# How to use Chrom-ART Accelerator to refresh an LCD-TFT display on STM32 MCUs

## Introduction

This application note highlights how to refresh an LCD-TFT display via the FMC (flexible memory controller) interface using the Chrom-ART Accelerator on STM32 microcontrollers listed in the table below.

This Chrom-ART Accelerator (DMA2D) is a specialized DMA dedicated to image manipulation.

The DMA2D can perform the following operations:

- Fill a part or the whole of a destination image with a specific color.
- Copy a part or the whole of a source image into a part or the whole of a destination image with a pixel format conversion.
- Blend a part and/or two complete source images with a different pixel format and copying the result into a part or the whole of a destination image with a different color format.

On STM32 microcontrollers, the FMC is used to access the LCD-TFT display through a parallel interface.

This application note explains:

- how to connect the LCD-TFT display to the FMC interface
- how to configure the DMA2D for the LCD-TFT display refresh
- how to use the DMA2D byte reordering features to directly drive Intel 8080 displays

To fully benefit from this document, the user can refer to the product reference manual to get familiar with the STM32 Chrom-ART Accelerator (DMA2D).

Table 1. **Applicable products**

| Type | Applicable products |
|---|---|
| Microcontrollers | STM32L4x6 line |
| | STM32L4R5/S5 line, STM32L4R7/S7 line, STM32L4R9/S9 line |
| | STM32U5 series |
| | STM32H7 series |

# 1 General information

This application note applies to STM32 microcontrollers Arm®-based devices.

*Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

### Reference documents

The following documents are available on *www.st.com*.

- Reference manual *STM32L4x6 advanced Arm®-based 32-bit MCUs* (RM0351)
- Reference manual *STM32L4Rxxx/L4Sxxx advanced Arm®-based 32-bit MCUs* (RM0432)
- Reference manual *STM32U5 series Arm ®-based 32-bit MCUs* (RM0456)
- Reference manual *STM32H7Rx/Sx Arm ®-based 32-bit MCUs* (RM0477)
- User manual *Discovery kit with STM32L496AG MCU* (UM2160)
- Embedded software for the STM32L4 Series and STM32L4 Series+ (STM32CubeL4)

# 2 Chrom-ART Accelerator (DMA2D) application use-case overview

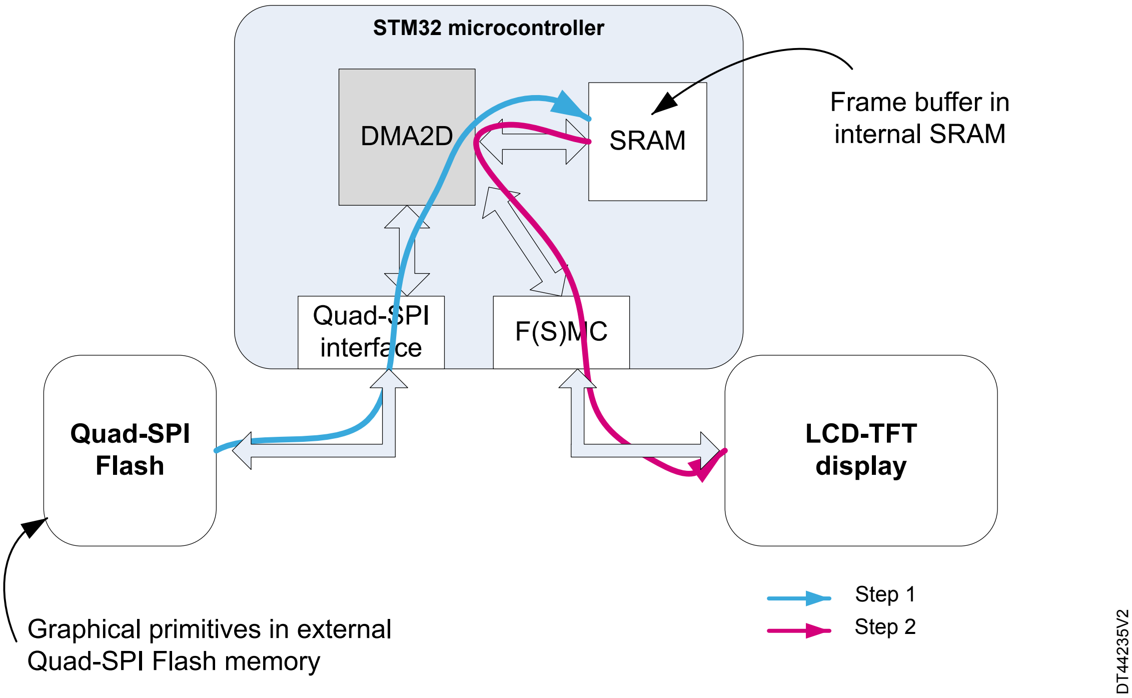
A typical application displaying an image into an LCD-TFT display is divided in two steps.

- Step 1: creation of the frame buffer content:
  - The frame buffer is built by composing graphical primitives like icons, pictures and fonts.
  - This operation is done by the CPU running a graphical library software.
  - It can be accelerated by a dedicated hardware used with the CPU through the graphical library (Chrom-ART Accelerator (DMA2D)).
  - The more often the frame buffer is updated, the more fluid are the animations.
- Step 2: display of the frame buffer onto the LCD-TFT display:
  - The frame buffer is transferred to the display through a dedicated hardware interface.
  - The transfer can be done using the CPU, the system DMA or using the Chrom-ART Accelerator (DMA2D).

