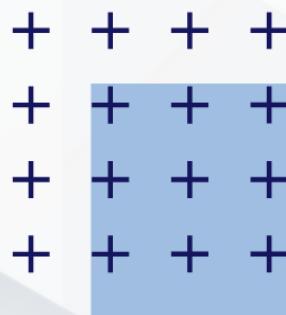
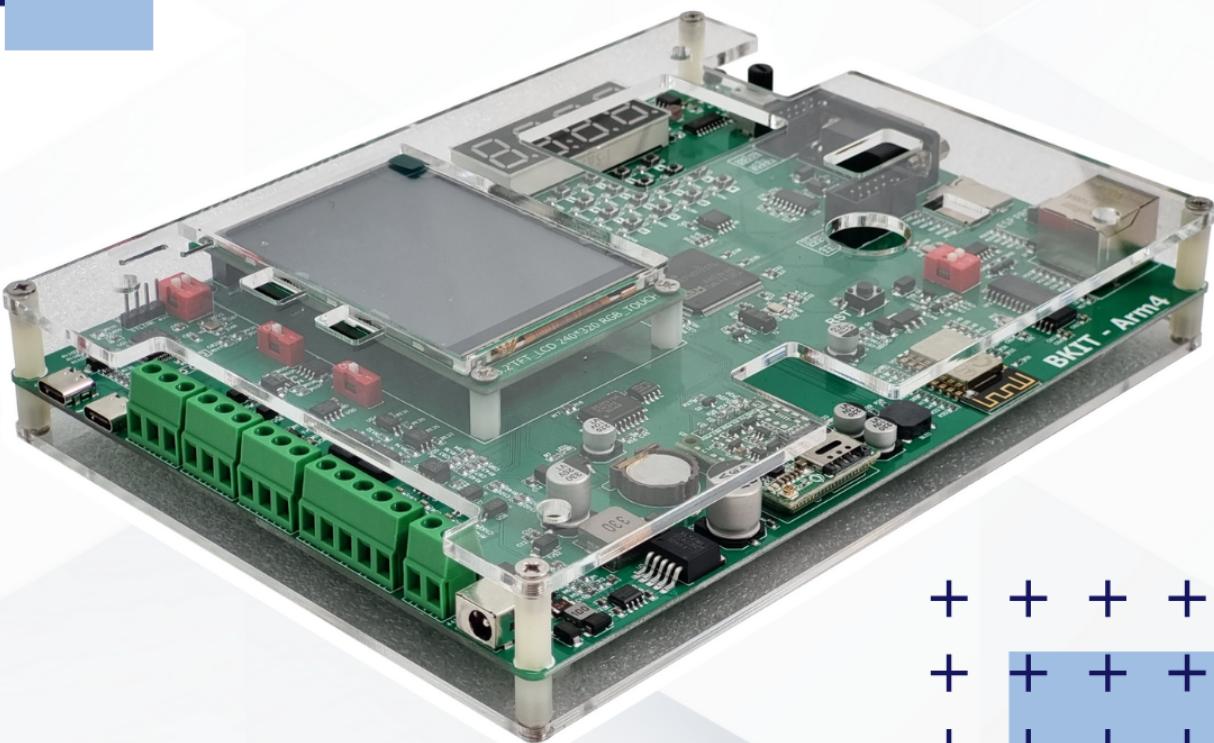


KIT THÍ NGHIỆM BKIT

ARM4



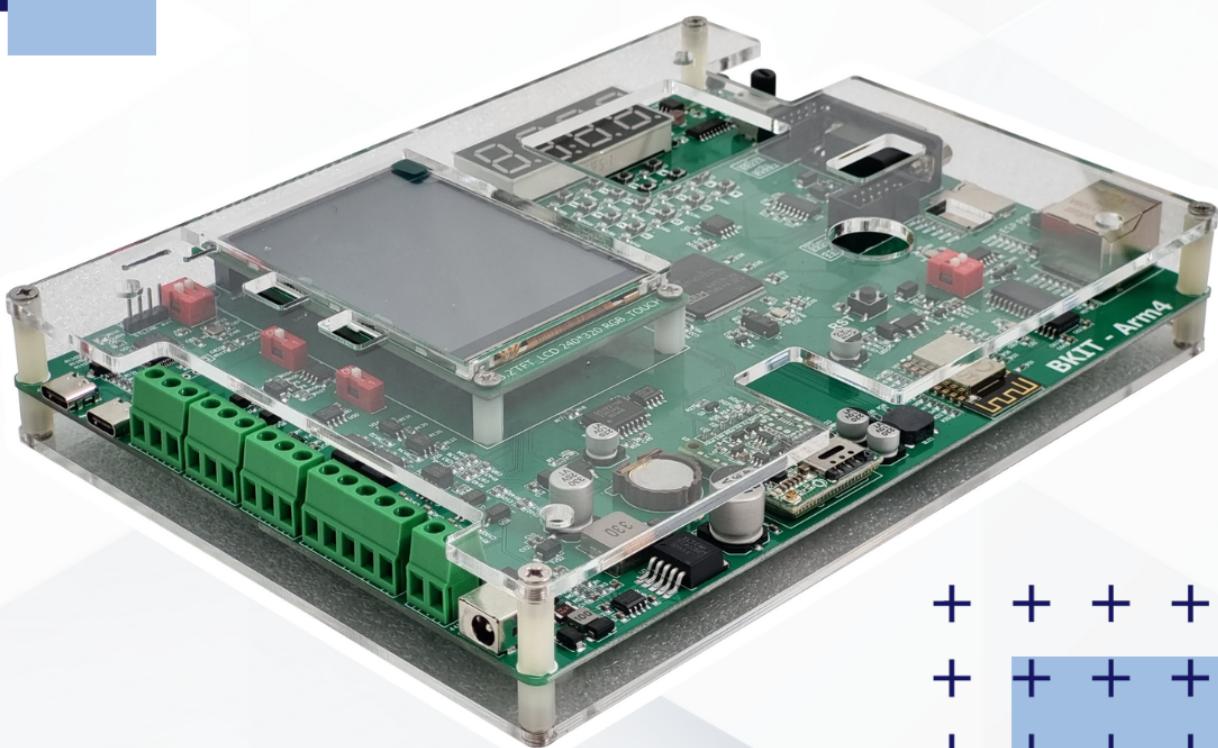
Mục lục

Chapter 3. LCD and Button matrix	4
1 Mục tiêu	5
2 Giới thiệu	5
3 Cơ sở lý thuyết	6
3.1 Sự cần thiết của điện trở kéo lên của nút nhấn	6
3.2 Chống rung nút nhấn	7
4 Hướng dẫn cấu hình	9
4.1 Cấu hình nút nhấn	9
4.2 Cấu hình LCD	11
5 Hướng dẫn lập trình	14
5.1 Sử dụng thư viện: button.h	14
5.2 Sử dụng thư viện: lcd.h	17
6 Bài tập và báo cáo	28
 Chapter 4. Real Time Clock	 29
1 Mục tiêu	30
2 Giới thiệu	30
3 Cơ sở lý thuyết	31
3.1 IC RTC DS3231	31
3.2 Giao thức I2C	33
4 Hướng dẫn cấu hình	36
5 Hướng dẫn lập trình	37
5.1 Thư viện utils.h	37

5.2	Thư viện ds3231.h	37
6	Bài tập và báo cáo	42
Chapter 5. Universal Asynchronous Receiver-Transmitter		43
1	Mục tiêu	44
2	Giới thiệu	44
3	Cơ sở lý thuyết	45
3.1	UART	45
3.2	RS232	46
4	Hướng dẫn cấu hình	47
4.1	Cấu hình UART-RS232	47
5	Hướng dẫn lập trình	48
5.1	Thư viện uart.h	48
6	Bài tập và báo cáo	55
6.1	Bài tập 1	55
6.2	Bài tập 2	55
6.3	Bài tập 3	55

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.

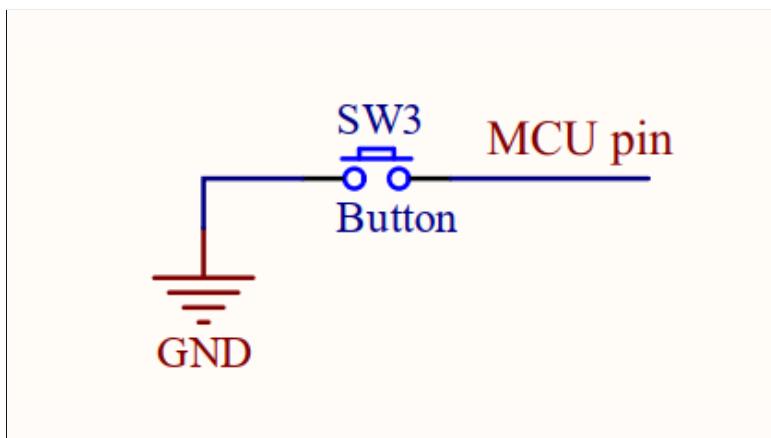
- Kích hoạt một máy bơm khi công tắc được bật.

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 thiết kế của điện trở kéo lên của nút nhấn

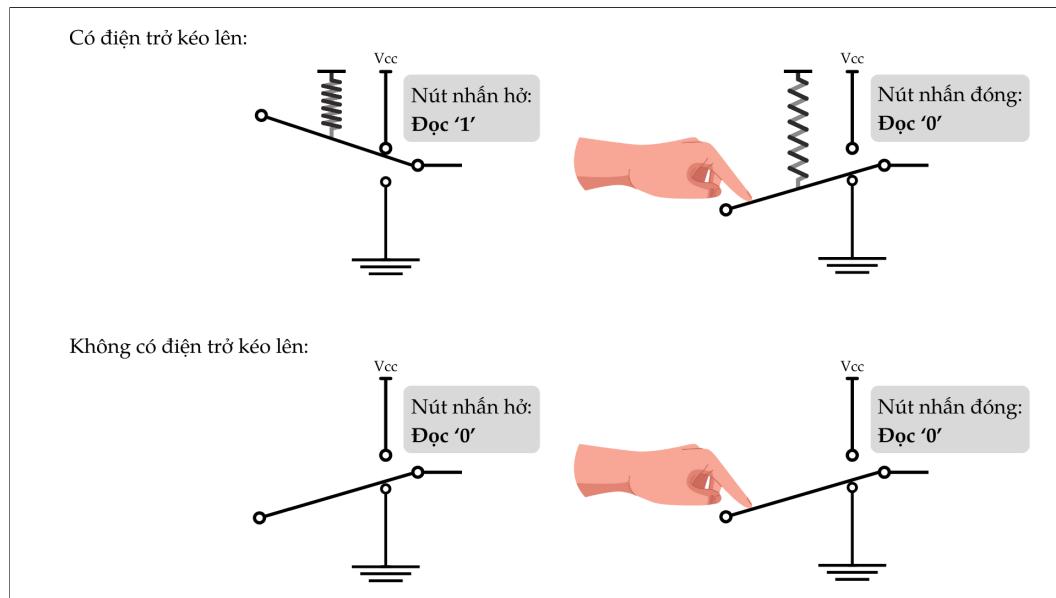


Hình 3.1: Kết nối nút nhấn với MCU

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

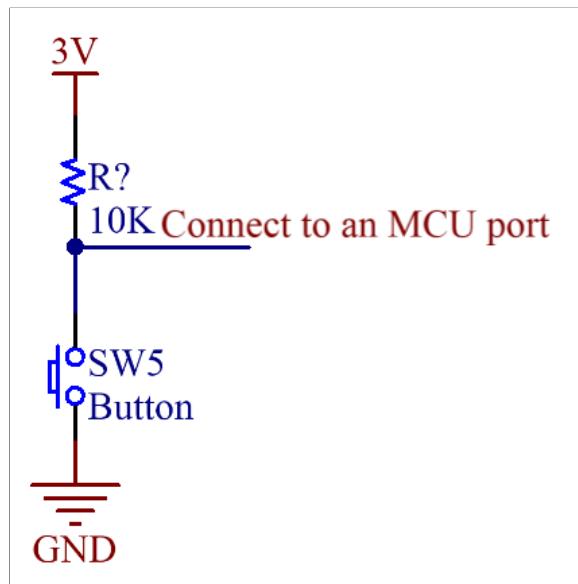
- Khi nút nhấn mở (không được nhấn), một điện trở nội bộ trong MCU sẽ "kéo" chân MCU lên nguồn (3.3V với STM32F407). Nếu chúng ta đọc giá trị của chân MCU, ta sẽ được giá trị "1".
- Khi nút nhấn đóng (được nhấn), chân MCU sẽ được nối với GND và sẽ có điện áp là 0V. Nếu ta đọc giá trị chân MCU, ta sẽ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

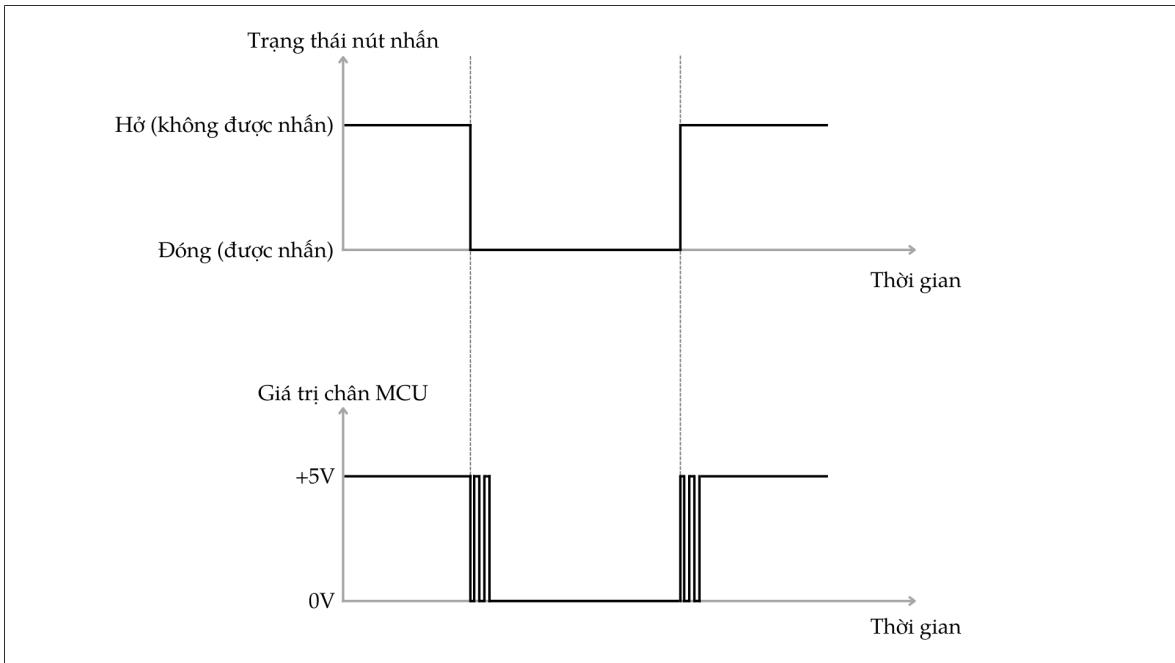
Vậy nên khi kết nối nút nhấn hoặc công tắc với MCU thì ta nên sử dụng một điện trở kéo lên như hình 3.2.



