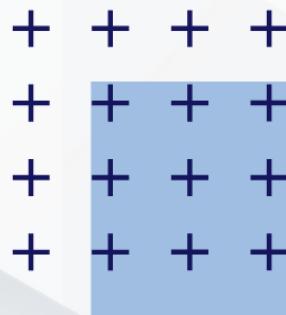
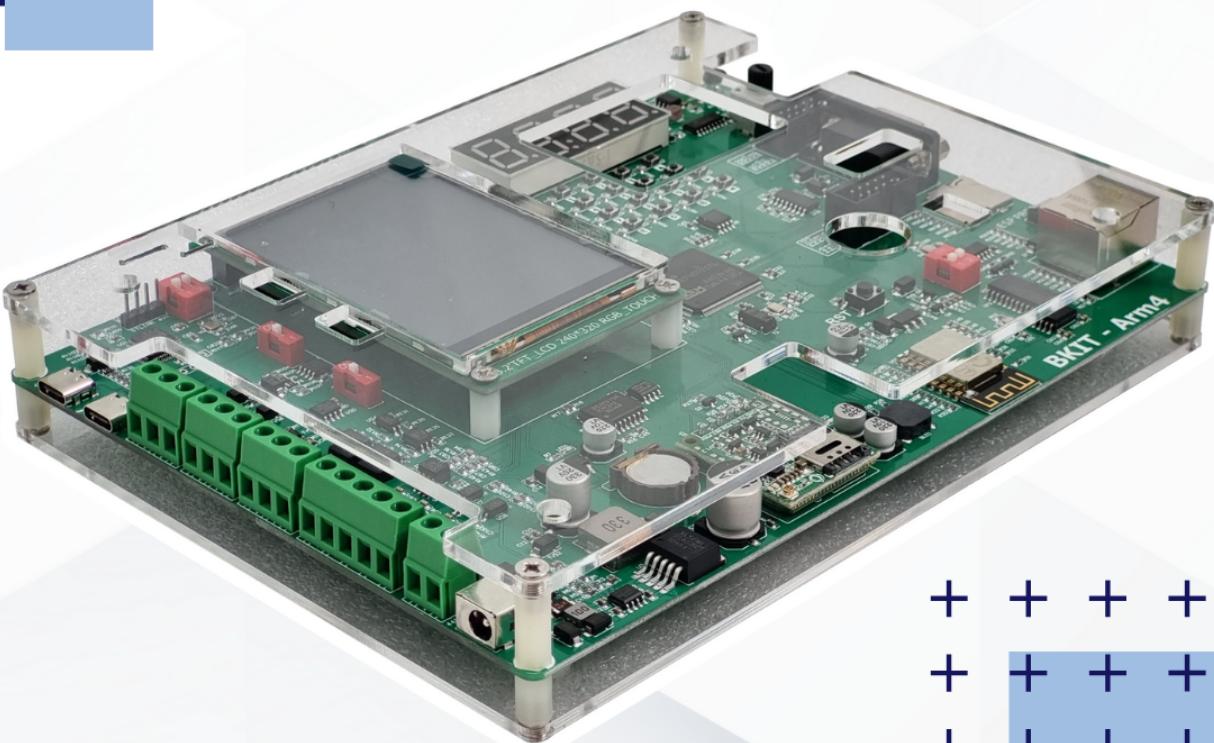


# KIT THÍ NGHIỆM BKIT

# ARM4



---

# Mục lục

---

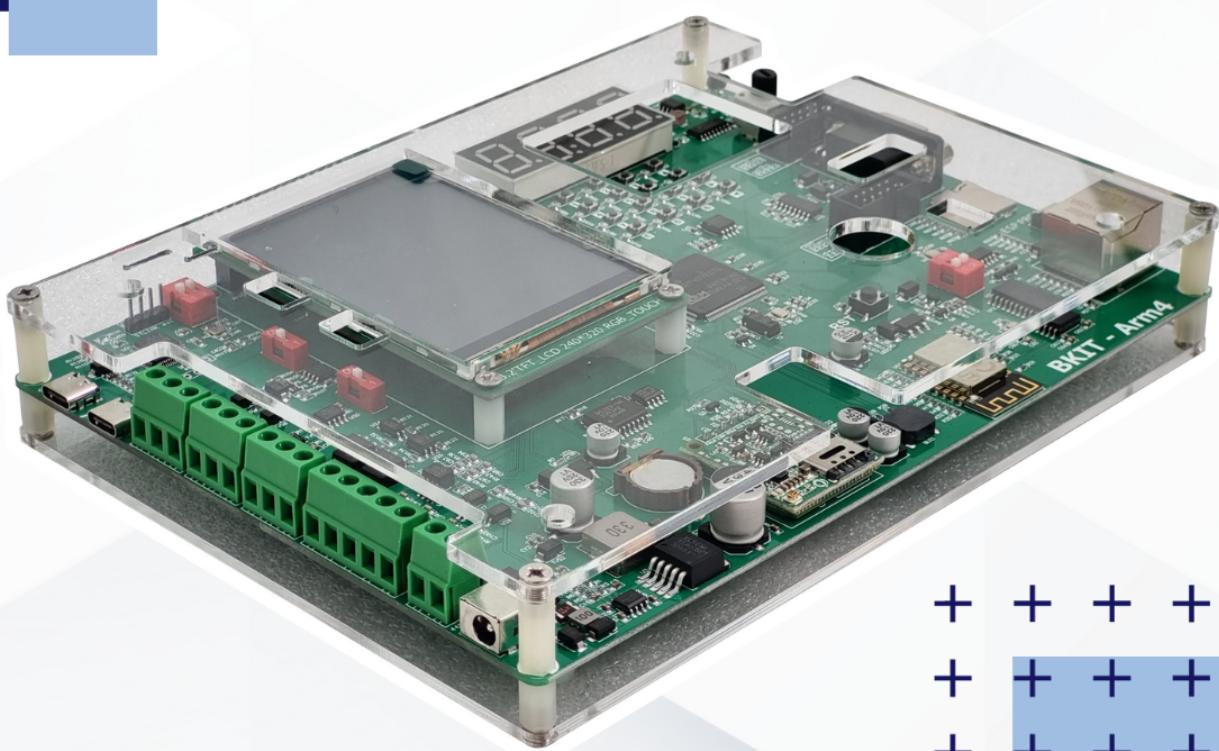
<b>Chapter 3. LCD and Button matrix</b>	<b>3</b>
1    Mục tiêu . . . . .	4
2    Giới thiệu . . . . .	4
3    Cơ sở lý thuyết . . . . .	5
3.1    Sự cần thiết của điện trở kéo lên của nút nhấn . . . . .	5
3.2    Chống rung nút nhấn . . . . .	6
4    Hướng dẫn cấu hình . . . . .	8
4.1    Cấu hình nút nhấn . . . . .	8
4.2    Cấu hình LCD . . . . .	10
5    Hướng dẫn lập trình . . . . .	13
5.1    Sử dụng thư viện: button.h . . . . .	13
5.2    Sử dụng thư viện: lcd.h . . . . .	16
6    Bài tập và báo cáo . . . . .	27