In a typical display application example using the STM32 microcontrollers, the F(S)MC is used as the hardware interface to the LCD-TFT display, the graphical primitives like pictures, icons or fonts are stored in the external Quad-SPI Flash memory and the frame buffer is stored in the internal SRAM. The transfer of the frame buffer to the LCD-TFT display can also be managed by the Chrom-ART Accelerator (DMA2D), hence not using the CPU or the DMA resources.

This is showed in the figure below.

**Figure 1. Display application typical use case**




The Chrom-ART Accelerator (DMA2D) can update the whole image on the display (full refresh) or only a part of it (partial refresh).

The configuration of the Chrom-ART Accelerator (DMA2D) (full or partial refresh) is done by programming specific registers through the high level HAL library function as shown in Section 4.

# 3 LCD-TFT display on F(S)MC

## 3.1 Hardware interface description

Signals in the table below are used to connect the F(S)MC to the LCD-TFT display.

**Table 2. F(S)MC signals**

| Signal name | F(S)MC I/O | Function |
|---|---|---|
| A[25:0] | O | Address bus |
| D[15:0] | I/O | Bidirectional data bus |
| NE[x] | O | Chip select, x=1..4 |
| NOE | O | Output enable |
| NWE | O | Write enable |

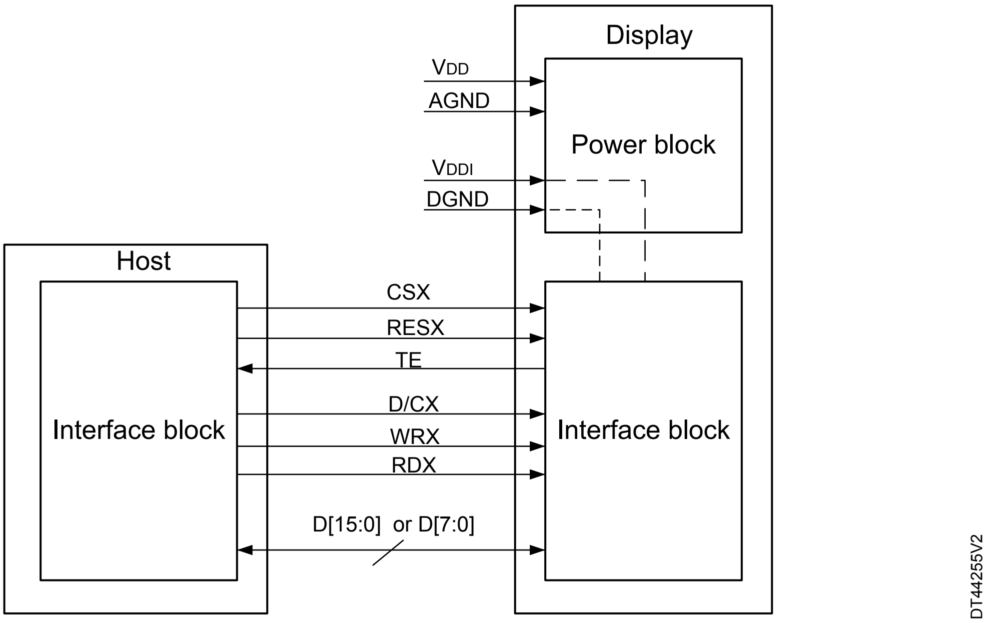
In the table below the signal names are provided according the Type B display bus interface (DBI) as described in the MIPI® Alliance standard for display bus interface.

**Table 3. LCD-TFT signals**

| Signal name | LCD-TFT I/O | Function |
|---|---|---|
| D/CX | I | Data/command control signal |
| D[15:0] | I/O | Bidirectional information signals bus |
| CSX | I | Chip select control signal |
| RDX | I | Read control signal |
| WRX | I | Write control signal |
| TE | O | Tearing effect |
| RESX | I | Reset |

A typical connection is showed the figure below.

**Figure 2. Display bus interface specification**




## 3.2 Display command set (DCS) software interface

The LCD-TFT displays can be controlled through the physical interface (here the F(S)MC bus) using software commands according to the display command set (DCS), as defined in the MIPI Alliance specification for DCS.

The DCS commands are used to configure the display module and to transfer the frame buffer to the display.

## 3.3 Controlling the D/CX signal with STM32 microcontrollers

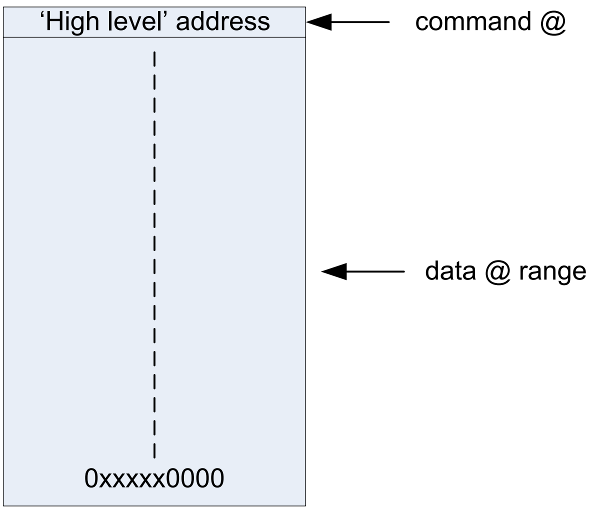
The 'Data/Command control' (D/CX) signal of the DBI protocol is used to distinguish the commands (when D/CX = 0) from the data (when D/CX = 1) transfers.

