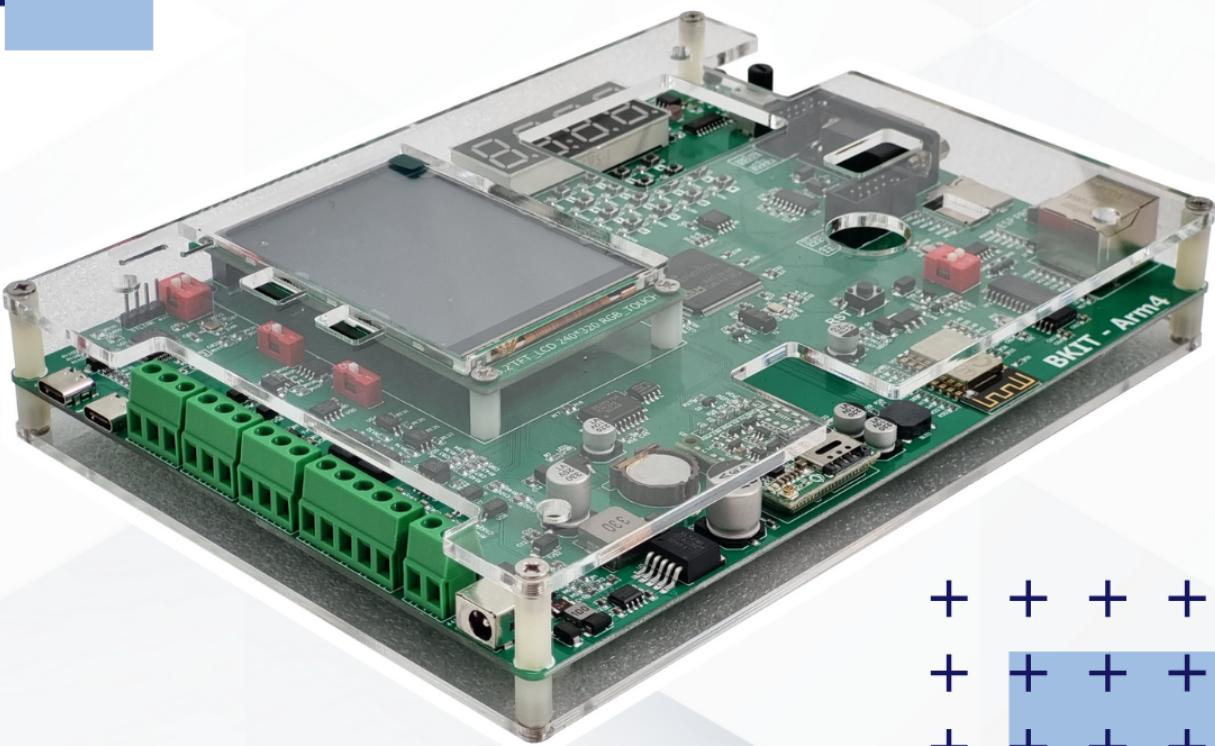


KIT THÍ NGHIỆM BKIT

ARM4



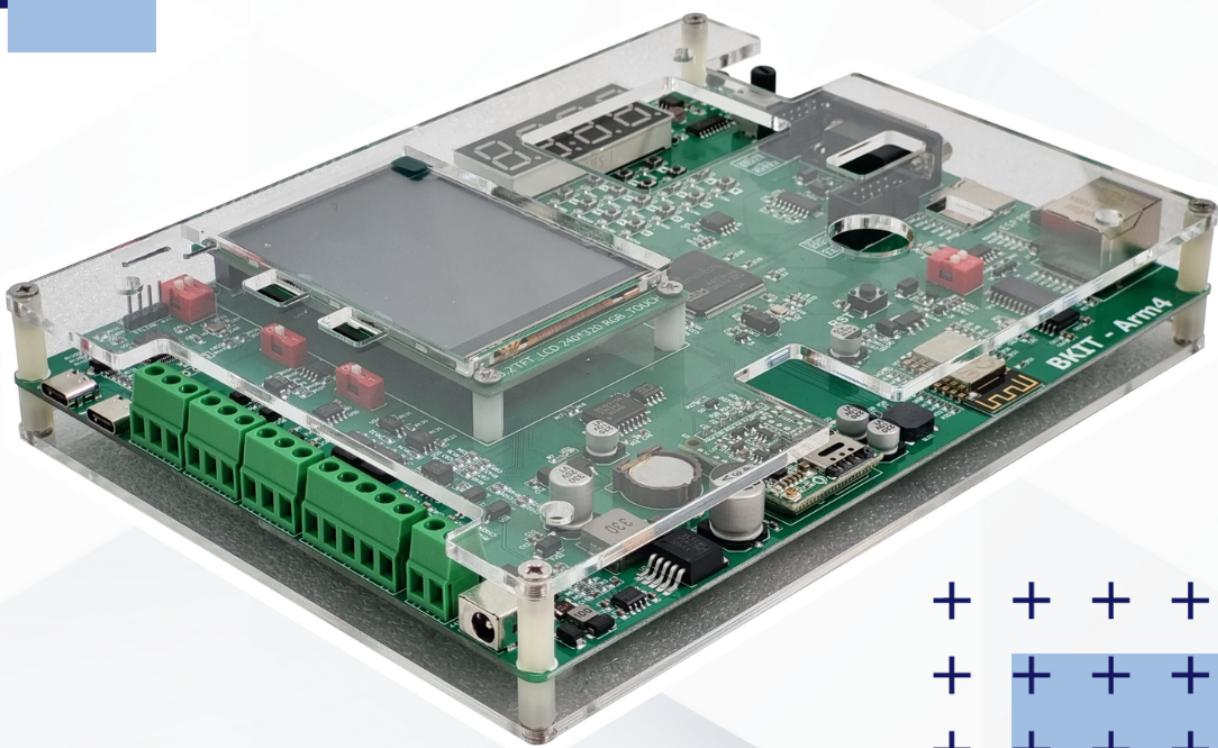
Mục lục

| | |
|--|-----------|
| Chapter 1. General Purpose Input Output | 4 |
| 1 Mục tiêu | 5 |
| 2 Giới thiệu | 5 |
| 3 Project đầu tiên trên STM32Cube | 6 |
| 4 Hướng dẫn config | 10 |
| 5 Bài tập | 15 |
| 5.1 Bài tập 1 | 15 |
| 5.2 Bài tập 2 | 15 |
| 5.3 Bài tập 3 | 15 |
| | |
| Chapter 2. Timer Interrupt and LED Scanning | 16 |
| 1 Mục tiêu | 17 |
| 2 Giới thiệu | 17 |
| 3 Cấu hình timer | 18 |
| 4 Cấu hình LED đồng hồ | 20 |
| 5 Hướng dẫn lập trình | 24 |
| 6 Exercise and Report | 31 |
| 6.1 Exercise 1 | 31 |
| 6.2 Bài tập 1 | 31 |
| 6.3 Bài tập 2 | 31 |
| 6.4 Bài tập 3 | 31 |
| 6.5 Bài tập 4 | 32 |

| | |
|---|-----------|
| Chapter 3. LCD and Button matrix | 33 |
| 1 Mục tiêu | 34 |
| 2 Giới thiệu | 34 |
| 3 Cơ sở lý thuyết | 35 |
| 3.1 Sự cần thiết của điện trở kéo lên của nút nhấn | 35 |
| 3.2 Chống rung nút nhấn | 36 |
| 4 Hướng dẫn cấu hình | 38 |
| 4.1 Cấu hình nút nhấn | 38 |
| 4.2 Cấu hình LCD | 40 |
| 5 Hướng dẫn lập trình | 43 |
| 5.1 Sử dụng thư viện: button.h | 43 |
| 5.2 Sử dụng thư viện: lcd.h | 46 |
| 5.3 Lập trình máy trạng thái | 59 |
| 6 Bài tập và báo cáo | 63 |
| 6.1 Bài tập 1 | 63 |
| 6.2 Bài tập 2 | 63 |

CHƯƠNG 1

General Purpose Input Output



1 Mục tiêu

- Biết cách sử dụng phần mềm STM32CubeIDE để xây dựng ứng dụng trên vi điều khiển.
- Biết cách config và lập trình các chân GPIO.
- Biết cách điều khiển các ngoại vi liên quan đến GPIO trên kit thí nghiệm.

2 Giới thiệu

Trong hướng dẫn này, STM32CubeIDE được sử dụng làm trình soạn thảo để lập trình vi điều khiển ARM. STM32CubeIDE là platform nổi bật với các tính năng config ngoại vi, sinh code, biên dịch và gõ lỗi cho vi điều khiển và vi xử lý STM32.



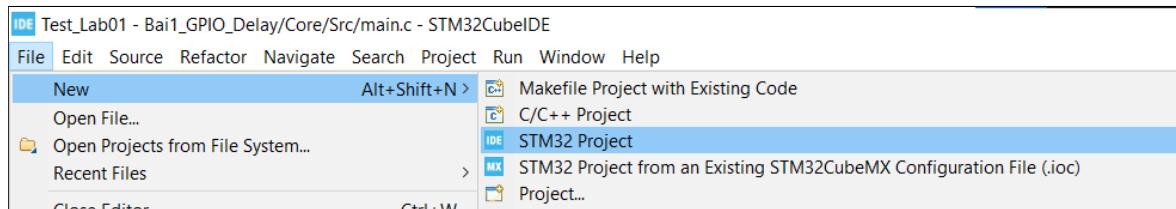
Hình 1.1: Phần mềm STM32Cube IDE

Điều thú vị nhất của STM32CubeIDE là sau khi chọn một MPU hoặc MCU STM32, hay một bộ vi điều khiển, bộ vi xử lý được config sẵn từ việc chọn bo mạch, mã khởi tạo sẽ được tạo tự động. Bất cứ khi nào trong quá trình lập trình, người dùng đều có thể quay lại quá trình khởi tạo và thay đổi config của các thiết bị ngoại vi. Việc config này không ảnh hưởng đến phần code đã được người lập trình viết trước đó. Tính năng này có thể đơn giản hóa quá trình khởi tạo các ngoại vi và dễ dàng phát triển các ứng dụng trên vi điều khiển STM32.

Phần còn lại của hướng dẫn này bao gồm tạo một project trên STM32Cube IDE, chạy kiểm thử trên mạch thực. Sau đó, sinh viên sẽ hoàn thành một số bài tập để hiểu thêm.

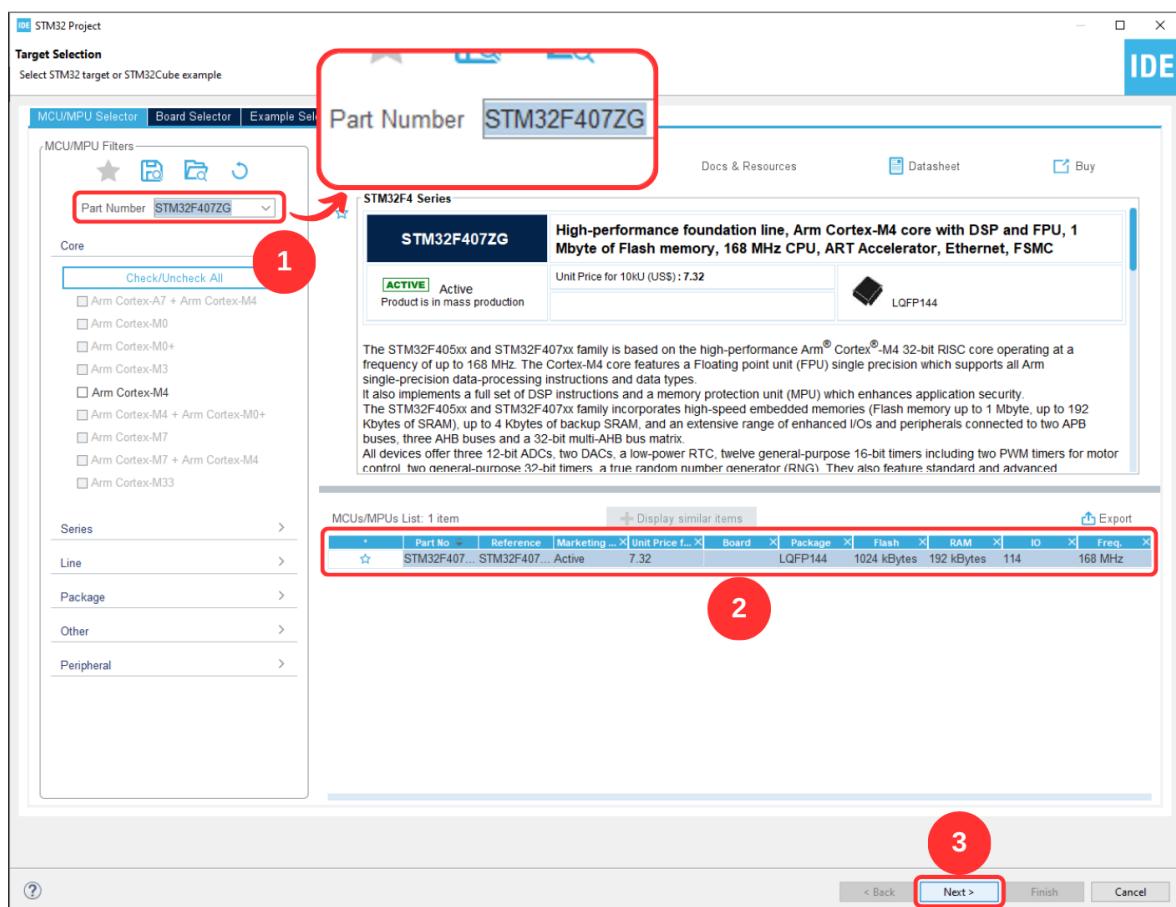
3 Project đầu tiên trên STM32Cube

Bước 1: Chạy phần mềm STM32CubeIDE, chọn menu **File**, chọn **New**, sau đó chọn **STM32 Project**. STM32CubeIDE sẽ cần tải xuống một số packages, thường tồn một ít thời gian trong lần đầu tiên tạo một dự án mới.



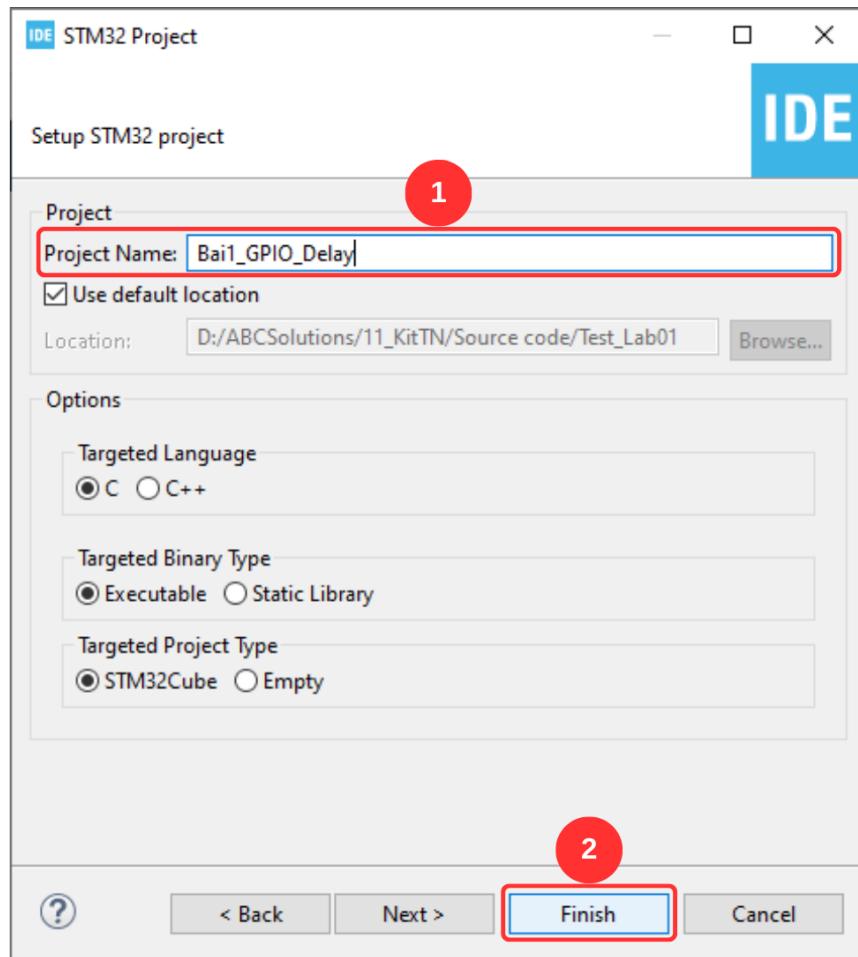
Hình 1.2: Tạo một project mới trên STM32CubeIDE

Bước 2: Tìm chip STM32F407ZG. Để dễ dàng tìm, chúng ta nhập tên vi điều khiển trong thanh tìm kiếm **Part Number**. Sau đó, chọn chip tại phần **MCUs/MPUs List** và chọn next để tiếp tục.



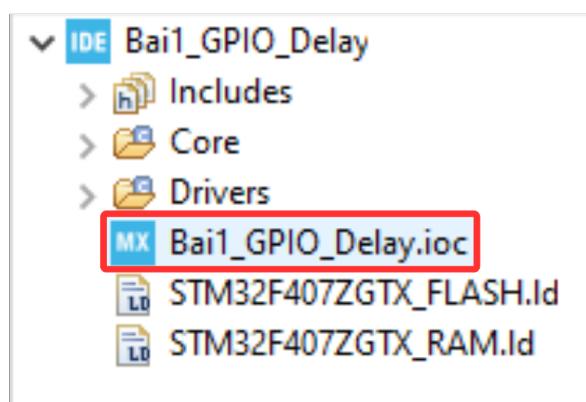
Hình 1.3: Tìm kiếm chip

Bước 3: Đặt tên project trong **Project Name** và đường dẫn lưu project trong **Location**. Lưu ý: Đường dẫn không được chứa ký tự tiếng Việt và ký tự đặc biệt (ví dụ như space).



