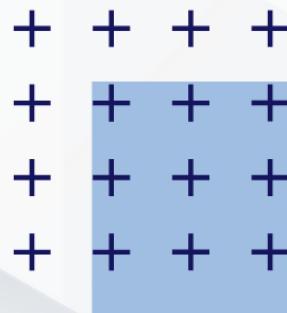
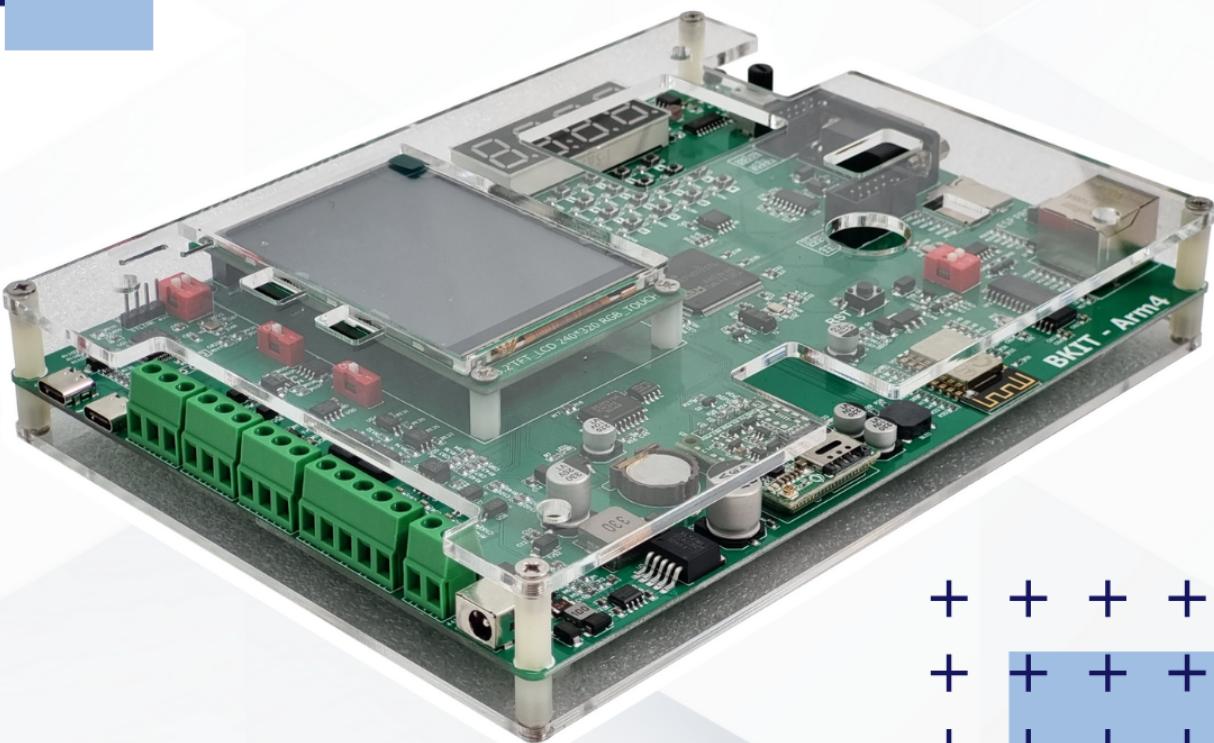


# KIT THÍ NGHIỆM BKIT

# ARM4





---

# Mục lục

---

<b>Chapter 1. General Purpose Input Output</b>	<b>5</b>
1    Mục tiêu . . . . .	6
2    Giới thiệu . . . . .	6
3    Project đầu tiên trên STM32Cube . . . . .	7
4    Cơ sở lý thuyết . . . . .	12
4.1    GPIO Output Mode . . . . .	12
4.2    GPIO Input Mode . . . . .	13
5    Hướng dẫn config . . . . .	14
6    Bài tập và Báo cáo . . . . .	20
6.1    Bài tập 1 . . . . .	20
6.2    Bài tập 2 . . . . .	20
6.3    Bài tập 3 . . . . .	20
 <b>Chapter 2. Timer Interrupt and LED Scanning</b>	 <b>21</b>
1    Mục tiêu . . . . .	22
2    Giới thiệu . . . . .	22
3    Cấu hình timer . . . . .	23
4    Cấu hình LED đồng hồ . . . . .	25
4.1    Giao tiếp SPI . . . . .	25
4.2    IC 74HC595 . . . . .	26
4.3    Config chân cho vi điều khiển . . . . .	27
5    Hướng dẫn lập trình . . . . .	30
6    Bài tập và Báo cáo . . . . .	35

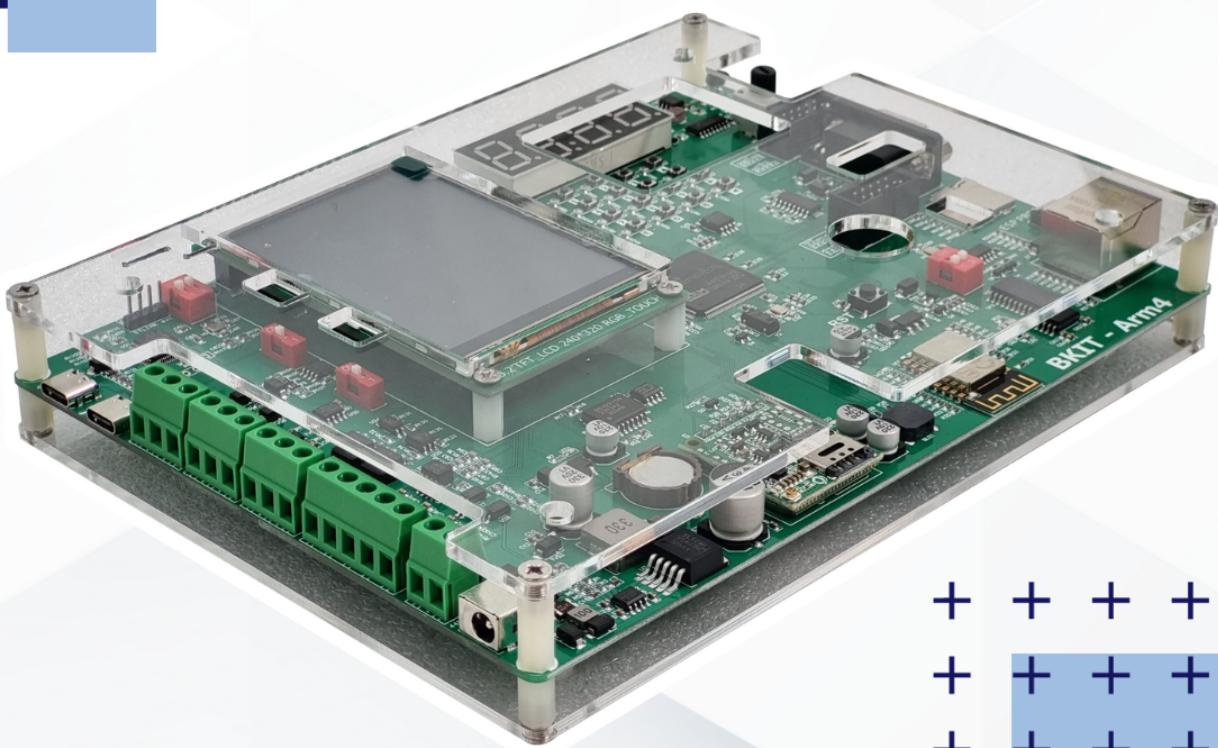
6.1	Bài tập 1 . . . . .	35
6.2	Bài tập 2 . . . . .	36
6.3	Bài tập 3 . . . . .	36
6.4	Bài tập 4 . . . . .	36
6.5	Bài tập 5 . . . . .	36

# CHƯƠNG 1

---

## General Purpose Input Output

---



# 1 Mục tiêu

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

# 2 Giới thiệu

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



Hình 1.1: Phần mềm STM32Cube IDE

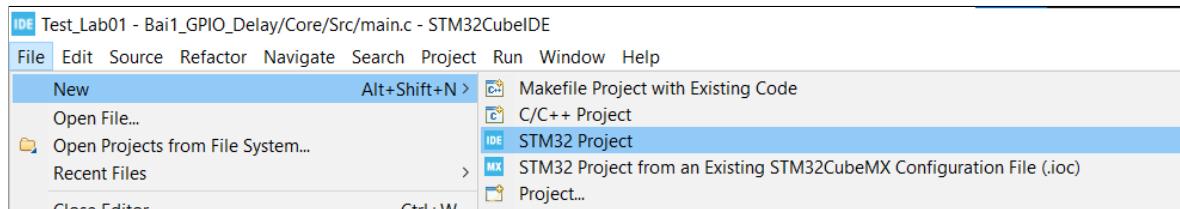
Điều thú vị nhất của STM32CubeIDE là sau khi chọn một MPU hoặc MCU STM32, hay một bộ vi điều khiển, bộ vi xử lý được config sẵn từ việc chọn bo mạch, mã khởi tạo sẽ được tạo tự động. Bất cứ khi nào trong quá trình lập trình, người dùng đều có thể quay lại quá trình khởi tạo và thay đổi config của các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

[https://ubc.sgp1.digitaloceanspaces.com/BKU\\_Softwares/STM32/stm32cubeide\\_1.7.0.zip](https://ubc.sgp1.digitaloceanspaces.com/BKU_Softwares/STM32/stm32cubeide_1.7.0.zip)

