Giao thức SPI (Serial Peripheral Interface):

- Một chút lịch sử: SPI là một chuẩn đồng bộ nối tiếp do công ty Motorola thiết kế nhằm đảm bảo sự liên lạc giữa các vi điều khiển và thiết bị ngoại vi một cách đơn giản giá thành rẻ vào năm 1980 . Và thật không may nó đã trở thành một tiêu chuẩn thực tế mà mỗi chúng ta phải tìm hiểu.

Lý do tại sao có uart và i2c mà ta vẫn phải có thêm spi:

- Theo cá nhân mình thì do nó sài xung clock nếu ở một tầm nào đó nó sẽ chính xác hơn uart và còn với i2c thủ tục nó đơn giản hơn, cộng với truyền song công nên SPI tồn tại được là như vậy, còn khách quan thì do thích thì nó còn thôi.
- **Note**: khi dữ liệu chỉ có thể truyền theo một hướng (master->slave hay slave->master) trong một thời điểm thì đó được gọi là bán song công (half duplex), còn trong một thời điểm dữ liệu có thể truyền theo cả hai hướng được gọi là song công (full duplex)

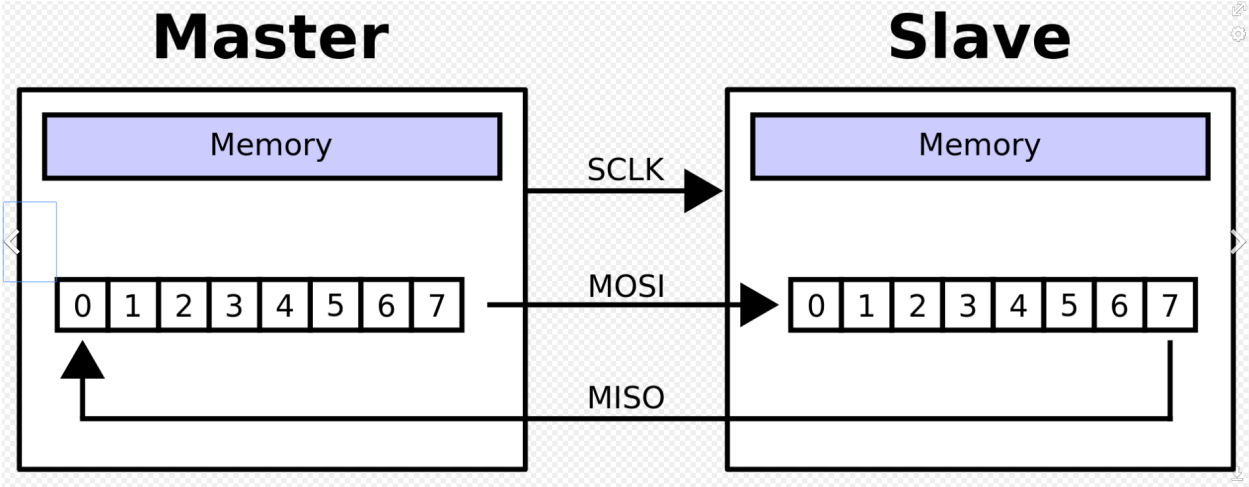
Cấu tạo của một giao thức SPI:

Vì là giao thức đồng bộ nên việc truyền nhận dữ liệu bằng tín hiệu xung clock phát ra từ vi điều khiển.

Giao thức SPI cơ bản giữa một master và slave cần 3 dây, với nhiều hơn 2 slave cần thêm dây slave select. Công dụng các dây:

- SCLK: Cung cấp xung clock.
- MOSI: Master out slave in, dành cho việc truyền dữ liệu từ master đến slave.
- MISO: Master in slave ou, dành cho việc nhận dữ liệu từ slavs về master.
- SS: slave select, chân chọn slave, active

Nguyên lý truyền nhận:

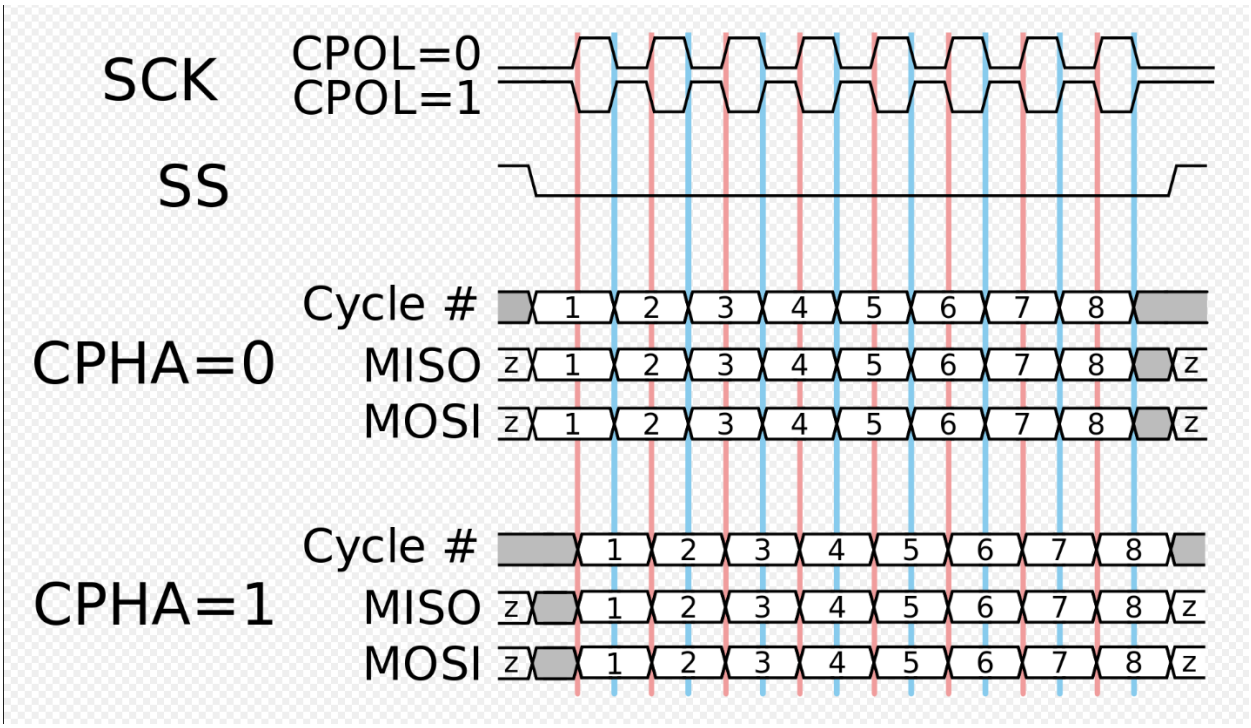


Khi master muốn nói chuyện với 1 slave nó sẽ kéo chân SS xuống mức 0. Sau đó quá trình thuền nhận bắt đầu.