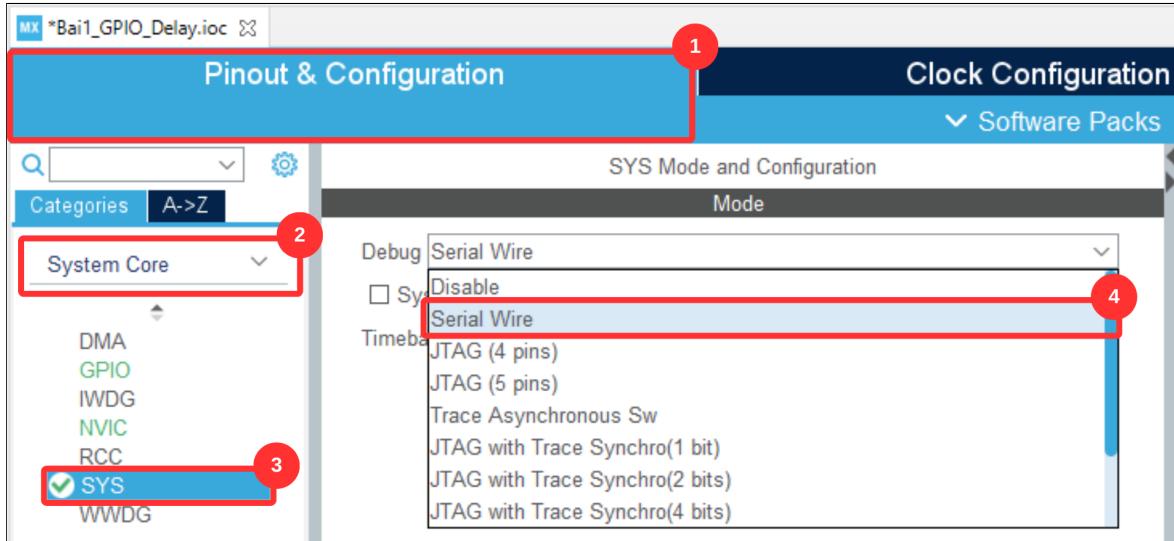
Hình 1.4: Đặt tên và tìm vị trí lưu project

Bước 4: Sau khi tạo xong project, màn hình config được hiển thị (file .ioc). Tính năng này của CubeIDE có thể đơn giản hóa quy trình config cho bộ vi điều khiển ARM như STM32. Trường hợp file chưa được mở, ta có thể mở file trong project.



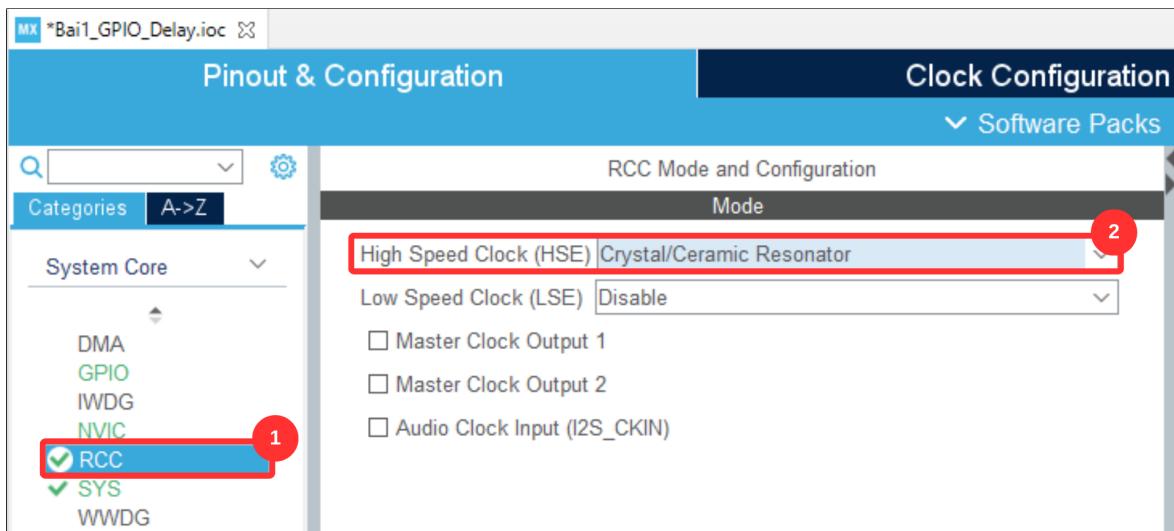
Hình 1.5: File config các chân của vi điều khiển

Bước 5: Điều chỉnh phương thức gõ lỗi:



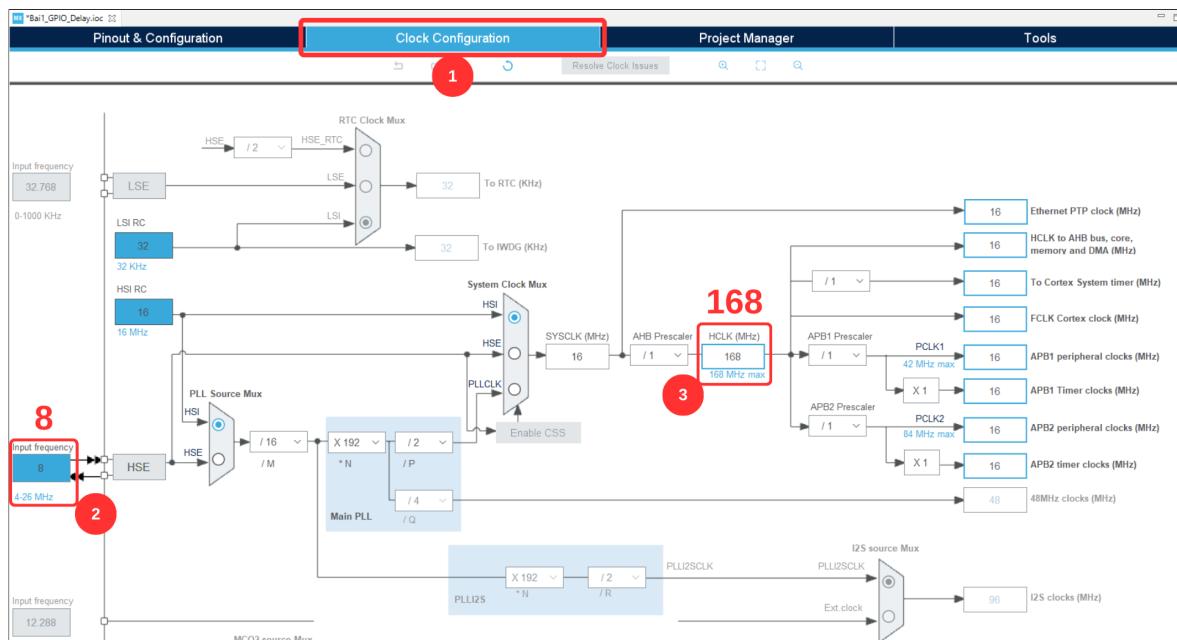
Hình 1.6: Chọn phương thức gõ lỗi

Bước 6: Thạch anh ngoài giúp tối ưu hóa tần số hoạt động của vi điều khiển. Để sử dụng thạch anh ngoài, chúng ta sẽ cần config trong file .ioc như hình 1.7.



Hình 1.7: Config thạch anh ngoài

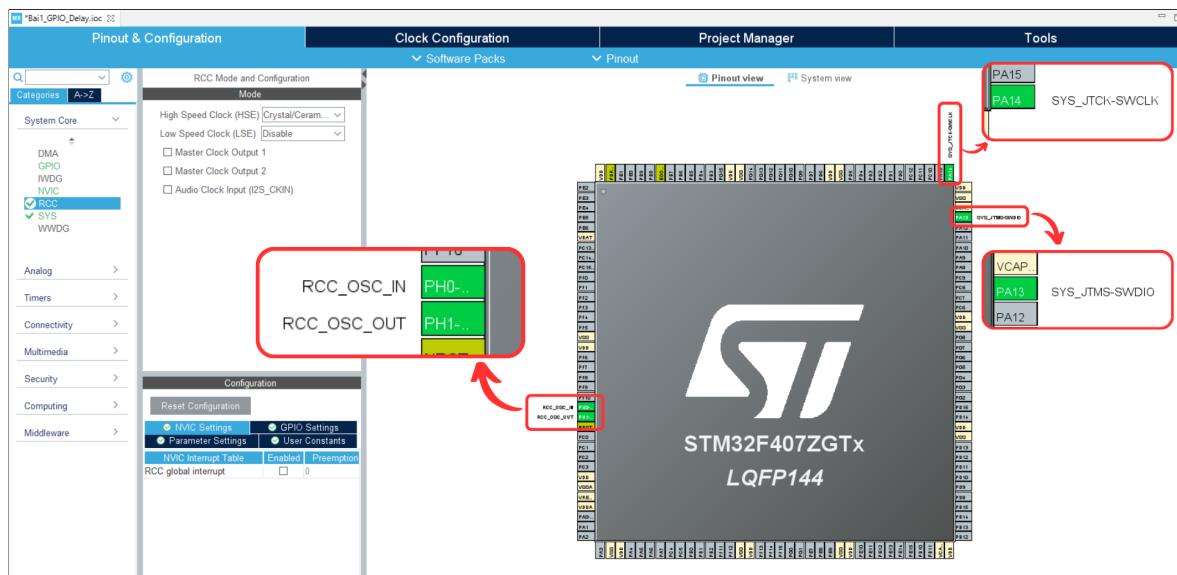
Chỉnh tần số trong cửa sổ **Clock Configuration**. Vì kit thí nghiệm sử dụng thạch anh ngoài có thông số **8MHz** và chúng ta muốn điều chỉnh tối đa tần số hoạt động của kit thí nghiệm (**168MHz**) nên ta sẽ điều chỉnh các thông số như hình 1.8.



Hình 1.8: Config thạch anh ngoài

Chọn OK và đợi tính toán:

Sau khi config xong thì có 4 chân được config như hình 1.9. Sau đó, bấm tổ hợp phím **Ctrl + S** để phần mềm sinh code.



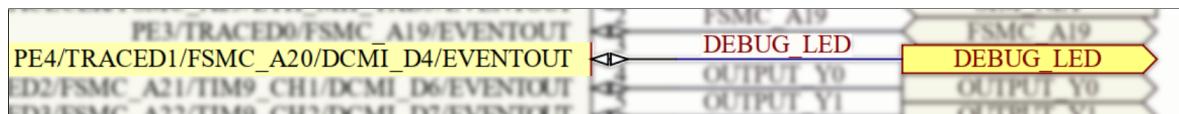
Hình 1.9: Vị điều khiển sau khi config

4 Hướng dẫn config

Trong nội dung này, ta sẽ tiến hành kiểm thử trên kit thí nghiệm bằng cách điều khiển LED3 (LED DEBUG). Theo như schematic, LED3 sẽ được điều khiển bằng chân PE4 của MCU và sẽ tích cực mức cao (LED sẽ sáng khi đầu ra của vi điều khiển ở mức logic 1).



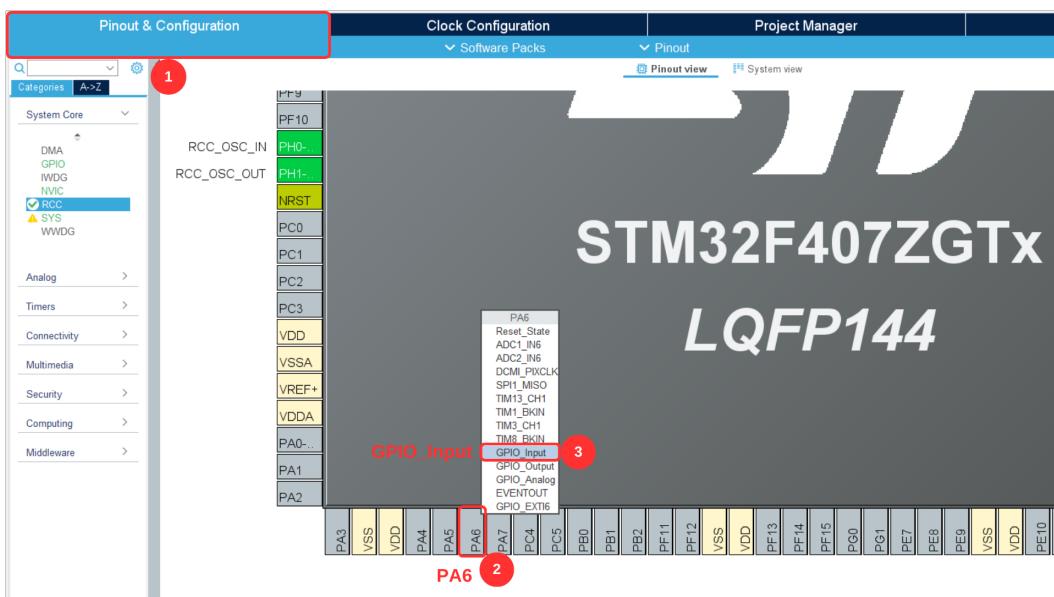
Hình 1.10: Sơ đồ nguyên lý của LED3



Hình 1.11: MCU điều khiển LED3 bằng chân PE4

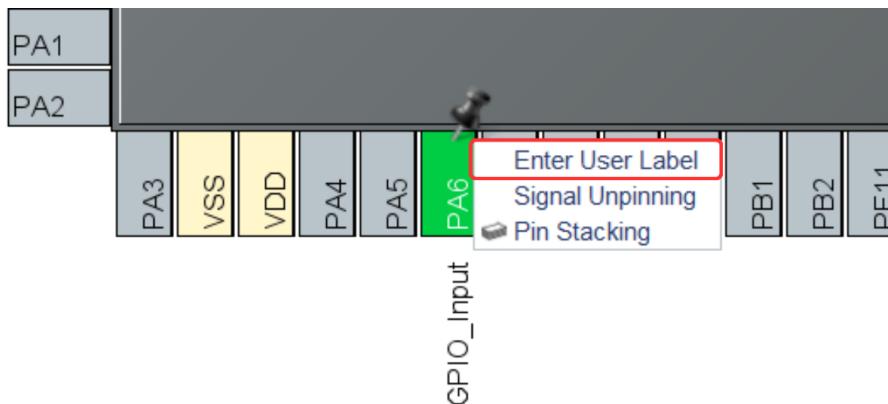
Ngoài ra, ta cũng có thể luyện tập config trước các chân ở khối input X0->X3 và khối output Y0,Y1. Lưu ý: Các khối input X và output Y trên kit thí nghiệm được thiết kế thông qua opto, dùng để tương tác với các thiết bị công suất lớn. Tương ứng với mỗi input và output nói trên, kit thí nghiệm sẽ có một đèn LED.

Để config các chân của vi điều khiển, ta mở file cấu hình (**.ioc**) và chọn vào cửa sổ **Pinout & Configuration**. Tiếp đó nhấp chọn chuột trái vào chân muốn config. Để config input, ta chọn **GPIO_Input**.

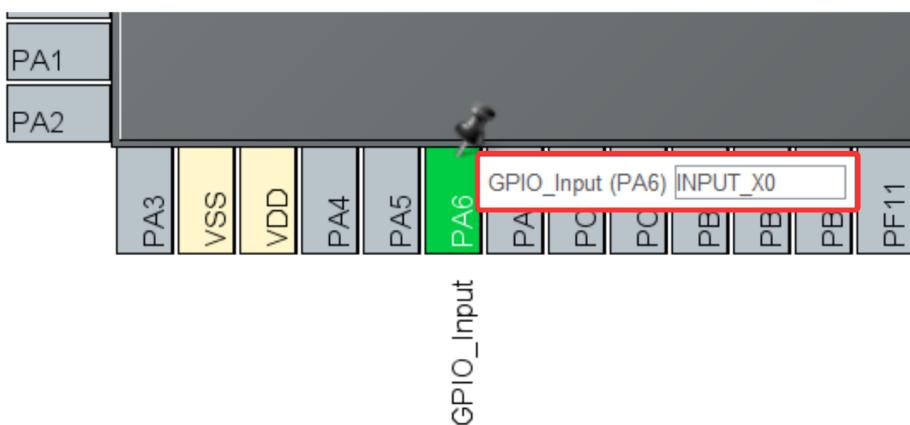


Hình 1.12: Config PA6 thành chân input

Tiếp theo, ta có thể đặt tên mới cho các chân này để thuận tiện cho việc lập trình. Đặt tên mới bằng cách nhấp chuột phải vào chân muốn đặt tên. Sau đó chọn Enter User Label và nhập tên mới.

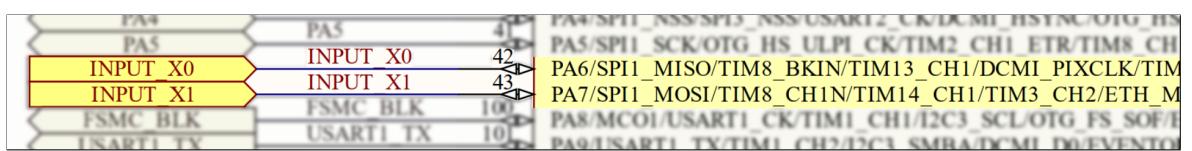


Hình 1.13: Nhấp chuột phải vào chân PA6 và chọn Enter User Label

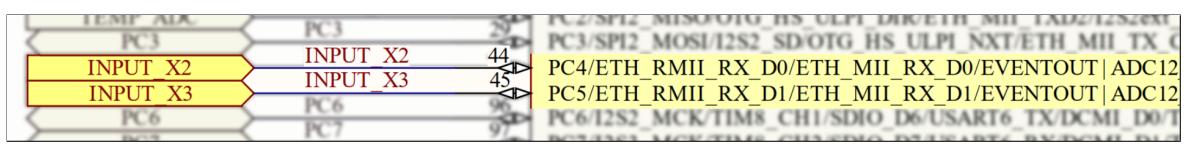


Hình 1.14: Nhập tên mới cho chân input PA6 là INPUT_X0

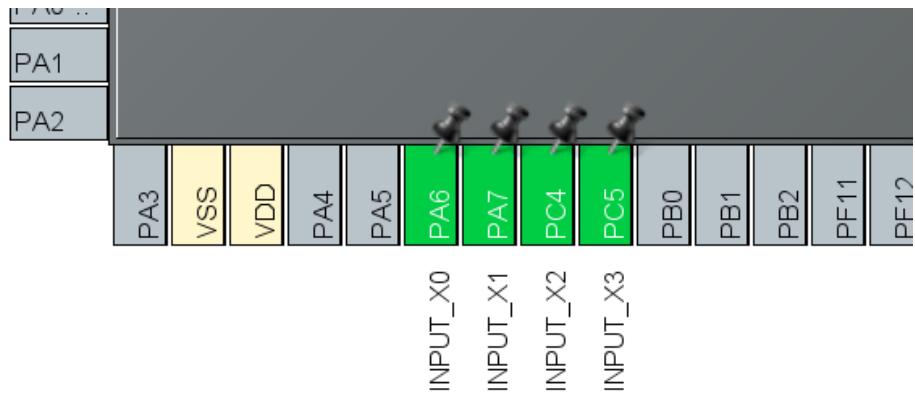
Thực hiện tương tự với các chân input còn lại.