# 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.

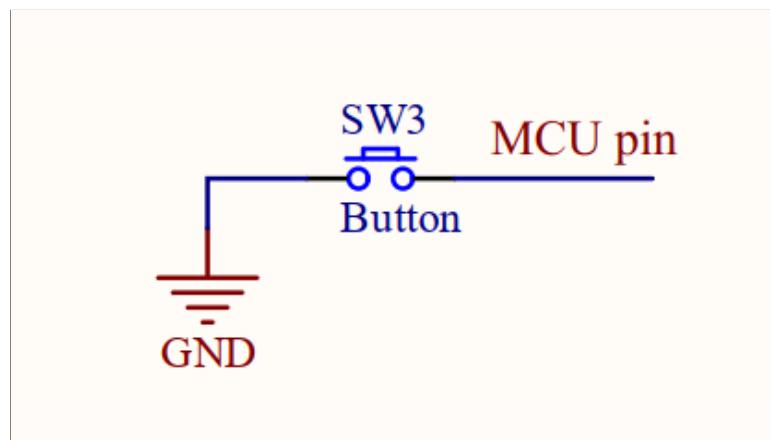
- Kích hoạt một máy bơm khi công tắc được bật.

**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

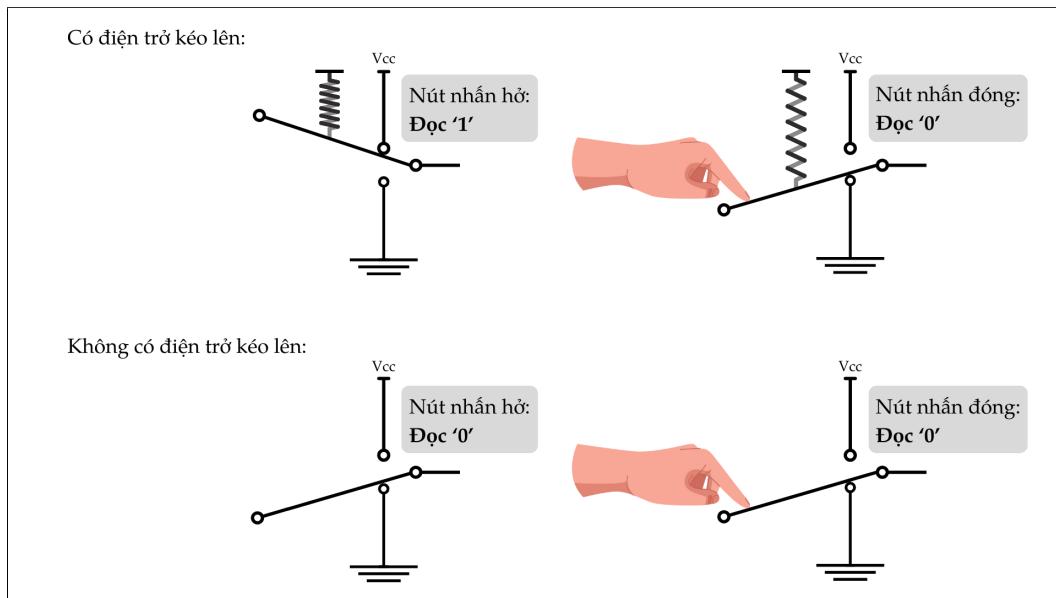


Hình 3.1: Kết nối nút nhấn với MCU

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

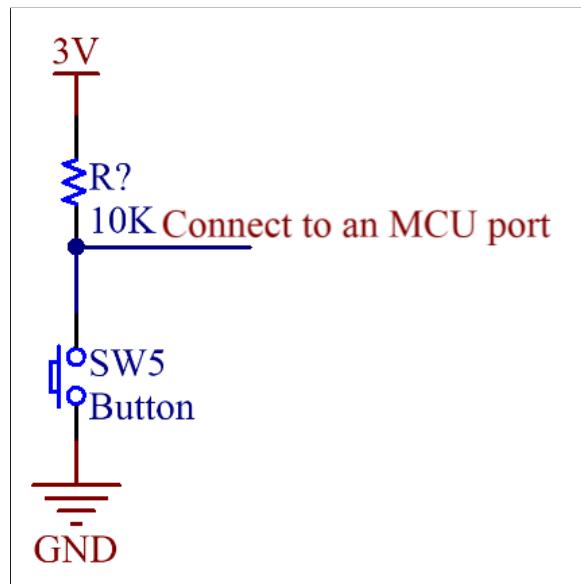
- Khi nút nhấn mở (không được nhấn), một điện trở nội bộ trong MCU sẽ "kéo" chân MCU lên nguồn (3.3V với STM32F407). Nếu chúng ta đọc giá trị của chân MCU, ta sẽ được giá trị "1".
- Khi nút nhấn đóng (được nhấn), chân MCU sẽ được nối với GND và sẽ có điện áp là 0V. Nếu ta đọc giá trị chân MCU, ta sẽ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

