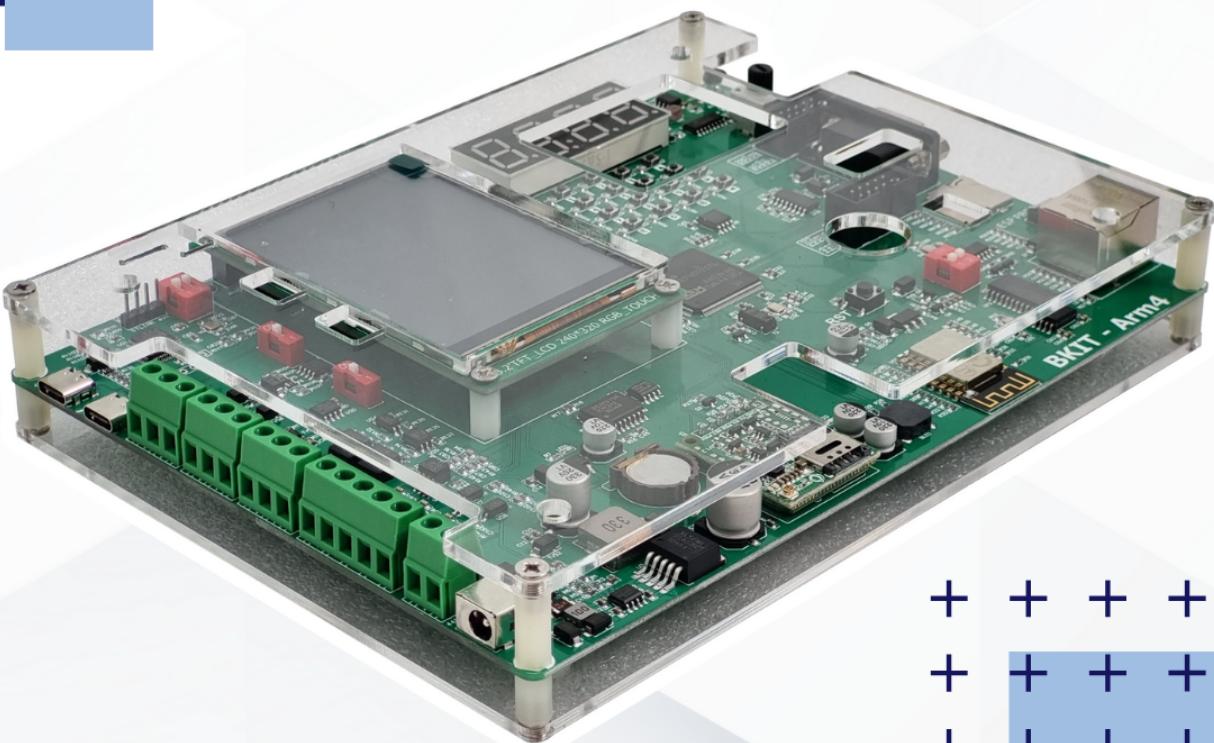


KIT THÍ NGHIỆM BKIT

ARM4

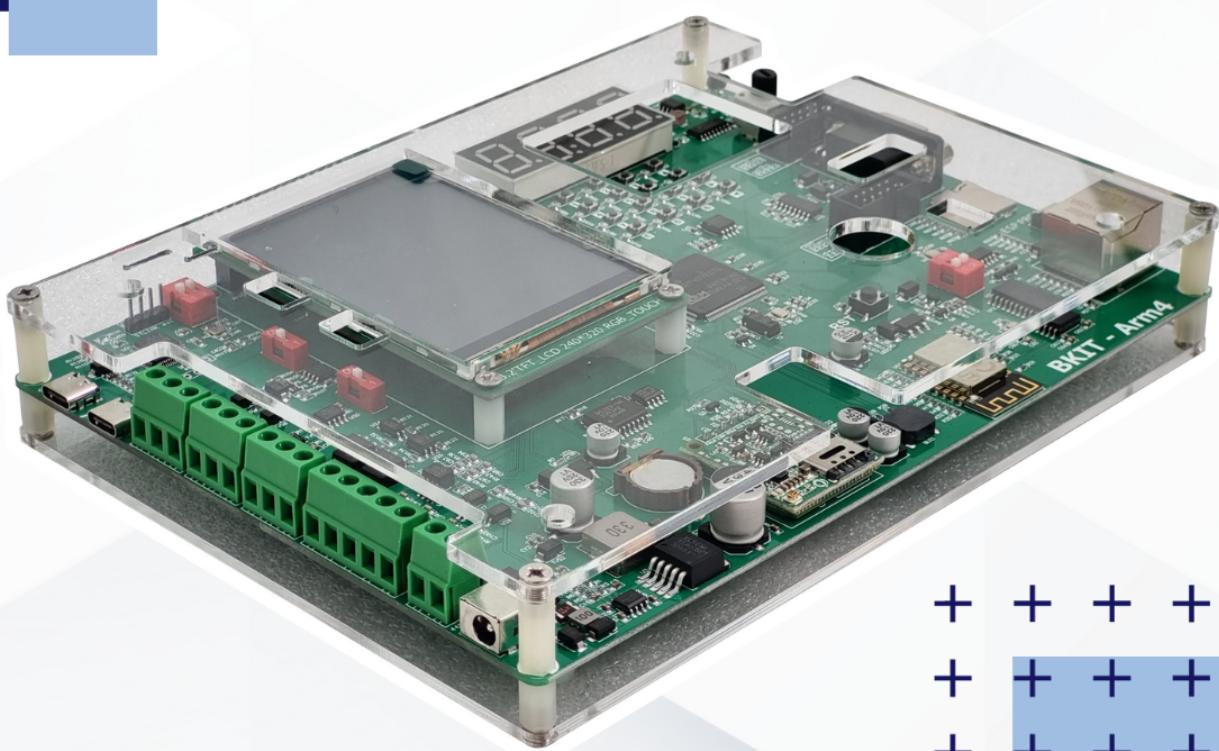


Mục lục

| | |
|--|----------|
| Chapter 7. LCD Touch | 3 |
| 1 Mục tiêu | 4 |
| 2 Giới thiệu | 4 |
| 3 Cơ sở lý thuyết | 4 |
| 3.1 Touch sensor | 4 |
| 3.1.1 Cảm biến điện trở | 4 |
| 3.1.2 Cảm biến điện dung | 5 |
| 3.2 Bộ nhớ EEPROM AT24C | 6 |
| 3.2.1 Quá trình hình thành EEPROM | 6 |
| 3.2.2 EEPROM AT24C | 7 |
| 4 Hướng dẫn cấu hình | 7 |
| 4.1 Cấu hình Touch: | 7 |
| 5 Hướng dẫn lập trình | 10 |
| 5.1 Thư viện touch.h | 10 |
| 5.2 Thư viện software_timer.c | 11 |
| 5.3 Thư viện at24c.h | 11 |
| 6 Bài tập và báo cáo | 19 |

CHƯƠNG 7

LCD Touch



1 Mục tiêu

- Tìm hiểu về 2 loại cảm biến touch.
- Tìm hiểu về bộ nhớ EEPROM AT24C.
- Hướng dẫn cách sử dụng màn hình LCD touch, từ cơ bản như hiển thị thông tin đến các tương tác cảm ứng.

2 Giới thiệu

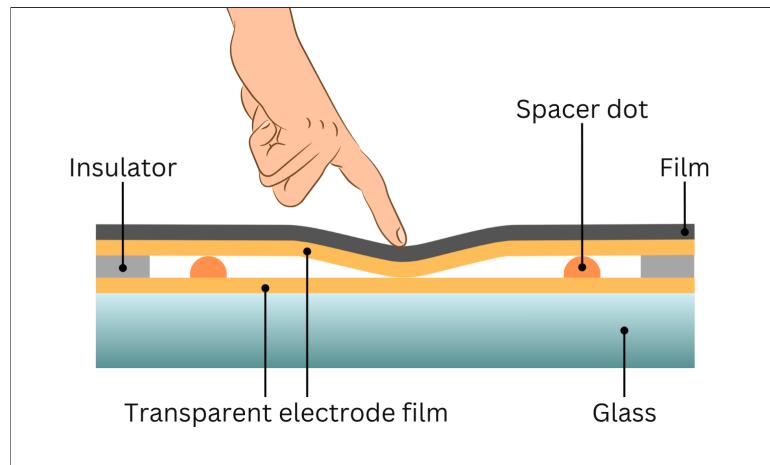