Hình 1.15: Sơ đồ nguyên lý của các chân input X0 và X1

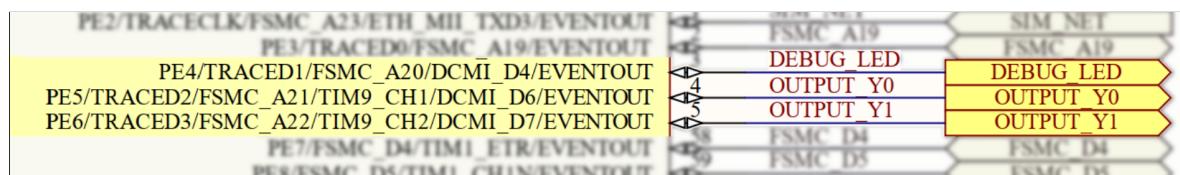


Hình 1.16: Sơ đồ nguyên lý của các chân input X2 và X3

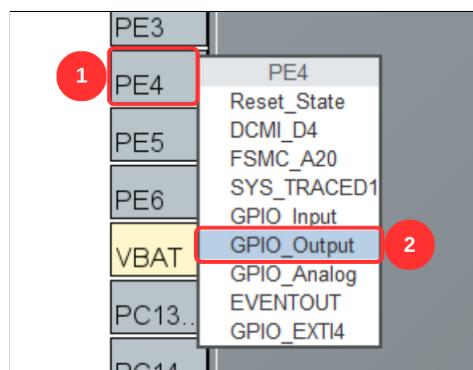


Hình 1.17: Các chân input sau khi được config và đặt lại tên

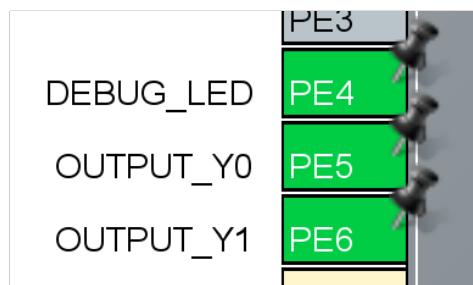
Tương tự như vậy, ta có thể config các chân output.



Hình 1.18: Sơ đồ nguyên lý của các chân output DEBUG_LED, Y0 và Y1



Hình 1.19: Config PE4 thành chân output



Hình 1.20: Các chân output sau khi được config và đặt lại tên

Sau đó ta lưu file config lại để phần mềm sinh code. Sau khi sinh code, ta được file **main.c** như sau:

```
1 int main(void)
2 {
3     /* USER CODE BEGIN 1 */
4
5     /* USER CODE END 1 */
6
7     /* MCU configuration
8     -----
9     */
10
11    /* Reset of all peripherals, Initializes the Flash
12       interface and the Systick. */
13    HAL_Init();
14
15
16    /* USER CODE BEGIN Init */
17
18
19    /* USER CODE END Init */
20
21
22    /* configure the system clock */
23    SystemClock_Config();
24
25
26    /* USER CODE BEGIN SysInit */
27
28
29    /* Initialize all configured peripherals */
30    MX_GPIO_Init();
31
32    /* USER CODE BEGIN 2 */
33
34    /* USER CODE END 2 */
35
36
37    /* Infinite loop */
38    /* USER CODE BEGIN WHILE */
39
40
41    while (1)
42    {
43        HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin);
44    }
45}
```

```

35     HAL_Delay(1000);
36     /* USER CODE END WHILE */
37
38     /* USER CODE BEGIN 3 */
39 }
40 /* USER CODE END 3 */
41 }
```

Program 1.1: Project chớp tắt led đầu tiên

Dòng 34 và 35 là các dòng ta chủ động thêm vào. Lưu ý khi ta lập trình cần phải để mã nguồn vào đúng các vị trí được cho phép, nếu không mã nguồn sẽ mất khi chỉnh sửa config và sinh lại code. Khi lập trình nên tập thói quen sử dụng phím tắt **Ctrl + Space** để gợi ý mã nguồn.

Đoạn mã trên là chương trình mẫu chớp tắt LED mỗi 1 giây. Bước tiếp theo là **Build** chương trình, bước này sẽ giúp kiểm tra các lỗi về mặt cú pháp. Sau đó, để nạp chương trình vào kit thí nghiệm, ta chọn **Run**. Lưu ý: SW1 trên kit thí nghiệm phải ở trạng thái ON để nạp code.



Hình 1.21: Build và Run chương trình

Bên cạnh sử dụng câu lệnh **Toggle**, ta còn có thể sử dụng câu lệnh **Set** và **Reset**.

```

1 while (1){
2     HAL_GPIO_WritePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin, 1);
3     HAL_Delay(1000);
4     HAL_GPIO_WritePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin, 0);
5     HAL_Delay(1000);
6 }
```

Program 1.2: Ví dụ chương trình chớp tắt LED

Sau khi nạp chương trình, ta được kết quả LED3 trên kit thí nghiệm sẽ chớp tắt mỗi 1 giây. Đoạn code trên có thể được chỉnh sửa để điều khiển output Y0, Y1, hoặc có thể tham khảo project mẫu **Bai1_GPIO_Delay**.

5 Bài tập

5.1 Bài tập 1

Viết chương trình điều khiển LED3 sáng trong 2 giây, sau đó tắt trong 4 giây, quá trình này được lặp lại mãi mãi.

5.2 Bài tập 2

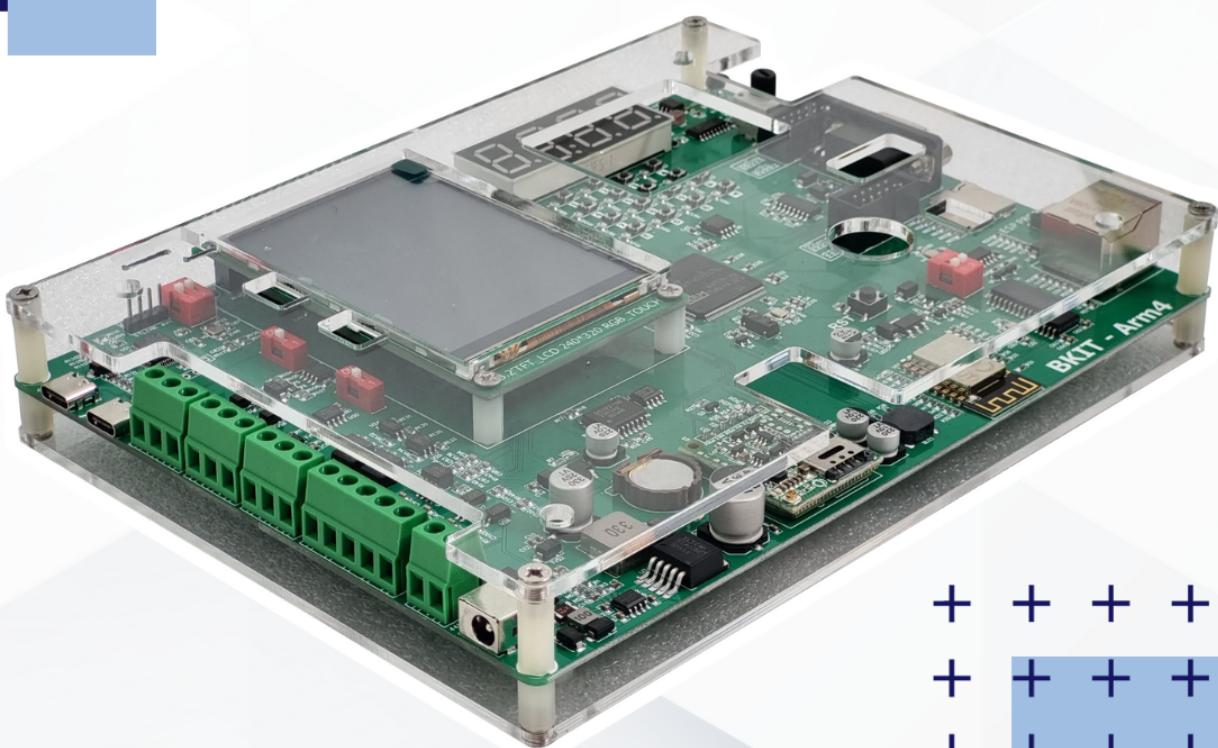
Lập trình lại chương trình ở Bài tập 1 nhưng chỉ sử dụng một câu lệnh delay duy nhất ở cuối vòng lặp while. Gợi ý: sử dụng biến đếm và biến để lưu trạng thái của đèn.

5.3 Bài tập 3

Sử dụng LED3 và các LED ở output Y0, Y1 để mô phỏng tín hiệu đèn giao thông. Giả sử mỗi LED đại diện cho một tín hiệu trên đèn giao thông với chu kỳ đèn đỏ là 5 giây, đèn xanh là 3 giây, đèn vàng là 1 giây. Chỉ sử dụng duy nhất 1 câu lệnh delay.

CHƯƠNG 2

Timer Interrupt and LED Scanning

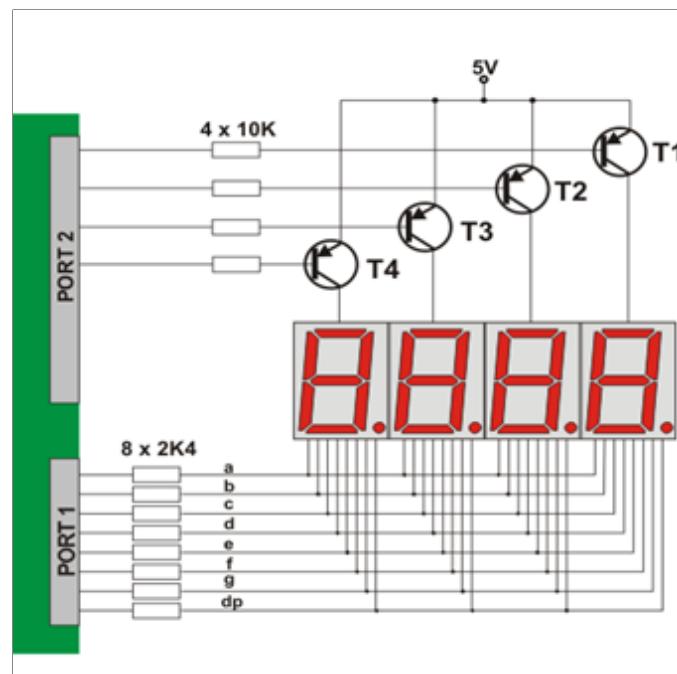


1 Mục tiêu

- Biết cách cấu hình và sử dụng timer trên kit thí nghiệm.
- Biết cách cấu hình và sử dụng thư viện LED bảy đoạn trên kit thí nghiệm.

2 Giới thiệu

Timer là một tính năng quan trọng trong vi điều khiển. Chúng giúp quản lý thời gian thực hiện tác vụ, tạo non-blocking code, kiểm soát thời gian kích hoạt các chân vi điều khiển và chạy hệ điều hành. Trong hướng dẫn này, chúng ta sẽ tìm hiểu cách cấu hình và sử dụng timer để chớp tắt đèn LED. Sử dụng timer interrupt để quét LED.

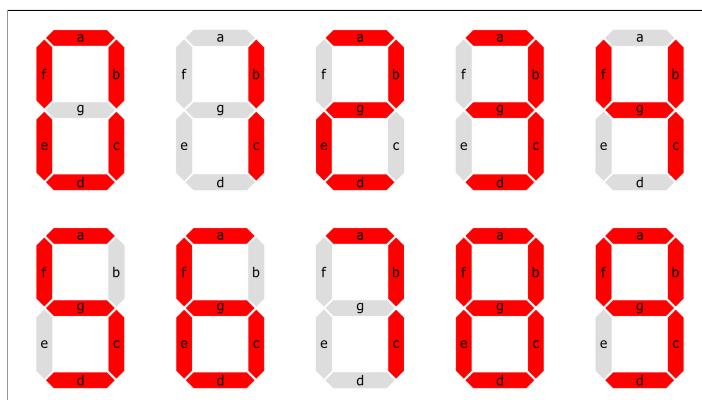


Hình 2.1: Giao diện của 4 LED bảy đoạn kết nối với vi điều khiển

Một trong những giao diện hiển thị thường thấy, trong các hệ thống là màn hình LED bảy đoạn. Mỗi LED bảy đoạn gồm 7 đèn LED (được kí hiệu từ a-g) hiển thị cho mỗi phân đoạn và 1 đèn LED hiển cho dấu chấm. Như vậy, với cách tiếp cận đơn giản nhất, mỗi đèn LED sẽ được điều khiển bằng 1 chân GPIO, như vậy với 1 LED bảy đoạn ta cần điều khiển bởi 8 chân GPIO, và giả sử ta cần điều khiển 4 LED bảy đoạn ta sẽ cần 32 chân GPIO. Trong thực tế, cách trên không được sử dụng vì tồn quá nhiều chân GPIO. Thay vào đó, các phân đoạn trên nhiều LED 7 đoạn sẽ được kết nối như hình 2.2 để tiết kiệm chân.

Theo như hình trên, LED7 đoạn được thiết kế theo loại dương chung (trong 1 LED bảy đoạn, chân dương của 8 LED phân đoạn được nối chung với nhau). Để tiết kiệm chân thì cực âm tương ứng của các LED phân đoạn (a-g và dấu ".") sẽ được nối chung với nhau. Cực dương của các LED bảy đoạn sẽ được điều khiển riêng biệt thông qua các chân GPIO kết hợp với Mosfet (Sử dụng Mosfet để tăng cường dòng điện cấp cho LED). Như vậy, tại một thời điểm, ta chỉ có thể hiển thị một giá trị trên cả 4 LED bảy đoạn. Và ta có thể điều khiển bật tắt hiển thị của 1 LED bảy đoạn thông qua 4 chân GPIO. Để có thể hiển thị 4 số khác nhau cùng lúc trên 4 LED bảy đoạn, ta cần sử dụng kĩ thuật quét LED. Để hiển thị số "1234", ta cần làm như sau:

- Đầu tiên, ta hiển thị số "1" trên LED đầu tiên, các LED còn lại tắt.
- Sau 1 giây, ta chuyển sang hiển thị số "2", tại LED tiếp theo, các LED còn lại tắt.
- Cứ như vậy, sau mỗi 1 giây, ta hiển thị số ở vị trí tiếp theo. Cứ thế lặp đi lặp lại, ta được hiệu ứng các số "1", "2", "3", "4" lần lượt hiển thị.
- Sau mỗi 4 giây, thì ta sẽ quét qua hết 4 LED, khoảng thời gian này được gọi là chu kỳ quét LED. Như vậy tần số quét LED sẽ là 0.25 Hz.
- Bây giờ ta tiến hành tăng tần số lên khoảng 60 Hz. Lúc này việc quét LED tuần tự diễn ra rất nhanh mà mắt người không nhìn kịp. Khi này hiệu ứng ta thấy được là số "1234" được hiển thị đồng thời trên 4 LED bảy đoạn.

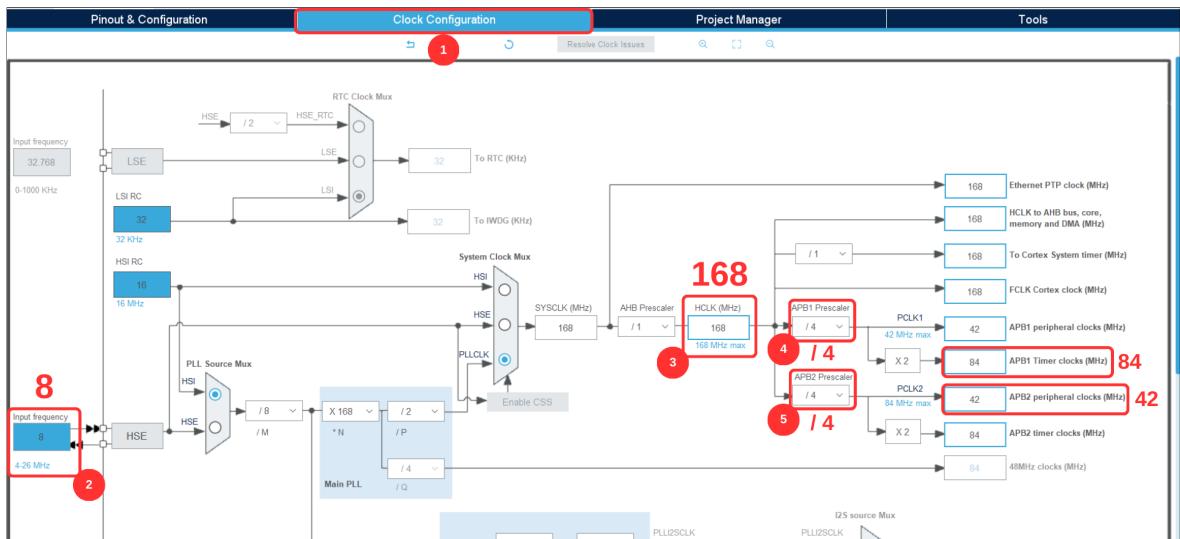


Hình 2.2: Biểu diễn ký tự số bằng LED 7 đoạn

3 Cấu hình timer

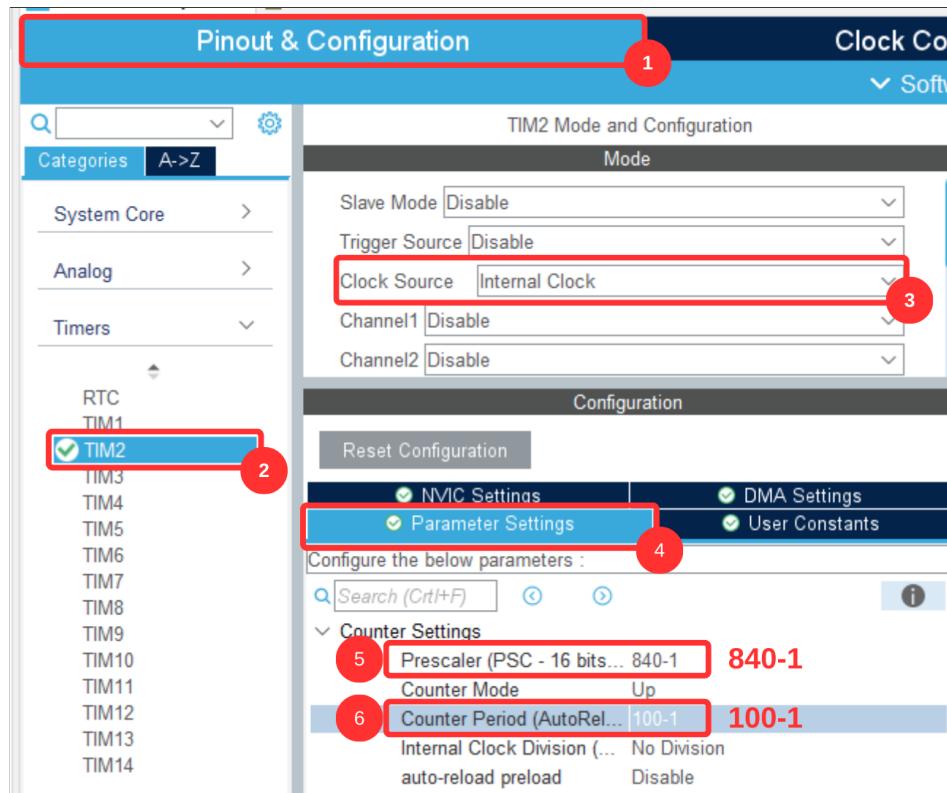
Bước 1: Tạo project mới được cấu hình như Lab 1.