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

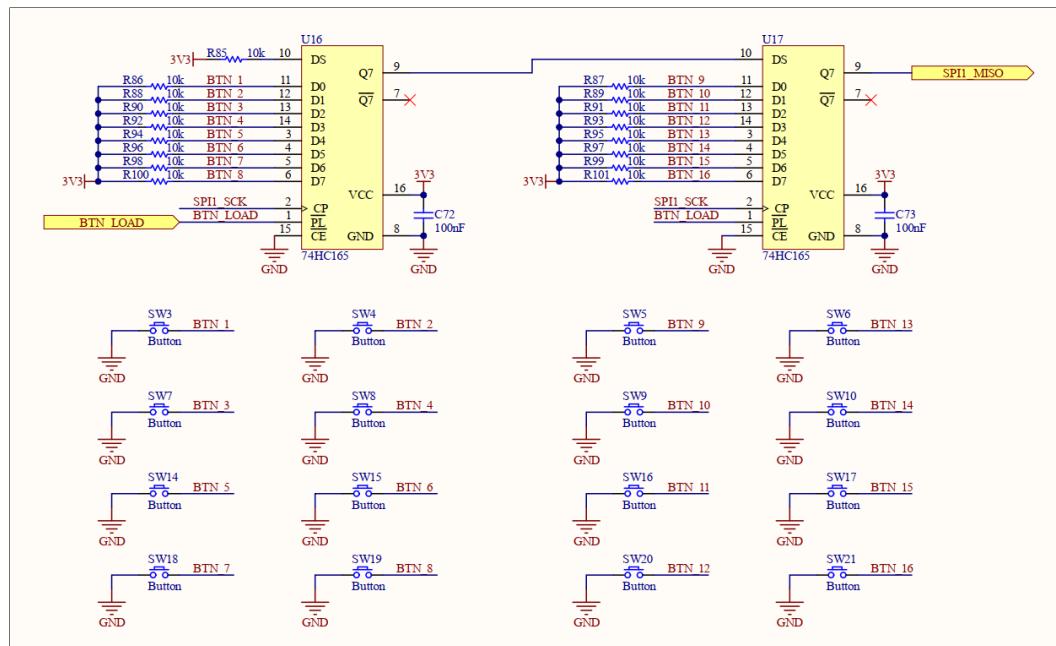
Có nhiều phương pháp chống rung nút nhấn sử dụng phần cứng hoặc phần mềm. Một cách chống rung bằng phần mềm là đọc nút nhấn sau mỗi chu kỳ (ví dụ: 10ms) và lưu lại trạng thái trước đó của nút nhấn. Một tín hiệu nút nhấn sẽ được chấp nhận khi không có sự thay đổi giữa tín hiệu hiện tại và tín hiệu trước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



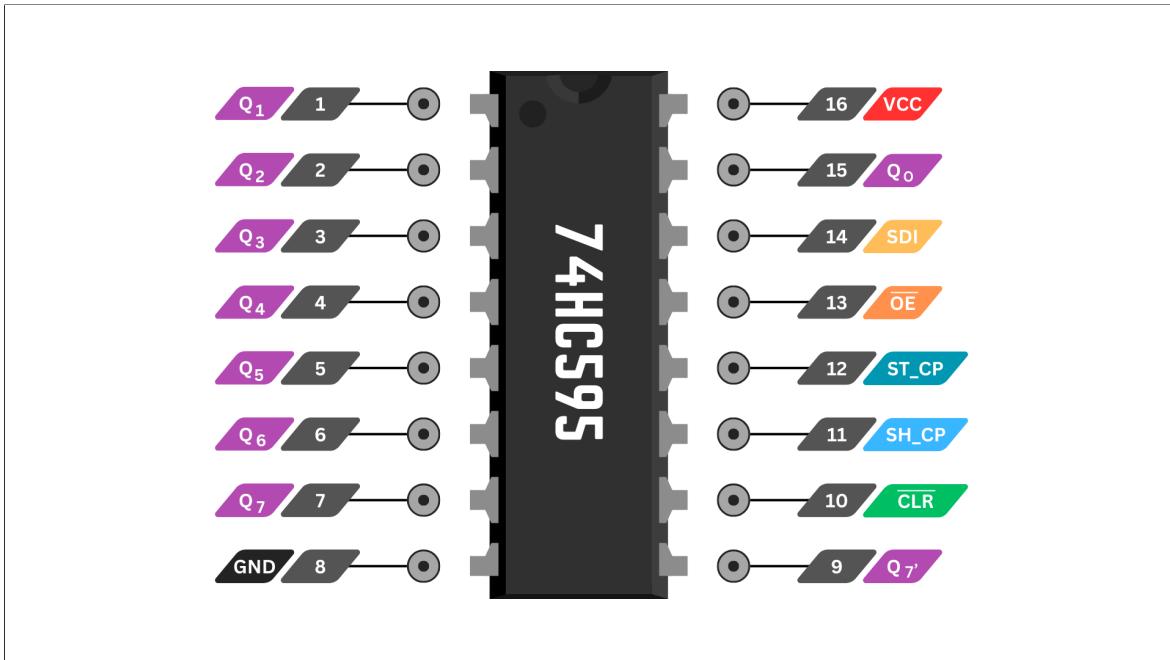
Hình 3.5: Kết nối ma trận phím trong Kit thí nghiệm

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:

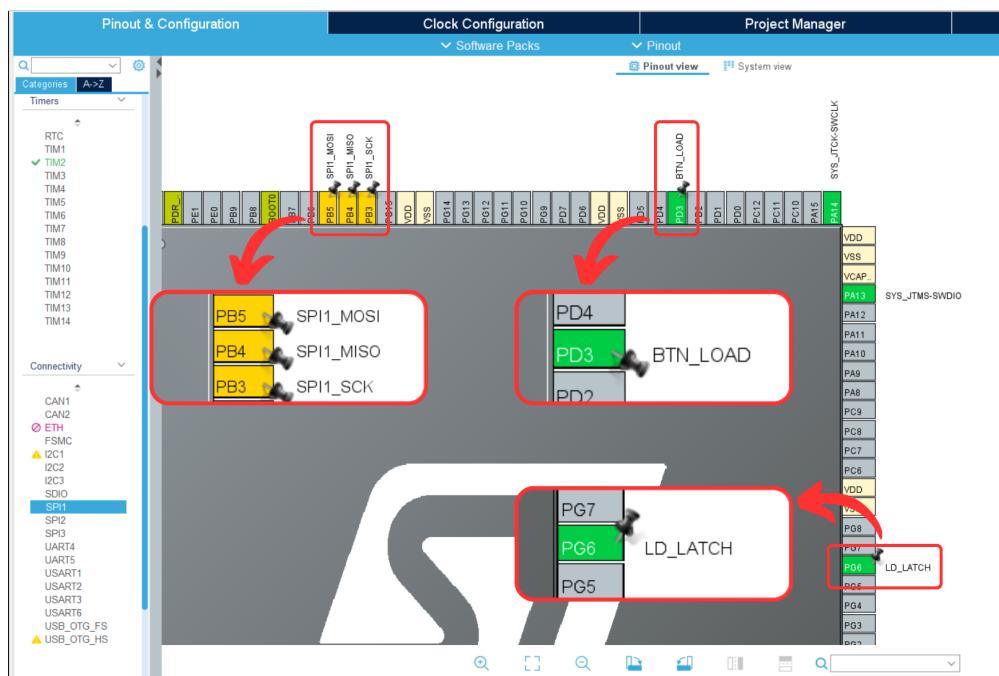
- Khi tín hiệu chân **PL** ở mức thấp, giá trị từ các ngõ vào D0-D7 sẽ được nạp song song vào thanh ghi trong IC.
- Khi tín hiệu chân **PL** ở mức cao và phát hiện cạnh lên ở chân **CP**, các giá trị bên trong thanh ghi sẽ được dịch và tín hiệu **Q7** sẽ lần lượt thể hiện các giá

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.



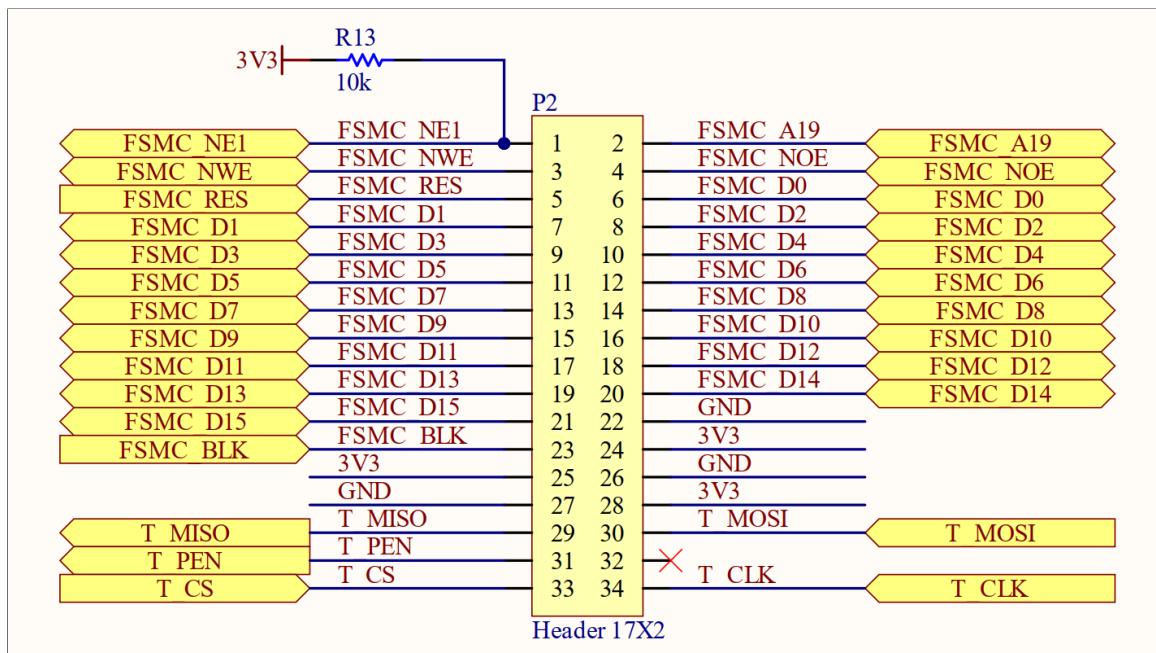
Hình 3.7: Các chân cần config

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:

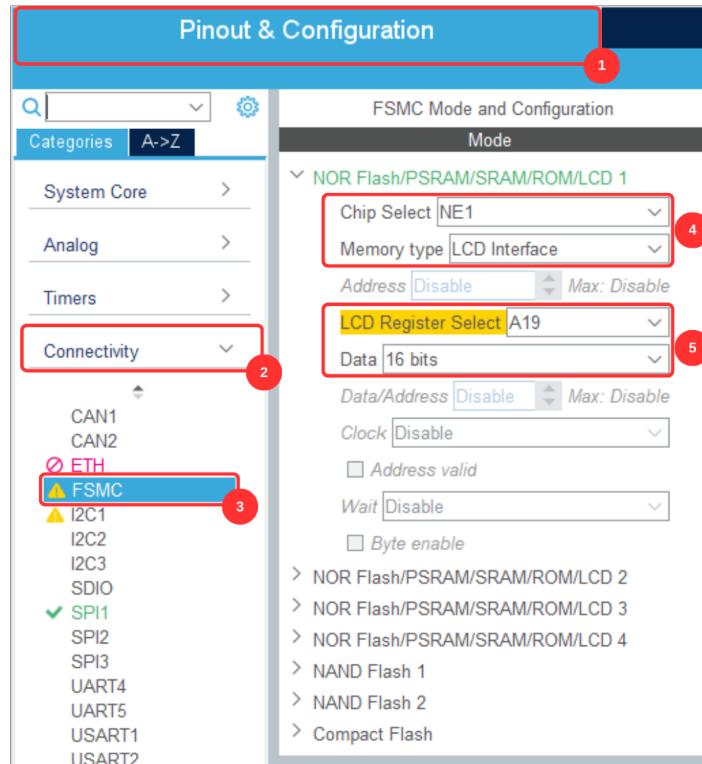
- Độ phân giải: 320x240 pixels.
- Chế độ màu: RGB 16 bit.
- Số bit dữ liệu giao tiếp với driver: 16.