Giả sử cả bên master và slave đều truyền nhận bằng thanh ghi 8 bit . Bit có trọng số thấp nhất được truyền nhận tại mỗi cạnh xung, mỗi bit dữ liệu truyền nhận thì các bit lại được đổi trọng số một lần, sau khi xác nhận thanh ghi đã được truyền nhận đúng hai bên thay đổi thanh ghi và tiếp tục quá trình truyền nhận. Quá trình này diễn ra cho đến khi hết data cần truyền.

Thanh ghi dịch có thể là 8 bit hoặc một kich thước khác ví dụ 16 bit như trong màn hình cảm ứng, 12 bit trong các bộ chuyển đổi ADC hay DAC

Clock và phase (theo Motorola SPI block guide):



CPOL loại clock :

CPOL=0 là loại clock bằng 0 ở trạng thái bình thường và lên 1 ở trên xung: xung mở đầu sẽ là xung cạnh lên, xung kết thúc là xung cạnh xuống.

CPOL=1 là loại clock bằng 1 ở trạng thái bình thường và xuống 0 khi xảy ra xung: xung mở đầu sẽ là xung cạnh xuống, xung kết thúc là xung cạnh lên.

CPHA cách lấy dữ liệu:

CPHA=0 : dữ liệu sẽ được lấy ở xung cạnh lên và thay đổi ở cạnh xuống, Trong chu kì đầu bit đầu cần nạp vào MISO trước khi clock diễn ra.

CPHA=1 : dữ liệu sẽ được lấy ra ở xung cạnh xuống của clock và thay đổi ở cạnh lên của xung clock. Trong chu kì cuối, slave nạp vào giá trị MOSI khi chân slave select chuyể trạng thái từ 0 sang 1.

Khi kết hợp 2 bit quy định CPOL là bit cao và CPHA là bit hấp lại ta được.

Với các dòng microchip PIC, ARM-base , với dòng (PIC32MX: set SPI mode bằng cách set CKP=1, và CKP, CKE config theo bảng

SPI mode			**Cạnh xung lấy dữ liệu**

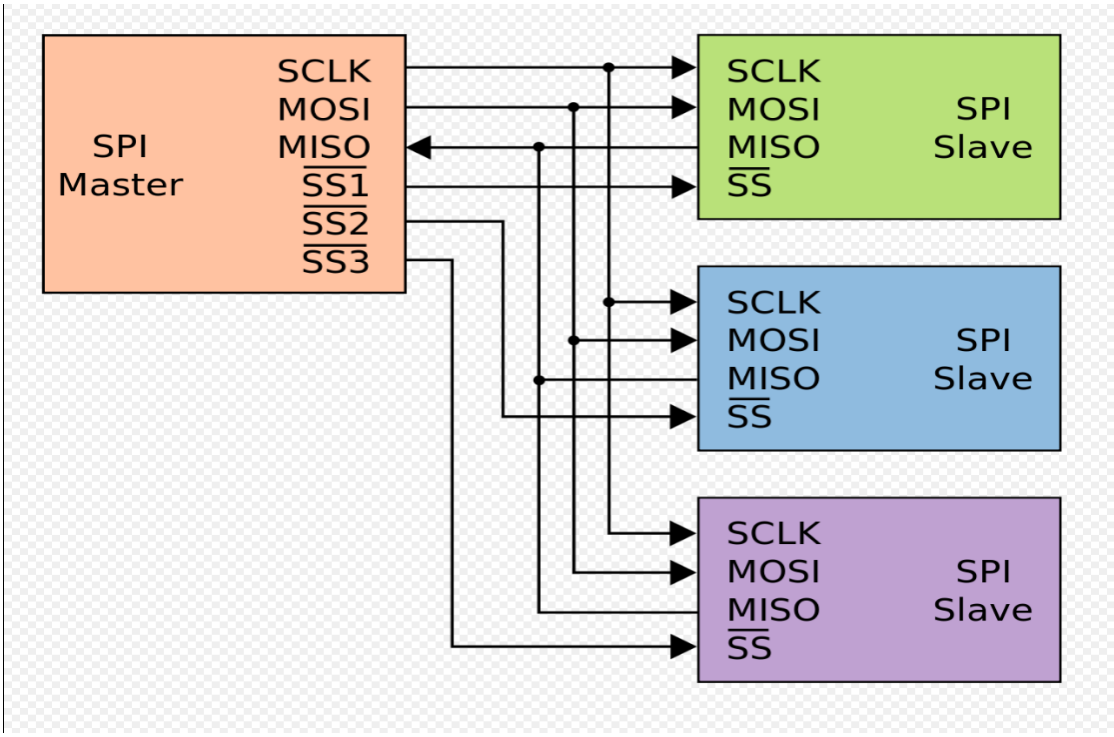
	(CPOL/CKP)	(CPHA)	(CKE/NCPHA)
0	0	0	1(xung cạnh lên)
1	0	1	0(xung cạnh xuống)
2	1	0	1(xung cạnh lên)
3	1	1	0(xung cạnh xuống)

Với các vi điều khiển khác cần.

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Sơ đồ kết nối:

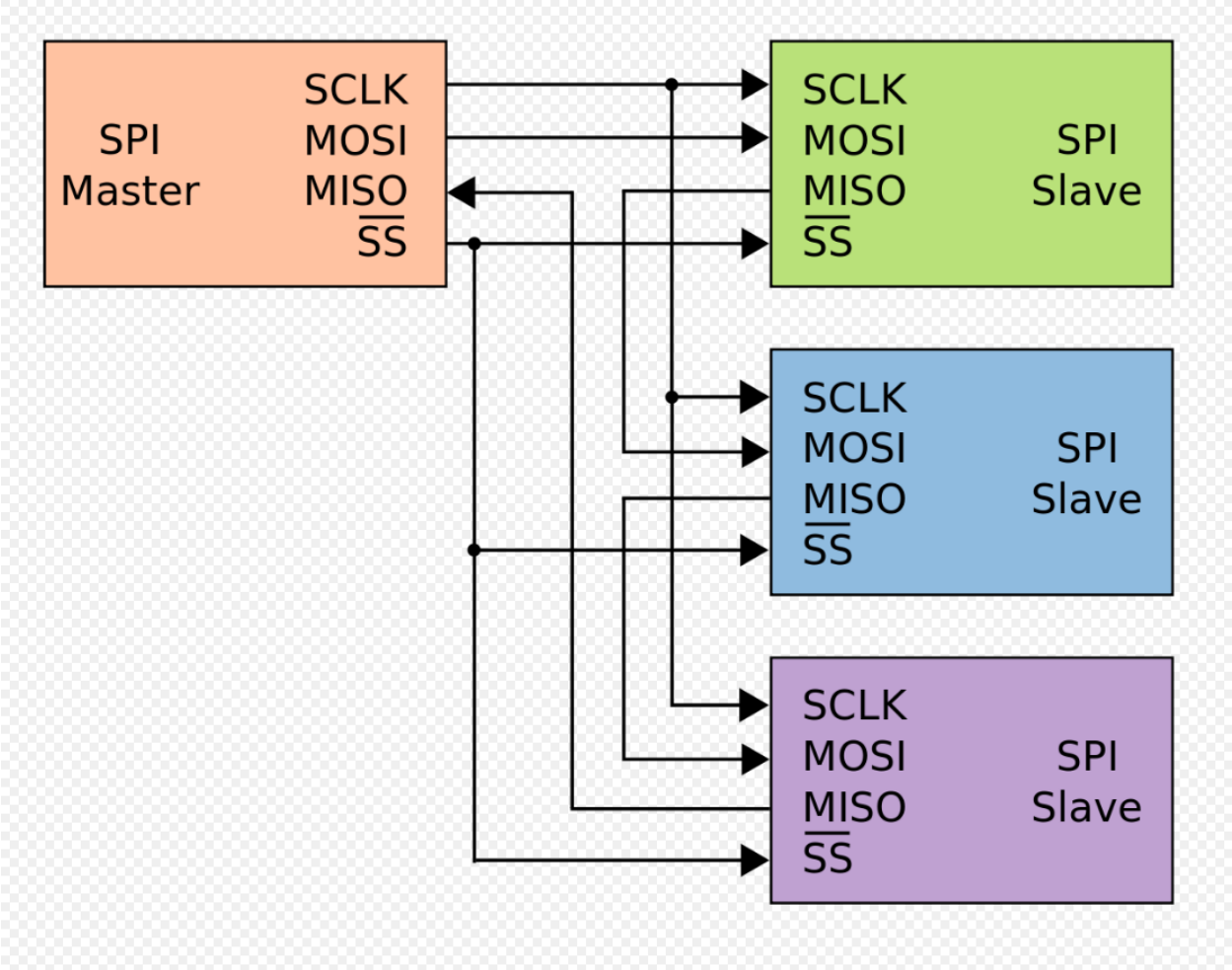
1) Các slave độc lập với nhau(independent slave configuration).



Trong kiểu nối này, mỗi con chip có một chân select nối thằng đến master chân này sẽ được kéo xuống mức 0 khi master muốn giao tiếp với slave đó. Điều kiện cần của các slave khi MOSI, hay MISO phải là cổng 3 trạng thái, nếu không phải thì cần phải có một ic đệm.

Để tiết kiệm chân chip select ta có thể dùng thêm các ic giải mã để tiết kiệm chân chip select.

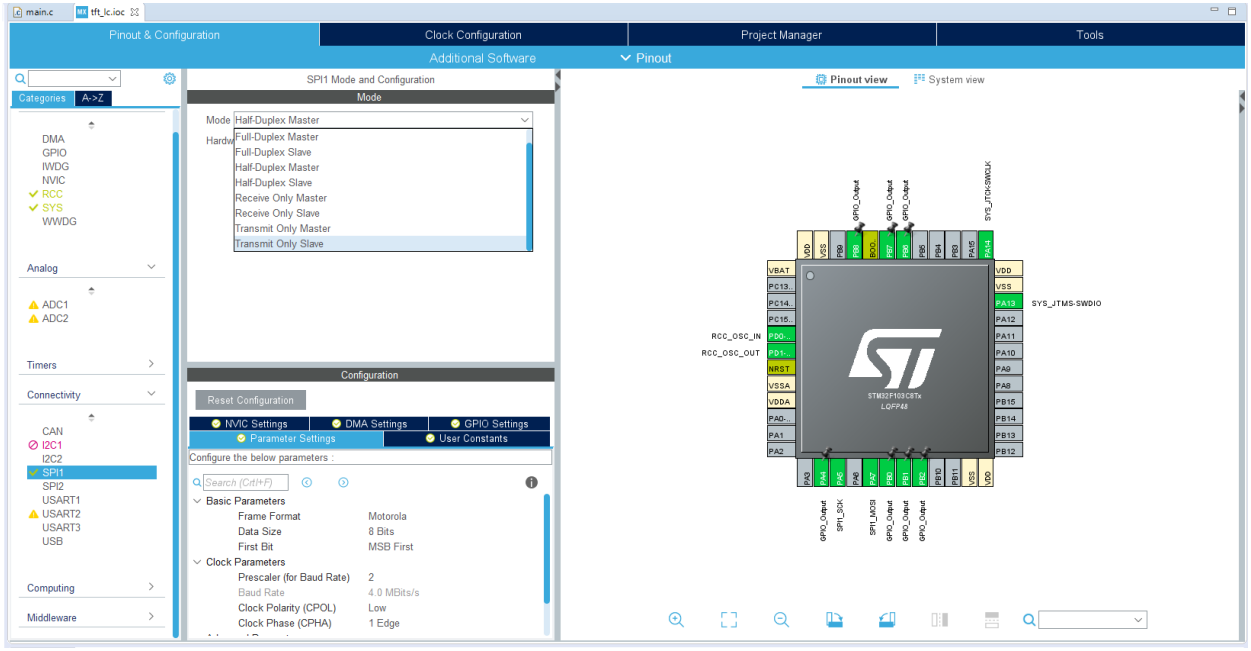
2) Mắc theo chuỗi



Trong kiểu nối dây này các dây slave select được nối với nhau , một MOSI của slave đầu được nối với master, MISO của slave thứ nhất nối với MOSI của slave thứ hai và cứ tiếp tục như vậy cho tới slave cuối cùng.

Cách giao tiếp dữ liệu. Sau khi slave nhận dữ liệu từ master, slave sẽ sử lý dữ liệu và truyền đi một bản copy tương tự đến slave thứ 2 cho đến slave cuối cùng hoặc khi chân chip select được đưa lên mức cao.

Thực hành: cho stm32c8t6 giao tiếp với màn hình lcd st 7735



Config: chọn nguồn cấp xung clock từ thạch anh ngoài trong RCC, chọn debug bằng serial wire.

Vào mục connectivity:

Chọn mode :half-duplex Master vì chỉ cần giao tiếp theo 1 chiều.

Chọn theo frame Motorola vì nó nghĩ ra cái này.

Data size 8 bít, Fist bit MSB, prescaler 2 (Cpol) low :idle state = 0, CPHA : 1 edge lấy mẫu tại cạnh lên.