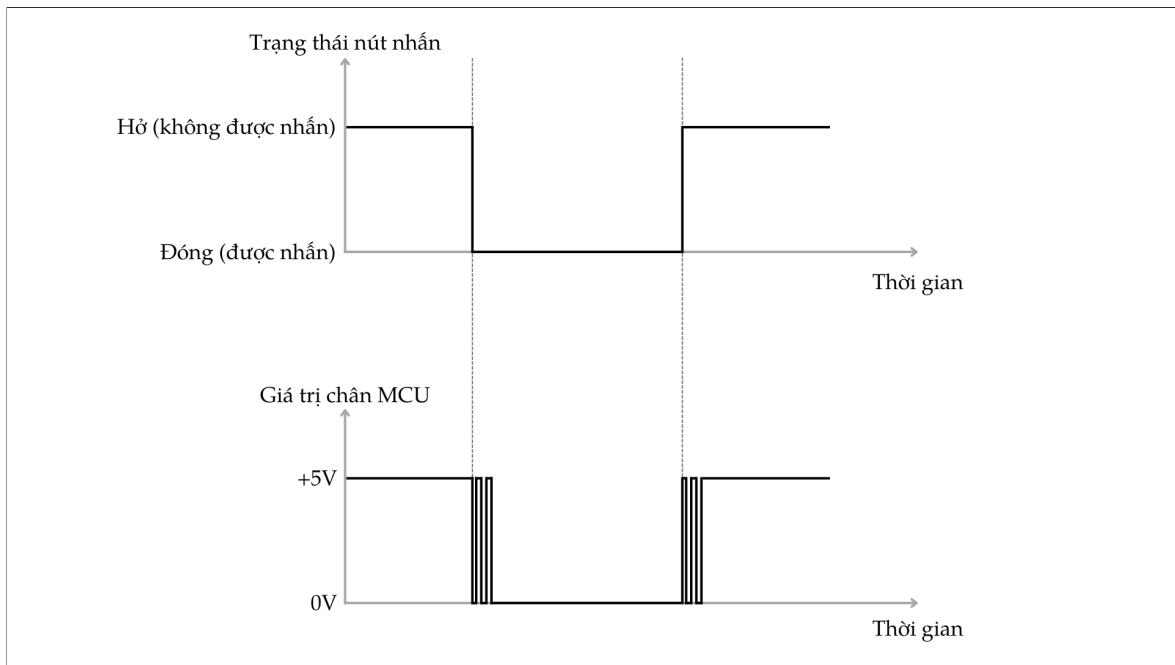
Vậy nên khi kết nối nút nhấn hoặc công tắc với MCU thì ta nên sử dụng một điện trở kéo lên như hình 3.2.



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*

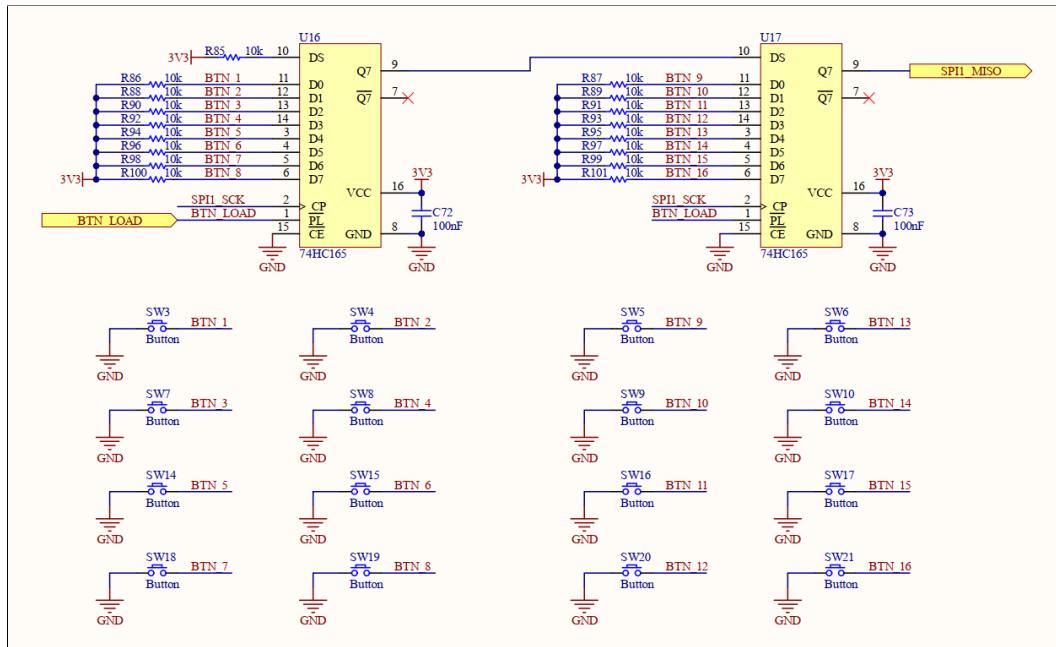
Có nhiều phương pháp chống rung nút nhấn sử dụng phần cứng hoặc phần mềm. Một cách chống rung bằng phần mềm là đọc nút nhấn sau mỗi chu kì (ví dụ: 10ms) và lưu lại trạng thái trước đó của nút nhấn. Một tín hiệu nút nhấn sẽ được chấp nhận khi không có sự thay đổi giữa tín hiệu hiện tại và tín hiệu trước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