There are two ways to control the D/CX signal:

- By using a dedicated GPIO:
  1. Set the D/CX signal in "command mode" (setting the GPIO connected to the D/CX signal to 0 by software).
  2. Send the command.
  3. Set the D/CX signal in "data mode" (setting the GPIO connected to the D/CX signal to 1 by software).
  4. Send the data (frame buffer).
- By using an address bit of the F(S)MC address bus:
  1. Reserve a “low level” address in the memory map for the command transfer.
  2. Reserve the higher memory map range for the data transfer.

When using the DMA2D to access the LCD-TFT display on F(S)MC interface, remember that even if the LCD-TFT display target is at a fixed address, the DMA2D increments the address bus of the transmitted data at each access (like a memory-to-memory access). Thus the F(S)MC address bus is incremented to cover the full data range address in the memory map.

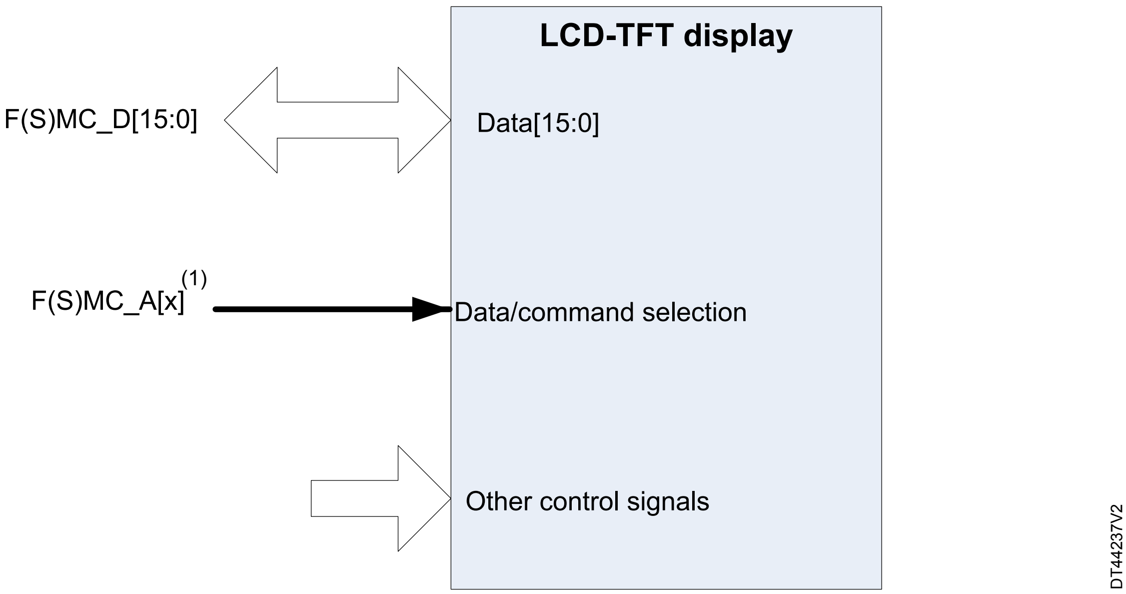
**Figure 3. Memory map for LCD-TFT display access**




The second option "an address bit of the F(S)MC address bus" makes the software simpler than the first option with a dedicated GPIO, but it requires using the “high level” address to control the ‘data or command select signal’.

The user cannot use for example the F(S)MC address LSB bit (F(S)MC_A0) to control the ‘data or command select signal’. The user must use a “high enough” F(S)MC address bit in order to keep for this bit the same value during the whole image frame buffer transfer.

**Figure 4. Automatic control of LCD-TFT display data/command by F(S)MC interface**




1. *‘x’ as high as possible according to Table 4.*

For example, if the image buffer size is 240 x 240 pixels and the transfer is done using 16 bits in RGB565 mode (one pixel transferred per access to LCD), the number of accesses are 240 x 240 = 57600 accesses and the F(S)MC address increments from 0x0000 0000 to 0x0000 E0FF.

Thus the first address bit that does not change during the transfer is the bit 16. In this specific case the F(S)MC_A16 or a higher address bit can be used.

The table below shows the minimum F(S)MC address bit that can be used depending on some image size.

**Table 4. Minimum F(S)MC address bit to use depending on image size (16-bit RGB565 access)**

| Image size | Number of pixels | Number of accesses | Max address | Min usable FSMC address bit |
|---|---|---|---|---|
| VGA | 640 x 480 | 307200 | 0x4AFFF | F(S)MC_A19 |
| HVGA | 480 x 320 | 153600 | 0x257FF | F(S)MC_A18 |
| QVGA | 320 x 240 | 76800 | 0x12BFF | F(S)MC_A17 |
| - | 240 x 240 | 57600 | 0x0E0FF | F(S)MC_A16 |

# 4 Chrom-ART Accelerator (DMA2D) configuration in STM32CubeL4

## 4.1 LCD partial refresh

An example configuring the DMA2D for an LCD partial refresh is provided in the following folder:

`STM32Cube_FW_L4\Firmware\Projects\STM32L496G-Discovery\Examples\DMA2D\DMA2D_MemToMemWithLCD`.