Touch sensor là một thành phần điện tử quan trọng, được sử dụng để phát hiện sự tương tác với môi trường xung quanh thông qua việc chạm hoặc áp lực. Cảm biến chạm được sử dụng rộng rãi từ điện thoại thông minh đến thiết bị gia dụng, giúp tạo ra giao diện người dùng thân thiện và hiệu quả. Cảm biến chạm hoạt động dựa trên nhiều nguyên tắc khác nhau, bao gồm cảm biến điện trở, cảm biến điện dung, cảm biến siêu âm và cảm biến hồng ngoại. Trong tương tác với màn hình, người ta thường quan tâm đến 2 loại cảm biến là cảm biến điện trở và cảm biến điện dung. Trong bài lab này, chúng ta sẽ tìm hiểu về LCD touch sử dụng cảm biến điện trở và bộ nhớ EEPROM AT24C512.

3 Cơ sở lý thuyết

3.1 Touch sensor

3.1.1 Cảm biến điện trở

- Cảm biến chạm điện trở hoạt động bằng cách sử dụng lớp mỏng của vật liệu dẫn điện nằm giữa hai lớp cách điện. Khi có chạm, sự tiếp xúc giữa hai lớp tạo ra một đường dẫn điện, làm thay đổi điện trở của hệ thống.

