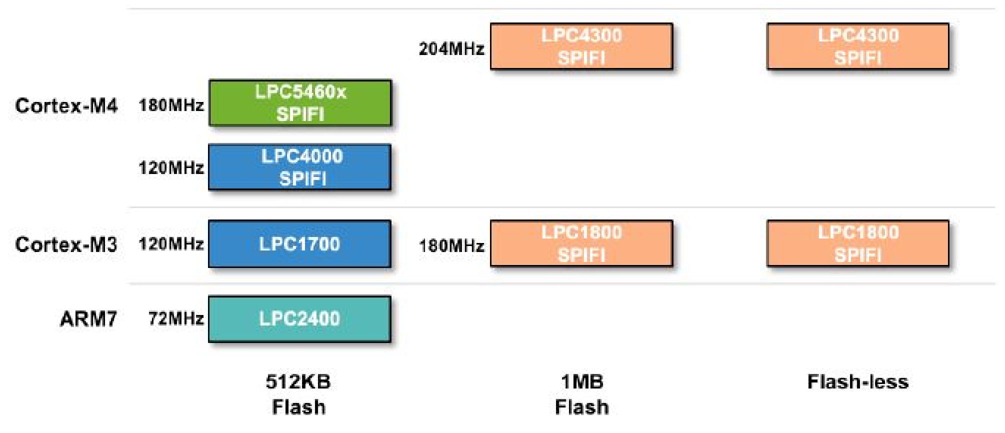


Fig 1. LPC MCUs with on-chip LCD controller

The LPC MCUs with on-chip LCD controller allow users to use a "bare" parallel LCD panel instead of LCD modules with integrated controllers. The bare LCD panels are less expensive and have a better bandwidth than the LCD modules with integrated controllers. The LCD modules with integrated controllers use SPI or GPIO interfaces.

2.1 Features

The LCD controller has, but is not limited to below key features

- AHB master interface to access frame buffer
- Dual 16-deep, 64-bit wide FIFOs for buffering incoming display data
- Supports Thin Film Transistor (TFT) color display
- Programmable display resolution up to 1024 × 768 (XGA)
- 16 bpp high-color non-palettized, for color STN and TFT
- 24 bpp true-color non-palettized, for color TFT
- Programmable timing for different display panels

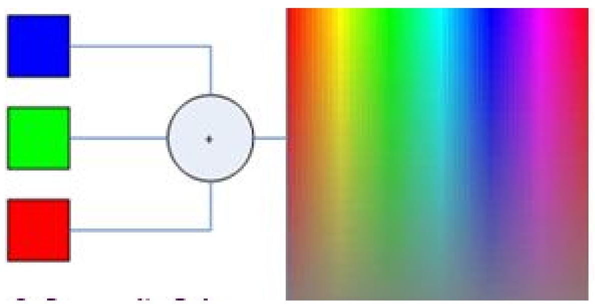
- Frame, line, and pixel clock signals
- Supports little- and big-endian data formats
- LCD panel clock is generated from the LCD peripheral clock

The LCD controller in LPC MCU supports both TFT and STN LCD panels. The application note describes the working of the LCD controller with true/high-color TFT LCD panels.

3. Driving TFT LCD

3.1 LCD color format

The basic unit of an LCD display is the pixel. A TFT pixel consists of a triplet of dots (red, green, blue) called as three element colors. Brightness of each element color within the pixel is controlled by pixel data in the framebuffer; Section 3.4. The combination of element colors with different brightness levels generate different colors; see Fig 2.



.

Fig 2. Composite Colors

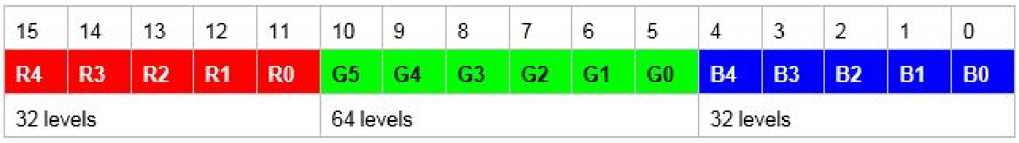
The brightness data of each element color have various bit lengths. Usually, they have length less than or equal to eight bits. If all colors are presented by 8-bit data, the color is referred as "RGB888" or "BGR888" according to the sequence.

As it is difficult for digital logic to address multiples of three, a dummy byte is appended to every 8-bit color. As a result, RGB888 requires four bytes per pixel. In embedded systems, most popular color format is RGB565. It packs the RGB colors into a 16-bit word, for easier hardware implementation and less memory requirement.

The capability of expressing 65536 colors (using 16-bit word) is sufficient for a high-quality image.

3.1.1 RGB565 format

The pixel data in memory (little-endian) for RGB565 format is shown in Fig 3.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0
32 levels					64 levels						32 levels				

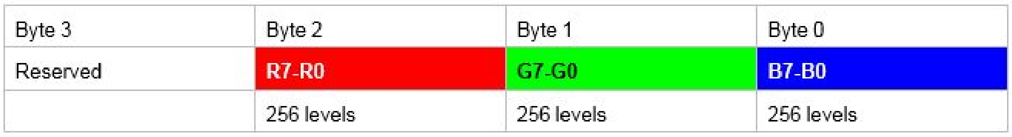
.

Fig 3. Pixel data in memory of RGB565 format

Five bits (32 levels) are used to control the brightness of red and blue color while six bits (64 levels) are used to control the brightness of green color. More number of levels for brightness control are available for green color, as the human eyes are sensitive to green.

3.1.2 RGB888 format

In the RGB888 format, every pixel uses four bytes (32 bits) in which the higher byte is not used (dummy byte); see Fig 4.



Byte 3	Byte 2	Byte 1	Byte 0
Reserved	R7-R0	G7-G0	B7-B0
	256 levels	256 levels	256 levels

.

Fig 4. Pixel data in memory of RGB888 format

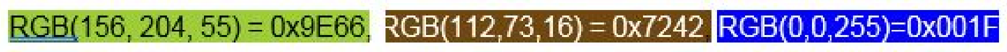
3.1.3 Macros for colors

The combination of different brightness levels of the three colors give human eyes the experience of different colors.

It is easier to use a macro instead of calculating 16-bit hexadecimal values for different colors. The pseudo code with the macro implementation is as follows:

```
#if COLOR_FORMAT == RGB565
#define RGB(r,g,b) ( ((((uint8_t)b)>>3)<<0U) + ((((uint8_t)g)>>2)<<5U) +
((((uint8_t)r)>>3)<<11U) )
#elif COLOR_FORMAT == RGB888
#define RGB(r,g,b) (((uint8_t)b)<<0UL) + (((uint8_t)b)<<8UL) +
(((uint8_t)r)<<16UL)
#endif
```

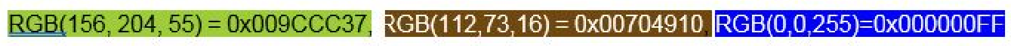
The composite color for RGB565 is shown in Fig 5. The shaded colors are the composite color result of the R,G,B brightness level settings.



.

Fig 5. Composite color for RGB565

The same color examples for RGB888 is shown in Fig 6.



.

Fig 6. Composite color for RGB888

The color data for the whole LCD panel is organized in a 2D array called as framebuffer; see Section 3.4.

Though color format configurations are flexible, LCD panels usually take eight pins for each color. Section 3.3.1 discusses about the unused least significant pins.

There is also a special palletized mode where colors in above format are saved in a special palette RAM, and pixel colors are the indexes of the RAM. This mode is for early 256-color mode and the discussion is beyond the scope of this document.

3.1.4 LCD register to configure color format

The LCD controller has registers to determine pixel format; see Table 1.

Note: Register and its field is written in format "LCD_<register name>.[MSb:LSb](<field name>)". This application note describes the functions of register bit fields. Refer User Manual for detailed information.