The code used to configure and start the DMA2D is shown below.

```
/* Configure LCD before image display: set first pixel position and image
size */
/* the position of the partial refreshed window is defined here. A rectangle
in the middle of the screen */
LCD_ImagePreparation((ST7789H2_LCD_PIXEL_WIDTH - LAYER_SIZE_X)/2,
(ST7789H2_LCD_PIXEL_HEIGHT - LAYER_SIZE_Y)/2, LAYER_SIZE_X, LAYER_SIZE_Y);
/*##-2- DMA2D configuration
################################################*/
DMA2D_Config();
/*##-3- Start DMA2D transfer
###############################################*/
hal_status = HAL_DMA2D_Start_IT(&Dma2dHandle,
(uint32_t)&RGB565_240x160, /* Source buffer in format RGB565 and size
240x160 */
(uint32_t)&(LCD_ADDR->REG), /* LCD data address */
1, LAYER_SIZE_Y * LAYER_SIZE_X); /* number of pixel to transfer */
OnError_Handler(hal_status != HAL_OK);
…
…
…
/**
* @brief DMA2D configuration.
* @note This function configure the DMA2D peripheral :
* 1) Configure the transfer mode : memory to memory
* 2) Configure the output color mode as RGB565
* 3) Configure the transfer from FLASH to SRAM
* 4) Configure the data size : 240x160 (pixels)
* @retval
* None
*/
static void DMA2D_Config(void)
{
HAL_StatusTypeDef hal_status = HAL_OK;
/* Configure the DMA2D Mode, color Mode and output offset */
Dma2dHandle.Init.Mode = DMA2D_M2M; /* DMA2D Mode memory to memory */
Chrom-ART Accelerator™ (DMA2D) configuration in STM32CubeL4 AN4943
12/22 DocID029937 Rev 2
Dma2dHandle.Init.ColorMode = DMA2D_OUTPUT_RGB565; /* Output color mode
is RGB565: 16 bpp */
Dma2dHandle.Init.OutputOffset = 0x0; /* No offset in output */
Dma2dHandle.Init.RedBlueSwap = DMA2D_RB_REGULAR; /* No R&B swap for
the output image */
Dma2dHandle.Init.AlphaInverted = DMA2D_REGULAR_ALPHA; /* No alpha
inversion for the output image */
```

```
/* DMA2D Callbacks configuration */
Dma2dHandle.XferCpltCallback = TransferComplete;
Dma2dHandle.XferErrorCallback = TransferError;
/* Foreground configuration: Layer 1 */
Dma2dHandle.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
Dma2dHandle.LayerCfg[1].InputAlpha = 0xFF; /* Fully opaque */
Dma2dHandle.LayerCfg[1].InputColorMode = DMA2D_INPUT_RGB565; /* Foreground
layer format is RGB565 : 16 bpp */
Dma2dHandle.LayerCfg[1].InputOffset = 0x0; /* No offset in input */
Dma2dHandle.LayerCfg[1].RedBlueSwap = DMA2D_RB_REGULAR; /* No R&B
swap for the input foreground image */
Dma2dHandle.LayerCfg[1].AlphaInverted = DMA2D_REGULAR_ALPHA; /* No alpha
inversion for the input foreground image */
```

```
Dma2dHandle.Instance = DMA2D;
/* DMA2D initialization */
hal_status = HAL_DMA2D_Init(&Dma2dHandle);
OnError_Handler(hal_status != HAL_OK);
hal_status = HAL_DMA2D_ConfigLayer(&Dma2dHandle, 1);
OnError_Handler(hal_status != HAL_OK);
}
```

A full refresh is of course done in the same way but initializing the LCD first pixel at (0, 0) and the image size to the LCD size.

```
LCD_ImagePreparation(0, 0, ST7789H2_LCD_PIXEL_WIDTH,
ST7789H2_LCD_PIXEL_HEIGHT);
```

Changing the number of pixels to be transferred in the DMA2D start command:

```
hal_status = HAL_DMA2D_Start_IT(&Dma2dHandle,
(uint32_t)&RGB565_240x240, /* Source buffer in format RGB565 and size
240x240 */
(uint32_t)&(LCD_ADDR->REG), /* LCD data address */
1, ST7789H2_LCD_PIXEL_HEIGHT * ST7789H2_LCD_PIXEL_WIDTH); /* number of
pixel to transfer */
OnError_Handler(hal_status != HAL_OK);
```

# 5 New DMA2D features to support Intel 8080 displays

On the STM32 microcontrollers, the pixel data are stored in the frame buffer memory in little-endian format. This means that the least significant byte is stored at the lowest address and the most significant byte is stored at the highest address.

For example: in case of the RGB888 pixel format, the blue component is stored at address 0 while the red component is stored at address 2.

When the pixel data are transmitted to the LCD display via the F(S)MC, it starts with the least significant byte first, which is the blue component in this example.

This creates a mismatch with some Intel 8080 LCD display color coding which requires the most significant byte to be transmitted first (red component in case of the RGB888 pixel format).

This mismatch requires extra byte reordering steps to get the right byte order before transmitting the pixel data through the F(S)MC.

The new DMA2D byte reordering features allow the user to reorder the data in the DMA2D output FIFO, enabling to directly drive the LCD displays from a frame buffer with a classic RGB order without any extra software manipulation.