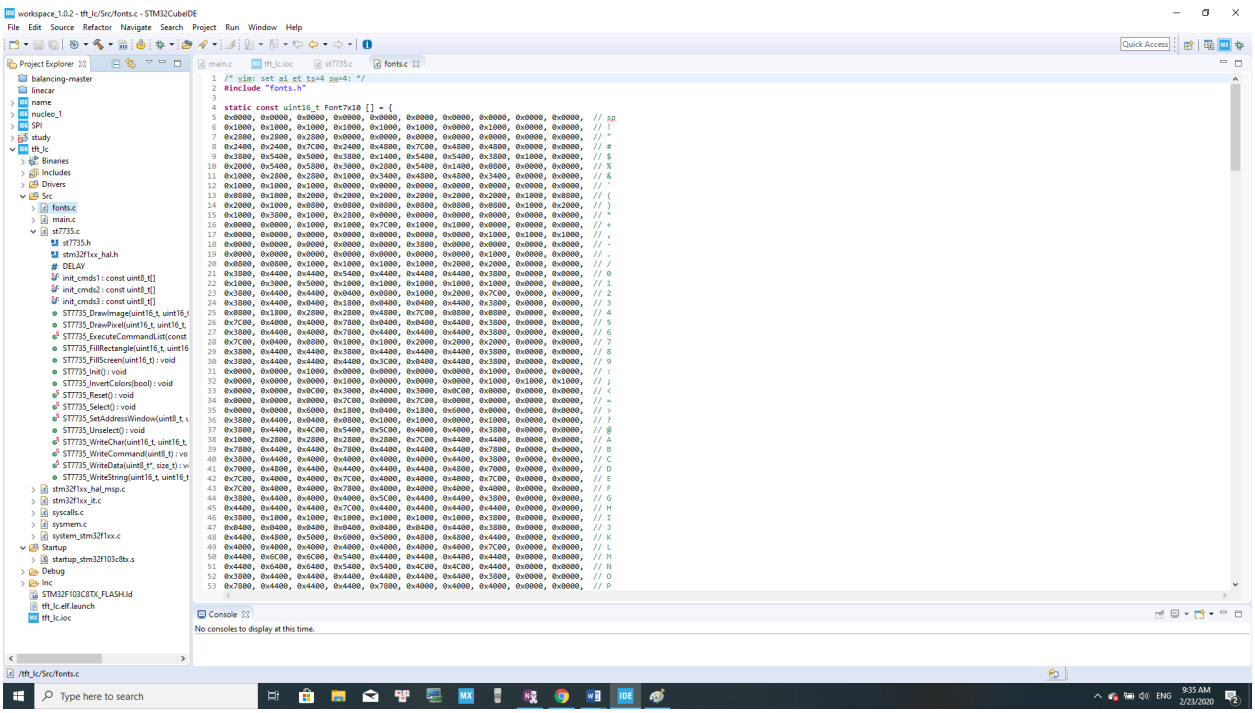
Sau khi generate code ta được:

```
 */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}
/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_1LINE;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
```

Hàm systemClock_Config và hàm MX_SPI1_Init là hàm để config các thông số cơ bản.

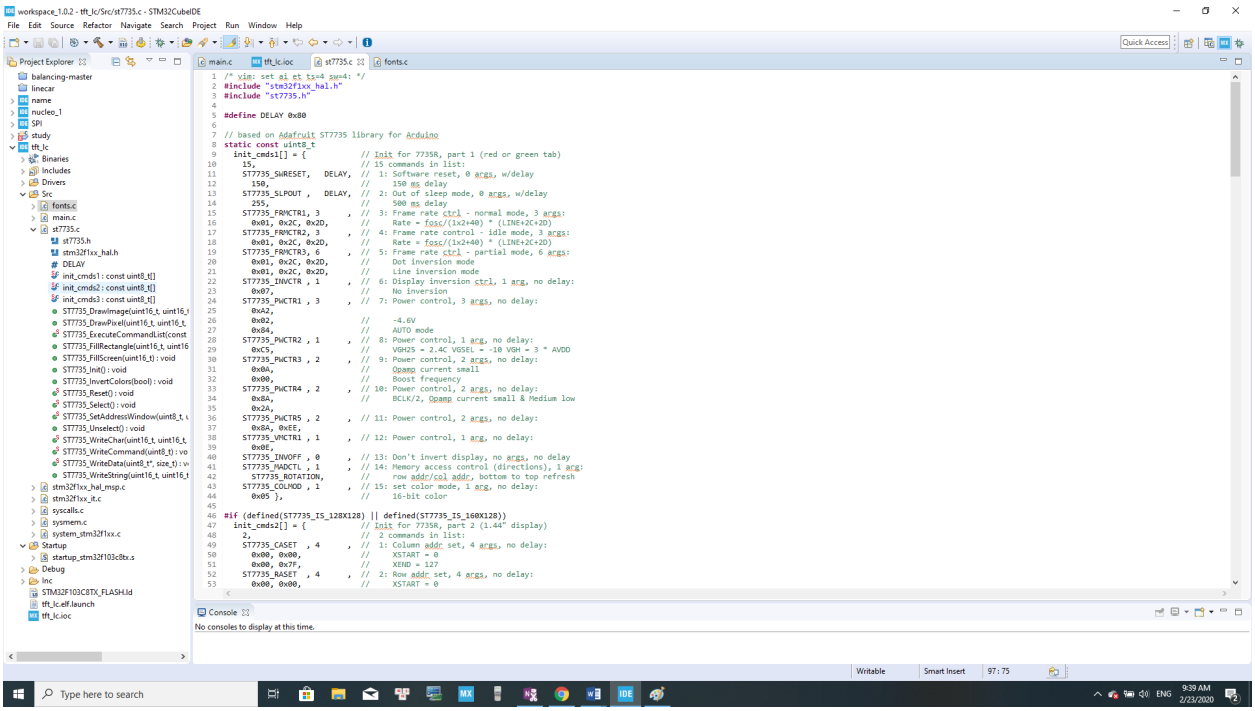
Để có thể điều khiển được các con led (một điểm ảnh ta cần một đoạn code theo datasheet). Mà ở đây cần xây dựng một thư viện nên cách nhanh nhất là đi địa thư viện và học cách sài nó.

Thư viện font.c



Cách điều khiển các mà hình dạng điểm ảnh là đưa con trỏ đến vị trí mong muốn và hiện kí tự ra để làm được việc này ta cần một thư viện nữa.

St7735.c



Để sử dụng màn hình này các bạn cần vào file st7735.h trong thư mục Inc và sửa các trạng thái như sau:

```
/*** Redefine if necessary ***/
#define ST7735_SPI_PORT hspi1
extern SPI_HandleTypeDef ST7735_SPI_PORT;

#define ST7735_RES_Pin       GPIO_PIN_2
#define ST7735_RES_GPIO_Port GPIOB
#define ST7735_CS_Pin        GPIO_PIN_1
#define ST7735_CS_GPIO_Port  GPIOB
#define ST7735_DC_Pin        GPIO_PIN_0
#define ST7735_DC_GPIO_Port  GPIOB
```

Trên board Chicken đã có sẵn sơ đồ nối, các em nối chân và setup project như board và chạy thử.