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:

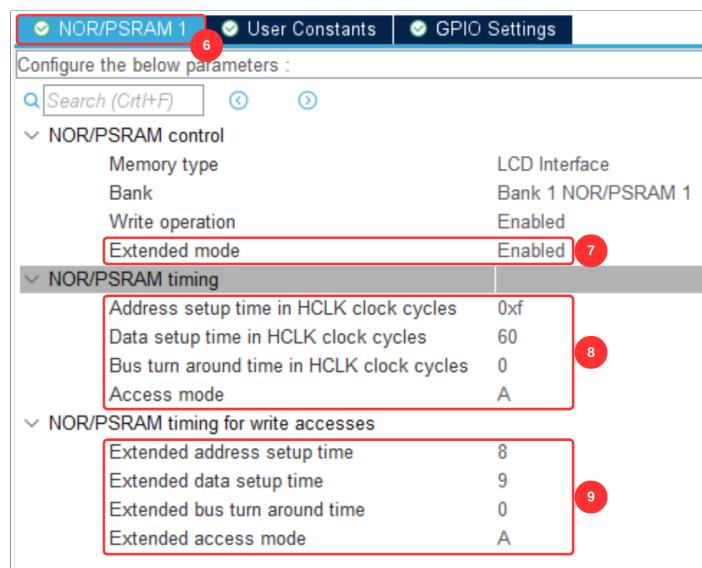


Hình 3.8: Schematic của LCD

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

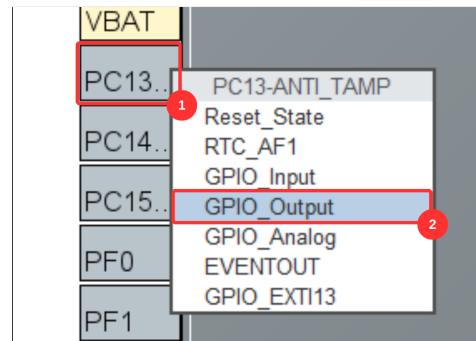


Hình 3.9: Cấu hình FSMC

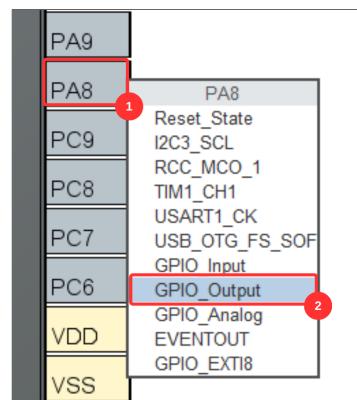


Hình 3.10: Cấu hình FSMC (tiếp theo)

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

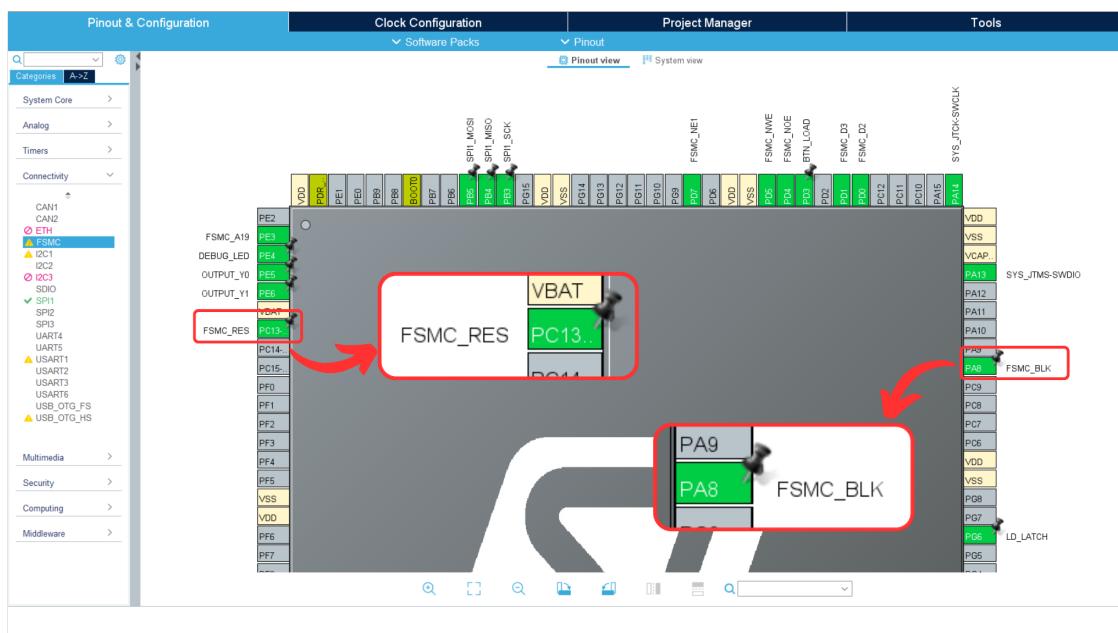


Hình 3.11: Cấu hình **FSMC_RES**



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  0XEF5B
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 ký tự trên LCD.
- **Tham số:**
 - **x:** Tọa độ x muốn in.
 - **y:** Tọa độ y muốn in.
 - **str:** Mảng chứa chuỗi các ký 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 ký 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 ký 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 muốn in.
 - **y:** Tọa độ y muốn in.
 - **num:** Số thập phân 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_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 .

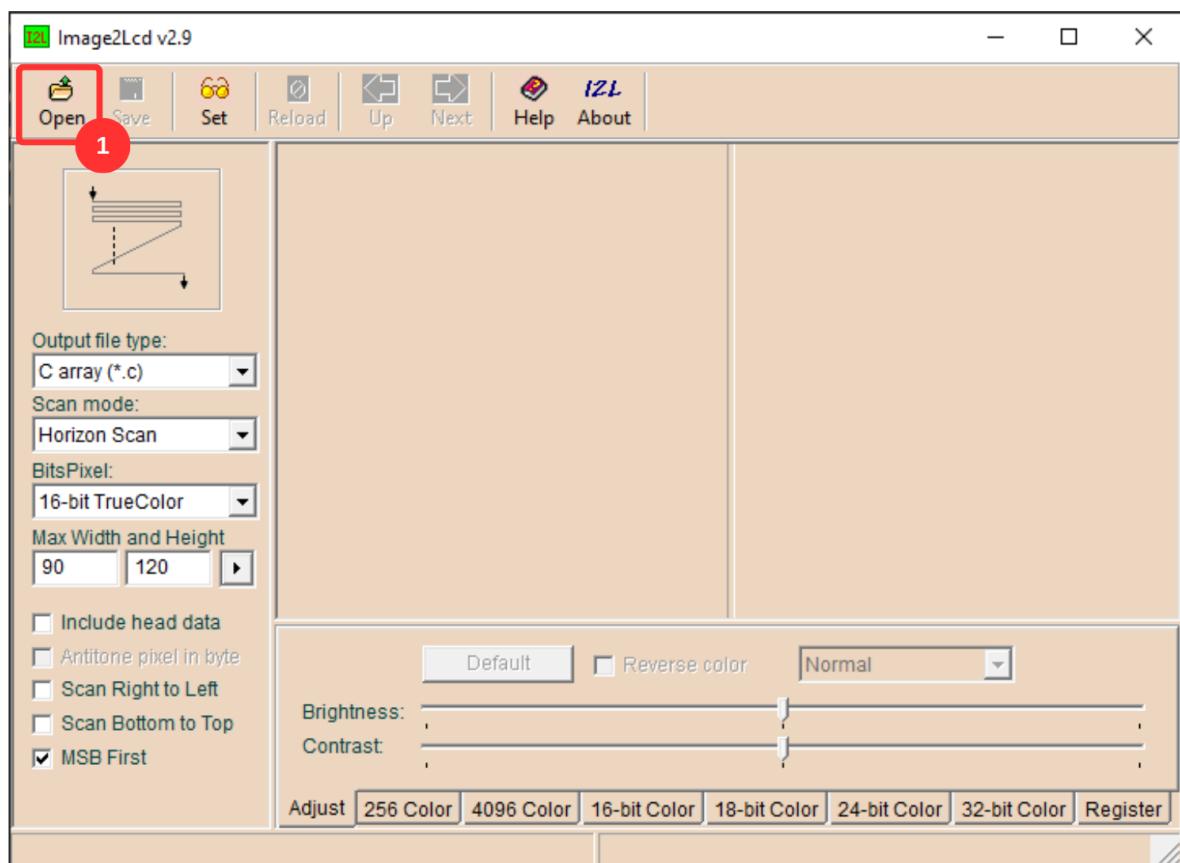
- **Tham số:**

- **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.

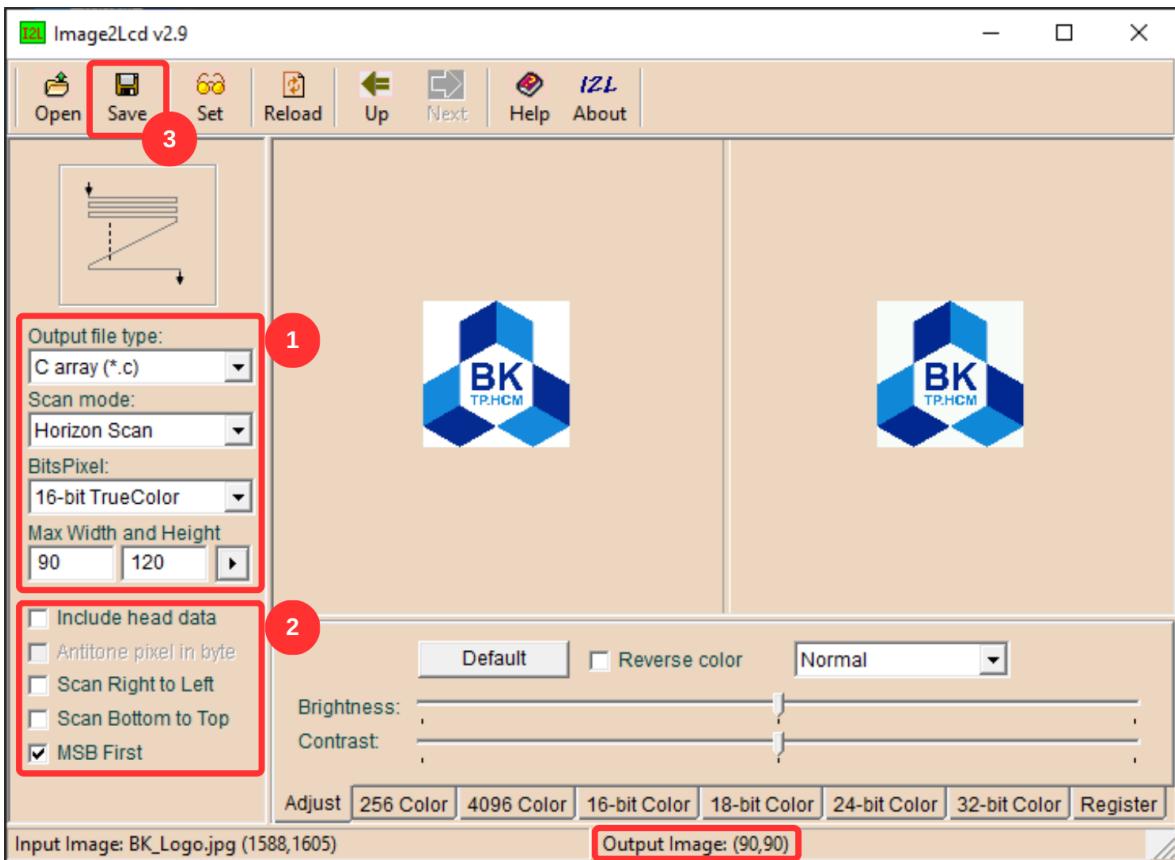
Để có thể hiển thị hình ảnh lên màn hình LCD, ta cần chuyển hình ảnh sang mảng dữ liệu có thể giao tiếp với màn hình LCD, mỗi điểm ảnh dùng 2 byte dữ liệu tương ứng với giá trị của màu RGB 16 bit. Để chuyển một hình ảnh thành dạng mảng nêu trên, chúng ta sẽ sử dụng sự trợ giúp của phần mềm Image2LCD.exe.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

```

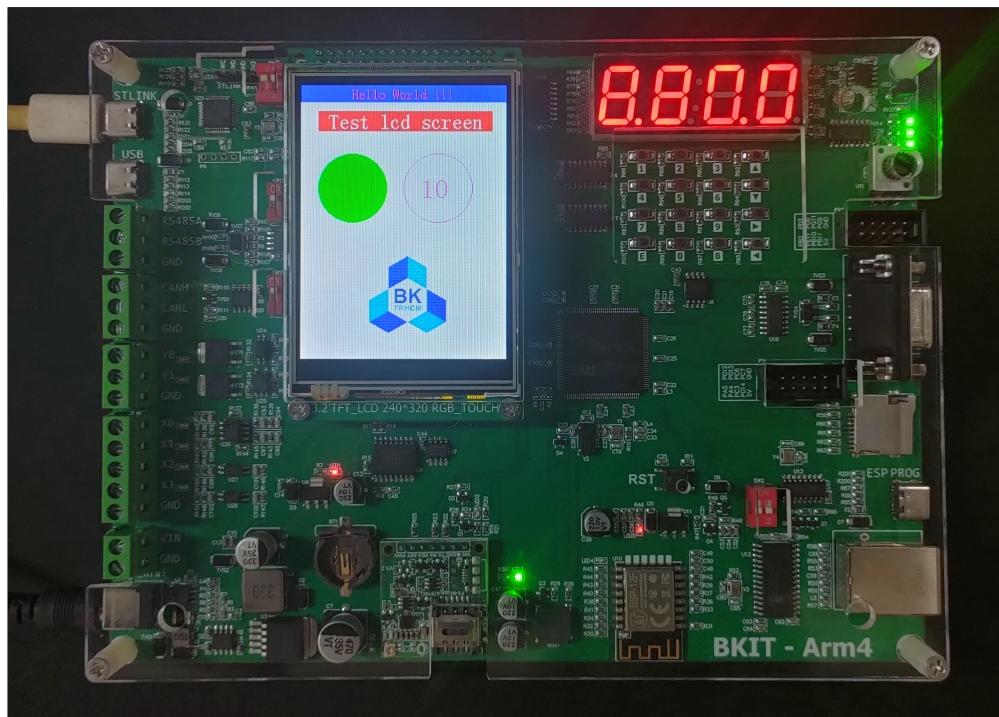
```