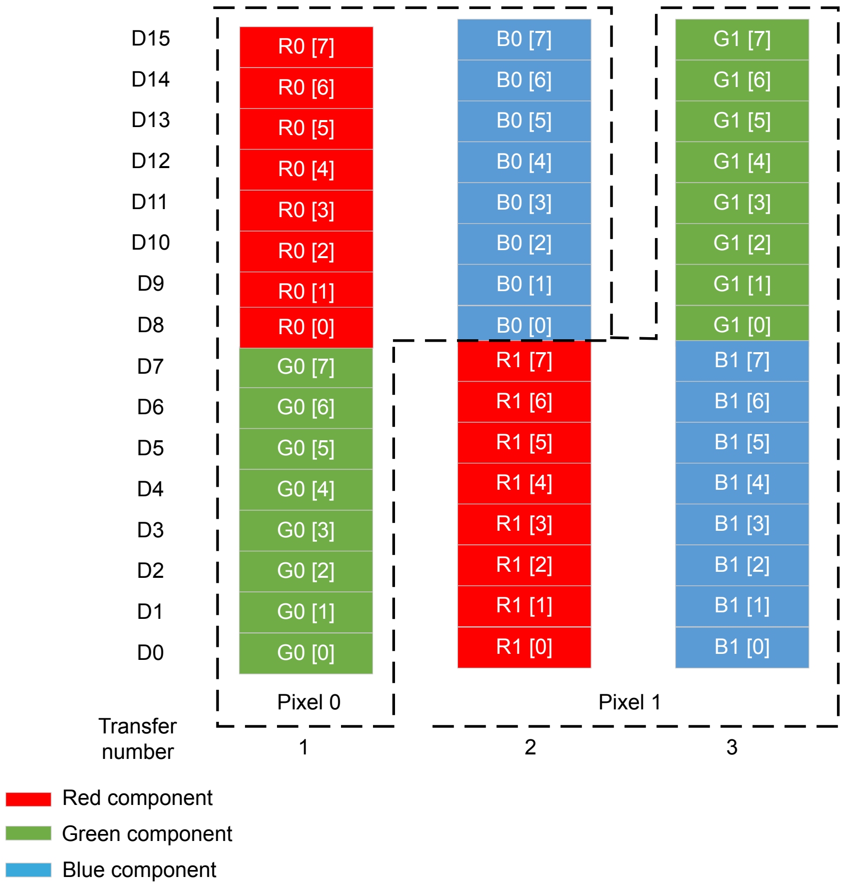
## 5.1 Intel 8080 interface color coding

The Intel 8080 is a common interface standard for the LCD displays. It is a parallel bus interface supporting 8, 9, 16 and 18-bit bus.

This section shows the Intel 8080 display color coding that creates a mismatch with a classic RGB order in the STM32 memory. Various cases are detailed below:

- 24 bpp (16.7M colors) and 18 bpp (262k colors) over 16-bit interface
  The figure below shows the color coding for transmitting 24 bpp data over a 16-bit bus interface on Intel 8080 displays.

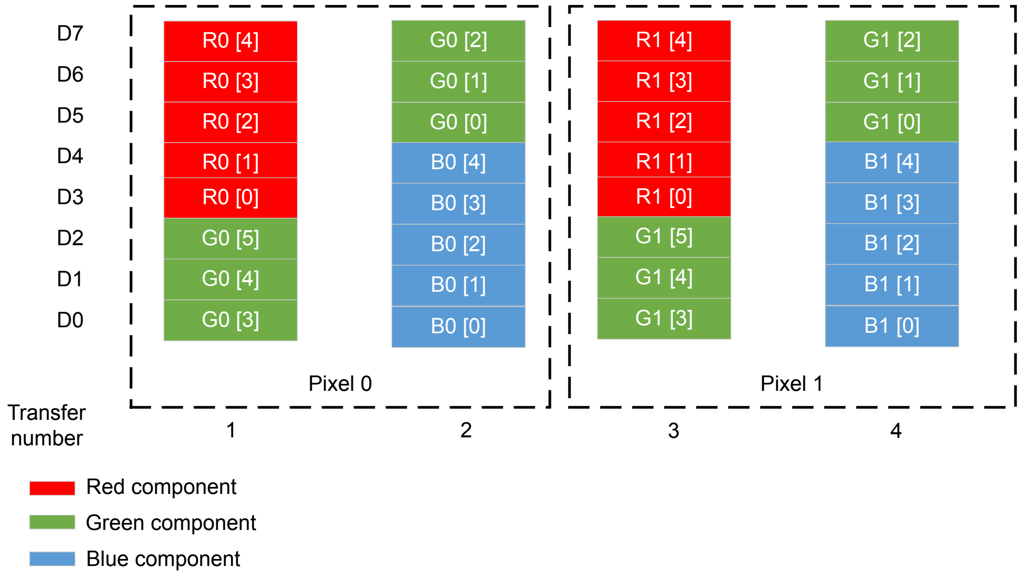
**Figure 5. 24 bpp over 16-bit interface color coding**




*Note: The 18 bpp displays have the same color coding except that in case of 18 bpp, R/G/B[6:0] are placed in the most significant bits of the bus and the data lines D9, D8, D1 and D0 are ignored.*

