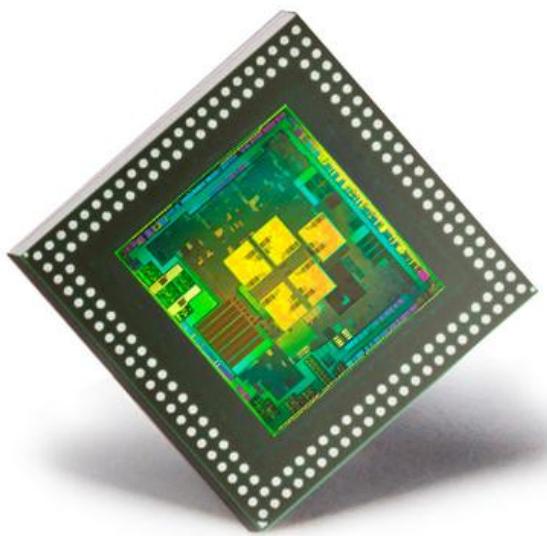


# LẬP TRÌNH CĂN BẢN ARM CORTEX M3 STM32F103C8T6



By

Nguyễn Ngọc Hà

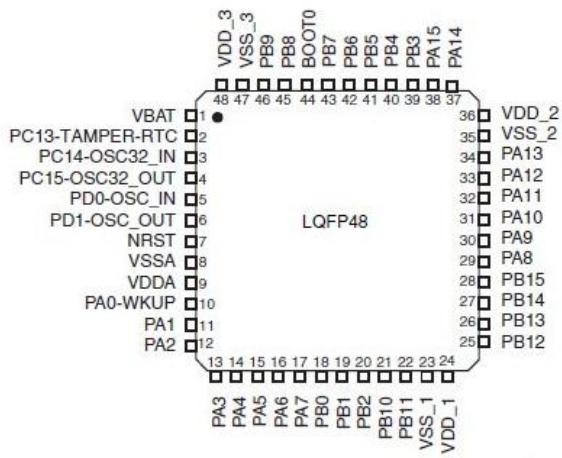
Email : [hanguyen92205@gmail.com](mailto:hanguyen92205@gmail.com)

Skype : hanguyen92205

ĐT: 0982 969 872

TpHCM, Tháng 5, 2014

## VỊ ĐIỀU KHIỂN ARM STM32F103C8T6



ai14393b

Manufacturer Part Number	STM32F103C8T6
Description	MCU ARM 64KB FLASH MEM 48-LQFP
Vendor	STMicroelectronics
Category	Integrated Circuits (ICs)
Program Memory Size	64KB (64K x 8)
RAM Size	20K x 8
Number of I/O	37
Package / Case	48-LQFP
Speed	72MHz
Oscillator Type	Internal
Packaging	Tray
Program Memory Type	FLASH
EEPROM Size	-
Core Processor	ARM® Cortex®-M3
Data Converters	A/D 10x12b
Core Size	32-Bit
Operating Temperature	-40°C ~ 85°C
Connectivity	CAN, I <sup>2</sup> C, IrDA, LIN, SPI, UART/USART, USB
Peripherals	DMA, Motor Control PWM, PDR, POR, PVD, PWM, Temp Sensor, WDT
Voltage - Supply (Vcc/Vdd)	2 V ~ 3.6 V
Lead Free Status	Lead Free
RoHS Status	RoHS Compliant

Other Names	STM32F103C8T6 STM32F103C8T6 497 6063 ND 4976063ND 497-6063
-------------	--

# MỤC LỤC

<b>LỜI MỞ ĐẦU .....</b>	<b>1</b>
<b>CHƯƠNG I : TÌM HIỂU VỀ ARM CORTEX M3 STM32F103 .....</b>	<b>2</b>
1.1    Giới thiệu về ARM Cortex M3 STM32F103.....	2
1.1.1    Cortex là gì ? .....	2
1.1.2    Đặc điểm nổi bật của STM32 .....	2
1.2    Tổng quát về ARM Cortex M3 STM32F103 .....	4
1.2.1    Các phiên bản cấu trúc ARM .....	5
1.2.2    Bộ xử lý và đơn vị xử lý trung tâm Cortex .....	5
1.2.3    Đơn vị xử lý trung tâm Cortex ( Cortex CPU ) .....	5
1.2.4    Bộ xử lý Cortex.....	7
1.2.5    Các chế độ năng lượng .....	11
1.2.6    Kiểu đóng gói chip và kiểu chân linh kiện.....	12
1.2.7    Nguồn cung cấp điện.....	12
1.2.8    Mạch Reset .....	13
1.3    Kiến trúc hệ thống .....	13
1.3.1    Cấu trúc bộ nhớ.....	14
1.3.2    Tối đa hiệu năng.....	14
1.4    Các ngoại vi.....	18
1.4.1    Ngoại vi đa dụng.....	18
1.4.2    Kết nối với các giao tiếp khác .....	28
1.5    Chế độ tiêu thụ năng lượng thấp .....	32
1.5.1    Chế độ bình thường – RUN Mode.....	32
1.5.2    Các chế độ sử dụng công suất tiêu thụ thấp .....	33
1.5.3    Standby .....	34
1.5.5    Hỗ trợ Debug .....	34
1.6    Tính an toàn.....	35
1.6.1    Reset Control .....	35
1.6.2    Kiểm tra điện áp nguồn .....	35
1.6.3    Hệ thống an toàn xung nhịp.....	36
1.6.4    Watchdogs .....	36
1.6.5    Tính năng ngoại vi .....	37
1.7    Flash.....	38
1.7.1    Lập trình và đảm bảo an toàn cho FLASH nội.....	38
1.7.2    Hoạt động xóa và ghi .....	38
1.7.3    Các byte Opteron .....	38

<b>CHƯƠNG II: THIẾT KẾ KIT THÍ NGHIỆM ARM STM32F103 .....</b>	39
2.1 Giới thiệu .....	39
2.2 Thiết kế mạch nguyên lý .....	39
2.3 Thiết kế mạch in .....	42
<b>CHƯƠNG III: LẬP TRÌNH ỦNG DỤNG CHO KIT STM32F103 .....</b>	45
3. 1 Hướng dẫn cơ bản cho một ứng dụng với KIT STM32F103.....	45
3.1.1 Các bước tạo một Project mới trên Keil C MDK .....	45
3.1.2 Nạp chương trình vào vi điều khiển.....	52
3.2 Lập trình ứng dụng.....	54
3.2.1 Nguồn Clock trong STM32 .....	54
3.2.2 Tạo thư viện delay sử dụng SYSTICK trong STM32.....	56
3.2.3 Lập trình GPIO điều khiển led đơn và thư viện GPIO .....	59
3.2.5 Lập trình hiển thị Led 7 đoạn .....	64
3.2.6 Lập trình hiển thị LCD 1602 .....	66
3.2.7 Giao tiếp USART với KIT STM32F103C8T6 .....	67
3.2.8 Đo giá trị ADC và hiển thị LCD .....	69
3.2.9 Giao tiếp cảm biến nhiệt độ DS18B20.....	71
3.2.10 Giao tiếp I2C với IC EEPROM 24C02 .....	72
<b>CHƯƠNG IV : THIẾT KẾ WEB HOCARM.NET .....</b>	75
4.1 Giới thiệu về mã nguồn mở Nukeviet.....	75
4.2 Thiết kế giao diện website .....	76
<b>TÀI LIỆU THAM KHẢO .....</b>	80

# DANH MỤC HÌNH

Hình 1.1. Kiến trúc vi xử lí ARM Cortex-M3 .....	2
Hình 1.2. Kiến trúc của STM32 nhánh Performance và Access .....	3
Hình 1.3. Đặc điểm của bốn nhánh trong họ STM32.....	4
Hình 1.4. Các phiên bản kiến trúc của lõi ARM.....	5
Hình 1.5. Kiến trúc đường ống của ARM Cortex-M3 .....	6
Hình 1.6 Kiến trúc load và store của ARM Cortex-M3 .....	6
Hình 1.7. Bản đồ bộ nhớ tuyến tính 4Gbyte của bộ xử lí Cortex-M3.....	7
Hình 1.8. Cấu trúc của NVIC trong bộ xử lí Cortex .....	9
Hình 1.9. Đáp ứng thời gian khi một ngắt bất kì xảy ra của Cortex-M3 .....	9
Hình 1.10. Đáp ứng thời gian khi hai ngắt xảy ra đồng thời của Cortex-M3.....	10
Hình 1.11. Hệ thống gỡ lõi CoreSight bên trong Cortex.....	12
Hình 1.12. Các miền năng lượng bên trong STM32 .....	12
Hình 1.13. Cách bố trí tụ chống nhiễu cho STM32 .....	13
Hình 1.14. Đặc tính của mạch reset bên trong STM32 .....	13
Hình 1.15. Cấu trúc Bus.....	14
Hình 1.16. Vùng nhớ Flash trên STM32 .....	14
Hình 1.17 STM32 bao gồm 2 bộ tạo xung nhịp nội và 2 bộ tạo xung nhịp ngoại thêm vào đó là bộ vòng khóa pha( Phase Lock Loop-PLL).....	15
Hình 1.18 Mỗi thao tác bộ nhớ DMA bao gồm 4 giai đoạn. ....	16
Hình 1.19 Bộ DMA được thiết kế cho truyền dữ liệu tốc độ và kích thước nhỏ.....	16
Hình 1.20 Ở giai đoạn Bus Access CPU sẽ có 3 chu kỳ rảnh. ....	17
Hình 1.21 Mỗi kênh DMA được gán với ngoại vi nhất định. Khi được kích hoạt, các thiết bị ngoại vi sẽ điều khiển bộ DMA tương ứng.....	17
Hình 1.22 bộ ADC STM32 .....	20
Hình 1.23 Các mức thoái gian chuyển đổi ADC.....	20
Hình 1.24 Analogue Watchdog có thể dùng giám sát một hay nhiều kênh ADC với vùng ngưỡng được cấu hình bởi người dùng .....	21
Hình 1.25 Hai thanh ghi điều khiển cấu hình hoạt động của khối ADC .....	21
Hình 1.26 Cả hai khối ADC cùng hoạt động ở cùng chế độ Regular hoặc Injected ....	22
Hình 1.27 Cả hai khối cùng hoạt động ở 2 chế độ Regular và Injected xen kẽ .....	22
Hình 1.28 Hoạt động xen kẽ nhanh và chậm Regular .....	23
Hình 1.29 Chế độ kích hoạt thay thế .....	23
Hình 1.30 4 khối định thời với các thanh ghi 16-bit Prescaler .....	24
Hình 1.31 Mỗi một kênh Capture/Compare đều có một thanh ghi đơn cấu hình chế độ hoạt động.....	24
Hình 1.32 4 kênh vào của khối Capture có các bộ lọc dữ liệu và phát hiện xung cạnh riêng. ....	25

Hình 1.33 Ngõ vào Capture và xung PWM .....	25
Hình 1.34 Mỗi khối Timer đều có khả năng tạo ra các xung PWM .....	26
Hình 1.35 Chế độ One Pulse .....	27
Hình 1.36 Mỗi khối Timer có đầu vào là các xung sự kiện .....	27
Hình 1.37 Khối RTC có thể lấy nguồn xung nhịp từ LSI, LSE và HSE .....	28
Hình 1.38 giao tiếp SPI .....	28
Hình 1.39 Giao tiếp I2C .....	29
Hình 1.40 Kiểm tra lỗi trên I2C .....	29
Hình 1.41 Giao diện USART có khả năng hỗ trợ giao tiếp không đồng bộ UARTS, modem cũng như giao tiếp hồng ngoại và Smartcard .....	30
Hình 1.42 Hỗ trợ giao tiếp ở chế độ half-duplex dựa trên một đường truyền .....	30
Hình 1.43 Giao tiếp smartcard và hồng ngoại .....	30
Hình 1.44 Hỗ trợ giao tiếp đồng bộ SPI .....	31
Hình 1.45 Khối điều khiển CAN .....	31
Hình 1.46 Khối CAN có 3 mailbox cho truyền dữ liệu với đánh nhãn thời gian tự động cho chuẩn TTCAN .....	31
Hình 1.47 Giao tiếp USB 2.0 .....	32
Hình 1.48 Thời gian đánh thức trong chế độ Stop dài nhất là 5,5 us với bộ điều áp chạy bình thường và 7,3 us với bộ điều áp trong chế độ công suất thấp .....	33
Hình 1.49 Trong chế độ Standby tiêu thụ điện năng là 2uA với thời gian đánh thức là 50uS .....	34
Hình 1.50 STM32 có thể có nhiều tín hiệu điều khiển để đưa hệ thống về trạng thái Reset. Trạng thái hệ thống sẽ được lưu lại trong thanh ghi RCC .....	35
Hình 1.51 Đầu ra của PVD được kết nối với 16 đường của bộ ngắt ngoài .....	35
Hình 1.52 Hệ thống dao động thạch anh nội .....	36
Hình 1.53 Windowed Watchdog .....	36
Hình 1.54 Independent Watchdog .....	37
Hình 2.1 Sơ đồ nguyên lý KIT STM32F103 .....	41
Hình 2.2 Mạch in lớp TOP .....	42
Hình 2.3 Mạch in lớp BOTTOM .....	42
Hình 2.4 PCB 3D lớp TOP .....	43
Hình 2.5 PCB 3 D Lớp BOTTOM .....	43
Hình 2.6 Mạch in 3D .....	44
Hình 2.7. Mạch sau khi lắp ráp .....	44
Hình 2.7 Các thư mục của một Project .....	45
Hình 3.1 Khối hiển thị led đơn .....	62
Hình 3.2 Khối hiển thị LED 7 đoạn .....	64
Hình 3.3 Mã hiển thị Led 7 đoạn .....	64
Hình 3.4 Khối hiển thị LCD16x2 .....	66

Hình 3.5 Khối giao tiếp USART .....	67
Hình 3.6 Khối giao tiếp ADC .....	69
Hình 3.7 Khối giao tiếp cảm biến nhiệt độ DS18B20.....	71
Hình 3.8 Khối giao tiếp I2C với EEPROM .....	72
Hình 4.1 Giao diện trang quản trị.....	76
Hình 4.2 Trang chủ website hocarm.net.....	76
Hình 4.3 Giao diện trang bài viết .....	77
Hình 4.4 Giao diện trang download .....	78
Hình 4.5 Bố cục trang web.....	78

## MỞ ĐẦU

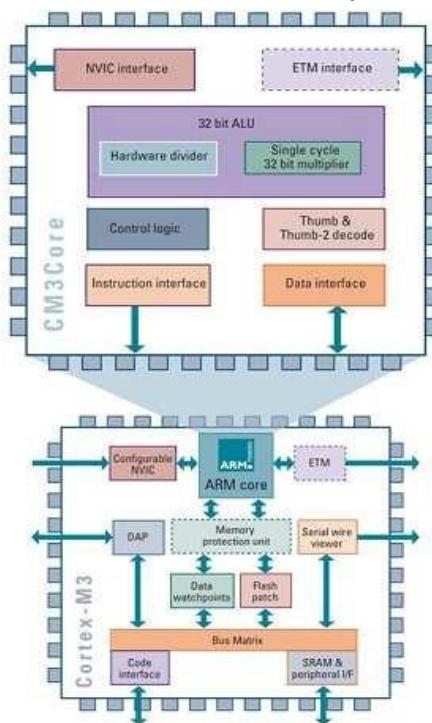
Trong vài năm trở lại đây, một trong những xu hướng chủ yếu trong các thiết kế với vi điều khiển là sử dụng các chip lõi ARM như một vi điều khiển đa dụng. Ngày nay các nhà sản xuất IC đưa ra thị trường rất nhiều dòng vi điều khiển sử dụng lõi ARM. Tập đoàn ST Microelectronic đã cho ra mắt dòng STM32, vi điều khiển đầu tiên dựa trên nền lõi ARM Cortex-M3 thế hệ mới do hãng ARM thiết kế, lõi ARM Cortex-M3 là sự cải tiến của lõi ARM7 truyền thống, từng mang lại sự thành công vang dội cho công ty ARM. Dòng STM32 thiết lập các tiêu chuẩn mới về hiệu suất, chi phí, cũng như khả năng đáp ứng các ứng dụng tiêu thụ năng lượng thấp và tính điều khiển thời gian thực khắt khe. Với ứng dụng rộng rãi: từ điện tử dân dụng, xe hơi đời mới, game, mobile , laptop, chỗ nào ARM cũng có mặt. Dòng STM32 tiêu thụ năng lượng cực thấp trong khi đó hiệu suất cực cao và việc lập trình cũng rất dễ dàng. Với sự đồ sộ về ngoại vi (GPIO, I2C, SPI, ADC, USB, Ethernet, CAN....), ST cung cấp cho chúng ta các thư viện trực tiếp cho mỗi dòng ARM (gọi là CMSIS - Cortex Microcontroller Software Interface Standard), nhiệm vụ của chúng ta không thể dễ dàng hơn: khai báo và sử dụng mà thôi. Mà giá của con này cũng khá rẻ so với các dòng chip hiện có trên thị trường.

## CHƯƠNG I : TÌM HIỂU VỀ ARM CORTEX M3 STM32F103

### 1.1 Giới thiệu về ARM Cortex M3 STM32F103

#### 1.1.1 Cortex là gì ?

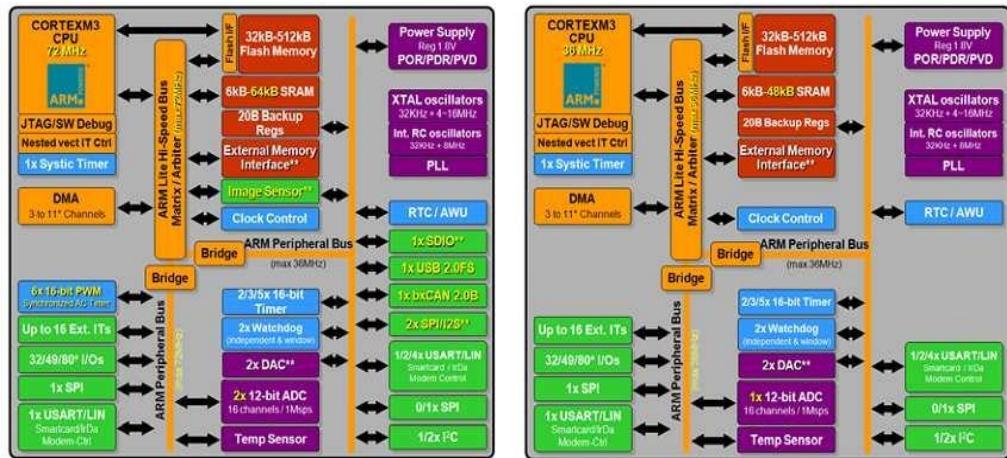
Dòng ARM Cortex là một bộ xử lí thế hệ mới đưa ra một kiến trúc chuẩn cho nhu cầu đa dạng về công nghệ. Không giống như các chip ARM khác, dòng Cortex là một lõi xử lí hoàn thiện, đưa ra một chuẩn CPU và kiến trúc hệ thống chung. Dòng Cortex gồm có 3 phân nhánh chính: dòng A dành cho các ứng dụng cao cấp, dòng R dành cho các ứng dụng thời gian thực như các đầu đọc và dòng M dành cho các ứng dụng vi điều khiển và chi phí thấp. STM32 được thiết kế dựa trên dòng Cortex-M3, dòng Cortex-M3 được thiết kế đặc biệt để nâng cao hiệu suất hệ thống, kết hợp với tiêu thụ năng lượng thấp, CortexM3 được thiết kế trên nền kiến trúc mới, do đó chi phí sản xuất đủ thấp để cạnh tranh với các dòng vi điều khiển 8 và 16-bit truyền thống.