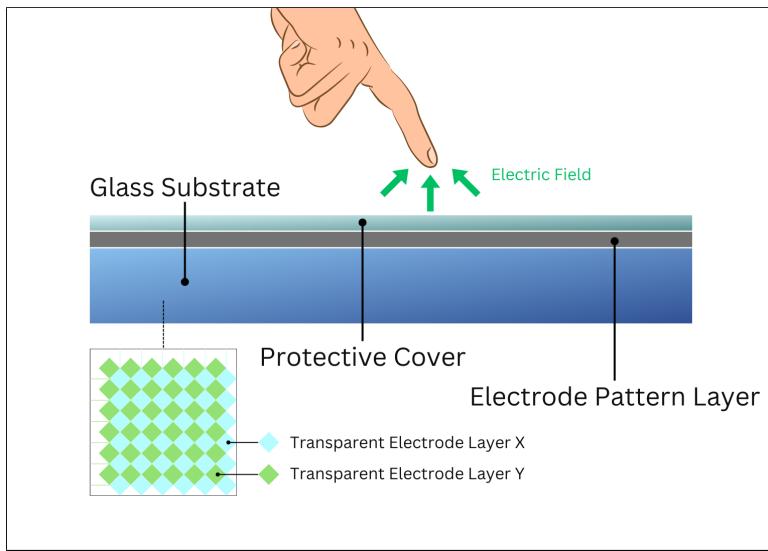


Hình 7.1: Minh họa cảm biến điện trở

- Cảm ứng điện trở xác định vị trí chạm thông qua việc đo điện trở tại các điểm khác nhau trên bề mặt, từ đó có thể xác định được vị trí chạm.
- Cảm biến điện trở thường được sử dụng trong các ứng dụng yêu cầu xác định chính xác vị trí chạm, như màn hình chạm trong các thiết bị đầu cuối công nghiệp.
- So với một số loại cảm biến chạm khác, cảm biến điện trở thường có chi phí thấp hơn.
- Bên cạnh đó, một số mô hình cảm biến điện trở có khả năng chịu ấn (cảm biến áp lực), nghĩa là chúng có thể phản ứng không chỉ với chạm mà còn với áp lực áp dụng lên bề mặt.

3.1.2 Cảm biến điện dung

- Cảm biến điện dung sử dụng nguyên tắc của điện dung để đo lường sự thay đổi trong dung tích của hệ thống khi có sự chạm. Bề mặt cảm biến có thể được làm từ các vật liệu dẫn điện và được cách điện từ môi trường bên ngoài. Nghĩa là khi có một vật dẫn điện (ngón tay hoặc bút cảm ứng) chạm vào cảm biến thì điện dung giữa hai bản kim loại giảm đi, điều đó giống như ngón tay đã “đánh cắp” một lượng điện trường.



Hình 7.2: Minh họa cảm biến điện dung

- Cảm ứng điện dung nhạy cảm và phản ứng nhanh. Cảm biến điện dung thường rất nhạy và có thể phản ứng với những chạm nhẹ từ ngón tay. Phản ứng nhanh giúp tạo ra trải nghiệm người dùng mượt mà và thú vị.
- Cảm ứng điện dung được sử dụng khá phổ biến: màn hình cảm ứng điện thoại thông minh, máy tính bảng; bảng điều khiển trong các thiết bị gia dụng như tủ lạnh thông minh.
- Có thể tạo ra cảm biến điện dung dựa trên dấu vết, nơi một tia xung được gửi qua bề mặt và các thay đổi trong dung tích được đo.

3.2 Bộ nhớ EEPROM AT24C

3.2.1 Quá trình hình thành EEPROM

Từ những năm 1940, bộ nhớ ROM (Read-Only Memory) xuất hiện trong máy tính đầu tiên, lưu trữ dữ liệu không thay đổi. Điều này có nghĩa là dữ liệu không thể thay đổi sau khi được ghi trong quá trình sản xuất. Những năm 1970, EPROM (Erasable Programmable Read-Only Memory) được giới thiệu có khả năng lập trình lại để giải quyết hạn chế của ROM, nhưng cần quá trình xóa bằng tia UV, điều đó làm cho việc ghi lại khá là khó khăn. Để dễ dàng ghi lại hơn, EEPROM (Electrically Erasable Programmable Read-Only Memory) được phát triển, loại bỏ sự cần thiết của tia UV và cho phép xóa và lập trình điện tử mà không cần tháo rời từ mạch điện tử. Từ lúc được giới thiệu cho đến thời điểm hiện tại, EEPROM và các công nghệ lưu trữ tương tự ngày càng phổ biến, liên tục cải tiến để đáp ứng nhu cầu ngày càng cao của công nghệ.

3.2.2 EEPROM AT24C

EEPROM AT24C là một dòng non-volatile memory được sản xuất bởi nhiều nhà sản xuất khác nhau, trong đó tập trung nhiều nhất là dòng sản phẩm của Microchip Technology. Trên Kit thí nghiệm này, chúng ta sẽ làm việc cụ thể với EEPROM AT24C.