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,
11);
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 - Điều chỉnh chu kỳ đèn đỏ:

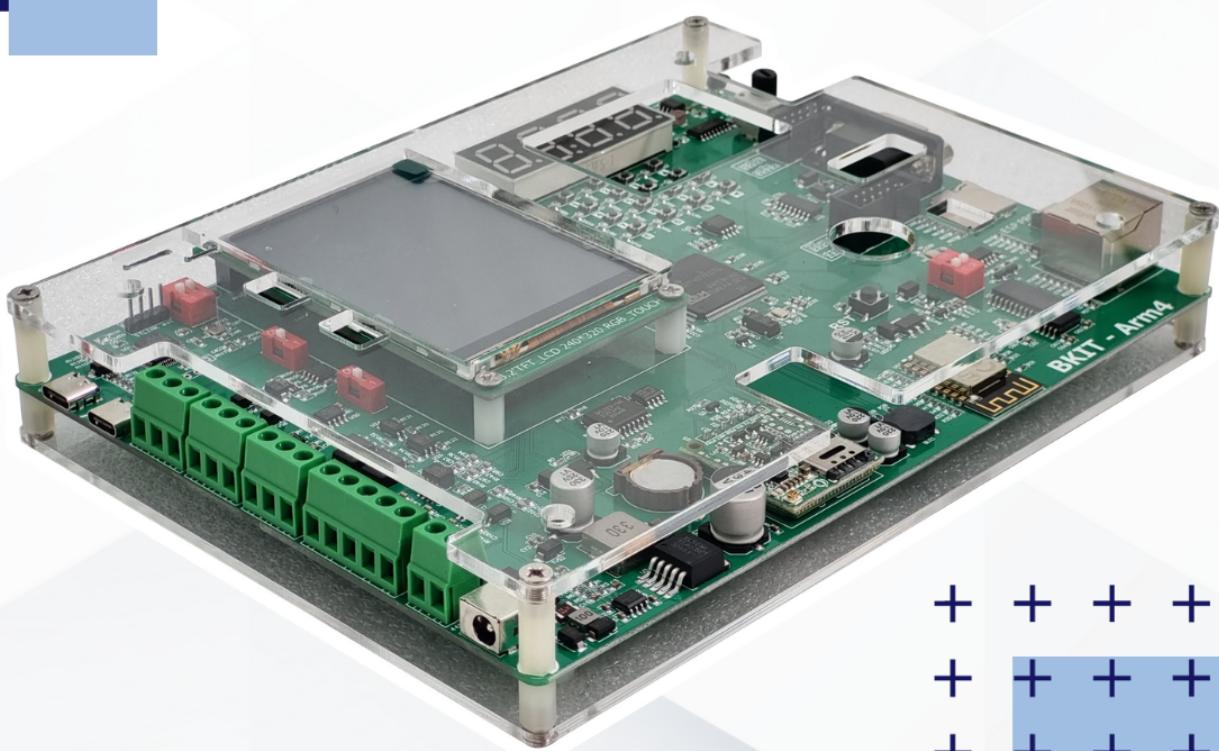
- Tín hiệu đèn đỏ chớp tắt với tần số 2Hz
- Hiển thị số được điều chỉnh trên màn hình LCD.
- Hiển thị chế độ đang hoạt động lên màn hình LCD.
- Nút nhấn thứ 2 dùng để tăng giá trị chu kỳ của đèn đỏ.
- Giá trị của chu kỳ đèn đỏ nằm trong khoảng 1-99.
- Nút nhấn thứ 3 dùng để xác nhận giá trị được chọn.

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.

CHƯƠNG 4

Real Time Clock



1 Mục tiêu

- Tìm hiểu về IC DS3231.
- Tìm hiểu giao tiếp I2C.
- Biết cách sử dụng module RTC trên Kit thí nghiệm và thư viện liên quan.

2 Giới thiệu

Bài lab này sẽ giới thiệu IC RTC DS3231 để tạo một đồng hồ thời gian thực. DS3231 cung cấp thông tin như giây, phút, giờ, ngày, thứ, tháng và năm. Chip này xử lý các chức năng như đếm thời gian để theo dõi thời gian thực và giao tiếp với vi điều khiển thông qua giao tiếp I2C.



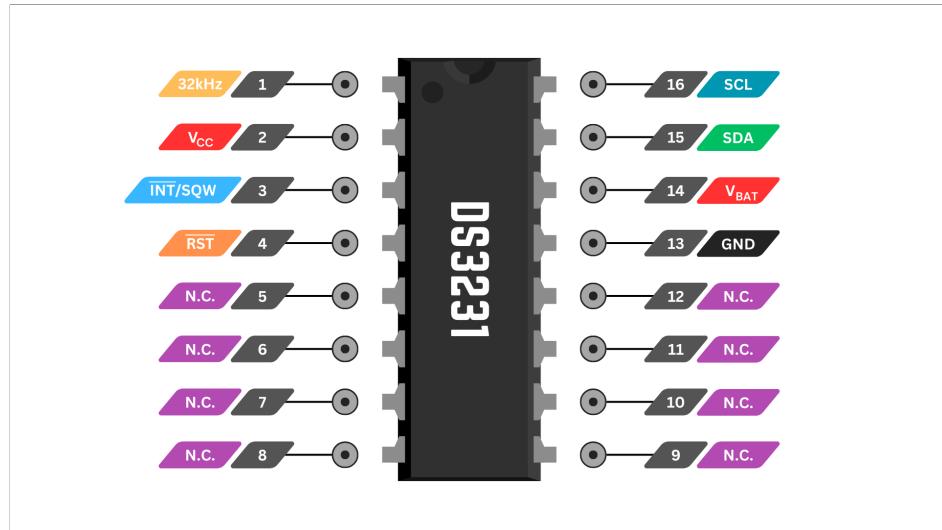
Hình 4.1: IC DS3231

I2C (Inter-Integrated Circuit), giao thức do Philips Semiconductors phát triển, truyền dữ liệu giữa các IC qua hai đường tín hiệu. Dữ liệu truyền từng bit theo tín hiệu đồng hồ và được dùng để giao tiếp với nhiều loại IC như vi điều khiển, cảm biến, EEPROM,

3 Cơ sở lý thuyết

3.1 IC RTC DS3231

DS3231 là một mạch RTC (Real-Time Clock) phổ biến có 16 chân và giao tiếp vi điều khiển thông qua giao thức I2C. Dưới đây là mô tả đầy đủ về các chân của DS3231:



Hình 4.2: Sơ đồ chân IC DS3231

- 32kHz Output (32kHz): Chân này cung cấp tín hiệu đồng hồ với tần số 32kHz. Thường được sử dụng để cung cấp tín hiệu đồng hồ chính xác cho các ứng dụng khác.
- VCC (Positive Supply Voltage): Chân này cung cấp nguồn (VCC) cho DS3231.
- \overline{INT}/SQW (Interrupt or Square Wave Output): Chân này có thể được cấu hình là chân ngắn hoặc đầu ra tín hiệu sóng vuông tùy thuộc vào cài đặt. Chân này có thể được sử dụng để tạo một tín hiệu ngắn khi một sự kiện thời gian cụ thể xảy ra.
- \overline{RST} (Reset): Chân này có thể được sử dụng để đặt lại DS3231 bằng cách đặt về mức mặc định (thường là GND). Điều này thường không được sử dụng trong ứng dụng thời gian thực tiêu chuẩn.
- SCL (Serial Clock Line): Chân này là dây đồng hồ nối tiếp (SCL) trong giao thức I2C. Nó được sử dụng để đồng bộ hóa truyền dữ liệu giữa DS3231 và mạch điều khiển.

- SDA (Serial Data Line): Chân này là dây dữ liệu nối tiếp (SDA) trong giao thức I2C. Nó được sử dụng để truyền dữ liệu giữa DS3231 và mạch điều khiển.
- GND (Ground): Chân này được kết nối đến đất (Ground) để tạo đất chung cho mạch.
- VBAT (Battery Input): Chân này được sử dụng để cung cấp nguồn năng lượng từ pin CR2032 hoặc pin lithium khác để duy trì thời gian thời gian thực và cài đặt khi nguồn chính bị ngắt. Thường được kết nối đến pin dương của pin lithium.
- NC (No Connect): Chân này không được sử dụng và thường để trống.

Để có thể sử dụng DS3231, chúng ta cần tìm hiểu các bit thời gian. DS3231 lưu trữ thời gian trong các thanh ghi (registers) bên trong nó, và mỗi phần tử thời gian (năm, tháng, ngày, thứ, giờ, phút, giây) được lưu trữ trong các bit cụ thể tại các thanh ghi.

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE	
00h	0	10 Seconds			Seconds			Seconds	00–59		
01h	0	10 Minutes			Minutes			Minutes	00–59		
02h	0	12/24	AM/PM	10 Hour	Hour			Hours	1–12 + AM/PM 00–23		
03h	0	0	0	0	0	Day		Day	1–7		
04h	0	0	10 Date		Date			Date	01–31		
05h	Century	0	0	10 Month	Month			Month/ Century	01–12 + Century		
06h	10 Year				Year			Year	00–99		
07h	A1M1	10 Seconds			Seconds			Alarm 1 Seconds	00–59		
08h	A1M2	10 Minutes			Minutes			Alarm 1 Minutes	00–59		
09h	A1M3	12/24	AM/PM	10 Hour	Hour			Alarm 1 Hours	1–12 + AM/PM 00–23		
0Ah	A1M4	DY/DT	10 Date		Day			Alarm 1 Day	1–7		
					Date			Alarm 1 Date	1–31		
0Bh	A2M2	10 Minutes			Minutes			Alarm 2 Minutes	00–59		
0Ch	A2M3	12/24	AM/PM	10 Hour	Hour			Alarm 2 Hours	1–12 + AM/PM 00–23		
0Dh	A2M4	DY/DT	10 Date		Day			Alarm 2 Day	1–7		
					Date			Alarm 2 Date	1–31		
0Eh	EOSC	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—	
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—	
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—	
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—	
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—	

Hình 4.3: Địa chỉ các thanh ghi trong DS3231

Dưới đây mô tả các thanh ghi thời gian trong DS3231:

- Giây (Seconds): DS3231 lưu giây trong một thanh ghi có 7 bit, từ 0 đến 59 giây.

- Phút (Minutes): DS3231 lưu phút trong một thanh ghi có 7 bit, từ 0 đến 59 phút.
- Giờ (Hours): DS3231 lưu giờ dưới hai chế độ thời gian (12 giờ hoặc 24 giờ) trong một thanh ghi có 6 bit. Đối với chế độ 12 giờ, một bit AM/PM được sử dụng để xác định buổi sáng (AM) hoặc buổi chiều (PM).
- Ngày (Date): DS3231 lưu ngày trong một thanh ghi có 6 bit, từ 1 đến 31 ngày.
- Tháng (Month): DS3231 lưu tháng trong một thanh ghi có 5 bit, từ 1 đến 12 tháng.
- Năm (Year): DS3231 lưu năm trong một thanh ghi có 7 bit, từ 0 đến 99 năm (tương ứng với 2000-2099 trong thực tế).