Hình 1.1. Kiến trúc vi xử lí ARM Cortex-M3

#### 1.1.2 Đặc điểm nổi bật của STM32

ST đã đưa ra thị trường 4 dòng vi điều khiển dựa trên ARM7 và ARM9, nhưng STM32 là một bước tiến quan trọng trên đường cong chi phí và hiệu suất (price/performance), giá chỉ gần 1 Euro với số lượng lớn, STM32 là sự thách thức thật sự với các vi điều khiển 8 và 16-bit truyền thống. STM32 đầu tiên gồm 14 biến thể khác nhau, được phân thành hai nhóm: dòng Performance có tần số hoạt động của CPU lên tới 72Mhz và dòng Access có tần số hoạt động lên tới 36Mhz. Các biến thể STM32 trong hai nhóm này tương thích hoàn toàn về cách bố trí chân (pin) và phần mềm, đồng thời kích thước bộ nhớ FLASH ROM có thể lên tới 128K và 20K SRAM.



Hình 1.2. Kiến trúc của STM32 nhánh Performance và Access

Dòng STM32 có hai nhánh, nhánh Performance hoạt động với xung nhịp lên đến 72Mhz và có đầy đủ các ngoại vi, nhánh Access hoạt động với xung nhịp tối đa 36Mhz và có ít ngoại vi hơn so với nhánh Performance.

#### a. Sự tinh vi

Thoạt nhìn thì các ngoại vi của STM32 cũng giống như những vi điều khiển khác, như hai bộ chuyển đổi ADC, timer, I2C, SPI, CAN, USB và RTC. Tuy nhiên mỗi ngoại vi trên đều có rất nhiều đặc điểm thú vị. Ví dụ như bộ ADC 12-bit có tích hợp một cảm biến nhiệt độ để tự động hiệu chỉnh khi nhiệt độ thay đổi và hỗ trợ nhiều mode chuyển đổi. Mỗi bộ timer có 4 khối capture compare, mỗi khối timer có thể liên kết với các khối timer khác để tạo ra một mảng các timer tinh vi.

#### b. Sự an toàn

Ngày nay các ứng dụng hiện đại thường phải hoạt động trong môi trường khắc khe, đòi hỏi tính an toàn cao, cũng như đòi hỏi sức mạnh xử lý và càng nhiều thiết bị ngoại vi tinh vi. Để đáp ứng các yêu cầu khắc khe đó, STM32 cung cấp một số tính năng phần cứng hỗ trợ các ứng dụng một cách tốt nhất. Chúng bao gồm một bộ phát hiện điện áp thấp, một hệ thống bảo vệ xung clock và hai bộ watchdogs.

#### c. Tính bảo mật

Một trong những yêu cầu khắc khe khác của thiết kế hiện đại là nhu cầu bảo mật mã chương trình để ngăn chặn sao chép trái phép phần mềm. Bộ nhớ Flash của STM32 có thể được khóa để chống truy cập đọc Flash thông qua cổng debug. Khi tính năng bảo vệ đọc được kích hoạt, bộ nhớ Flash cũng được bảo vệ chống ghi để ngăn chặn mã không tin cậy được chèn vào bảng vector ngắt.

#### d. Phát triển phần mềm

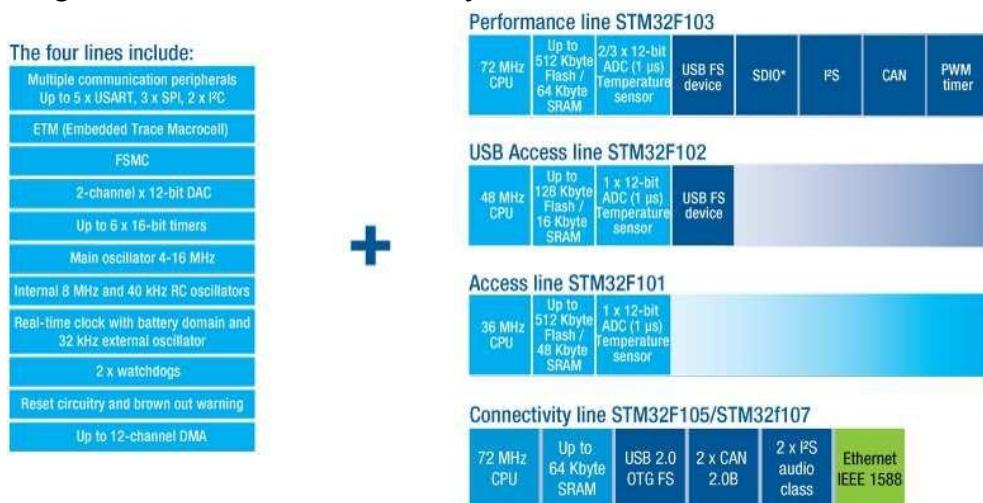
Nếu như đã sử dụng một vi điều khiển dựa trên lõi ARM, các công cụ phát triển đã được hỗ trợ tập lệnh Thumb-2 và dòng Cortex. Ngoài ra ST cũng cung cấp một thư viện điều khiển thiết bị ngoại vi, một bộ thư viện phát triển USB như là một thư viện ANSI C và mã nguồn đó là tương thích với các thư viện trước đó được công bố cho vi điều

khiển STR7 và STR9. Có rất nhiều RTOS mã nguồn mở và thương mại và middleware (TCP/IP, hệ thống tập tin, v.v.) hỗ trợ cho họ Cortex. Dòng Cortex-M3 cũng đi kèm với một hệ thống gỡ lỗi hoàn toàn mới gọi là CoreSight. Truy cập vào hệ thống CoreSight thông qua cổng truy cập Debug (Debug Access Port), cổng này hỗ trợ kết nối chuẩn JTAG hoặc giao diện 2 dây (serial wire-2 Pin), cũng như cung cấp trình điều khiển chạy gỡ lỗi.

### e. Dòng Performance và Access của STM32

Họ STM32 có hai nhánh đầu tiên riêng biệt: dòng Performance và dòng Access. Dòng Performance tập hợp đầy đủ các thiết bị ngoại vi và chạy với xung nhịp tối đa 72MHz. Dòng Access có các thiết bị ngoại vi ít hơn và chạy tối đa 32MHz. Quan trọng hơn là cách bố trí chân (pins layout) và các kiểu đóng gói chip (package type) là như nhau giữa dòng Access và dòng Performance. Điều này cho phép các phiên bản khác nhau của STM32 được hoán vị mà không cần phải sửa đổi sáp lại footprint (mô hình chân của chip trong công cụ layout bo mạch) trên PCB (Printed Circuit Board).

Ngoài hai dòng Performance và Access đầu tiên, hiện nay ST đã đưa ra thị trường thêm hai dòng USB Access và Connectivity như hình bên dưới.



Hình 1.3. Đặc điểm của bốn nhánh trong họ STM32

## 1.2 Tổng quát về ARM Cortex M3 STM32F103

Như đã thấy trong phần giới thiệu, bộ xử lý Cortex là thế hệ lõi nhúng kế tiếp từ ARM. Cortex thừa kế các ưu điểm từ các bộ xử lý ARM trước đó, nó là một lõi xử lý hoàn chỉnh, bao gồm bộ xử lý trung tâm Cortex và một hệ thống các thiết bị ngoại vi xung quanh, Cortex cung cấp phần xử lý trung tâm của một hệ thống nhúng. Để đáp ứng yêu cầu khắc khe và đa dạng của các hệ thống nhúng, bộ xử lý Cortex gồm có 3 nhánh, được biểu hiện bằng các ký tự sau tên Cortex như sau:

**Cortex-A** : bộ vi xử lý dành cho hệ điều hành và các ứng dụng của người dùng phức tạp. Hỗ trợ các tập lệnh ARM, Thumb và Thumb-2.

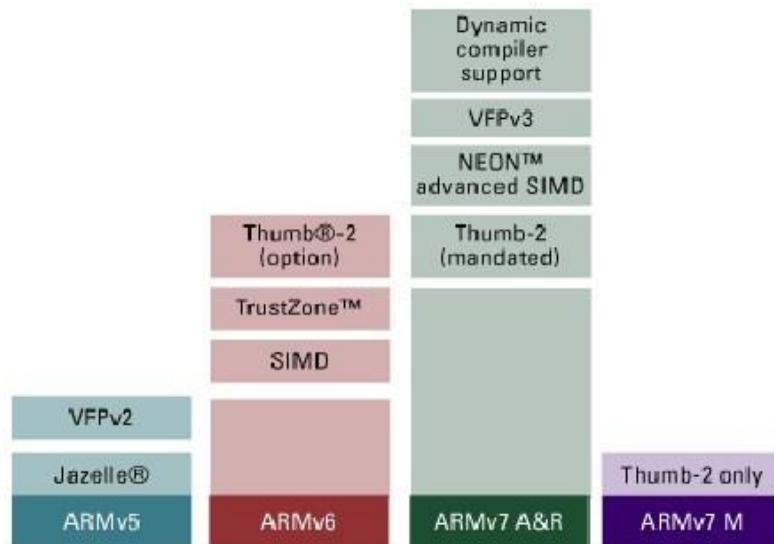
**Cortex-R** : bộ xử lí dành cho các hệ thống đòi hỏi khắc khe về tính thời gian thực. Hỗ trợ các tập lệnh ARM, Thumb, và Thumb-2.

**Cortex-M** : bộ xử lí dành cho dòng vi điều khiển, được tối ưu hóa cho các ứng dụng nhạy cảm về chi phí. Chỉ hỗ trợ tập lệnh Thumb-2.

Con số nằm cuối tên Cortex cho biết mức độ hiệu suất tương đối, với 1 là thấp nhất và 8 là cao nhất. Hiện nay dòng Cortex-M có mức hiệu suất cao nhất là mức 3. STM32 dựa trên bộ xử lý Cortex-M3.

### 1.2.1 Các phiên bản cấu trúc ARM

Tính đến thời điểm hiện tại thì phiên bản kiến trúc mới nhất của lõi ARM là ARMv7 (Trước đó có ARMv4, ARMv5, ARMv6). Bộ xử lý Cortex-M3 dựa trên kiến trúc ARMv7 M và có khả năng thực hiện tập lệnh Thumb-2.



Hình 1.4. Các phiên bản kiến trúc của lõi ARM

Các tài liệu hướng dẫn kỹ thuật cho Cortex-M3 và kiến trúc ARMv7-M có thể được tải về từ website của ARM tại [www.arm.com](http://www.arm.com)

### 1.2.2 Bộ xử lý và đơn vị xử lý trung tâm Cortex

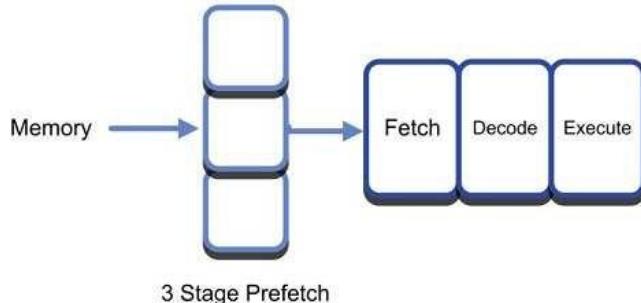
Trong suốt phần còn lại của tài liệu này, các thuật ngữ bộ xử lí Cortex (Cortex processor) và đơn vị xử lí trung tâm Cortex (Cortex CPU) sẽ được sử dụng để phân biệt giữa những lõi Cortex hoàn chỉnh và bộ xử lí trung tâm RISC nội (internal RISC CPU). Trong phần tiếp theo ta sẽ xem xét các đặc điểm chính của đơn vị xử lí trung tâm Cortex, tiếp theo là hệ thống thiết bị ngoại vi bên trong bộ xử lý Cortex.

### 1.2.3 Đơn vị xử lý trung tâm Cortex ( Cortex CPU )

Trung tâm của bộ xử lý Cortex là một CPU RISC 32-bit. CPU này có một phiên bản được đơn giản hóa từ mô hình lập trình (programmer's model) của ARM7/9, nhưng có một tập lệnh phong phú hơn với sự hỗ trợ tốt cho các phép toán số nguyên, khả năng thao tác với bit tốt hơn và khả năng đáp ứng thời gian thực tốt hơn.

#### a. Kiến trúc đường ống

CPU Cortex có thể thực thi hầu hết các lệnh trong một chu kỳ đơn. Giống như CPU của ARM7 và ARM9, việc thực thi này đạt được với một đường ống ba tầng. Tuy nhiên Cortex-M3 khả năng dự đoán việc rẽ nhánh để giảm thiểu số lần làm rỗng (flush) đường ống.

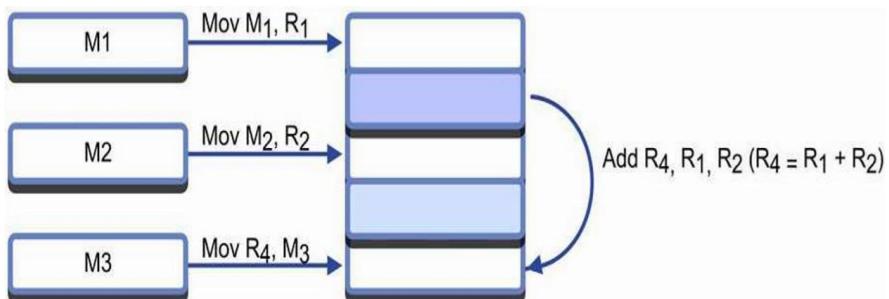


*Hình 1.5. Kiến trúc đường ống của ARM Cortex-M3*

Trong khi một lệnh đang được thực thi, thì lệnh tiếp theo sẽ được giải mã và lệnh tiếp theo nữa sẽ được lấy về từ bộ nhớ. Phương thức hoạt động này sẽ phát huy hiệu quả tối đa cho mã tuyến tính (linear code), nhưng khi gặp phải một rẽ nhánh (ví dụ câu trúc lệnh if...else) thì các đường ống phải được làm rỗng (flush) và làm đầy (refill) trước khi mã có thể tiếp tục thực thi.

### b. Mô hình lập trình

CPU Cortex là bộ xử lý dựa trên kiến trúc RISC, do đó hỗ trợ kiến trúc nạp và lưu trữ (load and store architecture). Để thực hiện lệnh xử lý dữ liệu, các toán hạng phải được nạp vào một tập thanh ghi trung tâm, các phép tính dữ liệu phải được thực hiện trên các thanh ghi này và kết quả sau đó được lưu lại trong bộ nhớ.



*Hình 1.6 Kiến trúc load và store của ARM Cortex-M3*

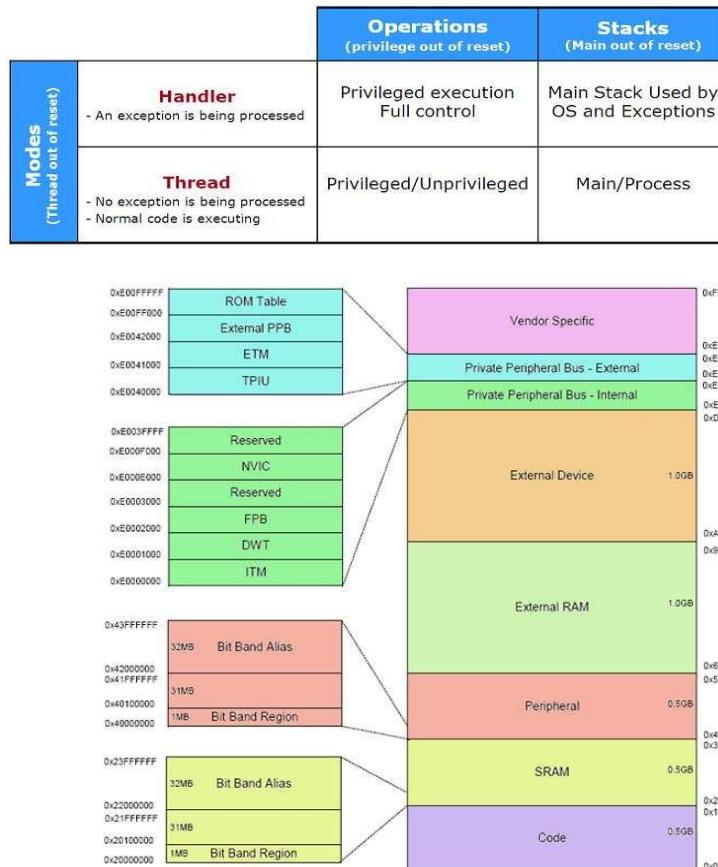
### c. Các chế độ hoạt động

Bộ vi xử lý Cortex được thiết kế với mục tiêu giảm số bóng bán dẫn, nhanh chóng và dễ sử dụng lõi vi điều khiển, nó có được thiết kế để hỗ trợ việc sử dụng hệ điều hành thực hành thời gian. Bộ xử lý Cortex có hai chế độ hoạt động: chế độ Thread và chế độ Handler. CPU sẽ chạy ở chế độ Thread trong khi nó đang thực thi ở chế độ nền không có ngắt xảy ra và sẽ chuyển sang chế độ Handler khi nó đang thực thi các ngắt đặc biệt

(exceptions). Ngoài ra, CPU Cortex có thể thực thi mã trong chế độ đặc quyền hoặc không đặc quyền (privileged or non-privileged mode).

#### d. Bản đồ bộ nhớ

Bộ xử lý Cortex-M3 là một lõi vi điều khiển được tiêu chuẩn hóa, như vậy nó có một bản đồ bộ nhớ cũng được xác định. Mặc dù có nhiều bus nội, bản đồ bộ nhớ này là một không gian địa chỉ 4 Gbyte tuyến tính. Bản đồ bộ nhớ này là chung cho tất cả các thiết bị dựa trên lõi Cortex.



Hình 1.7. Bản đồ bộ nhớ tuyến tính 4Gbyte của bộ xử lý Cortex-M3

#### e. Truy cập bộ nhớ không xếp hàng

Bộ xử lý Cortex-M3 có thể truy cập bộ nhớ không xếp hàng, việc đó đảm bảo rằng SRAM được sử dụng một cách hiệu quả.

CPU Cortex có các chế độ định địa chỉ cho word, half word và byte, nhưng có thể truy cập bộ nhớ không xếp hàng (unaligned memory). Điều này cho phép trình liên kết của trình biên dịch tự do sắp xếp dữ liệu chương trình trong bộ nhớ. Việc bổ sung hỗ trợ tính năng dải bit (bit banding) vào CPU Cortex cho phép các cờ chương trình được đóng gói vào một biến word hoặc half-word hơn là sử dụng một byte cho mỗi cờ.

#### 1.2.4 Bộ xử lý Cortex

Bộ xử lý Cortex được tạo thành từ CPU Cortex kết hợp với nhiều thiết bị ngoại vi như Bus, system timer...

## a. Bus

Bộ vi xử lý Cortex-M3 được thiết kế dựa trên kiến trúc Harvard với bus mã và bus dữ liệu riêng biệt. Chúng được gọi là các bus Icode và Dcode. Cả hai bus đều có thể truy cập mã và dữ liệu trong phạm vi bộ nhớ từ 0x00000000 – 0x1FFFFFFF. Một bus hệ thống bổ sung được sử dụng để truy cập vào không gian điều khiển hệ thống Cortex trong phạm vi 0x20000000 – 0xDFFFFFFF và 0xE0100000 – 0xFFFFFFFF. Hệ thống gỡ lỗi trên chip của Cortex có thêm một cấu trúc bus được gọi là bus ngoại vi riêng.

## b. Ma trận Bus

Bus hệ thống và bus dữ liệu được kết nối với vi điều khiển bên ngoài thông qua một tập các bus tốc độ cao được sắp xếp như một ma trận bus. Nó cho phép một số đường dẫn song song giữa bus Cortex và các bus chủ (bus master) khác bên ngoài như DMA đến các nguồn tài nguyên trên chip như SRAM và các thiết bị ngoại vi. Nếu hai bus chủ (ví dụ CPU Cortex và một kênh DMA) cố gắng truy cập vào cùng một thiết bị ngoại vi, một bộ phân xử nội sẽ giải quyết xung đột và cho truy cập bus vào ngoại vi có mức ưu tiên cao nhất.

