

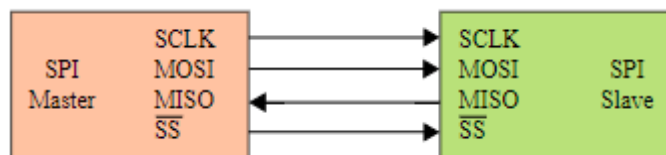
Giao thức SPI (Serial Peripheral Interface):

- 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 sai 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:

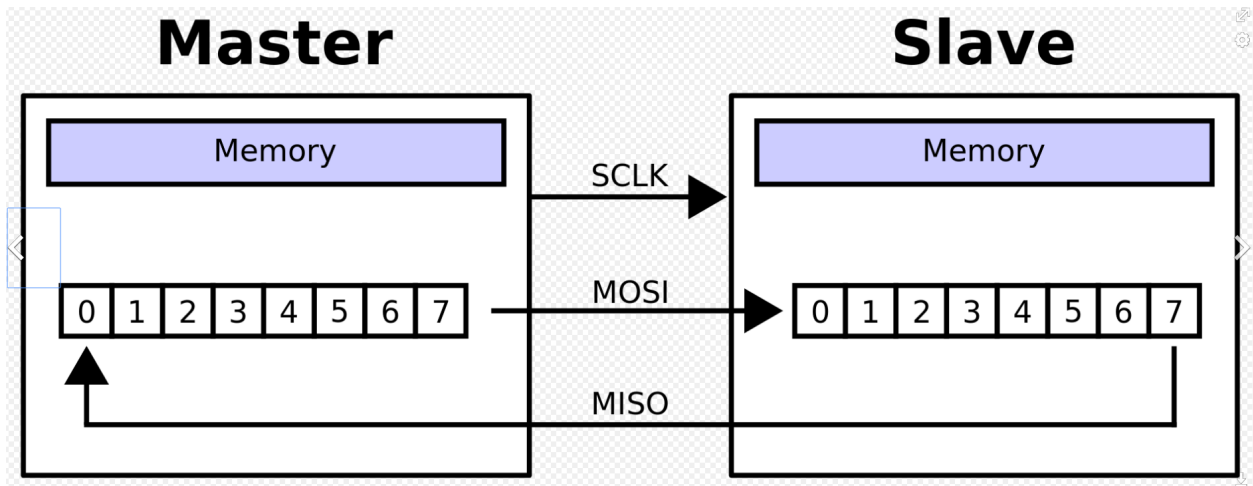


Vì là giao thức đồng bộ nên việc truyền nhận dữ liệu bằng tín hiệu xung clock phát ra từ vi điều khiển.

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 out, dành cho việc nhận dữ liệu từ slave về master.
- ss: slave select, chân chọn slave, active

Nguyên lý truyền nhận:

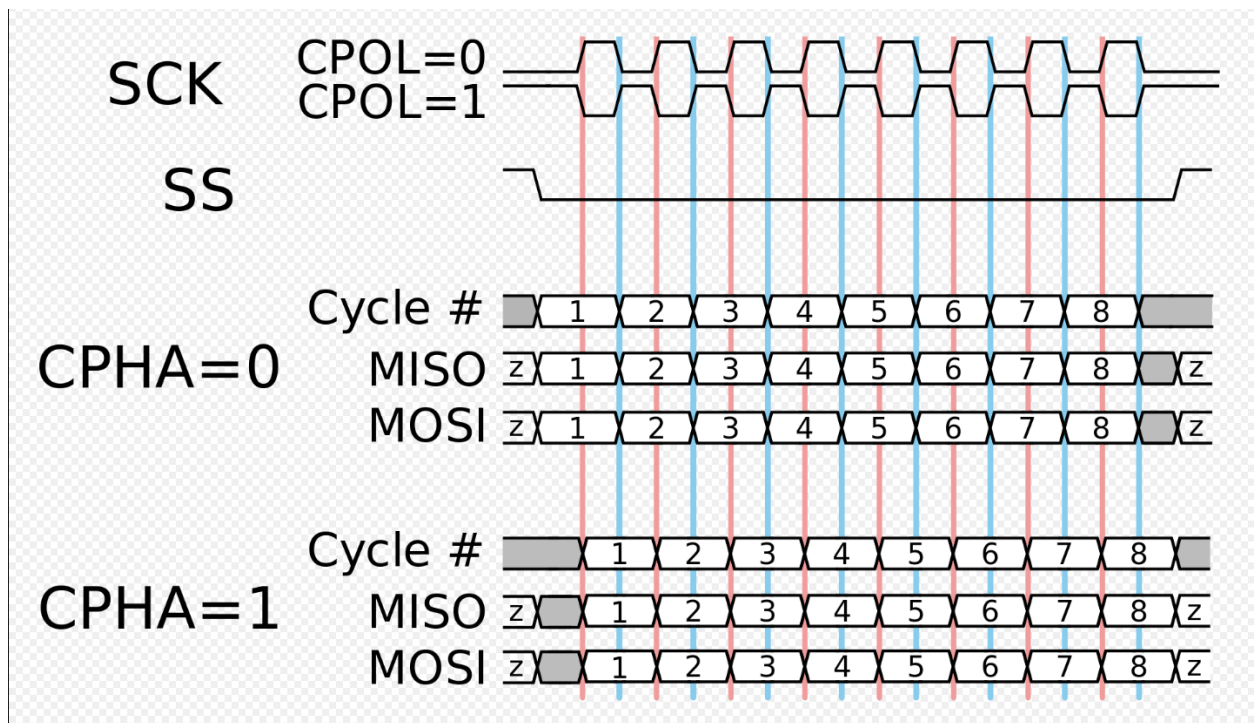


Khi master muốn nói chuyện với 1 slave nó sẽ kéo chân SS xuống mức 0. Sau đó quá trình truyền nhận bắt đầu.

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 kích 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 chuyển trạng thái từ 0 sang 1.

Khi kết hợp 2 bit quy định CPOL là bit cao và CPHA là bit thấp 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
----------	--	--	-----------------------

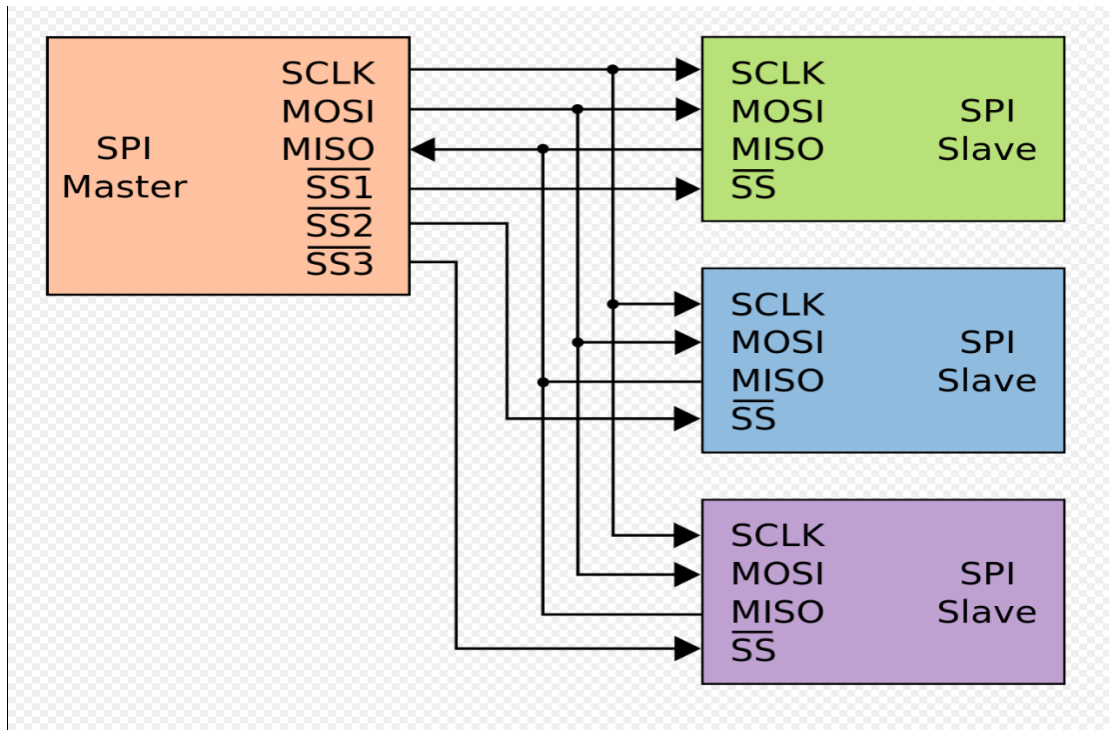
	(CPOL/CKP)	(CPHA)	(CKE/NCPHA)
0	0	0	1(xung cạnh lên)
1	0	1	0(xung cạnh xuống)
2	1	0	1(xung cạnh lên)
3	1	1	0(xung cạnh xuống)

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:

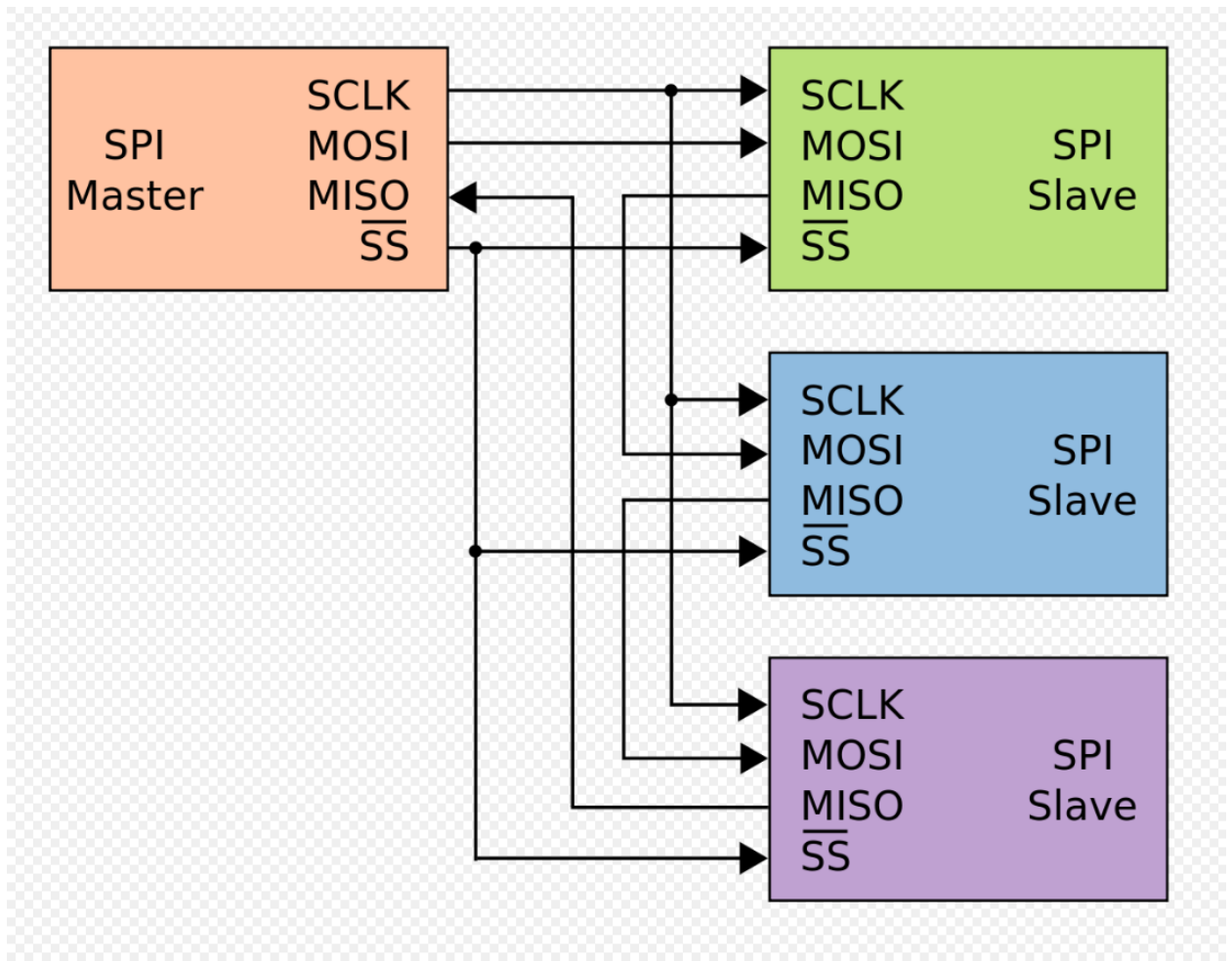
- 1) Các slave độc lập với nhau(independent slave configuration).



Trong kiểu nối này, mỗi con chip có một chân select nối thẳng 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 xử 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.

Vào mục connectivity:

Chọn theo frame Motorola vì nó nghĩ ra cái này.

Sau khi generate code ta được:

```

199  */
200 void SystemClock_Config(void)
201 {
202     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
203     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
204
205     /** Initializes the CPU, AHB and APB busses clocks
206     */
207     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
208     RCC_OscInitStruct.HSISource = RCC_HSI_ON;
209     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
210     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
211     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
212     {
213         Error_Handler();
214     }
215     /** Initializes the CPU, AHB and APB busses clocks
216     */
217     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
218         |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
219     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
220     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
221     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
222     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
223
224     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
225     {
226         Error_Handler();
227     }
228 }
229
230 /**
231  * @brief SPI1 Initialization Function
232  * @param None
233  * @retval None
234  */
235 static void MX_SPI1_Init(void)
236 {
237     /* USER CODE BEGIN SPI1_Init 0 */
238
239     /* USER CODE END SPI1_Init 0 */
240
241     /* USER CODE BEGIN SPI1_Init 1 */
242
243     /* USER CODE END SPI1_Init 1 */
244     /* SPI1 parameter configuration*/
245     hspi1.Instance = SPI1;
246     hspi1.Init.Mode = SPI_MODE_MASTER;
247     hspi1.Init.Direction = SPI_DIRECTION_1LINE;
248     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
249     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
250     hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
251

```

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.