Table 1. LCD registers to configure color format

LCD_CTRL.[3:1] (LCDBPP)	5 = RGB888 (24 bpp), 6 = RGB565 (16 bpp), 7 = RGB444 (12 bpp)
LCD_CTRL.[5:5] (LCDTFT)	1 = TFT
LCD_CTRL.[8:8] (BGR)	0 = RGB, 1 = BGR

3.1.5 Code snippet for configuring color format

The code snippet to configure the color format for LCD is shown below. The code is based on the SDK API. The complete code for LCD setup is discussed in Section 4 of the application note. For example, the LCD panel used in LPCXpresso54608 board with the part number is "RK043FN02H- CT". According to the parameters given by the datasheet, the following code snippet (clipped from main( )) shows the necessary steps to setup LCD display color format:

```
lcdc_config_t lcdConfig;
...
lcdConfig.bpp = kLCDC_16BPP565; // Use RGB565 color format  lcdConfig.display =
kLCDC_DisplayTFT; // Use TFT type of LCD  lcdConfig.swapRedBlue = false;   // Do not swap
red and blue (BGR)
...
```

The code first initializes SDRAM that contain framebuffers, and then setup LCD controller clock.

3.2 LCD pixel organization

An LCD panel is a 2D matrix of pixels. The row and column pixel count determines the resolution of an LCD panel, expressed in column × row format. For example, if an LCD panel has 240 rows (lines) and 320 columns, its resolution is 320 × 240. Popular resolutions used in embedded systems are 320 × 240, 480 × 272, 640 × 480, 800 × 480, and 800 × 600. The LCD controller in LPC MCUs support resolution up to 1024 × 768. Example of an LCD panel with resolution 480 × 272 (480 columns, 272 rows) is shown in Fig 7.

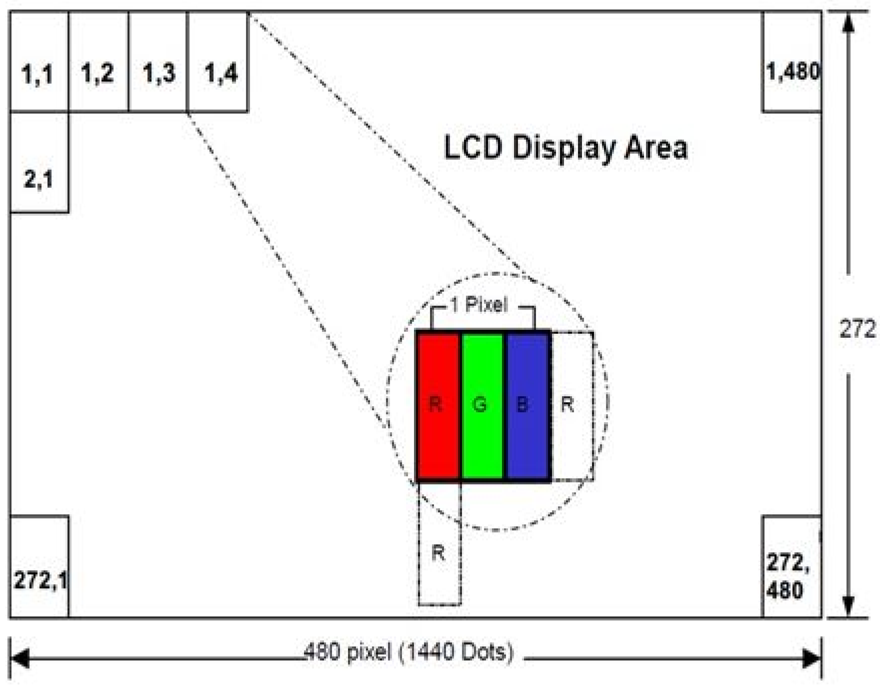


Fig 7. LCD pixel organization for a color TFT panel with 480 × 272 resolution

3.2.1 LCD register to configure LCD resolution

The LCD controller has the following registers to configure the resolution of an LCD.

Note: The register, field and field name are written in format <Register name>.[MSb:LSb](<field name>). For example, REG1.[5:2](FieldName) means register REG1, bit 5 to bit 2 (4 bits), and its name is FieldName.

Table 2. LCD registers to configure LCD resolution

LCD_TIMH.[7:2] (PPL)	columns/16 -1, columns MUST be multiplied of 16
LCD_TIMV.[9:0] (LPP)	Rows - 1
LCD_POL.[25:16] (CPL)	Set to the number of columns – 1for TFT. i.e., same as (PPL + 1) * 16 - 1

3.2.2 Code snippet of configuring LCD resolution

The data sheet for the LCD panel used in LPCXpresso54608 board RK043FN02H-CT specifies the LCD display resolution as 480 × 272.

```
#define LCD_PPL   480 // pixel per line
...
#define LCD_LPP   272 // Line per panel
...
// >>> configure LCD panel related parameters  lcdConfig.ppl = LCD_PPL;
lcdConfig.lpp = LCD_LPP;
...
```

The SDK API “LCDC_Init()” converts row and column numbers to register settings.

3.3 Signals to drive a bare LCD panel

Table 3 summarizes all the required signals (outputs) for driving a bare LCD panel.

Table 3. Signals to drive a bare LCD panel

Signal	Description
LCD PWR	LCD panel power enable
LCD_DCLK	LCD panel clock
LCD_AC	TFT data enable output
LCD_FP	vertical synchronization pulse (TFT), starting a new LCD frame
LCD_LE	line end signal
LCD_LP	horizontal synchronization pulse (TFT), starting a new LCD line
LCD_VD[23:0]	LCD panel data. Bits used depend on the panel configuration

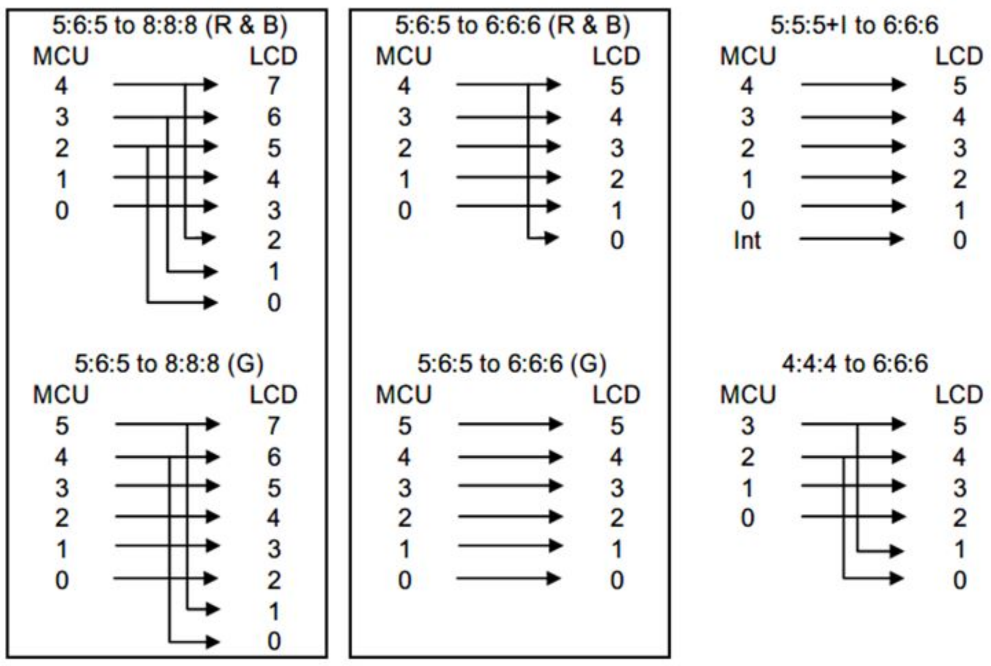
Depending on the LCD technology, 10 to 31 pins are used to drive the LCD. The LCD_DCLK is also known as the pixel clock. Configurations to connect TFT LCD panels with typical color formats are shown in Table 4.