## c. Timer hệ thống (System timer)

Lõi Cortex có một bộ đếm xuống 24-bit, với tính năng tự động nạp lại (auto reload) giá trị bộ đếm và tạo sự kiện ngắt khi đếm xuống zero. Nó được tạo ra với dụng ý cung cấp một bộ đếm thời gian chuẩn cho tất cả vi điều khiển dựa trên Cortex. Đồng hồ SysTick được sử dụng để cung cấp một nhịp đập hệ thống cho một RTOS, hoặc để tạo ra một ngắt có tính chu kỳ để phục vụ cho các tác vụ được lập lịch. Thanh ghi trạng thái và điều khiển của SysTick trong đơn vị không gian điều khiển hệ thống Cortex-M3 cho phép chọn các nguồn xung clock cho SysTick.

## d. Xử lí ngắt (Interrupt Handling)

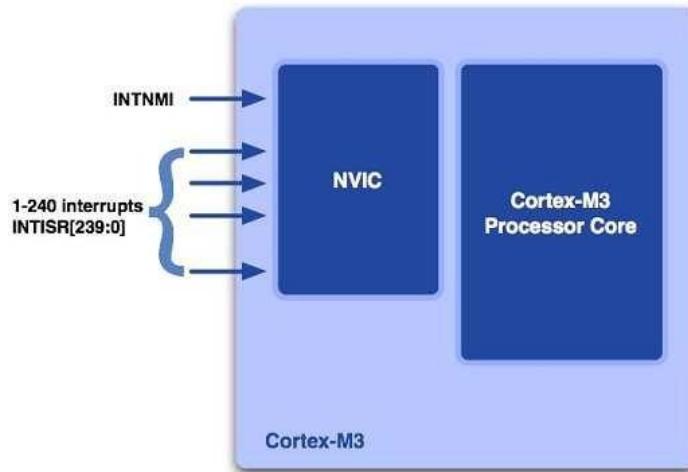
Một trong những cải tiến quan trọng của lõi Cortex so với các CPU ARM trước đó là cấu trúc ngắt của nó và xử lý các ngắt ngoại lệ (exception handling). CPU ARM7 và ARM9 có hai đường ngắt: ngắt nhanh (fast interrupt-FIQ) và ngắt đa dụng (general purpose interrupt hay còn gọi là interrupt request-RIQ). Hai đường tín hiệu ngắt này phục vụ tất cả các nguồn ngắt bên trong một vi điều khiển, trong khi kỹ thuật được sử dụng là như nhau, nhưng việc thực hiện lại khác biệt giữa các nhà sản xuất chip.

## e. Bộ điều khiển vector ngắt lồng nhau (Nested Vector Interrupt

### Controller)

NVIC (Nested Vector Interrupt Controller) là một đơn vị tiêu chuẩn bên trong lõi Cortex. Điều này có nghĩa là tất cả các vi điều khiển dựa trên lõi Cortex sẽ có cùng một cấu trúc ngắt, bất kể nhà sản xuất chip là ST, Atmel, Luminary hoặc NXP... Vì vậy, mă

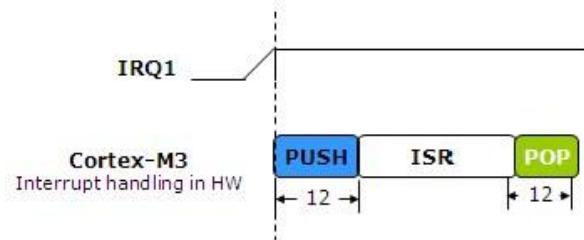
ứng dụng và hệ điều hành có thể dễ dàng được chuyển từ vi điều khiển này sang vi điều khiển khác và lập trình viên khác không cần phải tìm hiểu một tập các thanh ghi hoàn toàn mới. NVIC cũng được thiết kế để có một độ trễ khi đáp ứng ngắt rất thấp.



Hình 1.8. Cấu trúc của NVIC trong bộ xử lý Cortex

- **Phương pháp nhập và thoát khỏi một ngoại lệ của NVIC (NVIC Operation Exception Entry And Exit)**

Khi một ngắt được sinh ra bởi một thiết bị ngoại vi, NVIC sẽ kích khởi CPU Cortex phục vụ ngắt. Khi CPU Cortex đi vào chế độ ngắt của nó, nó sẽ đẩy một tập các thanh ghi vào vùng ngăn xếp (stack). Thao tác này được thực hiện trong vi chương trình (microcode), vì vậy không cần viết thêm bất kỳ lệnh nào trong mã ứng dụng. Trong khi khung ngăn xếp (stack frame) đang được lưu trữ, địa chỉ bắt đầu của trình dịch vụ ngắt đã được lấy về trên bus Icode (instruction bus). Vì vậy, thời gian từ lúc ngắt được sinh ra cho tới khi lệnh đầu tiên của trình dịch vụ ngắt được thực thi chỉ có 12 chu kỳ.



Hình 1.9. Đáp ứng thời gian khi một ngắt bắt đầu xảy ra của Cortex-M3

- **Các chế độ xử lý ngắt cao cấp (Advanced Interrupt Handling Modes)**

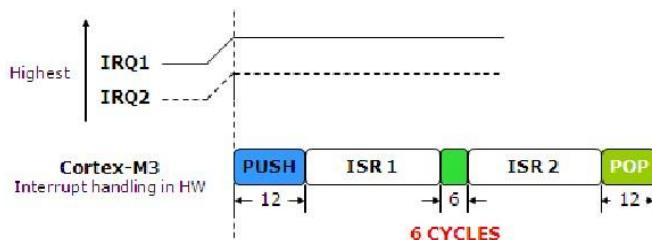
Với khả năng xử lý một ngắt đơn rất nhanh, NVIC được thiết kế để xử lý hiệu quả nhiều ngắt trong một ứng dụng đòi hỏi khắc khe tính thời gian thực. NVIC có một số phương pháp xử lý thông minh nhiều nguồn ngắt, sao cho độ trễ giữa các ngắt là tối thiểu và để đảm bảo rằng các ngắt có mức ưu tiên cao nhất sẽ được phục vụ đầu tiên.

#### **Quyền ưu tiên ngắt (Interrupt Pre-emption)**

NVIC được thiết kế để cho phép các ngắt có mức ưu tiên cao sẽ dành quyền ưu (pre-empt) so với một ngắt có mức ưu tiên thấp hơn đang chạy. Trong trường hợp này ngắt đang chạy sẽ bị dừng và một khung ngăn xếp mới (new stack frame) được lưu lại, thao tác này chỉ mất 12 chu kỳ sau đó ngắt có mức ưu tiên cao hơn sẽ chạy. Khi ngắt có mức ưu tiên cao thực hiện xong, dữ liệu lưu trên ngăn xếp trước đó sẽ được tự động lấy ra (automatically POPed) và ngắt ưu tiên thấp hơn có thể tiếp tục thực hiện.

### Kỹ thuật Tail Chaining trong NVIC

Nếu một ngắt có mức ưu tiên cao đang chạy và đồng thời một ngắt có mức ưu tiên thấp hơn cũng được kích hoạt, NVIC sử dụng một phương pháp gọi là **Tail Chaining** để đảm bảo thời gian trễ là tối thiểu giữa các lần phục vụ ngắt. Nếu hai ngắt được nâng lên, ngắt có mức ưu tiên cao nhất sẽ được phục trước và sẽ bắt đầu thực hiện chỉ sau 12 chu kỳ xung nhịp kể từ lúc xuất hiện ngắt.



Hình 1.10. Đáp ứng thời gian khi hai ngắt xảy ra đồng thời của Cortex-M3

Điều này chỉ mất 6 chu kỳ xung nhịp và sau đó trình phục vụ ngắt kế tiếp có thể bắt đầu được thực thi. Vào cuối các ngắt đang chờ, ngăn xếp được khôi phục và địa chỉ trả về được lấy, tiếp đó chương trình ứng dụng nên có thể bắt đầu thực thi chỉ trong 12 chu kỳ xung nhịp.

#### - Cấu hình và sử dụng NVIC

Để sử dụng NVIC cần phải qua ba bước cấu hình. Đầu tiên cấu hình bảng vector cho các nguồn ngắt mà chúng ta muốn sử dụng. Tiếp theo cấu hình các thanh ghi NVIC để cho phép và thiết lập các mức ưu tiên của các ngắt trong NVIC và cuối cùng cần phải cấu hình các thiết bị ngoại vi và cho phép ngắt tương ứng.

Trong trường hợp của bộ đếm thời gian SysTick, chúng ta có thể tạo ra một trình phục vụ ngắt bằng cách khai báo một hàm C với tên phù hợp:

```
void SysTick_Handler (void)
{
    ....
}
```

Các ngắt đặc biệt bên trong Cortex được cấu hình thông qua các thanh ghi điều khiển và thanh ghi cấu hình mức ưu tiên của hệ thống, trong khi đó các thiết bị ngoại vi người dùng được cấu hình bằng cách sử dụng các thanh ghi IRQ (Interrupt Request). Ngắt của SysTick là một ngắt đặc biệt bên trong Cortex và được xử lý thông qua các thanh ghi hệ thống. Một số ngắt đặc biệt khác bên trong lõi Cortex luôn ở trạng thái cho phép, bao

gồm các ngắt reset và NMI (Non-Maskable Interrupt), tuy nhiên ngắt của timer hệ thống-SysTick lại không được kích hoạt bên trong NVIC.

```
SysTickCurrent = 0x9000; //Start value for the sys Tick counter  
SysTickReload = 0x9000; //Reload value  
SysTickControl = 0x07; //Start and enable interrupt
```

Mỗi thiết bị ngoại vi được điều khiển bởi các khối thanh ghi IRQ. Mỗi ngoại vi có một bit cho phép ngắt. Những bit nằm trên hai thanh ghi cho phép ngắt có chiều dài là 32-bit. Việc cấu hình ngắt cho một thiết bị ngoại vi cũng giống với cấu hình một exception bên trong Cortex. Trong trường hợp ngắt của ADC, trước tiên chúng ta phải thiết lập vector ngắt và cung cấp hàm phục vụ ngắt-ISR:

```
DDC ADC_IRQHandler ;  
void ADC_Handler(void)  
{ }
```

Sau đó, ADC phải được khởi tạo và các ngắt phải được cho phép trong các thiết bị ngoại vi và các NVIC:

```
ADC1→CR2 = ADC_CR2; //Switch on the ADC and continuous conversion  
ADC1→SQR1 = sequence1; //Select number of channels in sequence conversion  
ADC1→SQR2 = sequence2; //and select channels to convert  
ADC1→SQR3 = sequence3;  
ADC1→CR2 |= ADC_CR2; //Rewrite on bit  
ADC1→CR1 = ADC_CR1; //Start regular channel group, enable ADC interrupt  
GPIOB→CRH = 0x33333333; //Set LED pins to output  
NVIC→Enable[0] = 0x00040000; //Enable ADC interrupt NVIC→Enable[1] = 0x00000000;
```

## 1.2.5 Các chế độ năng lượng

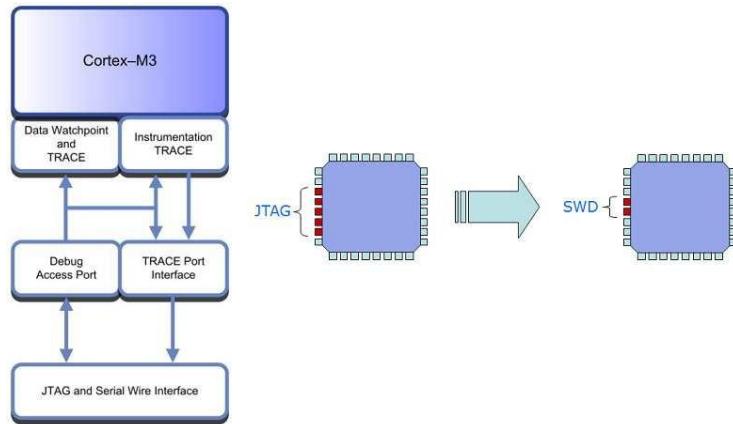
Trong phần này, chúng ta sẽ xem xét các chế độ quản lý năng lượng bên trong lõi Cortex. Các tùy chọn đầy đủ về quản lý năng lượng của STM32 sẽ được xem xét ở phần sau. CPU Cortex có một chế độ ngủ (sleep mode), sẽ đặt lõi Cortex vào chế độ năng lượng thấp của nó và ngừng thực thi các lệnh bên trong của CPU Cortex. Một phần nhỏ của NVIC vẫn được hoạt động bình thường, do đó ngắt tạo ra từ các thiết bị ngoại vi của STM32 có thể đánh thức lõi Cortex.

### a. Cách đi vào chế độ năng lượng thấp của CPU Cortex

Lõi Cortex có thể được đặt vào chế độ sleep bằng cách thực hiện lệnh WFI (Wait For Interrupt) hoặc WFE (Wait For Sự kiện). Trong trường hợp thực thi lệnh WFI, lõi Cortex sẽ tiếp tục thực hiện và phục vụ ngắt đang chờ xử lý. Khi trình phục vụ ngắt-ISR kết thúc, sẽ có hai khả năng xảy ra. Trước tiên, CPU Cortex có thể trở về từ ISR này và tiếp tục thực hiện chương trình ứng dụng nền như bình thường. Bằng cách đặt bit **SLEEPON EXIT** trong thanh ghi điều khiển hệ thống, lõi Cortex sẽ tự động đi vào chế độ ngủ một khi ISR này kết thúc. Ngắt WFE cho phép lõi Cortex tiếp tục thực hiện chương trình từ điểm mà nó được đặt vào chế độ sleep. Nó sẽ không nhảy đến và thực thi một trình phục vụ nào. Một sự kiện đánh thức (wake-up) chỉ đơn giản đến từ một thiết bị ngoại vi dù cho nó không được kích hoạt như là một ngắt bên trong NVIC.

### b. Khối hỗ trợ gỡ lỗi CoreSight

Tất cả các CPU ARM đều trang bị hệ thống gỡ lỗi riêng của nó ngay trên chip. CPU ARM7 và ARM9 CPU có tối thiểu một cổng JTAG cho phép một công cụ gỡ lỗi chuẩn kết nối với CPU và tải chương trình vào bộ nhớ RAM nội hoặc bộ nhớ Flash. Cổng JTAG cũng hỗ trợ điều khiển động cơ的基本 (thiết lập chạy từng bước và các breakpoint v.v...) cũng như có thể xem nội dung của các vị trí trong bộ nhớ.



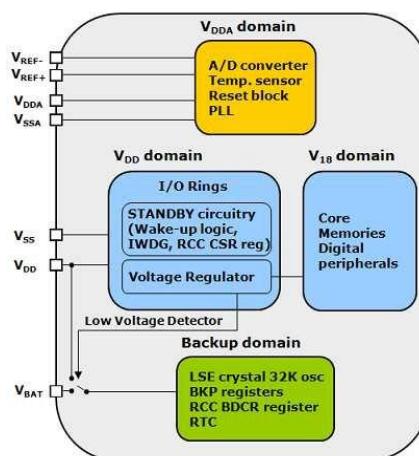
Hình 1.11. Hệ thống gỡ lỗi CoreSight bên trong Cortex

Hệ thống gỡ lỗi Cortex CoreSight sử dụng giao diện JTAG hoặc SWD (Serial Wire Debug). CoreSight cung cấp chức năng chạy kiểm soát và theo dõi. Nó có thể chạy khi STM32 đang ở một chế độ năng lượng thấp. Đây là một bước cải tiến lớn về chuẩn gỡ lỗi JTAG. Phần cứng cơ bản cho một thiết kế

#### 1.2.6 Kiểu đóng gói chip và kiểu chân linh kiện

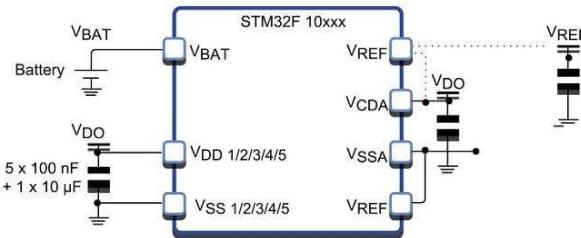
Các biến thể của dòng Access, USB, Performance và Connectivity của STM32 được thiết kế để phù hợp với nhiều kiểu đóng gói, để cho phép nâng cấp phần cứng một cách dễ dàng mà không cần phải thiết kế lại PCB (Printed Circuit Board). Tất cả các vi điều khiển STM32 đều có sẵn dạng đóng gói LQFP, từ 48 chân đến 144 chân.

#### 1.2.7 Nguồn cung cấp điện



Hình 1.12. Các miền năng lượng bên trong STM32

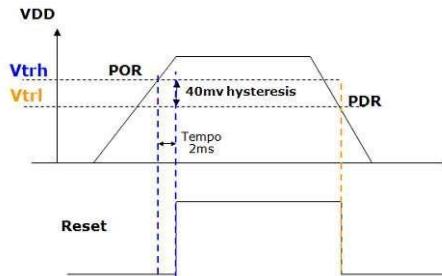
Tùy chọn cung cấp năng lượng thứ hai được sử dụng để cung cấp cho ADC. Nếu ADC được sử dụng, nguồn điện chính VDD được giới hạn trong phạm vi 2.4V đến 3.6V. Đối với chip đóng gói 100 chân, khối ADC có thêm chân điện áp tham khảo VREF+ và VREF-. Chân VREF- phải được kết nối với VDDA và VREF+ có thể thay đổi từ 2,4V đến VDDA. Tất cả các kiểu đóng gói chip còn lại thì điện áp tham khảo được kết nối bên trong với các chân cung cấp điện áp ADC. Mỗi nguồn cung cấp năng lượng cần một tụ chống nhiễu đi kèm.



Hình 1.13. Cách bố trí tụ chống nhiễu cho STM32

### 1.2.8 Mạch Reset

STM32 chứa một mạch reset nội, mạch này giữ cho chip ở trạng thái reset cho tới khi nào VDD vẫn còn dưới mức 2.0V.



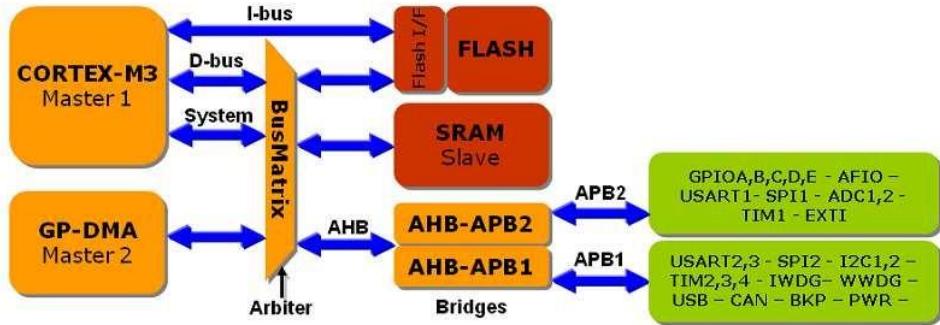
Hình 1.14. Đặc tính của mạch reset bên trong STM32

Bộ POR (Power On Reset) và PDR (Power Down Reset) đảm bảo xử lý chỉ chạy với một nguồn cấp điện ổn định, và không cần bất kì một mạch reset bên ngoài.

Một mạch reset bên ngoài không cần thiết trong thiết kế của STM32. Tuy nhiên, trong quá trình phát triển chân nRST có thể được kết nối với một nút reset đơn giản, đồng thời chân nRST cũng được kết nối đến cổng JTAG, để công cụ phát triển có thể tạo ra tín hiệu reset vi điều khiển.