Để tạo ảnh, chúng ta sử dụng đoạn code gen_img.py đã có trong github của anh (code python, đã yêu cầu cài opencv):

```python
#!/usr/bin/env python

# vim: set ai et ts=4 sw=4:

import cv2
import sys
import os
import numpy as np

img = cv2.imread("pif.png")
img = cv2.resize(img, (128,128))
img = cv2.flip( img, 1 )
img = cv2.flip( img, 0 )
# img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

for y in range(0, img.shape[0]):
    s = "{"
    for x in range(0, img.shape[1]):
        (b, g, r) = img[x, y,:]
        color565 = ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >> 3)
        # for right endiness, so ST7735_DrawImage would work
        color565 = ((color565 & 0xFF00) >> 8) | ((color565 & 0xFF) << 8)
        s += "0x{:04X},".format(color565)
    s += "},"
    print(s)

print("};")
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
```

Phần giải thích sẽ có trong các bài tut OpenCV sau này, để sử dụng cần lưu một ảnh định dạng pif.png vào chung thư mục source và chạy file này (có thể dùng terminal hoặc IDE).

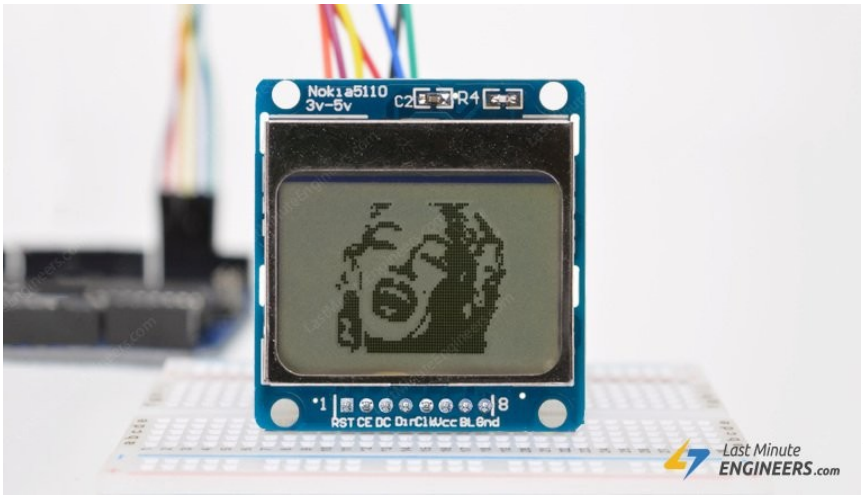
*Bài tập: Hiển thị hình người yêu lên màn hình :)))))

Ví dụ về software SPI:

Trong thực tế, ta có thể sử dụng chân GPIO để thực hiện các giao tiếp UART, I2C, SPI,…

Trong đó anh có một chương trình mẫu là một game nhỏ trên màn hình Nokia5110:

Màn Hình Nokia 5110



Màn hình Nokia5110 sử dụng chip driver PCD8544 của Philips, driver này cho phép sử dụng và quản lý màn hình graphic 84*48 pixel Nokia5110. Thông số của IC này như sau:

Single chip LCD controller/driver

•48 row, 84 column outputs

• Display data RAM 48 × 84 bits

• On-chip: – Generation of LCD supply voltage (external supply also possible) – Generation of intermediate LCD bias voltages – Oscillator requires no external components (external clock also possible).

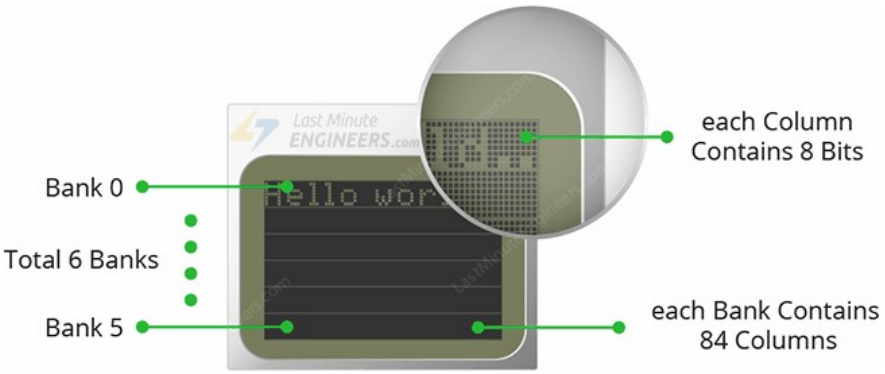
• External RES (reset) input pin

• Logic supply voltage range VDD to VSS: 2.7 to 3.3 V • Display supply voltage range VLCD to VSS – 6.0 to 8.5 V with LCD voltage internally generated (voltage generator enabled) – 6.0 to 9.0 V with LCD voltage externally supplied (voltage generator switched-off).

(Trích datasheet của IC).

Memory Map

Bộ nhớ 84 * 48 được tổ chức thành 6 băng thanh ghi với mỗi băng gồm 8bit mỗi hàng và 84 hàng quản lý toàn bộ các điểm ảnh của màn hình. Tổng số bit do đó là 8*84*48 = 4032 bits



Màn hình Nokia5110 sử dụng chip điều khiển PCD8544 của Philips, sử dụng giao thức SPI, trong đề tài này phương pháp software SPI được sử dụng.

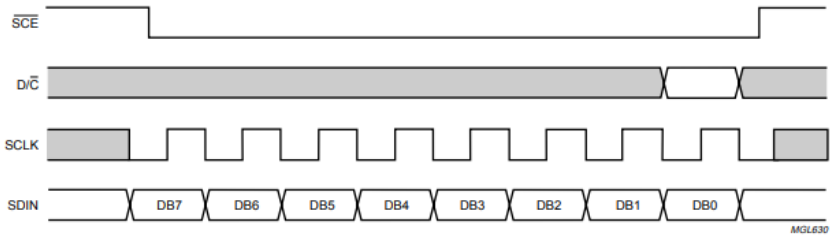


Fig.10 Serial bus protocol - transmission of one byte.

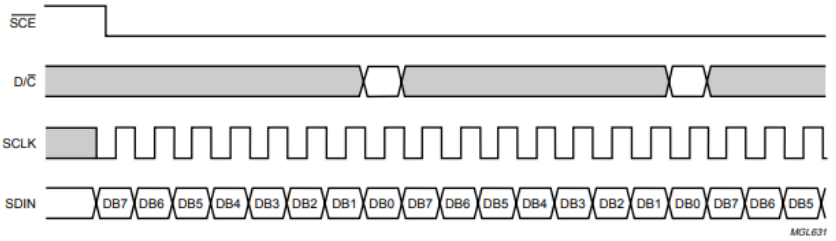


Fig.11 Serial bus protocol - transmission of several bytes.

Cấu trúc frame truyền của IC driver.