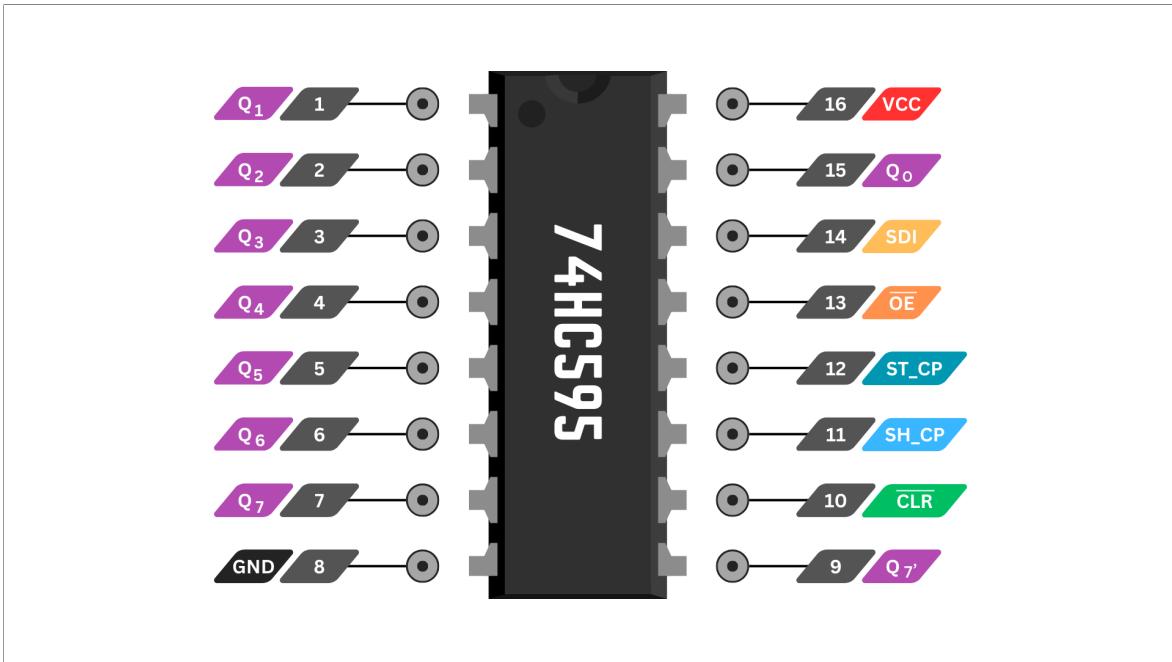
Hình 3.5: Kết nối ma trận phím trong Kit thí nghiệm

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:

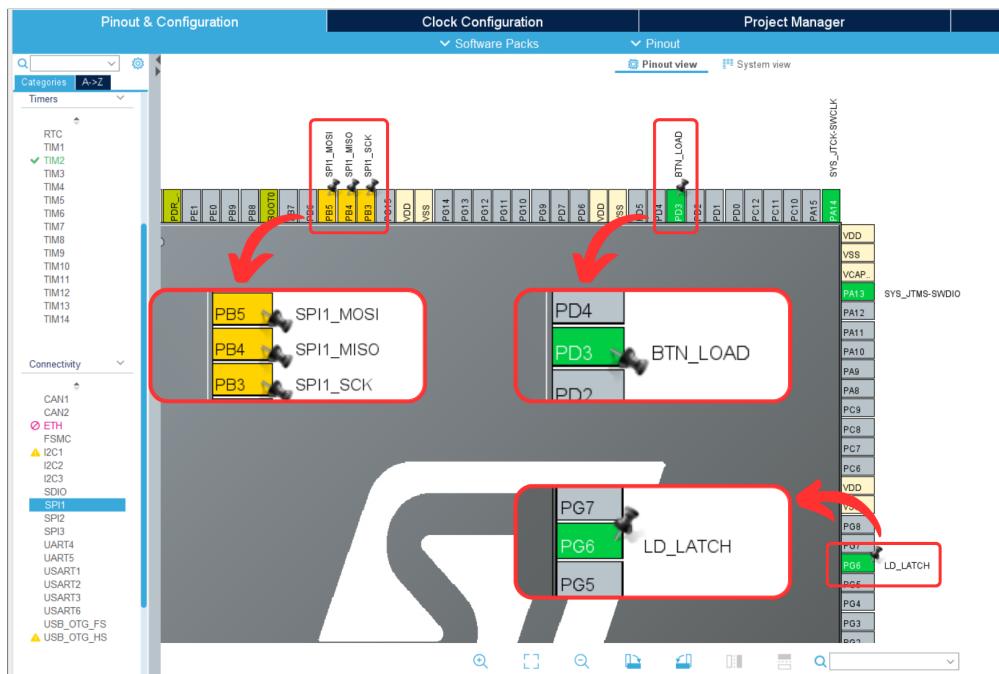
- Khi tín hiệu chân **PL** ở mức thấp, giá trị từ các ngõ vào D0-D7 sẽ được nạp song song vào thanh ghi trong IC.
- Khi tín hiệu chân **PL** ở mức cao và phát hiện cạnh lên ở chân **CP**, các giá trị bên trong thanh ghi sẽ được dịch và tín hiệu **Q7** sẽ lần lượt thể hiện các giá

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.



