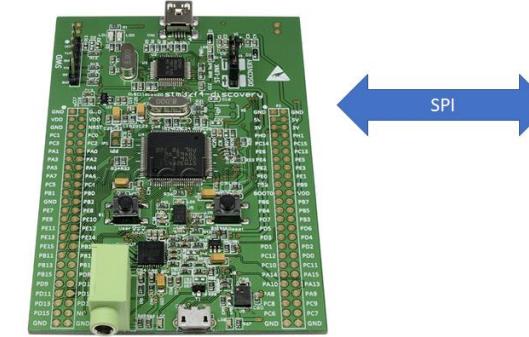
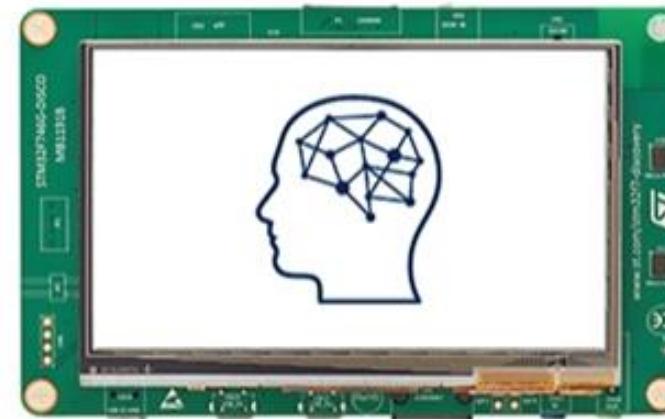
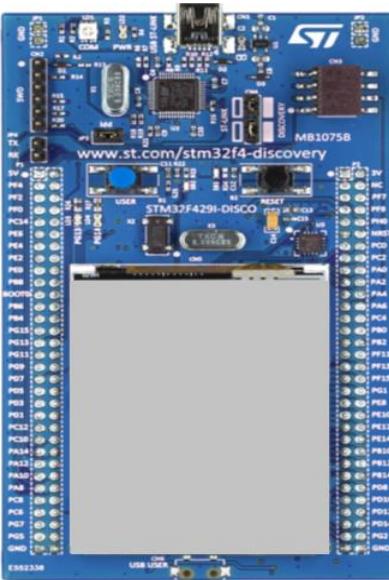
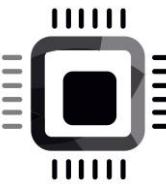


Mastering Microcontroller: STM32-LTDC, LCD-TFT, LVGL (MCU3)



 **LVGL**
Light and Versatile
Graphics Library

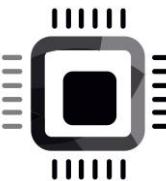


License

This power point presentation by BHARATI SOFTWARE is licensed under
[CC BY-SA 4.0](#)

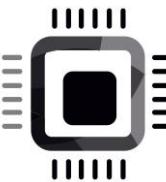
To view a copy of this license, visit

<https://creativecommons.org/licenses/by-sa/4.0>



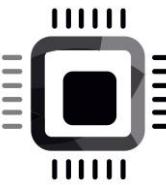
Important Links

- For the full video course please visit
 - <https://www.udemy.com/course/mastering-microcontroller-stm32-ltdc-lcd-tft-lvgl/>
- Course repository
 - <https://github.com/niekiran/EmbeddedGraphicsLVGL-MCU3>
 - Explore all FastBit EBA courses
 - <http://fastbitlab.com/course1/>
- For suggestions and feedback
 - contact@fastbitlab.com



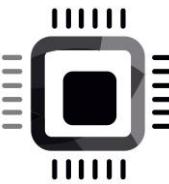
Social media

- Join our Facebook private group for technical discussion
 - <https://www.facebook.com/groups/fastbiteba/>
- Linkedin
 - <https://www.linkedin.com/company/fastbiteba/>
- Facebook
 - <https://www.facebook.com/fastbiteba/>
- YouTube
 - <https://www.youtube.com/channel/UCa1REBV9hyrzGp2mjJCagBg>
- Twitter
 - <https://twitter.com/fastbiteba>



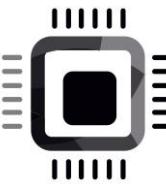
Highlights of the course

- Introduction to the embedded graphics system
- LCD-TFT interfacing using both MPI-DPI and MPI-DBI
- Using LTDC(LCD-TFT Display Controller) peripheral of STM32 MCU
- Bare metal step-by-step code implementation
- Creating STM32 UI projects using LVGL



Embedded graphics system

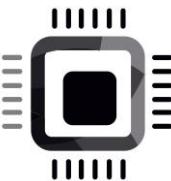
A microcontroller based system which runs embedded graphics application



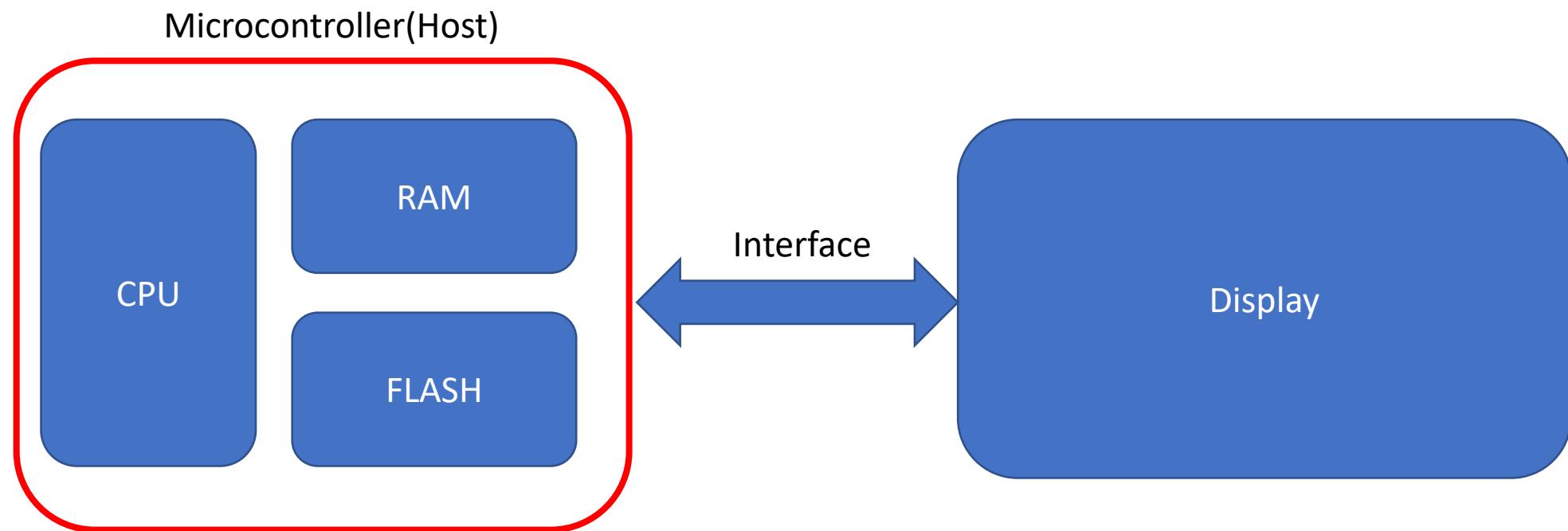
Graphics application

A microcontroller based interactive application which may involve any of the below components to display on the display device

- Colours
 - Color of the text
 - Background color / layer color
- Shapes
 - Box
 - Circle
 - Arrows
 - Lines
- Images
- Texts
- Graphical effects(scrolling, sliding, swipe, press, release)
- Widgets(simple button, radio button, check box)
- Videos
- 3D rendering
- Gaming

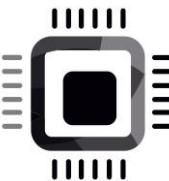


Embedded graphics system

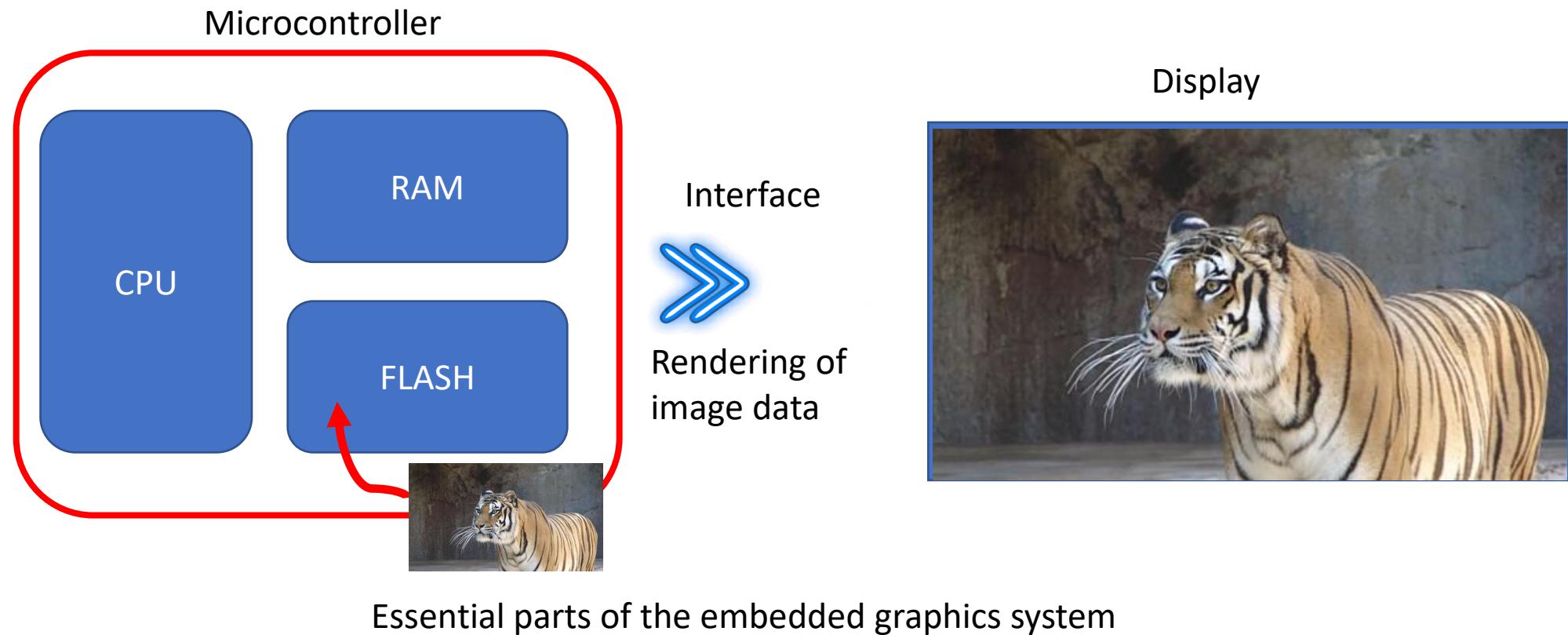


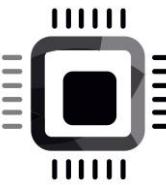
A simple microcontroller based graphics system

- LCD
 - TFT
 - IPS
- LED
- Plasma
- CRT
- OLED



Embedded graphics system



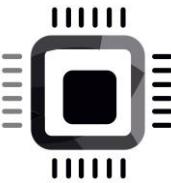


Important parts of the embedded graphics system

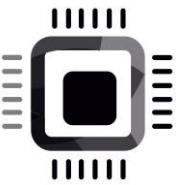
- A Microcontroller(Host)
 - Processor (Executes your code, updates frame buffer)
 - RAM (Frame buffer)
 - Flash (static images, fonts, texts,etc)
- Display Module
 - Glass(Where you see the graphics)
 - Driver chip(Interprets the signals sent by the host, generates required electric signals and voltages to lit the pixels of the display panel)

Frame buffer → A memory area which holds the pixel values of the frame to be updated on the display

Other important parts of the embedded graphics system



- Display controller
 - Present at the HOST side
 - Generates display timing-related signals
 - Transmits pixel data
- External memories
 - External Flash(your code + graphics components(images, texts) may not fit inside the internal flash)
 - External RAM
- Graphics library (LVGL, TouchGFX)
- UI designer tool
 - To create an interactive UI application
- Touch sensing
 - Touch panel
 - Touch screen controller(Which senses touch panel and informs HOST)
- DMA
 - Helps to transfer frame buffer to display without the intervention of the CPU
 - Helps to transfer graphics details from flash to frame buffer without the intervention of the CPU

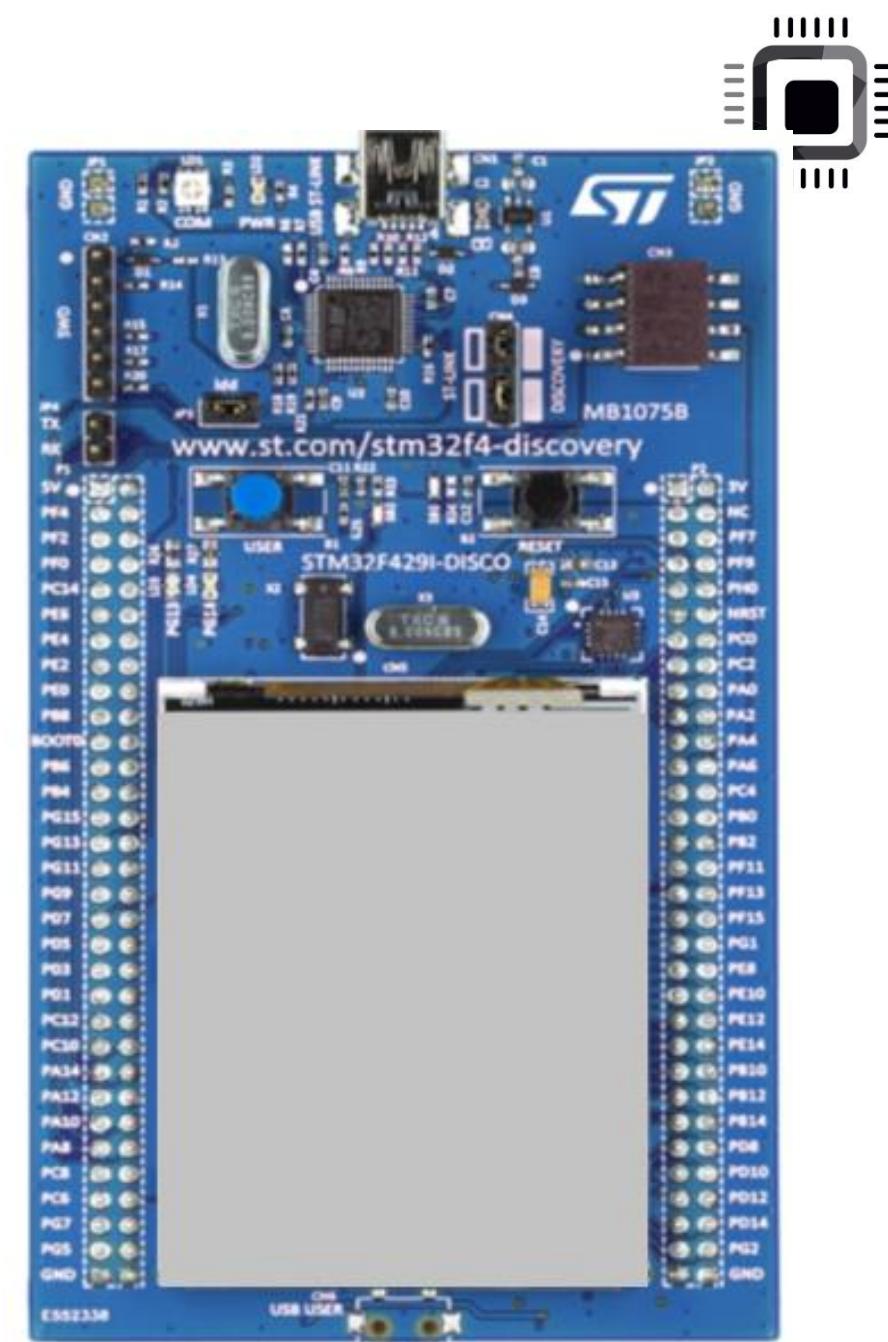


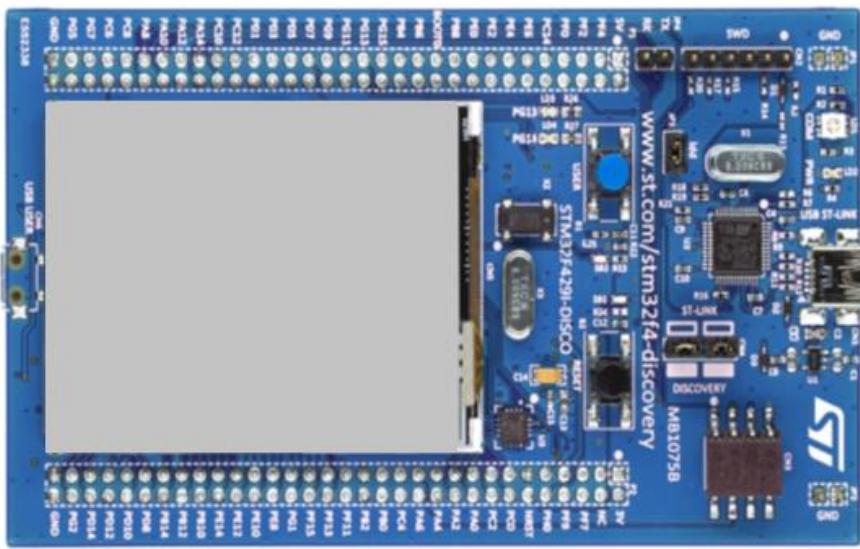
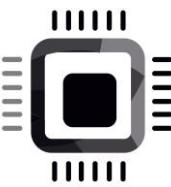
Hardware to use in this course

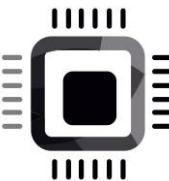
Use any one of these boards

Option 1:
32F429IDISCOVERY Discovery kit
with STM32F429ZI MCU

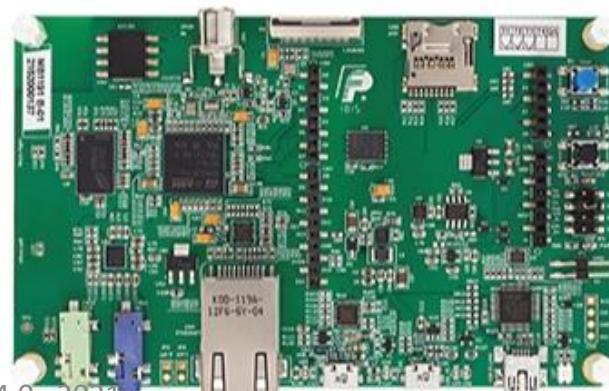
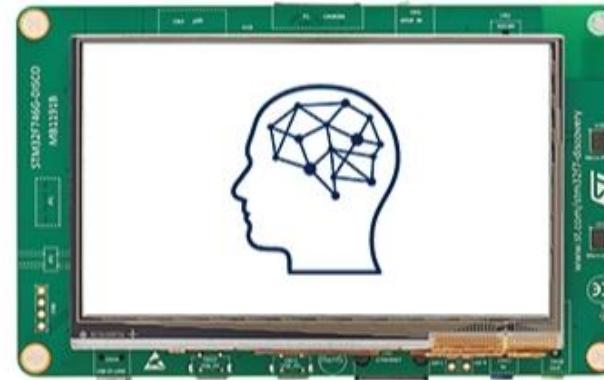
Recommended

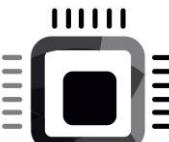






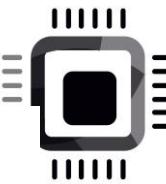
Option 2: 32F746GDISCOVERY Discovery kit (STM32F746NG MCU)





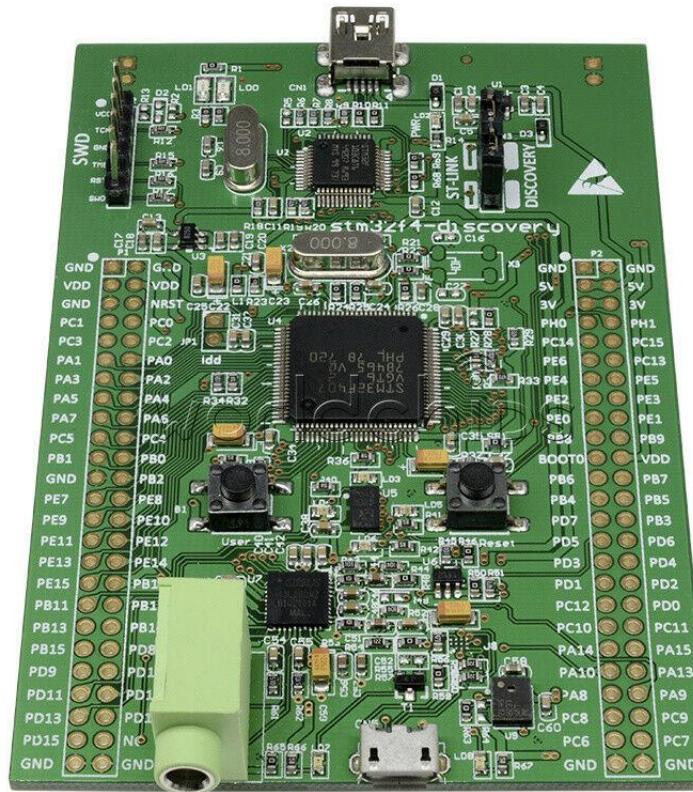
Option 3:
STM32F7508-DK
Discovery kit with STM32F750N8 MCU



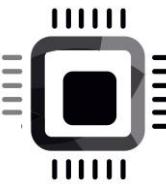


Option 4:

STM32F4DISCOVERY Discovery kit with
STM32F407VG MCU

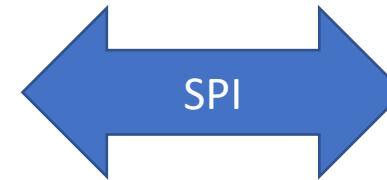
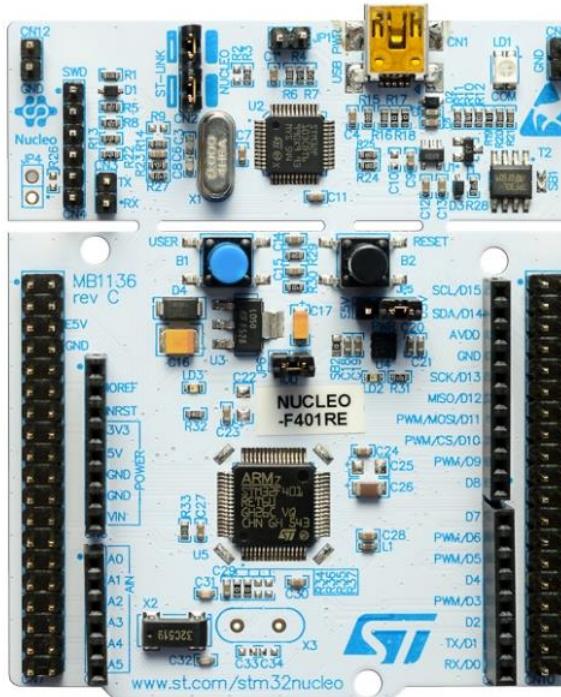


2.4 inch SPI Interface
240×320 Touch Screen TFT
Colour Display Module

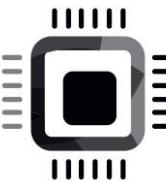


Option 5:

NUCLEO-F4x STM32 Nucleo-64 development board

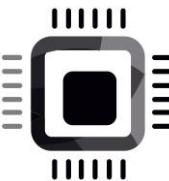


2.4 inch SPI Interface
240x320 Touch Screen TFT
Colour Display Module



Board	MCU	Processor	On-chip RAM (KiB)	On-chip Flash (MiB)	On board External SDRAM	On board external flash	LCD controller	Chrome ART Accelerator™	On board LCD-TFT
32F746GDISCOVERY	STM32F746NG	Cortex-M7	340	1	Yes 128Mbit	Yes 128Mbit	Yes	Yes	Yes 480x272
STM32F7508-DK	STM32F750N8	Cortex-M7	340	64KiB	Yes 128Mbit	Yes 128Mbit	Yes	Yes	Yes 480x272
32F429IDISCOVERY	STM32F429ZI	Cortex-M4	256	2	Yes 64Mbit	No	Yes	Yes	Yes 240x320
STM32F4DISCOVERY	STM32F407VG	Cortex-M4	192	1	No	No	Yes	Yes	No
NUCLEO-F4x	??	Cortex-M4	??	??	No	No	No	No	No

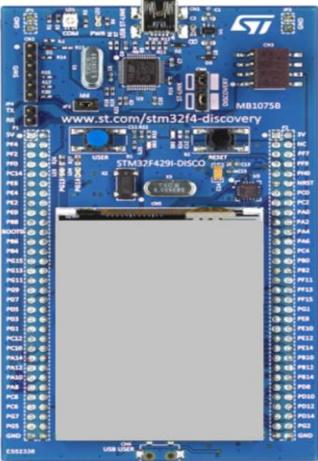
<https://www.st.com/en/evaluation-tools/stm32-mcu-mpu-eval-tools.html>



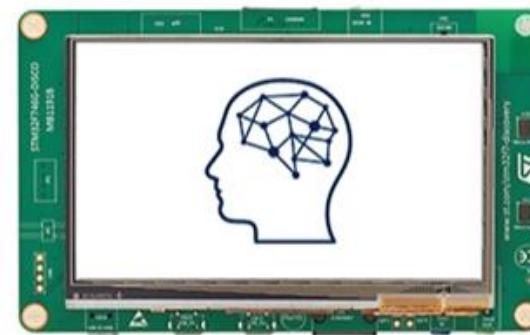
Roaring tiger demo

Runs on,

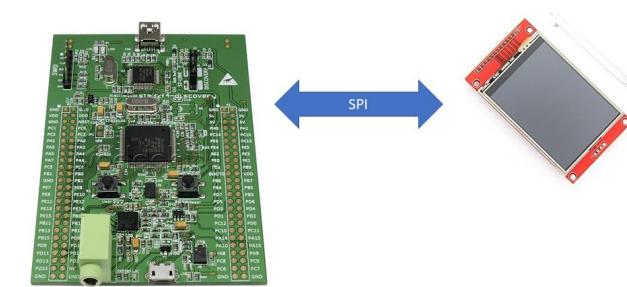
32F429IDISCOVERY

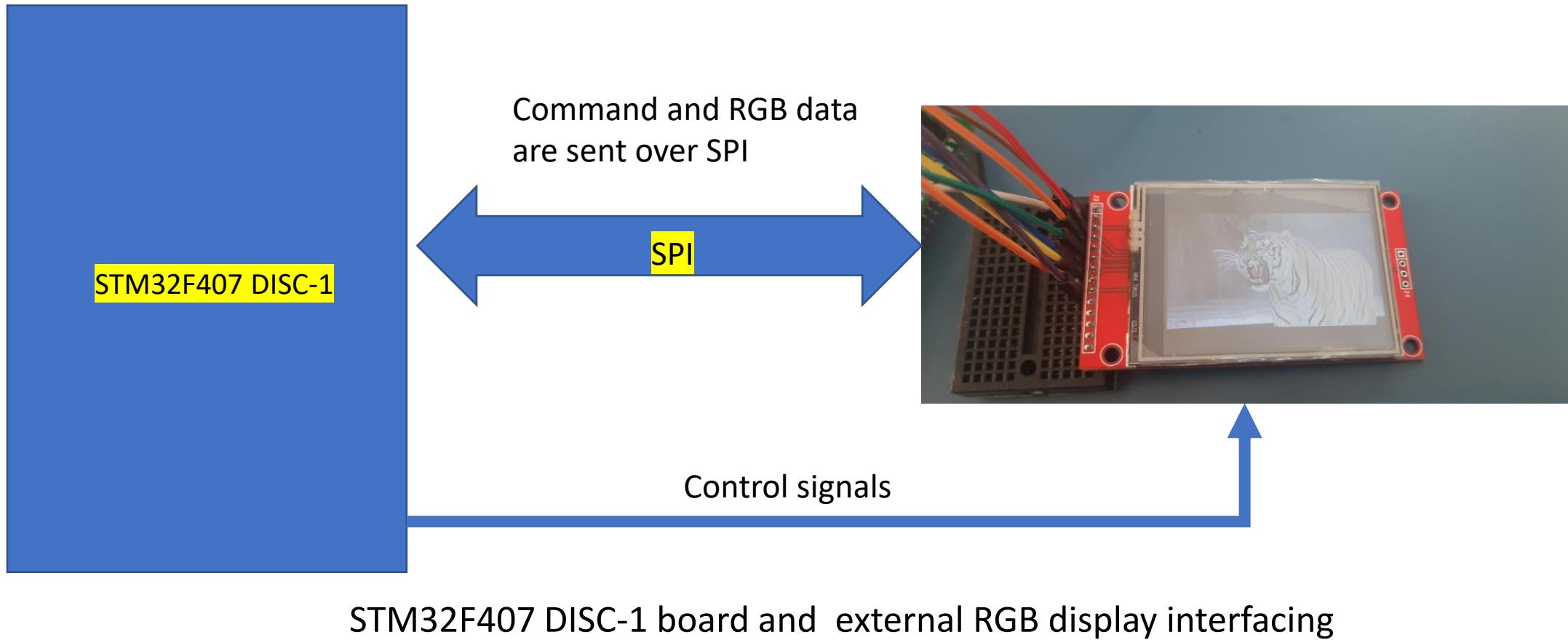
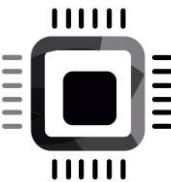


32F746GDISCOVERY

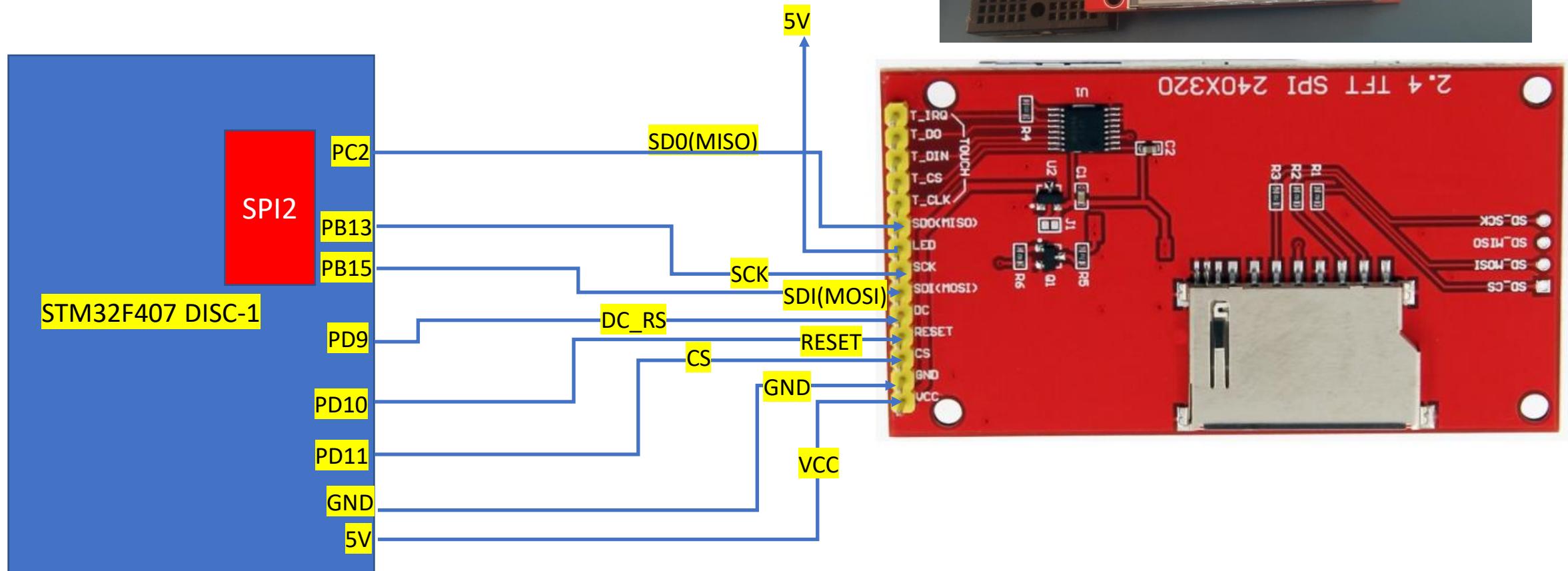


32F407DISCOVERY + SPI
based LCD module

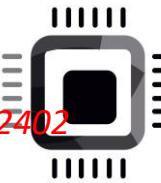




Roaring tiger demo



STM32F407 DISC-1 board and external RGB display interfacing



Number	Pin Label	Description
1	VCC	5V/3.3V power input
2	GND	Ground
3	CS	LCD chip select signal, low level enable
4	RESET	LCD reset signal, low level reset
5	DC/RS	LCD register / data selection signal, high level: register, low level: data
6	SDI(MOSI)	SPI bus write data signal
7	SCK	SPI bus clock signal
8	LED	Backlight control, high level lighting, if not controlled, connect 3.3V always bright
9	SDO(MISO)	SPI bus read data signal, if you do not need to the read function, you can not connect it
(The following is the touch screen signal line wiring, if you do not need to touch function or the module itself does not have touch function, you can not connect them)		
10	T_CLK	Touch SPI bus clock signal
11	T_CS	Touch screen chip select signal, low level enable
12	T_DIN	Touch SPI bus input
13	T_DO	Touch SPI bus output
14	T_IRQ	Touch screen interrupt signal, low level when touch is detected

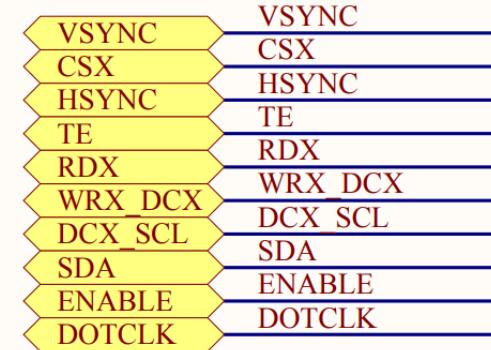
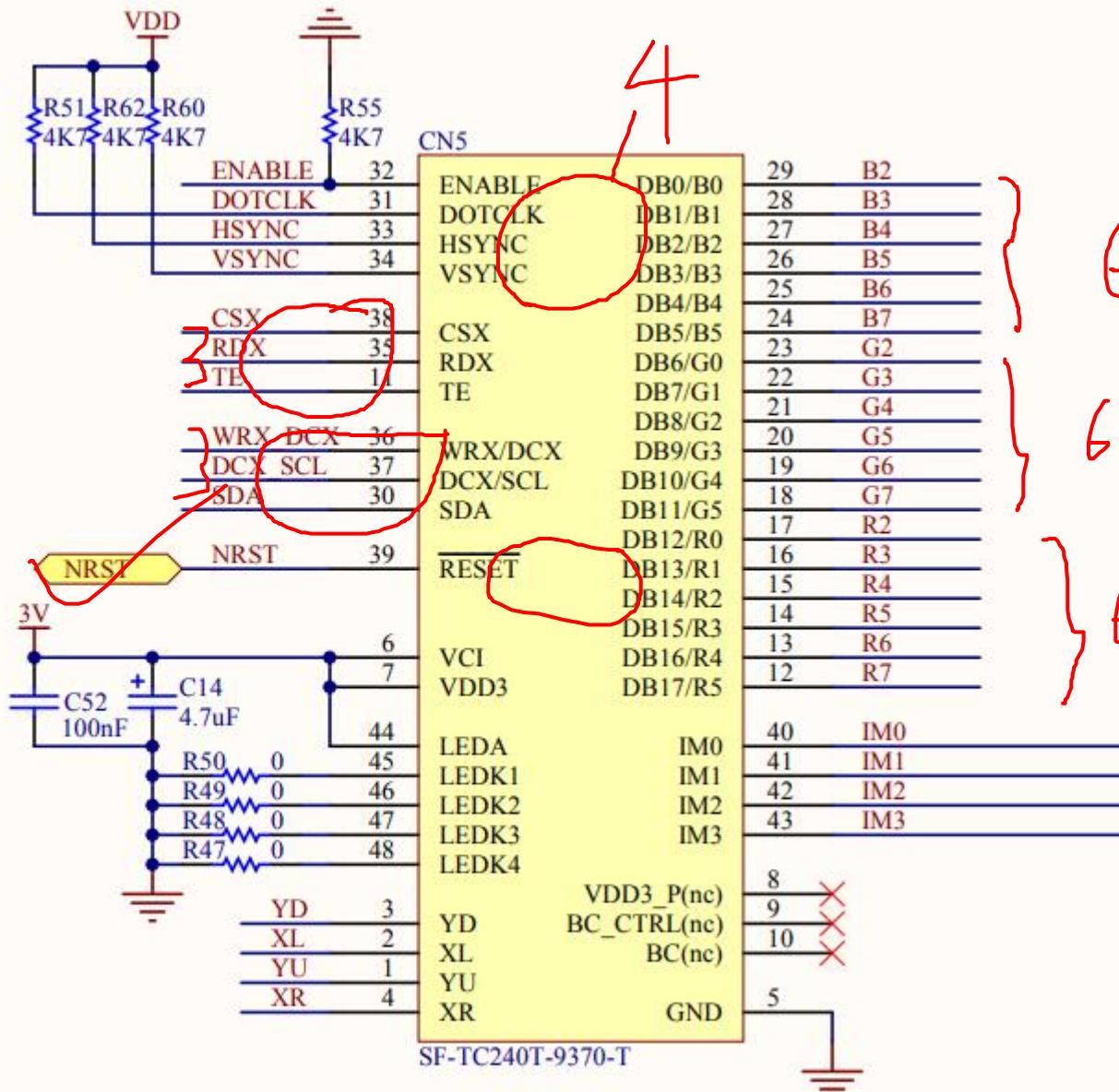
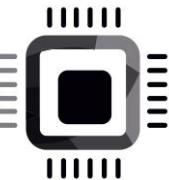
Pin details

Source :

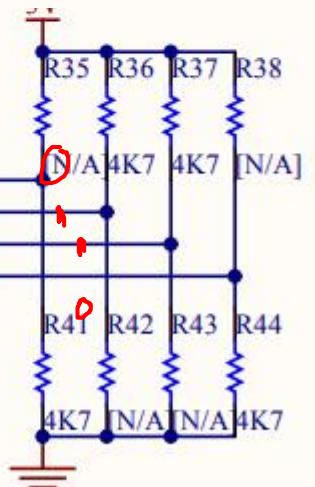
http://www.lcdwiki.com/2.4inch_SPI_Module_ILI9341_SKU:MSP2402

Name	Parameter
Display Color	RGB 65K color
SKU	have touch screen: MSP2402
	have no touch screen: MSP2401
Screen Size	2.4(inch)
Type	TFT
Driver IC	ILI9341
Resolution	320*240 (Pixel)
Module Interface	4-wire SPI interface
Active Area (AA area)	36.72x48.96(mm)
Module PCB Size	77.18x42.72(mm)
Operating Temperature	-20°C~60°C
Storage Temperature	-30°C~70°C
VCC power voltage	3.3V~5V
Logic IO port voltage	3.3V(TTL)
Power Consumption	TBD
Rough Weight(Package containing)	No touch: 26(g) / With touch: 36(g)

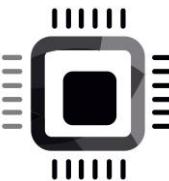
Product parameters



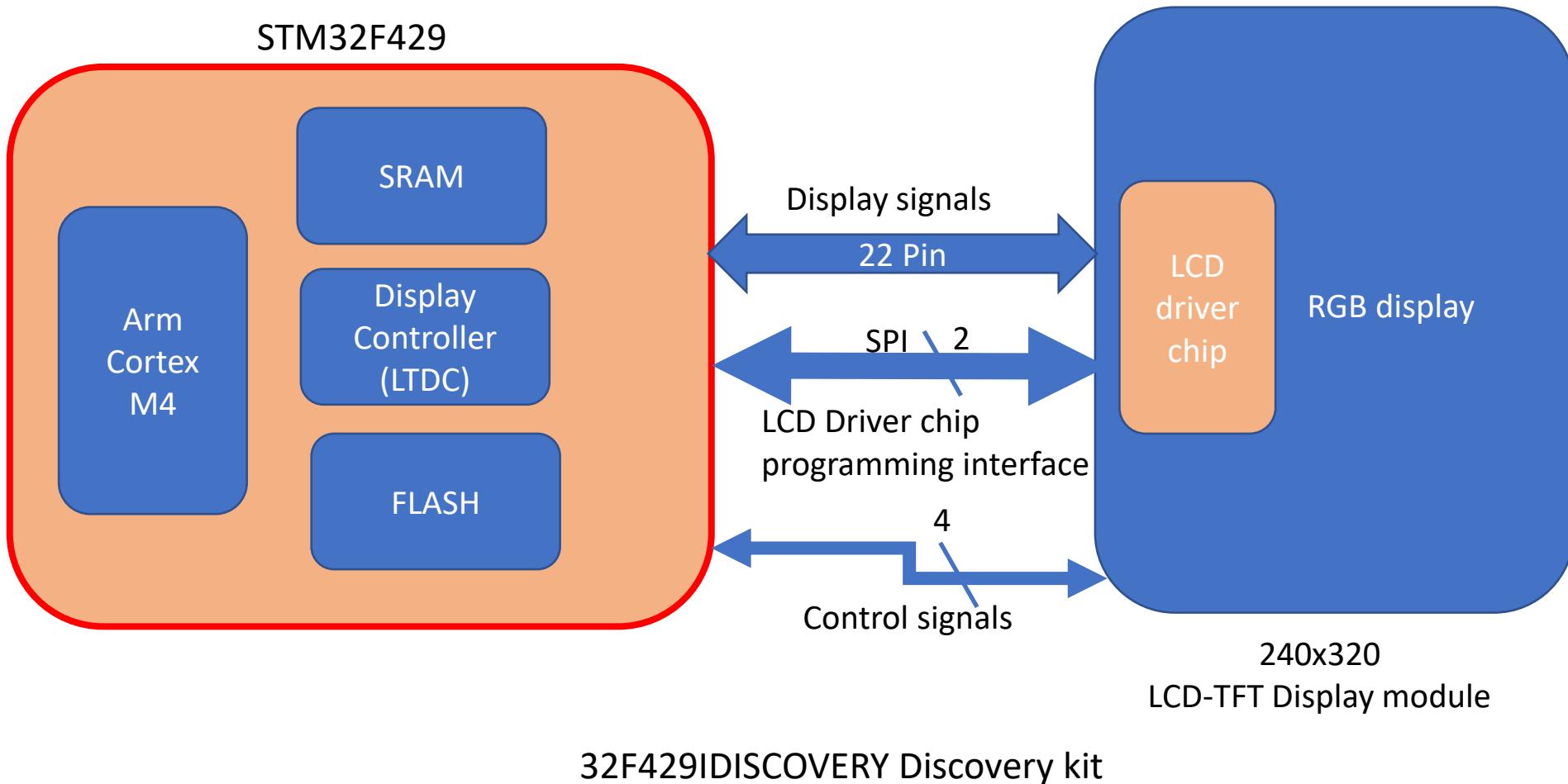
SPI5_SCK
SPI5_MOSI



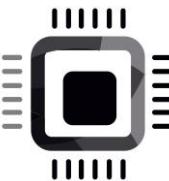
IM[0..3] = 0110 --> 4-wire 8-bit serial



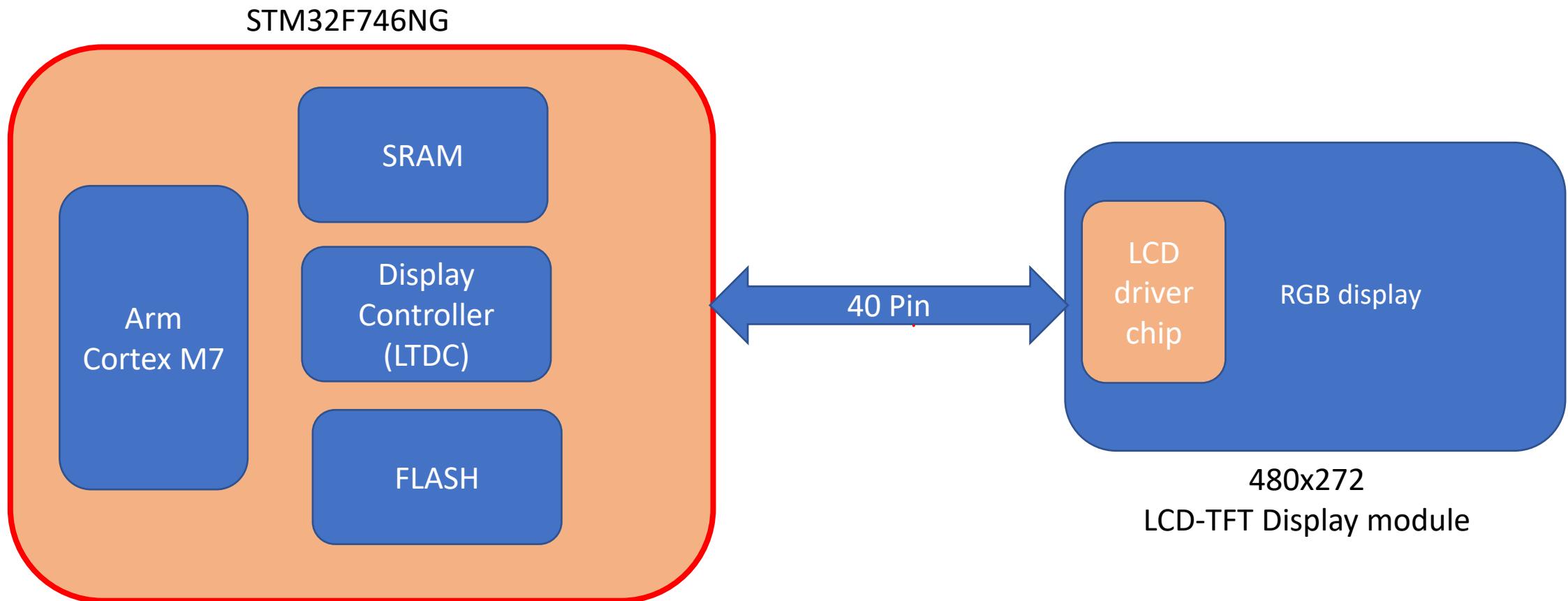
Embedded graphics system

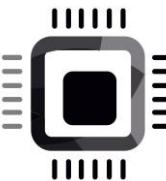


For more information refer the schematics of 32F429IDISCOVERY Discovery kit



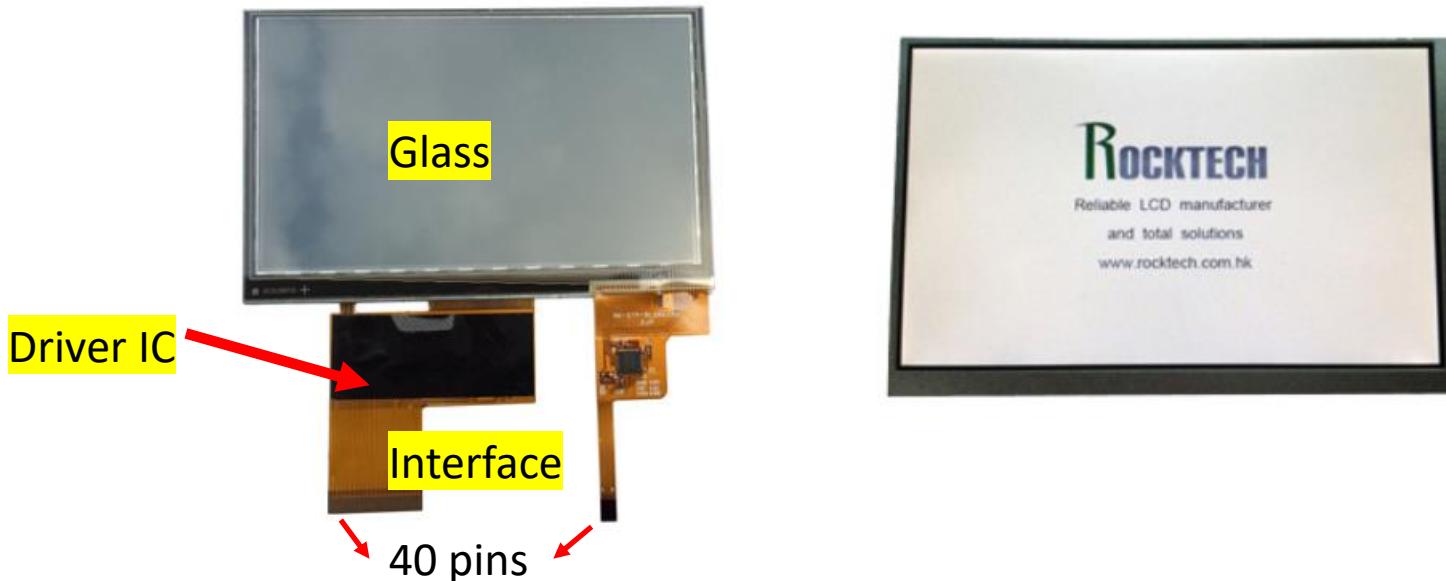
Embedded graphics system





32F746GDISCOVERY display module

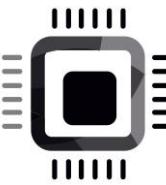
4.3" TFT with CTP(Capacitive Touch Panel)



6.1 TFT LCD Pin Description

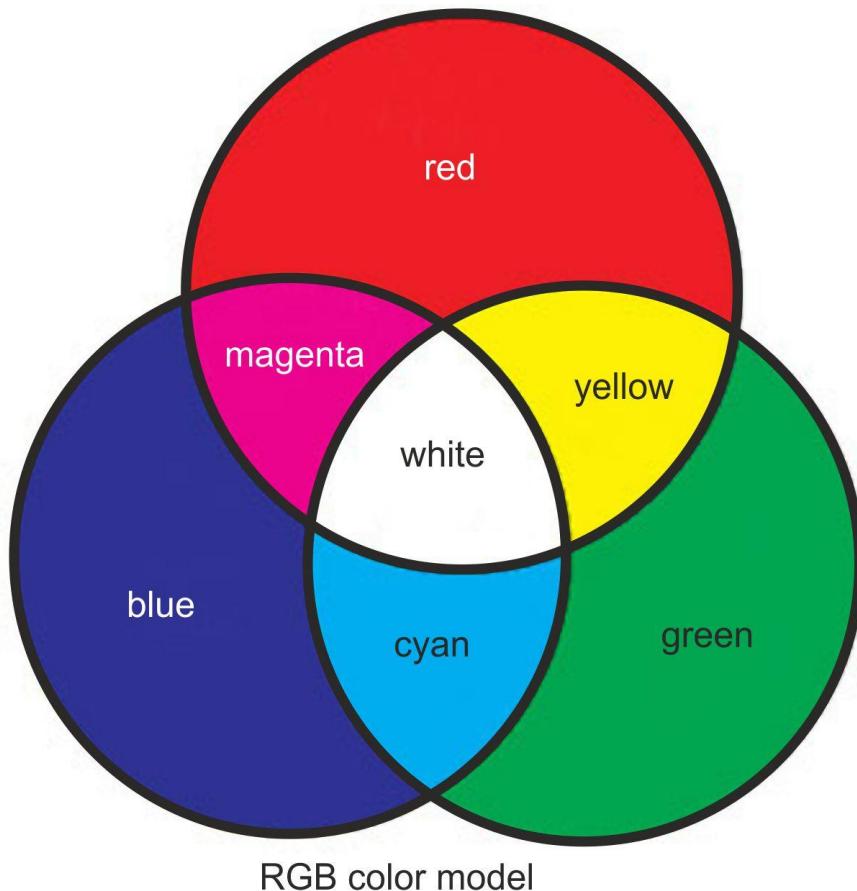
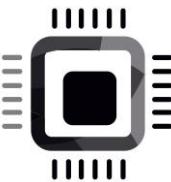
Pin	Symbol	Description
1	K	B/L Power input PIN Cathode
2	A	B/L Power input PIN anode
3	GND	Ground
4	VDD	Power input
5~12	R0~R7	Red Data
13~20	G0~G7	Green Data
21~28	B0~B7	Blue Data
29	GND	Ground
30	DCLK	Data clock signal
31	DISP	Standby Mode DISP="1", Normal operation DISP="0", Standby mode.
32	H SYNC	Horizontal synchronizing signal
33	V SYNC	Vertical synchronizing signal
34	DE	Data ENABLE signal
35	GND	Ground
36	GND	Ground
37	RESET	RESET
38	INT	Interrupt
39	SCL	I2C clock
40	SDA	I2C data

Display Module: A Functional module to show image on it, can consists of display glass, display driver IC, other peripheral components and circuits and display interface

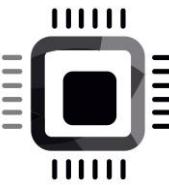


What is an RGB display?