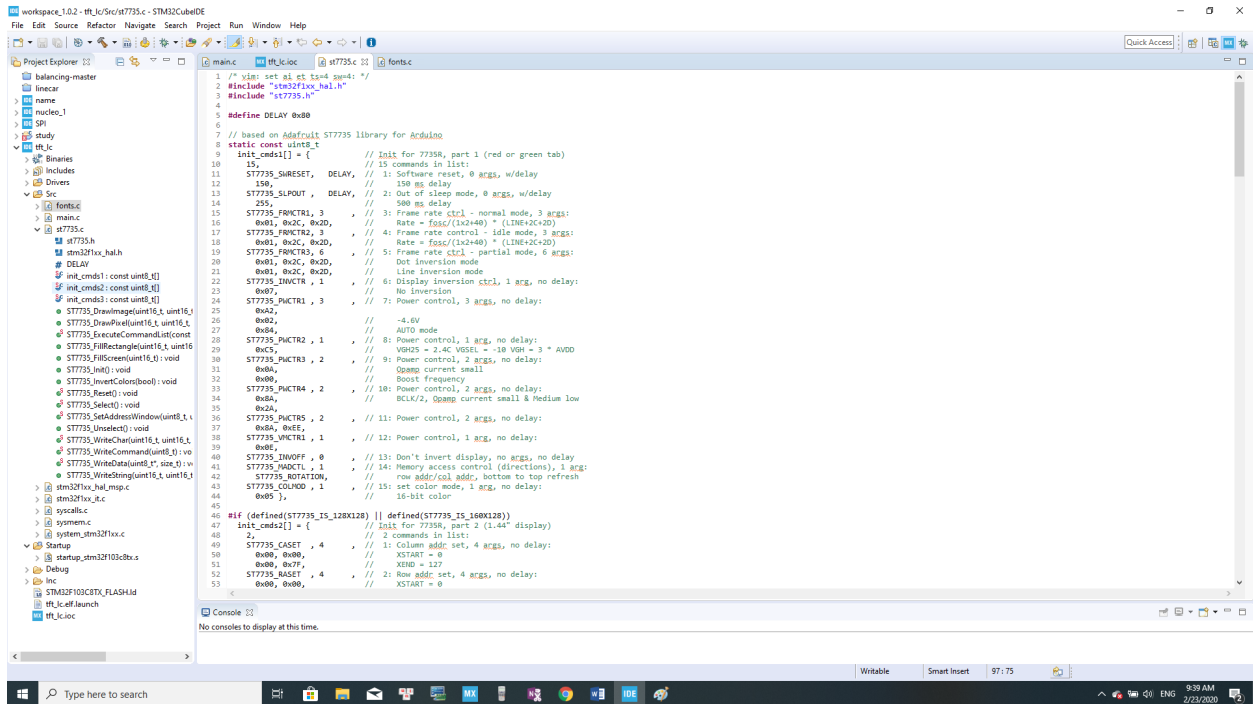
Thư viện font.c

```

1  /* vim: set ai et ts4 sw4: */
2  #include "font.h"
3
4  static const uint16_t Font7x10[] = {
5      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 0
6      0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 1
7      0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 2
8      0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 3
9      0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 4
10     0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 5
11     0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 6
12     0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 7
13     0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 8
14     0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 9
15     0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // A
16     0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // B
17     0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // C
18     0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // D
19     0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // E
20     0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // F
21     0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 10
22     0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 11
23     0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 12
24     0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 13
25     0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 14
26     0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 15
27     0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 16
28     0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 17
29     0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 18
30     0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 19
31     0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 20
32     0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 21
33     0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 22
34     0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 23
35     0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 24
36     0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 25
37     0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 26
38     0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 27
39     0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 28
40     0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 29
41     0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 30
42     0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 31
43     0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 32
44     0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 33
45     0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 34
46     0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 35
47     0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 36
48     0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 37
49     0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 38
50     0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 39
51     0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 40
52     0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 41
53     0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 42
54     0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 43
55     0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 44
56     0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 45
57     0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 46
58     0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 47
59     0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 48
60     0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 49
61     0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 50
62     0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 51
63     0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 52
64     0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 53
65     0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 54
66     0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 55
67     0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 56
68     0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 57
69     0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 58
70     0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 59
71     0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 60
72     0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 61
73     0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 62
74     0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 63
75     0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 64
76     0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 65
77     0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 66
78     0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 67
79     0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 68
80     0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 69
81     0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 70
82     0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 71
83     0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 72
84     0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 73
85     0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 74
86     0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 75
87     0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 76
88     0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 77
89     0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 78
90     0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 79
91     0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 80
92     0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 81
93     0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 82
94     0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 83
95     0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 84
96     0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 85
97     0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 86
98     0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 87
99     0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 88
100    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 89
101    0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 90
102    0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 91
103    0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 92
104    0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 93
105    0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 94
106    0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 95
107    0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 96
108    0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 97
109    0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 98
110    0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 99
111    0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 100
112    0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 101
113    0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 102
114    0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 103
115    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 104
116    0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 105
117    0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 106
118    0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 107
119    0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 108
120    0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 109
121    0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 110
122    0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 111
123    0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 112
124    0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 113
125    0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 114
126    0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 115
127    0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 116
128    0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 117
129    0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 118
130    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 119
131    0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 120
132    0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 121
133    0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 122
134    0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 123
135    0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 124
136    0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 125
137    0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 126
138    0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 127
139    0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 128
140    0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 129
141    0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 130
142    0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 131
143    0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 132
144    0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 133
145    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 134
146    0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 135
147    0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 136
148    0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 137
149    0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 138
150    0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 139
151    0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 140
152    0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 141
153    0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, 0xD000, // 142
154    0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, 0xE000, // 143
155    0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, 0xF000, // 144
156    0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // 145
157    0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, // 146
158    0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, // 147
159    0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, // 148
160    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, // 149
161    0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, // 150
162    0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, 0x7000, // 151
163    0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 152
164    0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, 0x9000, // 153
165    0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, 0xA000, // 154
166    0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, 0xB000, // 155
167    0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, 0xC000, // 156
168    0xD000, 0xD00
```