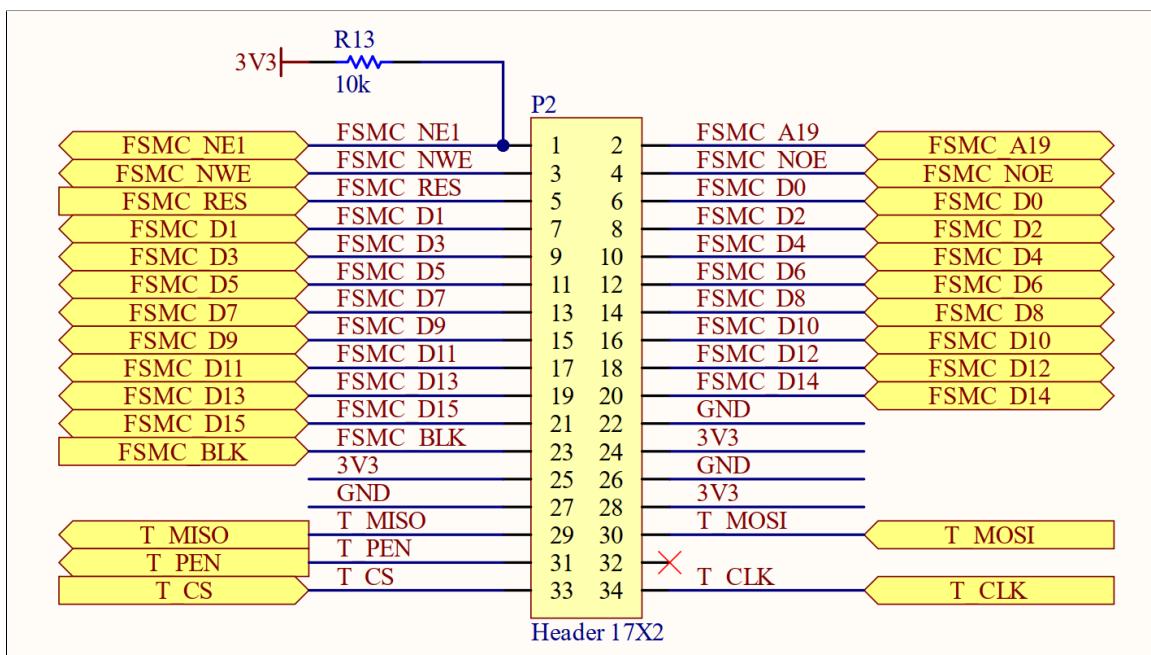
Hình 3.7: Các chân cần config

## 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:

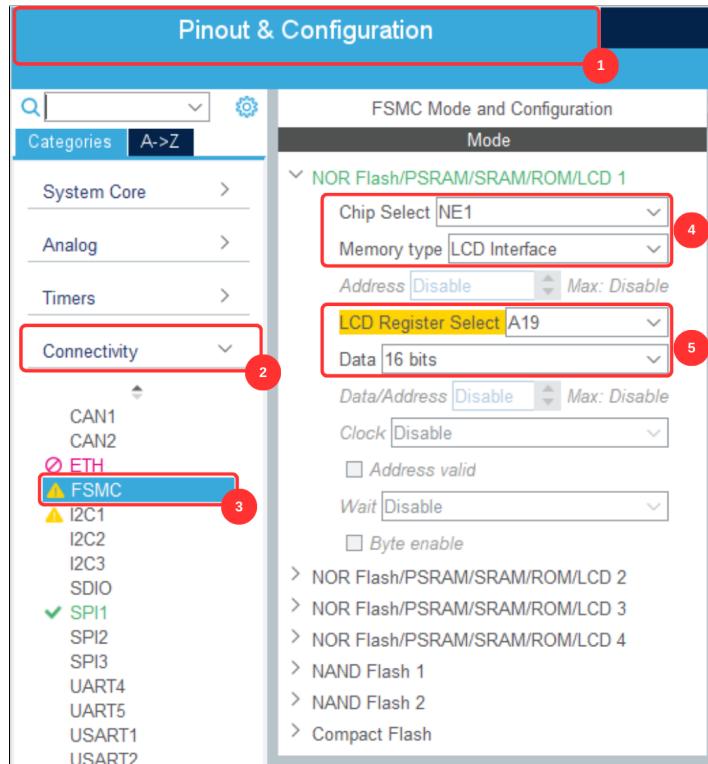
- Độ phân giải: 320x240 pixels.
- Chế độ màu: RGB 16 bit.
- Số bit dữ liệu giao tiếp với driver: 16.

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:

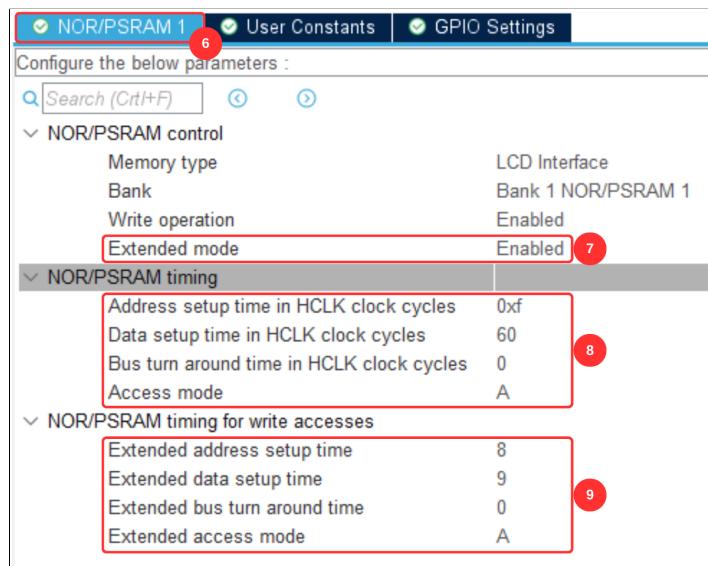


Hình 3.8: Schematic của LCD

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

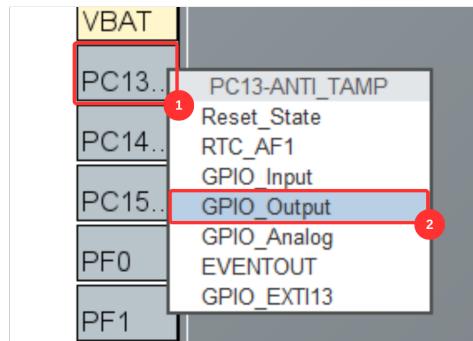


Hình 3.9: Cấu hình FSMC

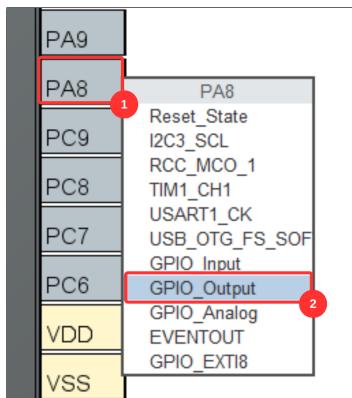


Hình 3.10: Cấu hình FSMC (tiếp theo)

Ngoài ra, ta cần cấu hình 2 chân: **FSMC\_RES** (reset LCD) và **FSMC\_BLK** (đèn nền LCD):

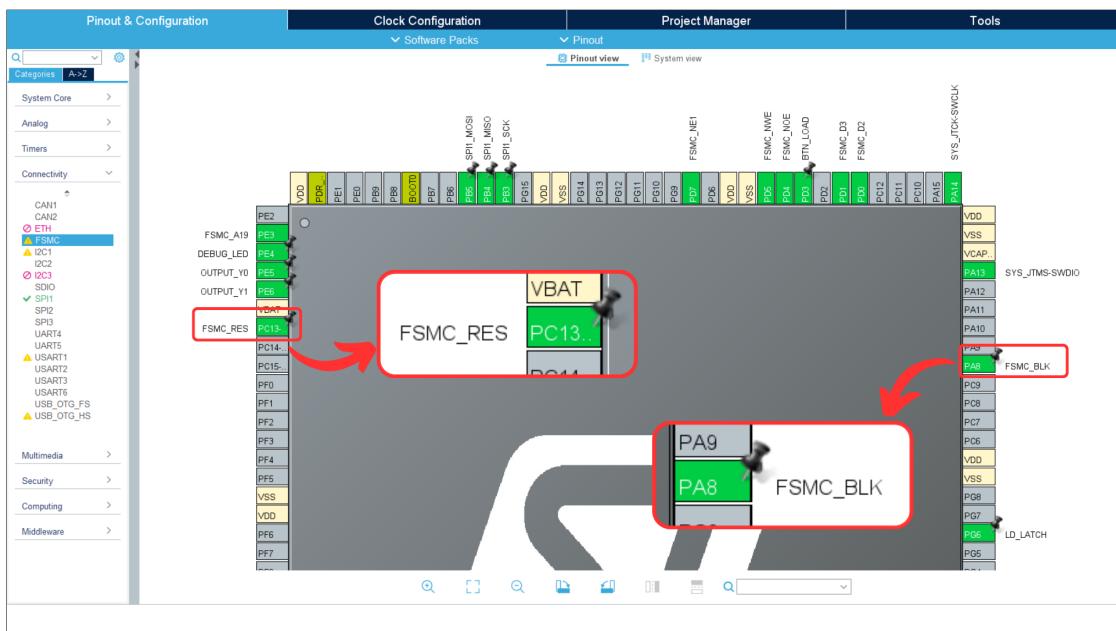


Hình 3.11: Cấu hình **FSMC\_RES**



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      /* USER CODE BEGIN 3 */
28  }
29  /* USER CODE END 3 */
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\_ShowChar(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\_ShowStr(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\_StrCenter(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\_ShowIntNum(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_ShowFloatNum1(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_DrawPoint(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_DrawLine(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\_DrawRectangle(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\_DrawCircle(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\_ShowPicture(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:**