Ngoài ra, DS3231 hỗ trợ hai báo thức độc lập, được gọi là Alarm 1 và Alarm 2. Mỗi báo thức có thể được cấu hình để kích hoạt vào một thời điểm cụ thể hoặc hàng ngày/giờ/phút nhất định. Các bit của báo thức được cấu hình trong các thanh ghi tương ứng:

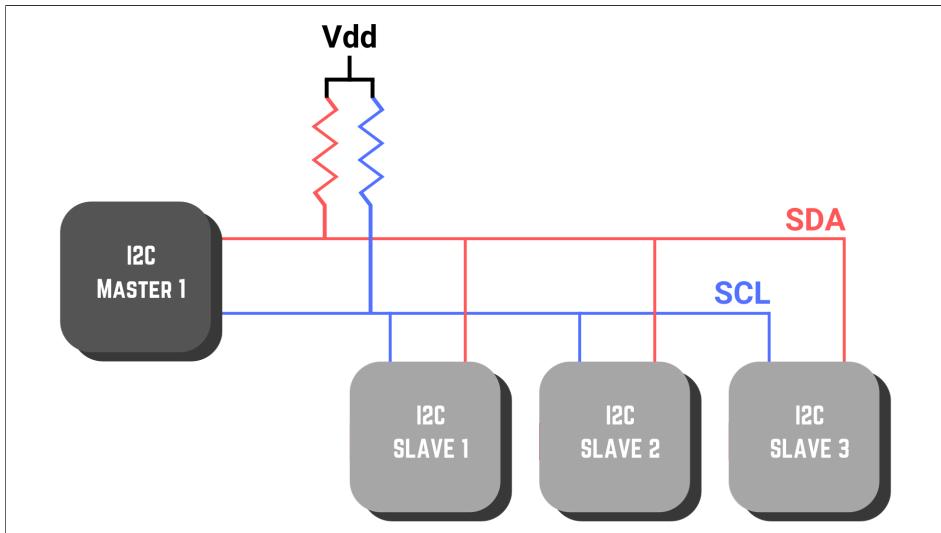
- Báo thức 1 (Alarm 1): Cấu hình trong thanh ghi ALM1 (0x07). Báo thức 1 có thể được cấu hình để kích hoạt vào một thời điểm cụ thể hàng ngày/giờ/phút.
- Báo thức 2 (Alarm 2): Cấu hình trong thanh ghi ALM2 (0x0B). Báo thức 2 cũng có thể được cấu hình để kích hoạt vào một thời điểm cụ thể hàng ngày/giờ/phút.

Để cài đặt và quản lý các bit thời gian và báo thức trong DS3231, chúng ta cần giao tiếp với nó thông qua giao thức I2C và ghi/đọc các giá trị vào các thanh ghi tương ứng.

3.2 Giao thức I2C

Cấu tạo:

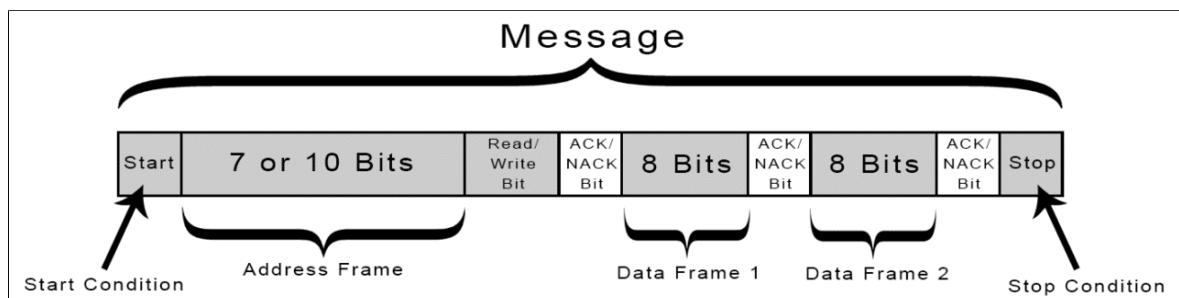
- I2C sử dụng 2 đường truyền tín hiệu:
 - SCL: Tạo xung nhịp đồng hồ do Master phát đi.
 - SDA: Đường truyền nhận dữ liệu.
- Giao tiếp I2C bao gồm quá trình truyền nhận dữ liệu giữa các thiết bị chủ (Master) và thiết bị phụ (Slave).
- Thiết bị Master là một vi điều khiển, có nhiệm vụ điều khiển đường tín hiệu SCL và gửi/nhận dữ liệu hoặc lệnh thông qua đường SDA đến các thiết bị khác.



Hình 4.4: Kết nối I2C giữa Master và Slave

- Các thiết bị nhận tín hiệu điều khiển từ thiết bị Master được gọi là các thiết bị Slave. Thiết bị Slave thường là các IC, hoặc thậm chí là vi điều khiển.
- Master và Slave được kết nối với nhau theo hình 4.4 Hai đường bus SCL và SDA đều hoạt động ở chế độ Open Drain, tức là bất kỳ thiết bị nào kết nối với mạng I2C cũng chỉ có thể kéo 2 đường bus này xuống mức thấp (LOW), nhưng không thể kéo chúng lên mức cao.

Dataframe của I2C:



Hình 4.5: Dataframe của I2C

Quá trình truyền nhận dữ liệu:

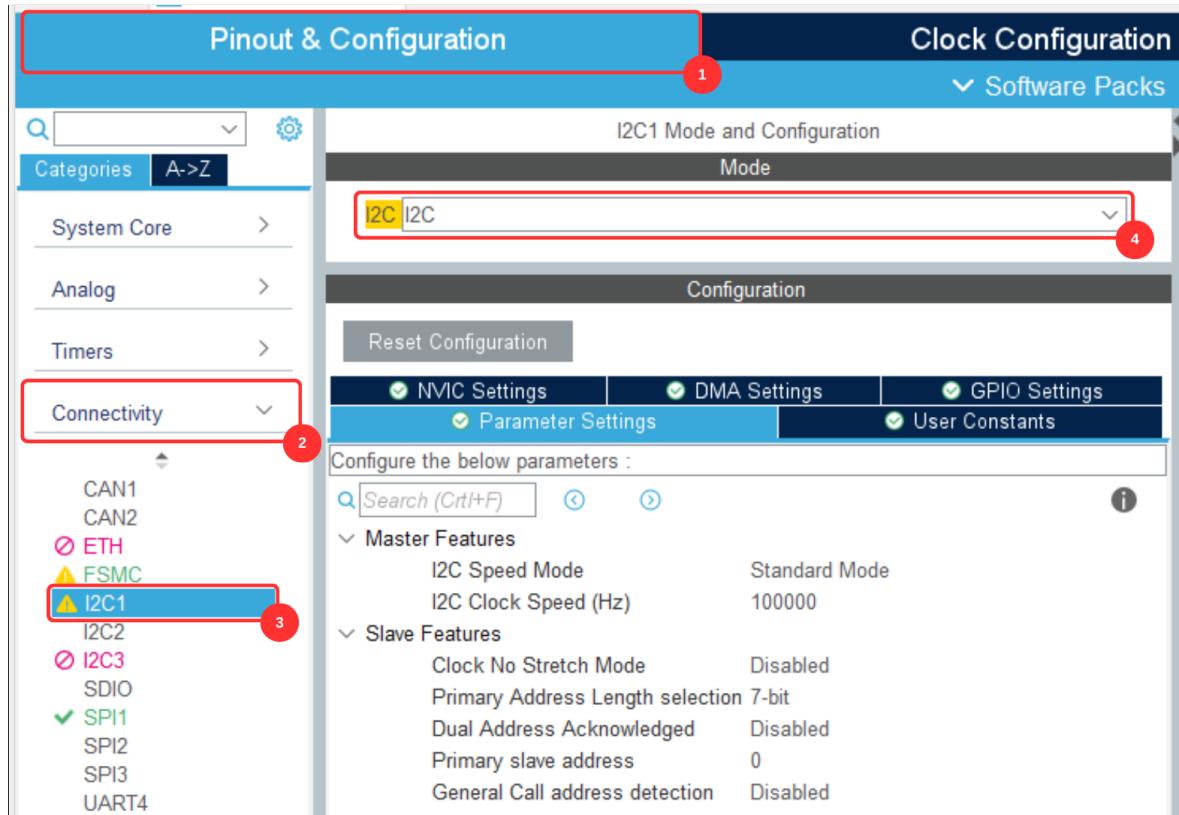
- Start: Thiết bị Master sẽ gửi một xung Start bằng cách hạ lần lượt các dây SDA, SCL từ mức 1 xuống 0.
- Sau đó, Master sẽ gửi 7 bit địa chỉ đến Slave mà nó muốn giao tiếp, kèm theo bit Read/Write.
- Slave sẽ so sánh địa chỉ vật lý của nó với địa chỉ vừa nhận được. Nếu khớp, Slave sẽ xác nhận bằng cách hạ dây SDA xuống 0 và đặt bit ACK/NACK là '0'. Nếu không khớp, dây SDA và bit ACK/NACK mặc định sẽ là '1'.

- Thiết bị Master sẽ tiếp tục gửi hoặc nhận khung bit dữ liệu. Nếu Master gửi đến Slave, bit Read/Write sẽ ở mức 0. Ngược lại, nếu Master nhận dữ liệu, bit này sẽ ở mức 1.
- Nếu khung dữ liệu được truyền đi thành công, bit ACK/NACK sẽ được đặt về mức 0 để báo hiệu cho Master tiếp tục.
- Khi tất cả dữ liệu đã được gửi thành công đến Slave, Master sẽ phát tín hiệu Stop để thông báo cho các Slave rằng quá trình truyền đã kết thúc, bằng cách chuyển lần lượt SCL, SDA từ mức 0 lên mức 1.

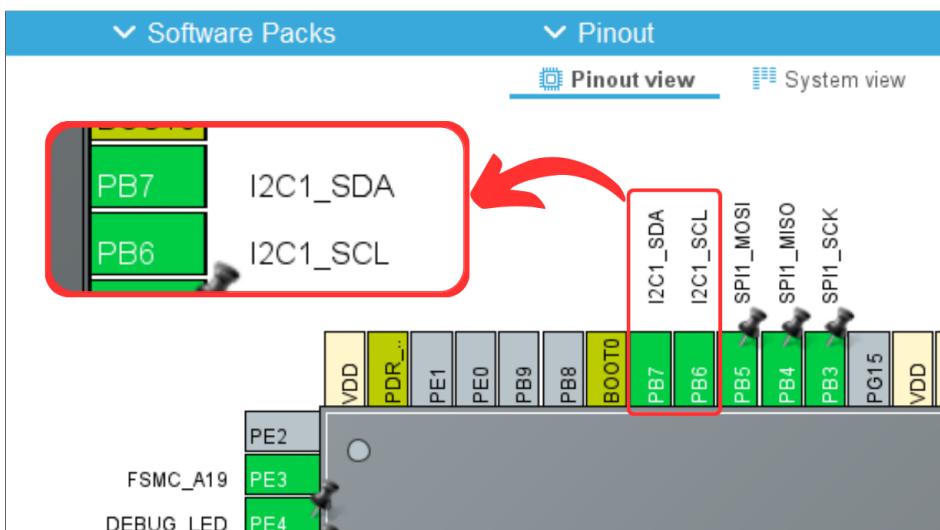
Bên cạnh đó, giao tiếp I2C giúp chúng ta có thể trao đổi dữ liệu giữa các thiết bị Master và Slave một cách dễ dàng. Tuy nhiên, một số trường hợp có thể xảy ra xung đột hoặc lỗi dữ liệu. Một cách để giảm thiểu vấn đề này là mỗi thiết bị Master nên kiểm tra trạng thái của đường SDA trước khi bắt đầu truyền dữ liệu.

4 Hướng dẫn cấu hình

Trên Kit thí nghiệm, IC DS3231 được điều khiển thông qua module I2C1 của MCU.



Hình 4.6: Cấu hình I2C



Hình 4.7: Pinout view sau khi cấu hình I2C

5 Hướng dẫn lập trình

5.1 Thư viện utils.h

Đây là thư viện chứa các hàm để hỗ trợ cho việc tính toán. Các file **utils.h** và **utils.c** có thể được sao chép từ project mẫu **Bai4_I2C_Realtimedclock**.

uint8_t BCD2DEC(uint8_t data)

- **Mô tả:** Chuyển đổi mã BCD sang số thập phân.
- **Tham số:**
 - **data:** Mã BCD cần chuyển đổi.
- **Giá trị trả về:** Số thập phân.

uint8_t DEC2BCD(uint8_t data)

- **Mô tả:** Chuyển đổi số thập phân sang mã BCD.
- **Tham số:**
 - **data:** Số thập phân cần chuyển đổi.
- **Giá trị trả về:** Mã BCD.

5.2 Thư viện ds3231.h

Các file **ds3231.h** và **ds3231.c** có thể được sao chép từ project mẫu **Bai5_I2C_Realtimedclock**.