### 1.3 Kiến trúc hệ thống

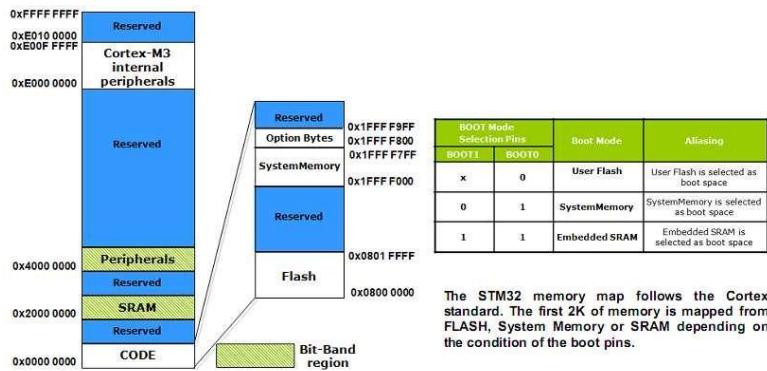
STM32 gồm nhân Cortex kết nối với bộ nhớ FLASH thông qua đường bus lệnh chuyên biệt. Các bus dữ liệu(Cortex Data busses) và hệ thống(Cortex System busses) được kết nối tới ma trận busses tốc độ cao( ARM Advanced High Speed Busses- AHB).



Hình 1.15. Cấu trúc Bus

Cấu trúc bus nội cung cấp đường truyền chuyên biệt dành cho tập lệnh thực thi và ma trận bus đường dữ liệu cho nhân Cortex and bộ điều khiển DMA truy cập tài nguyên trên vi xử lý.

### 1.3.1 Cấu trúc bộ nhớ

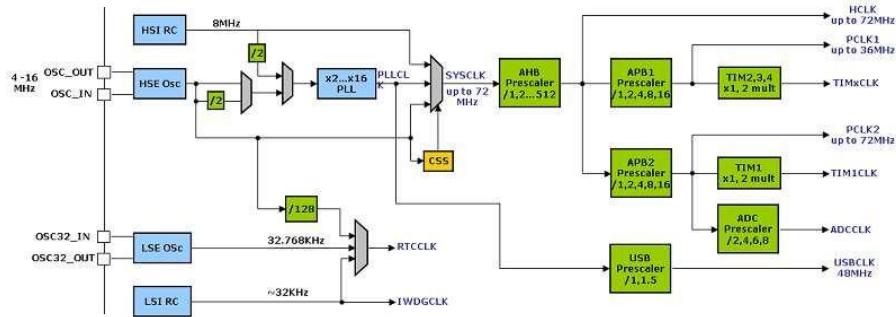


Hình 1.16. Vùng nhớ Flash trên STM32

Vùng nhớ dành cho flash được chia nhỏ thành 3 vùng. Vùng thứ nhất gọi là User Flash bắt đầu từ địa chỉ 0x00000000. Kế tiếp là System Memory hay còn gọi là vùng nhớ lớn. Vùng này có độ lớn 4Kbytes thông thường sẽ được nhà sản xuất cài đặt bootloader. Cuối cùng là vùng nhớ nhỏ bắt đầu từ địa chỉ 0x1FFFF80 chứa thông tin cấu hình dành cho STM32. Bootloader thường được dùng để tải chương trình thông qua USART1 và chứa ở vùng User Flash.

### 1.3.2 Tối đa hiệu năng

Ngoài việc hỗ trợ 2 bộ tạo xung nhịp ngoại STM32 cung cấp thêm 2 bộ tạo xung nhịp nội. Sau khi reset đồng hồ tạo xung của nhân Cortex, bộ tạo xung nhịp tốc độ cao( High Speed Internal Oscillator) hoạt động ở mức thấp 8MHz. Bộ tạo xung nội còn lại là Low Speed Internal Oscillator hoạt động ở mức 32768KHz.



Hình 1.17 STM32 bao gồm 2 bộ tạo xung nhịp nội và 2 bộ tạo xung nhịp ngoại thêm vào đó là bộ vòng khóa pha( Phase Lock Loop-PLL).

Nhân Cortex có thể được cấp xung nhịp từ bộ tạo dao động nội và ngoại, đồng thời từ PLL nội. Có một vấn đề là đối với bộ tạo dao động nội tốc độ cao xung nhịp không hoạt động chính xác ở 8MHz do đó khi sử dụng các thiết bị ngoại vi như: giao tiếp serial hay sử dụng định thời thời gian thực thì nên dùng bộ tạo dao động ngoại tốc độ cao. Tuy vậy, cho dù sử dụng bộ tạo dao động nào đi nữa thì nhân Cortex luôn phải sử dụng xung nhịp tạo ra từ bộ PLL. Tất cả thanh ghi điều khiển PLL và cấu hình bus đều được bố trí ở nhóm RCC ( Reset and Clock Control).

### a. Vòng khóa pha ( Phase Lock Loop )

Sau khi hệ thống reset STM32 nhận xung nhịp từ bộ tạo dao động HIS. Tại thời điểm đó các bộ tạo dao động ngoại sẽ bị tắt. Bước đầu tiên để STM32 hoạt động ở mức xung nhịp cao nhất là bật bộ tạo dao động HSE và chờ cho đến khi đi vào hoạt động ổn định. Bộ tạo dao động ngoại có thể được kích hoạt thông qua các thanh ghi điều khiển RCC\_Control. Sẽ có 1 bit trạng thái được bật khi chúng đi vào hoạt động ổn định. Một khi bộ tạo dao động ngoại hoạt động ổn định, nó có thể được chọn là đầu vào cho bộ PLL. Xung nhịp ra được tạo bởi PLL được xác định bằng cách thiết lập các bội số nguyên trong thanh ghi cấu hình RCC\_PLL. Trong trường hợp xung nhịp đầu vào của PLL là 8MHz khi đó cần cấu hình bội số nhân cho PLL là 9 để tạo xung nhịp 72MHz ở đầu ra.

#### Đoạn mã cấu hình STM32 sử dụng dao động từ PLL

```
//HSE clock, PLLx9
RCC->CFGR = 0x001D0000; //Enable PLL
RCC->CR |= 0x01000000;
While( !(RCC->CR & 0x02000000));
//Set the remaining control fields
RCC->CR |= 0x00000001;
//Set the remaining configuration fields RCC->CFGR |= 0x005D0402;
```

#### - Cấu hình cho bus

Khi PLL đã được chọn là bộ tạo dao động cho hệ thống, Cortex CPU sẽ hoạt động ở mức 72MHz. Để cho toàn bộ các phần còn lại của hệ thống hoạt động ở mức tối ưu người dùng cần phải cấu hình AHB và APB thông qua các thanh ghi cầu nối.

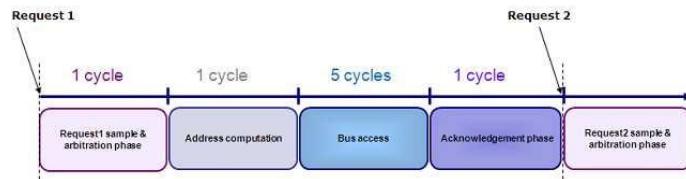
```
//Enable clocks to the AHB,APB1 and APB2 busses
AHBENR = 0x00000014;
RCC->APB2ENR = 0x00005E7D;
RCC->APB1ENR = 0x1AE64807;
//Release peripheral reset line on APB1 and APB2 buses
RCC->APB2RSTR = 0x00000000;
RCC->APB1RSTR = 0x00000000;
```

### - Flash Buffer

Khi xem xét kiến trúc hệ thống của STM32 chúng ta có thể thấy nhân Cortex kết nối với Flash thông qua đường dữ liệu chuyên biệt I-Bus. Bus dữ liệu này hoạt động cùng tần số với CPU, do vậy nếu CPU lấy dao động từ PLL thì bus dữ liệu sẽ hoạt động ở mức xung nhịp cao nhất 72Mhz. Cortex CPU sẽ truy cập vào Flash cứ mỗi 1.3ns. Khi mới hoạt động, nhân STM32 sử dụng bộ tạo dao động nội, do đó thời gian truy cập Flash là không đáng kể.

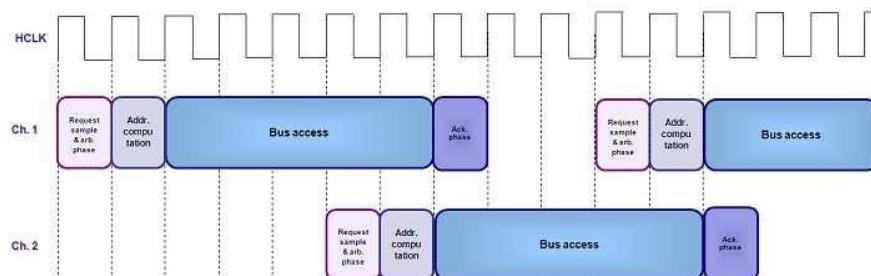
### - Direct Memory Access

Mặc dù có thể sử dụng chính nhân Cortex để trao đổi dữ liệu giữa các thiết bị ngoại vi và SRAM nội, tuy nhiên chúng ta có thể hoàn toàn sử dụng cơ chế tự động cho việc này với bộ quản lý DMA. STM32 có 7 kênh DMA độc lập dùng để chuyển dữ liệu từ: bộ nhớ sang bộ nhớ, ngoại vi tới bộ nhớ, bộ nhớ tới ngoại vi và ngoại vi tới ngoại vi.



Hình 1.18 Mỗi thao tác bộ nhớ DMA bao gồm 4 giai đoạn.

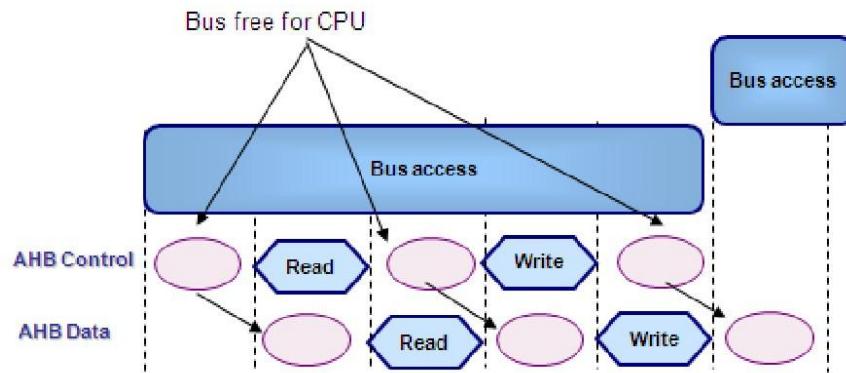
Quá trình truyền dữ liệu gồm 4 giai đoạn: lấy mẫu và phân xử, tính toán địa chỉ, truy cập đường truyền, và cuối cùng là hoàn tất.



Hình 1.19 Bộ DMA được thiết kế cho truyền dữ liệu tốc độ và kích thước nhỏ.

Bộ DMA chỉ sử dụng bus dữ liệu khi ở giai đoạn truy cập đường truyền.

Bộ DMA có thể thực hiện việc phân xử tài nguyên và tính toán địa chỉ trong khi bộ DMA khác đang ở giai đoạn truy cập đường truyền như mô tả ở hình trên. Ngay khi bộ DMA thứ nhất kết thúc việc truy cập đường truyền, bộ DMA 2 có thể ngay lập tức sử dụng đường truyền dữ liệu.

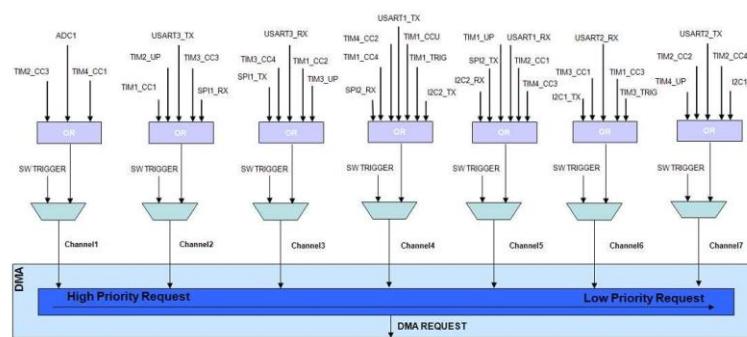


Hình 1.20 Ở giai đoạn Bus Access CPU sẽ có 3 chu kỳ rảnh.

Trong trường hợp trao đổi dữ liệu từ vùng nhớ sang vùng nhớ mỗi kênh DMA chỉ sử dụng đường truyền dữ liệu ở giai đoạn Bus Access và 5 chu kỳ CPU để chuyển 2 bytes dữ liệu. Trong đó 1 chu kỳ để đọc và 1 chu kỳ để ghi, 3 chu kỳ còn lại được bố trí xen kẽ nhằm giải phóng đường truyền dữ liệu cho nhân Cortex. Điều đó có nghĩa là bộ DMA chỉ sử dụng tối đa 40% băng thông của đường truyền dữ liệu. Việc sử dụng DMA rất đơn giản. Đầu tiên là kích hoạt đồng hồ xung nhịp

**RCC->AHBENR |= 0x00000001; //enable DMA clock**

Mỗi kênh DMA có thể được gắn với một mức ưu tiên: rất cao, cao, trung bình và thấp. Có ba loại ngắt hỗ trợ cho DMA: hoàn thành chuyển dữ liệu, hoàn thành một nửa, và lỗi. Sau khi cấu hình hoàn tất, chúng ta kích hoạt Channel Enable Bit để thực hiện quá trình chuyển dữ liệu



Hình 1.21 Mỗi kênh DMA được gắn với ngoại vi nhất định. Khi được kích hoạt, các thiết bị ngoại vi sẽ điều khiển bộ DMA tương ứng.

Kiểu truyền dữ liệu từ bộ nhớ sang bộ nhớ thường hay được dùng để khởi tạo vùng nhớ, hay chép các vùng dữ liệu lớn. Phần lớn tác vụ DMA hay được sử dụng để chuyển dữ liệu giữa ngoại vi và vùng nhớ.

## 1.4 Các ngoại vi

Phần này sẽ giới thiệu các thiết bị ngoại vi trên các phiên bản STM32. Để tiện theo dõi, chúng tôi chia ra thành 2 loại: ngoại vi đa dụng và ngoại vi giao tiếp. Tất cả ngoại vi trên STM32 được thiết kế và dựa trên bộ DMA. Mỗi ngoại vi đều có phần điều khiển mở rộng nhằm tiết kiệm thời gian xử lý của CPU

### 1.4.1 Ngoại vi đa dụng

Ngoại vi đa dụng trên STM32 bao gồm: các cổng I/O đa dụng, bộ điều khiển ngắt ngoại, bộ chuyển đổi ADC, bộ điều khiển thời gian đa dụng và mở rộng, đồng hồ thời gian thực, và chân “tamper”.

#### a. Các cổng I/O đa dụng

STM32 có 5 cổng I/O đa dụng với 80 chân điều khiển.

Các cổng I/O được đánh số từ A->E và mức áp tiêu thụ ở 5V. Nhiều chân ngoại có thể được cấu hình như là Input/Output tương tác với các thiết bị ngoại vi riêng của người dùng như USART hay I2C. Thêm nữa có thể cấu hình các chân này như là nguồn ngắt ngoại kết hợp với cổng GPIO khác.

Mỗi cổng GPIO đều có 2 thanh ghi 32-bit điều khiển. Như vậy ta có 64-bit để cấu hình 16 chân của một cổng GPIO. Như vậy mỗi chân của cổng GPIO sẽ có 4 bit để điều khiển: 2 bit sẽ quy định hướng ra vào dữ liệu: input hay output, 2 bit còn lại sẽ quy định đặc tính dữ liệu. Sau khi cổng được cấu hình, ta có thể bảo vệ các thông số cấu hình bằng cách kích hoạt thanh ghi bảo vệ. Trong thanh ghi này, mỗi chân trong cổng đều có một bit bảo vệ tương ứng để tránh các thay đổi vô ý ở các 4 bit cấu hình. Để kích hoạt chế độ bảo vệ, ta ghi lần lượt giá trị 1,0,1 vào bit 16:

```
uint32_t tmp = 0x00010000; //bit 16
tmp |= GPIO_Pin;           //chân cần được bảo vệ
/* Set LCKK bit */
GPIOx->LCKR = tmp; //ghi giá trị 1 vào bit 16 và chân cần bảo vệ
/* Reset LCKK bit */
GPIOx->LCKR = GPIO_Pin; //ghi giá trị 0 vào bit 16
/* Set LCKK bit */
GPIOx->LCKR = tmp; //ghi giá trị 1 vào bit 16
```

Sau đó đọc lại bit 16 liên tục 2 lần, nếu giá trị trả về lần lượt là 0 và 1 thì thiết lập khóa đã hoàn thành

```
tmp = GPIOx->LCKR; tmp = GPIOx->LCKR;
```

Để dễ dàng đọc và ghi dữ liệu trên cổng GPIO, STM32 cung cấp 2 thanh ghi Input và Output data. Kỹ thuật bit banding được hỗ trợ nhằm thực hiện các thao tác bit trên thanh ghi dữ liệu. Thanh ghi 32-bit Set/Reset, với 16 bit cao ánh xạ tới mỗi chân của cổng điều khiển reset khi được thiết lập giá trị 1. Tương tự vậy 16 bit thấp điều khiển Set khi được gán giá trị 1.

## - Các chức năng thay thế

Chức năng thay thế cho phép người dùng sử dụng các cổng GPIO với các ngoại vi khác. Để thuận tiện cho thiết kế phần cứng, một thiết bị ngoại vi có thể được ánh xạ tới một hay nhiều chân của vi xử lý STM32. Sử dụng các tính năng thay thế của STM32 được điều khiển bởi các thanh ghi “Remap & Debug I/O”. Mỗi thiết bị ngoại vi(USART, CAN, Timers, I2C và SPI) có 1 hoặc 2 trường bit điều khiển ánh xạ tới các chân của vi điều khiển. Một khi các chân được cấu hình sử dụng chức năng thay thế, các thanh ghi điều khiển GPIO sẽ được sử dụng để điều khiển các chức năng thay thế thay vì tác vụ I/O. Các thanh ghi Remap còn điều khiển bộ JTAG. Khi hệ thống khởi động, cổng JTAG được kích hoạt tuy nhiên chức năng theo dõi dữ liệu(data trace) vẫn chưa khởi động. JTAG khi đó có thể chuyển sang chế độ debug, xuất dữ liệu theo dõi ra ngoài, hoặc đơn giản chỉ sử dụng như cổng GPIO.