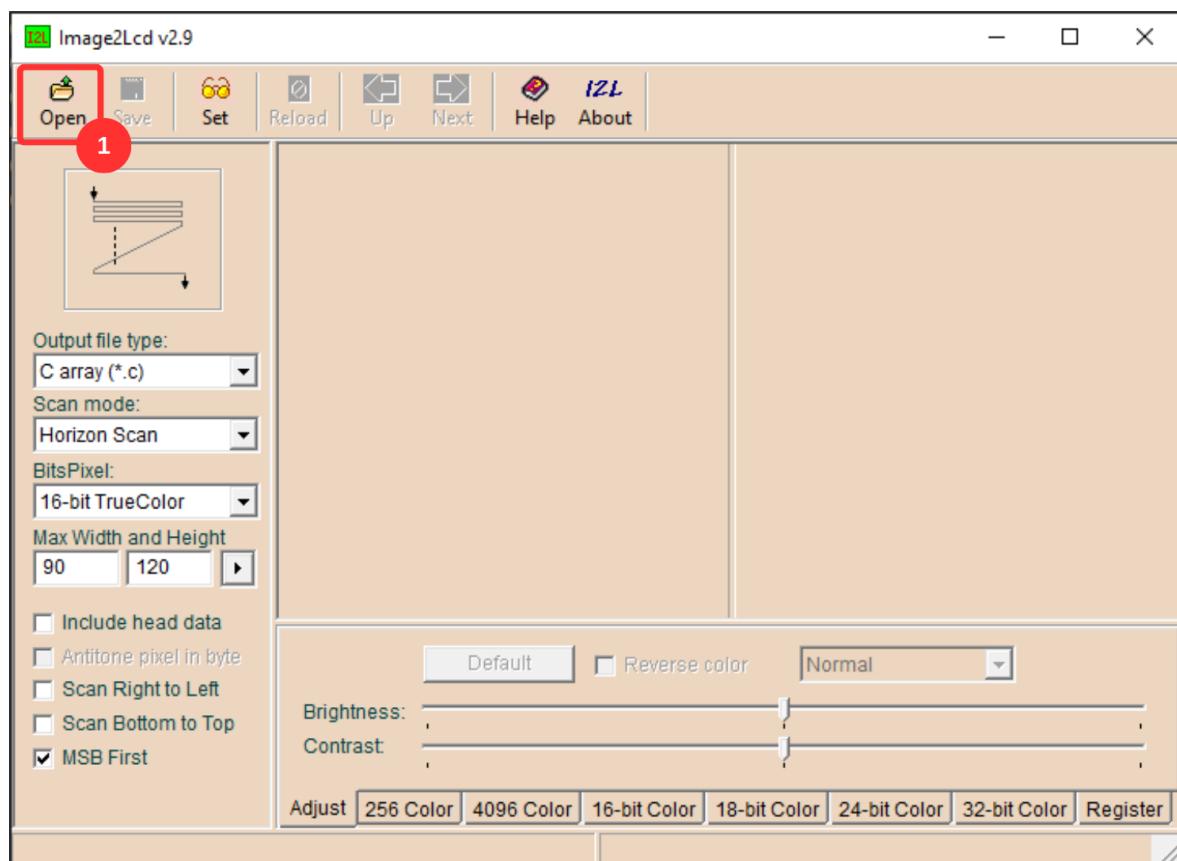
- **x:** Tọa độ x của điểm đầu bức ảnh.

- **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.

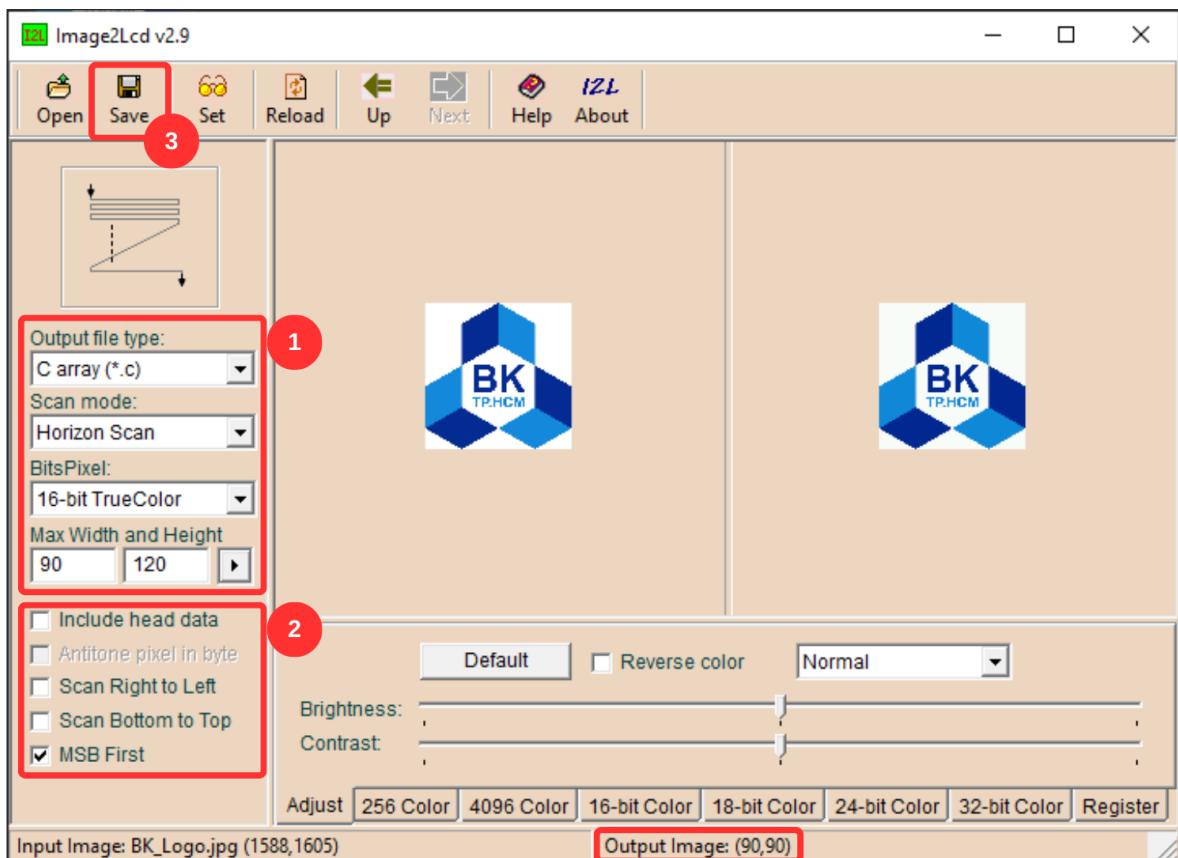
Để có thể hiển thị hình ảnh lên màn hình LCD, ta cần chuyển hình ảnh sang mảng dữ liệu có thể giao tiếp với màn hình LCD, mỗi điểm ảnh dùng 2 byte dữ liệu tương ứng với giá trị của màu RGB 16 bit. Để chuyển một hình ảnh thành dạng mảng nêu trên, chúng ta sẽ sử dụng sự trợ giúp của phần mềm Image2LCD.exe. 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 3.15. **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 3.15. 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 extern const unsigned char gImage_logo[16200];
5
6#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 /* Private includes
11  *
12  *-----*
13  */
14 /* USER CODE BEGIN Includes */
15 #include "software_timer.h"
16 #include "led_7seg.h"
17 #include "button.h"
18 #include "lcd.h"
19 #include "picture.h"
20 /* USER CODE END Includes */
21 // ...
22 void SystemClock_Config(void);
23 /* USER CODE BEGIN PFP */
24 void system_init();
25 void test_LedDebug();
26 void test_LedY0();
27 void test_LedY1();
28 void test_7seg();
29 void test_button();
```