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

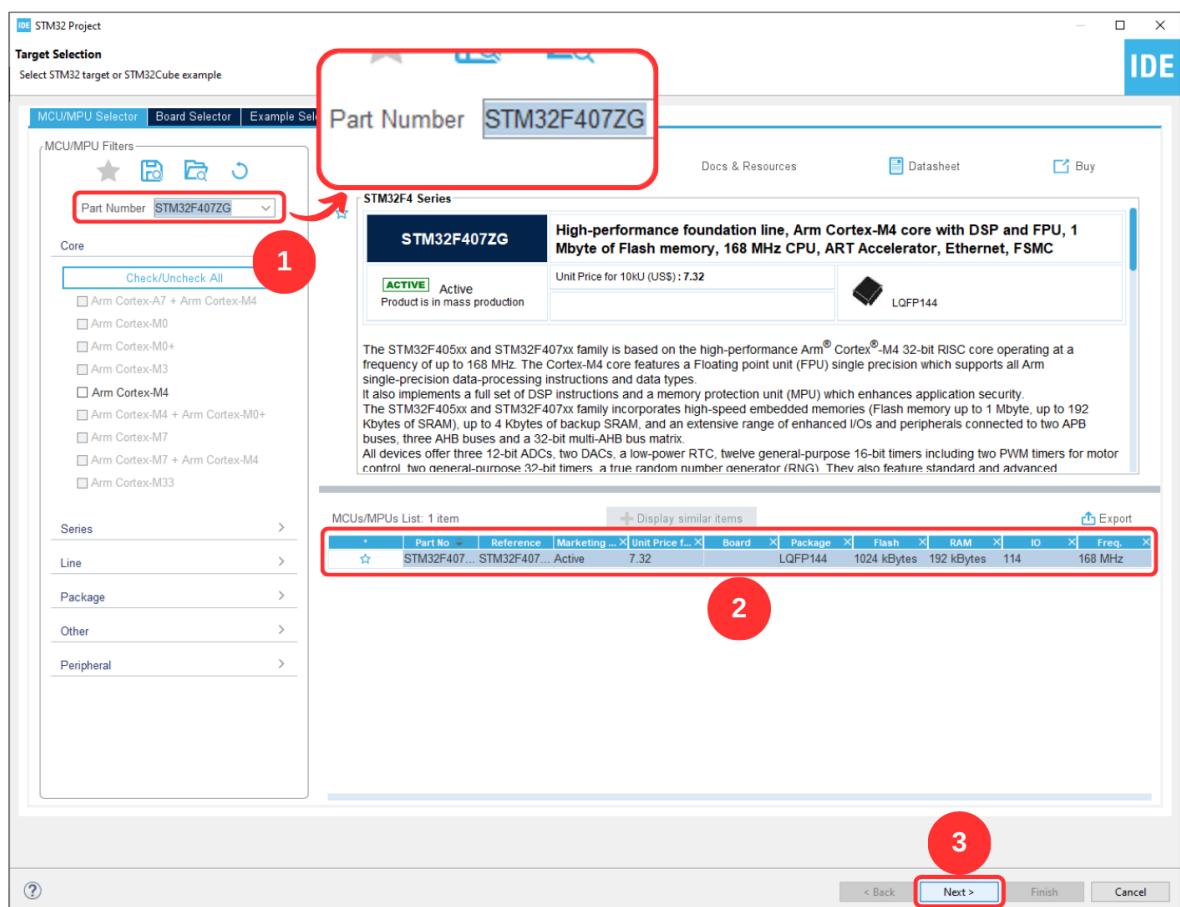
### 3 Project đầu tiên trên STM32Cube

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



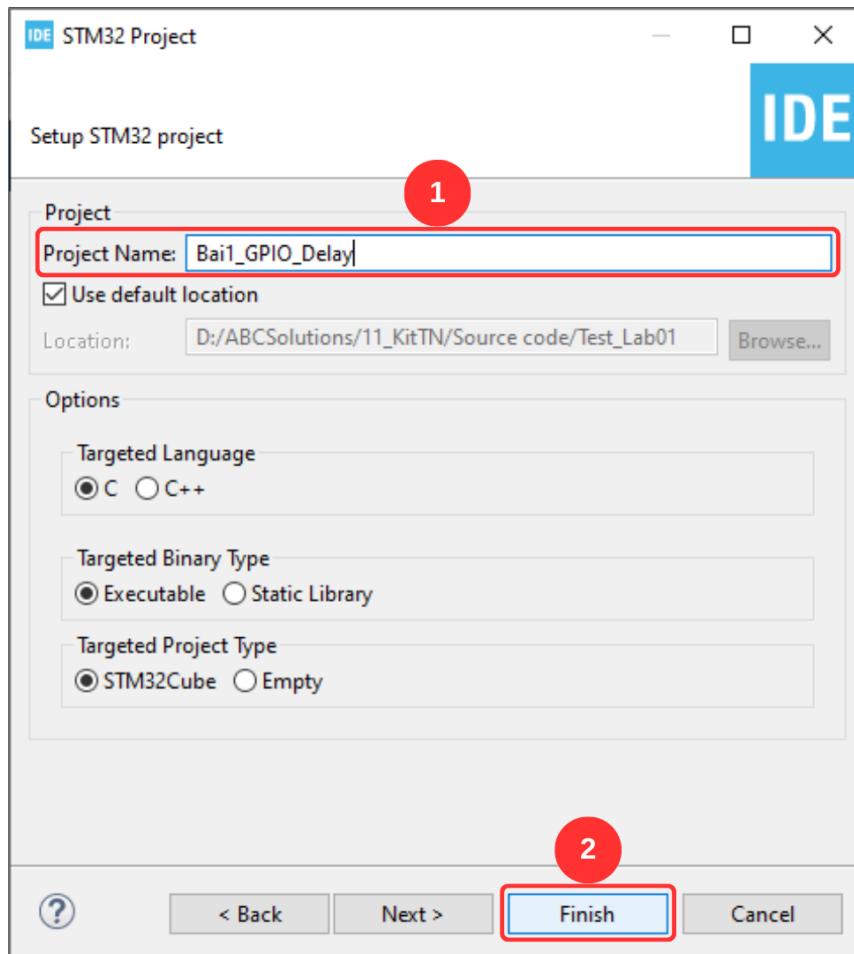
Hình 1.2: Tạo một project mới trên STM32CubeIDE

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



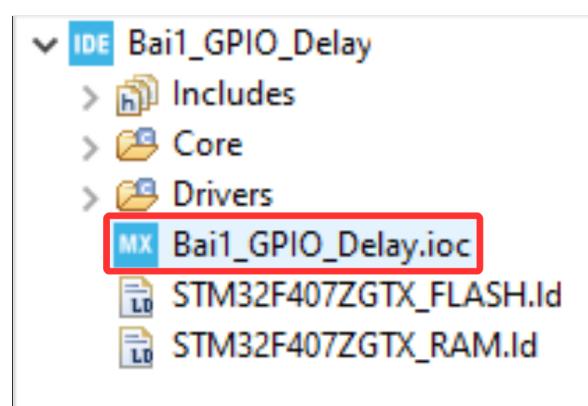
Hình 1.3: Tìm kiếm chip

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



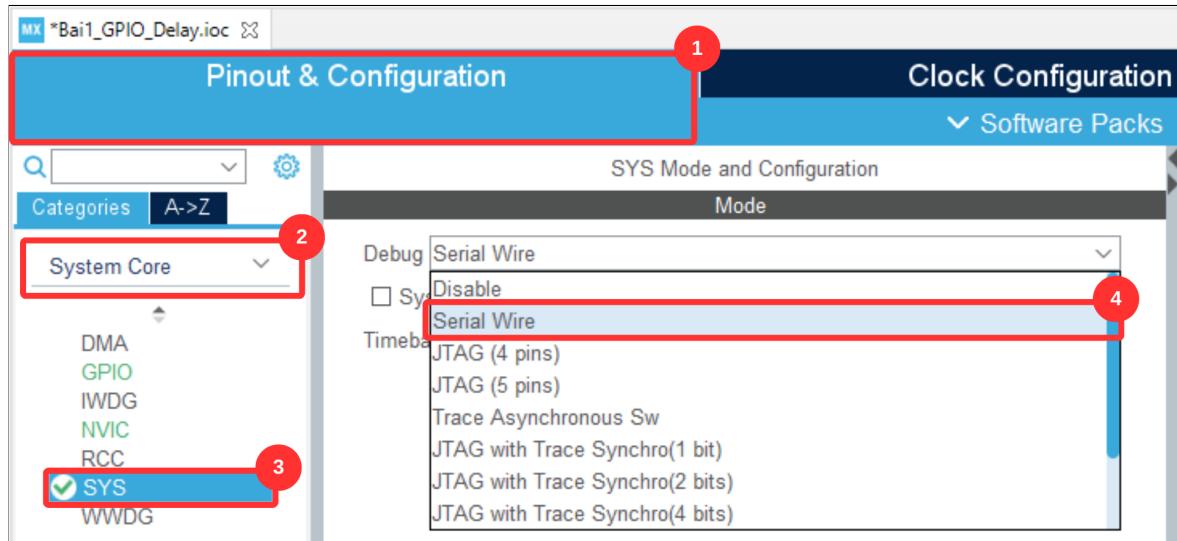
Hình 1.4: Đặt tên và tìm vị trí lưu project

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



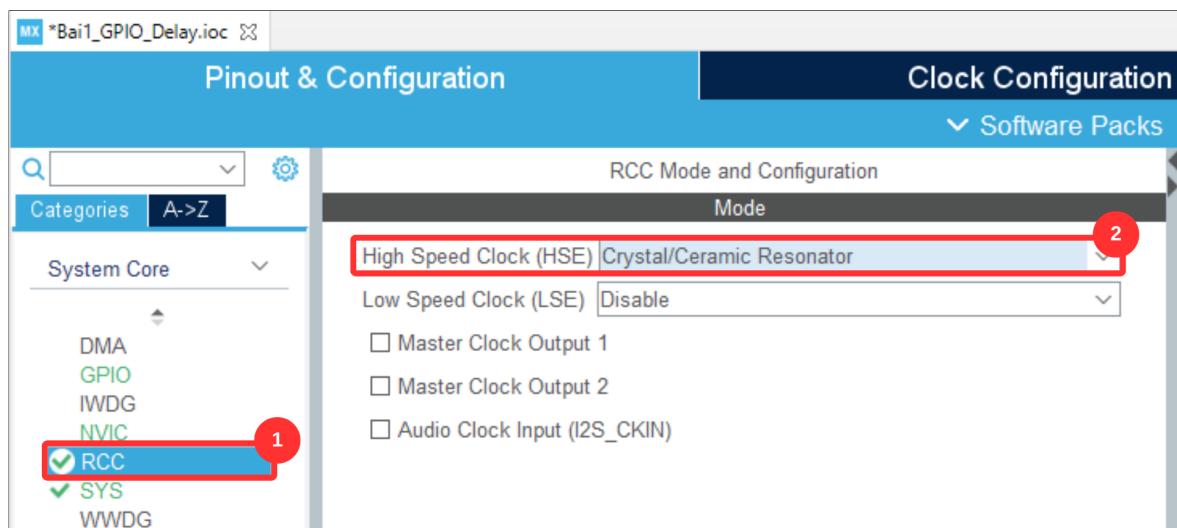
Hình 1.5: File config các chân của vi điều khiển