Bước 2: Kiểm tra cấu hình Clock tại cửa sổ **Clock Configuration** (ở file .ioc). Kiểm tra cấu hình xung clock theo như hình dưới.



Hình 2.3: Clock source mặc định cho hệ thống

Bước 3: Cấu hình tham số TIM2 theo hình sau:



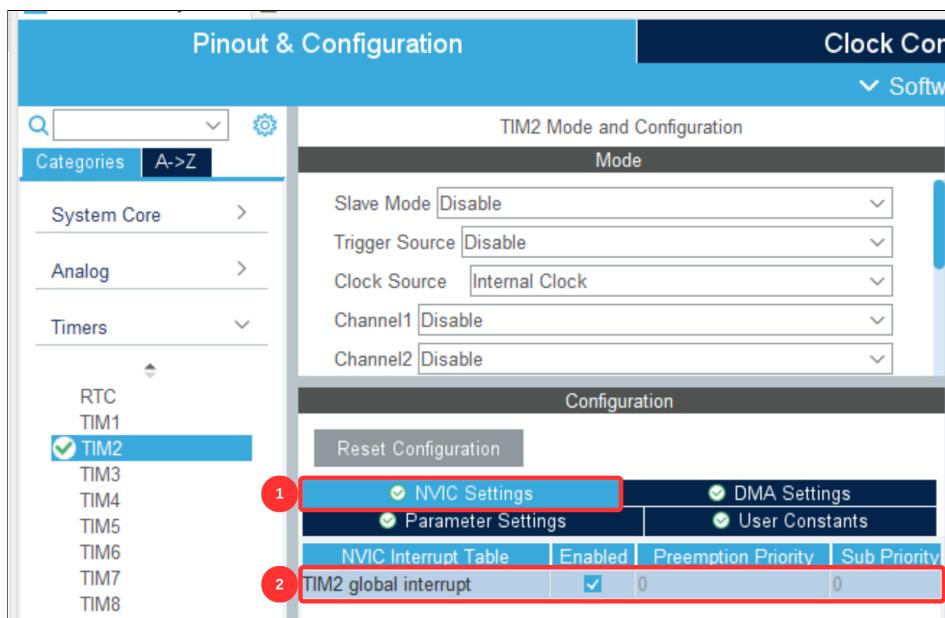
Hình 2.4: Cấu hình cho Timer 2

Chọn Clock Source cho timer 2 là **Internal Clock**. Cuối cùng cấu hình **Prescaller** là **840-1** và **Counter** là **100-1**. Những thông số này được giải thích như sau:

- Mục đích là tạo một timer với chu kỳ 1ms.

- Theo hướng dẫn sử dụng của chip STM32F407ZGT6, nguồn clock cấp cho module **TIM2** được kết nối với **APB1 Timer Clock**, mà theo chúng ta cấu hình ở hình 2.3 là 84MHz.
- Bằng cách đặt prescaler là 840-1, xung clock đầu vào của timer sẽ là $84\text{MHz}/(840-1+1) = 100\text{KHz}$.
- Hàm interrupt sẽ được gọi mỗi khi bộ đếm của timer đếm từ 0 tới 99, nghĩa là tần số lại tiếp tục được chia cho 100, trở thành **1kHz**, tương đương với chu kỳ **1ms**.

Bước 4: Kích hoạt timer interrupt tại cửa sổ **NVIC Settings** như sau:



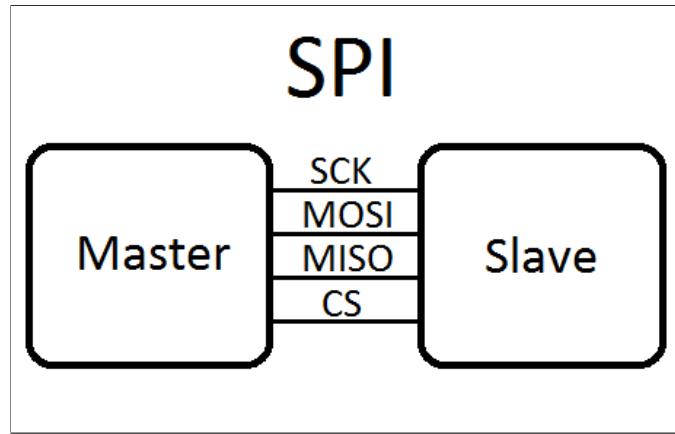
Hình 2.5: Kích hoạt timer interrupt

Ngoài ra, ta có thể cấu hình để các file mã nguồn của các ngoại vi sẽ được hệ thống sinh ra theo cặp file *.c và *.h

Cuối cùng lưu lại file cấu hình để hệ thống tự sinh code.

4 Cấu hình LED đồng hồ

LED đồng hồ trên kit thí nghiệm không được điều khiển trực tiếp từ chân MCU. Để tiết kiệm chân, LED đồng hồ được điều khiển thông qua 2 IC 74HC595. IC này có thể giao tiếp với vi điều khiển thông qua giao tiếp SPI.

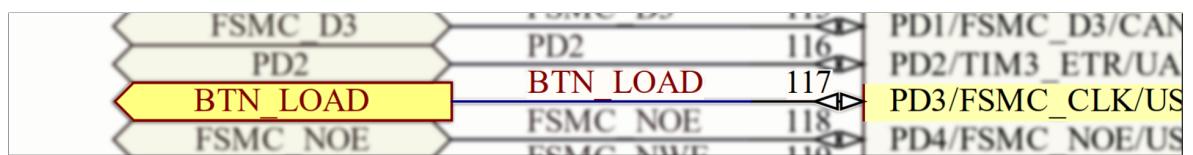


Hình 2.6: Sơ đồ giao tiếp SPI

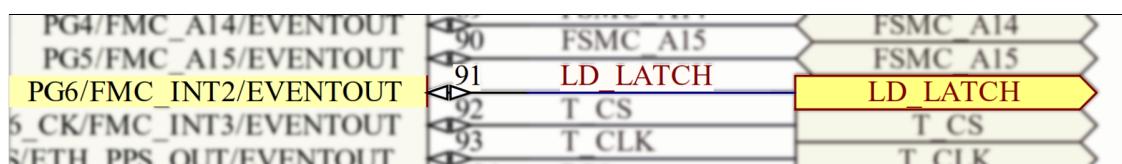
Theo schematic, ta có những chân cần config như sau:



Hình 2.7: Sơ đồ nguyên lý của các chân SPI1_SCK, SPI1_MISO, SPI1_MOSI

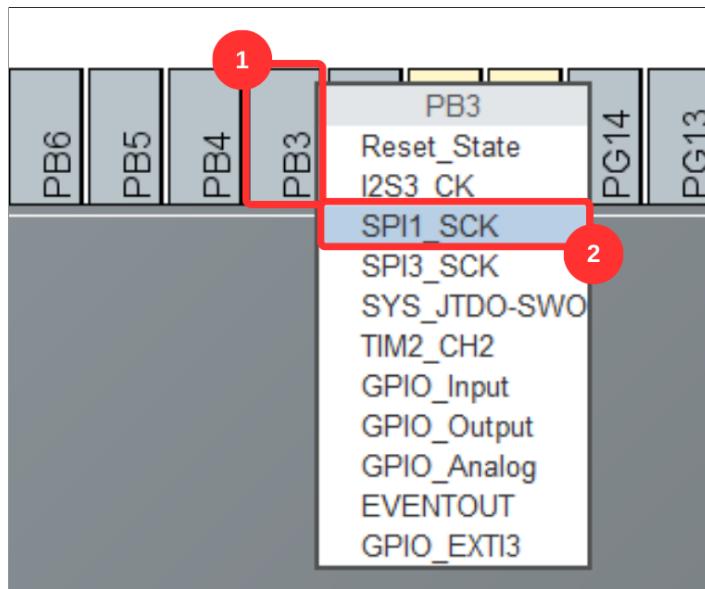


Hình 2.8: Sơ đồ nguyên lý của chân BTN_LOAD



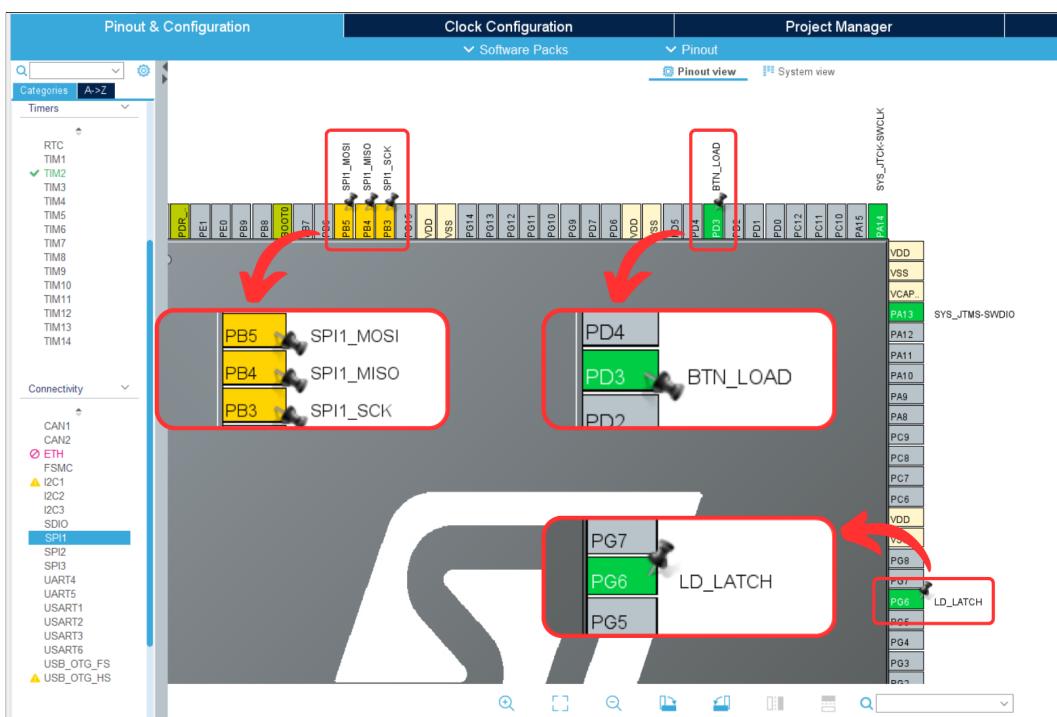
Hình 2.9: Sơ đồ nguyên lý của chân LD_LATCH

Ta thực hiện config các chân SPI1_SCK, SPI1_MISO, SPI1_MOSI bằng cách nhấp chuột phải vào chân tương ứng, sau đó chọn kiểu chân dựa theo schematic (hình 2.7). Hai chân BTN_LOAD và LD_LATCH sẽ được config thành chân output.



Hình 2.10: Config các chân SPI1_SCK, SPI1_MISO, SPI1_MOSI

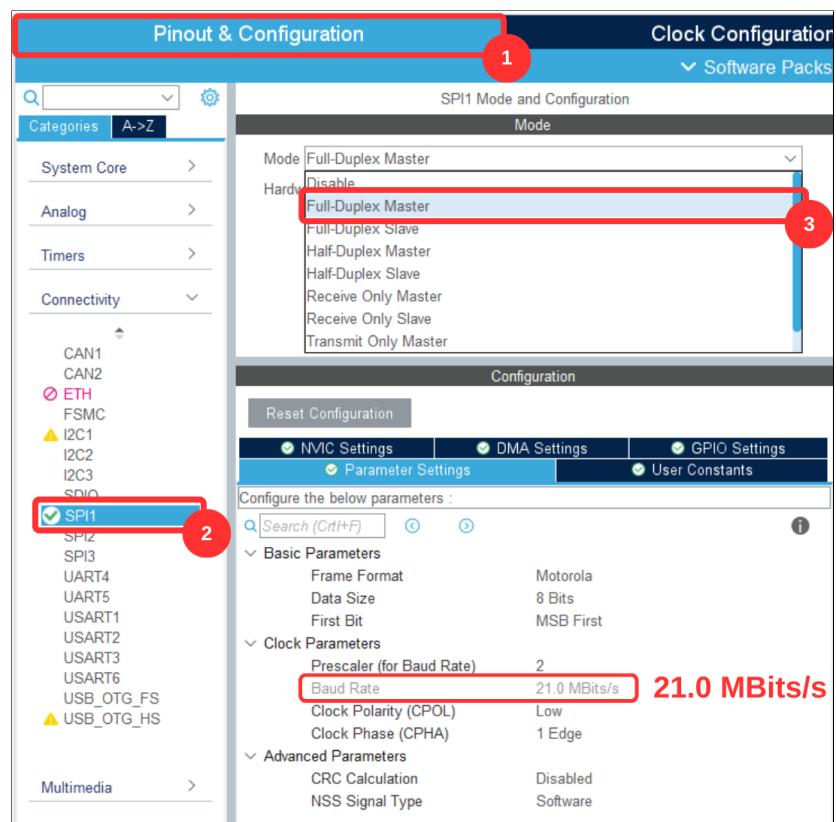
Sau khi config, ta có được Pinout View như sau:



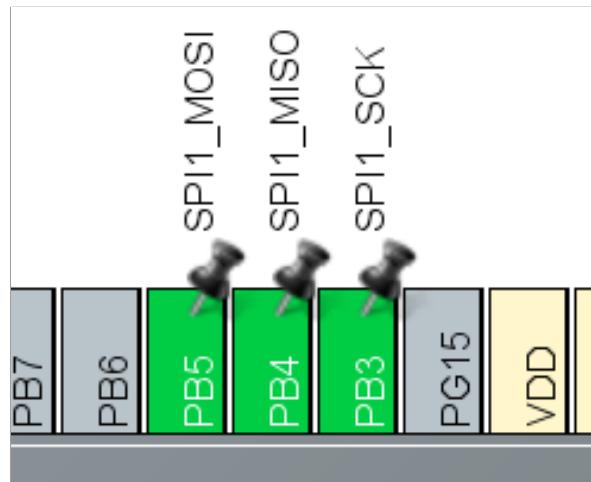
Hình 2.11: Các chân cần config

Để vi điều khiển giao tiếp với LED đồng hồ, ta cần config SPI1 như hình sau:

Sau khi config SPI1, ta sẽ có Pinout View như sau:



Hình 2.12: Config SPI



Hình 2.13: Sau khi config SPI1

5 Hướng dẫn lập trình

Các file **software_timer** và **LED7_seg** có thể được copy từ project mẫu **Bai2_Timer_1**. từ project mẫu đã cung cấp.

Ý tưởng của software timer là từ timer interrupt 1ms, ta sẽ dùng các biến đếm để có thể tạo ra các timer mềm với tần số thấp hơn.

- Hàm **HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)**: là hàm ngắt interrupt service routine (isr). Với timer2 đã được cấu hình như trên, isr sẽ được thực thi mỗi 1ms. Trong hàm này, các counter của software timer sẽ được đếm, và khi hết thời gian một tín hiệu flag_timer sẽ được bật để chương trình chính nhận biết.

```
1 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *  
2     htim) {  
3     if (htim->Instance == TIM2) {  
4         //timer2_isr  
5         if (timer2_counter > 0) {  
6             timer2_counter--;  
7             if (timer2_counter == 0) {  
8                 timer2_flag = 1;  
9                 timer2_counter = timer2_mul;  
10            }  
11        }  
12    }  
13 }
```

Program 2.1: Ví dụ timer interrupt

- Hàm **timer2_init**: sẽ được gọi để khởi tạo timer 2 của vi điều khiển.
- Hàm **timer2_set**: sẽ khởi tạo chu kỳ của software timer 2. Nếu như cần sử dụng nhiều timer hơn thì có thể tạo thêm các biến đếm và hàm tương tự.

```
1 #ifndef INC_LED_7SEG_H_  
2 #define INC_LED_7SEG_H_  
3  
4 /* Includes */  
5 #include "stdint.h"  
6
```

```

7 /* Variables */
8
9 /* Functions */
10 extern void led_7seg_init();
11 extern void led_7seg_set_digit(int num, int position,
12     uint8_t show_dot);
12 extern void led_7seg_set_colon(uint8_t status);
13 extern void led_7seg_display();
14
15 extern void led_7seg_debug_turn_on(uint8_t index);
16 extern void led_7seg_debug_turn_off(uint8_t index);
17
18 #endif /* INC_LED_7SEG_H_ */

```

Program 2.2: LED7_seg.h

- **led_7seg_init:** hàm khởi tạo LED đồng hồ
- **led_7seg_display:** hàm quét LED. Hàm này nên được gọi trong timer interrupt 1ms.Có nghĩa tần số quét hiện là **1KHz/4 = 250Hz**. Để thử nghiệm thay đổi tần số quét thì có thể dùng software timer và thay đổi vị trí đặt hàm này.
- **led_7seg_set_digit:** hàm hiển thị số trên LED, tham số đầu tiên là số muốn hiển thị, tham số thứ 2 là vị trí của LED muốn hiển thị số đó (0-4), tham số cuối cùng thể hiện rằng dấu chấm tại LED 7 đoạn đó có được hiển thị hay không.
- **led_7seg_set_colon:** hàm hiển thị hoặc tắt dấu hai chấm.
- **led_7seg_debug_turn_on:** hàm bật các LED 6,7,8; tham số đầu vào là index của các LED muốn bật.
- **led_7seg_debug_turn_off:** hàm tắt các LED 6,7,8; tham số đầu vào là index của các LED muốn tắt.

```

1
2 int main(void) {
3     /* USER CODE BEGIN 1 */
4
5     /* USER CODE END 1 */
6

```



```

43     ReverseOutput(1);
44 }
45
46 if (timer3_flag == 1) {
47     timer3_flag = 0;
48     ReverseOutput(0);
49 }
50
51 /* USER CODE END WHILE */
52
53 /* USER CODE BEGIN 3 */
54 }
55 /* USER CODE END 3 */
56 }
57 // ...
58 /* USER CODE BEGIN 4 */
59 void init_system(void) {
60     timer2_init();
61     timer3_init();
62     timer2_set(1000);
63     timer3_set(5000);
64
65     delay_ms(1000);
66 }
```

Program 2.3: hàm main

Để rõ ràng cho việc lập trình, các code khởi tạo sẽ được để trong hàm **system_init** và hàm này sẽ được gọi trước vòng **while** trong hàm **main**.