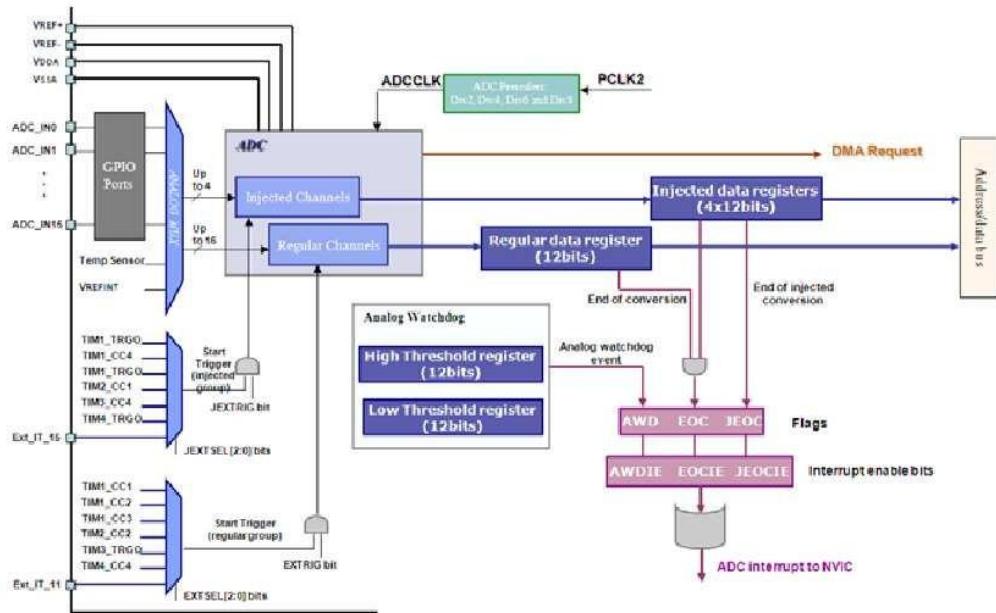
## b. Ngắt ngoại ( EXTI )

NVIC cung cấp bảng vector ngắt riêng biệt dành cho các ngắt từ 0-4, ngắt RTC, ngắt Power detect và ngắt USB wake up. Các ngắt ngoại còn lại chia làm 2 nhóm 5-10, và 11-15 được cung cấp thêm 2 bảng ngắt bổ sung. Các ngắt ngoại rất quan trọng trong quản lý tiêu thụ năng lượng của STM32. Chúng có thể được sử dụng để “đánh thức” nhân vi xử lý từ chế độ STOP khi cả 2 nguồn tạo xung nhịp chính ngưng hoạt động. 16 ngắt ngoại có thể được ánh xạ tới bất kỳ chân nào của vi xử lý thông qua 4 thanh ghi cấu hình điều khiển. Mỗi ngắt được điều khiển bởi trường 4 bit, đoạn mã sau mô tả cách cấu hình ngắt cho chân GPIO

```
//Map the external interrupts to port pins
AFIO->EXTICR[0] = 0x0000000;
//Enable external interrupt sources
EXTI->IMR = 0x00000001;
//Enable wake up event
EXTI->EMR = 0x00000000;
//Select falling edge trigger sources
EXTI->FTSR = 0x00000001;
//Select rising edge trigger sources
EXTI->RTSR = 00000000;
//Enable interrupt sources in NVIC
NVIC->Enable[0] = 0x00000040;
NVIC->Enable[1] = 0x00000000;
```

### c. ADC

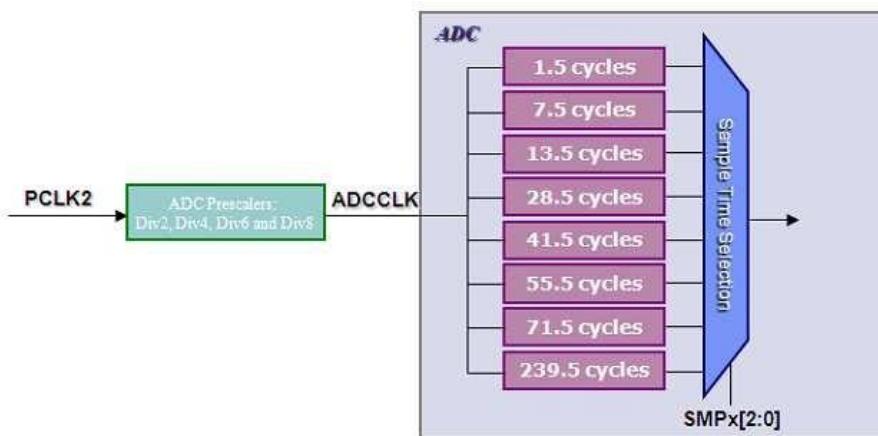
STM32 có thể có 2 bộ chuyển đổi tín hiệu tương tự sang tín hiệu số tùy vào các phiên bản. Bộ ADC có thể được cung cấp nguồn riêng từ 2.4V đến 3.6V. Nguồn cung cấp cho bộ ADC có thể được kết nối trực tiếp hoặc thông qua các chân chuyên biệt. Bộ ADC có độ phân giải 12-bit và tần suất lấy mẫu là 12Mhz. Với 18 bộ ghép kênh, trong đó 16 kênh dành cho các tín hiệu ngoại, 2 kênh còn lại dành cho cảm biến nhiệt và vôn kế nội.



Hình 1.22 bộ ADC STM32

- Thời gian chuyển đổi và nhóm chuyển đổi

Bộ ADC cho phép người dùng có thể cấu hình thời gian chuyển đổi riêng biệt cho từng kênh. Có 8 mức thời gian chuyển đổi riêng biệt từ 1.5 đến 239.5 chu kỳ.

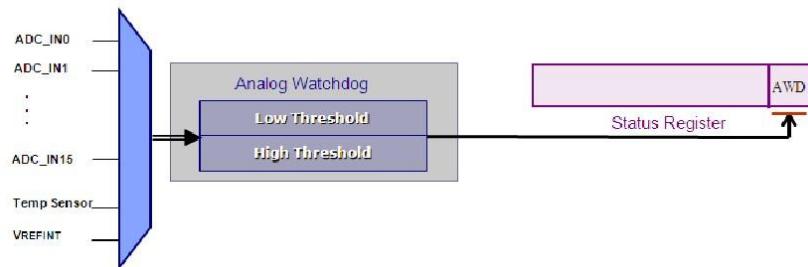


Hình 1.23 Các mức thời gian chuyển đổi ADC

Mỗi bộ ADC có 2 chế độ chuyển đổi: thông thường(regular) và injected. Ở chế độ regular cho phép một hay một nhóm các kênh kết hợp với nhau thực thi tác vụ chuyển đổi. Một nhóm kênh tối đa có thể gồm 16 kênh. Thứ tự chuyển đổi trong nhóm có thể được cấu hình bởi phần mềm, và trong một chu kỳ chuyển đổi của nhóm, một kênh có thể được sử dụng nhiều lần. Chuyển đổi regular có thể được kích hoạt bằng sự kiện phần cứng của Timer hay ngắt ngoại EXTI 1.

### - **Analogue WatchDog**

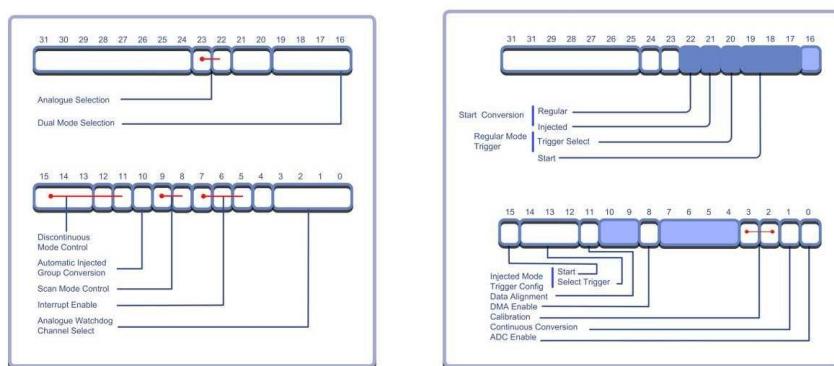
Ngoài 2 chế độ Regular và Injected, khối ADC còn được bổ sung thêm Analogue WatchDog. Khối này hỗ trợ phát hiện dữ liệu tương tự nằm ngoài vùng hoạt động bình thường của một kênh ADC cho trước. Khi được cấu hình ngưỡng trên và ngưỡng dưới, nếu tín hiệu tương tự đầu vào nằm ngoài vùng trên, thì ngắt sẽ được phát sinh.



*Hình 1.24 Analogue Watchdog có thể dùng giám sát một hay nhiều kênh ADC với vùng ngưỡng được cấu hình bởi người dùng*

### - **Cấu hình ADC**

Có hai thanh ghi điều khiển ADC\_CR1 và ADC\_CR2 để cấu hình hoạt động của khối ADC.



*Hình 1.25 Hai thanh ghi điều khiển cấu hình hoạt động của khối ADC*

```

ADC1->CR2 = 0x005E7003; //Switch on ADC1 and enable continuous conversion
ADC1->SQR1 = 0x0000; //set sequence length to one
ADC1->SQR2 = 0x0000; //select conversion on channel zero
ADC1->SQR3 = 0x0001;
ADC1->CR2 = 0x005E7003; //rewrite on bit
NVIC->Enable[0] = 0x00040000; //enable ADC interrupt NVIC->Enable[1] = 0x00000000;

```

# CHƯƠNG I : TÌM HIỂU VỀ ARM CORTEX M3 STM32F103

Ở hàm xử lý ngắn ADC

```
Void ADC_IRQHandler(void)
{
    GPIOB->ODR =ADC1->DR << 5; //Copy ADC result to port B }
```

Hoặc chúng ta có thể sử dụng DMA thay vì ngắn :

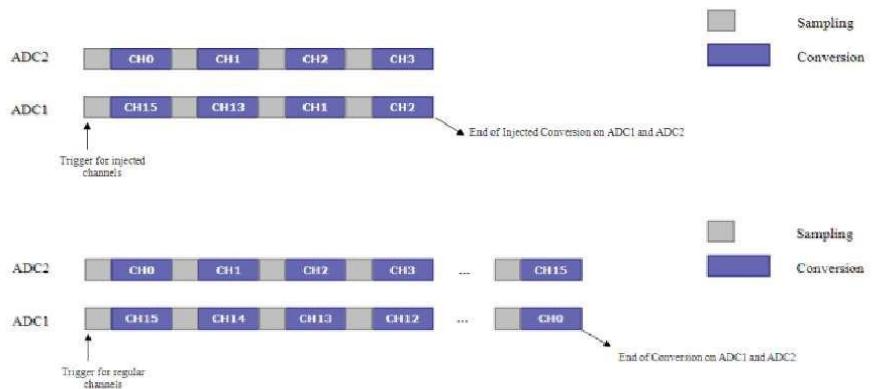
```
DMA_Channel1->CCR = 0x00003A28; //Circular mode, peripheral and memory increased disable
//Load destination address into peripheral register, GPIO port data register
DMA_Channel1->CMAR = (unsigned int) 0x04001244C;
DMA_Channel1->CNDTR = 0x0001; //number of words to transfer DMA_Channel1->CCR =
0x00000001;/Enable the DMA transfer
```

Chúng ta kích hoạt chế độ DMA của khối ADC :

```
ADC1->CR2 |= 0x0100;
```

- **Dual mode:** Ở một số phiên bản, ST cung cấp 2 khối ADC nhằm đáp ứng các tác vụ phức tạp hơn

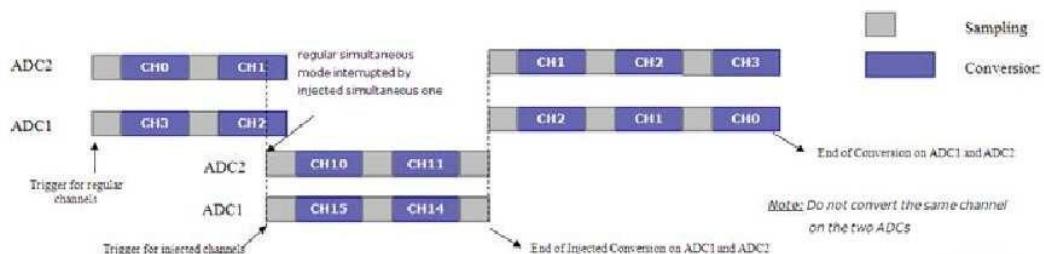
**Cả hai khối ADC cùng hoạt động ở cùng chế độ Regular hoặc Injected**



Hình 1.26 Cả hai khối ADC cùng hoạt động ở cùng chế độ Regular hoặc Injected

Khi hoạt động ở chế độ này, cùng lúc khối ADC1 và ADC2 sẽ chuyển đổi dữ liệu từ 2 kênh khác nhau. Ví dụ trong các ứng dụng cần theo dõi cùng lúc điện áp và cường độ dòng.

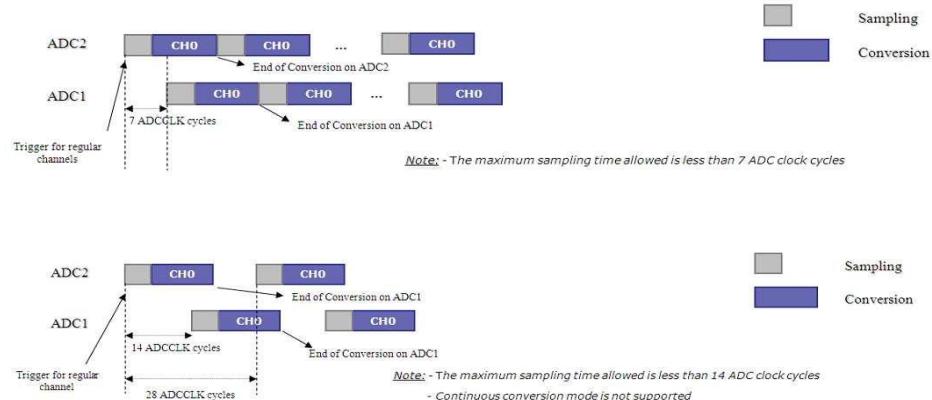
**Cả hai khối cùng hoạt động ở 2 chế độ Regular và Injected xen kẽ**



Hình 1.27 Cả hai khối cùng hoạt động ở 2 chế độ Regular và Injected xen kẽ

Như hình trên mô tả, cả hai khối ADC hoạt động ở cùng một chế độ tại cùng thời điểm. Khi chế độ Injected được kích hoạt, cả khối ADC1 và ADC2 tạm thời rời trạng thái Regular để thực thi chuyển đổi các kênh trong chế độ Injected.

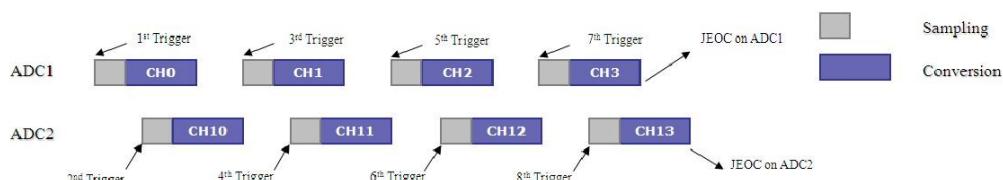
### Hoạt động xen kẽ nhanh và chậm Regular



Hình 1.28 Hoạt động xen kẽ nhanh và chậm Regular

Ở chế độ xen kẽ nhanh, một kênh có thể liên tục chuyển đổi bởi hai khối ADC, thời gian nhỏ nhất để kích hoạt lần chuyển đổi kế tiếp là 7 chu kỳ xung nhịp của ADC. Ở chế độ xen kẽ chậm khoảng cách thời gian tối thiểu là 14 chu kỳ xung nhịp. Hai chế độ kết hợp này làm tăng hiệu suất chuyển đổi của khối ADC.

### Chế độ kích hoạt thay thế



Hình 1.29 Chế độ kích hoạt thay thế

Ban đầu phần cứng sẽ kích hoạt kênh đầu tiên trong nhóm chuyển đổi Injected của khối ADC1, sau đó sẽ kích hoạt tiếp nhóm Injected của ADC2.

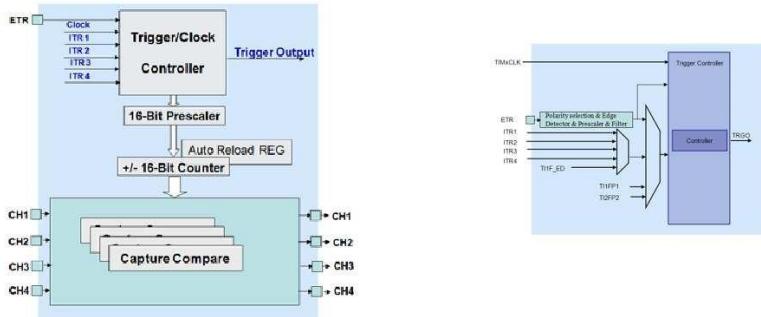
#### d. Bộ định thời đa nhiệm và nâng cao

STM32 có bốn khối định thời. Timer1 là khối nâng cao dành cho điều khiển động cơ. 3 khối còn lại đảm nhiệm chức năng đa nhiệm. Tất cả chúng đều có chung kiến trúc, khối nâng cao sẽ có thêm các đặc tính phần cứng riêng biệt.

##### - Bộ định thời đa nhiệm

Tất cả các khối định thời đều gồm bộ đếm 16-bit với thanh ghi chia tần số dao động 16-bit(prescaler) và thanh ghi tự nạp(auto-reload). Bộ đếm của khối định thời có thể được cấu hình để đếm lên, đếm xuống hay trung tính(lên xuống xen kẽ nhau). Xung nhịp cho đồng hồ có thể được lựa chọn dựa trên 8 nguồn khác nhau: từ đồng hồ chuyên

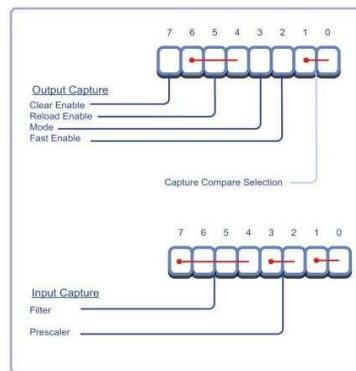
biệt được lấy từ đồng hồ hệ thống, từ xung nhịp chân ra lấy từ khối định thời khác, hoặc từ nguồn xung nhịp ngoại.



Hình 1.30 4 khối định thời với các thanh ghi 16-bit Prescaler

### Khối Capture/Compare

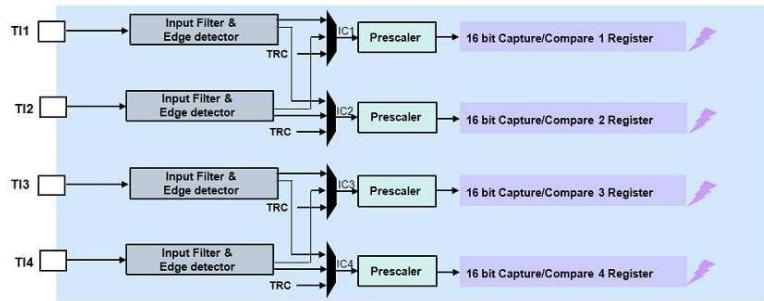
Mỗi kênh Capture/Compare được điều khiển bởi duy nhất một thanh ghi. Chức năng của thanh ghi này có thể thay đổi tùy thuộc cấu hình. Ở chế độ Capture, thanh ghi này có nhóm các bit đảm nhận thiết lập lọc dữ liệu đầu vào và chế độ đánh giá các ngõ PWM. Ở chế độ Compare, STM32 cung cấp hàm chuẩn so sánh và bộ tạo xung PWM.



Hình 1.31 Mỗi một kênh Capture/Compare đều có một thanh ghi đơn cấu hình chế độ hoạt động.

### Khối Capture

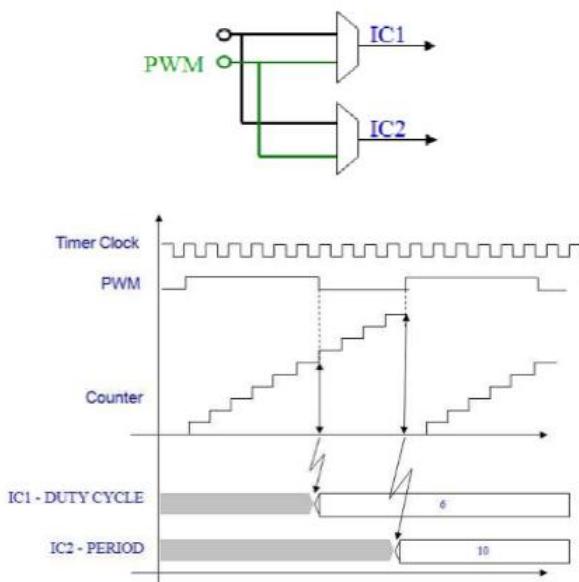
Một khối Capture cơ bản gồm có bốn kênh vào để cấu hình bộ phát hiện xung(Edge Detector). Khi một xung lên(rising edge) hay xung cạnh xuống( falling edge) được phát hiện, bộ đếm hiện thời của sẽ được cập nhật vào các thanh ghi 16-bit Capture/Compare.