- EEPROM AT24C được thiết kế để lưu trữ dữ liệu non-volatile, có khả năng xóa và lập trình lại mà không cần sự tháo rời từ mạch điện tử.
- Có nhiều phiên bản với các dung lượng khác nhau, từ vài kilobits đến vài nghìn kilobits, như AT24C32 (32 Kbits) hoặc AT24C256 (256 Kbits).
- AT24C hỗ trợ giao tiếp chuẩn I2C (Inter-Integrated Circuit), làm cho quá trình kết nối với các vi điều khiển và các thiết bị khác trở nên thuận tiện.
- Thường có thể vận hành ở điện áp thấp như 1.7V, giúp tiết kiệm năng lượng và phù hợp với nhiều ứng dụng di động.
- Sử dụng rộng rãi trong các ứng dụng yêu cầu lưu trữ cài đặt hệ thống, dữ liệu cảm biến, và thông tin cấu hình trong các thiết bị điện tử.

4 Hướng dẫn cấu hình

4.1 Cấu hình Touch:

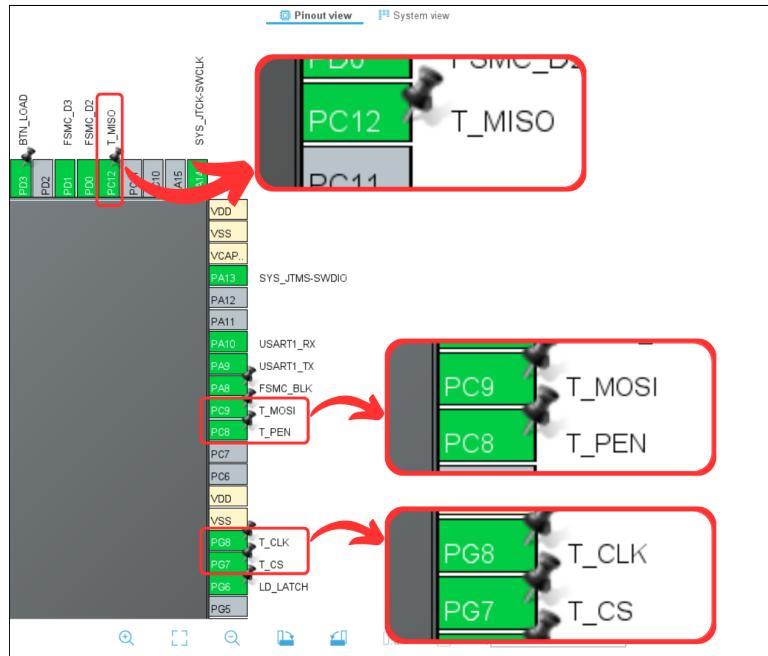
Trong Kit thí nghiệm, vi điều khiển sẽ cần phải giao tiếp với IC Touch Controller XPT2046 trên bo mạch LCD để quản lý các tác vụ về màn hình cảm ứng. IC trên sẽ giao tiếp với vi điều khiển thông qua SPI và một chân khác để báo hiệu màn hình được chạm. Các chân được cấu hình như bảng sau:

| Ngoại vi | Chân vi điều khiển | Chức năng |
|----------|--------------------|-------------|
| T_CLK | PG8 | GPIO Output |
| T_CS | PG7 | GPIO Output |
| T_MISO | PC12 | GPIO Input |
| T_MOSI | PC9 | GPIO Output |
| T_PEN | PC8 | GPIO Input |

Bảng 7.1: Kết nối ngoại vi

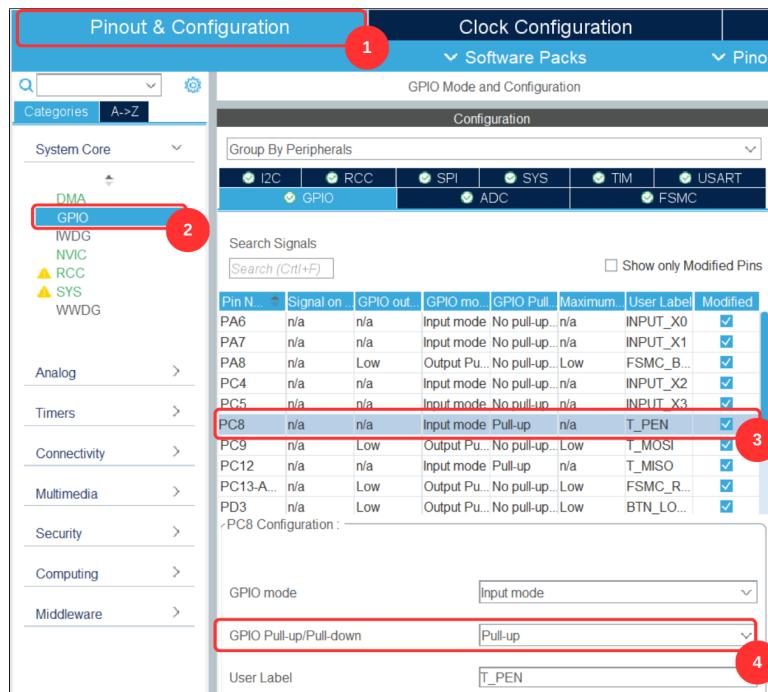
Các chân được dùng để giao tiếp SPI trong phần trên không có chức năng SPI sẵn trên vi điều khiển. Vậy nên ta sẽ config các chân trên là GPIO và sẽ lập trình các