Câu lệnh **timer2_set(1000)** sẽ khởi tạo một software timer có chu kỳ là 1000ms, tương tự với timer3. Đoạn code trên giúp ta khảo sát hoạt động của các timer khác nhau.

Trong vòng lặp vô tận while, ta sẽ chờ cho đến khi tín hiệu **flag_timer2** được bật (mỗi 1000ms) sau đó sẽ thực thi tác vụ ta mong muốn, tương tự với timer3.

```

1 void delay_ms(int value) {
2     HAL_Delay(1000);
3 }
4
```

```

5 void OpenOutput(int index) {
6     if (index >= 0 && index <= 1) {
7         HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port, arrayMapOfOutput
8             [index], 1);
9         statusOutput[index] = ON;
10    }
11}
12
13 void CloseOutput(int index) {
14     if (index >= 0 && index <= 1) {
15         HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port, arrayMapOfOutput
16             [index], 0);
17         statusOutput[index] = OFF;
18    }
19}
20
21 void ReverseOutput(int index) {
22     if (statusOutput[index] == ON) {
23         CloseOutput(index);
24         statusOutput[index] = OFF;
25     } else {
26         OpenOutput(index);
27         statusOutput[index] = ON;
28     }
29}
30
31 unsigned char statusLed_0 = 0;
32
33 void Led_0() {
34     statusLed_0 = (statusLed_0 + 1) % 20; // 50 (ms) * 20 =
35     1000 (ms)
36     if (statusLed_1 < 4)
37         OpenOutput(0);
38     else
39         CloseOutput(0);
40}
41
42 unsigned char statusLed_1 = 0;

```

```

41 void Led_1() {
42     statusLed_1 = (statusLed_1 + 1) % 40; // 50 (ms) * 40 =
43     2000 (ms)
44     if (statusLed_1 < 10)
45         OpenOutput(1);
46     else
47         CloseOutput(1);
}

```

Program 2.4: Các hàm liên quan khác

Trong các ứng dụng thực tế, 50ms thường là chu kì ổn định cho các việc đọc input, xử lí và xuất tín hiệu output do đó ta chỉ cần dung 1 timer duy nhất để định thời mỗi 50ms. Và ta sẽ định thời các chu kì lớn hơn bằng cách sử dụng biến đếm. Xem xét đoạn chương trình sau và các hàm **Led_0** và **Led_1** để hiểu hơn về cách thực thi các tác vụ với chu kì khác nhau của chương trình.

```

1 int main(void) {
2     /* USER CODE BEGIN 1 */
3
4     /* USER CODE END 1 */
5
6     /* MCU Configuration
7     -----
8     */
9     /* Reset of all peripherals, Initializes the Flash
10    interface and the Systick. */
11    HAL_Init();
12
13
14    /* Configure the system clock */
15    SystemClock_Config();
16
17    /* USER CODE BEGIN SysInit */
18    /* USER CODE END SysInit */
19
20    /* Initialize all configured peripherals */

```

```

21    MX_GPIO_Init();
22    MX_TIM2_Init();
23    MX_TIM3_Init();
24    MX_TIM4_Init();
25    /* USER CODE BEGIN 2 */
26    init_system();
27
28    OpenOutput(0);
29    delay_ms(2000);
30    CloseOutput(0);
31
32    OpenOutput(1);
33    delay_ms(2000);
34    CloseOutput(1);
35    /* USER CODE END 2 */
36
37    /* Infinite loop */
38    /* USER CODE BEGIN WHILE */
39    while (1) {
40        while(!timer2_flag);
41        timer2_flag = 0;
42        Led_0();
43        Led_1();
44
45        /* USER CODE END WHILE */
46
47        /* USER CODE BEGIN 3 */
48    }
49    /* USER CODE END 3 */
50}
51
52 void init_system(void) {
53     timer2_init();
54     timer2_set(50);
55
56     delay_ms(1000);
57 }
```

Program 2.5: hàm main

6 Exercise and Report

6.1 Exercise 1

6.2 Bài tập 1

Hiển thị 4 số khác nhau trên LED đồng hồ, sử dụng kĩ thuật quét led với các tần số 1Hz, 10Hz, 100Hz, 1000Hz và quan sát.

6.3 Bài tập 2

Để nhấp nháy LED mỗi 1 giây, thay vì sử dụng software timer, ta có thể cấu hình hardware timer với chu kì 1 giây, sau đó thực hiện bật tắt LED trong hàm ngắt isr của timer hay không ?

6.4 Bài tập 3

Sử dụng thư viện LED 7 đoạn trên Kit thí nghiệm, điều khiển LED 7 đoạn hiển thị như hình sau:



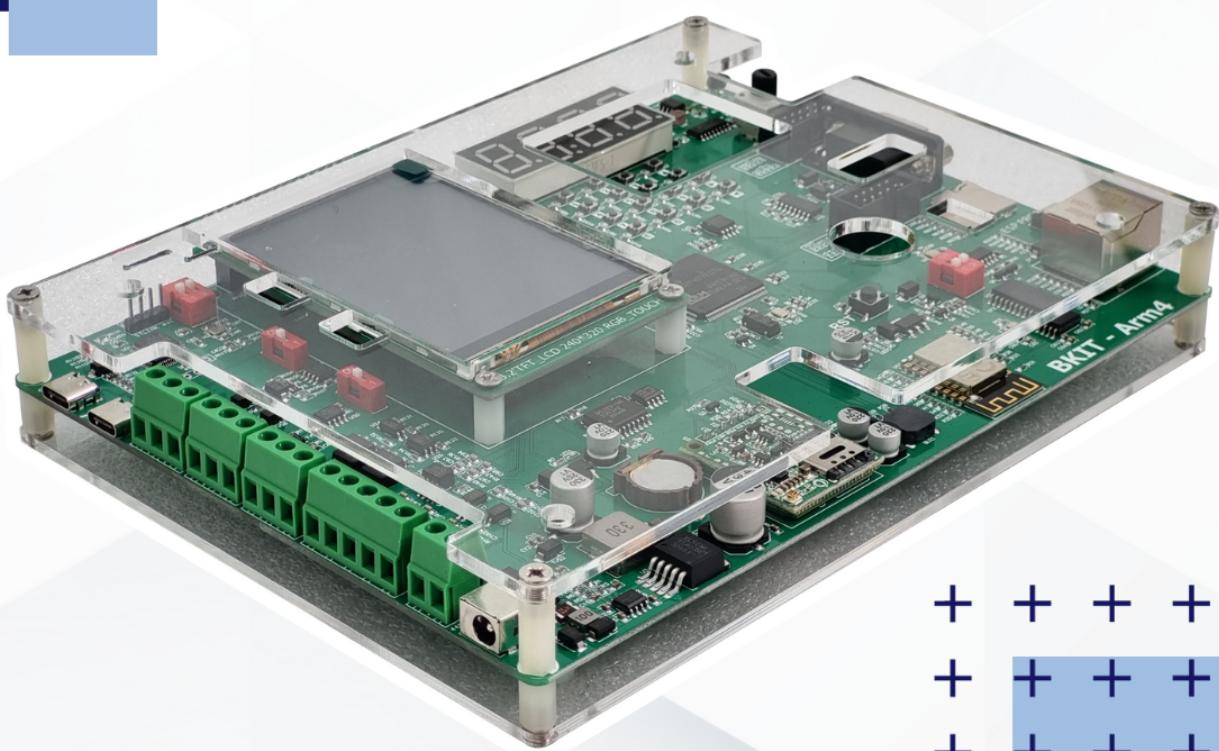
Hình 2.14: Kết quả hiện thực trên Kit thí nghiệm

6.5 Bài tập 4

Sử dụng module LED đồng hồ, và khung chương trình timer 50ms để tạo ứng dụng đồng hồ điện tử đơn giản. LED đồng hồ sẽ dùng để hiển thị phút và giây của đồng hồ.

CHƯƠNG 3

LCD and Button matrix



1 Mục tiêu

- Hiểu về nguyên lý hoạt động của nút nhấn sử dụng điện trở kéo lên.
- Hiểu về nguyên lý hoạt động và cách điều khiển LCD.
- Biết cách cấu hình sử dụng ma trận phím và LCD trên Kit thí nghiệm.
- Biết cách sử dụng các thư viện về ma trận phím và LCD.

2 Giới thiệu

Hệ thống nhúng thường sử dụng nút nhấn để tương tác với người dùng. Điều này áp dụng từ những hệ thống điều khiển từ xa cơ bản nhất như mở cửa nhà xe, cho đến hệ thống máy bay không người lái tinh vi nhất. Cho dù bạn phát triển hệ thống nào, bạn cũng cần có khả năng xử lý tín hiệu nút nhấn hiệu quả.

Một nút nhấn thường được nối với MCU để tạo ra một mức logic nhất định khi được ấn hoặc đóng hoặc "active" và mức logic ngược lại khi không được nhấn hoặc mở hoặc "inactive". Mức logic hoạt động có thể là '0' hoặc '1', nhưng vì lý do lịch sử và điện, mức hoạt động '0' là phổ biến hơn.

Chúng ta có thể sử dụng một nút nếu muốn thực hiện các thao tác như:

- Điều khiển một chiếc xe khi công tắc được nhấn.
- Bật một bóng đèn khi công tắc được nhấn.
- Kích hoạt một máy bơm khi công tắc được nhấn.

Các thao tác này có thể được thực hiện bằng nút nhấn điện mà không cần sử dụng vi điều khiển; tuy nhiên, việc sử dụng vi điều khiển có thể phù hợp nếu chúng ta yêu cầu những hành vi phức tạp hơn. Ví dụ:

- Khởi động một chiếc xe motor khi công tắc được nhấn.

Điều kiện: Nếu không đảm bảo an toàn, xe sẽ kêu 2 lần.

- Bật đèn khi công tắc được bật.

Điều kiện: Để tiết kiệm năng lượng, bỏ qua yêu cầu bật đèn vào ban ngày.

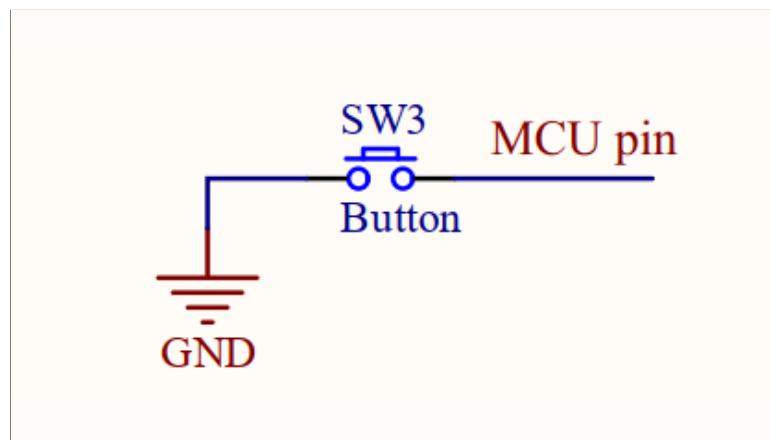
- Kích hoạt một máy bơm khi công tắc được bật.

Điều kiện: Nếu bình chứa nước chính dưới 300 lít, không khởi động máy bơm chính. Thay vào đó, khởi động máy bơm dự trữ và rút nước từ bể khẩn cấp.

Trong bài lab này, chúng ta sẽ tìm hiểu cách đọc đầu vào từ các nút cơ học trong lập trình nhúng bằng vi điều khiển.

3 Cơ sở lý thuyết

3.1 Sơ đồ kết nối nút nhấn với MCU

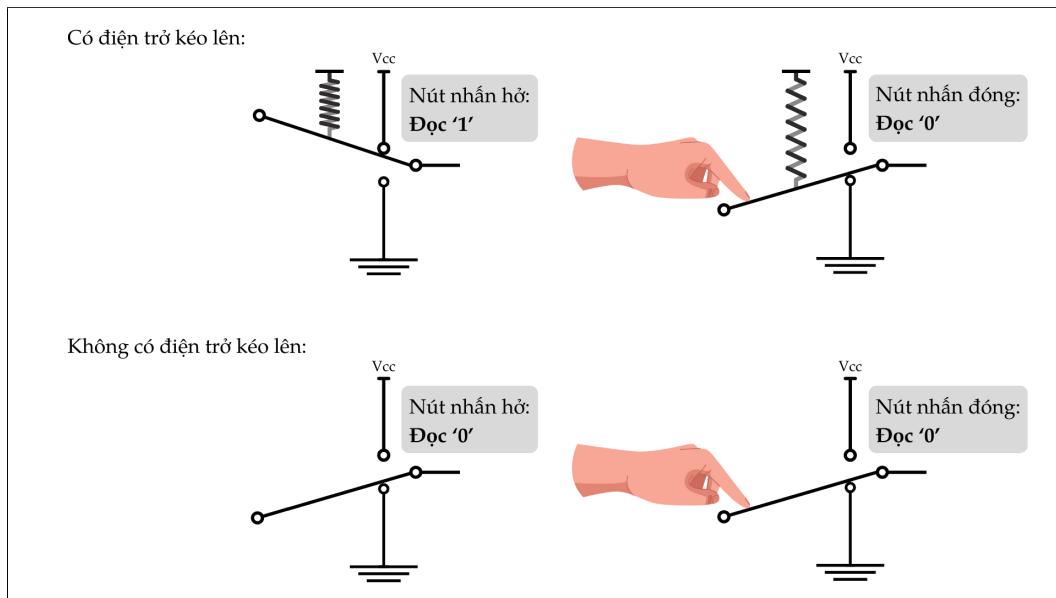


Hình 3.1: Kết nối nút nhấn với MCU

Hình 3.1 thể hiện một cách kết nối nút nhấn với MCU. Kết nối trên được mô tả như sau:

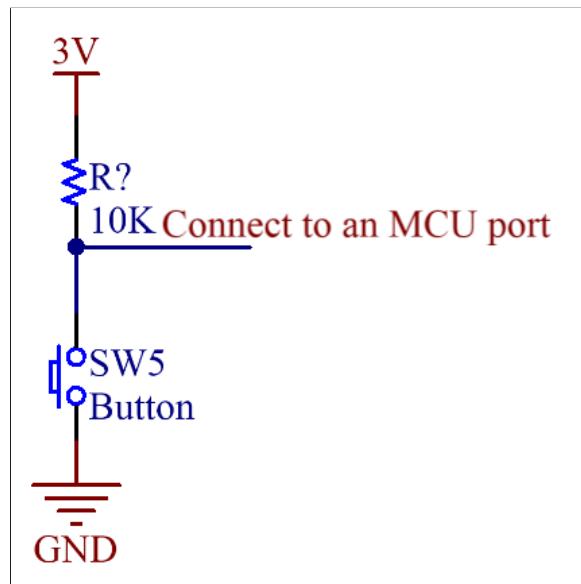
- Khi nút nhấn mở (không được nhấn), một điện trở nội bộ trong MCU sẽ "kéo" chân MCU lên nguồn (3.3V với STM32F407). Nếu chúng ta đọc giá trị của chân MCU, ta sẽ được giá trị "1".
- Khi nút nhấn đóng (được nhấn), chân MCU sẽ được nối với GND và sẽ có điện áp là 0V. Nếu ta đọc giá trị chân MCU, ta sẽ được giá trị "0".

Tuy nhiên, nếu MCU không có sẵn điện trở nội bộ kéo lên, khi nhấn nút thì giá trị đọc được sẽ là "0". Nhưng khi nút nhấn được thả, lúc này chân MCU sẽ ở trạng thái "thả nổi" và mức logic sẽ không được xác định.



Hình 3.2: Sự cản thiết của điện trở kéo lên

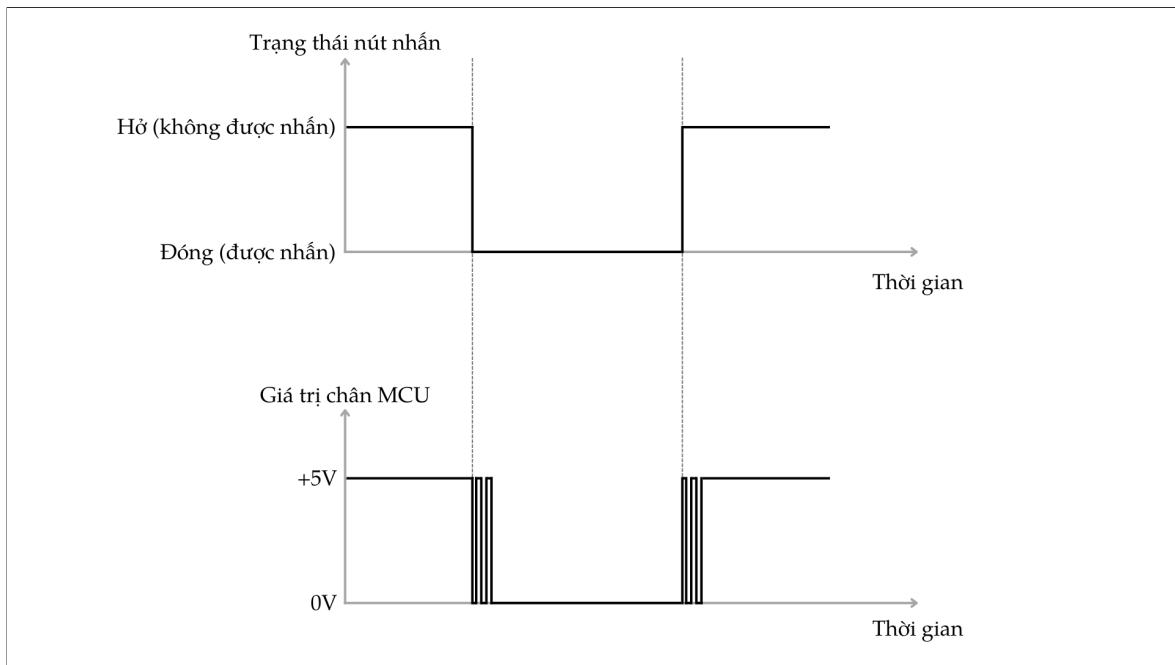
Vậy nên khi kết nối nút nhấn hoặc công tắc với MCU thì ta nên sử dụng một điện trở kéo lên như hình 3.2.