Table 4. Configure to connect TFT LCD panels with typical color format

Pin name	12-bit, 4:4:4 mode (18 pins)	16-bit, 1:5:5:5 mode (22 pins)	16-bit, 1:5:5:5 mode (24 pins)	24-bit (30 pins)
LCD_PWR	Y	Y	Y	Y
LCD_DCLK	Y	Y	Y	Y
LCD_ENAB_M	Y	Y	Y	Y
LCD_FP	Y	Y	Y	Y
LCD_LE	Y	Y	Y	Y
LCD_LP	Y	Y	Y	Y
LCD_VD[1:0]	-	-	-	RED[1:0]
LCD_VD[2]	-	-	intensity	RED[2]
LCD_VD[3]	-	RED[0]	RED[0]	RED[3]
LCD_VD[7:4]	RED[3:0]	RED[4:1]	RED[4:1]	RED[7:4]
LCD_VD[9:8]	-	-	-	GREEN[1:0]
LCD_VD[10]	-	GREEN[0]	intensity	GREEN[2]
LCD_VD[11]	-	GREEN[1]	GREEN[0]	GREEN[3]
LCD_VD[15:12]	GREEN[3:0]	GREEN[5:2]	GREEN[4:1]	GREEN[7:4]
LCD_VD[17:16]	-	-	-	BLUE[1:0]
LCD_VD[18]	-	-	intensity	BLUE[2]
LCD_VD[19]	-	BLUE[0]	BLUE[0]	BLUE[3]
LCD_VD[23:20]	BLUE[3:0]	BLUE[4:1]	BLUE[3:0]	BLUE[7:4]

3.3.1 Unused color pins on LCD panel

If the LCD panel has eight pins for every color, user should take care of the unused least significant bits for color pins. For example, RGB565 has three unused pins for red (LCD_VD[1:0] and LCD_VD[2]) , three unused pins for blue (LCD_VD[17:16] and LCD_VD[18]) and two unused pins for green (LCD_VD[9:8]). The unused pins should not be connected to the GND, LCD cannot reach its maximum brightness. An effective method is to connect unused MSBs to used MSBs. For example, connect pin 7 to pin 1, pin 6 to pin 0 for green color; see Fig 8. It provides full brightness to the LCD and smoother color steps. Fig 8 summarizes the different cases.



.

Fig 8. Connections of unused LCD pins (least significant pins)

The leftmost connection is frequently used for RGB565 format to connect to LCD panels with 24-bit data lines.

Fig 9 shows the example of LCD connection on LPCXPresso5460x board. It uses a simple connecting to ground method:

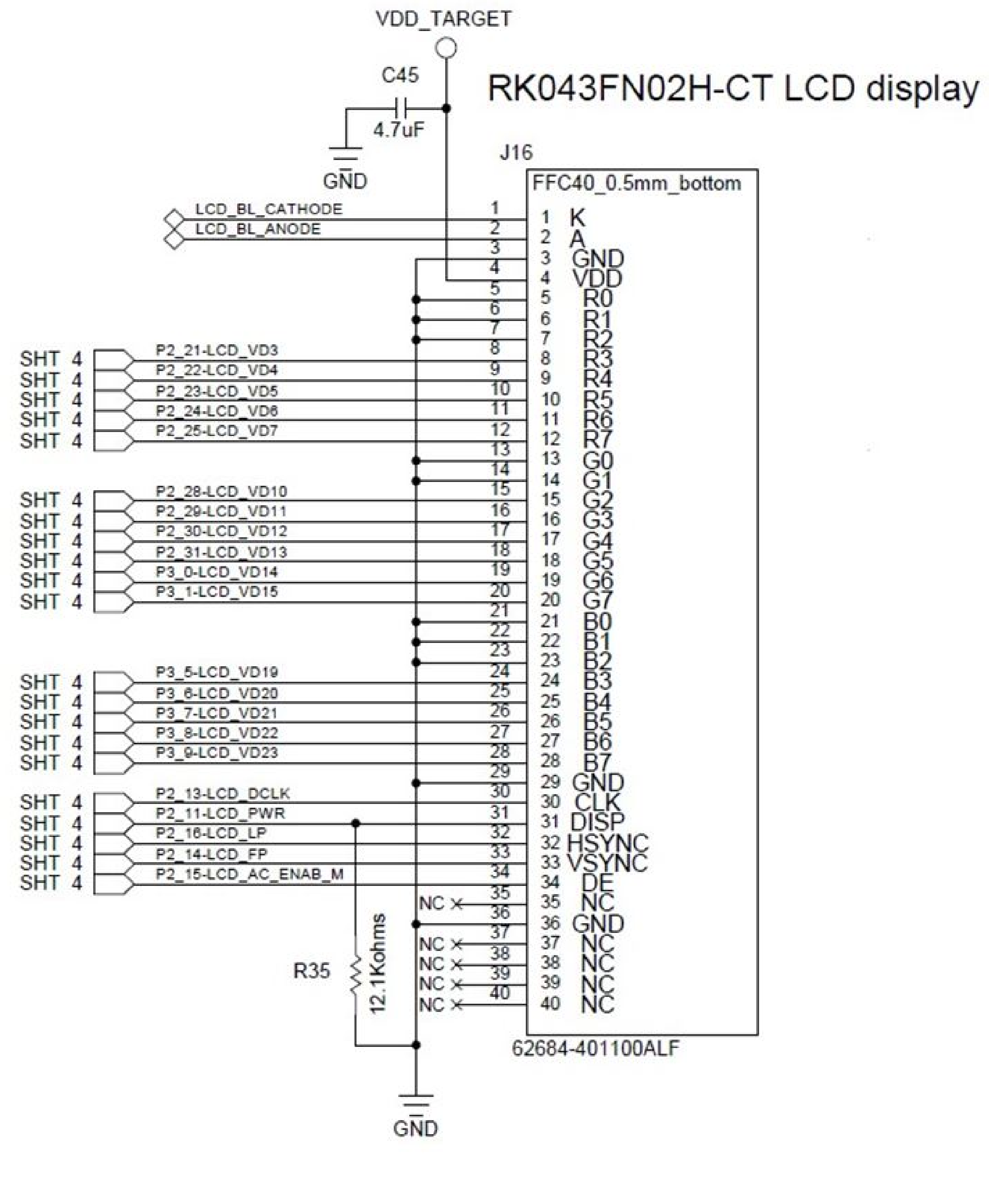


Fig 9. LCD connection on LPCXPresso54608 board

3.4 FrameBuffer (FB)

Every pixel data is saved in a separate memory known as the FrameBuffer (FB). The LCD controller reads the framebuffer and sends the pixel data to LCD data lines. It refreshes the LCD with FB data at a fixed frequency at about dozens of Frames Per Second (FPS). Each time when the LCD pixels are refreshed, data from the Framebuffer (FB) is written on the LCD_VD[ ] lines. As the data transfer rate (FPS) is usually greater than or equal to 30 FPS and the DMA of the LCD controller accesses the framebuffer memory frequently, the temperature of the framebuffer memory rises significantly.

Framebuffer uses a block of continuous memory larger than a typical on-chip SRAM for most MCUs. For example, even for a very low resolution of 320 × 240 RGB565, the framebuffer size is 320 × 240 × 2 = 150 kB. As it is difficult to have an on-chip SRAM of such a size, an external SDRAM is used to store framebuffer(s) and other data of graphical software. Later sections of the application note discuss techniques to connect multiple FBs.

The sizes of FB for some common resolution and color settings are mentioned below:

- 320 × 240 @ 16 bpp: 150 kB
- 480 × 272 @ 16 bpp: 255 kB (uses a typical external SRAM chip)
- 640 × 480 @ 24 bpp: 1200 kB (24 bpp requires 32 bits per pixel to store)
- 1024 × 768 @ 16 bpp: 1536 kB

The LCD controller accesses FB as an array of four byte words. The words (pixel data) are stored either in little- endian or big-endian format; see Table 5.

Table 5. Pixel organization in framebuffer

Byte order	Little-endian bye, 16bpp	Big-endian, 16 bpp	TGB888, either endian
0	pixel 0	pixel 1	Pixel 0
1			
2	pixel 1	pixel 0	
3			not used

Old STN panels with 1/2/4 bpp, has endian format settings within one byte. The discussion is beyond the scope of this application note.

The LCD controller consists of a FIFO with a capacity of storing 16 words at a time. The size of each word is 64 bits. LCD FIFO helps smoothing the data transfer from FB to LCD. DMA only accesses FB when FIFO is drained below a programmable watermark level. Also, the 64-bit width of FIFO limits the FB address to be aligned to eight bytes i.e. the three Least Significant Bits (LSB) of FB address is always 0.