A display which takes **red-green-blue** signal components of a pixel to generate the colors to be displayed on the screen

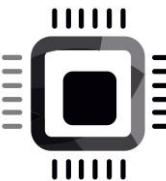


A model in which R-G-B primary colors are added together in various ways to reproduce a broad array of colors



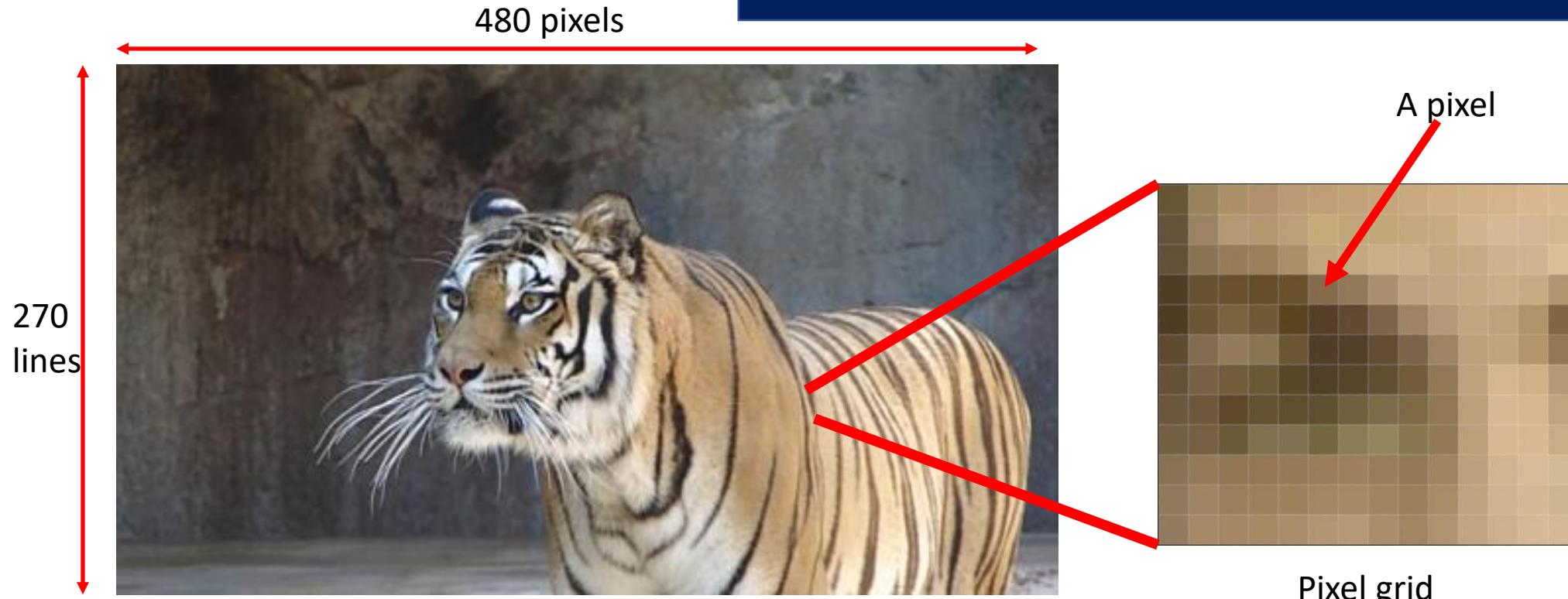
Some important terminologies

- Pixel
- Pixel density (PPI)
- Pixel color depth (Bit depth)
- Pixel format
- Resolution



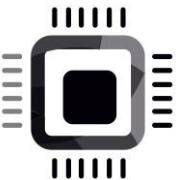
Pixel(Picture element)

Pixel of the image : A smallest information(color information) of the image stored in computer memory

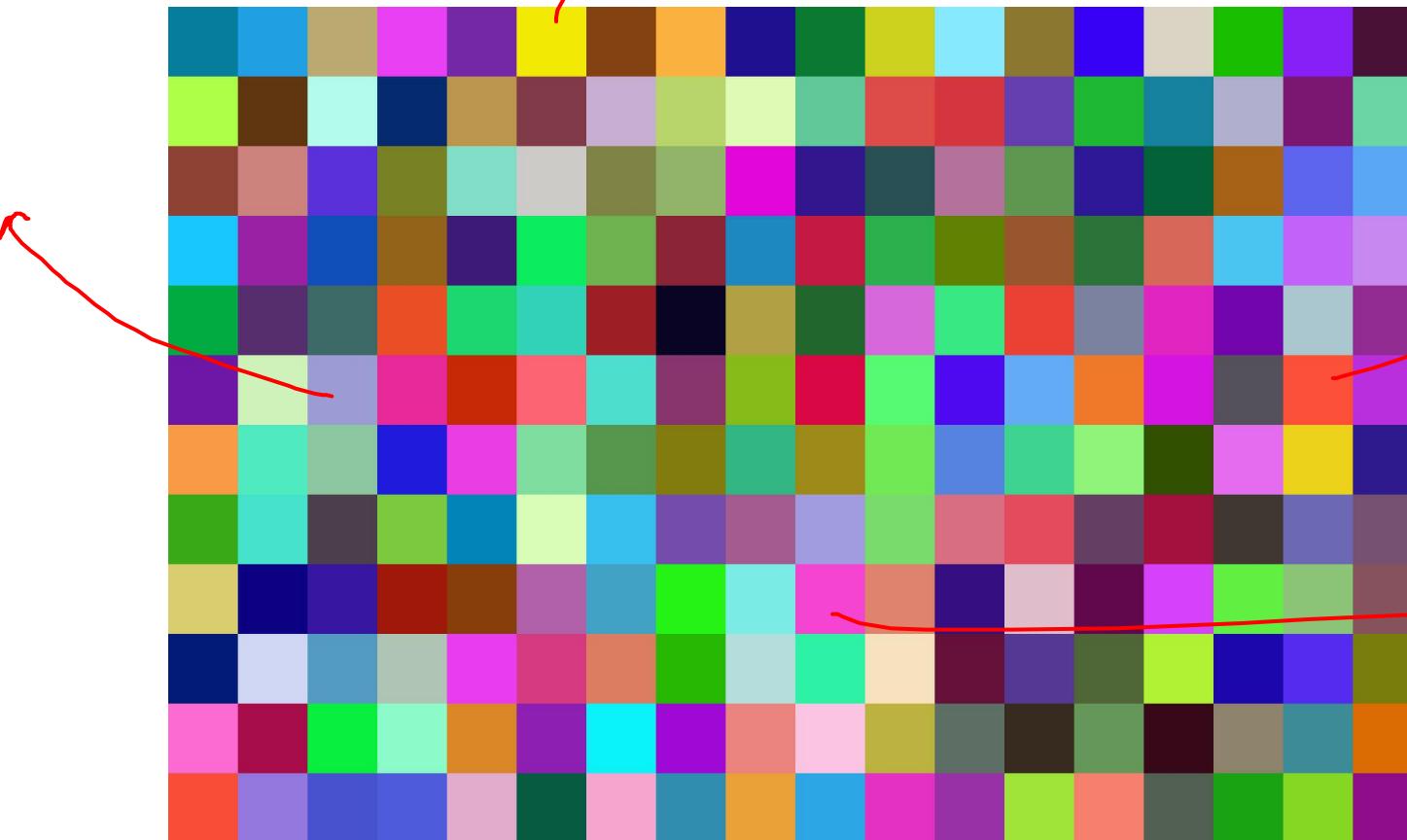


Dimensions	480 x 270
Width	480 pixels
Height	270 pixels
Bit depth	32

A color image

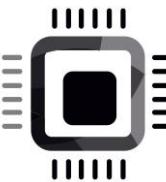


$R = 255, G = 0, B = 255$



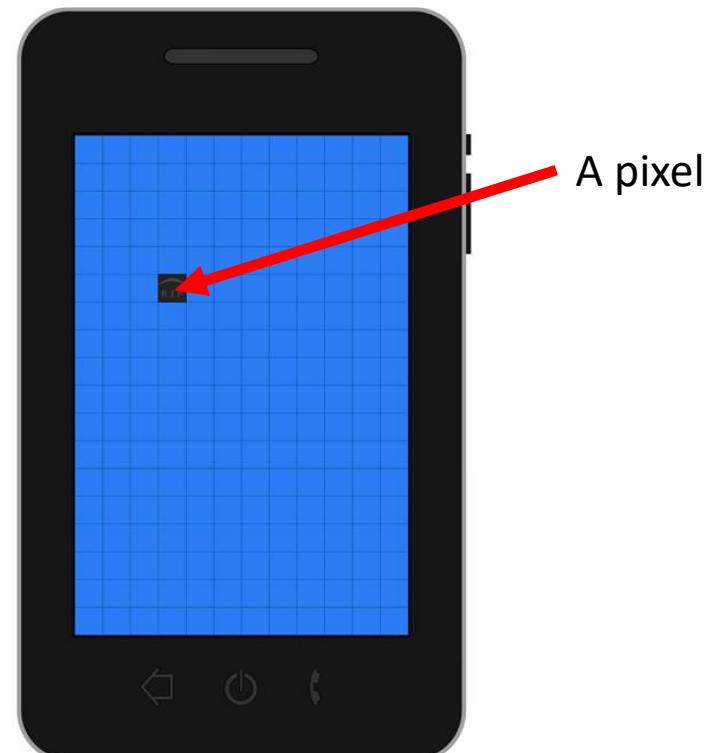
$R = 255, G = 0, B = 0$

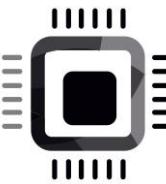
Grid of pixels



Pixel of the display

A smallest glowing(color producing) electronic element of the display module is also called as a pixel



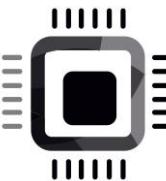


A Pixel of the image : The smallest information(color information) of the image stored in computer memory

How much memory does one pixel consume in compute memory ? (**Pixel depth/color depth/bit depth**)

- Depends on pixel format
- measured in bpp(bits per pixel)

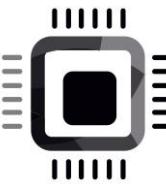
What are the different types of **pixel formats/color formats**?



Important pixel formats(Color format)

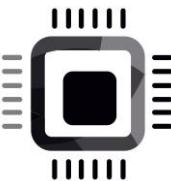
Pixel format	Meaning
ARGB8888	32 bits; RGB with Alpha component; 8bits/component .
RGB888	24 bits; 8bits/component; No alpha
RGB565	16bits; 5 bits R , 6 bits G , 5 bits B; No alpha
ARGB1555	16bits; 1 bit for alpha , 5 bits for color
ARGB4444	16bits; 4bits/component
RGB666	16bits; 6bits/component; No alpha
ARGB2222	8bits; 2bits/component
ABGR2222	8bits; 2bits/component
L8 (8-bit luminance or CLUT)	
L8_RGB888	8 bit Indexed color look up table ; size = 24
L8_RGB565	8 bit Indexed color look up table ; size = 16
AL44	4-bit alpha + 4-bit luminance
AL88	8-bit alpha + 8-bit luminance

Pixel format	Meaning
L4	
A8	8 bits alpha ; No RGB
A4	4 bits alpha ; No RGB
GRAY4	4bits Gray intensity value
GRAY2	3bits Gray intensity value
BW	1 bit black & white

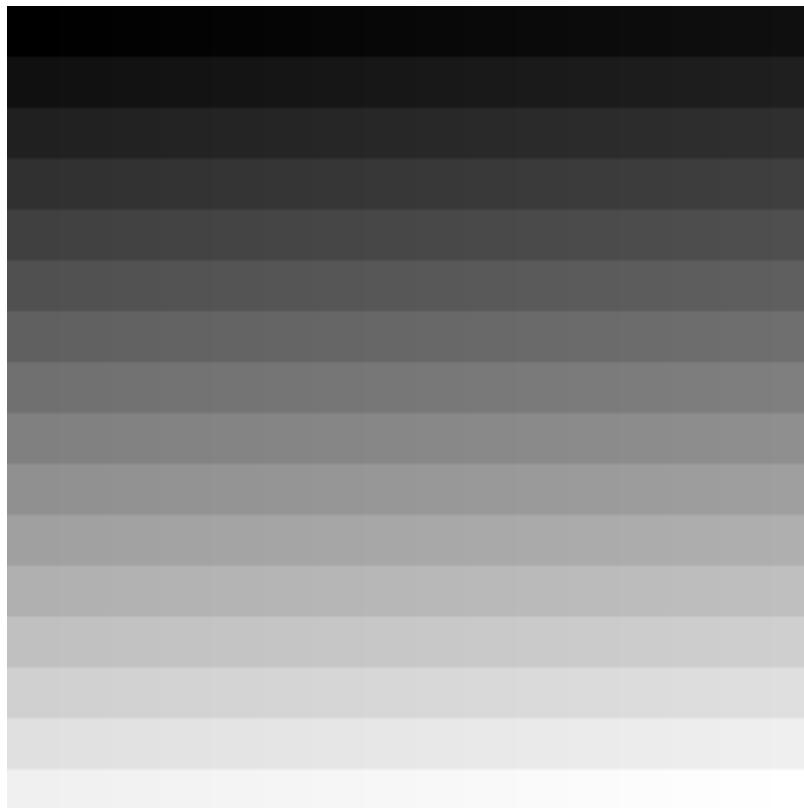


If an image is represented in ARGB8888 pixel format, that means each pixel of that image consumes 32bits(32bpp) in memory, and each pixel has 4 components

1. Alpha(opacity) component of 8bits
2. Red color component of 8bits
3. Green color component of 8bits
4. Blue color component of 8bits



GRAY8



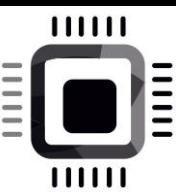
0

255

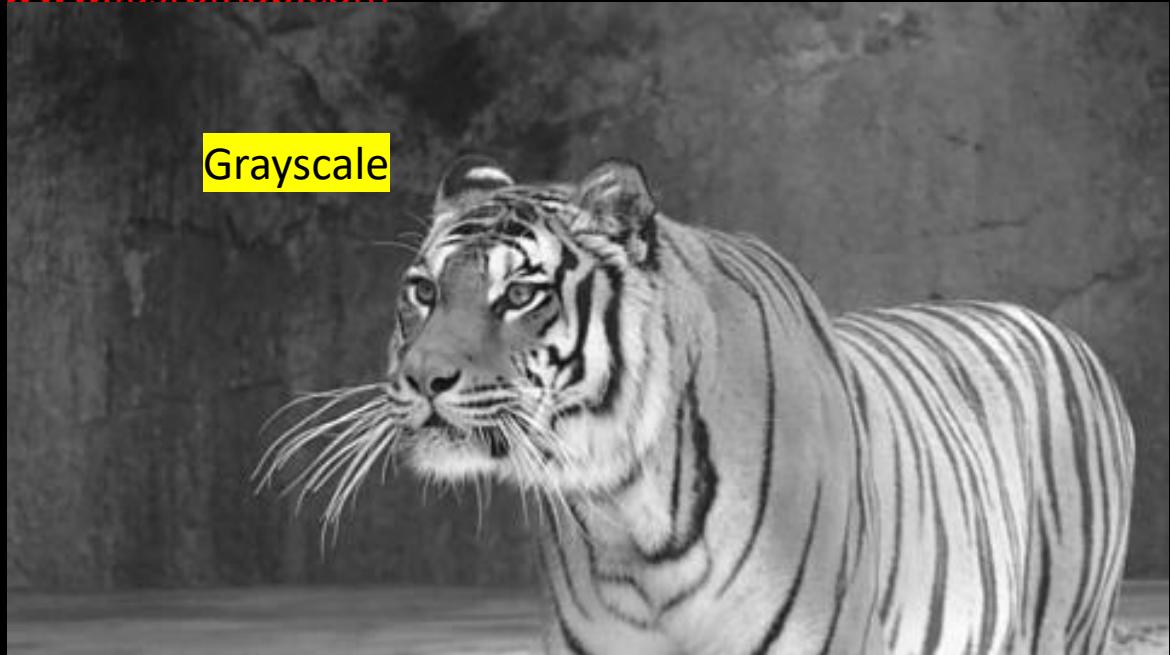
Gray intensity

The intensity of the color varies while moving from black to white.

When R=G=B, Gray color is produced



Grayscale

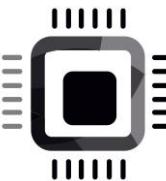


B&W



True Color





L8 format

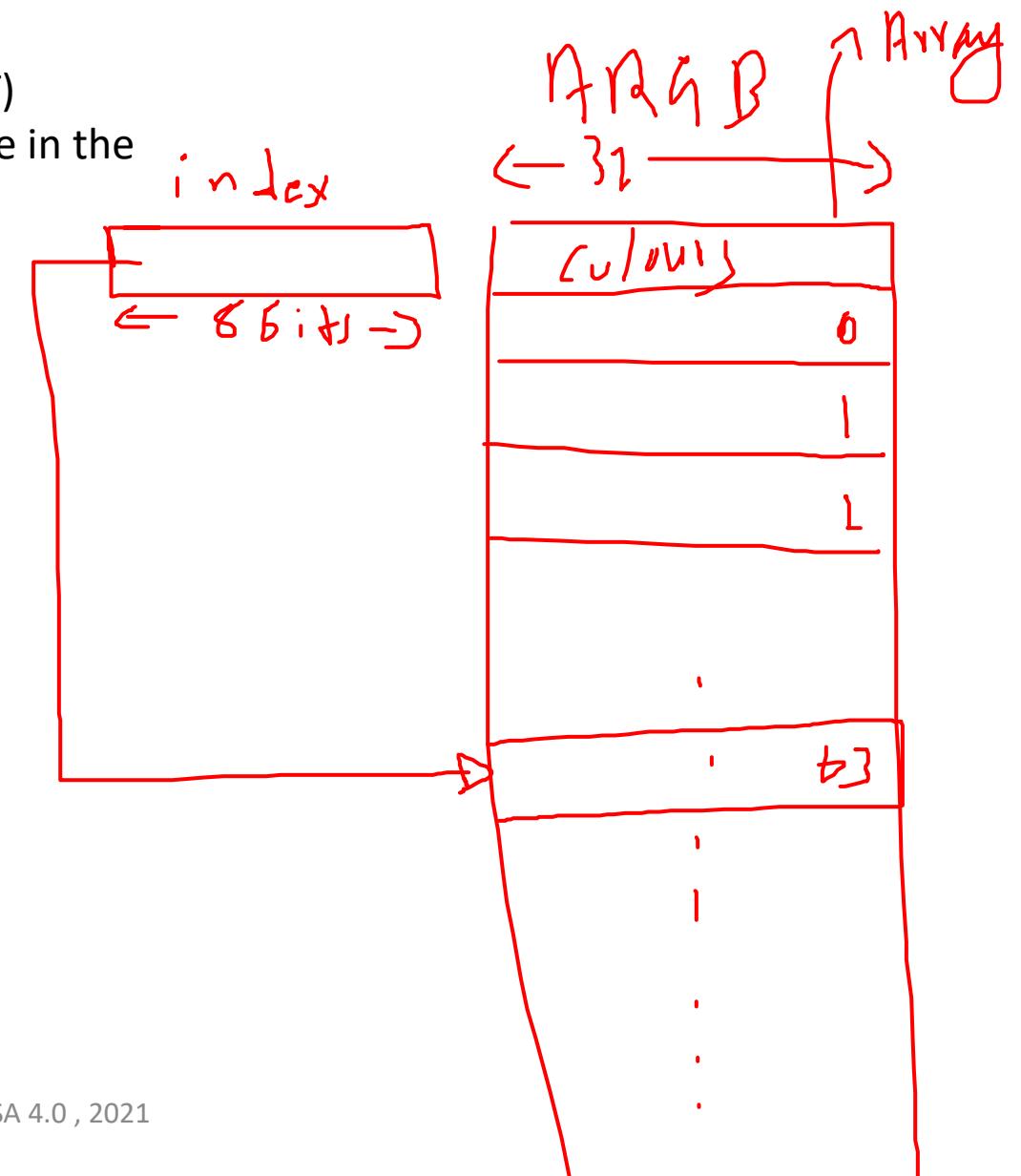
Represents the colors of the image by color look up table(CLUT)

In L8 format 8bit index value is used to look for the color value in the predefined color look up table.

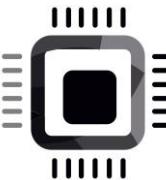
$$480 \times 270 \times 4 =$$

$$480 \times 270 \times 8 + 32 \times 255$$

=



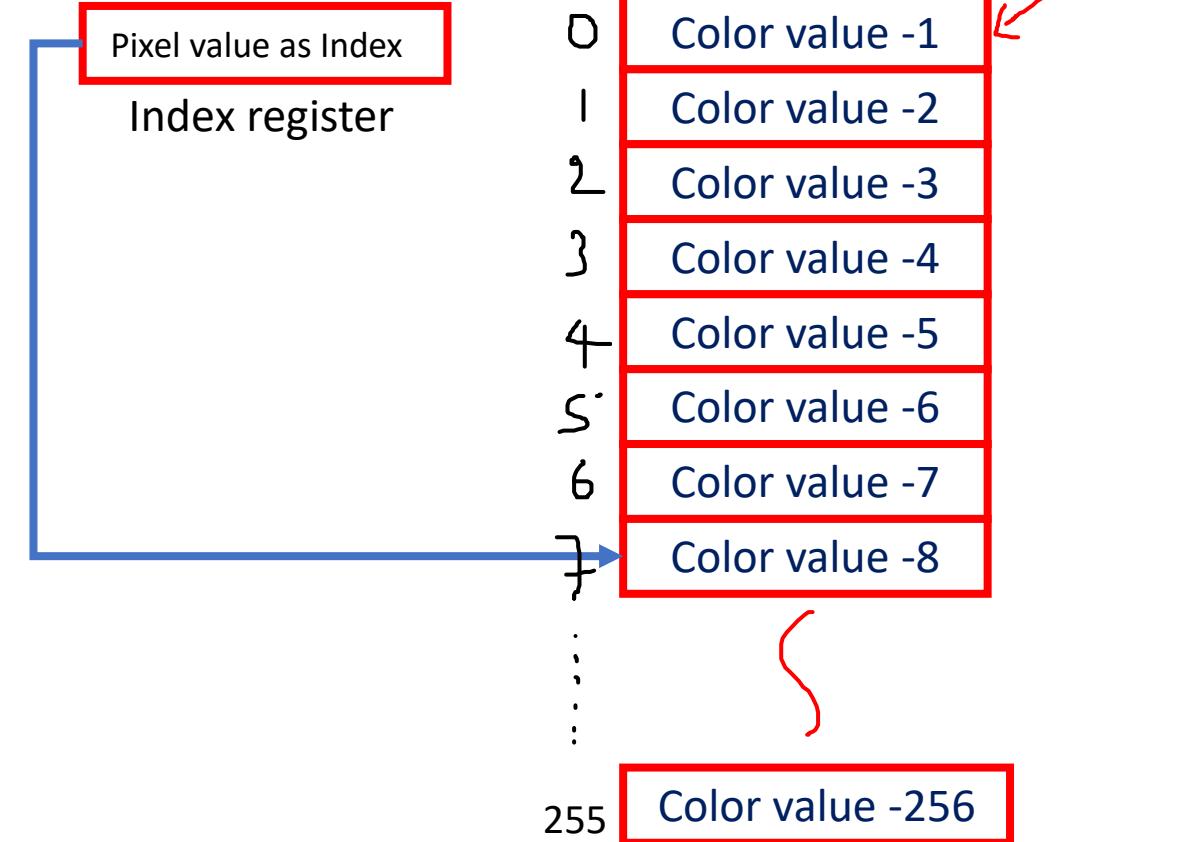
L8 format

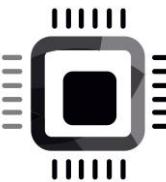


Represents the colors of the image by color lookup table(CLUT).

In L8 format, 8bit index value is used to look for the color value in the predefined color lookup table.

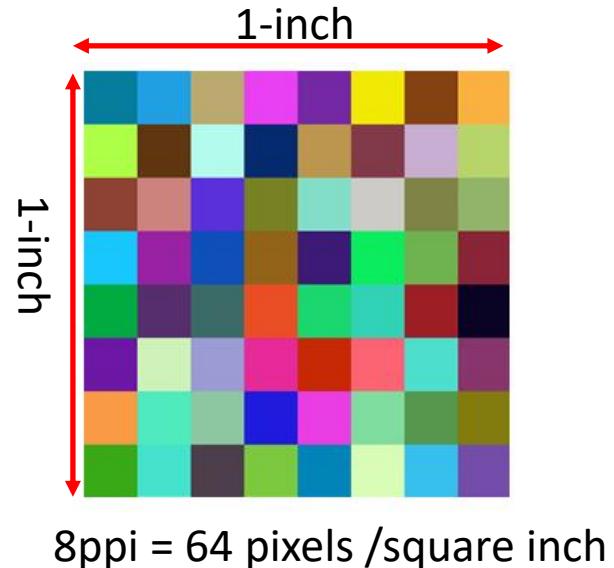
Frame buffer Resolution	Pixel format	Total memory consumed
480x270	RGB888	480x270x3 = 3805KiB
480x270	L8_RGB888	480x270x1 + 256 x 3 = 127.3KiB

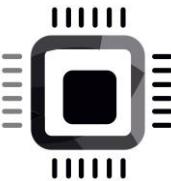




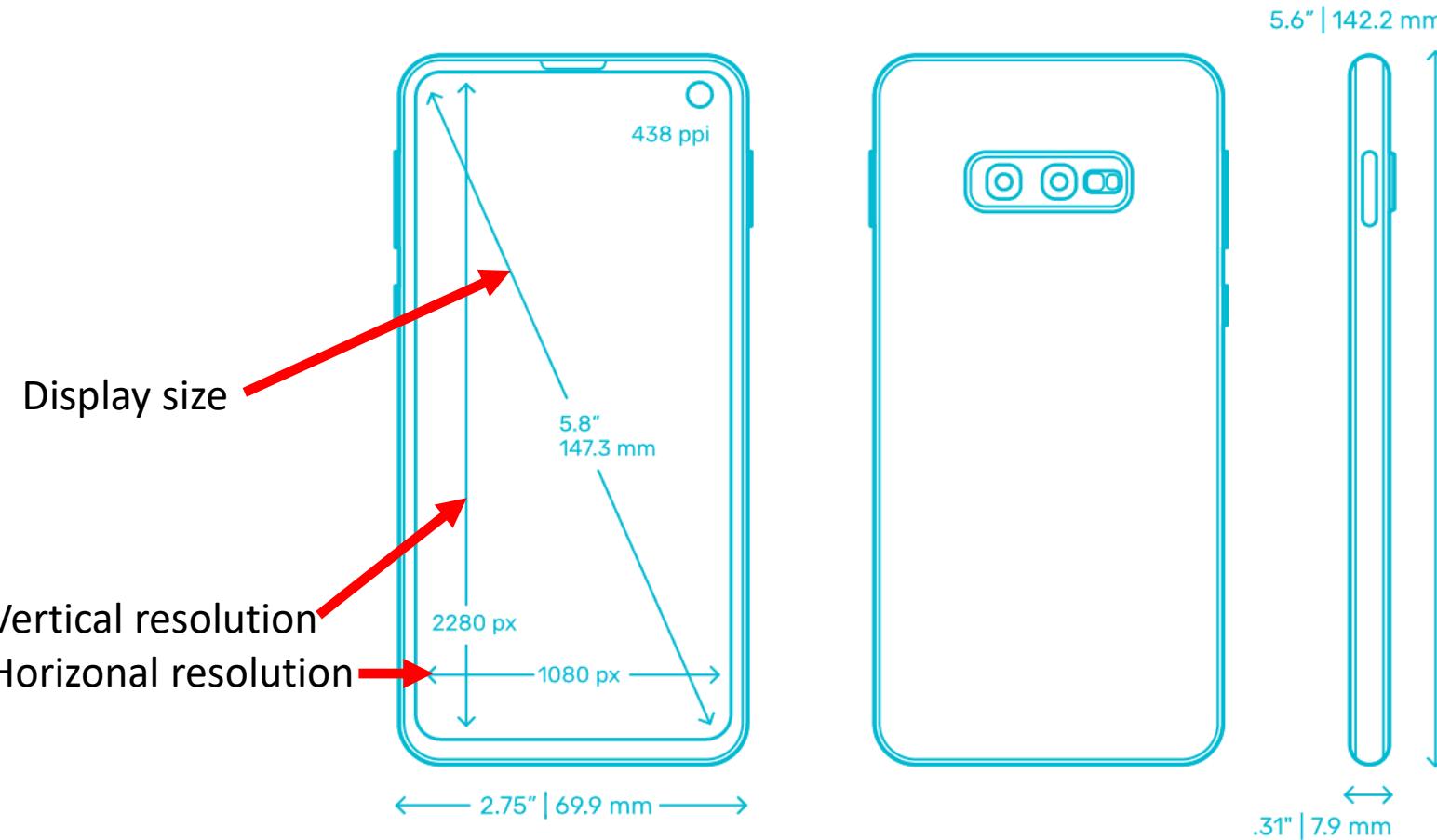
Pixel density of the display device(ppi)

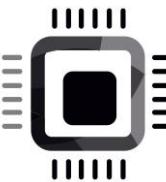
- Pixel density is measured in ppi(pixel per inch)
- It denotes how many pixels are there in per inch of the display
- More pixels/inch means more clarity of the image and texts shown on the display
- As ppi of the display increases, it is capable of showing high resolution images(more pixel information) in less space without or with less degradation of quality of the image





Pixel density of the display device(PPI)





Resolution

Image resolution:

Number of pixels used to represent an image arranged in standard graphics display width and height dimensions

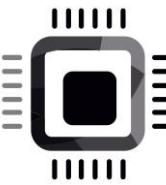
An image of 1600x900(HD+ resolution) will portray more details about the image than 1280x720(HD resolution) image

Display resolution:

Number of pixels available in a given size of the display arranged in standard graphics display width and height dimensions

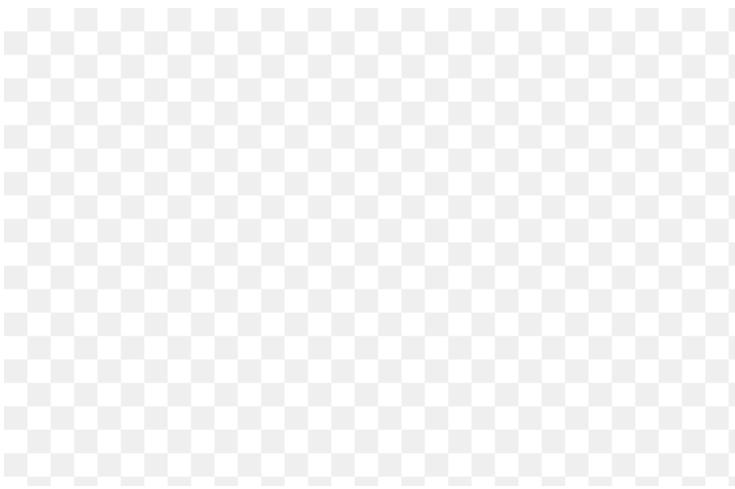
Refer the below wiki page for different graphic display resolution available

https://en.wikipedia.org/wiki/Graphics_display_resolution



What is an alpha component ?

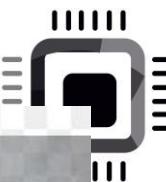
- Its an optional opacity component
- Colors with opacity are referred to as ARGB colors
- Opacity level 0 to 255 if alpha component is of 8bits
 - 255 means fully opaque with the background color
 - 0 means fully transparent with the background color



Opacity: 25%



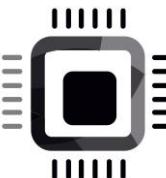
Opacity: 50%



Opacity: 75%



Opacity: 100%



How many bytes of memory does an image consume?

Given :

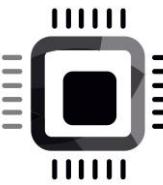
Resolution of the bitmap image : 480x270

Pixel format : ARGB8888

$$\begin{aligned}\text{Total memory consumed} &= \text{Width} \times \text{Height} \times (\text{bpp}/8) \\ &= 480 \times 270 \times (32/8) \\ &= 506.25 \text{ KiB}\end{aligned}$$



Type of file:	BMP File (.bmp)
Opens with:	Photos Change...
Location:	Downloads
Size:	506 KB (5,18,538 bytes)



BMP(Bitmap) images

1. An BMP image file contains uncompressed pixel data, hence consumes more space on the disk
 2. You can directly read the pixel values and display them on the display module



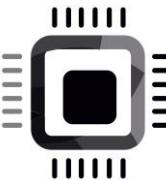
Display module showing the image

Image on disk in .bmp format

Uncompressed pixel data

00000008a	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
000000000	42	4d	8a	e9	0								00	00	7c	00
000000010	00	00	e0	01	0								00	00	03	00
000000020	00	00	00	e9	0	00	74	12	00	00	74	12	00	00	00	00
000000030	00	00	00	00	00	00	00	00	ff	00	00	ff	00	00	ff	00
000000040	00	00	00	00	00	ff	42	47	52	73	8f	c2	f5	28	51	b8
000000050	1e	15	1e	85	eb	01	33	33	33	13	66	66	66	26	66	66
000000060	66	06	99	99	99	09	3d	0a	d7	03	28	5c	8f	32	00	00
000000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000008a	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000090	80	ff	83	7d	79	ff	82	7d	79	ff	85	82	7e	ff	83	81
0000000a0	7d	ff	80	7f	7b	ff	80	7d	7a	ff	81	7d	7b	ff	82	7c
0000000b0	7b	ff	85	7f	7d	ff	84	7e	7b	ff	81	7c	78	ff	7f	7b
0000000c0	7a	ff	7f	7c	7d	ff	7d	7b	7b	ff	78	75	76	ff	75	75
0000000d0	73	ff	71	73	6e	ff	73	74	71	ff	7b	7c	7a	ff	82	83
0000000e0	81	ff	80	80	80	ff	7f	80	81	ff	80	7d	7f	ff	7c	7e
0000000f0	81	ff	7a	7d	81	ff	7a	7d	81	ff	7a	7d	81	ff	7a	7e
000000100	7f	ff	7b	80	7f	ff	7b	80	7f	ff	7b	80	7f	ff	7b	7f
000000110	80	ff	7b	7f	80	ff	7b	80	7a	ff	7c	7f	7c	ff	7c	80
000000120	7e	ff	7d	7f	7f	ff	7d	80	7e	ff	7e	81	7e	ff	7e	82
000000130	81	ff	7d	81	83	ff	7c	8	81	ff	7d	82	81	ff	80	84
000000140	83	ff	7c	81	80	ff	7b	81	83	ff	7a	7f	7c	ff	79	7e
000000150	7b	ff	79	7e	7b	ff	77	7a	ff	77	7d	7a	ff	77	7d	
000000160	7b	ff	76	7b	7a	ff	77	7c	7a	ff	77	7c	7b	ff	74	78
000000170	79	ff	74	78	7a	ff	7d	7f	77	7b	7d	ff	75	79		
000000180	7b	ff	75	78	7c	ff	79	7d	ff	76	79	7d	ff	7a	7d	
000000190	80	ff	7d	81	83	ff	80	82	ff	7c	7f	83	ff	7c	7f	
0000001a0	83	ff	7d	80	8	ff	81	86	ff	80	83	87	ff	86	8a	
0000001b0	8d	ff	8b	8f	8c	90	93	ff	8d	91	95	ff	8d	91		
0000001c0	94	ff	8f	93	90	93	96	ff	93	96	9b	ff	95	99		
0000001d0	9d	ff	99	99	ff	9d	a1	a6	ff	9d	9f	a5	ff	9d	9f	
0000001e0	a5	ff	9f	a2	ff	a0	a4	a6	ff	a2	a6	a8	ff	a6	a7	
0000001f0	a9	ff	a9	a8	aa	ff	ac	aa	ff	ae	aa	a8	ff	af	ab	
000000200	a9	ff	ee	aa	a8	ff	aa	a7	ff	a1	a4	a1	ff	9f	a2	
000000210	9f	ff	9d	a1	9f	ff	9d	a0	9f	ff	9c	9f	9e	ff	9b	9e
000000220	9d	ff	9c	9f	9e	ff	9c	9f	9e	ff	99	9b	9b	ff	96	99

Image on disk in .bmp format



You cannot directly read and display the .jpg file contents on the display module

00000000	ff d8 ff e0 00 10 4a 46 49 46 00 01 01 01 01 2c	y0yà..JFIF.....
00000010	01 2c 00 00 ff e1 00 16 45 78 69 66 00 00 4d 4d	...yá..Exif..MM
00000020	00 2a 00 00 00 08 00 00 00 00 00 ff db 00 43	*.....y0.c
00000030	00 03 02 02 02 02 02 03 02 02 02 03 03 03 03 04
00000040	06 04 04 04 04 04 08 06 06 05 06 09 08 0a 08 09
00000050	08 09 09 0a 0c 0f 0e 0a 08 0e 0b 09 09 0d 11 0d
00000060	0e 0f 10 10 11 10 0e 0c 12 13 12 10 13 0f 10 10
00000070	10 ff db 00 43 01 03 03 03 04 03 04 08 04 04 08	.y0.c.
00000080	10 0b 09 0b 10 10 10 10 10 10 10 10 10 10 10 10
00000090	10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
000000a0	10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
000000b0	10 10 10 10 10 ff c0 00 11 08 01 68 02 80 03yÁ...h.e.
000000c0	01 22 00 02 11 01 03 11 03 54 04 00 1d 00 00 02	.*.....yA.
000000d0	02 03 01 01 01 00 00 00 00 00 00 00 00 06 05
000000e0	04 07 03 08 02 00 01 09 44 10 00 01 03yA.D.
000000f0	03 03 03 03 02 04 05 02 05 03 05 01 02 03
00000100	04 05 11 00 06 21 12 37 13 51 22 61 14 32!1.A.Q'a.2
00000110	71 08 15 81 23 42 91 16 24 33 c1 62 d1	q...#B;Rt.S3ÁhN
00000120	43 e1 92 f0 34 72 17 55 62 53 63 a2 ff c4	cÁ.4r.R.%,ScyÁ
00000130	00 18 01 01 00 02 00 00 00 00 00 00 00 00 00
00000140	00 00 00 00 01 03 06 07 ff c4 00 3a 11yA.:
00000150	00 01 03 02 04 03 04 04 02 01 04 02 03 00 00
00000160	01 00 02 03 11 22 12 41 51 05 61 71 22 f0!1.AQ.aq"ô
00000170	91 13 a1 81 b1 14 e1 06 f1 23 42 52 a2	'.; tñÁ2.å.R#Rc
00000180	62 33 72 15 b? da 00 0c 03 01 00 02 11	b3r.ºØyÚ.....
00000190	03 11 00 00 00 00 00 00 00 00 00 00 00 00 00	...?uaz h.Eó#sP
000001a0	db 64 71 b2 20 49 71 45 29 41 b0 f3 98	Ódq"u.v(jgE)A"ô"
000001b0	94 1a 69 b2 a3 a6 11 62 52 d4 f5 db 52 b7	".i"uc;.brôSÔR.
000001c0	56 7d 45 20 61 41 47 23 27 9b c8 85 a2 f6	vjE("žaAc#",i.ôçô
000001d0	be 3d 53 f3 b8 08 f3 90 45 54 d6 9b 88	¶Gso",ô,ô EMÖ."
000001e0	c1 a2 a6 4d f6 cc 53 e9 e8 71 24 c7 40 49	Áç.ºyáiséeqScøI
000001f0	38 d7 c0 4d a1 29 47 c9 5d 4e 76 fc 17 1b 63	Sø-i-)G€]Nvu..c
00000200	9a 52 4d 00 28 84 d5 0f c4 a7 ad 92 0a 80 56	SRHñe(,ô,AS-'..v
00000210	4c 54 4d a1 4c 21 36 0a 1d 71 b4 6a 73 8e 36	Lrsb;Ll16..q'jážé
00000220	16 4d 34 c6 07 34 92 2f 63 63 82 d0 ec d0	.GMôME.4"/cc,BiD
00000230	12 4d 34 51 21 92 f7 33 ce f9 f6 a1 49 9c	.2xéQj'-'3luo;Ie
00000240	ca d2 0f 4c 79 15 31 c3 5f ca 05 24 0b 90	Éò'r.Ly.1A.É.S.
00000250	33 e3 ct bc fa 42 ad 7b 0d af 98 86 9d 55 53 51	3aiwQb(.--t USQ
00000260	44 85 c5 98 cd 86 97 f8 86 65 a6 3c cb ee 7c 98	D.Áít-óte<Éil"
00000270	b0 f7 cc ff 00 b3 99 0f 97 5f 7a de c3 32 b8 c3	"-iy."m -_zþÅ2,Å
00000280	70 da f4 a3 00 a3 ef 8d 00 e6 9a 88 c9 46 93 1d	pôéz.tz .æ"EF".

.jpg image on disk



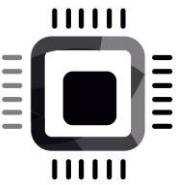
JPEG Decoder

Bitmap
data

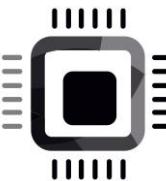


Display module

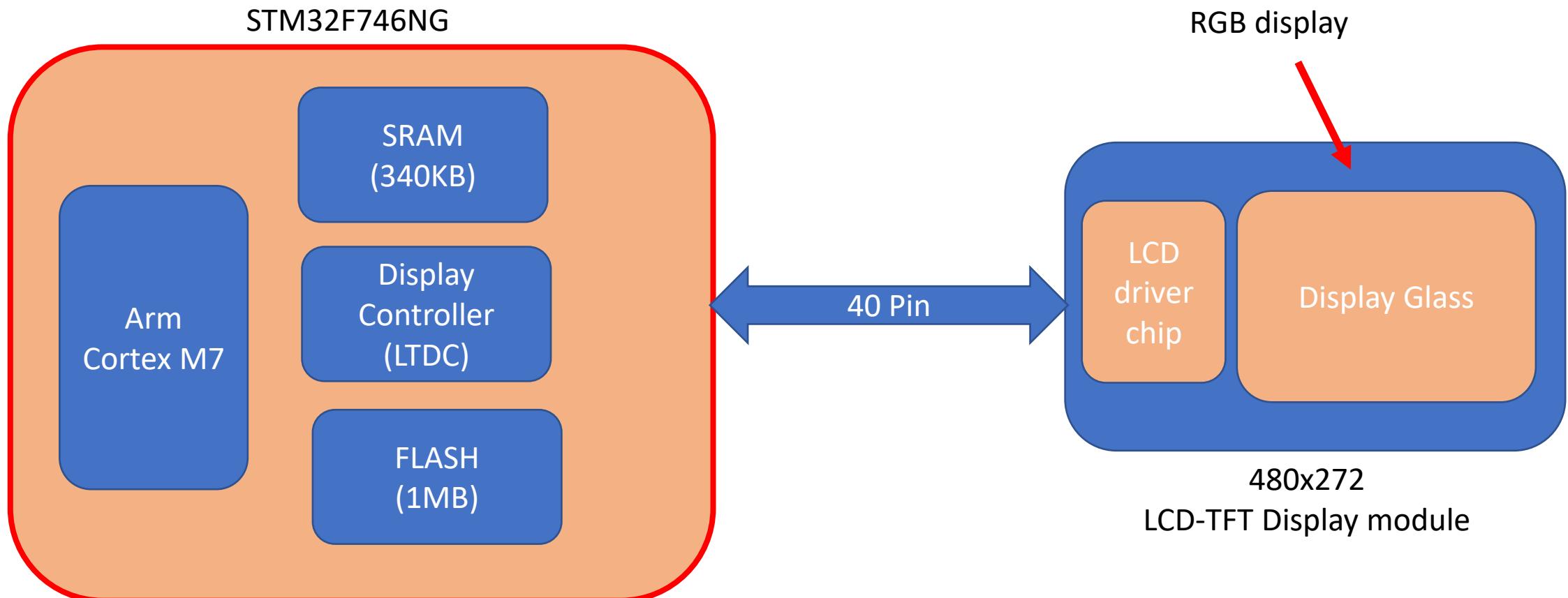
- 1) JPEG decoder middleware
- 2) JPEG decoder peripheral of the MCU



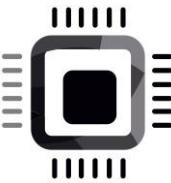
Storing images in the application



Embedded graphics system



32F746GDISCOVERY Discovery kit

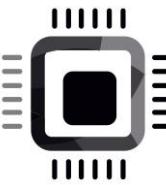


Frames



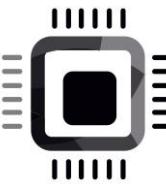
.png

Dimensions 480 x 270
Width 480 pixels
Height 270 pixels
Bit depth 32



Storing images in the application

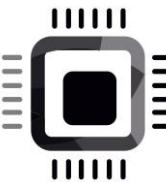
1. Convert the images to bitmap ‘C’ array (pixel value array) with desired pixel format
2. Store them in internal flash memory
3. Use appropriate pixel format



Selecting pixel format for your application

Depends on ,

- 1) Pixel format support by the LCD driver chip
- 2) Desired color range
- 3) Microcontroller RAM space availability
- 4) Microcontroller Flash space availability



LCD Driver : ILI9341

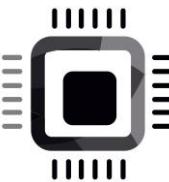
Supports RGB565/RGB666/RGB222 Pixel formats only

RGB565

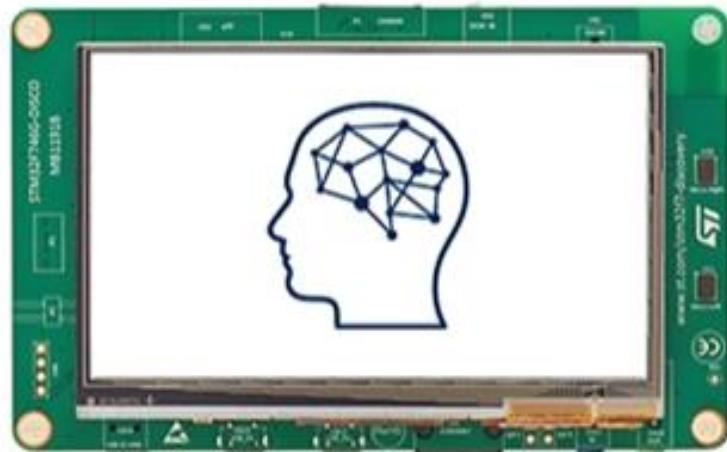
- 1 pixel consumes 2 bytes
- Total flash consumed for 3 frames = $(320 \times 240 \times 2) * 3 = 450\text{KiB}$

Total On-chip Flash: 2MB

Total on-chip RAM : 256KB



32F746GDISCOVERY



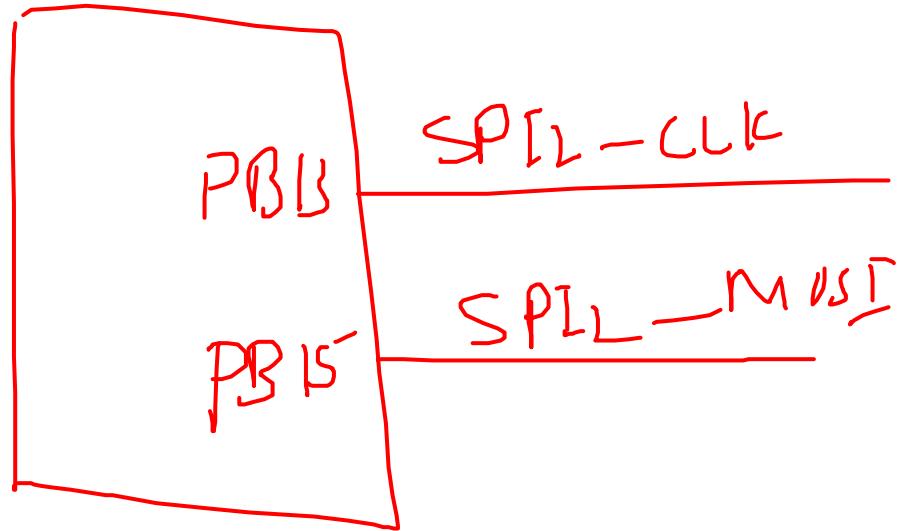
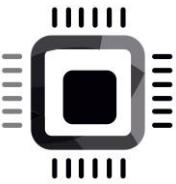
Total on-chip Flash : 1MB

LCD Driver : OTA5180A

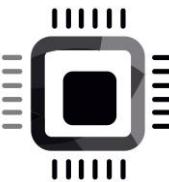
Supports up-to RGB888 Pixel formats

RGB888

- 1 pixel consumes 3 bytes
- Total flash space consumed for 3 frames =
 $(480 \times 270 \times 3) * 3 = 1139\text{KiB}$



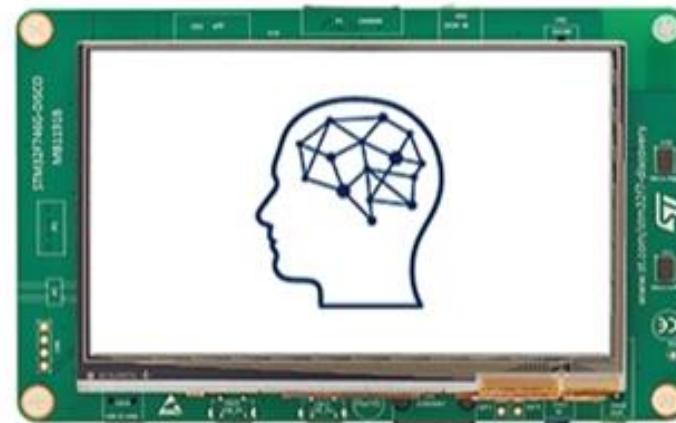
http://www.lcdwiki.com/2.4inch_SPI_Module_ILI9341_SKU:MSP2402



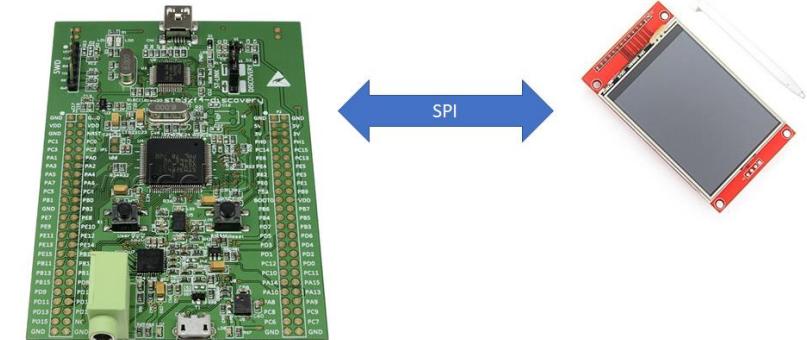
32F429IDISCOVERY

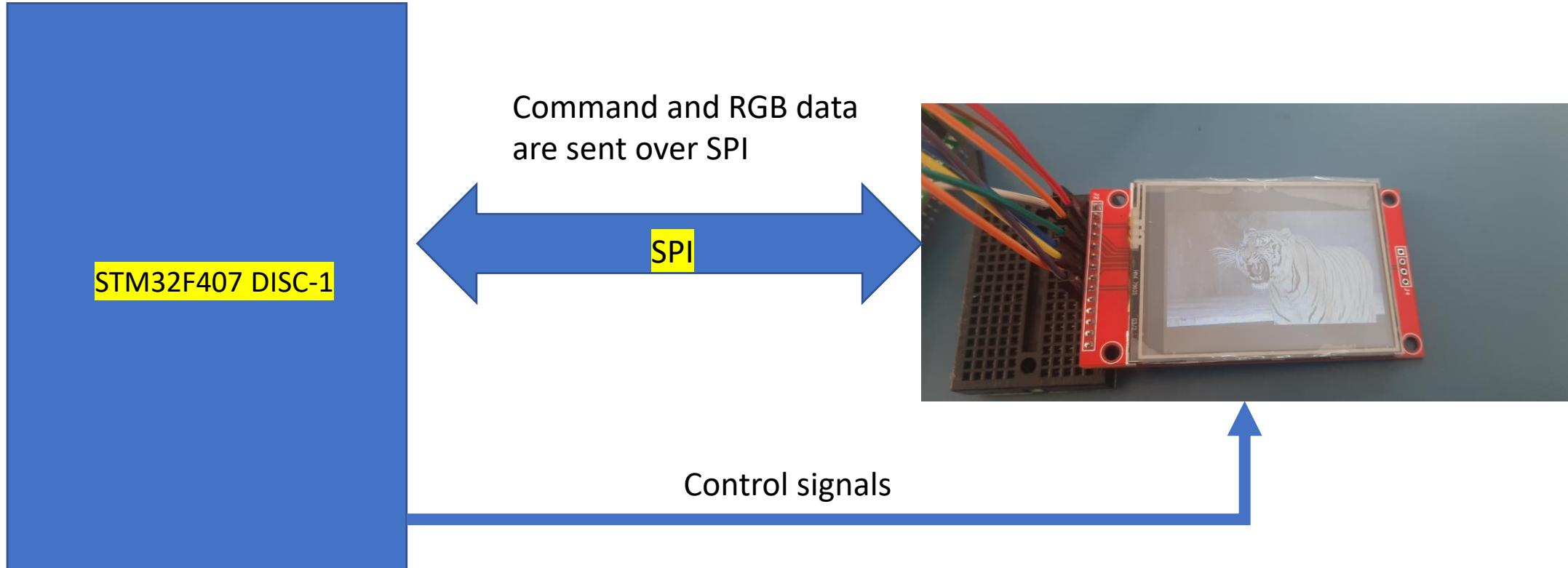
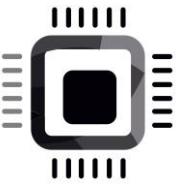