chân này theo quy tắc của SPI. Cách làm này thường được gọi là SPI mềm do được điều khiển bởi phần mềm.



Hình 7.3: Config màn hình cảm ứng

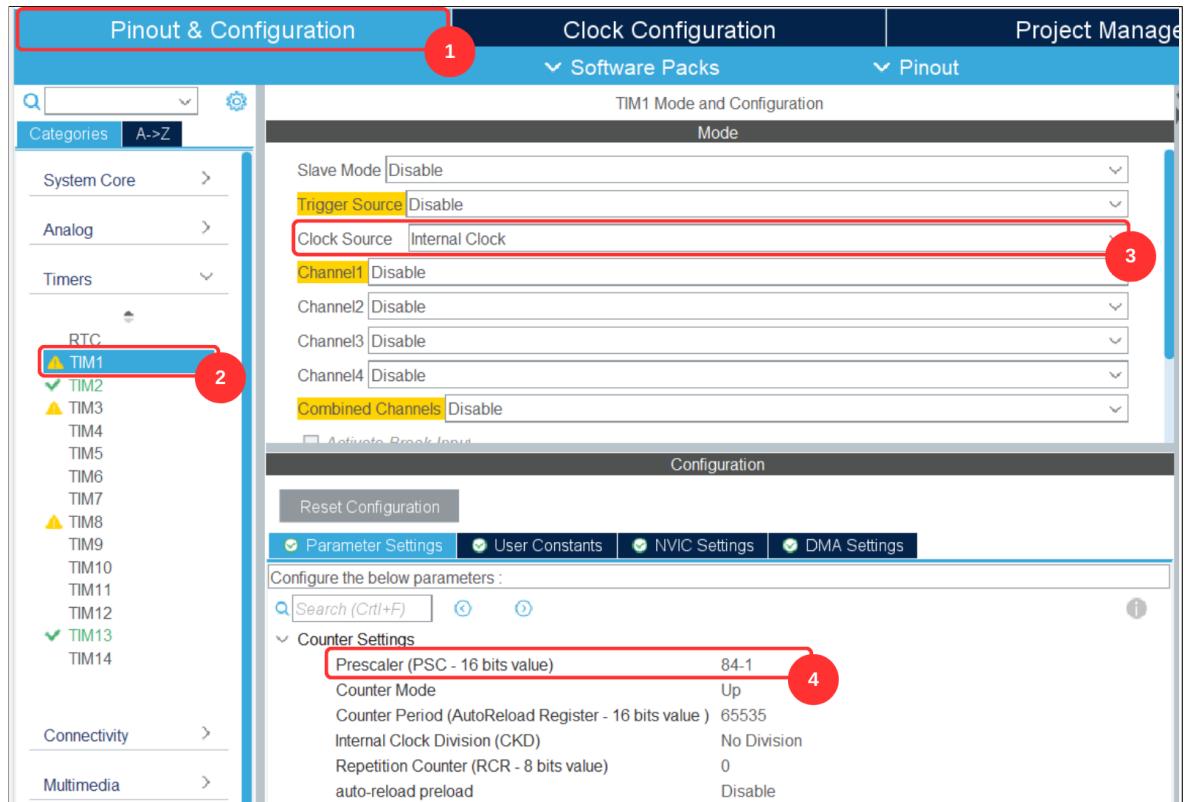
Lưu ý: Các chân Input sẽ phải được config điện trở kéo lên.



Hình 7.4: Config điện trở kéo lên

Để lập trình điều khiển SPI bằng phần mềm, ta sẽ cần hàm để tạo ra độ trễ với đơn vị micro giây. Vì vậy ta sẽ phải cấu hình thêm timer TIM1 cho công việc này. Để có

độ chia là micro giây, tức tần số là 1MHz, với xung clock đầu vào là 84 MHz (xem lại Lab 2) thì ta sẽ phải cấu hình **Prescaler** là **84-1**.