3.4.1 LCD registers to configure framebuffer

LCD controller supports up to two panels for STN, and one panel for TFT. The address of FB for each panel are given by the registers shown below:

Table 6. LCD registers to configure framebuffer

LCD_LPBASE.[31:3]	Address of FB for lower panel
LCD_UPBASE.[31:3]	Address of FB for upper panel
For TFT, set both registers to the same value - - - only one active FB. Also note that since LSB [2:0] is not considered, FB address must be multiples of 8.	
LCD_CTRL.[7:7] (LCDDUAL)	Always set to 0 for one TFT panel, does NOT support two TFT panels
LCD_CTRL.[9:9] (BEBO)	0 = Little endian bytes within a 32-bit word, 1 = big endian bytes within a 32-bit word
LCD_CTRL.[16:16] (WATERMARK)	LCD DMA (TX) FIFO watermark, 0 = 4 or more empty locations, 1 = 8 or more empty locations

3.4.2 Code snippet of configuring framebuffer

Usually we use at least RGB565 color depth and put framebuffer in external SDRAM with alignment to at least 8 bytes, and framebuffer is not required to initialize. We can define a 2D array with compiler extensions and corresponding linker configurations to guide tool chain to put framebuffer in SDRAM w/o initializing it, or simply define a pointer of framebuffer type and initialize it to the address we want to lay framebuffer at, and make sure framebuffer range is not visible to linker.

The following snippet shows the 2D array approach (KEIL). The “s_FB[ ][ ]” is the instance of framebuffer.

```
attribute  ((section("FB"), zero_init)) uint16_t s_FB[IMG_HEIGHT][IMG_WIDTH];
...
(in linker scatter file)
LR_Flash 0 512*1024 {
...
RW_FB 0xA0000000 UNINIT 1024*1024 {
*(FB)
}
…
}
```

The code specifies that s_FB is in FB section and the linker scatter file specifies FB section is placed in RW_FB region that starts at 0xA0000000, which is the base address of the SDRAM range.

The following snippet shows the pointer approach (used in MCUXpresso IDE).

```
typedef uint16_t s_FB_t[IMG_WIDTH];
s_FB_t *s_FB = (s_FB_t*) 0xA0000000;
```

Note : s_FB is a 1D array of pointers and is initialized to 0xA0000000, (starting address of the SDRAM range). The size of s_FB array is IMG_WIDTH and the format s_FB[row][col] is used to address a pixel.

3.5 LCD timings

As discussed in Section 3.4, the LCD controller refreshes the LCD at a fixed Frame-Per-Second (FPS) rate. Refreshing a new frame on the LCD is like printing a new page. If the LCD refresh rate is slowed down by thousand times, the contents are seen like being "printed" on a paper line by line with page margin settings shown in Fig 10.

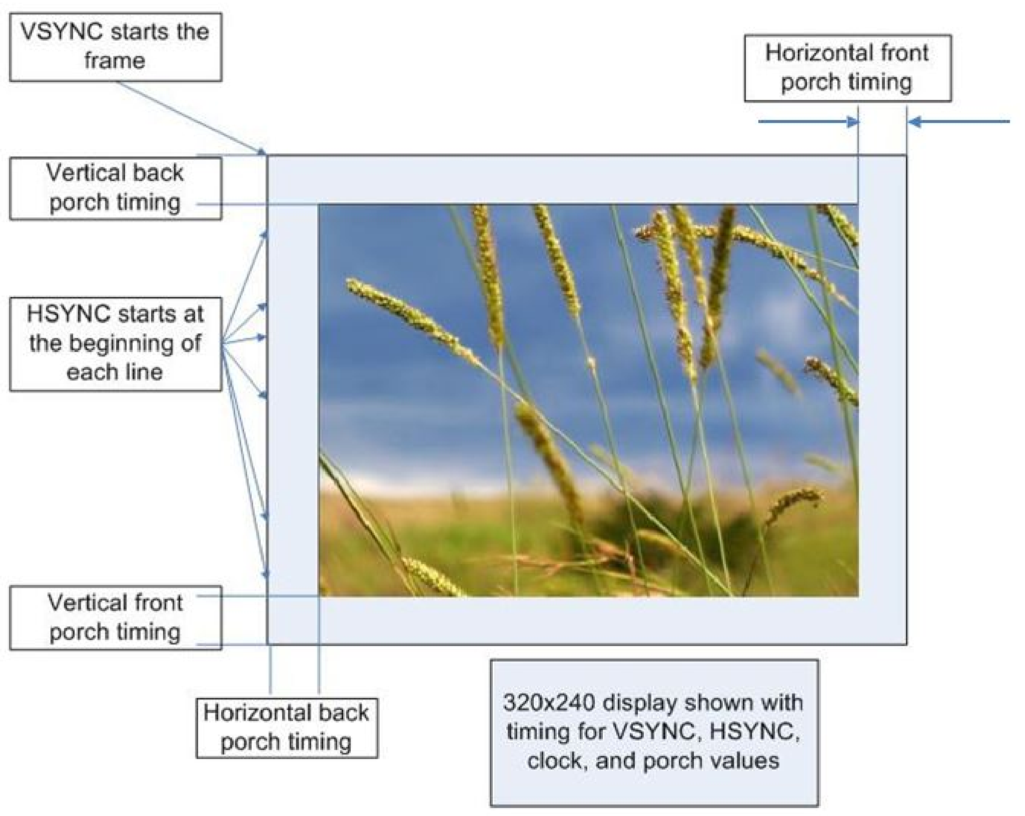

VSYNC starts the frame
Horizontal front porch timing
Vertical back porch timing
HSYNC starts at the beginning of each line
Vertical front porch timing
Horizontal back porch timing
320x240 display shown with timing for VSYNC, HSYNC, clock, and porch values

Fig 10. LCD timings

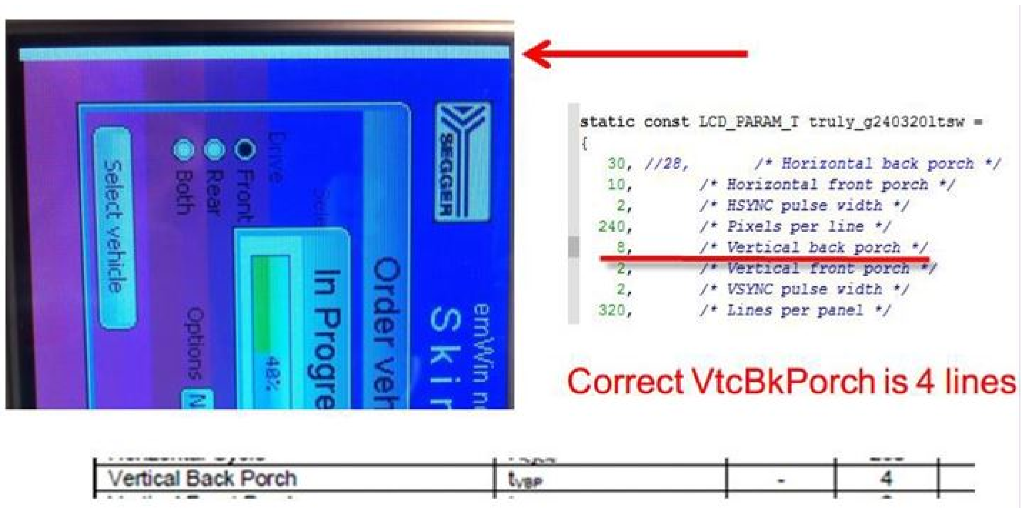
Fig 10 shows the parameters configured with the LCD controller timing registers and their visualization on a real panel. Fig 10 is a landscape display format. The LCD controller supports both landscape and portrait formats. For better understanding, assume that it is a picture on a paper. The blank areas on the edges are the page margins. In LCD terminology, these page margins are called porches. Horizontal porches are measured in pixel clocks while vertical porches are measured in lines. Though there is no image data in the porch regions, LCD pixel clock is still required.

VSYNC pulse starts a new frame, followed by some blank lines of Vertical Back Porch (VBP). Then the valid data lines start. After refreshing all valid data lines, few more blank lines of Vertical Front Porch (VFP) are present.

Every blank line and valid data line is started by a HSYNC pulse. It is followed by a blank segment as Horizontal Back Porch (HBP), then a segment with length of pixels per line, ended with another blank segment as Horizontal Front Porch (HFP). It is to be noted that the back porch comes before the front porch.