Table 1 Instruction set

INSTRUCTION	D/$\overline{C}$	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H = 0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	no operation
Function set	0	0	0	1	0	0	PD	V	H	power down control; entry mode; extended instruction set control (H)
Write data	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	writes data to display RAM
(H = 0)										
Reserved	0	0	0	0	0	0	1	X	X	do not use
Display control	0	0	0	0	0	1	D	0	E	sets display configuration
Reserved	0	0	0	0	1	X	X	X	X	do not use
Set Y address of RAM	0	0	1	0	0	0	Y_2	Y_1	Y_0	sets Y-address of RAM; $0 \le Y \le 5$
Set X address of RAM	0	1	X_6	X_5	X_4	X_3	X_2	X_1	X_0	sets X-address part of RAM; $0 \le X \le 83$
(H = 1)										
Reserved	0	0	0	0	0	0	0	0	1	do not use
	0	0	0	0	0	0	0	1	X	do not use
Temperature control	0	0	0	0	0	0	1	TC_1	TC_0	set Temperature Coefficient (TC_x)
Reserved	0	0	0	0	0	1	X	X	X	do not use
Bias system	0	0	0	0	1	0	BS_2	BS_1	BS_0	set Bias System (BS_x)
Reserved	0	0	1	X	X	X	X	X	X	do not use
Set V_{OP}	0	1	V_{OP6}	V_{OP5}	V_{OP4}	V_{OP3}	V_{OP2}	V_{OP1}	V_{OP0}	write V_{OP} to register

Table 2 Explanations of symbols in Table 1

BIT	0	1
PD	chip is active	chip is in Power-down mode
V	horizontal addressing	vertical addressing
H	use basic instruction set	use extended instruction set
D and E		
00	display blank	
10	normal mode	
01	all display segments on	
11	inverse video mode	
TC_1 and TC_0		
00	V_{LCD} temperature coefficient 0	
01	V_{LCD} temperature coefficient 1	
10	V_{LCD} temperature coefficient 2	
11	V_{LCD} temperature coefficient 3	

Bảng command của IC

*Giải thích ngắn gọn quy trình truyền:

Dựa theo cấu trúc frame truyền đã được kể ở trên, ta thấy quá trình truyền data có thể được mô tả như sau:

3) Để bắt đầu qúa trình truyền, chân CE của IC được kéo xuống thấp.

2. Quá trình truyền data được thực hiện khi có cạnh lên của tín hiệu chân CLK, IC sẽ đọc data ở chân DIN

3. Một frame truyền có độ dài 8bit (1byte) được ghi vào thanh ghi dịch của chip để đọc 1byte 1 lần.

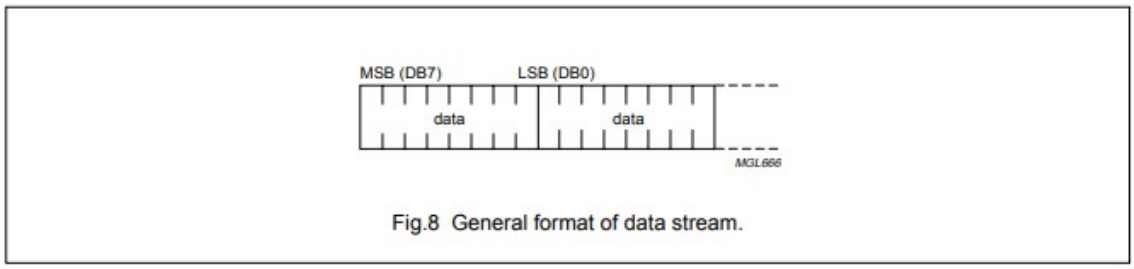


Fig.8 General format of data stream.

Tùy thuộc vào trạng thái của chân DC mà IC driver có thể hiểu lệnh được thực thi là commnad điều khiển IC hoặc là dữ liệu được đưa vào IC.

*Hàm truyền dữ liệu:

```
/*
 * @brief Send information to the LCD using configured GPIOs
 * @param val: value to be sent
 */
void LCD_send(uint8_t val){
  uint8_t i;
  for(i = 0; i < 8; i++){
    HAL_GPIO_WritePin(lcd_gpio.DINPORT, lcd_gpio.DINPIN, (val & (1 << (7 - i))));
    HAL_GPIO_WritePin(lcd_gpio.CLKPORT, lcd_gpio.CLKPIN, GPIO_PIN_SET);
    HAL_GPIO_WritePin(lcd_gpio.CLKPORT, lcd_gpio.CLKPIN, GPIO_PIN_RESET);
  }
}
/*
 * @brief Writes some data into the LCD
 * @param data: data to be written
 * @param mode: command or data
 */
void LCD_write(uint8_t data, uint8_t mode){
  if(mode == LCD_COMMAND){
    HAL_GPIO_WritePin(lcd_gpio.DCPORT, lcd_gpio.DCPIN, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_RESET);
    LCD_send(data);
    HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_SET);
  }
  else{
    HAL_GPIO_WritePin(lcd_gpio.DCPORT, lcd_gpio.DCPIN, GPIO_PIN_SET);
    HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_RESET);
    LCD_send(data);
    HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_SET);
  }
}
```

Hàm LCD_send sẽ gửi đi một byte dữ liệu, hàm LCD_write sẽ thực hiện việc set pin DC để chọn mode hoạt động send data hoặc send command.

Để màn hình hoạt động, đầu tiên ta cần Init màn hình như sau:

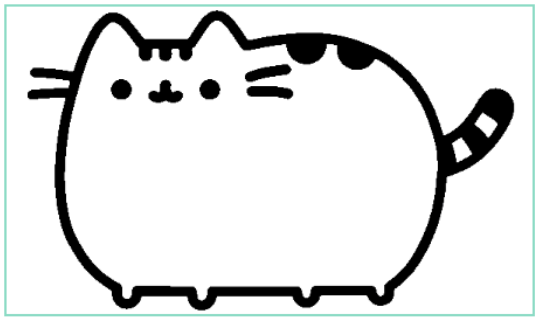
Các thông số init này dựa trên bảng command đã được đề cập.

```
108 void LCD_init(){
109   HAL_GPIO_WritePin(lcd_gpio.RSTPORT, lcd_gpio.RSTPIN, GPIO_PIN_RESET);
110   HAL_GPIO_WritePin(lcd_gpio.RSTPORT, lcd_gpio.RSTPIN, GPIO_PIN_SET);
111   LCD_write(0x21, LCD_COMMAND); //LCD extended commands.
112   LCD_write(0xBA, LCD_COMMAND); //set LCD Vop(Contrast).
113   LCD_write(0x04, LCD_COMMAND); //set temp coefficent.
114   LCD_write(0x14, LCD_COMMAND); //LCD bias mode 1:40.
115   LCD_write(0x20, LCD_COMMAND); //LCD basic commands.
116   LCD_write(0x0C, LCD_COMMAND); //LCD basic commands.
117
118   LCD_clrScr();
119  // LCD_write(0x09, LCD_COMMAND);
120 }
```

Để hiển thị ảnh trên màn hình:

Khi truyền data vào màn hình, từng byte data sẽ được đưa vào lần lượt theo từng băng thanh ghi từ byte 0 đến byte 504, mỗi byte sẽ quy đinh mỗi pixel là sáng hay tối. Do đó để hiển thị ảnh trên màn hình ta cần lưu vào một buffer 504 byte để hiển thị ra màn hình

Để hiển thị một ảnh bitmap bất kì ta sử dụng công cụ Image2Cpp như sau:



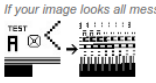


image2cpp

image2cpp is a simple tool to change images into byte arrays (or your array back into an image) for use with Arduino and (monochrome) displays such as OLEDs. It was originally made to work with the Adafruit OLED library. An example sketch for Arduino and this library can be found here.