### Bước 5: Điều chỉnh phương thức gõ lỗi:



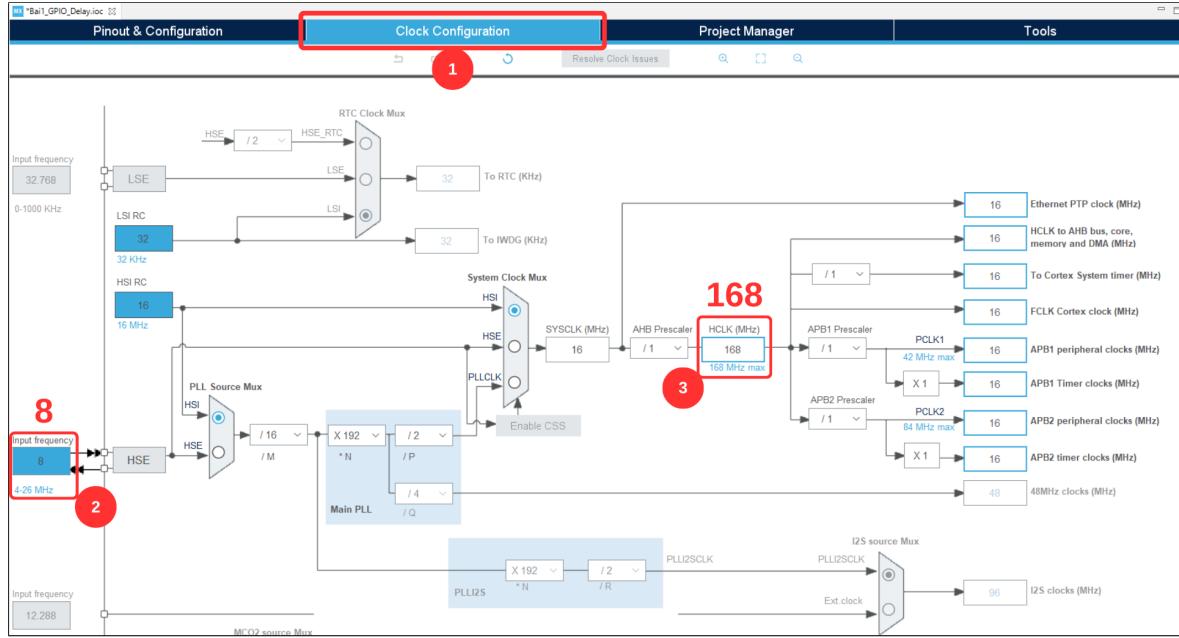
Hình 1.6: Chọn phương thức gõ lỗi

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



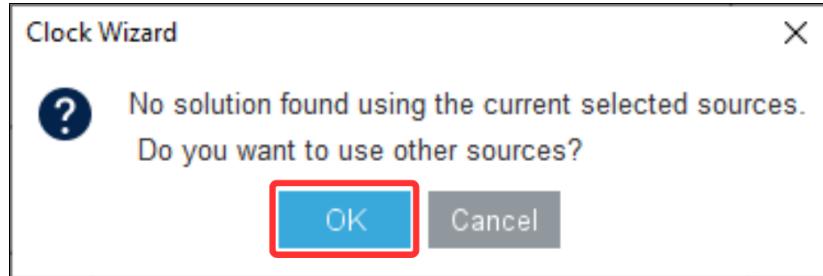
Hình 1.7: Config thạch anh ngoài

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



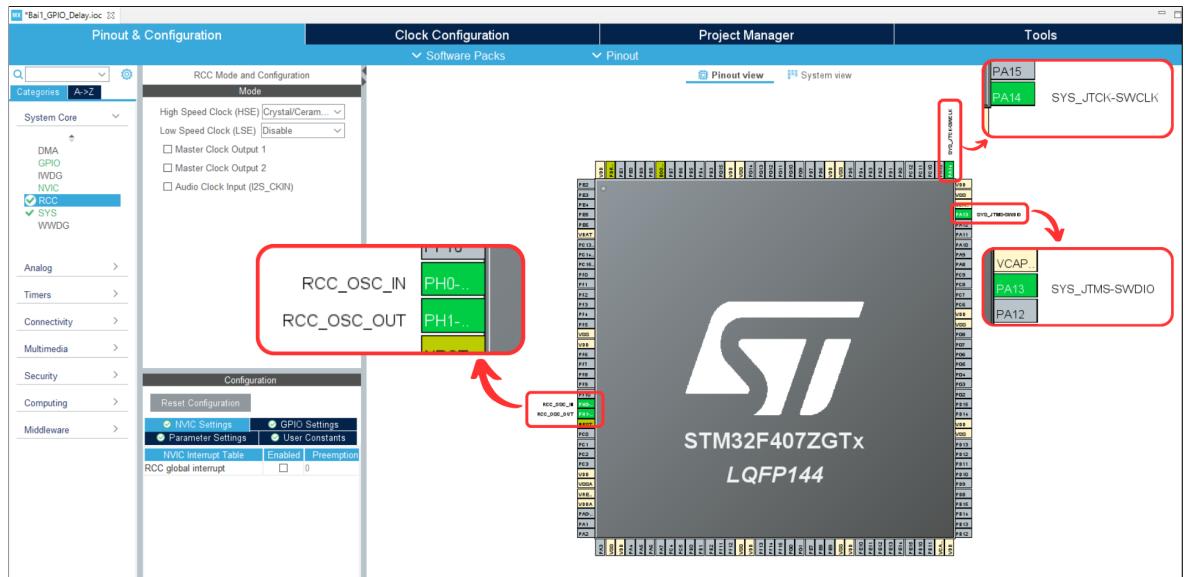
Hình 1.8: Config thạch anh ngoài

Chọn OK và đợi tính toán:



Hình 1.9: Config thạch anh ngoài

Sau khi config xong thì có 4 chân được config như hình 1.10. Sau đó, bấm tổ hợp phím **Ctrl + S** để phần mềm sinh code.



Hình 1.10: Vị trí điều khiển sau khi config

## 4 Cơ sở lý thuyết

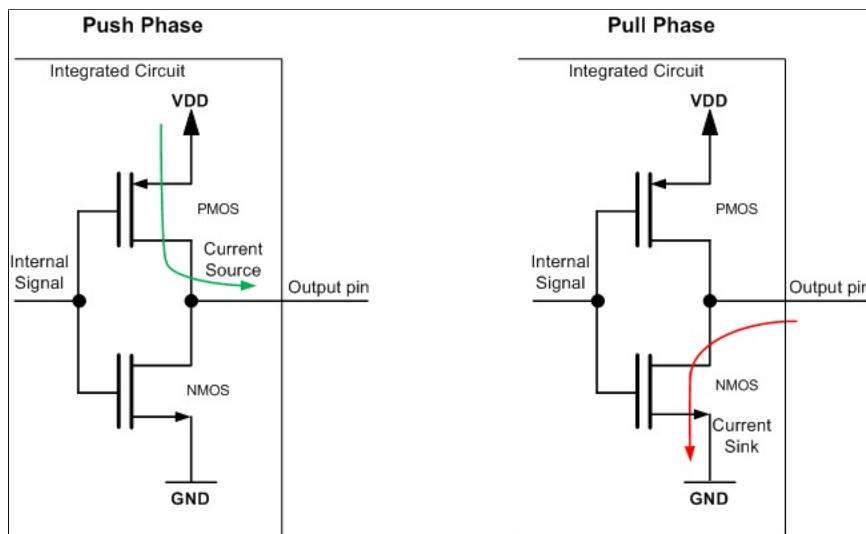
Trong phần này, chúng ta sẽ tìm hiểu về GPIO (viết tắt của General Purpose Input Output), là một thuật ngữ phổ biến trong lĩnh vực nhúng. Tín hiệu trên các chân GPIO cho phép thực hiện chức năng và giao tiếp với các thiết bị bên ngoài.

### 4.1 GPIO Output Mode

GPIO được sử dụng để truyền tín hiệu điện (cao hoặc thấp) đến chân khi nó được config làm output. Có hai tùy chọn config chủ yếu:

- **Push-pull**

- Trạng thái này là trạng thái mặc định của chế độ đầu ra GPIO. Chân này có thể “push” tín hiệu lên cao hoặc “pull” tín hiệu xuống thấp bằng cách sử dụng transistor PMOS hoặc transistor NMOS.
- Không cần điện trở kéo lên hay kéo xuống vì các transistor đã thực hiện công việc đó.

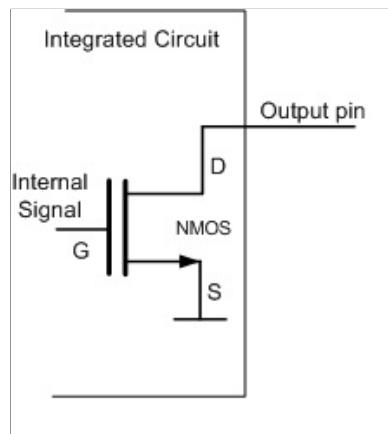


Hình 1.11: Sơ đồ config của push-pull

- **Open-drain**

- Trong Output mode, có thể sử dụng transistor PMOS và NMOS để tạo bộ đệm đầu ra. Khi loại bỏ transistor PMOS, trạng thái này sẽ trở thành "open-drain".
- Tên gọi "open-drain" xuất phát từ việc MOSFET không có kết nối nội bộ với bất kỳ điểm nào.

- Khi ta đưa NMOS lên mức cao, nó sẽ cung cấp một mức đất (GND), đồng thời đầu ra GPIO sẽ ở mức thấp.
- Khi tắt NMOS, đầu ra GPIO sẽ ở trạng thái nổi, không kết nối hoặc nối với Vcc hoặc GND. Do đó, đầu ra có thể là mức thấp hoặc mức cao trớn (nổi), có khả năng kéo chân xuống mức đất, nhưng không thể đẩy lên mức cao.
- Chế độ open-drain thường được sử dụng trong các giao diện truyền thông, nơi nhiều thiết bị được kết nối trên cùng một đường truyền (ví dụ: I2C, One-Wire). Khi tắt cả các đầu ra của các thiết bị kết nối với đường truyền ở trạng thái Hi-Z (không kết nối), đường truyền được đẩy đến một mức logic mặc định 1 bằng cách sử dụng trở kéo lên. Bất kỳ thiết bị nào cũng có thể pull xuống mức logic 0 bằng cách sử dụng open-drain và tắt cả các thiết bị có thể nhận thấy mức này.



Hình 1.12: Sơ đồ config của open-drain

## 4.2 GPIO Input Mode

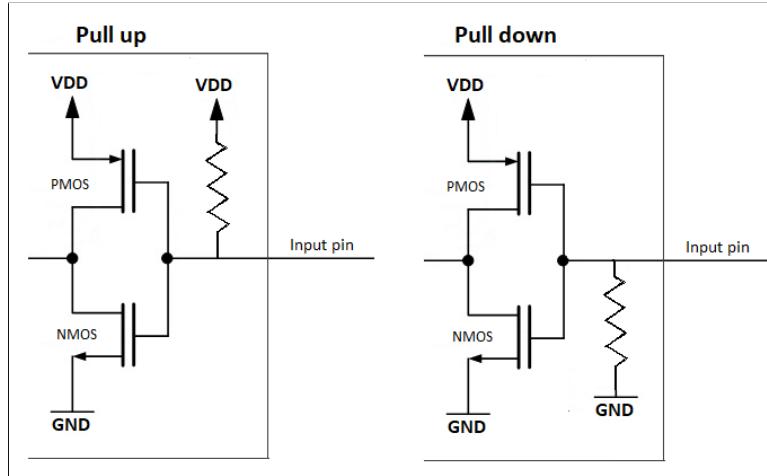
Để tránh hiện tượng thả nổi và đảm bảo rằng chân input có giá trị xác định khi không được kết nối với bất kỳ thiết bị nào, có hai chế độ để xác định giá trị ban đầu của một chân input: pull up và pull down.

- Pull up

- Điện trở kéo lên bên trong được kết nối với chân vi điều khiển. Vì vậy, trạng thái sẽ ở mức Cao trừ khi sử dụng điện trở kéo xuống bên ngoài.
- Việc sử dụng điện trở kéo lên có thể được config bởi người dùng.

- Pull down

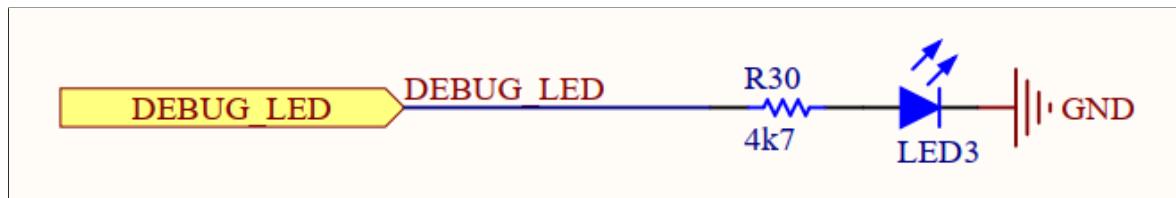
- Điện trở kéo xuống bên trong được kết nối với chân vi điều khiển. Vì vậy, trạng thái sẽ ở mức Thấp trừ khi sử dụng điện trở kéo lên bên ngoài.
- Việc sử dụng điện trở kéo xuống có thể được config bởi người dùng.



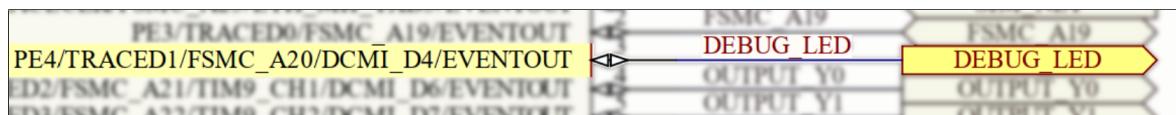
Hình 1.13: Sơ đồ config của input

## 5 Hướng dẫn config

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



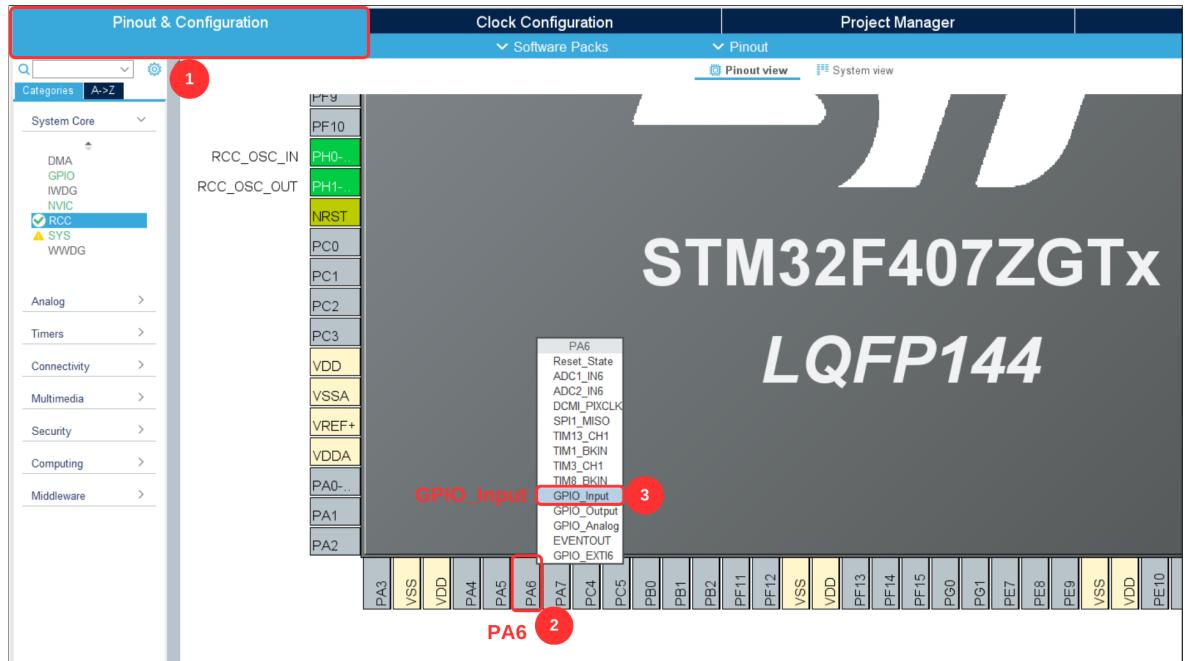
Hình 1.14: Sơ đồ nguyên lý của LED3



Hình 1.15: MCU điều khiển LED3 bằng chân PE4

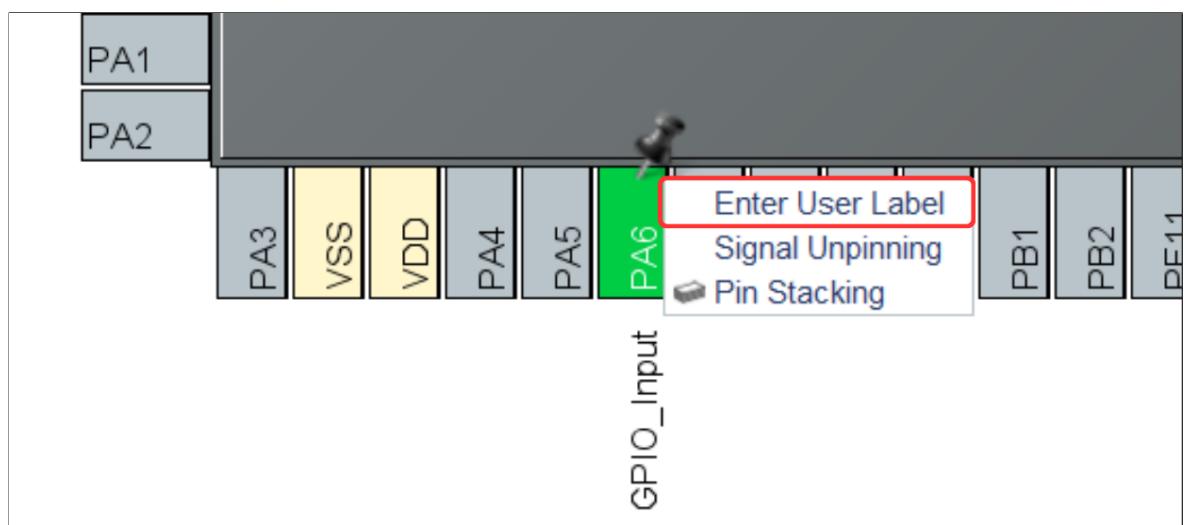
Ngoài ra, ta cũng có thể luyện tập config trước các chân ở khối input X0->X3 và khối output Y0,Y1. Lưu ý: Các khối input X và output Y trên kit thí nghiệm được thiết kế thông qua opto, dùng để tương tác với các thiết bị công suất lớn. Tương ứng với mỗi input và output nói trên, kit thí nghiệm sẽ có một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

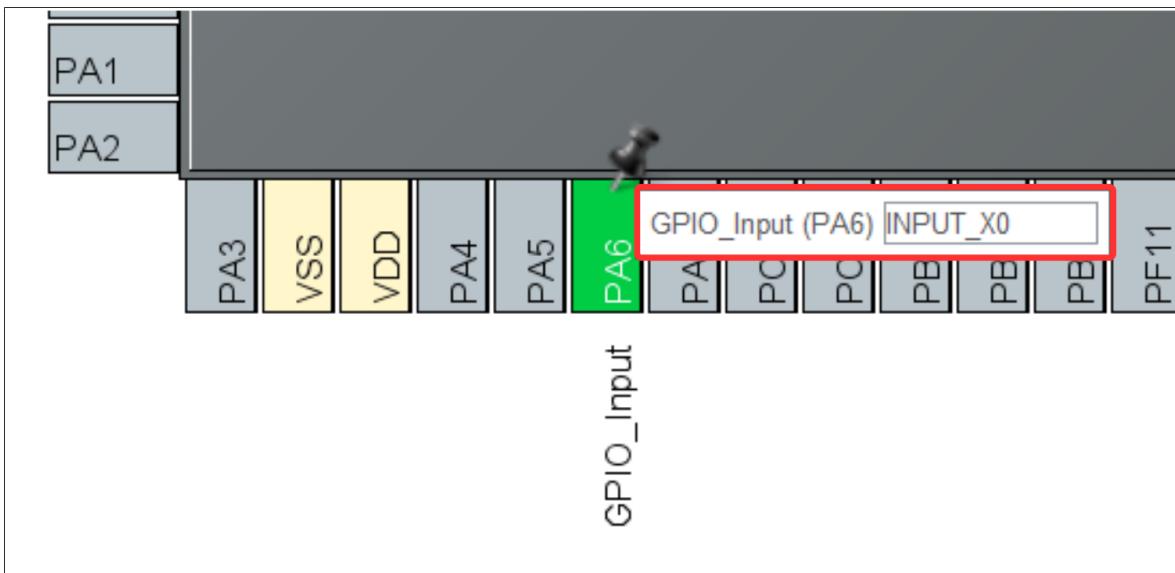
Để config các chân của vi điều khiển, ta mở file cấu hình (.ioc) và chọn vào cửa sổ **Pinout & Configuration**. Tiếp đó nhấp chọn chuột trái vào chân muốn config. Để config input, ta chọn **GPIO\_Input**.



Hình 1.16: Config PA6 thành chân input

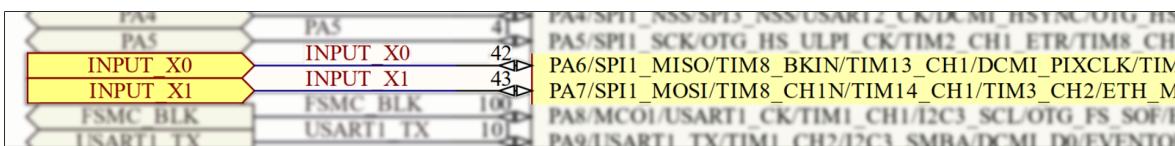
Tiếp theo, ta có thể đặt tên mới cho các chân này để thuận tiện cho việc lập trình. Đặt tên mới bằng cách nhấp chuột phải vào chân muốn đặt tên. Sau đó chọn Enter User Label và nhập tên mới.



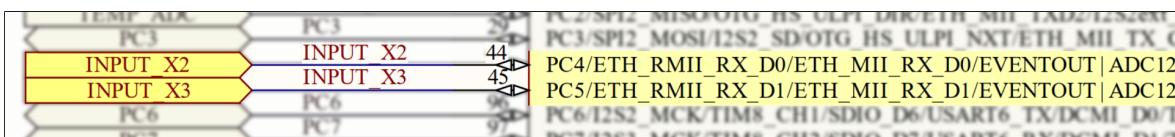


Hình 1.18: Nhập tên mới cho chân input PA6 là INPUT\_X0

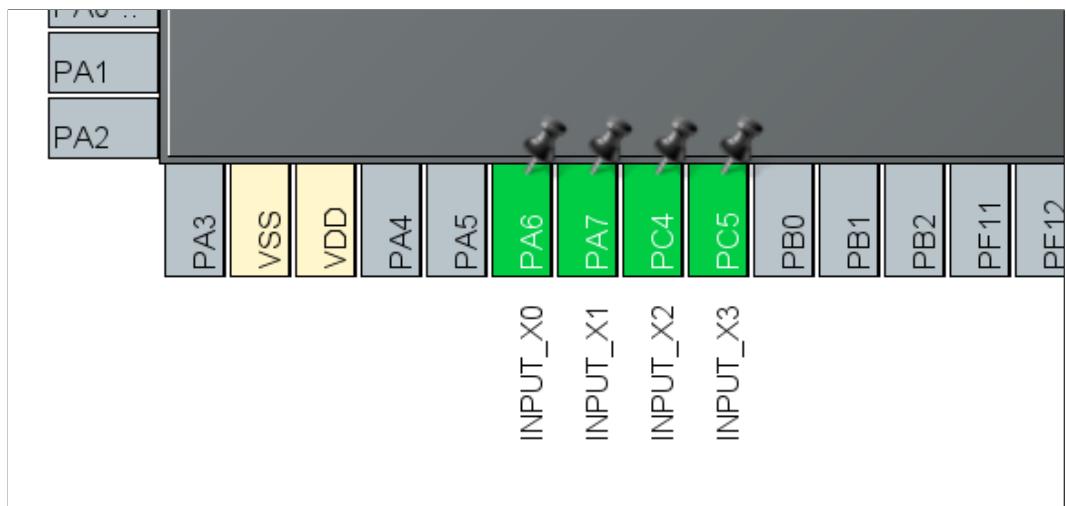
Thực hiện tương tự với các chân input còn lại.



Hình 1.19: Sơ đồ nguyên lý của các chân input X0 và X1

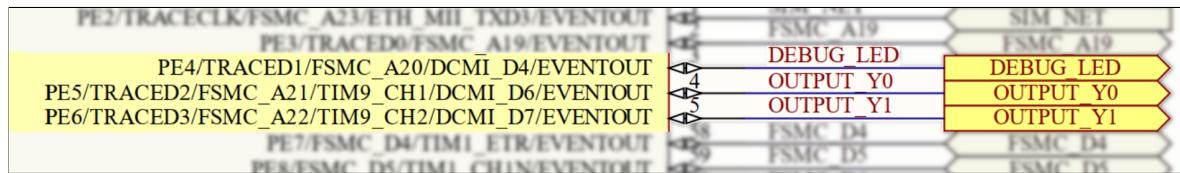


Hình 1.20: Sơ đồ nguyên lý của các chân input X2 và X3

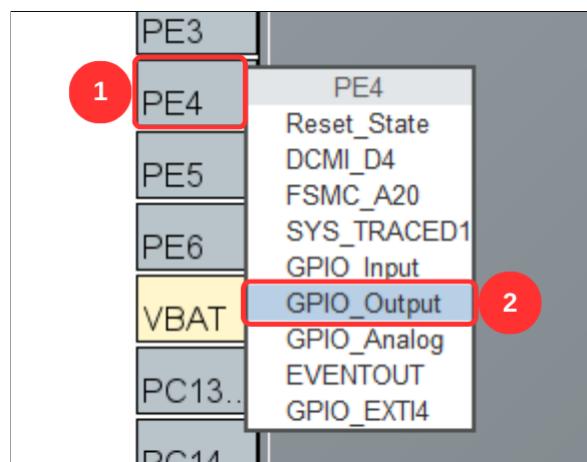


Hình 1.21: Các chân input sau khi được config và đặt lại tên

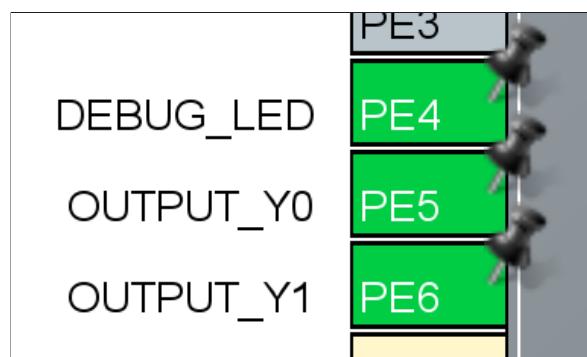
Tương tự như vậy, ta có thể config các chân output.



Hình 1.22: Sơ đồ nguyên lý của các chân output DEBUG\_LED, Y0 và Y1



Hình 1.23: Config PE4 thành chân output



Hình 1.24: Các chân output sau khi được config và đặt lại tên

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

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

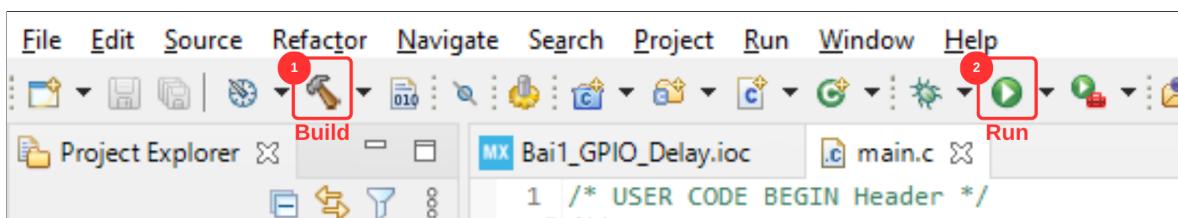
```

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

Program 1.1: Project chớp tắt led đầu tiên

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

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



Hình 1.25: Build và Run chương trình

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

```

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

Program 1.2: Ví dụ chương trình chớp tắt LED

Sau khi nạp chương trình, ta được kết quả LED3 trên kit thí nghiệm sẽ chớp tắt mỗi 1 giây. Đoạn code trên có thể được chỉnh sửa để điều khiển output Y0, Y1, hoặc có thể tham khảo project mẫu **Bai1\_GPIO\_Delay**.

## **6 Bài tập và Báo cáo**

### **6.1 Bài tập 1**

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

### **6.2 Bài tập 2**

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

### **6.3 Bài tập 3**

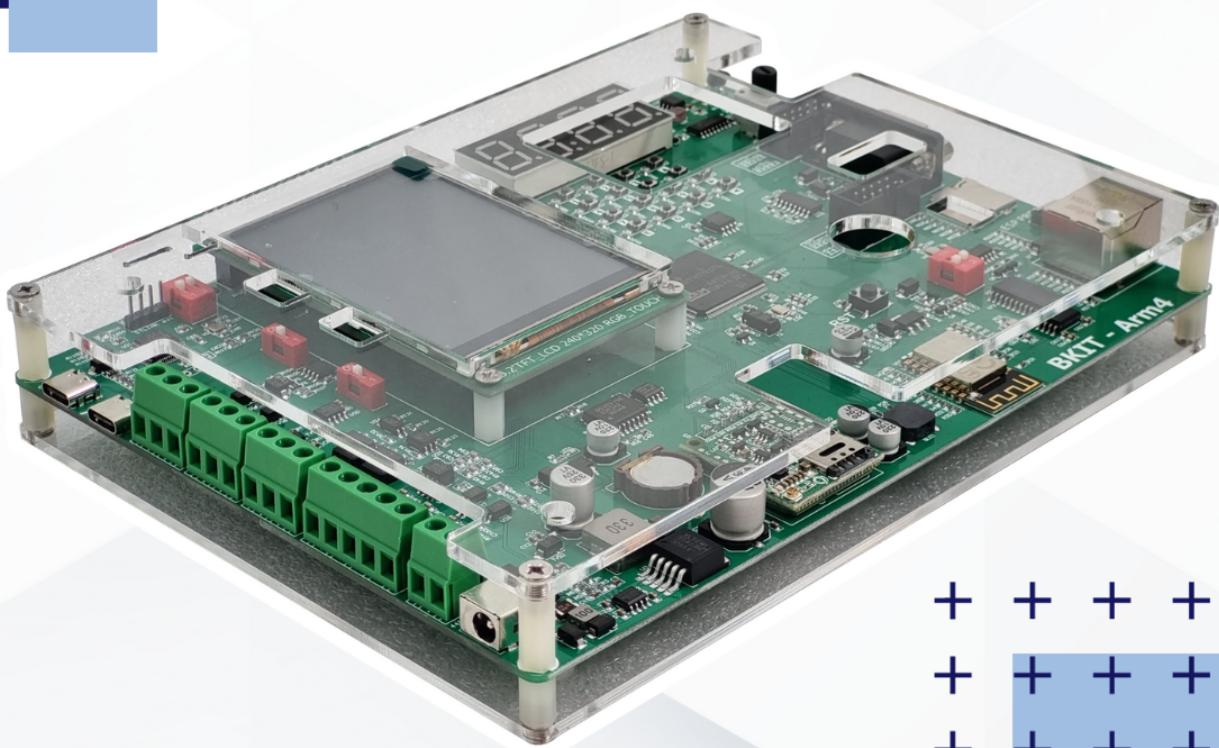
Sử dụng LED3 và các LED ở output Y0, Y1 để mô phỏng tín hiệu đèn giao thông. Giả sử mỗi LED đại diện cho một tín hiệu trên đèn giao thông với chu kì đèn đỏ là 5 giây, đèn xanh là 3 giây, đèn vàng là 1 giây. Chỉ sử dụng duy nhất 1 câu lệnh delay.

# CHƯƠNG 2

---

## Timer Interrupt and LED Scanning

---

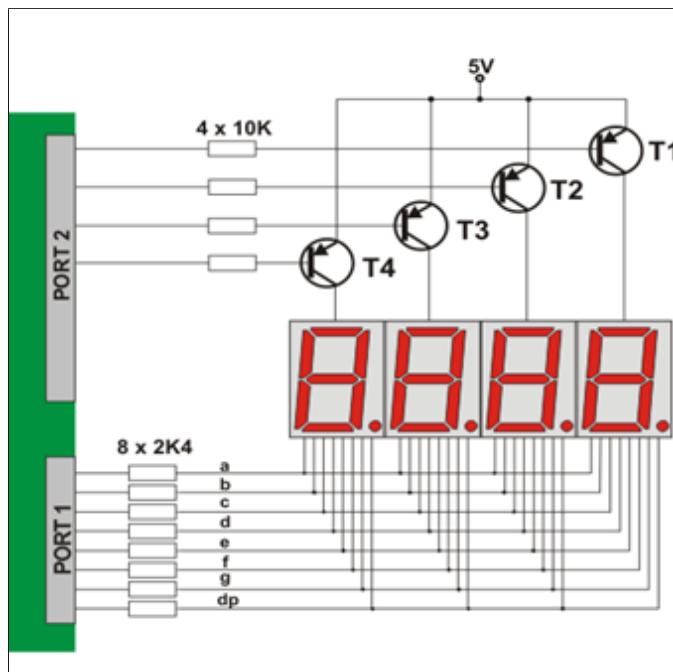


# 1 Mục tiêu

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

## 2 Giới thiệu

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

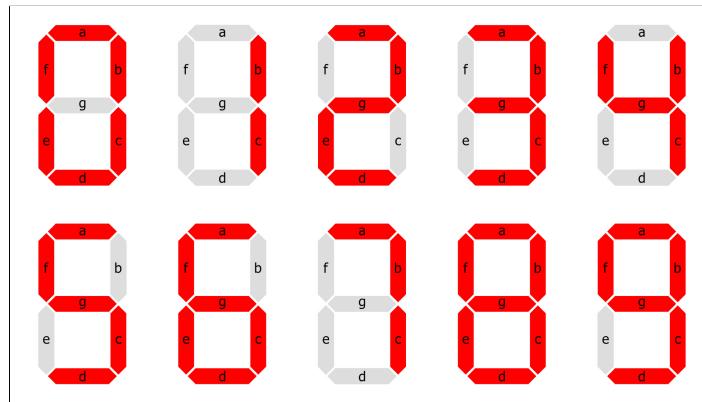


Hình 2.1: Giao diện của 4 LED bảy đoạn kết nối với vi điều khiển

Thiết kế giao diện cho màn hình nhiều LED bảy đoạn sử dụng input và output để điều khiển tắt cả các LED trong màn hình đã cho: dòng điện vào, khả năng cấp nguồn hay truyền nguồn cho thiết bị khác của mỗi chân, tốc độ gửi tín hiệu điều khiển. Với các thông số kỹ thuật này, giao tiếp có thể được thực hiện cho màn hình nhiều LED bảy đoạn với một vi điều khiển.

Mỗi LED bảy đoạn có bảy phân đoạn được đánh dấu từ a đến g (hình 2.2) và phân đoạn thứ tám là dấu chấm thập phân. Trong diagram, mỗi LED bảy đoạn có 8 đèn LED bên trong tương ứng với 8 phân đoạn, nên tổng số đèn LED trong ma trận 4 LED bảy đoạn là 32. Tuy nhiên, không cần phải bật tất cả các đèn LED, chỉ cần một

trong số chúng. Do đó, chỉ cần 12 chân để điều khiển 4 LED bảy đoạn. Sử dụng vi điều khiển, ta có thể bật các đèn cùng một khoảng thời gian  $T_S$ . Vì vậy, chu kỳ để điều khiển 4 LED bảy đoạn sẽ là  $4T_S$ . Nói cách khác, những LED này được quét với tần số  $f = 1/4T_S$ . Cuối cùng, dễ thấy nếu tần số quét lớn hơn 30Hz (e.g.  $f = 50\text{Hz}$ ) thì các LED dường như đang sáng cùng một lúc. Trong bài hướng dẫn này, timer interrupt được sử dụng để thiết kế một khoảng thời gian  $T_S$  cho việc quét LED.

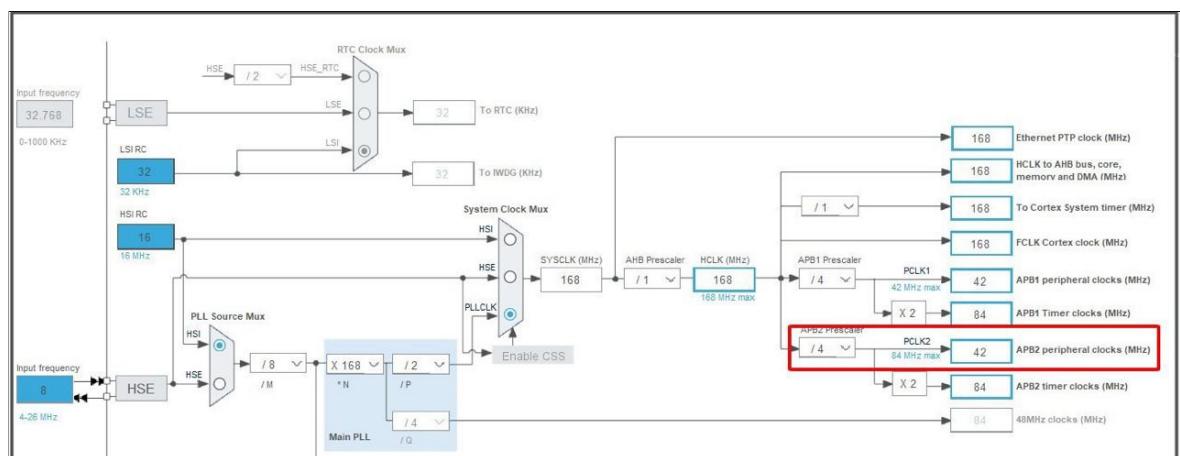


Hình 2.2: Biểu diễn ký tự số bằng LED 7 đoạn

### 3 Cấu hình timer

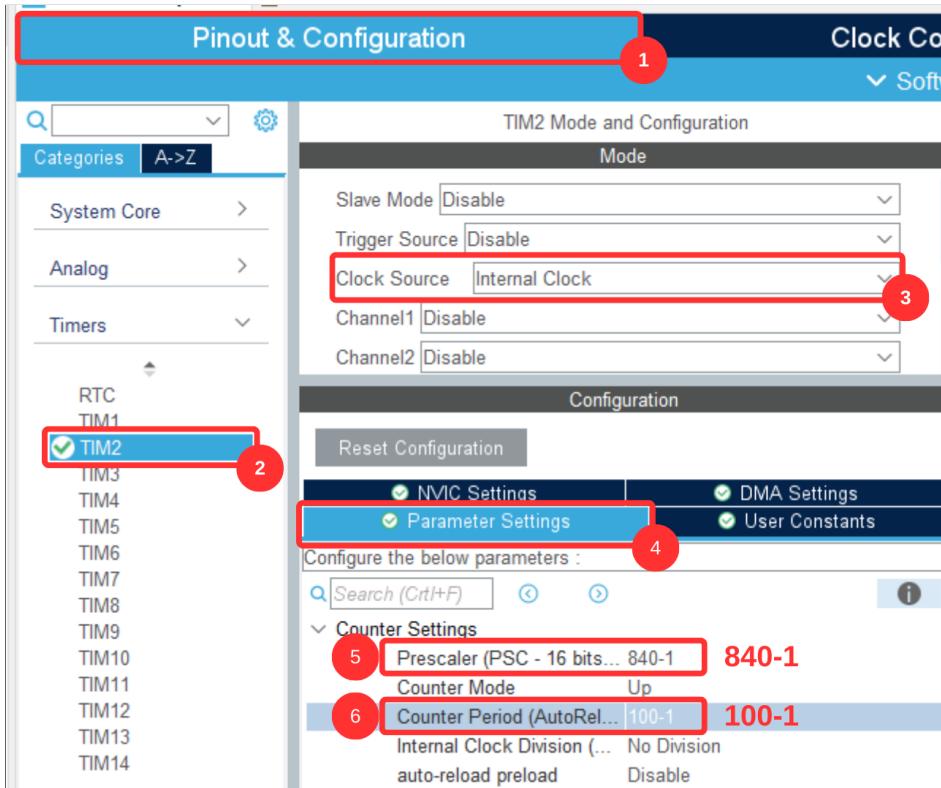
**Bước 1:** Tạo project mới được cấu hình như Lab 1.

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



Hình 2.3: Clock source mặc định cho hệ thống

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

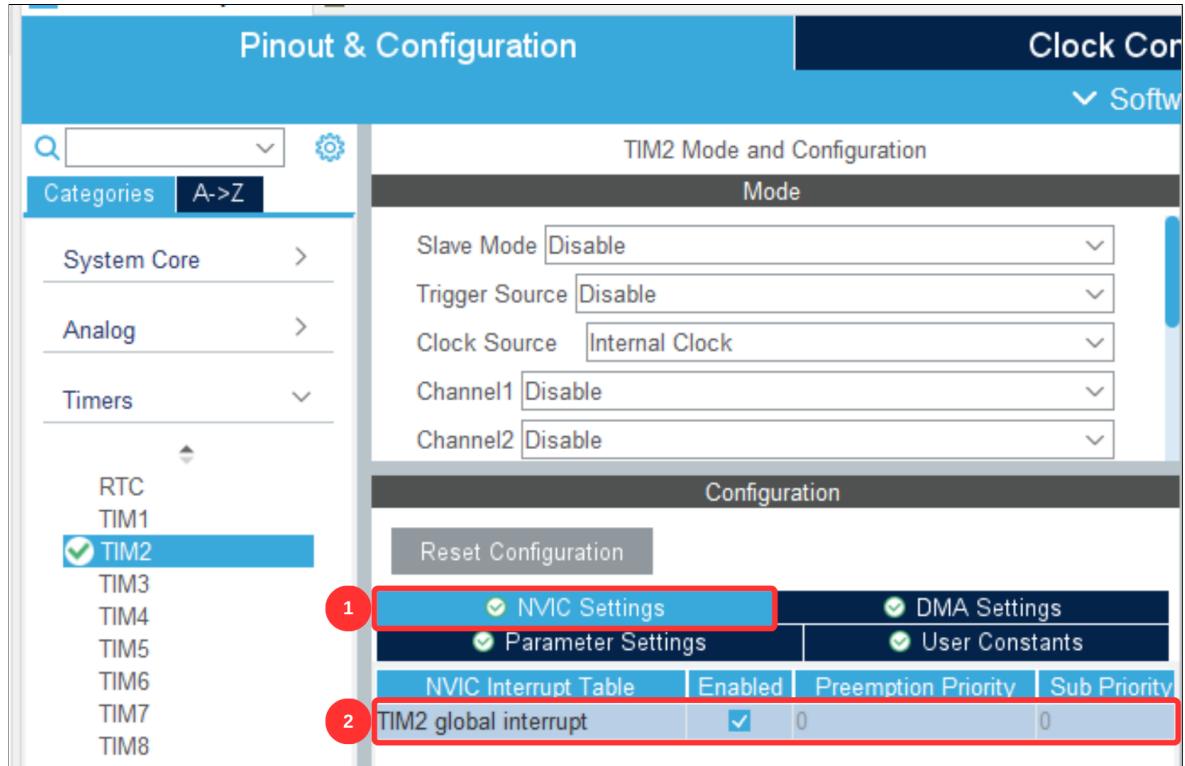


Hình 2.4: Cấu hình cho Timer 2

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

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

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



Hình 2.5: Kích hoạt timer interrupt

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

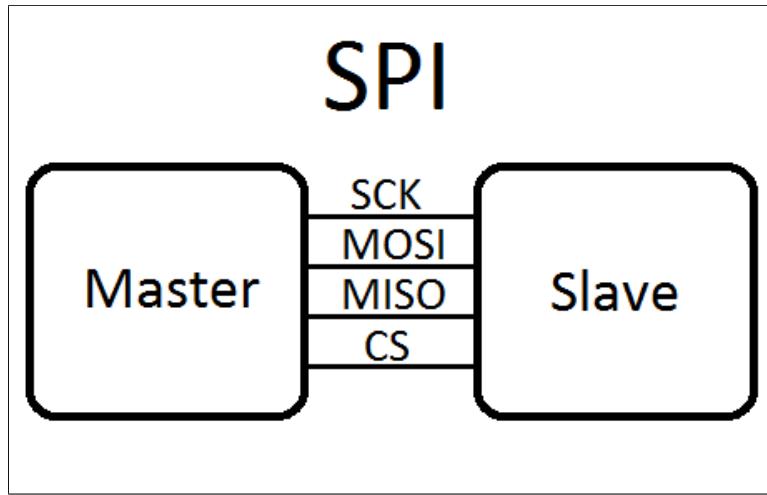
Cuối cùng lưu lại file cấu hình để hệ thống tự sinh code.

## 4 Cấu hình LED đồng hồ

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

### 4.1 Giao tiếp SPI

- SPI – Serial Peripheral Interface – còn gọi là giao diện ngoại vi nối tiếp, chuẩn đồng bộ nối truyền dữ liệu ở chế độ full-duplex (song công toàn phần). Nghĩa là tại 1 thời điểm có thể xảy ra đồng thời quá trình truyền và nhận.
- SPI là giao tiếp đồng bộ, bất cứ quá trình nào cũng đều được đồng bộ với xung clock sinh ra bởi thiết bị Master. Do đó, không cần phải lo lắng về tốc độ truyền dữ liệu.



Hình 2.6: Sơ đồ giao tiếp SPI

- Hoạt động của giao tiếp SPI: sử dụng 4 đường giao tiếp nên đôi khi được gọi là chuẩn truyền thông “ 4 dây”. 4 đường đó là :
  - SCK (Serial Clock): Thiết bị Master tạo xung tín hiệu SCK và cung cấp cho Slave. Xung này giữ nhịp cho giao tiếp SPI và truyền dữ liệu. Điều này giúp giảm lỗi và tăng tốc độ truyền.
  - MISO (Master Input Slave Output): Tín hiệu tạo bởi thiết bị Slave và nhận bởi thiết bị Master. Đường MISO phải được kết nối giữa thiết bị Master và Slave.
  - MOSI (Master Output Slave Input): Tín hiệu tạo bởi thiết bị Master và nhận bởi thiết bị Slave. Đường MOSI phải được kết nối giữa thiết bị Master và Slave.
  - CS (Chip Select) hoặc SS(Slave Select): Chọn Slave để giao tiếp. Master kéo chân CS xuống mức 0 (Low) để chọn Slave. Vi điều khiển có thể tạo chân CS bằng cách cấu hình 1 chân GPIO chế độ Output.

## 4.2 IC 74HC595

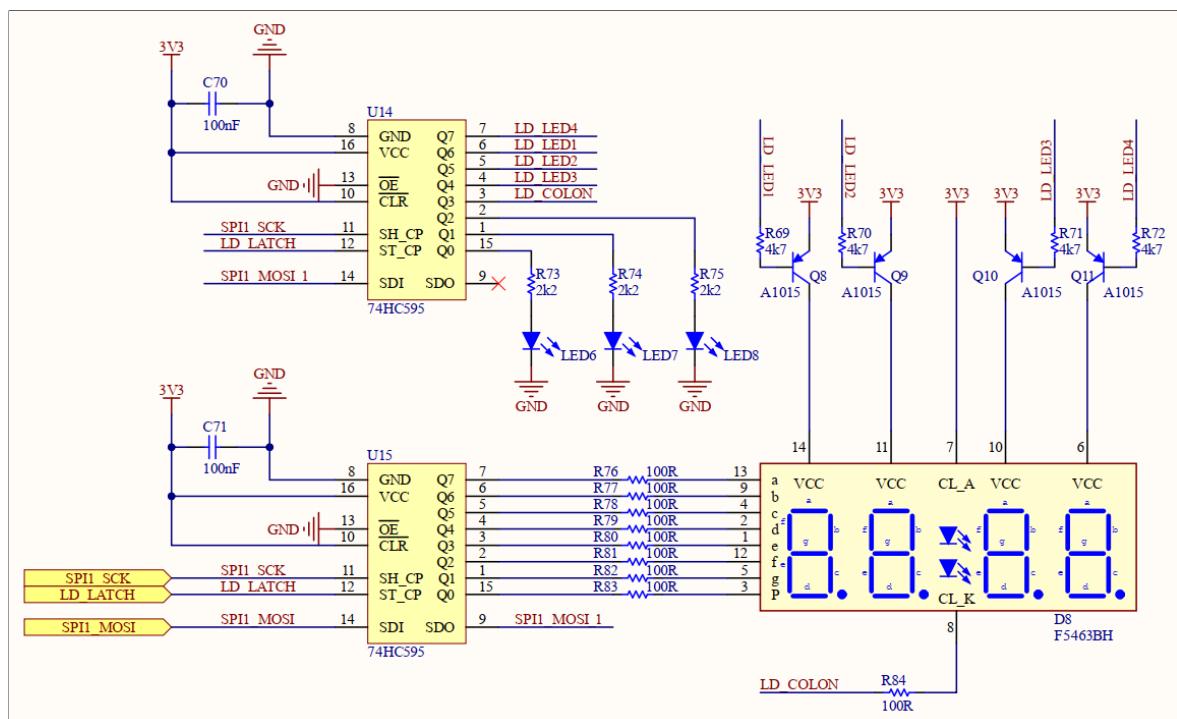
74HC595 là một thanh ghi dịch (shift register) hoạt động trên giao thức nối tiếp vào song song ra (Serial IN Parallel OUT), tức là nó có thể nhận dữ liệu đầu vào nối tiếp và điều khiển 8 chân đầu ra song song. Điều này rất tiện dụng khi không có đủ chân GPIO trên vi điều khiển hoặc vi xử lý để kiểm soát số lượng đầu ra cần thiết.

Đối với IC 74HC595, chúng ta cần quan tâm tới các chân **SH\_CP**, **ST\_CP**, **SDI**, **SDO**, **Q0-Q7**.

**Nguyên lý hoạt động:** khi phát hiện cạnh lên tại tín hiệu **SH\_CP** thì dữ liệu của chân **SDI** sẽ được đưa vào thanh ghi dịch. Ghi tín hiệu chân **ST\_CP** xuống mức thấp, thì giá trị trong thanh ghi dịch sẽ được cập nhật ra các chân output **Q7->Q0**. Để mở rộng thêm tín hiệu output thì ta sẽ dùng 2 IC 74HC595 và kết nối chân SDO của một IC với chân **SDI** của IC còn lại.

Với 2 IC thì ta có thể điều khiển được 16 output. Để điều khiển LED đồng hồ, ta cần 8 chân để điều khiển tín hiệu LED 7 đoạn, 4 chân để điều khiển chớp tắt 4 LED 7 đoạn, 1 chân để điều khiển dấu hai chấm. Như vậy chúng ta vẫn còn 3 output chưa được sử dụng, 3 output này sẽ được thiết kế để điều khiển LED6, LED7, LED8. Lưu ý: tất cả các chân tín hiệu kể trên đều tích cực mức thấp.

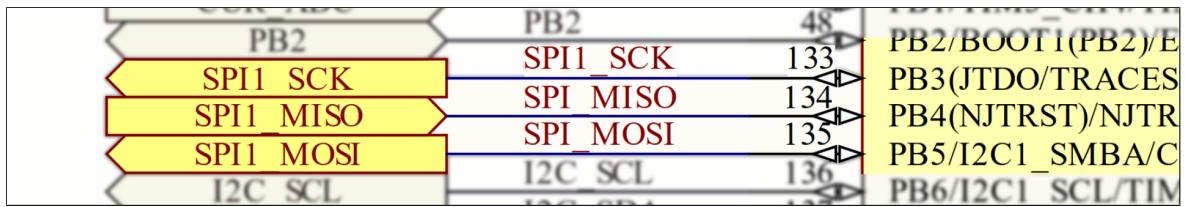
Với nguyên lý hoạt động của 74HC595, ta thấy rằng có thể sử dụng SPI để hiện thực giao tiếp giữa MCU và IC. Chân **SCK** của MCU sẽ kết nối với chân **SH\_CP** của cả 2 IC, chân **MOSI** của MCU sẽ kết nối với chân **SDI** của IC đầu tiên. Trên kit thí nghiệm, module SPI1 sẽ được sử dụng với mục đích trên.



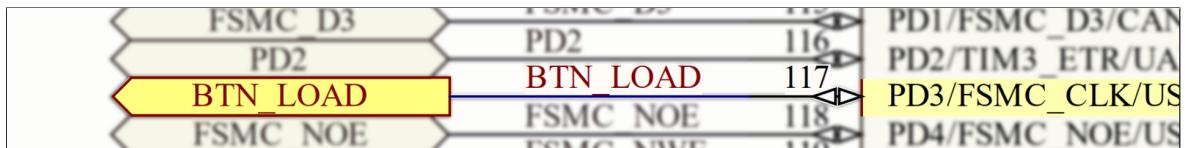
Hình 2.7: Sơ đồ nguyên lý điều khiển LED đồng hồ

#### 4.3 Config ch n cho vi điều khiển

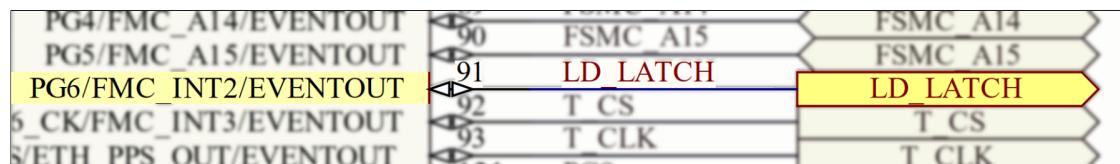
Theo schematic, ta có những chân cần config như sau:



Hình 2.8: Sơ đồ nguyên lý của các chân SPI1\_SCK, SPI1\_MISO, SPI1\_MOSI

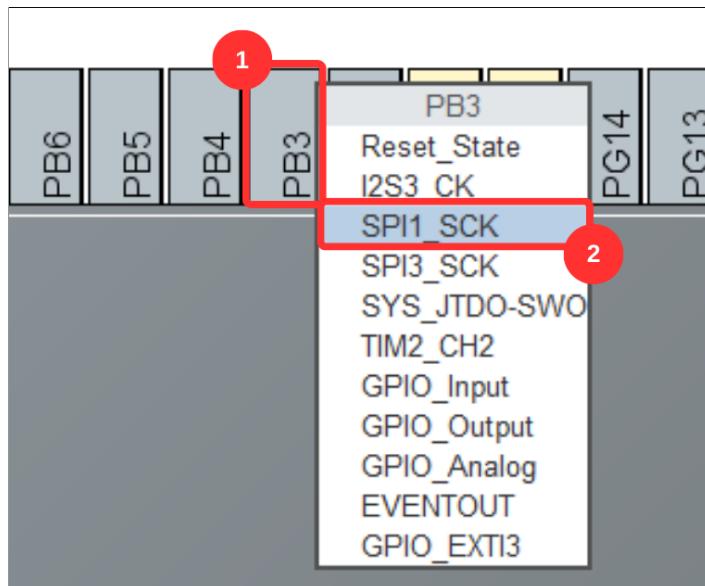


Hình 2.9: Sơ đồ nguyên lý của chân BTN\_LOAD



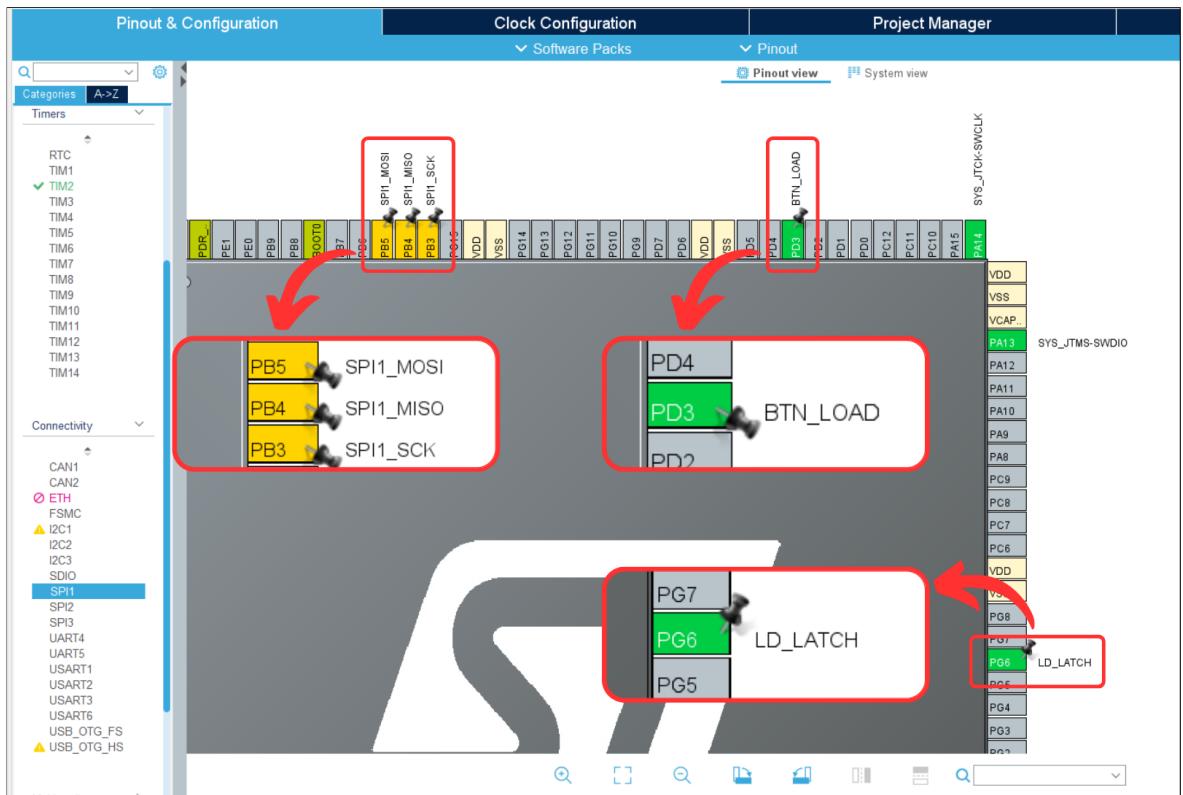
Hình 2.10: Sơ đồ nguyên lý của chân LD\_LATCH

Ta thực hiện config các chân SPI1\_SCK, SPI1\_MISO, SPI1\_MOSI bằng cách nhấp chuột phải vào chân tương ứng, sau đó chọn kiểu chân dựa theo schematic (hình 2.8). Hai chân BTN\_LOAD và LD\_LATCH sẽ được config thành chân output.