Besides the porches, after one line is refreshed (after front porch stage), some panels require a programmable "line end delay".

Many LCD panels accept a range of porch parameters, and can display images normally even if some porch settings are out of specification, such as the LCD panel used by LPCXpresso5460x board. But incorrect settings distort images for older panels. Fig 11 shows image on Truly G240320LTSW panel which wrongly sets vertical back porch to 8 while the correct value is 4, leading to 4 visible blank lines by LCD panel.



```
static const LCD_PARAM_T truly_g240320ltsw =
{
   30, //28,        /* Horizontal back porch */
   10,         /* Horizontal front porch */
    2,         /* HSYNC pulse width */
  240,         /* Pixels per line */
    8,         /* Vertical back porch */
    2,         /* Vertical front porch */
    2,         /* VSYNC pulse width */
  320,         /* Lines per panel */
```

Correct VtcBkPorch is 4 lines

Vertical Back Porch	t_{VBP}	-	4

.

Fig 11. An example of incorrect porch settings

3.5.1 LCD registers to configure timings

Besides the clock/line settings, timing configuration also includes the polarity of HSYNC and VSYNC pulses, output enable. Table 7 summarizes the configuration in LCD controller of the above timing parameters.

Table 7. LCD registers to configure timings

LCD_TIMH.[15:8] (HSW)	HSync pulse width, in pixel clocks – 1
LCD_TIMH.[23:16] (HFP)	Horizontal front porch (after image data of one line), in pixel clocks - 1
LCD_TIMH.[31:24] (HBP)	Horizontal back porch (before image data of one line), in pixel clocks - 1
LCD_TIMV[.15:8] (VSW)	VSync pulse width, in horizontal lines (rows) – 1
LCD_TIMV.[23:16] (VFP)	Vertical front porch (after image data of one line), in rows - 1
LCD_TIMV.[31:24] (VBP)	Vertical back porch (before image data of one line), in rows - 1
LCD_POL.[11:11] (IVS)	VSync is active high = 0/low = 1
LCD_POL.[12:12] (HIS)	HSync is active high = 0/low = 1
LCD_POL.[13:13] (IPC)	Pixel clock active edge is rising = 0/falling = 1 (active edge drives data to lines)
LCD_POL.[14:14] (IOE)	Output enable (LCD_AC pin) is active high =0/low = 1
LCD_POL.[6:0] (LED)	Line end delay, in pixel clicks - 1

3.5.2 Code snippets of configuring LCD timings

Relevant macros

```
#define LCD_HSW   2    // HSync width
#define LCD_HFP   8    // Horizontal front porch
#define LCD_HBP   43   // Horizontal back porch
#define LCD_VSW   10   // VSync width
#define LCD_VFP   4    // Vertical front porch
#define LCD_VBP   12   // Vertical back porch
#define LCD_POL_FLAGS    kLCDC_InvertVsyncPolarity | kLCDC_InvertHsyncPolarity
```

Configuration code

```
...
lcdc_config_t lcdConfig;
...
lcdConfig.hsw = LCD_HSW;  lcdConfig.hfp = LCD_HFP;  lcdConfig.hbp = LCD_HBP;
lcdConfig.vsw = LCD_VSW;  lcdConfig.vfp = LCD_VFP;  lcdConfig.vbp = LCD_VBP;
lcdConfig.polarityFlags = LCD_POL_FLAGS;
...
```

3.6 LCD power-up and power-down sequences

LCD panels require specific power-up and power-down sequences. The LCD controller enables or disables itself and controls power to the LCD panel. The power-up sequence consists of four steps:

1. When power is first applied, all LCD signals are held low
2. After power has stabilized, all signals except LCD_VD [23:0] and LCD_PWR are active
3. After above signals are stabilized, contrast voltage is applied to panel
4. If required, a timer can be used to generate a delay and then let the LCD controller to bring LCD_VD [23:0] and LCD_PWR active

Power-down sequence is symmetric with the power-up sequence. Fig 12 shows the sequences.

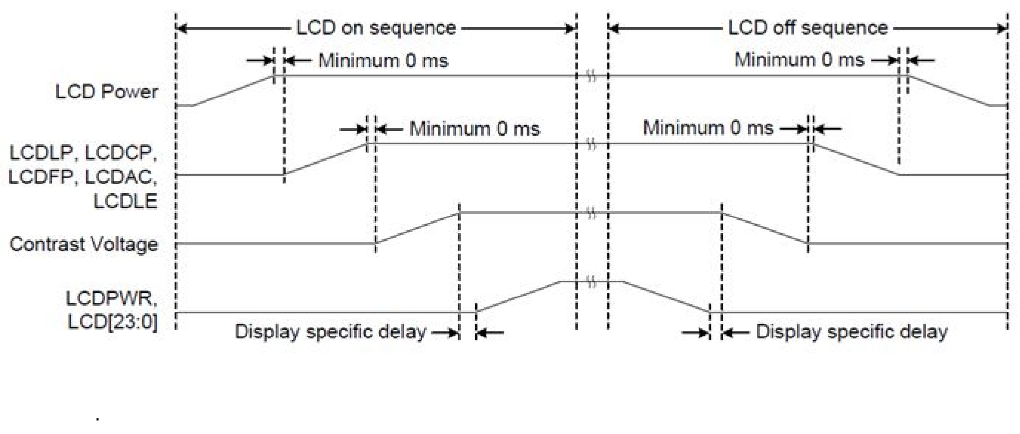


Fig 12. LCD power up and power down sequences

3.6.1 LCD registers for power sequence

Table 8. LCD registers for power sequence control

LCD_CTRL.[0:0] : (LCDEN)	0 =disable LCD signals (held low); 1 =enable LCD signals
LCD_CTRL.[11:11]: (LCDPWR)	0 = LCD panel is not powered and VD[23:0] are low; 1 = powered and VD[23:0] operated normally

3.6.2 Code snippets to bring up LCD

SDK provides the APIs for necessary operation.

```
...

LCDC_Start(LCD);

LCDC_PowerUp(LCD);
```

3.7 Flags and interrupts

LCD controller has status flags to represent its current working state and errors. These flags can be programmed to trigger interrupts.

The flags are:

- **FIFO underflow (bit 1)**: Set when either the upper or lower DMA FIFOs have been read accessed when empty. Helpful to diagnose memory bandwidth issues; see Section 4.2
- **Framebuffer next address base update (bit 2):** Used for multi-frame buffering. This bit indicates that the next FB address has been loaded. Next FB will be used for next LCD refresh cycle. Refer Section 4.1 for details about multi-frame buffering

- **Vertical compare (bit 3):** Set when specified progress of one LCD refresh cycle is reached. The progress can be selected from start of either VSYNC/back porch/active video/front porch
- **AHB master bus error (bit 4):** Set when LCD DMA encounters bus error response

3.7.1 LCD registers for status and interrupts

There are four registers that reflect raw flags, interrupt flags, interrupt mask, and interrupt clear. All these registers have the same bit field organization for the four flags.

Table 9. LCD registers for status and interrupts

LCD_INTRAW	Raw flags registers, show the instant value of the four status; can be used to request interrupt
LCD_INTMSK	Enable flag(s) in LCD_INTRAW to request interrupt, a one in a control bit means IRQ enabled
LCD_INTSTAT	Show the flags which are enabled to request interrupt after masking, once set, it is sticky until cleared by software
LCD_INTCLR	Write one(s) to clear corresponding sticky bits in LCD_INTSTAT

Flag bit organization is shown in Table 10.