Hình 7.5: Config TIM1

5 Hướng dẫn lập trình

5.1 Thư viện touch.h

Các file touch.h và touch.c có thể được sao chép từ project mẫu **Bai7_TouchScreen**.

void touch_init()

- **Mô tả:** Khởi tạo màn hình cảm ứng.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void touch_Adjust()

- **Mô tả:** Điều chỉnh thông số màn hình cảm ứng.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void touch_Scan()

- **Mô tả:** Đọc màn hình cảm ứng.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

uint8_t touch_IsTouch()

- **Mô tả:** Kiểm tra xem màn hình cảm ứng có được chạm không.
- **Tham số:** Không có.
- **Giá trị trả về:**
 - 1: Được chạm.
 - 0: Không được chạm.

uint8_t touch_GetX()

- **Mô tả:** Lấy tọa độ X điểm được chạm.

- **Tham số:** Không có.
- **Giá trị trả về:** Tọa độ X.

uint8_t touch_GetY0

- **Mô tả:** Lấy tọa độ Y điểm được chạm.
- **Tham số:** Không có.
- **Giá trị trả về:** Tọa độ Y.

5.2 Thư viện software_timer.c

Trong thư viện này sẽ được thêm các hàm hỗ trợ cho việc tạo delay.

void timer_EnableDelayUs()

- **Mô tả:** Cho phép tạo delay micro giây.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void delay_us(uint16_t us)

- **Mô tả:** Tạo delay micro giây.
- **Tham số:**
 - **us:** thời gian delay.
- **Giá trị trả về:** Không có.

5.3 Thư viện at24c.h

Các file at24c.h và at24c.c có thể được sao chép từ project mẫu **Bai7_TouchScreen**.

void at24c_init()

- **Mô tả:** Khởi tạo kết nối với eeprom.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void at24c_Read(uint16_t ReadAddr,uint8_t *pBuffer,uint16_t NumToRead)

- **Mô tả:** Đọc nhiều byte giá trị từ các thanh ghi liên tiếp.
- **Tham số:**
 - **ReadAddr:** Địa chỉ thanh ghi đầu tiên muốn đọc.
 - **pBuffer:** Mảng giá trị đọc được.
 - **NumToRead:** Số byte muốn đọc.
- **Giá trị trả về:** Không.

void at24c_Write(uint16_t WriteAddr,uint8_t *pBuffer,uint16_t NumToWrite)

- **Mô tả:** Ghi nhiều byte giá trị vào các thanh ghi liên tiếp.
- **Tham số:**
 - **WriteAddr:** Địa chỉ thanh ghi đầu tiên muốn ghi.
 - **pBuffer:** Mảng giá trị muốn ghi.
 - **NumToWrite:** Số byte muốn ghi.
- **Giá trị trả về:** Không.

Sau đây sẽ là phần hướng dẫn xây dựng một ứng dụng đơn giản để kiểm tra hoạt động của màn hình cảm ứng. Ta sẽ xây dựng một ứng dụng cho phép vẽ trên bảng đen và xóa bảng bằng cách chạm vào nút trên màn hình. Ứng dụng trên sẽ được xây dựng bởi máy trạng thái sau:

```
1 #define INIT 0
2 #define DRAW 1
3 #define CLEAR 2
4
5 int draw_Status = INIT;
6
7 uint8_t isButtonClear(){
8     if(!touch_IsTouched()) return 0;
9     return touch_GetX() > 60 && touch_GetX() < 180 &&
10    touch_GetY() > 10 && touch_GetY() < 60;
11}
12
13 void touchProcess(){
14     switch (draw_Status) {
```

```

14     case INIT:
15         // display blue button
16         lcd_Fill(60, 10, 180, 60, GBLUE);
17         lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
18         draw_Status = DRAW;
19         break;
20     case DRAW:
21         if(isButtonClear()){
22             draw_Status = CLEAR;
23             // clear board
24             lcd_Fill(0, 60, 240, 320, BLACK);
25             // display green button
26             lcd_Fill(60, 10, 180, 60, GREEN);
27             lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
28         }
29         break;
30     case CLEAR:
31         if(!touch_IsTouched()) draw_Status = INIT;
32         break;
33     default:
34         break;
35     }
36 }
```

Program 7.1: Hiện thực máy trạng thái

Máy trạng thái trên sẽ được xử lí mỗi 50ms. Tuy nhiên, để sử dụng touch sensor, ta cần phải gọi hàm khởi tạo **touch_init**. Để đọc dữ liệu từ màn hình cảm ứng, ta sẽ kiểm tra xem màn hình có được chạm không và đọc giá trị vị trí được chạm. Để tăng độ nhạy của cảm biến, việc kiểm tra và đọc giá trị của màn hình cảm ứng sẽ được thực hiện liên tục. Điều này dẫn đến một số thay đổi cấu trúc trong vòng lặp chính. Lưu ý: hàm **touch_Adjust** chỉ cần gọi khi sử dụng màn hình cảm ứng lần đầu tiên. Mục tiêu của hàm này là để căn chỉnh, lưu lại các giá trị cảm ứng đọc được ở 4 góc màn hình vào eeprom. Cụ thể việc đọc màn hình cảm ứng được thể hiện dưới đây:

```

1 system_init();
2 touch_Adjust();
3 lcd_Clear(BLACK);
4 while (1)
5 {
```

```

6   //scan touch screen
7   touch_Scan();
8   //check if touch screen is touched
9   if(touch_IsTouched() && draw_Status == DRAW){
10       //draw a point at the touch position
11       lcd_DrawPoint(touch_GetX(), touch_GetY(), RED);
12   }
13   // 50ms task
14   if(flag_timer2 == 1){
15       flag_timer2 = 0;
16       touchProcess();
17       test_LedDebug();
18   }
19
20   /* USER CODE END WHILE */
21
22   /* USER CODE BEGIN 3 */
23 }
```