More info (and credits) can be found in the Github repository. This is also where you can report any issues you might come across.

This tool also works offline. Simply save this page to your computer and open the file in your browser.

1. Select image

pusheen-copy.jpg

or

1. Paste byte array

128 x 64 px

2. Image Settings

Canvas size(s): pusheen-copy.jpg (file resolution: 600 x 350) 600 x 350 glyph

Background color: ● White ○ Black ○ Transparent

Invert image colors ☐

Brightness / alpha threshold: 128

0 - 255; if the brightness of a pixel is above the given level the pixel becomes white, otherwise they become black. When using alpha, opaque and transparent are used instead.

Scaling original size

3. Preview

4. Output

Code output format plain bytes

Draw mode: Horizontal - 1 bit per pixel

If your image looks all messed up on your display, like the image below, try using a different mode.

```
// 'pusheen-copy', 600x350px
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
```

Các file ảnh này được lưu trong các mảng:

```
char bird[] = {0x00, 0x20, 0x20, 0x30, 0x28, 0x48, 0x84, 0x04, 0x74, 0x0c, 0x84, 0xe8, 0x90, 0xe0, 0x80, 0x00,
       0x00, 0x01, 0x06, 0x0a, 0x0a, 0x11, 0x10, 0x10, 0x12, 0x15, 0x0a, 0x0a, 0x0a, 0x0a, 0x06, 0x01};
char bird_down[] = {0x00, 0x10, 0x88, 0x84, 0x08, 0x8c, 0xe4, 0x04, 0x44, 0xb4, 0x18, 0x08, 0x90, 0x60, 0x00, 0x00,
    0x00, 0x00, 0x02, 0x06, 0x08, 0x08, 0x10, 0x06, 0x16, 0x15, 0x2a, 0x2a, 0x34, 0x07, 0x00, 0x00};
char bar[] = {0x00, 0x00, 0x00, 0xbd, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81
};
// '0', 24x40px
const unsigned char myBitmap [10][120]  = {{
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xc0, 0xc0, 0xe0, 0xc0, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0xfc,
          0xfe, 0xff, 0x07, 0x03, 0x01, 0x83, 0xff, 0xff, 0xfc, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff, 0x00, 0xc0, 0x7c, 0x07, 0x07, 0xff,
          0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff,
          0xff, 0xff, 0x1e, 0x01, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x0f, 0x1c, 0x78, 0x70, 0x38, 0x1c, 0x0f,
          0x07, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
      },{
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0xc0, 0xc0, 0xe0, 0xf0,
          0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x03, 0x03, 0x43, 0xe3, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xdf, 0xff, 0xff, 0xff, 0x3f,
          0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
      },{
```

Để thực hiện in hình ra buffer ở một điểm xác định ta sử dụng hàm in từng pixel, sau đó ta in từng pixel ra buffer:

```
38  void LCD_set_pixel_im(uint8_t x, uint8_t y, uint8_t pixel){
39    if(x >= LCD_WIDTH)
40      x = LCD_WIDTH - 1;
41    if(y >= LCD_HEIGHT)
42      y = LCD_HEIGHT - 1;
43
44    if(pixel != 0){
45        buffer[x + (y / 8) * LCD_WIDTH] |= 1 << (y % 8);
46        if(buffer[x + (y / 8) * LCD_WIDTH] == 0){
47  }
48    }
49  else{
50
51  }
52  }
```

Như đã nói ở phần trên, buffer là một mảng 504 phần tử uint8_t, mỗi phần tử thể hiện 8 bits của mỗi thanh ghi trong băng thanh ghi. Để truy cập đến pixel có tọa độ (x,y), trong đó x có kích thước 84, y là số kí hiệu băng thanh ghi, gồm 1 - > 6, mỗi băng thanh ghi có 8 vị trí, do đó để chuyển đến vị trí (x,y) ta cần truy cập vào thanh ghi thứ x*84 + y/8 (phép chia lấy phần nguyên) và sau đó thực hiện phép dịch bit 1 << (y%8) để truy cập đến phần tử thứ y. Như trong code ở trên đã thể hiện phương pháp thực hiện này.

Sau đó để thực hiện in ảnh trong buffer ở vị trí (x,y) ta sử dụng hàm sau:

```
void printImage(uint8_t x, uint8_t y, char data [], uint8_t sx,uint8_t sy){
//   for(int i = 0; i<504; i++)
//         buffer[i] = 0x00;
          for(int i = 0 ; i<sx;i++ ){
                    for(int j = 0; j<sy; j++){
                        for(int k = 0; k<8;k++){
                         LCD_set_pixel_im(j+x , y + k + 8*(sx -1 - i) , data[i*sy+j] & (1 << (7 - k)));
                        }
                    }
              }
    }
```

Input của hàm là vị trí ảnh được in (x,y), ảnh được đưa vào là mảng char data [], 2 thông số kích thước ảnh sx và sy, để thực hiện in ảnh ta cần set từng pixel ảnh vào từng vị trí trong buffer. 2 vòng lặp đầu tiên quét qua tất cả các pixel của ảnh, vòng lặp trong cùng quét qua từng vị trí trong thanh ghi của buffer, trong đó từng pixel của data cũng được lấy ra nhờ câu lệnh data[i*sy + j] & (1<<7 - k)).

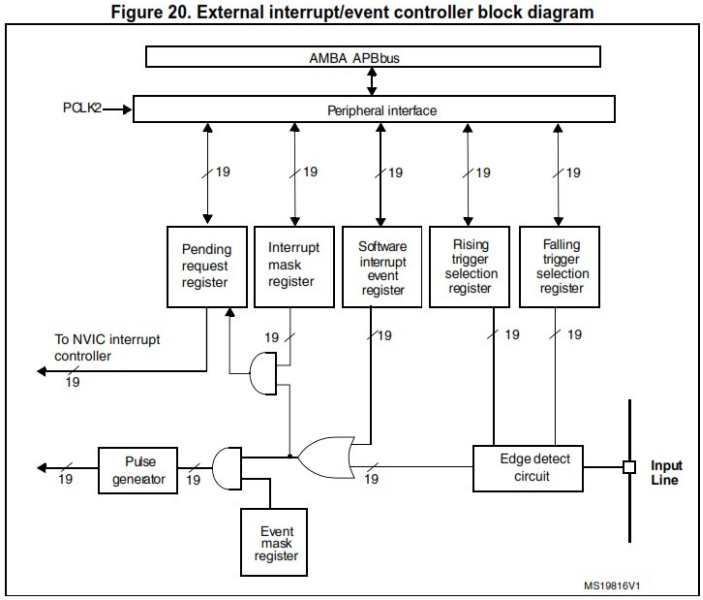
Hàm vẽ thanh chắn của game:

Hàm này vẽ các hình chữ nhật biểu thị cho các thanh chắn của game Flappy Bird, các thông số cần sử dụng là vị trí thanh chắn, chiều cao thanh chắn, độ rộng thanh chắn, khoảng giữa 2 thanh chắn để nhân vật vượt qua.