Table 10. Flag bits organization in registers

LCD_INTXXX.[01:01]	FIFO underflow
LCD_INTXXX.[01:02]	Framebuffer next address base updated
LCD_INTXXX.[03:03]	Vertical compare
LCD_INTXXX.[04:04]	AHB master bus error
LCD_CTRL.[13:12] (LCDVCOMP)	Select LCD refresh progress for vertical compare: 0 = VSync, 1 = back porch, 2 =active video, 3 =front porch

The multi-framebuffering (discussed later) makes use of the framebuffer next address update interrupt or vertical compare interrupt to synchronize buffer switch with LCD refreshment.

```
…
LCDC_EnableInterrupts(LCD,kLCDC_BaseAddrUpdateInterrupt);
NVIC_EnableIRQ(LCD_IRQn);
…
void LCD_IRQHandler(void)
{
uint32_t intStatus = LCDC_GetEnabledInterruptsPendingStatus(LCD);
LCDC_ClearInterruptsStatus(LCD, intStatus);
if (intStatus & kLCDC_BaseAddrUpdateInterrupt)
{
// notify background code that new framebuffer is loaded, can now draw on  previous
framebuffer safely
}
}
```

3.8 Introduction to hardware cursor support

The LCD controller also has a hardware, two bits per pixel cursor overlay support, including its bitmap, color palette, position, and clipping (display part of cursor image). Details of the hardware cursor support is beyond the scope of this application note.

4. Advanced topics about LCD control

4.1 Multi-framebuffering technology

If one framebuffer is being refreshed by LCD controller and rendered by software at the same time, LCD shows the intermediate render result. It often leads to a flickering effect.

To avoid this, dual-frame buffering is helpful. When one framebuffer is being refreshed - printed to LCD panel, rendering is done on the other background framebuffer(s). With this method, the framebuffer being refreshed by LCD controller should be “read only” for software, so that the LCD panel never displays intermediate drawings.

To support multi-framebuffer technique, LCD controller has LCD_LPBASE and LCD_UPBASE register (for TFT LCD always set them to the save value). It specifies the address of NEXT framebuffer. Note that writing to this register does not affect LCD to continue to refresh current FB.

4.1.1 Dual-framebuffering

A basic practice is to use dual-frame buffering where graphics software renders one FB (background FB) while another FB (foreground FB) is refreshed to LCD panel. After the background FB is fully rendered, the registers LCD_LPBASE and LCD_UPBASE are updated to the address of this background FB, and switch the background FB to the other FB.

Note that at this moment, the current foreground FB becomes the new background FB, and it may still be used by LCD controller to refresh the LCD panel. If the graphics software renders it immediately, it will compete the bus bandwidth against LCD controller and the LCD may show tearing and/or flickers, which is unacceptable for a good user-experience.

To avoid this issue, the LCD controller has the next base address update interrupt (bit 2 of LCD_INTSTAT register). The LCD next base address update interrupt asserts when either the LCDUPBASE or LCDLPBASE values have been transferred to the LCDUPCURR or LCDLPCURR incrementors respectively. The transfer happens at every LCD VSYNC signal. Only after this IRQ, the new background FB (original foreground FB) is safe to render. The side effect is wasting time on synchronizing to beginning of LCD refreshment (VSYNC) which can lower frames-per-second (FPS) index.

4.1.2 Tri-framebuffering

Dual-framebuffering prevents flickering. However, S/W have to wait until foreground FB is completed refreshed then do rendering, this lead to downgrade of frame-per-second (FPS). Also, if system load changes in a large range, S/W do not have chance to do more rendering work when system load is lite (no free background FB to render). As it may not be possible to render new frame in time when system load is heavy, the FPS is not stable.

Tri-frame buffering adopts the basic concept of dual-frame buffering and helps improving the FPS index to be more stable. In this scheme, there are three FBs:

1. one foreground FB being refreshed by LCD controller
2. one background FB which is being rendered and
3. one background FB which is free or rendered. This FB ensures that if render rate is lower LCD refresh rate, S/W can immediately render on it, instead of waiting until current foreground FB is fully refreshed.

After initialization, software marks first FB as foreground FB and marks the remaining FBs as free and then enters main loop. At the start point of main loop, the software searches for free background FB:

- If there is one free background FB, use it for rendering. After rendering is completed, immediately update the LCD_LPBASE and LCD_UPBASE to oldest rendered FB, so LCD controller uses it for next LCD refresh cycle. Oldest FB maybe the just rendered one or not. In the latter case, the update uses the same FB as previous update.
- If there is no free background FB, then skip rendering.

Then, the S/W waits for the flag set by *next base address update* ISR. After ISR signals the IRQ flag, clear the flag and enter next iteration of main loop.

In parallel, in the IRQ handler of *next base address update* interrupt, if there is at least one rendered background FB, mark the previous foreground FB as the new free background FB. Also, signal the IRQ flag so that the background graphics software continues rendering; otherwise, do nothing (especially, do **not** signal IRQ flag).

4.2 Avoid LCD tearing

If LCD resolution is high and refresh rate is high, or bus masters such as CPU and DMA accesses the same RAM where framebuffer is located, they consume significant bus bandwidth. If it exceeds the available bandwidth, LCD DMA may not be able to fill image data to LCD in time. As a result, LCD display will look like as if the displayed image is torn. It is called the LCD tearing effect; see Fig 13.

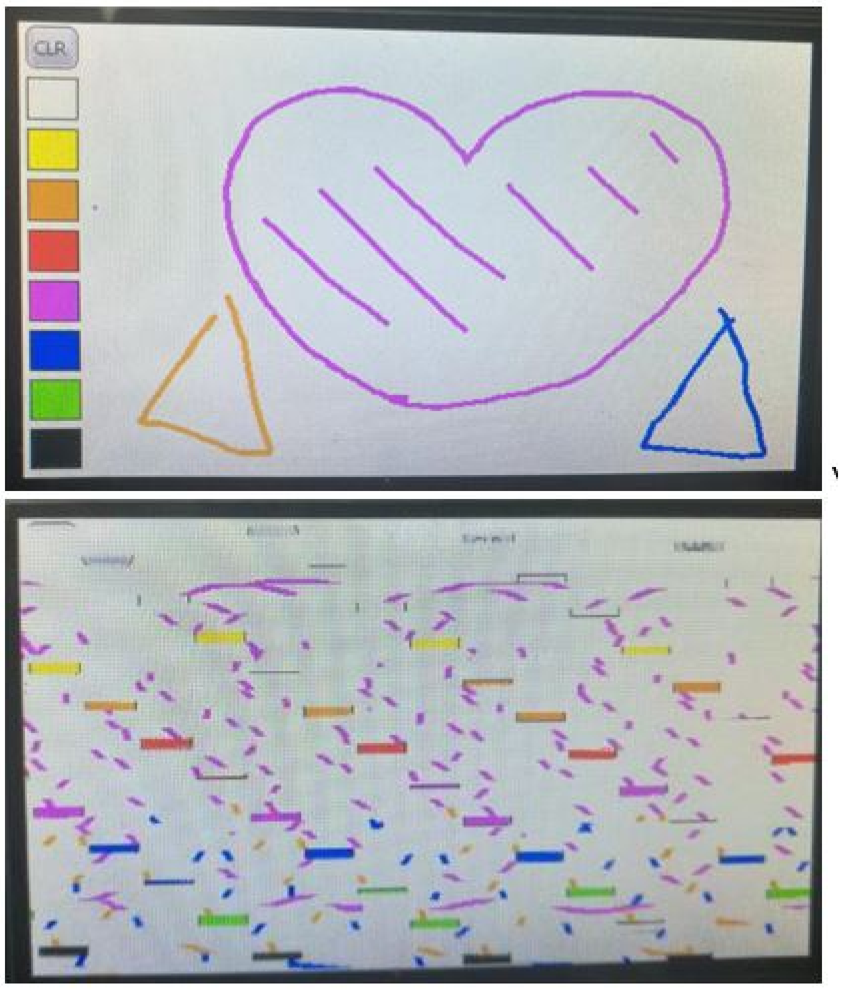


Fig 13. An example of LCD tearing

In Fig 13, upper image is normal while the lower image is LCD tearing. We got the effect by dividing EMC clock by nine or larger.