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



```

1  #!/usr/bin/env python
2
3  # vim: set ai et ts=4 sw=4:
4
5  import cv2
6  import sys
7  import os
8  import numpy as np
9
10 img = cv2.imread("pif.png")
11 img = cv2.resize(img, (128,128))
12 img = cv2.flip( img, 1 )
13 img = cv2.flip( img, 0 )
14 # img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
15
16 for y in range(0, img.shape[0]):
17     s = "{"
18     for x in range(0, img.shape[1]):
19         (b, g, r) = img[x, y,:]
20         color565 = ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >> 3)
21         # for right endiness, so ST7735_DrawImage would work
22         color565 = ((color565 & 0xFF00) >> 8) | ((color565 & 0xFF) << 8)
23         s += "0x{:04X}, ".format(color565)
24     s += "}, "
25     print(s)
26
27 print("};")
28 cv2.imshow('image',img)
29 k = cv2.waitKey(0)
30 if k == 27:          # wait for ESC key to exit
31     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).

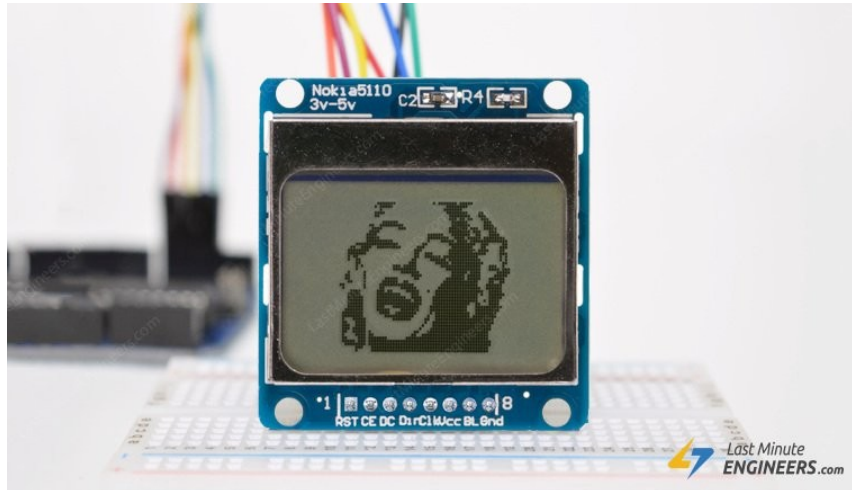
*Bài tập: Hiện thị hình người yêu lên màn hình :))))))

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 ảnh có một chương trình mẫu là một game nhỏ trên màn hình Nokia5110:

Màn Hình Nokia 5110



Màn hình
Nokia5110 sử
dụng chip driver
của Philips,

PCD8544

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

