

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA KỸ THUẬT ĐIỆN TỬ

-----oo-----

BÀI THỰC HÀNH
MÔN KỸ THUẬT VI XỬ LÝ

NGÀNH HỌC: ĐIỆN-ĐIỆN TỬ VÀ ĐIỆN TỬ-VIỄN THÔNG

ĐỐI TƯỢNG: ĐẠI HỌC

Hà Nội, tháng 9 năm 2020

TỔNG QUAN VỀ MÔN HỌC

1. Đối tượng tham gia học tập

- Sinh viên đam mê lập trình nói chung và đam mê lập trình vi điều khiển nói riêng.
- Sinh viên muốn qua kì thi “Kỹ thuật vi xử lí”

2. Yêu cầu

- Có ý thức Tự học tập, tự nghiên cứu.
- Sinh viên đã hiểu biết cơ bản ít nhất một ngôn ngữ lập trình C.
- Chăm chỉ hoàn thành các bài tập nếu như các bạn muốn việc học là hiệu quả và tiết kiệm thời gian của chính các bạn.

3. Hình thức học tập

- Không nặng về lý thuyết, các bài thực hành sẽ ngắn gọn, dễ hiểu. Tuy nhiên, cuối mỗi bài, yêu cầu các bạn nghiên cứu thêm và tự đặt ra các câu hỏi về ý nghĩa mỗi dòng code trong các bài tập.
- Ghi lại mọi thắc mắc và hỏi trong tiết học để chúng ta cùng thảo luận và tối ưu code được tốt hơn. Khuyến khích tự tìm kiếm câu trả lời trên google

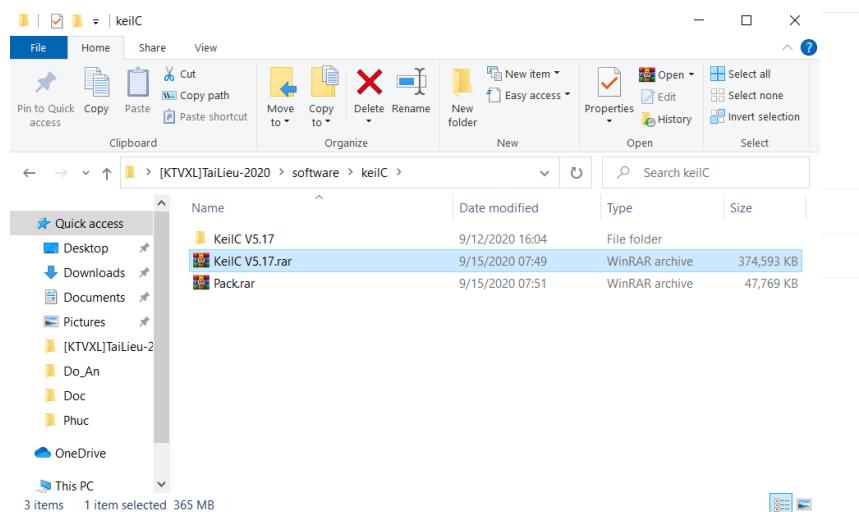
PHÂN 1 : TỔNG QUAN VỀ VĐK STM32F103C8T6 & MÔ PHỎNG TRÊN PROTEUS

I, Hướng dẫn cài đặt phần mềm lập trình KeilCv5

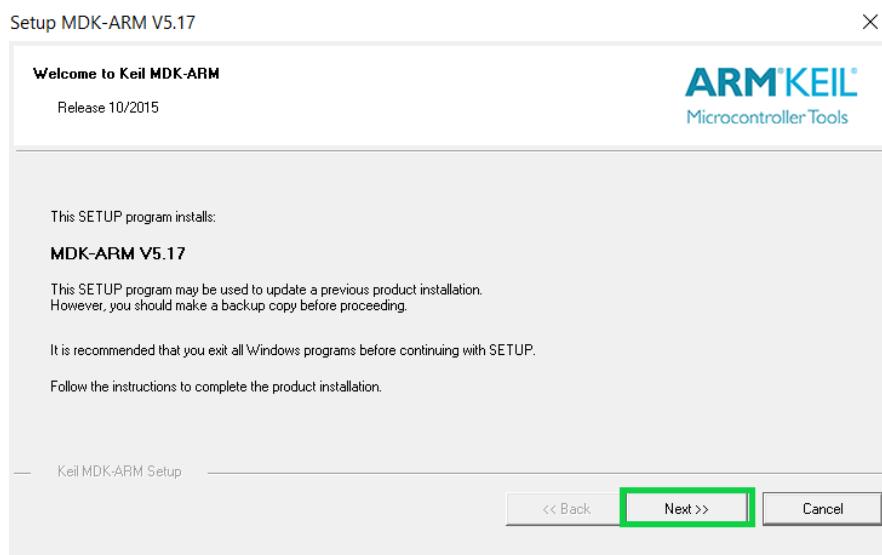
Có rất nhiều trình biên dịch hỗ trợ lập trình STM32F103C8T6 trong đó có một công cụ được sử dụng nhiều là KeilC. KeilC hỗ trợ lập trình ASM và C với họ vi điều khiển STM32F103C8T6 . Sau đây là các bước để cài đặt trình biên dịch này.

1.1. Hướng dẫn cài đặt:

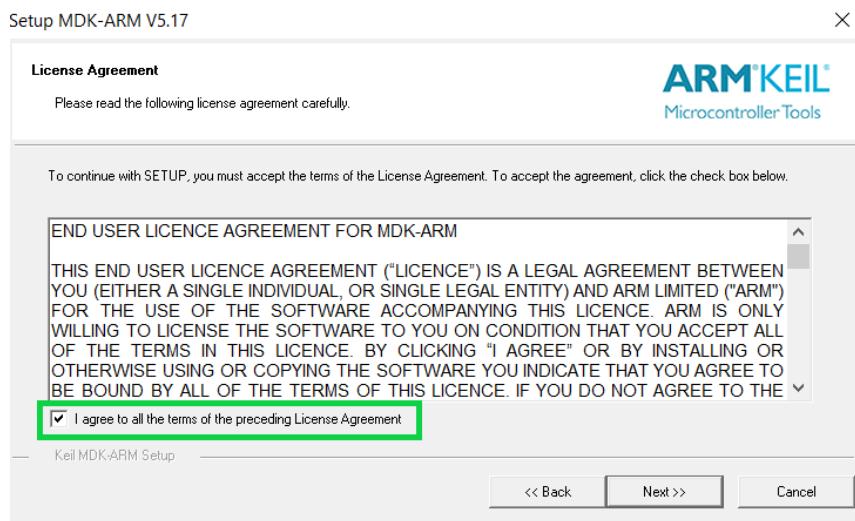
Bước 1: Giải nén file KeilC V5.17.rar.



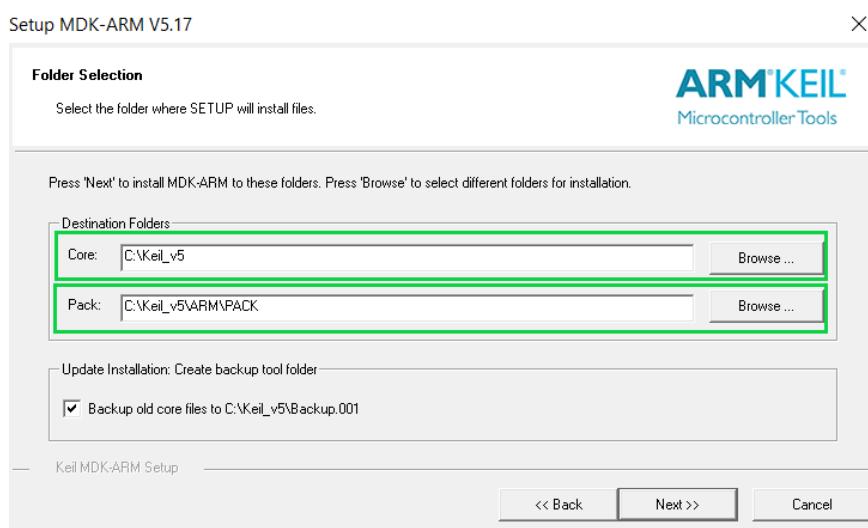
Bước 2: Vào thư mục vừa giải nén, sau đó các bạn chạy file “MDK517.EXE”, Chọn Next



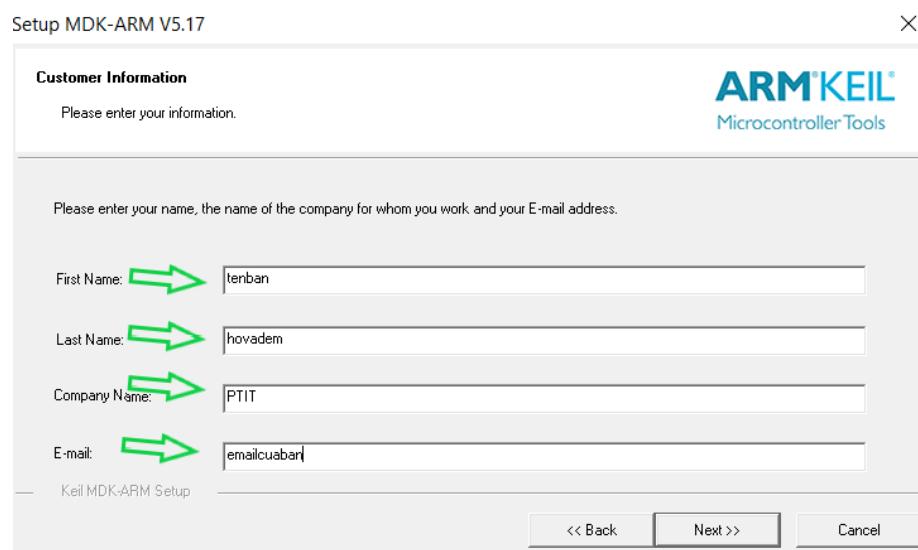
Bước 2: Tick vào ô “*I agree to all the tems of the preceding License Agreement*”, chọn Next.



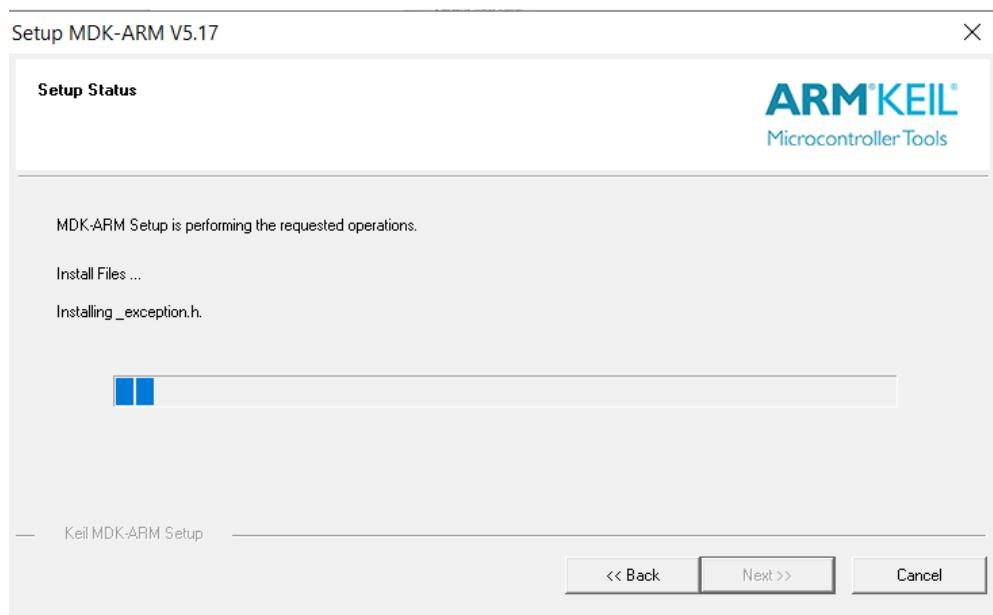
Bước 3: Chọn đường dẫn lưu thư mục cài đặt, sau đó chọn Next.



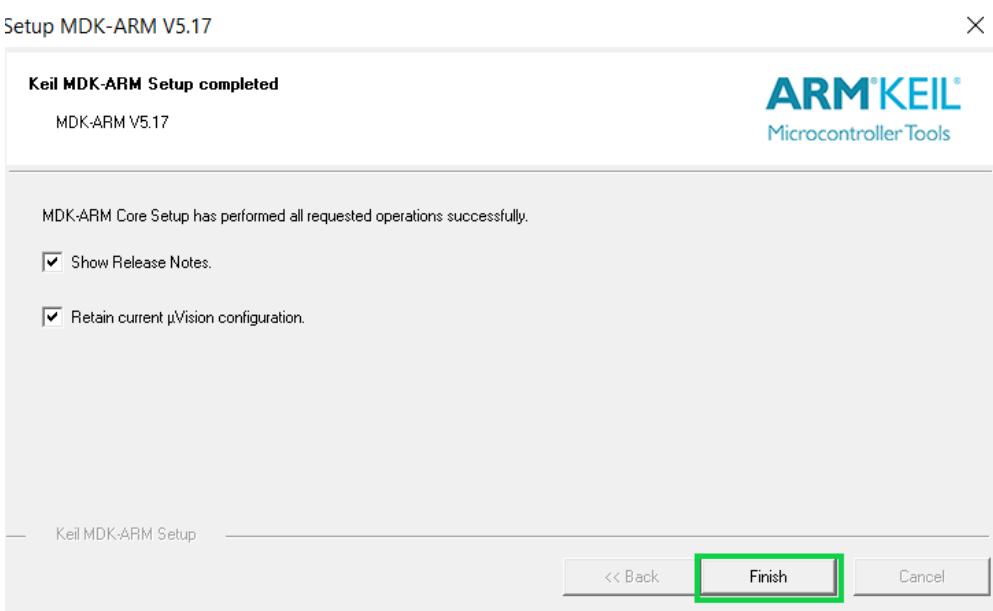
Bước 4: Điền thông tin cá nhân và chọn Next.



Bước 5: Chờ cho máy tiến hành cài đặt.



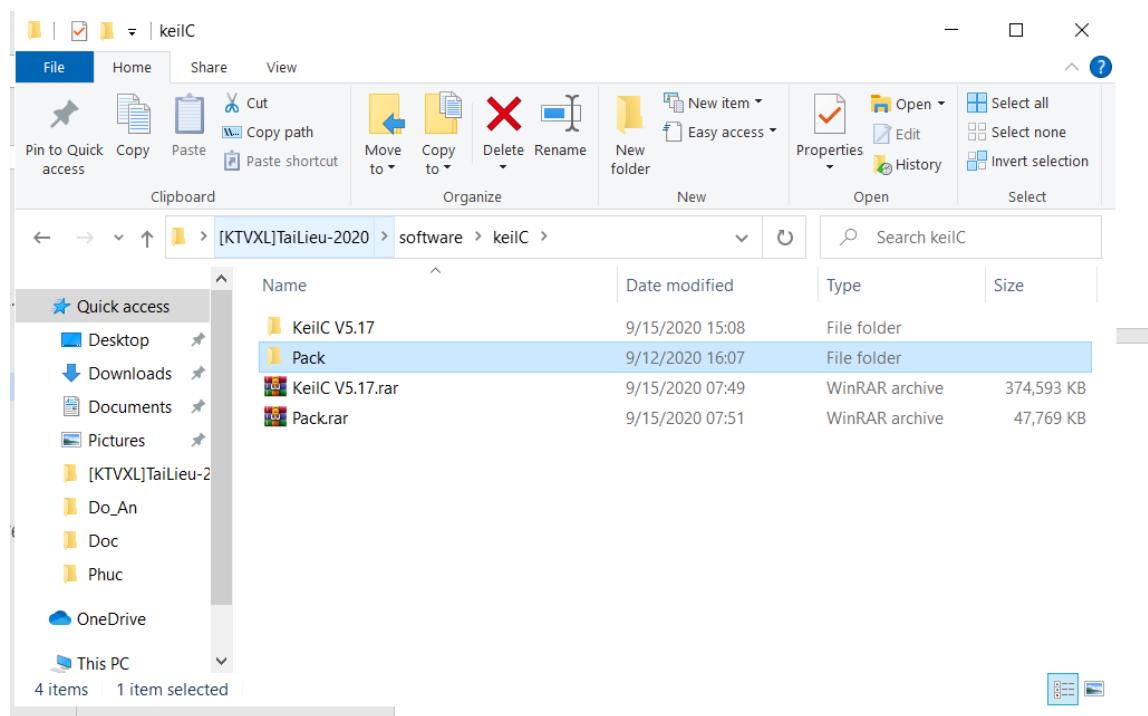
Bước 6: Sau khi cài đặt xong, chọn Finish.



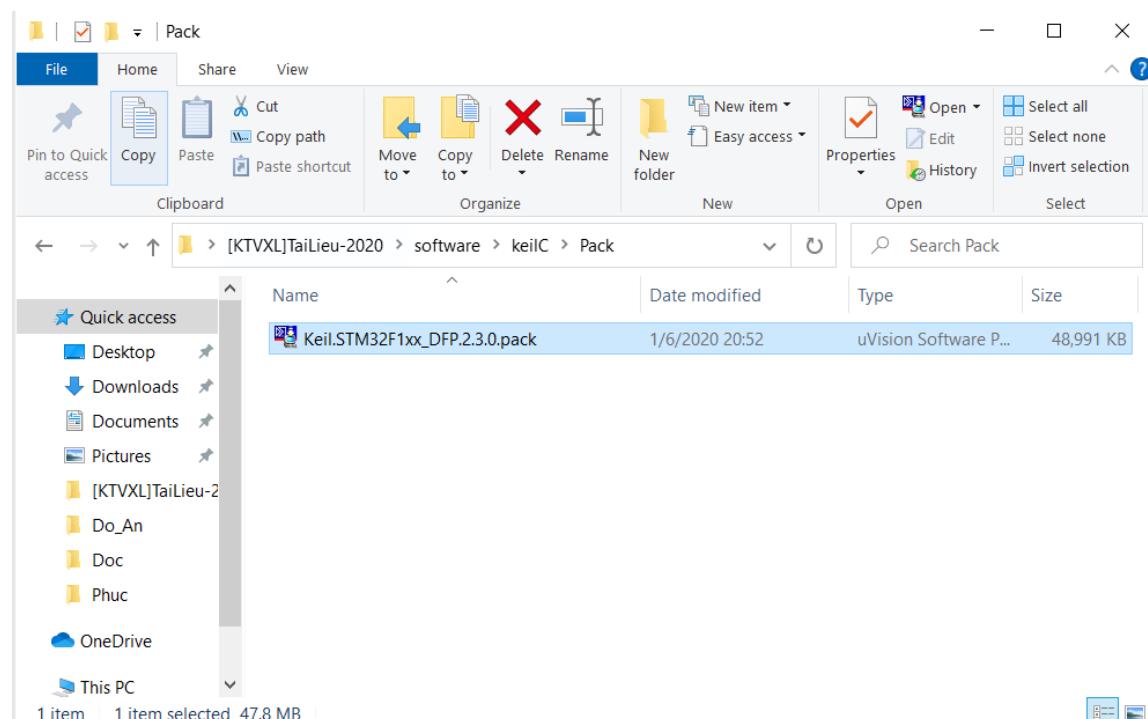
Vậy là bạn đã cài đặt xong phần mềm KeilC cho STM32.

1.2 Hướng dẫn cài đặt gói pack cho STM32F1XX

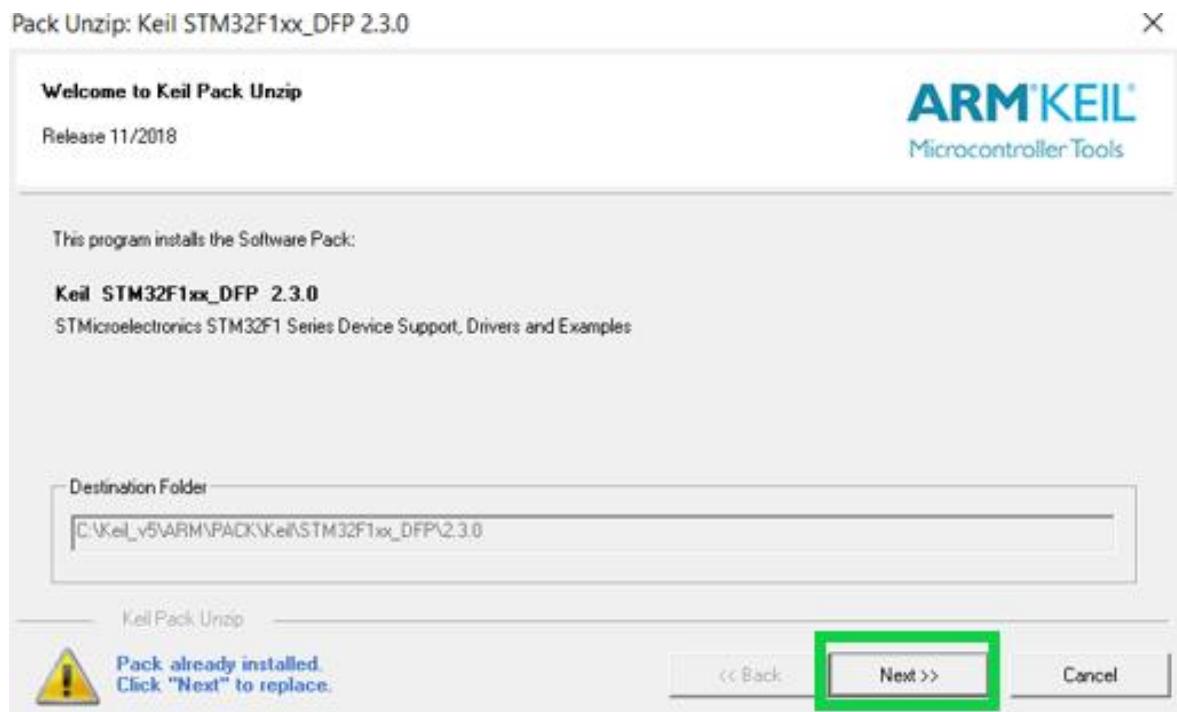
Bước 1: Giải nén thư mục Pack



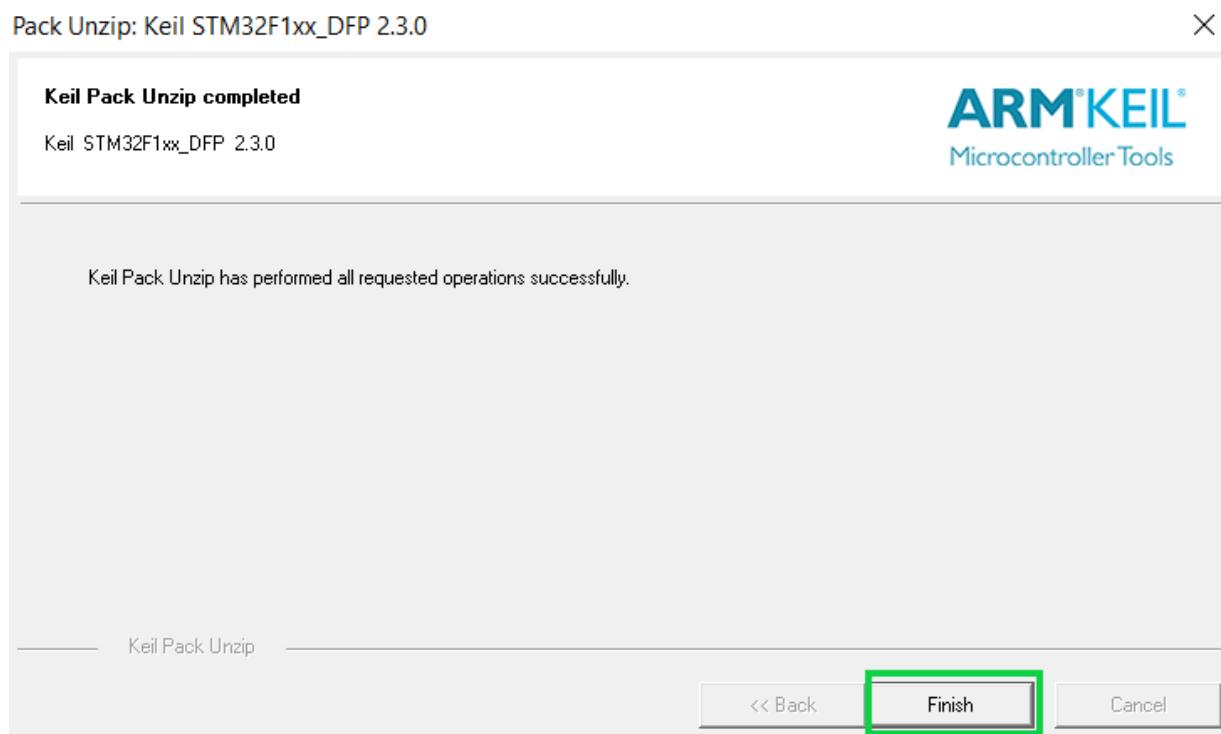
Bước 2: Vào thư mục Pack vừa giải nén và chạy file KeilSTM32F1xx_DFP.2.3.0.pack



Bước 3: Nhấn next để cài đặt

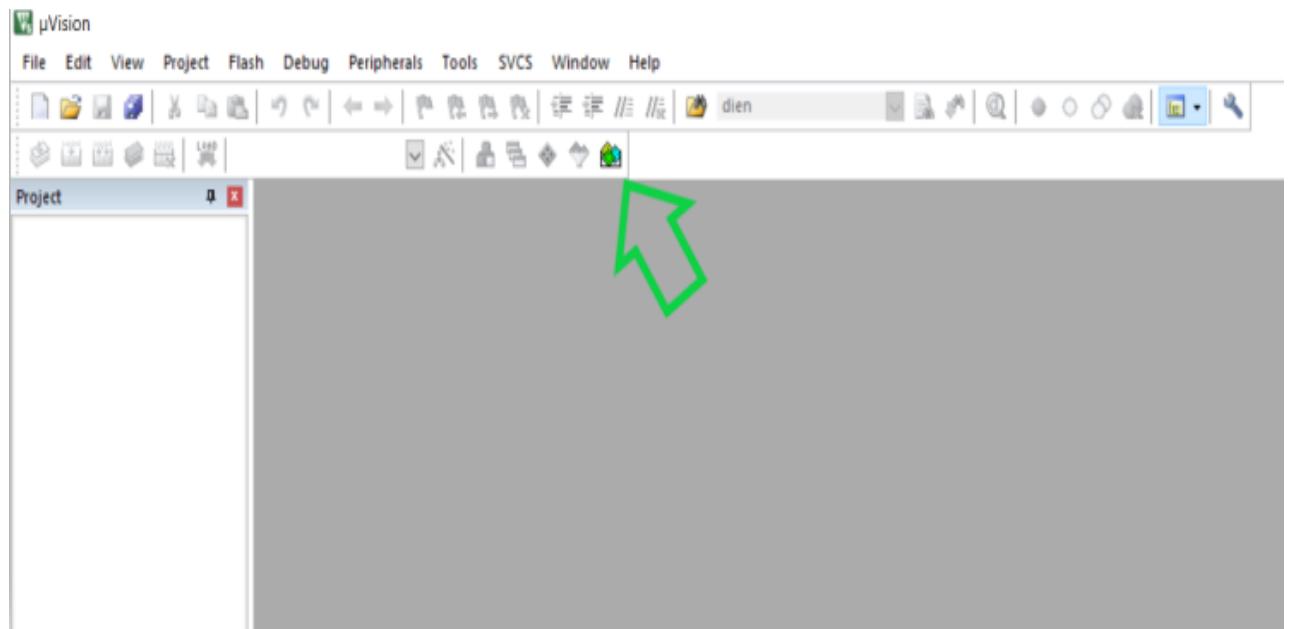


Bước 4: Nhấn Finish

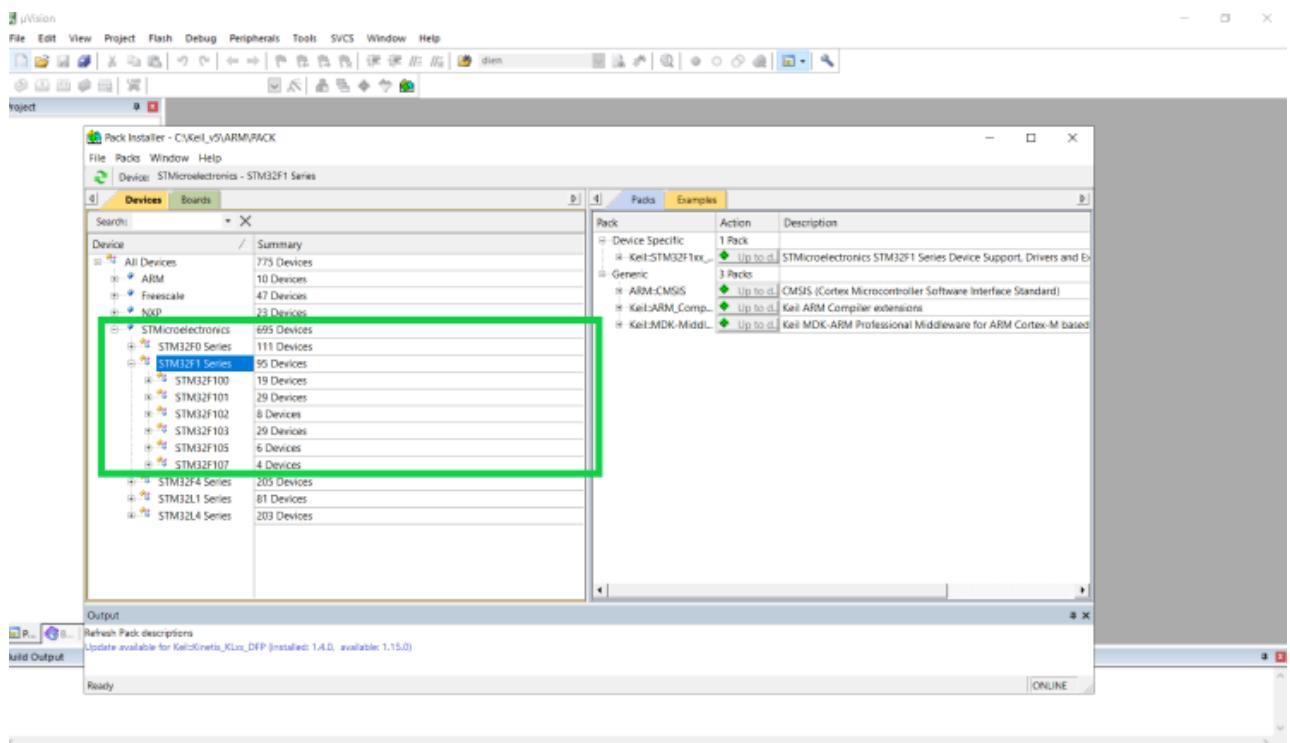


Bước 5: Kiểm tra

Vào keilC -> chọn Pack Installer



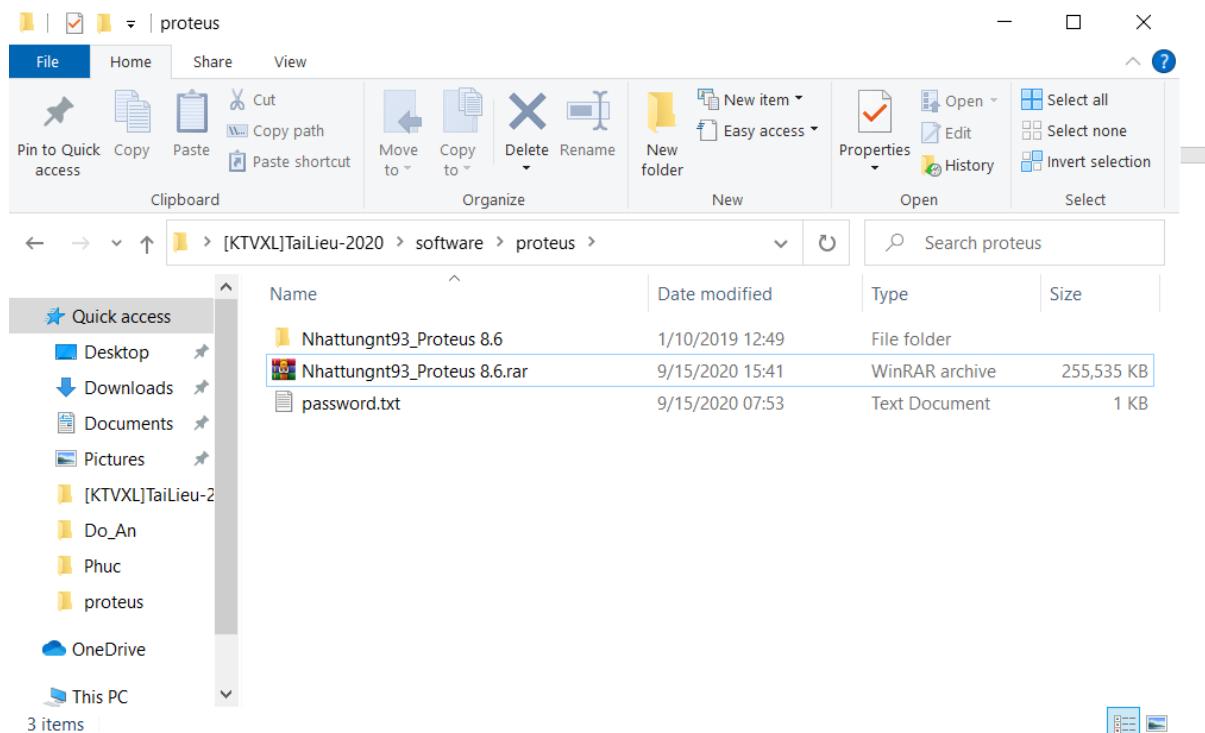
Dưới đây là hình ảnh gói pack cho STM32F1xx đã được cài đặt thành công.



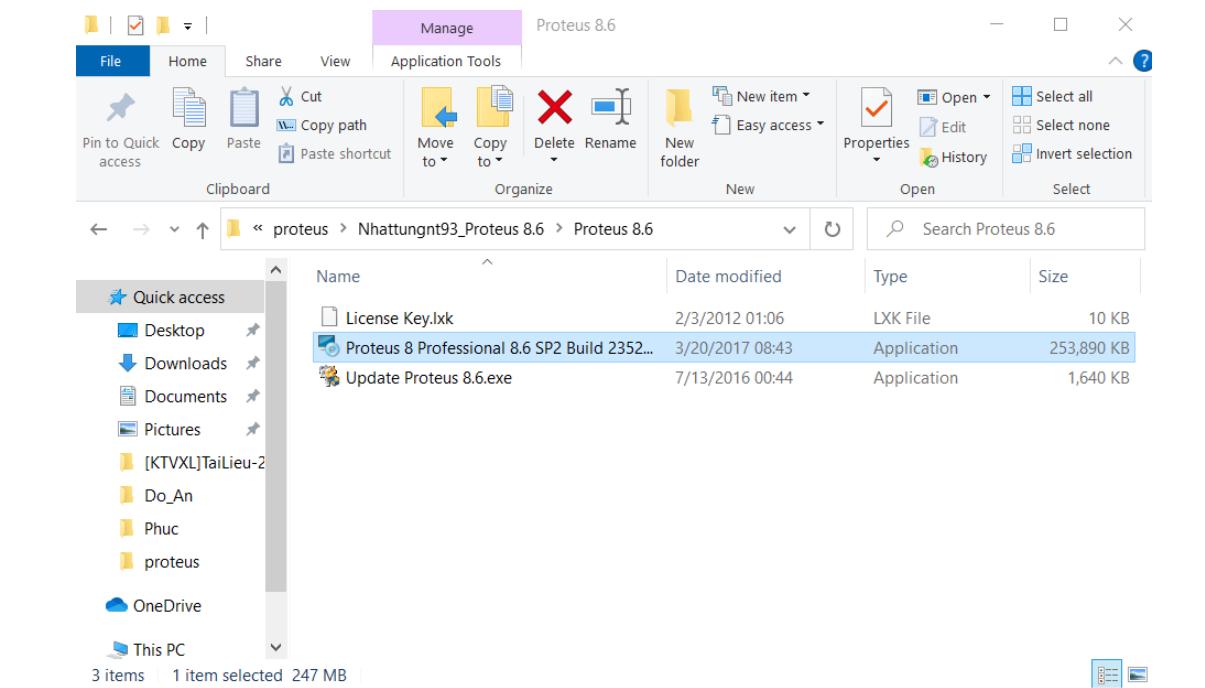
II, Hướng dẫn cài đặt phần mềm mô phỏng proteus 8.6

2.1. Hướng dẫn cài đặt.

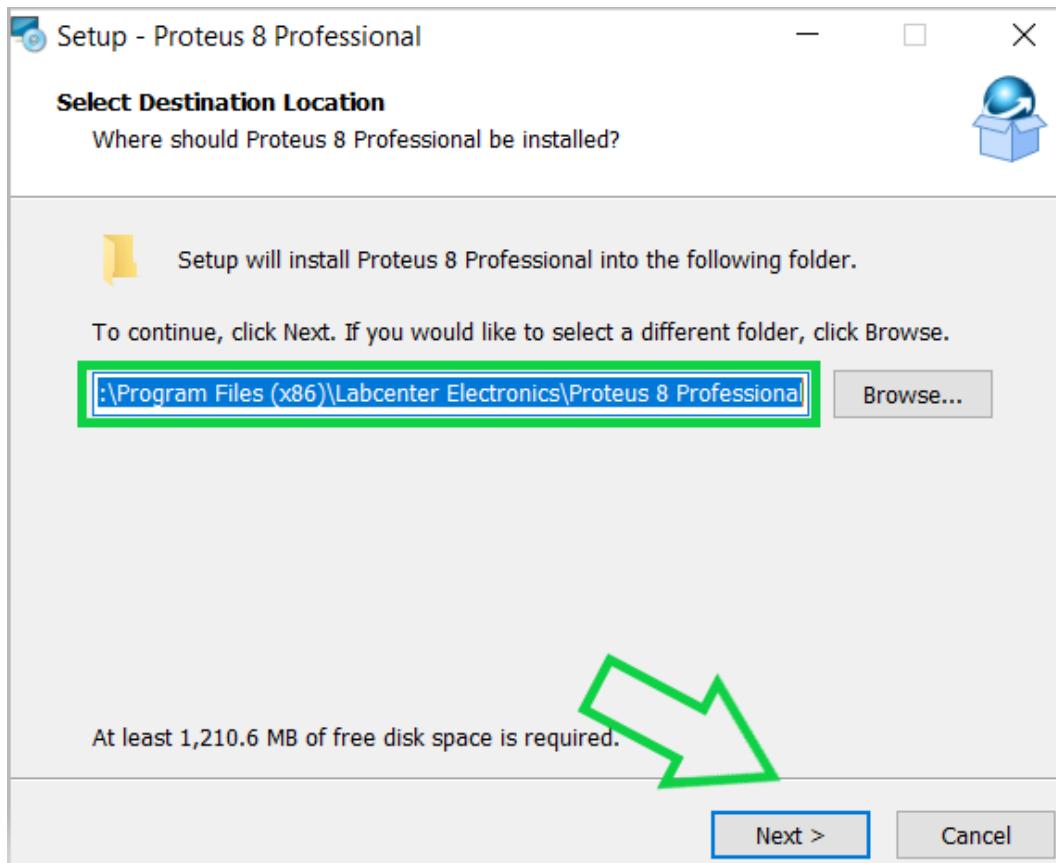
Bước 1: Giải nén file KeilC V5.17.rar, pass là Nhattungnt93



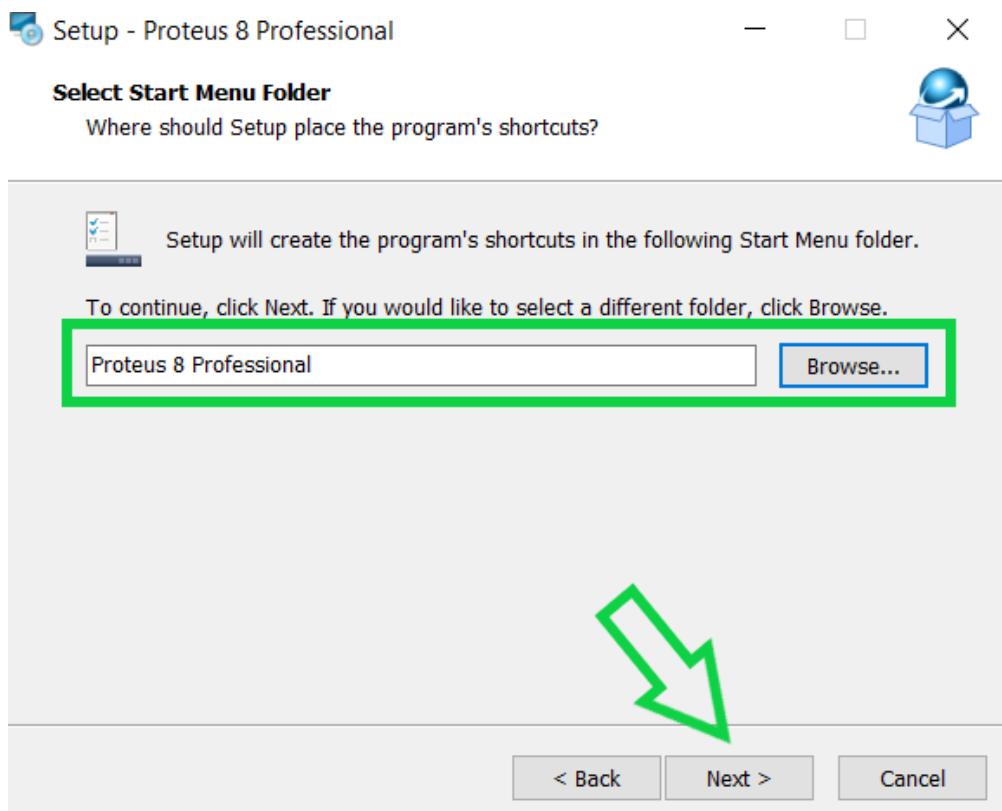
Bước 2: Vào thư mục vừa giải nén, sau đó các bạn chạy file “**Proteus 8 Professional 8.6 SP2 Build 23525 RePack.kuyhAa.exe**”.



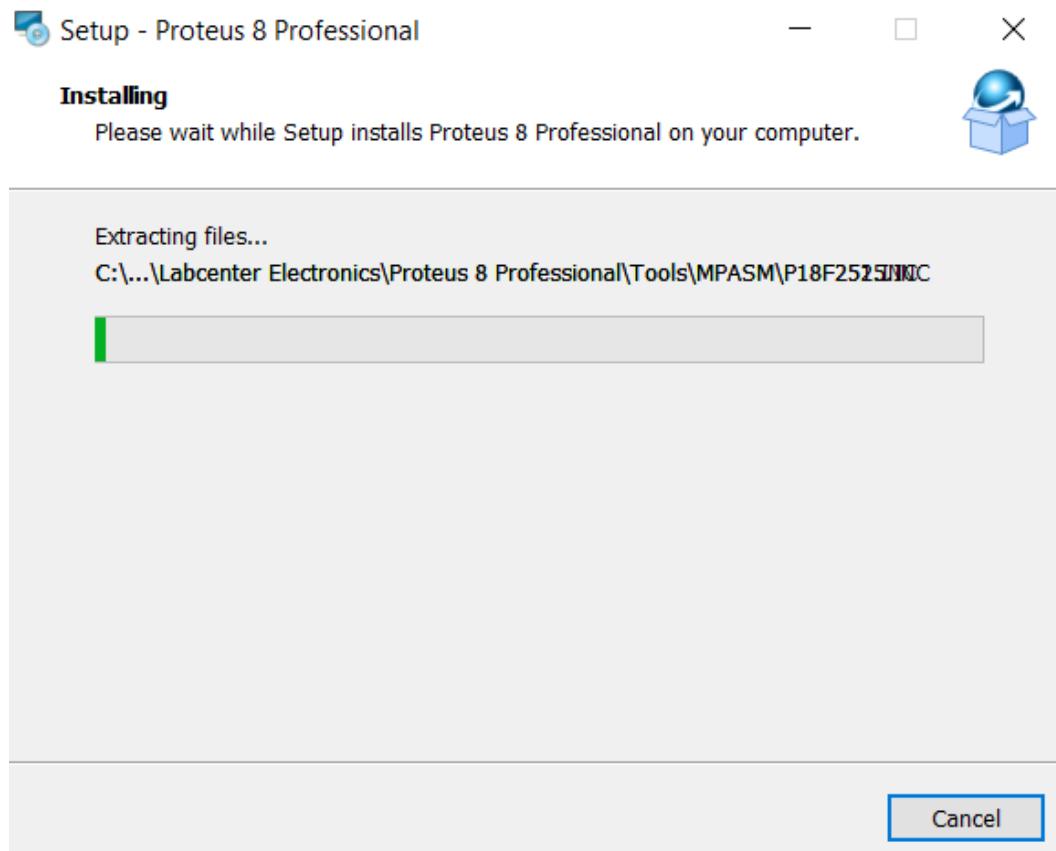
Bước 3: chọn được dẫn sau đó click next.



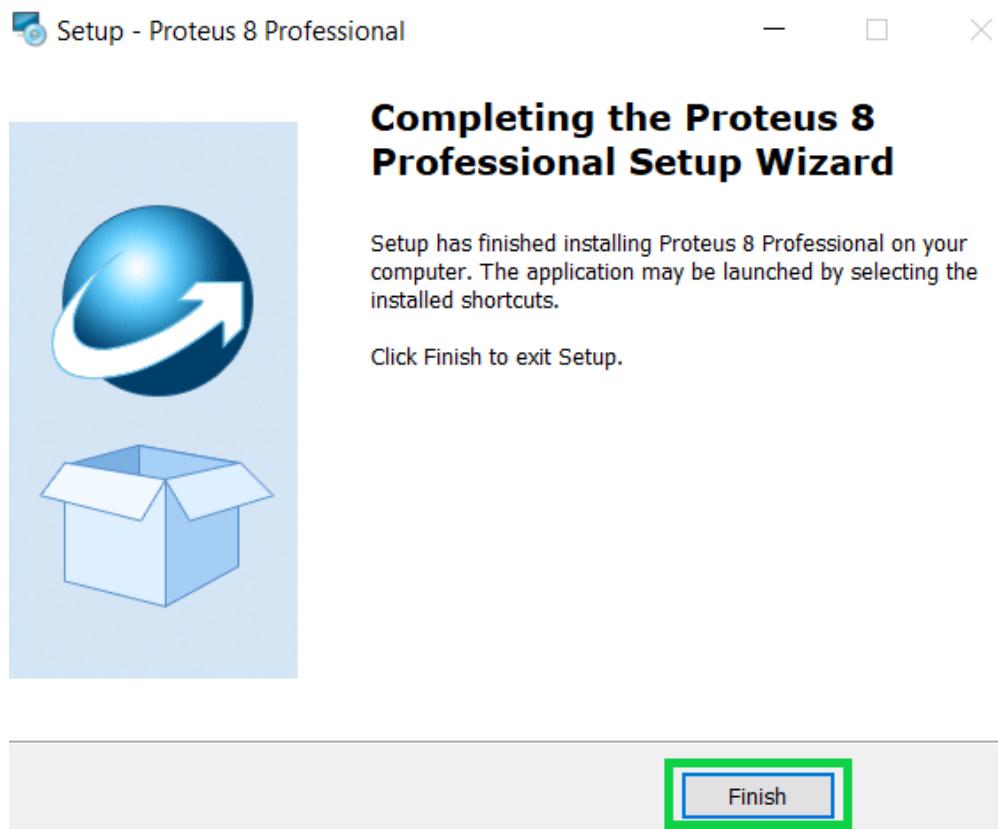
Bước 4: Lựa chọn start menu folder, sau đó click next



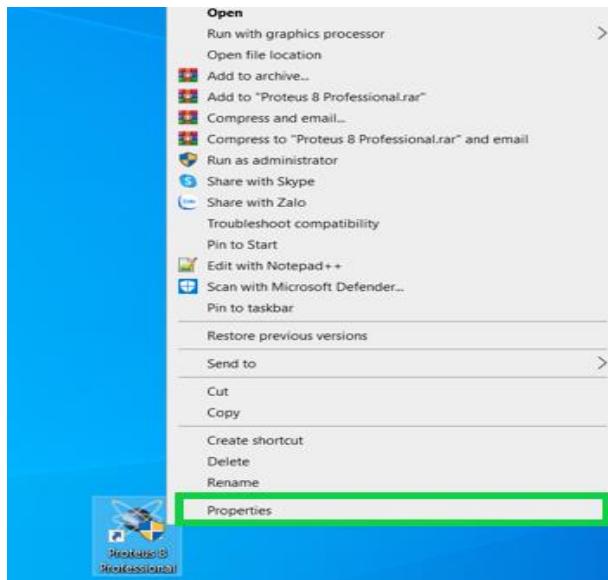
Bước 5: Đợi cho đến khi proteus cài đặt xong



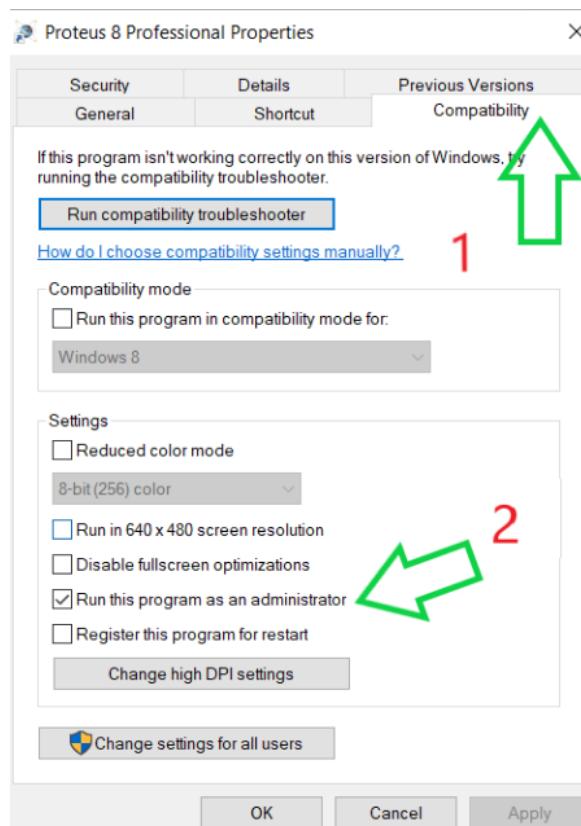
Bước 6: Sau khi cài đặt xong, click finish kết thúc cài đặt



Bước 7: Click chuột phải vào biểu tượng proteus ngoài desktop -> properties

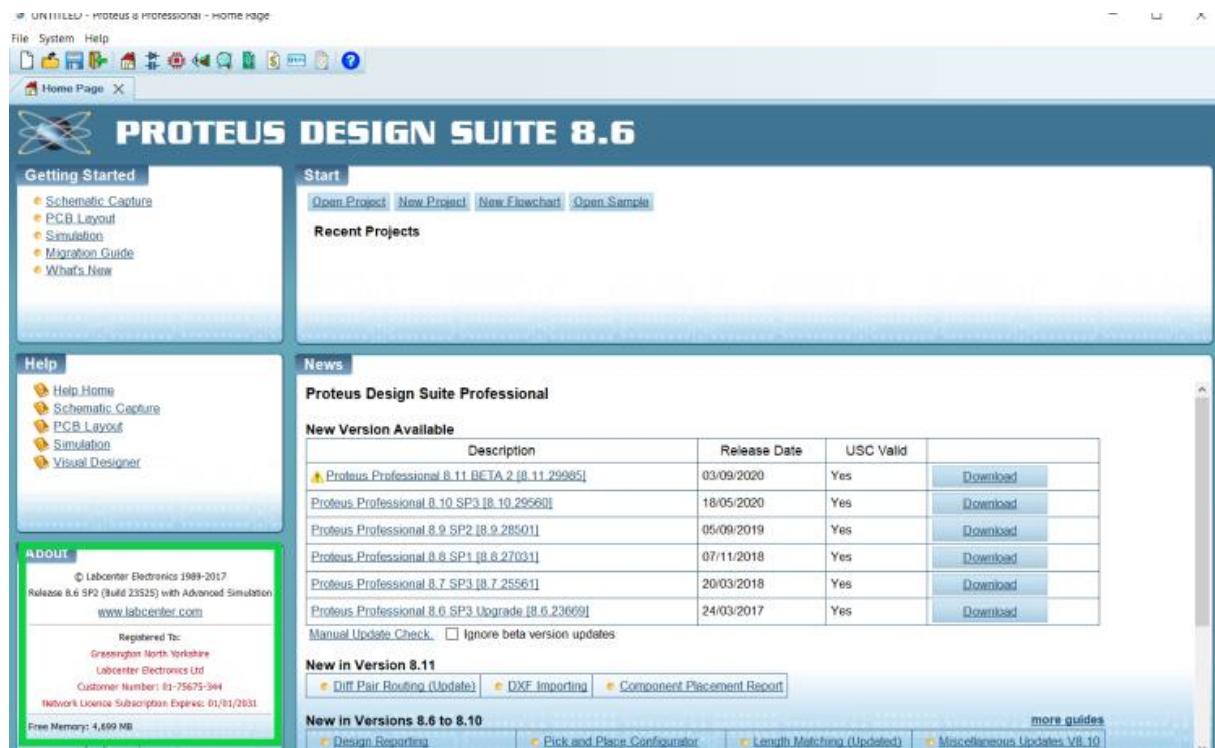


Bước 8: Chọn tab compatibility -> click chọn Run this program as an administrator.



Bước 9: Kiểm tra

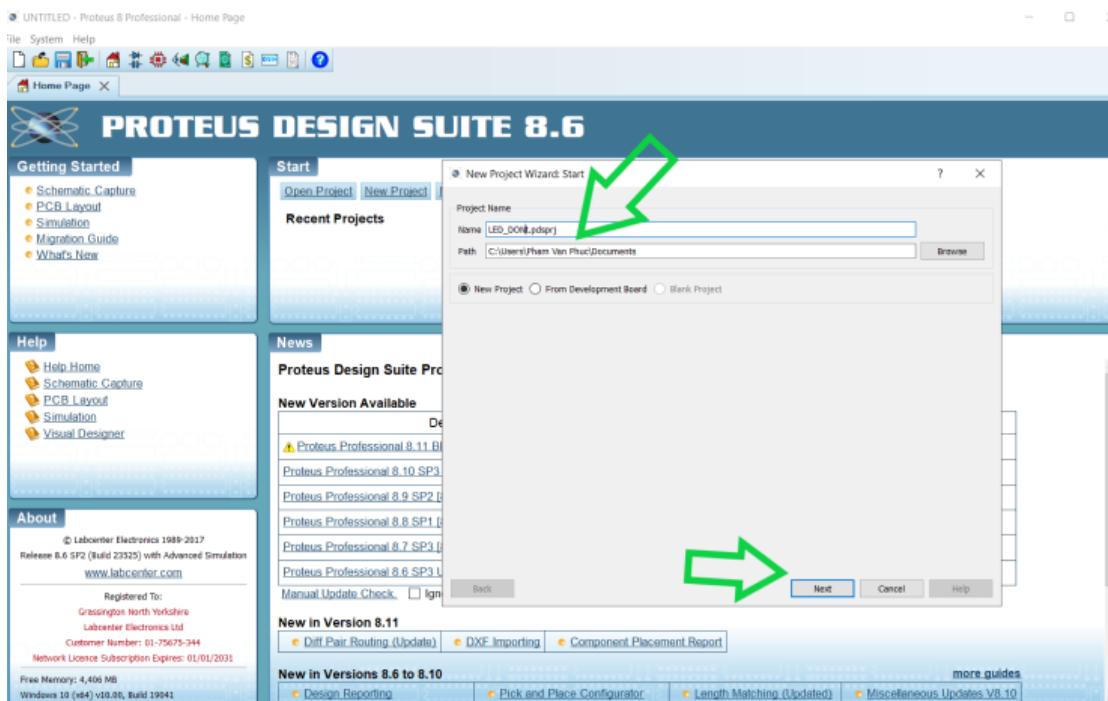
Dưới đây là hình ảnh khi bạn đã cài đặt thành công phần mềm mô phỏng proteus 8.6.



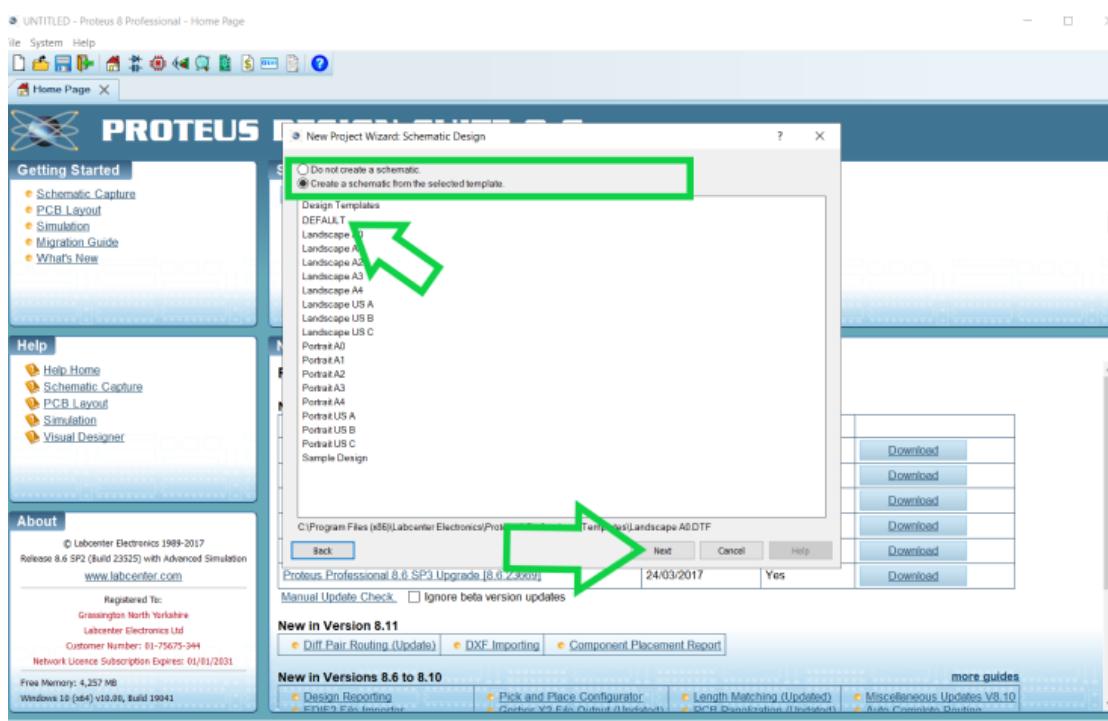
2.2 Hướng dẫn tạo project với STM32 trên proteus

Bước 1: Mở proteus 8.6. -> click chọn new project

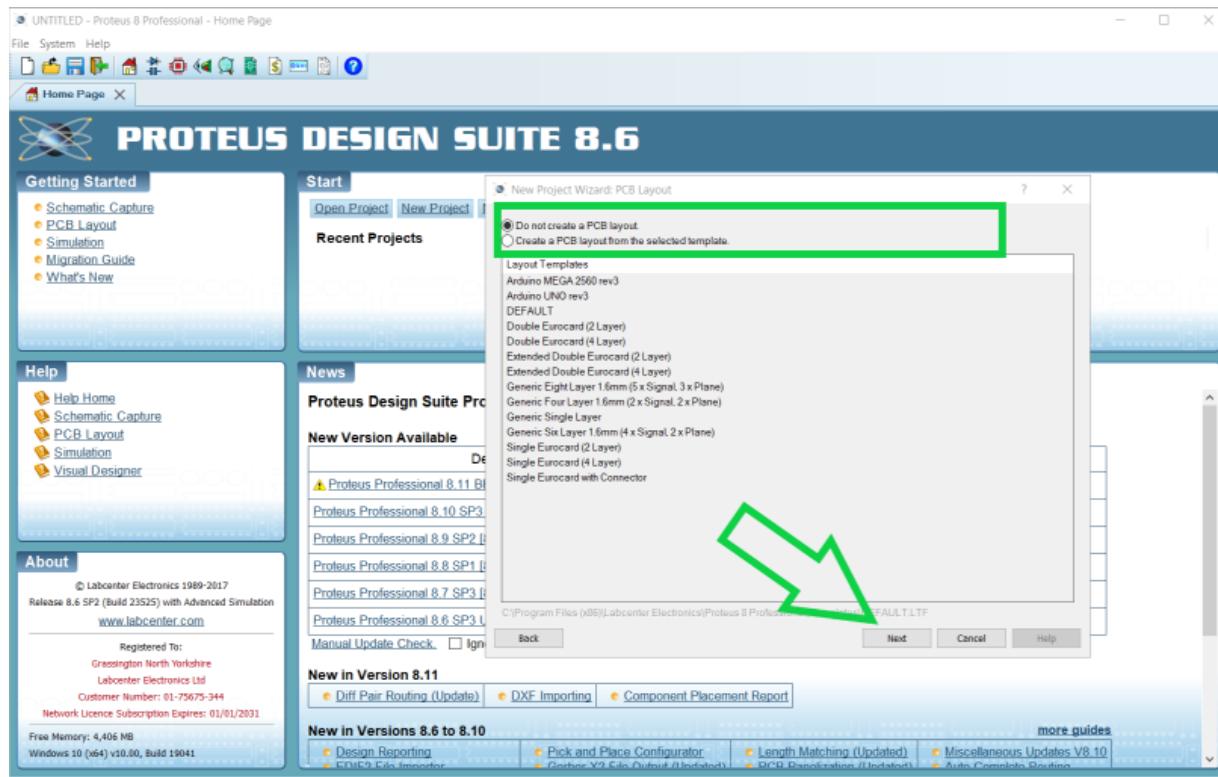
Các bạn điền tên project và chọn đường dẫn sau đó nhấn Next.



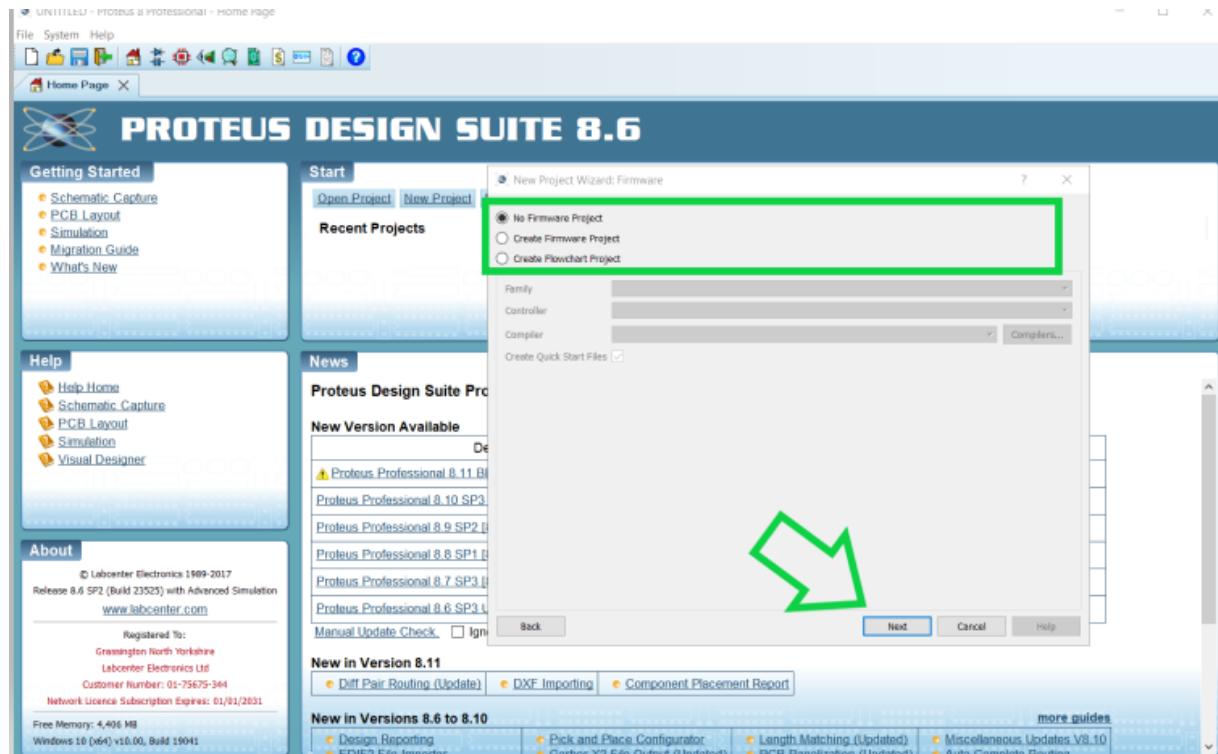
Bước 2: Click vào create a schematic from the selected template, DEFAULT-> sau đó click Next



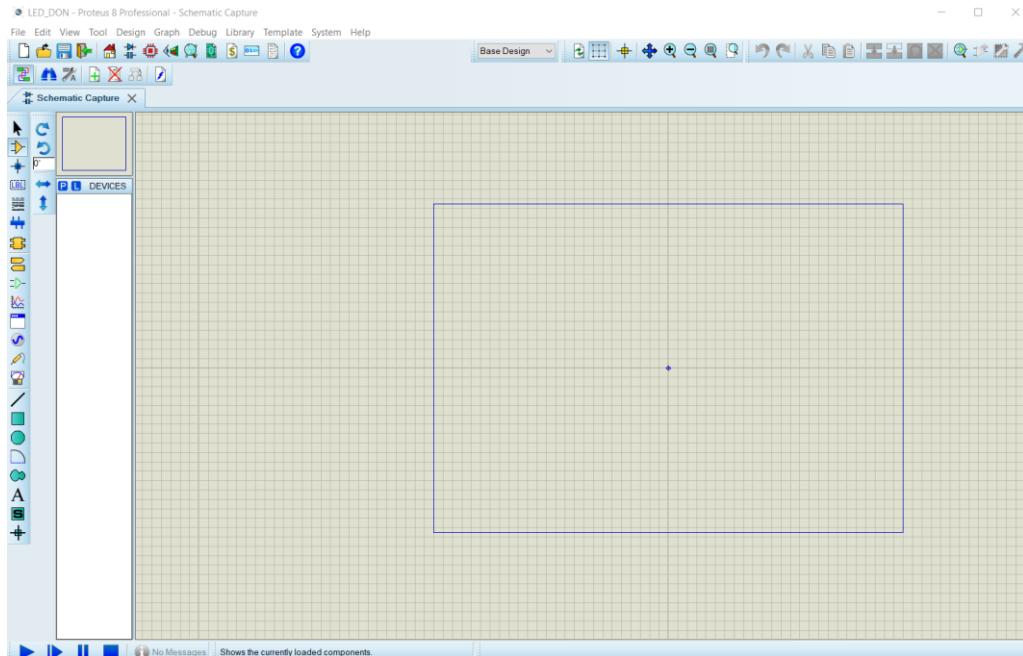
Bước 3: Click vào Do not create a PCB layout -> sau đó click Next



Bước 4: Click vào No Firmware Project -> sau đó click Next -> click Finish.



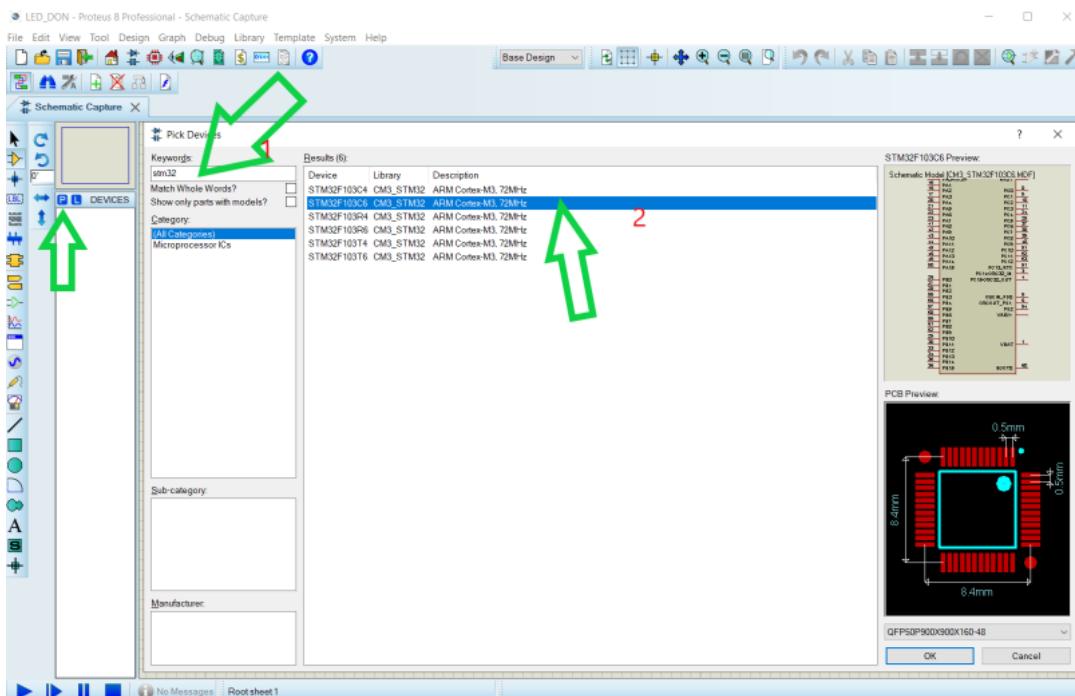
Màn hình làm việc chính.

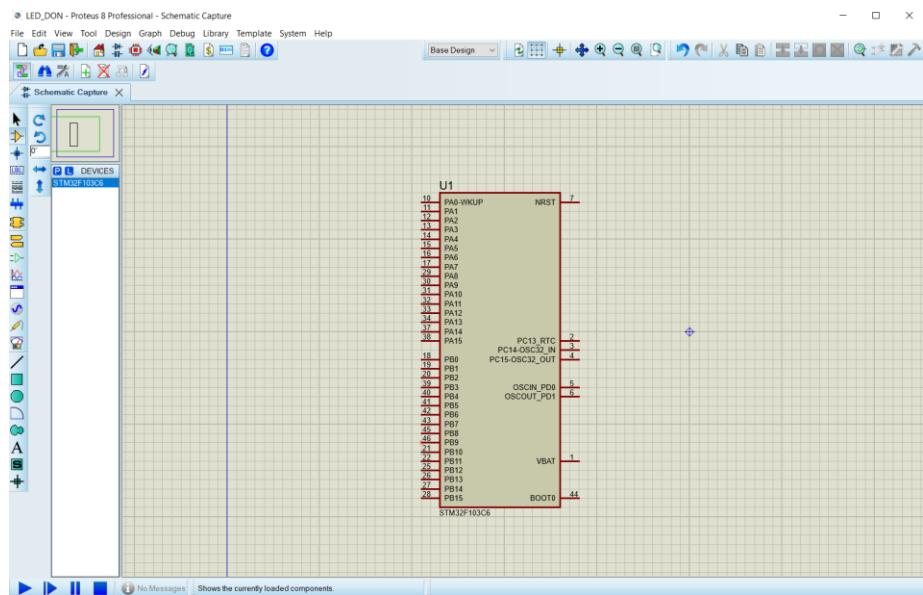


Bước 5: click vào icon P or gõ phím tắt P để vào giao diện lấy linh kiện.

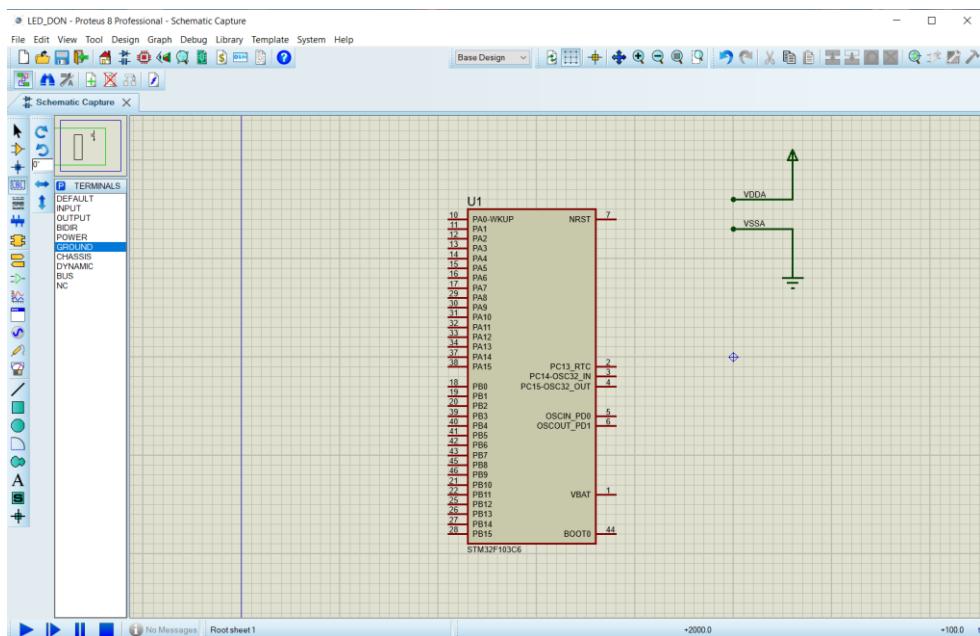
(1): gõ tên linh kiện vào tìm kiếm.

(2): Nhấp đúp để chọn linh kiện.





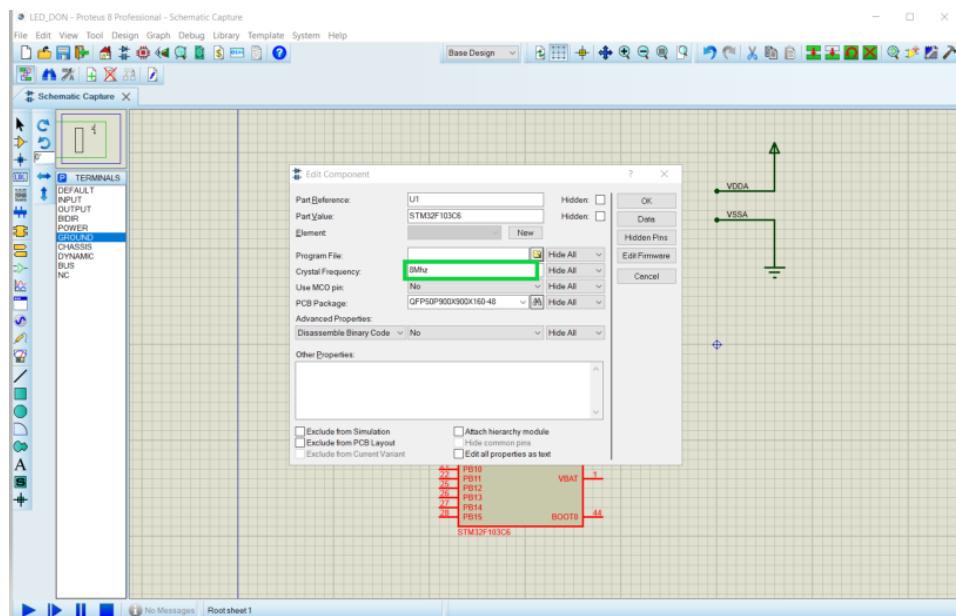
Bước 6: Cấu hình VSSA, VDDA



Bước 7: Cấu hình crystal frequency

Chú ý tần số phải khớp với tần số chúng ta đang sử dụng

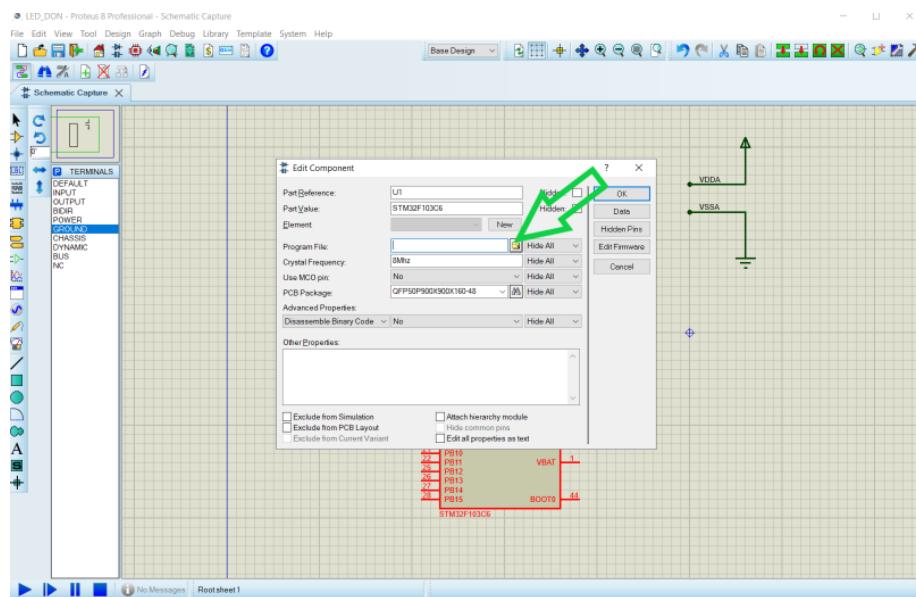
Click vào đúp chuột vào vi điều khiển -> sửa crystal frequency thành 8mhz



Bước 8: Nạp chương trình cho vi điều khiển.

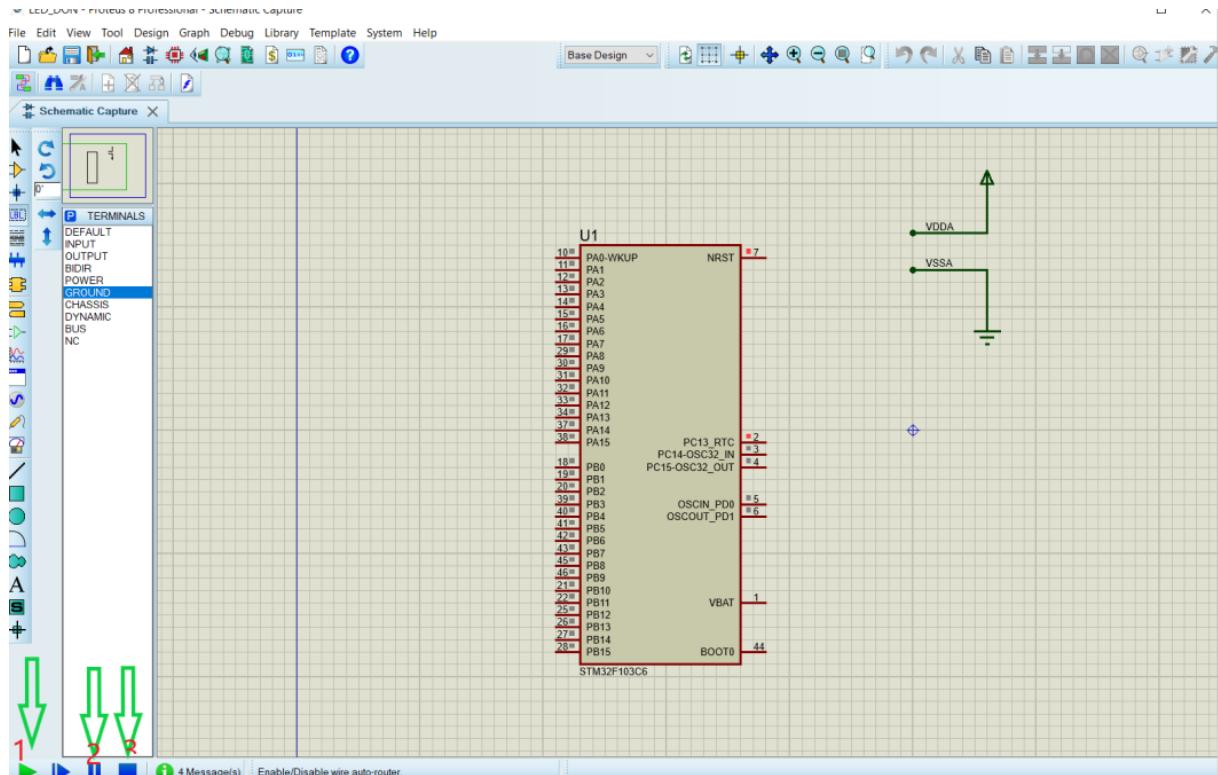
Các bạn click vào program file -> chọn file .hex để nạp vào.

Chú ý: file .hex sẽ được hướng dẫn tạo ở phần sau



Bước 9: Chạy chương trình.

- (1): Chạy mô phỏng
- (2): Tạm dừng mô phỏng
- (3): Dừng mô phỏng



III, Hướng dẫn thực hành với KIT STM32F103C8T6

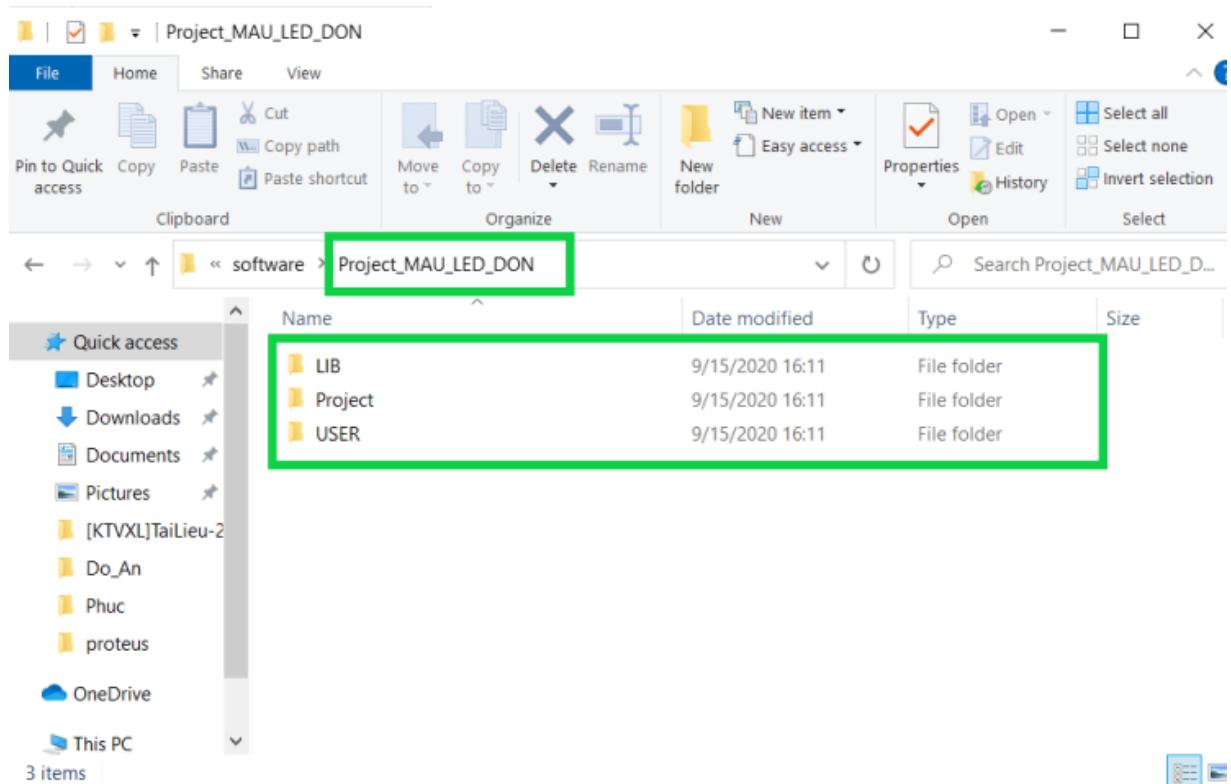
3.1. Tải thư viện chuẩn cho STM32F1XX

Tải thư viện chuẩn cho STM32F103C8T6 theo link sau và giải nén:

<https://drive.google.com/file/d/1832stosz-zLn9ZZG2bxk94i-o53YTVBE/view?usp=sharing>

3.2. Hướng dẫn tạo project

Bước 1: Tạo thư mục tên thường là tên Project_MAU_LED_DON. Trong thư mục này tạo 3 thư mục con **LIB**, **Project**, **USER**.



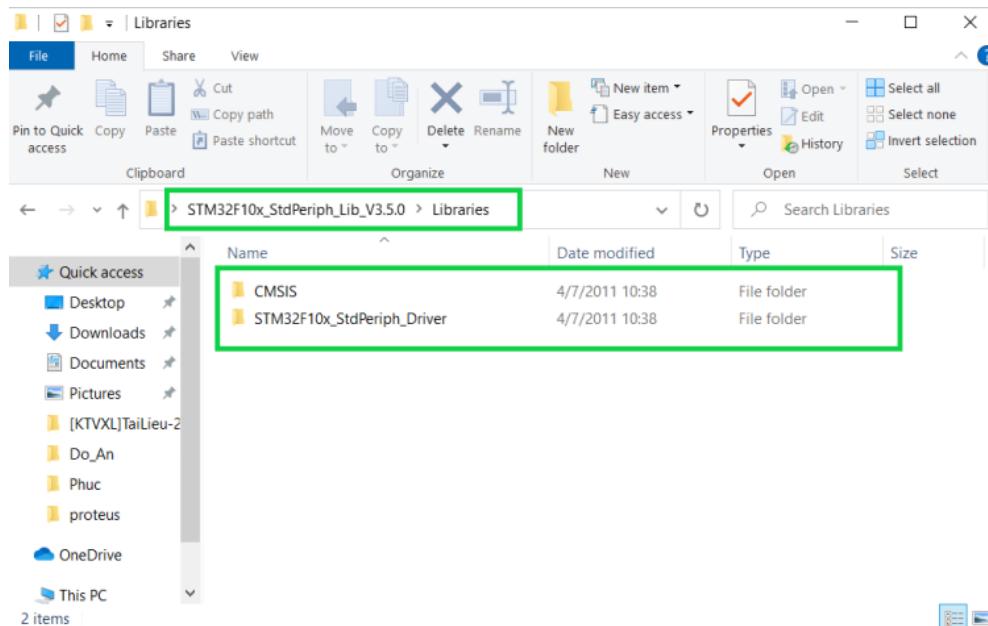
LIB: chứa thư viện chuẩn của nhà sản xuất.

Project: chứa file liên quan đến project KeikC như là file .hex, object

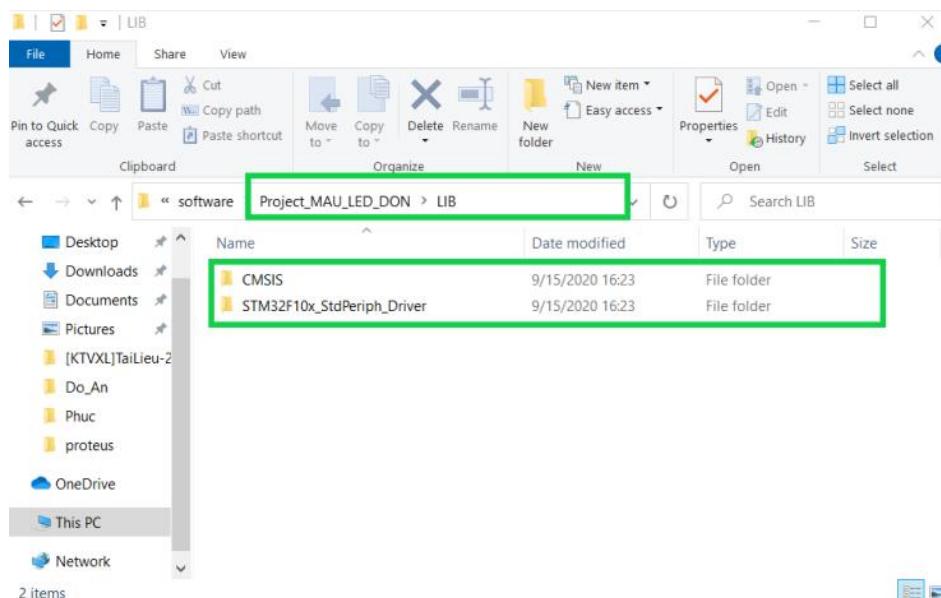
USER: chứa file source của người dùng.

Bước 2: Vào thư mục ta vừa giải nén trước đó(phần 3.1).

Copy 2 thư mục **CMSIS** và **STM32F10x_StdPeriph_Driver** trong đường dẫn
...keil\STM32F10x_StdPeriph_Lib_V3.5.0\Libraries.



Pase vào thư mục LIB.

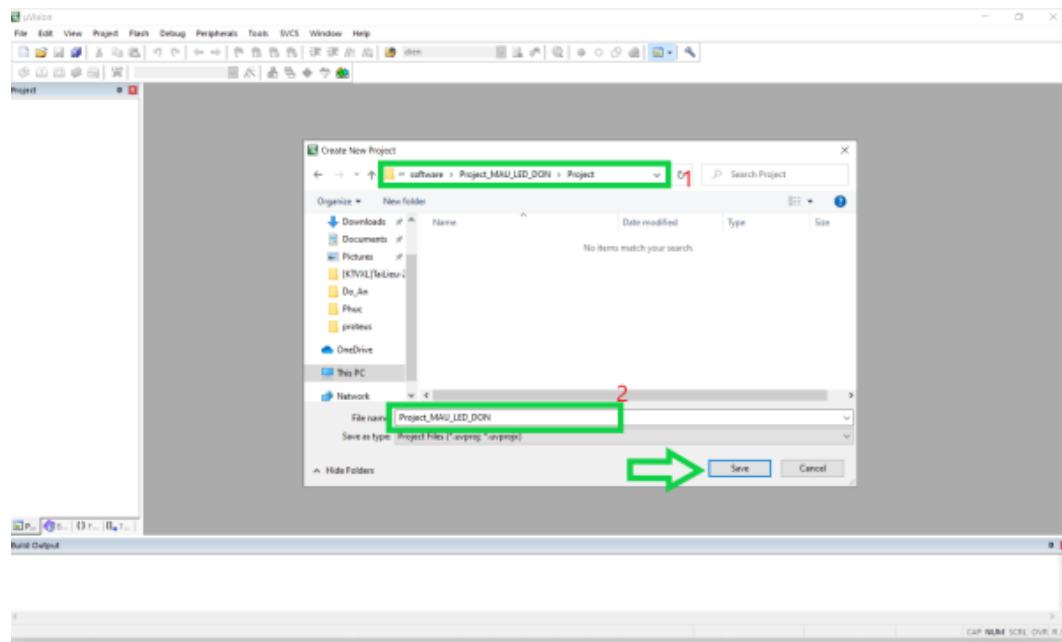


Bước 3: Khởi động keilC lên để bắt đầu tạo project mới

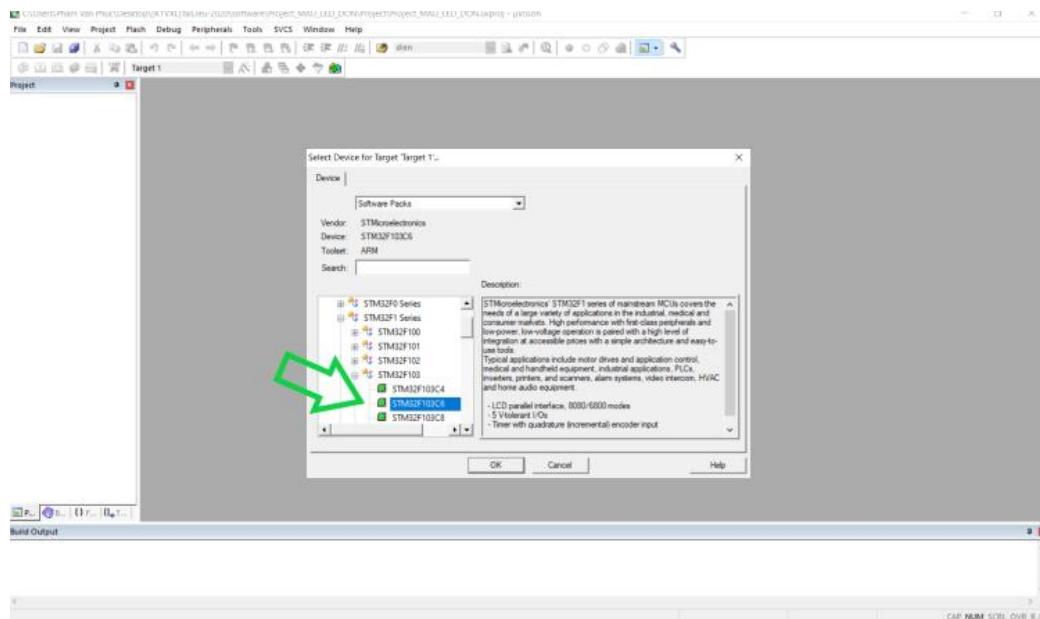
Chọn Project -> New Uvision project.

Chọn đường dẫn chứa file project : chỏ tới thư mục Project trong thư mục Project_MAU_LED_DON.(1)

Đặt tên cho project.(2)=> bấm Save.

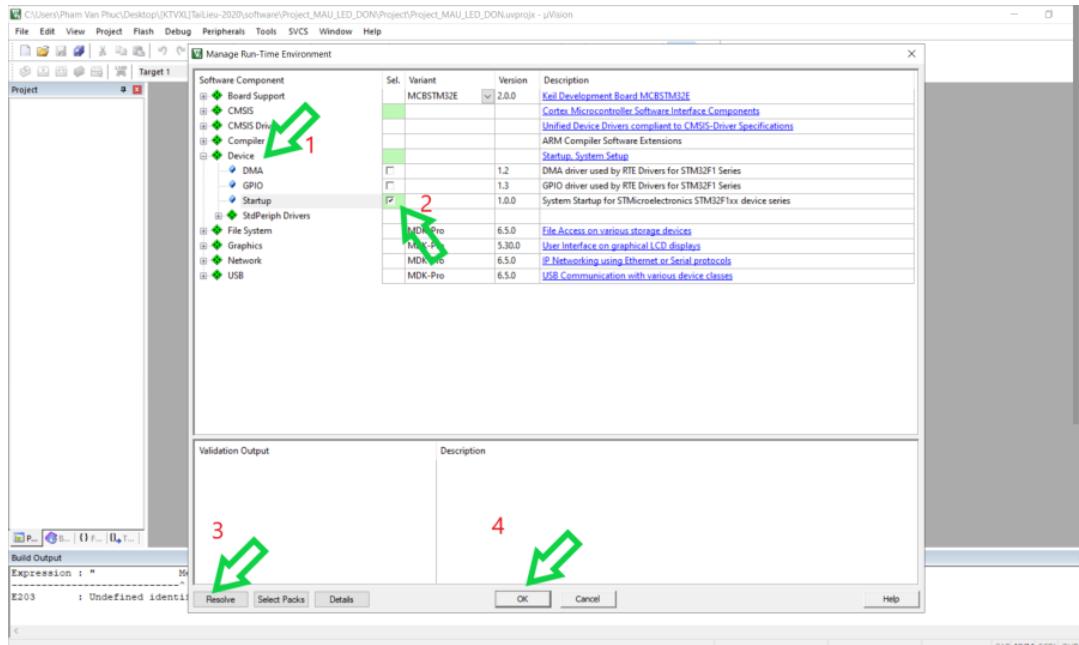


Bước 4: Cửa sổ mới được mở ra để cho chúng ta chọn chip. Ở đây chúng ta sử dụng STM32F103C6T8 nên sẽ chọn MCU là STM32F103C6 => nhấn OK => OK.



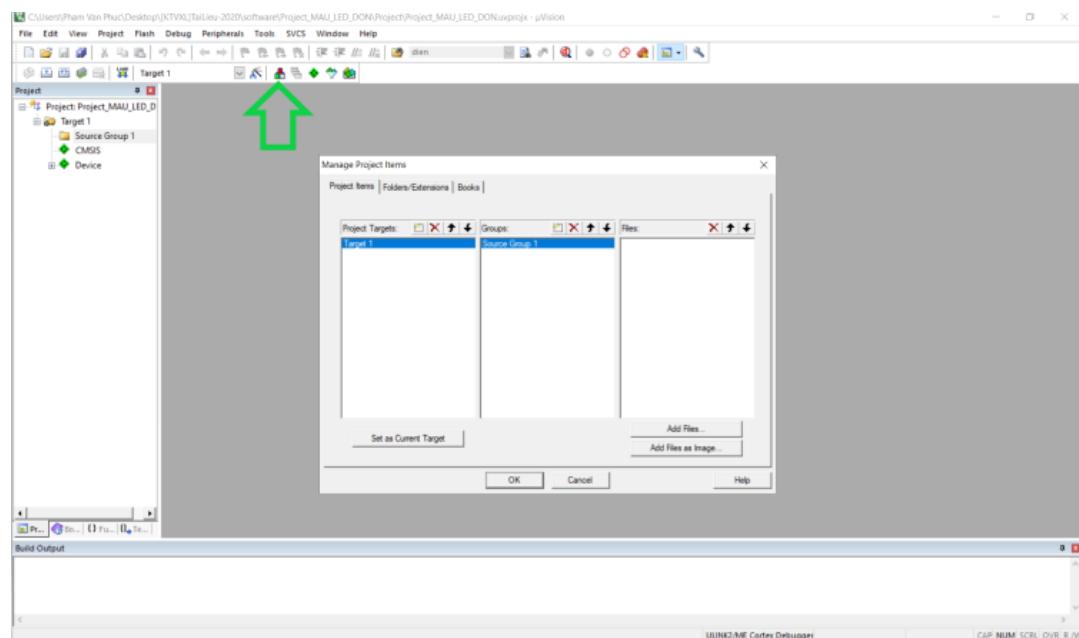
Bước 5: Cửa sổ manage run hiện ra.

Click Chọn Device(1) -> Startup(2) -> Resolve(3)->OK(4).

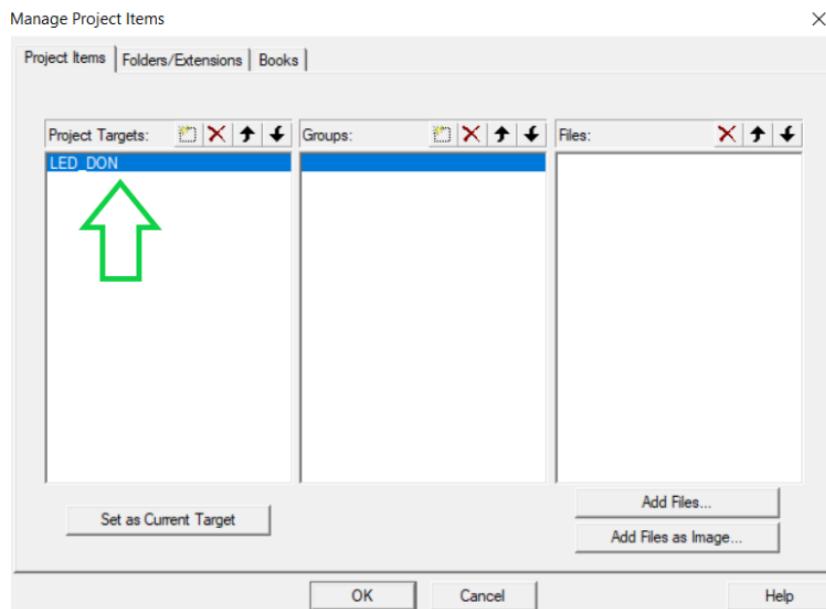


Bước 6: Thêm các file thư viện, source vào project

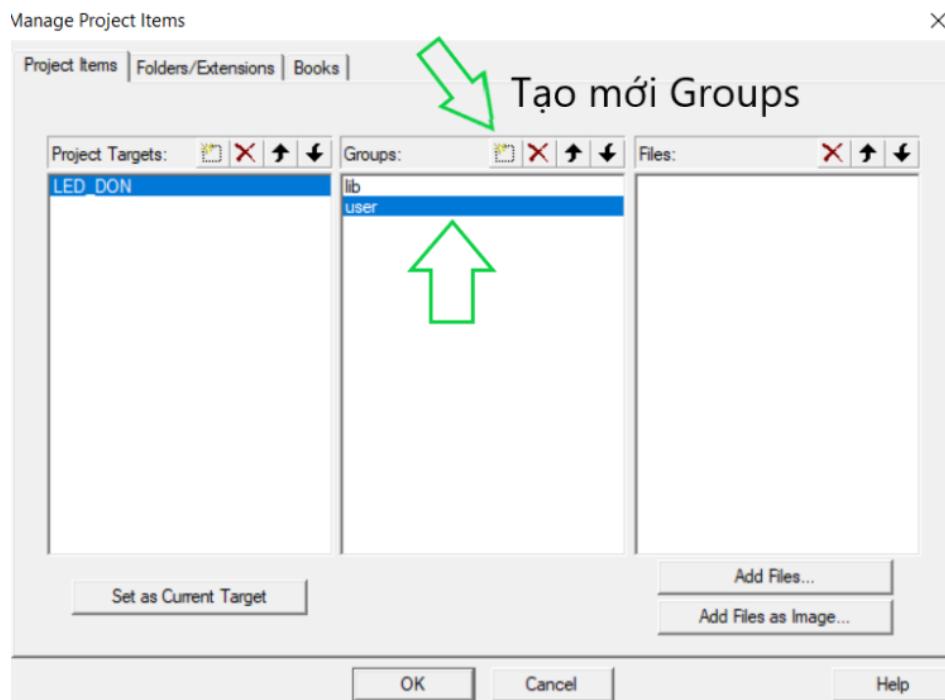
Mở cửa sổ **Manage project item** bằng cách kích vào biểu tượng hình 3 ô vuông màu xanh, đỏ trắng hoặc vào đường dẫn sau : **Project => manage => project items**.



1, Chính sửa tên ở ô **Projects target** thành tên bạn muốn ở đây chúng ta để GPIO (thường là tên project) :



2, sửa tên ở ô **Groups** thành **lib** (dùng để chứa các thư viện mẫu) và thêm **group user** (thường chứa file main, file chương trình và các thư viện tự phát triển).

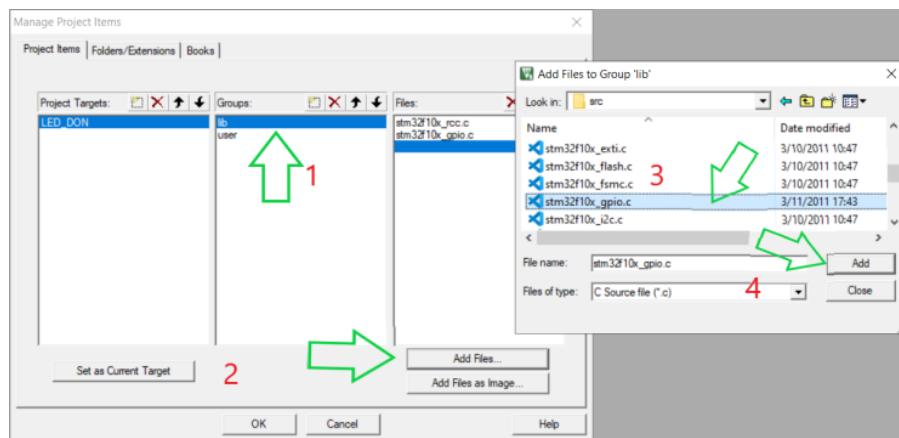


3, Add Files trong thư viện mẫu có sẵn vào project của chúng ta vừa tạo.

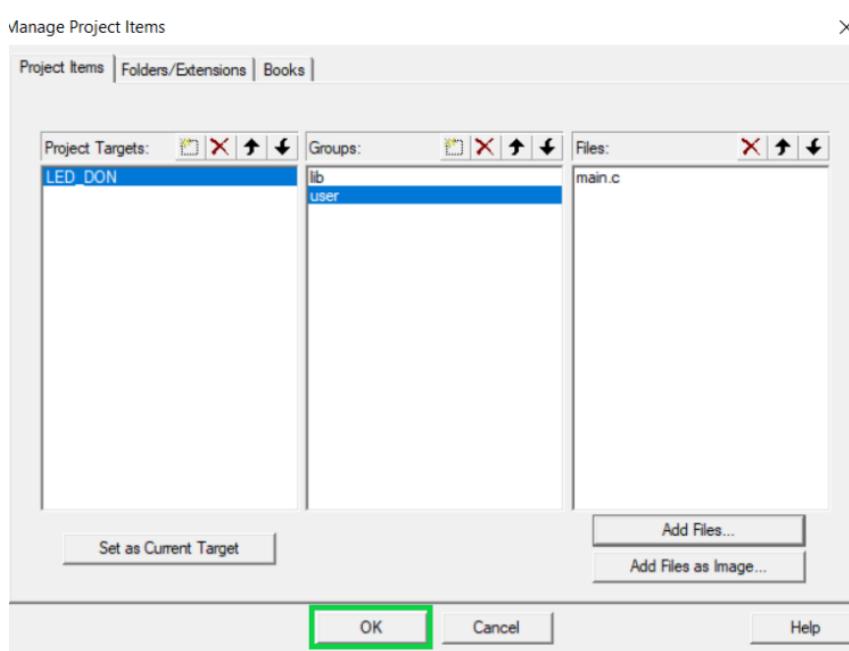
Ta cần add 2 file là stm32f10x_gpio.c và stm32f10x_rcc.c. Do ta chỉ làm việc gói ngoại vi GPIO nên ta chỉ cần add 2 file trên là đủ.

Add File vào Group lib

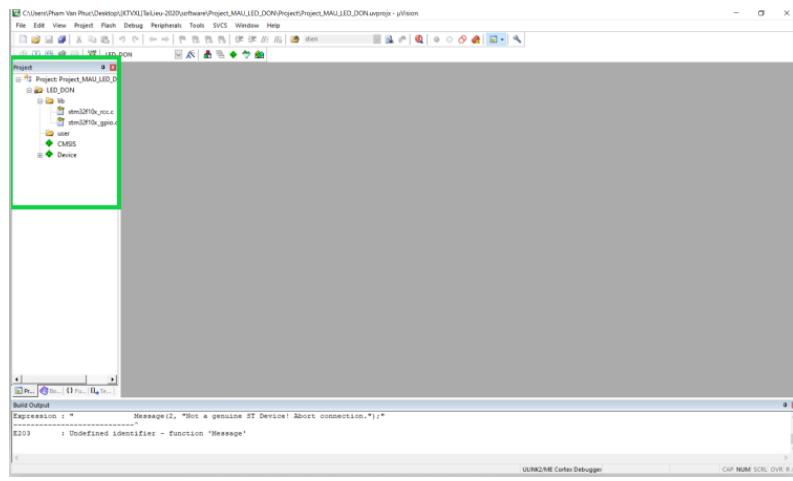
(1)Click chọn lib -> (2)Add File ->(3)Chọn đường dẫn tới nơi chứa thư viện (“Project_MAU_LED_DON\LIB\STM32F10x_StdPeriph_Driver\src”) -> (4)add 2 File trên vào project.



Click vào OK để kết thúc việc add file.



Nếu add thành công, kết quả bên sidebar xuất hiện các file, group mình vừa tạo.



Add File main.c vào Group user.

Do file **main.c** chưa được tạo, vì vậy ta phải tạo file vào add vào group user.

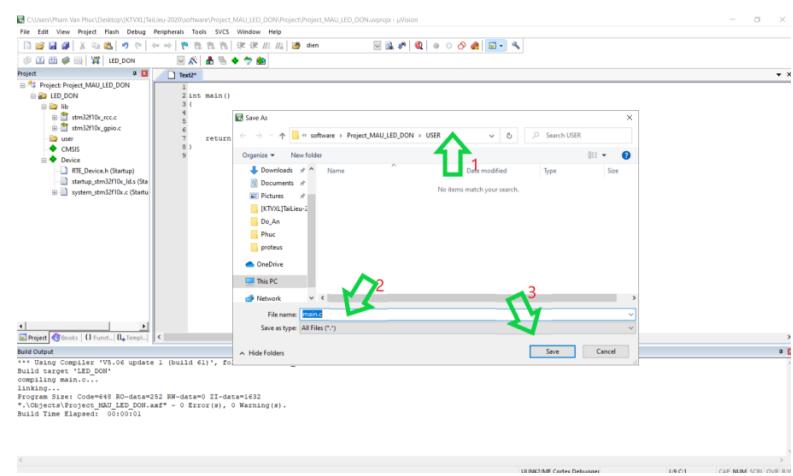
Tạo file main.c

Ấn tổ hợp phím **ctr+N** or **file -> new**

Thêm đoạn mã vào file main.c

```
int main()
{
    return 0;
}
```

sau đó ấn **ctr+S** để lưu file với tên main.c

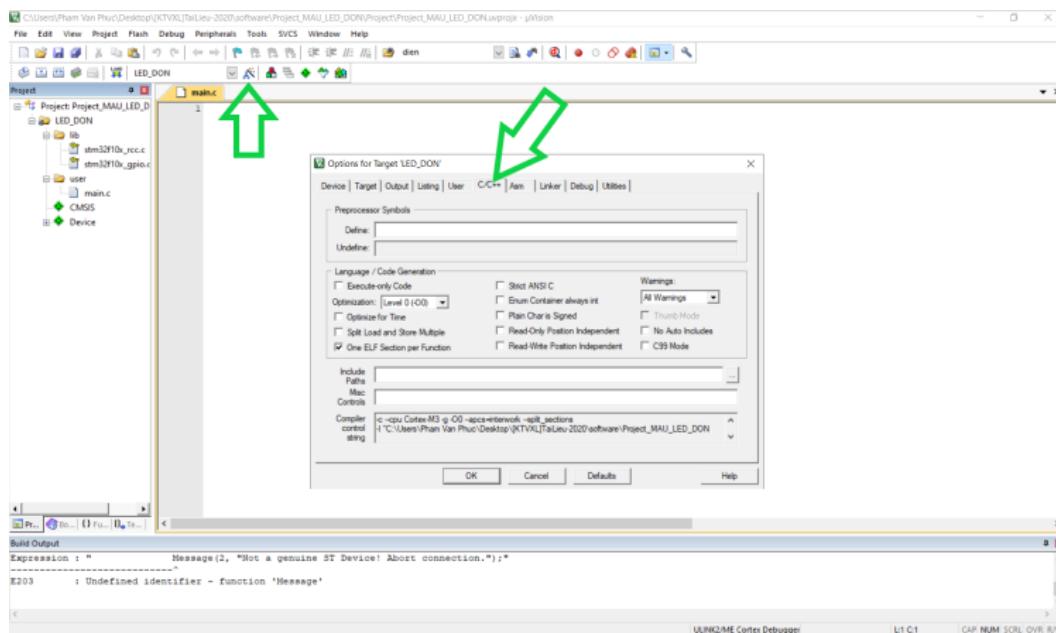


- (1) đường dẫn.
- (2) Tên file
- (3) Lưu file

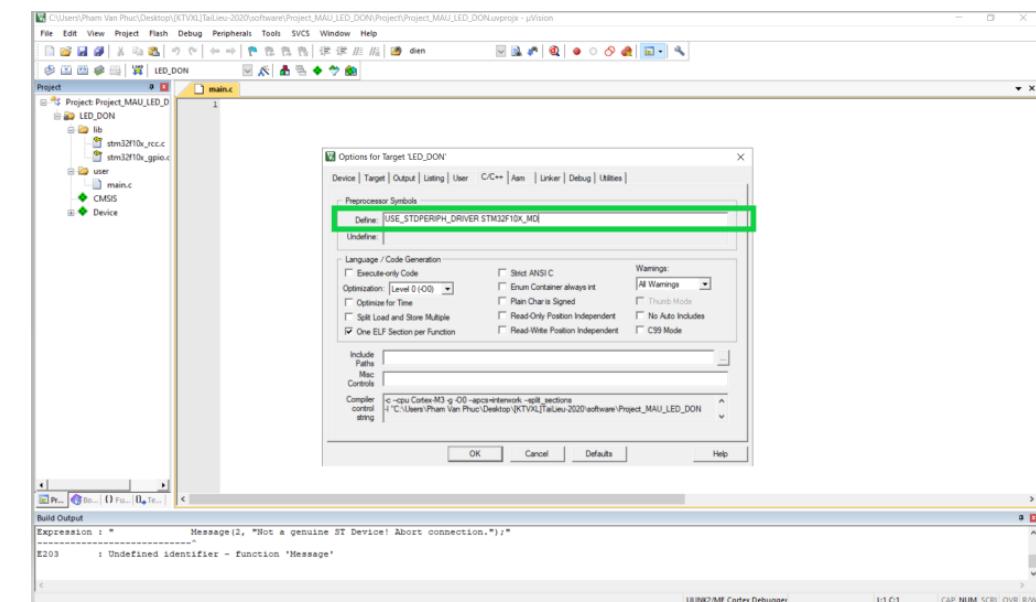
Add file main.c vào group user tương tự như ta 2 file là stm32f10x_gpio.c và stm32f10x_rcc.c

Bước 7: Cấu hình compiler C/C++

Click chọn Target -> tab C/C++

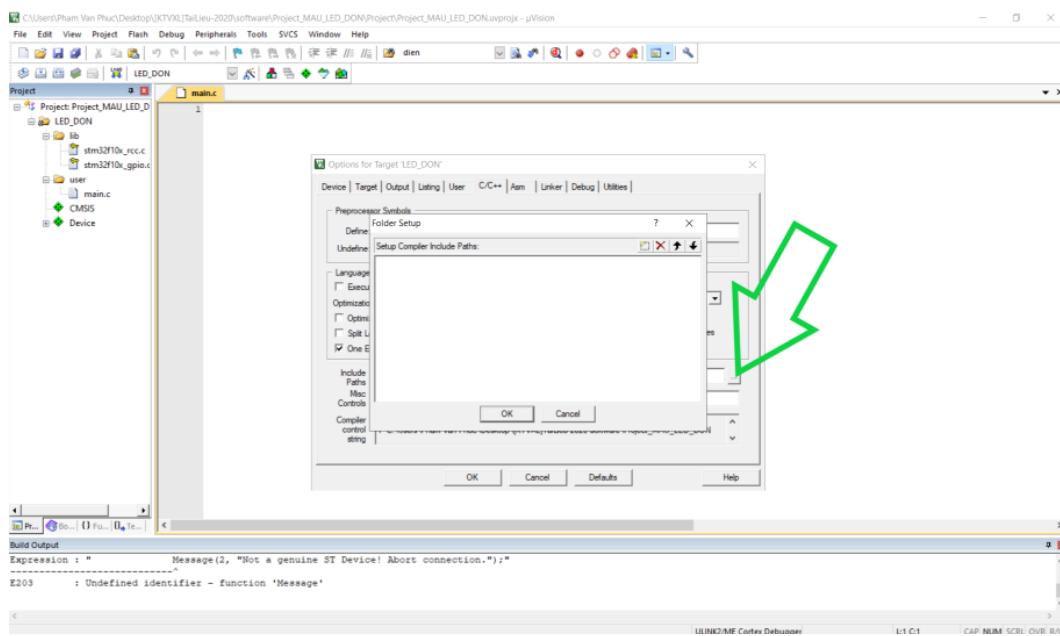


Thêm macro sau vào dòng define “USE_STDPERIPH_DRIVER STM32F10X_MD”



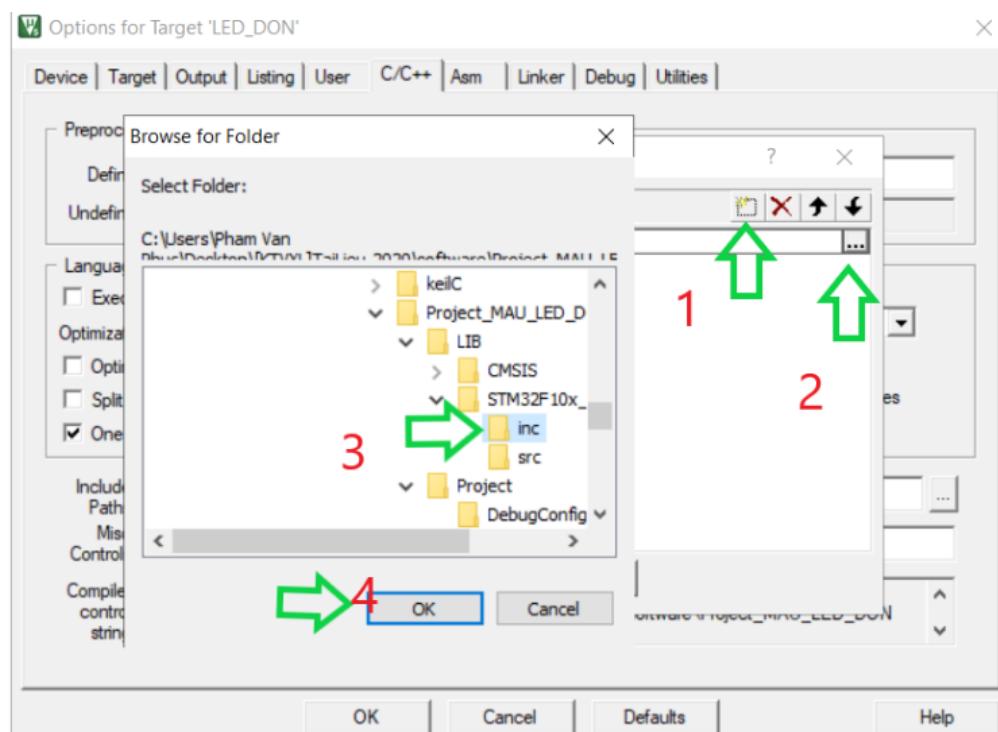
Chọn đường dẫn tới header file

Click vào **Include Path**



Để thêm một đường dẫn ta làm như sau:

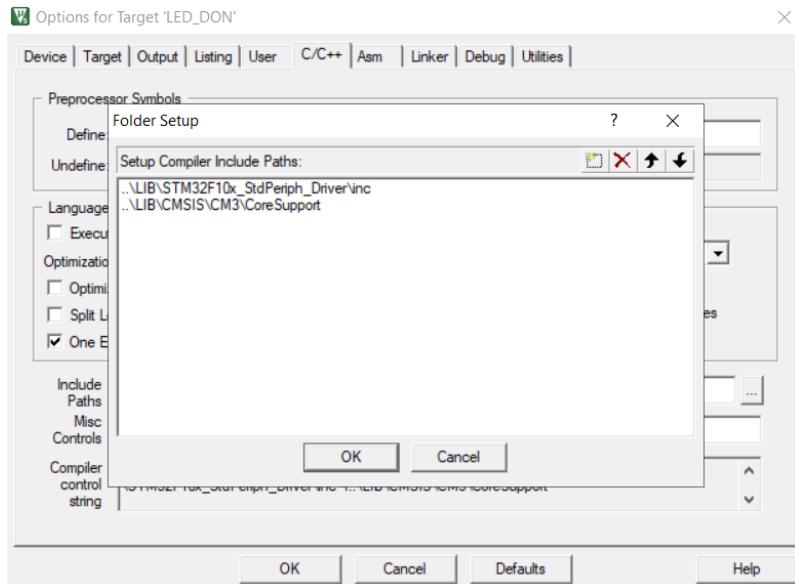
- (1) Tạo một đường dẫn mới ->(2) chọn đường dẫn -> (3) Trỏ đến nơi chưa header file -> (4) OK để lưu.



Ta thêm 3 đường dẫn như sau

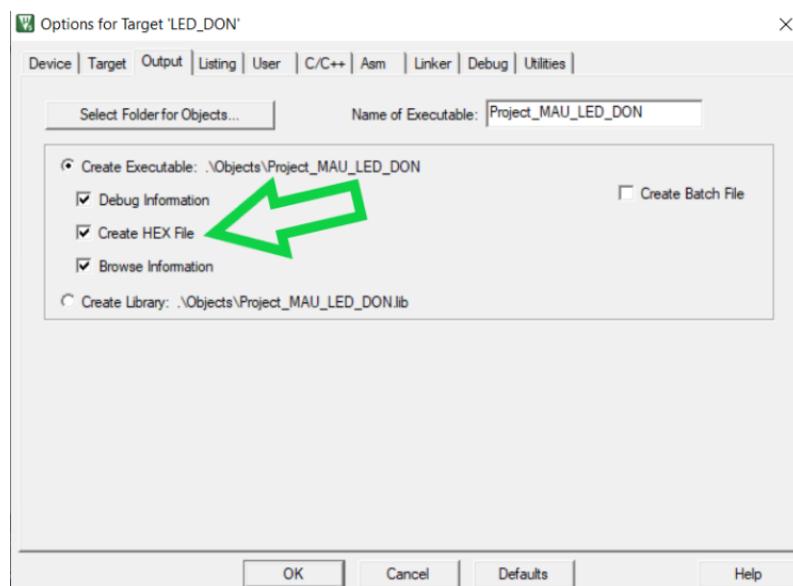
..\\LIB\\STM32F10x_StdPeriph_Driver\\inc

..\\LIB\\CMSIS\\CM3\\CoreSupport

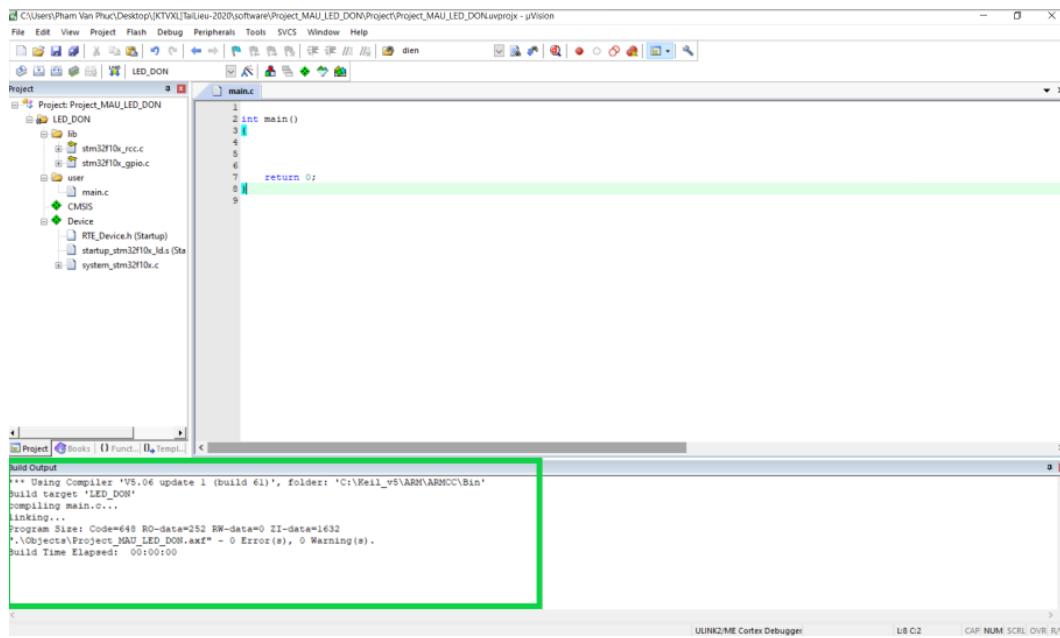


Bước 8: Lựa chọn tạo file hex khi biên dịch.

Vào target -> tab Output -> Click chọn Create HEX File

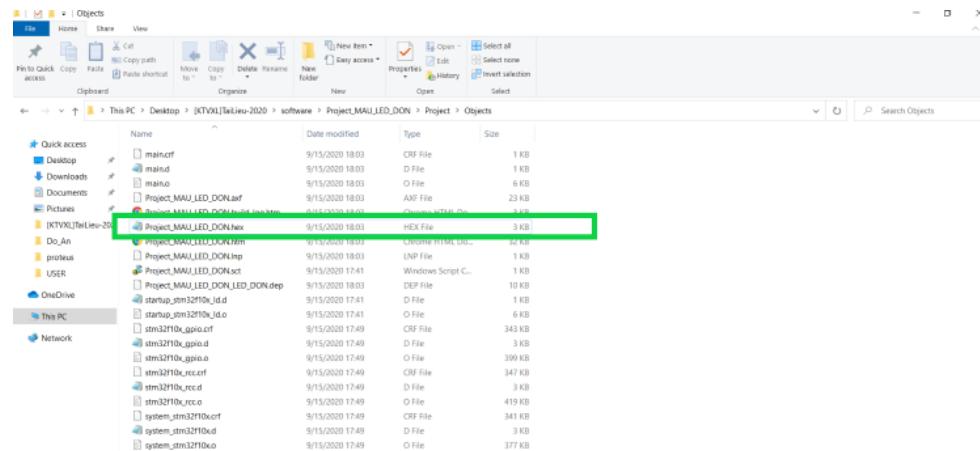


Bước 9: Kích vào biểu tượng build (F7) nếu build ở lần đầu tiên và rebuild cho các lần tiếp theo. Xem kết bảng thông báo build output nếu không xuất hiện 0 lỗi tức là đã thành công.



Bước 10: Kiểm tra File hex được tạo ra chưa.

⇒ Truy cập đường dẫn “Project_MAU_LED_DON\Project\Objects\”.



CHÚ Ý

Nếu làm theo hướng dẫn mà vẫn không tạo được project:

Các bạn click vào link sau để tải về.

<https://drive.google.com/file/d/1DhqUxiQekW9AzgNHbmiFCKTi2q7zrsGj/view?usp=sharing>

3.3. Lập trình nháy LED với STM32F103C8T6.

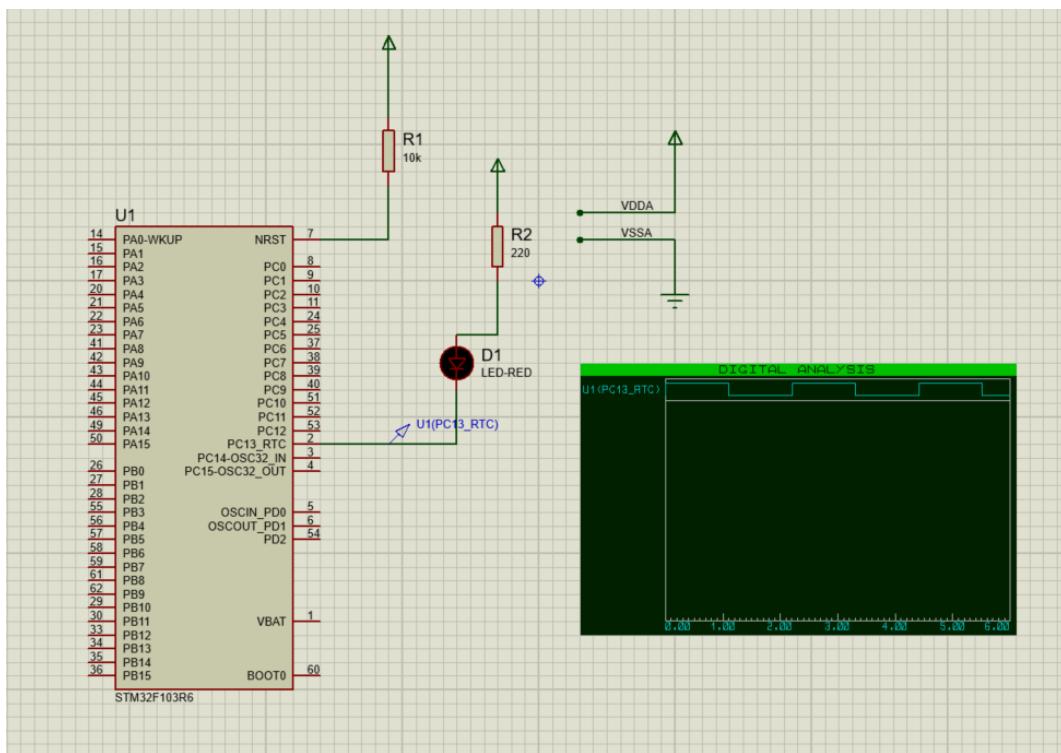
a, Sơ lược về lý thuyết.

GPIO là từ viết tắt của General purpose I/O ports tạm hiểu là nơi giao tiếp chung giữa tín hiệu ra và tín hiệu vào. GPIO là bài cơ bản, cần nắm vững khi học bất kì một VĐK nào đó. Cần hiểu được các thuật ngữ, chế độ, cấu hình, số lượng... của các chân GPIO. Ở STM32 thì các chân GPIO chia ra làm nhiều Port vd: PortA, PortB..... Số lượng Port phụ thuộc vào số lượng chân(pin) và cách gọi phụ thuộc vào nhà sản xuất (ví dụ VĐK X có PortA mà lại không có PortD). Mỗi Port thường có 16 chân đánh số từ 0 -> 15 tương ứng với mỗi chân là 1bit (nhiều dòng có 32 chân). Mỗi chân có 1 chức năng khác nhau như analog input, external interrupt.. hay đơn thuần chỉ là xuất tín hiệu on/off ở mức 0,1. Chức năng của mỗi chân thì chúng ta cần tra datasheet của nhà sản xuất trước khi lập trình hoặc thiết kế mạch. Các mode GPIO của STM32:

- **Input floating:** cấu hình chân I/O là ngõ vào và để nổi.
- **Input pull-up:** cấu hình chân I/O là ngõ vào, có trở kéo lên nguồn.
- **Input-pull-down:** cấu hình chân I/O là ngõ vào, có trở kéo xuống GND.
- **Analog:** cấu hình chân I/O là Analog, dùng cho các mode có sử dụng ADC/DAC
- **Output open-drain:** cấu hình chân I/O là ngõ ra, khi output control = 0 thì N-MOS sẽ dẫn, chân I/O sẽ nối VSS (GND), còn khi output control = 1 thì P-MOS và N-MOS đều không dẫn, chân I/O được để nổi – không có điện thế. Nếu bên ngoài nối với VCC qua điện trở thì mặc định xuất là 3.3V, khi đó điều khiển mức 0 sẽ xuống GND. (Sử dụng trở treo).
- **Output push-pull:** cấu hình chân I/O là ngõ ra, khi output control = 0 thì N-MOS sẽ dẫn, chân I/O sẽ nối VSS, còn khi output control = 1 thì P-MOS dẫn, chân I/O được nối VDD.
- **Alternate function push-pull:** sử dụng chân I/O vừa là ngõ ra và vừa là ngõ vào, tuy nhiên sẽ không có trở kéo lên và kéo xuống ở input, chức năng output giống Output push-pull. Ngoài ra nó còn để sử dụng cho chức năng remap.
- **Alternate function push-pull:** sử dụng chân I/O vừa là ngõ ra và vừa là ngõ vào, tuy nhiên sẽ không có trở kéo lên và kéo xuống ở input.

Bài 1: Nháy nháy đèn ở chân PortC chân 13.

a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

void Delay(uint32_t);
void GPIO_Config(void);
void Clock_Config(void);

int main(void){
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    GPIO_Config();

    while(1){
        GPIO_SetBits(GPIOC, GPIO_Pin_13);
        Delay(100);
        GPIO_ResetBits(GPIOC, GPIO_Pin_13);
        Delay(100);
    }
}
/*Delay tuong doi*/
void Delay(uint32_t t){
    unsigned int i,j;

    for(i=0;i<t;i++){
        for(j=0;j< 0x2AFF; j++);
    }
}

void GPIO_Config(){
    GPIO_InitTypeDef GPIO_InitStructure;

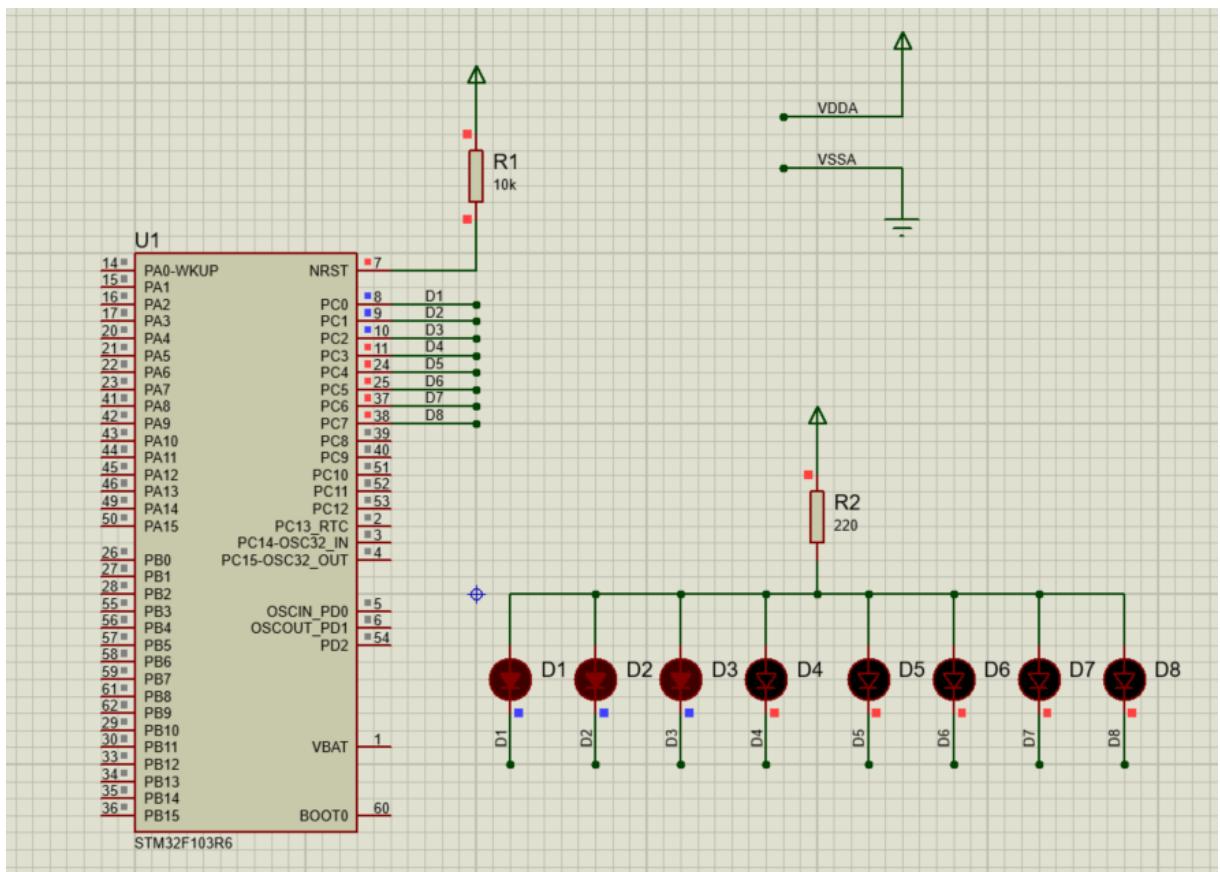
    /*enble clock for GPIOC*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    /*Configuration GPIO pin*/
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_13|GPIO_Pin_12|GPIO_Pin_11|GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOC,&GPIO_InitStructure);
}

void Clock_Config(void){
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div2);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div2);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
    /* Select PLL as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till PLL is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}
}
```

Bài 2: Điều khiển 8 led sáng tắt theo mong muốn.

a, Mô phỏng trên proteus



b, code

```
//dieu khien 8 led
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

uint16_t sangdan[8]={0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00};
void Delay(uint32_t);
void GPIO_Config(void);
void Clock_Config(void);

int main(void){
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    GPIO_Config();

    while(1){
        for( int i = 0; i < 8; i++){
            GPIO_Write(GPIOC, sangdan[i]);
            Delay(100);
        }
    }
    /*Delay tuong doi*/
    void Delay(uint32_t t){
        unsigned int i,j;

        for(i=0;i<t;i++){
            for(j=0;j< 0x2AFF; j++);
        }
    }
    void GPIO_Config(){
        GPIO_InitTypeDef GPIO_InitStructure;
        /*enable clock for GPIOC*/
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
        /*Configuration GPIO pin*/
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
        GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
    }
    void Clock_Config(void){
        /* RCC system reset */
        RCC_DeInit();
        /* HCLK = SYSCLK */
        RCC_HCLKConfig(RCC_SYSCLK_Div1);
        /* PCLK2 = HCLK */
        RCC_PCLK2Config(RCC_HCLK_Div2);
        /* PCLK1 = HCLK/2 */
        RCC_PCLK1Config(RCC_HCLK_Div2);
        /*enable HSI source clock*/
        RCC_HSICmd(ENABLE);
        /* Wait till PLL is ready */
        while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
        /* Select PLL as system clock source */
        RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
        /* Wait till PLL is used as system clock source */
        while(RCC_GetSYSCLKSource() != 0x00) {}
    }
}
```

Câu hỏi ôn tập

- Muốn thay đổi thời gian sáng, tắt thì làm thế nào? Tại sao trong hàm Delay lại dùng giá trị 0x2AFF?
- Muốn 1 đèn ở một vị trí bất kì (trong Port A/B/C) sáng tắt thì sao? Thủ code một chương trình nhấp nháy 4 led ở port A.
- Muốn 8 đèn sáng tắt luân phiên (40-04) thì làm thế nào?
- Muốn 8 đèn sáng dần rồi tắt dần thì làm thế nào?
- Lập trình để đèn sáng lần lượt từ trái qua phải, sau đó lại sáng lần lượt từ phải qua trái? Bạn cảm thấy thế nào?

III, Bài tập

Bài tập nâng cao theo nhóm

Lập trình trên 8 led (xếp thành 1 cột) của một Port của STM32 để thực hiện chức năng theo trò chơi xếp hình, hoặc nước nhỏ giọt. Cụ thể như sau:

- Đầu tiên trên cùng có 1 LED sáng sau đó LED rời dần xuống dưới và sẽ dừng lại ở vị trí cuối cùng.
- Khi LED số 1 chạm đến vị trí dưới cùng thì xuất hiện 1 LED sáng ở vị trí ở trên cùng và lặp lại việc rời xuống dưới và dừng lại khi đến vị trí ngay trên con LED cũ.
- Lặp lại như vậy cho đến khi số LED đầy trên cột và sẽ bắt đầu lại từ đầu.

Bài tập cá nhân

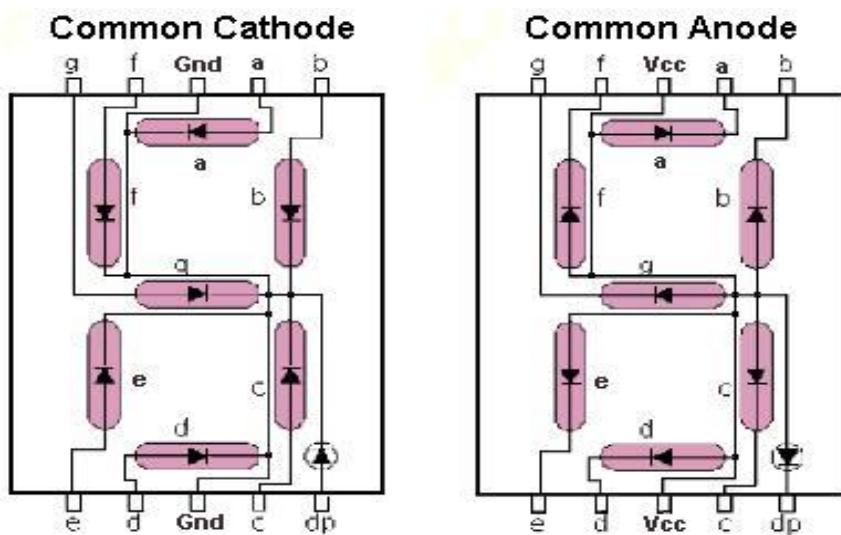
Hãy sắp xếp 16 đèn LED thành chữ cái đầu tiên trong tên của bạn. Code hiệu ứng cho nó mà bạn thấy đẹp và lạ nhất! Tìm trên mạng hoặc bạn tự nghĩ!

PHẦN 2: Lập trình LED 7 thanh với STM32F103C8T6.

I, Giới thiệu về LED 7 thanh

Bài trước chúng ta học lập trình giao tiếp để điều khiển Output (LED) theo Pin và theo Port. Ở bài này chúng ta sẽ áp dụng những kiến thức đó để thực hiện lập trình điều khiển hiển thị trên LED 7 thanh theo để hiển thị theo ý mình. Về cơ bản, chúng ta dùng 8 chân đầu ra để điều khiển LED 7 thanh với mục đích hiển thị số theo ý mình.

Sau đây là hình ảnh và chi tiết về 1 con LED 7 thanh và bảng dữ liệu mã hóa với các số từ 0-9 theo kiểu LED sử dụng Anot chung.



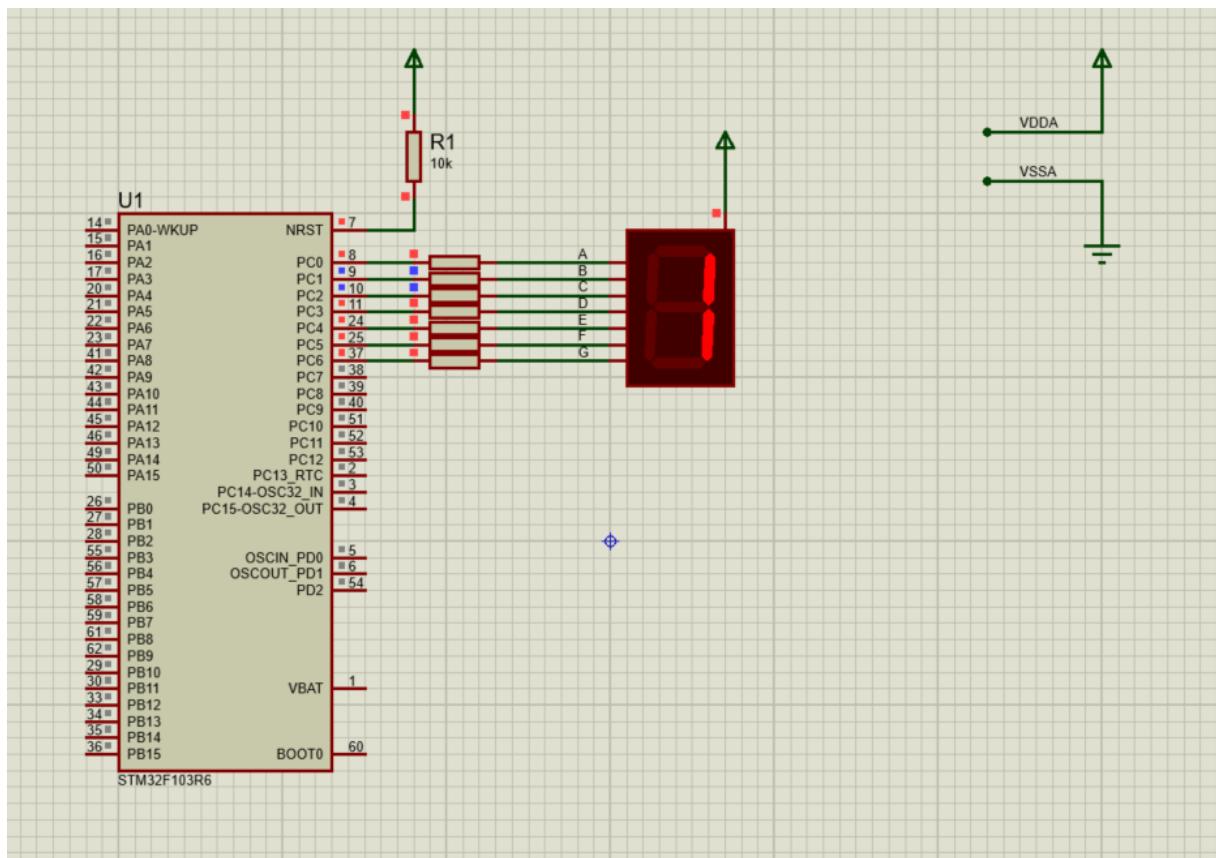
Chữ số	A	B	C	D	E	F	G	Dp	Mã
0	0	0	0	0	0	0	1	1	0xC0
1	1	0	0	1	1	1	1	1	0xF9
2	0	0	1	0	0	1	0	1	0xA4
3	0	0	0	0	1	1	0	1	0xB0
4	1	0	0	1	1	0	0	1	0x99
5	0	1	0	0	1	0	0	1	0x92
6	0	1	0	0	0	0	0	1	0x82
7	0	0	0	1	1	1	1	1	0xF8
8	0	0	0	0	0	0	0	1	0x80
9	0	0	0	0	1	0	0	1	0x90

Các bạn ghi nhớ bảng trên để có thể áp dụng trong các bài toán liên quan đến LED 7 thanh mà không cần lặp lại việc này. Chúng ta sẽ dùng chúng trong việc tạo mảng hiển thị các con số trong các bài thực hành sau.

II, Hướng dẫn thực hành với KIT STM32F103C8T6

Bài 1: Điều khiển 1 LED 7 thanh đếm từ 0-9.

a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

uint16_t LED7SEG[10]={0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,
0x90};

void Delay(uint32_t);
void GPIO_Config(void);
void Clock_Config(void);

int main(void){
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    GPIO_Config();
    while(1)
    {
        for(int i = 0; i < 10; i++){
            GPIO_Write(GPIOC, LED7SEG[i]);
            Delay(100);
        }
    }
}

/*Delay tuong doi*/
void Delay(uint32_t t)
{
    unsigned int i,j;
    for(i=0;i<t;i++){
        for(j=0;j< 0x2AFF; j++);
    }
}
```

```

void GPIO_Config(){

    GPIO_InitTypeDef GPIO_InitStructure;

    /*enable clock for GPIOC*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    /*Configuration GPIO pin*/
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

}

void Clock_Config(void){

    /* RCC system reset */
    RCC_DeInit();

    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);

    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div2);

    /* PCLK1 = HCLK/2 */

    RCC_PCLK1Config(RCC_HCLK_Div2);

    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);

    /* Wait till PLL is ready */

    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}

    /* Select PLL as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

    /* Wait till PLL is used as system clock source */

    while(RCC_GetSYSCLKSource() != 0x00) {}

}

```

Câu hỏi 1: Muốn hiển thị các số chẵn hoặc lẻ thì sao?

Câu hỏi 2: Muốn hiển thị theo thứ tự ngược lại (đếm ngược) thì làm ntn?

Bài 2: Điều khiển 2 LED 7 thanh đếm từ 00–99.

Ở bài toán điều khiển LED 7 thanh chúng ta có thể áp dụng theo bài toán bước 1 nối mỗi con LED 7 thanh vào 8 chân data độc lập. Tuy nhiên việc này sẽ gây lãng phí số chân điều khiển LED và limit số LED có thể điều khiển (4 chiếc). Với số LED tăng lên đủ lớn số chân cần cũng tăng lên rất nhiều. Để giải quyết bài này có một kỹ thuật nêu ra là kỹ thuật “*Quét LED*”.

Kỹ thuật Quét LED thực hiện theo nguyên tắc một thời điểm chỉ bật một LED 7 thanh với dữ liệu nó cần hiển thị, các LED còn lại được tắt. Việc quét LED thực hiện luôn phiền sáng các LED với yêu cầu trên. Có một hiện tượng hay xảy ra với những người mới thực hiện lập trình quét LED là hiện tượng “bóng ma” đó là hiện tượng xuất hiện các bóng mờ LED không mong muốn do quá trình điều khiển. Quá trình quét LED chuẩn được thực hiện theo các bước sau:

Bước 1: Xuất ra mã hiển thị.

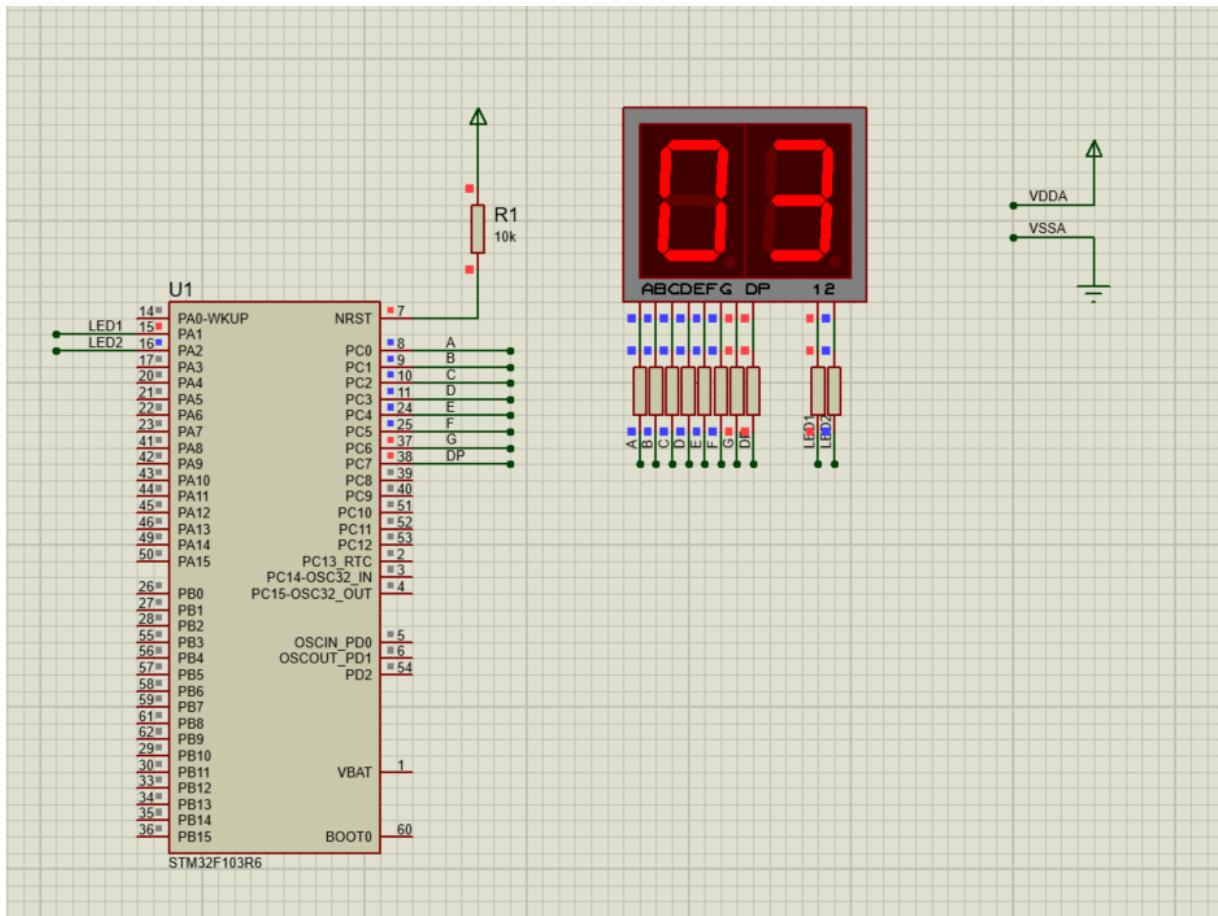
Bước 2: Cấp nguồn cho LED muốn hiển thị.

Bước 3: Trễ 1 khoảng thời gian để duy trì sáng.

Bước 4: Cắt nguồn LED vừa hiển thị.

Các em chú ý thực hiện việc quét LED trong code theo quy trình để tránh gặp những rắc rối không mong muốn. Sau đây là một đoạn chương trình hiển thị các số từ 00-99.

a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

#define LED7SEG_A      GPIO_Pin_0
#define LED7SEG_B      GPIO_Pin_1
#define LED7SEG_C      GPIO_Pin_2
#define LED7SEG_D      GPIO_Pin_3
#define LED7SEG_E      GPIO_Pin_4
#define LED7SEG_F      GPIO_Pin_5
#define LED7SEG_G      GPIO_Pin_6

#define LED7SEG_DP     GPIO_Pin_7
#define LED1           GPIO_Pin_1
#define LED2           GPIO_Pin_2

#define PORT_LED7SEG_CODE GPIOC
#define PORT_LED          GPIOA

#define PORT_LED7SEG_CODE_CLOCK      RCC_APB2Periph_GPIOC
#define PORT_LED_CLOCK              RCC_APB2Periph_GPIOA

uint16_t LED7SEG[10]={0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,
0x90};

void Delay(uint32_t);
void GPIO_Config(void);
void Clock_Config(void);

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    GPIO_Config();

    GPIO_ResetBits(PORT_LED, LED1); //LED1 = 0;
    GPIO_ResetBits(PORT_LED, LED2); //LED2 = 0;

    while(1){
        for(uint8_t i = 0; i < 100; i++){
            for(uint8_t j = 0; j < 24; j++){
                GPIO_Write(PORT_LED7SEG_CODE, LED7SEG[i/10]);
                GPIO_SetBits(PORT_LED, LED1); // LED1 = 1
                Delay(1);

                GPIO_ResetBits(PORT_LED, LED1); //LED1 = 0
                GPIO_Write(PORT_LED7SEG_CODE, LED7SEG[i%10]);
                GPIO_SetBits(PORT_LED, LED2); //LED2 = 1
                Delay(1);

                GPIO_ResetBits(PORT_LED, LED2); //L7S2 = 0
            }
        }
    }
}
```

```

/*Delay tuong doi*/
void Delay(uint32_t t)
{
    unsigned int i,j;

    for(i=0;i<t;i++)
        for(j=0;j< 0x2AFF; j++);
}

void GPIO_Config()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*enble clock for GPIOC*/
    RCC_APB2PeriphClockCmd(PORT_LED7SEG_CODE_CLOCK|PORT_LED_CLOCK, ENABLE);

    /*Configuration GPIO pin*/
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin =
LED7SEG_A|LED7SEG_B|LED7SEG_C|LED7SEG_D|LED7SEG_E|LED7SEG_F|LED7SEG_G|LED7S
EG_A|LED7SEG_DP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(PORT_LED7SEG_CODE, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = LED1|LED2;
    GPIO_Init(PORT_LED, &GPIO_InitStructure);
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div2);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div2);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
    /* Select PLL as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till PLL is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}
}

```

III, Bài tập

Bài tập 1: Hiển thị các số chẵn tăng dần từ 0-99, hiển thị số chẵn, số lẻ...

Bài tập 2: Hiển thị các số lẻ giảm dần từ 99-0, hiển thị số chia hết cho 3...

PHẦN 3: Lập trình nút nhấn

I, Giới thiệu về nút nhấn

Nút nhấn là một loại công tắc đơn giản điều khiển hoạt động của máy hoặc một số loại quá trình. Hầu hết, các nút nhấn là nhựa hoặc kim loại. Hình dạng của nút nhấn có thể phù hợp với ngón tay hoặc bàn tay để sử dụng dễ dàng. Tất cả phụ thuộc vào thiết kế cá nhân. Nút nhấn có 2 loại chính là nút nhấn thường mở hoặc nút nhấn thường đóng.



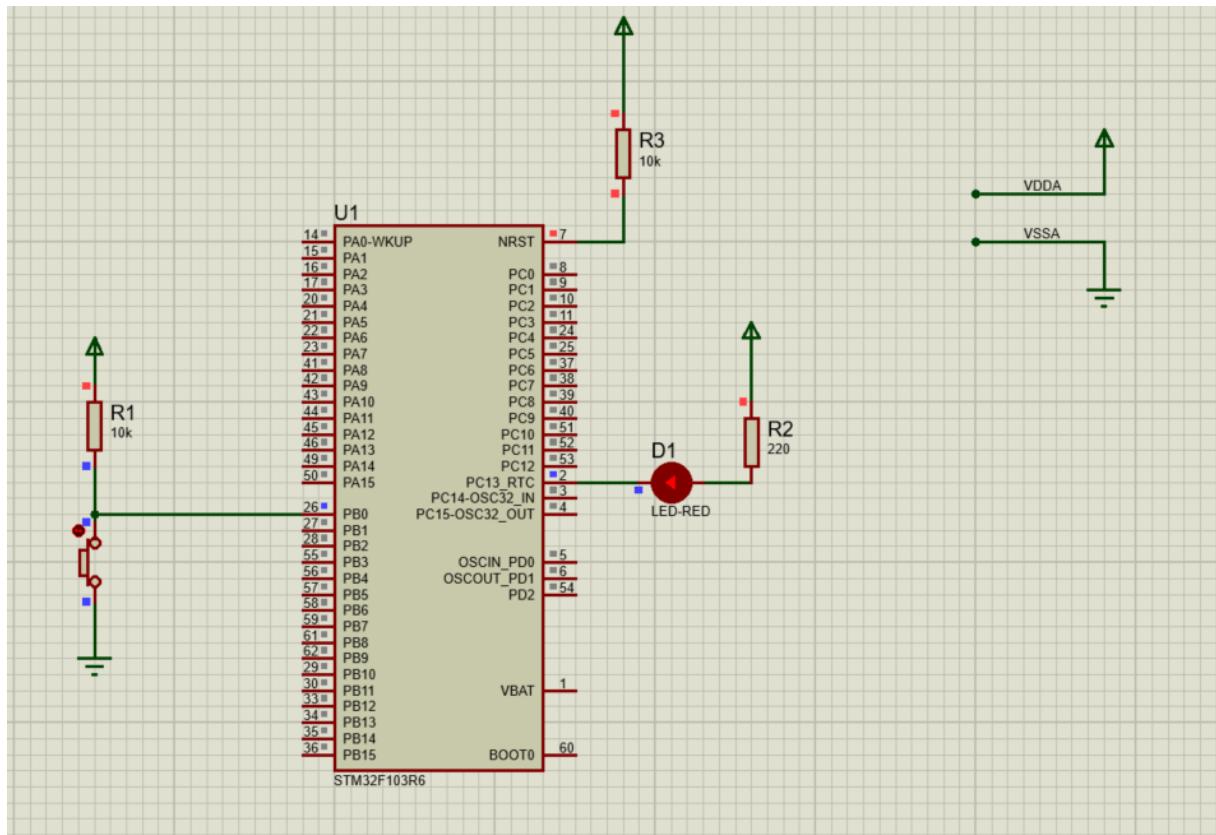
Nguyên lý hoạt động

Nút nhấn có ba phần: Bộ truyền động, các tiếp điểm cố định và các rãnh. Bộ truyền động sẽ đi qua toàn bộ công tắc và vào một xy lanh mỏng ở phía dưới. Bên trong là một tiếp điểm động và lò xo. Khi nhấn nút, nó chạm vào các tiếp điểm tĩnh làm thay đổi trạng thái của tiếp điểm. Trong một số trường hợp, người dùng cần giữ nút hoặc nhấn liên tục để thiết bị hoạt động. Với các nút nhấn khác, chốt sẽ giữ nút bật cho đến khi người dùng nhấn nút lần nữa.

II, Hướng dẫn thực hành với KIT STM32F103C8T6

Bài 1: Đọc trạng thái nút nhấn cơ bản

Trước tiên việc đầu tiên chúng ta cần làm là đọc được dữ liệu đầu vào ở chân vi điều khiển đang ở mức 0 hay 1. Để hiểu rõ, chúng ta cùng thực hiện một bài toán đơn giản như sau: Đọc dữ liệu chân PB0 của vi điều khiển và bật LED tại chân PC13.



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

#define BUTTON           GPIO_Pin_0
#define PORT_BUTTON      GPIOB
#define PORT_BUTTON_CLOCK RCC_APB2Periph_GPIOB

#define LED              GPIO_Pin_13
#define PORT_LED         GPIOC
#define PORT_LED_CLOCK   RCC_APB2Periph_GPIOC

#define LED_ON           GPIO_ResetBits(PORT_LED, LED);
#define LED_OFF          GPIO_SetBits(PORT_LED, LED);

void Delay(uint32_t);
void GPIO_Config(void);
void Clock_Config(void);

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    GPIO_Config();

    while(1) {

        if(GPIO_ReadInputDataBit(PORT_BUTTON, BUTTON)==0) {
            LED_ON
        }
        else{
            LED_OFF
        }
    }
}

/*Delay tuong doi*/
void Delay(uint32_t t){
    unsigned int i,j;

    for(i=0;i<t;i++) {
        for(j=0;j< 0x2AFF; j++);
    }
}

void GPIO_Config(){
    GPIO_InitTypeDef GPIO_InitStructure;

    /*enble clock for GPIOC*/
    RCC_APB2PeriphClockCmd(PORT_BUTTON_CLOCK|PORT_LED_CLOCK, ENABLE);

    /*Configuration GPIO pin*/
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Pin = BUTTON;
    GPIO_Init(PORT_BUTTON, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = LED;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(PORT_LED, &GPIO_InitStructure);
}
```

```
void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div2);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div2);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET) {}
    /* Select PLL as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till PLL is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}
}
```

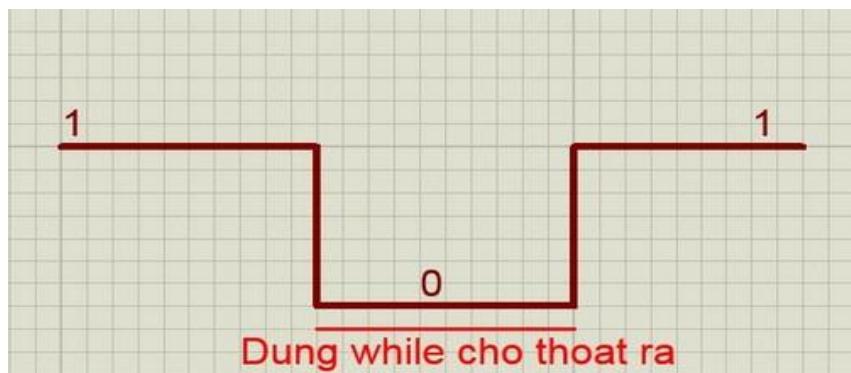
Câu hỏi 1: Khi không nhấn nút thì đèn thé nào? Tại sao?

Câu hỏi 2: Thay 1 đèn bằng 8 đèn để điều khiển sáng tắt đồng thời 8 đèn?

Bài 2: Đọc đầu vào xử lý nút nhấn nâng cao

Ở bài toán tiếp theo chúng ta sẽ nghĩ đến việc thực hiện bài toán nhấn nút thì đèn sáng nhấn thêm lần nữa thì đèn tắt.

Dùng hàm while để chờ chuyển mức.



Ở cách này, sau khi chúng ta phát hiện dữ liệu xuống mức 0, chúng ta sẽ dùng vòng lặp while chờ cho đến khi nào dữ liệu thoát khỏi mức 0 để xác nhận nút đã được nhấn. Lúc đó chúng ta có thể thực hiện các chức năng mà chúng ta mong muốn tương ứng với việc nhấn nút. Đoạn code như sau (Nút bấm nối với PB0, LED nối với PC13):

b, code

```
int main(void)
{
    Clock_Config(); // configuration clock
    SystemCoreClockUpdate(); // update SystemCoreClock variable
    GPIO_Config();

    GPIO_SetBits(PORT_LED, LED);
    while(1){

        if(GPIO_ReadInputDataBit(PORT_BUTTON, BUTTON) == 0)
        {
            while(GPIO_ReadInputDataBit(PORT_BUTTON, BUTTON) == 0);
            GPIO_WriteBit(PORT_LED, LED,
            (BitAction)(1^GPIO_ReadOutputDataBit(PORT_LED, LED)));
        }
    }
}
```

Câu hỏi 1: Lệnh “while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_12)==0) {}” có vai trò gì? Nếu bỏ đi thì chương trình chạy như thế nào ?

Trong thực tế, nút nhấn rất có khả năng bị chập chờn, dao động liên tục mức 0-1 (run tay khi nhấn chẳng hạn). Như thế, một lần nhấn, thả nút có thể khiến vi điều khiển hiểu là rát nhiều nhấn, thả liên tiếp. Để tránh hiện tượng này, khi phát hiện nút nhấn được nhấn chúng ta delay một khoảng thời gian nhỏ để “vượt qua” thời gian dao động thường gấp. Sau đó, chúng ta kiểm tra lại nút nhấn và thực hiện vòng chờ như trên.

```
/* Chuong trinh chinh */
void Delay_ms(uint16_t time){
    uint32_t time_n = time*12000;
    while(time_n != 0){};
}

void main (void)
{
    while(1){
        if(GPIO_ReadInputDataBit(PORT_BUTTON, BUTTON)==0) {
            Delay_ms(2);
            if(GPIO_ReadInputDataBit(PORT_BUTTON, BUTTON) == 0) {
                while(GPIO_ReadInputDataBit(PORT_BUTTON, BUTTON)== 0) {}
                GPIO_WriteBit(PORT_LED,
LED, (BitAction)(1^GPIO_ReadOutputDataBit(PORT_LED, LED)));
            }
        }
    }
}
```

Câu hỏi 2: Giải thích đoạn code trên?

Câu hỏi 3: Thay 1 đèn bằng 4 đèn, mô phỏng tương tự?

III, Bài tập

Bài tập 1: Lập trình để khi nhấn nút nhán thì 8 đèn cùng chuyển trạng thái.

Bài tập 2: Lập trình và mô phỏng bài toán 2 nút nhán sau: Bình thường 8 đèn tắt, khi nhấn 1 nút thì 8 đèn cùng sáng, 8 đèn chuyển sang tắt khi nhấn nút còn lại.

Bài tập 3: Lập trình và mô phỏng bài toán 2 nút nhán sau: Sử 2 nút nhán, đèn Led 7 thanh (1 chữ số). Nhấn 1 nút thì số hiển thị tăng lên, nhấn nút kia thì số giảm đi.

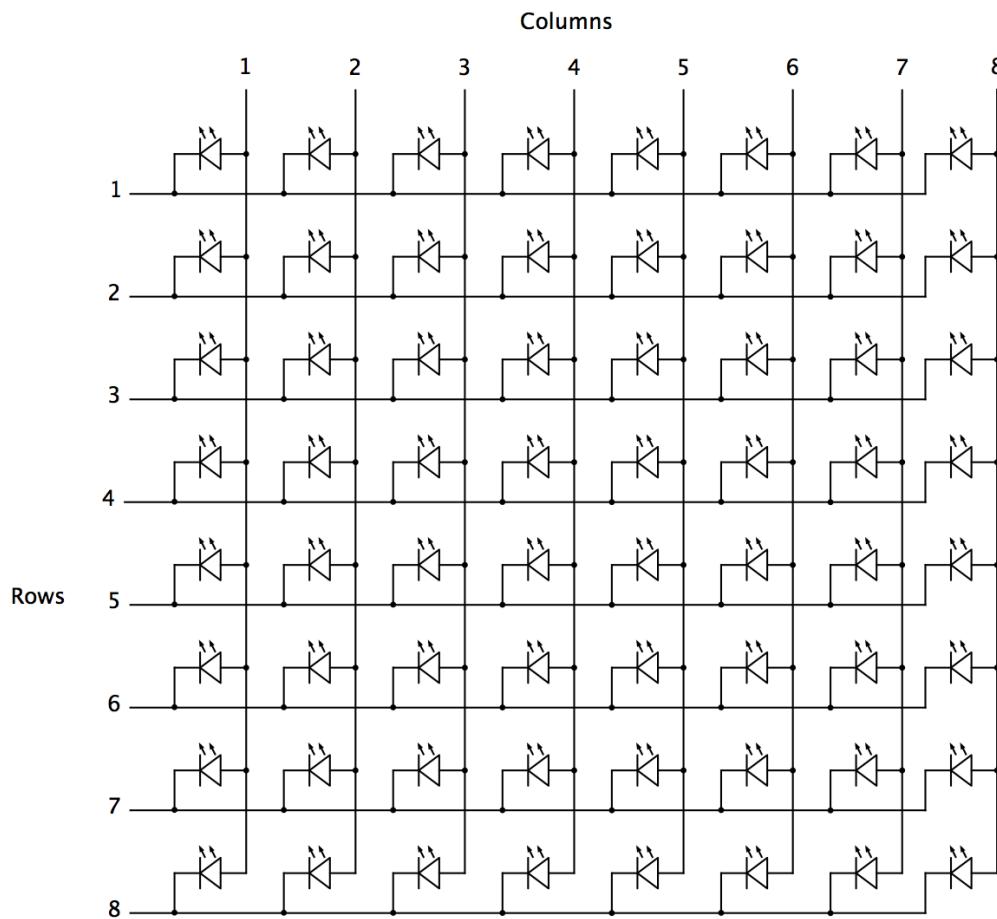
PHẦN 4: Lập trình LED Matrix, LCD1602 với STM32F1

I, LED Matrix

1. Giới thiệu về LED Matrix

Hôm nay chúng ta tiếp tục tìm hiểu về bài toán LED Matrix. Về cơ bản quá trình điều khiển LED Matrix tuân theo nguyên tắc điều khiển LED 7 thanh nhưng dữ liệu hiển thị cho mỗi lần là một cột. Cần hiển thị nhiều lần để có thể hiển thị trên toàn bộ một màn hình LED Matrix

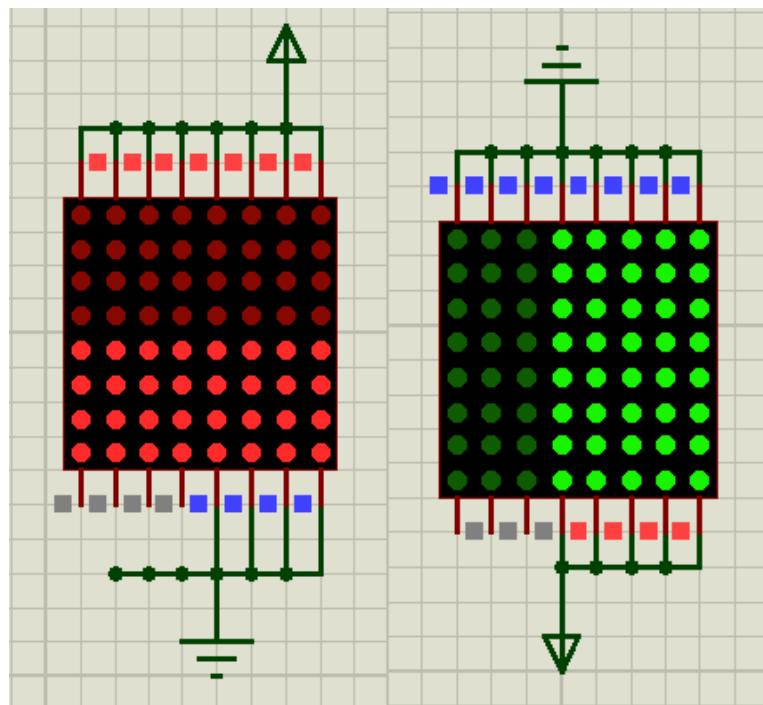
Cấu tạo LED Matrix có dạng như hình dưới đây. Về cấu tạo rất giống với một hệ nhiều LED 7 thanh, ví dụ gồm 8 LED 7 thanh ghép với nhau



Để chuẩn bị cho bài toán về LED matrix các em chuẩn bị một mạch điện mô phỏng đơn giản theo hình sau. Đánh vào từ khóa “Matrix”, kéo xuống chọn loại 8x8 (màu xanh, đỏ tùy các em).

Ở đây PORT2 dùng để điều khiển 8 hàng LED, PORT3 dùng để điều khiển 8 cột LED. LED sẽ sáng nếu thỏa mãn 2 điều kiện là Port3 xuất ra 1 và Port2 xuất ra 0. Port 2 cấp giá trị theo chân, PORT3 được dùng để xuất dữ liệu ra LED. Để đèn sáng thì chân tương ứng ở PORT3 phải có giá trị bằng 0. Chi tiết xem hình dưới.

Hãy chọn Led và thử nghiệm theo hình dưới. **Chú ý, led đỏ hình dưới đã được quay 180 độ so với mặc định để cho giống với hình vẽ đầu tiên!** Nếu để mặc định, Led (xanh) sẽ sáng theo hình dưới (hàng cột đổi chỗ), và khi đó đoạn code sau sẽ không chạy theo ý muốn. Đoạn code và ma trận được xây dựng cho Led đã được quay 180 độ.

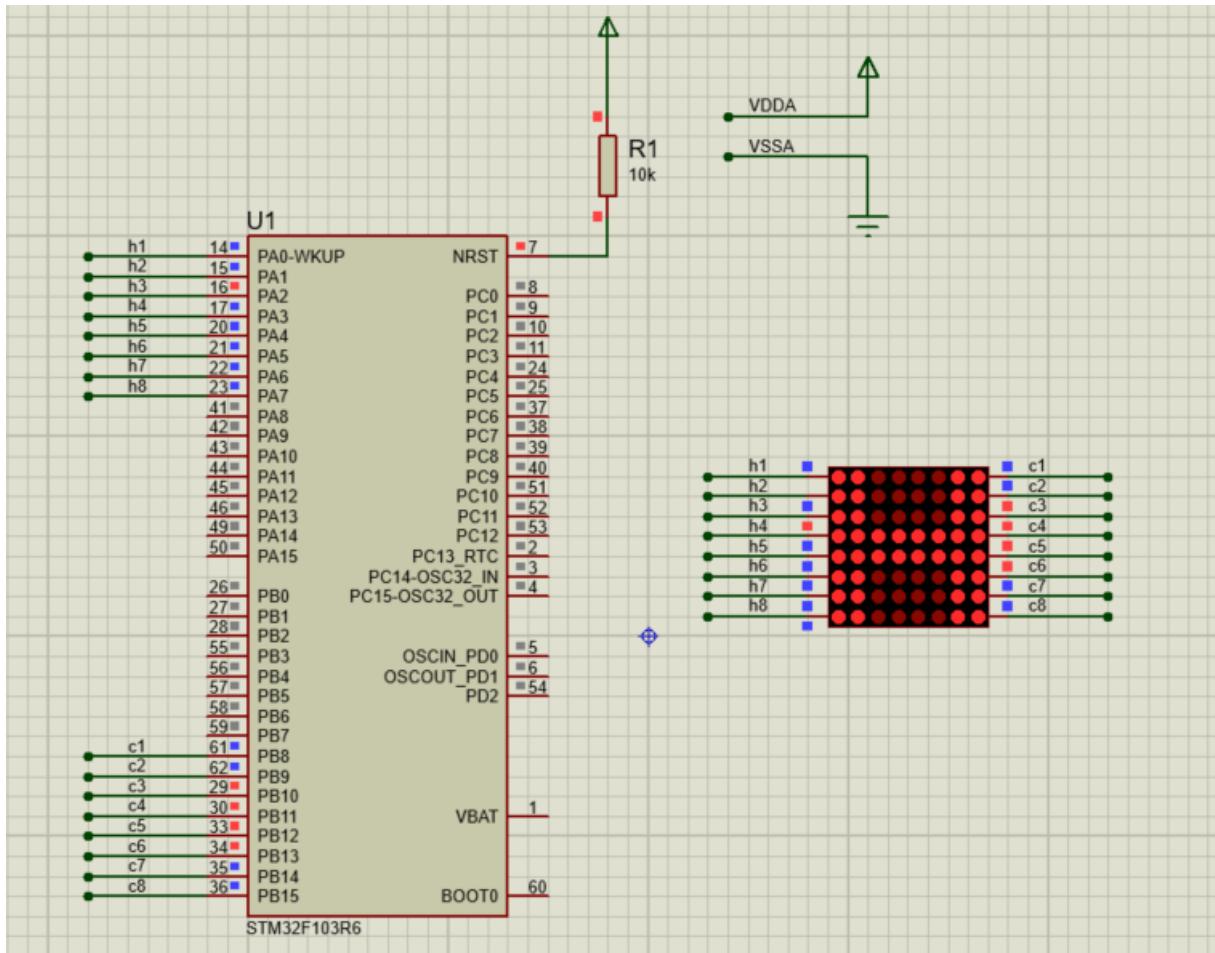


Ở bài toán này chúng ta tập trung tìm cách hiển thị một hình ảnh chúng ta mong muốn lên LED matrix 8×8 này. Việc chúng ta cần là xây dựng bảng dữ liệu cho hình ảnh mà chúng ta muốn hiển thị và lưu lại như lưu lại bảng dữ liệu của LED 7 thanh ở bài trước. Ở đây chúng ta sẽ lập bảng dữ liệu cho chữ cái H. Sau khi có bảng dữ liệu chúng ta thực hiện lập trình xuất mã tương tự LED 7 thanh. Ý tưởng là sử dụng thuật toán quét Led để mỗi vòng lặp, hiển thị led sáng trên một cột.

2, Hướng dẫn thực hành với KIT STM32F103C8T6

Bài 1: Hiển thị chữ cái ‘H’ trên LED matrix 8x8

a, mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

#define RCC_GPIO_ROW          RCC_APB2Periph_GPIOA
#define RCC_GPIO_COL          RCC_APB2Periph_GPIOB

#define GPIO_ROW               GPIOA
#define GPIO_COL               GPIOB

#define GPIO_PIN_ROW_1         GPIO_Pin_0 // khai bao hang 1
#define GPIO_PIN_ROW_2         GPIO_Pin_1 // khai bao hang 2
#define GPIO_PIN_ROW_3         GPIO_Pin_2 // khai bao hang 3
#define GPIO_PIN_ROW_4         GPIO_Pin_3 // khai bao hang 4
#define GPIO_PIN_ROW_5         GPIO_Pin_4 // khai bao hang 5
#define GPIO_PIN_ROW_6         GPIO_Pin_5 // khai bao hang 6
#define GPIO_PIN_ROW_7         GPIO_Pin_6 // khai bao hang 7
#define GPIO_PIN_ROW_8         GPIO_Pin_7 // khai bao hang 8

#define GPIO_PIN_COL_1         GPIO_Pin_8 // khai bao cot 1
#define GPIO_PIN_COL_2         GPIO_Pin_9 // khai bao cot 2
#define GPIO_PIN_COL_3         GPIO_Pin_10 // khai bao cot 3
#define GPIO_PIN_COL_4         GPIO_Pin_11 // khai bao cot 4
#define GPIO_PIN_COL_5         GPIO_Pin_12 // khai bao cot 5
#define GPIO_PIN_COL_6         GPIO_Pin_13 // khai bao cot 6
#define GPIO_PIN_COL_7         GPIO_Pin_14 // khai bao cot 7
#define GPIO_PIN_COL_8         GPIO_Pin_15 // khai bao cot 8

unsigned int
chuH[]={0x3C00,0x3C00,0x3C00,0x0000,0x0000,0x3C00,0x3C00,0x3C00};

void Delay(uint32_t);
void GPIO_Config(void);
void Clock_Config(void);

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    GPIO_Config();

    while(1){
        for(uint8_t i = 0; i < 8; i++){
            GPIO_Write(GPIO_ROW, 0x01 << i);
            GPIO_Write(GPIO_COL, chuH[i]);
            Delay(1);
        }
    }
}

/*Delay tuong doi*/
```

```

void Delay(uint32_t t)
{
    unsigned int i,j;

    for(i=0;i<t;i++){
        for(j=0;j< 0x2AFF/4; j++);
    }
}

void GPIO_Config()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*enable clock for GPIOC*/
    RCC_APB2PeriphClockCmd(RCC_GPIO_ROW|RCC_GPIO_COL, ENABLE);

    /*Configuration GPIO pin*/
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin =
    GPIO_PIN_ROW_1|GPIO_PIN_ROW_2|GPIO_PIN_ROW_3|GPIO_PIN_ROW_4|GPIO_PIN_ROW_5|
    GPIO_PIN_ROW_6|GPIO_PIN_ROW_7|GPIO_PIN_ROW_8;;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIO_ROW, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin =
    GPIO_PIN_COL_1|GPIO_PIN_COL_2|GPIO_PIN_COL_3|GPIO_PIN_COL_4|GPIO_PIN_COL_5|
    GPIO_PIN_COL_6|GPIO_PIN_COL_7|GPIO_PIN_COL_8;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIO_COL, &GPIO_InitStructure);
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div2);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div2);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
    /* Select PLL as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till PLL is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}
}

```

Còn lại chương trình tuân theo các bước của quy trình “*Quét LED*” đã nhắc ở bài thực hành số 3 trước.

Sau khi hoàn thiện chương trình các bạn thực hiện Build, các bạn đưa file hex vào mạch điện chúng ta đã có và chạy mô phỏng. Các bạn sẽ thấy có kết quả như sau:

Trên đây chúng ta đã thực hiện điều khiển được 01 LED martrix. Về phần điều khiển nhiều LED matrix cũng tương tự như thế nhưng với điều kiện khi số LED martrix tăng lên yêu cầu số chân tăng lên rất nhanh. Chúng ta có thể sử dụng một số IC mở rộng như 74HC595 để thực hiện mở rộng để có thể điều khiển được nhiều LED matrix hơn.

3, Bài tập

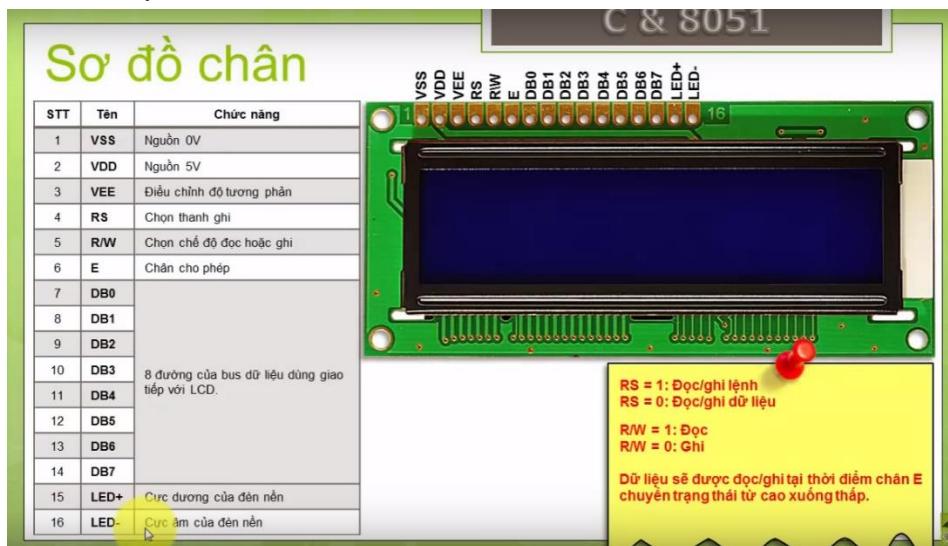
Bài tập1: Hiển thị lần lượt các chữ cái trong tên của bạn! Khi trình bày theo nhóm thì hiển thị lần lượt các kí tự “NHOM MOT, NHOM HAI...”).

Good luck and Have fun!

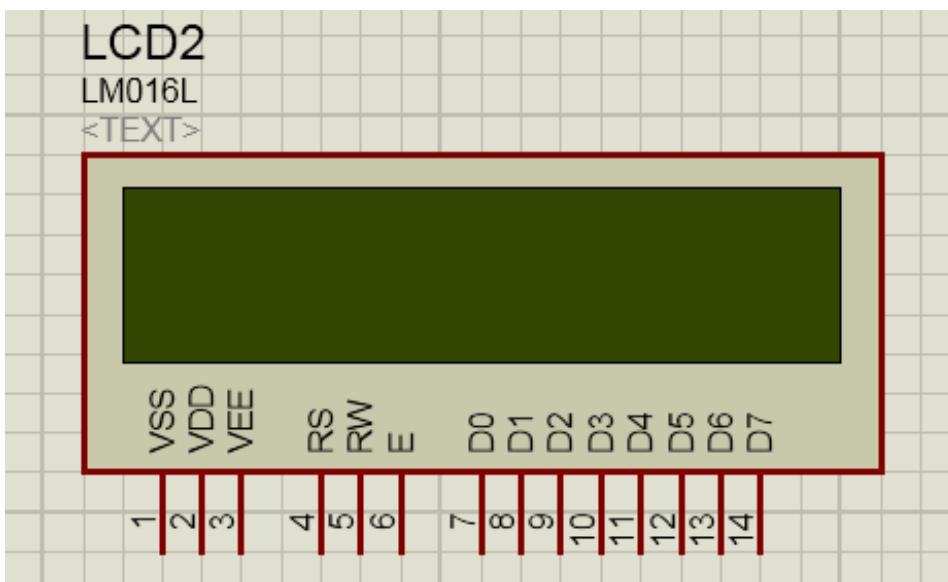
II, LCD1602

1, Giới thiệu về LCD1602

Chúng ta đến với một phần rất quan trọng: Hiển thị thông tin ra LCD. LCD là một màn hình nhiều điểm ảnh, có thể coi là một Led ma trận dạng lớn, tuy nhiên chúng ta không cần đi vào điều khiển từng Pixel hay là từng chấm nhỏ như trong Phần 1 mà chúng ta sẽ điều khiển qua lệnh, con trỏ... để hiển thị thông tin một cách dễ dàng hơn. Có nhiều loại LCD, trong bài này chúng ta dùng loại đơn giản 16x2. Trước tiên chúng ta tìm hiểu về cấu tạo của nó.



Dưới đây là hình mô phỏng, các em tìm nó với từ khóa “lcd 16x2”.



Tiếp theo chúng ta học về cách kết nối, lập trình để sử dụng LCD. Trước tiên chúng ta phải gửi lệnh vào cho LCD biết nó phải làm gì. Cách gửi lệnh như sau:

Gửi lệnh ra LCD

Để gửi một lệnh ra LCD, các bạn phải làm theo trình tự sau:

- B1.** Kéo chân RW xuống mức thấp để chọn chế độ là “ghi”.
- B2.** Kéo chân RS xuống mức thấp để cho LCD hiểu là chúng ta muốn ghi lệnh chứ không phải ghi dữ liệu.
- B3.** Gửi byte lệnh ra các chân D7...D0
- B4.** Tạo xung trên chân E bằng cách cho E xuống mức 0 rồi lên lại mức 1 hoặc ngược lại để cho phép lệnh được ghi vào LCD.
- B5.** Delay 1 khoảng thời gian để LCD **thực hiện** xong lệnh.

Các em có thể xem một số lệnh sau đây. Những lệnh này gọi là lệnh khởi tạo LCD.

Tập lệnh của LCD

Chú ý: Ở đây Dâng chỉ liệt kê các lệnh thường dùng, chi tiết hơn về tập lệnh các bạn xem thêm trong tài liệu tham khảo.

Mã lệnh	Chức năng	T _{exe}
0x01	Xoá toàn bộ nội dung đang hiển thị trên màn hình.	1.52ms
0x02	Di chuyển con trỏ về vị trí đầu màn hình.	1.52ms
0x06	Tự động di chuyển con trỏ đến vị trí tiếp theo mỗi khi xuất ra LCD 1 ký tự.	37us
0x0C	Bật hiển thị và tắt con trỏ	37us
0x0E	Bật hiển thị và bật con trỏ	37us
0x80	Di chuyển con trỏ về đầu dòng 1	37us
0xC0	Di chuyển con trỏ về đầu dòng 2	37us
0x38	Giao tiếp 8 bit, hiển thị 2 dòng, kích thước font 5x7	37us
0x28	Giao tiếp 4 bit, hiển thị 2 dòng, kích thước font 5x7	37us

Sau khi gửi các lệnh vào LCD (qua các chân D0 – D7) để nó biết sẽ phải làm gì (bật màn hình, hiển thị hay tắt con trỏ, di chuyển con trỏ ...), chúng ta sẽ phải gửi lệnh chứa các kí tự mà chúng ta muốn hiển thị. Cách gửi dữ liệu ra cũng tương tự như cách gửi lệnh, các em có thể xem trong hình sau:

Gửi dữ liệu ra LCD

Để gửi ký tự ra LCD, các bạn phải làm theo trình tự sau:

- B1. Kéo chân RW xuống mức thấp để chọn chế độ là “ghi”.
- B2. Kéo chân RS lên mức cao để cho LCD hiểu là chúng ta muốn ghi dữ liệu chứ không phải ghi lệnh.
- B3. Gửi mã của ký tự (ASCII) ra các chân D7...D0.
- B4. Tạo xung trên chân E bằng cách cho E xuống mức 0 rồi lên lại mức 1 hoặc ngược lại để cho dữ liệu được ghi vào LCD.
- B5. Delay 1 khoảng thời gian (37us) để LCD hiển thị ký tự

Lệnh Set DDRAM address

Lệnh này dùng để di chuyển con trỏ đến vị trí bất kỳ trong vùng nhớ DDRAM. Do đó, chúng ta có thể hiển thị ký tự hoặc chuỗi tại một vị trí bất kỳ.

Mã lệnh: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
1 [AD] [AD] [AD] [AD] [AD] [AD] [AD]

Ví dụ: Chúng ta muốn xuất ký tự A ra LCD tại vị trí dòng 1, cột 5. Như vậy, chúng ta cần di chuyển con trỏ đến ô nhớ có địa chỉ là 0x04.

Mã lệnh: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
1 0 0 0 0 1 0 0 0x84

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
				A												
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50

Lcd_Cmd(0x84);
Lcd_Chr_Cp('A');

Sau khi hiển thị một ký tự tại một vị trí có sẵn, chúng ta có thể tiến đến bước xa hơn, dịch chuyển ký tự mà chúng ta muốn (dịch trái, dịch phải) với câu lệnh như hình dưới.

Lệnh dịch nội dung hiển thị

- Lệnh 0x18: Dịch toàn bộ nội dung đang hiển thị sang trái
- Lệnh 0x1C: Dịch toàn bộ nội dung đang hiển thị sang phải

01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	...	27
O	I		Y	E	U		N	U	O	C		V	I	E	T		N	A	M	...	
																				...	

Cuối cùng, khi dòng chữ đang chạy, các em muốn dừng không cho chạy nữa, trở lại vị trí như ban đầu, có thể dùng lệnh sau:

Lệnh return home

- **Mã lệnh:** 0x02 hoặc 0x03

- **Chức năng:**

Trả lại kiểu hiển thị gốc nếu nó bị thay đổi.

Nội dung của DDRAM không thay đổi.

Con trỏ trở về vị trí đầu dòng đầu tiên.

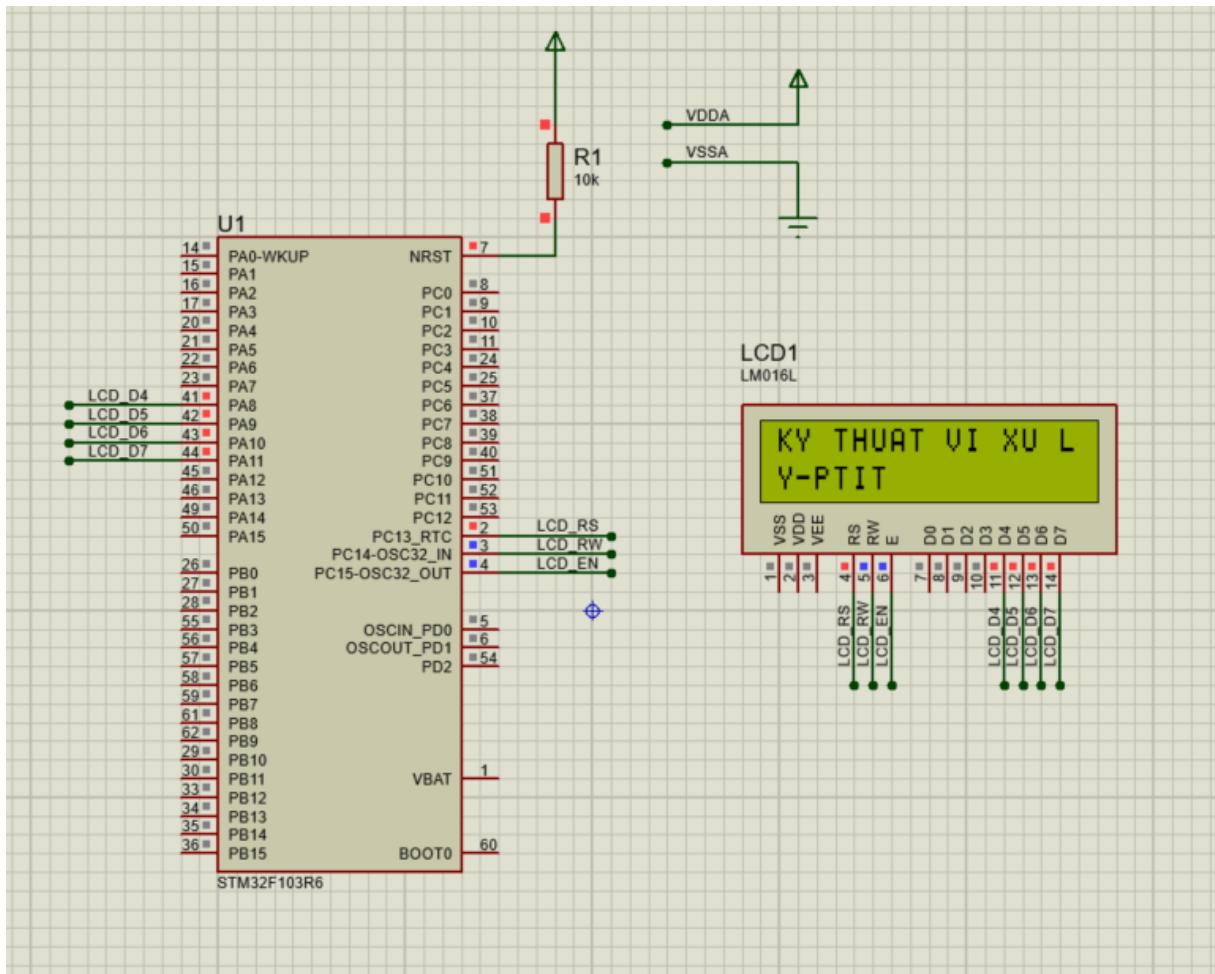


00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	...	27
T	O	I		Y	E	U		N	U	O	C		V	I	E	T		N	A	M	...	
																					...	

2. Hướng dẫn thực hành với KIT STM32F103C8T6

Bài 1: Gửi kí tự lên màn hình LCD1602

a, Mô phỏng trên proteus



Sơ đồ nối chân:

- LCD_RS PC13
- LCD_RW PC14
- LCD_EN PC15
- LCD_D4 PA8
- LCD_D5 PA9
- LCD_D6 PA10
- LCD_D7 PA11

***Lưu ý: Các em add thư viện cần thiết vào project như delay.h, lcd.h b, code**

File main.c:

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "lcd16x2.h"
#include "delay.h"

int main(void)
{
    // Delay initialization
    DelayInit();
    // LCD initialization
    LCD_Init();

    while(1) {
        LCD_Gotoxy(0,0);
        LCD_Puts("KY THUAT VI XU");
        LCD_Gotoxy(0,1);
        LCD_Puts("LY - PTIT");
        DelayMs(500);
    }
}
```

File delay.c

```
#include "delay.h"

// For store tick counts in us
static __IO uint32_t usTicks;

// SysTick_Handler function will be called every 1 us
void SysTick_Handler()
{
    if (usTicks != 0)
    {
        usTicks--;
    }
}

void DelayInit()
{
    // Update SystemCoreClock value
    SystemCoreClockUpdate();
    // Configure the SysTick timer to overflow every 1 us
    SysTick_Config(SystemCoreClock / 1000000);
}

void DelayUs(uint32_t us)
{
    // Reload us value
    usTicks = us;
    // Wait until usTick reach zero
    while (usTicks);
}

void DelayMs(uint32_t ms)
{
    // Wait until ms reach zero
    while (ms--)
    {
        // Delay 1ms
        DelayUs(1000);
    }
}
```

File delay.h

```
#ifndef __DELAY_H
#define __DELAY_H

#ifndef __cplusplus
extern "C" {
#endif

#include "stm32f10x.h"

void DelayInit(void);
void DelayUs(uint32_t us);
void DelayMs(uint32_t ms);

#endif /* __cplusplus */

}

#endif

#endif
```

File LCD.c

```
#include "lcd16x2.h"

#define LCD_RS      GPIO_Pin_13
#define LCD_RW      GPIO_Pin_14
#define LCD_EN      GPIO_Pin_15

#define LCD_D4      GPIO_Pin_8
#define LCD_D5      GPIO_Pin_9
#define LCD_D6      GPIO_Pin_10
#define LCD_D7      GPIO_Pin_11

void GPIO_LCD_Config(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOC,
                           ENABLE);
    GPIO_InitTypeDef      GPIO_LCD_InitStruct;
    /*Configure GPIO pin Output Level */
    GPIO_WriteBit(GPIOA, LCD_D4|LCD_D5|LCD_D5|LCD_D7, 0);
    GPIO_WriteBit(GPIOC, LCD_RS|LCD_RW|LCD_EN, 0);

    GPIO_LCD_InitStruct.GPIO_Mode     = GPIO_Mode_Out_PP;
    GPIO_LCD_InitStruct.GPIO_Pin     = LCD_D4|LCD_D5|LCD_D6|LCD_D7;
    GPIO_LCD_InitStruct.GPIO_Speed   = GPIO_Speed_10MHz;
    GPIO_Init(GPIOA, &GPIO_LCD_InitStruct);

    GPIO_LCD_InitStruct.GPIO_Pin = LCD_EN|LCD_RS|LCD_RW;
    GPIO_Init(GPIOC, &GPIO_LCD_InitStruct);
}

void LCD_Enable(void)
{
    GPIO_SetBits(GPIOC, LCD_EN);
    DelayMs(1);
    GPIO_ResetBits(GPIOC, LCD_EN);
    DelayMs(1);
}

void LCD_Send4Bit(unsigned char Data)
{
    GPIO_WriteBit(GPIOA, LCD_D4, Data & 0x01);
    GPIO_WriteBit(GPIOA, LCD_D5, (Data>>1)&1);
    GPIO_WriteBit(GPIOA, LCD_D6, (Data>>2)&1);
    GPIO_WriteBit(GPIOA, LCD_D7, (Data>>3)&1);
}

void LCD_SendCommand(unsigned char command)
{
    LCD_Send4Bit(command >> 4);
    LCD_Enable();
    LCD_Send4Bit(command);
    LCD_Enable();
}

void LCD_Clear()
{
    LCD_SendCommand(0x01);
    DelayUs(10);
```

```

}

void LCD_Init()
{
    GPIO_LCD_Config();

    LCD_Send4Bit(0x00);
    GPIO_WriteBit(GPIOC, LCD_RS, 0);
    LCD_Send4Bit(0x03);
    LCD_Enable();
    LCD_Enable();
    LCD_Enable();
    LCD_Send4Bit(0x02);
    LCD_Enable();
    LCD_SendCommand(0x28); // giao thuc 4 bit, hien thi 2 hang, ki tu 5x8
    LCD_SendCommand(0x0C); // cho phep hien thi man hinh
    LCD_SendCommand(0x06); // tang ID, khong dich khung hinh
    LCD_SendCommand(0x01); // xoa toan bo khung hinh
}

void LCD_Gotoxy(unsigned char x, unsigned char y)
{
    unsigned char address;
    if(y == 0) address=(0x80+x);
    else if(y == 1) address=(0xc0+x);
    LCD_SendCommand(address);
}

void LCD_PutChar(unsigned char Data)
{
    GPIO_SetBits(GPIOC, LCD_RS);
    LCD_SendCommand(Data);
    GPIO_ResetBits(GPIOC, LCD_RS);
}

void LCD_Puts(char *s)
{
    while (*s)
    {
        LCD_PutChar(*s);
        s++;
    }
}

```

File LCD.h

```
#ifndef __LCD_H
#define __LCD_H

#ifndef __cplusplus
extern "C" {
#endif

/* Includes -----
---*/
#include "stm32f10x.h"
#include "delay.h"

void GPIO_LCD_Config(void);
void LCD_Enable(void);
void LCD_Send4Bit(unsigned char Data);
void LCD_SendCommand(unsigned char command);
void LCD_Clear();
void LCD_Init();
void LCD_Gotoxy(unsigned char x, unsigned char y);
void LCD_PutChar(unsigned char Data);
void LCD_Puts(char *s);

#endif /* __cplusplus
}

#endif /* __MISC_H */
```

Add hết các file cần thiết vào và chạy chương trình

III, Bài tập

Bài tập 1: Hiển thị 2 dòng chữ - Số thứ tự nhóm và Họ và tên một bạn trong nhóm.

Bài tập 2: Hiển thị Tên 2 bạn ở vị trí chính giữa màn hình, dòng 1 và dòng 2.

Bài tập 3: Dịch phải và dịch trái một đoạn text dài hơn 16 kí tự.

PHẦN 5: UART

I. Giới thiệu UART

USART là giao tiếp đơn giản, nhưng cực kì quan trọng và phổ biến. Hầu hết vi điều khiển đều hỗ trợ giao tiếp USART không thẻ không nói đến Arduino với cộng đồng cực kì lớn. Ngoài ra rất nhiều module giao tiếp với vi điều khiển qua USART như module SIM, bluetooth HC05, ESP8266,..

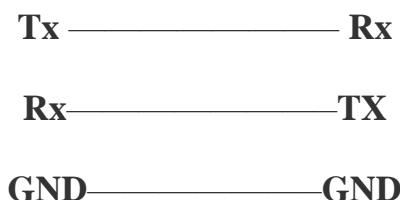
STM32F10x hỗ trợ 3 cổng USART với chân map và remap như sau:

Alternate functions	USART2_REMAP = 0	USART2_REMAP = 1 ⁽¹⁾
USART2_CTS	PA0	PD3
USART2_RTS	PA1	PD4
USART2_TX	PA2	PD5
USART2_RX	PA3	PD6
USART2_CK	PA4	PD7

Alternate function	USART3_REMAP[1:0] = "00" (no remap)	USART3_REMAP[1:0] = "01" (partial remap) ⁽¹⁾	USART3_REMAP[1:0] = "11" (full remap) ⁽²⁾
USART3_TX	PB10	PC10	PD8
USART3_RX	PB11	PC11	PD9
USART3_CK	PB12	PC12	PD10
USART3_CTS	PB13		PD11
USART3_RTS	PB14		PD12

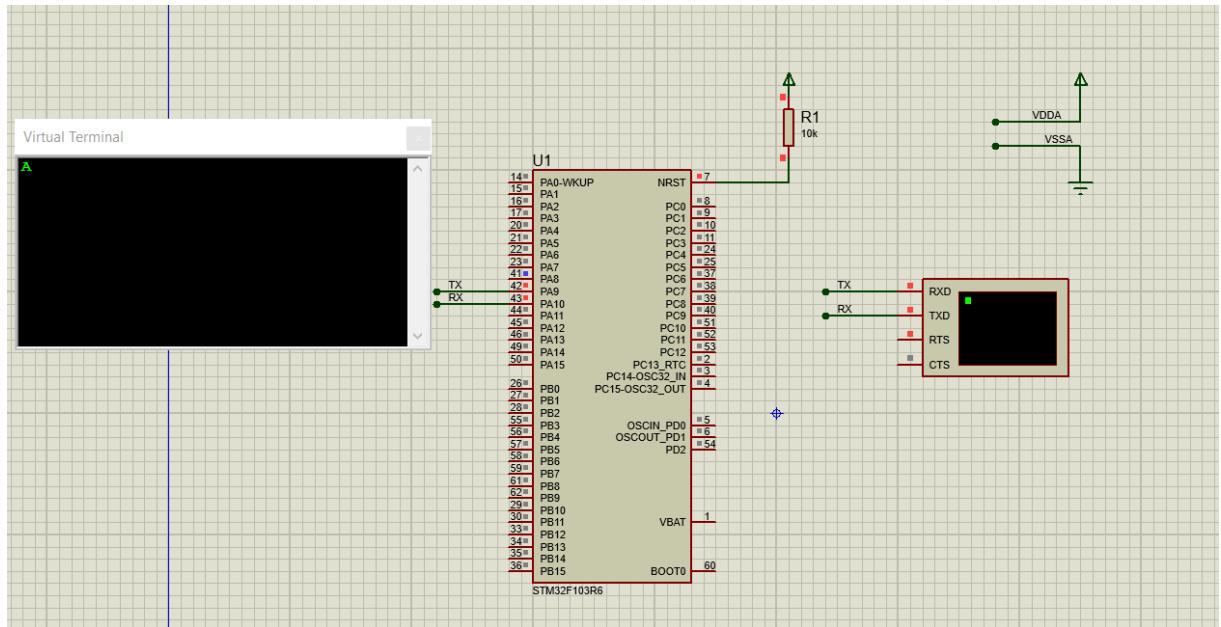
Khi kết nối lên máy tính mình hay dùng module USB-USART PL2303 hoặc CP1202

Kết nối vi điều khiển với module USART



II, Bài tập thực hành

Bài 2.1: Viết chương trình gửi ký tự “A” lên máy tính với tốc độ baud là 9600
a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"

void Clock_Config(void);
void UART_Configuration(void);
void USART_SendChar(USART_TypeDef *USART, char data);
void USART_SendString(USART_TypeDef *USART, char *Str);
void Delay_t(uint32_t t);

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    UART_Configuration();

    while(1)
    {
        USART_SendChar(USART1, 'A');
        Delay_t(100);
    }
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div1);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
    /* Select HSI as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till HSI is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}

    /*Delay tuong doi*/
    void Delay_t(uint32_t t)
    {
        unsigned int i,j;

        for(i=0;i<t;i++){
            for(j=0;j< 0x2AFF; j++);
        }
    }
}
```

```

void UART_SendChar(USART_TypeDef *USARTx, char data){

    USARTx->DR = 0x00;
    USART_SendData(USARTx, data);
    //TxE = 1: Data is transferred to the shift register
    //TxE = 0; Data is not transferred to the shift register
    while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
}

void UART_SendString(USART_TypeDef *USARTx, char *Str)
{
    while(*Str){
        UART_SendChar(USARTx, *Str);
        Str++;
    }
}

void UART_configuration(void)
{
    GPIO_InitTypeDef      GPIO_InitStructure;
    USART_InitTypeDef     USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE ); /*enable clock*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE );
    /*
        USART1_Rx : PA10  input floating
        USART1_Tx : PA9   alternate function push-pull
    */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure );

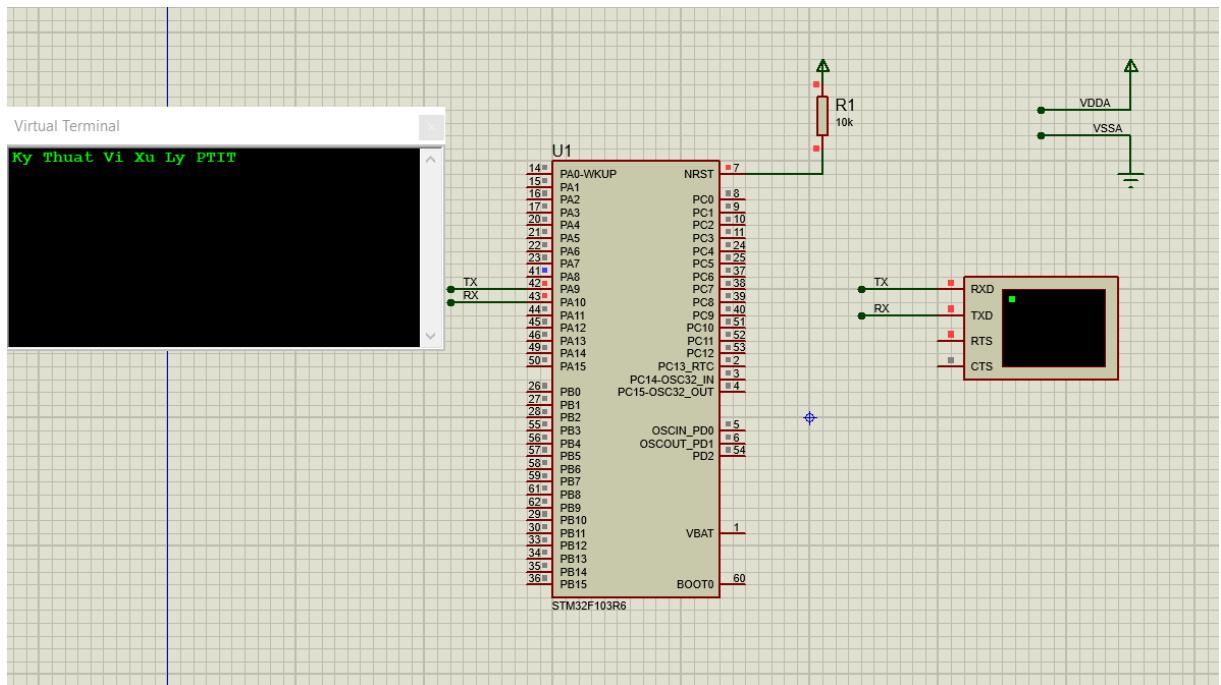
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure );

    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
    USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE); //kick hoat usart1
}

```

**Bài 2.2: Gửi chuỗi “Ky Thuat Vi Xu Ly PTIT” lên máy tính sử dụng hàm printf
a, Mô phỏng trên proteus**



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stdio.h"

void Clock_Config(void);
void USART_configuration(void);
void USART_SendChar(USART_TypeDef *USART, char data);
void USART_SendString(USART_TypeDef *USART, char *Str);
void Delay_t(uint32_t t);

struct __FILE {
    int dummy;
};

FILE __stdout;

int fputc(int ch, FILE *f){
    /* Send your custom byte */
    USART_SendData(USART1, ch);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET){};
    /* If everything is OK, you have to return character written */
    return ch;
}

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale
    USART_configuration();

    while(1)
    {
        printf("Ky Thuat Vi Xu Ly PTIT");
        Delay_t(100);
    }
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div1);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
    /* Select HSI as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till HSI is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}
}
```

```

/*Delay tuong doi*/
void Delay_t(uint32_t t)
{
    unsigned int i,j;

    for(i=0;i<t;i++) {
        for(j=0;j< 0x2AFF; j++);
    }
}

void UART_SendChar(USART_TypeDef *USARTx, char data){

    USARTx->DR = 0x00;
    USART_SendData(USARTx, data);
    //TxE = 1: Data is transferred to the shift register
    //TxE = 0; Data is not transferred to the shift register
    while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
}

void UART_SendString(USART_TypeDef *USARTx,char *Str)
{
    while(*Str){
        UART_SendChar(USARTx, *Str);
        Str++;
    }
}

void UART_configuration(void)
{
    GPIO_InitTypeDef      GPIO_InitStructure;
    USART_InitTypeDef     USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE ); /*enable clock*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE );
    /*
        USART1_Rx : PA10  input floating
        USART1_Tx : PA9   alternate function push-pull
    */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure );

    USART_InitStructureUSART_BaudRate = 9600;
    USART_InitStructureUSART_WordLength = USART_WordLength_8b;
    USART_InitStructureUSART_StopBits = USART_StopBits_1;
    USART_InitStructureUSART_Parity = USART_Parity_No;
    USART_InitStructureUSART_HardwareFlowControl =
    USART_HardwareFlowControl_None;
    USART_InitStructureUSART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE); //kich hoat usart1
}

```

II, Bài tập

Bài tập 1: Viết chương trình vi điều khiển nhận được ký tự nào từ máy tính thì gửi lại ký tự đó lên máy tính đó với tốc độ 115200.

PHẦN 6: TIMER

I, Giới thiệu về timer

STM32f103C8 có tất cả 7 timer nhưng trong đó đã bao gồm 1 systick timer, 2 watchdog timer. Vậy chỉ còn lại 4 timer dùng cho các chức năng như ngắt, timer base, PWM, Encoder, Input capture.... Trong đó TIM1 là Timer đặc biệt, chuyên dụng cho việc xuất xung với các mode xuất xung, các mode bảo vệ đầy đủ hơn so với các timer khác. TIM1 thuộc khối clock APB2, còn các TIM2,TIM3,TIM4 thuộc nhóm APB1.

Có 3 vấn đề cần phải tìm hiểu trong bài này đó là :

- Timer clock.
- Prescaler
- Auto Reload Value.

Khi không có cấu hình gì liên quan đến clock và đã gắn đúng thạch anh ngoài trên chân PD0(5) và PD1(6) thì clock tương ứng của TIM1,TIM2,TIM3,TIM4 đã là 72Mhz. Cần ghi nhớ là sử dụng timer nào thì cấp clock cho timer đó theo đúng nhánh clock.

Prescaler là bộ chia tần số của timer. Bộ chia này có giá trị tối đa là 16 bit tương ứng với giá trị là 65535. Các giá trị này có thể được thay đổi và điều chỉnh bằng lập trình. Tần số sau bộ chia này sẽ được tính là:

$$f_{CK_CNT} = f_{CK_PSC}/(PSC+1).$$

- FCK_CNT: tần số sau bộ chia.
- fCK_PSC: tần số clock đầu vào cấp cho timer.
- PSC: chính là giá trị truyền vào được lập trình bằng phần mềm

Auto Reload value là giá trị bộ đếm tối đa có thể được điều chỉnh để nạp vào cho timer. Giá trị bộ đếm này được cài đặt tối đa là 16bit tương ứng với giá trị là 65535.Từ các thông số trên ta rút ra công thức cần tính cuối cùng đó là:

FTIMER= fSYSTEM/[(PSC+1)(Period+1)]

- Ftimer : là giá trị cuối cùng của bài toán, đơn vị là hz.
- F system : tần số clock hệ thống được chia cho timer sử dụng, đơn vị là hz.
- PSC : giá trị nạp vào cho bộ chia tần số của timer. Tối đa là 65535.
- Period : giá trị bộ đếm nạp vào cho timer. Tối đa là 65535.

Ngắt timer: khi giá trị đếm của bộ đếm timer(thanh ghi CNT) vượt qua giá trị của Auto Reload Value thì cờ báo tràn sẽ được kích hoạt. Trình phục vụ ngắt tràn sẽ xảy ra nếu được cấu hình cho phép trước đó.

II, Bài tập thực hành

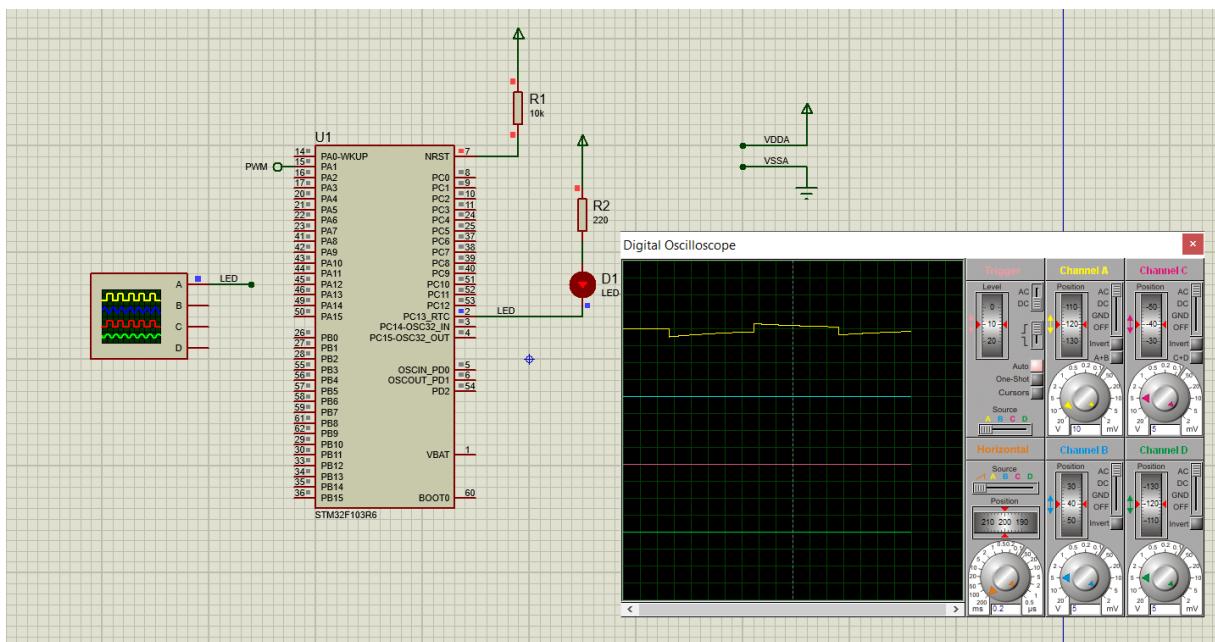
Bài 1: Cấu hình ngắt tràn TIM4. Led trên chân PB8 sáng tắt với tốc độ là 1s sáng 1s tắt.

*Hướng dẫn: Giả sử

APB1 = 8Mhz

- Được nhân 1 thành 8Mhz rồi cấp cho TIM3
 - Timer phải đếm 8 000 000 lần thì được 1s
 - Timer phải đếm 4 000 000 lần thì được 0.5s
 - Giá trị đếm tối đa chỉ được 65635 -> phải sử dụng bộ chia trước.
 - Chọn bộ chia trước là 1000
- > Timer phải đếm $8\ 000\ 000 / 1000 = 8000$ xung thì được 1s
- > Timer phải đếm 8000 xung thì được 1s
- ➔ Chọn giá trị tự động nạp lại là $8000-1=7999$

a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"

#define LED          GPIO_Pin_13
#define PORT_LED    GPIOC
#define RCC_LED      RCC_APB2Periph_GPIOC

void Clock_Config(void);
void GPIO_configuration(void);
void TIM3_Configuration(void);
void NVIC_Configuration(void);

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale

    GPIO_Configuration();
    TIM3_Configuration();
    NVIC_Configuration();

    GPIO_SetBits(PORT_LED, LED);
    while(1)
    {

    }
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div1);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET){}
    /* Select HSI as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till HSI is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}

}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_LED, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = LED;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(PORT_LED, &GPIO_InitStructure);
}
```

```

void TIM3_Configuration(void)
{
    TIM_TimeBaseInitTypeDef TIM_InitStructure;

    /*
        Luu y: voi phien ban proteus 8.6, chung ta phai enable TIM1 thi moi
        co the dung TIMx.
        => bug cua phan men proteus.
    */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
    TIM_TimeBaseStructInit(&TIM_InitStructure);
    TIM_Cmd(TIM1, ENABLE);

    /*configuration TIM3*/
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    TIM_InitStructure.TIM_Period = 7999;
    TIM_InitStructure.TIM_Prescaler = 999;
    TIM_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_InitStructure.TIM_RepetitionCounter = 0x00;
    TIM_TimeBaseInit(TIM3, &TIM_InitStructure);

    TIM_TimeBaseInit(TIM3, &TIM_InitStructure);
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); //enable update interrupt
    TIM_Cmd(TIM3, ENABLE);
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    NVIC_InitStructure.NVIC IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void TIM3_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) //if update flag turn
    on
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update); //clear update flag
        GPIO_WriteBit(PORT_LED, LED,
        (BitAction)(1^GPIO_ReadOutputDataBit(PORT_LED, LED)));
    }
}

```

Bài 2: Chương trình phát xung PWM dùng TIM2 channel 3 với tần số 2hz duty 50% (TIM2 channel 3 là chân PA2)

*Hướng dẫn: Giả sử

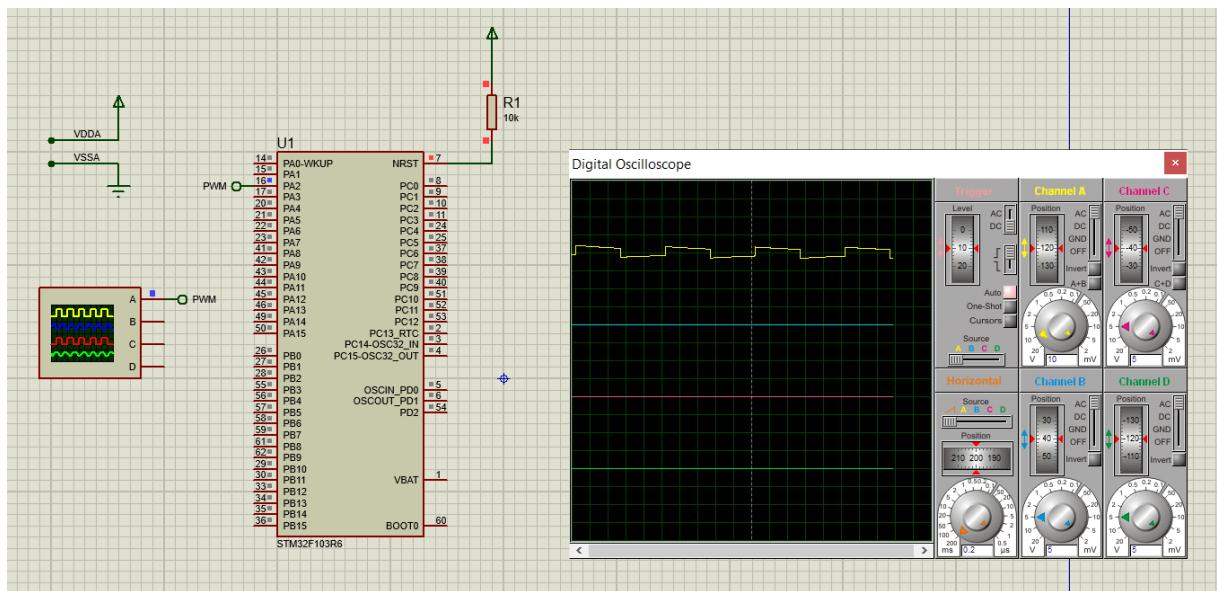
APB1 = 8Mhz

$$F_{PWM} = 2\text{hz} \rightarrow T_{PWM} = 1/2\text{s}$$

Chọn bộ chia là 999 $\rightarrow F_{timer} = 8\text{Mhz}/(999+1) = 8000\text{hz}$

Chọn giá trị tự nạp lại là 3999. $\Rightarrow F_{PWM} = 8000/(3999+1) = 2\text{hz}$

a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"

#define PWM          GPIO_Pin_2
#define PORT_PWM    GPIOA
#define RCC_PWM     RCC_APB2Periph_GPIOA

void Clock_Config(void);
void GPIO_configuration(void);
void TIM2_PWM_CH3_configuration(void);

int main(void)
{
    Clock_Config(); // configuraion clock
    SystemCoreClockUpdate(); // update SystemCoreClock varibale

    GPIO_configuration();
    TIM2_PWM_CH3_configuration();

    while(1)
    {

    }
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div1);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET) {}
    /* Select HSI as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till HSI is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}

}

void GPIO_configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_PWM, ENABLE);

    GPIO_InitStructure.GPIO_Pin = PWM;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(PORT_PWM, &GPIO_InitStructure);
}

void TIM2_PWM_CH3_configuration(void)
```

```

{
    TIM_TimeBaseInitTypeDef TIM_InitStructure;

    /*
        Luu y: voi phien ban proteus 8.6, chung ta phai enable TIM1 thi moi
        co the dung TIMx.
        => bug cua phan men proteus.
    */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
    TIM_TimeBaseStructInit(&TIM_InitStructure);
    TIM_Cmd(TIM1, ENABLE);

    /*configuration TIM2*/
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_InitStructure.TIM_Period = 7999;
    TIM_InitStructure.TIM_Prescaler = 999;
    TIM_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_InitStructure.TIM_RepetitionCounter = 0x00;
    TIM_TimeBaseInit(TIM2, &TIM_InitStructure);

    TIM_TimeBaseInit(TIM2, &TIM_InitStructure);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); //enable update interrupt
    TIM_Cmd(TIM2, ENABLE);

    //cau hinh TIM2 channel 3 xuat PWM
    TIM_OCInitTypeDef TIM_OCInitStructure;

    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //che do tao xung PWM
    o ngo ra loai 1
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //cho
    phep output compare xuat tin bieu ra o Channel 3
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //cau hinh
    muc tich cuc
    TIM_OCInitStructure.TIM_Pulse = 3999; //cau hinh gia tri so sanh cho
    output compare
    TIM_OC3Init(TIM2, &TIM_OCInitStructure);

    TIM_CtrlPWMOutputs(TIM2, ENABLE); //cho phep xuat tin hieu ra o TIM2
    TIM_Cmd(TIM2, ENABLE);
}

```

III, Bài tập

Bài tập 1: Cấu hình ngắt tràn TIM4. Led trên chân PB8 sáng tắt với tốc độ là 1s sáng 1s tắt.

Bài tập 2: Chương trình phát xung PWM dùng TIM4 channel 4 với tần số 1Khz duty 20% (TIM4 channel 4 là PB9).

PHẦN 7: Ngắt Ngoài

I, Giới thiệu về ngắt ngoài

NVIC - Nested vectored interrupt controller là bộ vector ngắt lồng nhau. Nghĩa là chúng ta có thể sử dụng kết hợp nhiều ngắt trong một chương trình. Ngắt là một phần quan trọng và thiết yếu của chương trình. Nếu không có ngắt thì chương trình sẽ thực hiện theo 1 trình tự từ trên xuống dưới mà không có bất kì sự can thiệp nào. Điều đó là bất lợi khi có 1 tác động ngoài xảy ra, chương trình sẽ không xử lý kịp thời dẫn đến việc bỏ qua tác động đó. Ngắt ra đời để phục vụ cho các sự cố đó.

Một số thông số ngắt chính của STM32F103:

- 16 mức ưu tiên có thể lập trình được.
- Độ trễ thấp (xảy ra ngắt cực kỳ nhanh).
- Có quản lý năng lượng cho vector ngắt.
- Có các thanh ghi điều khiển quá trình ngắt.
- 68 vector ngắt(xem thêm trong reference manual).

Ngắt ngoài nằm trong 1 phần của ngắt NVIC. Mỗi EXTI – interrupt/event controller có thể được lập trình chọn loại sự kiện/ ngắt, chọn cạnh lên, cạnh xuống hoặc cả 2, mức ưu tiên ngắt.

Một số tính năng chính của ngắt ngoài:

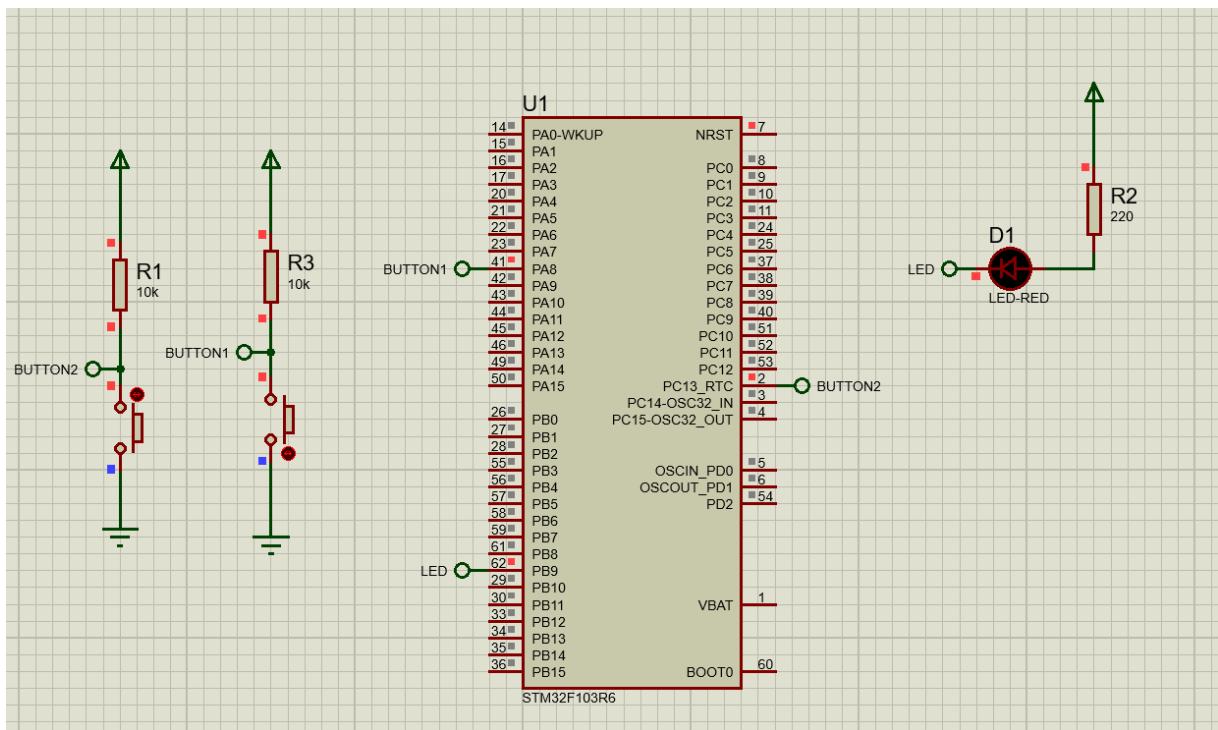
- Kích hoạt độc lập và mặt nạ cho mỗi line sự kiện/ngắt.
- Có bit trạng thái riêng cho mỗi line ngắt.
- Có thể có tối đa 20 sự kiện/ ngắt, tham khảo thêm trong reference manual.
- Kiểm tra tín hiệu ngoài có độ rộng xung nhỏ hơn clock trên APB2.
- Sơ đồ khói của các khói điều khiển ngắt ngoài:

II, Hướng dẫn thực hành với KIT STM32F103C8T6

Có 2 loại ngắt ngoài chính đó là ngắt ngoài trên các chân điều khiển ở dạng thông thường và ngắt ngoài trên các ứng dụng như : PVD, RTC, USB, Ethernet.

Bài 1: Chương trình điều khiển led (B9) sáng hoặc tắt nhờ hai nút bấm On (C13), Off(A8). Cấu hình chân A8 có mức ưu tiên cao hơn.

a, mô phỏng trên proteus



b, code

```
#include "stm32f10x.h"

#define BUTTON_1      GPIO_Pin_8
#define PORT_BUTTON_1 GPIOA
#define RCC_BUTTON_1   RCC_APB2Periph_GPIOA

#define BUTTON_2      GPIO_Pin_13
#define PORT_BUTTON_2 GPIOC
#define RCC_BUTTON_2   RCC_APB2Periph_GPIOC

#define LED          GPIO_Pin_9
#define PORT_LED     GPIOB
#define RCC_LED       RCC_APB2Periph_GPIOB

void LED_Config(void);
void GPIO_EXTI_Config(void);
void EXTI_Config(void);
void NVIC_EXTI_Config(void);
void EXTI9_5_IRQHandler(void);
void EXTI15_10_IRQHandler(void);

// cau hinh cho LED PB9
void LED_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_LED, ENABLE); //cap clock cho GPIOB

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = LED;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(PORT_LED, &GPIO_InitStructure); //cai dat cac cau hinh tren
cho GPIOB
}

//cau hinh ngat ngoai cho 2 chan PA8 va PC13
/*
    B1:cap xung va cau hinh chan PA8 va PC13 nhan tin hieu ngat ngoai la
input pull-up
*/
void GPIO_EXTI_Config(void){
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_BUTTON_1 | RCC_BUTTON_2 |
RCC_APB2Periph_AFIO, ENABLE);
    //do ngat ngoai la chuc nang thay the nen phai bat AIFO

    //cau hinh chan PA8
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Pin = BUTTON_1;
    GPIO_Init(PORT_BUTTON_1, &GPIO_InitStructure);

    //cau hinh chan PC13
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
    GPIO_Init(PORT_BUTTON_2, &GPIO_InitStructure);

    //chon chan PA8 va PC13 la chan nhan tin hieu ngat ngoai
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource8);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource13);
```

```

}

/*
     B2:Cau hinh va cho phep ngat ngoai o EXTI
*/
void EXTI_Config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;

    EXTI_InitStructure EXTI_Line = EXTI_Line8 | EXTI_Line13; // chon kenh 8
    va kenh 13 ung voi A8 va C13
    EXTI_InitStructure EXTI_Mode = EXTI_Mode_Interrupt; //chon che do ngat
    ngoai
    EXTI_InitStructure EXTI_Trigger = EXTI_Trigger_Falling; //chon canh
    tich cuc la canh xuong
    EXTI_InitStructure EXTI_LineCmd = ENABLE; //cho phep kenh ngat ngoai
    duoc cau hinh
    EXTI_Init(&EXTI_InitStructure); //lenh cau hinh cac thong so duoc luu
    trong EXTI_InitStructure
}

/*
     B3: cau hinh cap do uu tien va cho phep ngat ngoai o NVIC
*/
void NVIC_EXTI_Config()
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0); //0 cap
    PreemptionPriority va 16 cap SubPriority
    NVIC_InitStructure.NVIC IRQChannel = EXTI9_5_IRQn; // chon cac kenh tu
    5-9
    NVIC_InitStructure.NVIC IRQChannelSubPriority = 0; //chon thu tu uu
    tien
    NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE; // cho phep ngoai vi
    dang dc cau hinh o NVIC
    NVIC_Init(&NVIC_InitStructure); // lenh cau hinh cac thong so duoc luu
    trong NVIC_InitStructure cho NVIC

    NVIC_InitStructure.NVIC IRQChannel = EXTI15_10_IRQn;//chon kenh tu 10
    den 15
    NVIC_InitStructure.NVIC IRQChannelSubPriority = 1; //chon muc uu tien
    NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

/*
     B4: Viet chuong trinh con phuc vu ngat ngoai
*/
// chuong trinh con phuc vi ngat ngoai cho chan PA8
void EXTI9_5_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_IMR_MR8) != RESET) //kiem tra co phai la kenh
    8 ngat khong?
    {
        EXTI_ClearITPendingBit(EXTI_IMR_MR8); //xoa co ngat kenh 8
        GPIO_SetBits(PORT_LED, LED); //mo Led
    }
}

//chuong trinh con phuc vu ngat ngoai cho chanP C13

```

```

void EXTI15_10_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_IMR_MR13) != RESET) //kiem tra co phai la kenh
13 ngat khong
    {
        EXTI_ClearITPendingBit(EXTI_IMR_MR13); //xoa co ngat kenh 13
        GPIO_ResetBits(PORT_LED, LED); //tat Led
    }
}

int main(void)
{
    LED_Config();
    GPIO_EXTI_Config();
    EXTI_Config();
    NVIC_EXTI_Config();
    EXTI9_5_IRQHandler();
    EXTI15_10_IRQHandler();

    while(1);
}

```

III, Bài tập

Bài tập 1: Viết chương trình đọc nút nhấn trên chân PC13 sử dụng ngắt ngoài, nhấn lần 1 đèn trên chân PB9 sáng, lần thứ 2 led trên chân PB9 tắt... lặp đi lặp lại như vậy.

PHẦN 8: ADC

I, Giới thiệu về ADC

a, ADC là gì

Mạch chuyển đổi tương tự ra số hay ADC (viết tắt tiếng Anh: Analog-to-Digital Converter) là hệ thống mạch thực hiện chuyển đổi một tín hiệu analog (tín hiệu tương tự) liên tục, ví dụ như tín hiệu âm thanh thanh micro, hay tín hiệu ánh sáng trong máy ảnh kỹ thuật số, thành tín hiệu kỹ thuật số. Một hệ thống ADC có thể bao gồm một bộ phận phần cứng (như một bộ tính toán độc lập) làm nhiệm vụ chuyển đổi tín hiệu analog (dưới dạng điện áp hay dòng điện) thành các giá trị số (digital) đại diện cho cường độ điện áp hay tín hiệu đó. *Theo wikipedia*

b, quá trình chuyển đổi ADC

Bộ chuyển đổi tương tự sang số – ADC (Analog to Digital Converter) lấy mức điện thế vào tương tự sau đó một thời gian sẽ sinh ra mã đầu ra dạng số biểu diễn đầu vào tương tự. Tiến trình biến đổi A/D thường phức tạp và mất nhiều thời gian hơn tiến trình chuyển đổi D/A. Do đó có nhiều phương pháp khác nhau để chuyển đổi từ tương tự sang số.

Các bước chuyển đổi AD, Quá trình chuyển đổi A/D nhìn chung được thực hiện qua 4 bước cơ bản, đó là:

- lấy mẫu
- nhớ mẫu
- lượng tử hóa
- mã hóa

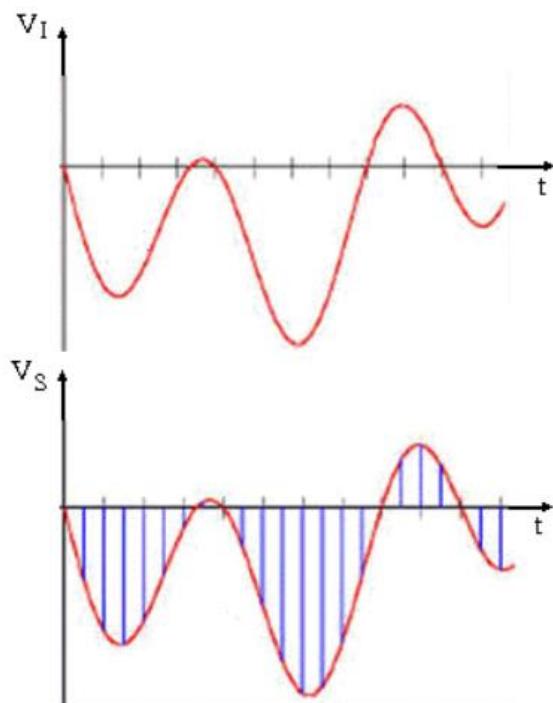
Định lý lấy mẫu : Đối với tín hiệu tương tự VI thì tín hiệu lấy mẫu VS sau quá trình lấy mẫu có thể khôi phục trở lại VI một cách trung thực nếu điều kiện sau đây thỏa mãn:

$$F_s \geq 2F_{max}$$

Trong đó f_s : tần số lấy mẫu

f_{max} : là giới hạn trên của giải tần số tương tự

Vì mỗi lần chuyển đổi điện áp lấy mẫu thành tín hiệu số tương ứng đều cần có một thời gian nhất định nên phải nhớ mẫu trong một khoảng thời gian cần thiết sau mỗi lần lấy mẫu. Điện áp tương tự đầu vào được thực hiện chuyển đổi A/D trên thực tế là giá trị VI đại diện, giá trị này là kết quả của mỗi lần lấy mẫu.



Lượng tử hóa và mã hóa: Tín hiệu số không rời rạc trong thời gian mà còn không liên tục trong biến đổi giá trị. Một giá trị bất kỳ của tín hiệu số đều phải biểu thị bằng bội số nguyên làn giá trị đơn vị nào đó, giá trị này là nhỏ nhất được chọn. Nghĩa là nếu dùng tín hiệu số biểu thị điện áp lấy mẫu thì phải bắt điện áp lấy mẫu hóa thành bội số nguyên làn giá trị đơn vị. Quá trình này gọi là lượng tử hóa. Đơn vị được chọn theo qui định này gọi là đơn vị lượng tử, kí hiệu D. Như vậy giá trị bit 1 của LSB tín hiệu số bằng D. Việc dùng mã nhị phân biểu thị giá trị tín hiệu số là mã hóa. Mã nhị phân có được sau quá trình trên chính là tín hiệu đầu ra của chuyên đổi A/D.

Mạch lấy mẫu và nhớ mẫu: Khi nối trực tiếp điện thế tương tự với đầu vào của ADC, tiến trình biến đổi có thể bị tác động ngược nếu điện thế tương tự thay đổi trong tiến trình biến đổi. Ta có thể cải thiện tính ổn định của tiến trình chuyển đổi bằng cách sử dụng mạch lấy mẫu và nhớ mẫu để ghi nhớ điện thế tương tự không đổi trong khi chu kỳ chuyển đổi diễn ra. Hình 5.18 là một sơ đồ của mạch lấy mẫu và nhớ mẫu.

c, Bộ chuyển đổi ADC trong STM32F103C8T6

STM32 có tổng cộng 18 kênh ADC nhưng 2 kênh nằm trong chip, như vậy có thể dùng 16 kênh đo tín hiệu analog bên ngoài. PCLK2 cung cấp tần số cho bộ ADC và lưu ý tần số không được vượt quá 14Mhz

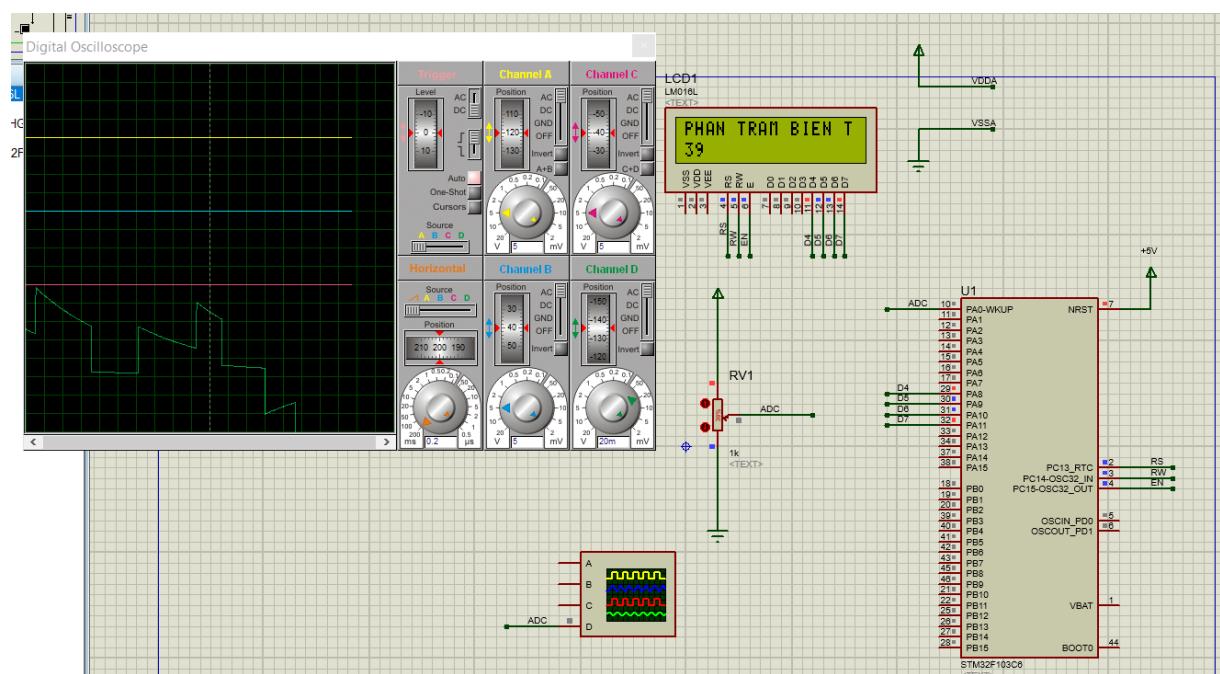
Đặc điểm chính:

- + Độ phân giải 12bit và có thể config 10bit 8bit hoặc 6 bit tăng tốc độ chuyển đổi
- + Ngắt sự kiện khi chuyển đổi xong, event và watchdog xảy ra khi không thể chuyển đổi.
- + Chuyển đổi liên tục.
- + Scan mode: Quét và chuyển đổi từ kênh 0-n
- + Điều khiển chuyển đổi từ bên ngoài sử dụng tín hiệu trigger.
- + Dual mode: Cả 2 bộ ADC cùng convert các kênh một lúc. Ví dụ ADC1 từ 0-15 và ADC2 từ 15-0. Tăng tốc độ chuyển đổi
- + Sample time: Thời gian để điện áp nạp đầy tụ để bộ ADC lấy mẫu. Thời gian phải đủ để điện áp tụ đến bằng điện áp input. Như vậy phụ thuộc vào điện trở nội của điện áp input. Với điện trở nội thấp -> sample time sẽ thấp hơn và ngược lại.
- + Continuous mode và discontinuous mode: Với continuous mode ADC sẽ tự động chuyển đổi lại khi chuyển đổi xong và discontinuous mode ngược lại. Đối với chuyển đổi nhiều kênh cùng một lúc nên dùng discontinuous mode như thế sẽ giảm thời gian đọc một kênh nhất định nào đó mà không phải đọc liên tục từ kênh 0-n.
- + Thời gian chuyển đổi STM32F103xx 1us tại 56Mhz (1.17us với 72Mhz) Vì xung clock cấp cho bộ ADC max 14Mhz ->khi HCLK = 56Mhz suy ra xung clock cấp cho bộ max ADC = $56/2/2 = 14\text{Mhz}$ và nếu HCLK = 72Mhz -> max xung clock bộ ADC = $72/1/6 = 12\text{Mhz}$.
- + Vref: điện áp so sánh. Đối với chip 144 chân sẽ có chân input điện áp so sánh $3.6\text{V} \geq V_{\text{ref}} \geq 2.4\text{V}$. và phải có lọc cẩn thận để ADC hoạt động ổn định. Với chip 64 chân trỏ xuống bạn không cần quan tâm vì điện áp so sánh lấy ở trong chip.
- + Điện áp input cho kênh ADC $V_{\text{ref-}} \leq V_{\text{in}} \leq V_{\text{ref+}}$
- + Bộ DMA để tăng tốc độ chuyển đổi

II, Hướng dẫn lập trình ADC trên KIT STM32F103C8T6

Bài 1 : Đọc phân trám giá trị biên trở

a, Mô phỏng trên proteus



b, code

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_adc.h"
#include "lcd16x2.h"
#include "delay.h"
#include "stdio.h"

void Clock_Config(void);
void ADC_configuration();

uint16_t raw_value, sum_adc, num_percent;
char str_percent[2];

int main(void)
{
    DelayInit();      // Delay initialization
    LCD_Init();       // LCD initialization
    ADC_configuration();

    while(1)
    {
        sum_adc = 0;
        //doc gia tri ADC 10 lan roi lay ket qua
        for(int i=0;i<10;i++)
        {

            raw_value = ADC_GetConversionValue(ADC1);
            sum_adc += raw_value;
            DelayMs(1);
        }
        sum_adc /= 10;
        num_percent = (sum_adc*100)/4096;
        sprintf(str_percent, "%d *C", num_percent);
        LCD_Gotoxy(0,0);
        LCD_Puts("PHAN TRAM BIEN TRO");
        LCD_Gotoxy(0,1);
        LCD_Puts(str_percent);
        DelayMs(500);
    }
}

void Clock_Config(void)
{
    /* RCC system reset */
    RCC_DeInit();
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);
    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div1);
    /*enable HSI source clock*/
    RCC_HSICmd(ENABLE);
    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET) {}
    /* Select HSI as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    /* Wait till HSI is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x00) {}
}
```

```

void ADC_Configuration()
{
    //cau hinh cho chan GPIO va bo ADC hoat dong
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;

    /*cau hinh chan Input cua bo ADC1 la chan PA0*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //cau hinh ADC
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);

    /* Cau hinh channel, rank, thoi gian lay mau */
    ADC-RegularChannelConfig(ADC1, ADC_Channel_0, 1,
    ADC_SampleTime_55Cycles5);
    /* Cho phep bo ADC1 hoat dong */
    ADC_Cmd(ADC1, ENABLE);

    /* Bat dau chuyen doi ADC */
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

```

III, Bài tập

Bài 1: Đọc giá trị nhiệt độ sử dụng cảm biến LM35 và hiển thị nhiệt độ lên LCD, tạo thêm cảnh báo khi nhiệt độ vượt quá 30 độ C.