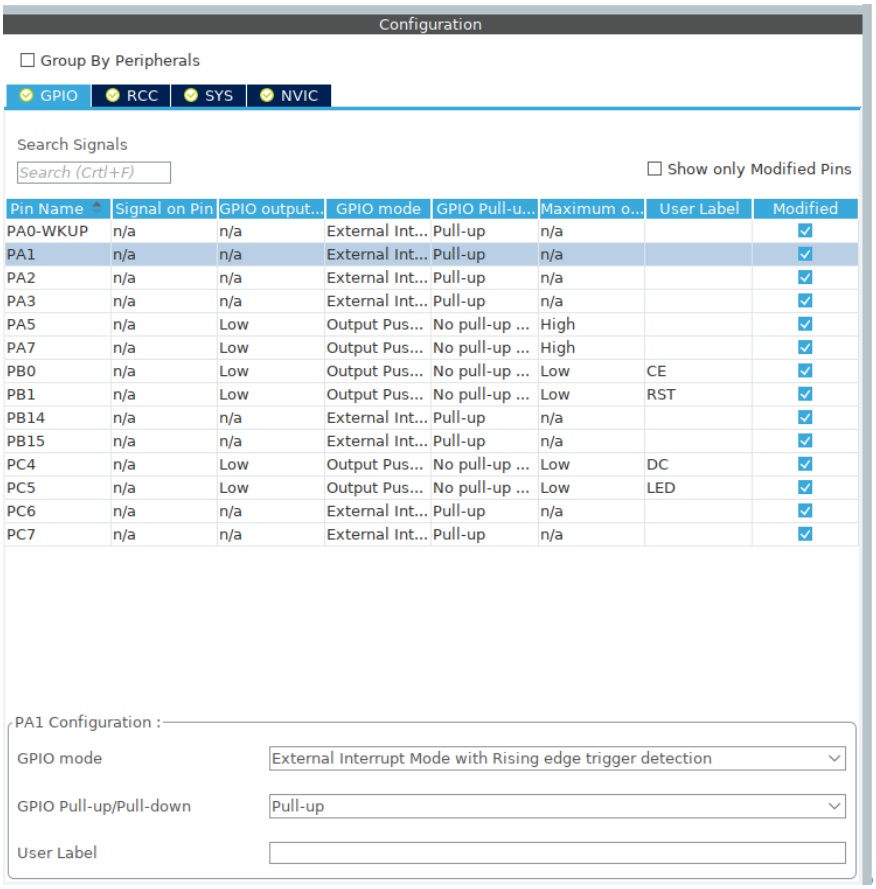
```
void draw_bar(uint8_t x, uint8_t y, uint8_t height, uint8_t width, uint8_

    for(int i = x; i<x+width; i++){
        for(int j = y;j<y+height-6;j++){
            LCD_set_pixel(i,j, 1);
        }
    }
    for(int i = x; i<x+width; i++){
        for(int j = 48;j>  (y+height)+ gap + 6;j--){
            LCD_set_pixel(i,j, 1);
        }
    }

    if(x<82 && x>2){
    for(int i = x-2; i<x+width +2 ; i++){
            for(int j = y+height-5;j<y+height;j++){
                LCD_set_pixel(i,j, 1);
            }
        }
    for(int i = x-2; i<x+width +2 ; i++){
                for(int j = (y+height)+ gap + 5;j>  (y+height)+ gap;j--){
                    LCD_set_pixel(i,j, 1);
                }
            }

//  for(int i = x-2; i<x+width +2 ; i++){
//              for(int j = 48;j>(y+height)- gap ;j++){
//                  LCD_set_pixel(i,j, 1);
//              }
//          }
    }
//  LCD_printBuffer(buffer);
}
```

3. **Điều Khiển Nhân Vật:**

Figure 20. External interrupt/event controller block diagram



Để điều khiển nhân vật, External interrupt/event controller (EXTI) của STM32 được sử dụng, cho phép kiểm soát ngắt trên chân GPIO của STM32. Các GPIO này được setup như trong hình dưới, được nối với các button đã được chống rung hardware và được kéo trở pull up nội.



Bảng ưu tiên ngắt NVIC như sau:


```
43 /* Private variables ---------------------------------------------------------*/
44 /* USER CODE BEGIN PV */
45 extern bird_pos;
46 extern mode;
47 /* USER CODE END PV */
48
49 /* Private function prototypes -----------------------------------------------*/
50 /* USER CODE BEGIN PFP */
```

Trong file stm32f1xx_it.c có các hàm xử lý ngắt EXTI như sau

Các biến toàn cục bird_pos để điều chỉnh vị trí của nhân vật game và mode để điều khiển mode chạy của máy chơi game.

```
222 void EXTI2_IRQHandler(void)
223 {
224   /* USER CODE BEGIN EXTI2_IRQn 0 */
225
226   /* USER CODE END EXTI2_IRQn 0 */
227   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
228   /* USER CODE BEGIN EXTI2_IRQn 1 */
229   HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_5);
230   /* USER CODE END EXTI2_IRQn 1 */
231 }
232
233 /**
234   * @brief This function handles EXTI line3 interrupt.
235   */
236 void EXTI3_IRQHandler(void)
237 {
238   /* USER CODE BEGIN EXTI3_IRQn 0 */
239
240   /* USER CODE END EXTI3_IRQn 0 */
241   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3);
242   /* USER CODE BEGIN EXTI3_IRQn 1 */
243
244       bird_pos += 4;
245   /* USER CODE END EXTI3_IRQn 1 */
246 }
```

Các hàm EXTIx_IRQHandler để thực hiện xử lý ngắt từ EXTI

Sản phẩm :3

Dưới đây là toàn bộ chia sẻ về SPI .

Bài tập kết thúc. Các bạn hãy thử làm mà hình hiện tên mình lên nhé.

Note: sau khi nắm rõ các cách thức hãy sáng tạo cho mình một hình ảnh đặc biệt.

Link code tham khảo code :https://github.com/DangLamTung/stm32code/tree/master/tft_lc/Src?fbclid=IwAR0HYTLboxxsqee5T-nlIi4xsgeZ0rSf4f-eFI03qezALd9iK-0pxIZ3CL0

https://github.com/DangLamTung/stm32Gamer

Link cho datasheet: https://html.alldatasheet.com/html-pdf/326213/SITRONIX/ST7735/383/1/ST7735.html