Các biến có thể truy xuất

```
1 extern uint8_t ds3231_hours;
2 extern uint8_t ds3231_min;
3 extern uint8_t ds3231_sec;
4 extern uint8_t ds3231_date;
5 extern uint8_t ds3231_day;
6 extern uint8_t ds3231_month;
7 extern uint8_t ds3231_year;
```

Program 4.1: Các biến chứa giá trị về thời gian

void ds3231_init()

- **Mô tả:** Khởi tạo module DS3213.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void ds3231_Write(uint8_t address, uint8_t value)

- **Mô tả:** Ghi giá trị vào thanh ghi trong DS3231.
- **Tham số:**
 - **address:** Địa chỉ của thanh ghi.
 - **value:** Giá trị muốn ghi.
- **Giá trị trả về:** Không có.

```
1 #define ADDRESS_SEC      0x00
2 #define ADDRESS_MIN      0x01
3 #define ADDRESS_HOUR     0x02
4 #define ADDRESS_DAY       0x03
5 #define ADDRESS_DATE      0x04
6 #define ADDRESS_MONTH    0x05
7 #define ADDRESS_YEAR      0x06
```

Program 4.2: Một số địa chỉ thanh ghi đã được định nghĩa sẵn trong ds3231.h

void ds3231_ReadTime()

- **Mô tả:** Đọc giá trị các thông tin về ngày giờ và lưu vào các biến.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

Để sử dụng module thời gian thực, ta cần khởi tạo để kiểm tra kết nối với DS3231 bằng hàm **ds3231_init**. Hàm **updateTime** là hàm ví dụ giúp ghi các thời gian ban đầu vào **DS3231**. Hiện tại hàm này chỉ nên được gọi trong lần đầu tiên sử dụng module đồng hồ thời gian thực (RTC) để khởi tạo các giá trị thời gian ban đầu, sau đó các giá trị thời gian trong DS3231 sẽ tự động tăng lên theo thời gian thực. Để lấy thời gian từ DS3231 và lưu vào các biến thời gian đã được khai báo ta dùng hàm **ds3231_ReadTime** (hiện hàm này được gọi mỗi 50ms). Hàm **displayTime** là ví dụ đơn giản để hiển thị các giá trị thời gian được lấy từ DS3231 lên màn hình LCD.

```

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 /* USER CODE END Includes */
10 // ...
11 /* Private variables
   -----
   */
12
13 /* USER CODE BEGIN PV */
14 uint8_t count_led_debug = 0;
15 /* USER CODE END PV */
16
17 /* Private function prototypes
   -----
   */
18 void SystemClock_Config(void);
19 /* USER CODE BEGIN PFP */
20 void system_init();
21 void test_LedDebug();
22 void displayTime();
23 void updateTime();
24 // ...
25 int main(void)
26 {
27 // ...
28     /* USER CODE BEGIN 2 */
29     system_init();
30     /* USER CODE END 2 */
31     /* Infinite loop */
32     /* USER CODE BEGIN WHILE */
33     lcd_Clear(BLACK);
34     updateTime();
35     while (1)
36     {

```

```

37     while(!flag_timer2);
38     flag_timer2 = 0;
39     button_Scan();
40     ds3231_ReadTime();
41     displayTime();
42     /* USER CODE END WHILE */
43     /* USER CODE BEGIN 3 */
44 }
45 /* USER CODE END 3 */
46 }
47 // ...
48 /* USER CODE BEGIN 4 */
49 void system_init(){
50     HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port ,OUTPUT_Y0_Pin ,0);
51     HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port ,OUTPUT_Y1_Pin ,0);
52     HAL_GPIO_WritePin(DEBUG_LED_GPIO_Port ,DEBUG_LED_Pin ,0);
53     timer_init();
54     led7_init();
55     button_init();
56     lcd_init();
57     ds3231_init();
58     setTimer2(50);
59 }
60
61 void test_LedDebug(){
62     count_led_debug = (count_led_debug + 1)%20;
63     if(count_led_debug == 0){
64         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port , DEBUG_LED_Pin);
65     }
66 }
67
68 void updateTime(){
69     ds3231_Write(ADDRESS_YEAR , 23);
70     ds3231_Write(ADDRESS_MONTH , 10);
71     ds3231_Write(ADDRESS_DATE , 20);
72     ds3231_Write(ADDRESS_DAY , 6);
73     ds3231_Write(ADDRESS_HOUR , 20);
74     ds3231_Write(ADDRESS_MIN , 11);
75     ds3231_Write(ADDRESS_SEC , 23);

```

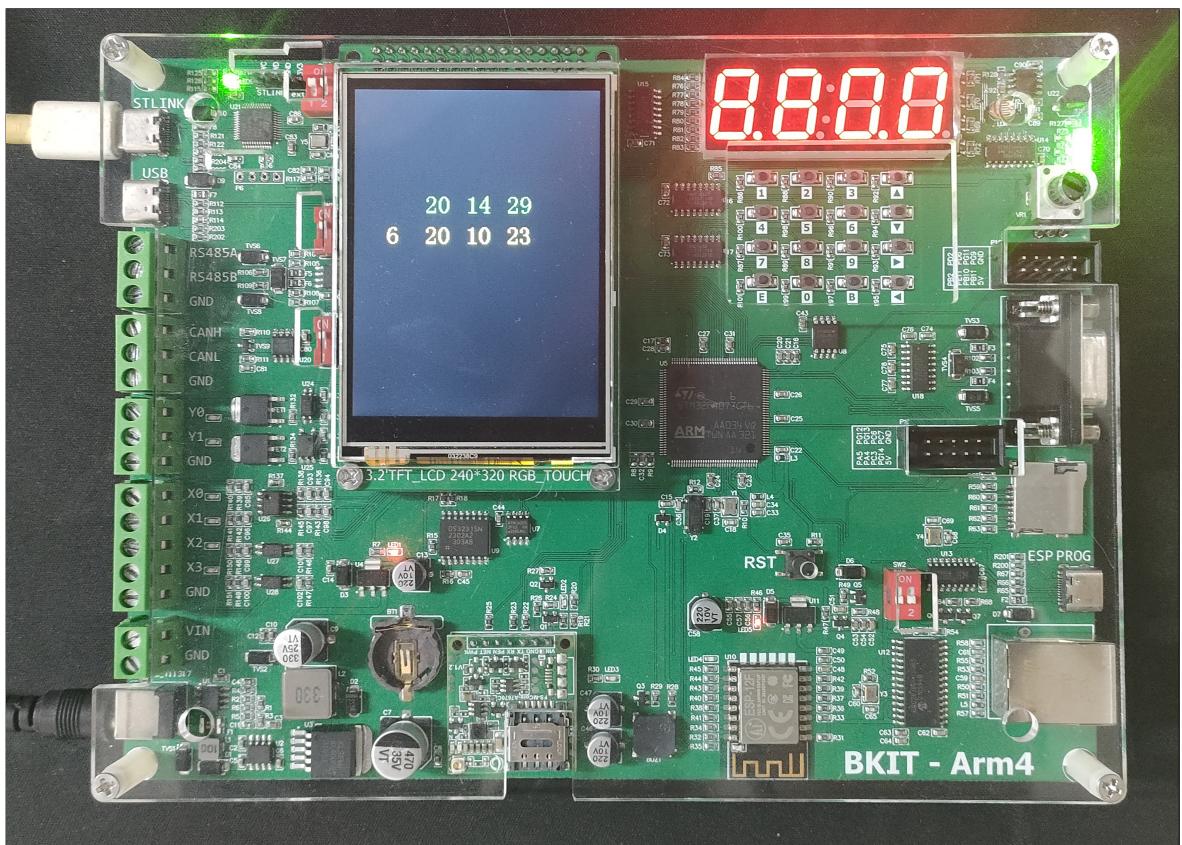
```

76 }
77
78 void displayTime(){
79     lcd_ShowIntNum(70,100,ds3231_hours,2,WHITE,BLACK,24);
80     lcd_ShowIntNum(110,100,ds3231_min,2,WHITE,BLACK,24);
81     lcd_ShowIntNum(150,100,ds3231_sec,2,WHITE,BLACK,24);
82     lcd_ShowIntNum(20,130,ds3231_day,2,WHITE,BLACK,24);
83     lcd_ShowIntNum(70,130,ds3231_date,2,WHITE,BLACK,24);
84     lcd_ShowIntNum(110,130,ds3231_month,2,WHITE,BLACK,24);
85     lcd_ShowIntNum(150,130,ds3231_year,2,WHITE,BLACK,24);
86 }
87 /* USER CODE END 4 */

```

Program 4.3: Ví dụ sử dụng thư viện.

Hình ảnh kết quả trên Kit thí nghiệm:



Hình 4.8: Kết quả hiển thị thực trên Kit thí nghiệm

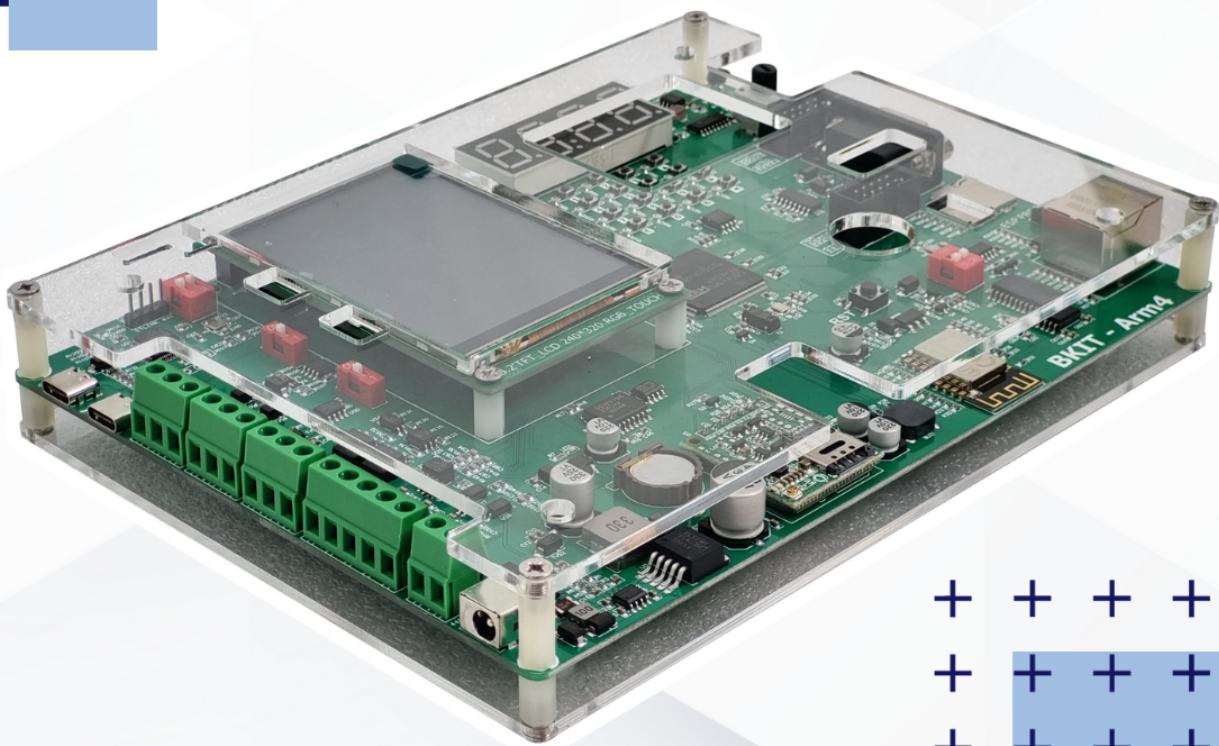
6 Bài tập và báo cáo

Xây dựng máy trạng thái và hiện thực hệ thống đồng hồ điện tử với các chức năng sau.

- Hệ thống bao gồm 3 chế độ:
 - Chế độ xem giờ: Màn hình LCD hiển thị các thông tin như thứ, ngày, tháng, năm, giờ, phút, giây theo một định dạng hợp lý.
 - Chế độ xem chỉnh giờ: Cho phép người dùng điều chỉnh lại các thông số của đồng hồ. Khi này màn hình đồng hồ sẽ ngừng chạy, khi đang chỉnh thông số nào thì thông số đó sẽ chớp tắt với tần số 2Hz trên màn hình. Có 2 nút nhấn sẽ được dùng trong chế độ này:
 - * Nút nhấn chỉnh thông số (gọi ý dùng **nút "mũi tên lên"**): khi nhấn thông số sẽ tăng lên 1, nếu nhấn giữ trong vòng 2s thì thông số sẽ tăng mỗi 200ms.
 - * Nút nhấn lưu thông số (gọi ý dùng **nút "E"**): nhấn để lưu thông số và chuyển sang thông số tiếp theo.
 - Chế độ hẹn giờ: Điều chỉnh tương tự chế độ chỉnh giờ. Khi đồng hồ đến giờ đã hẹn, hiển thị hiệu ứng báo động trên màn hình LCD.
- Sử dụng thêm một nút nhấn để chuyển đổi giữa các chế độ.
- Trên màn hình nên hiển thị thêm trạng thái hiện tại của hệ thống.

CHƯƠNG 5

Universal Asynchronous Receiver-Transmitter



1 Mục tiêu

- Tìm hiểu giao tiếp UART.
- Biết cách sử dụng module RS232 trên Kit thí nghiệm.
- Biết cách sử dụng phần mềm để giao tiếp giữa Kit thí nghiệm và máy tính thông qua RS232.

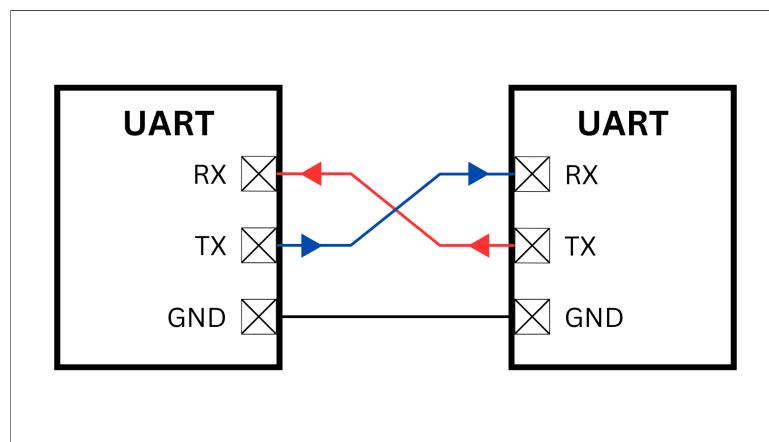
2 Giới thiệu

UART (Universal Asynchronous Receiver-Transmitter) là một giao thức truyền thông không đồng bộ đáng tin cậy và được sử dụng rộng rãi, thường được áp dụng cho việc liên lạc giữa các thiết bị và module như Wifi, Bluetooth, Xbee, module đọc thẻ RFID trong Raspberry Pi, Arduino và các vi điều khiển khác.