Hình 3.3: Cách kết nối nút nhấn đáng tin cậy

3.2 Chống rung nút nhấn

Trong thực tế, tất cả nút nhấn đều xảy ra hiện tượng rung (tín hiệu chuyển đổi liên tục giữa bật và tắt trong thời gian ngắn) do sự va đập của các chi tiết cơ khí.



Hình 3.4: Hiện tượng rung của nút nhấn

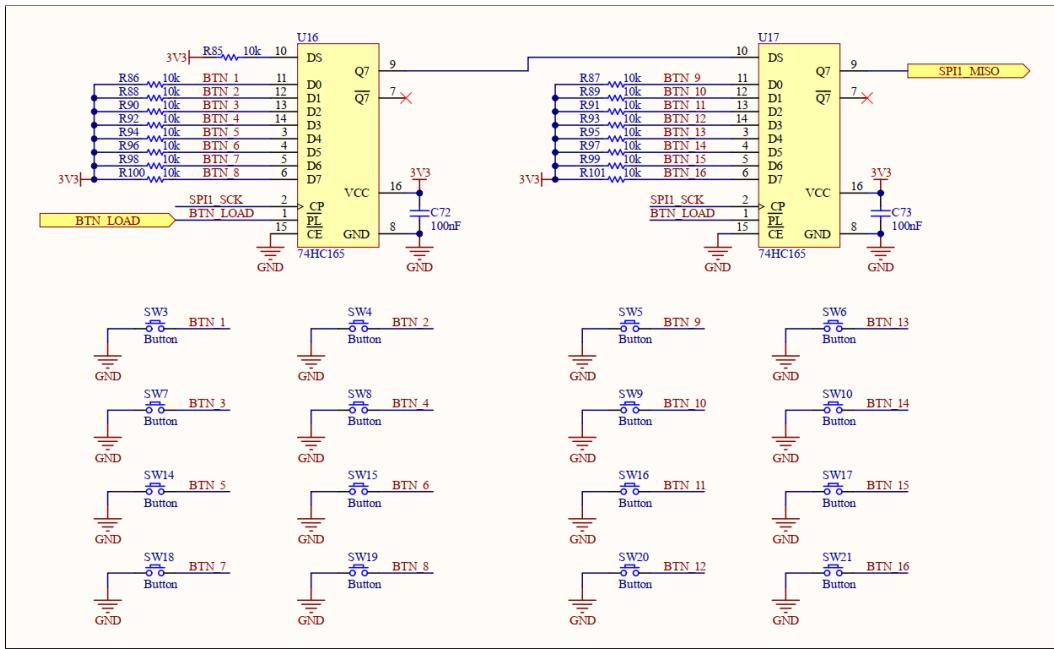
Có nhiều phương pháp chống rung nút nhấn sử dụng phần cứng hoặc phần mềm. Một cách chống rung bằng phần mềm là đọc nút nhấn sau mỗi chu kì (ví dụ: 10ms) và lưu lại trạng thái trước đó của nút nhấn. Một tín hiệu nút nhấn sẽ được chấp nhận khi không có sự thay đổi giữa tín hiệu hiện tại và tín hiệu trước đó.

Tuy nhiên, trong bài lab hiện tại, việc chống rung nút nhấn sẽ được xử lý đơn giản bằng cách đọc tín hiệu nút nhấn mỗi 50ms. Phương pháp này tuy đơn giản nhưng vẫn đạt được hiệu quả mong muốn với đa số các nút nhấn vì trong tiêu chuẩn công nghiệp, nhà sản xuất nút nhấn thường sẽ đảm bảo rằng tín hiệu của nút nhấn sẽ ổn định trong 50ms.

4 Hướng dẫn cấu hình

4.1 Cấu hình nút nhấn

Trong thiết kế của Kit thí nghiệm, các nút nhấn trên ma trận phím không được kết nối trực tiếp với MCU mà thông qua IC 74HC165. Đây là thanh ghi dịch ngõ vào song song, ngõ ra nối tiếp (trái ngược với IC 74HC595 được đề cập trong lab 2) được dùng với mục đích giảm số chân dùng để điều khiển nút nhấn.

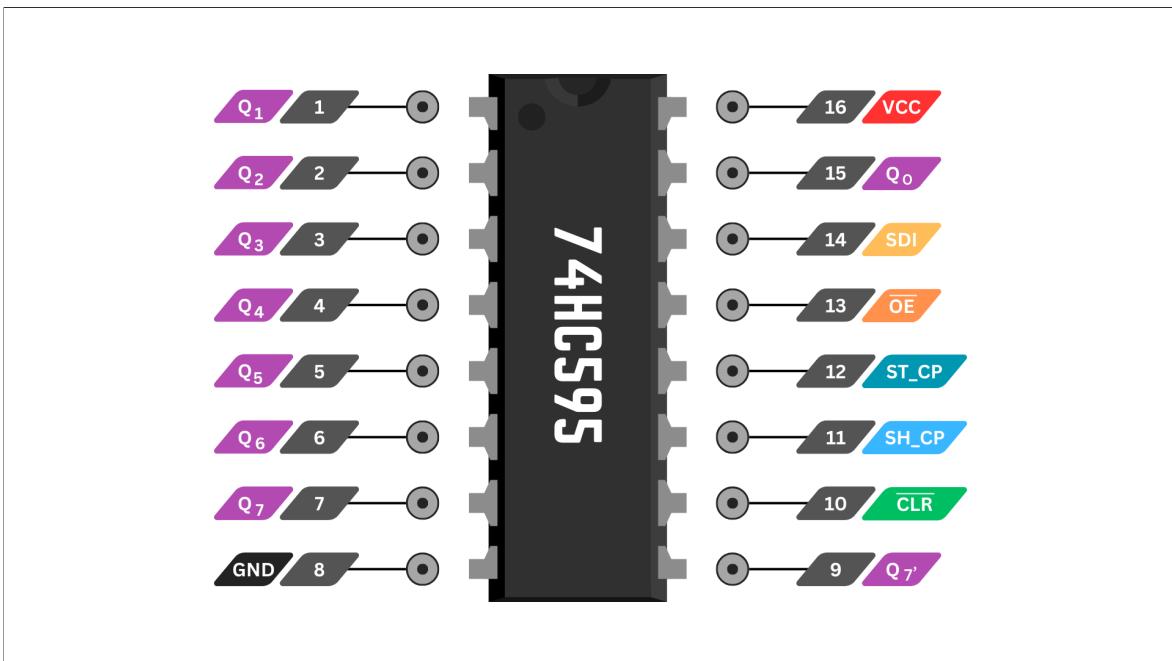


Hình 3.5: Kết nối ma trận phím trong Kit thí nghiệm

Nếu kết nối trực tiếp nút nhấn với MCU, với 16 nút nhấn sẽ phải tốn tới 16 chân của vi điều khiển, điều này khá lãng phí. Để giải quyết vấn đề này, ta có thể rút ngắn bằng việc xử lý chúng theo hàng và cột (theo ma trận). Khi đó, ta sẽ chuyển 16 nút nhấn thành 4 hàng và 4 cột (tương đương với 8 chân vi điều khiển).

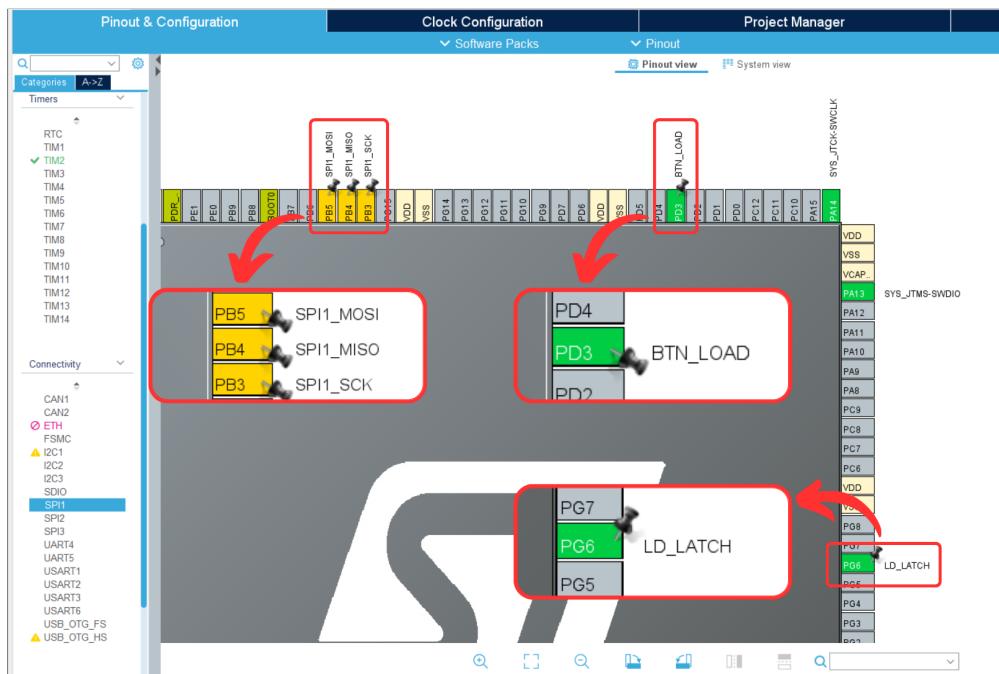
Với phương pháp sử dụng IC 74HC165 ta đang dùng, ta chỉ cần kết nối 3 chân của IC với MCU, cụ thể là các chân **PL**, **CP**, **Q7**. IC 74HC165 hoạt động với nguyên lý sau:

- Khi tín hiệu chân **PL** ở mức thấp, giá trị từ các ngõ vào D0-D7 sẽ được nạp song song vào thanh ghi trong IC.
- Khi tín hiệu chân **PL** ở mức cao và phát hiện cạnh lên ở chân **CP**, các giá trị bên trong thanh ghi sẽ được dịch và tín hiệu **Q7** sẽ lần lượt thể hiện các giá trị trong thanh ghi.



Hình 3.6: IC 74HC165

Tương tự với IC 74HC595 ở lab 2, cơ chế hoạt động của 74HC165 có thể được điều khiển bởi giao tiếp SPI và sử dụng module SPI1 trên MCU. Tín hiệu SCK, MISO của MCU sẽ lần lượt được kết nối với chân **CP** và **Q7** của IC. Lưu ý cần config chân **BTN_LOAD** thành **GPIO_OUTPUT** nếu chưa config.



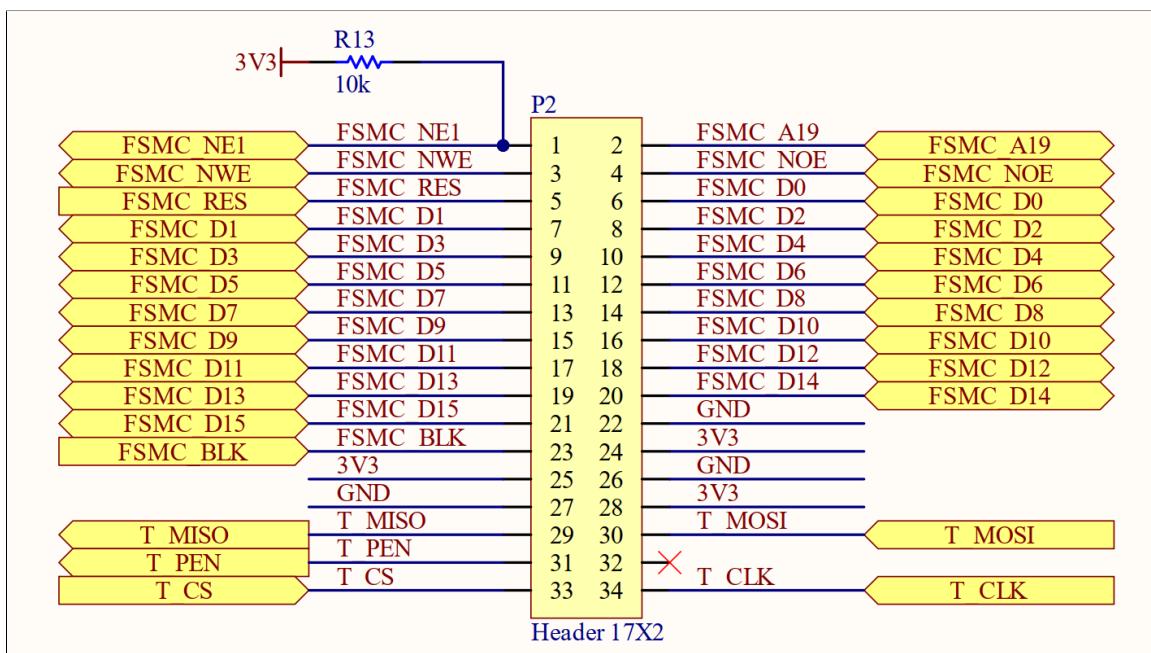
Hình 3.7: Các chân cần config

4.2 Cấu hình LCD

Để giao tiếp với màn hình LCD, chúng ta cần phải giao tiếp với 1 IC driver trung gian, đối với Kit thí nghiệm **BKIT - Arm 4** là driver ILI9341. Ta chỉ cần ghi các lệnh đã được nhà sản xuất đặc tả, IC driver sẽ tự điều khiển hiển thị trên màn hình LCD. Đối với thiết kế của Kit thí nghiệm, MCU sẽ giao tiếp với driver thông qua FSMC, một loại giao tiếp song song giúp MCU kết nối với bộ nhớ ngoài với tốc độ nhanh. Một số thông số quan trọng khi làm việc với LCD và driver điều khiển LCD:

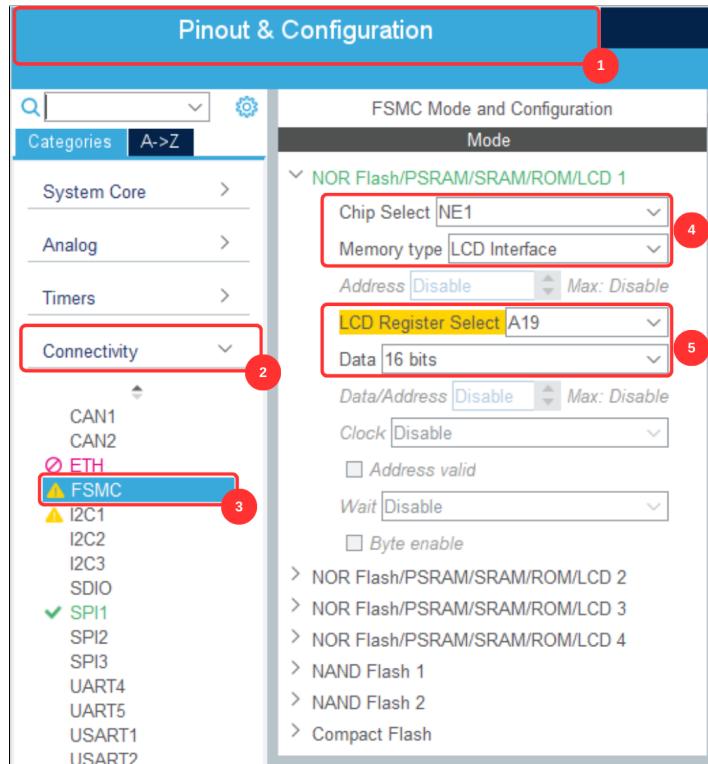
- Độ phân giải: 320x240 pixels.
- Chế độ màu: RGB 16 bit.
- Số bit dữ liệu giao tiếp với driver: 16.