Hình 1.32 4 kênh vào của khối Capture có các bộ lọc dữ liệu và phát hiện xung cạnh riêng.

### Chế độ PWM Input

Khối Capture có thể được cấu hình dùng 2 ngõ Capture đầu vào để đo tín hiệu PWM ở ngoài.



Hình 1.33 Ngõ vào Capture và xung PWM

Ở chế độ đo tín hiệu PWM, 2 kênh Capture được dùng để đo chu kỳ Period và Duty của sóng PWM.

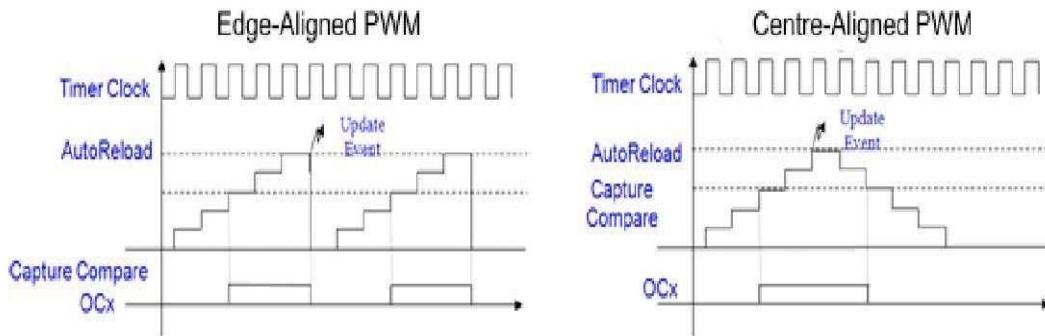
M3->CR1 = 0x00000000;//default
TIM3->PSC = 0x000000FF;//set max prescale
TIM3->ARR = 0x00000FFF;//set max reload count
TIM3->CCMR1 = 0x00000001;//Input IC1 mapped to TI1
TIM3->CCER  = 0x00000000;//IC1 triggers on rising edge
TIM3->CCMR1  = 0x00000200;//Input IC2 mapped to TI1
TIM3->CCMR1  = 0x00000020;//IC2 triggers on falling edge
TIM3->SMCR = 0x00000054; //Select TI1FP1 as input, rising edge trigger

<pre>//reset counter TIM3-&gt;CCER = 0x00000001;//enable capture channel TIM3-&gt;CR1</pre>	<pre>= 0x00000001;//enable timer</pre>
---	--

Ở chế độ PWM sử dụng 2 kênh Capture. Ở thời điểm bắt đầu chu kỳ PWM, bộ đếm được thiết lập giá trị 0 và bắt đầu đếm lên khi phát hiện ra các tín hiệu cạnh lên(rising edge). Khi tín hiệu cạnh xuống được phát hiện(falling edge) giá trị bộ đếm giá trị của chu kỳ Duty được tăng thêm.

### Chế độ PWM

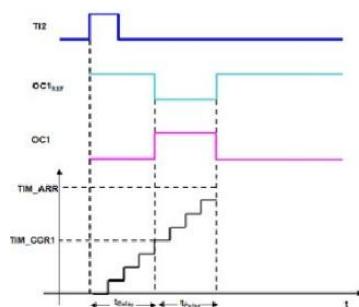
Mỗi khối Timer đều có khả năng tạo các xung nhịp PWM. Ở chế độ tạo xung PWM, giá trị Period được lưu trong thanh ghi Auto Reload. Trong khi đó giá trị Duty được lưu ở thanh ghi Capture/Compare. Có hai kiểu tạo xung PWM, một là canh lề(edge-aligned) và canh lề giữa(centre-aligned). Với edge-aligned cạnh xuống của tín hiệu trùng với thời điểm thanh ghi reload cập nhật lại giá trị. Với centre-aligned thời điểm thanh ghi reload cập nhật lại là khoảng giữa của chu kỳ Duty.



Hình 1.34 Mỗi khói Timer đều có khả năng tạo ra các xung PWM

### Chế độ One Pulse

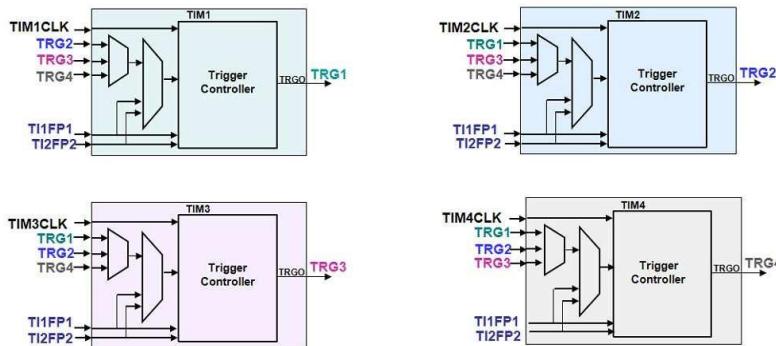
các chế độ đã trình bày trên, ta thấy xung nhịp PWM được tạo có dạng dãy các tín hiệu liên tiếp nhau. Khối Timer còn cung cấp một chế độ hoạt động riêng cho phép tạo duy nhất một xung PWM với tần số, bê rộng xung cùng với thời gian trễ có khả năng được cấu hình một cách linh động.



Hình 1.35 Chế độ One Pulse

- Đồng bộ hóa định thời

Mặc dù các bộ định thời hoạt động hoàn toàn độc lập với nhau, tuy nhiên chúng có thể được đồng bộ hóa từng đôi một hay toàn bộ.

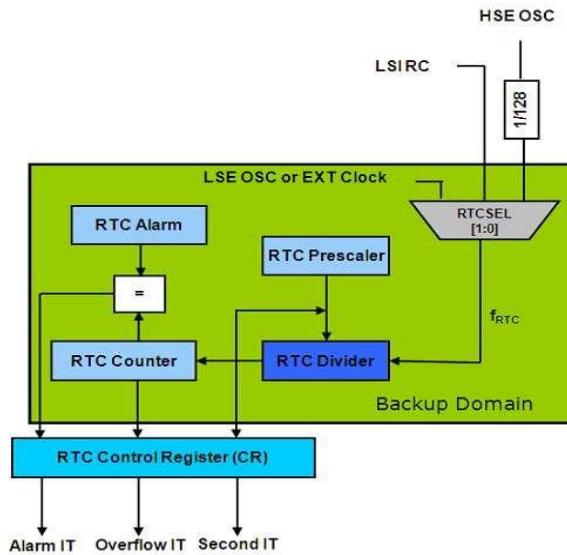


Hình 1.36 Mỗi khối Timer có đầu vào là các xung sự kiện

Mỗi khối Timer 3 đường vào hỗ trợ các xung sự kiện từ 3 khối Timers còn lại. Ngoài ra chân Capture từ Timer1 và Timer2(TIFP1 và TIFP2) cũng được đưa khói điều khiển sự kiện của mỗi Timer.

#### e. RTC và các thanh ghi Backup

STM32 bao gồm 2 khối nguồn chính: nguồn dành cho nhân CPU, các thiết bị ngoại vi và nguồn dành cho khối dự phòng. Cùng được thiết kế chung với khối dự phòng là 10 thanh ghi 16-bit, đồng hồ thời gian thực RTC và một khối Watchdog độc lập. Các thanh ghi dự phòng đơn giản chỉ là 10 vùng nhớ để lưu các giá trị dữ liệu quan trọng khi hệ thống đi vào chế độ Standby và nguồn chính của hệ thống bị ngắt. Ở chế độ tiết kiệm năng lượng, đồng hồ RTC và Watchdog có thể được dùng kích hoạt hệ thống hoạt động trở lại. STM32 có một đồng hồ thời gian thực với thanh ghi đếm 32-bit và giá trị tăng lên một sau mỗi giây nếu xung nhịp đầu vào của nó là 32.768KHz. Khi cấu hình xung nhịp hoạt động hệ thống, xung nhịp nguồn cho đồng hồ RTC này có thể được lấy từ 3 nguồn: LSI, LSE, HSE với giá trị chia là 128.



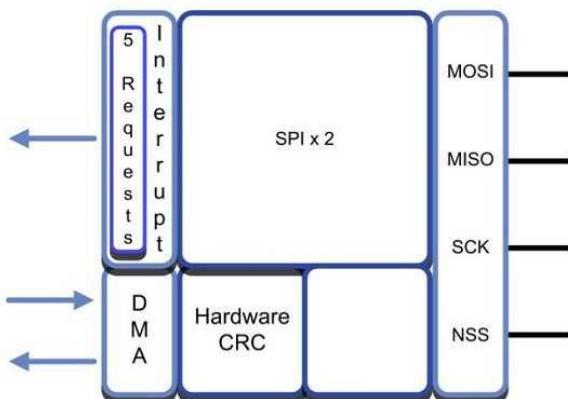
Hình 1.37 Khối RTC có thể lấy nguồn xung nhịp từ LSI, LSE và HSE.

#### 1.4.2 Kết nối với các giao tiếp khác

STM32 hỗ trợ 5 loại giao tiếp ngoại vi khác nhau. STM32 có giao diện SPI và I2C để giao tiếp với các mạch tích hợp khác. Hỗ trợ giao tiếp CAN cho các module, USB cho giao tiếp PC và giao tiếp USART.

##### a. SPI

Hỗ trợ giao tiếp tốc độ cao với các mạch tích hợp khác, STM cung cấp 2 khối điều khiển SPI có khả năng chạy ở chế độ song công(Full duplex) với tốc độ truyền dữ liệu lên tới 18MHz. Khối SPI tốc độ cao nằm trên APB2, khối SPI tốc độ thấp nằm trên APB1. Mỗi khối SPI có hệ thống thanh ghi cấu hình độc lập, dữ liệu truyền có thể dưới dạng 8-bit hoặc 16-bit, thứ tự hỗ trợ MSB hay LSB. Chúng ta có thể cấu hình mỗi khối SPI đóng vai trò master hay slave.



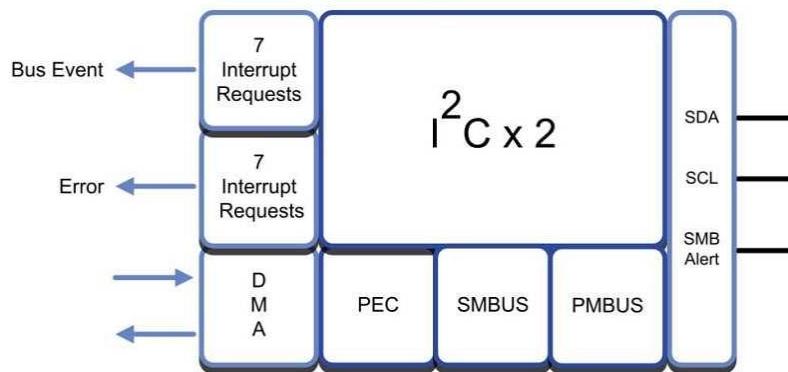
Hình 1.38 giao tiếp SPI

Để hỗ trợ truyền dữ liệu tốc độ cao, mỗi khối SPI có 2 kênh DMA dành cho gửi và nhận dữ liệu.Thêm vào đó là khối CRC dành cho cả truyền và nhận dữ liệu. Khối CRC

đều có thẻ hỗ trợ kiểm tra CRC8 và CRC16. Các đặc tính này rất cần thiết khi sử dụng SPI để giao tiếp với MMC/SD card.

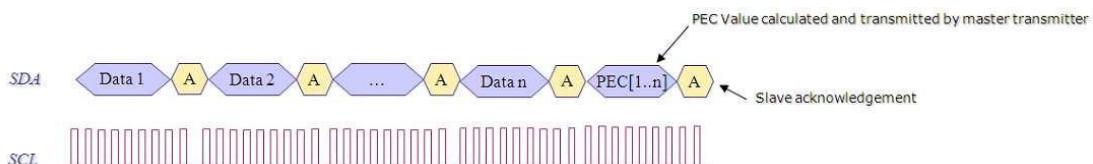
### b. I2C

Tương tự như SPI, chuẩn I2C cũng được STM32 hỗ trợ nhằm giao tiếp với các mạch tích hợp ngoài. Giao diện I2C có thể được cấu hình hoạt động ở chế độ slave, master hay đóng vai trò bộ phân xử đường trong hệ thống multi-master. Giao diện I2C hỗ trợ tốc độ truyền chuẩn 100kHz hay tốc độ cao 400kHz. Ngoài ra còn hỗ trợ 7 hoặc 10 bit địa chỉ. Được thiết kế nhằm đơn giản hóa quá trình trao đổi với 2 kênh DMA cho truyền và nhận dữ liệu. Hai ngắt một cho nhân Cortex, một cho định địa chỉ và truyền nhận



Hình 1.39 Giao tiếp I2C

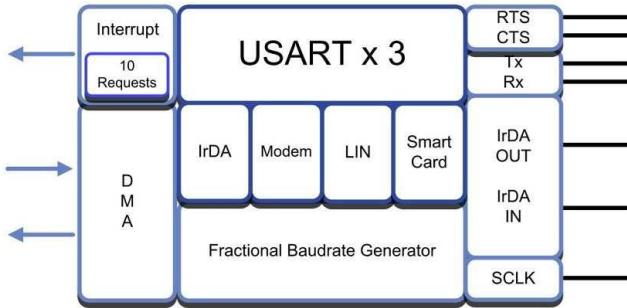
Thêm nữa để đảm bảo tính chính xác dữ liệu truyền, khôi kiểm tra lỗi dữ liệu( PAC – packet error checking) được tích hợp thêm vào giao diện I2C cho phép kiểm tra mã CRC-8 bit. Thao tác này được thực hiện hoàn toàn tự động bởi phần cứng.



Hình 1.40 Kiểm tra lỗi trên I2C

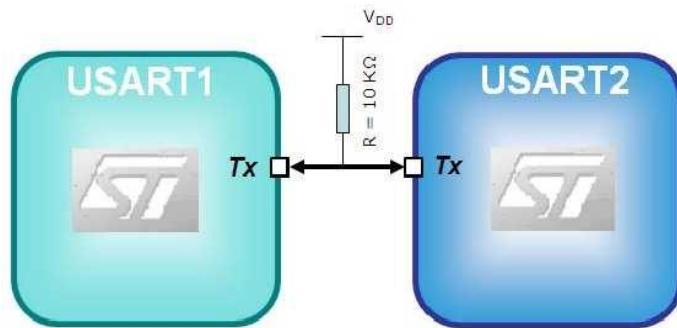
### c. USART

Mặc dù các giao diện trao đổi dữ liệu dạng nối tiếp dần dần không còn được hỗ trợ trên máy tính, chúng vẫn còn được sử dụng rất nhiều trong lĩnh vực nhúng bởi sự tiện ích và tính đơn giản. STM32 có đến 3 khôi USART, mỗi khôi có khả năng hoạt động đến tốc độ 4.5Mbps. Một khôi USART nằm trên APB1 với xung nhịp hoạt động 72MHz, các khôi còn lại nằm trên APB2 hoạt động ở xung nhịp 36MHz.



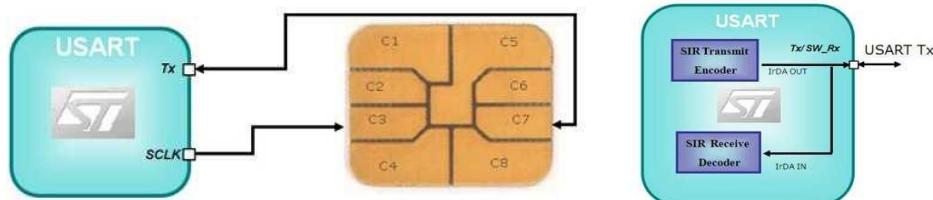
Hình 1.41 Giao diện USART có khả năng hỗ trợ giao tiếp không đồng bộ UARTS, modem cũng như giao tiếp hồng ngoại và Smartcard.

Với mạch tích hợp cho phép chia nhỏ tốc độ BAUD chuẩn thành nhiều tốc độ khác nhau thích hợp với nhiều kiểu trao đổi dữ liệu khác nhau. Mỗi khối USART có hai kênh DMA dành cho truyền và nhận dữ liệu. Khi hỗ trợ giao tiếp dạng UART, USART cung cấp nhiều chế độ giao tiếp. Có thể trao đổi dữ liệu theo kiểu chế độ half-duplex trên đường truyền Tx. Khi hỗ trợ giao tiếp modem và giao tiếp có sử dụng điều khiển luồng (hardware flow control) USART cung cấp thêm các tín hiệu điều khiển CTS và RTS.



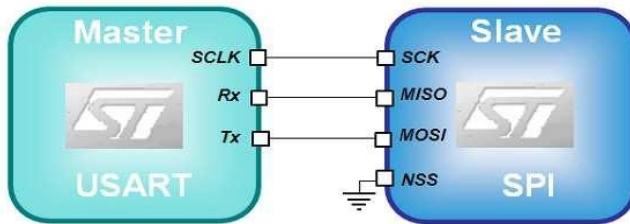
Hình 1.42 Hỗ trợ giao tiếp ở chế độ half-duplex dựa trên một đường truyền

Ngoài ra USART còn có thể dùng để tạo các giao tiếp nội (local interconnect bus). Đây là mô hình cho phép nhiều vi xử lý trao đổi dữ liệu lẫn nhau. USART còn có khối encoder/decoder dùng cho giao tiếp hồng ngoại với tốc độ hỗ trợ có thể đạt đến 1115200bps, hoạt động ở chế độ half-duplex NRZ khi xung nhịp hoạt động khoảng từ 1.4MHz cho đến 2.12Mhz. Để thực hiện giao tiếp với smartcard, USART còn hỗ trợ chuẩn ISO 7618-3.



Hình 1.43 Giao tiếp smartcard và hồng ngoại

Người dùng có thể cấu hình khối USART cho các giao tiếp đồng bộ tốc độ cao dựa trên 3 đường tín hiệu riêng biệt như SPI. Khi hoạt động ở chế độ này, khối USART sẽ đóng vai trò là SPI master và có khả năng cấu hình Clock Polarity/Phase nên hoàn toàn có thể giao tiếp với các SPI slave khác.



Hình 1.44 Hỗ trợ giao tiếp đồng bộ SPI

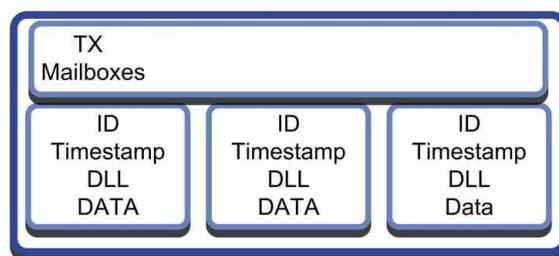
#### d. CAN

Khối điều khiển CAN cung cấp một điểm giao tiếp CAN đầy đủ hỗ trợ chuẩn CAB 2.0A và 2.0B Active và Passive với tốc độ truyền dữ liệu 1 Mbit/s. Ngoài ra khối CAN còn có khối mở rộng hỗ trợ giao tiếp truyền dữ liệu dạng deterministic dựa trên thời gian Time-trigger CAN(TTCAN).