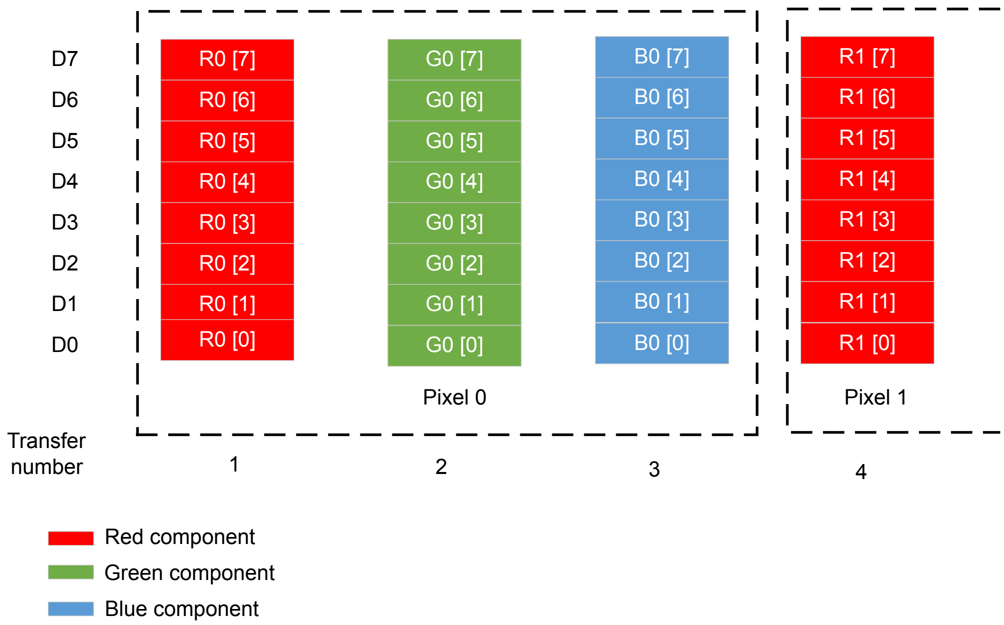
- 16 bpp (64k colors) over 8-bit interface
  The figure below shows the pixel color coding for 16 bpp displays over an 8-bit bus interface.

**Figure 6. 16 bpp over 8-bit interface color coding**




- 24 bpp (16.7M colors) and 18 bpp (262k colors) over 8-bit interface
  The figure below shows the pixel color coding for 24 bpp over an 8-bit bus interface.

**Figure 7. 24 bpp over 8-bit interface color coding**




*Note: The 18 bpp displays have the same color coding except that in case of 18 bpp, R/G/B[6:0] are placed in the most significant bits of the bus and the data lines D9, D8, D1 and D0 are ignored.*

## 5.2 DMA2D reordering features

The DMA2D output FIFO bytes can be reordered to support the display frame buffer update through a parallel interface (F(S)MC) directly from the DMA2D. The user can do combination of reordering operations to get the right byte endianness aligned with the display color coding.

### 5.2.1 Red and blue swap

The red and blue components can be swapped by setting the RBS bit in DMA2D_OPFCCR. This feature exists in all products stated in Table 1.

### 5.2.2 Byte swap

The MSB and the LSB bytes of a half-word can be swapped in the output FIFO by setting the SB bit in DMA2D_OPFCCR.

This feature exists in all products stated in Table 1 except STM32L4 series.

The table below shows the swap operations required to match the LCD display color coding depending on the display color depth and the bus interface width.

**Table 5. Swap operations**

| Color depth | Interface bus width | Required operation | |
|---|---|---|---|
| | | Red blue swap | Byte swap |
| 8 bpp (256 colors) | 8-bit | No | No |
| | 16-bit | No | Yes |
| 16 bpp (64k colors) | 8-bit | No | Yes |
| | 16-bit | No | No |
| 18 bpp (262k colors) | 8-bit | Yes | No |
| | 16-bit | Yes | Yes |
| 24 bpp (16.7M colors) | 8-bit | Yes | No |
| | 16-bit | Yes | Yes |

## 5.3 DMA2D reordering use case examples

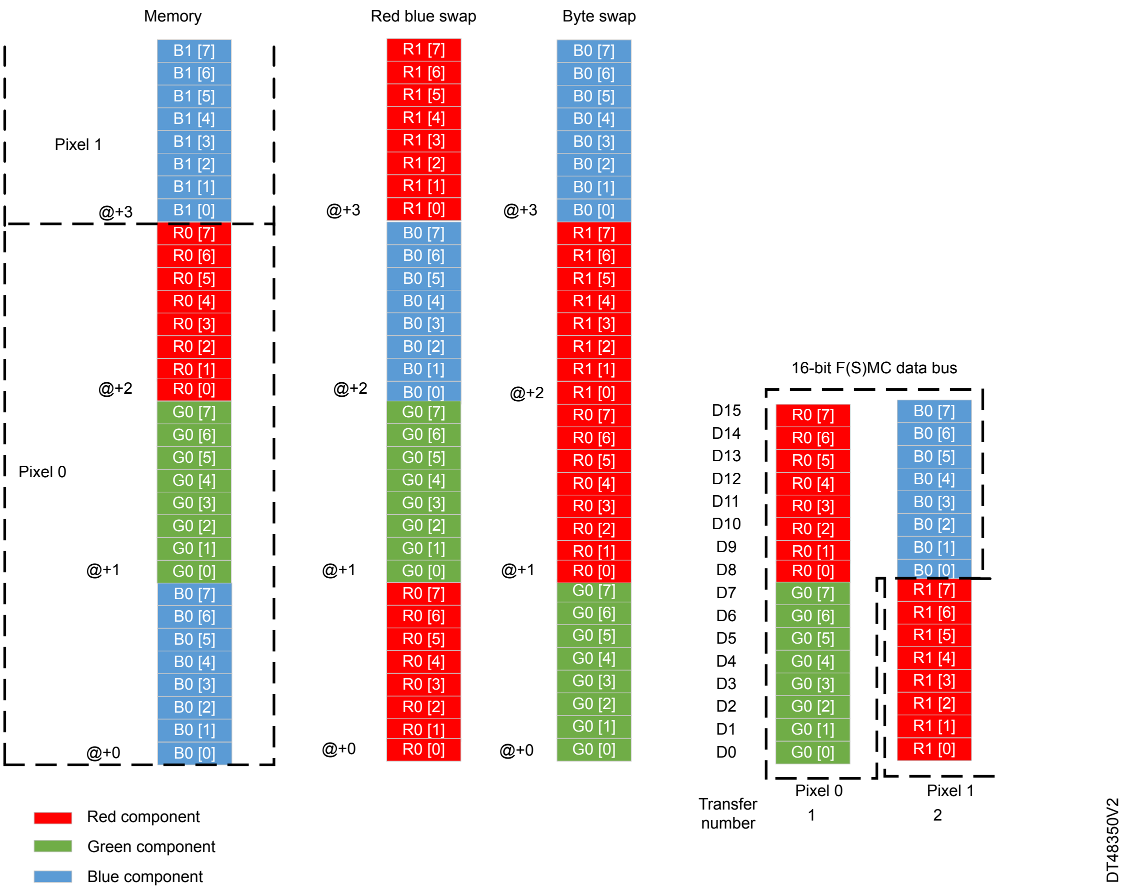
### 5.3.1 24 bpp/18 bpp over 16-bit F(S)MC data bus interface