3 Cơ sở lý thuyết

3.1 UART

UART là giao thức truyền thông được sử dụng trong giao tiếp nối tiếp. Giao thức này thường được sử dụng trong truyền thông khoảng cách ngắn (trong cùng một thiết bị hoặc các thiết bị trên cùng bo mạch). Trong giao tiếp UART, hai thiết bị UART giao tiếp trực tiếp với nhau. UART chuyển đổi dữ liệu song song từ một thiết bị điều khiển thành dạng nối tiếp, truyền dữ liệu nối tiếp tới thiết bị nhận UART, sau đó chuyển đổi dữ liệu nối tiếp thành dữ liệu song song cho thiết bị nhận. Chỉ cần hai dây để truyền dữ liệu giữa hai thiết bị UART. Luồng dữ liệu từ chân Tx của truyền UART đến chân Rx của UART nhận:



Hình 5.1: Sơ đồ giao tiếp giữa 2 thiết bị qua UART

Khi thiết bị UART nhận diện được bit khởi đầu, nó bắt đầu đọc bit theo một tần số cụ thể, được gọi là baudrate. Baudrate là chỉ số đo tốc độ truyền dữ liệu, thể hiện bằng bit/giây (bps). Cả hai UART đều phải hoạt động ở cùng một tốc độ truyền. Sự khác biệt về tốc độ truyền giữa thiết bị truyền và thiết bị nhận chỉ có thể chênh lệch khoảng 10%. Cả hai thiết bị UART cũng cần được cấu hình để truyền và nhận cùng một cấu trúc gói dữ liệu.

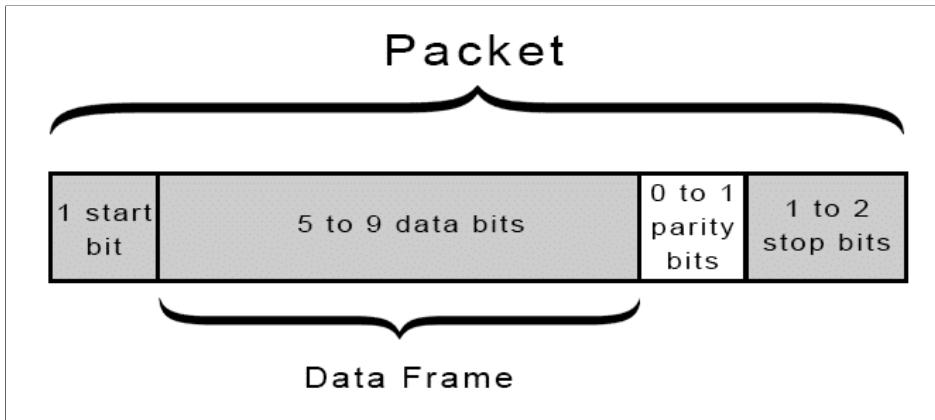
UART có thể được hiện thực theo một trong ba chế độ:

- Full duplex: Giao tiếp đồng thời đến và đi từ mỗi master và slave.
- Half duplex: Dữ liệu đi theo một hướng tại một thời điểm.
- Simplex: Chỉ giao tiếp một chiều.

Dữ liệu truyền UART được tổ chức thành các gói. Mỗi gói chứa 1 bit bắt đầu, 5 đến 9 bit dữ liệu, bit chẵn lẻ tùy chọn và 1 hoặc 2 bit dừng:

Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous?	Asynchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	1

Hình 5.2: Các điểm cần lưu ý về UART



Hình 5.3: Dataframe của UART

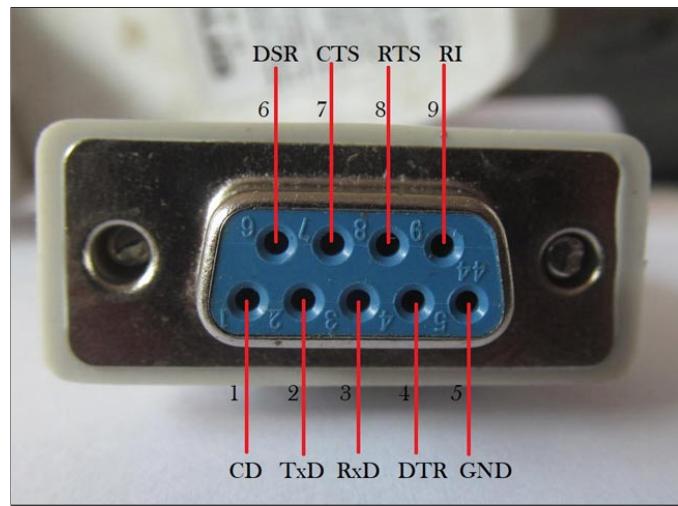
3.2 RS232

RS232 là một hình thức truyền dữ liệu nối tiếp. Nó được thiết kế để cho phép truyền và nhận dữ liệu theo cách nối tiếp, tức là dữ liệu được truyền từng bit một và theo thứ tự liên tiếp theo chuỗi thời gian.

RS232 cho phép giao tiếp full-duplex, tức là việc đường truyền và nhận dữ liệu có thể được sử dụng đồng thời. RS232 được dùng phổ biến để giao tiếp trong khoảng cách tương đối ngắn (tối đa khoảng 15m).

Cấu tạo của RS232:

- Thực tế là cổng RS232 có hai loại đầu nối: DB-25 và DB-9. DB-25 có 25 chân, phục vụ nhiều ứng dụng, nhưng không phải ứng dụng nào cũng cần dùng hết 25 chân. Do đó, đầu nối 9 chân được thiết kế để thuận lợi hơn cho việc kết nối các thiết bị.

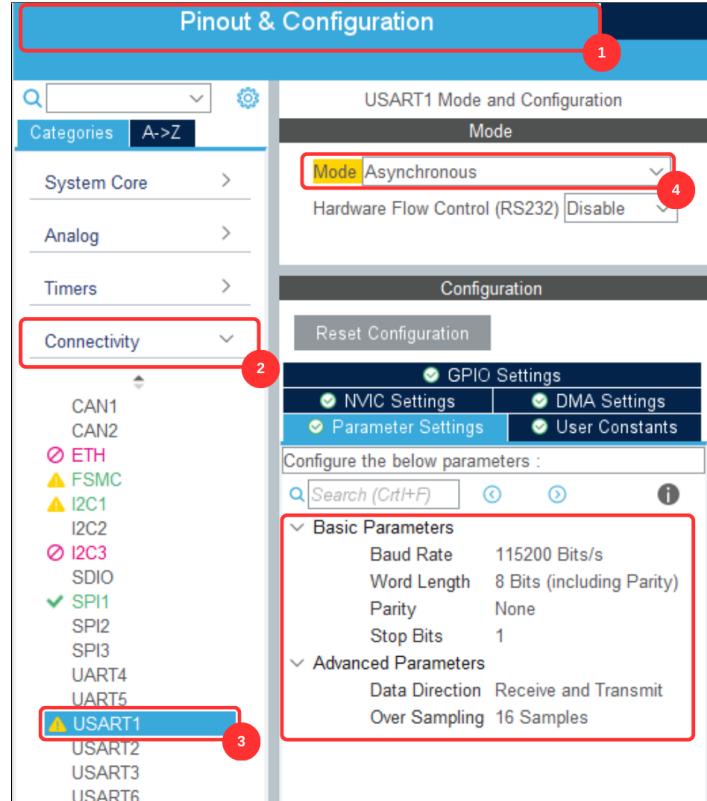


Hình 5.4: Cổng RS232 DB9

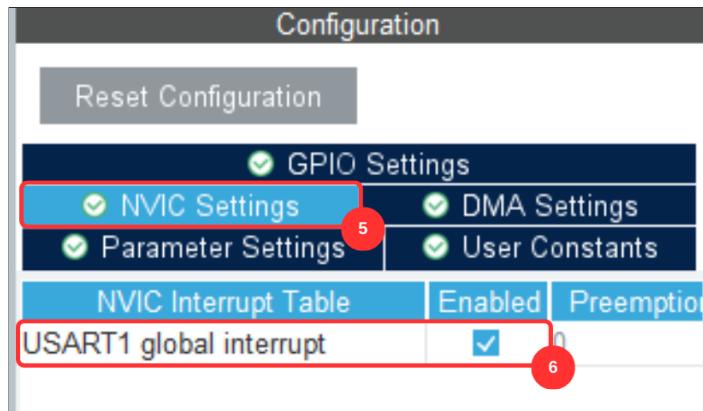
4 Hướng dẫn cấu hình

4.1 Cấu hình UART-RS232

Trên Kit thí nghiệm, module RS232 được điều khiển thông qua module UART1 trên MCU. Trong file .ioc chúng ta sẽ cấu hình USART1 như sau:

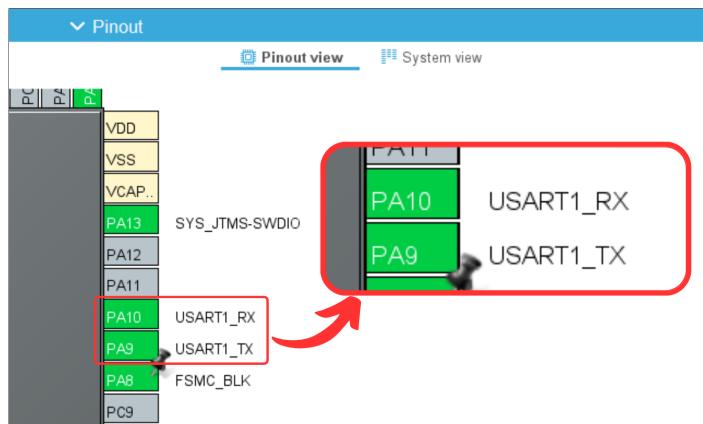


Hình 5.5: Config USART1



Hình 5.6: Config UART1 (tiếp theo)

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



Hình 5.7: Pinview out sau khi cấu hình UART1

5 Hướng dẫn lập trình

5.1 Thư viện uart.h

Các file **uart.h** và **uart.c** có thể được sao chép từ project mẫu **Bai5_UART**.

void uart_init_rs2320

- **Mô tả:** Khởi tạo RS232.
- **Tham số:** Không có
- **Giá trị trả về:** Không có.

void uart_Rs232SendString(uint8_t* str)

- **Mô tả:** Gửi chuỗi ký tự qua RS232.

- **Tham số:**

- **str:** Chuỗi ký tự cần gửi.

- **Giá trị trả về:** Không có.

void uart_Rs232SendBytes(uint8_t* bytes, uint16_t size)

- **Mô tả:** Gửi chuỗi byte qua RS232.

- **Tham số:**

- **bytes:** Mảng chứa dữ liệu.

- **size:** Độ dài dữ liệu.

- **Giá trị trả về:** Không có.

void uart_Rs232SendNum(uint32_t num)

- **Mô tả:** Gửi số dưới dạng ký tự qua RS232.

- **Tham số:**

- **num:** Số cần gửi.

- **Giá trị trả về:** Không có.

Ngoài ra ta có hàm **HAL_UART_RxCpltCallback** là hàm ngắn (interrupt service routine) sẽ được gọi mỗi khi nhận được dữ liệu. Sau này chúng ta có thể sửa đổi hàm này tùy theo mục đích xử lý dữ liệu nhận vào. Trong ví dụ hiện tại, hàm này sẽ gửi lại dữ liệu được nhận.

```

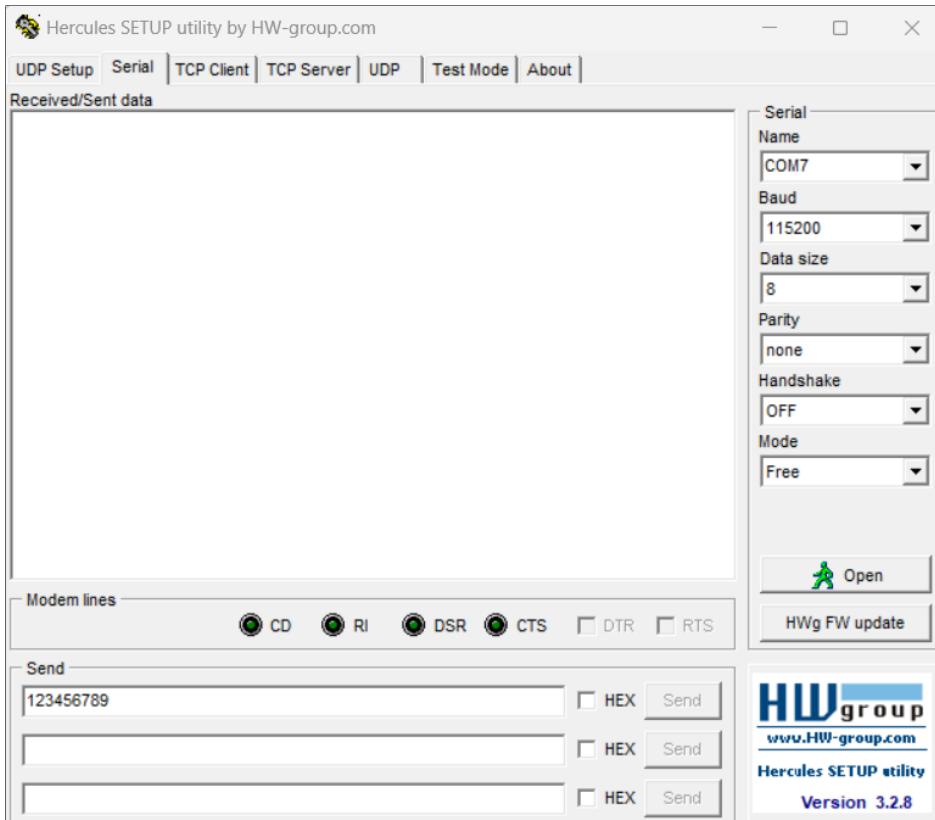
1 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
2     if(huart->Instance == USART1){
3         // rs232 isr
4         // can be modified
5         HAL_UART_Transmit(&huart1, &receive_buffer1, 1, 10);
6
7         // turn on the receive interrupt
8         HAL_UART_Receive_IT(&huart1, &receive_buffer1, 1);
9     }
10 }
```

Program 5.1: Hàm interrupt

Trong bài thí nghiệm này, cổng RS232 trên Kit thí nghiệm sẽ được kết nối với máy tính. Để thể hiện được dữ liệu nhận và truyền trên máy tính, ta cần sử dụng một trong các phần mềm hỗ trợ sau:

- Serial Debug Assistant (tải tại Microsoft Store).
- Hercules Terminal (tải tại đây).