Hình 1.45 khối điều khiển CAN

Tên đầy đủ của CAN là bxCAN, trong đó bx là viết tắt của Base eXtended. Một giao diện cơ bản CAN tối thiểu phải hỗ trợ bộ đệm đơn truyền và nhận dữ liệu, trong khi đó các giao diện mở rộng cung cấp nhiều bộ đệm. bxCan là sự kết hợp giữa hai kiến trúc trên. bxCan có 3 bộ đệm dữ liệu cho truyền và 2 bộ đệm nhận, các bộ đệm này thường được gọi là mailbox(hộp thư). Mỗi mailbox được tổ chức như một FIFO hàng đợi



Hình 1.46 Khối CAN có 3 mailbox cho truyền dữ liệu với đánh nhãn thời gian tự động cho chuẩn TTCAN

Một điểm quan trọng nữa của CAN là lọc gói tin nhận(receive message filter). Vì giao thức CAN truyền dữ liệu dựa trên địa chỉ đích nhận, do đó gói tin sẽ được phát trên toàn bộ mạng, chỉ có điểm nào có địa chỉ giống như địa chỉ nhận trên gói tin sẽ dùng gói tin đó. Lọc gói tin giúp các điểm trên mạng CAN tránh xử lý các gói tin không phù hợp. STM32 cung cấp 14 bộ lọc(14 filters bank) được đánh số từ 0-13 cho phép lọc toàn bộ các gói tin không cần thiết. Mỗi bộ lọc gồm 2 thanh ghi 32-bit CAN\_FxR0 và CAN\_FxR1.

### e. USB

Hỗ trợ giao tiếp Device USB với tốc độ Full Speed (12Mbps) có khả năng kết nối với một giao diện host usb. Khối giao diện này bao gồm Layer1 và Layer2 đảm nhận chức năng truyền vật lý(physical layer) và truyền dữ liệu logic (data layer). Ngoài ra còn hỗ trợ đầy đủ chế độ Suspend và Resume nhằm tiết kiệm năng lượng.



*Hình 1.47 Giao tiếp USB 2.0*

Với 8 endpoint, có thể hoạt động dưới các chế độ : Control, Interrupt, Bulk hoặc Isochronous. Vùng đệm dữ liệu 512 byte SRAM của các endpoint được chia sẻ với giao diện CAN. Khi được cấu hình, ứng dụng sẽ chia vùng đệm này thành các phần tương ứng với các endpoint. Các vùng đệm này đảm bảo dữ liệu được truyền nhận liên tục trên mỗi endpoint.

## 1.5 Chế độ tiêu thụ năng lượng thấp

STM32 có nhiều chế độ công suất thấp bên cạnh chế độ bình thường (normal RUN mode). Khi sử dụng một cách đúng đắn các chế độ công suất thấp(SLEEP, STOP, STANDBY) sẽ làm tối ưu nguồn pin. Khi vào chế độ công suất thấp, CPU và các ngoại vi Cortex được tạm dừng và tiêu thụ công suất tối thiểu. Một khi bộ xử lý Cortex vào chế độ công suất thấp, nó xuất một tín hiệu SLEEPDEEP đến các vi điều khiển xung quanh, để ra hiệu rằng nó đã vào một chế độ công suất thấp nào đó. CPU Cortex vào các chế độ công suất thấp bằng cách thực hiện lệnh WFI hoặc WFE. Tùy vào cấu hình thanh ghi điều khiển công suất (power control registers) mà STM32 sẽ đi vào các trạng thái tiêu thụ thấp tương ứng..

### 1.5.1 Chế độ bình thường – RUN Mode

RUN mode là chế độ STM32 thực hiện các lệnh chương trình và nó tiêu thụ công suất ở mức cao nhất. Trong phần này sẽ chỉ ra nhiều cách để giảm công suất tiêu thụ tổng thể của hệ thống trong chương trình thực thi. Điểm cốt lõi là tất cả các chức năng

của hệ thống nên được dùng một cách linh hoạt khi thực thi mã chương trình. Nghĩa là khi có thể, mã nên được chạy ở chế độ công suất thấp-hiệu năng xử lý thấp, và chuyển sang chế độ công suất cao-cấu hình hiệu năng cao để đáp ứng ngắt và chương trình sự kiện (program event).

### a. Chế độ Half-cycle và Prefetch-buffer

Khi chạy trực tiếp từ bộ dao động ngoài HSE với tần số tối đa 8MHz, ta có thể tắt bộ đệm lấy trước lệnh (FLASH Prefetch Buffer) và cho kích hoạt chế độ nửa chu kỳ (Half Cycle). Làm như vậy tuy phải chịu thêm một số trạng thái chờ, nhưng giảm được công suất tiêu thụ ở chế độ RUN mode.

#### 1.5.2 Các chế độ sử dụng công suất tiêu thụ thấp

Cấu hình chế độ RUN mode của STM32 một cách cẩn thận có thể giảm công suất tiêu thụ khoảng 8,5mA. Nhằm đạt được các ứng dụng công suất thấp thực chúng ta phải sử dụng các chế độ công suất thấp của STM32.

##### a. SLEEP

Mức đầu tiên của hoạt động công suất thấp là chế độ SLEEP mode. Mặc định, khi một lệnh WFE hoặc WFI được thực hiện bộ xử lý Cortex sẽ tạm dừng các đồng hồ xung nhịp nội và dừng thực thi mã ứng dụng. Trong chế độ SLEEP các phần còn lại của STM32 vẫn tiếp tục hoạt động. STM32 sẽ thoát khỏi SLEEP mode khi một ngoại vi nào đó phát sinh ngắt. Khi STM32 vào SLEEP mode cùng tất cả ngoại vi đang hoạt động và nó chạy ở 72MHz từ HSE thông qua bộ nhân PLL, công suất tiêu thụ của nó vào khoảng 14.4mA.

##### b. STOP Mode

STM32 có thể được cấu hình để vào chế độ công suất thấp STOP Mode bằng cách thiết lập bit SLEEPDEEP trong thanh ghi điều khiển công suất Cortex (the Cortex power control register) và xóa bit Power Down Deep Sleep (PDDS) trong thanh ghi điều khiển công suất STM32. Một khi STM32 đã vào chế độ STOP, tiêu thụ điện năng của nó giảm mạnh xuống khoảng 24 uA thay vì hàng mA ở chế độ RUN.

Conditions	$V_{DD}/V_{BAT} = 2.4\text{ V}$	$V_{DD}/V_{BAT} = 3.3\text{ V}$	Unit	Symbol	Parameter	Conditions	Type	Unit
Regulator in Run mode, low-speed and high speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog)	NA	24	$\mu\text{A}$	$t_{WUSTOP}$	Wakeup from Stop mode (regulator in run mode)	Wakeup on HSI RC clock	3.52	$\mu\text{s}$
Regulator in Low Power mode, low speed and high speed internal RC oscillators and high speed oscillator OFF (no independent watchdog)	NA	14			Wakeup from Stop mode (regulator in run mode + WFI)		5.42	
					Wakeup from Stop mode (regulator in Low power mode + WFE)		5.32	
					Wakeup from Stop mode (regulator in Low power mode + WFI)		7.21	

Hình 1.48 Thời gian đánh thức trong chế độ Stop dài nhất là 5,5 us với bộ điều áp chạy bình thường và 7,3 us với bộ điều áp trong chế độ công suất thấp

### 1.5.3 Stanby

STM32 có thể được cấu hình để vào chế độ Standby bằng cách thiết lập bit **SLEEPDEEP** trong thanh ghi điều khiển công suất Cortex và thiết lập bit Power Down Deep Sleep (PDDS) trong thanh ghi điều khiển công suất của STM32. Bây giờ, khi lệnh WFI hoặc WFE được thực hiện, STM32 sẽ vào chế độ năng lượng thấp nhất của nó. Trong chế độ Standby, STM32 thực sự tắt. Các điều kiện áp nội bộ được tắt các bộ tạo dao động HSE và HIS cũng tắt. Trong chế độ này STM32 chỉ chiếm 2uA.

Conditions	$V_{DD}/V_{BAT} = 2.4\text{ V}$	$V_{DD}/V_{BAT} = 3.3\text{ V}$	Unit
Low-speed internal oscillator and independent watchdog OFF, low speed oscillator and RTC OFF	NA	2	$\mu\text{A}$
Low-speed oscillator and RTC ON	1.08	1.4	
Reset	Wake-up from deep stand-by mode	HSI RC clock Wake-up on	20 $\mu\text{s}$
Standby	Deepsleep	Conditions	Unit

Hình 1.49 Trong chế độ Standby tiêu thụ điện năng là 2uA với thời gian đánh thức là 50uS.

Chúng ta có thể thoát khỏi chế độ Standby bằng cách sử dụng tín hiệu ngắn của khối RTC tương tự như trong chế độ STOP. Ngoài ra ta có thể sử dụng chân Reset ngoại hay chân Reset từ Watchdog độc lập.

### 1.5.4 Sự tiêu thụ công suất của nguồn dự phòng

Vùng năng lượng dự phòng (Backup Region) chứa pin dự phòng để duy trì RAM và RTC trong suốt thời gian các chế độ công suất thấp. Vùng này tiêu thụ khoảng 1,4 uA ở điện áp 3,3V.

### 1.5.5 Hỗ trợ Debug

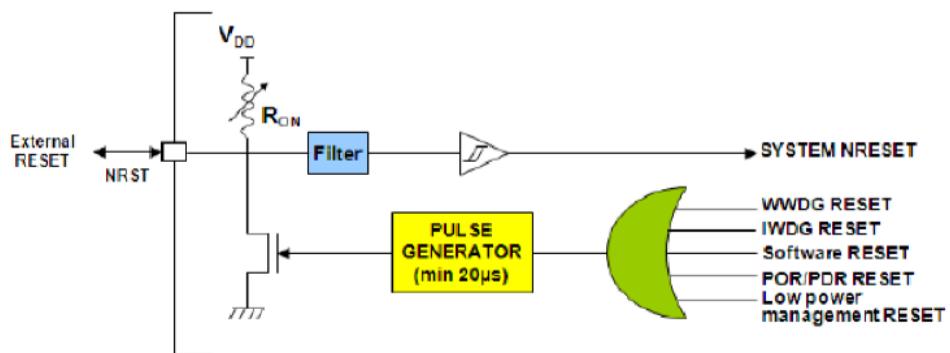
Trên hệ thống vi điều khiển truyền thống, gỡ lỗi một ứng dụng mà sử dụng chế độ công suất thấp có thể mất rất nhiều công sức. Ngay sau khi vi điều khiển vào các chế độ công suất thấp nó dừng đáp ứng các trình gỡ lỗi, điều này sẽ tạo một lỗi hoặc ngừng làm việc. Trong STM32 có thể cấu hình các chế độ công suất thấp để giữ bộ dao động nội HSI chạy trong các chế độ năng lượng thấp tương ứng, cung cấp một đường xung nhịp dành cho khối gỡ lỗi CoreSight.

## 1.6 Tính an toàn

STM32 cũng đã được thiết kế với một số tính năng vốn có mà sẽ phát hiện các hoạt động không chính xác của các mã ứng dụng hoặc của chính STM32. Để đảm bảo rằng có một nguồn cung cấp năng lượng đáng tin cậy, STM32 có một bộ reset nội thực hiện chức năng reset chip nếu các nguồn cung cấp điện áp dưới mức tối thiểu VDD.

### 1.6.1 Reset Control

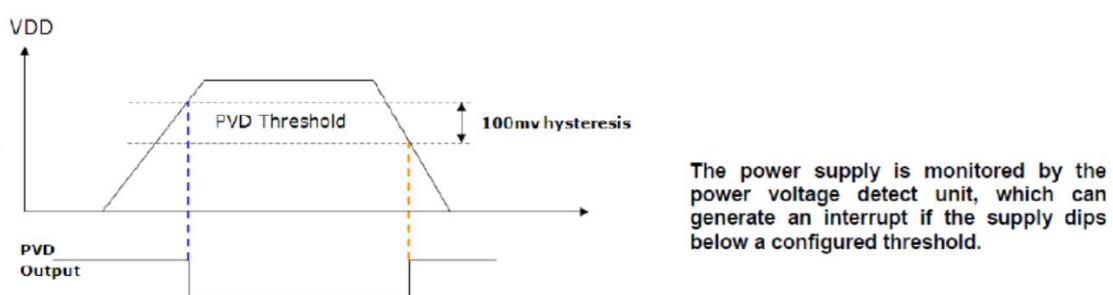
STM32 có nhiều nguồn Reset khác ngoài đường Reset bên ngoài. STM32 có thể bị buộc khởi động lại từ: các Watchdogs nội, một Reset mềm thông qua NVIC, bộ Reset mở/tắt nguồn nội và mạch phát hiện điện áp nguồn thấp. Nếu một hiệu lệnh Reset xuất hiện, một bộ cờ trong thanh ghi kiểm soát và trạng thái RCC có thể được đọc để xác định nguyên nhân gây ra Reset.



Hình 1.50 STM32 có thể có nhiều tín hiệu điều khiển để đưa hệ thống về trạng thái Reset. Trạng thái hệ thống sẽ được lưu lại trong thanh ghi RCC

### 1.6.2 Kiểm tra điện áp nguồn

Là một phần của bộ giám sát nguồn cung cấp nội của STM32 chứa một đơn vị giám sát nguồn được gọi là bộ Kiểm tra điện áp nguồn - Power Voltage Detect (PVD). PVD có ngưỡng khả trình có thể được thiết lập với sai số 0.1V từ 2.2V đến 2.9V. Ngưỡng này được cấu hình trong các thanh ghi kiểm soát nguồn.

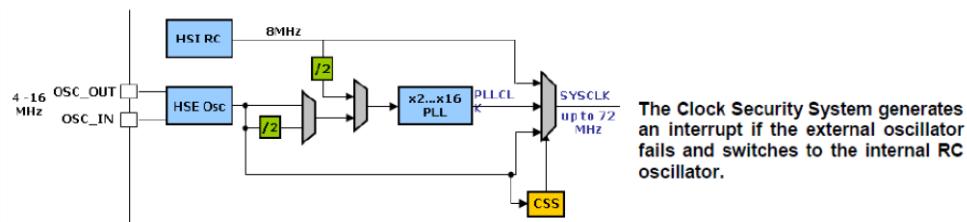


Hình 1.51 Đầu ra của PVD được kết nối với 16 đường của bộ ngắt ngoài.

Do các đường ngắt ngoài EXTI có thể được cấu hình để tạo ra ngắt từ tín hiệu cạnh lên hoặc cạnh xuống, hoặc cả hai, đơn vị PVD có thể được sử dụng để tạo ra một ngắt cho cả thấp áp và quá áp.

### 1.6.3 Hệ thống an toàn xung nhịp

Trong hầu hết các ứng dụng, xung nhịp hệ thống được được dùng bởi bộ xử lý Cortex và các thiết bị ngoại vi sẽ lấy từ một dao động thạch anh bên ngoài kết nối thông qua các chân HSE. Hệ thống tạo xung nhịp dao động này chứa một khối CSS thực hiện nhiệm vụ giám sát dao động thạch anh ngoài. Nếu nguồn dao động thạch anh ngoài có vấn đề, nó sẽ sử dụng hệ thống tạo dao động nội HSI 8MHz.



Hình 1.52 Hệ thống dao động thạch anh nội

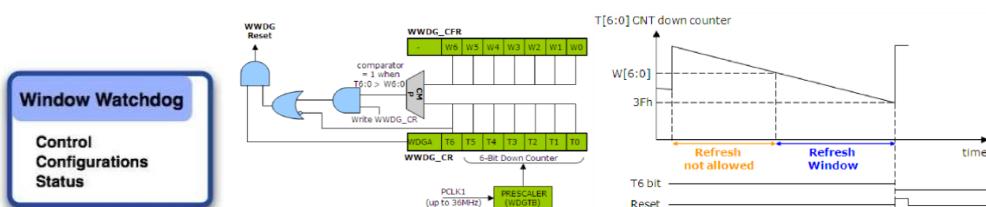
Khối CSS được kích hoạt bằng cách thiết lập bit Clock Security Enable trong thanh ghi điều khiển RCC.

### 1.6.4 Watchdogs

STM32 chứa hai Watchdogs riêng biệt. Watchdog độc lập(independent watchdog) là hoàn toàn tách biệt với hệ thống STM32 chính. Nó được đặt trong miền nguồn điện dự phòng và xung nhịp của nó bắt nguồn từ dao động nội tốc độ thấp (LSI- Low Speed Oscillator). Windowed Watchdog là một phần của hệ thống STM32 chính và được cấp xung clock thông qua đường truyền xung nhịp ngoại vi số 1(PCLK1).

#### a. Windowed Watchdog

Windowed Watchdog là một phiên bản nâng cao hơn của Watchdog truyền thống trên chip. Sau khi kích hoạt, Watchdog sẽ đếm ngược và sẽ tạo ra tín hiệu Reset khi giá trị của thanh ghi thay đổi từ 0x40 sang 0x3F tức là khi bit T6 bị xóa. Giá trị đếm ngừng trên được lưu trong thanh ghi cấu hình Windowed Watchdog.

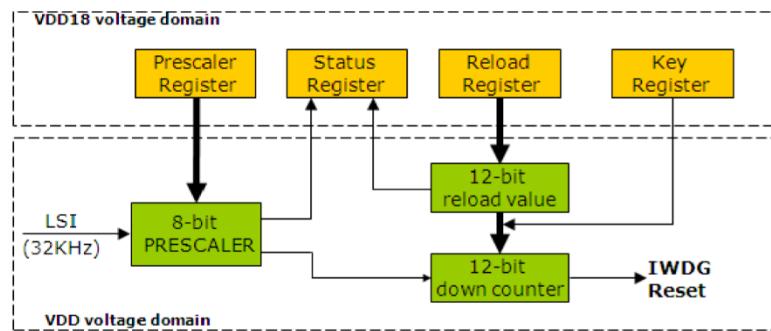


Hình 1.53 Windowed Watchdog

Windowed Watchdog là bộ đếm xuống 6 bit, được cung cấp xung clock từ PCLK1 thông qua bộ tiền chia 12 bit – chia PCKL1 xuống bởi 4096. Bộ tiền chia có thêm 4 bit khả lập trình cho phép người dùng chia thêm 1,2,4 hoặc 8.

### b. Independent Watchdog

Mặc dù Independent Watchdog được thiết kế nằm trên hệ thống chính, nhưng nó có bộ dao động riêng tách biệt với nguồn dao động của hệ thống chính. Independent Watchdog cũng được đặt trong miền điện áp  $V_{DD}$ , là miền mà vẫn được duy trì khi hệ thống ở trong chế độ STOP và STANDBY.



Hình 1.54 Independent Watchdog

Independent Watchdog là một định thời đếm xuống 12 bit, và sẽ phát sinh hiệu lệnh Reset cho hệ thống khi giá trị đạt dưới ngưỡng. Nó được cấp xung nhịp từ dao động nội tốc độ thấp thông qua một bộ chia 8 bit. Bộ tạo dao động LSI có một tần số danh nghĩa là 32kHz, nhưng trong thực tế nó có thể thay đổi giữa 30 KHz đến 60 KHz. Independent Watchdog được khởi động bằng cách, đầu tiên thiết lập thanh ghi Prescaler, khi qua bộ chia Prescaler dao động sẽ có giá trị trong khoảng từ 4 đến 256.

### 1.6.5 Tính năng ngoại vi

Các thiết bị ngoại vi cũng đã được thiết kế với một số tính năng hỗ trợ để đảm bảo sự vận hành an toàn các STM32.

#### a. GPIO Port Locking

Khi các cổng GPIO được khởi tạo, mỗi dòng IO sẽ được cấu hình như một đầu vào hoặc đầu ra. Một khi cấu hình hoàn tất ta có thể khóa nó lại. Điều này ngăn chặn bất cứ thay đổi tình cờ tiếp nào lên cấu hình cổng. Mỗi cổng có thể bị khoá trên một dựa trên bit điều khiển.

#### b. Analog Watchdog

Mỗi bộ chuyển đổi ADC có 2 analog watchdogs. Những Watchdog này có thể được thiết lập để phát ngắt khi trên hoặc dưới ngưỡng điện áp.