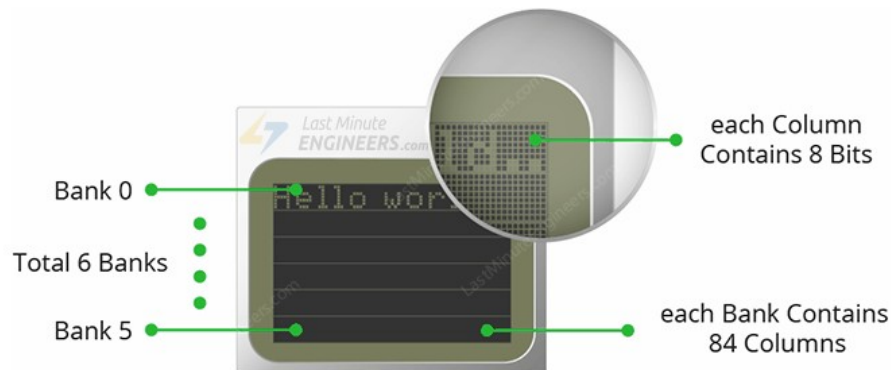
Single chip LCD controller/driver

- 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).

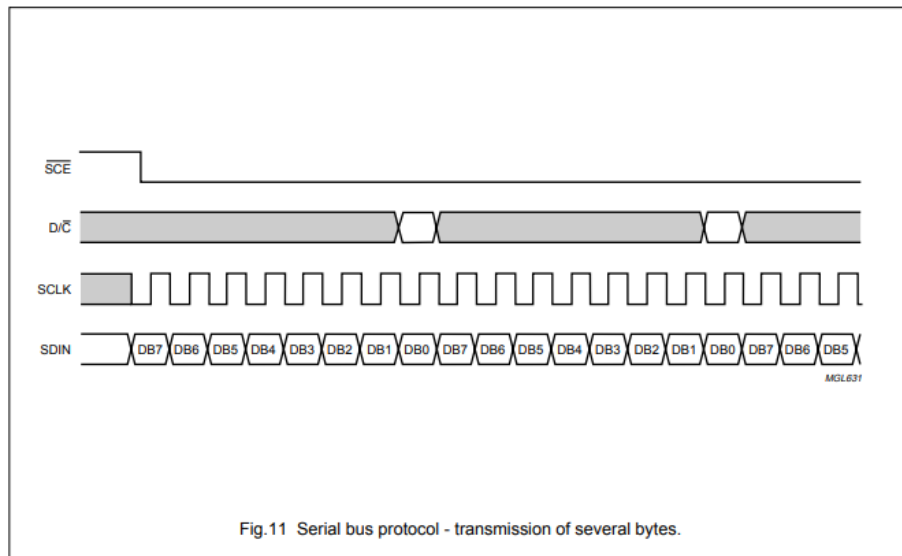
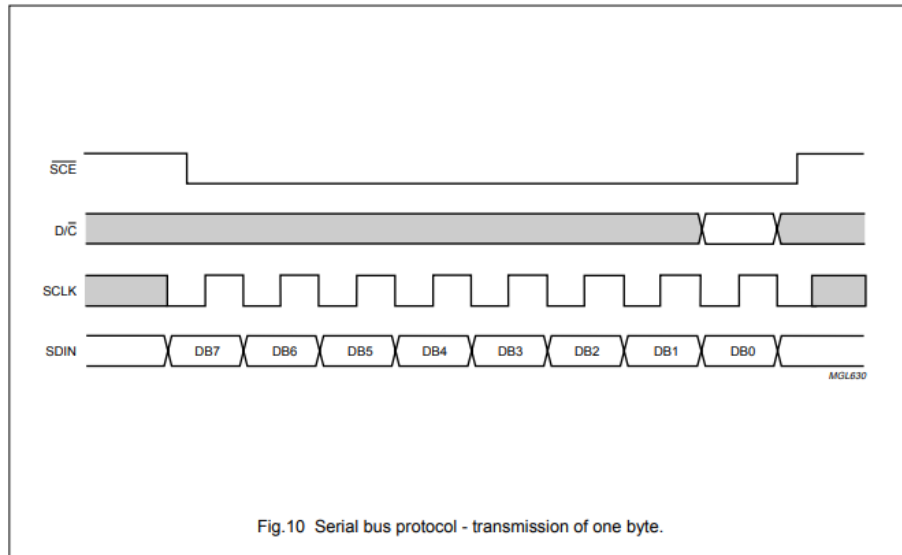
(Trích datasheet của IC).

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.



Cấu trúc frame truyền của IC driver.

48 × 84 pixels matrix LCD controller/driver

PCD8544

Table 1 Instruction set

INSTRUCTION	D/ \bar{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 ≤ Y ≤ 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 ≤ X ≤ 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	display blank 00 normal mode 10 all display segments on 01 inverse video mode 11	
TC ₁ and TC ₀	V _{LCD} temperature coefficient 0 00 V _{LCD} temperature coefficient 1 01 V _{LCD} temperature coefficient 2 10 V _{LCD} temperature coefficient 3 11	

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:

- Để bắt đầu quá trình truyền, chân CE của IC được kéo xuống thấp.
- Quá trình truyền data được thực hiện khi có cạnh lên của tín hiệu chân CLK, IC sẽ đọc data ở chân DIN
- Một frame truyền có độ dài 8bit (1byte) được ghi vào thanh ghi dịch của chip để đọc 1byte 1 lần.

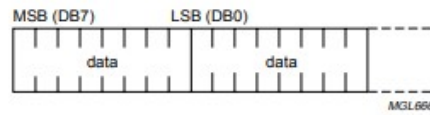


Fig.8 General format of data stream.

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

*Hàm truyền dữ liệu:

```

40 /*
41  * @brief Send information to the LCD using configured GPIOs
42  * @param val: value to be sent
43  */
44 void LCD_send(uint8_t val){
45     uint8_t i;
46     for(i = 0; i < 8; i++){
47         HAL_GPIO_WritePin(lcd_gpio.DINPORT, lcd_gpio.DINPIN, (val & (1 << (7 - i))));
48         HAL_GPIO_WritePin(lcd_gpio.CLKPORT, lcd_gpio.CLKPIN, GPIO_PIN_SET);
49         HAL_GPIO_WritePin(lcd_gpio.CLKPORT, lcd_gpio.CLKPIN, GPIO_PIN_RESET);
50     }
51 }
52 /*
53  * @brief Writes some data into the LCD
54  * @param data: data to be written
55  * @param mode: command or data
56  */
57 void LCD_write(uint8_t data, uint8_t mode){
58     if(mode == LCD_COMMAND){
59         HAL_GPIO_WritePin(lcd_gpio.DCPORT, lcd_gpio.DCPIN, GPIO_PIN_RESET);
60         HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_RESET);
61         LCD_send(data);
62         HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_SET);
63     }
64     else{
65         HAL_GPIO_WritePin(lcd_gpio.DCPORT, lcd_gpio.DCPIN, GPIO_PIN_SET);
66         HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_RESET);
67         LCD_send(data);
68         HAL_GPIO_WritePin(lcd_gpio.CEPORT, lcd_gpio.CEPIN, GPIO_PIN_SET);
69     }
70 }

```


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 coefficient.
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 định 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.

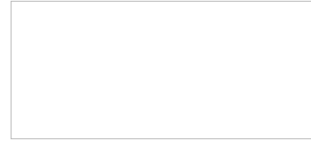
This tool also works offline. Simply save this page to your computer and open the file in your browser.

1. Select image

or

1. Paste byte array

Choose Files pusheen-copy.jpg



128 x 64 px

Read as horizontal Read as vertical

2. Image Settings

Canvas size(s):

pusheen-copy.jpg (file resolution: 600 x 350)
600 x 350 glyph remove

Background color:

☒ White ☐ Black ☐ Transparent

Invert image colors

Brightness / alpha threshold:

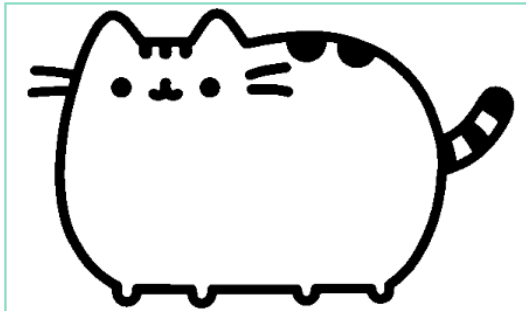
128

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

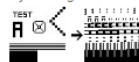
Code output format

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.



Generate code

```
// '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, 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, 0x06, 0x01};
char bird_down[] = {0x00, 0x10, 0x88, 0x84, 0x08, 0x8c, 0xe4, 0x04, 0x44, 0xb4, 0x18, 0x08, 0x90, 0x60, 0x00, 0x00,
                    0x00, 0x00, 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, 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, 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, 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, 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, 0x01, 0x07, 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 bảng thanh ghi. Để truy cập đến pixel có tọa độ (x,y), trong đó x có kích thước 84, y là số kí hiệu bảng thanh ghi, gồm 1 -> 6, mỗi bảng 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 $1 \ll (y\%8)$ 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:

```

148 void printImage(uint8_t x, uint8_t y, char data [], uint8_t sx, uint8_t sy){
149 //   for(int i = 0; i<504; i++){
150 //       buffer[i] = 0x00;
151       for(int i = 0 ; i<sx;i++){
152           for(int j = 0; j<sy; j++){
153               for(int k = 0; k<8;k++){
154                   LCD_set_pixel_im(j+x , y + k + 8*(sx -1 - i) , data[i*sy+j] & (1 << (7 - k)));
155               }
156           }
157       }
158 }