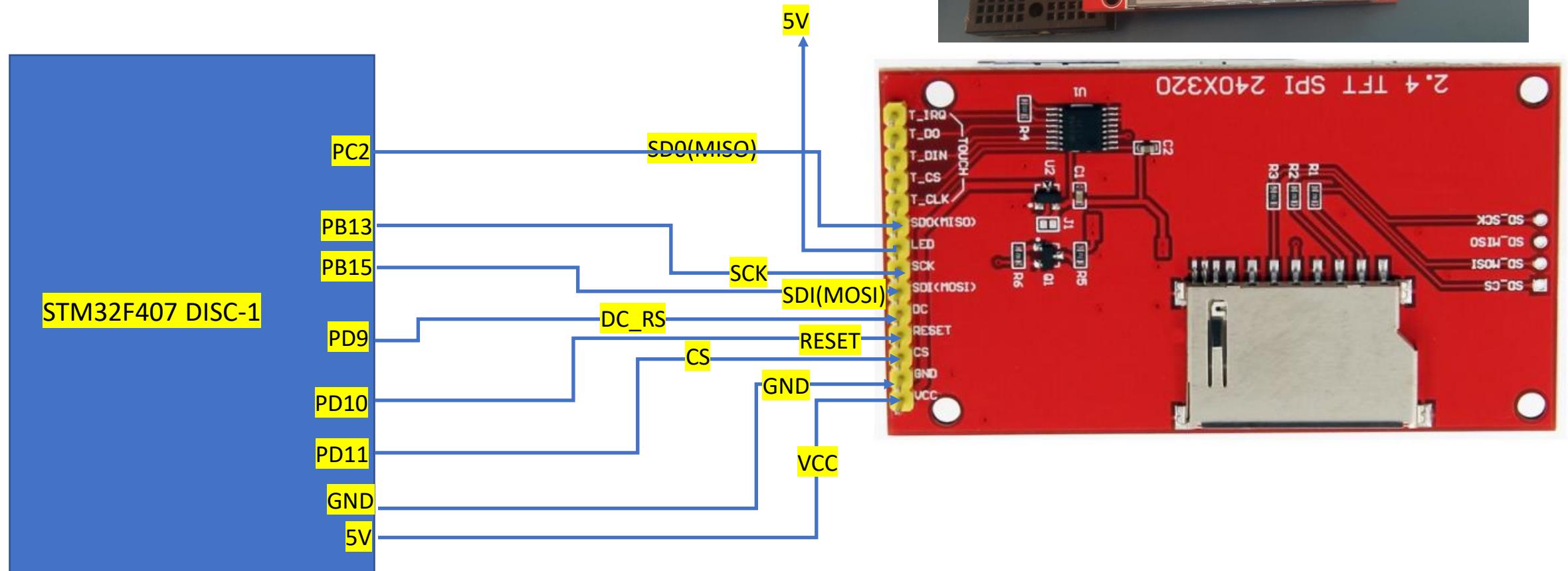
32F746GDISCOVERY



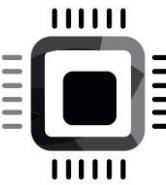
32F407DISCOVERY + SPI
based LCD module







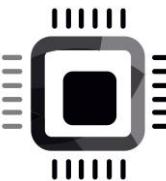
STM32F407 DISC-1 board with an external RGB display interfacing



LTDC of the ST Microcontroller

Configure and enable the LTDC peripheral (LCD-TFT Display Controller) of the microcontroller

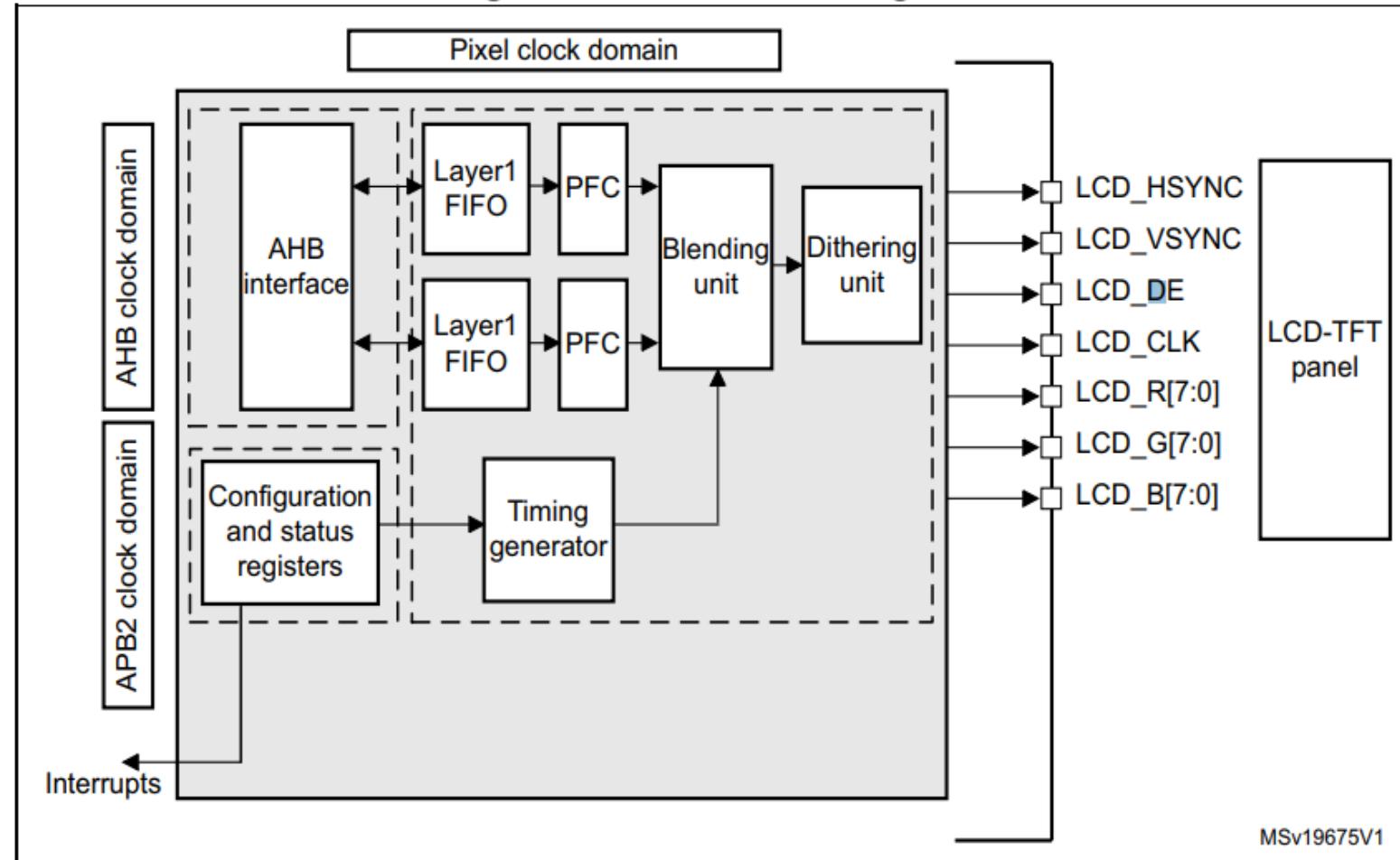
- LTDC of the MCU generates all synchronization and timing signals and transfers RGB components to the display.
- Display which has a display driver chip that interprets those signals and drives the display panel to light the desired pixels

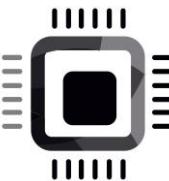


18.3.1 LTDC block diagram

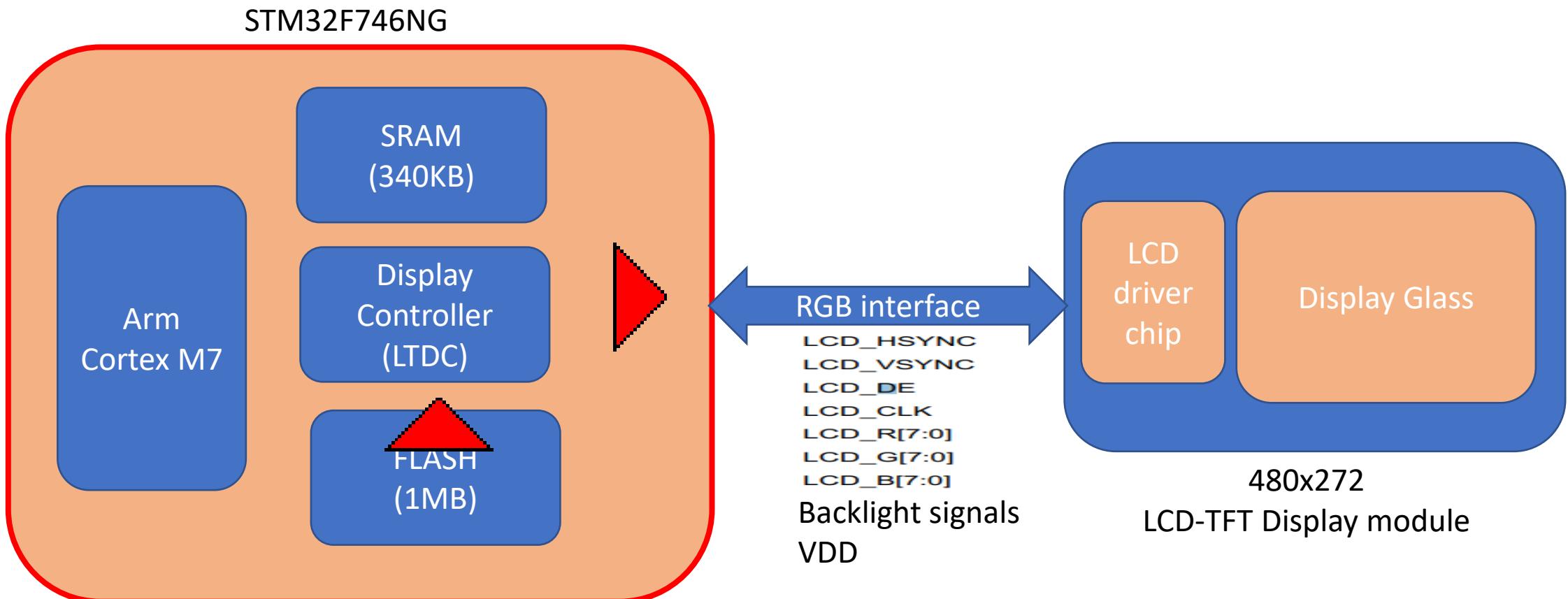
The block diagram of the LTDC is shown in [Figure 111: LTDC block diagram](#).

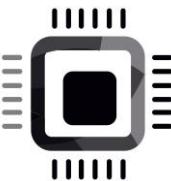
Figure 111. LTDC block diagram



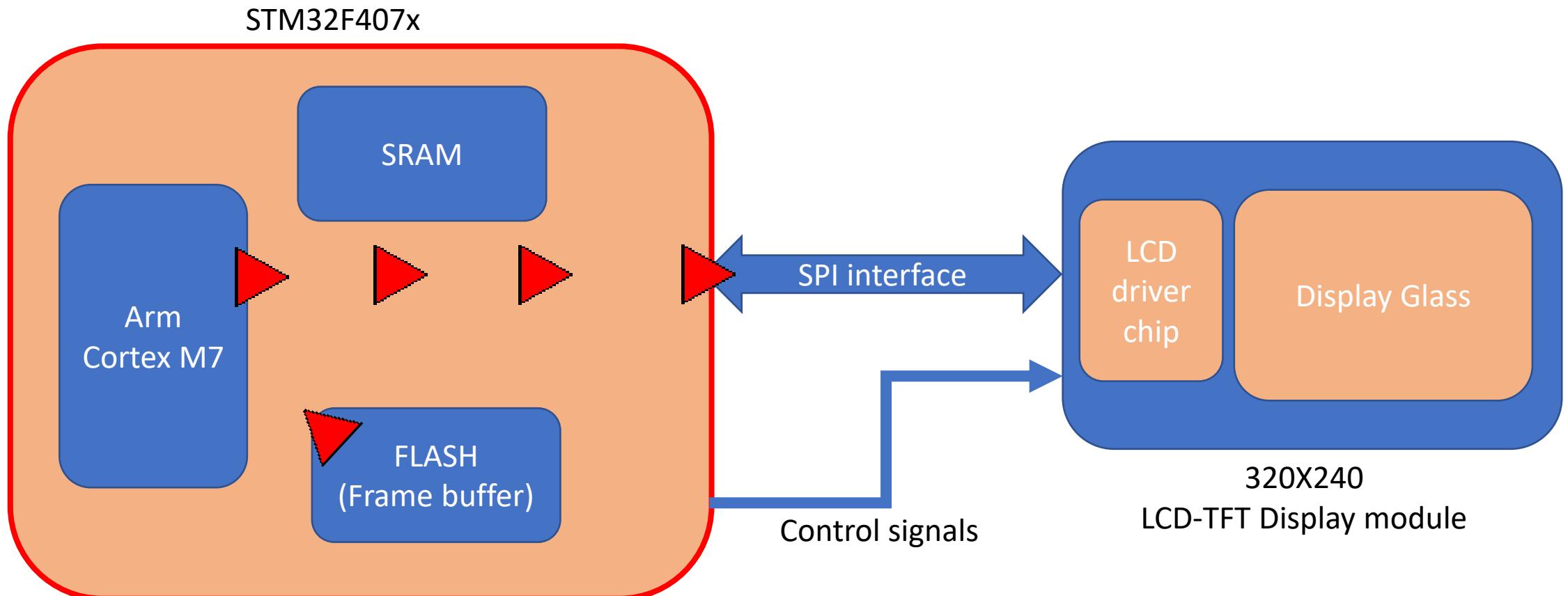


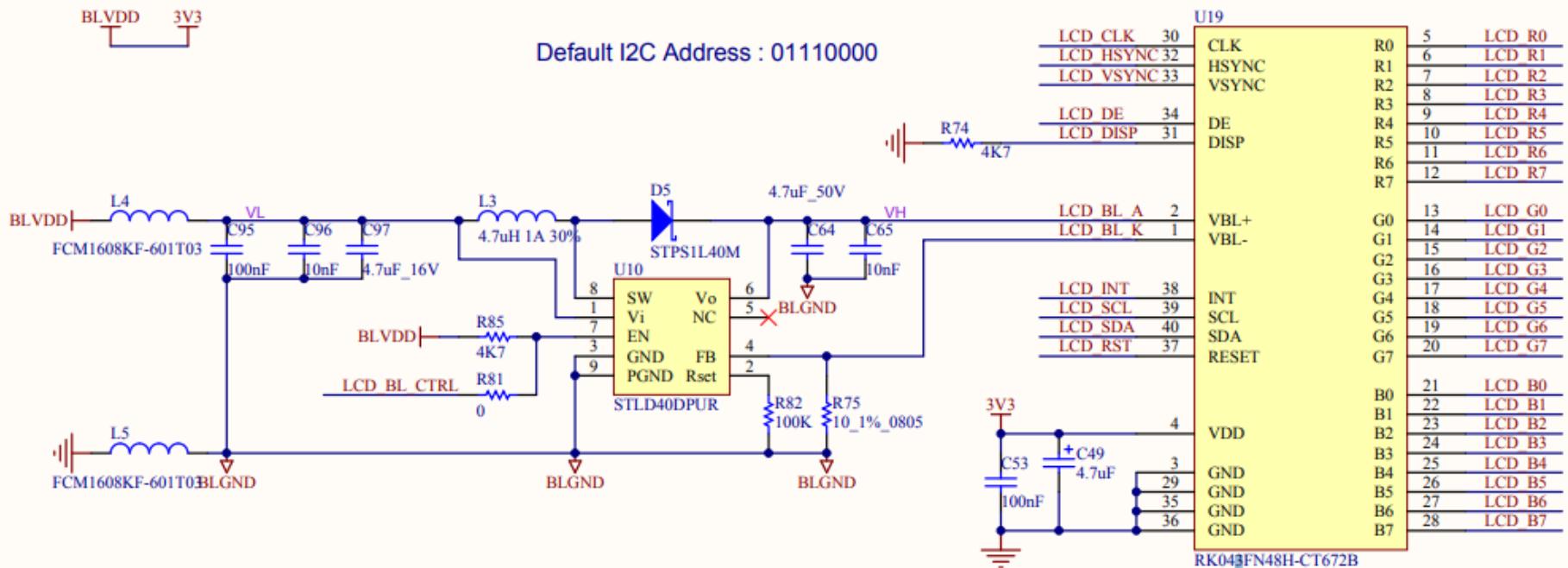
Embedded graphics system



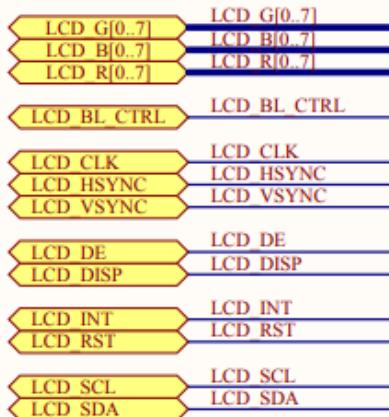


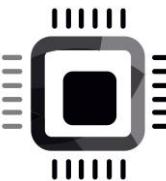
Embedded graphics system





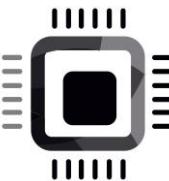
Backlight driver & PFC of LCD panel





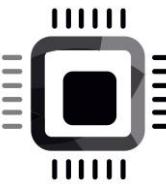
Some math

- Converted ARGB8888 pixel format image to RGB565 pixel format ‘C’ array
- Pixel depth = 16bpp(2 bytes)
- Array size/frame = $480 \times 270 \times 2 \rightarrow 253.125 \text{ KiB}$
- Array size of 3 frames $\rightarrow 759.375\text{KiB}$
- 759.375KiB of flash is used up to hold just 3 frames .
- Let's check the .map file of the project to verify the memory consumption



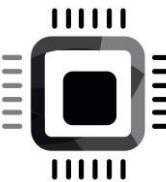
Display interface types

- MIPI DPI (Display Pixel Interface)
- MIPI DBI (Display Bus Interface/MCU interface)
- MIPI DSI (Display Serial Interface)



MIPI(Mobile Industry Processor Interface) Alliance

- MIPI Alliance develops interface specifications for mobile and mobile-influenced devices.
- In the mobile industry, companies use MIPI Alliance specifications when developing smartphones, tablets, laptops, and hybrid devices



MIPI DPI

- This is also called as RGB interface
- Applies to display interface which uses 16/18/24-bit data lines and control signals
- The MIPI DPI specification standardizes the data and controls signals to be used by manufacturers of mobile device processors, cameras, and display
- The DPI interface is typically used when the display module doesn't support the frame buffer (GRAM). The Host controller must stream the data in real-time to the display



The host processor interface must implement a 24-bit data width and accompanying control and timing signals .
The host processor must be capable of transferring data as 16/18/24 bit words .

The display module interface must be implemented with a 16-bit or 18-bit or 24-bit data width

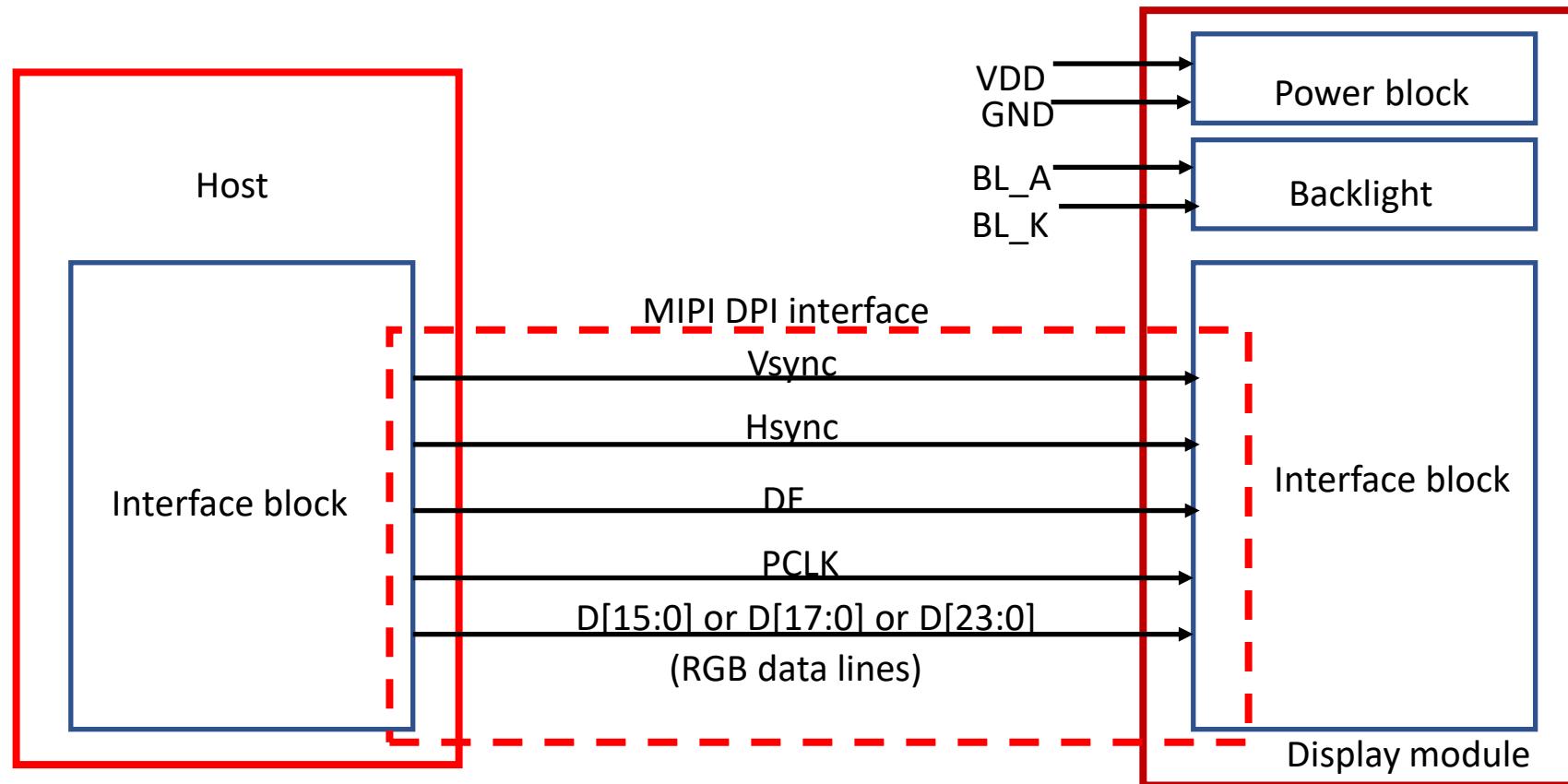
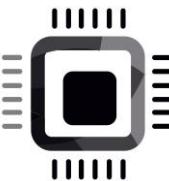
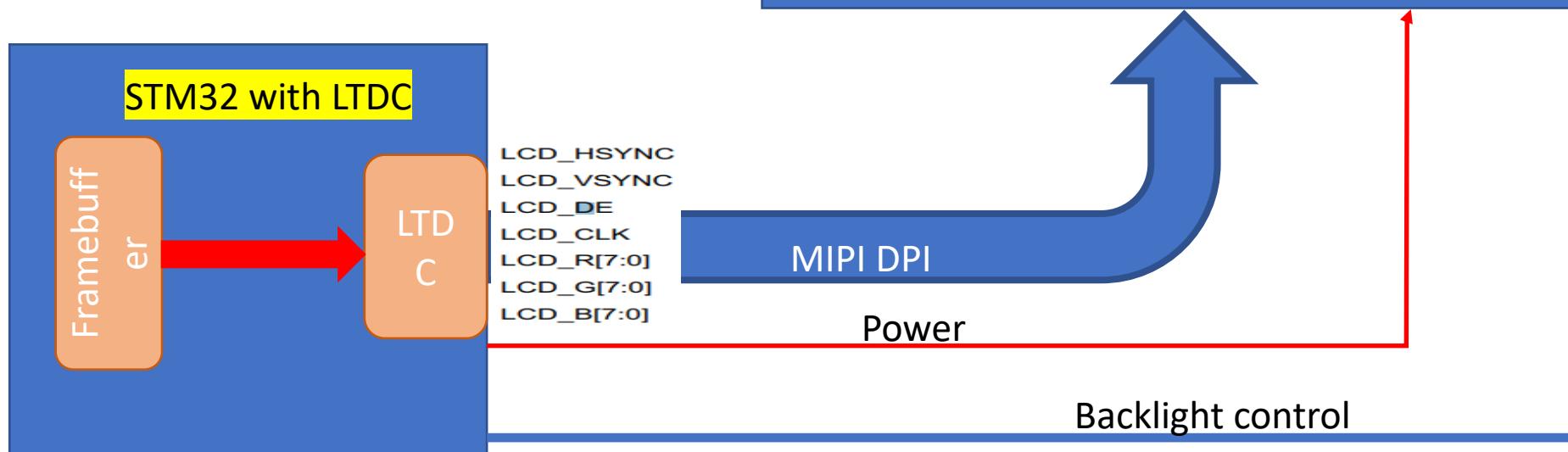
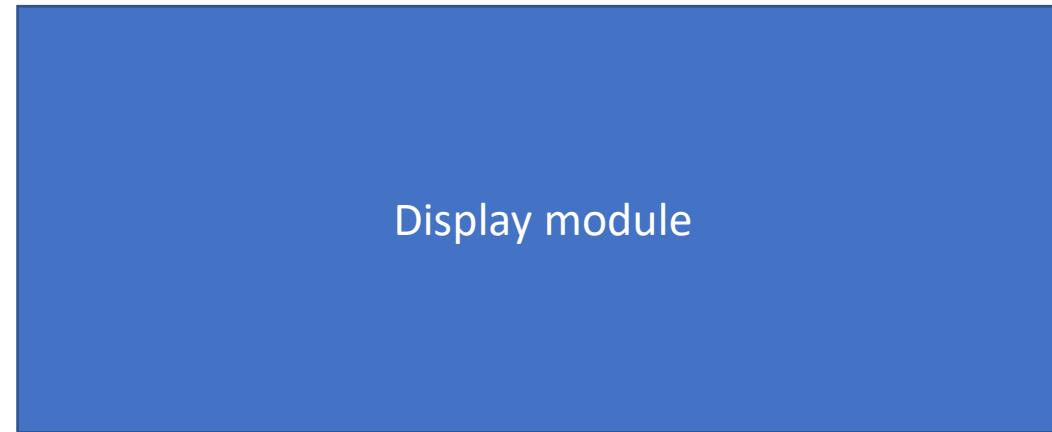
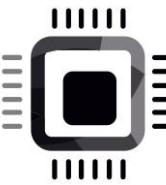


Figure shows a display module connected to the Host(Microcontroller) using MIPI DPI interface



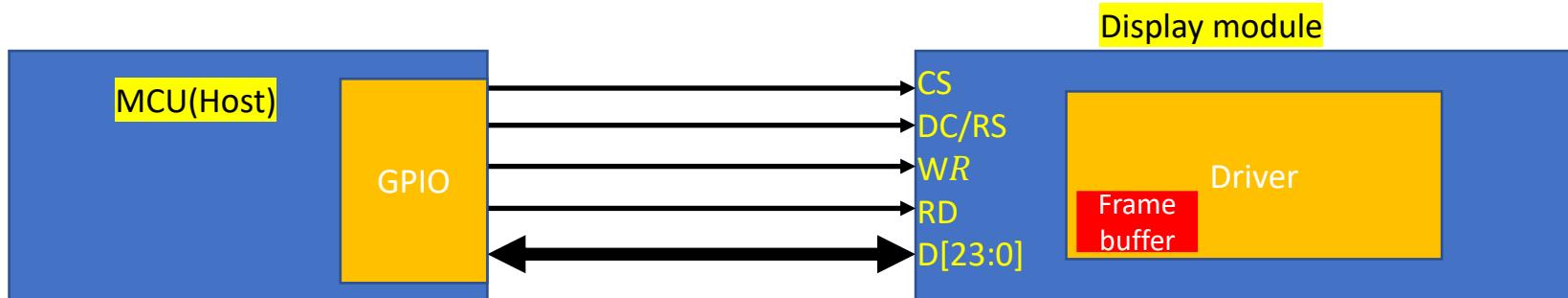
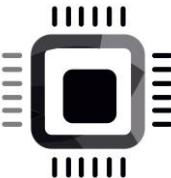
LTDC generates all the required synchronization and data signals as per the programmed parameters in its registers



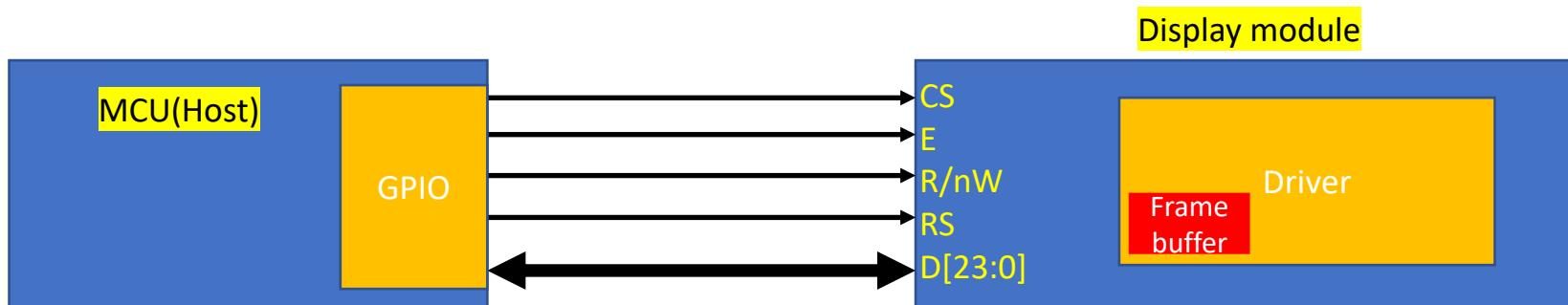


MIPI DBI(Display Bus Interface)

- Also known as MCU interface
- The MIPI-DBI is used to interface with a display module with an integrated graphic RAM (GRAM). The pixel data is first updated in the local GRAM of the display driver chip which repeatedly refreshes the display
- Host and display module can be connected by simple GPIOs
- Types of MIPI-DBI are:
 - DBI Type A: based on Motorola 6800 bus
 - 8/9/16/18/24 bit parallel data transmission is possible
 - DBI Type B: based on Intel® 8080 bus
 - 8/9/16/18/24 bit parallel data transmission is possible
 - DBI Type C: based on SPI protocol
 - 3 or 4 line SPI interface

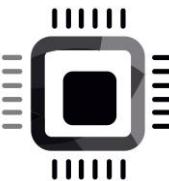


8/9/16/18/24-bit parallel MCU interface (DBI Type A or Intel 8080)

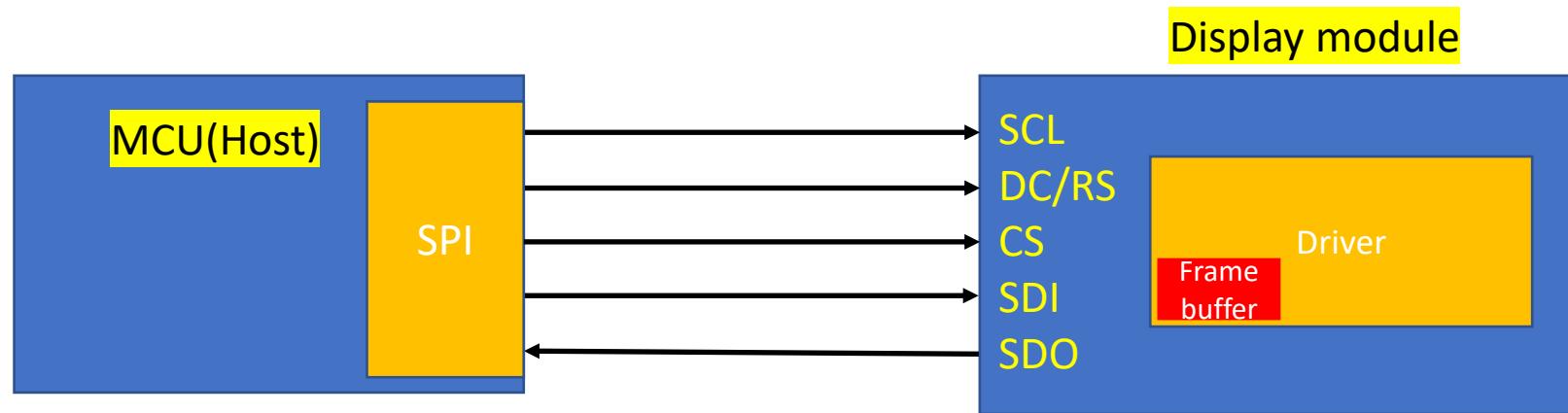


8/9/16/18/24-bit parallel MCU interface (DBI Type B or Motorola 6800)

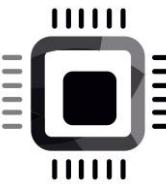
Using Type A/B/C interface , MCU first sends commands to the LCD driver, configures some important LCD parameters, and then writes data to the integrated GRAM to display it



4 Line DBI serial interface



8/9/16/18/24-bit serial MCU interface (DBI Type C)



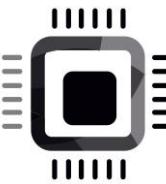
RGB Interface signals

Vsync :

- This is a vertical synchronization signal sent from the host display controller(LTDC) to the display module
- This signal marks the “start(beginning) of a new frame”. That means, when this signal asserts, the display module understands that the host controller is going to send a new frame

Hsync:

- This is a horizontal synchronization signal sent from the host display controller(LTDC) to the display module
- This signal marks the “start of a new line of the frame.” That means, when this signal asserts, the display module understands that the host controller will send a new line of the current frame.



RGB Interface signals

DE :

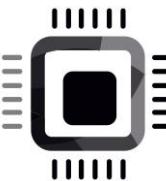
- This is sent from the host controller(LTDC) and indicates whether the RGB data is valid or not.
- When DE = 0, The display module doesn't read the RGB data because it is invalid
- When DE = 1 , the display module reads the RGB data and displays it

DOTCLK(PCLK):

- The clock signal is sent from the host controller(LTDC) to read the RGB data when DE = 1; Display module reads the RGB data during the rising edge of DOTCLK. This also indicates how fast RGB data is made available to the display module.

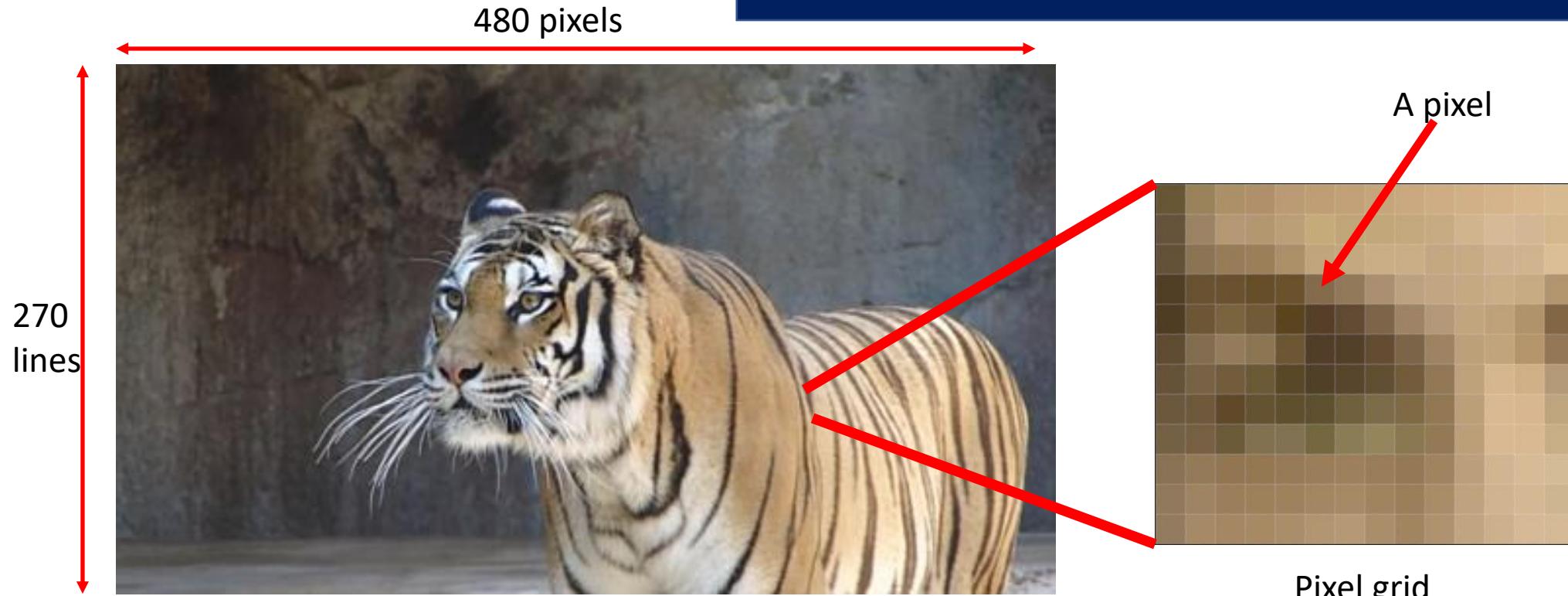
RGB data lines :

- The host controller must support 24bit data lines to carry RGB data. The display modules samples these data lines only when DE=1 during rising edge of DOTCLK. Note that the display module does not need to support 24 data lines to accept RGB data. You have to check the display module's pin capability while interfacing.



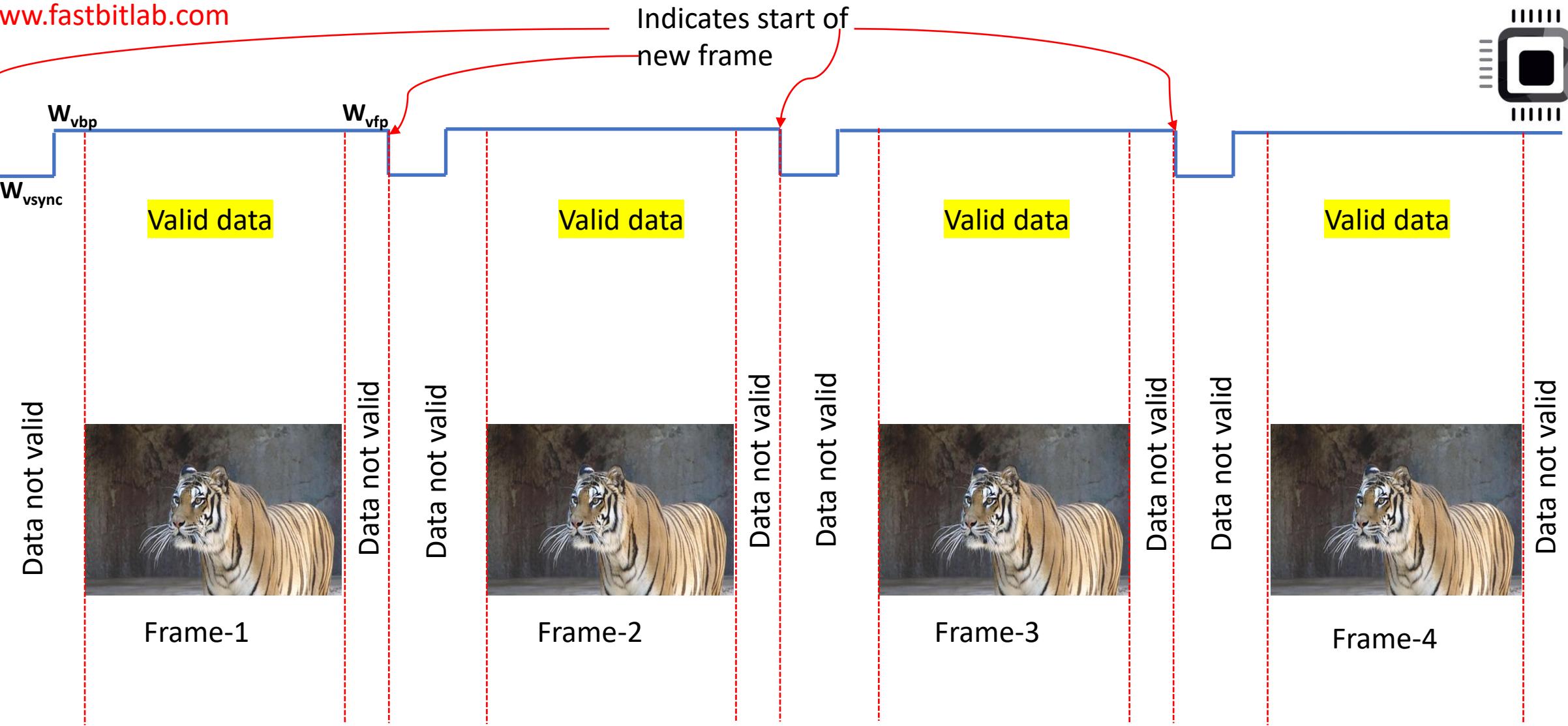
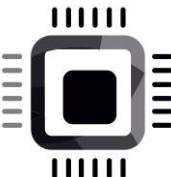
Pixel(Picture element)

Pixel of the image : A smallest information(color information) of the image stored in computer memory



Dimensions 480 x 270
Width 480 pixels
Height 270 pixels
Bit depth 32

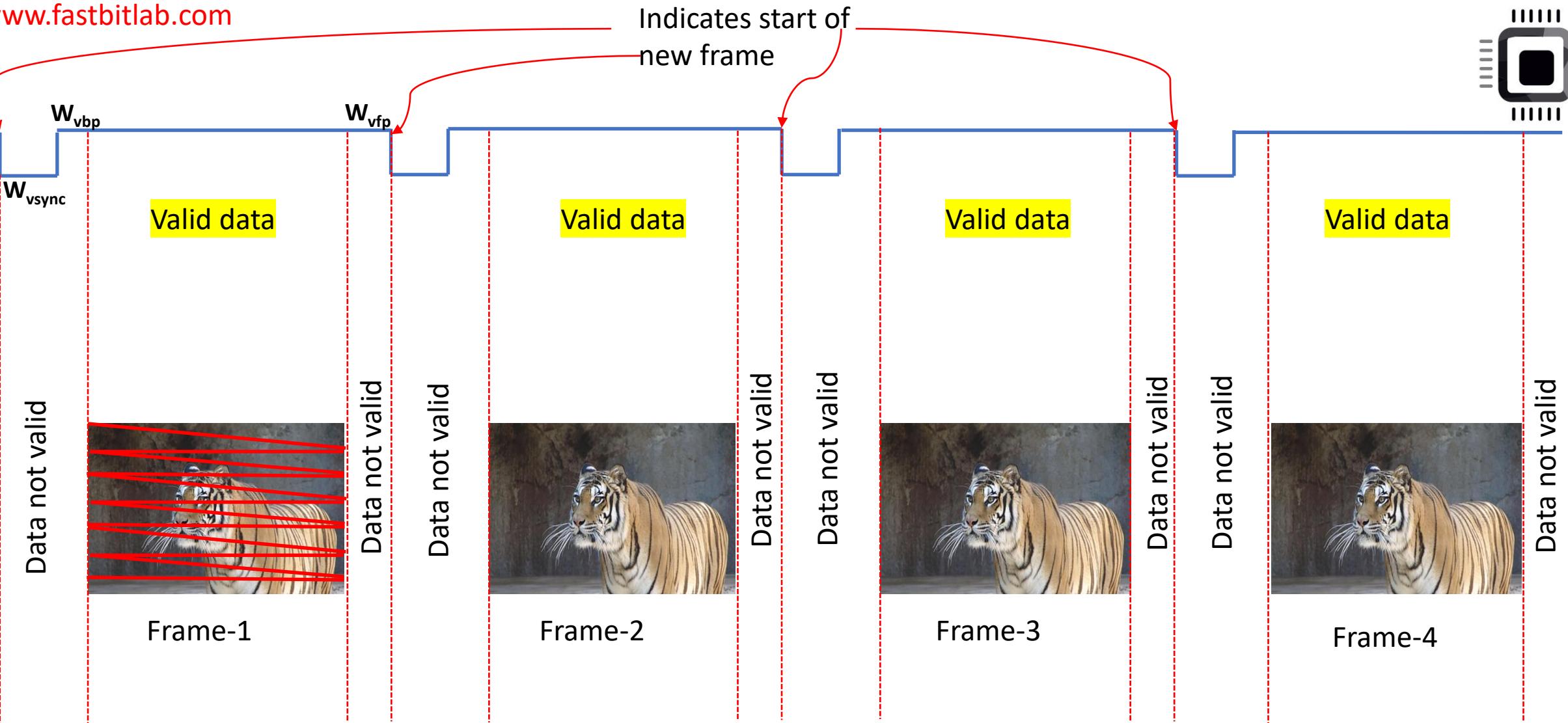
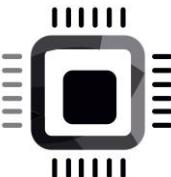
A color image



W_{vsync} → Width of the Vsync strobe

W_{vfp} → Width of the vertical front porch

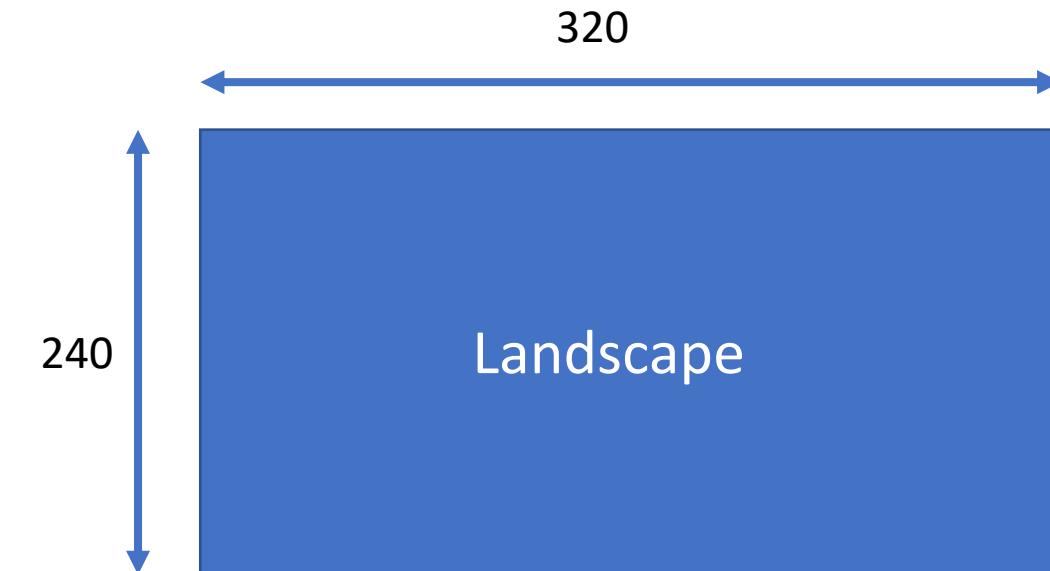
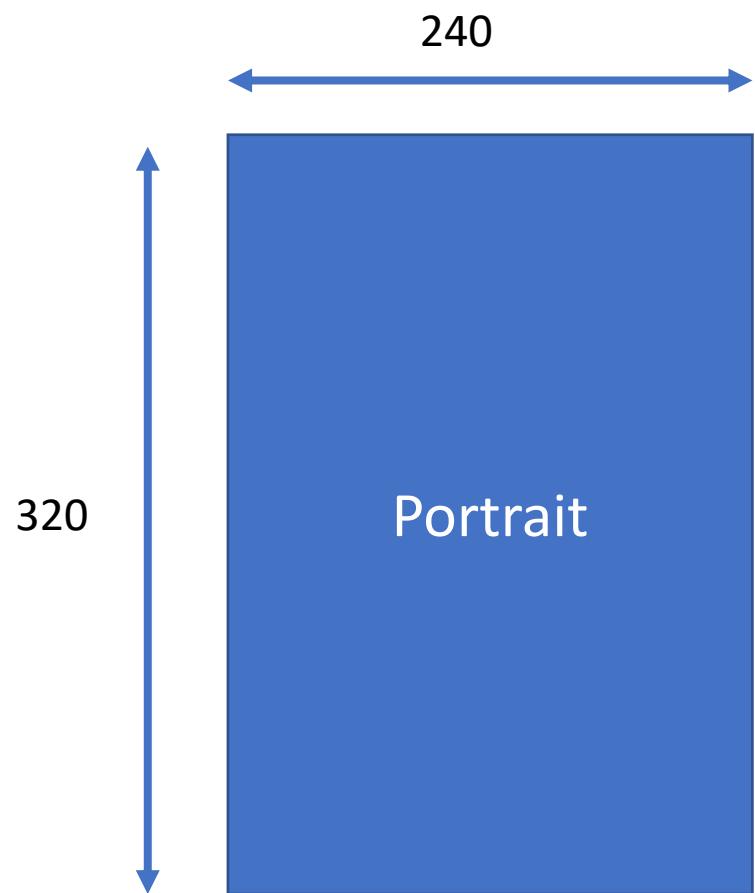
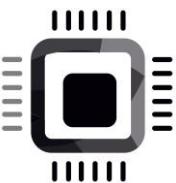
W_{vbp} → Width of the vertical back porch

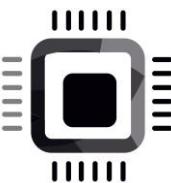


W_{vsync} → Width of the Vsync strobe

W_{vfp} → Width of the vertical front porch

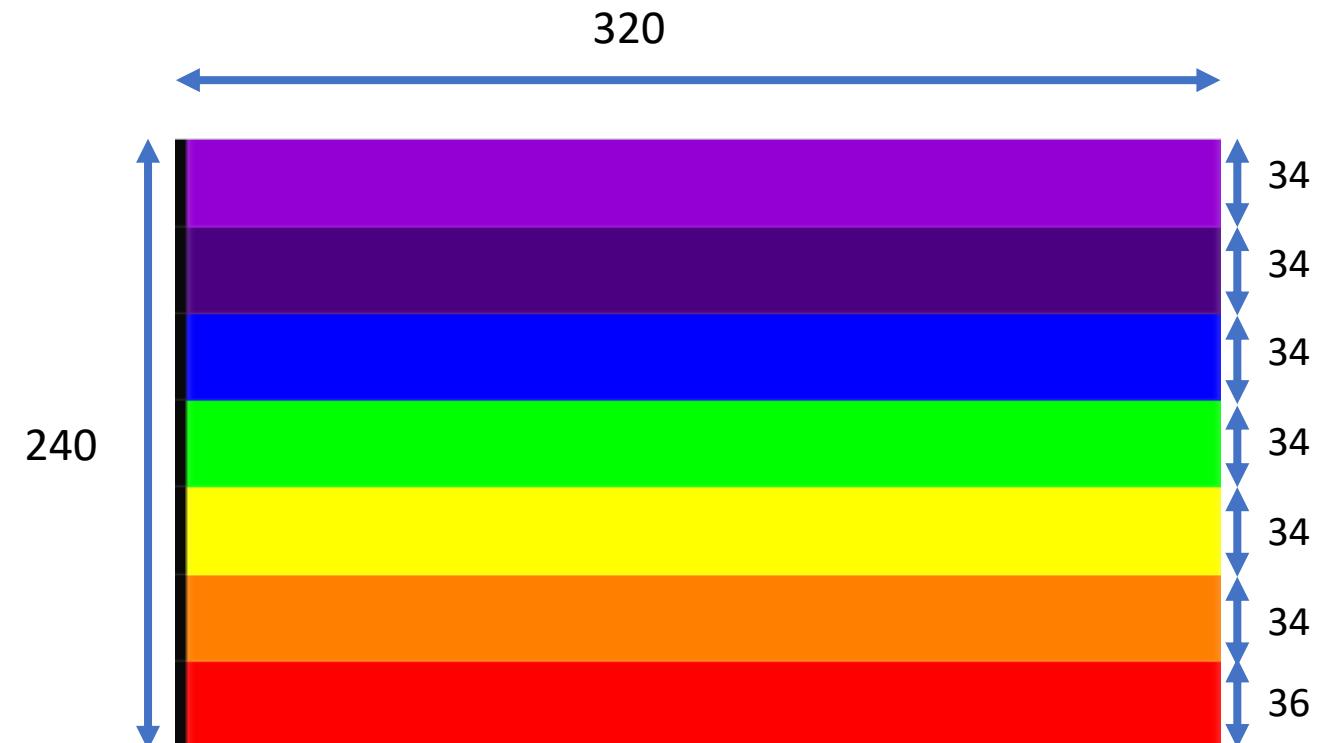
W_{vbp} → Width of the vertical back porch

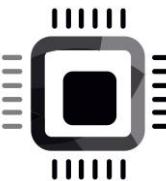




Exercise 001 : Display VIBGYOR bars

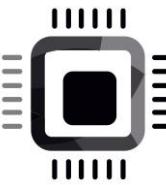
Color	RGB Value
Violet	148, 0, 211
Indigo	75, 0, 130
Blue	0, 0, 255
Green	0, 255, 0
Yellow	255, 255, 0
Orange	255, 127, 0
Red	255, 0 , 0





Steps

- 1) Create a new project for your microcontroller or board using stm32cubeide
- 2) Setup the main system clock
- 3) Setup AHB and APB clocks
- 4) Setup pixel clock (DOTCLOCK) (**Not required for STM32F407x-DISC1 board with external LCD**)
- 5) Configure the SPI peripheral (**Not required for STM32F746-DISC board**)
- 6) Setup framebuffer in RAM
- 7) Initialize the LCD module by sending LCD commands over SPI (**Not required for STM32F746-DISC board**)
- 8) Configure and enable the LTDC peripheral (**Not required for STM32F407x-DISC1**)
- 9) Send frame buffer contents to LCD over SPI (**Only in the case of STM32F407x-DISC1**)
- 10) Make sure that all error interrupts enabled and ISRs implemented

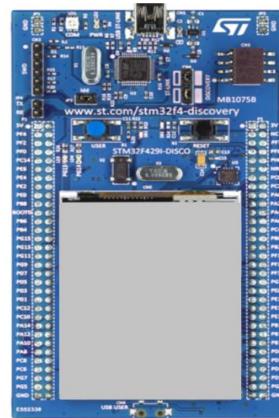


Setup main system clock(SYSCLK)

We will configure SYSCLK for the maximum value the microcontroller can achieve

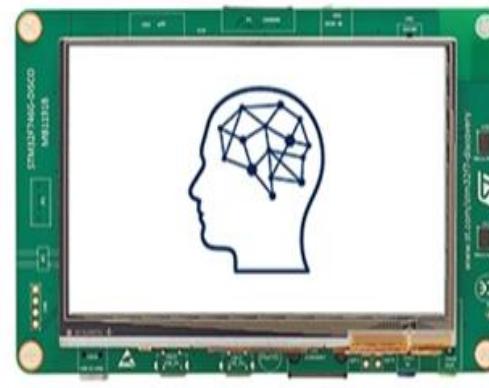
Maximum SYSCLK speed for different microcontrollers

32F429IDISCOVERY



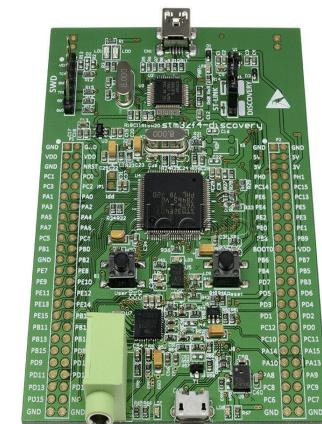
180MHz

32F746GDISCOVERY



216MHz

32F407DISCOVERY



168MHz

STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced

152 / 1751

156%

-

+

□

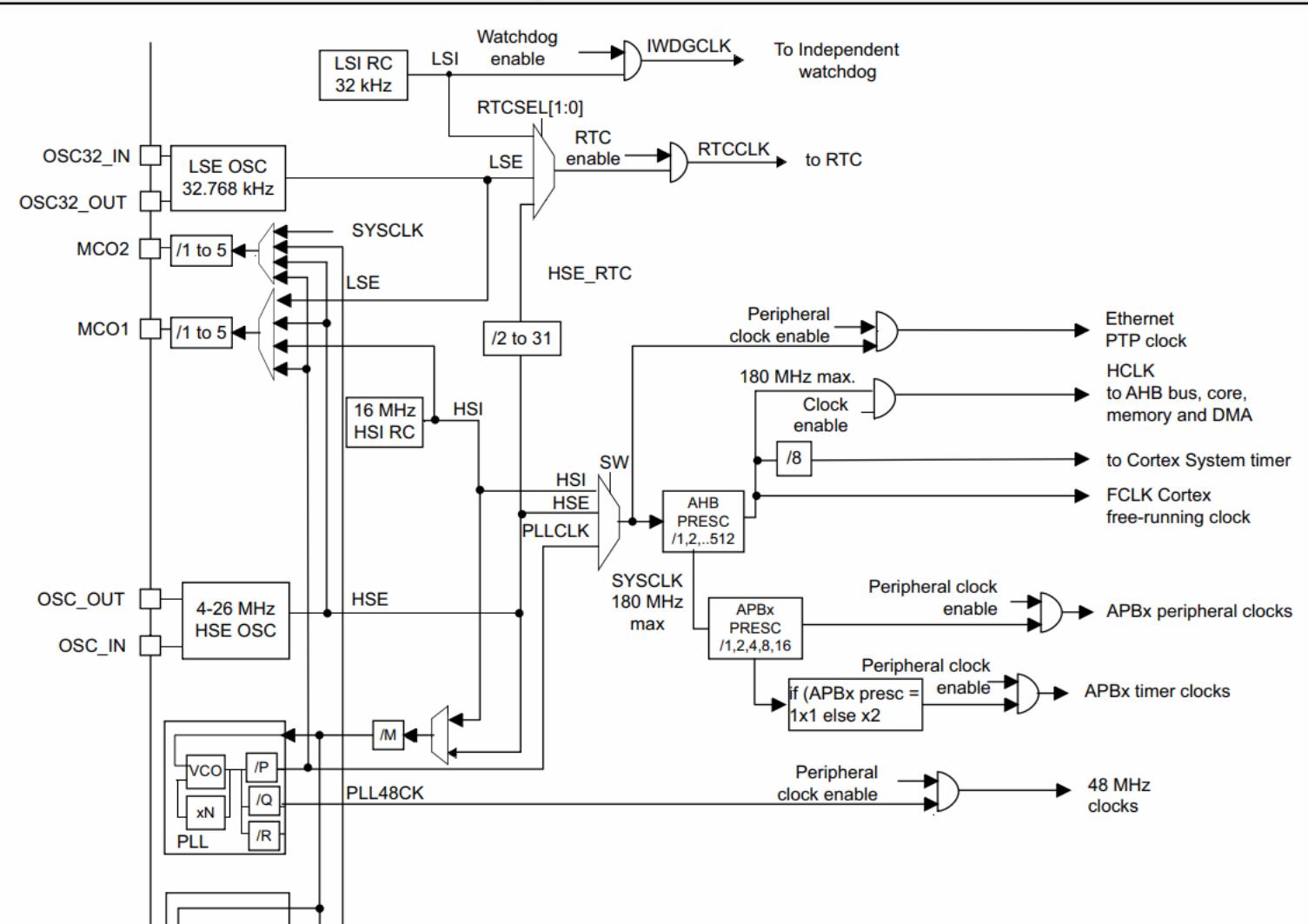
◊

Download

Print

⋮

- > 1 Documentation conventions
- > 2 Memory and bus architecture
- > 3 Embedded Flash memory interface
- > 4 CRC calculation unit
- > 5 Power controller (PWR)
- > 6 Reset and clock control for STM32F42xxx and STM32F43xxx (RCC)
- > 7 Reset and clock control for STM32F405xx/07xx and STM32F415xx/17xx(RCC)
- > 8 General-purpose I/Os (GPIO)
- > 9 System configuration controller (SYSCFG)
- > 10 DMA controller (DMA)
- > 11 Chrom-Art Accelerator™ controller (DMA2D)
- > 12 Interrupts and events
- > 13 Analog-to-digital converter (ADC)
- > 14 Digital-to-analog converter



STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced...

152 / 1751

156%

+

-

□

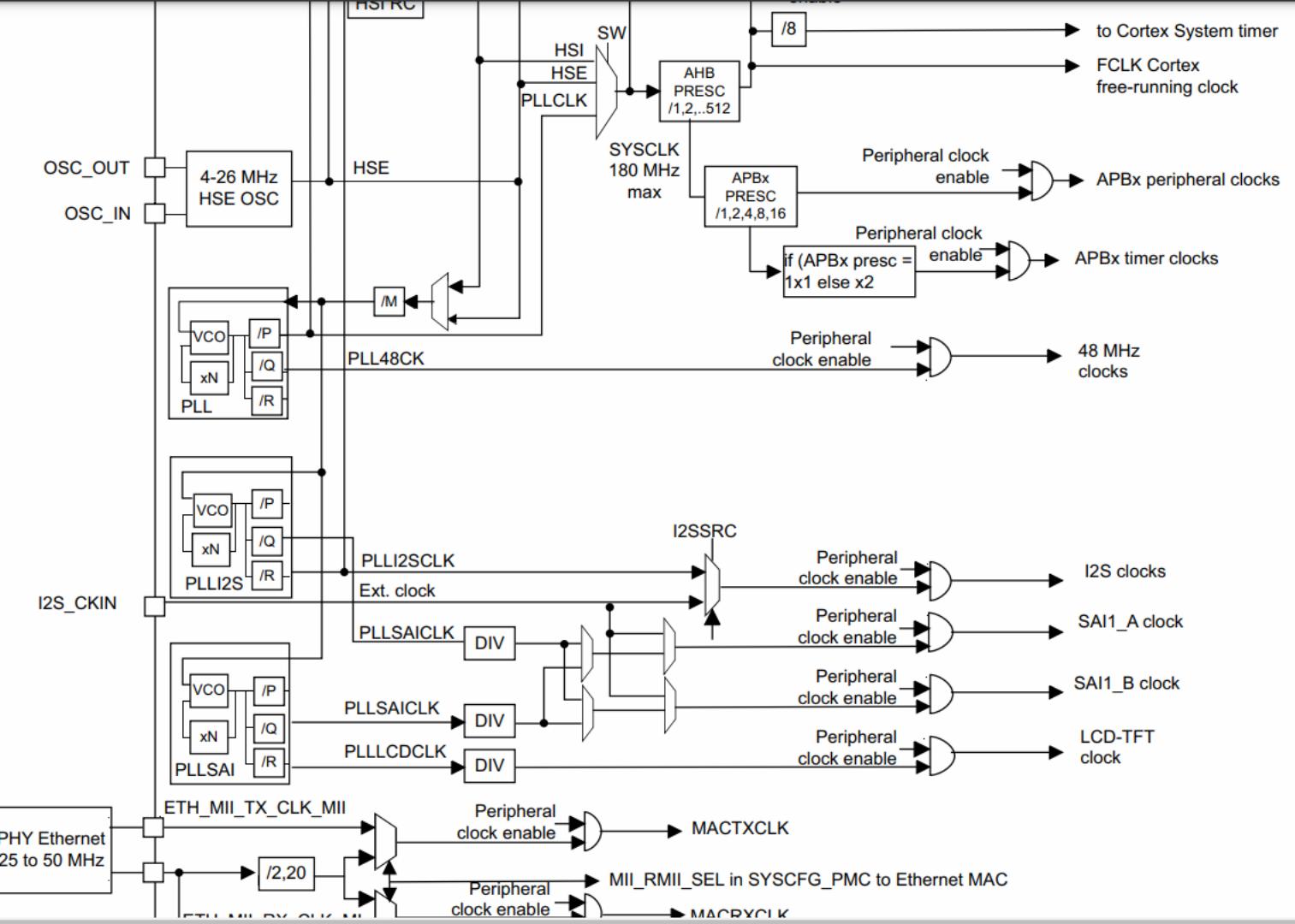
□

Download

Print

⋮

- > 1 Documentation conventions
- > 2 Memory and bus architecture
- > 3 Embedded Flash memory interface
- > 4 CRC calculation unit
- > 5 Power controller (PWR)
- > 6 Reset and clock control for STM32F42xxx and STM32F43xxx (RCC)
- > 7 Reset and clock control for STM32F405xx/07xx and STM32F415xx/17xx(RCC)
- > 8 General-purpose I/Os (GPIO)
- > 9 System configuration controller (SYSCFG)
- > 10 DMA controller (DMA)
- > 11 Chrom-Art Accelerator™ controller (DMA2D)
- > 12 Interrupts and events
- > 13 Analog-to-digital converter (ADC)
- > 14 Digital-to-analog converter



STM32F405/415, STM32F407/417 x Microsoft Word - ILI9341_DS_V1. x OTA5180A x +

File | D:/fastbit/courses/mcu3/docs/F4/RM_rm0090-stm32f405415-stm32f407417-stm32f427437-and-stm32f429439-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Bookmarks android_touchscreen v1 Messages Mail :: Inbox IEEE Arithmetic S-Touch® advance... Other bookmarks

STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advan... 152 / 1751 - 156% + ☰ ⚡

architecture

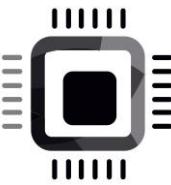
- > 3 Embedded Flash memory interface
- > 4 CRC calculation unit
- > 5 Power controller (PWR)
- 6 Reset and clock control for STM32F42xxx and STM32F43xxx (RCC)
 - > 6.1 Reset
 - > 6.2 Clocks
 - 6.3 RCC registers
 - 6.3.1 RCC clock control register (RCC_CR)
 - 6.3.2 RCC PLL configuration register (RCC_PLLCFGR)
 - 6.3.3 RCC clock configuration register (RCC_CFGR)
 - 6.3.4 RCC clock interrupt register (RCC_CIR)
 - 6.3.5 RCC AHB1 peripheral reset register (RCC_AHB1RSTR)

The diagram illustrates the clock generation logic for the STM32F4 series. It starts with an external I2S_CKIN source. This signal feeds into two PLLs: PLLI2S and PLLSAI. PLLI2S generates the PLLI2SCLK output. PLLSAI generates the PLLSAICLK output, which is then divided by a DIV block to produce the PLLLCDCLK output. Both PLLI2SCLK and PLLLCDCLK are used as reference clocks for other dividers. The PLLI2SCLK is also used to drive an I2SSRC block. The I2SSRC block provides clock enable signals to generate I2S clocks, SAI1_A clock, SAI1_B clock, and LCD-TFT clock. The PLLSAICLK is divided by a DIV block to generate the MACTXCLK output. The PLLSAICLK is also divided by a DIV block and then further processed by a /2,20 divider to generate the MACRXCLK and MACRMIICLK outputs. The MACRXCLK and MACRMIICLK outputs are generated via peripheral clock enable logic. The OTG_HS_ULPI_CK signal is also generated via peripheral clock enable logic. The entire diagram is labeled MS30434V2.

1. For full details about the internal and external clock source characteristics, refer to the Electrical characteristics section in the device datasheet.
2. When TIMPRE bit of the RCC_DCKCFGR register is reset, if APBx prescaler is 1, then TIMxCLK = PCLKx, otherwise TIMxCLK = 2x PCLKx.
3. When TIMPRE bit in the RCC_DCKCFGR register is set, if APBx prescaler is 1,2 or 4, then TIMxCLK = HCLK, otherwise TIMxCLK = 4x PCLKx.

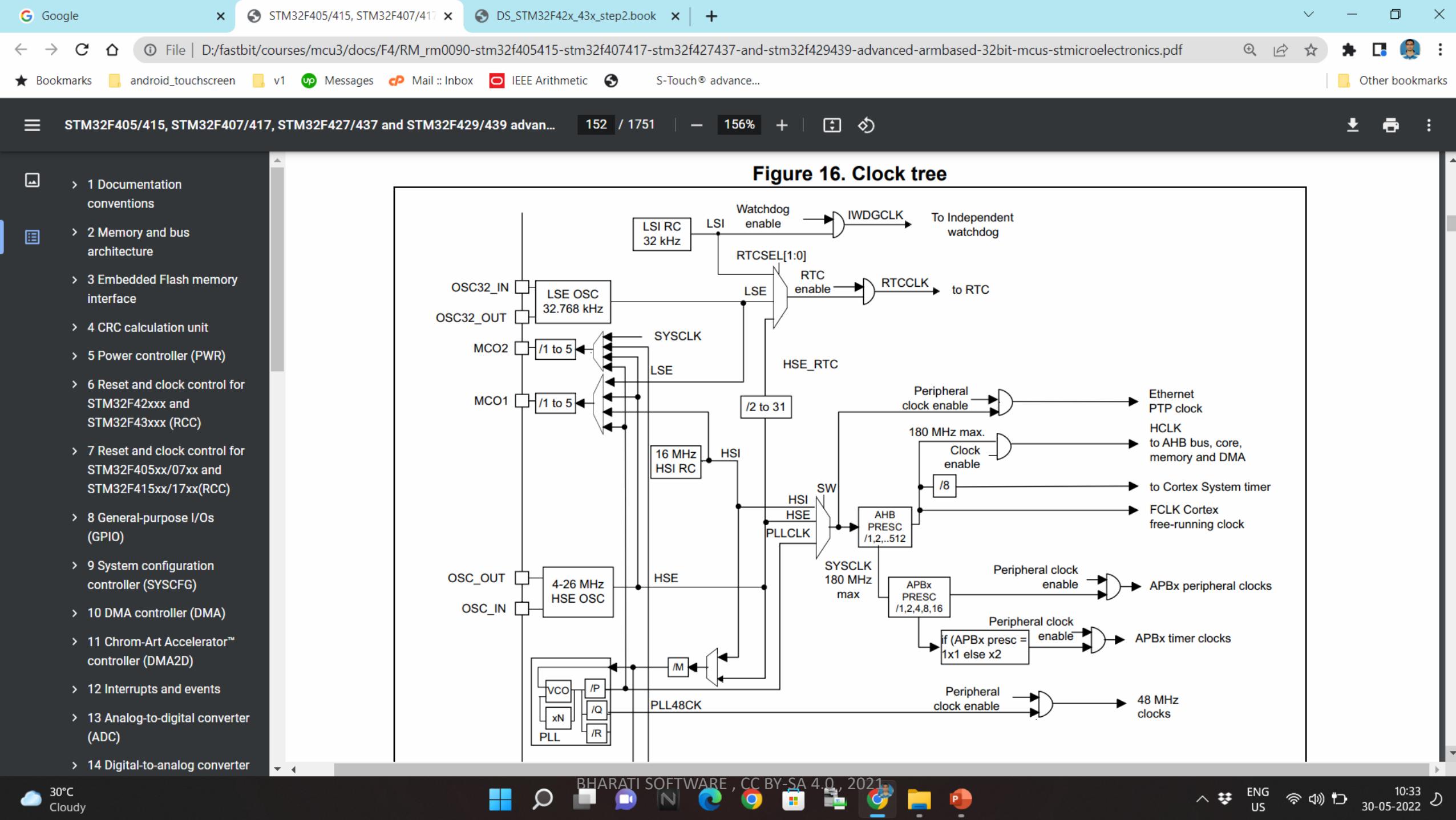
152/1751 RM0090 Rev 19 ST

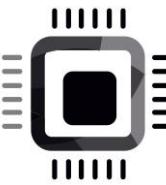
29°C Partly sunny BHARATI SOFTWARE, CC BY-SA 4.0 , 2021 IDE 17:29 ENG US 28-05-2022



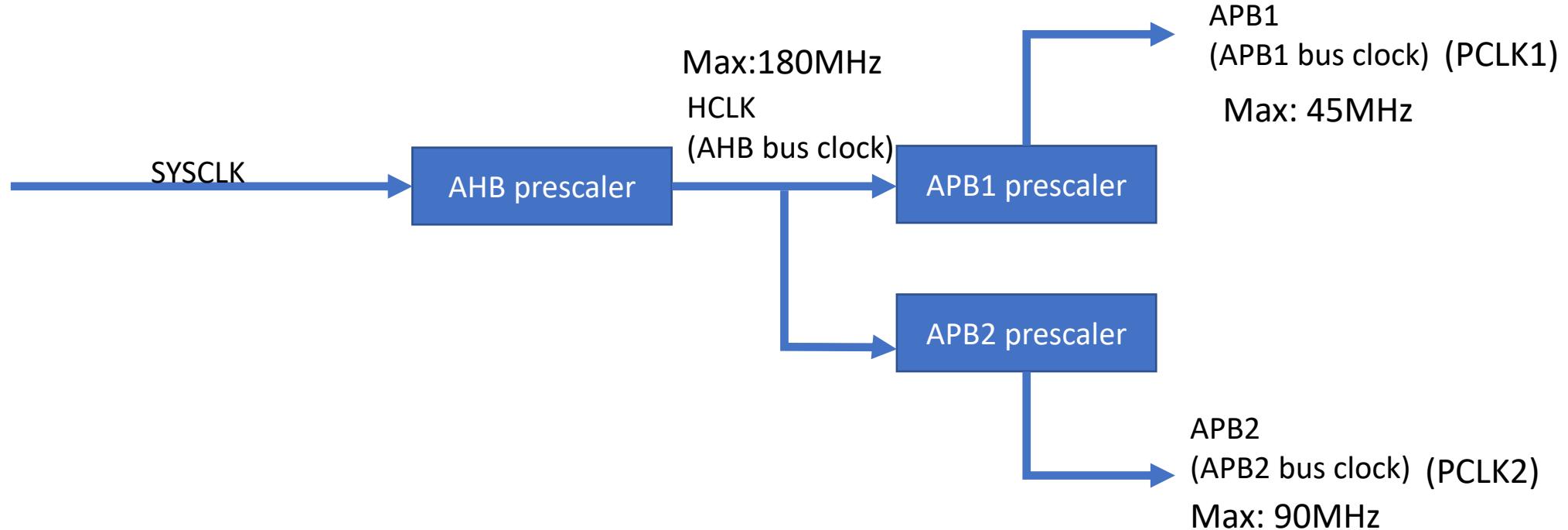
Main PLL configuration

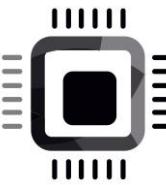
- PLL_M (Controls input to the PLL VCO and PLLSAI VCO)
- PLL_N (Multiplier)
- PLL_P(Divider)





Setup AHB and APBx clocks (STM32F429)

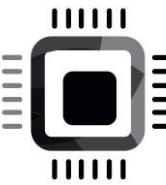




Setup AHB and APB clocks (STM32F429)

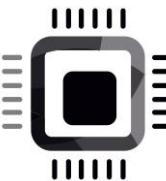
- Configure HCLK(AHB clock) to 180MHz (SYSCLK/1)
 - AHB prescaler to use → 1
- Configure PCLK1(APB1 clock) to 45MHz (HCLK/4)
 - APB1 prescaler to use → 4
- Configure PCLK2(APB2 clock) to 90MHz (HCLK/2)
 - APB2 prescaler to use → 2

Please check your MCU's RM to know maximum speed of HCLK,PCLK1,PCLK2



Setup main system clock(SYSCLK)

- Configure main PLL parameters
- Configure PLLSAI parameters
- Configure AHB, APBx prescalers
- Turn on the main PLL
- Wait until PLLCLK ready bit is set
- Switch PLLCLK as SYSCLK
- Wait for switch status
- Configure FLASH wait states :
 - To correctly read data from Flash memory, the number of wait states (LATENCY) must be correctly programmed in the Flash access control register (FLASH_ACR) according to the frequency of the CPU clock (HCLK) and the supply voltage of the device.
- Turn on PLLSAI
- Wait until PLLSAICLK ready bit is set



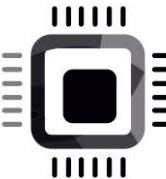
Configure FLASH read latency

**Table 11. Number of wait states according to CPU clock (HCLK) frequency
(STM32F42xxx and STM32F43xxx)**

Wait states (WS) (LATENCY)	HCLK (MHz)			
	Voltage range 2.7 V - 3.6 V	Voltage range 2.4 V - 2.7 V	Voltage range 2.1 V - 2.4 V	Voltage range 1.8 V - 2.1 V Prefetch OFF
0 WS (1 CPU cycle)	0 < HCLK ≤ 30	0 < HCLK ≤ 24	0 < HCLK ≤ 22	0 < HCLK ≤ 20
1 WS (2 CPU cycles)	30 < HCLK ≤ 60	24 < HCLK ≤ 48	22 < HCLK ≤ 44	20 < HCLK ≤ 40
2 WS (3 CPU cycles)	60 < HCLK ≤ 90	48 < HCLK ≤ 72	44 < HCLK ≤ 66	40 < HCLK ≤ 60
3 WS (4 CPU cycles)	90 < HCLK ≤ 120	72 < HCLK ≤ 96	66 < HCLK ≤ 88	60 < HCLK ≤ 80
4 WS (5 CPU cycles)	120 < HCLK ≤ 150	96 < HCLK ≤ 120	88 < HCLK ≤ 110	80 < HCLK ≤ 100
5 WS (6 CPU cycles)	150 < HCLK ≤ 180	120 < HCLK ≤ 144	110 < HCLK ≤ 132	100 < HCLK ≤ 120
6 WS (7 CPU cycles)		144 < HCLK ≤ 168	132 < HCLK ≤ 154	120 < HCLK ≤ 140
7 WS (8 CPU cycles)		168 < HCLK ≤ 180	154 < HCLK ≤ 176	140 < HCLK ≤ 160
8 WS (9 CPU cycles)			176 < HCLK ≤ 180	160 < HCLK ≤ 168

After reset, the CPU clock frequency is 16 MHz and 0 wait state (WS) is configured in the FLASH_ACR register.

It is highly recommended to use the following software sequences to tune the number of wait states needed to access the Flash memory with the CPU frequency.



Over drive mode enable

STM32F427xx STM32F429xx

Electrical characteristics

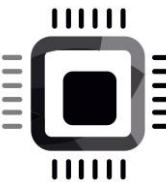
6.3 Operating conditions

6.3.1 General operating conditions

Table 17. General operating conditions

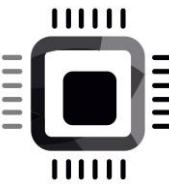
Symbol	Parameter	Conditions ⁽¹⁾	Min	Typ	Max	Unit	
f_{HCLK}	Internal AHB clock frequency	Power Scale 3 (VOS[1:0] bits in PWR_CR register = 0x01), Regulator ON, over-drive OFF	0	-	120	MHz	
		Power Scale 2 (VOS[1:0] bits in PWR_CR register = 0x10), Regulator ON	0	-	144		
				-	168		
		Power Scale 1 (VOS[1:0] bits in PWR_CR register= 0x11), Regulator ON	0	-	168		
				-	180		
		Over-drive OFF	0	-	42		
f_{PCLK1}	Internal APB1 clock frequency	Over-drive ON	0	-	45		
f_{PCLK2}		Over-drive OFF	0	-	84		
		Over-drive ON	0	-	90		
V_{DD}	Standard operating voltage		1.7 ⁽²⁾	-	3.6		
v	Analog operating voltage (ADC limited to 1.2 M samples)		1.7 ⁽²⁾	-	2.4		

PWR_CR
VOS → 0X11
Enable Over drive (ODEN)
Enable ODSWEN bit to switch to
Over drive mode

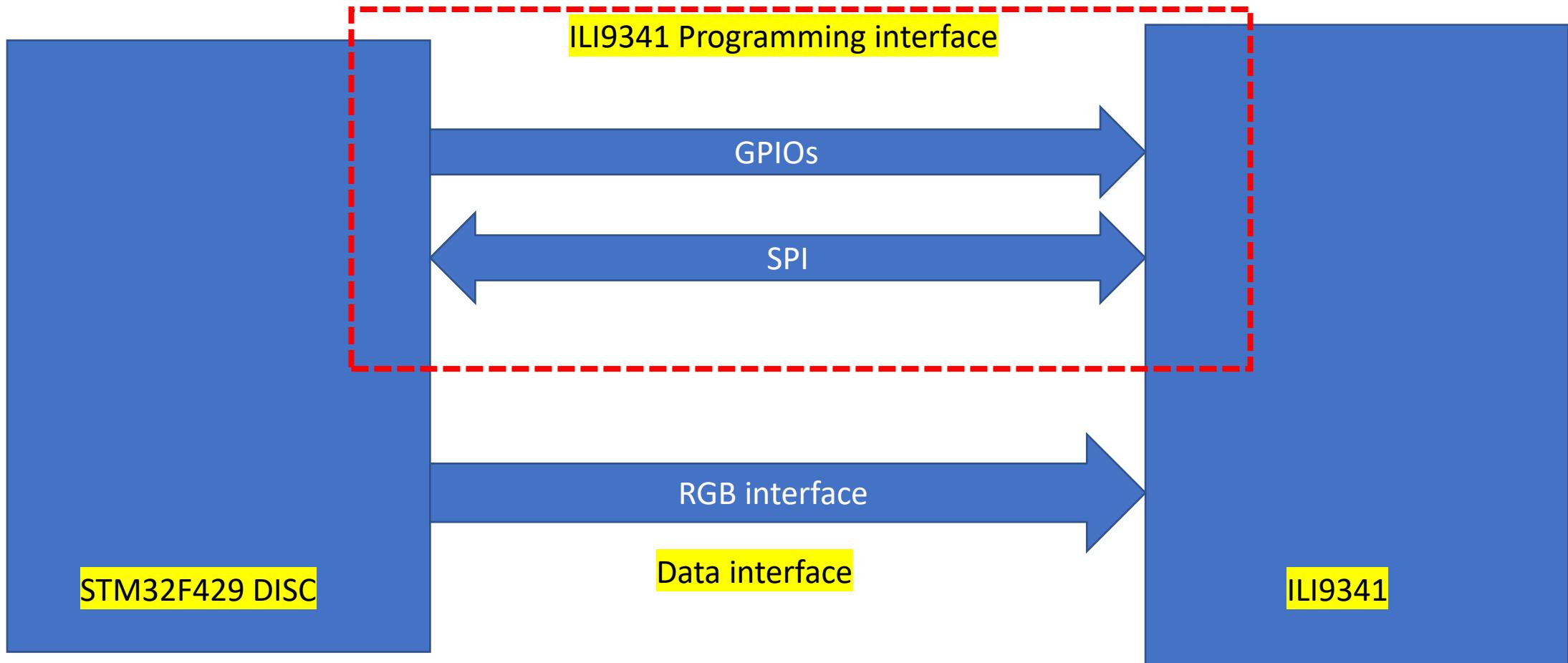


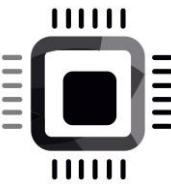
Configuring SPI

- The SPI interface is used to send commands to the display module, which is based on the ILI9341 LCD driver chip
- For STM32F407x + external display setup, this interface is also used to transmit the RGB data

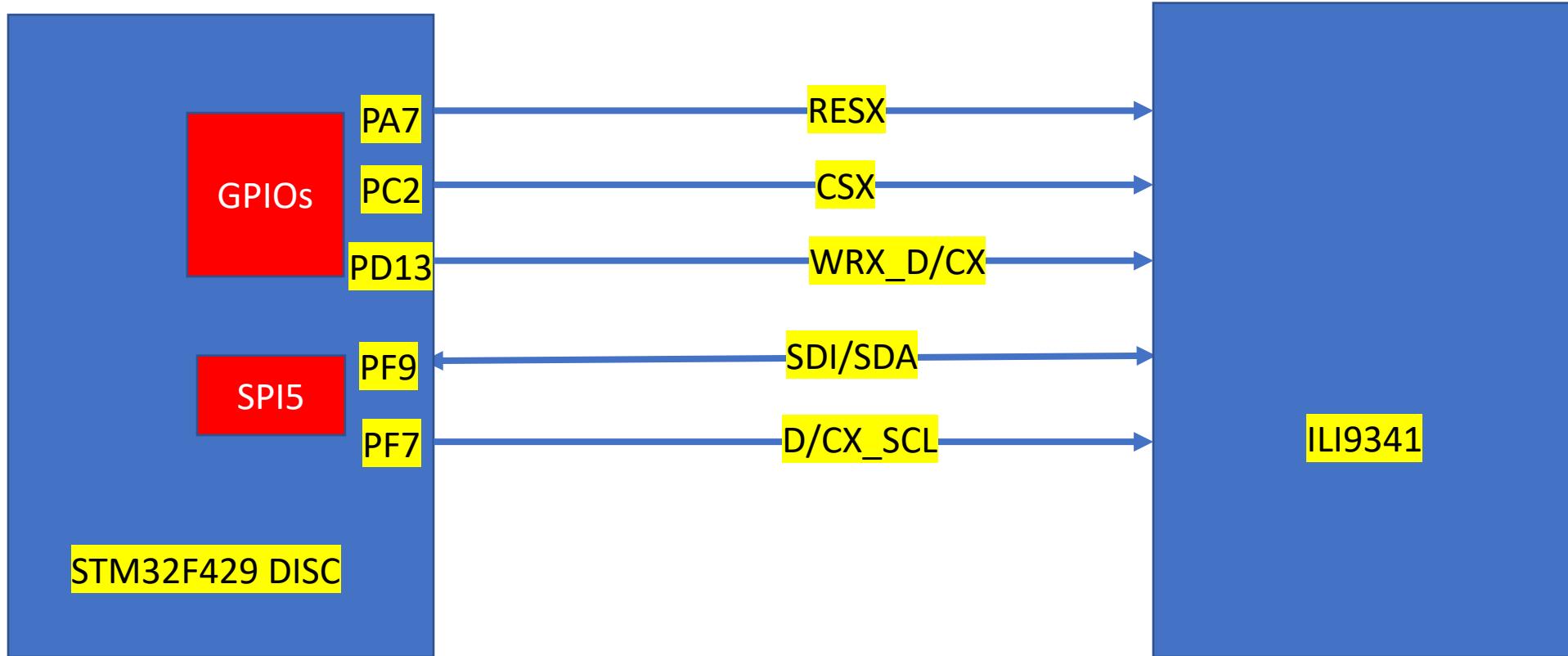


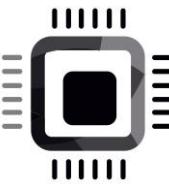
Interfacing (STM32F429 DISC)



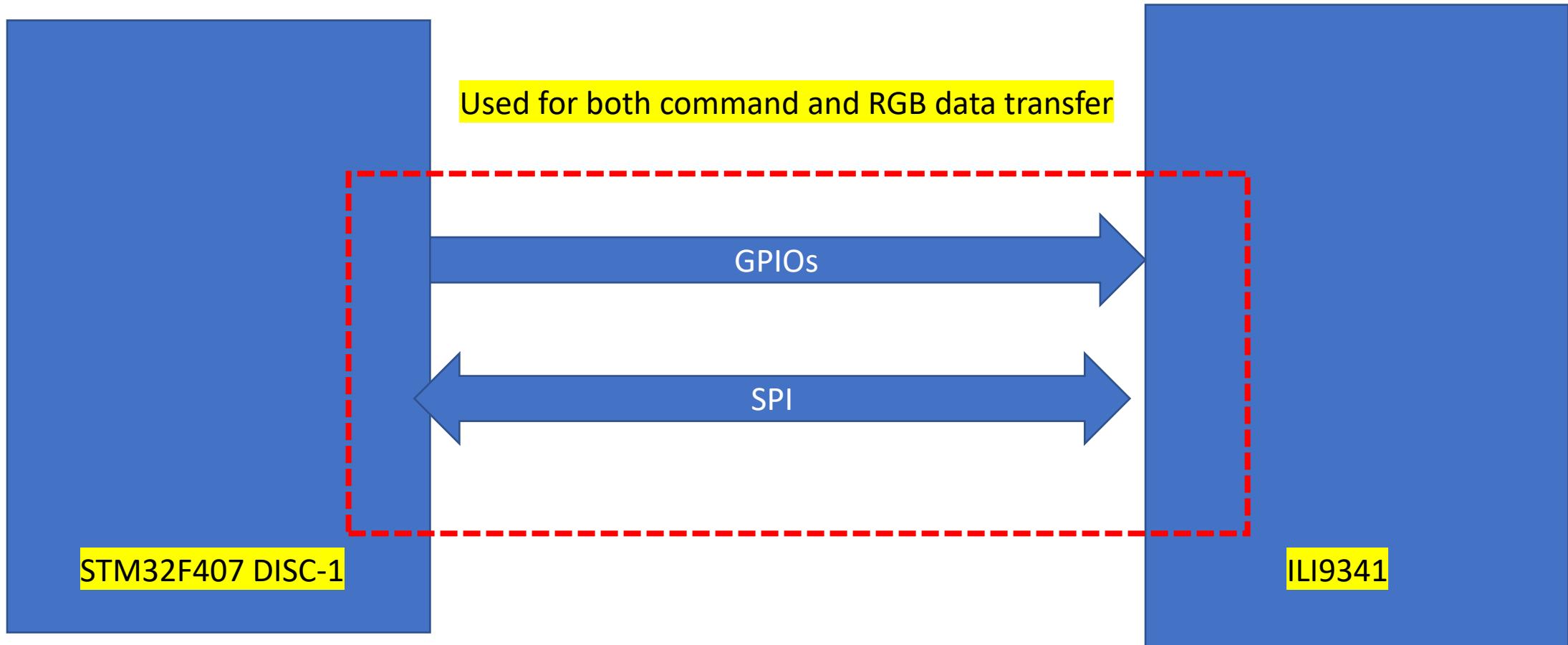


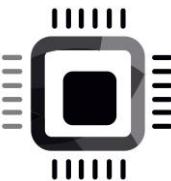
ILI9341 programming interface



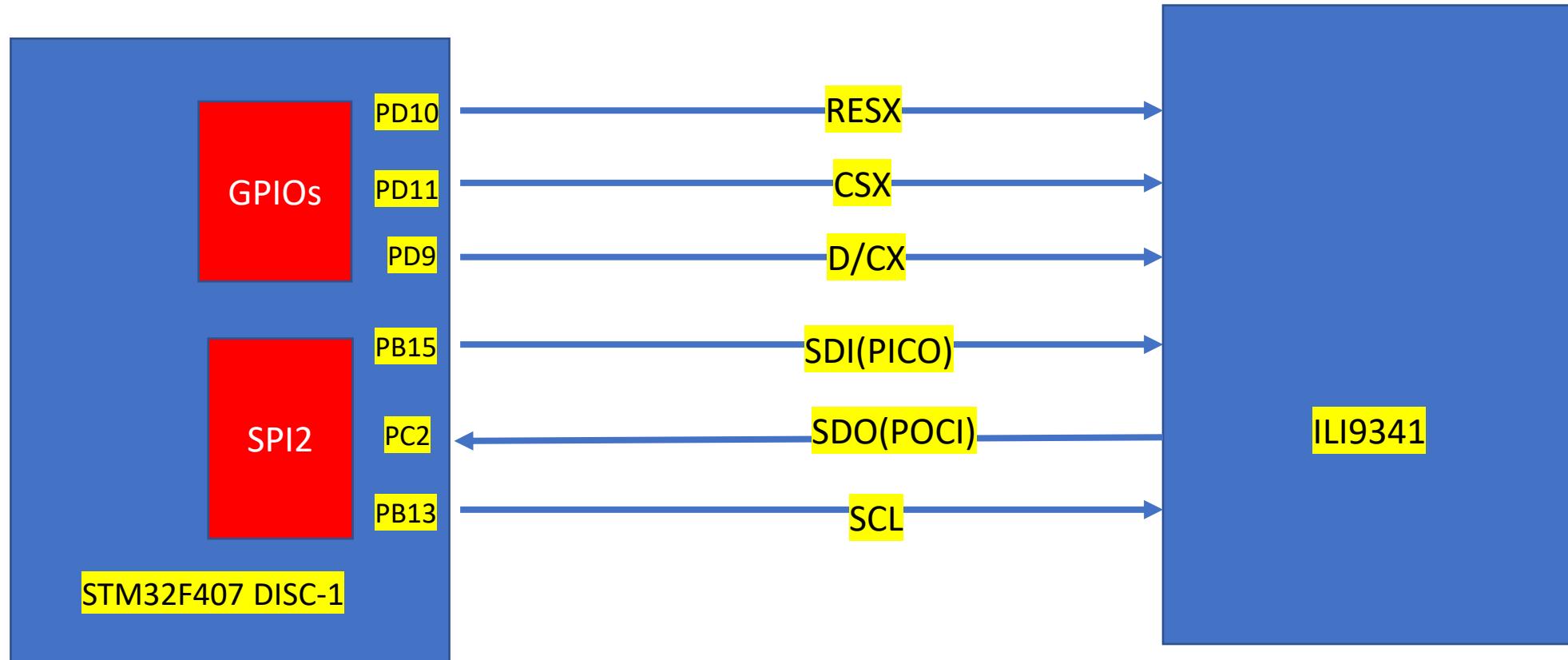


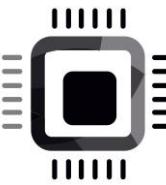
Interfacing (STM32F407x + external LCD)





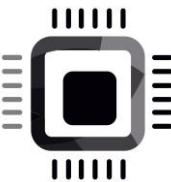
ILI9341 programming interface



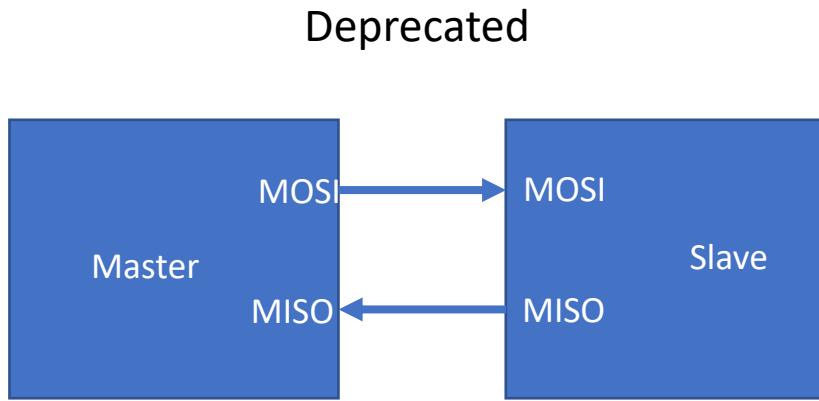


Configuring GPIO pins for SPI functionality

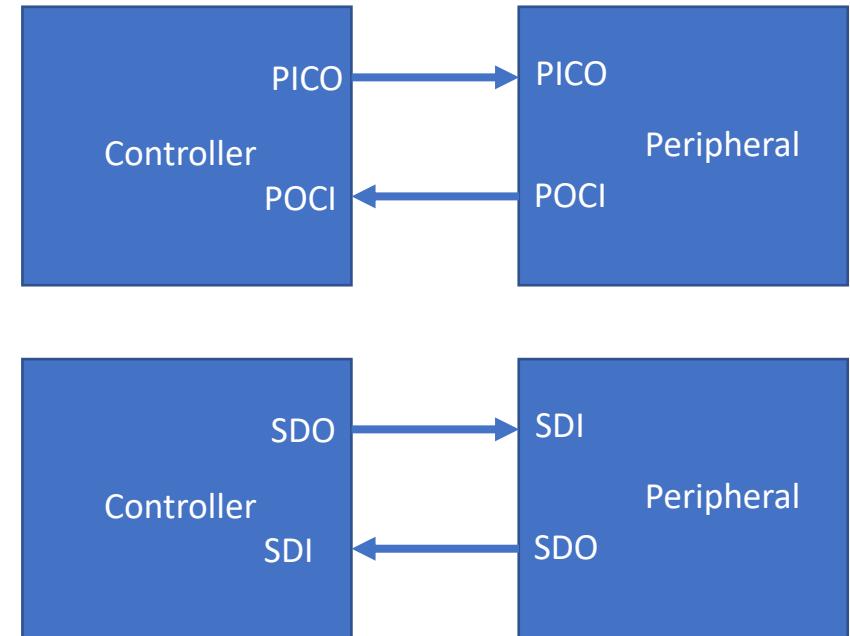
- Configure the GPIO pin to Alternate function mode
- Configure the alternate function mode number in alternate function register (you can get the alt function mode number from MCU datasheet)
- No pull up or pull down settings are required for SPI communication

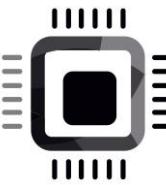


Redefining SPI signal names



New signal names

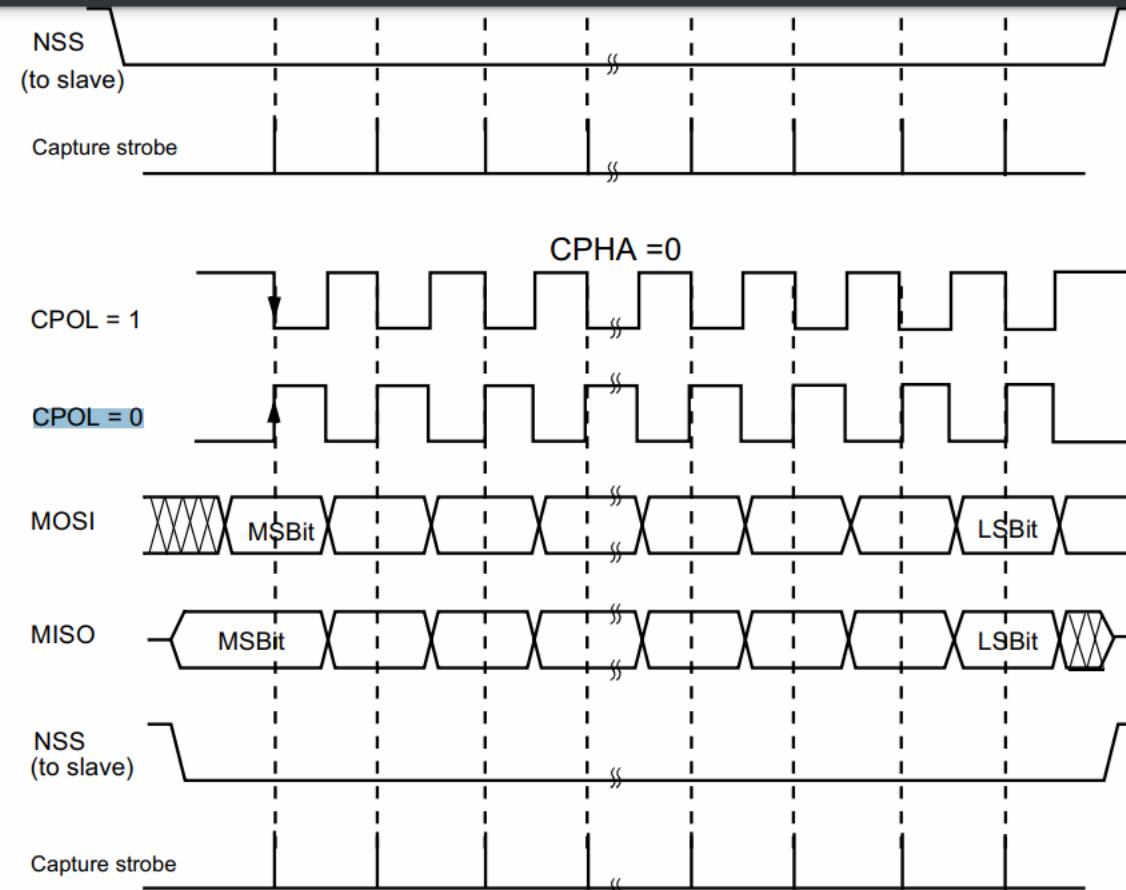




MCU SPI peripheral configuration

- Let's use the below settings for SPI configuration
 - SPI mode : Half-duplex Controller
 - Data format : 8bit, msb first
 - CPOL and CPHA : Check the peripheral datasheet to obtain the correct setting it requires
 - SPI Clock : Check the peripheral datasheet and obtain the max speed it can support (< 6MHz)
 - Chip select is handled by the software

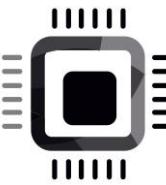
- > 24 Random number generator (RNG)
- > 25 Hash processor (HASH)
- > 26 Real-time clock (RTC)
- > 27 Inter-integrated circuit (I2C) interface
- > 28 Serial peripheral interface (SPI)
 - 28.1 SPI introduction
 - > 28.2 SPI and I2S main features
 - > 28.3 SPI functional description
 - > 28.4 I2S functional description
 - > 28.5 SPI and I2S registers
- > 29 Serial audio interface (SAI)
- > 30 Universal synchronous asynchronous receiver transmitter (USART)
- > 31 Secure digital input/output interface (SDIO)
- > 32 Controller area network (bxCAN)



ai17154d

- These timings are shown with the LSBFIRST bit reset in the SPI_CR1 register.

Data frame format

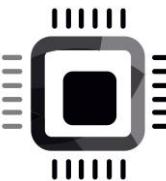


ILI9341 commands

Refer to the files attached with this lecture

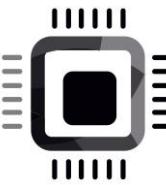
ILI9341_Cmd_RGB_Interface.c (Refer to this file if you are using RGB interface)

ILI9341_Cmd_SPI_Interface.c (Refer to this file if you are using SPI interface for RGB data)



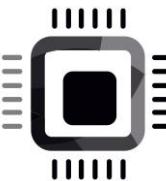
Steps

- 1) Create a new project for your microcontroller or board using stm32cubeide
- 2) Setup the main system clock
- 3) Setup AHB and APB clocks
- 4) Setup pixel clock (DOTCLOCK) (**Not required for STM32F407x-DISC1 board with external LCD**)
- 5) Configure the SPI peripheral (**Not required for STM32F746-DISC board**)
- 6) Setup framebuffer in RAM
- 7) Initialize the LCD module by sending LCD commands over SPI (**Not required for STM32F746-DISC board**)
- 8) Configure and enable the LTDC peripheral (**Not required for STM32F407x-DISC1**)
- 9) Send frame buffer contents to LCD over SPI (**Only in the case of STM32F407x-DISC1**)
- 10) Make sure that all error interrupts enabled and ISRs implemented



LTDC configuration

- 1) LTDC pin initialization (Check the schematic of your board to know which pins are used as LTDC signals)
- 2) LTDC peripheral initialization
- 3) LTDC layers initialization (layer 1 or layer 2 or both)

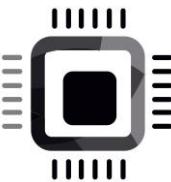


16.3.3 LCD-TFT pins and signal interface

The Table below summarizes the LTDC signal interface:

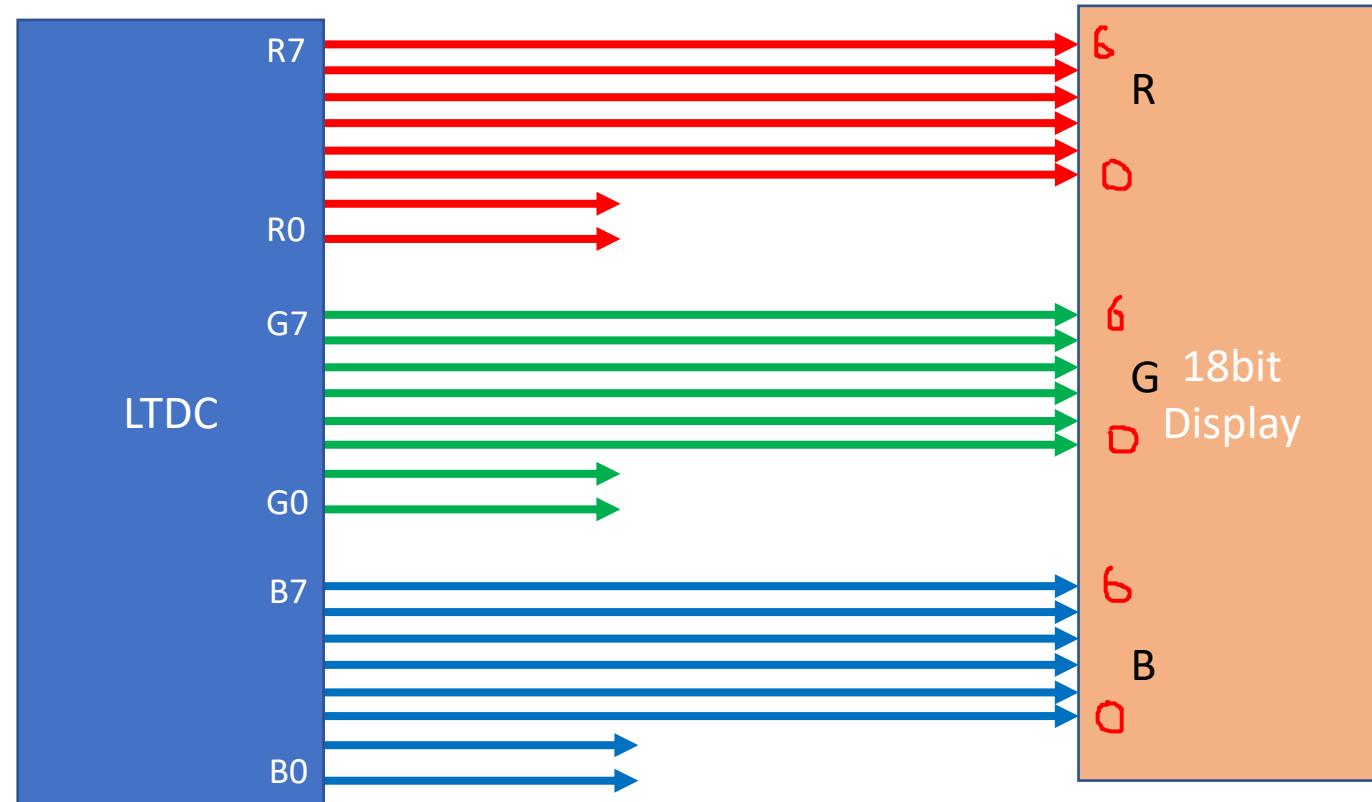
Table 89. LCD-TFT pins and signal interface

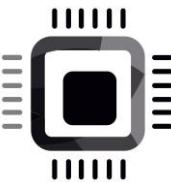
LCD-TFT signals	I/O	Description
LCD_CLK	O	Clock Output
LCD_HSYNC	O	Horizontal Synchronization
LCD_VSYNC	O	Vertical Synchronization
LCD_DE	O	Not Data Enable
LCD_R[7:0]	O	Data: 8-bit Red data
LCD_G[7:0]	O	Data: 8-bit Green data
LCD_B[7:0]	O	Data: 8-bit Blue data



STM32F429 DISC

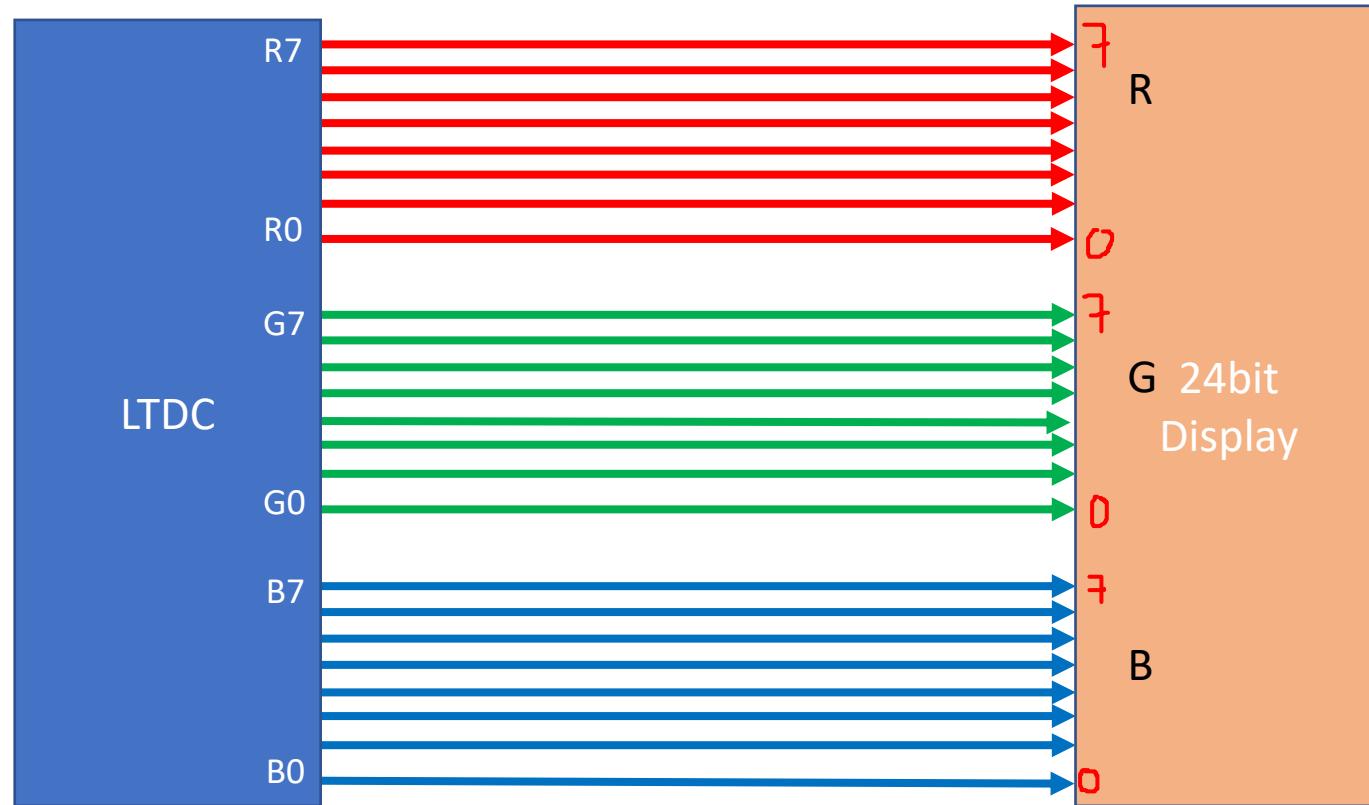
- This board uses 18bit display connected as shown below

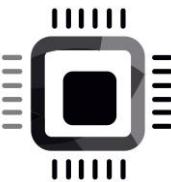




STM32F746ng DISC

- This board uses 24bit display connected as shown below





- 16bit display connected as shown below

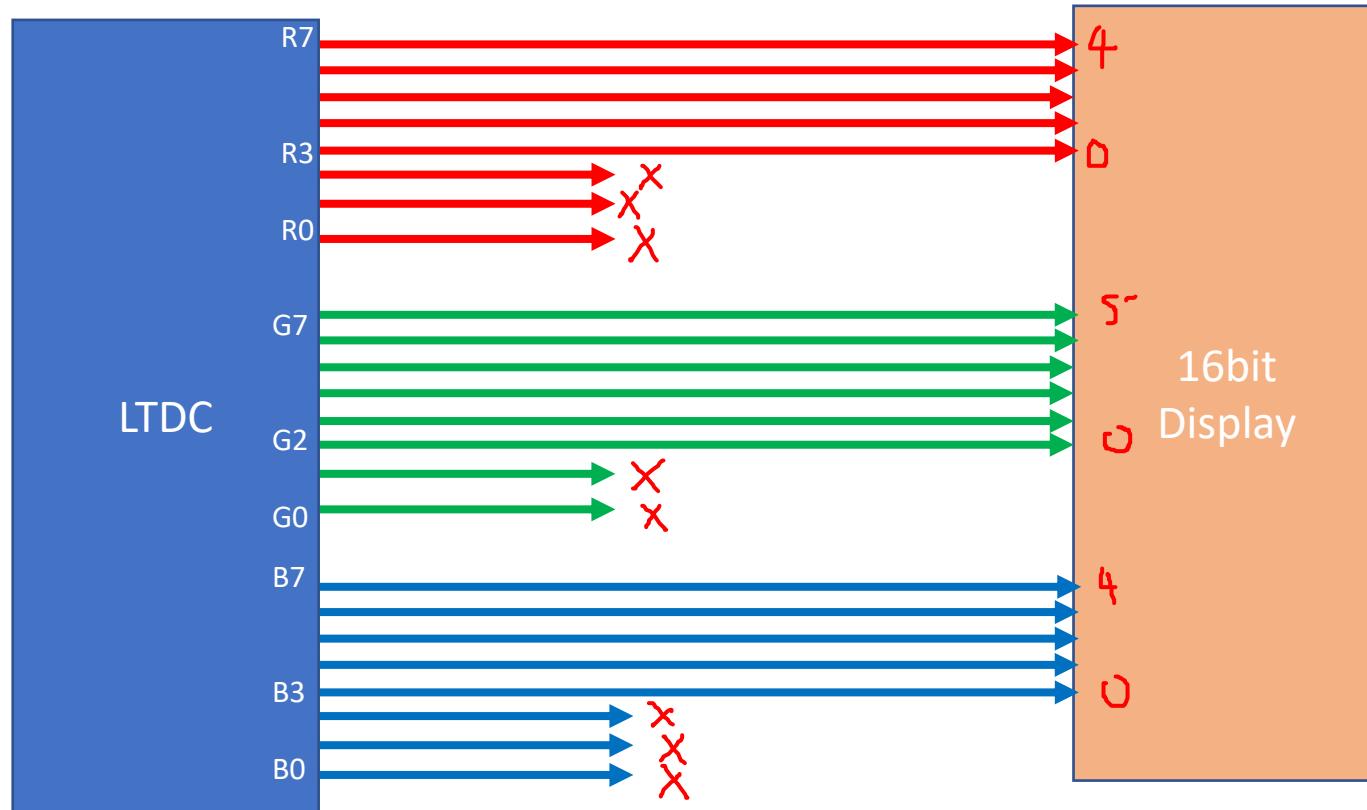
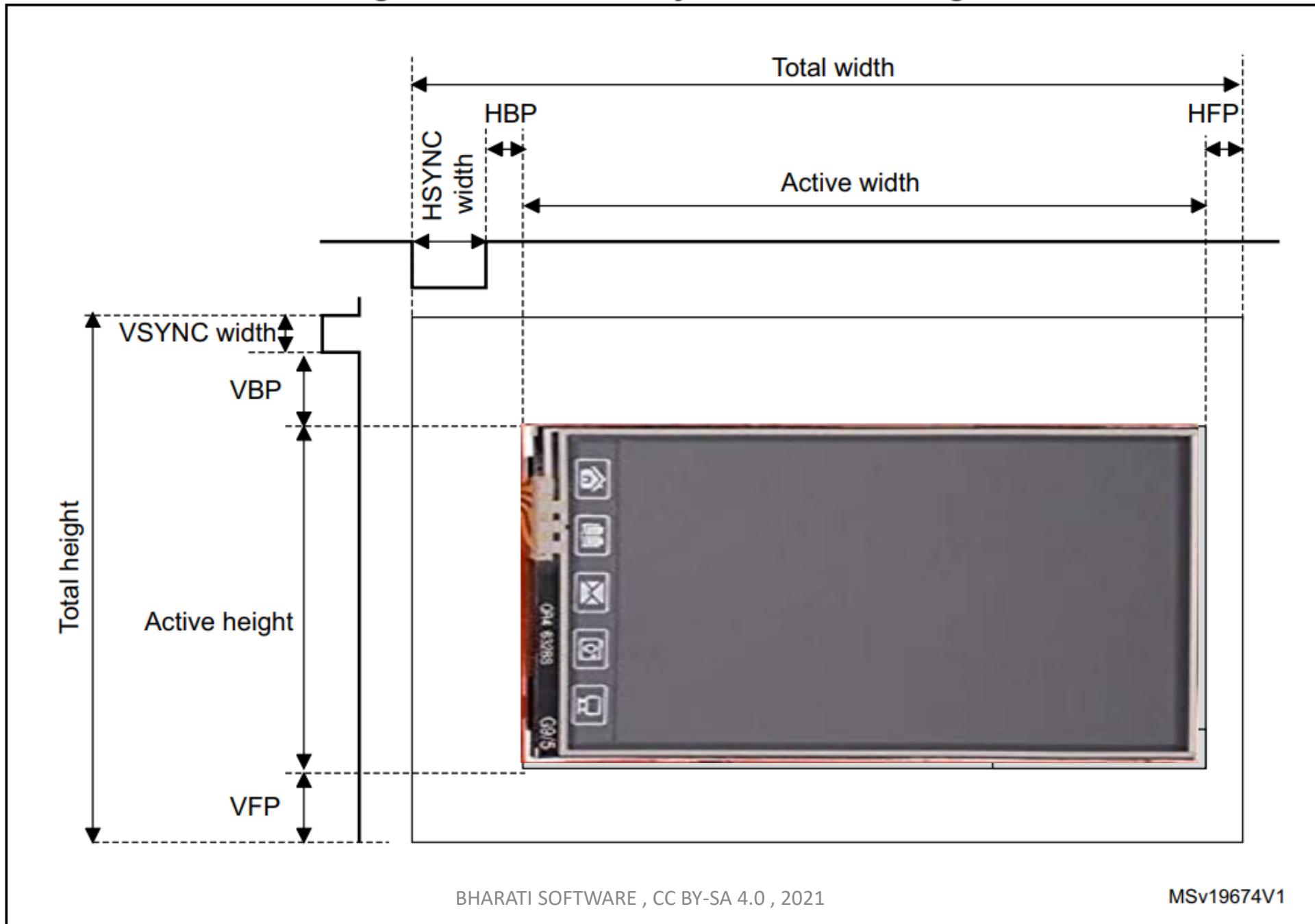
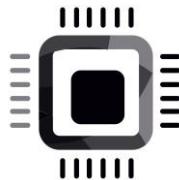
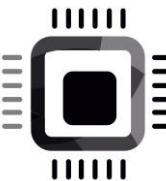


Figure 82. LCD-TFT Synchronous timings



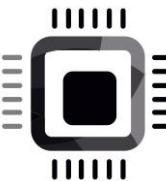


Check display driver chip datasheet to obtain synchronization related values

ILI9341

Parameters	Symbols	Condition	Min.	Typ.	Max.	Units
Horizontal Synchronization	Hsync		2	10	16	DOTCLK
Horizontal Back Porch	HBP		2	20	24	DOTCLK
Horizontal Address	HAdr		-	240	-	DOTCLK
Horizontal Front Porch	HFP		2	10	16	DOTCLK
Vertical Synchronization	Vsync		1	2	4	Line
Vertical Back Porch	VBP		1	2	-	Line
Vertical Address	VAdr		-	320	-	Line
Vertical Front Porch	VFP		3	4	-	Line

Typical values are setting example when used with panel resolution 240 x 320 (QVGA), clock frequency 6.35MHz and frame



Some math

Total Width(TW) = AW+HSW+HBP+HFP pixels

Total time required to send one line(T_{line}) = $(TW * T_{DOTCLK})$ sec

Total Lines(TL) = AH+VSW+VBP+VFP lines

Total time required to send 1 frame(T_{frame}) = $(TL * T_{line})$ sec

DOTCLK = 6.25MHz

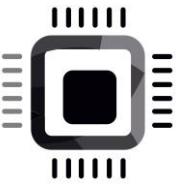
T_{DOTCLK} = 0.16 micro sec

$TW = AW+HSW+HBP+HFP = 320 + 10 + 20 + 10 \rightarrow 360$ pixels

$TL = AH+VSW+VBP+VFP = 240 + 2 + 2 + 4 \rightarrow 248$ lines

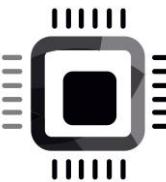
$T_{line} = (TW * T_{DOTCLK}) = 360 * 0.16 \rightarrow 57.6 \mu s$

$T_{frame} = (TL * T_{line}) = 248 * 57.6 \rightarrow 14.3$ ms



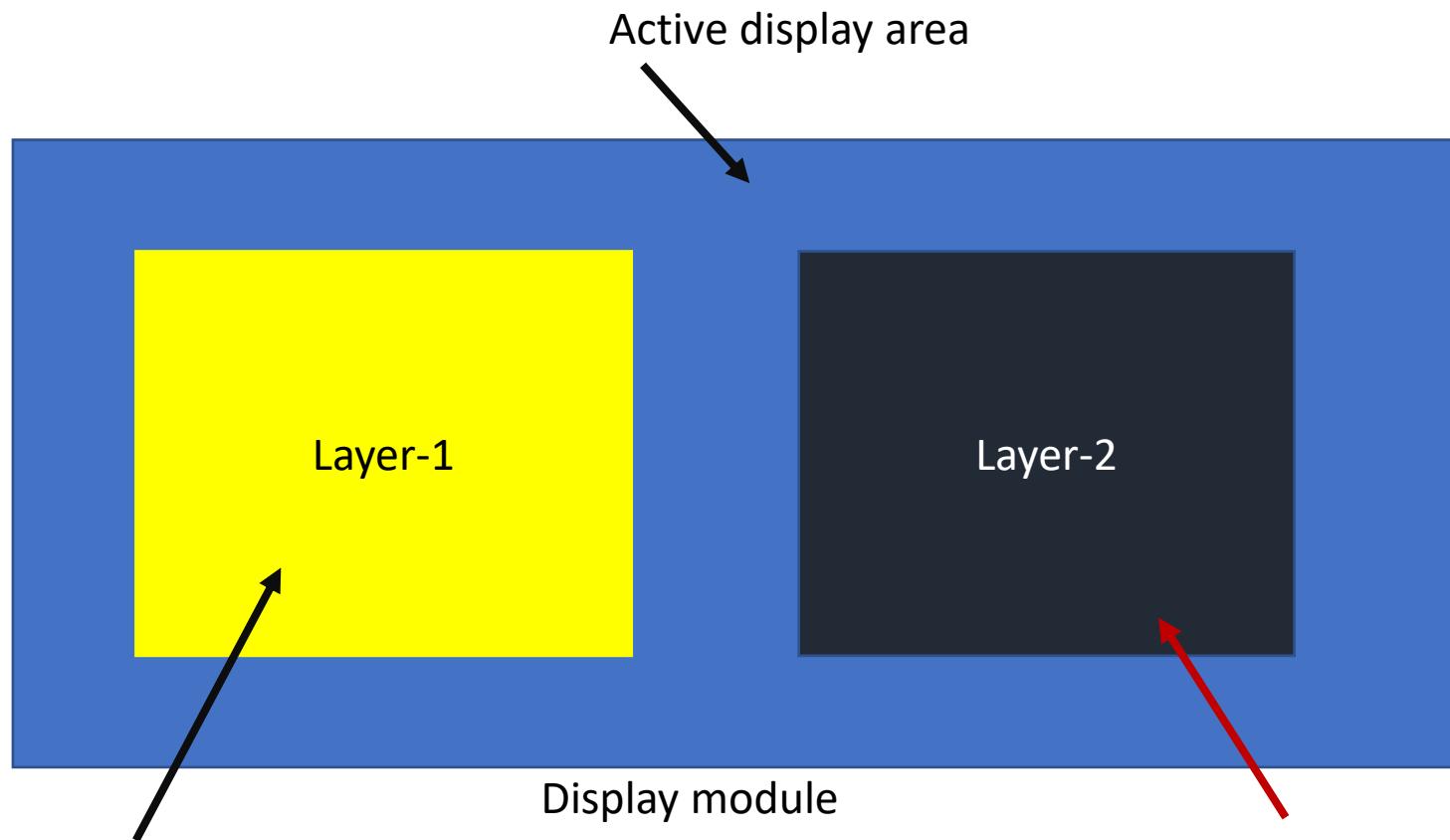
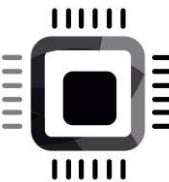
Task

Configure the LTDC peripheral to produce the background color RED.



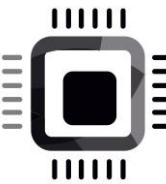
LTDC Layers

- LTDC supports 2 layers
- You can activate either of the layers or all layers as per your application need
- Programmable default color settings for each layer
- Programmable frame buffer address settings for each layer
- Flexible blending options (Constant and pixel alpha blending)
- Programmable window positioning for each layer
- Most LTDC layer registers are shadow registers, and you need to activate loading shadow values to real registers explicitly.



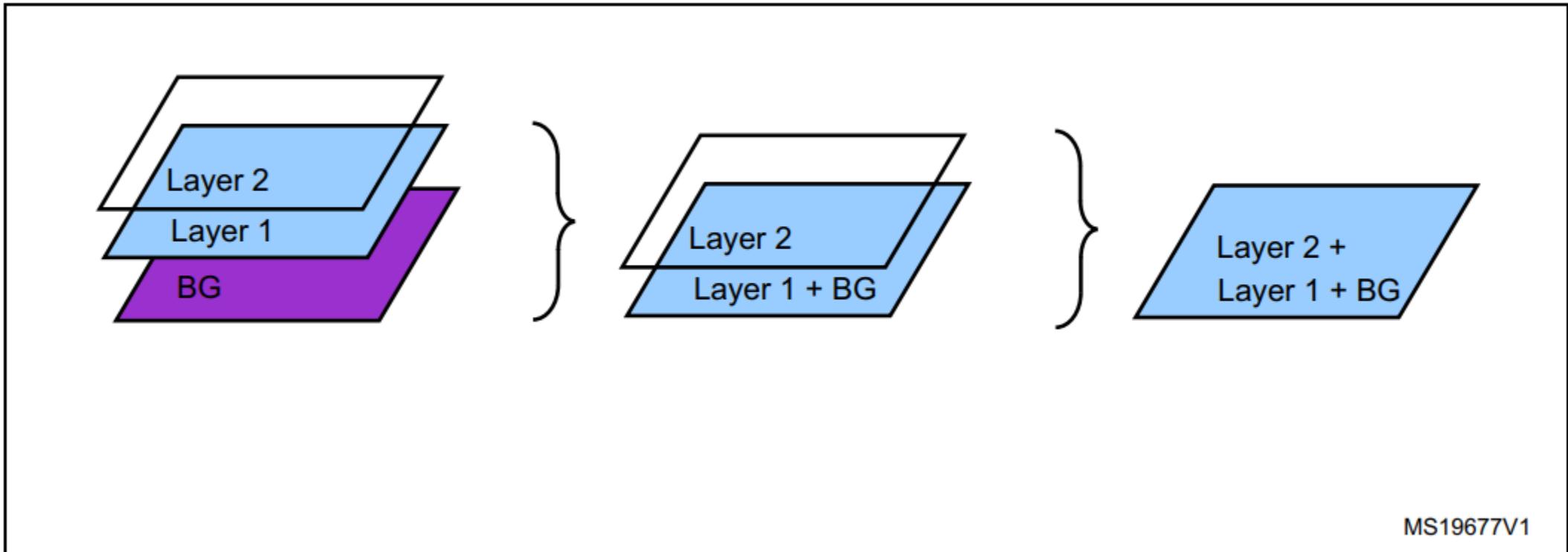
Layer-1 activated with windowing
and dedicated framebuffer holding
yellow color

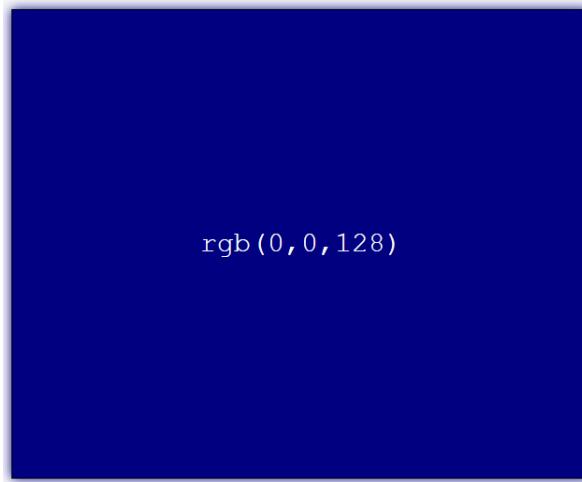
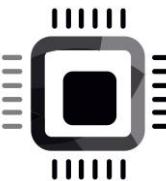
Layer-2 activated with
windowing
and dedicated framebuffer
holding black color



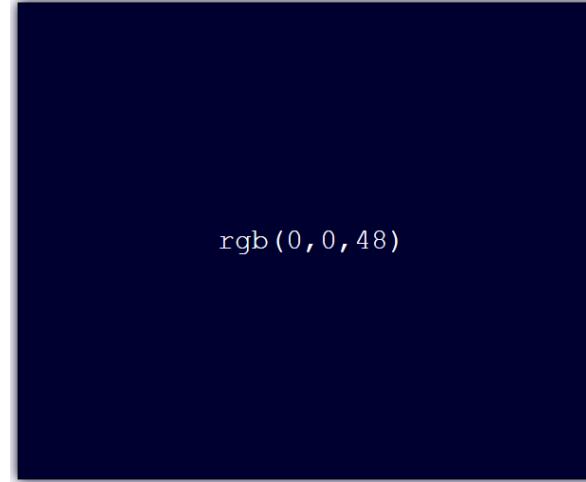
Layers and Blending

Figure 84. Blending two layers with background

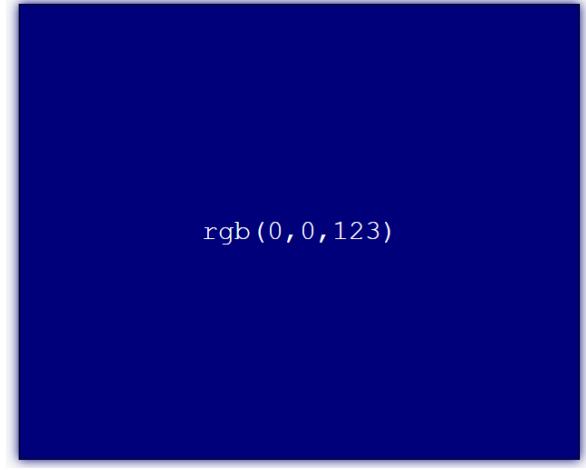




Layer-1 color



Background color



Blended color

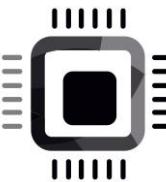
Constant Alpha (layer 1) = 240 , Constant Alpha value = $240/255= 0.94$

layer 1 : BF1 = constant alpha , BF2 = 1 - Constant Alpha

$BC = BF1 \times C + BF2 \times Cs$

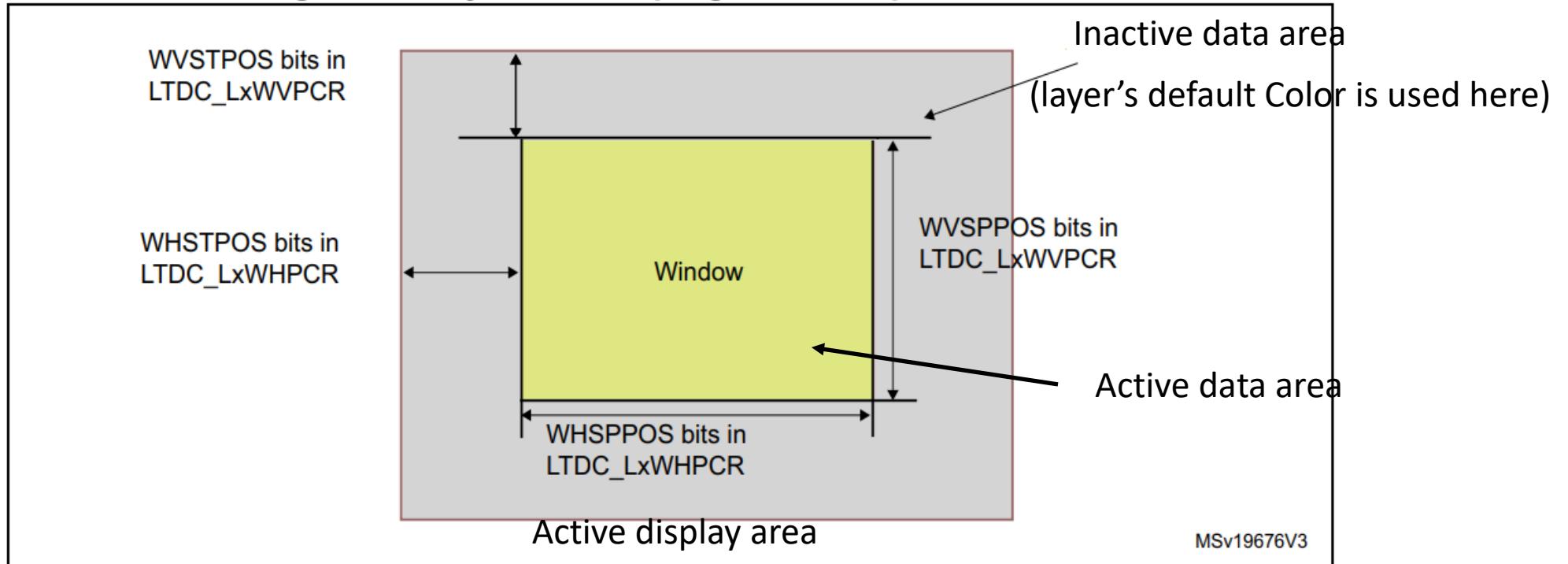
$$= 0.94 * 128 + 0.06 * 48 = 123$$

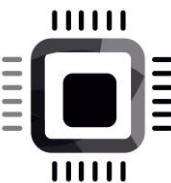
$Cs \rightarrow$ Subjacent color
 $C \rightarrow$ Color of top layer



Layer windowing

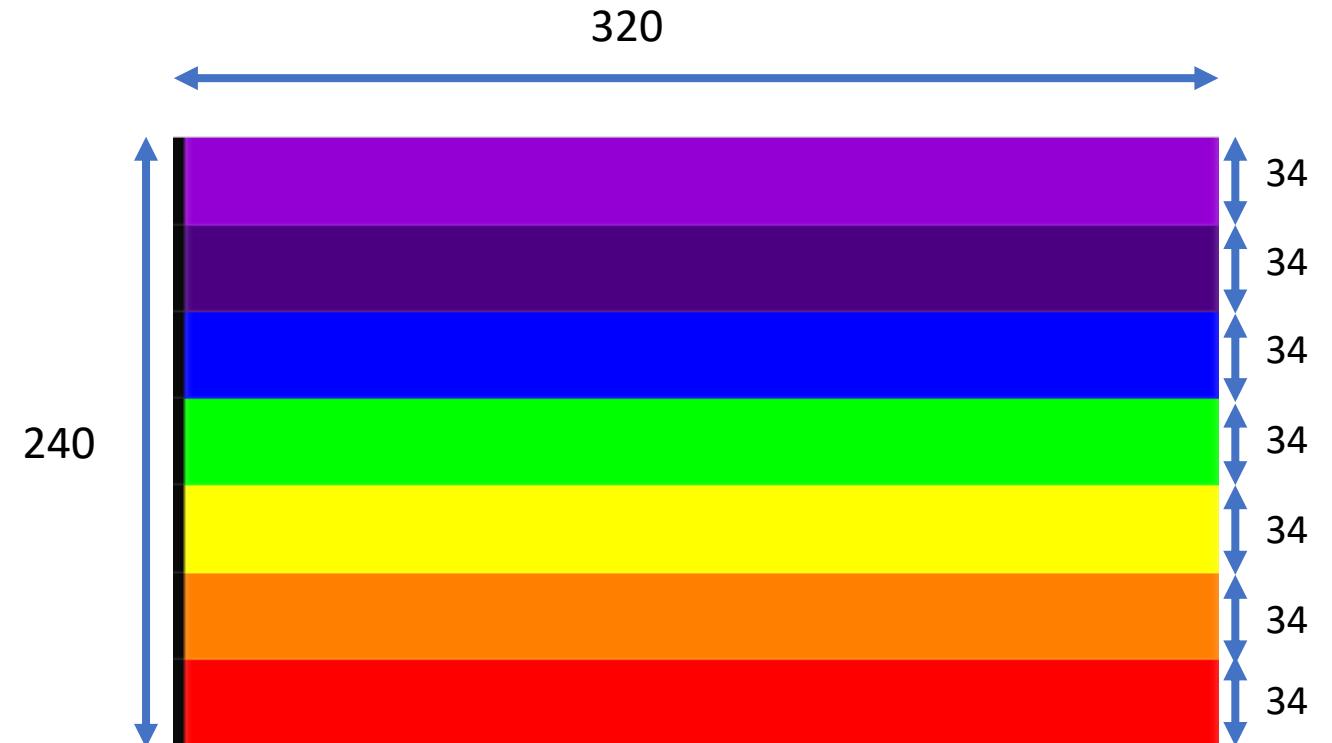
Figure 83. Layer window programmable parameters:

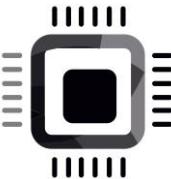




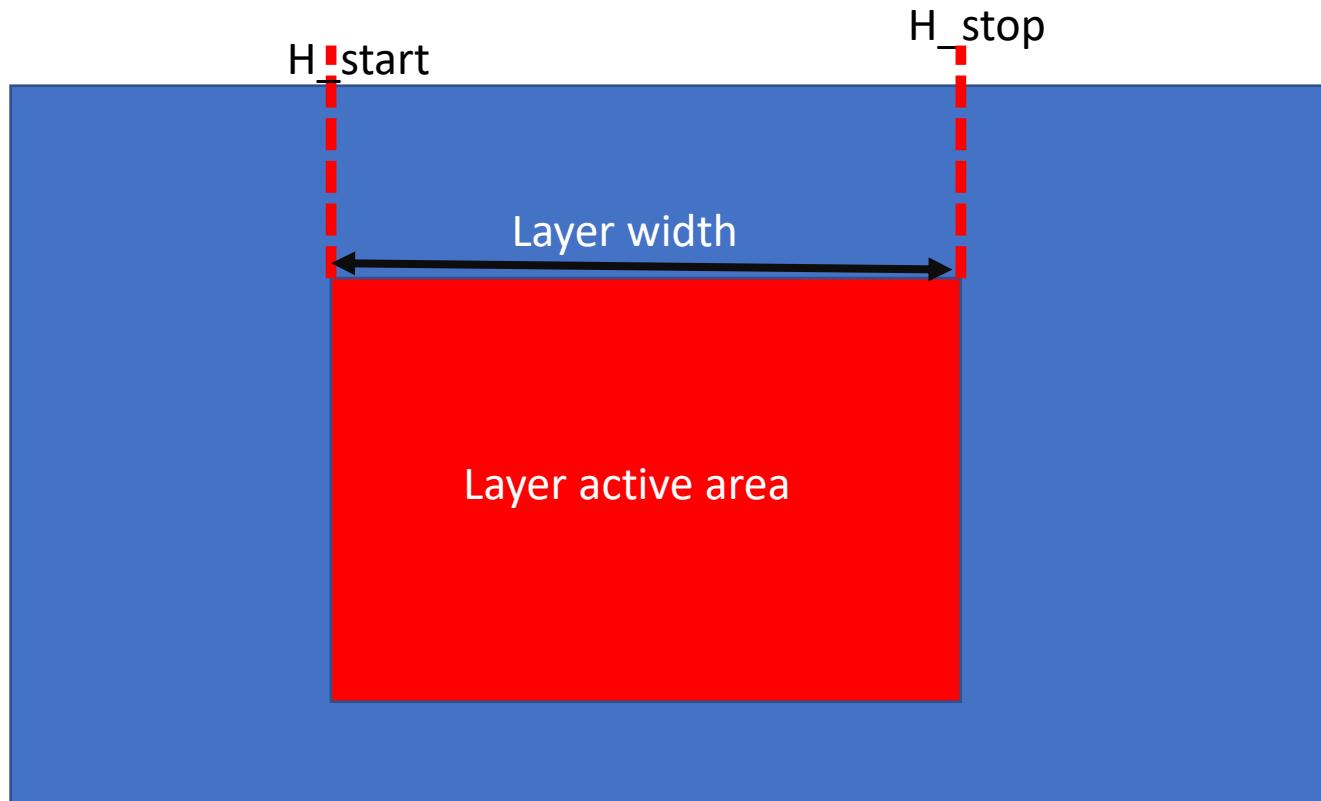
Exercise 001 : Display VIBGYOR bars

Color	RGB Value
Violet	148, 0, 211
Indigo	75, 0, 130
Blue	0, 0, 255
Green	0, 255, 0
Yellow	255, 255, 0
Orange	255, 127, 0
Red	255, 0 , 0



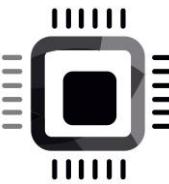


Layer window horizontal position configuration items

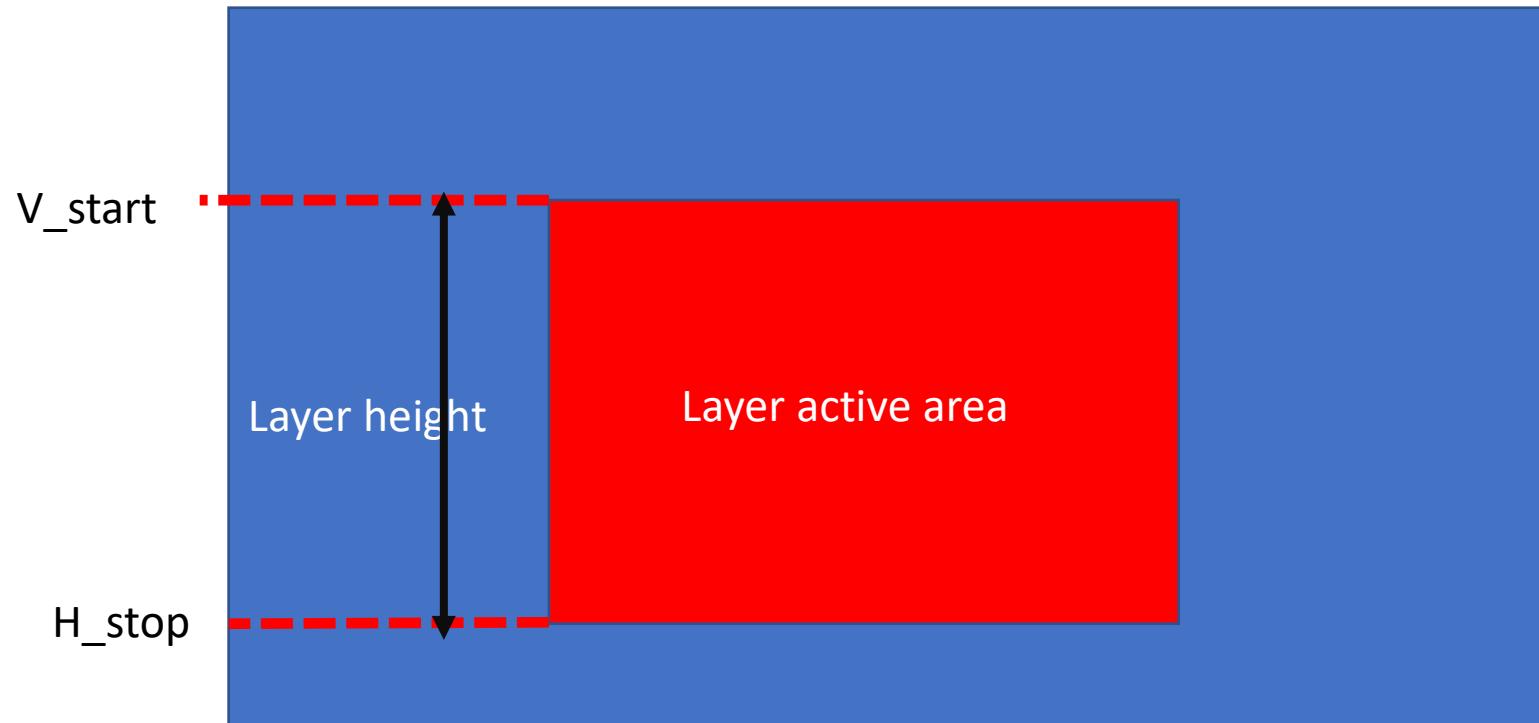


$$\text{WHSTPOS (Start)} = \text{AHBP} + \text{H_start} + 1$$

$$\text{WHSPPPOS (Stop)} = \text{AHBP} + \text{H_start} + \text{Layer_width} + 1 \quad (<= \text{AAW})$$

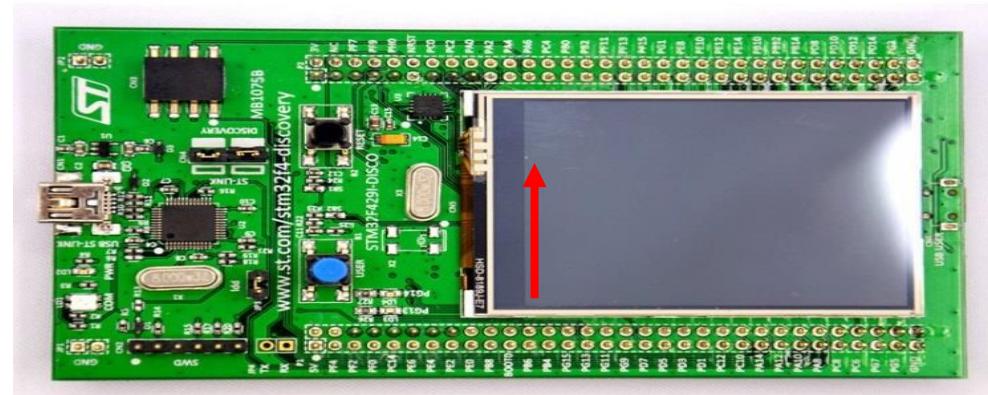
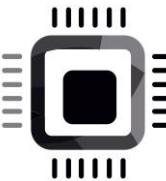


Layer window vertical position configuration items

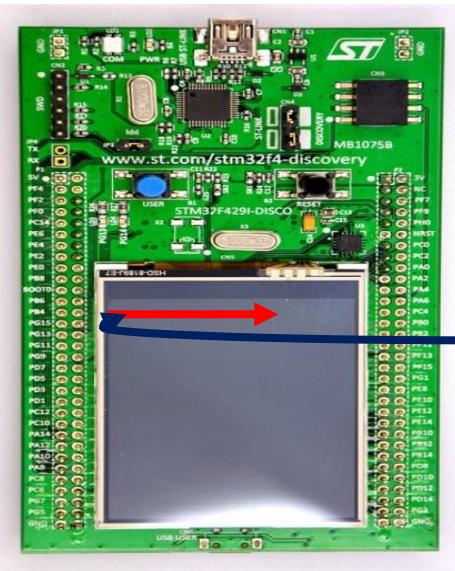


`WVSTPOS (Start) = AVBP + V_start + 1`

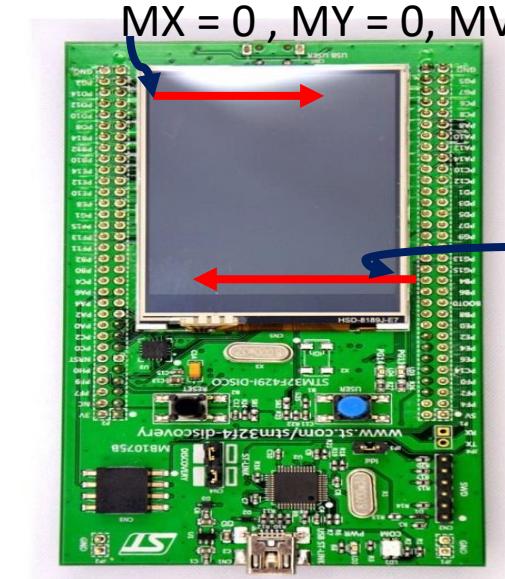
`WVSPPPOS (Stop) = AVBP + V_start + Layer_height+ 1 (<= AAH)`



$MX = 0, MY = 1, MV = 1$



$MX = 1, MY = 1, MV = 0$



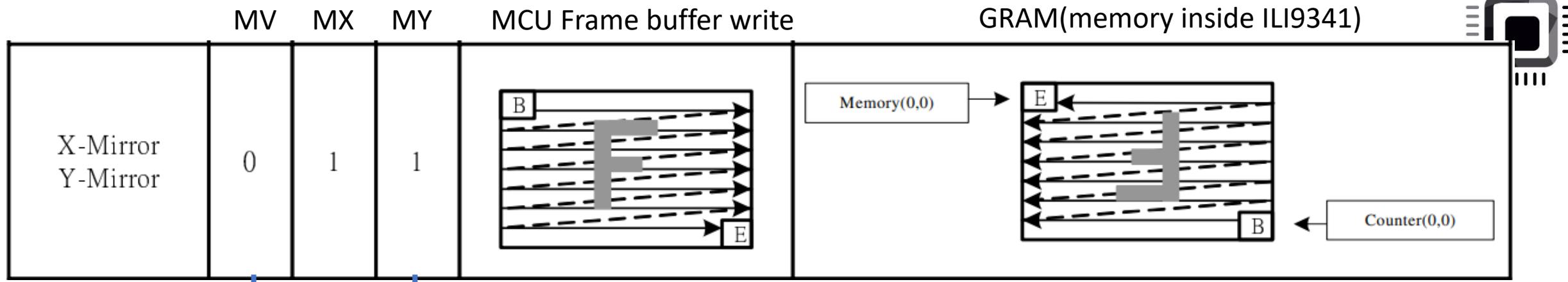
$MX = 0, MY = 0, MV = 0$

$MX = 1, MY = 1, MV = 0$

Portrait-1

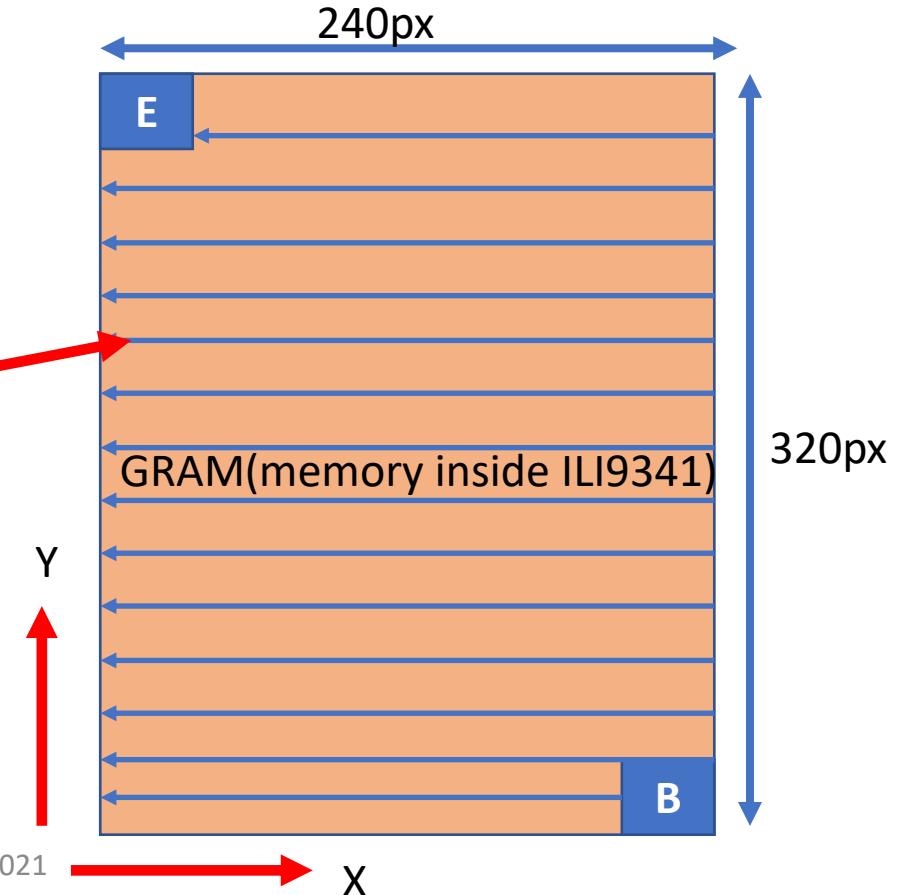
Portrait-2

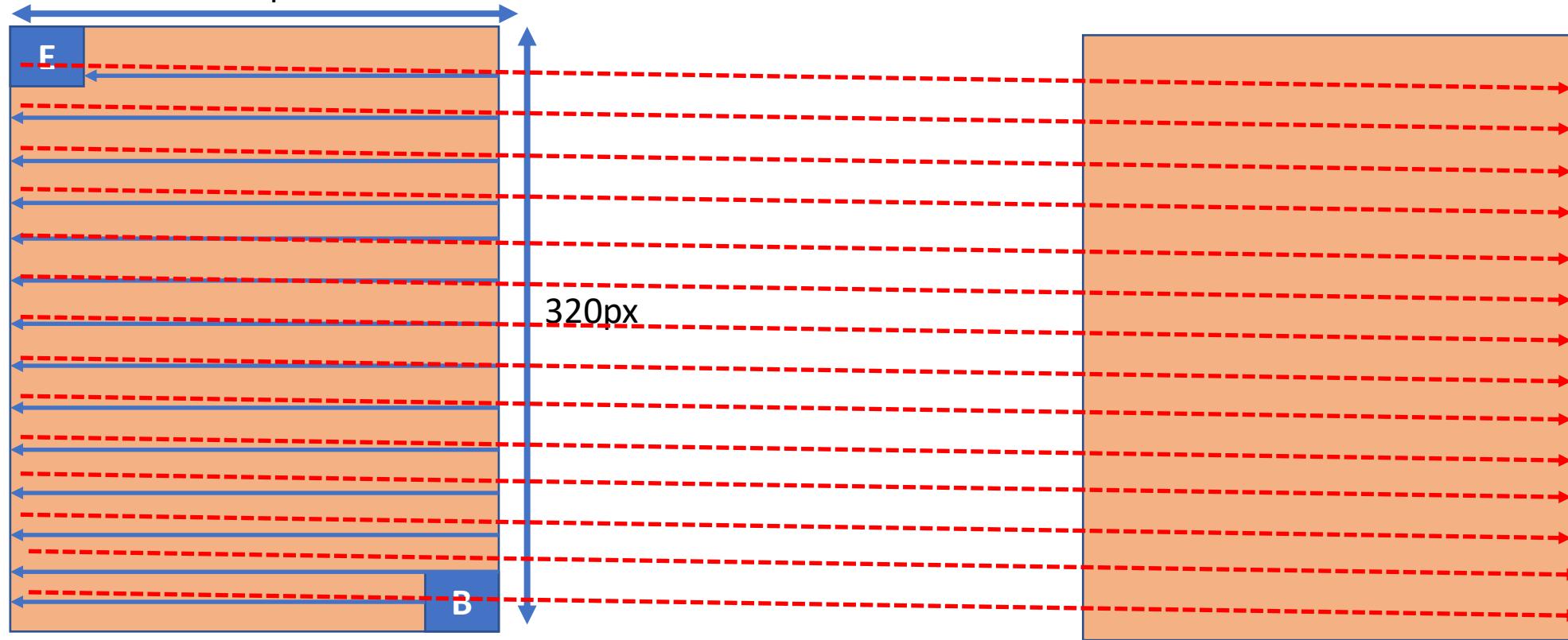
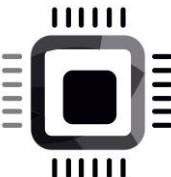
Direction of pixel display



This configuration can be used for Portrait

Memory write direction



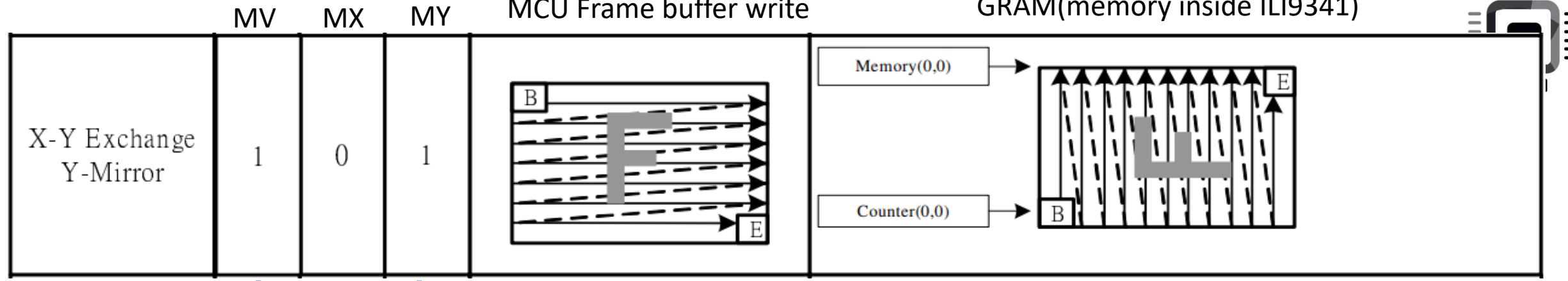


GRAM(memory inside ILI9341)

Display Glass

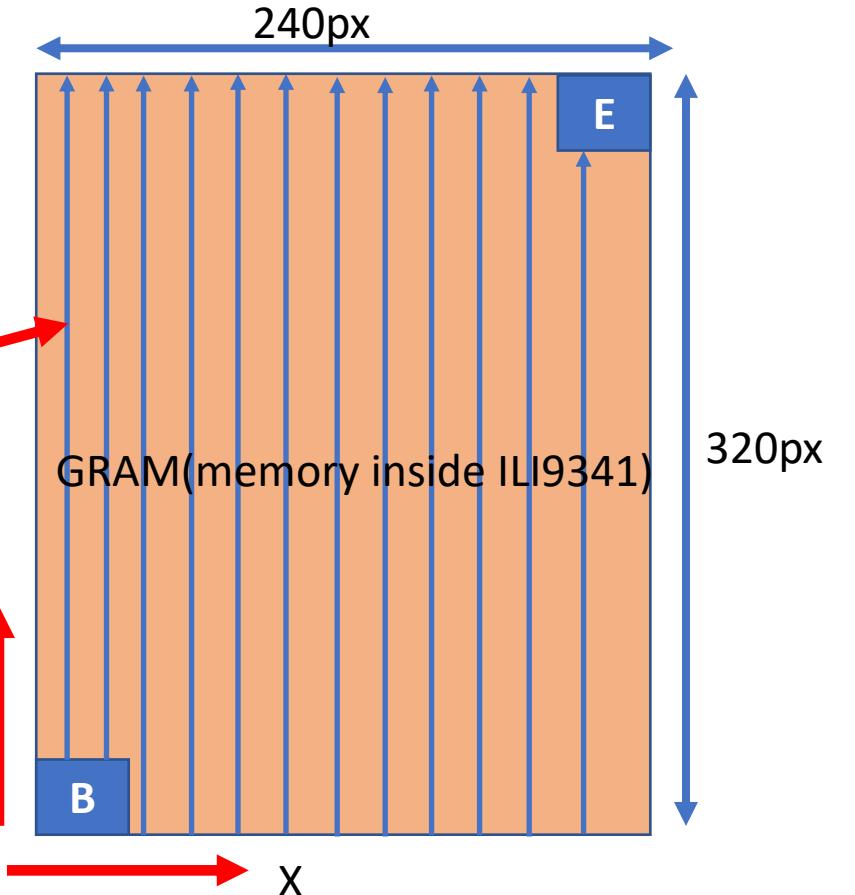
→ GRAM filling up as pixel bytes are written to LCD for every HSYNC strobe

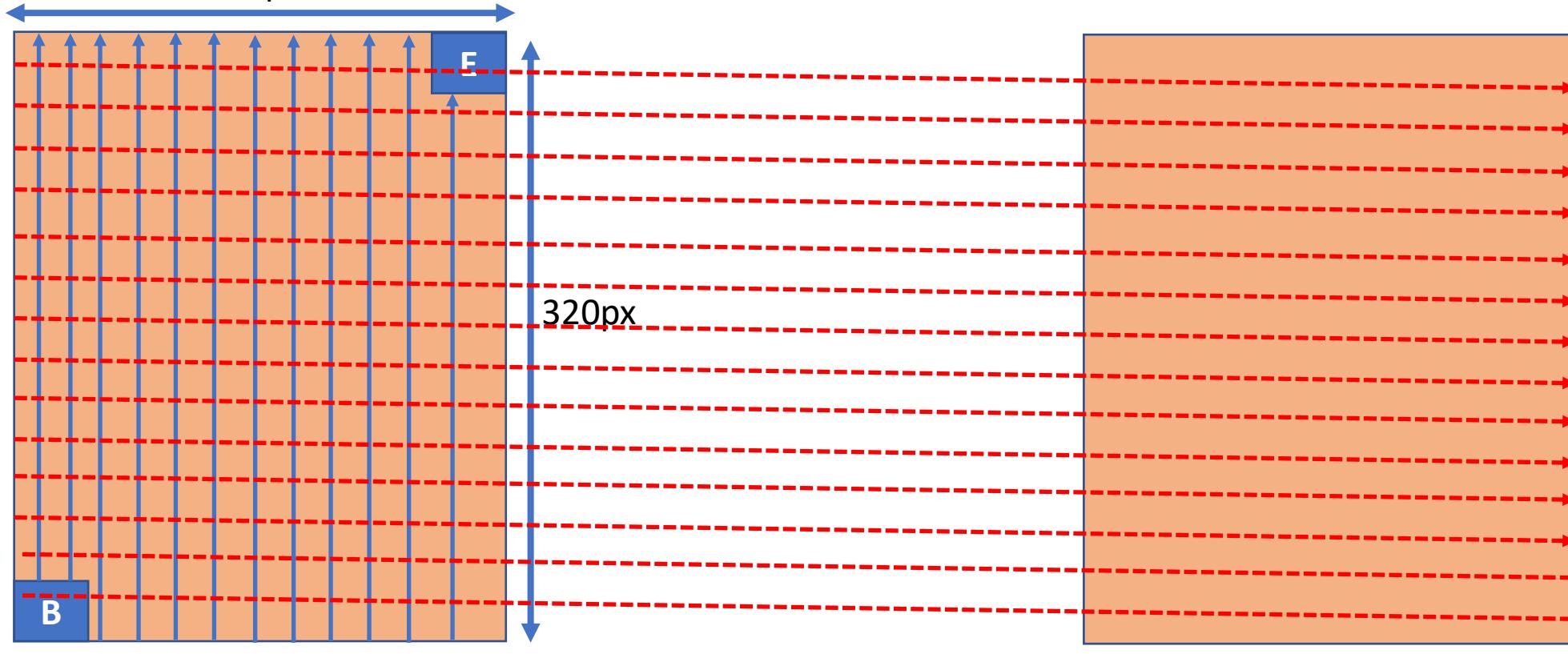
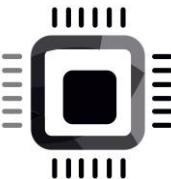
→ GRAM to display mapping for every HSYNC strobe



This configuration can be used for Landscape

Memory write direction



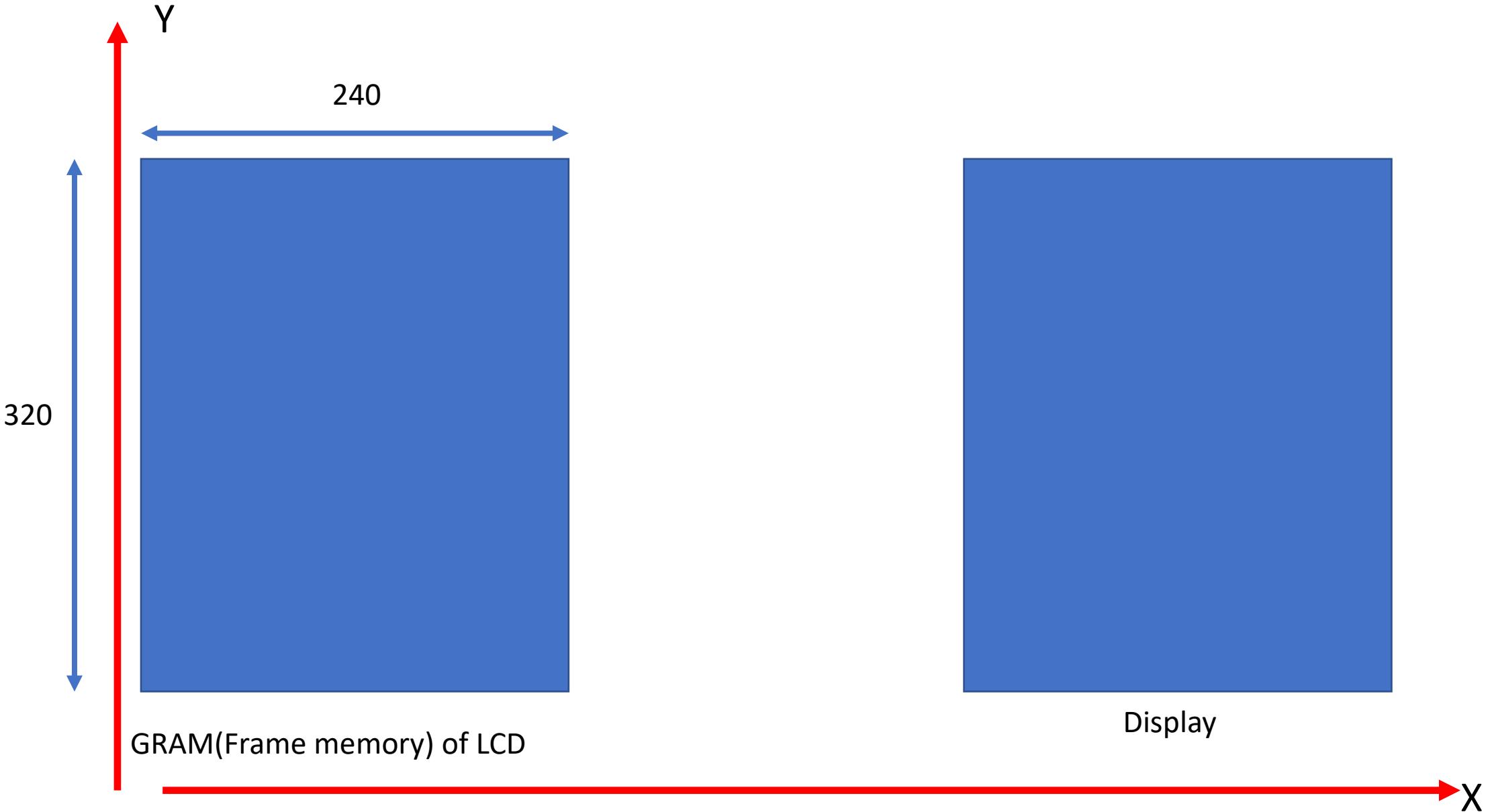
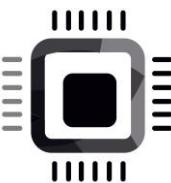


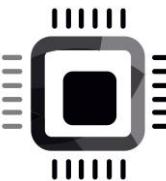
GRAM(memory inside ILI9341)

Display Glass

→ GRAM filling up as pixel bytes are written to LCD for every HSYNC strobe

→ GRAM to display mapping for every HSYNC strobe

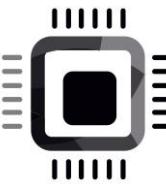




```
void LCD_Write_Data(uint8_t *buffer,uint32_t len)
{
    SPI_TypeDef *pSPI = SPI;
    for(uint32_t i = 0 ; i < len ;i++){
        LCD_CSX_LOW();
        while(!REG_READ_BIT(pSPI->SR,SPI_SR_TXE_Pos));
        REG_WRITE(pSPI->DR,buffer[i]);
        while(!REG_READ_BIT(pSPI->SR,SPI_SR_TXE_Pos));
        while(REG_READ_BIT(pSPI->SR,SPI_SR_BSY_Pos));
        LCD_CSX_HIGH();
    }
}
```

```
void lcd_write_data(uint8_t *buffer,uint32_t len)
{
    SPI_TypeDef *pSPI = SPI;
    LCD_CSX_LOW();
    for(uint32_t i = 0 ; i < len ;i++){
        while(!REG_READ_BIT(pSPI->SR,SPI_SR_TXE_Pos));
        REG_WRITE(pSPI->DR,buffer[i]);
    }
    while(!REG_READ_BIT(pSPI->SR,SPI_SR_TXE_Pos));
    while(REG_READ_BIT(pSPI->SR,SPI_SR_BSY_Pos));
    LCD_CSX_HIGH();
}
```

Function used to send LCD command parameters



Pixel writing in the case of STM32F407x + external LCD

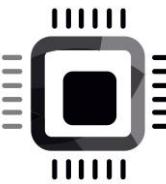
There are two methods

1) Non-DMA based data write over SPI (Processor involved)

- Check the function `void bsp_lcd_write(uint8_t *buffer, uint32_t nbytes)` in `bsp_lcd.c`

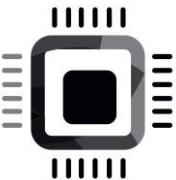
2) DMA based data write over SPI

- Check the function `void bsp_lcd_write_dma(uint32_t src_addr, uint32_t nbytes)` in `bsp_lcd.c`

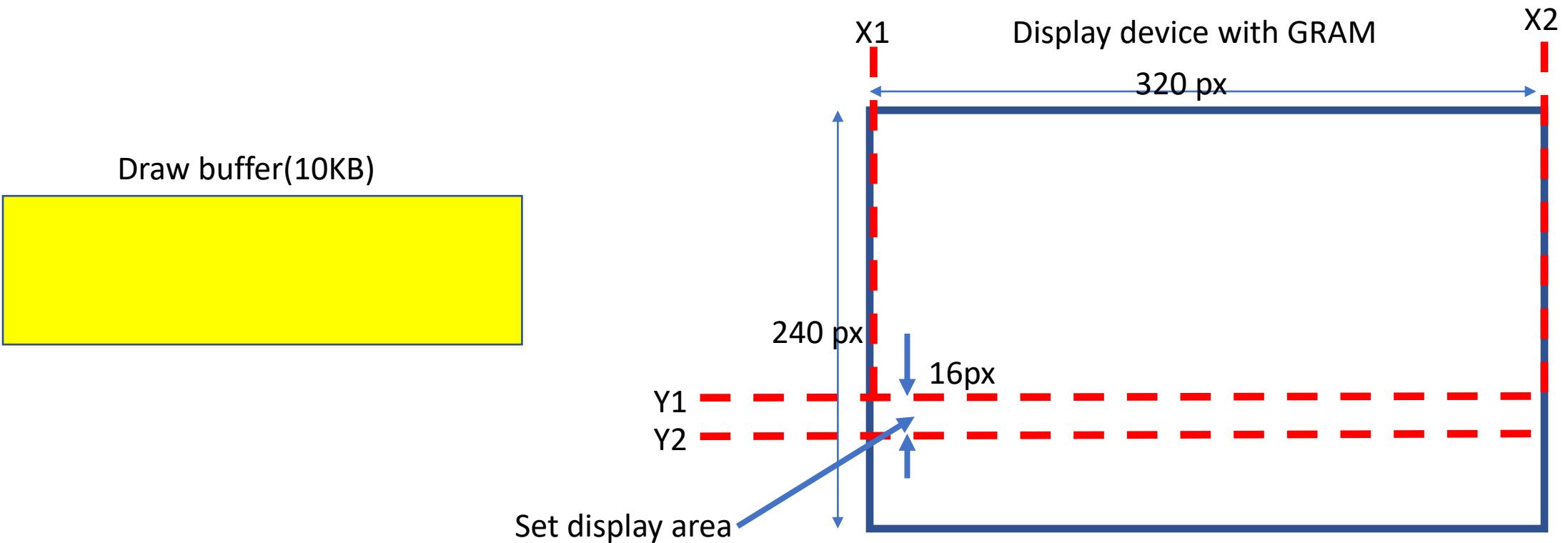


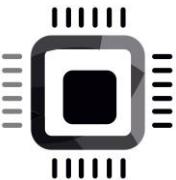
Pixel buffer

- In the case of STM32F407x + external LCD, you cannot use a screen-sized frame buffer like in the case of STM32F429 disc board.
 - On chip RAM of STM32F407x (168KB) < (320 x 240 x 2) Bytes (150KB)
- bsp_lcd.c uses non screen sized buffer of 10KB (Draw buffer)

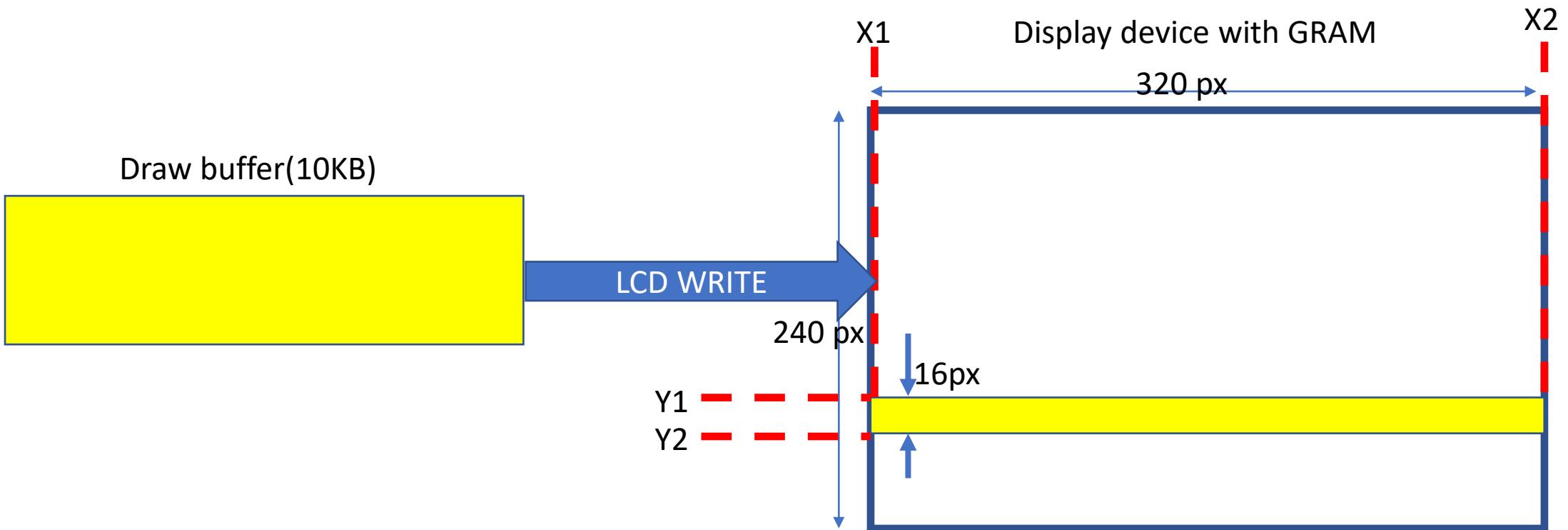


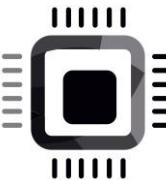
Pixel writing to display device





Pixel writing to display device



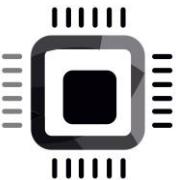


```
void bsp_lcd_set_display_area(uint16_t x1, uint16_t x2, uint16_t y1, uint16_t y2)
{
    lcd_area_t area;
    area.x1 = x1;
    area.x2 = x2;
    area.y1 = y1;
    area.y2 = y2;
    lcd_set_display_area(&area);
}

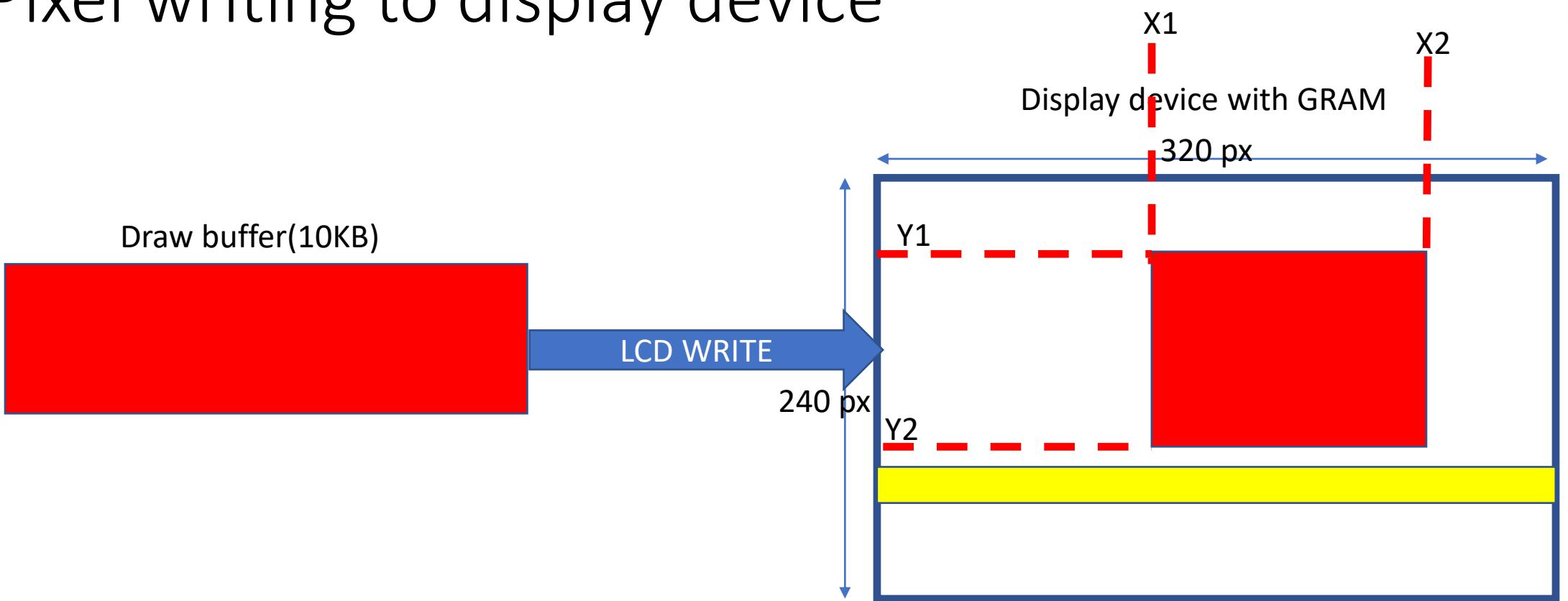
void lcd_set_display_area(lcd_area_t *area)
{
    uint8_t params[4];
    /*Column address set(2Ah) */
    params[0] = HIGH_16(area->x1);
    params[1] = LOW_16(area->x1);
    params[2] = HIGH_16(area->x2);
    params[3] = LOW_16(area->x2);
    lcd_write_cmd(ILI9341_CASET);
    lcd_write_data(params, 4);

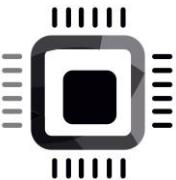
    params[0] = HIGH_16(area->y1);
    params[1] = LOW_16(area->y1);
    params[2] = HIGH_16(area->y2);
    params[3] = LOW_16(area->y2);
    lcd_write_cmd(ILI9341_RASET);
    lcd_write_data(params, 4);

}
```

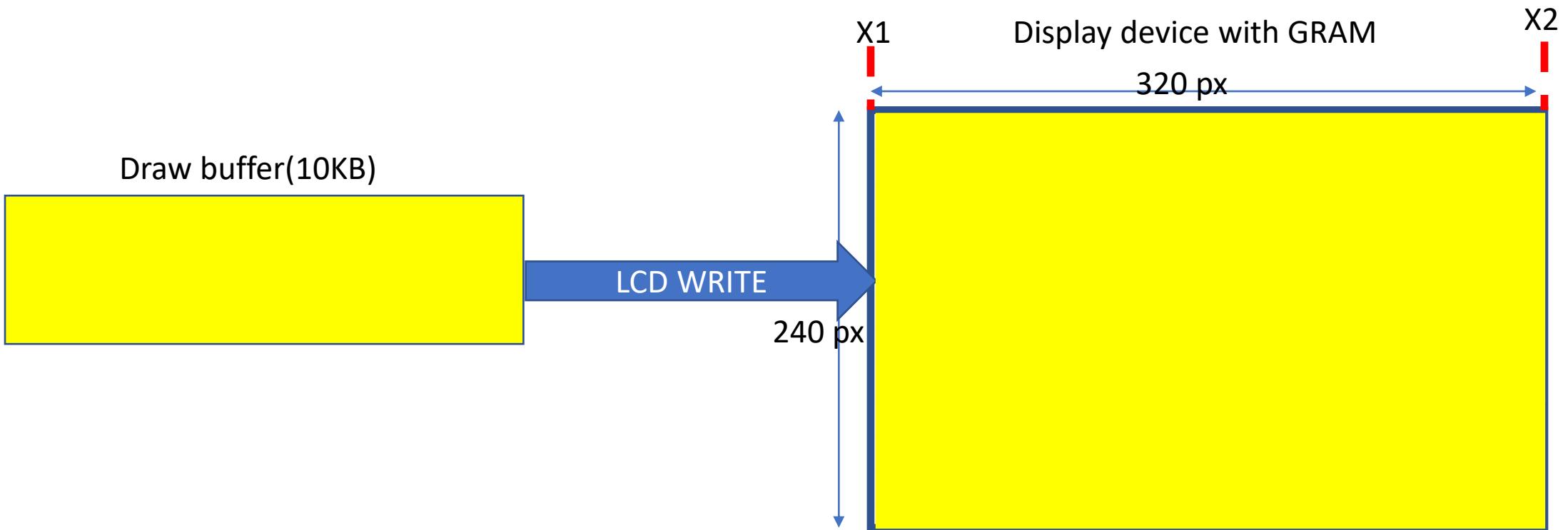


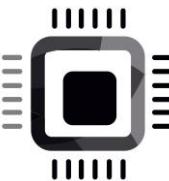
Pixel writing to display device



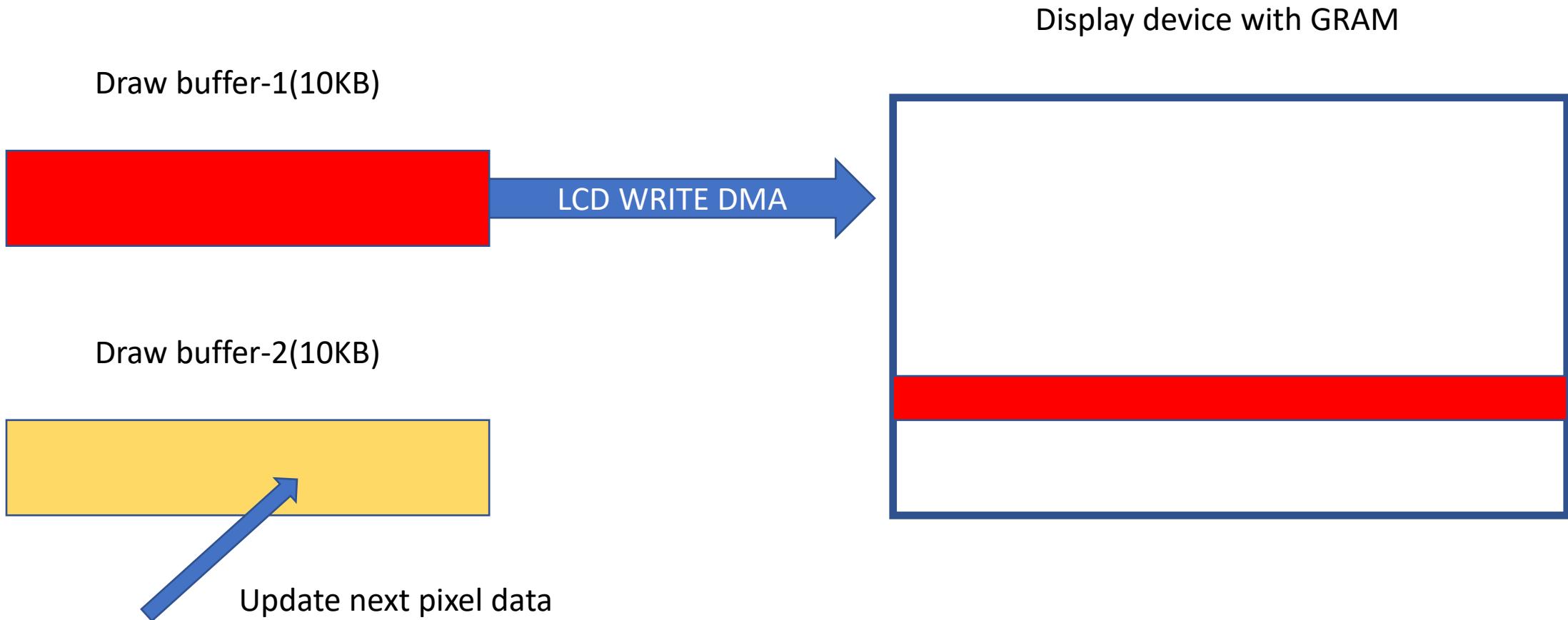


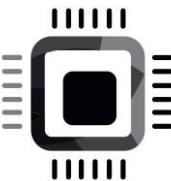
Pixel writing to display device



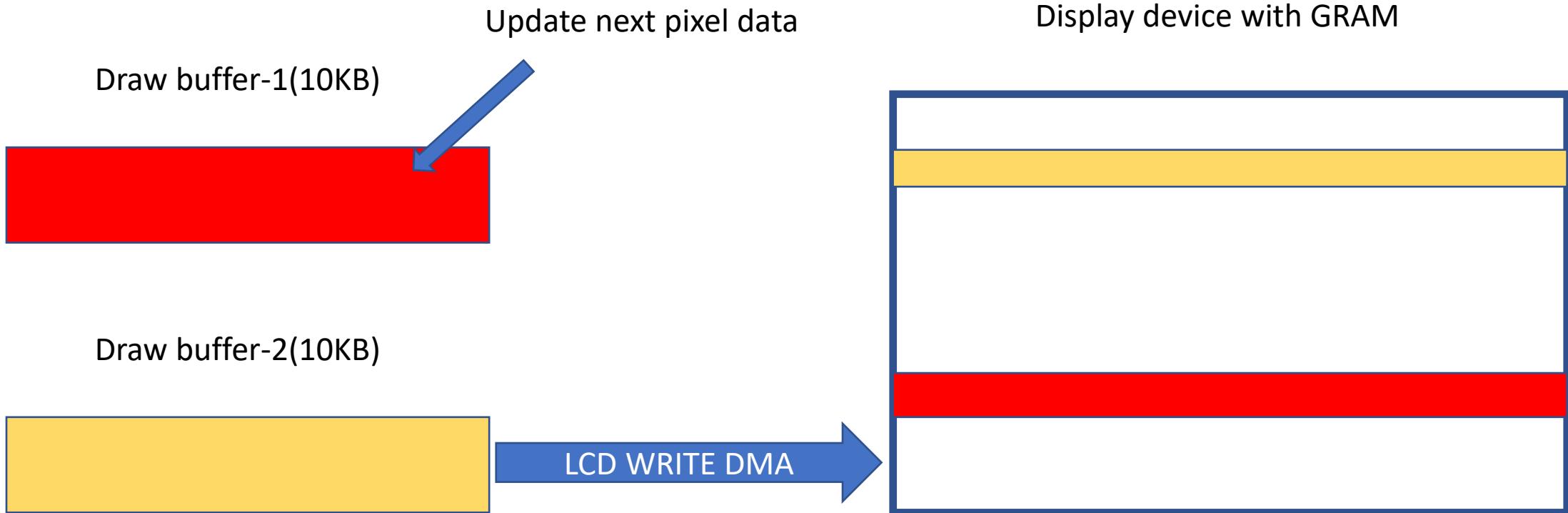


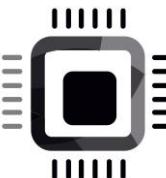
Dual buffering





Dual buffering





```
void bsp_lcd_write(uint8_t *buffer, uint32_t nbytes)
{
    uint16_t *buff_ptr;

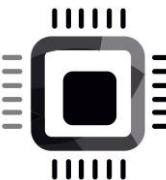
    __disable_spi();
    __spi_set_dff_16bit();
    __enable_spi();

    LCD_CSX_LOW();

    buff_ptr = (uint16_t*)buffer;
    while(nbytes)[
        while(!REG_READ_BIT(SPI->SR,SPI_SR_TXE_Pos));
        REG_WRITE(SPI->DR,*buff_ptr);
        ++buff_ptr;
        nbytes -= 2;
    ]

    __disable_spi();
    LCD_CSX_HIGH();
    __spi_set_dff_8bit();
    __enable_spi();

}
```



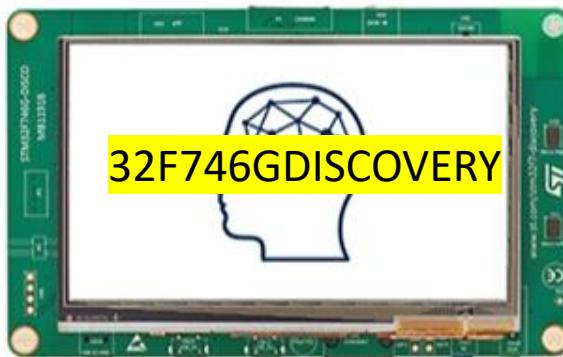
ROCKTECH

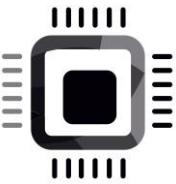
RK043FN48H-CT672B

7.3 Parallel RGB Data Format

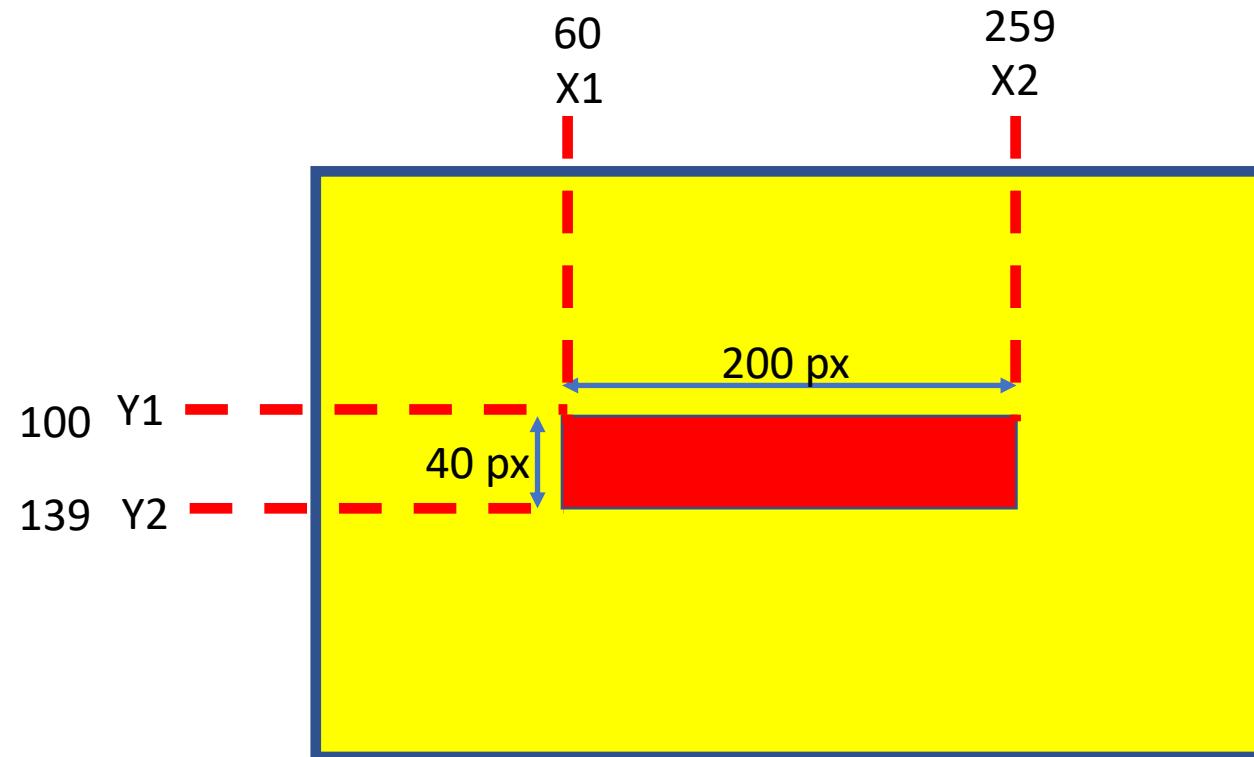
7.3.1 Parallel RGB Input Timing Table

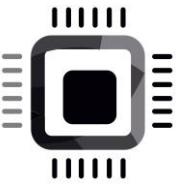
Item		Symbol	Min.	Typ.	Max.	Unit	
DCLK Frequency		Fclk	5	9	12	MHz	
DCLK Period		Tclk	83	110	200	ns	
Hsync	Period Time	Th	490	531	605	DCLK	
	Display Period	Thdisp		480		DCLK	
	Back Porch	Thbp	8	43		DCLK	By H_BLANKING setting
	Front Porch	Thfp	2	8		DCLK	
	Pulse Width	Thw	1	(51)		DCLK	
Vsync	Period Time	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 (16)		H	



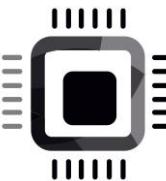


STM32F407x + external LCD





LVGL



Running LVGL on Simulator and STM32 microcontroller

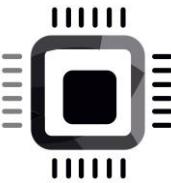
- About LVGL
- Using Simulator
- Executing LVGL sample applications on the simulator
- Adding LVGL source code to the STM32 project
- Adding LCD TFT and Touchscreen driver support
- Executing LVGL sample applications on target hardware

Source for all the discussion in this section

LVGL Documentation 8.3

<https://docs.lvgl.io/master/index.html>

About LVGL



← → C ⌂ lvgl.io

⠇ ⭐



Features Demos Boards Services ▾ Developers Tools ▾ About

Light and Versatile Graphics Library

LVGL is an open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.



Github



Release notes

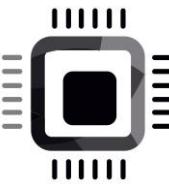


Docs



Forum





LVGL Simulator

<https://lvgl.io/developers>

Windows : Use visual studio based simulator

Linux/MAC : User Eclipse based simulator

Visual Studio Installer

Modifying — Visual Studio Community 2022 — 17.2.4

Workloads Individual components Language packs Installation locations

Web & Cloud (4)

-  **ASP.NET and web development**
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
-  **Python development**
Editing, debugging, interactive development and source control for Python.
-  **Azure development**
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
-  **Node.js development**
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

-  **Mobile development with .NET**
Build cross-platform applications for iOS, Android or Windows using Xamarin.
-  **.NET desktop development**
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
-  **Desktop development with C++**
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
-  **Universal Windows Platform development**
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Location
C:\Program Files\Microsoft Visual Studio\2022\Community

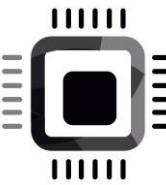
By continuing, you agree to the [license](#) for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the [3rd Party Notices](#) or in its accompanying license. By continuing, you also agree to those licenses.

Installation details

▼ Desktop development with C++
 ▼ Included
 ✓ C++ core desktop features
 ▼ Optional
 ✓ MSVC v143 - VS 2022 C++ x64/x86 build t...
 ✓ Windows 10 SDK (10.0.19041.0)
 ✓ Just-In-Time debugger
 ✓ C++ profiling tools
 ✓ C++ CMake tools for Windows
 ✓ C++ ATL for latest v143 build tools (x86 &...
 ✓ Test Adapter for Boost.Test
 ✓ Test Adapter for Google Test
 ✓ Live Share
 ✓ IntelliCode
 ✓ C++ AddressSanitizer
 □ MSVC v143 - VS 2022 C++ ARM64 build t...
 □ C++ MFC for latest v143 build tools (x86...
 □ C++/CLI support for v143 build tools (Late...
 □ C++ Modules for v143 build tools (x64/x8...
 □ C++ Clang tools for Windows (13.0.1 - x64...
 □ JavaScript diagnostics

Total space required 0 B

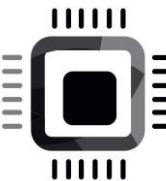
Install while downloading ▾ Close



Adding lvgl to STM32 project

We will do the following,

- 1) Add lvgl source files and lvgl.h
- 2) Add lvgl example source files
- 3) Add the tft and touchscreen controller drivers
- 4) Add lvgl_conf.h
- 5) Add include path settings in the IDE
- 6) Turn off DMA2D and SDRAM configuration code (We don't use these peripherals in this project)
- 7) Call lvgl_init() function before you call any the lvgl apis
- 8) Register display driver and input device driver(touchscreen) with lvgl
- 9) Call lv_tick_inc(x) every x milliseconds in an interrupt to report the elapsed time to LVGL
- 10) Call lv_timer_handler() every few milliseconds to handle LVGL related tasks



By using 2 draw buffers rendering
and refreshing of the display
become parallel operations

Draw buffer_1

This need not be screen sized

Draw buffer_2(optional)

This need not be screen sized

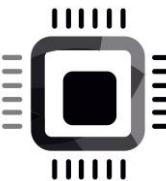
tft_flush

Frame buffer

Screen sized

LVGL uses these buffers to update
graphic elements which
Supposed to be drawn on the
screen

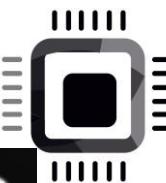
Display driver file (tft.c)



LVGL object

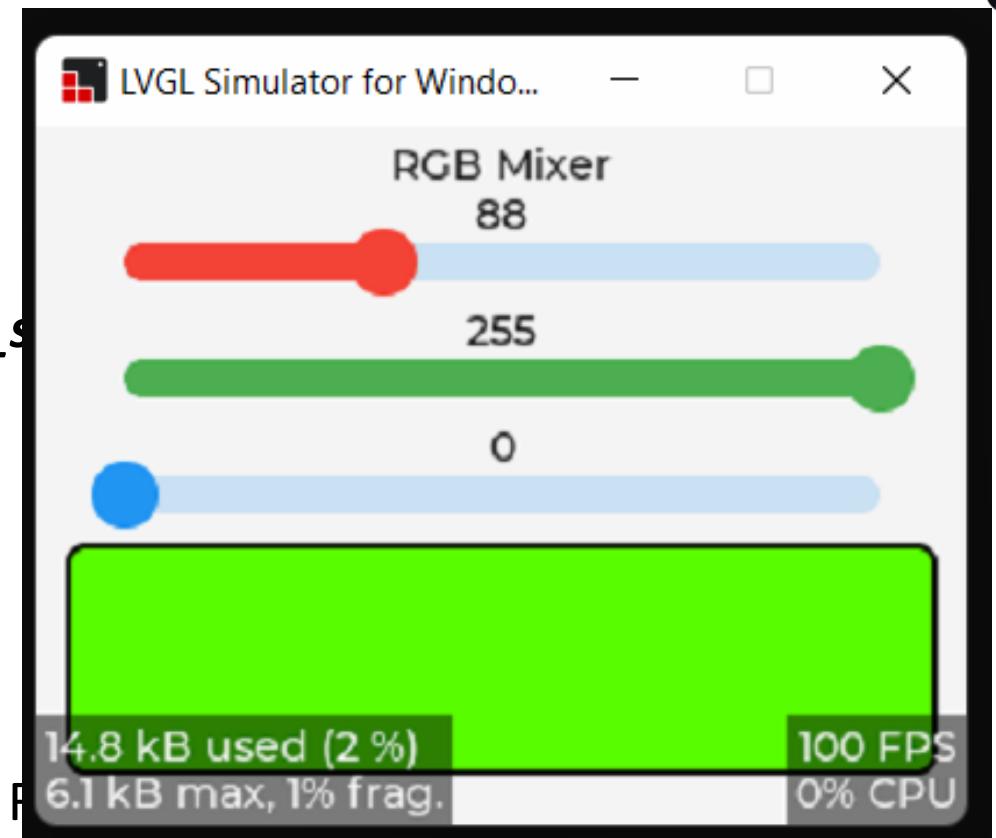
- In LVGL, the basic building blocks of a user interface are the objects, also called Widgets
- For example, a Button, Label, Image, List, Chart or Text area or just a base object
- All objects are referenced using a *lv_obj_t* pointer as a handle
- This pointer can later be used to set or get the object's attributes.
- For example, **button** and **label** both are lvgl objects of type *lv_obj_t*, but **button** objects may carry some unique attributes that **label** objects may not have.
- '**Base object**' is also a lvgl object or a widget that doesn't carry any special attributes.

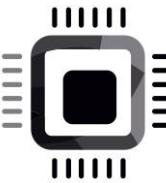
Source : LVGL documentation v8.3



Attributes of the object

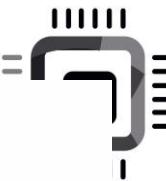
- Basic attributes
 - Applies to all objects of type *lv_obj_t*
 - You can set/get these attributes with *lv_obj_set_attr()*
 - Position
 - Size
 - Parent
 - Styles
 - Event handlers
- Special attributes
 - The object types have special attributes too. For example:
 - Minimum and maximum values
 - Current value
 - For these special attributes, every object type may have unique API functions





A screen object

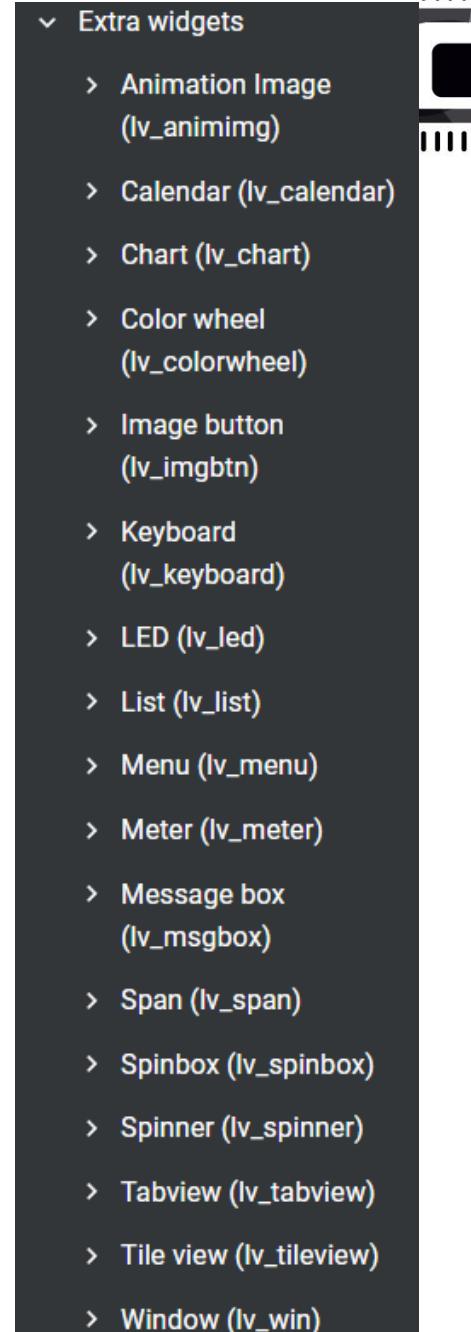
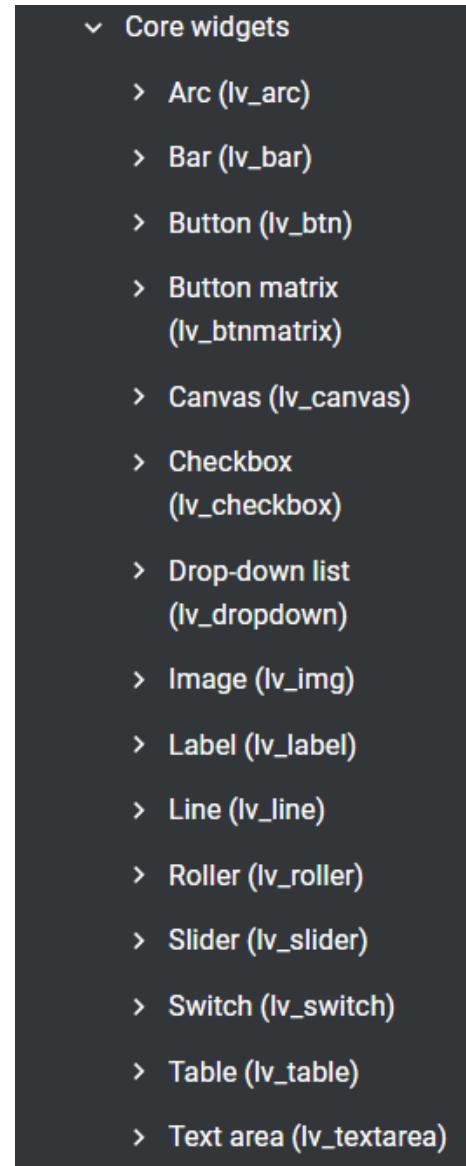
- When you register a display driver with lvgl, lvgl creates a ‘screen’ object of type ***lv_obj_t*** .
- This screen object is a parent object to which you can add your application-specific child objects.
- The screen object is associated with the display object of type ***lv_disp_t***
- There will be one display object for each physical display. For example, if you are using 2 physical displays, there may be 2 display objects. Lvgl provides dedicated APIs to change the properties of the display
 - ***lv_disp_set_rotation()***
 - ***lv_disp_set_default()***
 - ***lv_disp_get_default()*** and many
- Each display can have many screen objects, and current active one is accessed by the field ***act_scr*** of the display object
- You can manipulate screen objects by these APIs
 - Get the active screen ***lv_scr_act()***
 - Load screen ***lv_scr_load(scr1)***

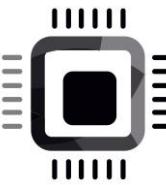


```
/**  
 * Display structure.  
 * @note `lv_disp_drv_t` should be the first member of the structure.  
 */  
typedef struct _lv_disp_t {  
    /**< Driver to the display*/  
    struct _lv_disp_drv_t * driver;  
  
    /**< A timer which periodically checks the dirty areas and refreshes them*/  
    lv_timer_t * refr_timer;  
  
    /**< The theme assigned to the screen*/  
    struct _lv_theme_t * theme;  
  
    /** Screens of the display*/  
    struct _lv_obj_t ** screens;      /**< Array of screen objects.*/  
    struct _lv_obj_t * act_scr;     /**< Currently active screen on this display*/  
    struct _lv_obj_t * prev_scr;    /**< Previous screen. Used during screen animations*/  
    struct _lv_obj_t * scr_to_load; /**< The screen prepared to load in lv_scr_load_anim*/  
    struct _lv_obj_t * top_layer;   /**< @see lv_disp_get_layer_top*/  
    struct _lv_obj_t * sys_layer;   /**< @see lv_disp_get_layer_sys*/  
    uint32_t screen_cnt;  
    uint8_t del_prev : 1;           /**< 1: Automatically delete the previous screen when the screen load animation is ready*/  
  
    lv_opa_t bg_opa;              /**< Opacity of the background color or wallpaper*/  
    lv_color_t bg_color;          /**< Default display color when screens are transparent*/  
    const void * bg_img;         /**< An image source to display as wallpaper*/  
  
    /** Invalidated (marked to redraw) areas*/  
    lv_area_t inv_areas[LV_INV_BUF_SIZE];
```

Widgets

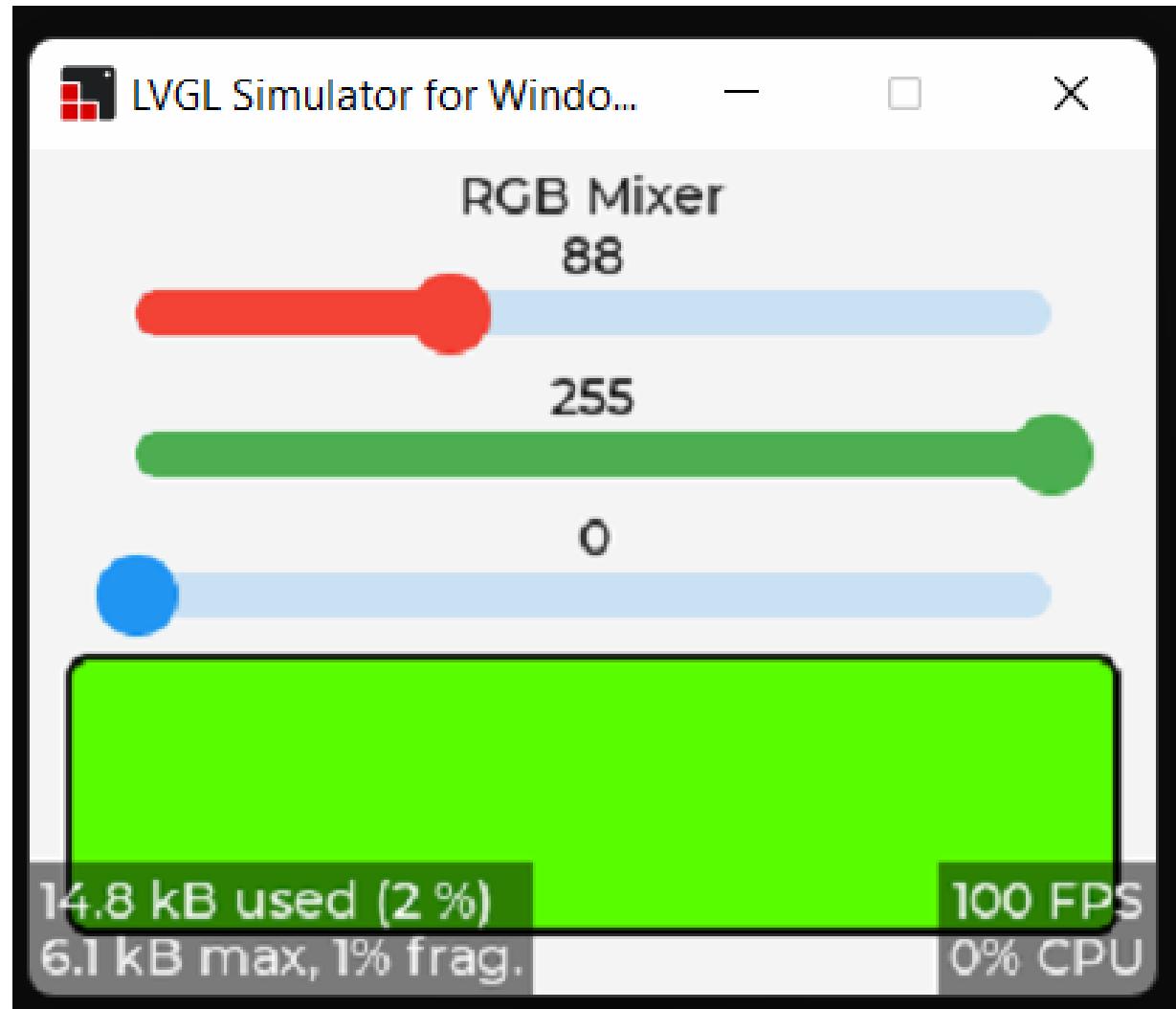
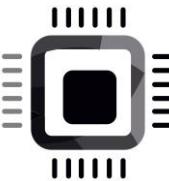
- All widgets are lvgl objects of type ***lv_obj_t***.
- A **base object** is also a widget which only has basic properties such as
 - coordinates
 - parent object
 - children
 - contains the styles
 - attributes like Clickable, Scrollable, etc
- In object-oriented thinking, a **base object** is like the base class from which all other objects in LVGL are inherited.
- The **base object** can be directly used as a simple widget: it's nothing more than a rectangle

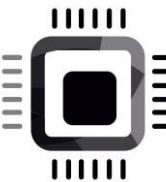




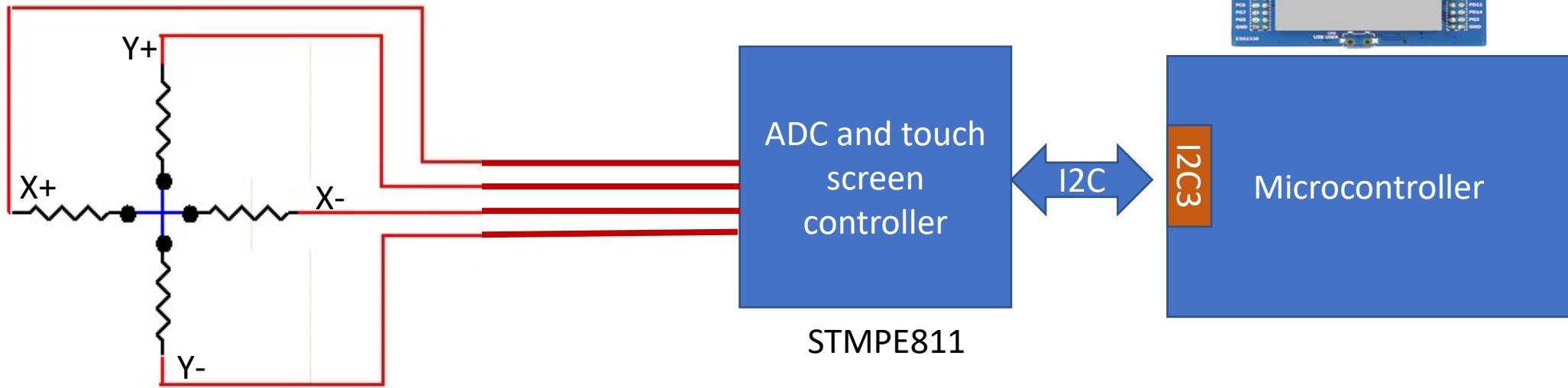
Exercise 003 : RGB mixer

1. Test using the simulator
2. Test on hardware
 - Create a new STM32 CubeIDE project
 - Add all the required peripherals using IDE's device configuration tool , LVGL and LVGL display , touchscreen drivers as explained in the previous exercise
 - Copy simulator tested code and test on the hardware



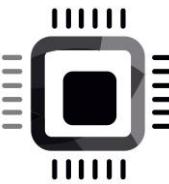


About touch screen controller

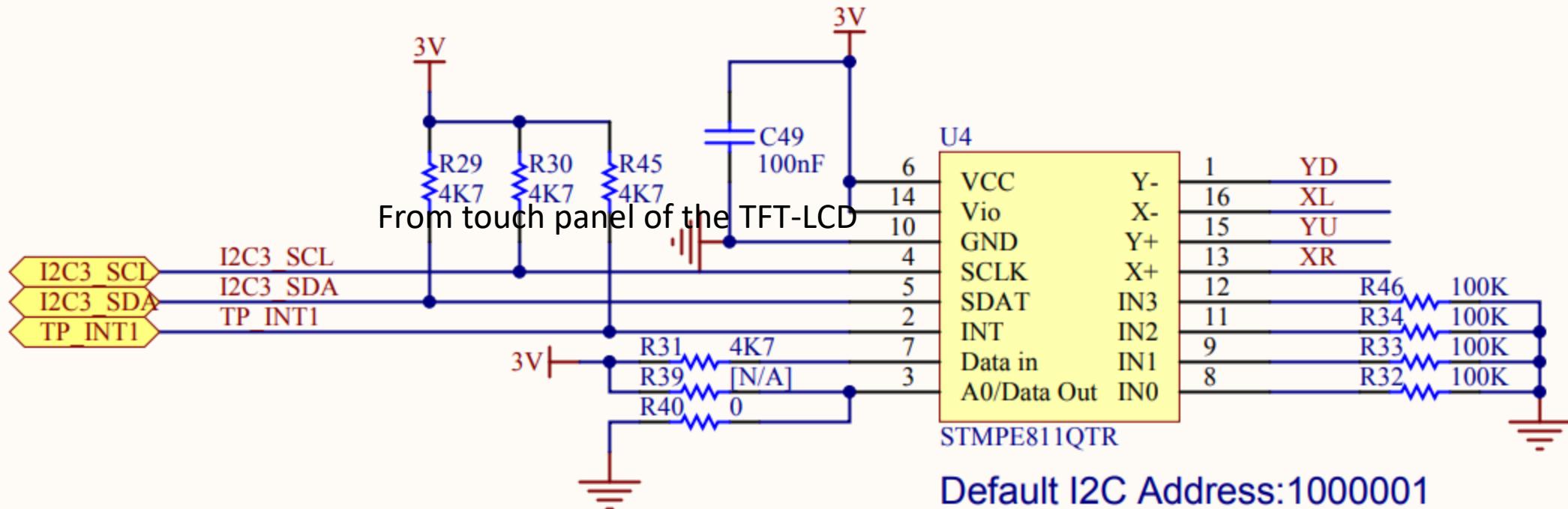


Analog 4 wire resistive touch panel

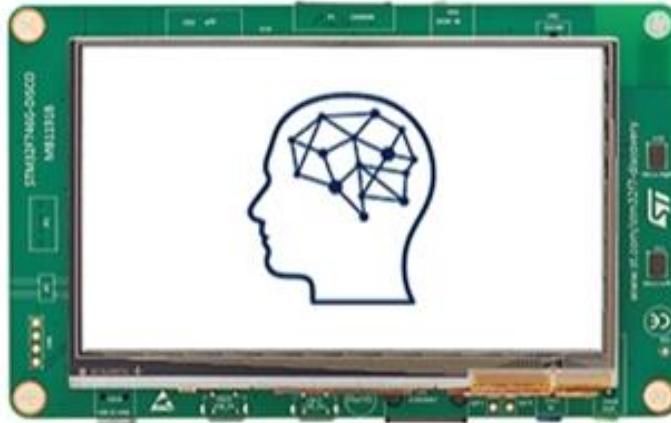
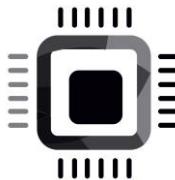
STM32F429 DISC board touch functionality



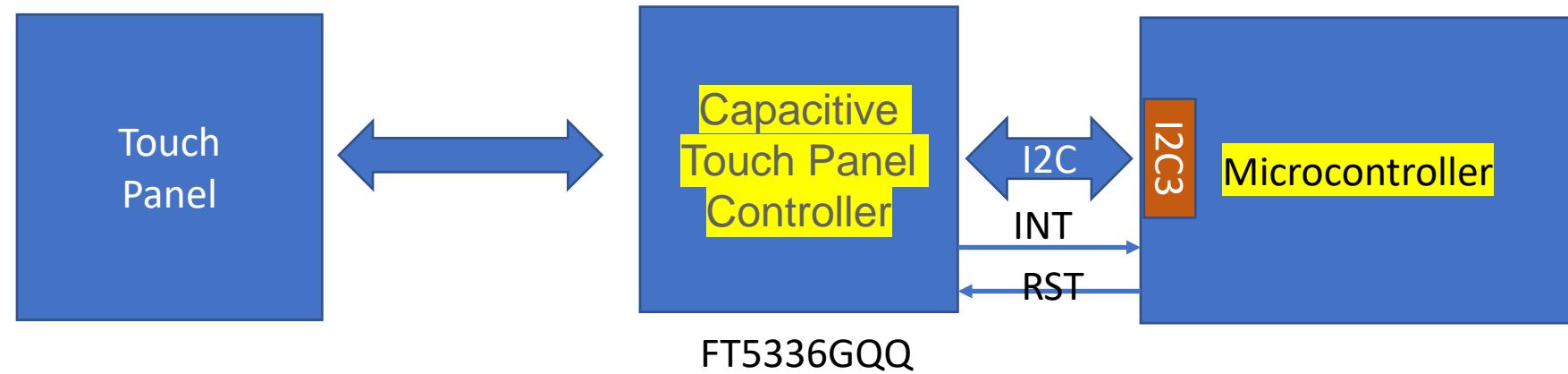
STM32F429 DISC board



On-board Touch screen controller

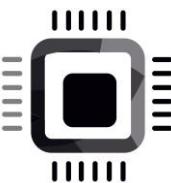


STM32F746 disc board

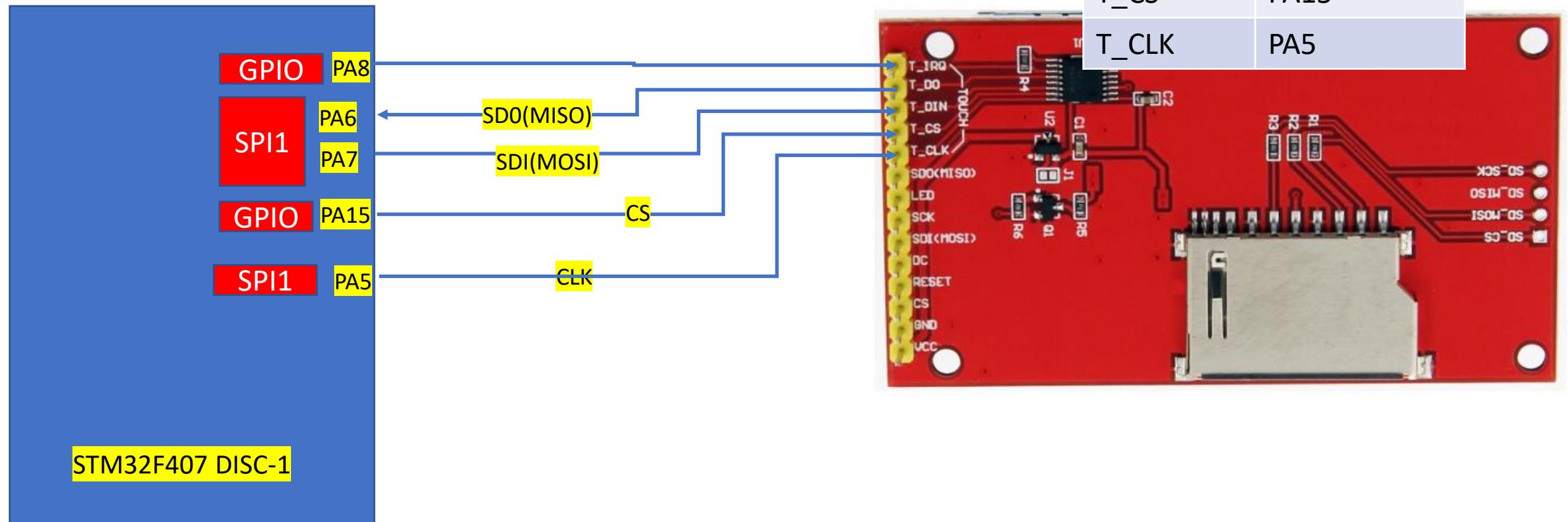


On-board Touch screen controller

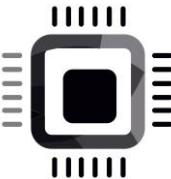
Touch screen interfacing



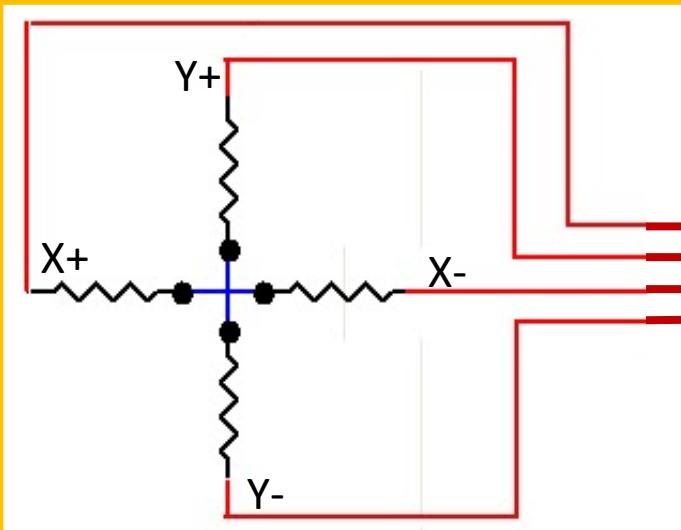
External LCD	STM32 board
T_IRQ	PA8
T_DO	PA6
T_DIN	PA7
T_CS	PA15
T_CLK	PA5



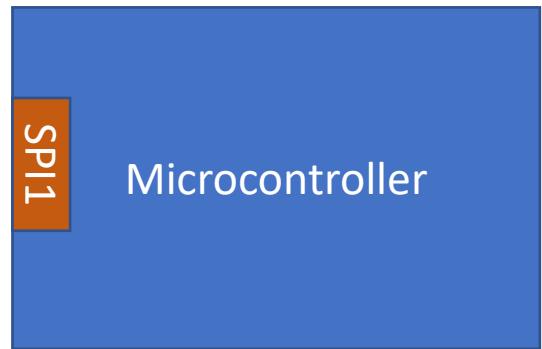
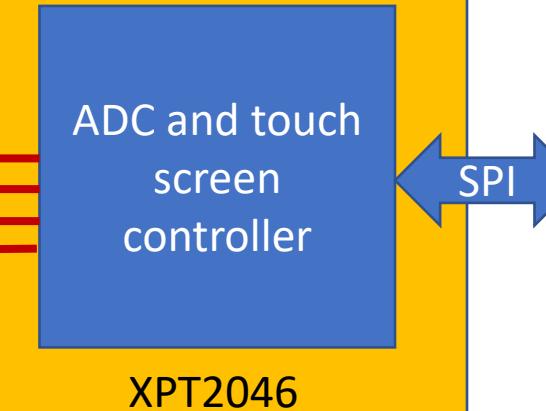
STM32F407 DISC-1 board and external RGB display interfacing



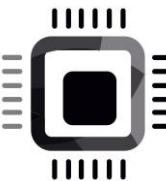
2.4 inch external TFT-LCD display with touchscreen controller



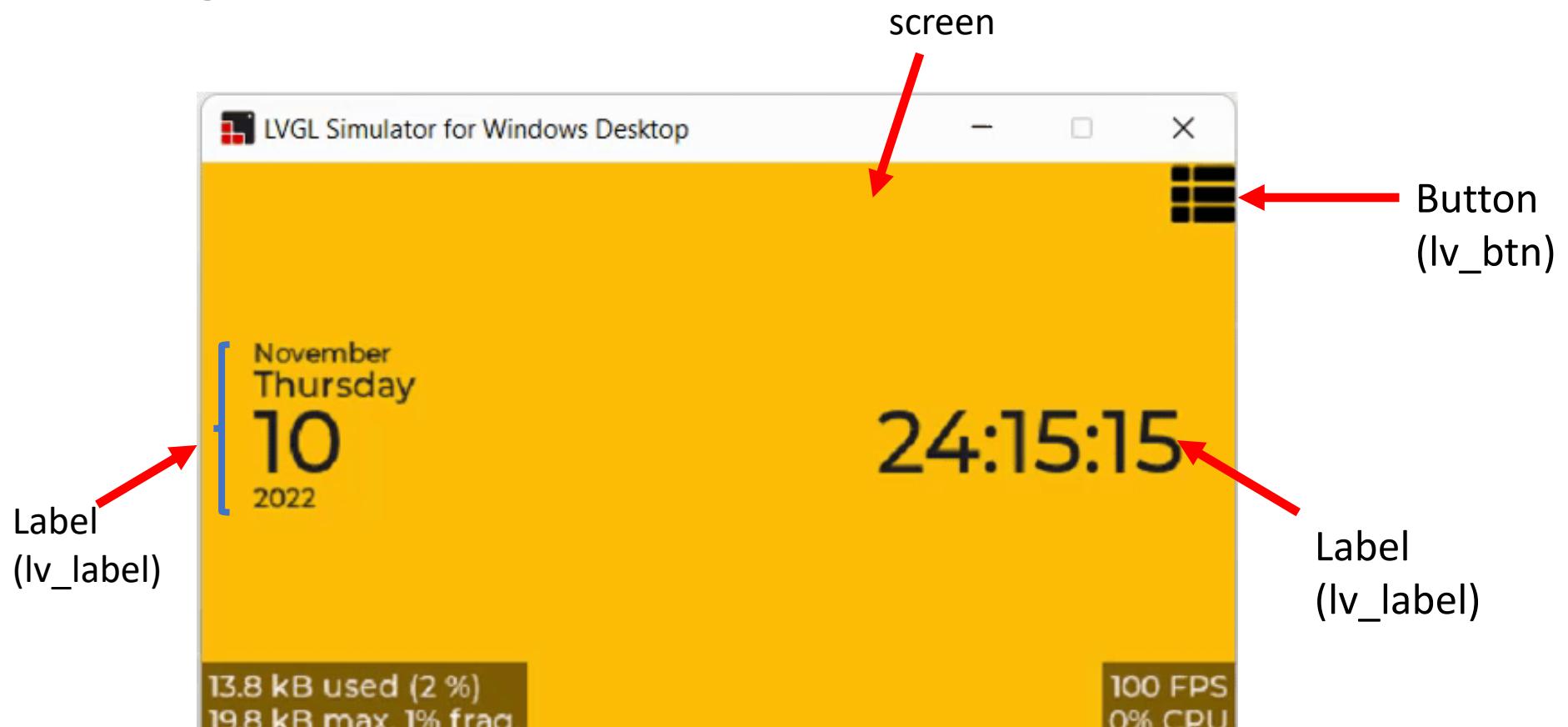
Analog 4 wire resistive touch panel



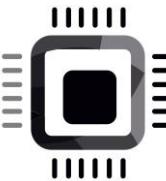
STM32F407x DISC board



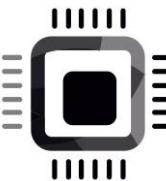
Exercise 004: Digital clock



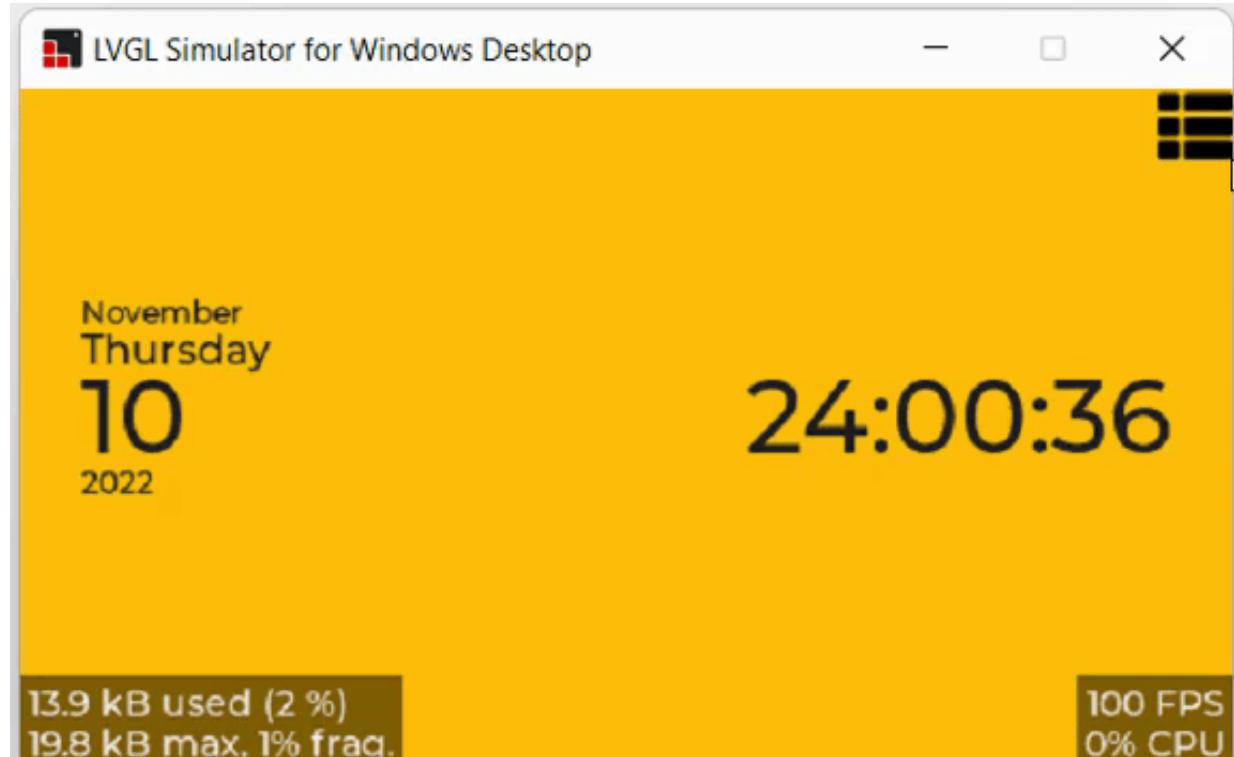
```
void Lvgl_main_page_set_style(Lvgl* const me) in glvgl.c
```



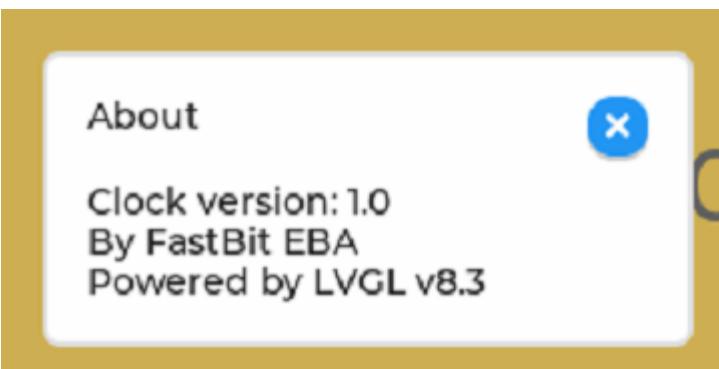
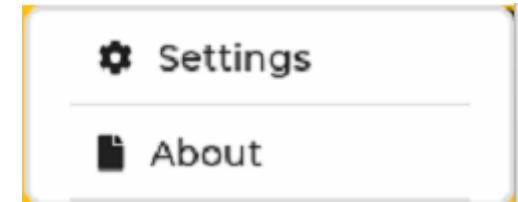
```
/*=====
 *   FONT USAGE
 *=====*/
/*Montserrat fonts with ASCII range and some symbols using bpp = 4
 *https://fonts.google.com/specimen/Montserrat*/
#define LV_FONT_MONTSERRAT_8      0
#define LV_FONT_MONTSERRAT_10     0
#define LV_FONT_MONTSERRAT_12     1
#define LV_FONT_MONTSERRAT_14     1
#define LV_FONT_MONTSERRAT_16     1
#define LV_FONT_MONTSERRAT_18     0
#define LV_FONT_MONTSERRAT_20     0
#define LV_FONT_MONTSERRAT_22     1
#define LV_FONT_MONTSERRAT_24     0
#define LV_FONT_MONTSERRAT_26     0
#define LV_FONT_MONTSERRAT_28     0
#define LV_FONT_MONTSERRAT_30     1
#define LV_FONT_MONTSERRAT_32     0
#define LV_FONT_MONTSERRAT_34     0
#define LV_FONT_MONTSERRAT_36     0
#define LV_FONT_MONTSERRAT_38     0
#define LV_FONT_MONTSERRAT_40     1
#define LV_FONT_MONTSERRAT_42     0
#define LV_FONT_MONTSERRAT_44     0
#define LV_FONT_MONTSERRAT_46     0
#define LV_FONT_MONTSERRAT_48     0
```



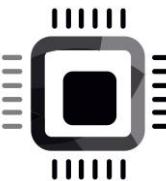
About info



List (lv_list)



Message box
(lv_msgbox)

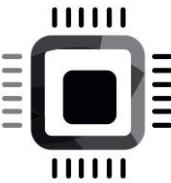


APIs used

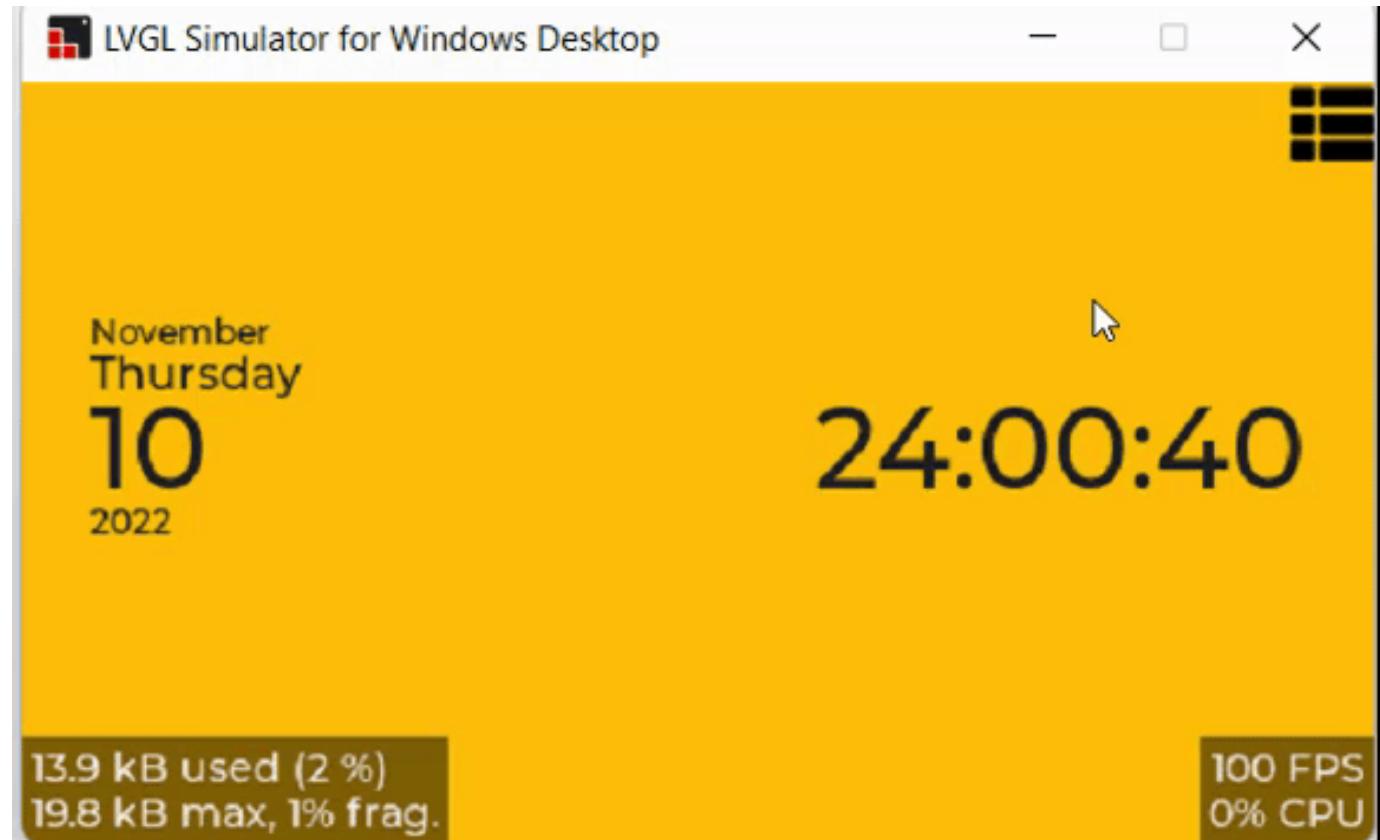
```
void event_handler_dropdown_btn(lv_event_t* e)
{
    lv_event_code_t code = lv_event_get_code(e);
    Lvgl* const lvgl_inst = (Lvgl*)lv_event_get_user_data(e);
    lv_obj_t* btn;

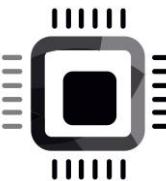
    if (code == LV_EVENT_CLICKED) {
        lvgl_inst->dropdwn_list = lv_list_create(parent:lvgl_inst->screen);
        btn = lv_list_add_btn(lvgl_inst->dropdwn_list, icon:LV_SYMBOL_SETTINGS, txt:"Settings");
        lv_obj_add_event_cb(obj:btn, event_cb:event_handler_dropdown_btn_opts_settings, filter:LV_EVENT_CLICKED, user_data:(void*)lvgl_inst);
        btn = lv_list_add_btn(lvgl_inst->dropdwn_list, icon:LV_SYMBOL_FILE, txt:"About");
        lv_obj_add_event_cb(obj:btn, event_cb:event_handler_dropdown_btn_opts_about, filter:LV_EVENT_CLICKED, user_data:(void*)lvgl_inst);
        lv_obj_set_height(obj:lvgl_inst->dropdwn_list, h:LV_SIZE_CONTENT);
        lv_obj_align(obj:lvgl_inst->dropdwn_list, LV_ALIGN_TOP_RIGHT, x_ofs:0, y_ofs:0);
    }
}
```

glvgl_widget.c

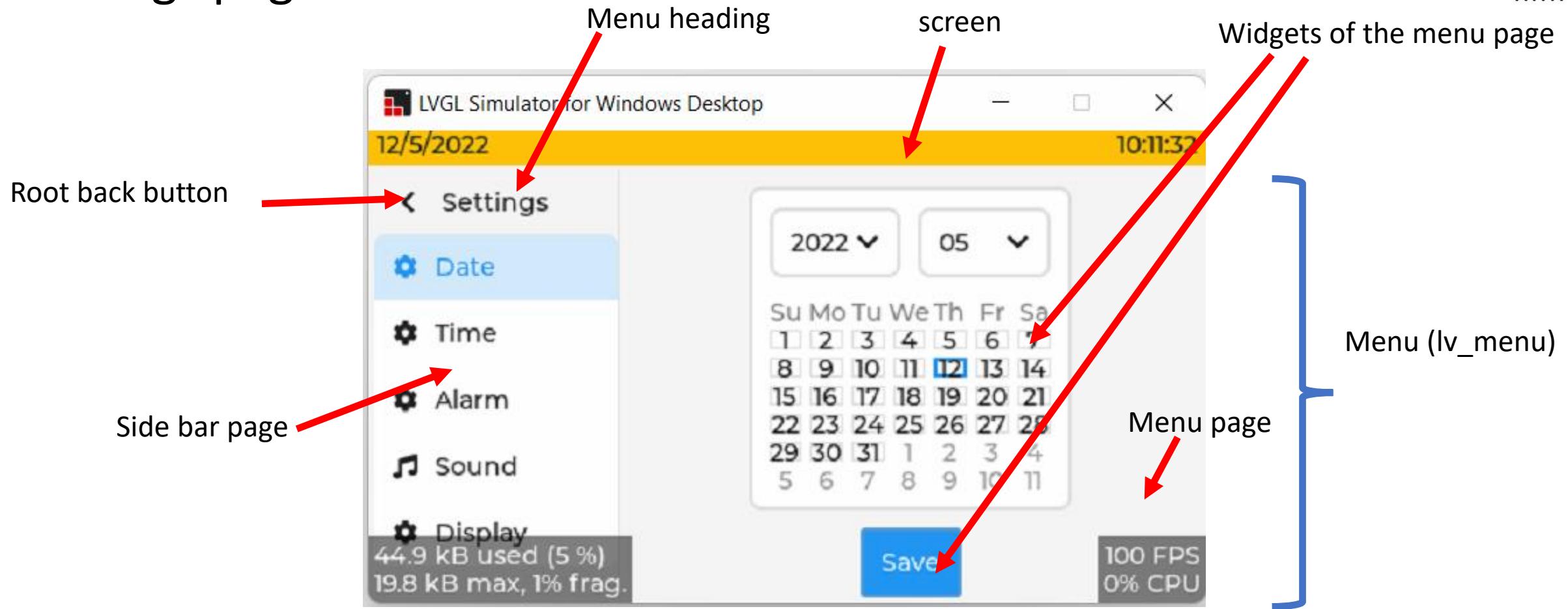


Settings page

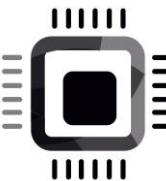




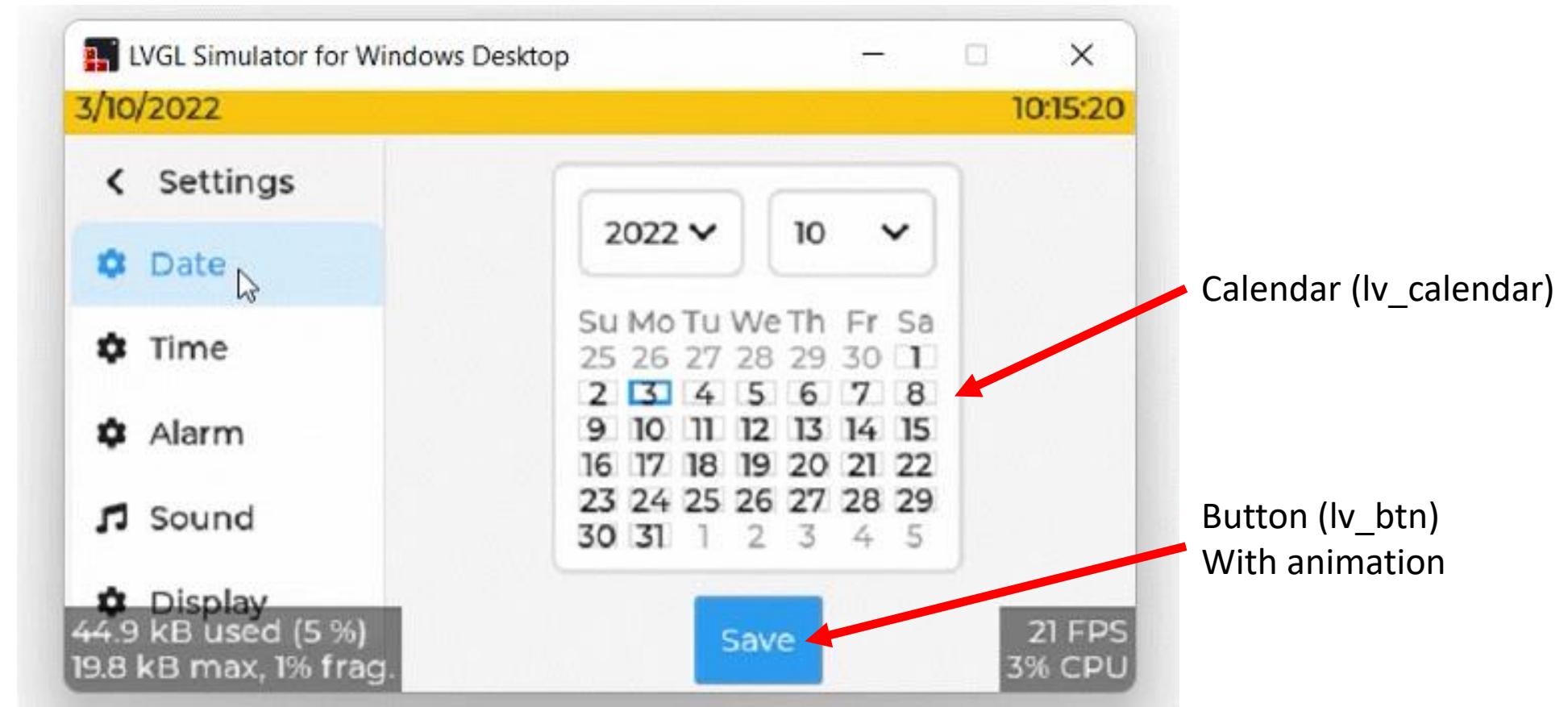
Settings page

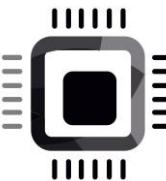


```
void Lvgl_create_setting_page(Lvgl * const me, lvgl_setting_page_data_t* data)
in glvgl.c
```

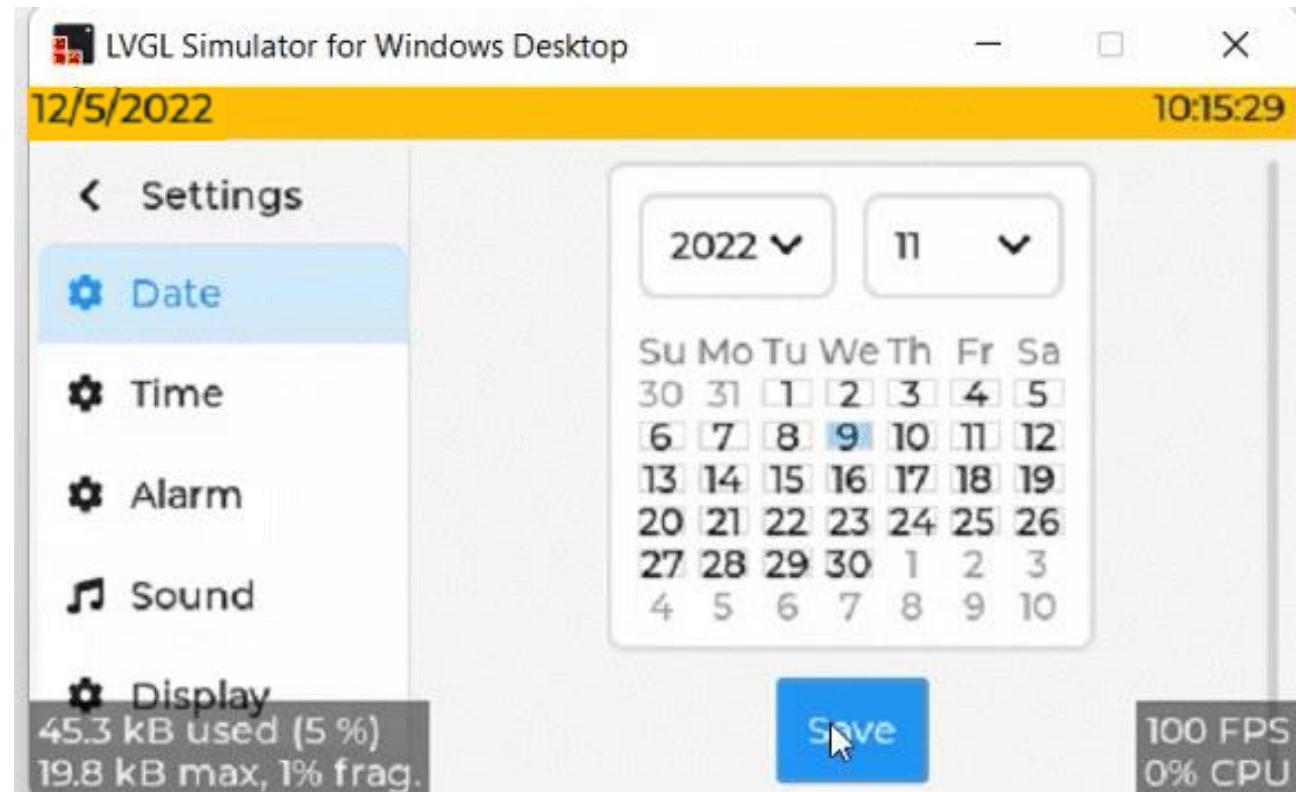


Date setting





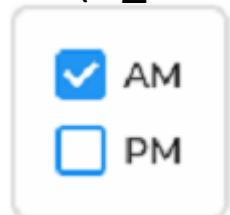
Time setting



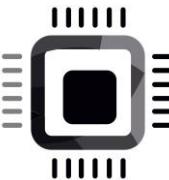
Roller(lv_roller)



Checkbox(lv_checkbox)

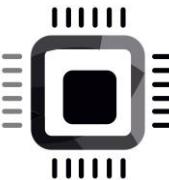


Switch(lv_switch)

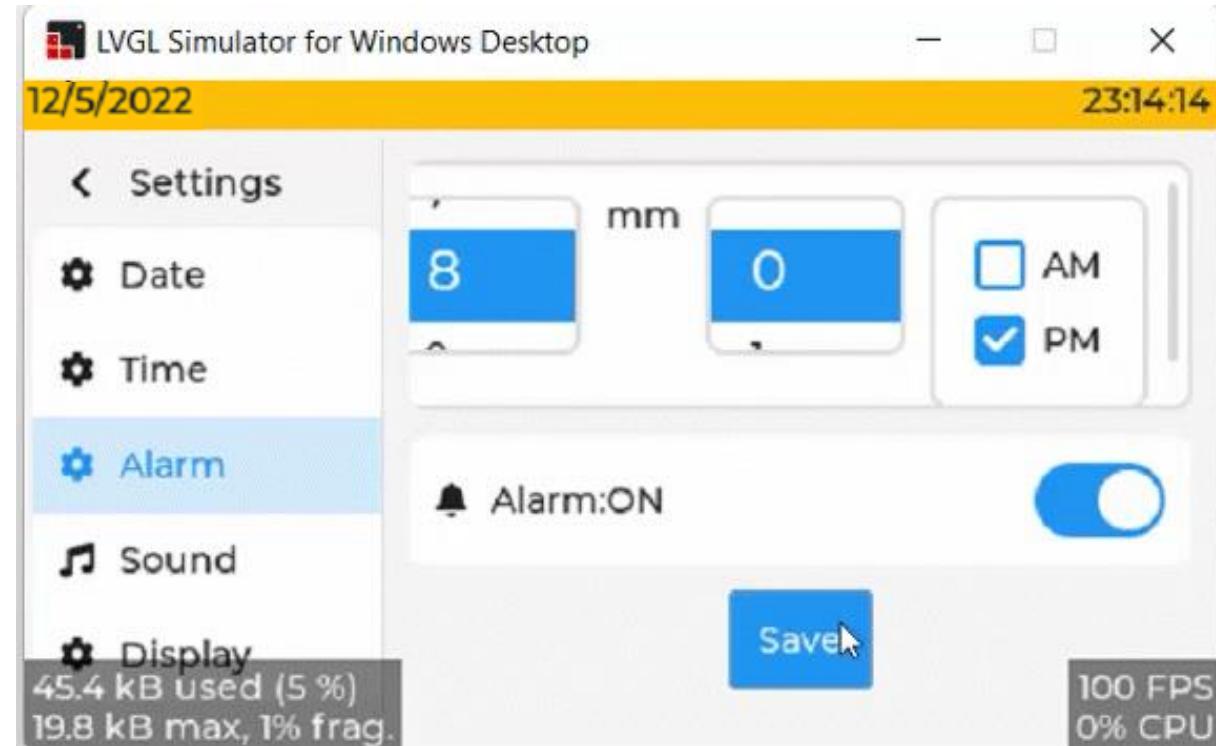


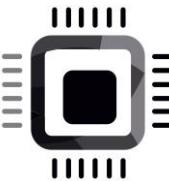
Alarm setting



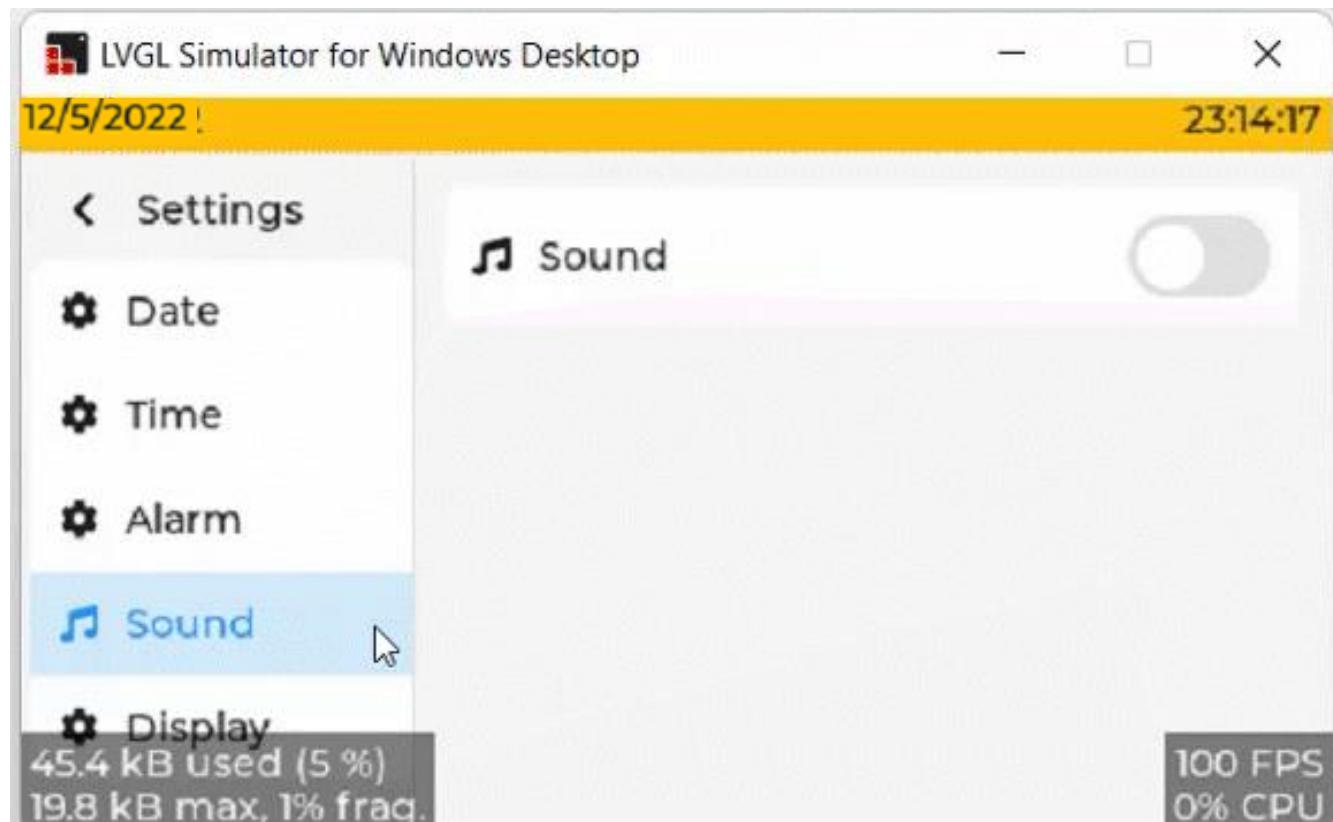


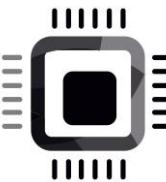
Sound setting





Brightness setting



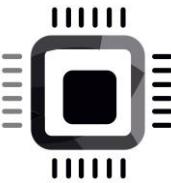


Alarm notification

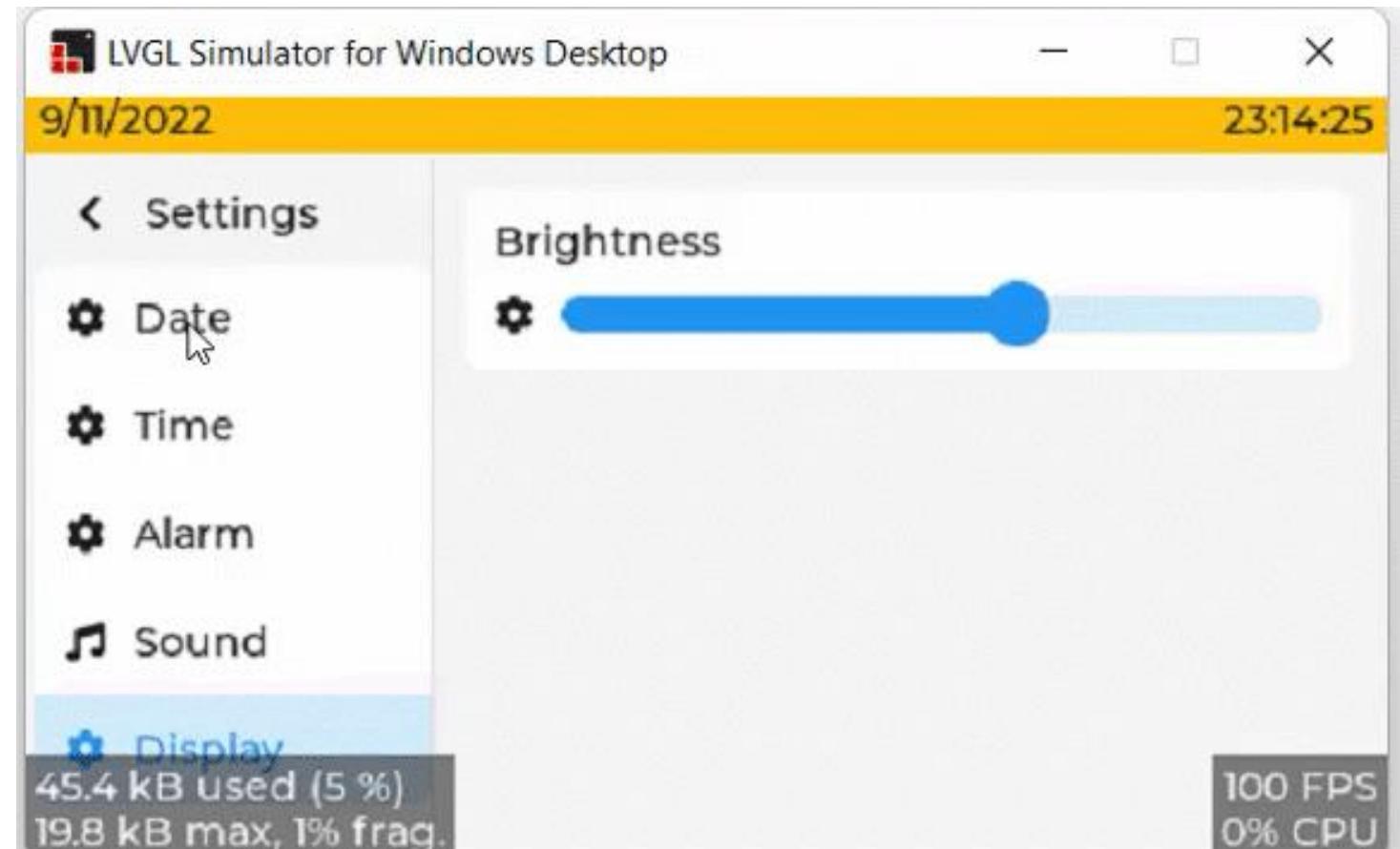


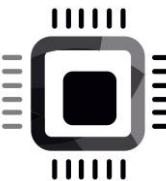
Gif image

```
void Lvgl_create_alarm_notif_page(Lvgl* const me, \
                                  lvgl_alarm_page_data_t *data) in glvgl.c
```



Message box alert





```
lv_refr.c



```
/*
 * Called periodically to handle the refreshing
 * @param tmr pointer to the timer itself
 */
void _lv_disp_refr_timer(lv_timer_t * tmr)
{
 TRACE_REFR("begin");

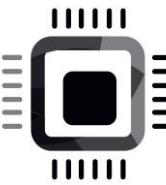
 uint32_t start = lv_tick_get();
 uint32_t elaps = 0;

 disp_refr = tmr->user_data;

#if LV_USE_PERF_MONITOR == 0 && LV_USE_MEM_MONITOR == 0
 /**
 * Ensure the timer does not run again automatically.
 * This is done before refreshing in case refreshing invalidates something else.
 */
 lv_timer_pause(tmr);
#endif

 /*Refresh the screen's layout if required*/
 lv_obj_update_layout(disp_refr->act_scr);
}
```


```

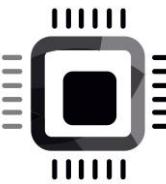


Lvgl and display driver interface

4.2 Display interface

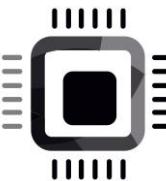
To register a display for LVGL, a `lv_disp_draw_buf_t` and a `lv_disp_drv_t` variable have to be initialized.

- `lv_disp_draw_buf_t` contains internal graphic buffer(s) called draw buffer(s).
- `lv_disp_drv_t` contains callback functions to interact with the display and manipulate low level drawing behavior.



LVGL draw buffer

- You can provide 1 or 2 draw buffers to the lvgl
- Draw buffers need not be size of screen
- They are simple byte arrays
- LVGL uses the draw buffer to render the screen content before its been transferred to the actual screen
- Once the rendering is ready the content of the draw buffer is sent to the display . For this lvgl calls flush method of the display driver.



Logging

4.8.1 Log level

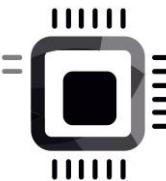
To enable logging, set `LV_USE_LOG 1` in `lv_conf.h` and set `LV_LOG_LEVEL` to one of the following values:

- `LV_LOG_LEVEL_TRACE` A lot of logs to give detailed information
- `LV_LOG_LEVEL_INFO` Log important events
- `LV_LOG_LEVEL_WARN` Log if something unwanted happened but didn't cause a problem
- `LV_LOG_LEVEL_ERROR` Only critical issues, where the system may fail
- `LV_LOG_LEVEL_USER` Only user messages
- `LV_LOG_LEVEL_NONE` Do not log anything

The events which have a higher level than the set log level will be logged too. E.g. if you `LV_LOG_LEVEL_WARN`, errors will be also logged.

Logging with printf

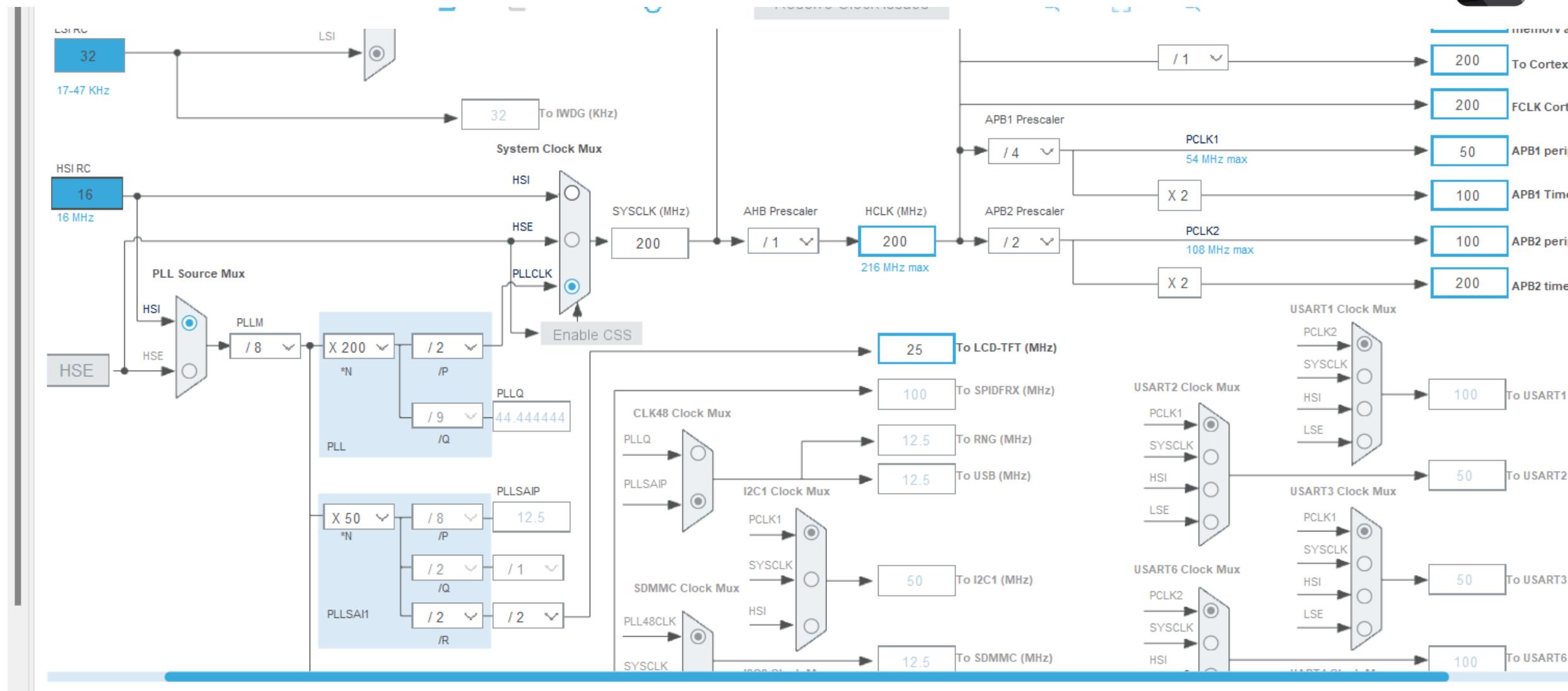
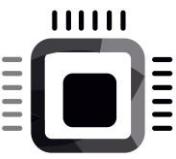
If your system supports `printf`, you just need to enable `LV_LOG_PRINTF` in `lv_conf.h` to send the logs with `printf`.

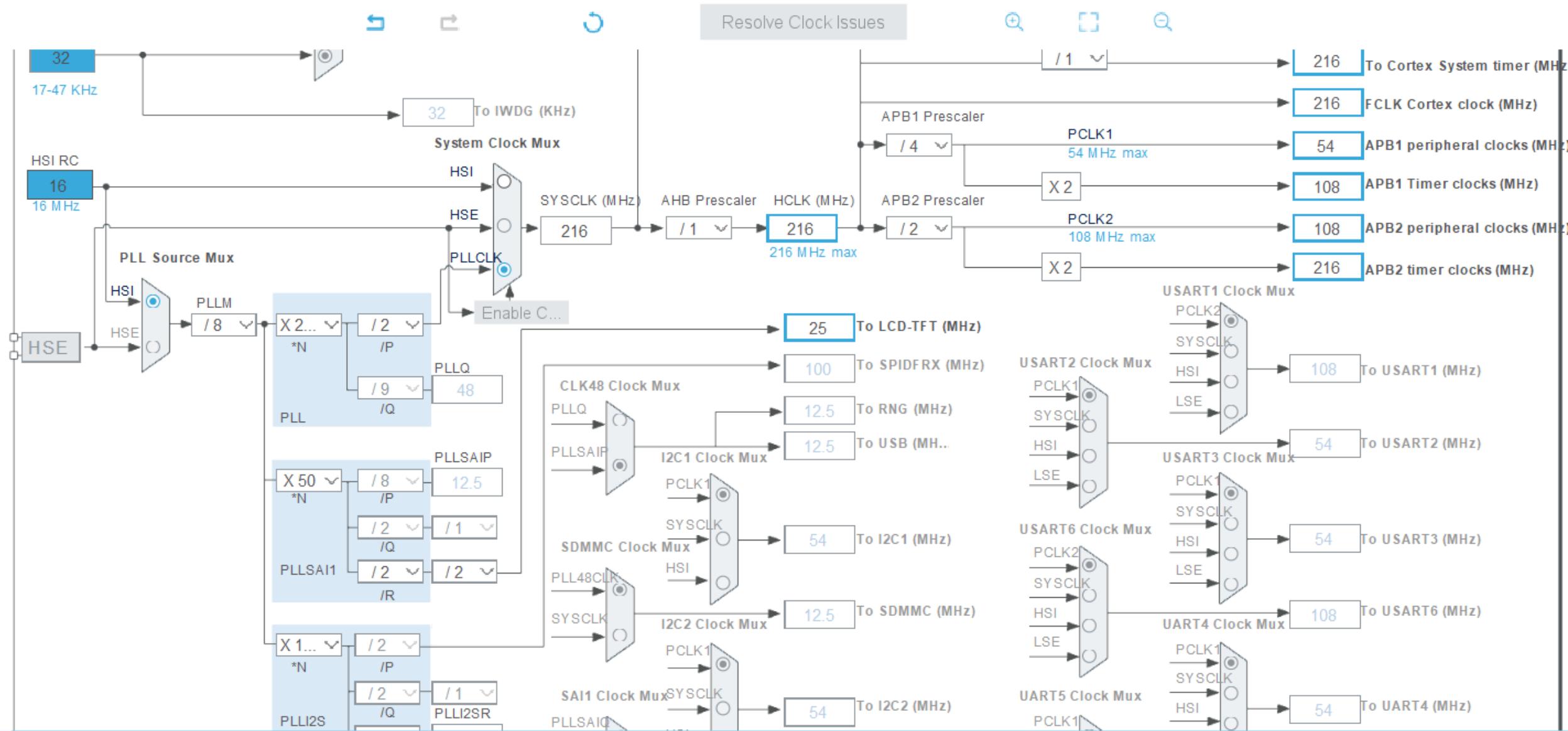
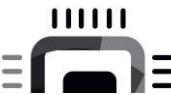


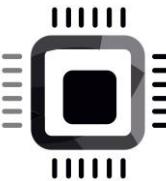
4.8.3 Add logs

You can also use the log module via the `LV_LOG_TRACE/INFO/WARN/ERROR/USER(text)` or `LV_LOG(text)` functions. Here:

- `LV_LOG_TRACE/INFO/WARN/ERROR/USER(text)` append following information to your `text`
- Log Level
- `_FILE_`
- `_LINE_`
- `_func_`
- `LV_LOG(text)` is similar to `LV_LOG_USER` but has no extra information attached.







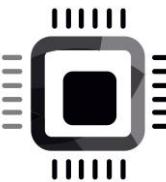
Display module	STM32F7x MCU
LCD_R0	PJ15
LCD_R1	PJ0
LCD_R2	PJ1
LCD_R3	PJ2
LCD_R4	PJ3
LCD_R5	PJ4
LCD_R6	PJ5
LCD_R7	PJ6

Display module	STM32F7x MCU
LCD_G0	PJ7
LCD_G1	PJ8
LCD_G2	PJ9
LCD_G3	PJ10
LCD_G4	PJ11
LCD_G5	PK0
LCD_G6	PK1
LCD_G7	PK2

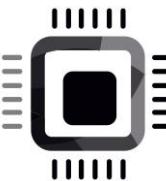
Display module	STM32F7x MCU
LCD_B0	PE4
LCD_B1	PJ13
LCD_B2	PJ14
LCD_B3	PJ15
LCD_B4	PG12
LCD_B5	PK4
LCD_B6	PK5
LCD_B7	PK6

Display module	STM32F7x MCU
LCD_VSYNC	PJ9
LCD_HSYNC	Pj10
LCD_PCLK	PJ14
LCD_DE	PK7

GPIOG,GPIOI,GPIOJ,GPIOK



- Program the new number of wait states to the LATENCY bits in the FLASH_ACR register
- Check that the new number of wait states is taken into account to access the Flash memory by reading the FLASH_ACR register
- Select the required AHB and APB prescalers
 - Configure PPREG=2, PPREG1=4
 - Configure HPREG = 0xxx: system clock not divider
- Configure PLLM = 000010: PLLM = 2, PLLN = 216 , PLLP = 8
- Configure RCC_PLLCFGR register and set PLLON bit of RCC_CR register.
- Wait for PLL lock.
 - : Main PLL (PLL) clock ready flag
- Switch the system clock to the PLL.
 - SW: System clock switch
- Wait for status
 - SWS: System clock switch status
- Check that the new CPU clock source or/and the new CPU clock prescaler value is/are taken into account by reading the clock source status (SWS bits) or/and the AHB prescaler value (HPREG bits), respectively, in the RCC_CFGR register.



Increasing the CPU frequency

1. Program the new number of wait states to the LATENCY bits in the FLASH_ACR register
2. Check that the new number of wait states is taken into account to access the Flash memory by reading the FLASH_ACR register
3. Modify the CPU clock source by writing the SW bits in the RCC_CFGR register
4. If needed, modify the CPU clock prescaler by writing the HPRE bits in RCC_CFGR
5. Check that the new CPU clock source or/and the new CPU clock prescaler value is/are taken into account by reading the clock source status (SWS bits) or/and the AHB prescaler value (HPRE bits), respectively, in the RCC_CFGR register.



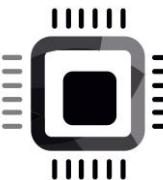
Table 5. Number of wait states according to CPU clock (HCLK) frequency

Wait states (WS) (LATENCY)	HCLK (MHz)			
	Voltage range 2.7 V - 3.6 V	Voltage range 2.4 V - 2.7 V	Voltage range 2.1 V - 2.4 V	Voltage range 1.8 V - 2.1 V
0 WS (1 CPU cycle)	0 < HCLK ≤ 30	0 < HCLK ≤ 24	0 < HCLK ≤ 22	0 < HCLK ≤ 20
1 WS (2 CPU cycles)	30 < HCLK ≤ 60	24 < HCLK ≤ 48	22 < HCLK ≤ 44	20 < HCLK ≤ 40
2 WS (3 CPU cycles)	60 < HCLK ≤ 90	48 < HCLK ≤ 72	44 < HCLK ≤ 66	40 < HCLK ≤ 60
3 WS (4 CPU cycles)	90 < HCLK ≤ 120	72 < HCLK ≤ 96	66 < HCLK ≤ 88	60 < HCLK ≤ 80
4 WS (5 CPU cycles)	120 < HCLK ≤ 150	96 < HCLK ≤ 120	88 < HCLK ≤ 110	80 < HCLK ≤ 100
5 WS (6 CPU cycles)	150 < HCLK ≤ 180	120 < HCLK ≤ 144	110 < HCLK ≤ 132	100 < HCLK ≤ 120
6 WS (7 CPU cycles)	180 < HCLK ≤ 210	144 < HCLK ≤ 168	132 < HCLK ≤ 154	120 < HCLK ≤ 140
7 WS (8 CPU cycles)	210 < HCLK ≤ 216	168 < HCLK ≤ 192	154 < HCLK ≤ 176	140 < HCLK ≤ 160
8 WS (9 CPU cycles)	-	192 < HCLK ≤ 216	176 < HCLK ≤ 198	160 < HCLK ≤ 180
9 WS (10 CPU cycles)	-	-	198 < HCLK ≤ 216	-



18.6 LTDC programming procedure

- Enable the LTDC clock in the RCC register.
- Configure the required pixel clock following the panel datasheet.
- Configure the synchronous timings: VSYNC, HSYNC, vertical and horizontal back porch, active data area and the front porch timings following the panel datasheet as



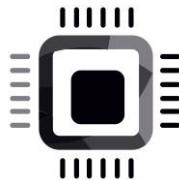
described in the [Section 18.4.1: LTDC global configuration parameters](#).

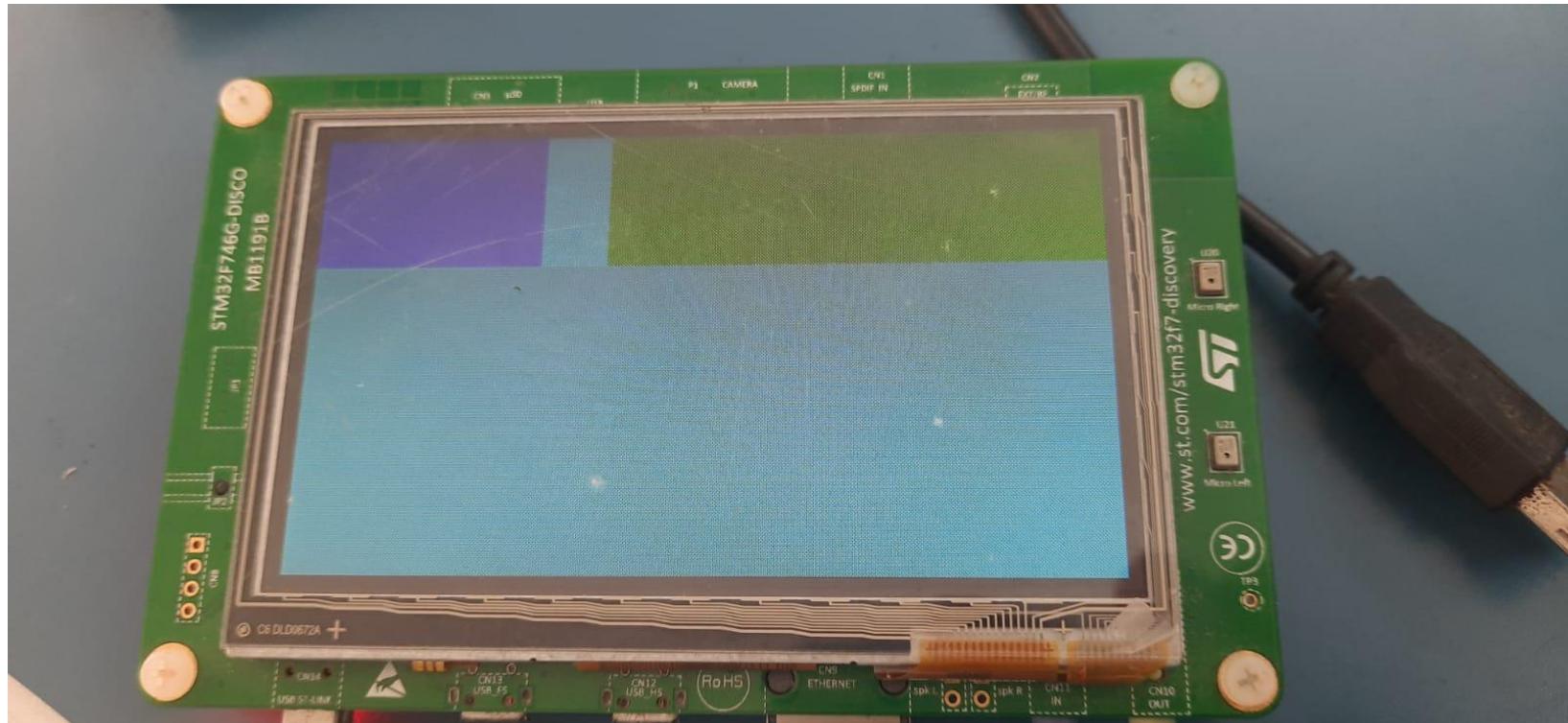
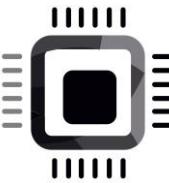
- Configure the synchronous signals and clock polarity in the LTDC_GCR register.
- If needed, configure the background color in the LTDC_BCCR register.
- Configure the needed interrupts in the LTDC_IER and LTDC_LIPCR register.
- Configure the layer1/2 parameters by:
 - programming the layer window horizontal and vertical position in the LTDC_LxWHPCR and LTDC_WVPCR registers. The layer window must be in the active data area.
 - programming the pixel input format in the LTDC_LxPFCR register
 - programming the color frame buffer start address in the LTDC_LxCFBAR register
 - programming the line length and pitch of the color frame buffer in the LTDC_LxCFBLR register
 - programming the number of lines of the color frame buffer in the LTDC_LxCFBLNR register
 - if needed, loading the CLUT with the RGB values and its address in the LTDC_LxCLUTWR register
 - If needed, configuring the default color and the blending factors respectively in the LTDC_LxDCCR and LTDC_LxBFCR registers
- Enable layer1/2 and if needed the CLUT in the LTDC_LxCR register.
- If needed, enable dithering and color keying respectively in the LTDC_GCR and LTDC_LxCKCR registers. They can be also enabled on the fly.
- Reload the shadow registers to active register through the LTDC_SRCR register.
- Enable the LCD-TFT controller in the LTDC_GCR register.
- All layer parameters can be modified on the fly except the CLUT. The new configuration has to be either reloaded immediately or during vertical blanking period by configuring the LTDC_SRCR register.

Note:

All layer's registers are shadowed. Once a register is written, it must not be modified again before the reload has been done. Thus, a new write to the same register overrides the previous configuration if not yet reloaded.

Alpha:
Layer1 = 255
Layer2 = 0

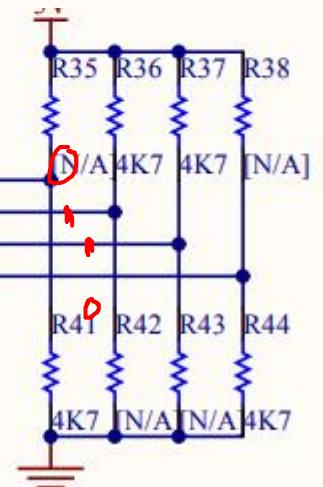
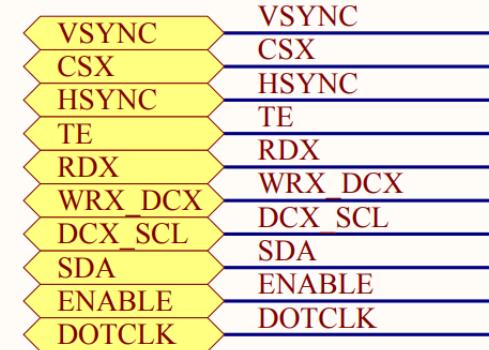
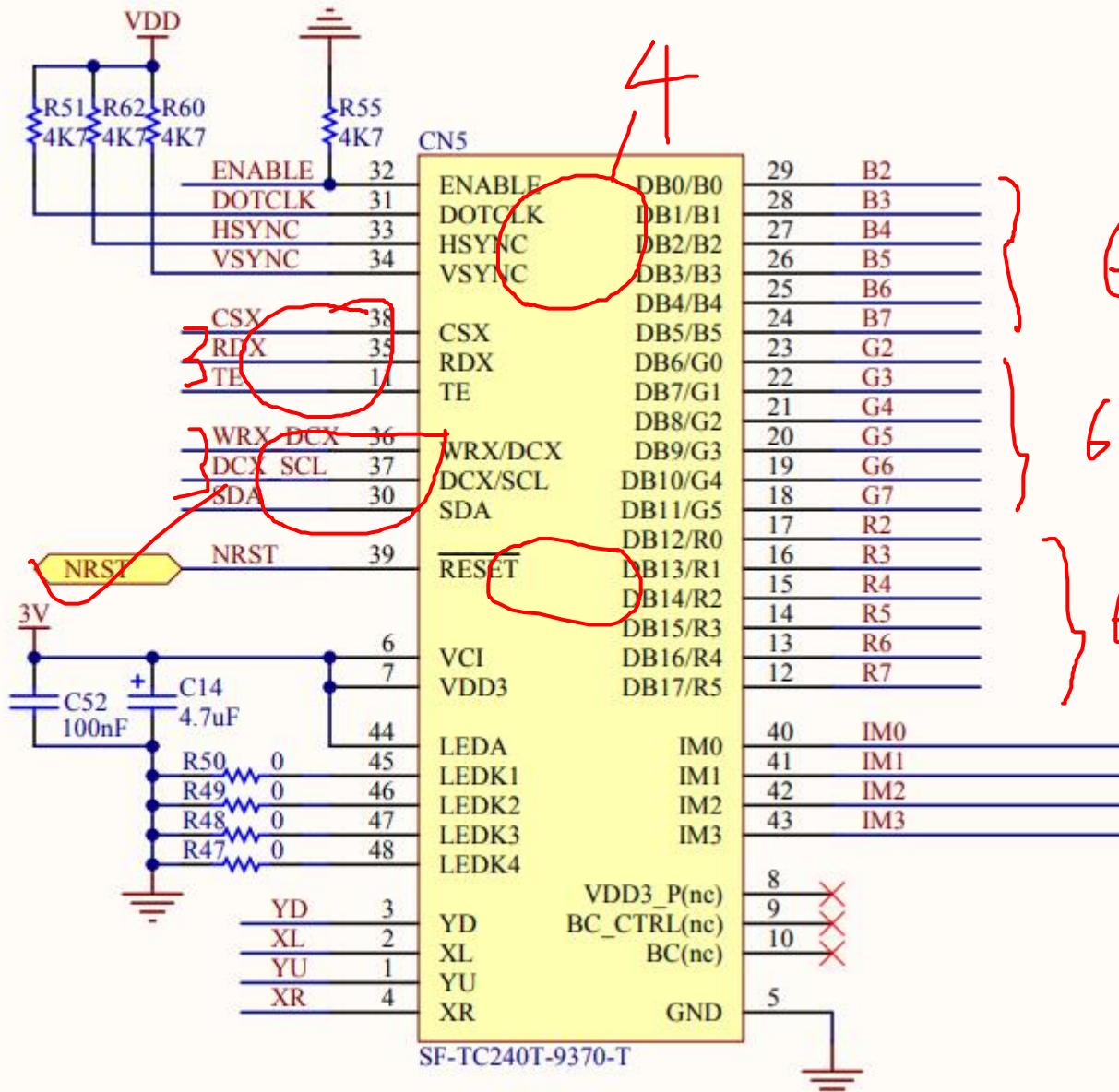
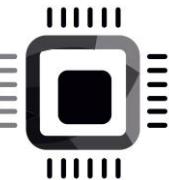


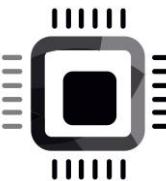


Alpha:

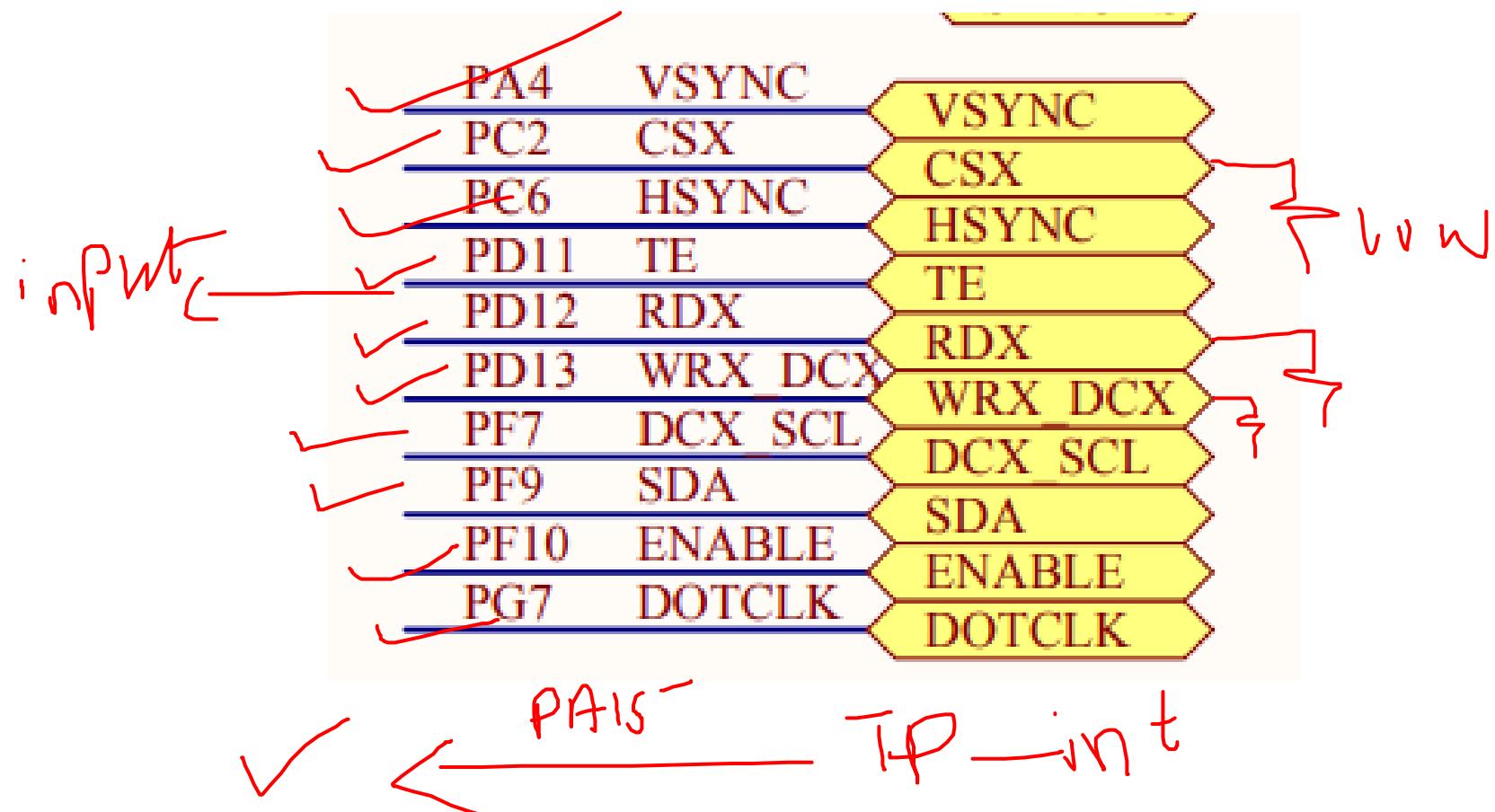
Layer1 = 255

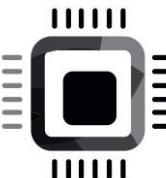
Layer2 = 127



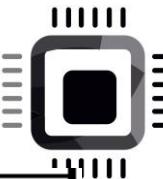


Initialization status

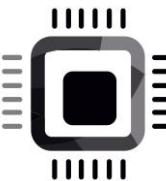




Interface Logic Signals							
Pin Name	I/O	Type	Descriptions				
IM[3:0]	I (VDDI/VSS)		- Select the MCU interface mode IM3 IM2 IM1 IM0 MCU-Interface Mode				
			0 0 0 0	80 MCU 8-bit bus interface I	D[7:0]	D[7:0]	
			0 0 0 1	80 MCU 16-bit bus interface I	D[7:0]	D[15:0]	
			0 0 1 0	80 MCU 9-bit bus interface I	D[7:0]	D[8:0]	
			0 0 1 1	80 MCU 18-bit bus interface I	D[7:0]	D[17:0]	
			0 1 0 1	3-wire 9-bit data serial interface I	SDA: In/OUT		
			0 1 1 0	4-wire 8-bit data serial interface I	SDA: In/OUT		
			1 0 0 0	80 MCU 16-bit bus interface II	D[8:1]	D[17:10], D[8:1]	
			1 0 0 1	80 MCU 8-bit bus interface II	D[17:10]	D[17:10]	
			1 0 1 0	80 MCU 18-bit bus interface II	D[8:1]	D[17:0]	
			1 0 1 1	80 MCU 9-bit bus interface II	D[17:10]	D[17:9]	
			1 1 0 1	3-wire 9-bit data serial interface II	SDI: In SDO: Out		
			1 1 1 0	4-wire 8-bit data serial interface II	SDI: In SDO: Out		
MPU Parallel interface bus and serial interface select If use RGB Interface must select serial interface. * : Fix this pin at VDDI or VSS.							

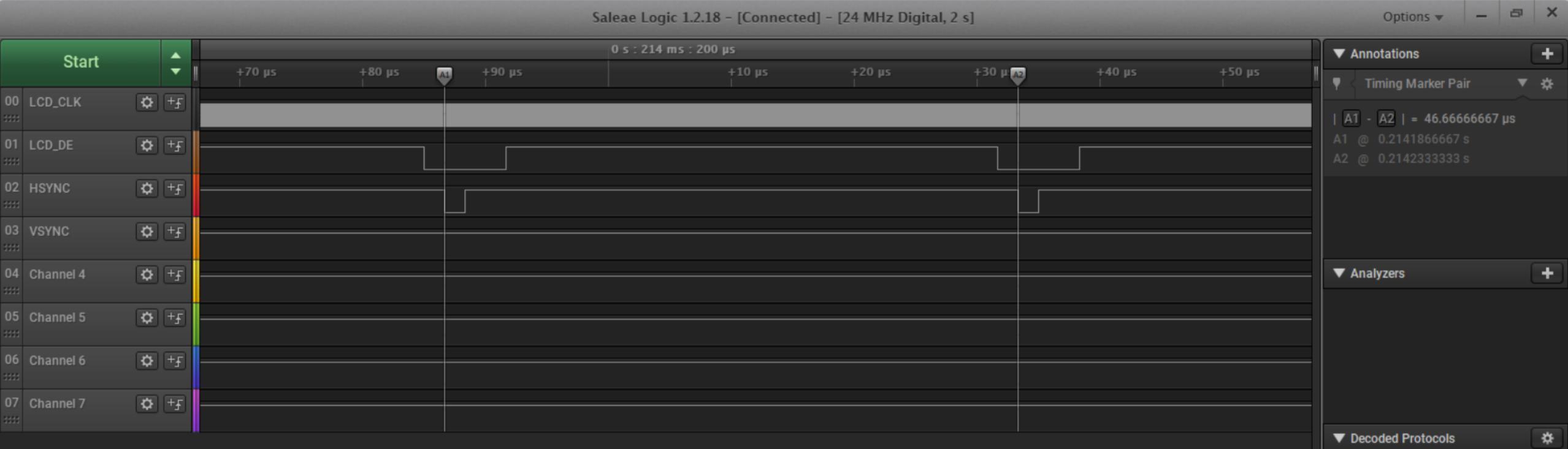
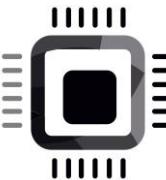


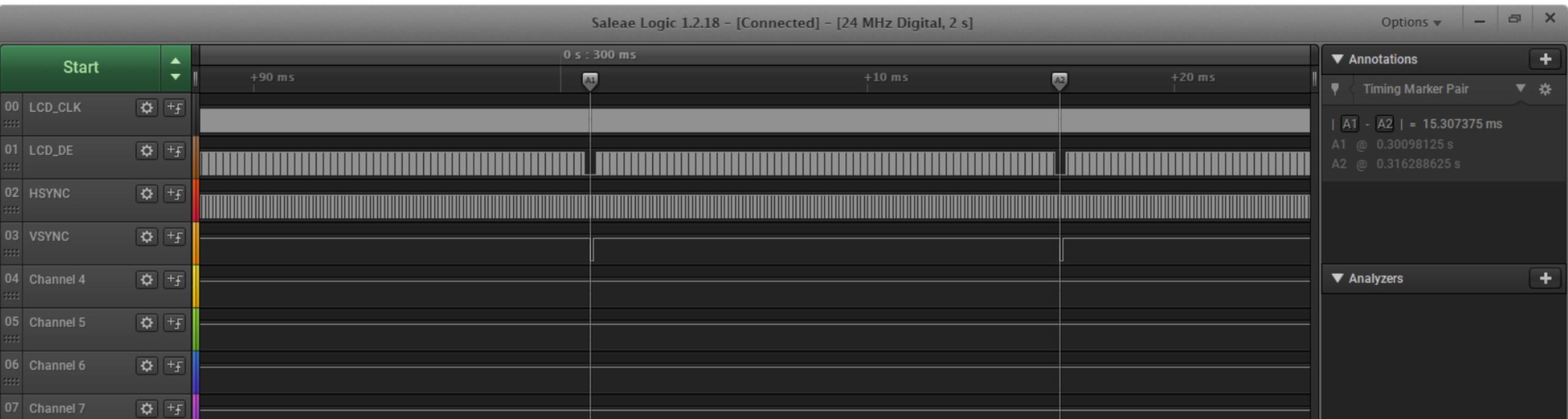
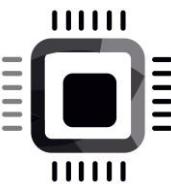
CSX	I	MCU (VDDI/VSS)	<p>Chip select input pin (“Low” enable). This pin can be permanently fixed “Low” in MPU interface mode only. * note1,2</p>
D/CX (SCL)	I	MCU (VDDI/VSS)	<p>This pin is used to select “Data or Command” in the parallel interface or 4-wire 8-bit serial data interface. When DCX = '1', data is selected. When DCX = '0', command is selected. This pin is used serial interface clock in 3-wire 9-bit / 4-wire 8-bit serial data interface. If not used, this pin should be connected to VDDI or VSS.</p>
RDX	I	MCU (VDDI/VSS)	<p>8080- I /8080- II system (RDX): Serves as a read signal and MCU read data at the rising edge. <i>Fix to VDDI level when not in use.</i></p>
WRX (D/CX)	I	MCU (VDDI/VSS)	<ul style="list-style-type: none"> - 8080- I /8080- II system (WRX): Serves as a write signal and writes data at the rising edge. - 4-line system (D/CX): Serves as command or parameter select. <p><i>Fix to VDDI level when not in use.</i></p>

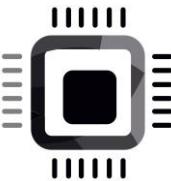


D[17:0]	I/O	MCU (VDDI/VSS)	18-bit parallel bi-directional data bus for MCU system and RGB interface mode <i>Fix to VSS level when not in use</i>
SDI/SDA	I/O	MCU (VDDI/VSS)	When IM[3] : Low, Serial in/out signal. When IM[3] : High, Serial input signal. The data is applied on the rising edge of the SCL signal. <i>If not used, fix this pin at VDDI or VSS.</i>

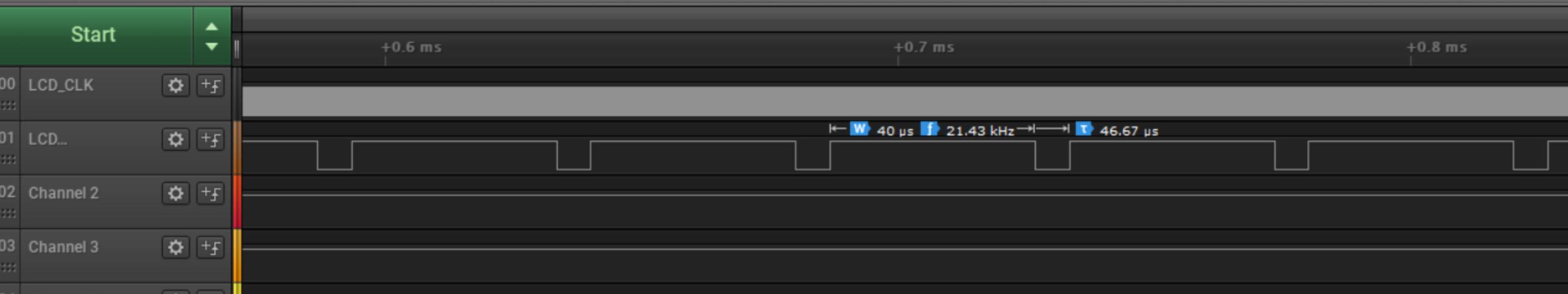
TE	O	MCU (VDDI/VSS)	Tearing effect output pin to synchronize MPU to frame writing, activated by S/W command. When this pin is not activated, this pin is low. <i>If not used, open this pin.</i>
----	---	-------------------	---

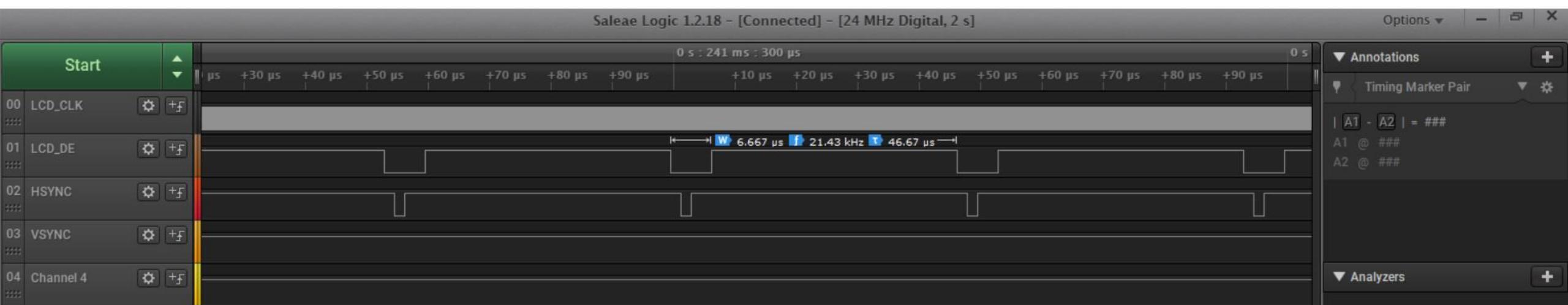
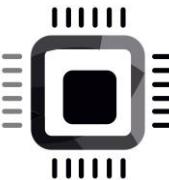


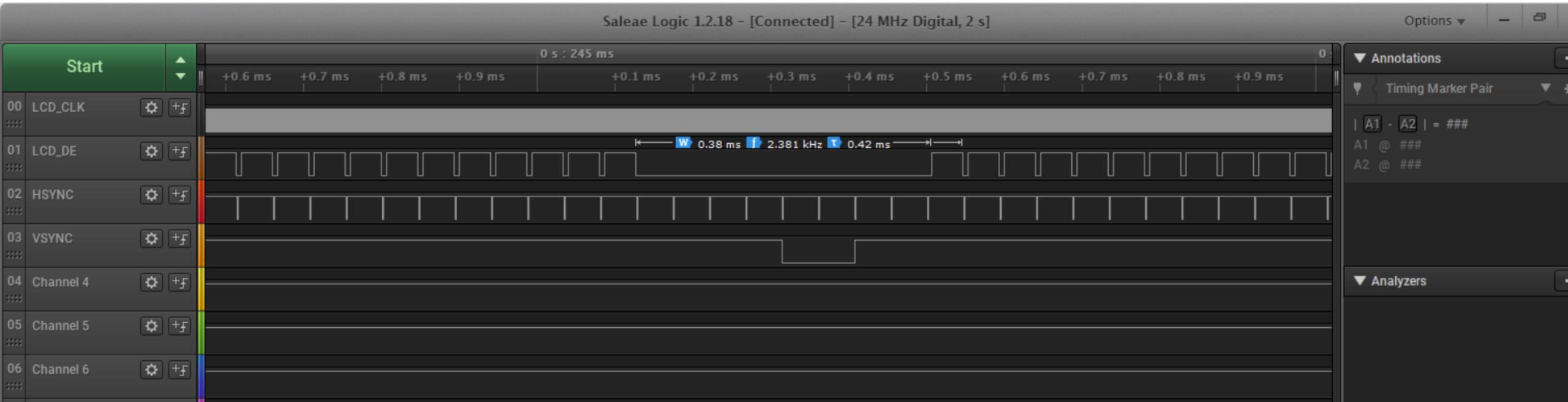
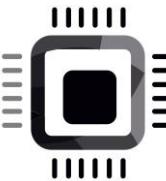


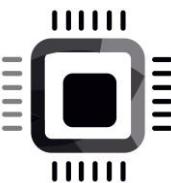


Saleae Logic 1.2.18 – [Connected] – [24 MHz Digital, 2 s]

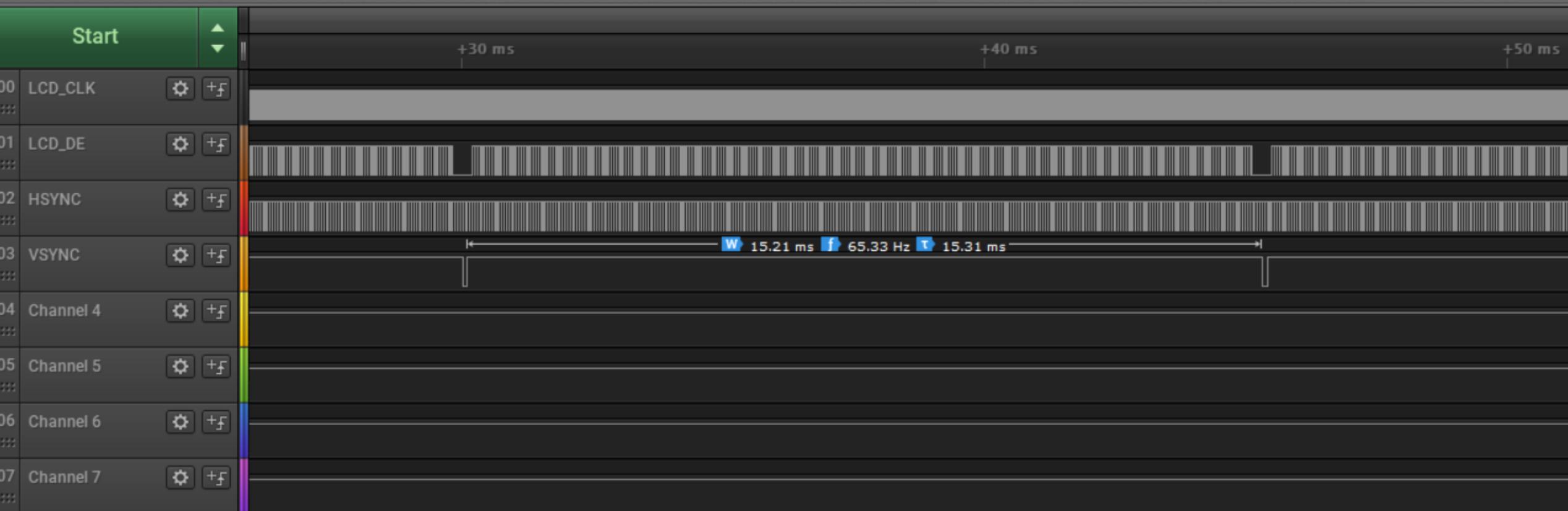






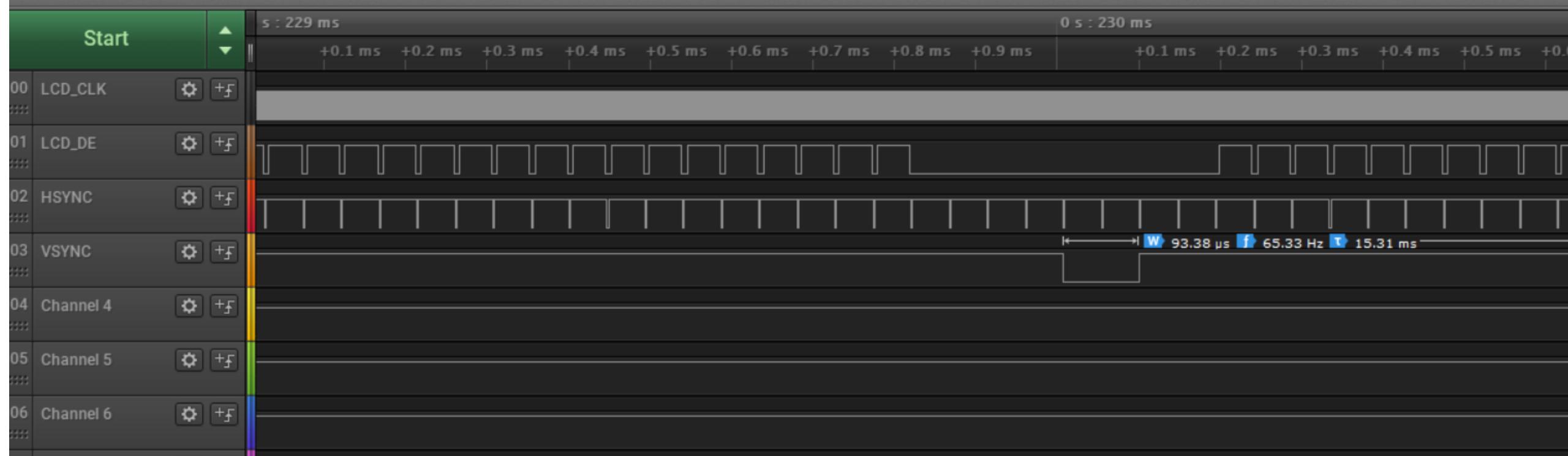


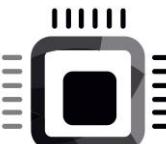
Saleae Logic 1.2.18 – [Connected] – [24 MHz Digital, 2 s]



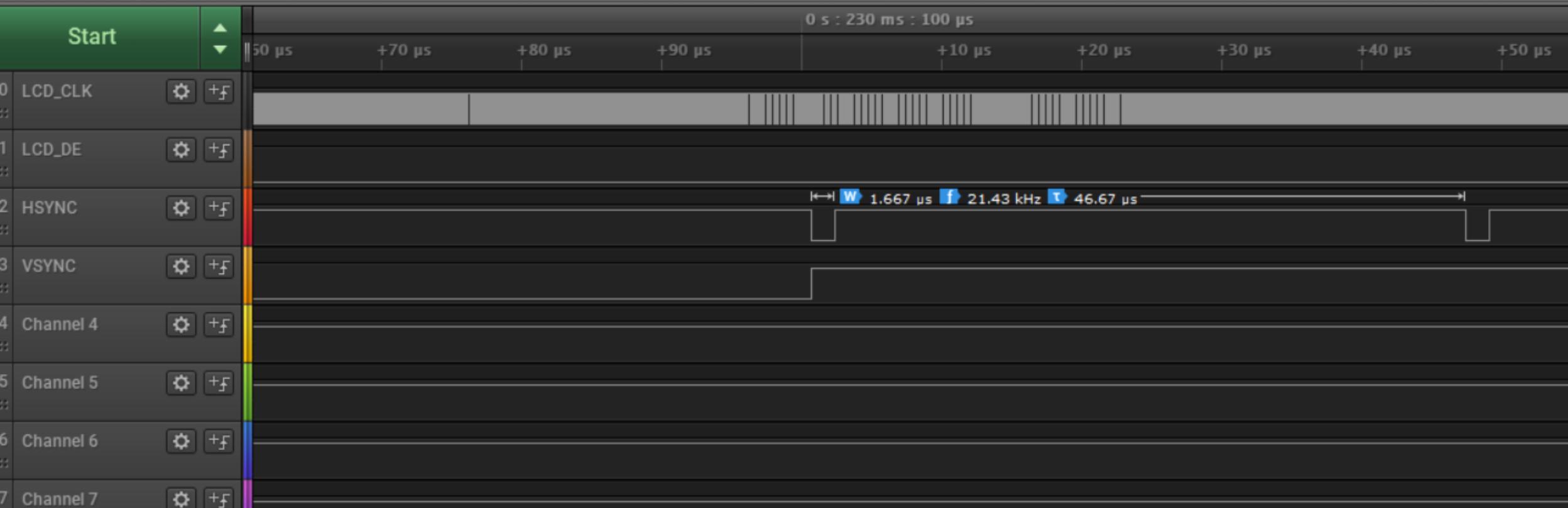


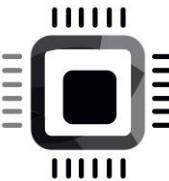
Saleae Logic 1.2.18 - [Connected] - [24 MHz Digital, 2 s]

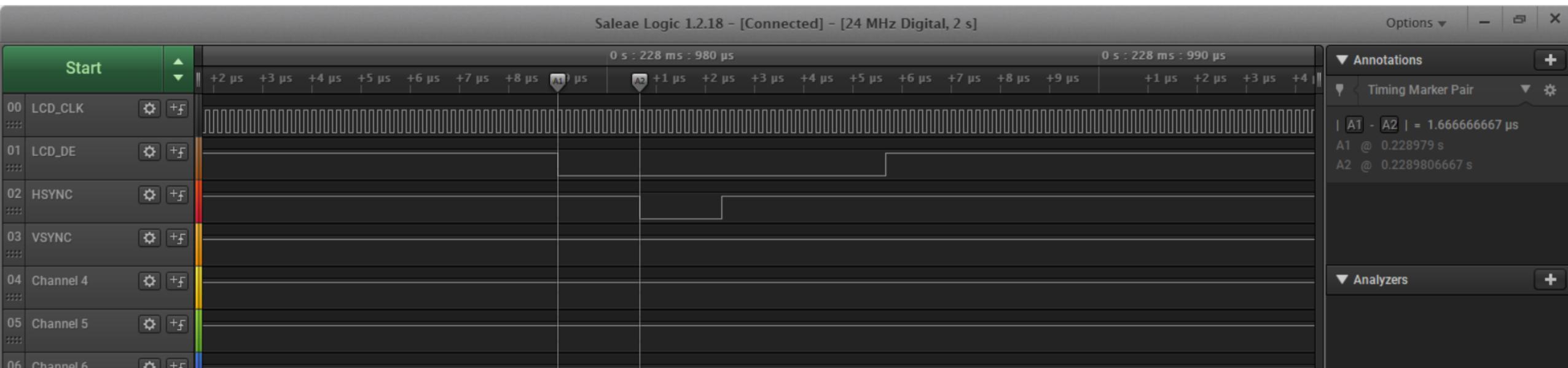
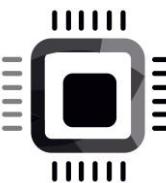


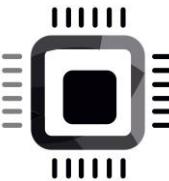


Saleae Logic 1.2.18 - [Connected] - [24 MHz Digital, 2 s]









Saleae Logic 1.2.18 – [Connected] – [24 MHz Digital, 2 s]