To detect potential LCD tearing, LCD controller provides the FIFO underflow interrupt (bit 1 of LCD_INTSTAT register) to monitor. If this IRQ happens often, then user should consider some methods to decrease bus bandwidth or prioritize LCD DMA as below approaches:

- Increase EMC bus clock
- Slow down the frame refresh rate, pixel clock if possible
- Some LPC MCUs (such as LPC17xx/40xx, LPC546xx) give option to configure AHB master priorities. Setting LCD_DMA priority higher than others can relieve or solve LCD tearing.
- However, this may cause FB rendering to slow down
- Use 32-bit SDRAM rather than 16-bit SDRAM

NXP provides an LCD resource requirements calculator in excel to estimate the risk of LCD tearing. It supports framebuffer in external SRAM/SDRAM and TFT LCD. If the parameters of external RAM and LCD are entered, the calculator estimates the framebuffer size, LCD data rate, and bus bandwidth requirements. The file is available in the package with this AN. An example of the LCD panel on LPC5460x board is calculated as follows:

Table 11. LCD resource requirements calculator usage example

Bus bandwidth on various LCD resolutions and colors depths at various refresh rate	
EMC bus clock (MHz)	96
Dynamic external memory configuration	
Bus width	16
Precharge command period, tRP	3
RAS latency (active to read/write delay), RAS (tRCD)	3
CAS latency, CAS	3
LCD parameters	
Horizontal (pixels)	480
Vertical (pixels)	272
Horizontal back porch (pixel clocks)	32
Horizontal back porch (pixel clocks)	8
Vertical front porch (lines)	12
Vertical back porch (lines)	4
Pixel clock rate (MHz)	9
Color depth (bpp)	16
Results	
Refresh rate (Hz)	60.1
Frame buffet (KB)	255
LCD data rate (Mpixels/s)	7.8
LCD data rate (Mwords/s)	3.9
LCD data rate (Mbutsts/s)	1.0
Dynamic external memory burst – burst (clocks)	22
Bus bandwidth required (%)	22.5 %

Note: The refresh rate is calculated by REFRESH_RATE (Hz) = pixel_clock_rate / [(rows + vertical_front_porch + vertical_back_porch) * (pixel_clocks_per_data_line + horizontal_front_porch + horizontal_back_porch))].

The above example shows on 96 MHz 16-bit slow SDRAM, an LCD with 480 × 272 × 16 bpp@9 MHz pixel clock requires about 22.5% bandwidth to refresh. An interesting thing is if SDRAM is 32-bit, then bandwidth requirement is about 18.4%, far more than half.

The direct factor for LCD tearing risk is the bus bandwidth needed by LCD. Never make it more than 100%. For a relatively safe system, better no more than 60%. If UI has rich animations, it should be even lower.

5. Example of configuring LCD controller and bring up LCD panel in SDK

SDK provides easy-to-use APIs to setup LCD controller and power up LCD.

To configure LCD controller, user should provide LCD panel parameters in a configure structure and pass to LCDC_Init( ) function, then start LCD controller with LCDC_Start( ) and finally power up LCD with LCDC_PowerUp.

As an example, for the LCD panel used in LPCXpresso54608 board, its part number is "RK043FN02H-CT", from its datasheet, we find below parameters, which are critical to correctly configure LCD controller.

Table 12. An example of LCD timing parameters of LCD panel "RK043FN02H-CT"

Item		**Symbol**	**Min.**	**Typ**	**Max.**	**Unit**	
DCLK frequency		Fclk	5	9	12	MHz	
DCLK period		Tclk	83	110	200	Ns	
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	355	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	

According to the parameters given in datasheet, we define the configuration values relating to LCD panel as follows:

```
#define LCD_PANEL_CLK   12000000 // pixel clock
#define LCD_PPL    480 // pixel per line
#define LCD_HSW    2   // HSync width
#define LCD_HFP    8   // Horizontal front porch
#define LCD_HBP    43  // Horizontal back porch
#define LCD_LPP    272 // Line per panel
#define LCD_VSW    10  // VSync width
#define LCD_VFP    4   // Vertical front porch
#define LCD_VBP    12  // Vertical back porch
#define LCD_POL_FLAGS   kLCDC_InvertVsyncPolarity | kLCDC_InvertHsyncPolarity
```

The following below code snippet (clipped from main( )) shows the necessary steps to bring up LCD.

```
lcdc_config_t lcdConfig;

BOARD_InitSDRAM(); // SDRAM contains framebuffer
CLOCK_AttachClk(kMCLK_to_LCD_CLK);  // Attach main clock to LCD controller
CLOCK_SetClkDiv(kCLOCK_DivLcdClk, 1, true);
// >>> configure LCD controller and initialize
LCDC_GetDefaultConfig(&lcdConfig);
// >>> configure LCD panel related parameters
lcdConfig.panelClock_Hz = LCD_PANEL_CLK;
lcdConfig.ppl = LCD_PPL;
lcdConfig.hsw = LCD_HSW;
lcdConfig.hfp = LCD_HFP;
lcdConfig.hbp = LCD_HBP;
lcdConfig.lpp = LCD_LPP;
lcdConfig.vsw = LCD_VSW;
lcdConfig.vfp = LCD_VFP;
lcdConfig.vbp = LCD_VBP;
lcdConfig.polarityFlags = LCD_POL_FLAGS;
// <<<
lcdConfig.upperPanelAddr = (uint32_t)s_FBs[!s_actFBNdx][0]; // specify FB addr
lcdConfig.bpp = kLCDC_16BPP565; // Use RGB565 color format
lcdConfig.display = kLCDC_DisplayTFT; // Use TFT type of LCD
lcdConfig.swapRedBlue = false;  // Do not swap red and blue (BGR)
LCDC_Init(LCD, &lcdConfig, LCD_INPUT_CLK_FREQ);
// <<<
// >>> Enable LCD "BaseAddrUpdateInterrupt" to safely switch FBs for dual-FB
LCDC_EnableInterrupts(LCD, kLCDC_BaseAddrUpdateInterrupt);
NVIC_EnableIRQ(LCD_IRQn);
// <<<
LCDC_Start(LCD);
LCDC_PowerUp(LCD);
```

The above code at first initializes SDRAM that will contain framebuffers. It then sets up LCD controller clock, configures LCD controller, enables LCD controller IRQ for FB switch, finally starts and powers up.

Note: For LCD controller configuration, some parameters come from LCD panel, while others, such as framebuffer address, color format, depend on application design.

The above code snippet is collected from the LCD TFT dual-FB example (lcdc_2_tft16bpp_2fb). The project of this example can be found in the companion package. The hands-on PDF file shows more details about how to do this hands-on.

6. Introduction to embedded GUIs

6.1 Why embedded GUI

Graphical LCD enabled applications often need a well-designed graphics user interface. Most of the times, it is rather cumbersome as they need to implement many features such as:

- Model LCD drivers and framebuffer management
- Basic graphics primitives such as drawing lines, polygons, circles, curves, and filling shapes
- Drawing text: manage fonts, character encoding, text formatting, etc
- Drawing images
- Window and widget management for richer UI design and better user experience
- Mouse and/or touch screen support
- A set of PC utilities to create and manage resources of fonts, images, UI design, etc

Recently, smart phones and wearable devices have become popular. Their rich UI has increased user-demand for UI experience for a vast range of embedded systems.

To address above common requirements, there are specialized graphics middleware to use which is called as GUI. These middlewares take responsibilities of fundamental graphical and window management and provide a rich set of PC utilities. Users can design their UI with PC utilities and call APIs and callbacks to implement application level UI logic, which saves lots of development time. Some frequently used GUIs suitable for LPC MCUs are discussed below.

Free options: SWIM, emWin

Paid options: TouchGFX, Permission UI

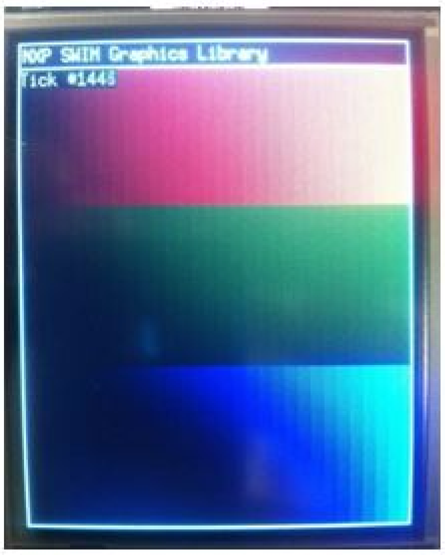
6.2 Basic Graphics Library - SWIM

SWIM is a free basic graphic library from NXP. It supports simple graphic functions such as boxes, lines, circles and some basic ASCII font options with 6 × 7, 6 × 13, and 8 × 8 as well as bitmap images with scaling support.

The SWIM graphics library supports varying color depths but can only be statically compiled for one color depth that must be defined before SWIM is compiled.