```

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.

```

79 void draw_bar(uint8_t x, uint8_t y, uint8_t height, uint8_t width, uint8_t gap){
80
81     for(int i = x; i<x+width; i++){
82         for(int j = y; j<y+height-6; j++){
83             LCD_set_pixel(i,j, 1);
84         }
85     }
86     for(int i = x; i<x+width; i++){
87         for(int j = 48; j> (y+height)+ gap + 6; j--){
88             LCD_set_pixel(i,j, 1);
89         }
90     }
91
92     if(x<82 && x>2){
93         for(int i = x-2; i<x+width +2 ; i++){
94             for(int j = y+height-5; j<y+height; j++){
95                 LCD_set_pixel(i,j, 1);
96             }
97         }
98         for(int i = x-2; i<x+width +2 ; i++){
99             for(int j = (y+height)+ gap + 5; j> (y+height)+ gap; j--){
100                 LCD_set_pixel(i,j, 1);
101             }
102         }
103
104 //     for(int i = x-2; i<x+width +2 ; i++){
105 //         for(int j = 48; j>(y+height)- gap ; j++){
106 //             LCD_set_pixel(i,j, 1);
107 //         }
108 //     }
109 }
110 // LCD_printBuffer(buffer);
111 }

```

3. Điều Khiển Nhân Vật:

The diagram illustrates the internal architecture of the Interrupt Controller (IC). At the top, the **AMBA APBbus** is connected to the **Peripheral interface** via a bidirectional arrow. The **PCLK2** clock signal is input to the **Peripheral interface**. The **Peripheral interface** is connected to five registers, each via a 19-bit bus:

- Pending request register**: Its output is connected to the **To NVIC interrupt controller** via a 19-bit bus.
- Interrupt mask register**: Its output is connected to an AND gate along with the output of the **Pending request register**.
- Software interrupt event register**: Its output is connected to an OR gate along with the output of the AND gate.
- Rising trigger selection register**: Its output is connected to the **Edge detect circuit** via a 19-bit bus.
- Falling trigger selection register**: Its output is connected to the **Edge detect circuit** via a 19-bit bus.

The **Edge detect circuit** is connected to the **Input Line**. The **Event mask register** is connected to the AND gate and the OR gate via 19-bit buses. The **Pulse generator** is connected to the AND gate and the OR gate via 19-bit buses. The output of the OR gate is connected to the **Edge detect circuit** via a 19-bit bus.

Configuration

☐ Group By Peripherals

GPIO

RCC

SYS

NVIC

Search Signals

☐ Show only Modified Pins

Pin Name	Signal on Pin	GPIO output...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA0-WKUP	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA1	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA2	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA3	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA5	n/a	Low	Output Pus...	No pull-up ...	High		<input checked="" type="checkbox"/>
PA7	n/a	Low	Output Pus...	No pull-up ...	High		<input checked="" type="checkbox"/>
PB0	n/a	Low	Output Pus...	No pull-up ...	Low	CE	<input checked="" type="checkbox"/>
PB1	n/a	Low	Output Pus...	No pull-up ...	Low	RST	<input checked="" type="checkbox"/>
PB14	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PB15	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PC4	n/a	Low	Output Pus...	No pull-up ...	Low	DC	<input checked="" type="checkbox"/>
PC5	n/a	Low	Output Pus...	No pull-up ...	Low	LED	<input checked="" type="checkbox"/>
PC6	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>
PC7	n/a	n/a	External Int...	Pull-up	n/a		<input checked="" type="checkbox"/>

PA1 Configuration :

GPIO mode

External Interrupt Mode with Rising edge trigger detection

GPIO Pull-up/Pull-down

Pull-up

User Label

Bảng ưu tiên ngắt NVIC như sau:

NVIC Mode and Configuration

Configuration

Priority Group: 4 bits for pre-emption priority 0 bits for sub-priority ☐ Sort by Preemption Priority and Sub Priority

Search: ☐ Show only enabled interrupts ☒ Force DMA channels interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
RTC global interrupt	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
EXTI line0 interrupt	<input type="checkbox"/>	0	0
EXTI line1 interrupt	<input checked="" type="checkbox"/>	0	0
EXTI line2 interrupt	<input checked="" type="checkbox"/>	0	0
EXTI line3 interrupt	<input checked="" type="checkbox"/>	0	0
EXTI line[9:5] interrupts	<input checked="" type="checkbox"/>	0	0
TIM4 global interrupt	<input type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	0	0

☐ Enabled Preemption Priority Sub Priority

```

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>