In order to support 24-bpp displays using the 8080 standard, two operations are required on the frame buffer data:

- red and blue swap
- MSB and LSB bytes of a half-word swap

The figure below shows the operations performed by the DMA2D to have the good byte order corresponding to the Intel 8080 protocol for 24-bpp color depth over a 16-bit interface.

**Figure 8. DMA2D operations to support 24 bpp over 16-bit interface**



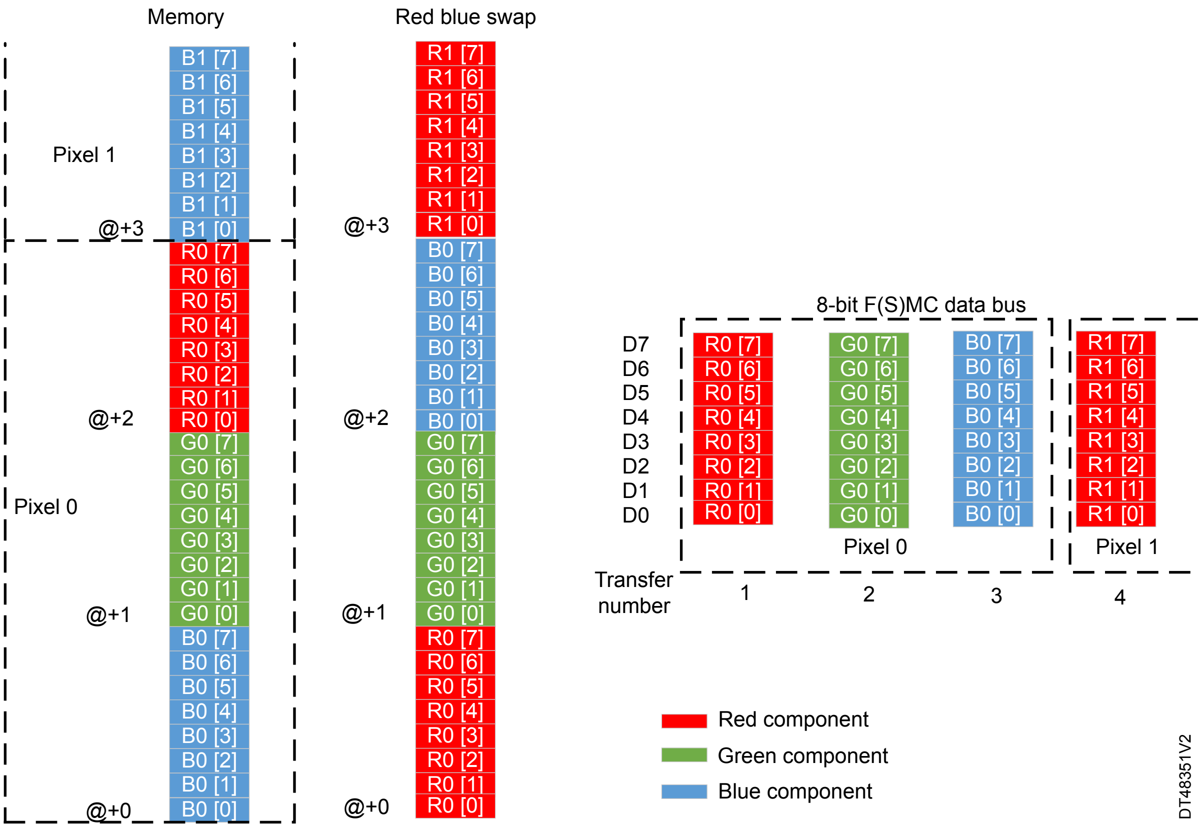

*Note: On MCUs not supporting the byte swap, a hardware fix can be implemented by swapping the data lines of the LCD interface on the board. The display D[15:8] lines are connected to the F(S)MC D[7:0] lines and the display D[7:0] lines are connected to the F(S)MC D[15:8] lines.*

### 5.3.2 24 bpp/18 bpp over 8-bit F(S)MC data bus interface

The red and blue swaps are required to get the correct order of bytes for 24 bpp displays using an 8-bit data bus.