SWIM supports any frame buffer that has 8-bit, 16-bit, or 32-bit addressable pixel color data.

NXP provides an application note AN10815 that illustrates how to configure SWIM for the different third-party hardware available.

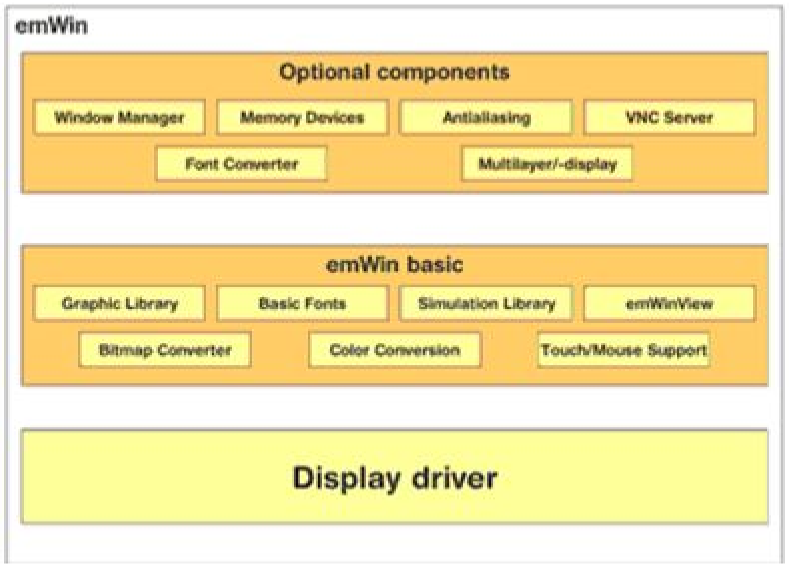


.

Fig 14. Snapshot of “SWIM” library

6.3 emWin

emWin is an old but proven and reliable embedded GUI middleware from Segger. emWin is hardware-friendly. It needs only few kB of flash and <= 1kB of RAM to run its core function and supports almost all kinds of LCD panels/modules. For feature rich applications, emWin also provides comprehensive GUI features including some modern elements to make UI experience close to smart phones.



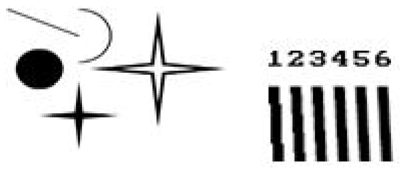
.

Fig 15. Block diagram of emWin

Besides core GUI library functions, emWin supports a vast range of some advanced features. Following are some of the features:

Memory devices: Memory device contexts allow creation of a section to output to the display in the memory, ready to render to framebuffer(s) in the later, allowing flicker free updates even with slow CPUs or slow displays.

Antialiasing: It is used for smoothening of lines and curves. It reduces the jagged, stair-step appearance of any line that is not exactly horizontal or vertical. emWin supports different antialiasing qualities, antialiased fonts and high-resolution coordinates.



.

Fig 16. Antialiasing examples in emWin

Windows and widgets: The window manager supplies a set of routines which allows you to easily create, move, resize, and otherwise manipulate any number of windows. It also provides low-level support by managing the layering of windows on the display and by alerting your application to display changes that affect its windows. Based on window support, emWin also provides predefined widgets as elements of a frame window, with skinning support to improve user experience. On the other hand, emWin provides templates of common dialogs such as file, calendar, color, and message box.

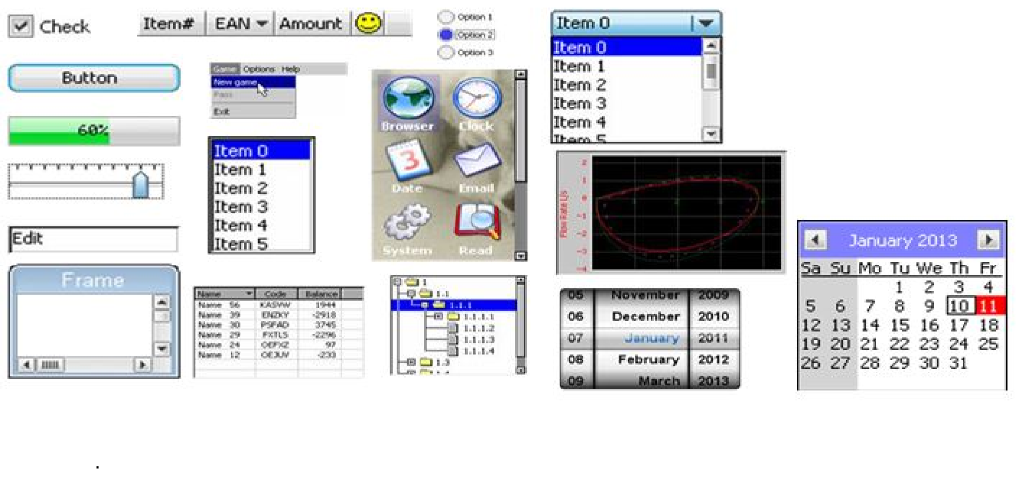


Fig 17. Widgets gallery of emWin

PC utilities: emWin provides a rich set of PC utilities, including bitmap converter, simulator, text converter, font converter, GUI builder.

NXP® Semiconductors now offers emWin in library form for free commercial use with NXP microcontrollers. The software bundle offered by NXP includes the emWin Color basic package, the Window Manager/Widgets module including the GUI Builder, the Memory Devices module for flicker-free animation, the Antialiasing module for smooth display of curves, lines and fonts, the Font Converter and the VNC (Virtual Network Computing) Server.

Segger provides a comprehensive user manual of emWin. NXP also provides application notes on the usage of emWin:

AN11244: emWin startup guide

AN11218: emWin Porting guide

6.4 TouchGFX

TouchGFX is a relatively new GUI. User can create modern and beautiful UI like those found on smart phones. TouchGFX is **not** free for commercial use.

TouchGFX is an excellent software framework that unlocks the graphical user interface (GUI) performance of low-resource hardware. The revolutionizing technology breaks existing restraints as it lets users create sophisticated GUIs that fully live up to smart phone standards of today at a fraction of the cost. By using TouchGFX, embedded product gets outstanding graphics and smooth animations with minimal resource and power consumption. It is a high-end product with a low cost per unit and a long battery life.

TouchGFX provides a very powerful PC designer studio TouchGFX Designer. It is an easy-to-use GUI builder that supports the development of embedded GUIs based on TouchGFX

7. LCD hands-on examples

This application note includes four LCD hands-on examples. They all use LPCXpresso54608 board (OM10392) and no special setting is required. To see the result, just open the relative projects, build, download and run. To practice more, also refer to the included hands-on guide "HOT_LCD_AN.pdf".

Example 1: Basic LCD drawing. This example uses 16 bpp mode. It also initializes SDRAM and locates the single FB in SDRAM. It draws eight rotating color stripes on LCD screen.

Example 2: Dual-framebuffering. Based on example 1 but allocates two FBs. Repeating drawings in main loop: first clear the screen to black, then draw color stripes. Use SysTick timer to limit draw rate.

If “SW5” button is not pressed, then use one FB to draw.

If “SW5” button is pressed, then waits for *base address update* IRQ, then draws on backup FB (the previous active FB). After drawing, set the next active FB to this FB.

Example 3: Palette mode. This example uses on-chip RAM as FB and 2 bpp mode, one byte contains four pixels. It defines used colors in palette. In main loop, it draws moving rectangle periodically. Every period is synchronized to a new LCD base address update IRQ. The example implements a rectangle draw and fill routine with 2 bpp mode.

Example 4: Hardware cursor. This example uses the same settings as example 3 and configures hardware cursor. Then in main loop, moves cursor periodically. Every period is synchronized to a new LCD vertical back porch IRQ. Users can see that there is a cursor moving smoothly and when it reaches an edge (either left, top, right, bottom), the cursor bounces. The inner color of cursor is the complementary to the background color.

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

8.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

8.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP Semiconductors N.V.

9. List of figures

10. List of tables

11. Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

© NXP Semiconductors N.V. 2017. All rights reserved.

For more information, visit: http://www.nxp.com

Date of release: 21 August 2017
Document identifier: AN12027