Theo schematic, chúng ta sẽ cấu hình các chân FSMC, và thiết lập các thông số liên quan đến thời gian trong giao tiếp như sau:

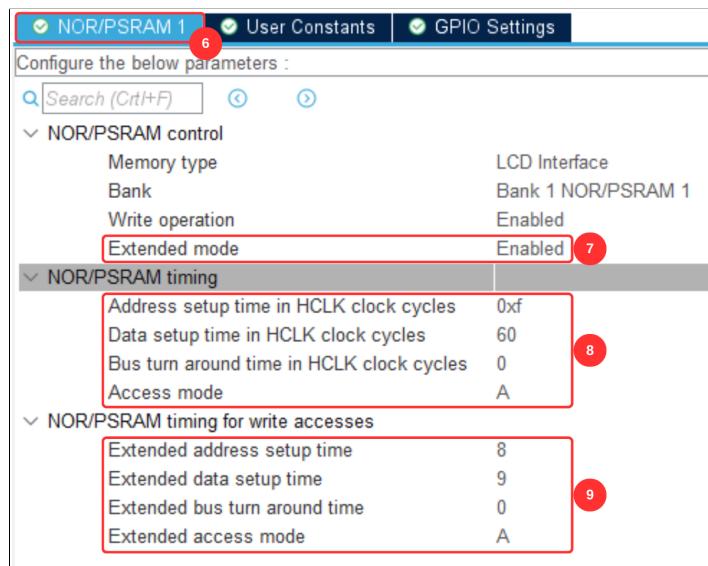


Hình 3.8: Schematic của LCD

Để vi điều khiển giao tiếp với LCD thì theo schematic, chúng ta cấu hình FSMC như sau:

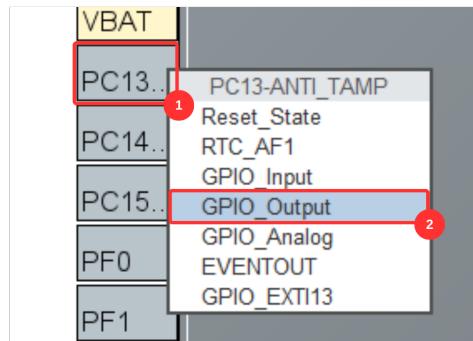


Hình 3.9: Cấu hình FSMC

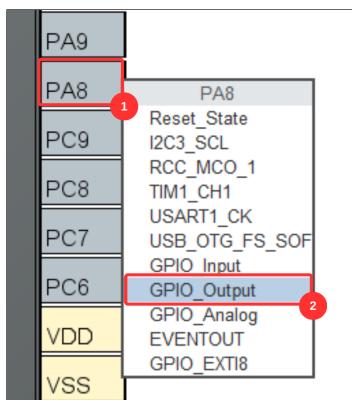


Hình 3.10: Cấu hình FSMC (tiếp theo)

Ngoài ra, ta cần cấu hình 2 chân: **FSMC_RES** (reset LCD) và **FSMC_BLK** (đèn nền LCD):

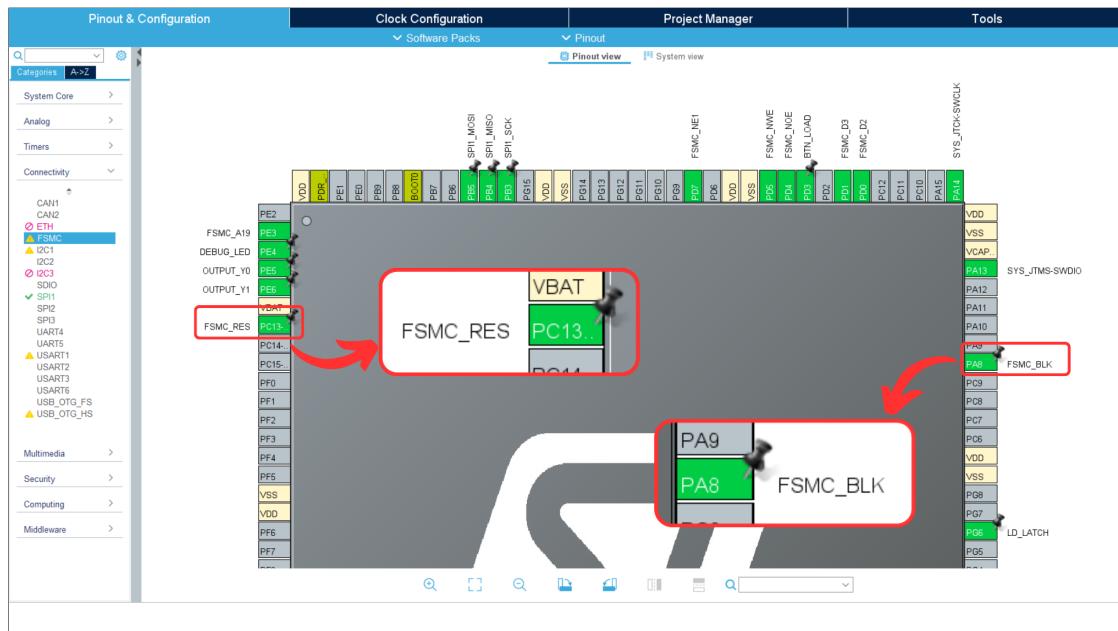


Hình 3.11: Cấu hình FSMC_RES



Hình 3.12: Cấu hình FSMC_BLK

Sau khi cấu hình, ta có kết quả Pinout view như sau:



Hình 3.13: Cấu hình FSMC RES và FSMC BLK

5 Hướng dẫn lập trình

5.1 Sử dụng thư viện: button.h

File **button.h** và **button.c** có thể sao chép từ project mẫu **Bai3_LCD_Button**.

extern uint16_t button_count[16]

- **Mô tả:** Có thể truy xuất để lấy giá trị đếm của các nút nhấn. Nếu nút nhấn được nhấn trong khi gọi hàm **button_scan** thì biến đếm tương ứng tăng 1 đơn vị, ngược lại nếu nút nhấn không được nhấn, giá trị của biến đếm sẽ về 0.

void button_init()

- **Mô tả:** Khởi tạo nút nhấn. Gợi ý: gọi trong hàm **system_init** trong **main.c**.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void button_scan()

- **Mô tả:** Quét đọc tín hiệu tất cả nút nhấn. Gợi ý: gọi mỗi 50ms trong vòng lặp while trong **main.c**.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

Để kiểm tra một nút nhấn i có được nhấn hay không, ta chỉ cần so sánh giá trị của biến **button_count[i]** vì khi được nhấn thì trong vòng 50ms biến đếm sẽ tăng lên 1. Dưới đây là một ví dụ đọc giá trị nút nhấn có thứ tự 0 trên ma trận phím (phím số "1") và đảo DEBUG_LED nếu phím đó được nhấn.

```
1 #include "software_timer.h"
2 #include "led7_seg.h"
3 #include "button.h"
4 //...
5 int main(void)
6 {
7     //...
8     /* USER CODE BEGIN 2 */
```

```

9   system_init();
10  /* USER CODE END 2 */

11
12  /* Infinite loop */
13  /* USER CODE BEGIN WHILE */
14  while (1)
15  {
16      while(!flag_timer2);
17      flag_timer2 = 0;
18      // main task, every 50ms
19      // read input
20      button_scan();
21      // process
22      // control output
23      if(button_count[0] == 1)
24          HAL_GPIO_Toggle(DEBUG_LED_GPIO_Por,
25 DEBUG_LED_Pin);
26      /* USER CODE END WHILE */
27
28      /* USER CODE BEGIN 3 */
29  }
30 }

31
32 /* USER CODE BEGIN 4 */
33 void system_init(){
34     timer_init();
35     led7_init();
36     button_init();
37     setTimer2(50);
38 }
```

Program 3.1: Ví dụ đọc nút nhấn

Để phát hiện một nút nhấn đang nhấn giữ, ta có thể làm như sau: Giả sử nút nhấn thứ 0 đang được nhấn, thì mỗi 50ms biến đếm sẽ tăng 1 đơn vị. Nếu ta muốn phát hiện nút nhấn được nhấn giữ trong 2s, thì ta cần so sánh biến đếm với giá trị 40 (vì 2s sẽ tương ứng với $2000/50 = 40$ chu kỳ đọc nút nhấn liên tiếp).

```
1 int main(void)
```

```

2 {
3     /* USER CODE BEGIN 2 */
4     system_init();
5     /* USER CODE END 2 */
6
7     /* Infinite loop */
8     /* USER CODE BEGIN WHILE */
9     while (1)
10    {
11        while(!flag_timer2);
12        flag_timer2 = 0;
13        // main task, every 50ms
14        // read input
15        button_scan();
16        // process
17        // control output
18        if(button_count[0] == 40)
19            HAL_GPIO_Toggle(DEBUG_LED_GPIO_Port,DEBUG_LED_Pin);
20     /* USER CODE END WHILE */
21
22     /* USER CODE BEGIN 3 */
23    }
24     /* USER CODE END 3 */
25 }
```

Program 3.2: Ví dụ nút nhấn được nhấn giữ trong 2s

Dưới đây là trường hợp ta muốn tạo hiệu ứng đèn LED đảo trạng thái khi nút nhấn được nhấn và nếu nhấn giữ thì sau mỗi 2s LED sẽ tiếp tục đảo trạng thái.

```

1 int main(void)
2 {
3     /* USER CODE BEGIN 2 */
4     system_init();
5     /* USER CODE END 2 */
6
7     /* Infinite loop */
8     /* USER CODE BEGIN WHILE */
9     while (1)
10    {
11        while(!flag_timer2);
```

```

12     flag_timer2 = 0;
13     // main task, every 50ms
14     // read input
15     button_scan()
16     // process
17     // control output
18     if(button_count[0] % 40 == 1)
19         HAL_GPIO_Toggle(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin);
20     /* USER CODE END WHILE */
21
22     /* USER CODE BEGIN 3 */
23 }
24 /* USER CODE END 3 */
25 }
```

Program 3.3: Ví dụ xử lí nút nhấn

5.2 Sử dụng thư viện: lcd.h

File **lcd.h** và **lcd.c** có thể sao chép từ project mẫu **Bai3_LCD_Button**. Ngoài ra còn có file **lcdfont.h** để lưu các font chữ có sẵn.

void lcd_init()

- **Mô tả:** Khởi tạo màn hình LCD. Gợi ý: gọi trong hàm **system_init** trong **main.c**.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void lcd_clear(uint16_t color)

- **Mô tả:** Tô toàn bộ màn hình với một màu RGB 16-bit.
- **Tham số:**
 - **color:** Giá trị màu RGB 16-bit (có thể chọn một số màu được định nghĩa trong **lcd.h**).
- **Giá trị trả về:** Không có.

```

1 #define WHITE      0xFFFF
2 #define BLACK     0x0000
3 #define BLUE      0x001F
4 #define BRED      0XF81F
5 #define GRED      0XFFE0
6 #define GBLUE     0X07FF
7 #define RED       0xF800
8 #define MAGENTA   0xF81F
9 #define GREEN     0x07E0
10 #define CYAN      0x7FFF
11 #define YELLOW    0xFFE0
12 #define BROWN    0XBC40
13 #define BRRED     0XFC07
14 #define GRAY      0X8430
15
16 #define DARKBLUE  0X01CF
17 #define LIGHTBLUE 0X7D7C
18 #define GRAYBLUE   0X5458
19
20
21 #define LIGHTGREEN 0X841F
22 #define LIGHTGRAY  0XE5B
23 #define LGRAY      0XC618
24
25 #define LGRAYBLUE  0XA651
26 #define LBBLUE     0X2B12

```

Program 3.4: Các màu đã được định nghĩa sẵn trong lcd.h

void lcd_fill(uint16_t xsta, uint16_t ysta, uint16_t xend, uint16_t yend, uint16_t color)

- **Mô tả:** Tô màu một vùng hình chữ nhật trên màn hình với màu RGB 16-bit.
- **Tham số:**
 - **xsta:** Tọa độ x của điểm đầu.
 - **ysta:** Tọa độ y của điểm đầu.
 - **xend:** Tọa độ x của điểm cuối.
 - **yend:** Tọa độ y của điểm cuối.

- **color:** Giá trị màu RGB 16-bit (có thể chọn một số màu được định nghĩa trong **lcd.h**).
- **Giá trị trả về:** Không có.

void lcd_show_char(uint16_t x, uint16_t y, uint8_t character, uint16_t fc, uint16_t bc, uint8_t sizey, uint8_t mode)

- **Mô tả:** Hiển thị ký tự trên LCD.
- **Tham số:**
 - **x:** Tọa độ x muốn in.
 - **y:** Tọa độ y muốn in.
 - **character:** Chữ cái muốn in.
 - **fc:** Màu chữ (16 bit RGB).
 - **bc:** Màu nền (16 bit RGB).
 - **sizey:** Kích thước chữ (có thể chọn các kích thước 12, 16, 24, 32 là chiều cao chữ tính theo pixel).
 - **mode:** Nhận các giá trị sau:
 - * **0:** Hiển thị cả chữ lẫn màu nền.
 - * **1:** Chỉ hiển thị chữ, không hiển thị màu nền.
- **Giá trị trả về:** Không có.

void lcd_show_string(uint16_t x, uint16_t y, char *str, uint16_t fc, uint16_t bc, uint8_t sizey, uint8_t mode)

- **Mô tả:** Hiển thị chuỗi kí tự trên LCD.
- **Tham số:**
 - **x:** Tọa độ x muốn in.
 - **y:** Tọa độ y muốn in.
 - **str:** Mảng chứa chuỗi các kí tự.
 - **fc:** Màu chữ (16 bit RGB).
 - **bc:** Màu nền (16 bit RGB).
 - **sizey:** Kích thước chữ (có thể chọn các kích thước 12, 16, 24, 32).
 - **mode:** Nhận các giá trị sau:

- * **0:** Hiển thị cả chữ lẫn màu nền.
- * **1:** Chỉ hiển thị chữ không hiển thị màu nền.
- **Giá trị trả về:** Không có.

void lcd_show_string_center(uint16_t x, uint16_t y, char *str, uint16_t fc, uint16_t bc, uint8_t sizey, uint8_t mode)

- **Mô tả:** Hiển thị chuỗi kí tự được căn giữa trên LCD.
- **Tham số:**
 - **x:** Tọa độ x muốn in.
 - **y:** Tọa độ y muốn in.
 - **str:** Mảng chứa chuỗi các kí tự.
 - **fc:** Màu chữ (16 bit RGB).
 - **bc:** Màu nền (16 bit RGB).
 - **sizey:** Kích thước chữ (có thể chọn các kích thước 12, 16, 24, 32).
 - **mode:** Nhận các giá trị sau:
 - * **0:** Hiển thị cả chữ lẫn màu nền.
 - * **1:** Chỉ hiển thị chữ không hiển thị màu nền.
- **Giá trị trả về:** Không có.

void lcd_show_int_num(uint16_t x, uint16_t y, uint16_t num, uint8_t len, uint16_t fc, uint16_t bc, uint8_t sizey)

- **Mô tả:** Hiển thị số nguyên trên LCD.
- **Tham số:**
 - **x:** Tọa độ x muốn in.
 - **y:** Tọa độ y muốn in.
 - **num:** Số nguyên muốn in.
 - **len:** Độ dài số muốn in.
 - **fc:** Màu số (16 bit RGB).
 - **bc:** Màu nền (16 bit RGB).
 - **sizey:** Kích thước chữ (có thể chọn các kích thước 12, 16, 24, 32).
- **Giá trị trả về:** Không có.

```
void lcd_show_float_num(uint16_t x, uint16_t y, float num, uint8_t len, uint16_t fc, uint16_t bc, uint8_t sizey)
```

- **Mô tả:** Hiển thị số thập phân trên LCD.
- **Tham số:**
 - **x:** Tọa độ x muôn in.
 - **y:** Tọa độ y muôn in.
 - **num:** Số thập phân muôn in.
 - **len:** Độ dài số muôn in.
 - **fc:** Màu số (16 bit RGB).
 - **bc:** Màu nền (16 bit RGB).
 - **sizey:** Kích thước chữ (có thể chọn các kích thước 12, 16, 24, 32).
- **Giá trị trả về:** Không có.

```
void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color)
```

- **Mô tả:** Hiển thị một điểm ảnh với màu sắc tùy chỉnh.
- **Tham số:**
 - **x:** Tọa độ x của điểm ảnh.
 - **y:** Tọa độ y của điểm ảnh.
 - **color:** Giá trị màu RGB 16-bit.
- **Giá trị trả về:** Không có.

```
void lcd_draw_line(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color)
```

- **Mô tả:** Hiển thị đường thẳng với màu sắc tùy chỉnh.
- **Tham số:**
 - **x1:** Tọa độ x của điểm đầu.
 - **y1:** Tọa độ y của điểm đầu.
 - **x2:** Tọa độ x của điểm cuối.
 - **y2:** Tọa độ y của điểm cuối.
 - **color:** Giá trị màu RGB 16-bit.

- **Giá trị trả về:** Không có.

void lcd_draw_rectangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color)

- **Mô tả:** Hiển thị hình chữ nhật với màu sắc tùy chỉnh trên LCD.

- **Tham số:**

- **x1:** Tọa độ x của điểm đầu.
- **y1:** Tọa độ y của điểm đầu.
- **x2:** Tọa độ x của điểm cuối.
- **y2:** Tọa độ y của điểm cuối.
- **color:** Giá trị màu RGB 16-bit.

- **Giá trị trả về:** Không có.

void lcd_draw_circle(int xc, int yc, uint16_t c, int r, int fill)

- **Mô tả:** Hiển thị hình tròn với màu sắc tùy chỉnh trên LCD.

- **Tham số:**

- **xc:** Tọa độ x của tâm hình tròn.
- **yc:** Tọa độ y của tâm hình tròn.
- **c:** Giá trị màu RGB 16-bit.
- **r:** Bán kính.
- **fill:** Lựa chọn có tô màu hình tròn hay không.
 - * **0:** Không tô màu hình tròn, chỉ tô màu đường tròn.
 - * **1:** Tô màu hình tròn.

- **Giá trị trả về:** Không có.

void lcd_show_picture(uint16_t x, uint16_t y, uint16_t length, uint16_t width, const uint8_t *pic[])

- **Mô tả:** Hiển thị một bức ảnh .

- **Tham số:**

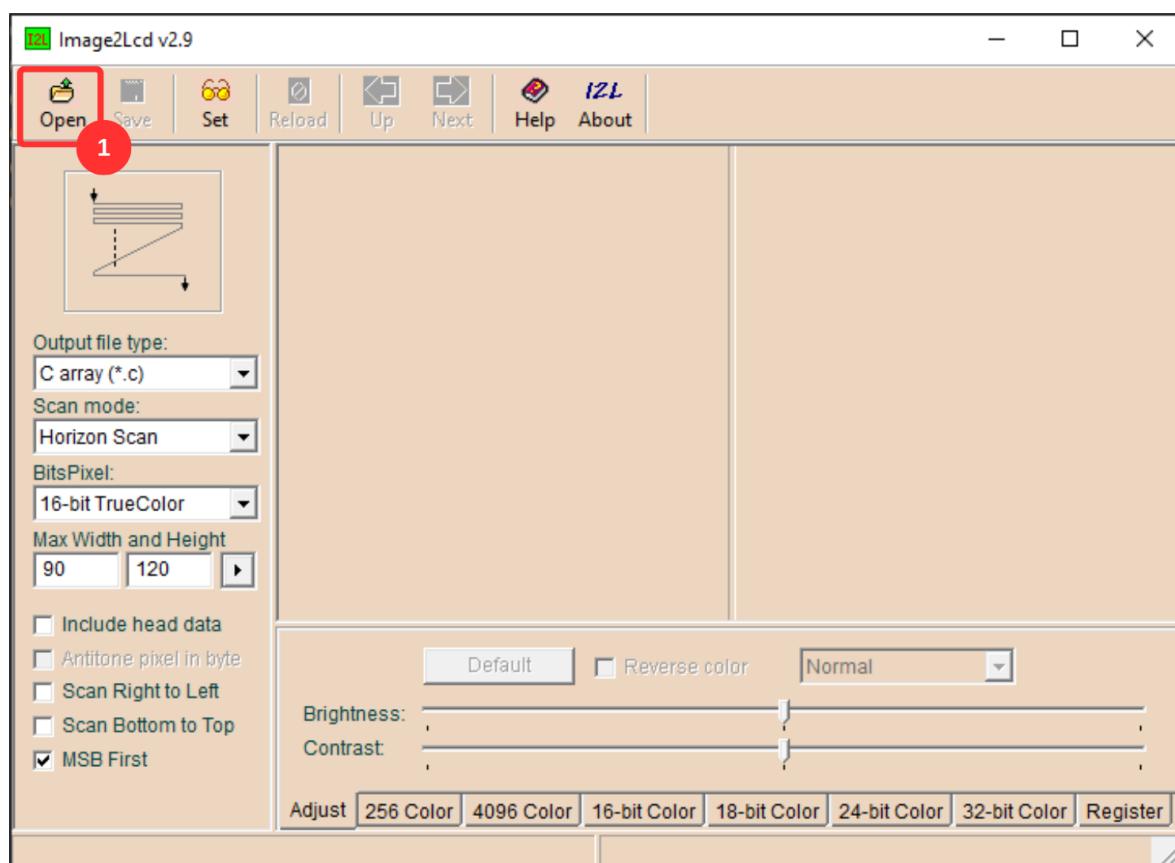
- **x:** Tọa độ x của điểm đầu bức ảnh.