The figure below shows the red and blue swap operation done by the DMA2D allowing to have the good bytes order.

**Figure 9. DMA2D operations to support 24 bpp over 8-bit interface**

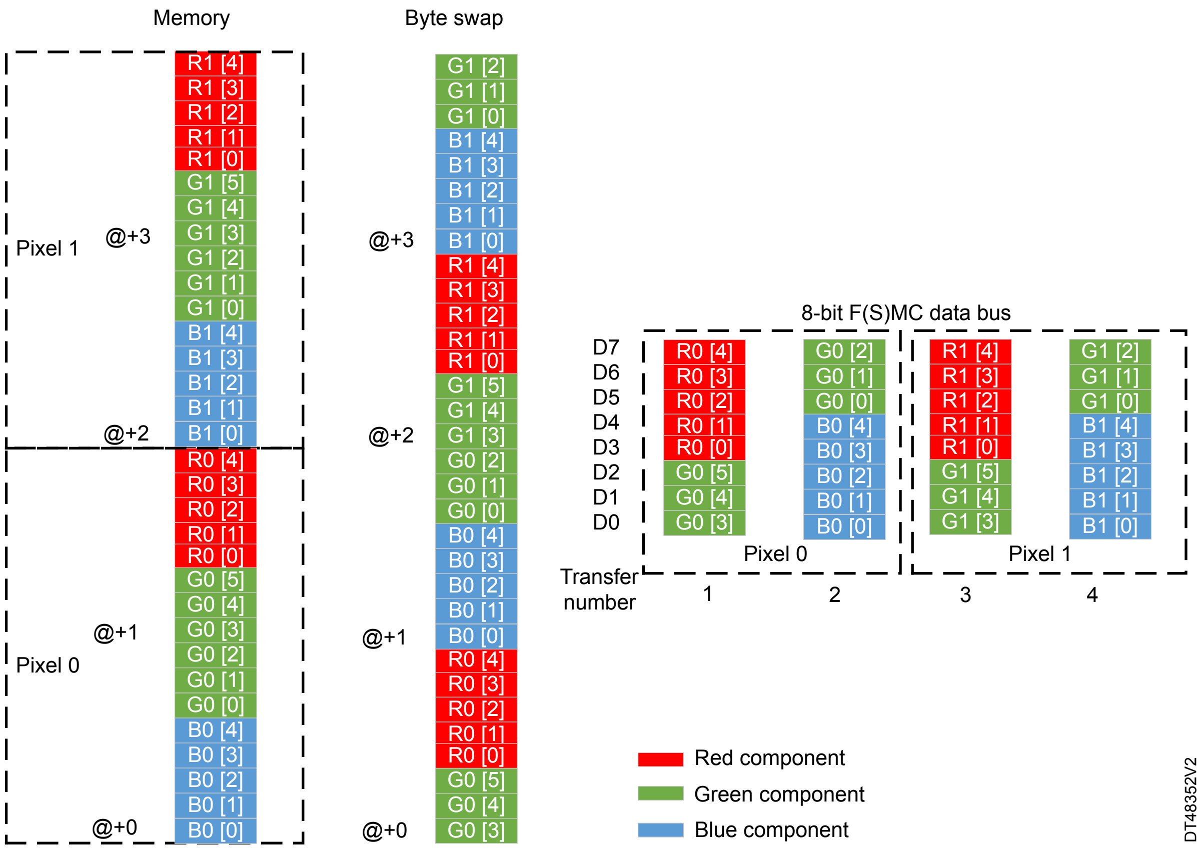


### 5.3.3 16 bpp over 8-bit F(S)MC data bus interface

In order to drive the 16-bpp Intel 8080 display over an 8-bit interface, the MSB and LSB bytes of a half word must be swapped.

The figure below shows how the swap operation allows having the good bytes order.

**Figure 10. DMA2D operations to support 16 bpp over 8-bit interface**



# 6 Conclusion

This application note explains how to easily transfer images to an LCD-TFT display via the F(S)MC interface using the Chrom-ART Accelerator (DMA2D), without using the CPU or the DMA resources. A focus is given to the correct control of the D/CX signal of the LCD-TFT display. Some code examples are provided to setup the DMA2D.

This document presents the new byte reordering features of the DMA2D used to support an update of 16.7M and 262k color Intel 8080 displays directly through the F(S)MC.

## Revision history

**Table 6. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 27-Jan-2017 | 1 | Initial release. |
| 23-Oct-2017 | 2 | Added STM32L4Rxxx/L4Sxxx devices in the whole document.<br>Added Section 5: New DMA2D features to support Intel 8080 displays. |
| 23-Sept-2021 | 3 | Updated:<br>• Document title<br>• Table 1. Applicable products<br>Section 5.2.1 Red and blue swap<br>Section 5.2.2 Byte swap<br>• Name of devices in the whole document |
| 22-Mar-2024 | 4 | Updated:<br>• Table 1. Applicable products<br>• Section 5.2.1: Red and blue swap<br>• Section 5.2.2: Byte swap<br>Terminology updated.<br>Applicable products updated. |

# Contents

# List of tables

# List of figures

**IMPORTANT NOTICE – READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved