Let's play

with the STM32F746G

Objectives

Explore some properties of the µC STM32F746G

Get familiar with the LCD-TFT display

Useful references (they are really useful !)

- STM32F7 Online Training

http://www.st.com/content/st_com/en/support/learning/stm32f7-online-training.html

- Datasheet of the STM32F746

http://www.st.com/content/ccc/resource/technical/document/datasheet/96/ed/61/9b/e0/6c/45/0b/DM00166116.pdf/files/DM00166116.pdf/jcr:content/translations/en.DM00166116.pdf

- Datasheet of the LCD-TFT of STM32 µC

http://www.st.com/content/ccc/resource/technical/document/application_note/group0/25/ca/f9/b4/ae/fc/4e/1e/DM00287603/files/DM00287603.pdf/jcr:content/translations/en.DM00287603.pdf

Outline

1. Getting started with the display
    1. LCD-TFT Display
    2. LTDC Display Controller Description
    3. LTDC Display Controller Configuration
    4. Let's practice

2. What about adding some colors
    1. Overview of the Programmable LTDC Layers
    2. Flexible Window Position and Size Configuration
    3. Color Frame Buffer
    4. Let's practice

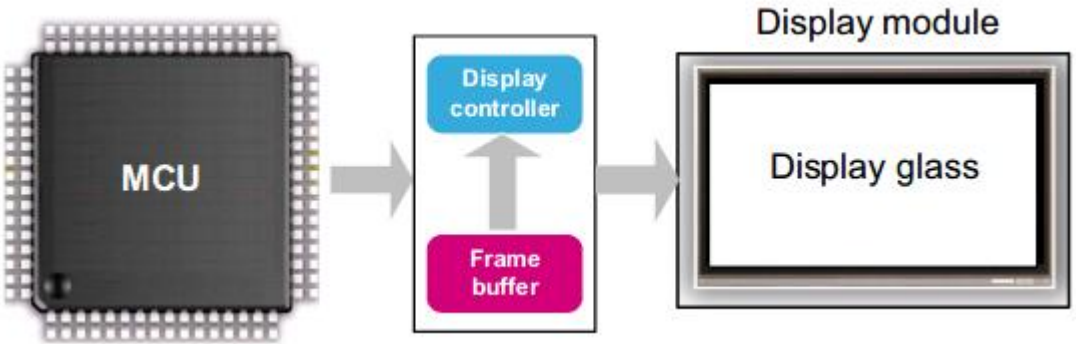
3. Using the touch screen
    1. LCD-TFT Touch Panel
    2. Let's Practice

Getting started with the display

Basic Graphic System

A basic embedded graphic system is composed of

- Microcontroller
- Frame buffer
- Display controller
- Display glass



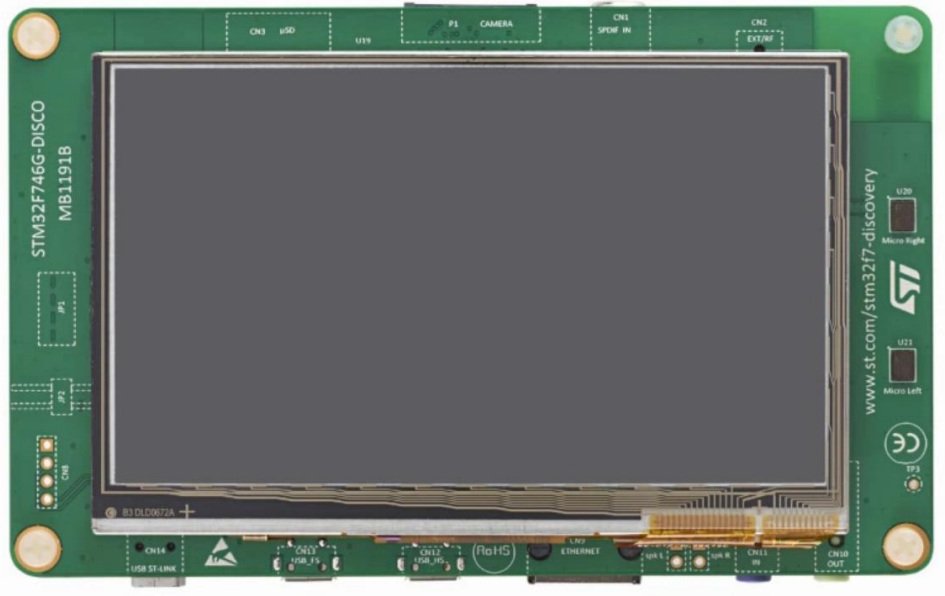
LCD-TFT Display

4.3-inch 480x272 color LCD-TFT

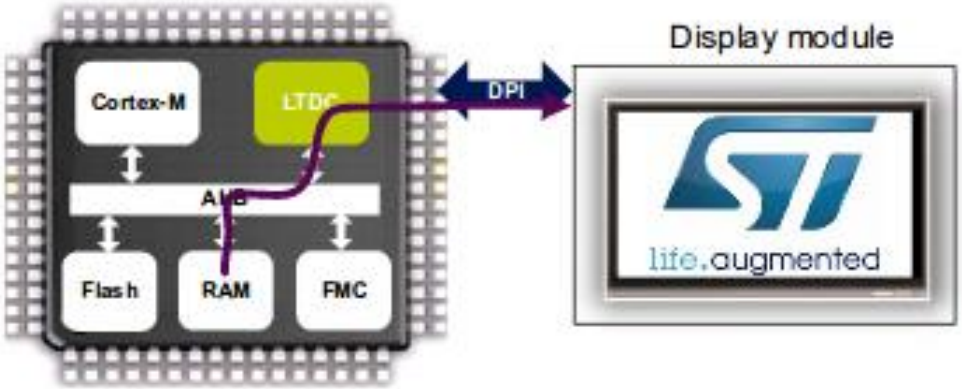
(Liquid Crystal Thin Film Transistor)

with capacitive touch screen is

on the front face of the STM32F7



LCD-TFT Display

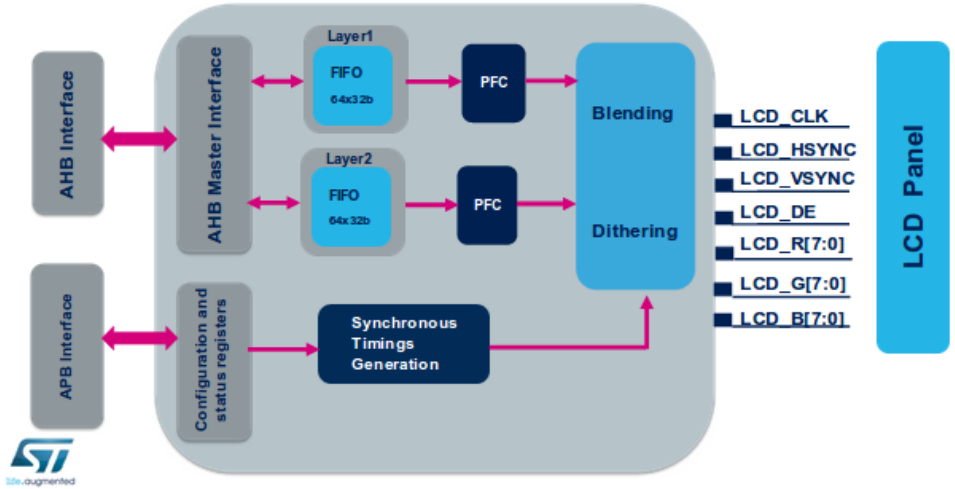


LTDC Display Controller Description

- is master on the AHB Bus Matrix and can access internal memories
- Reads the data of images in a line per line mode
- Provides 24-bit RGB interface with additional signals for horizontal and vertical synchronization

LTDC Display Controller Description

Block Diagram



LTDC Clock Domains

LTDC uses 3 clock domains:

- HCLK
  - AHB clk domain
  - Used to transfer data from memory to the frame buffer
- PCLK
  - To access the configuration and the status registers
- LCD_CLK
  - To generate LCD-TFT interface signals (LCD_HSYNC, LCD_VSYNC,...) and pixel data

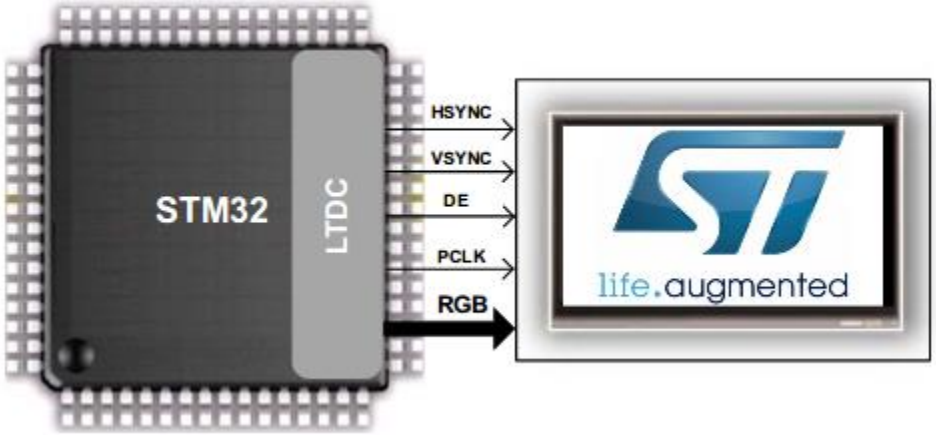
LTDC Reset

- Is managed by the Reset and Clock Controller (RCC)
- Is reset by setting the LTDC_RST bit in the RCC_APB2RSTR register

Reset and Clock Control (RCC)

- Manages the different system and peripheral resets
  - System reset
  - Power reset
  - Backup domain reset
- Has different clocks sources
  - 2 internal clock (HSI, LSI)
  - 2 external oscillators (HSE, LSE)
  - 3 PLLs (phase-locked loops)

Pins and Signal Interface



LTDC Pins and Signal Interface (2)

LTDC provides up to 28 signals including:

LCD-TFT Signals	Description
LCD_CLK	Pixel clock output
LCD_HSYNC	Horizontal synchronization
LCD_VSYNC	Vertical synchronization
LCD_DE	indicates that the data in the RGB bus is valid
LCD_R[7:0]	8-bit red data
LCD_G[7:0]	8-bit green data
LCD_B[7:0]	8-bit blue data

LTDC Pins and Signal Interface (3)

Other specific pins are :

LCD-TFT Signals	Description
LCD_DISP	enable/disable display standby mode
LCD_RST	reset the LCD-TFT
LCD_BL_A & LCD_BL_K	for LED backlight control

LTDC Display Controller Configuration

- GPIOs Configuration
- Peripheral Configuration
- FMC SDRAM Configuration

General-Purpose Input/Output (GPIO)

- enables the connectivity of the STM32F7 to the surrounding environment
- directly connected to AHB bus which allows faster operations
- ability to externally wake up the MCU
- enhanced robustness with the locking mode to freeze the I/O port configuration (GPIOx_LCKR)

General-Purpose Input/Output (GPIO) (2)

- 11 GPIOx ports on the STM32F7
- Each GPIO pin can be configured as
  - Input (floating, with or without pull-up or pull-down)
  - Output (push-pull or open-drain, with or without pull-up or pull-down)
- Each GPIO can host up to 16 I/O pins
- Bit set and reset operations using BSRR and BRR registers
- I/O pins are shared by several peripherals using an alternate function multiplexer

General-Purpose Input/Output (GPIO) (3)

- Declaration of the I/O

```
let mut gpio = Gpio::new(gpio_a,
                         gpio_b,
                         gpio_c,
                         gpio_d,
                         gpio_e,
                         gpio_f,
                         gpio_g,
                         gpio_h,
                         gpio_i,
                         gpio_j,
                         gpio_k);
```

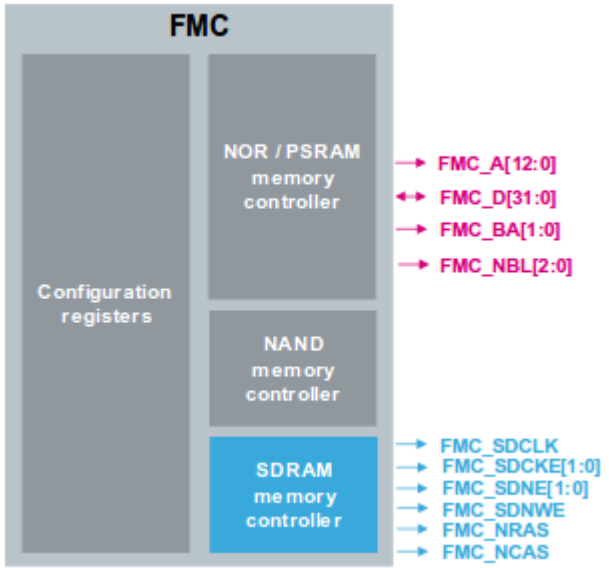
Peripheral Configuration

- System Clock (SYSCLK) configuration
- Pixel Clock (LCD_CLK) configuration
- LTDC Layer parameters configuration

FMC SDRAM Configuration

- The external SDRAM contains the LTDC framebuffer
- The Flexible Memory Controller (FMC)

generates the appropriate signals to drive the SDRAM memories



Configuration in Rust

- Hardware register structure

```
use stm32f7::{system_clock, sdram, lcd, board, embedded};
```

Configuration in Rust

- Hardware extraction in the main function

```
let board::Hardware {
        rcc,
        pwr,
        flash,
        fmc,
        ltdc,
        gpio_a,
        gpio_b,
        gpio_c,
        gpio_d,
        gpio_e,
        gpio_f,
        gpio_g,
        gpio_h,
        gpio_i,
        gpio_j,
        gpio_k,
        ..
    } = hw;
```

Configuration in Rust

- Initialisation of the LCD-TFT and the SDRAM in the main function

```
// init sdram (needed for display buffer)
sdram::init(rcc, fmc, &mut gpio);

// lcd controller
let mut lcd = lcd::init(ltdc, rcc, &mut gpio);
```

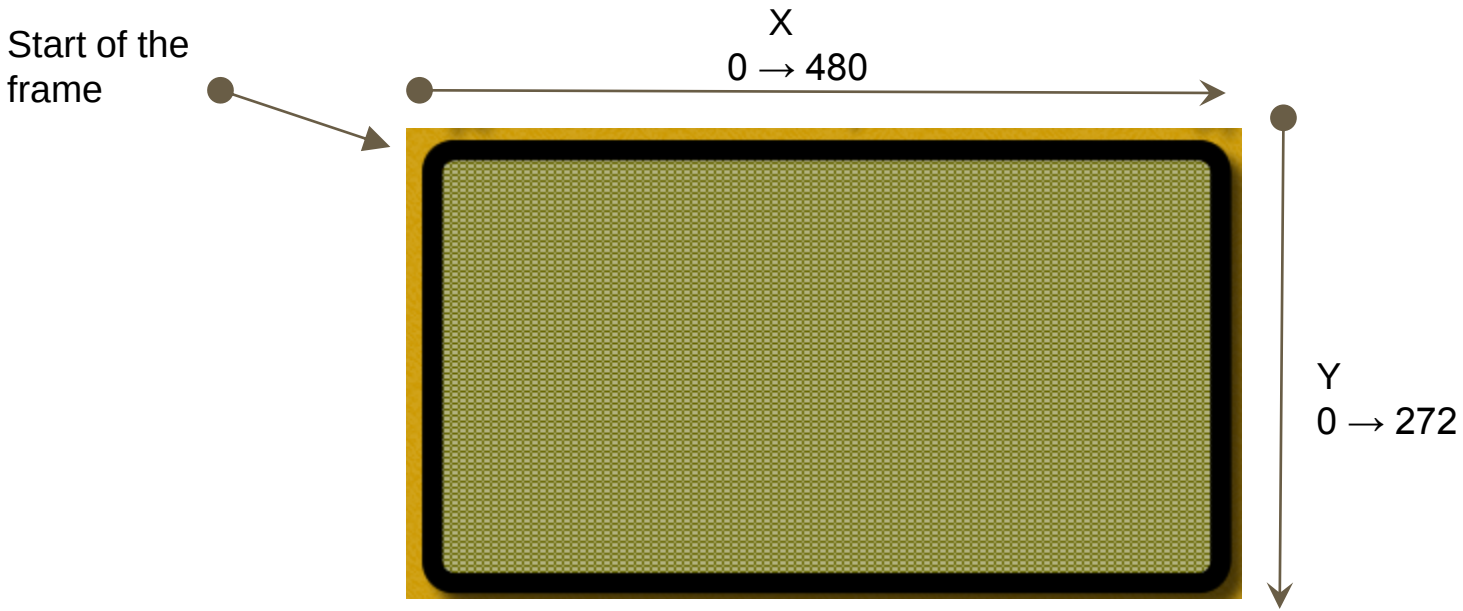
Let's Practice

- Clone / download the demo project from the GitHub

- Focus mainly on the main.rs file

```
fn main(hw: board::Hardware) -> ! {
    ...
    loop {
        // Add your code
    }
}
```

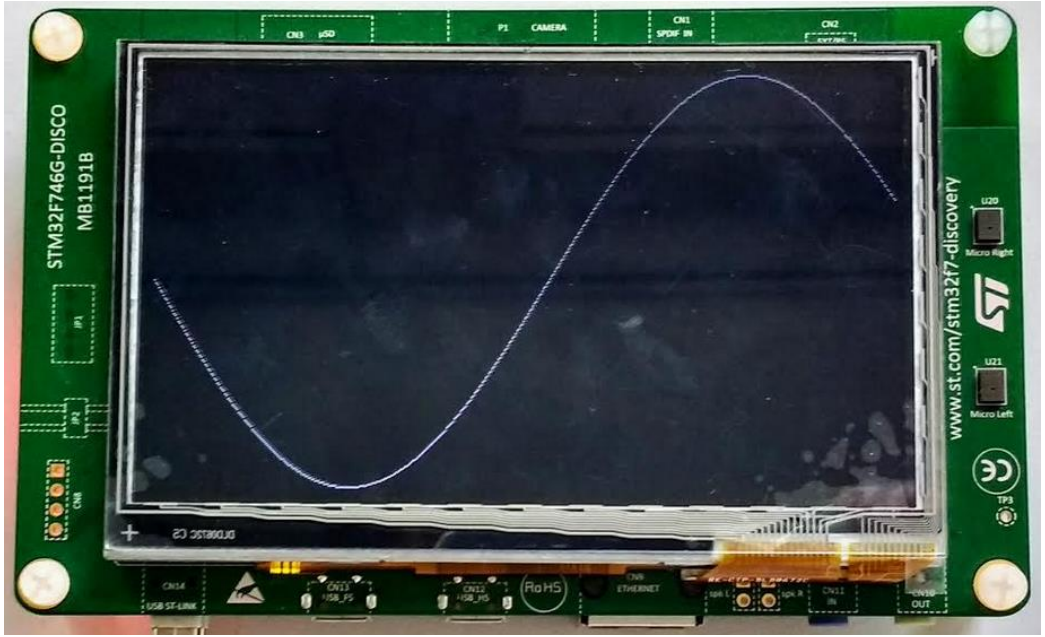
Let's Practice



Let's Practice

Task 1:

Generate one period of a sine wave on the display of the STM32F7 without using the maths library



Let's Practice

- To generate the sine wave without using any library
  - LookUp Table
  - Taylor series
  - CORDIC algorithm...

- To generate a LUT

http://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml

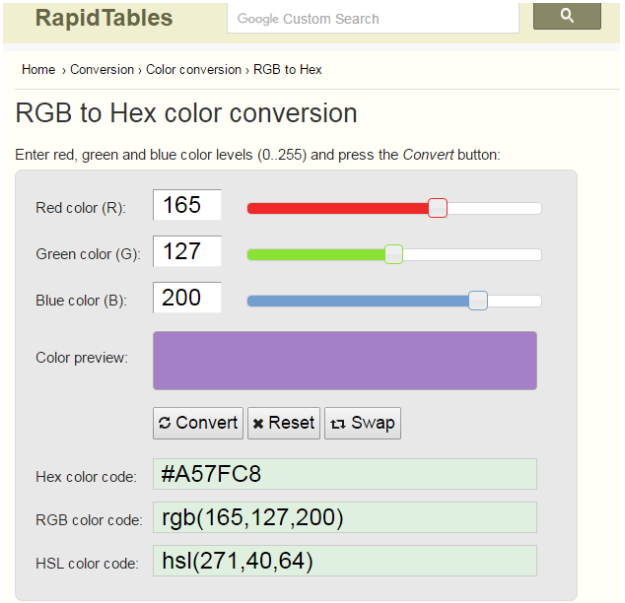
Let's Practice

Extra Tasks:

- Generate more periods of a sine wave on the display of the STM32F7
- Control the frequency of the sine wave

What about adding some colors

What about adding some colors



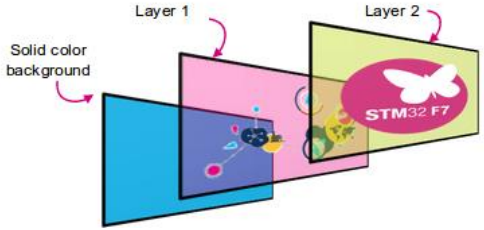
- RGB (red, green, blue) color values are used
- Each parameter (Red, Green,Blue)
  - defines the intensity of the color
  - can be an integer between 0..255 or hexadecimal between #000000 and #FFFFFF
  - http://www.rapidtables.com/convert/color/rgb-to-hex.htm
- The parameter alpha
  - Blending coefficient

What about adding some colors

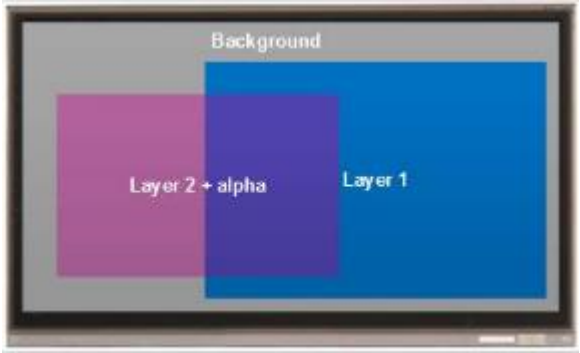
- Definition of the structure Color in Rust

```
pub struct Color {
    pub red: u8,
    pub green: u8,
    pub blue: u8,
    pub alpha: u8,
}
```

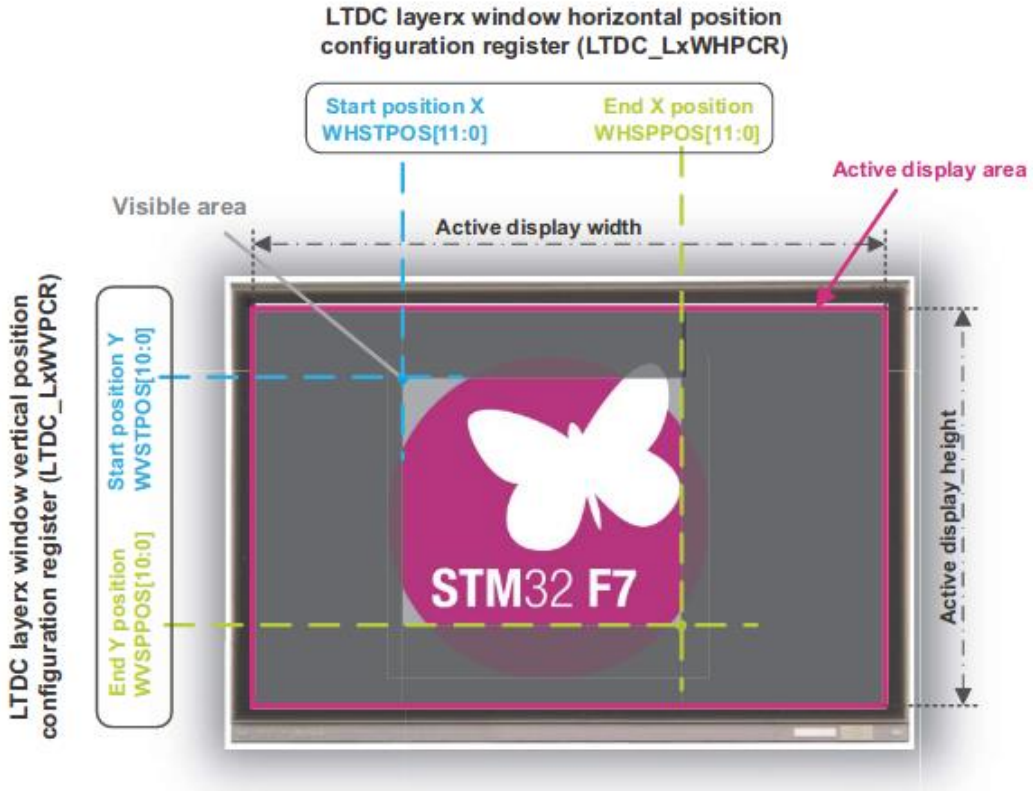
Programmable LTDC Layers



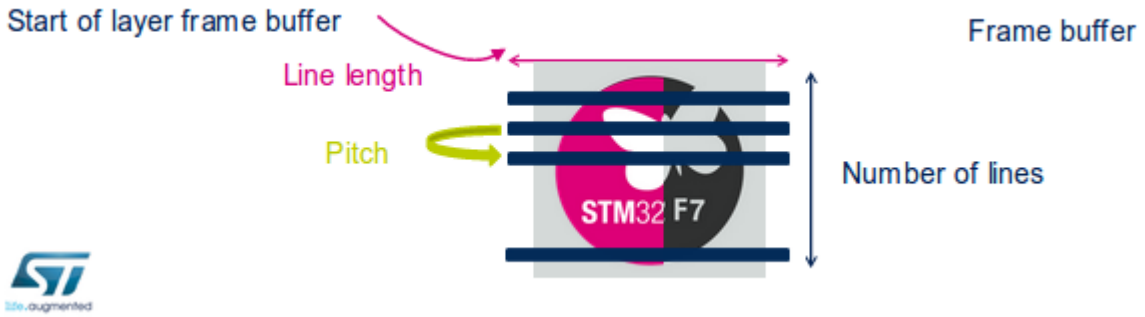
- LTCD has 3 layers (background + 2layers)
  - The layers can be enabled, disabled and configured separately
  - They can be blended according to the alpha coefficient
  - The blending order is fixed (bottom-up)



Flexible Window Position and Size Configuration

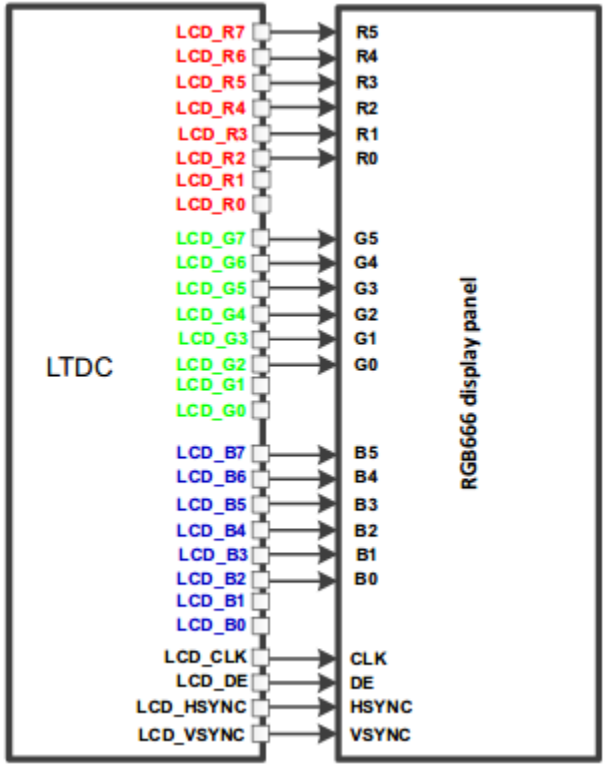


Color Frame Buffer



Registers	Description
LTDC_LxCFBAR	The start address for the color framebuffer
LTDC_LxCFBLR	The line length (in bytes)
LTDC_LxCFBLNR	The number of lines (in bytes)
LTDC_LxCFBLR	The pitch is the distance between the start of one line and the beginning of the next line in bytes

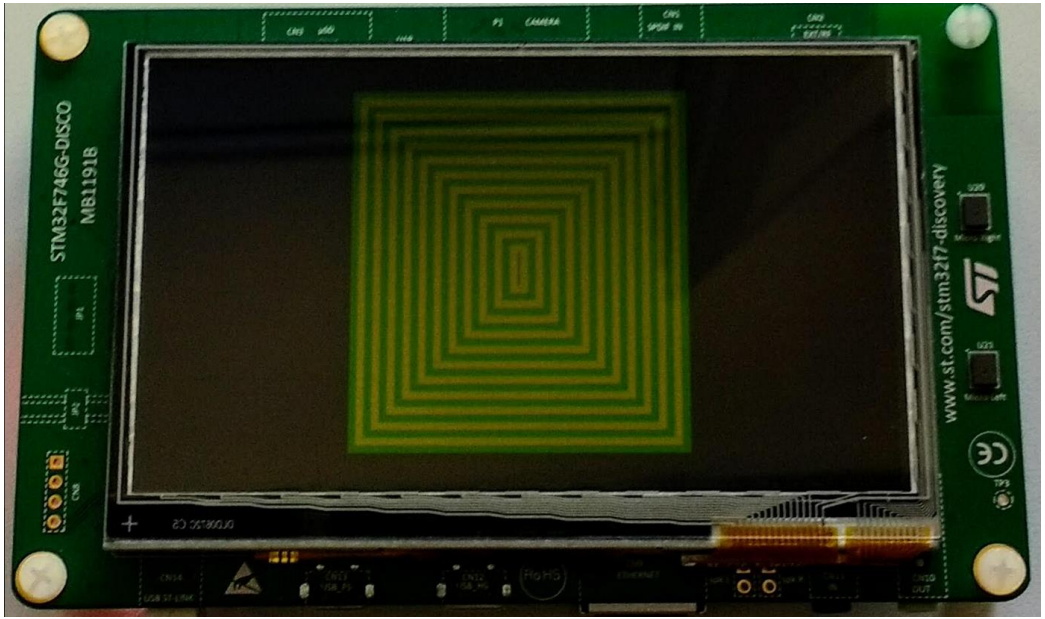
Display Panel Connection



Let's Practice

Task 2:

Generate concentric squares of different colors on the display in a loop without using the graphics library



Using the touch screen

LCD-TFT Touch Panel

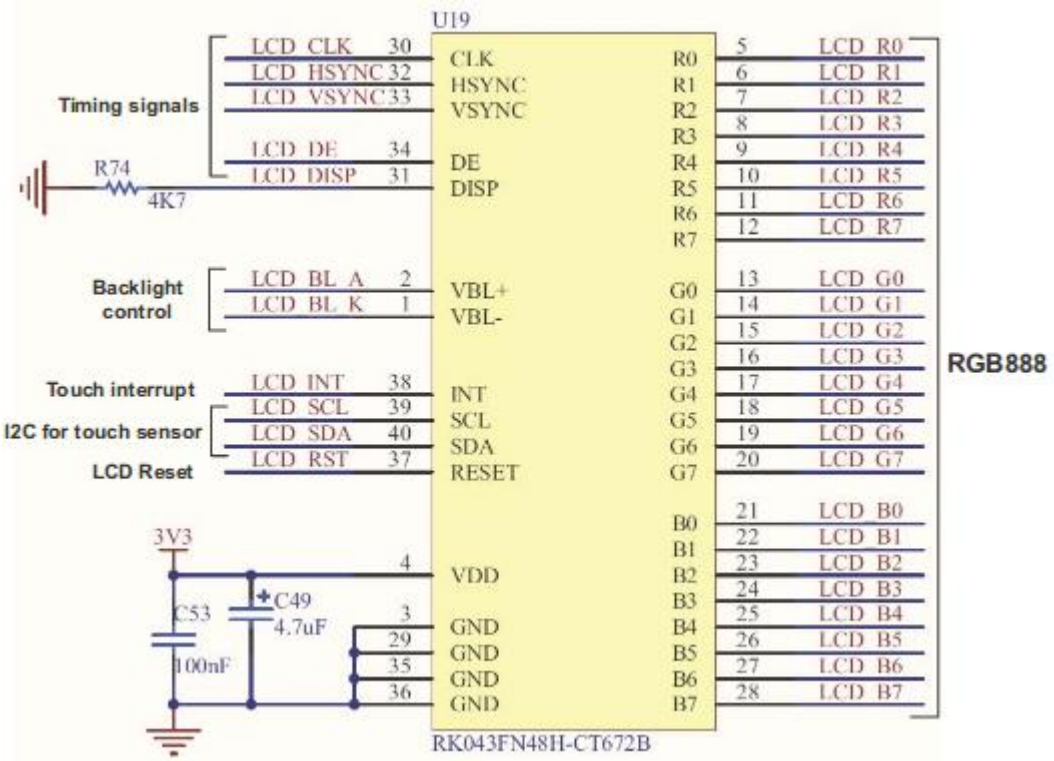
- Capacitive touch panel
- A serial interface I2C (Inter-Integrated Circuit) connects the display panel to the touch sensor

LTDC Pins and Signal Interface specific to the touch sensor

Other specific pins to the touch sensor are :

LCD-TFT Signals	Description
LCD_INT	allows the touch sensor to generate interrupts
LCD_SCL & LCD_SDA	control the touch sensor

LTDC Pins and Signal Interface specific to the touch sensor (2)



Configuration in Rust

- Hardware register structure

```
// hardware register structs with accessor methods
use stm32f7::{system_clock, sdram, lcd, i2c, touch, board, embedded};
```

Configuration in Rust

- Hardware extraction in the main function

```
let board::Hardware {
    rcc,
    pwr,
    flash,
    fmc,
    ltdc,
    gpio_a,
    gpio_b,
    gpio_c,
    gpio_d,
    gpio_e,
    gpio_f,
    gpio_g,
    gpio_h,
    gpio_i,
    gpio_j,
    gpio_k,
    i2c_3,
    ..
} = hw;
```

Configuration in Rust

- Initialisation of the I2C interface in the main function

```
// i2c
i2c::init_pins_and_clocks(rcc, &mut gpio);
let mut i2c_3 = i2c::init(i2c_3);
```

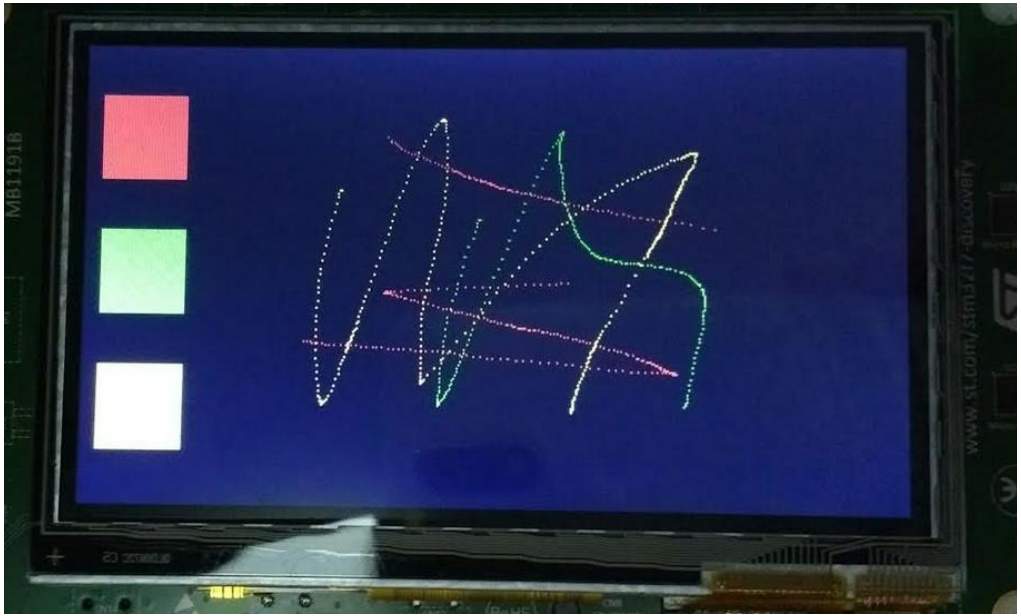
- Connecting the touch screen via the I2C interface

```
touch::check_family_id(&mut i2c_3).unwrap();
```

Let's Practice

Task 3:

Write on the touch screen of the STM32F7 using different colors



Let's Practice

Extra Tasks:

- Modify the size of the drawing on the display
- Erase some lines from your drawing...


That's all Folks!

Let's Practice: Task 1

```
loop {
    //Parameters to generate the
    //LUT (Nbre of pts:480 Max amplitude: 271)
    let y ; // y = LUT of sine
    for x in 1..480 {
        lcd.print_point_at(x, y[x as usize]);
    }
}
```

Let's Practice: Task 2

```
loop {
    let color = [0xff00, 0x0f00]; //yellow;green
    for c in 1..24 {
        let i1 = 124 + 5 * c;
        let i2 = 356 - 5 * c;
        let j1 = 10 + 5 * c;
        let j2 = 262 - 5 * c;
        for i in i1..i2 {
            for j in j1..j2 {
                lcd.print_point_color_at(i, j, color[c as usize & 1]);
            }
        }
    }
    lcd.clear_screen();
}
```

Let's Practice: Task 3

```
let color = [0xf000, 0x0f00, 0xff00, 0xffff];
let w = 80;
for i in 10..60 {
    for j in 30..80 {
        for c in 0..3 {
            lcd.print_point_color_at(i, j + w * c, color[c as usize]);
        }
    }
}
let mut c = 3;
loop {
    for touch in &touch::touches(&mut i2c_3).unwrap() {
        let x = touch.x;
        let y = touch.y;
        if x > 10 && x < 60 {
            if y > 30 && y < 80 {
                c = 0;
            } else if y > 30 && y < (80 + w) {
                c = 1;
            } else if y > 30 && y < (80 + 2 * w) {
                c = 2;
            }
        }
    }
    for touch in &touch::touches(&mut i2c_3).unwrap() {
        lcd.print_point_color_at(touch.x, touch.y, color[c as usize]);
    }
}
```