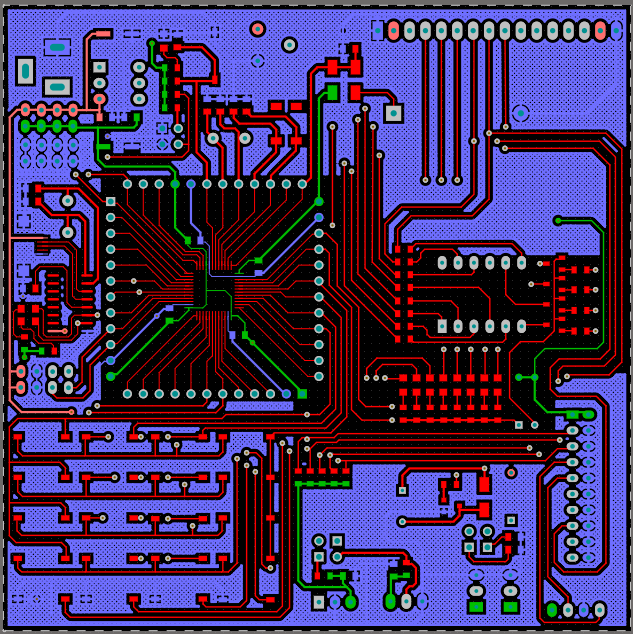


**LẬP TRÌNH CĂN BẢN ARM CORTEX M3 STM32F103C8T6**



**DESIGNER : TRẦN NGỌC KHIÊM**

**CODER : ĐỖ VĂN THÁI**

MỤC LỤC

[**MỤC LỤC 2**](#_Toc93354702)

[**I. TỔNG QUAN VỀ KIT LEARNING STM32F103C8T6 4**](#_Toc93354703)

[**1.1 Giới thiệu về Kit STM32F103C8T6 4**](#_Toc93354704)

[**1.2 Thiết kế mạch nguyên lý 5**](#_Toc93354705)

[**II. LẬP TRÌNH ỨNG DỤNG CHO KIT STM32F103C8T6 7**](#_Toc93354706)

[**2.1 Hướng dẫn cơ bản cho một ứng dụng với KIT STM32F103C8T6 7**](#_Toc93354707)

[**2.2 Các bước tạo một Project trên KeliC MDK 7**](#_Toc93354708)

[**2.3 Nguồn Clock trong STM32 16**](#_Toc93354709)

[**2.3.1 Bộ thanh ghi RCC (Register Clock Control) 18**](#_Toc93354710)

[**2.3.2 Thư viện stm32f10x\_rcc.h 19**](#_Toc93354711)

[**2.4 Lập trình GPIO 20**](#_Toc93354712)

[**2.4.1 Giới thiệu GPIO 20**](#_Toc93354713)

[**2.4.2 Các thanh ghi và lệnh liên quan đến GPIO 20**](#_Toc93354714)

[**2.4.3 Chức năng thay thế và thay đổi vị trí các I/O (Alternate function and remap I/O) 27**](#_Toc93354715)

[**2.4.4 Sử dụng các chân đặc biệt làm chân I/O 27**](#_Toc93354716)

[**2.4.5 Lập trình LED 28**](#_Toc93354717)

[**2.4.6 Lập trình BUTTON 33**](#_Toc93354718)

[**2.4.7 Lập trình LED 7 Thanh 37**](#_Toc93354719)

[**2.5 Lập trình INTERRUPTS và EVENTS 45**](#_Toc93354720)

[**2.5.1 Bộ điều khiển vector ngắt lồng nhau (Nested Vector Interrupt Controller) 45**](#_Toc93354721)

[**2.5.2 Cơ chế ưu tiên ngắt 45**](#_Toc93354722)

[**2.5.3 Các lệnh thông dụng liên quan đến NVIC 46**](#_Toc93354723)

[**2.5.4 Ngắt ngoài 49**](#_Toc93354724)

[**2.5.5 Các lệnh thông dụng liên quan đến INTERRUPTS And EVENTS 50**](#_Toc93354725)

[**2.5.6 Lập trình 51**](#_Toc93354726)

[**2.6. Lập trình USART 52**](#_Toc93354727)

[**2.6.1 Giới thiệu USART 52**](#_Toc93354728)

[**2.6.2 Các lệnh thông dụng liên quan đến USART 53**](#_Toc93354729)

[**2.6.3 Phần mềm và công cụ hỗ trợ 55**](#_Toc93354730)

[**2.6.4 Lập trình 57**](#_Toc93354731)

[**2.7 Lập trình TIMER 64**](#_Toc93354732)

[**2.7.1 Giới thiệu TIMER 64**](#_Toc93354733)

[**2.7.2 Các chế độ hoạt động của TIMER 65**](#_Toc93354734)

[**2.7.3 Các lệnh thông dụng 68**](#_Toc93354735)

[**2.7.4 Lập trình 73**](#_Toc93354736)

[**2.8 Lập trình ADC 75**](#_Toc93354737)

[**2.8.1 Giới thiệu ADC 75**](#_Toc93354738)

[**2.8.2 Những đặc điểm của ADC 75**](#_Toc93354739)

[**2.8.3 Các chế độ hoạt động cơ bản của ADC 76**](#_Toc93354740)

[**2.8.4. Sơ đồ khối và các chân của ADC 78**](#_Toc93354741)

[**2.8.5 Các lệnh thông dụng liên quan đến ADC 80**](#_Toc93354742)

[**2.8.6 Lập trình 87**](#_Toc93354743)

I. TỔNG QUAN VỀ KIT LEARNING STM32F103C8T6

Để thuận tiện cho việc học Vi xử lý ARM Cortex M3 STM32, team đã lựa chọn Vi xử lý STM32F103C8T6 để thiết kế KIT thí nghiệm và xây dựng thư viện để việc lập trình được dễ dàng.

* 1. Giới thiệu về Kit STM32F103C8T6

STM32 là một trong những dòng chip phổ biến của ST với nhiều họ thông dụng nhưF0,F1,F2,F3,F4….. STM32 có đang được thiết kế dựa trên 3 dòng sản phẩm chính như sau: Hiệu năng cao (High-performance), Phổ thông (Mainstream) và Tiết kiệm năng lượng (Ultra Low-Power). Hiệu năng cao (High-performance): là các dòng vi điều khiển có hiệu năng tính toán cao, tốc độ hoạt động lớn và thường được sử dụng trong các ứng dụng đa phương tiện. Dòng này bao gồm các dòng vi điều khiển Cortex-M3/4F/7 với tần số hoạt động từ khoảng 120MHz đến 268MHz. Các dòng này điều hỗ trợ công nghệ ART Accelerator.

Phổ thông (Mainstream): Là dòng vi điều khiển được thiết kế với mục tiêu tối ưu về giá. Các sản phẩm này thường có giảm thấp hơn 1$/pcs với tần số hoạt động trong khoảng từ 48MHz đến 72MHz.

Tiết kiệm năng lượng (Ultra Low-Power): Là nhóm các vi điều khiển hoạt động với tiêu chí tiết kiệm năng lượng tiêu thụ. Các nhóm vi điều khiển này đều có khả năng hoạt động với pin trong thời gian dài và hỗ trợ nhiều chế độ ngủ khác nhau. Với các sản phẩm của hãng STMicrochips, điển hình của nhóm này là dòng ARM Cortex M0+.

STM32F103 thuộc họ F1 với lõi là ARM Cortex M3. STM32F103 là vi điều khiển 32 bit, tốc độ tối đa là 72Mhz. Giá thành cũng khá rẻ so với các loại vi điều khiển có chức năng tương tự. Mạch nạp cũng như công cụ lập trình khá đa dạng và dễ sử dụng.

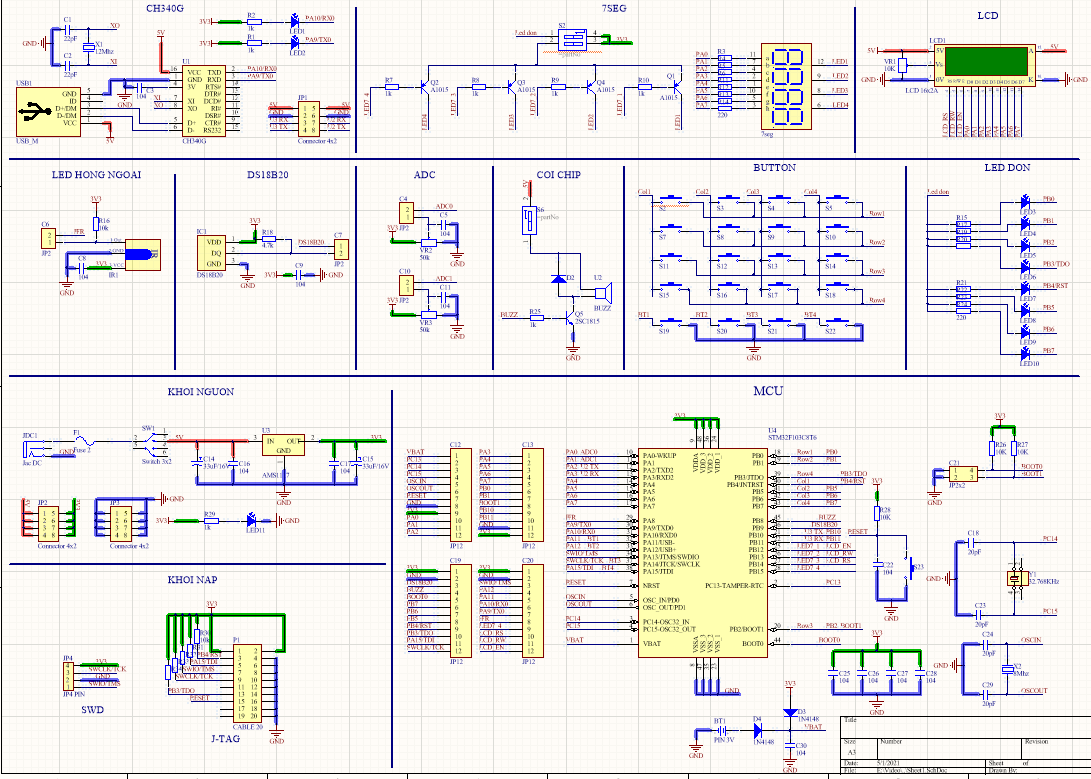
Một số ứng dụng chính: dùng cho driver để điều khiển ứng dụng, điều khiển ứng dụng thông thường, thiết bị cầm tay và thuốc, máy tính và thiết bị ngoại vi chơi game, GPS cơ bản, các ứng dụng trong công nghiệp, thiết bị lập trình PLC, biến tần, máy in, máy quét, hệ thống cảnh báo, thiết bị liên lạc nội bộ…

STM32F103 C8T6 là vi điều khiển 32bit của STMicroelectronics với 64 Kb bộ nhớ Flash, USB 2.0 full-speed, CAN, 7 bộ Timer, 2 bộ ADC và 9 giao diện kết nối

Thông tin chung về sản phẩm:

* Lõi : ARM 32 bit Cortex M3
* Tần số hoạt động lên tới 72 MHz
* Bộ nhớ : 64 Kb Flash , 20Kb SRAM
* Điện áp : 2~3.6 VDC
* Tổng số I/O : 37
* ADC : 2x12 bit, tần số lấy mẫu 1MHz
* DAC : Không
* DMA : Điều khiển 7 kênh DMA
* Timer : 4 bộ, 16 bit ( IC, OC, PWM )
* Giao diện kết nối : 2xI2C, 3xUSART, 2xSPI, CAN, 1xUSB 2.0 full- speed, 1xCAN
* Kiểu chân : LQFP48
* Ứng dụng : Những tính năng này làm cho vi điều khiển STM32F103C8T6 thích hợp cho một loạt các ứng dụng như điều khiển động cơ, kiểm soát các ứng dụng nâng cao, thiết bị y tế và thiết bị cầm tay, máy tính và thiết bị ngoại vi chơi game, GPS, ứng dụng công nghiệp, PLC, biến tần, máy in, máy quét , hệ thống báo động, hệ thống liên lạc video, và HVACs
  1. Thiết kế mạch nguyên lý
* Nguồn vào : 5V-DC
* Qua IC ổn áp AMS1117 để lấy nguồn 3.3 V cho vi xử lý
* Khối nút nhấn + ma trận phím
* Khối JTAG: Hỗ trợ nạp chương trình và debug
* Khối nạp SWD: Hỗ trợ nạp chuẩn 2 dây của ST, sử dụng mạch nạp ST-Link
* Khối hiển thị Led 7 đoạn: Sử dụng Led Anode chung 4 số kích thước 0.36
* Khối hiển thị LCD: giao tiếp 8 bit, 4bit
* Khối Led đơn
* Khối còi
* Khối ADC
* Khối thạch anh và chọn chế độ hoạt động của chip
* Khối reset, wkup, pin Bakup
* Khối giao tiếp USB
* Khối giao tiếp USART
* Khối đọc cảm biến nhiệt độ DS18B20
* Khối Led thu phát hồng ngoại

**Sơ đồ nguyên lý:**



Hình 1.1: Sơ đồ nguyên lý

II. LẬP TRÌNH ỨNG DỤNG CHO KIT STM32F103C8T6

2.1 Hướng dẫn cơ bản cho một ứng dụng với KIT STM32F103C8T6

Trong chương này sẽ đề cập đến cách thức để tạo một Project trên Keil C, các công cụ cần thiết và các ví dụ minh họa về lập trình ARM STM32F103 C8T6  
Các công cụ cần thiết:

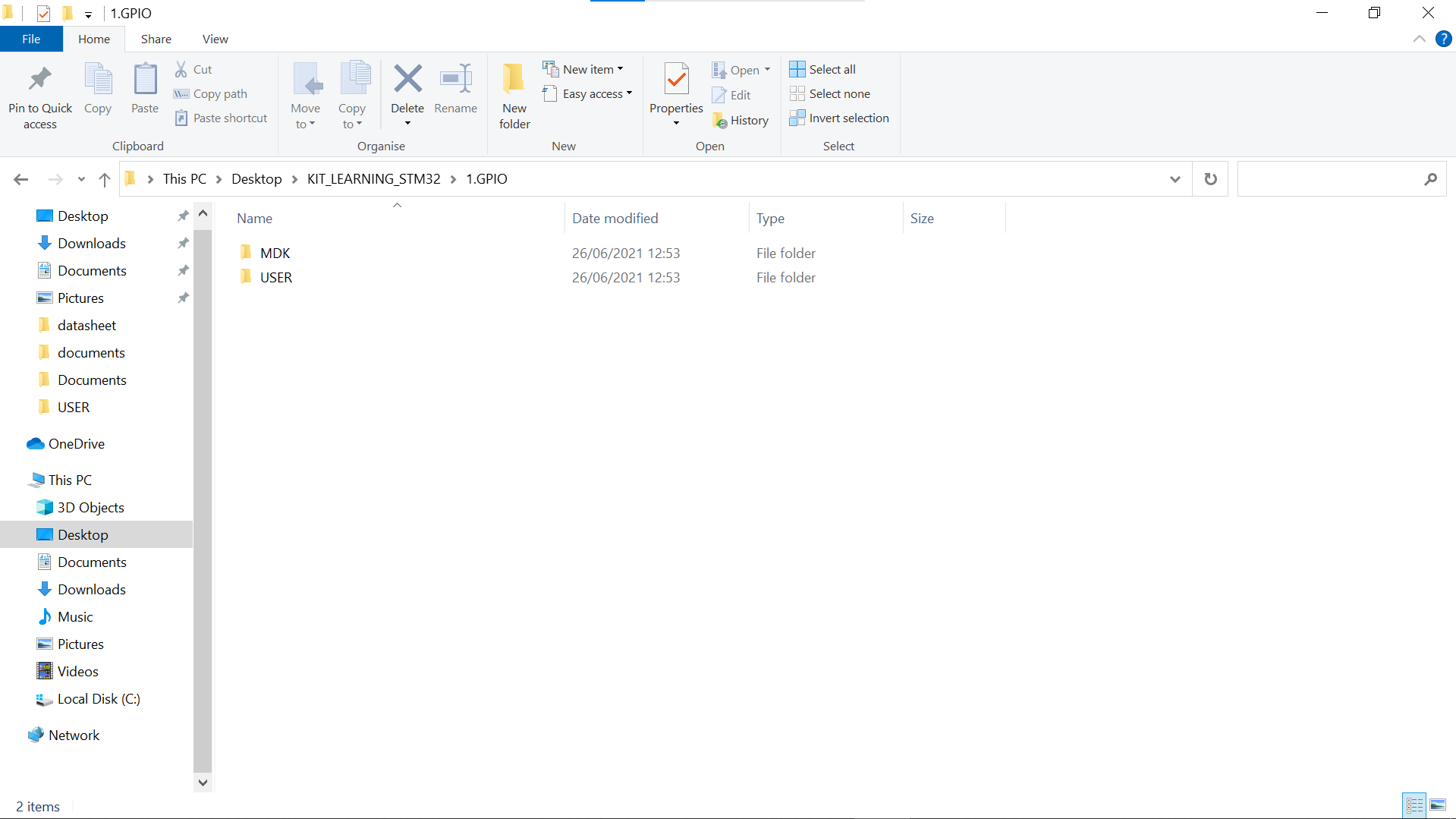
* Driver USB to CH340: sử dụng chip CH340 có chức năng giao tiếp giữa máy tính với MCU qua giao thức UART.
* Mạch nạp ST-Link: sử dụng để nạp code cho MCU.
* Phần mềm Kelic V5: sử dụng để viết chương trình cho MCU.

2.2 Các bước tạo một Project trên KeliC MDK

* Tạo 1 folder để lưu trữ các project:

Folder: ../KIT\_LEARNING\_STM32/1.GPIO/

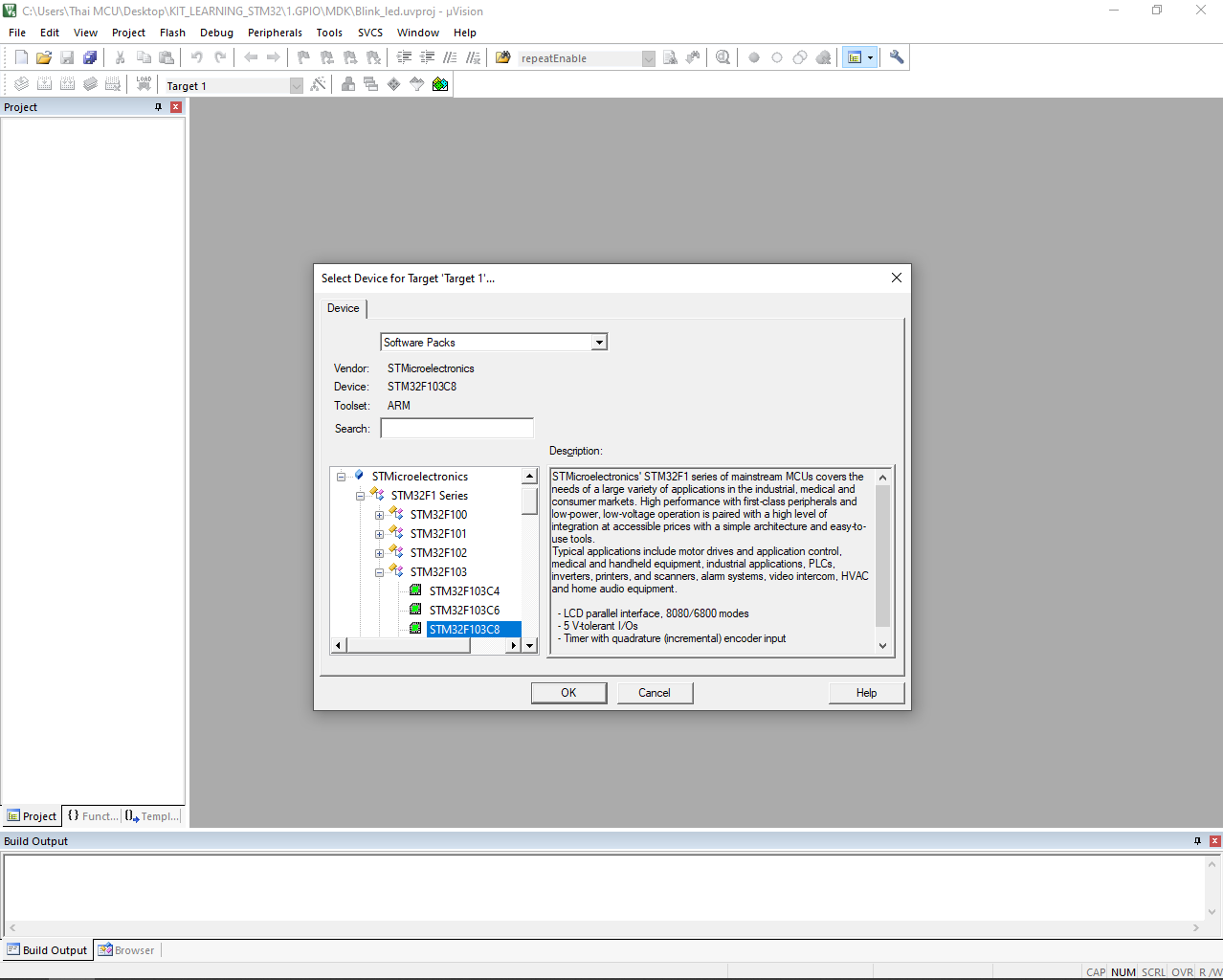
* Bên trong thư mục 1.GPIO tạo them 2 thư mục con là MDK và USER



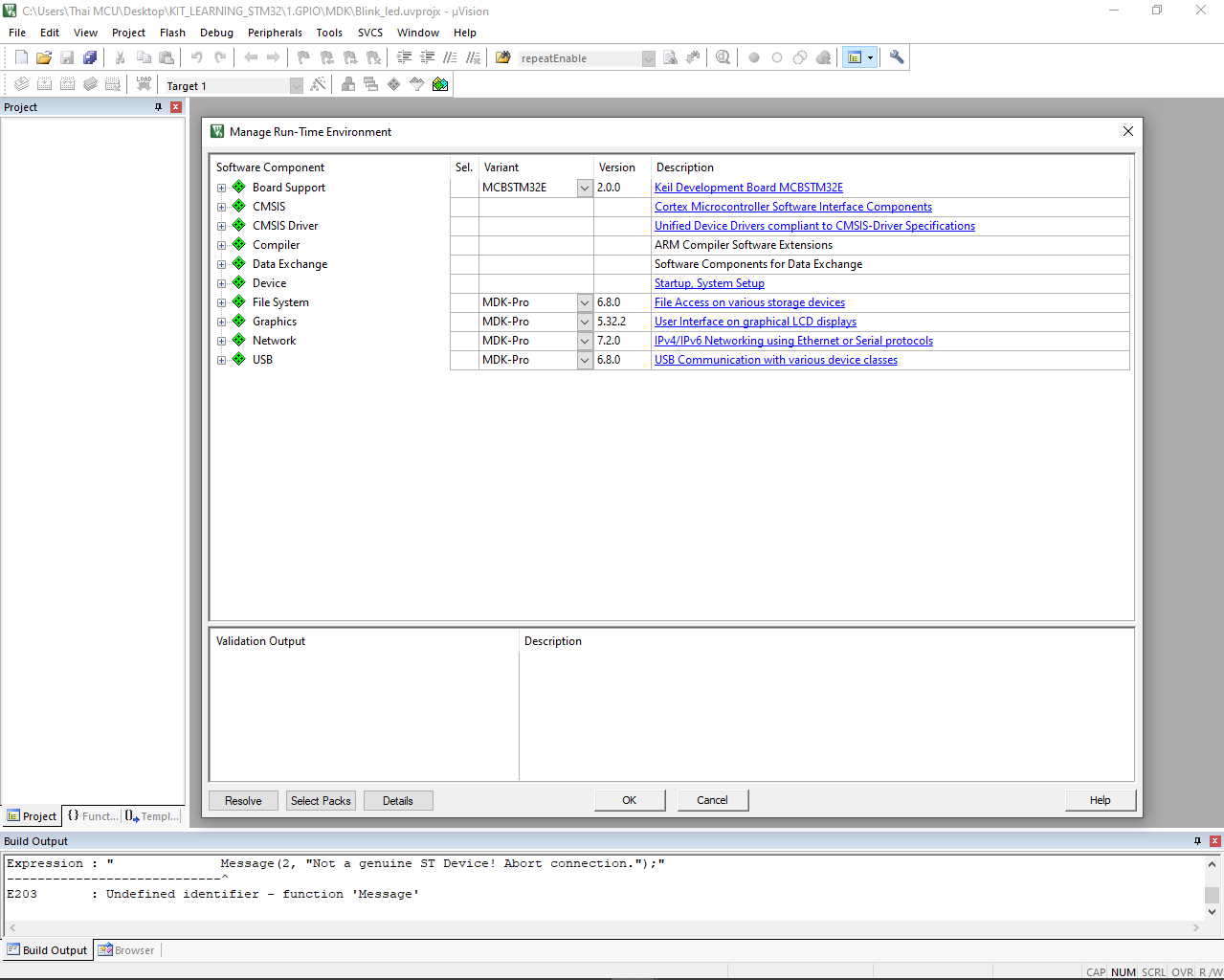
* Mở phần mềm Kelic lên và tạo project mới:



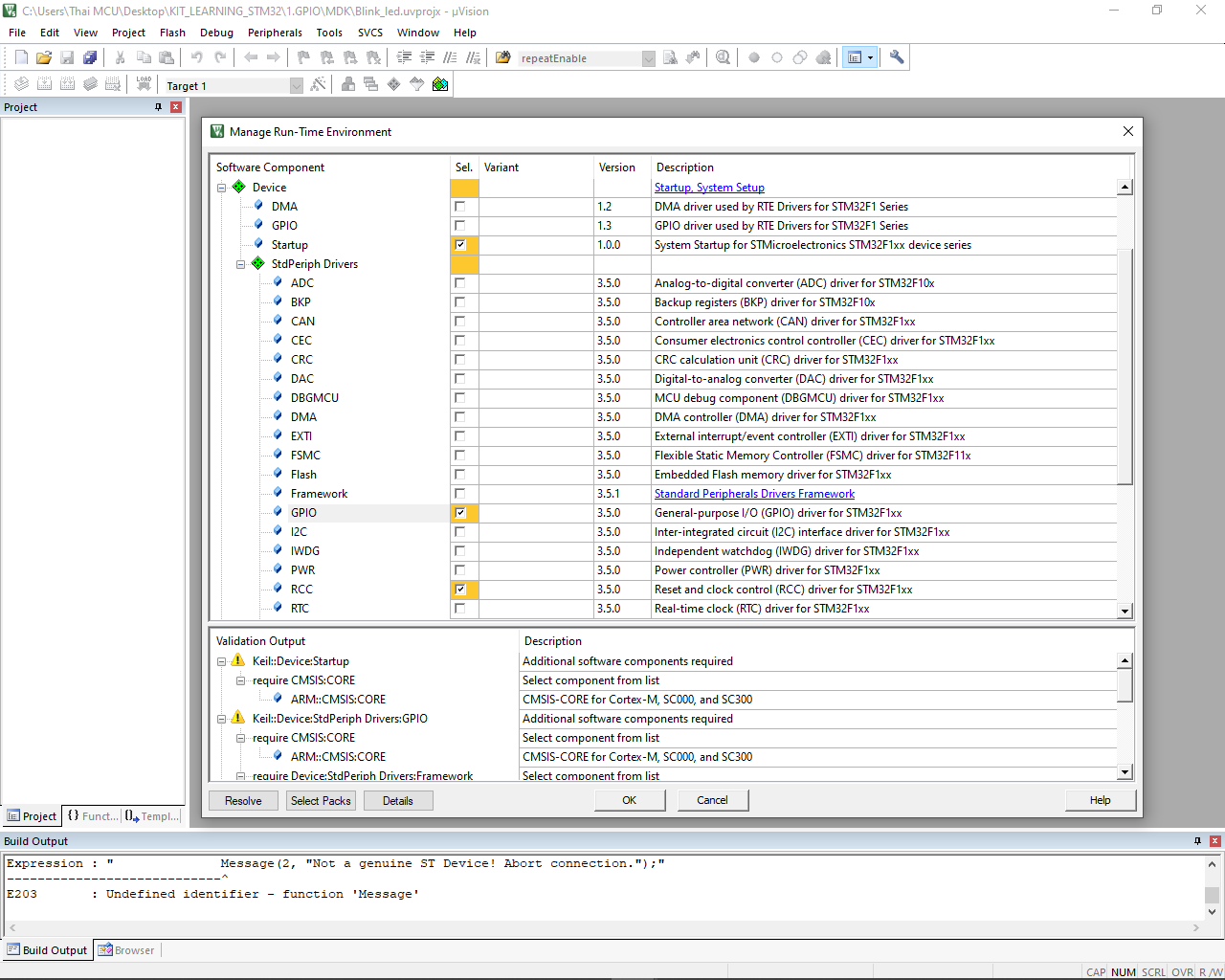
* Chọn Project->New uVision project
* Một hộp thoại Create New Project hiện ra, sau đó trỏ để thư mục MDK vừa tạo ở trên( ../KIT\_LEARNING\_STM32/1.GPIO/MDK ) đặt tên cho project rồi ấn Save.
* Hộp thoại Select Device for Taget hiện ra, tại đây chọn dòng chip stm32f103c8



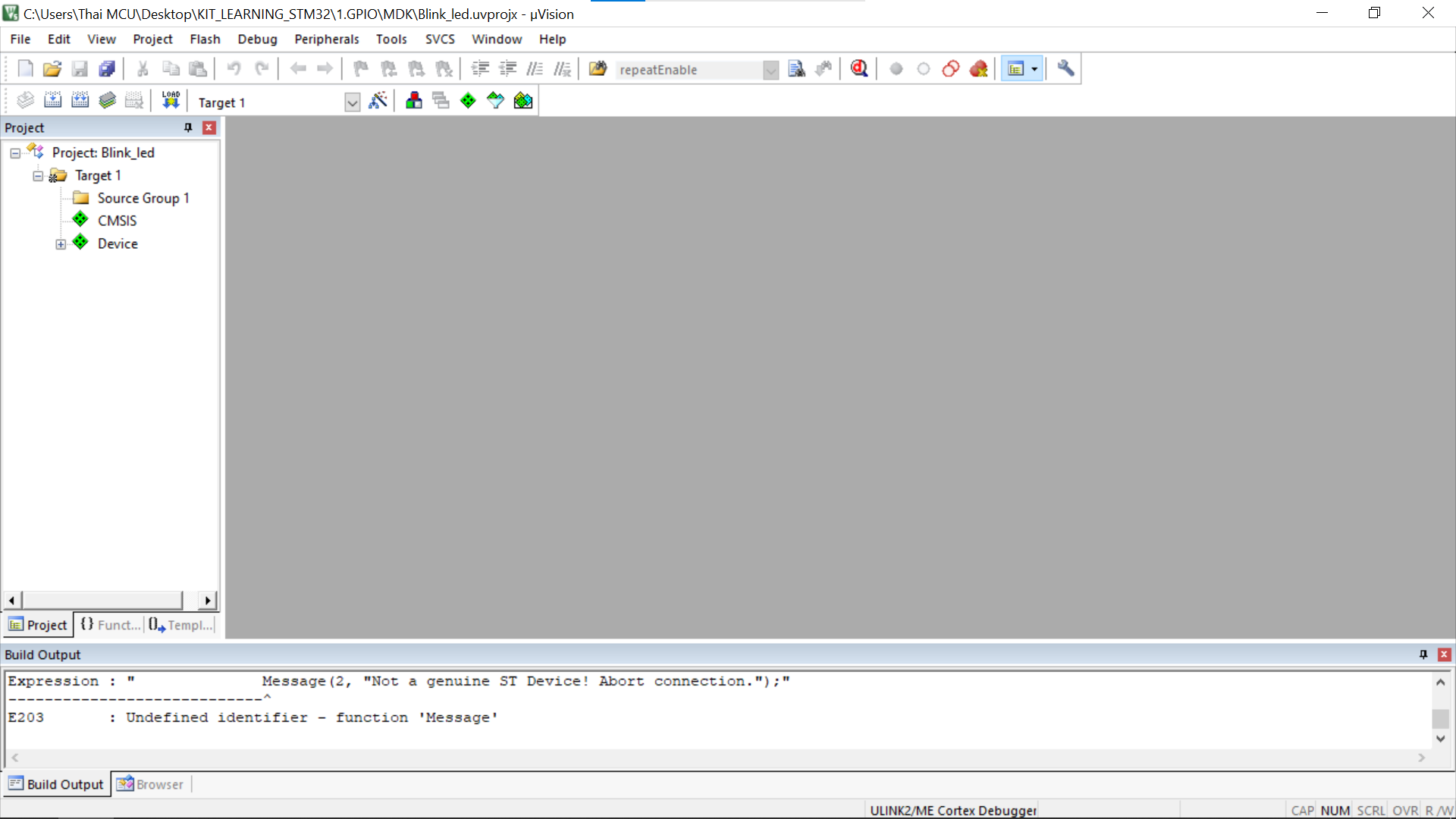
* Sau đó hộp thoại Manage Run-Time Environment hiện ra



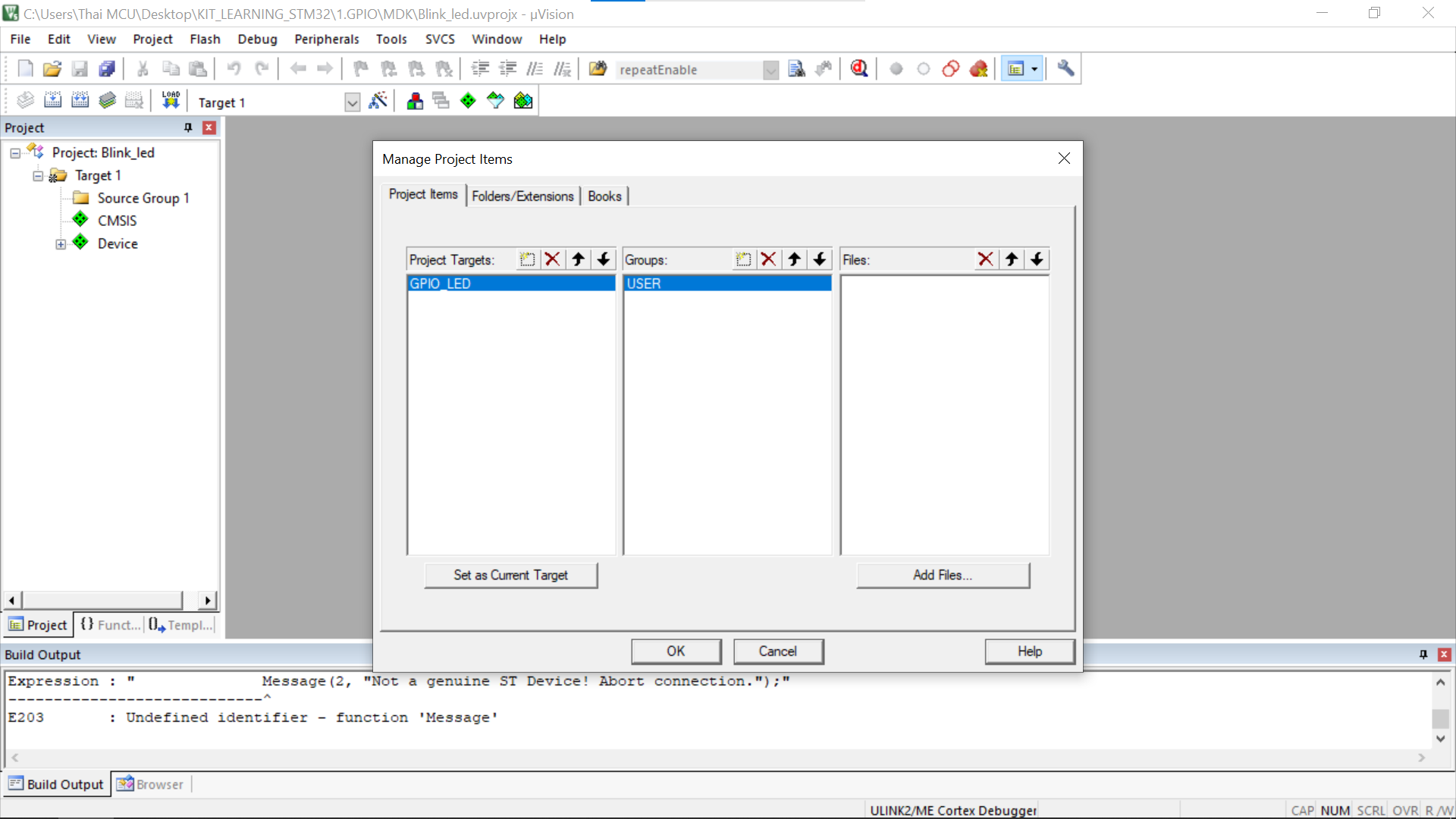
* Sau đó chọn vào phần Device -> StdPeriph Drivers để chọn các ngoại vi sử dụng:



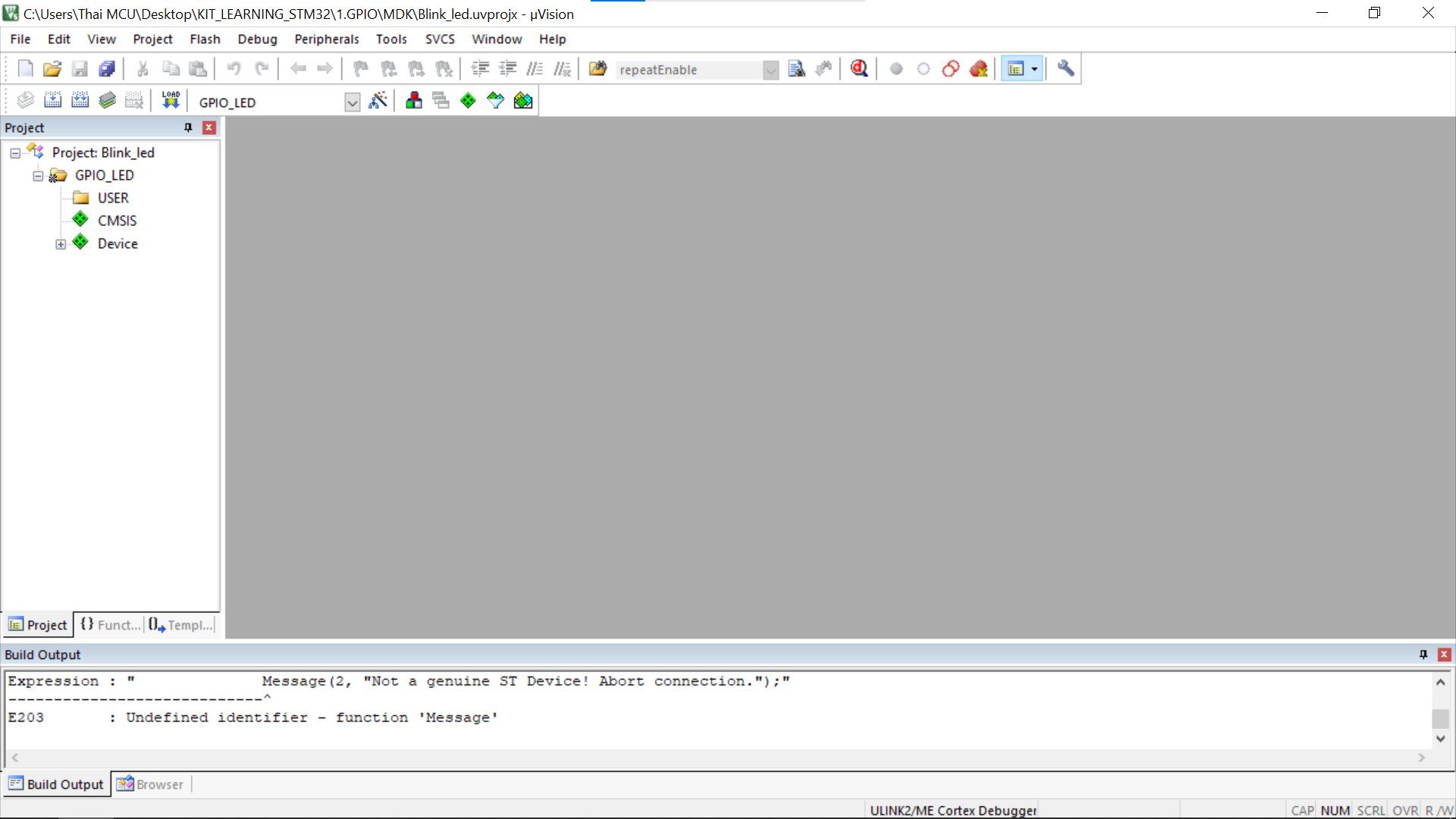
* Sau đó ấn Resolve để phần mềm tự động chọn thêm những device hỗ trợ rồi ấn OK ta sẽ được giao diện:



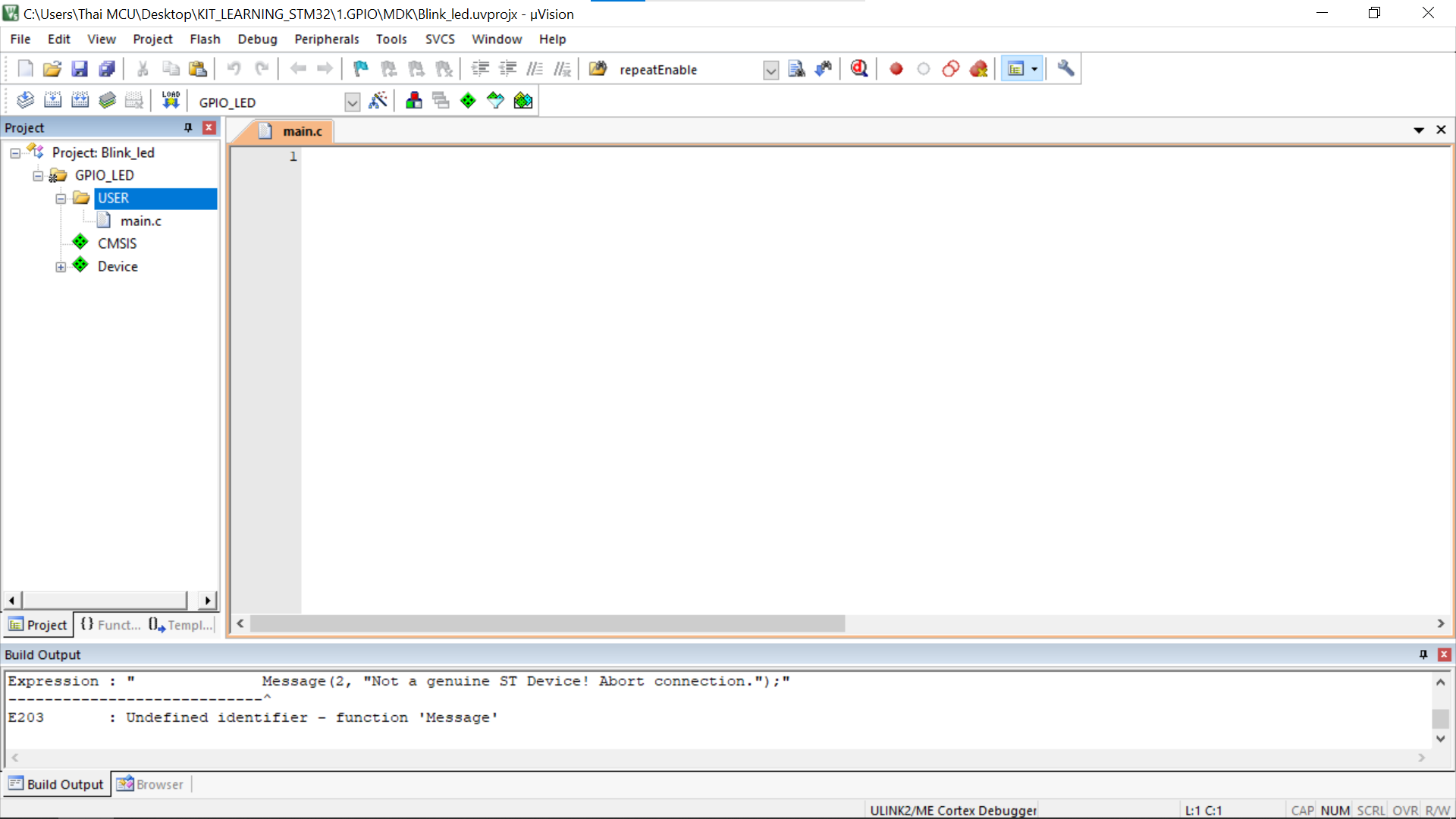
* Tiếp theo cọn biểu tượng  hoặc chọn: Project-> Manage -> project items…
* Hộp thoại Manage Project Items hiện ra. Tại đây phục vụ cho việc quản lý và chỉnh sửa project. Cụ thể trọng phần hướng dẫn này chúng ta sẽ thay đổi tên ở mục project targets thành: GPIO\_LED; đổi tên Groups thành: USER



* Tiếp theo chọn OK ta sẽ có 1 giao diện mới:



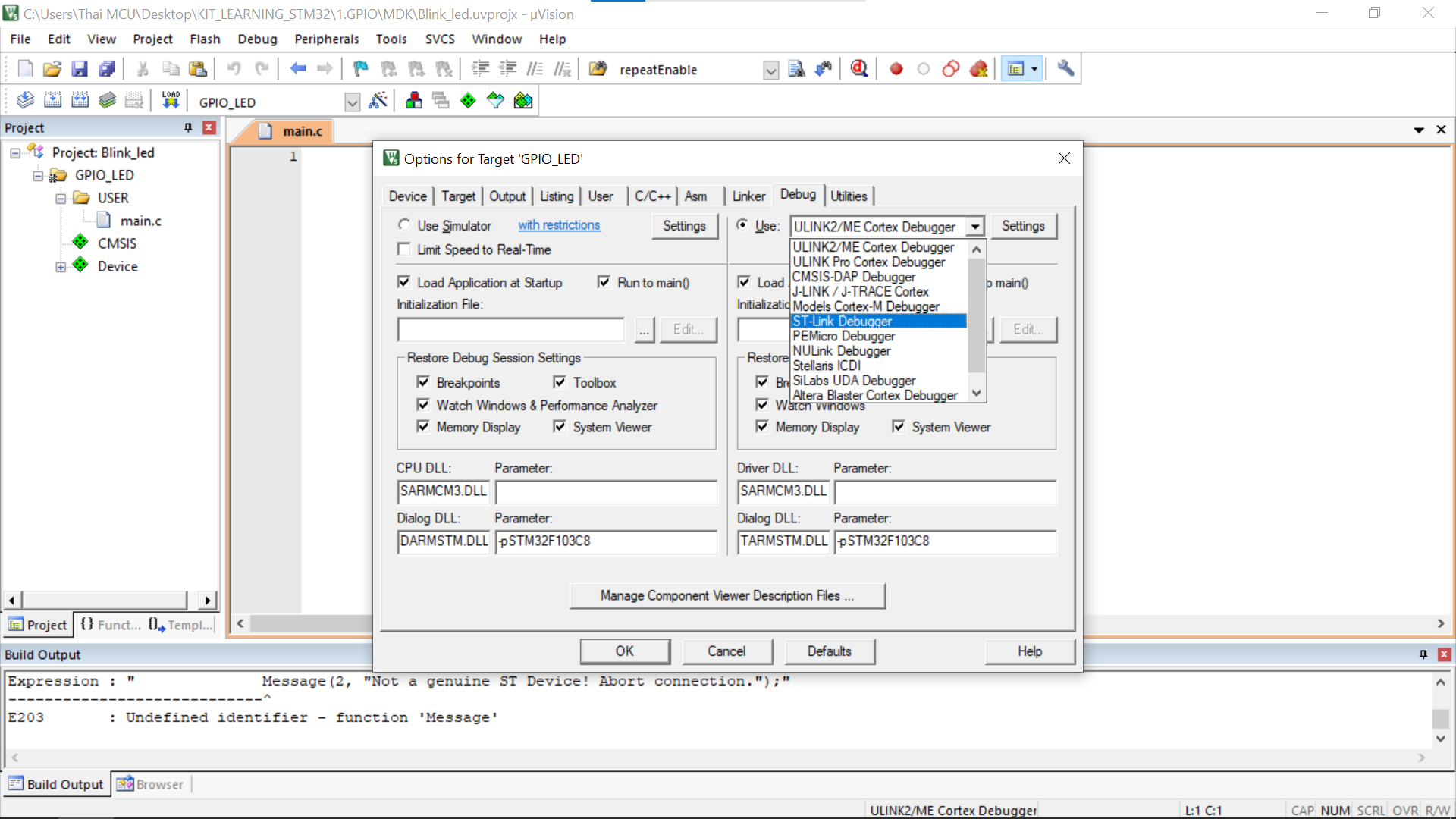
* Tiếp theo ta tạo file .c để viết code bằng cách chọn File -> New hoặc ấn tổ hợp phím Ctrl+N và lưu nó trong thư mục ../KIT\_LEARNING\_STM32/1.GPIO/USER
* Sau đó add file .c vào trong project của keliC mục USER bằng cách chọn vào USER -> add existing files to group ‘USER’ -> trỏ đến thư mục lưu file.c và add file .c đó vào project.



* Tiếp theo sẽ cấu hình để nạp code cho chip: chọn vào biểu tượng  hoặc chọn Project-> Options for target ‘GPIO\_LED’ hoặc ấn tổ hợp phím Alt+F7 hộp thoại Options for target ‘GPIO\_LED’ hiện ra:



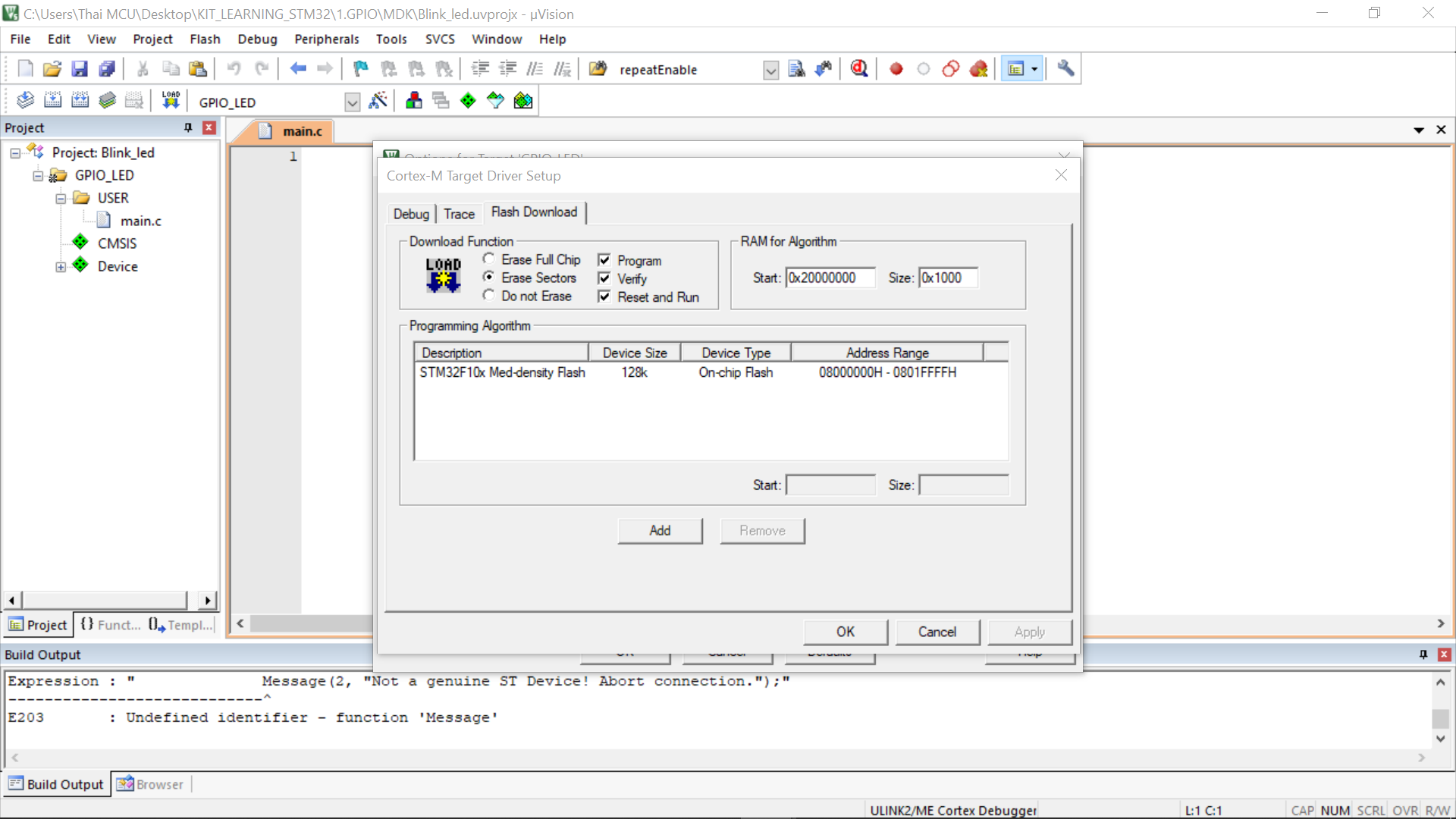
* Chọn vào hộp thoại Debug và chọn mạch nạp ST-Link Debugger



* Tiếp theo chọn Settings hộp thoại Cortex-M Target Driver Setup hiện ra,



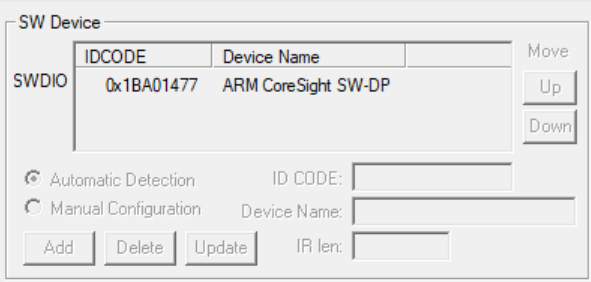
* Chọn hộp thoại Flash Download chọn Reset and Run.



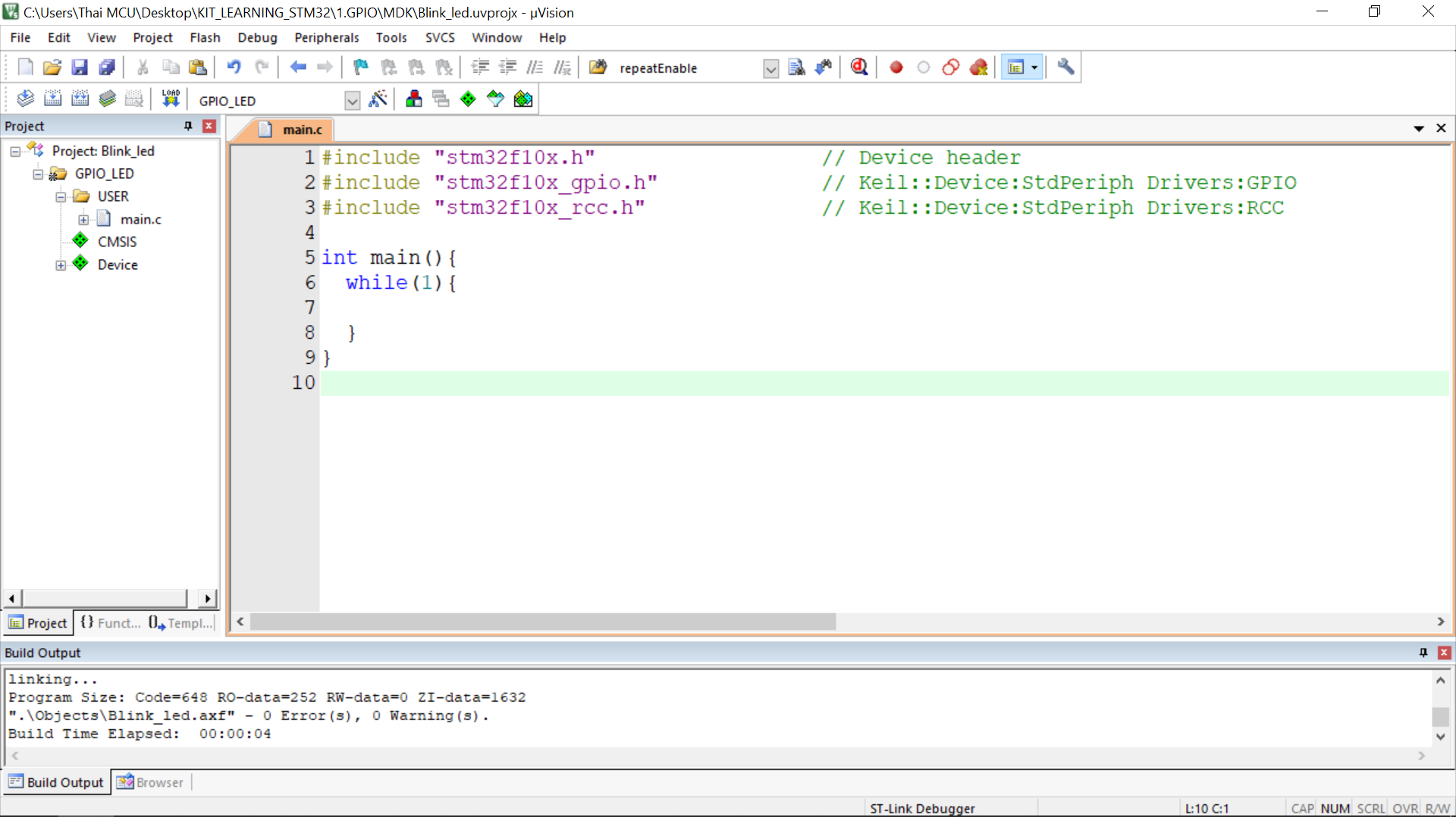
Mục đích sẽ giúp cho chương trình sau khi nạp xuống chip sẽ tự khởi động mà không cần ấn reset.

Một số chú ý: Mọi người kiểm tra các thông số trong hộp thoại Cortex-M Target Driver Setup xem có đúng như trong tài liệu hướng dẫn hay không. Chú ý ở mục Driver mạch nạp và PORT: SW

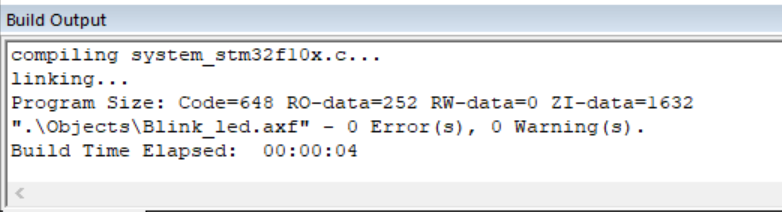
* Nếu nhận mạch nạp thì hộp thoại Cortex0M Target Driver Setup hiển thị:



* Viết Form cho chương trình:



* Để Build code ấn F7 hoặc chọn biểu tượng 
* Để Nạp code ấn F8 hoặc chọn biểu tượng 
* Sau khi Build code kết quả trả về 0 error(s), 0 warning(s) thì quá trình tạo project đã hoàn thành.



2.3 Nguồn Clock trong STM32

Trong vi điều khiển, nguồn clock như là “trái tim” của toàn bộ hệ thống, nó cung cấp xung nhịp cho lõi vi điều khiển và các thiết bị ngoại vi giúp chúng hoạt động được.

Vì vậy, khi mới bắt đầu tìm hiểu, nghiên cứu bất kỳ một dòng vi điều khiển nào chúng ta cần trả lời được 2 câu hỏi:

* Có bao nhiêu nguồn Clock cung cấp cho hệ thống? đặc điểm của các nguồn Clock?.
* Cách cấu hình chọn nguồn Clock cung cấp cho hệ thống?

Trong phần này sẽ trình bày rõ cho chúng ta nguồn Clock của dòng vi điều khiển STM32.

Đối với dòng vi điều khiển STM32, Clock hệ thống có thể được cung cấp bởi 3 nguồn:

* Nguồn Clock dao động nội tốc độ cao (HSI – High Speed Internal): lấy từ dao động RC nội 8MHz.
* Nguồn Clock dao động ngoại tốc độ cao (HSE – High Speed External): lấy từ nguồn dao động thạch anh 4 – 16MHz.
* Nguồn Clock PLL (PLL – Phase Lock Loop).

Ngoài ra, STM32 còn được hỗ trợ thêm 2 nguồn Clock phụ:

* Nguồn dao động RC nội (LSI RC – Low Speed Internal RC): lấy từ nguồn dao động RC nội 40KHz ( cung cấp cho IWDG – Independent Watchdog).
* Nguồn dao động thạch anh ngoại (LSE crystal – Low Speed External): lấy từ dao động thạch anh 32.768KHz (cung cấp cho RTC – Real Time Clock).

|  |
| --- |
|  |

Hình 2.1 Sơ đồ hệ thống clock trong STM32

Nhìn vào sơ đồ Clock ở hình trên, ta cần chú ý tới tần số Clock hoạt động của 3 Bus sau:

* AHB (Advanced High Speed Buses): Đây là Bus kết nối hệ thống, tần số hoạt động lớn nhất của Bus là 72MHz.
* APB1, APB2 (Advanced Peripheral Buses 1,2): Đây là các Bus kết nối với thiết bị ngoại vi và kết nối với hệ thống thông qua AHB.
* Fmax APB1 = 36MHz.
* Fmax APB2 = 72MHz.

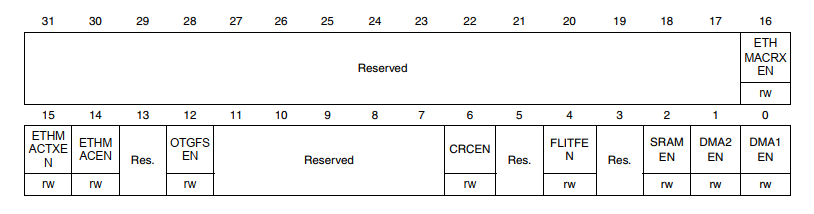
Điều này rất quan trọng, vì khi sử dụng bất kỳ một ngoại vi nào trong tài nguyên của STM32 cần phải cung cấp nguồn Clock cho ngoại vi đấy. Khi đó cần chú ý đến tần số Clock giới hạn của từng ngoại vi để phù hợp với ứng dụng.

2.3.1 Bộ thanh ghi RCC (Register Clock Control)

Bộ thanh ghi RCC là bộ thanh ghi dùng để điều khiển Clock của IC và tất cả các ngoại vi, tài nguyên của IC. Để một bộ ngoại vi như: GPIO, Timer, UART,... hoạt động ngoài Clock của hệ thống được kích hoạt chương trình phần mềm cần thực hiện các lệnh cấu hình để cấp clock cho bộ ngoại vi tương ứng.

Bộ RCC có số lượng thanh ghi rất lơn. Ở giáo trình này tập trung vào 3 thanh ghi điều khiển cấp Clock cho các bộ ngoại vi của vi điều khiển.

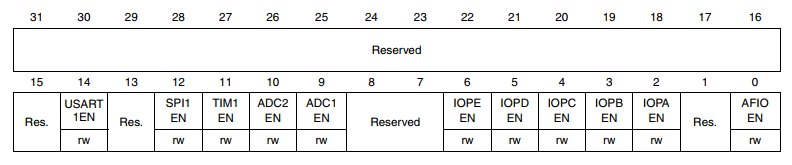
**a. Thanh ghi AHB Peripheral Clock enable register (RCC\_AHBENR)**



Hình 2.2: Thanh ghi RCC\_AHBENR

Thanh ghi này có nhiệm vụ kích hoạt clock cho các ngoại vi được nối với bus điều khiển ngoại vi AHB như: SDIO, CRC, DMA1, DMA2,…

**b. Thanh ghi APB2 peripheral clock enable register (RCC\_APB2ENR)**



Hình 2.3: Thanh ghi RCC\_APB2ENR

Thanh ghi này có nhiệm vụ kích hoạt clock cho các ngoại vi được nối với bus điều khiển ngoại vi APB2 như: Timer, ADC, USART, SPI, IO, AFIO,...

Các bộ GPIO của vi điều khiển đều được kích hoạt thông qua thanh ghi này.

**c. Thanh ghi APB1 peripheral clock enable register (RCC\_APB1ENR)**



Hình 2.4: Thanh ghi RCC\_APB1ENR

Thanh ghi này có nhiệm vụ kích hoạt clock cho các ngoại vi được nối với bus điều khiển ngoại vi APB1 như: DAC, CAN, USB, Timer,...

2.3.2 Thư viện stm32f10x\_rcc.h

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_rcc.h** | |
| **Thông số** | **Giải thích** |
| **RCC\_APB2PeriphClockCmd(A,B);**  (Cho phép hoặc không cho phép clock tốc độ cao của ngoại vi **A** hoạt động.) | |
| **A**:  RCC\_APB2Periph\_AFIO  RCC\_APB2Periph\_GPIOA  RCC\_APB2Periph\_ADC1  …  **B**:  ENABLE  DISABLE | **A**:  Cấp xung clock cho AFIO  Cấp xung clock cho GPIOA  Cấp xung clock cho ADC1  …  **B**:  Cho phép  Không cho phép |
| **RCC\_APB1PeriphClockCmd(A,B);**  (Cho phép hoặc không cho phép clock tốc độ thấp của ngoại vi **A** hoạt động.) | |
| **A**:  RCC\_APB2Periph\_I2C1  RCC\_APB2Periph\_USART2  RCC\_APB2Periph\_SPI2  …  **B**:  ENABLE  DISABLE | **A**:  Cấp xung clock cho I2C1  Cấp xung clock cho USART2  Cấp xung clock cho SPI 2  …  **B**:  Cho phép  Không cho phép |
| **RCC\_AHBPeriphClockCmd(A,B);**  (Cho phép hoặc không cho phép clock của ngoại vi **A** hoạt động.) | |
| A:  RCC\_AHBPeriph\_DMA1  RCC\_AHBPeriph\_SRAM  RCC\_AHBPeriph\_CRC  …  B:  ENABLE  DISABLE | A:  Cấp xung clock cho DMA1  Cấp xung clock cho SRAM  Cấp xung clock cho CRC  …  B:  Cho phép  Không cho phép |

**Ví dụ** để kích hoạt clock cho GPIOA của vi điều khiển chương trình phần mềm cần thực hiện lệnh sau:

Cách 1: sử dụng thư viện stm32f10x\_rcc.h

|  |
| --- |
| **RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);** |

Cách 2: sử dụng thanh ghi RCC

|  |
| --- |
| **RCC -> APB2ENR = 0x0004;**  // xem chi tiết tại hình 2.3 thanh ghi RCC\_APB2ENR |

2.4 Lập trình GPIO

2.4.1 Giới thiệu GPIO

**GPIO (General-purpose input/output):** ngõ vào -ngõ ra sử dụng chung.

* Đối với các dòng STM32 thì mỗi PORT có 16 chân I/O.
* Ngoài chức năng là I/O nếu muốn sử dụng các chân này làm ngõ vào hoặc ngõ ra của
* ngoại vi ta phải thiết lập chân này là chức năng thay thế (Alternate Function) hoặc
* Analog.
* Để tiện cho thiết kế phần cứng nhà sản xuất xây dựng chức năng remap cho phép người
* dùng có thể thay đổi vị trí I/O của ngoại vi trong một phạm vi nhất định .
* Mỗi chân I/O bên trong chip đều được gắn điện trở nội kéo lên và kéo xuống.

2.4.2 Các thanh ghi và lệnh liên quan đến GPIO

Mỗi bộ GPIO của STM32F1 có 2 thanh ghi cấu hình (GPIOx\_CRL, GPIOx\_CRH), 2 thanh ghi dữ liệu (GPIOx\_IDR, GPIOx\_ODR), một thanh ghi Lập/Xóa ((GPIOx\_BSRR), một thanh ghi (GPIOx\_BRR) và một thanh ghi khóa (GPIOx\_LCKR).

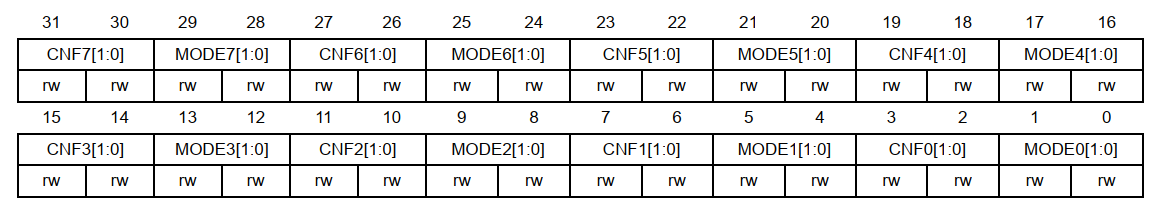
Mỗi bộ GPIO hỗ trợ điều khiển tối đa 16 chân GPIO. Số lượng chân của mỗi GPIO trên mỗi IC phụ thuộc vào số lượng cổng mà IC đó hỗ trợ. Các chân này có thể cấu hình độc lập với các chức năng hoạt động khác nhau. Danh sách các chế độ hoạt động của mỗi cổng như sau:

* **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 hoặc 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, còn khi output control = 1 thì P-MOS và N-MOS đều không dẫn, chân I/O được để nổi.
* **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 open-drain:** 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 open-drain. goài ra nó còn để sử dụng cho chức năng remap.

**a. Hai thanh ghi 32 bit GPIOx\_CRL và GPIOx\_CRH**

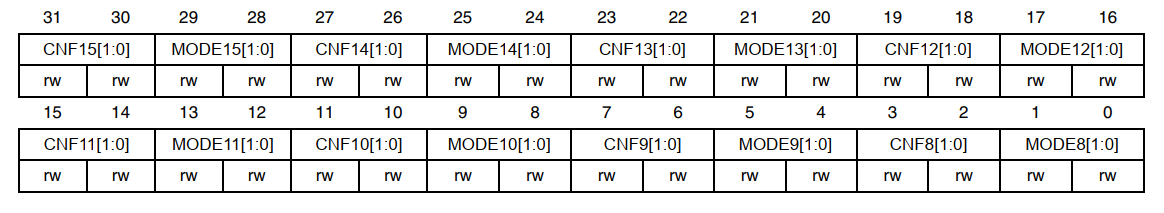
Mỗi GPIO của STM32F1 hỗ trợ điều khiển 16 cổng (0-15) thông qua giao diện thanh ghi 32 bit. Việc cấu hình hoạt động cho mỗi cổng của IC cần thông qua 4 bit điều khiển CNF[1:0] và MODE [1:0]. Do vậy thanh ghi cấu hình hoạt động cho cả bộ GPIO cần chia làm 2 thanh ghi: Một thanh ghi điều khiển chế độ của các cổng từ 0 đến 7 và một thanh ghi điều khiển chế độ hoạt động của các cổng từ 8 đến 15.

**GPIOx\_CRL:** dùng để cấu hình cho các chân 0 đến 7 của Portx

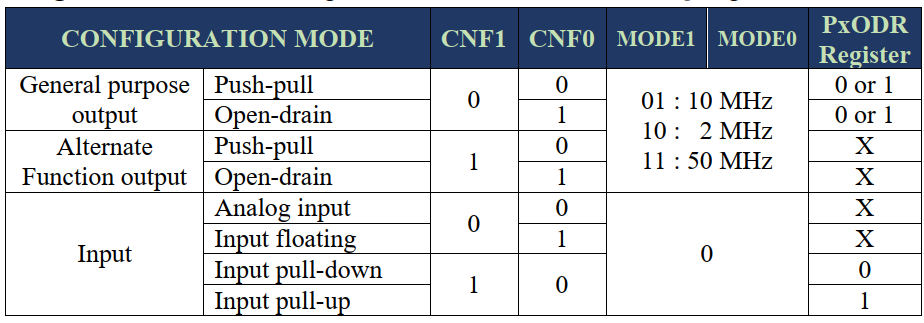


Hình 2.5: thanh ghi GPIOx\_CRL(P5rt configuration register low).

**GPIOx\_CRH:** dùng để cấu hình cho các chân 8 đến 15 của Portx



Hình 2.6: thanh ghi GPIOx\_CRH(Port configuration register high).



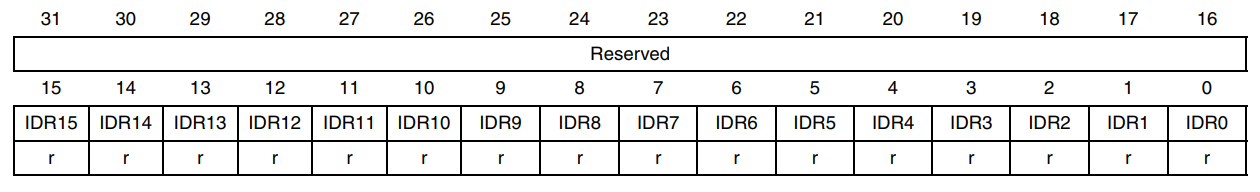
Hình 2.7: bảng các thông số cấu hình

Theo Hình 2.7 ta chú ý đến hai thông số CNF và MODE trong đó 2 bit CNF[1:0] dùng để lựa chọn chế độ và 2 bit MODE[1:0] dùng để lựa chọn tốc độ.

**b. Thanh ghi GPIOx\_IDR**

Thanh ghi này là thanh ghi chỉ đọc, dùng để đọc dữ liệu đầu vào từ các GPIO tương ứng.

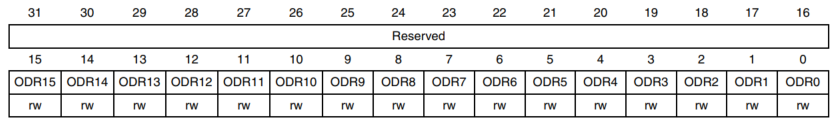
Mỗi bit IDR tương ứng đọc giá trị đầu vào của chân tương ứng. Thanh ghi này thường được sử dụng khi cần đọc dữ liệu đầu vào số của một cổng nào đó.



Hình 2.8: Thanh ghi GPIOx\_IDR(Port Input Data Register).

Thanh ghi này có thể truy cập đọc và ghi có chức năng dùng để điều khiển đầu ra số tại các chân đầu ra tương ứng. Khi tác động đến thanh ghi này các chân được cài đặt ở chế độ đầu ra tương ứng sẽ bị ảnh hưởng.

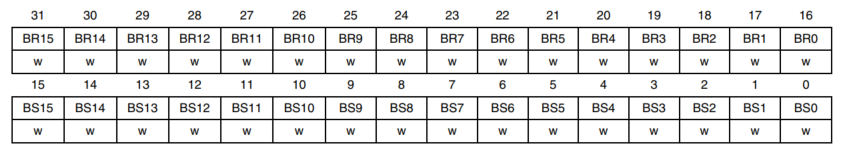
**c. Thanh ghi GPIOx\_ODR**



Hình 2.9: Thanh ghi GPIOx\_ODR(Port Output Data Register).

Thanh ghi này có thể truy cập đọc và ghi có chức năng dùng để điều khiển đầu ra số tại các chân đầu ra tương ứng. Khi tác động đến thanh ghi này các chân được cài đặt ở chế độ đầu ra tương ứng sẽ bị ảnh hưởng.

**d. Thanh ghi GPIOx\_BSRR**

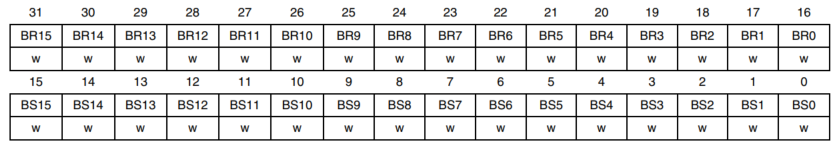


Hình 2.10: Thanh ghi GPIOx\_BSRR(Port Bit Set/Reset Register)

Thanh ghi này là thanh ghi chỉ ghi và không thể đọc về. Trong đó thanh ghi được chia làm 2 nửa từ BS0-BS15 và BR0-BR15.

BS0-BS15: Nếu ghi giá trị 0 vào các bit tương ứng này thì đầu ra của chân tương ứng không thay đổi, nếu ghi giá trị 1 vào các bit tương ứng thì bit tương ứng của thanh ghi ODR được lập lên 1 (Tức là điều khiển cổng tương ứng lên mức đầu ra là 1 – 3,3V). BR0-BR15: Nếu ghi giá trị 0 vào các bit tương ứng này thì đầu ra của chân tương ứng không thay đổi, nếu ghi giá trị 1 vào các bit tương ứng thì bit tương ứng của thanh ghi ODR được xóa về 0 (Tức là điều khiển cổng tương ứng với mức đầu ra là 0 – 3.3V).

**e. Thanh ghi GPIOx\_BRR**



Hình 2.11: Thanh ghi GPIOx\_BRR(Port Bit Reset Register)

Thanh ghi này chứa 16 bit có thể truy cập dưới dạng chỉ ghi có chức năng tương đương như các bit từ BR0-BR15 của thanh ghi BSRR.

**f. Các tập lệnh thông dụng để cấu hình GPIO**

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_gpio.h** | |
| **Thông số** | **Giải thích** |
| **GPIO\_InitTypedef X;**  (Khai báo biến X thuộc kiểu GPIO\_InitTypedef) | |
| **X.GPIO\_Pin = A;**  (Lệnh chọn chân cấu hình) | |
| **A:**  GPIO\_Pin\_0  GPIO\_Pin\_1  …  GPIO\_Pin\_15  GPIO\_Pin\_All | **A:**  Chọn chân 0  Chọn chân 1  …  Chọn chân 15  Chọn tất cả các chân 0-15  Chú ý: Nếu muốn chọn nhiều chân thì ta sử dụng lệnh OR “|” giữa các lựa chọn  Ví dụ: **X.GPIO\_Pin** = GPIO\_Pin\_0|GPIO\_Pin\_3; |
| **X.GPIO\_Mode = B;**  ( Lệnh cấu hình chế độ hoạt động) | |
| **B**:GPIO\_Mode\_AIN GPIO\_Mode\_IN\_FLOATING GPIO\_Mode\_IPD GPIO\_Mode\_IPU GPIO\_Mode\_Out\_OD GPIO\_Mode\_Out\_PP GPIO\_Mode\_AF\_OD GPIO\_Mode\_AF\_PP | **B:** Cấu hình chế độ hoạt động  Ngõ vào tương tự  Ngõ vào thả nổi  Ngõ vào kéo xuống  Ngõ vào kéo lên  Ngõ ra cực thu hở  Ngõ ra đẩy kéo  Chức năng thay thế cực thu hở  Chức năng thay thế đẩy kéo |
| **X.GPIO\_Speed = C;**  ( Lệnh cấu hình tốc độ) | |
| **C:** GPIO\_Speed\_10MHz GPIO\_Speed\_2MHz GPIO\_Speed\_50MHz | **C:** GPIO\_Speed\_10MHz GPIO\_Speed\_2MHz GPIO\_Speed\_50MHz |
| **GPIO\_Init(GPIOx,&X);** ( Lệnh cấu hình cho GPIOx theo các thông số được lưu trong biến **X**) | |
| **uint16\_t A;**  **A = GPIO\_ReadInputData(B);**  (Lệnh đọc trạng thái logic 16 chân của 1 PORT rổi lưu vào biến **A**) | |
| **B**:  GPIOA  GPIOB  … | **B**: PORT cần đọc dữ liệu  Đọc PORTA  Đọc PORTB  … |

|  |  |
| --- | --- |
| **GPIO\_Write(A, B);**  (Lệnh xuất dữ liệu ra 16 chân của PORT) | |
| **A**:  GPIOA  GPIOB  …  GPIOG | **A**: PORT cần xuất dữ liệu  Xuất ra PORTA  Xuất ra PORTB  …  Xuất ra PORTG  **B**: Số nguyên 16 bit ứng với trạng thái 16  chân của PORT cần xuất. |
| **GPIO\_SetBits(A, B);**  (Lệnh xuất mức ‘1’ ra 1 chân vi điều khiển) | |
| **A**:GPIOA … GPIOG **B**:GPIO\_Pin\_0 … GPIO\_Pin\_15 PIO\_Pin\_All | **A**: PORT cần xuất  Xuất ra PORTA  …  Xuất ra PORTG  **B**: Chân cần xuất  Chân số 0  …  Chân số 15  Tất cả các chân của PORT |
| **GPIO\_ResetBits(A, B);**  (Lệnh xuất mức ‘0’ ra 1 chân vi điều khiển) | |
| **A**:GPIOA … GPIOG **B**:GPIO\_Pin\_0 … GPIO\_Pin\_15 GPIO\_Pin\_All | **A**: PORT cần xuất  Xuất ra PORTA  …  Xuất ra PORTG  **B**: Chân cần xuất  Chân số 0  …  Chân số 15  Tất cả các chân của PORT |
| **BitAction C;**  **GPIO\_WriteBit(A, B, C);**  (Lệnh xuất mức logic của biến C ra 1 chân vi điều khiển) | |
| **A**:GPIOA … GPIOG **B**:GPIO\_Pin\_0 … GPIO\_Pin\_15 GPIO\_Pin\_All **C**:Bit\_RESET Bit\_SET | **A**: PORT cần xuất  Xuất ra PORTA  …  Xuất ra PORTG  **B**: Chân cần xuất  Chân số 0  …  Chân số 15  Tất cả các chân của PORT  **C**: Mức logic cần xuất  Mức ‘0’  Mức ‘1 |

|  |  |
| --- | --- |
| **Sử Dụng Thanh Ghi** | |
| **Thông số** | **Giải thích** |
| **uint16\_t A;**  **A= P->IDR**  (Lệnh đọc trạng thái logic 16 chân của 1 PORT rổi lưu vào biến A) | |
| **P**:GPIOA GPIOB … | **P**: PORT cần đọc dữ liệu  Đọc PORTA  Đọc PORTB  …  **A**: Biến 16 bit để lưu trạng thái logic 16 chân của PORT được đọc |
| **P->ODR =A;**  ( Lệnh xuất dữ liệu ra 16 chân của PORT) | |
| **P**:GPIOA GPIOB … | **P**: PORT cần đọc dữ liệu  Đọc PORTA  Đọc PORTB  …  **A**: Biến 16 bit để lưu trạng thái logic 16 chân của PORT được đọc |
| **P->BSRR = A;**  (Lệnh xuất mức ‘1’ hoặc ‘0’ ra chân vi điều khiển) | |
| **P**:GPIOA … GPIOG **A:** Số 32 bit | **P:** PORT cần xuất  Xuất ra PORTA  …  Xuất ra PORTG  **A:** Chân cần xuất  **Ví dụ:**  chân số 5 và 10 của PORTC là mức ‘1’  **A** = 0000 0100 0010 0000B  **A** = 0x00000420  => **GPIOC->BSRR =0x00000420;** |
| **P->BRR = A;**  (Lệnh xuất mức ‘0’ ra chân vi điều khiển) | |
| **P**:GPIOA … GPIOG **A**:[0-65535] | **P:** PORT cần xuất  Xuất ra PORTA  …  Xuất ra PORTG  **A:** Chân cần xuất  **Ví dụ:**  chân số 5 và 10 của PORTC là mức ‘0’  **A** = 0000 0100 0010 0000B = 0x0420  **GPIOC->BRR =0x0420**; |

2.4.3 Chức năng thay thế và thay đổi vị trí các I/O (Alternate function and remap I/O)

Do mỗi chân vi điều khiển có nhiều chức năng nên khi muốn sử dụng chức năng khác với mặc định thì ta phải cấu hình cho chân đó là chức năng thay thế( Alternate Function) hoặc Analog.

Khi được thiết lập là là chức năng thay thế, các chân này được gọi là AFIO( Alternate Function I/O). Để các AFIO có thể hoạt động được ta cần phải cấp xung clock bằng lệnh sau:

|  |
| --- |
| **RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO,ENABLE);** |

Để tiện cho thiết kế phần cứng nhà thiết kế cho phép ta có thể thay đổi vị trí các ngõ vào ra của ngoại vi trong một phạm vi nhất định, thao tác này gọi là “**Remap**”.

2.4.4 Sử dụng các chân đặc biệt làm chân I/O

**a. Sử dụng hai chân OSC32\_IN( PC14)/OSC32\_OUT(PC15)**

Hai chân này được dùng để nối thạch anh ngoài 32.768 Khz cho LSE.

Hai chân OSC32\_IN và OSC32\_OUT có thể được dùng như là các chân GPIO thông thường nếu ta tắt LSE. Khi bật LSE thì hai chân này trở thành chân để kết nối thạch anh 32.768 Khz.

**b. Sử dụng hai chân OSC\_IN(PD0)/OSC\_OUT(PD1)**

Hai chân này được dùng để kết nối thạch anh ngoài – thường chọn 8 Mhz- cho HSE

Việc tận dụng hai chân OSC\_IN(PD0)/OSC\_OUT(PD1) làm GPIO chỉ có thể thực hiện được trên các dòng ARM ít chân (36, 48, 64 chân) còn các dòng nhiều chân hơn thì hai chân PD0 và PD1 nằm tách biệt so với OSC\_IN và OSC\_OUT nên không cần remap.

Chú ý: Khi remap hai chân OSC\_IN và OSC\_OUT thành GPIO thì chúng không có khả năng tạo ngắt ngoài như các GPIO khác.

Đối với các dòng ARM ít chân hai chân PD0 và PD1 mặc định là ngõ kết nối thạch anh để remap chúng thành GPIO ta dùng lệnh sau:

|  |
| --- |
| **GPIO\_PinRemapConfig(GPIO\_Remap\_PD01, ENABLE);** |

**c. Sử dụng các chân JTDI(PA15), JTDO(PB3) và JNTRST(PB4)**

Các chân này kết nối với mạch nạp JTAG để nạp và debug chương trình.

Mặc định các chân này được dùng làm chân để nạp chương trình và debug qua cổng JTAG nhưng nếu như ta nạp và debug theo chuẩn SWD( Serial Wire Debug ) thì ta có thể tận dụng các chân này làm GPIO bằng lệnh sau:

|  |
| --- |
| **GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);** |

Chú ý : Sau khi remap các chân này thành GPIO thì khi nạp chương trình ta cần phải chuyển chế độ nạp sang SWD vì lúc này chuẩn JTAG không còn nạp được nữa.

**Ví dụ:** cấu hình chân A0-A3 là ngõ ra đẩy kéo tốc độ 50 Mhz.

**Cách 1: Sử dụng thư viện stm32f10x\_gpio.h**

|  |
| --- |
| GPIO\_InitTypeDef GPIO\_Structure;  GPIO\_Structure.GPIO\_Pin = GPIO\_Pin\_0|GPIO\_Pin\_1|GPIO\_Pin\_2|GPIO\_Pin\_3;  // Chọn chân số 0-3  GPIO\_Structure.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  // Chọn chế độ ngõ ra đẩy kéo  GPIO\_Structure.GPIO\_Speed = GPIO\_Speed\_50MHz;  // Chọn tốc độ 50 Mhz  GPIO\_Init(GPIOA,&GPIO\_Structure );  // Cài đặt các cấu hình trên cho PORTA |

**Cách 2: Sử dụng thanh ghi**

|  |
| --- |
| /\* Chân A0 được cấu hình bởi GPIOA\_CRH  Tốc độ 50 Mhz => MODE0[1:0]=11  Ngõ ra đẩy kéo => CNF8\0[1:0]=00  CNF0[1:0]MODE0[1:0] = 0011= 3  Tương tự với các chân còn lại \*/  GPIOA->CRL = 0x00003333; // Cấu hình cho chân A0…A3 |

2.4.5 Lập trình LED

**a. Cấu tạo**



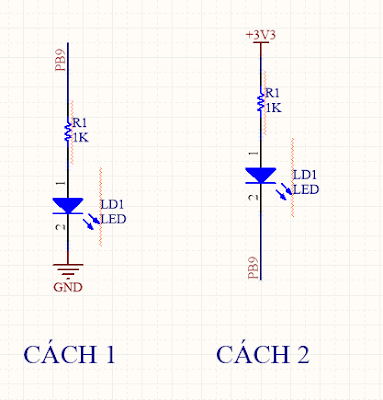
Led đơn với 2 cực **Anode** và **Cathode** trong đó:

**Anode:** cực dương (chân dài, bản cực nhỏ)

**Cathode:** cực âm (chân ngắn, bản cực lớn)

*Led đơn 5mm*

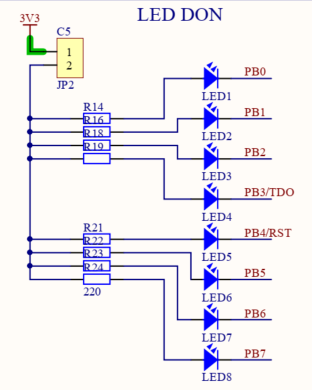
**b. Nguyên lý**



|  |  |
| --- | --- |
| **Cách 1** | **Cách 2** |
| Mức logic 0: LED OFF  Mức logic 1: LED ON | Mức logic 0: LED ON  Mức logic 1: LED OFF |

*Sơ đồ kết nối led đơn vói vi điều khiển*

|  |
| --- |
| Từ sơ đồ trên ta có thế nhận ra led được kết nối theo cách 2 và kết nối với GPIOB(B0-B7). Ta chú ý là có 2 chân B3 và B4 là 2 chân hỗ trợ cho mạch nạp JTAG. Để sử dụng B3, B4 như I/O bình thường thì ta cần phải **remap** các chân này thành GPIO theo mục 2.4.4 phần c. |



*Sơ đồ kết led đơn trên kit.*

**Ví dụ:** Lập trình điều khiển 8 led đơn( led đơn kết nối với port B, pin 0-7 ) sáng tắt với tần số 1Hz.

Phương pháp chung:

* Khai báo các thư viện sử dụng.
* Viết các hàm với các chức năng: Delay\_ms, cấu hình I/O( cấp xung clock, cấu hình chân), hàm xử lý.

**Cách 1: sử dụng thanh ghi GPIOx\_ODR**

|  |
| --- |
| #include "stm32f10x.h"  #include "stm32f10x\_gpio.h"  #include "stm32f10x\_rcc.h"  void Delay\_ms(uint16\_t \_time);  void led\_Init(void);  void led\_Blink(void);  int main(){  led\_Init();  while(1){  led\_Blink(5);  }  }  void Delay\_ms(uint16\_t \_time){  uint16\_t i,j;  for(i = 0; i < \_time; i++){  for(j = 0; j < 0x2AFF; j++);  }  }  void led\_Init(void){  // cấp xung clock cho GPIOB và chức năng AFIO  RCC->APB2ENR = 0x0009;  // cấu hình chân A0…A7 là ngõ ra, tốc độ 50MHz  GPIOB->CRL = 0x33333333;  }  void led\_Blink(void){  GPIOB->ODR = 0x00; //xuất mức 0 ra các chân A0…A7(LED ON)  Delay\_ms(1000);  GPIOB->ODR = 0xFF; //xuất mức 1 ra các chân A0…A7(LED OFF )  Delay\_ms(1000);  } |

**Cách 2: Sử dụng thư viện stm32f10x\_gpio.h**

#include "stm32f10x.h"

#include "stm32f10x\_gpio.h"

#include "stm32f10x\_rcc.h"

void Delay\_ms(uint16\_t \_time);

void led\_Init(void);

void led\_Blink(int n);

int main(){

led\_Init();

while(1){

led\_Blink(5);

}

}

void Delay\_ms(uint16\_t \_time){

uint16\_t i,j;

for(i = 0; i < \_time; i++){

for(j = 0; j < 0x2AFF; j++);

}

}

void led\_Init(void){

GPIO\_InitTypeDef led\_init;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);

GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);

led\_init.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

led\_init.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;

led\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_Init(GPIOB,&led\_init);

}

|  |
| --- |
| void led\_Blink(void){  //xuất mức ”0“ ra các chân vđk -> LED ON  GPIO\_ResetBits(GPIOB,GPIO\_Pin\_0|GPIO\_Pin\_1|GPIO\_Pin\_2|GPIO\_Pin\_3|GPIO\_Pin\_4|GPIO\_Pin\_5|GPIO\_Pin\_6|GPIO\_Pin\_7);  Delay\_ms(1000);  //xuất mức ”1“ ra các chân vđk -> LED OFF  GPIO\_SetBits(GPIOB,GPIO\_Pin\_0|GPIO\_Pin\_1|GPIO\_Pin\_2|GPIO\_Pin\_3|GPIO\_Pin\_4|GPIO\_Pin\_5|GPIO\_Pin\_6|GPIO\_Pin\_7);  Delay\_ms(1000);  } |

Giải thích:

|  |
| --- |
| #include "stm32f10x.h" // Device header  #include "stm32f10x\_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO  #include "stm32f10x\_rcc.h" // Keil::Device:StdPeriph Drivers:RCC  void Delay\_ms(uint16\_t \_time);  void led\_Init(void);  void led\_Blink(void); |

Khai báo các thư viện sử dụng:

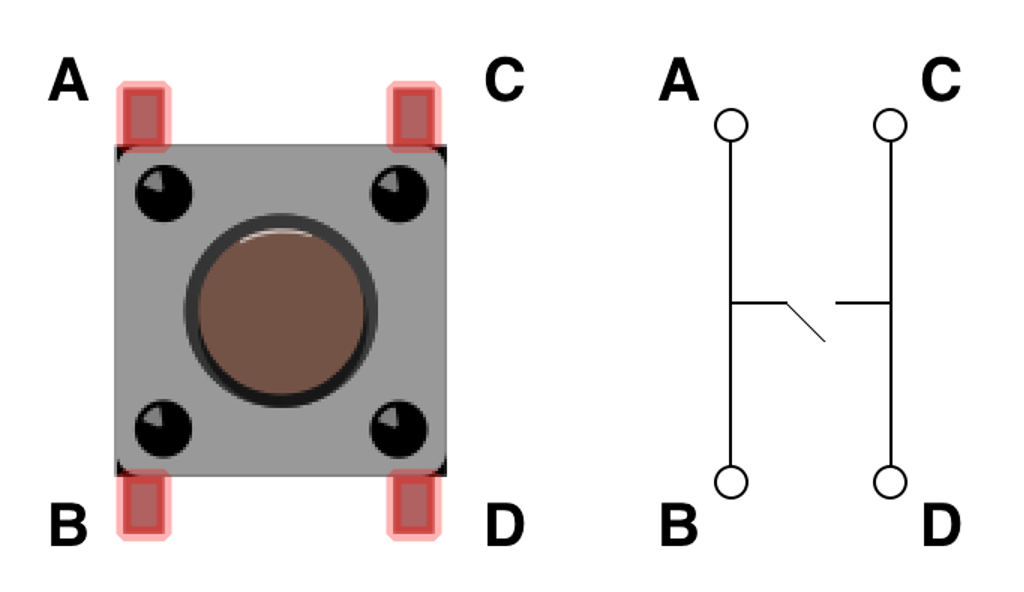
* Stm32f10x.h
* Stm32f10x\_gpio.h
* Stm32f10x\_rcc.h

Khởi tạo các hàm chức năng:

* Delay\_ms(unit16\_t \_time): hàm tạo độ chễ ms;
* led\_Init(void): hàm khởi tạo các chân I/O
* cấp Clock cho ngoại vi
* cấu hình gpio
* led\_Blink(int n): hàm thuật toán, nhấp nháy led với tần số 1Hz

2.4.6 Lập trình BUTTON

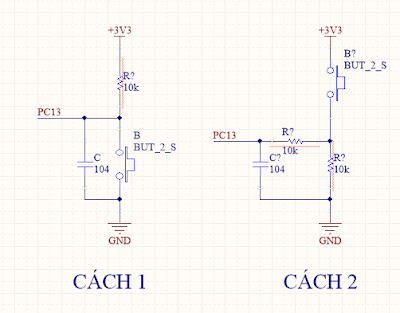
**a. Cấu tạo**



*Cấu tạo nút nhấn.*

**b. Nguyên lý**

Các cách kết nói nút nhấn với vi điều khiển:



*Sơ đồ kết nối nút nhán với vi điều khiển*

|  |  |  |
| --- | --- | --- |
|  | **Cách 1** | **Cách 2** |
| **Đặc điểm** | Tích cực mức 1  Mức logic 0: nhấn nút  Mức logic 1: nhả nút | Tích cực mức 0  Mức logic 0: nhả nút  Mức logic 1: nhấn nút |

**c. Lập trình**

**Ví dụ:** lập trình nút nhấn với yêu cầu nhấn nút thì đèn led sáng và nhả ra thì đèn led tắt( nút nhấn được kết nối với chân A11 tích cực mức thấp và 8 led đơn kết nối với B0-B7 tích cực mức thấp).

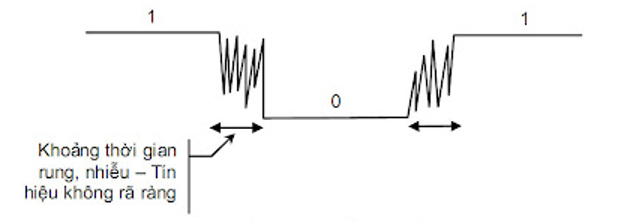
cách làm:

* đọc tín hiệu ngõ vào sử dụng hàm:
* GPIO\_ReadInputDataBit(GPIOx,GPIO\_Pin\_x);
* Cấp clock và cấu hình chân cho nút nhấn và led đơn.
* Viết hàm xử lý yêu câu bài toán.

|  |
| --- |
| void led\_Init(void){  GPIO\_InitTypeDef led\_init;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);  GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);  led\_init.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  led\_init.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;  led\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOB,&led\_init);  GPIOB->ODR = 0xFF;  }  void button\_Init(void){  GPIO\_InitTypeDef button\_init;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);  GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);    button\_init.GPIO\_Mode = GPIO\_Mode\_IPU;  button\_init.GPIO\_Pin = GPIO\_Pin\_11;  button\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;    GPIO\_Init(GPIOA,&button\_init);  }  void button\_Press(void){  if(!GPIO\_ReadInputDataBit(GPIOA,GPIO\_Pin\_11)){  GPIOB->ODR = 0x00;  }  else{  GPIOB->ODR = 0xFF;  }  } |

Link tải code mẫu: https://drive.google.com/drive/folders/1gKO9csTeUp25MKFIw-JtWLzavuHNhDfd?usp=sharing

Vi dụ trên là cách sử dụng cơ bản với nút nhấn nhưng trong quá trình nhấn nút thì chúng ta sẽ nhận thấy là có lúc mạch chạy không đúng và có độ trễ, đó là do tín hiệu bị nhiễu.



*Tín hiệu rung nhiễu khi nhấn nút*

hình ảnh bên trên là quá trình rung nhiễu khi ta nhấn nút, tức là khi tín hiệu chuyển từ mức logic cao xuống mức logic thấp hoặc từ mức logic thấp lên mức logic cao thì tín hiệu sẽ thay đổi liên tục giữa mức cao với mức thấp, quá trình đó xảy ra trong khoảng 20ms sau đó tín hiệu mới ở mức logic ổn định.

Để khắc phục hiện tượng rung phím(nhiễu) chúng ta có một số cách khắc phục:

* Chống nhiễu bằng phần cứng, như sơ đồ kết nối nút nhấn ở trên chúng ta có sử dụng tụ 104uf để lọc tín hiệu.
* Chống nhiễu bằng phần mềm tức là chúng ta xử lý quá trình rung phím bằng code.

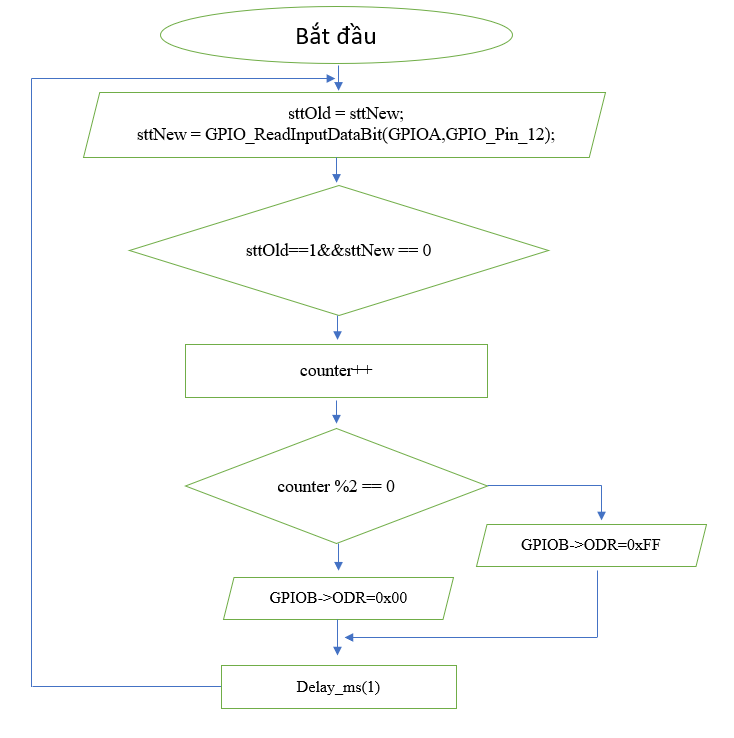
Chống nhiễu bằng phần mềm:

|  |
| --- |
| uint8\_t sttOld = 1, sttNew = 1;  uint16\_t counter = 0;  void button\_PressHold(void){  sttOld = sttNew;  sttNew = GPIO\_ReadInputDataBit(GPIOA,GPIO\_Pin\_12);  if(sttOld == 1 && sttNew == 0){  counter++;  }  if(counter%2==0){  GPIOB->ODR = 0xff;  }  else{  GPIOB->ODR = 0x00;  }  Delay\_ms(1);  } |

Giải thích:

* Hàm button\_PressHold() có chứa năng là nhấn nút thì đảo trạng thái của led có sử dụng chống nhiễu(mức cơ bản và đơn giản).
* Chống nhiễu: cứ mỗi 1ms chúng ta sẽ kiểm tra tin hiệu ngõ vào nếu trạng thái trước(sttOld) ở mức cao và trạng thái sau đó(sttNew) ở mức thấp thì đã có hiện tượng nhấn phím. Chi tiết xem tại lưu đồ thuật toán ở bên dưới.

Lưu đồ thuật toán:

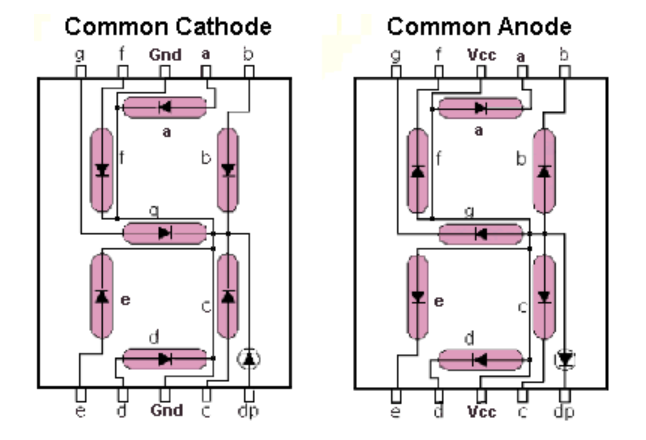


Link tải code mẫu:

https://drive.google.com/drive/folders/1r-x5Y66jDwoBrWnRlpYDO2wY10wbp6HS?usp=sharing

2.4.7 Lập trình LED 7 Thanh

**a. Cấu tạo**

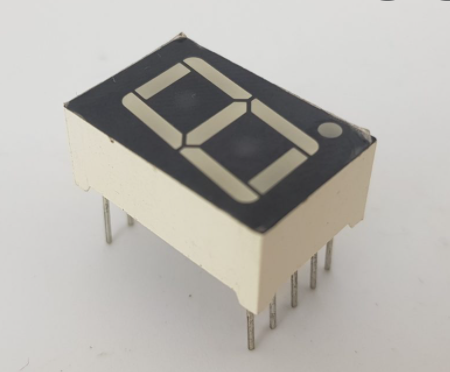


Cấu tạo của led 7 thanh gồm 8 led đơn được kết nối với nhau trong đó có các led thanh được kí hiệu là a,b,c,d,e,f,g; led ‘dấu chấm’ là dp và 2 chân GND đối với loại Cathode vs 2 chân VCC đối với loại Anode.

Led 7 thanh có hai loại:

* Led 7 thanh Anode chung: các chân dương của led nối chung với nhau và kéo lên VCC
* Led 7 thanh Cathode chung: các chân âm của led nối chung với nhau và kéo xuống GND

Do led 7 thanh được cấu tạo từ những led đơn nên cách kết nối và xuất các mứa logic giống với led đơn.

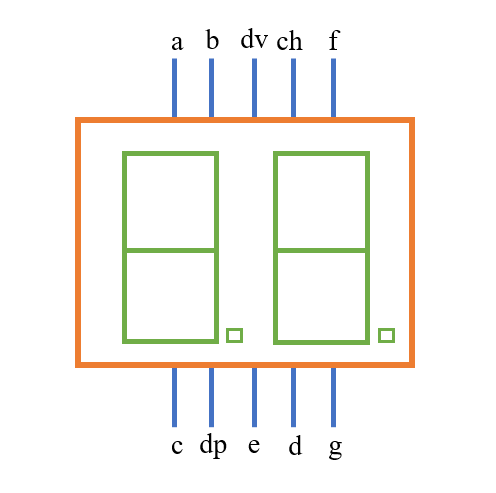
   
*Led 7 thanh loại 1 số.*  *Led 7 thanh loại 2 số*

*Led 7 thanh 4 số*  *led 7 thanh 3 số*

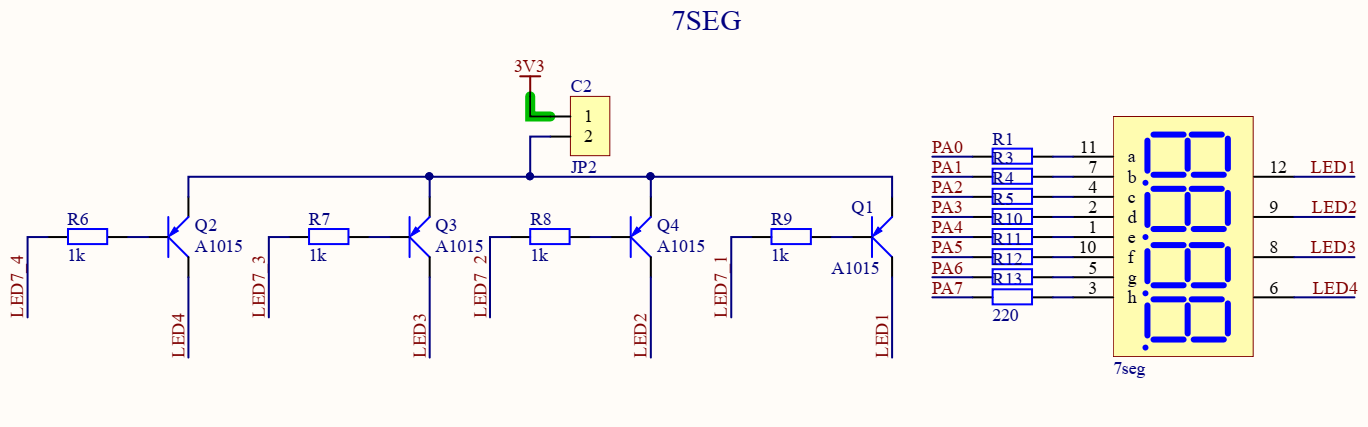
Trên đấy là một số hình ảnh về led 7 thanh, mỗi loại sẽ có số chân khác nhau, để xem chi tiết thì chúng ta sẽ google tra datasheet của chúng trước khi bắt tay vào lập trình.

**b. Nguyên lý**



*Sơ đồ chân led 7 thanh 2 số*

Led 7 thanh 2 sô được ghép bởi 2 led 7 thanh 1 số, có 10 chân điều khiển, trong đó có 8 chân a,b,c,d,e,f,g,dp là các chân hiển thị của led 7 thanh và 2 chân điều khiển bật tắt là dv(điều khiển led đơn vị), ch (điều khiển led hàng chục).



*Sơ đồ nguyên lý*

Theo sơ đồ nguyên lý thí trên kít sử dụng led 7 thanh 4 số và là loại Anode chung. Các chân a,b,…,dp được kéo với điện trở để hạn dòng cho led, các chân điều khiển LED1,.., LED4 được thông qua transistor đùng để đệm dòng và điều khiển bật tắt.

**b. Lập trình**

cách lấy mã Hex cho các số:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mã LED 7 Thanh Anode(1: OFF/ 0: ON)** | | | | | | | | | |
| **Num** | **dp** | **g** | **f** | **e** | **d** | **c** | **b** | **a** | **Hex** |
| **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | **0xC0** |
| **1** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | **0xF9** |
| **2** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | **0xA4** |
| **3** | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | **0xB0** |
| **4** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | **0x99** |
| **5** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | **0x92** |
| **6** | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | **0x82** |
| **7** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | **0xF8** |
| **8** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0x80** |
| **9** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | **0x90** |

*mã các sô cho led 7 thanh loại anode*

**Ví dụ:** Lập trình hiển thị các số từ 0-9 trên led 7 thanh, các chân dc kết nối với A0-A7.

Cách làm:

* Tạo mảng để lưu mã các số từ 0-9:

uint8\_t Num[10]={0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

* Viết các hàm cấu hình: void seg\_Init(void);
* Viết hàm xử lý yêu cầu bài toán: void seg\_scan(void);
* Tạo độ trễ để quan sát được sự thay đổi: void Delay\_ms(unit16\_t \_time);

|  |
| --- |
| #include "stm32f10x.h"  #include "stm32f10x\_gpio.h"  #include "stm32f10x\_rcc.h"  uint8\_t Num[10]={0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};  void Delay\_ms(uint16\_t \_time);  void seg\_Init(void);  void seg\_Scan(void);  int main(void){  seg\_Init();  while(1){  seg\_Scan();  }  }  void Delay\_ms(uint16\_t \_time){  uint16\_t i,j;  for(i = 0; i < \_time; i++){  for(j = 0; j < 0x2AFF; j++);  }  }  void seg\_Init(void){    GPIO\_InitTypeDef led\_init;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);    led\_init.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  led\_init.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;  led\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;    GPIO\_Init(GPIOA,&led\_init);    GPIOA->ODR = 0XFF;  }  void seg\_Scan(void){  uint8\_t i = 0;  for(i = 0; i < 10; i++){  GPIOA->ODR = Num[i];  Delay\_ms(1000);  }  } |

**Ví dụ:** Lập trình hiển thị led 7 thanh 0000-9999 trong đó led a,b,…,dp kết nối với chân A0-A7; 4 chân điều khiển kết nối với B12-B15.

Cách làm:

* Tạo mảng để lưu mã của các số 0-9:
* uint8\_t Num[10]={0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
* Viết các hàm khởi tạo và cấu hình chân led và chân điều khiển:

void seg\_Init(void); void control\_Init(void);

* Viết hàm xử lý yêu cầu bài toán: void seg\_Disp(void);
* Hàm tạo độ trễ: void Delay\_ms(uint16\_t \_time);

Để hiển thị led từ 0000-9999 thì chúng ta phải sử dụng phương pháp quét led.

Phương pháp quét led: phương pháp quét Led dựa trên sự lưu ảnh của mắt người. Mắt người phải mất khoảng 25 ms để xử lý một hình ảnh (40 hình/s), vậy nếu cho Led sáng tắt với khoảng thời gian nhỏ hơn 25 ms hoặc nói cách khác thì số hình quét được trong 1s lớn hơn 40 hình thì giá trị hiển thị trên Led giống như luôn sáng.

* Bật led hàng đơn vị -> hiển thị số -> Delay\_ms(1) -> tắt led hàng đơn vị;
* Bật led hàng chục-> hiển thị số -> Delay\_ms(1) -> tắt led hàng chục;
* Bật led hàng trăm-> hiển thị số -> Delay\_ms(1) -> tắt led hàng trăm;
* Bật led hàng nghìn-> hiển thị số -> Delay\_ms(1) -> tắt led hàng nghìn;

Như vậy thì 1 lần hiển thị sẽ mất 4ms -> quét được 250 hình/s > 40 hình/s

|  |
| --- |
| #include "stm32f10x.h"  #include "stm32f10x\_gpio.h"  #include "stm32f10x\_rcc.h"  /\* Định nghĩa các chân điều khiển led \*/  #define LED\_DONVI GPIO\_Pin\_15  #define LED\_CHUC GPIO\_Pin\_14  #define LED\_TRAM GPIO\_Pin\_13  #define LED\_NGHIN GPIO\_Pin\_12  uint8\_t Num[10]={0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};  void Delay\_ms(uint16\_t \_time);  void seg\_Init(void);  void control\_Init(void);  void seg\_Disp(void);  int main(){  seg\_Init();  control\_Init();  while(1){  seg\_Disp();  }  }  void Delay\_ms(uint16\_t \_time){  uint16\_t i,j;  for(i = 0; i < \_time; i++){  for(j = 0; j < 0x2AFF; j++);  }  }  void seg\_Init(void){  GPIO\_InitTypeDef led\_init;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  led\_init.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  led\_init.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;  led\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA,&led\_init);  GPIOA->ODR = 0XFF;  }  void control\_Init(void){  GPIO\_InitTypeDef control\_init;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);  control\_init.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  control\_init.GPIO\_Pin = LED\_NGHIN|LED\_TRAM|LED\_CHUC|LED\_DONVI;  control\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOB,& control\_init);  }  void seg\_Disp(void){  uint16\_t i,j;  for(i=0; i<10000; i++){ // đếm từ 0000 - 9999  for(j=0; j<125; j++){// quét 1 lần hết 4ms -> 125 \* 4 = 500ms  /\*quet led hang don vi\*/  GPIO\_ResetBits(GPIOB, LED\_DONVI);  GPIOA->ODR = Num[i%10];  Delay\_ms(1);  GPIO\_SetBits(GPIOB, LED\_DONVI);  /\*quet led hang chuc vi\*/  GPIO\_ResetBits(GPIOB, LED\_CHUC);  GPIOA->ODR = Num [(i/10)%10];  Delay\_ms(1);  GPIO\_SetBits(GPIOB, LED\_CHUC);  /\*quet led hang tram\*/  GPIO\_ResetBits(GPIOB, LED\_TRAM);  GPIOA->ODR = Num [(i/100)%10];  Delay\_ms(1);  GPIO\_SetBits(GPIOB, LED\_TRAM);  /\*quet led hang nghin\*/  GPIO\_ResetBits(GPIOB, LED\_NGHIN);  GPIOA->ODR = Num [i/1000];  Delay\_ms(1);  GPIO\_SetBits(GPIOB, LED\_NGHIN);  }  }  } |

2.5 Lập trình INTERRUPTS và EVENTS

2.5.1 Bộ điều khiển vector ngắt lồng nhau(NVIC-Nested Vector Interrupt Controller)

**a. Ngắt là gì?**

Ngắt là việc vi điều khiển dừng tạm thời chương trình đang thực thi mà đi thực thi một chương trình khác có yêu cầu cấp thiết hơn. Sau khi thực thi xong CPU sẽ quay trở về thực hiện lại chương trình cũ.

**b. NVIC là gì?**

NVIC là một khối nằm trong lõi Cortex có nhiệm vụ quản lý các nguồn ngắt, do có rất nhiều nguồn có khả năng yêu cầu CPU ngắt nên chúng cần có một **“người quản lý”** chung để sắp xếp, điều khiển mọi hoạt động ngắt được diễn ra đúng theo ý muốn tránh trường hợp chồng chéo, xung đột giữa các yêu cầu.

NVIC là một khối chuẩn trong lõi Cortex. Điều này có nghĩa là tất cả các vi điều khiển dựa trên lỗi Cortex sẽ có cùng một cấu trúc ngắt bất kể nhà sản xuất chip Atmel, Ti , ST... Nhờ vậy mà người lập trình có thể kế thừa chương trình của của hãng này cho chip hãng khác.

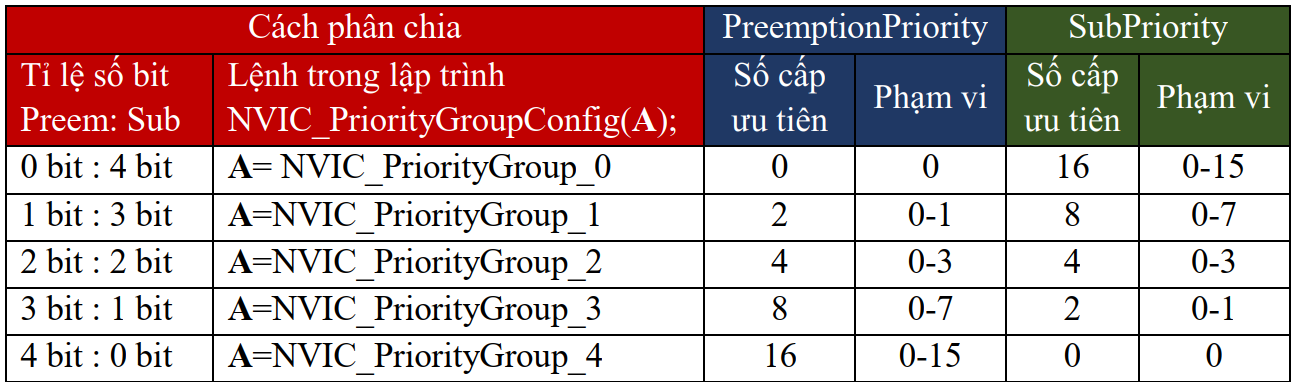
NVIC được thiết kế để điều khiển các nguồn ngắt lồng nhau và trên STM32 NVIC có:

* 16 cấp độ ưu tiên ngắt.
* 68 kênh ngắt có thể che được(maskable –có thể cho phép hoặc cấm được) – Chưa kể đến 16 đường ngắt của lõi Cortex™-M3.

2.5.2 Cơ chế ưu tiên ngắt

Khi có nhiều nguồn ngắt xảy ra cùng một lúc thì CPU sẽ xử lý yêu cầu của ngắt có độ ưu tiên cao nhất dần dần cho đến thấp nhất. Chỉ trừ một số ngắt không thể che thuộc về phần cứng thì thứ tự ưu tiên của các nguồn ngắt còn lại có thể được quy định bởi người lập trình.

Có 16 cấp độ ưu tiên ngắt( được điều khiển bằng 4 bit) và 4 bit này có thể được phân chia theo 5 tỉ lệ giữa nhóm ưu tiên chính (PreemptionPriority) và nhóm ưu tiên phụ (SubPriority) theo hình 2.12.



Hình 2.12: Bảng phân chia số cấp ưu tiên gữa 2 nhóm PreemptionPriority và SubPriority

**Chú ý:** Nhóm PreemptionPriority có mức độ ưu tiên cao hơn SubPriority và cấp độ ưu tiên càng nhỏ thì ưu tiên càng cao. Có nghĩa là cấp 0 của nhóm PreemptionPriority sẽ có mức độ ưu tiên cao nhất.

2.5.3 Các lệnh thông dụng liên quan đến NVIC

|  |  |
| --- | --- |
| **Thư viện misc.h** | |
| **Thông số** | **Giải thích** |
| **NVIC\_SetVectorTable(A,B);**  (Lệnh chọn nơi lưu bảng vector ngắt) | |
| **A**:NVIC\_VectTab\_RAM NVIC\_VectTab\_FLASH | **A**: Nơi lưu Lưu trong RAM Lưu trong FLASH **B**: Địa chỉ đầu vùng nhớ cần lưu. Địa chỉ này phải là bội số của 0x200 |
| **NVIC\_PriorityGroupConfig(A);** (Lệnh chọn cách phân chia số cấp ưu tiên của 2 nhóm PreemptionPriority và SubPriority) | |
| **A**:NVIC\_PriorityGroup\_0 NVIC\_PriorityGroup\_1 NVIC\_PriorityGroup\_2 NVIC\_PriorityGroup\_3 NVIC\_PriorityGroup\_4 | **A**: Nhóm ưu tiên được chọn Nhóm 0: 0 Pre – 16 Sub Nhóm 1: 2 Pre – 8 Sub Nhóm 2: 4 Pre – 4 Sub Nhóm 3: 8 Pre – 2 Sub Nhóm 4: 16 Pre – 0 Sub |
| **NVIC\_InitTypeDef A;**  **A.NVIC\_IRQChannel = B;**  (Lệnh chọn ngoại vi cần cho phép ngắt) | |
| **B**:USART1\_IRQn SPI1\_IRQn EXTI4\_IRQn …TIM2\_IRQn | **B**: Ngoại vi cần ngắt Ngắt UART1 Ngắt SPI1 Ngắt ngoài kênh số 4 …Ngắt Timer2 **Chú ý**: Xem trong file “stm32f10x.h” từ dòng 169 đến 470 để biết cách viết tên ngoại vi cần cho phép ngắt. |
| **A.NVIC\_IRQChannelPreemptionPriority = C;** (Lệnh chọn thứ tự ưu tiên ngắt của ngoại vi đang cấu hình trong nhóm chính) | |
| **C**:0  … 15 | **C**: thứ tự ưu tiên 0: Ưu tiên cao nhất … 15: Ưu tiên thấp nhất |
| **A.NVIC\_IRQChannelSubPriority = D;** (Lệnh chọn thứ tự ưu tiên ngắt của ngoại vi đang cấu hình trong nhóm phụ) | |
| **D**:0 … 15 | **D**: thứ tự ưu tiên 0: Ưu tiên cao nhất … 15: Ưu tiên thấp nhất |
| **A.NVIC\_IRQChannelCmd = E;** (Lệnh cho phép hoặc cấm ngắt ngoại vi đang được cấu hình tại NVIC) | |
| **E**:ENABLE DISABLE | **E**: Cho phép hoặc cấm Cho phép ngắt Cấm ngắt |
| **NVIC\_Init(&A);** ( Lệnh cấu hình các thông số được lưu trong biến A cho NVIC) | |

**Ví dụ**: Viết chương trình con cấu hình NVIC cho phép ngắt ngoài kênh số 4.

|  |
| --- |
| void CauhinhNVIC\_EXTI4(){  #ifdef VECT\_TAB\_RAM  NVIC\_SetVectorTable(NVIC\_VectTab\_RAM, 0x0);  // Nếu có định nghĩa "VECT\_TAB\_RAM" thì  // lưu bảng vector ngắt vào RAM  #else  NVIC\_SetVectorTable(NVIC\_VectTab\_FLASH, 0x0);  // Nếu không định nghĩa "VECT\_TAB\_RAM"  // thì lưu bảng vector ngắt vào FLASH  #endif  NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_0);  // 0 cấp PreemptionPriority và 16 cấp SubPriority  NVIC\_InitStructure.NVIC\_IRQChannel = EXTI4\_IRQn;  // Chọn ngoại vi cần ngắt là ngắt ngoài kênh 4  //NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority=0;  //Bỏ dòng trên vì chọn chế độ PreemPtionPriority có 0 cấp  NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 1;  // Chọn thứ tự ưu tiên 1 trong SubPriority cho kênh 4  NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;  // Cho phép ngắt  NVIC\_Init(&NVIC\_InitStructure);  // Cài đặt các cấu hình trên vào NVIC  } |

**Chú ý:** Khi đã cấu hình NVIC cho phép ngoại vi ngắt thì ta phải viết chương trình con phục vụ ngắt cho ngoại vi đó. Cấu trúc của chương trình con phục vụ ngắt giống với cấu trúc của một chương trình con bình thường và chúng chỉ khác nhau ở phần tên. Có nghĩa là muốn viết chương trình con phục vụ ngắt ta viết chương trình con như bình thường và đặt tên cho nó đúng với cú pháp sau:

|  |
| --- |
| **Tên ngoại vi** khai báo ngắt trong NVIC **bỏ đi** chữ **“n”** ở cuối **thêm vào** chữ **“Handler”** |

`

Như ví dụ trên ta sẽ đặt tên chương trình con phục vụ ngắt là:

**EXTI4\_IRQn** bỏ đi chữ “**n**” ở cuối thêm vào chữ “**Handler**” = **EXTI4\_IRQHandler**

|  |
| --- |
| **void** **EXTI4\_IRQHandler** **(void){**  // Các lệnh thực thi kh ngắt ngoài xảy ra viết ở đây  **}** |

2.5.4 Ngắt ngoài

Hầu hết tất cả các chân ARM của ST đều có thể thiết lập ngắt ngoài chỉ ngoại từ một số chân đặc biệt.

Có 19 kênh [EXT0 – EXT18] phục vụ cho ngắt ngoài. Trong đó các kênh từ 0 đến 15 phục vụ yêu ngắt ngoài tương ứng với các chân từ 0 đến 15 của các PORT, kênh 16 nối với PVD (Programmable Voltage Detector), kênh 17 kết nối RTC Alarm và kênh 18 là USB Wakeup.

|  |
| --- |
|  |

bảng đồ GPIO của các kênh ngắt ngoài và sự kiện

Mỗi kênh có thể được cấu hình riêng để có thể phát hiện xung hoặc cạnh (cạnh lên, cạnh xuống, cả cạnh lên lẫn cạnh xuống), cho phép hoặc cấm ngắt.

2.5.5 Các lệnh thông dụng liên quan đến INTERRUPTS And EVENTS

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_exti.h** | |
| **Thông số** | **Thông số** |
| **EXTI\_InitTypeDef A; GPIO\_EXTILineConfig(B, C);** (Lệnh chọn chân nhận tín hiệu ngắt ngoài/sự kiện) | |
| **B:** GPIO\_PortSourceGPIOA…GPIO\_PortSourceGPIOG **C:** GPIO\_PinSource0…GPIO\_PinSource15 | **B:** Chọn PORTChọn PORTA…Chọn PORTG **C:** Chọn chânChọn chân 0…Chọn chân 15 |
| **A.EXTI\_Line = D;** (Lệnh chọn kênh ngắt ngoài/sự kiện) | |
| **D:** EXTI\_Line0…EXTI\_Line18 | **D:** EXTI\_Line0…EXTI\_Line18 |
| **A.EXTI\_Mode = E ;** (Lệnh chọn chế độ ngắt hoặc sự kiện) | |
| **E:** EXTI\_Mode\_InterruptEXTI\_Mode\_Event | **E:** EXTI\_Mode\_InterruptEXTI\_Mode\_Event |
| **A.EXTI\_Trigger = F;** (Lệnh chọn cạnh tích cực) | |
| **F:** EXTI\_Trigger\_RisingEXTI\_Trigger\_FallingEXTI\_Trigger\_Rising\_Falling | **F:** EXTI\_Trigger\_RisingEXTI\_Trigger\_FallingEXTI\_Trigger\_Rising\_Falling |
| **A.EXTI\_Trigger = F;** (Lệnh chọn cạnh tích cực) | |
| **F:** EXTI\_Trigger\_RisingEXTI\_Trigger\_FallingEXTI\_Trigger\_Rising\_Falling | **F:** EXTI\_Trigger\_RisingEXTI\_Trigger\_FallingEXTI\_Trigger\_Rising\_Falling |
| **EXTI\_Init(&A);** ( Lệnh cấu hình các thông số được lưu trong biến A cho EXTI) | |

2.5.6 Lập trình

Các bước để cấu hình và sử dụng ngắt ngoài cho 1 chân ARM STM32F1X:

* Cho phép cấp xung và cấu hình chân cần ngắt ngoài là ngõ vào (GPIO)
* Cấu hình và cho phép ngắt ngoài (chọn chân, kênh, cạnh tích cực… EXTI)
* Cho phép ngắt ngoài và cấu hình chế độ ưu tiên (NVIC)
* Viết chương chình con phục vụ ngắt ngoài

**Ví dụ:** Viết chương trình điều khiển LED kết nối với chân B0-B7 sáng hoặc tắt nhờ hai nút nhấn ON và OFF được kiểm tra bằng ngắt ngoài. Biết nút ON được nối với chân C13 và nút OFF nối với chân A8

Link code mẫu: https://drive.google.com/drive/folders/1CgL8hYUHxRBBPqKtIsQkI\_Ss4Jpr-oya?usp=sharing

2.6. Lập trình USART

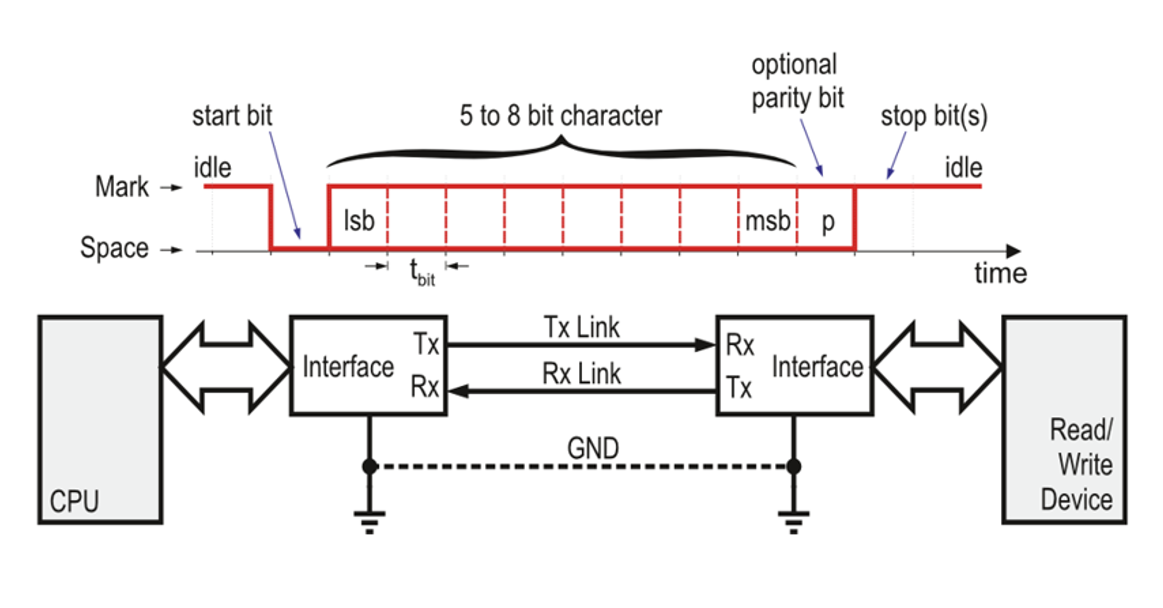
2.6.1 Giới thiệu USART

**a. USART là gì?**

USART - Universal synchronous/asynchronous receiver transmitter bộ truyền nhận nối tiếp đồng bộ/không đồng bộ là một ngoại vi cơ bản sử dụng hai đường Tx và Rx để truyền và nhận dữ liệu và thường được dùng để giao tiếp với các module như Zigbee, Bluetooth, Wifi… Khi giao tiếp UART kết hợp với các IC giao tiếp như MAX232CP, SP485EEN…. thì sẽ tạo thành các chuẩn giao tiếp RS232, RS485. Đây là các chuẩn giao tiếp thông dụng và phổ biến trong công nghiệp từ trước đến nay.

Bộ USART có hai chế độ truyền là truyền dữ liệu đồng bộ (Synchronous) và truyền dữ liệu không đồng bộ (Asynchronous).

**b. Cấu trúc khung dữ liệu**



Sơ đồ kết nối và khung dữ liệu

Ngoài việc giống nhau của tốc độ baud 2 thiết bị truyền nhận thì khung truyền của 2 bên cũng được cấu hình giống nhau. Khung truyền quy định số bit trong mỗi lần truyền, **Start bit, Data, Stop bit, Parity bit**, ngoài ra số bit quy định trong một gói dữ liệu cũng được quy định bởi khung truyền.

**Start bit:** Là bit đầu tiên của frame nó báo hiệu rằng dòng dữ liệu đã rời khỏi trạng thái idle (UART luôn ở trạng thái cao mức HIGH cho đến khi chip muốn truyền dữ liệu đi thì nó gởi bit start bằng cách kéo xuống mức LOW) Data sẽ được truyền từ thời điểm này.

**Data:** Data hay dữ liệu là thông tin mà chúng ta nhận được trong quá trình truyền và nhận. Data trong STM32 có quy định khung truyền là 8bit hoặc 9bit. Trong quá trình truyền UART, bit có trọng số thấp nhất (LSB – least significant bit – bên phải) sẽ được truyền trước và cuối cùng là bit có ảnh hưởng cao nhất (MSB – most significant bit – bên trái).

**Parity bit:** Dùng để kiếm tra tính đúng đắn của dữ liệu. Có 2 loại Parity đó là Parity chẵn (even parity) và parity lẽ (odd parity), bit này không bắt buộc.

**Stop bit:** Là bit cuối cùng của frame nó báo hiệu rằng Data đã được gửu xong. Stop bits có thể là 1bit, 1.5bit, 2bit, 0.5bit.

**Baudrate:** Số bit truyền được trong 1s, ở truyền nhận không đồng bộ thì ở các bên truyền và nhận phải thống nhất Baudrate. Các chỉ số baudrate 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200…

**Quá trình truyền data**: data từ đường bus sẽ được lưu trong bộ đệm TDR. Khi bit cho phép truyền (TE) được set, dữ liệu sẽ được đưa xuống thanh ghi Transmit Shift Register và bit có trọng số thấp nhất (LSB) sẽ được gửi đi đầu tiên trên chân TX MCU. Sau khi truyền đi một byte, thanh ghi TDR sẽ được làm trống và byte tiếp theo có thể được truyền đi. Ở chế độ này dữ liệu của thanh ghi USART\_DR chính là dữ liệu của bộ đệm TDR.

**Quá trình nhận data:** Khi có dữ liệu được truyền từ thiết bị bên ngoài, bit có trọng số thấp nhất của frame data đầu tiên được truyền qua chân RX của MCU và lưu vào thanh ghi Receive Shift Register. Khi một byte đã nhận xong, bit RXNE (Read data register not empty) set lên 1, dữ liệu chứa trong thanh ghi Receive Shift Register được đưa đến bộ đệm RDR, dữ liệu lúc này đã sẵn sàng để đọc. Nếu bit RXNE và bit RXNEIE (RXNE interrupt enable) cùng set lên 1 thì một tín hiệu ngắt sẽ được sinh ra. Bit RXNE được reset khi có hoạt động đọc dữ liệu trên thanh ghi USART\_DR hoặc khi ghi bit ‘0’ vào nó.

2.6.2 Các lệnh thông dụng liên quan đến USART

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_usart.h** | |
| **Thông số** | **Giải thích** |
| **USART\_InitTypeDef A;**  (Khai báo biến **A** thuộc kiểu dữ liệu **USART\_InitTypeDef**) | |
| **A.USART\_BaudRate = X;**  (Cấu hình tốc độ truyền dữ liệu)  **X** = 9600, 115200… | |
| **A.USART\_Mode = B;**  (Cấu hình chế độ truyền hoặc nhận dữ liệu) | |
| **B**:  USART\_Mode\_Rx  USART\_Mode\_Tx | **B**:  Chế độ nhận dữ liệu  Chế độ truyền dữ liệu |
| **A.USART\_Parity = B;**  (Cấu hình bít kiểm tra) | |
| **B**:  USART\_Parity\_No  USART\_Parity\_Even  USART\_Parity\_Odd | **B**:  Không sử dụng bít kiểm tra  Sử dụng bít kiểm tra chẵn  Sử dụng bit kiểm tra lẻ |
| **A.USART\_StopBits = B;**  (Cấu hình bit kết thúc cho quá trình truyền nhận dữ liệu) | |
| **B**:  USART\_StopBits\_1  USART\_StopBits\_0\_5  USART\_StopBits\_2  USART\_StopBits\_2\_5 | **B**:  Số bit kết thúc là 1  Số bit kết thúc là 0.5  Số bit kết thúc là 2  Số bit kết thúc là 2.5 |
| **A.** **USART\_WordLength = B:**  (Cấu hình số bit dữ liệu) | |
| **B:**  USART\_WordLength\_8b  USART\_WordLength\_9b | **B**:  Số bít dữ liệu là 8  Số bít dữ liệu là 9 |
| **A.** **USART\_HardwareFlowControl = B;**  (Cấu hình luồn điều khiển phần cứng)  Chủ yếu dùng không sử dụng. **B** = USART\_HardwareFlowControl\_None | |
| **USART\_Init(B,&A);**  (Lệnh cấu hình cho UARTx theo các thông số được lưu trong biến **A**) | |
| **B:** USARTx  USART1  USART2  … | **B:**  Sử dụng bộ USART1  Sử dụng bộ USART2  … |
| **USART\_Cmd(B,C);**  (Cấu hình cho phép hoặc không cho phép USARTx hoạt động) | |
| **B:** USARTx  USART1  USART2  …  **C:**  ENABLE  DISABLE | B:  Sử dụng bộ USART1  Sử dụng bộ USART2  …  **C:**  Cho phép  Không cho phép |
| **USART\_SendData(A,B);**  (Hàm truyền dữ liệu) | |
| **A:** USARTx  USART1  USART2  …  **B:** Data | **A:**  Sử dụng bộ USART1  Sử dụng bộ USART2  …  **B:** Dữ liệu cần truyền đi. |
| **USART\_ReceiveData(A);**  (Hàm nhận dữ liệu) | |
| **A:** USARTx  USART1  USART2  … | **A:**  Sử dụng bộ USART1  Sử dụng bộ USART2  … |

2.6.3 Phần mềm và công cụ hỗ trợ

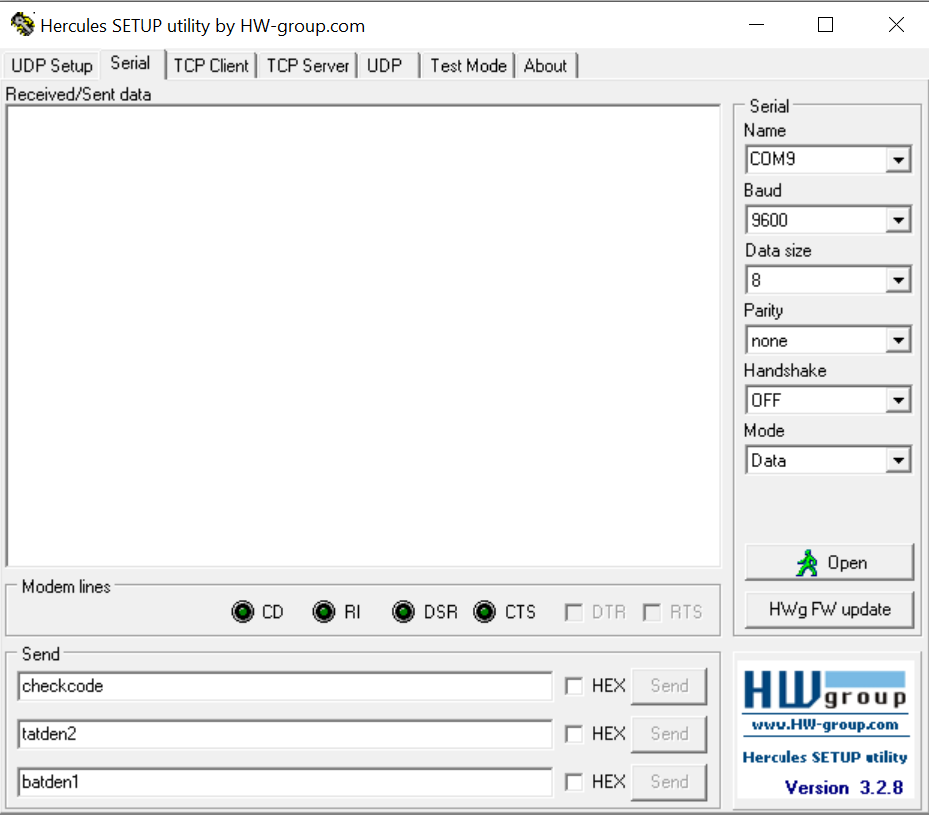
**a. Phần mềm hỗ trợ**

Để thực làm việc hiệu quả với giao tiếp USART thì ta cần một số phần mềm hỗ trợ hiển thị kết quả lên máy cụ thể là giao tiếp với máy tình.

Một số phần mềm giúp giao tiếp với mạch tính: Hercules, PuTTY,… chủ yếu chúng ta sử dụng phần mềm Hercules.

*Link phần mềm: https://drive.google.com/file/d/1g-U5GmtlT9sMrnAct9d\_Nez5N889ZqCZ/view?usp=sharing*

Sau khi cài đặt phần mềm thì chúng ta mở phần mềm và chuyển sang tab **“Serial”.**



Tại giao diện của tab **“Serial”** chúng ta sẽ cấu hinh các thông số như trong hình ảnh trên.

**Baud, Data sizze** là thông số mà chúng ta cấu hình trong code.

**Name:** khi ta kết nối với các công cụ hỗ trợ (cp2102, pl2303…) thì cổng **COM** sẽ hiển thị khác nhau với mỗi máy tính. Như máy của mình khi kết nối thì nó sẽ hiển thị ra là **COM9**.

**b. Công cụ hỗ trợ**

Công cụ hỗ trợ ở đây là các module dùng để giao tiếp uart như CP2102, CH340, PL2303,… với mỗi module khi kết nốt với máy tính thì chúng ta phải tiến hành cài **Driver** cho module đó. Tùy vào từng máy tinh nó sẽ tự động update driver, nếu không thì chúng ta phải tự cài đặt.



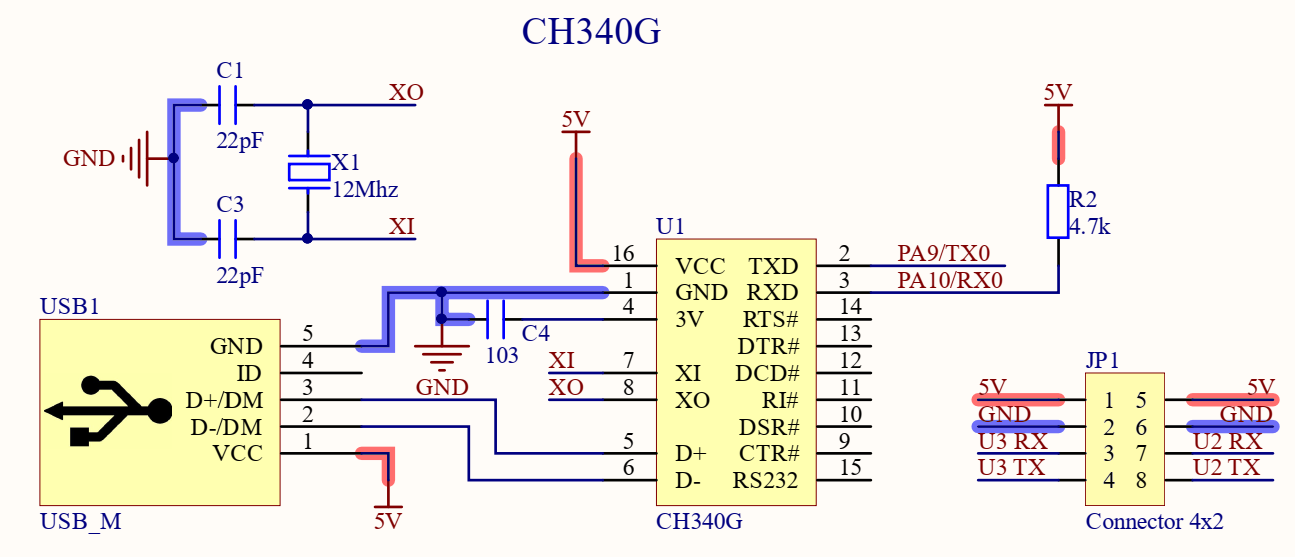
*Module Cp2102*



*Module CH340*

2.6.4 Lập trình

**a. Nguyên lý**



*Sơ đồ nguyên lý CH340*

Để thuận tiện cho việc giao tiếp UART thì trên mạch đã thiết kế sẵn cổng MicroUSB kết nối vói chip CH340G và sử dụng bộ USART1 của chíp STM32F1.

Ngoài ra trên mạch còn thiết kế thêm jum cắm cho 2 bộ USART2 và USART3 để phục vụ thêm cho quá trình sử dụng.

Như vậy thì trên Kit có tổng cộng 3 bộ USART gồm:

|  |  |  |
| --- | --- | --- |
| USARTx | Pin Tx | Pin Rx |
| USART1 | A9 | A10 |
| USART2 | A2 | A3 |
| USART3 | B10 | B11 |

**b. Lập trình**

Các bước lập trình giao tiếp USART:

* Cấp clock cho ngoại vi(GPIO, USART)
* Cấu hình GPIO cho các chân của USART trong đó:
* Tx: GPIO\_Mode\_AF\_PP
* Rx: GPIO\_Mode\_IN\_FLOATING
* Cấu hình USART và cho phép USART hoạt động
* Viết chương chình truyền nhận dữ liệu

**Ví dụ:** Viết chương trình truyền kí tự ‘A’, hiển thị kết quả thông qua cổng com trên máy tính, sử dụng USART1, Baudrate = 9600.

* Cấp clock cho ngoại vi

|  |
| --- |
| RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1, ENABLE); |

* Cấu hình chân Tx,Rx

|  |
| --- |
| GPIO\_InitTypeDef gpio\_typedef;  // congifgure pin Tx - A9;  gpio\_typedef.GPIO\_Pin = GPIO\_Pin\_9;  gpio\_typedef.GPIO\_Mode = GPIO\_Mode\_AF\_PP;  gpio\_typedef.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA,&gpio\_typedef);  // configure pin Rx - A10;  gpio\_typedef.GPIO\_Pin = GPIO\_Pin\_10;  gpio\_typedef.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;  gpio\_typedef.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA,&gpio\_typedef); |

* Cấu hình USART và cho phép USART hoạt động

|  |
| --- |
| USART\_InitTypeDef usart\_typedef;  // usart configure  usart\_typedef.USART\_BaudRate = 9600;  usart\_typedef.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;  usart\_typedef.USART\_Mode = USART\_Mode\_Tx;  usart\_typedef.USART\_Parity = USART\_Parity\_No;  usart\_typedef.USART\_StopBits = USART\_StopBits\_1;  usart\_typedef.USART\_WordLength = USART\_WordLength\_8b;  USART\_Init(USART1, &usart\_typedef);  USART\_Cmd(USART1, ENABLE); |

* Chương trình truyền kí tự

|  |
| --- |
| void uart\_SendChar(char \_chr){  USART\_SendData(USART1, ’A’);  // chờ đến khi truyền xong.  while(USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE)==RESET);  } |

**Ví dụ:** Viết chương trình truyền chuỗi kí tự “Lap Trinh Nhung”, hiển thị kết quả thông qua cổng Com trên máy tính, sử dụng USART1, Baudrate = 9600.

Các bước và cách làm giống như ví dụ trên, khác chương trình truyền kí tự.

Chương trình truyền chuỗi kí tự:

**Cách 1:**

|  |
| --- |
| void uart\_SendStr(char \*str){  while(\*str != NULL){  uart\_SendChar(\*str++);  }  }  // uart\_SendStr(“Lap Trinh Nhung”); |

**Cách 2:** Sử dụng hàm **printf** trong thư viện “**stdio.h**”

|  |
| --- |
| struct \_\_FILE {  int dummy;  };  FILE \_\_stdout;    int fputc(int ch, FILE \*f) {  uart\_SendChar(ch);  return ch;  }  //printf(“Lap Trinh Nhung”); |

*Link tải code mẫu:*

https://drive.google.com/drive/folders/1ARkC2Mb0vK3qFWtpf7u7ap6IBl1MpAcf?usp=sharing

**Ví dụ:** Viết chương trình truyền nhận dữ liệu điều khiển LED.

Nếu chuỗi nhận về là:

**“BatDen” -> LED ON**

**“TatDen”-> LED OFF**

Với LED được kết nối với B0-B7, sử dụng USART1, baudrate = 9600.

Giải pháp:

* Để nhận được dữ liệu một cách chính xác nhất thì chúng ta sẽ sử dụng ngắt USART để nhận dữ liệu.
* Để sử dụng ngắt nhận USART thì chúng ta cần phải cấu hình ngắt cho chức năng này. Một số hàm sử dụng để cấu hình ngắt:

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_usart.h** | |
| **Thông số** | **Giải thích** |
| **USART\_ITConfig(A,B,C);**  (Hàm cho phép hoặc không cho phép ngắt USART) | |
| **A**: USARTx  USART1  USART2  USART3  **B:**  USART\_IT\_TXE  USART\_IT\_RXN  …  **C:**  ENABLE  DISBALE | **A**: Bộ USARTx  Sử dụng bộ USART1  Sử dụng bộ USART2  Sử dụng bộ USART3  **B**:  Ngắt truyền  Ngắt nhận  …  **C**:  Cho phép  Không cho phép |
| **NVIC\_EnableIRQ(A);**  (cho phép ngắt tổng quát) | |
| A:  USART1\_IRQn  USART2\_IRQn  USART3\_IRQn  … | **B**:  Ngắt tổng quát USART1  Ngắt tổng quát USART2  Ngắt tổng quát USART3  …  **Chú ý:** tên của hàm ngắt được viết như sau:  **USART1\_IRQn** bỏ ‘**n**’ + “**Handler**”  = **USART1\_IRQHandler**() |

* Cấp clock, cấu hình GPIO, USART như bình thường. Với yêu cầu nhận dữ liệu thì chúng ta thêm một bước cấu hình ngắt nhận và cho phép ngắt USARTx tổng quát.
* Viết hàm ngắt USARTx

|  |
| --- |
| #include "stm32f10x.h"  #include "stm32f10x\_gpio.h"  #include "stm32f10x\_rcc.h"  #include "string.h"  #include "stdio.h"  #define MAX 100  char ONLED[]=”BatDen”;  char OFFLED[]=”TatDen”;  char vrc\_Getc; // bien kieu char, dung de nhan du lieu tu PC gui xuong;  uint16\_t vri\_Stt; // bien danh dau trang thai.  uint16\_t vri\_Count = 0; // bien diem  char vrc\_Res[MAX]; //Mang kieu char vrc\_Res[MAX]: mang luu tru tam thoi;  void Delay\_ms(uint16\_t \_time);  void uart\_Init(void);  void led\_Init(void);  void uart\_SendChar(char \_chr);  uint16\_t UARTx\_Getc(USART\_TypeDef\* USARTx);  int main(){  uart\_Init();  led\_Init();  while(1){  if(vri\_Stt){  if(strstr(vrc\_Res ,ONLED)!=NULL){  GPIOB->ODR = 0x00;  }  else if(strstr(vrc\_Res ,OFFLED)!=NULL){  GPIOB->ODR = 0xFF;  }  vri\_Stt = 0;  }  }  }  struct \_\_FILE {  int dummy;  };  FILE \_\_stdout;    int fputc(int ch, FILE \*f) {  uart\_SendChar(ch);  return ch;  }  void led\_Init(void){    GPIO\_InitTypeDef led\_init;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);  GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);    led\_init.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  led\_init.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;  led\_init.GPIO\_Speed = GPIO\_Speed\_50MHz;    GPIO\_Init(GPIOB,&led\_init);  }  void uart\_Init(void){  GPIO\_InitTypeDef gpio\_typedef;  USART\_InitTypeDef usart\_typedef;  // enable clock  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1, ENABLE);  // congifgure pin Tx - A9;  gpio\_typedef.GPIO\_Pin = GPIO\_Pin\_9;  gpio\_typedef.GPIO\_Mode = GPIO\_Mode\_AF\_PP;  gpio\_typedef.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA,&gpio\_typedef);  // configure pin Rx - A10;  gpio\_typedef.GPIO\_Pin = GPIO\_Pin\_10;  gpio\_typedef.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;  gpio\_typedef.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA,&gpio\_typedef);  // usart configure  usart\_typedef.USART\_BaudRate = 9600;  usart\_typedef.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;  usart\_typedef.USART\_Mode = USART\_Mode\_Tx | USART\_Mode\_Rx;  usart\_typedef.USART\_Parity = USART\_Parity\_No;  usart\_typedef.USART\_StopBits = USART\_StopBits\_1;  usart\_typedef.USART\_WordLength = USART\_WordLength\_8b;  USART\_Init(USART1, &usart\_typedef);  USART\_ITConfig(USART1, USART\_IT\_RXNE, ENABLE);  /\* Enable USART1 global interrupt \*/  NVIC\_EnableIRQ(USART1\_IRQn);  USART\_Cmd(USART1, ENABLE);  }  void uart\_SendChar(char \_chr){  USART\_SendData(USART1,\_chr);  while(USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE)==RESET);  }  // hàm nhận về 1 kí tự  uint16\_t UARTx\_Getc(USART\_TypeDef\* USARTx){  return USART\_ReceiveData(USARTx);  }  void USART1\_IRQHandler(){  if(USART\_GetITStatus(USART1, USART\_IT\_RXNE)!=RESET){  vrc\_Getc = UARTx\_Getc(USART1);  if(vrc\_Getc == '13'){// 13: enter; bao hieu ket thuc chuoi  vri\_Stt = 1; // dieu kien kiem tra de thuc hien cac chuong trinh theo yeu cau.  vrc\_Res[vri\_Count] = NULL;  vri\_Count = 0;  }  else{  vrc\_Res[vri\_Count] = vrc\_Getc;  vri\_Count++;  }  }  } |

2.7 Lập trình TIMER

2.7.1 Giới thiệu TIMER

Họ STM32F1x có 3 loại timer là:

* Các timer điều khiển nâng cao (Advanced control timers – TIM1&TIM8)
* Các timer cơ bản ( Basic timers – TIM6&TIM7)
* Các timer sử dụng cho mục đích thông thường(General purpose timers –timer còn lại)

|  |
| --- |
|  |

Cấu trúc bộ Tỉmer Advanced control

Bộ Timer Advanced-Control bao gồm TIM1 & TIM8 đối với dòng vi điều khiển STM32F1. Các bộ timer này có một số đặc điểm như sau:

* Bộ đếm 16 bit có thể đếm tiến, lùi, tiến/lùi tự động nạp lại
* Hỗ trợ bộ chia tần 16 bit có thể lập trình được.
* Hỗ trợ 4 kênh hoạt động độc lập cho các chức năng:
* Caputre đầu vào (Input Capture)
* So sánh đầu ra (Output Compare)
* Điều chế độ rộng xung (PWM)
* Ngõ ra của chế độ 1 xung( One-pulse mode output)
* Hỗ trợ kích hoạt các sự kiến DMA
* Hỗ trợ kích hoạt ngắt
* Hỗ trợ trigger cho các sự kiện

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.

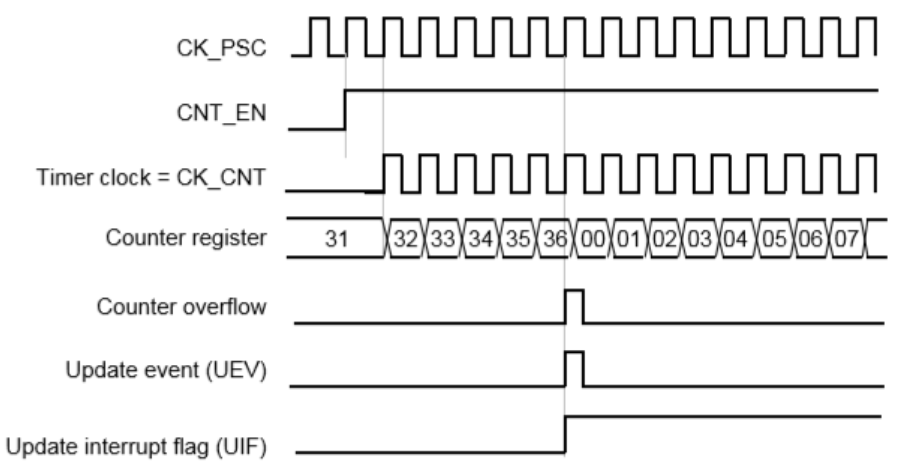
2.7.2 Các chế độ hoạt động của TIMER

**a. Chế độ đếm lên- upcounting mode**

Ở chế độ này, TIMx\_CNT đếm lên từ 0 (hoặc một giá trị nào đó được người dùng ghi vào TIMx\_CNT trước) đến giá trị của thanh ghi TIMx\_ARR, sau đó TIMx\_CNT bắt đầu lại từ 0. Lúc này có sự kiện tràn counter – overflow, sự kiện này có thể tạo yêu cầu ngắt nếu người dùng cấu hình cho phép ngắt.

Khi sự kiện cập nhật xảy ra thì:

* Cờ cập nhật được set
* Giá trị đếm lặp được nạp lại từ thanh ghi TIMx\_RCR
* Giới hạn đếm được nạp lại từ thanh ghi TIMx\_ARR.
* Giá trị của bộ chia được nạp lại từ thanh ghi TIMx\_PSC



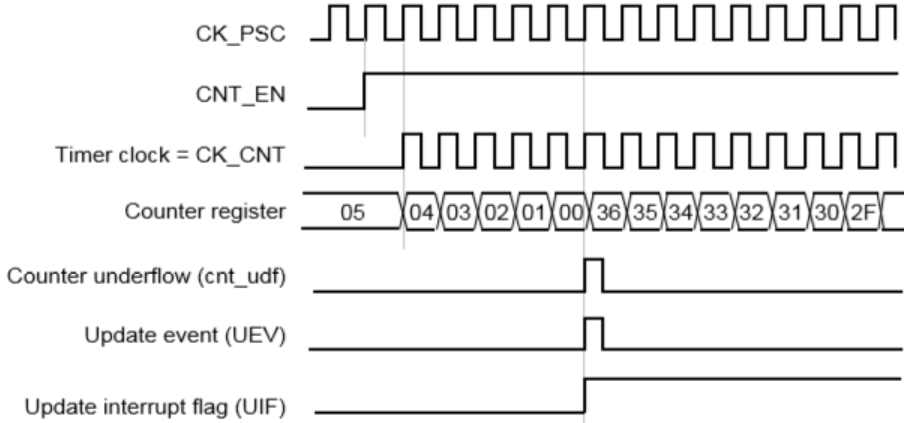
Chế độ đếm lên với giá trị nạp lại là 0x36

**b. Chế độ đếm xuống – downcounting mode**

Ở chế độ này, TIMx\_CNT đếm xuống từ giá trị thanh ghi TIMx\_ARR (hoặc 1 giá trị nào đó do người dùng ghi trực tiếp vào TIMx\_CNT trước) đến 0, sau đó TIMx\_CNT bắt đầu lại từ giá trị TIMx\_ARR, lúc này có sự kiện tràn counter – underflow, sự kiện này có thể tạo yêu cầu ngắt nếu người dùng cấu hình cho phép ngắt.

Khi sự kiện cập nhật xảy ra thì:

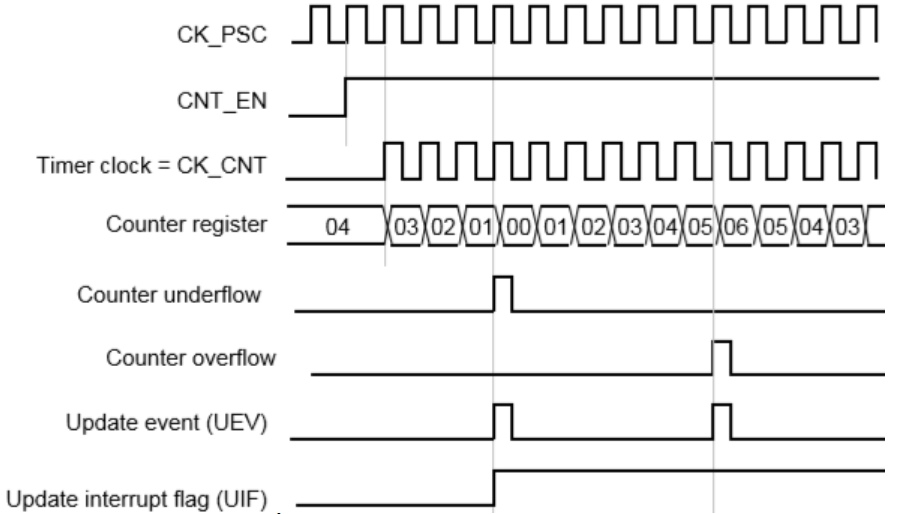
* Cờ cập nhật được set
* Giá trị đếm lặp được nạp lại từ thanh ghi TIMx\_RCR
* Giới hạn đếm được nạp lại từ thanh ghi TIMx\_ARR



Chế độ đếm lên với giá trị nạp lại là 0x36

**c. Chế độ canh giữa( đếm lên/xuống)-Center-aligned mode (up/down counting)**

Ở chế độ này, counter sẽ đếm lên từ 0 (hoặc một giá trị nào đó được người dùng ghi vào TIMx\_CNT trước) đến giá trị thanh ghi TIMx\_ARR – 1, lúc này xuất hiện sự kiện tràn counter – overflow, tiếp theo TIMx\_CNT sẽ đếm xuống từ TIMx\_ARR tới 1, lúc này có sự kiện tràn counter – underflow, sau đó TIMx\_CNT sẽ về giá trị 0 và bắt đầu lại quá trình đếm lên.



Chế độ canh giữa với giá trị tự động nạp lại la 0x06

2.7.3 Các lệnh thông dụng

**a. Các lệnh thông dụng dùng để cấu hình timer base**

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_tim.h** | |
| **Thông số** | **Giải thích** |
| **TIM\_TimeBaseInitTypeDef A;**  (Khai báo biến **A** thuộc kiểu **TIM\_TimeBaseInitTypeDef** ) | |
| **A.TIM\_Period = B;**  ( Lệnh chọn chu kỳ đếm cho timer- giá trị tự động nạp lại ) | |
| **B:**  [0-65535] | **B:** Giá trị cài đặt  Giới hạn từ 0 đến 65535 |
| **A. TIM\_Prescaler = C;**  ( Lệnh chọn bộ chia(xung) trước khi vào bộ đếm) | |
| **B**:  [0-65535] | **B**: Giá trị cài đặt  Giới hạn từ 0 đến 65535 |
| **A. TIM\_ClockDivision= D;**  ( Lệnh chọn bộ chia xung clock cho timer ) | |
| **D**:  TIM\_CKD\_DIV1  TIM\_CKD\_DIV2  TIM\_CKD\_DIV4 | **D**: Bộ chia  Chia 1  Chia 2  Chia 4 |
| **A.TIM\_RepetitionCounter= E;**  (Lệnh chọn số lần đếm lặp cho timer ) | |
| **E**:  [0-255] | **E**: Số lần đếm lặp  Giá trị từ 0 đến 255 |
| **A. TIM\_CounterMode= F;**  ( Lệnh chọn chế độ hoạt động cho timer ) | |
| **F**:  TIM\_CounterMode\_Up  TIM\_CounterMode\_Down  TIM\_CounterMode\_CenterAligned1  TIM\_CounterMode\_CenterAligned2  TIM\_CounterMode\_CenterAligned3 | **F**: Chế độ hoạt động  Đếm lên  Đếm xuống  Đếm lên xuống 1  Đếm lên xuống 2  Đếm lên xuống 3 |
| **TIM\_TimeBaseInit(G, &A);**  (Lệnh cài đặt các thông số đã cấu hình cho Timer ) | |
| **G**:  TIM1  …  TIMn | **G**: Timer cần cấu hình  Timer 1  …  Timer n |

**b.** **Các lệnh thông dụng dùng để cấu hình input capture**

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_tim.h** | |
| **Thông số** | **Giải thích** |
| **TIM\_ICInitTypeDef A;** (Khai báo biến A thuộc kiểu TIM\_ICInitTypeDef ) | |
| **A. TIM\_Channel =B;** (Lệnh chọn kênh Input Capture) | |
| **B:** TIM\_Channel\_1 TIM\_Channel\_2 TIM\_Channel\_3 TIM\_Channel\_4 | **B: Kênh được chọn** Kênh 1 Kênh 2 Kênh 3 Kênh 4 |
| **A. TIM\_ICPolarity = C;** (Lệnh chọn cạnh tích cực cho tín hiệu Input Capture) | |
| **C:** Tích cực cạnh lên Tích cực cạnh xuống Tích cực cả cạnh lên lẫn cạnh xuống | **C:** Tích cực cạnh lên Tích cực cạnh xuống Tích cực cả cạnh lên lẫn cạnh xuống |
| **A. TIM\_ICSelection= D;** (Lệnh cấu hình cách kết nối các tín hiệu Input Capture) | |
| **D:** TIM\_ICSelection\_DirectTI TIM\_ICSelection\_IndirectTI TIM\_ICSelection\_TRC | **D: Cách kết nối** Timer Input 1, 2, 3, 4 được kết nối tương ứng với Input Capture 1, 2, 3, và 4 Timer Input 1, 2, 3, 4 được kết nối tương ứng với Input Capture 2, 1, 4, 3 Các ngõ Input Capture kết nối với TRC |
| **A. TIM\_ICPrescaler = E;** ( Lệnh chọn bộ chia trước cho tín hiệu Input Captuer) | |
| **E:** TIM\_ICPSC\_DIV1 TIM\_ICPSC\_DIV2 TIM\_ICPSC\_DIV4 TIM\_ICPSC\_DIV8 | **E: Bộ chia trước** Chia 1 Chia 2 Chia 4 Chia 8 |
| **A. TIM\_ICFilter = F;** (Lệnh chọn hệ số lọc cho tín hiệu Input Capture) | |
| **F:** [0-15] | **F: Hệ số lọc** Giá trị từ 0 đến 15 |
| **TIM\_ICInit (G,&A);** (Lệnh cài đặt các thông số đã chọn để Timer hoạt động ở chế độ Capture bình thường) | |
| **G:** TIM1 … TIMn | **G: Timer cần cấu hình** Timer 1 … Timer n |

|  |  |
| --- | --- |
| **TIM\_SelectInputTrigger(G, H) ;** ( Lệnh chọn tín hiệu kích hoạt từ bên ngoài) | |
| **H:** TIM\_TS\_ITR0 TIM\_TS\_ITR1 TIM\_TS\_ITR2 TIM\_TS\_ITR3 TIM\_TS\_TI1F\_ED TIM\_TS\_TI1FP1 TIM\_TS\_TI2FP2 TIM\_TS\_ETRF | **H: Tín hiệu kích hoạt được chọn** ITR0 ( từ timer 1 trigger out) ITR1 ( từ timer 2 trigger out) ITR2 ( từ timer 3 trigger out) ITR3 ( từ timer 4 trigger out) TI1( CH1) qua mạch phát hiện cạnh TI1( CH1) qua mạch lọc TI2( CH2) qua mạch lọc ETR qua mạch lọc |

1. **Các lệnh thông dụng dùng để cấu hình ouput compare**

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_tim.h** | |
| **Thông số** | **Giải thích** |
| **TIM\_OCInitTypeDef A;**  (Khai báo biến A thuộc kiểu TIM\_OCInitTypeDef ) | |
| **A.TIM\_OCMode=B;**  (Lệnh chọn chế độ hoạt động cho ouput compare) | |
| **B:** TIM\_OCMode\_Timing TIM\_OCMode\_Active TIM\_OCMode\_Inactive TIM\_OCMode\_Toggle TIM\_OCMode\_PWM1 TIM\_OCMode\_PWM2 | **B: chế độ hoạt động** Chế độ định thời Chế độ tích cực Chế độ không tích cực Chế độ nghịch đảo ngõ ra Chế độ tạo xung PWM ở ngõ ra loại 1 Chế độ tạo xung PWM ở ngõ ra loại 2 (PWM loại 1 và loại 2 nghịch đảo nhau) |
| **A.TIM\_OutputState = C;**  ( Lệnh cho phép hoặc cấm Output Compare xuất tín hiệu ra chân CHx của Timer) | |
| **C:** TIM\_OutputState\_Disable TIM\_OutputState\_Enable | **C:** Cho phép hoặc cấm Không cho phép Cho phép |
| **A.TIM\_OutputNState = D;**  (Lệnh cho phép hoặc cấm Output Compare xuất tín hiệu ra các chân CHxN)  (Chỉ dùng cho TIMER 1 và TIMER 8) | |
| **D:** TIM\_OutputNState\_Disable TIM\_OutputNState\_Enable | **D:** Cho phép hoặc cấm Không cho phép Cho phép |

|  |  |
| --- | --- |
| **A.TIM\_OCPolarity = E;**  ( Lệnh cấu hình mức tích cực của Output Compare đối với các chân CHx) | |
| **E:** TIM\_OCPolarity\_High TIM\_OCPolarity\_Low | **E:** Mức tích cực Tích cực mức cao Tích cực mức thấp |
| **A.TIM\_OCNPolarity = F;**  ( Lệnh cấu hình mức tích cực của Output Compare đối với các chân CHxN)  (Chỉ dùng cho TIMER 1 và TIMER 8) | |
| **F:** TIM\_OCNPolarity\_High TIM\_OCNPolarity\_Low | **F:** Mức tích cực Tích cực mức cao Tích cực mức thấp |
| **A.TIM\_Pulse = G;**  (Lệnh cấu hình giá trị so sánh cho Output Compare) | |
| **G:** [0-65535] | **G:** Giá trị so sánh Giới hạn từ 0 đến 65535 |
| **TIM\_OC1Init(H, &A);**  **TIM\_OC2Init(H, &A);**  **TIM\_OC3Init(H, &A);**  **TIM\_OC4Init(H, &A);**  (Các lệnh cấu hình những thông số được lưu trong biến A các bộ output compare ) | |
| **H:** TIM1 … TIMn | **H:** Timer cần cấu hình Timer 1 … Timer n |
| **TIM\_SelectOutputTrigger (H, I);**  (Lệnh cấu hình tín hiệu kích hoạt xuất ra bên ngoài- TRGO) | |
| **I:** TIM\_TRGOSource\_Reset TIM\_TRGOSource\_Enable TIM\_TRGOSource\_Update TIM\_TRGOSource\_OC1 TIM\_TRGOSource\_OC1Ref TIM\_TRGOSource\_OC2Ref TIM\_TRGOSource\_OC3Ref TIM\_TRGOSource\_OC4Ref | **I**: Nguồn xung kích hoạt Xuất xung kích hoạt khi timer reset Xuất xung kích hoạt khi cho phép timer Xuất xung kích hoạt khi timer cập nhật Xung kích hoạt là tín hiệu OC1 Xung kích hoạt là tín hiệu OC1Ref Xung kích hoạt là tín hiệu OC2Ref Xung kích hoạt là tín hiệu OC3Ref Xung kích hoạt là tín hiệu OC4Ref |
| **TIM\_CtrlPWMOutputs(H, J);**  (Lệnh cho phép hoặc cấm xuất xung PWM ở ngõ ra) | |
| **J:** ENABLE DISABLE | **J:** Cho phép hoặc cấm Cho phép Không cho phép |

1. **Các lệnh thông dụng liên quan đến ngắt timer**

|  |  |
| --- | --- |
| **Thư viện stm32f10x\_tim.h** | |
| **Thông số** | **Giải thích** |
| **TIM\_ITConfig (A, B,C);**  ( Lệnh cấu hình ngắt cho timer ) | |
| **A:** TIM1 … TIMn **B:** TIM\_IT\_Update TIM\_IT\_CC1 TIM\_IT\_CC2 TIM\_IT\_CC3 TIM\_IT\_CC4 TIM\_IT\_COM TIM\_IT\_Trigger TIM\_IT\_Break **C:** ENABLE DISABLE | **A:** Timer cần cho phép ngắt Timer 1 … Timer n **B:** Chọn nguồn ngắt Ngắt khi timer cập nhật Ngắt của bộ Capture Compare 1 Ngắt của bộ Capture Compare 2 Ngắt của bộ Capture Compare 3 Ngắt của bộ Capture Compare 4 Ngắt có sự kiện Comunication Ngắt khi có sự kiện Trigger Ngắt khi có sự kiện Break Input **C:** Cho phép hoặc không cho phép Cho phép Không cho phép |
| **TIM\_GetITStatus(A, B);**  ( Lệnh đọc cờ ngắt của timer ) | |
| **B:** TIM\_IT\_Update TIM\_IT\_CC1 TIM\_IT\_CC2 TIM\_IT\_CC3 TIM\_IT\_CC4 TIM\_IT\_COM TIM\_IT\_Trigger TIM\_IT\_Break | **B:** Cờ ngắt cần đọc Ngắt khi timer cập nhật Ngắt của bộ Capture Compare 1 Ngắt của bộ Capture Compare 2 Ngắt của bộ Capture Compare 3 Ngắt của bộ Capture Compare 4 Ngắt có sự kiện Comunication Ngắt khi có sự kiện Trigger Ngắt khi có sự kiện Break Input |
| **TIM\_ClearITPendingBit(A, B);**  ( Lệnh xóa cờ ngắt của Timer ) | |
| **B:** TIM\_IT\_Update … | **B:** Cờ ngắt cần xóa Ngắt khi timer cập nhật … giống B của TIM\_GetITStatus(A, B, C); |

2.7.4 Lập trình

**a. Tạo độ trễ ms**

**Ví dụ:** viết chương trình tạo độ trễ 1ms sử dụng Timer2.

Giải pháp:

* Sử dụng Timer 2 để tạo độ trễ thì chúng ta sử dụng chế độ đếm của Tỉmer ( mode counter ). Sử dụng công thức chia tần để tính toán các thông số PSC, PD

|  |
| --- |
| * T = 0.001s, chọn giá trị PSC = 2   Ta có 2 giá trị để cấu hình Timer:   * PSC = 2 – 1 * PD = 36000 – 1 |

* Để sử dụng được Timer thì chúng ta cần phải cấp clock và cấu hình cho Tỉmer 2.

|  |
| --- |
| void timer\_Init(void){  TIM\_TimeBaseInitTypeDef timer\_init;  RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2, ENABLE);  timer\_init.TIM\_CounterMode = TIM\_CounterMode\_Up;  timer\_init.TIM\_Period = 65535; // có thể để giá trị max  timer\_init.TIM\_Prescaler = 2 - 1;  timer\_init.TIM\_RepetitionCounter = 0;  TIM\_TimeBaseInit(TIM2,&timer\_init);  } |

* Viết chương trình tạo độ trễ ms

|  |
| --- |
| void Delay\_1ms(void){  // set gia tri dem cua timer  timer\_Init();  TIM\_Cmd(TIM2,ENABLE); // cho phép Timer2 hoạt động  TIM\_SetCounter(TIM2,0); // đặt giá trị ban dầu cho Timer2  while(TIM\_GetCounter(TIM2)<36000); // lặp đến khi giá trị bộ đếm < 36000  TIM\_Cmd(TIM2,DISABLE); // không cho phép Timer2 hoạt động  }  void Delay\_ms(unsigned int time){  while(time--){  Delay\_1ms();  }  } |

**b. Sử dụng ngắt Tiner**

**Ví dụ:** Viết chương trình blink led B0-B7 sử dụng ngắt Timer2 với t = 0.5s.

Giải pháp:

* Sử dụng ngắt Timer2 với t = 0.5s:

|  |
| --- |
| * T = 0.5s, chọn giá trị PSC = 3600   Ta có 2 giá trị để cấu hình Timer:   * PSC = 3600 – 1 * PD = 10000 – 1 |

* Cấp clock, cấu hình tỉmer2 và ngắt timer2:

|  |
| --- |
| void timer\_Init(void){  TIM\_TimeBaseInitTypeDef timer\_init;  RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2, ENABLE);  timer\_init.TIM\_CounterMode = TIM\_CounterMode\_Up;  timer\_init.TIM\_Period = 10000 - 1;  timer\_init.TIM\_ClockDivision = 0;  timer\_init.TIM\_RepetitionCounter = 0;  timer\_init.TIM\_Prescaler = 3600 - 1;  TIM\_TimeBaseInit(TIM2,&timer\_init);  TIM\_ClearFlag(TIM2,TIM\_FLAG\_Update);  TIM\_ITConfig(TIM2,TIM\_IT\_Update,ENABLE);  NVIC\_EnableIRQ(TIM2\_IRQn); // thay thể cho chương trình cấu hình NVIC  TIM\_Cmd(TIM2,ENABLE);  } |

* Chương trình ngắt Tỉmer2

|  |
| --- |
| volatile char vrvc\_SttBlink = 0;  void TIM2\_IRQHandler(){  if(TIM\_GetITStatus(TIM2, TIM\_IT\_Update)!=RESET){  vrvc\_SttBlink = !vrvc\_SttBlink;// sau 0.5s sẽ đảo trạng thái biến vrvc\_SttBlink  }  TIM\_ClearITPendingBit(TIM2,TIM\_IT\_Update);  } |

*Link tải code mẫu: https://drive.google.com/drive/folders/1KJuk9RooGQ4loPjKYn-dwjVDwUp30Aja?usp=sharing*

2.8 Lập trình ADC

2.8.1 Giới thiệu ADC

ADC(Analog to Digital Converter): bộ chuyển đổi tín hiệu tương tự sang tín hiệu số.

ADC của họ ARM STM32 F1 có độ phân giải là 12 bit và có tất cả 18 kênh tương tự trong đó 16 kênh nhận tín hiệu bên ngoài và 2 kênh nhận tín hiệu bên trong để chuyển đổi.

Mỗi kênh ADC có thể được cấu hình độc lập để chuyển đổi tín hiệu tương tự sang số theo nhiều chế độ khác nhau như là chuyển đổi đơn( single), liên tục( continuous), quét( scan) hoặc không liên tục( discontinuous).

Đối với kết quả chuyển đổi ADC người dùng có thể cài đặt canh phải hoặc canh trái khi lưu vào thanh ghi dữ liệu ADC.

Bộ giám sát tín hiệu tương tự( analog watchdog) cho phép phát hiện việc điện áp vượt khỏi ngưỡng cao hoặc thấp do người dùng cài đặt.

Xung clock cấp cho ADC được tạo ra bởi PLCK2 và đi qua một bộ chia sao cho khi cấp cho ADC tần số xung này không vượt quá 14 Mhz

2.8.2 Những đặc điểm của ADC

* Độ phân giải 12 bit.
* Có khả năng tạo ra ngắt khi chuyển đổi hoàn tất, và khi bộ giám sát tín hiệu tương tự(
* Anglog Watchdog) phát hiện điện áp vào vượt quá ngưỡng cài đặt.
* Có thể hoạt động được ở chế độ chuyển đổi 1 lần( Single) hoặc liên tục( Continuous)
* Hỗ trợ chế độ quét( Scan) cho phép tự động chuyển đổi từ kênh 0 đến kênh “n”.
* Có bộ hiệu chỉnh thông số cho phép kết quả chuyển đổi được chính xác.
* Dữ liệu sau khi chuyển đổi cho phép canh phải hoặc canh trái.
* Dữ liệu sau khi chuyển đổi cho phép canh phải hoặc canh trái
* Mỗi kênh có thể được lập trình thời gian lấy mẫu riêng
* Hỗ trợ tín hiệu yêu cầu chuyển đổi từ bên ngoài cho cả hai nhóm regular và injected
* Hỗ trợ chế độ không liên tục( Discontinuous mode)
* Có thể hoạt động ở chế độ song song( chỉ có trên các dòng ARM có từ 2 ADC trở lên)
* Thời gian chuyển đổi:
* STM32F103xx: 1 µs khi hoạt động ở 56 MHz (1.17 µs tại 72 MHz)
* STM32F101xx: 1 µs khi hoạt động ở 28 MHz (1.55 µs tại 36 MHz)
* STM32F102xx: 1.2 µs khi hoạt động ở 48 MHz
* STM32F105xx và STM32F107xx: 1 µs khi hoạt động ở 56 MHz (1.17 µs tại 72 MHz)
* Điện áp cấp cho ADC từ 2.4V đến 3.6V
* Điện áp tưng tự vào ADC: VREF- ≤ VIN ≤ VREF+
* Tạo tín hiệu yêu cầu DMA trong suốt quá trình chuyển đổi của kênh regular

**Chú ý:** Nếu dòng ARM đang sử dụng có chân VREF- thì phải nối chân này với VSSA

2.8.3 Các chế độ hoạt động cơ bản của ADC

**a. chế độ chuyển đổi đơn (Single Conversion Mode)**

Trong chế độ chuyển đổi đơn ADC chỉ thực hiện 1 chuyển đổi và khi hoàn tất thì:

* Nếu đang sử dụng nhóm regular thì kết quả chuyển đổi sẽ được lưu trong thanh ghi 16 bit ADC\_DR, cờ EOC( End Of Conversion) được set lên ‘1’ và tạo ra tín hiệu yêu cầu ngắt nếu trước đó đã cho phép báo ngắt khi chuyển đổi hoàn tất.
* Nếu đang sử dụng nhóm injected thì kết quả chuyển đổi sẽ đươc lưu trong các thanh ghi 16 bit ADC\_DRJx, cờ JEOC(End Of Conversion Injected) được set lên ‘1’ và đồng thời tạo ra tín hiệu yêu cầu ngắt nếu trước đó đã cho phép báo ngắt khi chuyển đổi hoàn tất
* Sau khi thực hiện xong các công việc trên ADC sẽ dừng hoạt động.

**b. Chế độ liên tục (Continuous Mode)**

Trong chế độ liên tục ADC sẽ bắt đầu chuyển đổi mới ngay khi vừa hoàn tất chuyển đổi trước đó. Sau mỗi lần chuyển đổi xong thì:

* Nếu đang sử dụng nhóm regular thì kết quả chuyển đổi sẽ được lưu trong thanh ghi 16 bit ADC\_DR, cờ EOC( End Of Conversion) được set lên ‘1’ và tạo ra tín hiệu yêu cầu ngắt nếu trước đó đã cho phép báo ngắt khi chuyển đổi hoàn tất.
* Nếu đang sử dụng nhóm injected thì kết quả chuyển đổi sẽ đươc lưu trong các thanh ghi 16 bit ADC\_DRJx, cờ JEOC(End Of Conversion Injected) được set lên ‘1’ và đồng thời tạo ra tín hiệu yêu cầu ngắt nếu trước đó đã cho phép.

**c. Chế độ hoạt động song song 2 ADC (lDual ADC mode)**

**Chú ý:** chỉ hoạt động đối với các dòng ARM có từ 2 ADC trở lên.

Trong chế độ song song 2 ADC tín hiệu yêu cầu chuyển đổi được kích hoạt luân phiên hoặc đồng thời bởi ADC chủ( ADC1) tới ADC tớ( ADC2) phụ thuộc vào chế độ hoạt động được chọn bởi bit DUALMOD[2:0] trong thanh ghi ADC1\_CR1.

**Chú ý:**

* Ở chế độ hoạt động song song khi cấu hình tín hiệu yêu cầu chuyển đổi từ bên ngoài người dùng chỉ cài đặt tín hiệu này cho ADC chủ còn ADC tớ thì cấu hình tín hiệu yêu cầu bằng phần mềm để ngăn chặn việc ADC tớ chuyển đổi ngoài ý muốn.
* Phải cho phép cả 2 ADC chủ và tớ được nhận tín hiệu yêu cầu chuyển đổi từ bên ngoài.

Có 6 chế độ có thể cấu hình cho ADC hoạt động ở chế độ song song:

* Injected đồng thời (Injected simultaneous mode)
* Regular đồng thời(Regular simultaneous mode)
* Xen kẽ nhanh( Fast interleaved mode)
* Xen kẽ chậm(Low interleaved mode)
* Kích hoạt chuyển đổi luân phiên( Alternate trigger mode)
* Độc lập giữa 2 ADC (Independent mode)

Có thể kết hợp 6 chế độ trên để tạo thành các chế độ khác như:

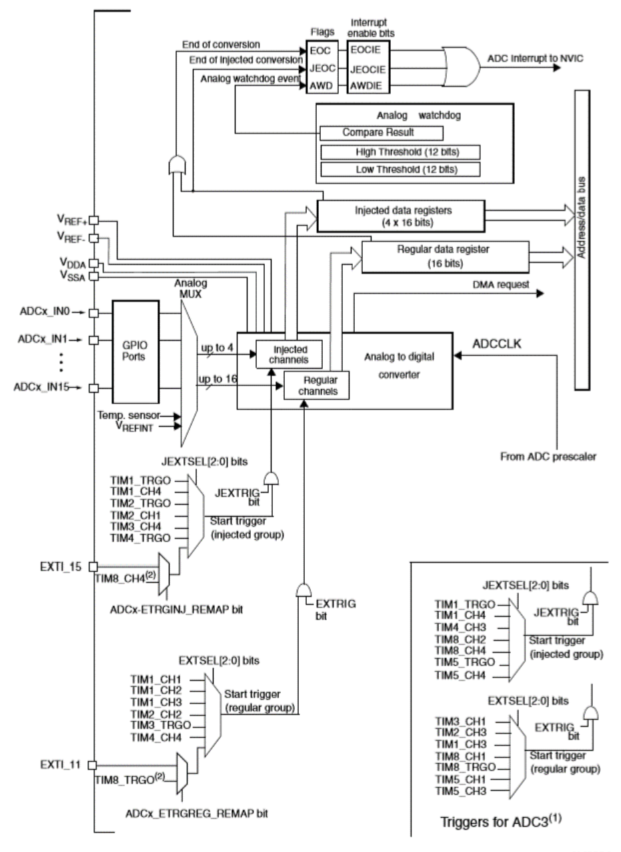
* Injected đồng thời + Regular đồng thời (Injected simultaneous mode + Regular simultaneous mode)
* Regular đồng thời + kích hoạt chuyển đổi luân phiên ( Regular simultaneous mode + Alternate trigger mode)
* Injected đồng thời + xen kẽ (Injected simultaneous mode + Interleaved mode)

**Chú ý:** trong chế độ song song để đọc được dữ liệu chuyển đổi của ADC tớ trong thanh ghi dữ liệu của ADC chủ thì cần phải cho phép DMA ngay cả khi nó không được sử dụng để vận chuyển kết quả chuyển đổi của nhóm regular.

**Chế độ injected đồng thời (Injected simultaneous mode):**

* Chế độ này cho phép cả 2 ADC chuyển đổi một nhóm các kênh injected cùng một lúc nhưng 2 ADC này không được chuyển đổi cùng một kênh tại cùng một thời điểm. Sau khi chuyển đổi xong kết quả chuyển đổi sẽ được lưu trong thanh ghi ADC\_JDRx của mỗi ADC và tín hiệu báo ngắt JEOC được tạo ra nếu trước đó đã cho phép ngắt ở 1 trong 2 ADC khi tất cả các kênh thuộc nhóm injected được chọn đã chuyển đổi xong.
* Nếu trước đó đã cho phép ngắt ở 1 trong 2 ADC thì sau khi tất cả các kênh thuộc nhóm injected chuyển đổi xong sẽ tạo ra yêu cầu ngắt.
* **Chú ý:** trong chế độ injected đồng thời người dùng cần cấu hình chính xác thời gian lấy mẫu cho 2 kênh mà sẽ được ADC1 và ADC2 lấy mẫu cùng lúc.

2.8.4. Sơ đồ khối và các chân của ADC



Sơ đồ khối của ADC

|  |  |  |
| --- | --- | --- |
| **Tên chân** | **Chức năng** | **Chú ý** |
| VREF+, VREF- | Ngõ vào cấp điện áp tham chiếu cho ADC | 2.4V <= VREF+ <= VDDA  VREF- = VSS |
| VDDA, VSSA | Nguồn cấp | 2.4V <= VDDA <= 3.6V  VSSA = VSS |
| ADCx\_IN[15:0] | Ngõ vào tín hiệu tương tự |  |

*Chức năng các chân của ADC*

16 kênh tương tự ngõ vào được chia ra làm 2 nhóm là regular và injected.

•Nhóm regular có thể được cài đặt tối đa 16 yêu cầu chuyển đổi.  
•Nhóm injected có thể được cài đặt tối đa 4 yêu cầu chuyển đổi.

Số kênh và thứ tự chuyển đổi của các kênh trong 2 nhóm regular và injected được cấu  
hình bằng phần mềm. Nếu những cấu hình này bị thay đổi trong khi ADC đang chuyển đổi thì công việc chuyển đổi hiện tại sẽ bị reset và đến khi có tín hiệu yêu cầu chuyển đổi tiếp theo thì ADC sẽ chuyển đổi theo cấu hình mới.

Cảm biến đo nhiệt độ bên trong vi điều khiển được nối với ngõ vào tương tự thứ 16  
(ADCx\_IN16). Điện áp tham chiếu nội được nối với kênh 17( ADCx\_IN17). Cả 2 kênh này có thể cấu hình hoạt động theo nhóm injected hay regular đều được. Khi sử dụng 2 kênh này ta cần chú ý là phải sử dụng ADC1 vì chỉ ADC1 mới kết nối với chúng

2.8.5 Các lệnh thông dụng liên quan đến ADC

|  |  |
| --- | --- |
| **Sử dụng thư viện “stm32f10x\_adc.h”** | |
| **Thông số** | **Giải thích** |
| **ADC\_InitTypeDef A;**  (Khai báo biến A thuộc kiểu ADC\_InitTypeDef) | |
| **A.ADC\_Mode = B;**  ( Lệnh chọn chế độ hoạt động cho ADC ) | |
| **B**:ADC\_Mode\_Independent ADC\_Mode\_RegInjecSimult ADC\_Mode\_RegSimult\_AlterTrig ADC\_Mode\_InjecSimult\_FastInterl ADC\_Mode\_InjecSimult\_SlowInterl ADC\_Mode\_InjecSimult ADC\_Mode\_RegSimult ADC\_Mode\_FastInterl ADC\_Mode\_SlowInterl ADC\_Mode\_AlterTrig | **B**: Chế độ hoạt động  Các ADC hoạt động độc lập với nhau  Regular, Injected đồng thời  Regular đồng thời + chuyển đổi luân phiên  Injected đồng thời + xen kẽ nhanh  Injected đồng thời + xen kẽ chậm  Injected đồng thời  Regular đồng thời  Xen kẽ nhanh  Xen kẽ chậm  Kích hoạt chuyển đổi luân phiên |
| **A.ADC\_ScanConvMode = C;**  ( Lệnh cho phép hoặc cấm chế độ Scan ) | |
| **C**:ENABLE DISABLE | **C**: Cho phép hoặc cấm Cho phép Cấm |
| **A.ADC\_ContinuousConvMode= D;**  ( Lệnh cho phép hoặc cấm chuyển đổi liên tục ) | |
| **D**:ENABLE DISABLE | **D**: Cho phép hoặc cấm  Cho phép  Cấm |
| **A.ADC\_DataAlign = E;**  ( Lệnh canh lề dữ liệu ) | |
| **E**:ADC\_DataAlign\_Right ADC\_DataAlign\_Left | **E**: Canh lề phải hoặc trái Canh lề phải Canh lề trái |

|  |  |
| --- | --- |
| **A.ADC\_NbrOfChannel = F;**  (Lệnh khai báo số kênh regular được sử dụng ) | |
| **F**:1  … 16 | **F**: Số kênh 1 kênh regular … 16 kênh regular |
| **A.ADC\_ExternalTrigConv = G;**  (Lệnh chọn nguồn tín hiệu yêu cầu chuyển đổi từ bên ngoài cho nhóm regular ) | |
| **G**:ADC\_ExternalTrigConv\_T1\_CC1 ADC\_ExternalTrigConv\_T1\_CC2 ADC\_ExternalTrigConv\_T2\_CC2 ADC\_ExternalTrigConv\_T3\_TRGO ADC\_ExternalTrigConv\_T4\_CC4 ADC\_ExternalTrigConv\_Ext\_IT11\_TIM8\_TRGO ADC\_ExternalTrigConv\_T1\_CC3 ADC\_ExternalTrigConv\_None ADC\_ExternalTrigConv\_T3\_CC1 ADC\_ExternalTrigConv\_T2\_CC3 ADC\_ExternalTrigConv\_T8\_CC1 ADC\_ExternalTrigConv\_T8\_TRGO ADC\_ExternalTrigConv\_T5\_CC1 ADC\_ExternalTrigConv\_T5\_CC3 | **G**: Nguồn tín hiệu yêu cầu chuyển đổi Capture Compare 1 của Timer 1 Capture Compare 2 của Timer 1 Capture Compare 2 của Timer 2 Trigger Out của Timer 3 Capture Compare 4 của Timer 4 EXIT line 11 và Trigger Out của Timer 8 Capture Compare 3 của Timer 1 Không sài yêu cầu chuyển đổi bên ngoài Capture Compare 1 của Timer 3 Capture Compare 3 của Timer 2 Capture Compare 1 của Timer 8 Trigger Out của Timer 8 Capture Compare 1 của Timer 5 Capture Compare 3 của Timer 5 |
| **ADC\_Init( H, &A);**  ( Lệnh cài đặt các thông số đã cấu hình trong biến A cho ADC ) | |
| **H**:ADC1 ADC2 ADC3 | **H**: ADC cần cấu hình Cấu hình cho ADC1 Cấu hình cho ADC2 Cấu hình cho ADC3 |

|  |  |
| --- | --- |
| **ADC\_RegularChannelConfig (A, B, C, D);**  ( Lệnh cấu hình nhóm regular cho ADC ) | |
| **A**:ADC1 ADC2 ADC3 **B**:ADC\_Channel\_0 … ADC\_Channel\_17 **C**:1 đến 16 **D**:ADC\_SampleTime\_1Cycles5 ADC\_SampleTime\_7Cycles5 ADC\_SampleTime\_13Cycles5 ADC\_SampleTime\_28Cycles5 ADC\_SampleTime\_41Cycles5 ADC\_SampleTime\_55Cycles5 ADC\_SampleTime\_71Cycles5 ADC\_SampleTime\_239Cycles5 | **A**: ADC cần cấu hình Cấu hình cho ADC1 Cấu hình cho ADC2 Cấu hình cho ADC3 **B**: Kênh cần cấu hình Kênh tương tự 0 … Kênh tương tự 17 **C**: Hạng của kênh trong nhóm regular Hạng từ 1 đến 16 **D**: Thời gian lấy mẫu 1.5 chu kỳ ADCCLK 7.5 chu kỳ ADCCLK 13.5 chu kỳ ADCCLK 28.5 chu kỳ ADCCLK 41.5 chu kỳ ADCCLK 55.5 chu kỳ ADCCLK 71.5 chu kỳ ADCCLK 239.5 chu kỳ ADCCLK |
| **ADC\_ExternalTrigConvCmd(A, B);**  ( Lệnh cho phép hoặc cấm yêu cầu chuyển đổi từ bên ngoài đối với nhóm regular) | |
| **A**:ADC1 ADC2 ADC3 **B**:ENABLE DISABLE | **A**: ADC cần cấu hình Cấu hình cho ADC1 Cấu hình cho ADC2 Cấu hình cho ADC3 **B**: Cho phép hoặc cấm Cho phép Cấm |
| **ADC\_DMACmd(A, B);**  ( Lệnh cho phép hoặc cấm ADC sử dụng DMA ) | |
| **A:**  ADC1  ADC2  ADC3  **B:**  ENABLE  DISABLE | **A:** ADC cần cấu hình  Cấu hình cho ADC1  Cấu hình cho ADC2  Cấu hình cho ADC3  **B:** cho phép hoặc cấm  Cho phép  Cấm ngắt |
| **ADC\_Cmd(A, B);**  ( Lệnh cho phép hoặc cấm ADC hoạt động ) | |
| **A**:ADC1 ADC2 ADC3 **B**:ENABLE DISABLE | **A**: ADC cần cấu hình Cấu hình cho ADC1 Cấu hình cho ADC2 Cấu hình cho ADC3 **B**: cho phép hoặc cấm Cho phép Cấm ngắt |
| **ADC\_ResetCalibration(A);**  **while(ADC\_GetResetCalibrationStatus(A));**  **ADC\_StartCalibration(A);**  **while(ADC\_GetCalibrationStatus(A));**  ( Các lệnh reset và hiệu chỉnh các thông số của ADC ) | |
| **A**:ADC1 ADC2 ADC3 | **A**: ADC cần cấu hình Cấu hình cho ADC1 Cấu hình cho ADC2 Cấu hình cho ADC3 |
| **ADC\_SoftwareStartConvCmd(A, B);**  ( Lệnh cho phép hoặc cấm bắt đầu chuyển đổi bằng phần mềm- sử dụng khi không chọn yêu cầu chuyển đổi từ bên ngoài ) | |
| **A**:ADC1 ADC2 ADC3 **B**:ENABLE DISABLE | **A**: ADC cần cấu hình Cấu hình cho ADC1 Cấu hình cho ADC2 Cấu hình cho ADC3 **B**: cho phép hoặc cấm Cho phép Cấm ngắt |

|  |  |
| --- | --- |
| **ADC\_GetFlagStatus(A, B) ;**  ( Lệnh đọc cờ trạng thái của ADC) | |
| **A**:ADC1 ADC2 ADC3 **B**:ADC\_FLAG\_AWD ADC\_FLAG\_EOC ADC\_FLAG\_JEOC ADC\_FLAG\_JSTRT ADC\_FLAG\_STRT | **A**: ADC cần đọc Đọc ADC1 Đọc ADC2 Đọc ADC3 **B**: Cờ cần đọc Cờ vượt ngưỡng của bộ giám sát tương tự Cờ báo chuyển đổi xong của nhóm regular Cờ báo chuyển đổi xong của nhóm injected Cờ báo bắt đầu chuyển đổi nhóm injected Cờ báo bắt đầu chuyển đổi nhóm regular |
| **ADC\_ClearFlag(A, B) ;**  ( Lệnh xóa cờ trạng thái của ADC) | |
| **A**:ADC1 ADC2 ADC3 **B**:ADC\_FLAG\_AWD ADC\_FLAG\_EOC ADC\_FLAG\_JEOC ADC\_FLAG\_JSTRT ADC\_FLAG\_STRT | **A**: ADC cần xóa ADC1 ADC2 ADC3 **B**: Cờ cần xóa Cờ vượt ngưỡng của bộ giám sát tương tự Cờ báo chuyển đổi xong của nhóm regular Cờ báo chuyển đổi xong của nhóm injected Cờ báo bắt đầu chuyển đổi nhóm injected Cờ báo bắt đầu chuyển đổi nhóm regular |
| **unsinged short A;**  **A = ADC\_GetConversionValue(B);**  ( Lệnh đọc giá trị chuyển đổi được của kênh regular ) | |
| **B**:ADC1 ADC2 ADC3 | **B:** ADC cần đọc Đọc ADC1 Đọc ADC2 Đọc ADC3 |
| **unsinged long A;**  **A = ADC\_GetDualModeConversionValue();**  ( Lệnh đọc giá trị chuyển đổi được ADC1 và ADC2 ở chế độ song song chủ tớ) | |

|  |  |
| --- | --- |
| **ADC\_InjectedChannelConfig(A, B, C, D);**  ( Lệnh cấu hình cho nhóm Injected ) | |
| **A:** ADC1 ADC2 ADC3  **B:** ADC\_Channel\_0 … ADC\_Channel\_17 **C:**  1  …  4  **D:**  ADC\_SampleTime\_1Cycles5  ADC\_SampleTime\_7Cycles5  ADC\_SampleTime\_13Cycles5  ADC\_SampleTime\_28Cycles5  ADC\_SampleTime\_41Cycles5  ADC\_SampleTime\_55Cycles5  ADC\_SampleTime\_71Cycles5  ADC\_SampleTime\_239Cycles5 | **A: ADC cần đọc** Đọc ADC1 Đọc ADC2 Đọc ADC3 **B: Kênh cần cấu hình**  Kênh tương tự 0  …  Kênh tương tự 17  **C: Hạng của kênh trong nhóm injected**  Hạng 1  ...  Hạng 4  **D: Thời gian lấy mẫu**  1.5 chu kỳ ADCCLK  7.5 chu kỳ ADCCLK  13.5 chu kỳ ADCCLK  28.5 chu kỳ ADCCLK  41.5 chu kỳ ADCCLK  55.5 chu kỳ ADCCLK  71.5 chu kỳ ADCCLK  239.5 chu kỳ ADCCLK |
| **ADC\_InjectedSequencerLengthConfig(A, B);**  ( Lệnh khai báo số kênh thuộc nhóm Injected được sử dụng ) | |
| **A:** ADC1 ADC2 ADC3  **B:** 1  …  4 | **A: ADC cần xóa** ADC1 ADC2 ADC3  **B:** Số lượng kênh Injected được sử dụng1 kênh…4 kênh |
| **ADC\_ExternalTrigInjectedConvConfig(A, B);**  ( Lệnh chọn tín hiệu yêu cầu chuyển đổi từ bên ngoài cho nhóm Injected ) | |
| **A:** ADC1 ADC2 ADC3  **B:**  ADC\_ExternalTrigInjecConv\_T1\_TRGO  ADC\_ExternalTrigInjecConv\_T1\_CC4  ADC\_ExternalTrigInjecConv\_T2\_TRGO  ADC\_ExternalTrigInjecConv\_None  … | **A:** ADC cần đọc Đọc ADC1 Đọc ADC2 Đọc ADC3  **B: Nguồn tín hiệu yêu cầu chuyển đổi**  Trigger Out của Timer 1  Capture Compare 2 của Timer 1  Trigger Out của Timer 2  Không dùng yêu cầu chuyển đổi ngoại  … |

|  |  |
| --- | --- |
| **ADC\_AutoInjectedConvCmd(A, B);**  ( Lệnh cho phép hoặc cấm chế độ AutoInjected) | |
| **A:** ADC1 ADC2 ADC3  **B**:  ENABLE  DISABLE | **A: ADC cần đọc** Đọc ADC1 Đọc ADC2 Đọc ADC3 **B: Cho phép hoặc cấm**  Cho phép  Cấm |
| **unsinged short A;**  **A= ADC\_GetInjectedConversionValue(B, C);**  ( Lệnh đọc giá trị chuyển đổi được của kênh injected ) | |
| **B:** ADC1 ADC2 ADC3  **C:**  ADC\_InjectedChannel\_1  …  ADC\_InjectedChannel\_4 | **B: ADC cần xóa** ADC1 ADC2 ADC3  **C: Kênh injected cần đọc**  Kênh injected 1  …  Kênh injected 4 |

2.8.6 Lập trình

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ADC1** | **ADC2** | **ADC3** | **GPIO\_Pin\_x** | **ADC\_Channel\_x** |
| 1 | 1 | 1 | PA0 | ADC\_Channel\_0 |
| 1 | 1 | 1 | PA1 | ADC\_Channel\_1 |
| 1 | 1 | 1 | PA2 | ADC\_Channel\_2 |
| 1 | 1 | 1 | PA3 | ADC\_Channel\_3 |
| 1 | 1 | 0 | PA4 | ADC\_Channel\_4 |
| 1 | 1 | 0 | PA5 | ADC\_Channel\_5 |
| 1 | 1 | 0 | PA6 | ADC\_Channel\_6 |
| 1 | 1 | 0 | PA7 | ADC\_Channel\_7 |
| 1 | 1 | 0 | PB8 | ADC\_Channel\_8 |
| 1 | 1 | 0 | PB9 | ADC\_Channel\_9 |
| 1 | 1 | 1 | PC0 | ADC\_Channel\_10 |
| 1 | 1 | 1 | PC1 | ADC\_Channel\_11 |
| 1 | 1 | 1 | PC2 | ADC\_Channel\_12 |
| 1 | 1 | 1 | PC3 | ADC\_Channel\_13 |
| 1 | 1 | 0 | PC4 | ADC\_Channel\_14 |
| 1 | 1 | 0 | PC5 | ADC\_Channel\_15 |

Pin và channel tương ứng

**a. Ví dụ về chế độ chuyển đổi đơn (Single Conversion Mode)**

**Ví dụ:** Viết chương trình điều khiển ADC1 tự động chuyển đổi bằng phần mềm 1 lần duy nhất. Sử dụng ADC\_Channel\_0, ADC1.

|  |
| --- |
| // Cấu hình GPIO cho pin PA0 (ADC\_Channel\_0)  void GPIO\_Config(void){  GPIO\_InitTypeDef GPIO\_InitStructure;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN;  GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0;  GPIO\_Init(GPIOA, &GPIO\_InitStructure);  } |

|  |
| --- |
| // Cấu hình ADC  void ADC\_Config(void){  ADC\_InitTypeDef ADC\_InitStructure;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);    ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;  ADC\_InitStructure.ADC\_ScanConvMode = DISABLE;  ADC\_InitStructure.ADC\_ContinuousConvMode = DISABLE;  ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;  ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;  ADC\_InitStructure.ADC\_NbrOfChannel = 1;  ADC\_Init(ADC1, &ADC\_InitStructure);    ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_0, 1, ADC\_SampleTime\_55Cycles5);  ADC\_Cmd(ADC1, ENABLE);  ADC\_ResetCalibration(ADC1);  while(ADC\_GetResetCalibrationStatus(ADC1));  ADC\_StartCalibration(ADC1);  while(ADC\_GetCalibrationStatus(ADC1));  ADC\_SoftwareStartConvCmd(ADC1, ENABLE);  } |

|  |
| --- |
| // Hàm đọc giá trị ADC  unsigned int ADCx\_Read(void){  return(ADC\_GetConversionValue(ADC1));  } |

|  |
| --- |
| // Sử dụng hàm Map của arduino để chuyển đổi dải do.  int Map(int x, int in\_min, int in\_max, int out\_min, int out\_max) {  return (x - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min;  } |

**b. Ví dụ về chế độ chuyển đổi liên tục (Continuous Mode)**

**Ví dụ:** Viết chương trình điều khiển ADC1 tự động chuyển đổi liên lục bằng phần mềm. Sử dụng ADC\_Channel\_0.

|  |
| --- |
| // Cấu hình các thông số tương tự như chế độ chuyển đổi đơn chỉ khác lệnh:  // Thay đỏi DISABLE thành ENABLE.  ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE; |

**c. Ví dụ về chế độ song song 2 ADC – injected dồng thời (Injected simultaneous mode)**

**Ví dụ:** Viết chương trình điều khiển ADC1 và ADC2 chuyển đỏi liên tục 2 kênh 0 và 1 theo chế độ inject đồng thời.

|  |
| --- |
| // Cấu hình GPIO cho pin PA0 (ADC\_Channel\_0). PA1 (ADC\_Channel\_1)  void GPIO\_Config(void){  GPIO\_InitTypeDef GPIO\_InitStructure;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN;  GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0 | GPIO\_Pin\_1;  GPIO\_Init(GPIOA, &GPIO\_InitStructure);  } |

|  |
| --- |
| Void ADC\_Config(void){  ADC\_InitTypeDef ADC\_InitStructure;  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC2, ENABLE);    ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE;  ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;  ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;  ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;  ADC\_InitStructure.ADC\_NbrOfChannel = 1;  ADC\_InitStructure.ADC\_ScanConvMode = DISABLE;  ADC\_Init(ADC1, &ADC\_InitStructure);  ADC\_Init(ADC2, &ADC\_InitStructure);  ADC\_InjectedSequencerLengthConfig(ADC1, 1);// tong so kenh can doc  ADC\_InjectedSequencerLengthConfig(ADC2, 1);  ADC\_InjectedChannelConfig(ADC1, ADC\_Channel\_0, 1, ADC\_SampleTime\_239Cycles5);  ADC\_InjectedChannelConfig(ADC2, ADC\_Channel\_1, 2, ADC\_SampleTime\_239Cycles5);    ADC\_ExternalTrigInjectedConvConfig(ADC1, ADC\_ExternalTrigInjecConv\_None);  ADC\_ExternalTrigInjectedConvConfig(ADC2, ADC\_ExternalTrigInjecConv\_None);  ADC\_AutoInjectedConvCmd(ADC1, ENABLE);  ADC\_AutoInjectedConvCmd(ADC2, ENABLE);  ADC\_Cmd (ADC1, ENABLE);  ADC\_Cmd (ADC2, ENABLE);  ADC\_ResetCalibration(ADC1); // Reset hieu Chuan ADC  while(ADC\_GetResetCalibrationStatus(ADC1));  ADC\_StartCalibration(ADC1); // Hieu chuan lai  while(ADC\_GetCalibrationStatus(ADC1));  ADC\_SoftwareStartInjectedConvCmd (ADC1 , ENABLE);  ADC\_ResetCalibration(ADC2); // Reset hieu Chuan ADC  while(ADC\_GetResetCalibrationStatus(ADC2));  ADC\_StartCalibration(ADC2); // Hieu chuan lai  while(ADC\_GetCalibrationStatus(ADC2));  ADC\_SoftwareStartInjectedConvCmd (ADC2 , ENABLE);  } |

|  |
| --- |
| unsigned int ADCx\_Read1 (void){  return(ADC\_GetInjectedConversionValue(ADC1, ADC\_InjectedChannel\_1));  }  unsigned int ADCx\_Read2 (void){  return(ADC\_GetInjectedConversionValue(ADC2, ADC\_InjectedChannel\_2));  } |