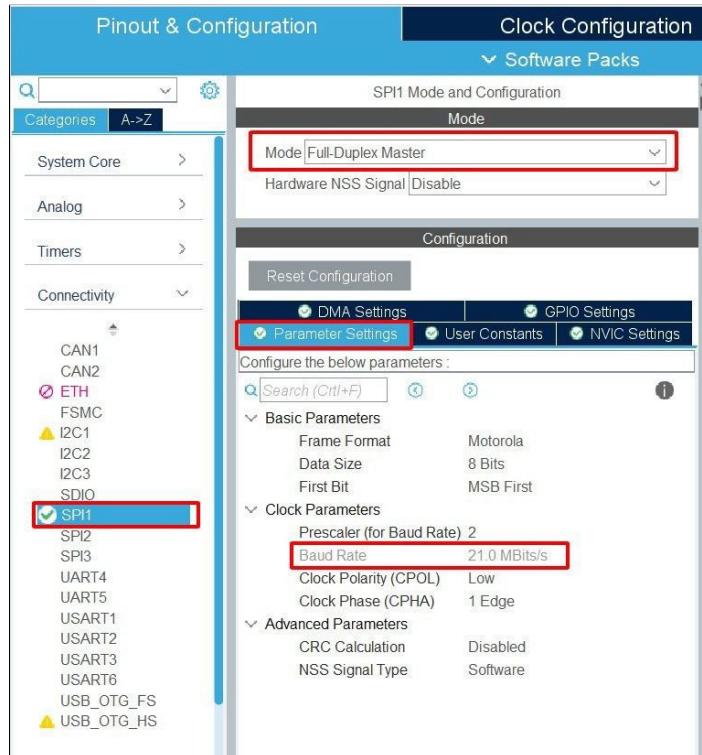
Hình 2.11: Config các chân SPI1\_SCK, SPI1\_MISO, SPI1\_MOSI

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



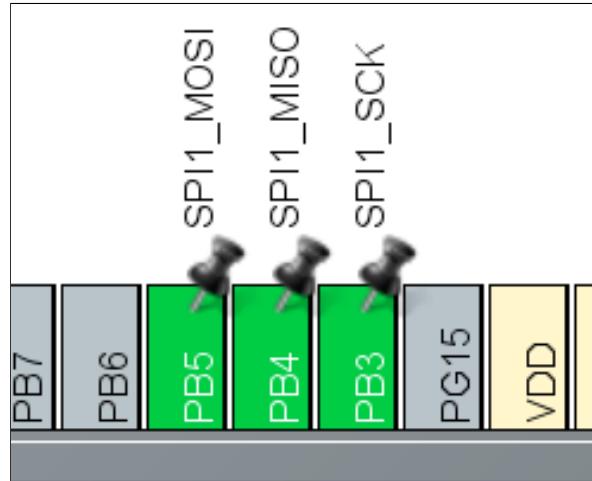
Hình 2.12: Các chân cần config

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



Hình 2.13: Config SPI

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



Hình 2.14: Sau khi config SPI1

## 5 Hướng dẫn lập trình

Các file **software\_timer** và **LED7\_seg** có thể được copy từ project mẫu **Bai2\_Timer\_7seg**.

```
1 #ifndef INC_SOFTWARE_TIMER_H_
2 #define INC_SOFTWARE_TIMER_H_
3
4 #include "tim.h"
5 #include "software_timer.h"
6 #include "LED_7seg.h"
7
8 extern uint16_t flag_timer2;
9
10 void timer_init();
11 void setTimer2(uint16_t duration);
12
13#endif /* INC_SOFTWARE_TIMER_H_ */
```

Program 2.1: software\_timer.h

```

1 #include "software_timer.h"
2
3 #define TIMER_CYCLE_2 1
4
5 uint16_t flag_timer2 = 0;
6 uint16_t timer2_counter = 0;
7 uint16_t timer2_MUL = 0;
8
9 void timer_init(){
10     HAL_TIM_Base_Start_IT(&htim2);
11 }
12
13 void setTimer2(uint16_t duration){
14     timer2_MUL = duration/TIMER_CYCLE_2;
15     timer2_counter = timer2_MUL;
16     flag_timer2 = 0;
17 }
18
19 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
20     if(htim->Instance == TIM2){
21         if(timer2_counter > 0){
22             timer2_counter--;
23             if(timer2_counter == 0) {
24                 flag_timer2 = 1;
25                 timer2_counter = timer2_MUL;
26             }
27         }
28         // 1ms interrupt here
29         LED7_Scan();
30     }
31 }

```

Program 2.2: software\_timer.c

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

- Hàm **HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef\*htim)**: là hàm interrupt service routine (isr) được gọi mỗi 1ms. Trong hàm này, các counter

của software timer sẽ được đếm lùi, và khi hết thời gian một tín hiệu flag\_timer sẽ được bật để chương trình chính nhận biết.

- Hàm **timer\_init**: sẽ được gọi để khởi tạo timer của vi điều khiển.
- Hàm **setTimer2**: sẽ khởi tạo tần số của software timer 2. Nếu như cần sử dụng nhiều software timer hơn thì có thể tạo thêm các biến đếm và hàm tương tự.

```
1 #ifndef INC_LED7SEG_H_
2 #define INC_LED7SEG_H_
3
4 #include "spi.h"
5
6 void LED7_init();
7 void LED7_Scan();
8 void LED7_SetDigit(int num, int position, uint8_t show_dot)
9     ;
10 void LED7_SetColon(uint8_t status);
11 void LED_On(uint8_t index);
12 void LED_Off(uint8_t index);
13
14#endif /* INC_LED7SEG_H_ */
```

Program 2.3: LED7\_seg.h

- **LED7\_init**: hàm khởi tạo LED đồng hồ
- **LED7\_Scan**: hàm quét LED. Hiện tại hàm này được gọi trong timer interrupt. Có nghĩa tần số quét hiện là **1KHz/4 = 250Hz**. Để thử nghiệm thay đổi tần số quét thì có thể dùng software timer và thay đổi vị trí đặt hàm này.
- **LED7\_SetDigit**: hàm hiển thị số trên LED, tham số đầu tiên là số muốn hiển thị, tham số thứ 2 là vị trí của LED muốn hiển thị số đó (0-4), tham số cuối cùng thể hiện rằng dấu chấm tại LED 7 đoạn đó có được hiển thị hay không.
- **LED7\_SetColon**: hàm hiển thị hoặc tắt dấu hai chấm.
- **LED\_On**: hàm bật các LED 6,7,8; tham số đầu vào là index của các LED muốn bật.
- **LED\_Off**: hàm tắt các LED 6,7,8; tham số đầu vào là index của các LED muốn tắt.

```

1 int main(void)
2 {
3     //...
4     /* USER CODE BEGIN 2 */
5     system_init();
6     LED7_SetColon(1);
7     /* USER CODE END 2 */
8
9     /* Infinite loop */
10    /* USER CODE BEGIN WHILE */
11    while (1)
12    {
13        while(!flag_timer2);
14        flag_timer2 = 0;
15        // main task, every 50ms
16        test_LEDDebug();
17        test_LEDY0();
18        test_LEDY1();
19        test_7seg();
20        /* USER CODE END WHILE */
21
22        /* USER CODE BEGIN 3 */
23    }
24    /* USER CODE END 3 */
25 }
26
27 /* USER CODE BEGIN 4 */
28 void system_init(){
29     HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port, OUTPUT_Y0_Pin,
30     0);
31     HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port, OUTPUT_Y1_Pin,
32     0);
33     HAL_GPIO_WritePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin,
34     0);
35     timer_init();
36     LED7_init();
37     setTimer2(50);
38 }
```

---

#### Program 2.4: hàm main

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

Câu lệnh **SetTimer2(50)** sẽ khởi tạo một software timer có chu kì là 50ms. Trong công nghiệp, 50ms thường là chu kì ổn định cho các việc đọc input, xử lý và xuất tín hiệu output.

Trong vòng lặp vô tận while, ta sẽ chờ cho đến khi tín hiệu **flag\_timer2** được bật (mỗi 50ms) sau đó sẽ gọi các hàm đọc input, xử lý hoặc xuất tín hiệu output.

---

```
1 uint8_t count_LED_debug = 0;
2 uint8_t count_LED_Y0 = 0;
3 uint8_t count_LED_Y1 = 0;
4
5 void test_LEDDebug(){
6     count_LED_debug = (count_LED_debug + 1)%20;
7     if(count_LED_debug == 0){
8         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port , DEBUG_LED_Pin);
9     }
10}
11
12 void test_LEDY0(){
13     count_LED_Y0 = (count_LED_Y0+ 1)%100;
14     if(count_LED_Y0 > 40){
15         HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port , OUTPUT_Y0_Pin ,
16         1);
17     } else {
18         HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port , OUTPUT_Y0_Pin ,
19         0);
20     }
21}
22 void test_LEDY1(){
23     count_LED_Y1 = (count_LED_Y1+ 1)%40;
24     if(count_LED_Y1 > 10){
25         HAL_GPIO_WritePin(OUTPUT_Y1_GPIO_Port , OUTPUT_Y1_Pin ,
26         0);
```

```

25     } else {
26         HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port, OUTPUT_Y1_Pin,
27             1);
28     }
29 }
30 void test_7seg(){
31     //write number1 at LED index 0 (not show dot)
32     LED7_SetDigit(1, 0, 0);
33     LED7_SetDigit(5, 1, 0);
34     LED7_SetDigit(4, 2, 0);
35     LED7_SetDigit(7, 3, 0);
36 }
```

Program 2.5: Các hàm test

Các hàm trên được tạo với mục đích kiểm tra các module trên kit thí nghiệm và làm quen với cách sử dụng các hàm trong thư viện. Các hàm này sẽ được gọi trong vòng while mỗi 50ms.

Như vậy đối với hàm **test\_LEDDebug** sau 50ms thì biến **count\_LED\_debug** được tăng lên và cứ sau 20 lần gọi hàm ( $20 \times 50 = 1000\text{ms} = 1\text{s}$ ) thì biến **count\_LED\_debug** sẽ bằng 0 và tác vụ toggle LED sẽ được thực thi. Như vậy LED3 sẽ chớp tắt mỗi 1s.

Với cách suy luận tương tự thì hàm **test\_LEDY0** và **test\_LEDY1** sẽ tạo ra hiệu ứng chớp tắt LED với thời gian khác nhau.

Hàm **test\_7seg** sẽ hiển thị các giá trị lên LED đồng hồ, cụ thể LED đồng hồ sẽ hiển thị số "15:47".

## 6 Bài tập và Báo cáo

### 6.1 Bài tập 1

Dùng software timer và tham khảo các hàm ở trên để tạo hiệu ứng trên các LED:

- **DEBUG\_LED** chớp tắt mỗi 2s.
- **LED\_Y0** sáng trong vòng 2s và tắt trong 4s.

- **LED\_Y1** sáng trong vòng 5s và tắt trong 1s.

## 6.2 Bài tập 2

Hiện thực lại bài tập về đèn giao thông ở Lab 1 sử dụng timer.

## 6.3 Bài tập 3

Thay đổi tần số quét của LED đồng hồ thành 1Hz, 25Hz, 100Hz.

## 6.4 Bài tập 4

Dùng LED đồng hồ để mô phỏng đồng hồ kỹ thuật số, 2 số đầu tiên sẽ hiển thị giờ, 2 số tiếp theo sẽ hiển thị phút. Thời gian sẽ thay đổi như thời gian thực. Đầu 2 chấm ở giữa đồng hồ sẽ chớp tắt với tần số 2Hz.

## 6.5 Bài tập 5

Hiển thị 4 số khác nhau trên LED đồng hồ, tạo hiệu ứng các số này sẽ được dịch qua phải mỗi 1s.