- **y**: Tọa độ y của điểm đầu bức ảnh.
- **length**: Chiều dài bức ảnh.
- **width**: Chiều rộng bức ảnh.
- **pic**: Mảng chứa màu các điểm ảnh trong bức ảnh.

- **Giá trị trả về**: Không có.

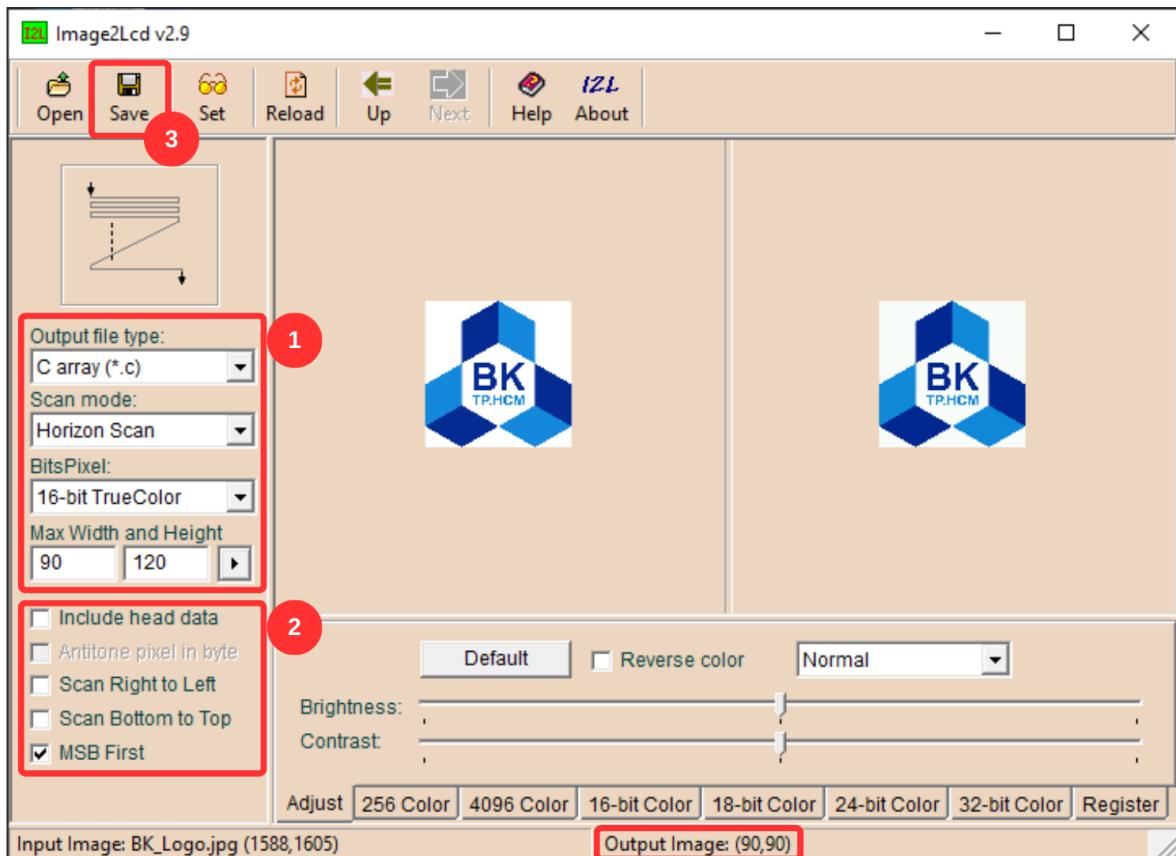
Để có thể hiển thị hình ảnh lên màn hình LCD, ta cần chuyển hình ảnh sang mảng dữ liệu có thể giao tiếp với màn hình LCD, mỗi điểm ảnh dùng 2 byte dữ liệu tương ứng với giá trị của màu RGB 16 bit. Để chuyển một hình ảnh thành dạng mảng nêu trên, chúng ta sẽ sử dụng sự trợ giúp của phần mềm Image2LCD.exe. Truy cập link tải phần mềm tại đây.

Chọn Open để chọn hình tải lên:



Hình 3.14: Sử dụng phần mềm Img2LCD.exe

Chỉnh các lựa chọn như hình 3.17. **Max Width and Height** là kích thước tối đa mà ta mong muốn chỉnh. Sau đó nhấn nút kế bên để chuyển đổi kích thước hình, **Output image** là kích thước chính xác của hình.



Hình 3.15: Sử dụng phần mềm Img2Lcd.exe

Sau khi chọn các thông số như hình 3.17. Để tạo mảng của hình ảnh, ta nhấn **Save** trên thanh công cụ. Sau khi save, ta sẽ có được file mảng dữ liệu như sau:

Hình 3.16: Dữ liệu hình ảnh sau khi được chuyển đổi bởi phần mềm Img2Lcd.exe

Để thuận tiện cho việc quản lý, chúng ta sẽ tạo thêm file **picture.h** và **picture.c** để lưu giữ dữ liệu hình ảnh. Mảng dữ liệu trên sẽ được sao chép vào file **picture.c**. Để có thể truy xuất mảng này ta cần khai báo file **picture.h** như sau:

```
1 #ifndef INC_PICTURE_H_
2 #define INC_PICTURE_H_
3
4 /* Variables */
5 extern const unsigned char gImageLogo [16200];
6 extern const unsigned char gImagePic [86400];
7
8#endif /* INC_PICTURE_H_ */
```

Program 3.5: picture.h

Sau đây là ví dụ sử dụng các hàm điều khiển nút nhấn và LCD.

```
1 /* Includes
2  -----
3   */
4 #include "main.h"
5 #include "spi.h"
6 #include "tim.h"
7 #include "gpio.h"
8 #include "fsmc.h"
9
10 /*
11  * Private includes
12  -----
13   */
14 /* USER CODE BEGIN Includes */
15 #include "software_timer.h"
16 #include "led.h"
17 #include "led_7seg.h"
18 #include "button.h"
19 #include "lcd.h"
20 #include "picture.h"
21
22 /* USER CODE END Includes */
23
24 /*
25  * Private typedef
26  -----
27   */
28
29 /* USER CODE BEGIN PTD */
```

```

20
21 /* USER CODE END PTD */
22
23 /* Private define
24 -----
25 */
24 /* USER CODE BEGIN PD */
25 /* USER CODE END PD */
26
27 /* Private macro
28 -----
29 */
28 /* USER CODE BEGIN PM */
29
30 /* USER CODE END PM */
31
32 /* Private variables
33 -----
34 */
34 /* USER CODE BEGIN PV */
35 int numberOfRowsPushButton;
36 /* USER CODE END PV */
37
38 /* Private function prototypes
39 -----
40 */
39 void SystemClock_Config(void);
40 /* USER CODE BEGIN PFP */
41 void init_system();
42
43 int IsButtonUp();
44 int IsButtonDown();
45 void TestButtonMatrix();
46 void TestLcd();
47 /* USER CODE END PFP */
48
49 /* Private user code
50 -----
51 */

```

```

50 /* USER CODE BEGIN 0 */
51
52 /* USER CODE END 0 */
53
54 /**
55 * @brief The application entry point.
56 * @retval int
57 */
58 int main(void) {
59     /* USER CODE BEGIN 1 */
60
61     /* USER CODE END 1 */
62
63     /* MCU Configuration
64     -----
65     */
66     /* Reset of all peripherals, Initializes the Flash
67     interface and the Systick. */
68     HAL_Init();
69
70     /* USER CODE BEGIN Init */
71
72     /* USER CODE END Init */
73
74     /* Configure the system clock */
75     SystemClock_Config();
76
77     /* USER CODE BEGIN SysInit */
78
79     /* Initialize all configured peripherals */
80     MX_GPIO_Init();
81     MX_TIM2_Init();
82     MX_TIM4_Init();
83     MX_SPI1_Init();
84     MX_FSMC_Init();
85     /* USER CODE BEGIN 2 */

```

```

86     init_system();
87
88     lcd_clear(WHITE);
89     TestLcd();
90     /* USER CODE END 2 */
91
92     /* Infinite loop */
93     /* USER CODE BEGIN WHILE */
94     while (1) {
95         while (!timer2_flag)
96             ;
97         timer2_flag = 0;
98
99         button_scan();
100        TestButtonMatrix();
101
102        /* USER CODE END WHILE */
103
104        /* USER CODE BEGIN 3 */
105    }
106    /* USER CODE END 3 */
107}
108
109//...
110
111/* USER CODE BEGIN 4 */
112void init_system() {
113    HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port, OUTPUT_Y0_Pin, 0);
114    HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port, OUTPUT_Y1_Pin, 0);
115    HAL_GPIO_WritePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin, 0);
116
117    button_init();
118    led_7seg_init();
119    lcd_init();
120
121    timer2_init();
122    timer2_set(50);
123
124    timer4_init();

```

```

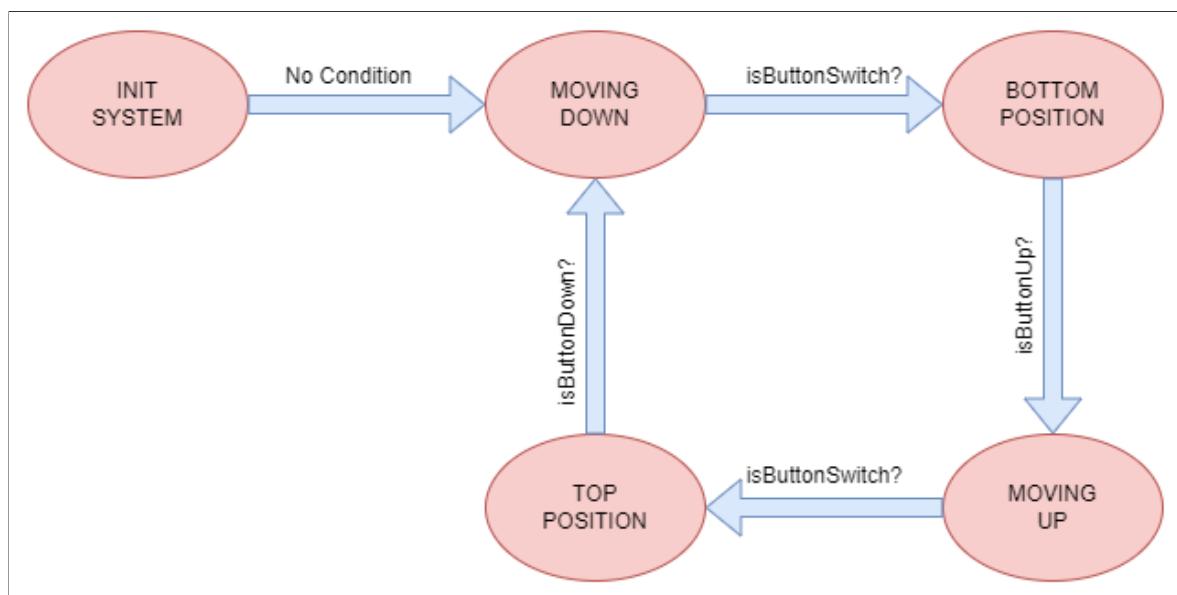
125     timer4_set(1);
126 }
127
128 int IsButtonUp(){
129     if(button_count[3] == 1 || (button_count[3] > 20 &&
130         button_count[3] % 2 == 0)){
131         return 1;
132     }
133     return 0;
134 }
135
136 int IsButtonDown(){
137     if(button_count[7] == 1){
138         return 1;
139     }
140     return 0;
141 }
142 void TestButtonMatrix() {
143     for (int i = 0; i < 16; i++) {
144         if (button_count[i] != 0) {
145             lcd_show_int_num(140, 105, i, 2, BRED, WHITE, 32);
146         }
147     }
148     if (IsButtonUp()) numberOfPushButton++;
149     if (IsButtonDown()) numberOfPushButton--;
150     lcd_show_int_num(40, 105, numberOfPushButton, 2, GREEN,
151                     WHITE, 32);
152 }
153 void TestLcd() {
154     lcd_fill(0, 0, 240, 20, BLUE);
155     lcd_show_string_center(0, 2, "Hello World !!!", RED, BLUE
156                           , 16, 1);
157     lcd_show_string(20, 30, "Test LCD Screen", WHITE, RED,
158                     24, 0);
159     lcd_draw_rectangle(20, 80, 100, 160, GREEN);
160     lcd_draw_circle(160, 120, BRED, 40, 0);
161     lcd_show_picture(80, 200, 90, 90, gImageLogo);
162 }
```

```
160 /* USER CODE END 4 */  
161 // ...
```

Program 3.6: main.c

5.3 Lập trình máy trạng thái

Máy trạng thái (FSM: Finite State Machine) là phương pháp thiết kế và lập trình ứng dụng giúp ta có thể kiểm soát được các trạng thái của hệ thống, khiến việc phát triển ứng dụng, kiểm thử, gỡ lỗi trở nên dễ dàng hơn. Một mô hình máy trạng thái gồm hai thành phần chính là **Trạng thái** và **Điều kiện chuyển trạng thái**.



Hình 3.17: Sử dụng phần mềm Img2Lcd.exe

Ta lấy ví dụ về một hệ thống kéo cờ. Một cột cờ được điều khiển bằng 2 nút nhấn lên và xuống. Khi lá cờ ở vị trí dưới cùng, ta nhấn nút lên thì lá cờ sẽ di chuyển lên, và khi tới đỉnh, một công tắc sẽ được kích hoạt, báo hiệu cho lá cờ dừng lại. Tương tự cho trường hợp ngược lại. Bài toán trên sẽ được mô hình hóa thành máy trạng thái sau:

Trong lập trình, máy trạng thái sẽ được hiện thực như sau:

```
1 //define the state  
2 #define INIT_SYSTEM 0  
3 #define MOVING_UP 1  
4 #define MOVING_DOWN 2  
5 #define STOP_MOVING 3
```

```

6 #define TOP_POSITION      4
7 #define BOTTOM_POSITION   5
8
9 uint8_t IsButtonUp();
10 uint8_t IsButtonDown();
11 uint8_t IsButtonStop();
12 uint8_t IsBottomSwitch();
13 uint8_t IsTopSwitch();
14 void FlagMovingDown();
15 void FlagMovingUp();
16 void FlagStopMoving();
17 void BaiTapFlag();

18
19 uint8_t statusFlag = INIT_SYSTEM;

20
21 int main(void) {
22     //...
23     /* USER CODE BEGIN 2 */
24     init_system();
25     /* USER CODE END 2 */

26
27     /* Infinite loop */
28     /* USER CODE BEGIN WHILE */
29     while (1) {
30         while (timer2_flag == 0)
31             ;
32         timer2_flag = 0;

33         // INPUT
34         button_scan();

35
36         // PROCESS
37         BaiTapFlag();

38         // OUTPPUT
39
40     /* USER CODE END WHILE */

41
42     /* USER CODE BEGIN 3 */

```

```

45 }
46 /* USER CODE END 3 */
47 }

48

49 void BaiTapFlag() {
50     switch (statusFlag) {
51         case INIT_SYSTEM:
52             lcd_clear(0x875c);
53             lcd_fill(0, 0, 240, 20, BLUE);
54             lcd_show_picture(80, 100, 90, 209, gImage_c_flag);

55
56         statusFlag = BOTTOM_POSITION;
57         break;
58     case MOVING_UP: // state
59         // State action
60         lcd_show_string_center(0, 2, "MOVING UP", WHITE,
61         BLUE, 16, 0);
62         FlagMovingUp();
63         // Condition
64         if (IsButtonDown()) {
65             statusFlag = MOVING_DOWN; // state transition
66         }

67         if (IsTopSwitch()) {
68             statusFlag = TOP_POSITION;
69         }
70         break;

71     case BOTTOM_POSITION:
72         //...
73         break;

74     case MOVING_DOWN:
75         //...
76         break;

77     case TOP_POSITION:
78         //...
79         break;

```

```
83
84     case STOP_MOVING:
85         // ...
86         break;
87
88     default:
89         statusFlag = INIT_SYSTEM;
90         break;
91     }
92 }
```

Program 3.7: Thực thi máy trạng thái

6 Bài tập và báo cáo

6.1 Bài tập 1

Nạp chương trình **LAB3_LCD_Button**, quan sát và tìm hiểu chương trình.

6.2 Bài tập 2

Xây dựng máy trạng thái và hiện thực hệ thống đèn giao thông ở ngã tư với các tính năng:

- Ứng dụng sẽ có 6 đèn tín hiệu giao thông tương ứng với 2 tuyến đường (2 đèn xanh, 2 đèn đỏ, 2 đèn vàng). Các đèn giao thông sẽ được mô phỏng trên màn hình LCD.
- Ứng dụng có 3 nút nhấn dùng để:
 - Chọn chế độ.
 - Điều chỉnh chu kỳ các đèn.
 - Xác nhận thông số đã được điều chỉnh.
- Ứng dụng có ít nhất 4 chế độ được điều chỉnh bởi nút nhấn thứ nhất. Chế độ 1 là chế độ **NORMAL**, chế độ 2,3,4 là chế độ **MODIFICATION**. Nút nhấn thứ nhất sẽ được dùng để chuyển đổi giữa các chế độ. Chế độ sẽ được thay đổi từ 1 tới 4 và quay về 1.

Chế độ 1 - NORMAL:

- Đèn giao thông chạy bình thường.

Chế độ 2 - Điều chỉnh chu kỳ đèn đỏ:

- Tín hiệu đèn đỏ chớp tắt với tần số 2Hz
- Hiển thị số được điều chỉnh trên màn hình LCD.
- Hiển thị chế độ đang hoạt động lên màn hình LCD.
- Nút nhấn thứ 2 dùng để tăng giá trị chu kỳ của đèn đỏ.
- Giá trị của chu kỳ đèn đỏ nằm trong khoảng 1-99.
- Nút nhấn thứ 3 dùng để xác nhận giá trị được chọn.

Chế độ 3 - Điều chỉnh chu kỳ đèn xanh:

Tương tự chế độ 2.

Chế độ 4 - Điều chỉnh chu kỳ đèn vàng:

Tương tự chế độ 2.