Program 7.2: Vòng lặp vô tận

Khi hiện thực xong chương trình, kết quả là ta có thể vẽ được trên LCD (có thể sử dụng vật cứng như đầu bút tương tác chạm) và có thể xóa màn hình đã vẽ bằng cách chạm vào nút **CLEAR** trên màn hình. Nếu có hàm **touch_Adjust** được gọi, thì ban đầu một màn hình sẽ hiện ra yêu cầu chạm vào các điểm được đánh dấu để lưu lại các thông số căn chỉnh. Dưới đây là toàn bộ source code của hàm file main.c.

```

1 // ...
2 /* USER CODE BEGIN Includes */
3 #include "software_timer.h"
4 #include "led_7seg.h"
5 #include "button.h"
6 #include "lcd.h"
7 #include "picture.h"
8 #include "ds3231.h"
9 #include "sensor.h"
10 #include "buzzer.h"
11 #include "touch.h"
12 /* USER CODE END Includes */
13
```

```

14 // Define statuses
15
16 #define INIT 0
17 #define DRAW 1
18 #define CLEAR 2
19
20 int draw_Status = INIT;
21 // ...
22 /* USER CODE BEGIN PFP */
23 void system_init();
24 void test_LedDebug();
25 void touchProcess();
26 /* USER CODE END PFP */
27
28 int main(void)
29 {
30
31 /* USER CODE BEGIN 2 */
32 system_init();
33 touch_Adjust();
34 lcd_Clear(BLACK);
35 /* USER CODE END 2 */
36
37 /* Infinite loop */
38 /* USER CODE BEGIN WHILE */
39
40 while (1)
41 {
42     //scan touch screen
43     touch_Scan();
44     //check if touch screen is touched
45     if(touch_IsTouched() && draw_Status == DRAW){
46         //draw a point at the touch position
47         lcd_DrawPoint(touch_GetX(), touch_GetY(), RED);
48     }
49     // 50ms task
50     if(flag_timer2 == 1){
51         flag_timer2 = 0;
52         touchProcess();

```

```

53     test_LedDebug();
54 }
55
56 /* USER CODE END WHILE */
57
58 /* USER CODE BEGIN 3 */
59 }
60 /* USER CODE END 3 */
61 }
62
63 // ...
64
65 /* USER CODE BEGIN 4 */
66 void system_init(){
67     timer_init();
68     button_init();
69     lcd_init();
70     touch_init();
71     setTimer2(50);
72 }
73
74 uint8_t count_led_debug = 0;
75
76 void test_LedDebug(){
77     count_led_debug = (count_led_debug + 1)%20;
78     if(count_led_debug == 0){
79         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port , DEBUG_LED_Pin);
80     }
81 }
82
83 uint8_t isButtonClear(){
84     if(!touch_IsTouched()) return 0;
85     return touch_GetX() > 60 && touch_GetX() < 180 &&
86     touch_GetY() > 10 && touch_GetY() < 60;
87 }
88
89 void touchProcess(){
90     switch (draw_Status) {
91         case INIT:

```

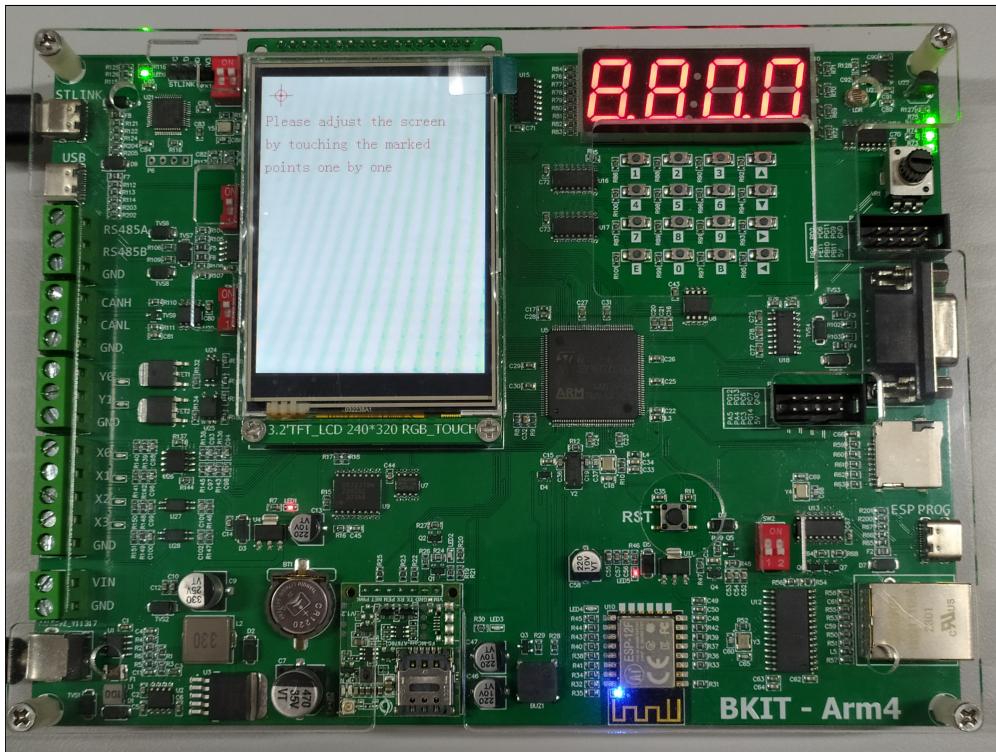
```

91                     // display blue button
92         lcd_Fill(60, 10, 180, 60, GBLUE);
93         lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
94         draw_Status = DRAW;
95         break;
96     case DRAW:
97         if(isButtonClear()){
98             draw_Status = CLEAR;
99                 // clear board
100            lcd_Fill(0, 60, 240, 320, BLACK);
101                // display green button
102            lcd_Fill(60, 10, 180, 60, GREEN);
103            lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
104        }
105        break;
106    case CLEAR:
107        if(!touch_IsTouched()) draw_Status = INIT;
108        break;
109    default:
110        break;
111    }
112}
113/* USER CODE END 4 */

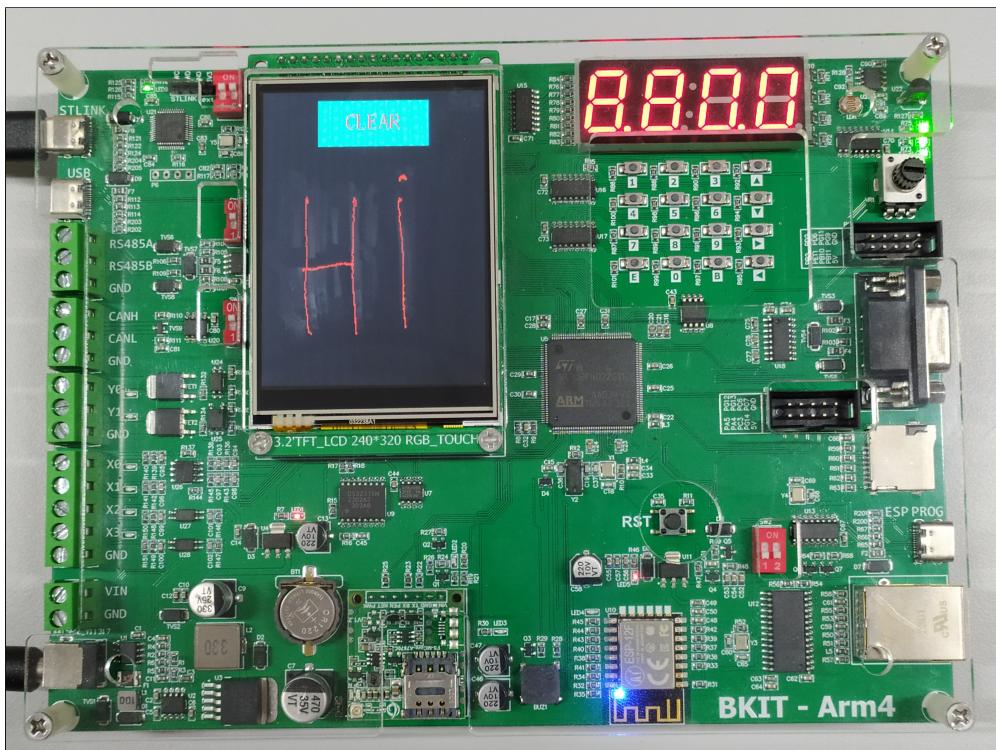
```

Program 7.3: Chương trình mẫu kiểm tra mà hình cảm ứng

Hình ảnh kết quả trên Kit thí nghiệm:



Hình 7.6: Kết quả hiển thị thực trên Kit thí nghiệm (1)



Hình 7.7: Kết quả hiển thị thực trên Kit thí nghiệm (2)

6 Bài tập và báo cáo

Hiện thực trò chơi rắn săn mồi cổ điển trên LCD. Trong đó, toàn bộ tương tác với người dùng sẽ được thao tác trên màn hình cảm ứng, ví dụ như:

- Chạm vào nút "Start" trên màn hình để bắt đầu trò chơi.
- Chạm vào các nút điều hướng trên màn hình để điều khiển con rắn.