```

24 void test_lcd();
25 /* USER CODE END PFP */
26 // ...
27 int main(void)
28 {
29     // ...
30     /* USER CODE BEGIN 2 */
31     system_init();
32     lcd_Clear(WHITE);
33     test_lcd();
34     /* USER CODE END 2 */
35     /* Infinite loop */
36     /* USER CODE BEGIN WHILE */
37     while (1)
38     {
39         while(!flag_timer2);
40         flag_timer2 = 0;
41         button_Scan();
42         test_button();
43         /* USER CODE END WHILE */
44         /* USER CODE BEGIN 3 */
45     }
46     /* USER CODE END 3 */
47 }
48 // ...
49 /* USER CODE BEGIN 4 */
50 void system_init(){
51     HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port, OUTPUT_Y0_Pin,
52     0);
53     HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port, OUTPUT_Y1_Pin,
54     0);
55     HAL_GPIO_WritePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin,
56     0);
57     timer_init();
58     led7_init();
59     button_init();
60     lcd_init();
61     setTimer2(50);
62 }

```

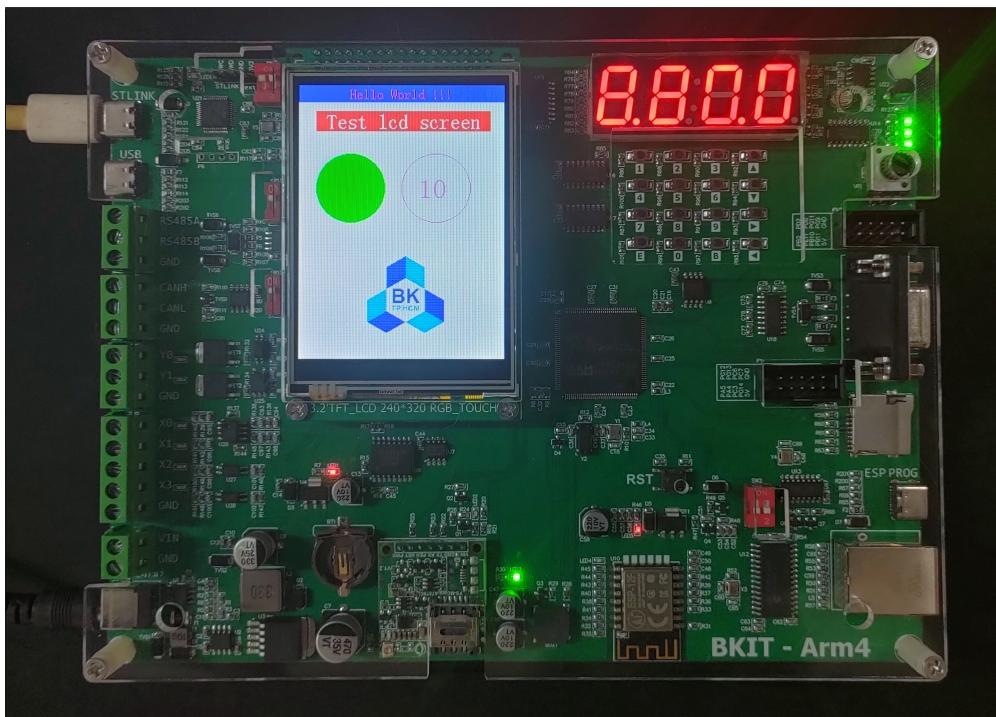
```

60 void test_button(){
61     for(int i = 0; i < 16; i++){
62         if(button_count[i] == 1){
63             lcd_ShowIntNum(140, 105, i, 2, BRED, WHITE, 32);
64         }
65     }
66 }
67 void test_lcd(){
68     lcd_Fill(0, 0, 240, 20, BLUE);
69     lcd_StrCenter(0, 2, "Hello World !!!", RED, BLUE, 16,
10);
70     lcd_ShowStr(20, 30, "Test lcd screen", WHITE, RED, 24,
0);
71     lcd_DrawCircle(60, 120, GREEN, 40, 1);
72     lcd_DrawCircle(160, 120, BRED, 40, 0);
73     lcd_ShowPicture(80, 200, 90, 90, gImage_logo);
74 }
75 /* USER CODE END 4 */
76 // ...

```

Program 3.6: main.c

Hình ảnh kết quả trên Kit thí nghiệm:



Hình 3.17: Kết quả hiện thực trên Kit thí nghiệm

## 6 Bài tập và báo cáo

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 - Chính sửa 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 - Chính sửa chu kỳ đèn xanh:** Tương tự chế độ 2.

### **Chế độ 4 - Chính sửa chu kỳ đèn vàng:** Tương tự chế độ 2.