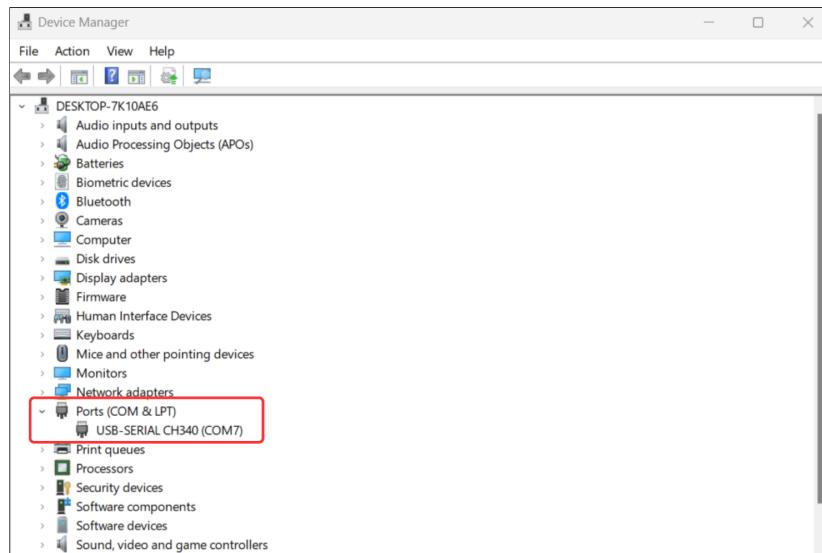
Phần hướng dẫn này sẽ sử dụng phần mềm **Hercules Terminal**.



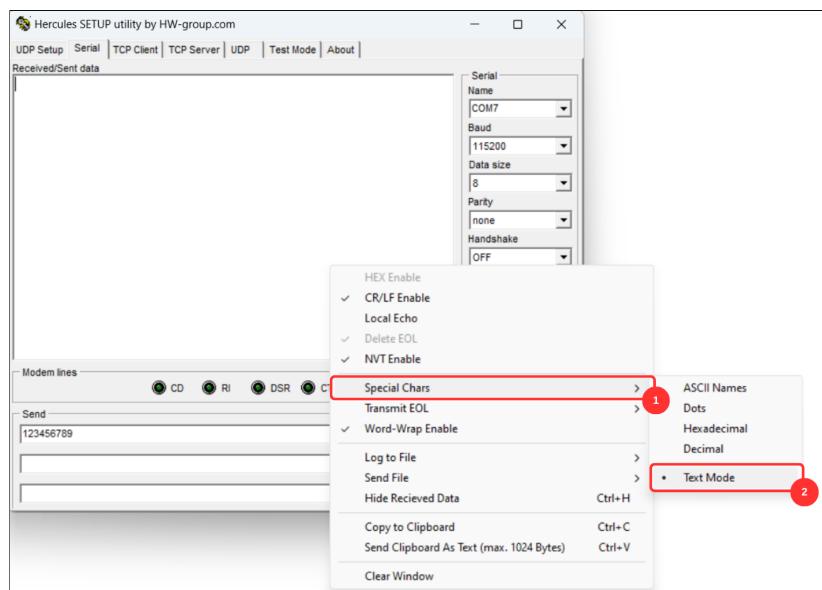
Hình 5.8: Giao diện phần mềm Hercules Terminal

Để phần mềm có thể gửi và nhận dữ liệu, ta cần chọn cổng COM đúng với cổng đang kết nối với dây cáp RS232. Ta cần phải kiểm tra cổng COM đang sử dụng trong **Device Manager** (Window + X -> Device Manager).

Như hình trên ta đang sử dụng cổng COM7. Ta tiến hành điều chỉnh các thông số **Name** thành COM7, thông số **Baud**, **Data size**, **Parity** giống với những gì ta đã config UART trên Kit thí nghiệm. Sau đó chọn **Open**. Ta cũng cần phải chỉnh chế độ hiển thị dữ liệu để có thể dễ dàng quan sát.



Hình 5.9: Kiểm tra cổng COM



Hình 5.10: Chỉnh chế độ hiển thi

Bây giờ chúng ta sẽ hiện thực gửi UART với dữ liệu là thời gian lấy được từ module DS3231 mỗi khi nút "E" trên ma trận phím được nhấn. Đầu tiên, chúng ta khởi tạo module UART-RS232 trong **system_init()** bằng hàm **uart_rs232_init()**. Sau đó, một hàm mới **test_Uart()** được tạo và gọi mỗi 50ms. Trong hàm này ta sẽ bắt tín hiệu nút nhấn "E" trên Kit thí nghiệm như đã hiện thực ở lab 3. Mỗi khi nút "E" được nhấn, ta sẽ gọi các hàm để gửi dữ liệu giờ, phút, giây theo định dạng **<giờ>:<phút>:<giây>** sang máy tính thông qua module UART-RS232 (dữ liệu về thời gian sẽ được đọc từ đồng hồ thời gian thực như ở lab 4). Để thuận tiện cho việc quan sát dữ liệu trên màn hình máy tính, ta gửi thêm kí tự xuống dòng ('\n').

1 // ...

```

2
3 /* USER CODE BEGIN Includes */
4 #include "software_timer.h"
5 #include "led_7seg.h"
6 #include "button.h"
7 #include "lcd.h"
8 #include "picture.h"
9 #include "ds3231.h"
10 #include "uart.h"
11 /* USER CODE END Includes */
12
13 // ...
14
15 /* USER CODE BEGIN PFP */
16 void system_init();
17 void test_LedDebug();
18 void test_Uart();
19 /* USER CODE END PFP */
20
21 /* Private user code
22 -----
23 */
24 /* USER CODE BEGIN 0 */
25
26 /**
27 * @brief The application entry point.
28 * @retval int
29 */
30 int main(void)
31 {
32 // ...
33 /* USER CODE BEGIN 2 */
34 system_init();
35 /* USER CODE END 2 */
36
37 /* Infinite loop */
38 /* USER CODE BEGIN WHILE */

```

```

39 while (1)
40 {
41     /* USER CODE END WHILE */
42     while(!flag_timer2);
43     flag_timer2 = 0;
44     button_Scan();
45     test_LedDebug();
46     ds3231_ReadTime();
47     test_Uart();
48     /* USER CODE BEGIN 3 */
49 }
50 /* USER CODE END 3 */
51 }
52
53 // ...
54
55 /* USER CODE BEGIN 4 */
56 void system_init(){
57     timer_init();
58     led7_init();
59     button_init();
60     lcd_init();
61     uart_init_rs232();
62     setTimer2(50);
63 }
64
65 uint16_t count_led_debug = 0;
66
67 void test_LedDebug(){
68     count_led_debug = (count_led_debug + 1)%20;
69     if(count_led_debug == 0){
70         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin);
71     }
72 }
73
74 void test_Uart(){
75     if(button_count[12] == 1){
76         uart_Rs232SendNum(ds3231_hours);
77         uart_Rs232SendString(":");

```

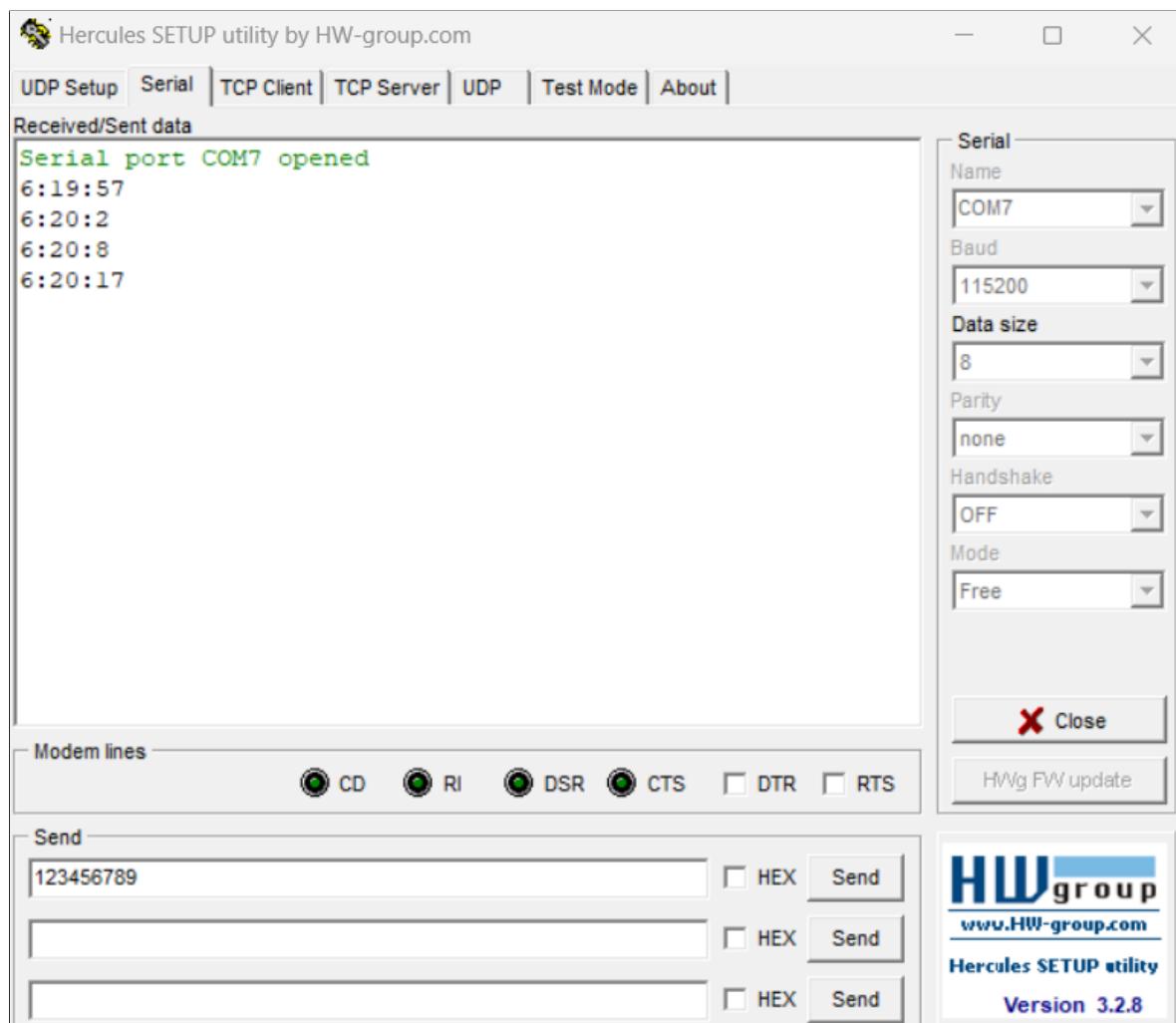
```

78     uart_Rs232SendNum(ds3231_min);
79     uart_Rs232SendString(":");
80     uart_Rs232SendNum(ds3231_sec);
81     uart_Rs232SendString("\n");
82 }
83 */
84 /* USER CODE END 4 */
85 // ...

```

Program 5.2: main.c

Kết quả:



Hình 5.11: Kết quả thu được từ ví dụ mẫu

6 Bài tập và báo cáo

6.1 Bài tập 1

Sử dụng ring buffer để lưu trữ các dữ liệu nhận được trong **Uart Receive Interrupt**. Sau đó bật cờ và xử lí dữ liệu nhận được trong màn main.

6.2 Bài tập 2

Nâng cấp bài tập về đồng hồ điện tử đã được hiện thực ở lab 4. Thêm một chế độ cho phép cập nhật thời gian thông qua giao tiếp RS232 với máy tính. Ở chế độ này, các giá trị thời gian sẽ được cập nhật lần lượt. Sau đây là một ví dụ khi hệ thống yêu cầu cập nhật giờ:

- Màn hình LCD sẽ hiển thị thêm dòng chữ "Updating hours ..."
- Kit thí nghiệm sẽ gửi dữ liệu **request** "Hours" đến máy tính.
- Máy tính sẽ gửi lại dữ liệu **response** là chuỗi kí tự chứa giá trị giờ muốn cập nhật. Gợi ý: sử dụng chế độ gửi dữ liệu của phần mềm **Hercules**.
- Hệ thống lưu lại giá trị vừa nhận được và chuyển sang yêu cầu cập nhật giá trị tiếp theo.
- Sau khi hệ thống đã nhận được tất cả giá trị thời gian, hệ thống sẽ lưu lại giá trị đó vào IC thời gian thực DS3231.

6.3 Bài tập 3

Tiếp tục phát triển ứng dụng ở bài tập 2, thêm các chức năng sau:

- Tại thời điểm 10s sau khi hệ thống gửi request đến máy tính, nếu không có phản hồi thì hệ thống sẽ gửi lại request. Nếu sau 3 lần gửi mà không được phản hồi, hệ thống sẽ báo lỗi thông qua LCD và quay về chế độ hoạt động bình thường.
- Nếu dữ liệu response từ máy tính không hợp lệ, hệ thống sẽ gửi lại request.