#### c. Brak Input

Đối với các ứng dụng dựa trên động cơ, đường Break trong khối định thời nâng cao có thể được dùng để đặt ba ngõ ra PWM bổ sung vào trạng thái định sẵn, để đáp ứng với tín hiệu đầu vào trên chân Break, hoặc mất nguồn tạo dao động chính.

## 1.7 Flash

Bộ nhớ FLASH on-chip của STM32 được bố trí trong 3 vùng chính. Đầu tiên, vùng nhớ FLASH chính được thiết kế để chứa các lệnh chương trình. Đây là vùng nhớ 64 bit, để cung cấp sự truy cập bộ nhớ hiệu quả với bộ đệm lấy lệnh. Vùng nhớ này được chia thành từng trang 4K đối với hoạt động xóa và ghi chương trình flash. Bộ nhớ có độ bền 10.000 chu kỳ, duy trì dữ liệu 30 năm ở 85°C. Hầu hết các bộ nhớ FLASH vi điều khiển chỉ duy trì dữ liệu tốt ở 25 °C, bộ nhớ FLASH của STM32 là một ngoại lệ. Bên cạnh vùng nhớ chương trình chính, là 2 vùng nhớ nhỏ hơn: khói thông tin lớn và khói thông tin nhỏ. Khối thông tin lớn là 2k bộ nhớ FLASH chứa chương trình **bootloader** được cung cấp bởi nhà sản xuất, là chương trình được thiết kế sẵn để tải code về qua cổng USART1.

### 1.7.1 Lập trình và đảm bảo an toàn cho FLASH nội

Bộ nhớ FLASH nội có thể được cập nhật bởi bootloader, bằng JTAG, hoặc bằng chương trình ứng dụng bên trong thông qua một bộ thanh ghi chuyên dụng được gọi là chương trình FLASH và xóa điều khiển FPEC. FPEC cũng được dùng để lập trình các byte tùy chọn trong khói thông tin nhỏ. Khối FPEC được dùng để cho phép trong ứng dụng lập trình của bộ nhớ FLASH. Bộ nhớ FLASH cũng có thể được bảo vệ đọc khỏi các công cụ debug và bảo vệ ghi.

### 1.7.2 Hoạt động xóa và ghi

Sau khi reset, các thanh ghi FPEC được bảo vệ và phải được mở khóa bằng cách ghi một chuỗi đặc biệt vào thanh ghi khóa (key register). Để mở khóa FPEC ta phải ghi 0x45670123, theo sau bởi 0xCDEF89AB. Nếu có lỗi xuất hiện trong chuỗi này, FPEC sẽ ở trạng thái bị khóa cho đến lần reset kế tiếp. Một khi FPEC đã được mở khóa, nó có thể thực hiện hoạt động xóa và ghi trên bộ nhớ FLASH chính. Trong khói bộ nhớ FLASH chính nó có thể thực hiện xóa theo khói hoặc xóa một trang 4k được chọn. Hoạt động xóa khói được thực hiện bằng cách thiết lập đơn giản xóa khói và khởi động các bit trong thanh ghi điều khiển.

### 1.7.3 Các byte Option

Khối thông tin nhỏ có 8 byte Option cấu hình dành cho người dùng. Trong đó 4 byte được dùng để thiết lập hoạt động bảo vệ ghi trên bộ nhớ FLASH chính. Byte thứ năm dùng để thiết lập bảo vệ đọc để tránh sự truy nhập vào các vùng nhớ khi chip ở trong chế độ debug. Byte thứ sáu được dùng để cấu hình công suất thấp và hoạt động reset. Hai byte cuối cùng là những ô bộ nhớ FLASH đơn giản, sẵn có cho người dùng tùy chọn. Trước khi các byte Option có thể được ghi vào, FPEC phải được mở khóa như mô tả ở trên.

## CHƯƠNG II: THIẾT KẾ KIT THÍ NGHIỆM ARM STM32F103

Để thuận tiện cho việc học Vi xử lý ARM Cortex M3 STM32 , em 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

### 2.1 Giới thiệu

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

### 2.2 Thiết kế mạch nguyên lý

Mục đích của đồ án này là phục vụ cho người mới tìm hiểu về ARM Cortex M3 STM32 nên em chỉ thiết kế với một số ngoại vi đơn giản sử dụng GPIO, I2C, SPI,.

Và để thuận tiện cho việc thiết kế bài giảng điện tử và trực quan cho người sử dụng, em đã chọn phần mềm thiết kế mạch in Altium Designer

- Sơ đồ khói Vi xử lý: để thuận tiện cho người dùng về sau, các IO được đưa ra header
- Khối nguồn

Nguồn vào : 7-12VDC qua IC ổn áp LM7805 để tạo áp 5V

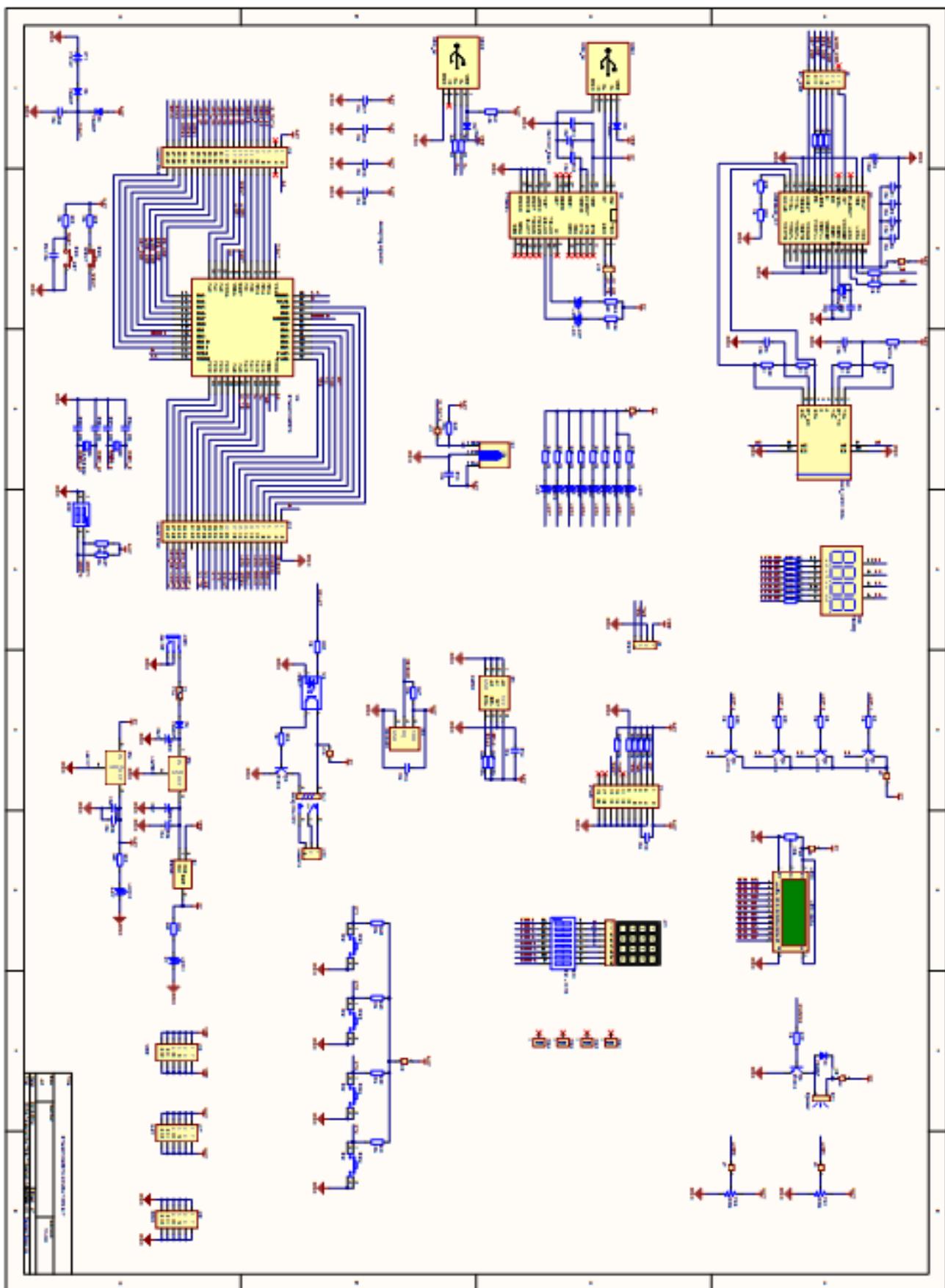
Qua IC ổn áp AMS1117 để lấy nguồn 3.3 V cho vi xử lý

Sử dụng công tắc gạt để lựa chọn nguồn cung cấp ngoài hoặc là nguồn từ cổng USB

- Khối nút nhấn : Mặc định ở trạng thái logic 1( sử dụng trở treo lên 3.3V ), khi nhấn nút thì xuống logic 0
- Khối 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 STLink
- Khối hiển thị Led 7 đoạn: Sử dụng Led anode chung
- Khối hiển thị LCD
- Khối Led đơn
- Khối còi
- Khối ADC
- Khối giao tiếp mạng sử dụng ENC28J60
- Khối thạch anh và chọn chế độ hoạt động của chip
- Khối reset, wake-up, pin Backup
- Khối giao tiếp USB
- Khối giao tiếp USART : Sử dụng chip chuyên dụng FT232 cho phép chuyển đổi từ USB sang RS232 để giao tiếp với vi xử lý thông qua USART và nạp chương trình cho chip qua Bootloader

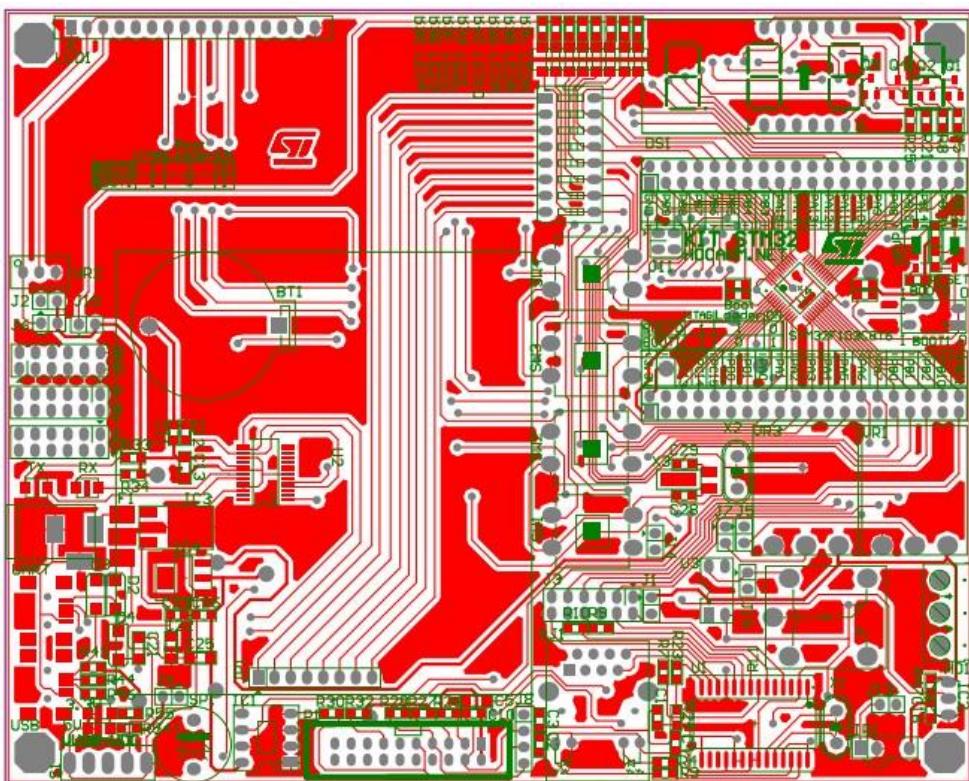
Hình 2.1 Sơ đồ nguyên lý KIT STM32F103



### 2.3 Thiết kế mạch in

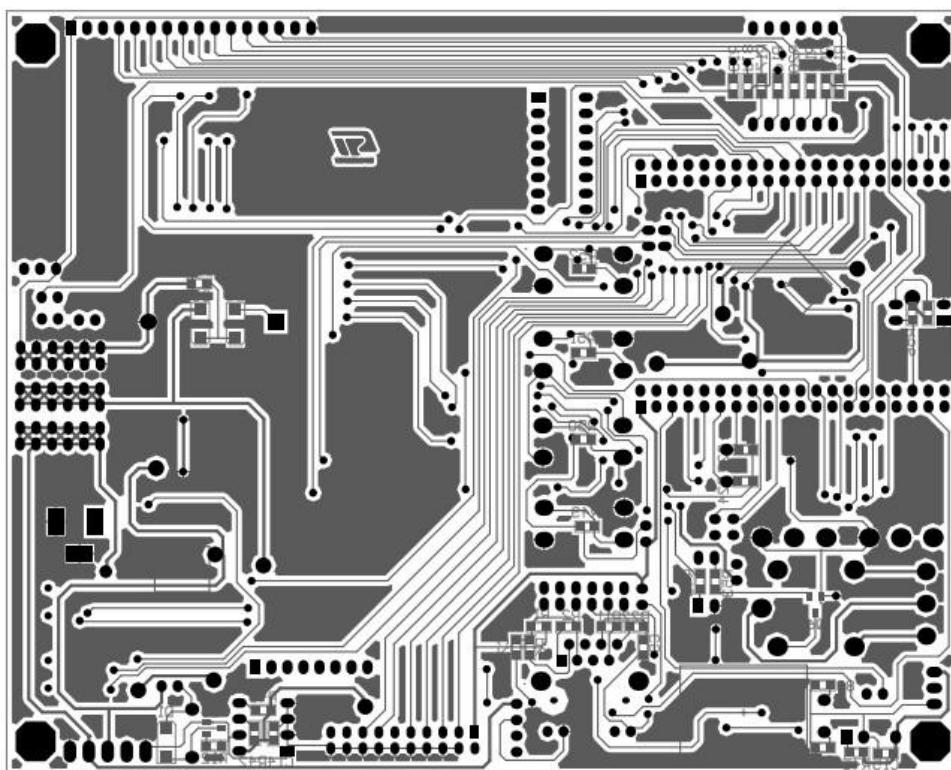
Mạch in được vẽ trên Altium 3D

- Mạch in lớp TOP



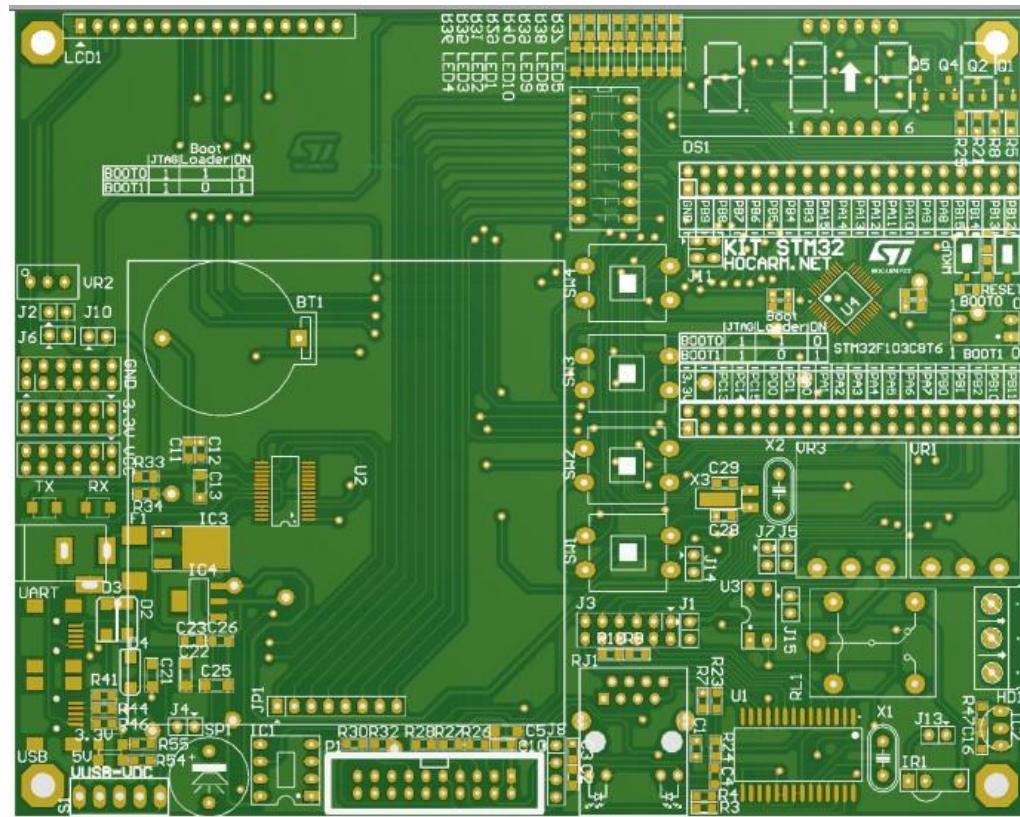
*Hình 2.2 Mạch in lớp TOP*

- Mạch in lớp BOTTOM



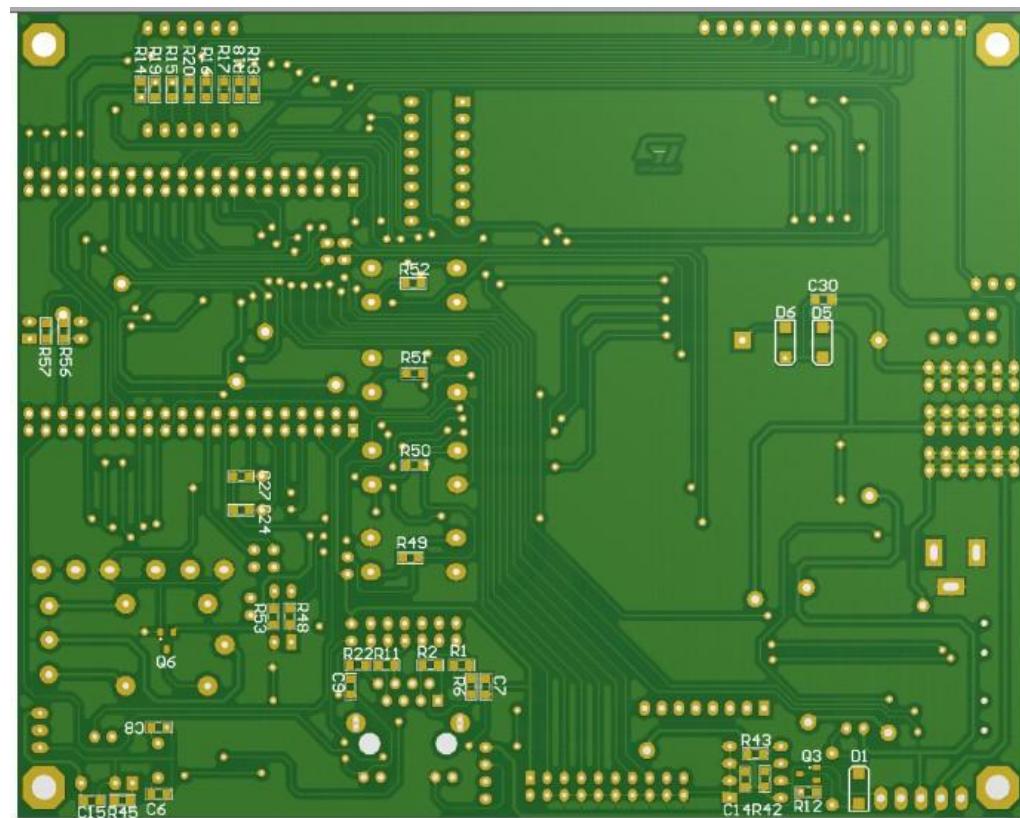
Hình 2.3 Mạch in lớp BOTTOM

- PCB 3D lớp TOP:



Hình 2.4 PCB 3D lớp TOP

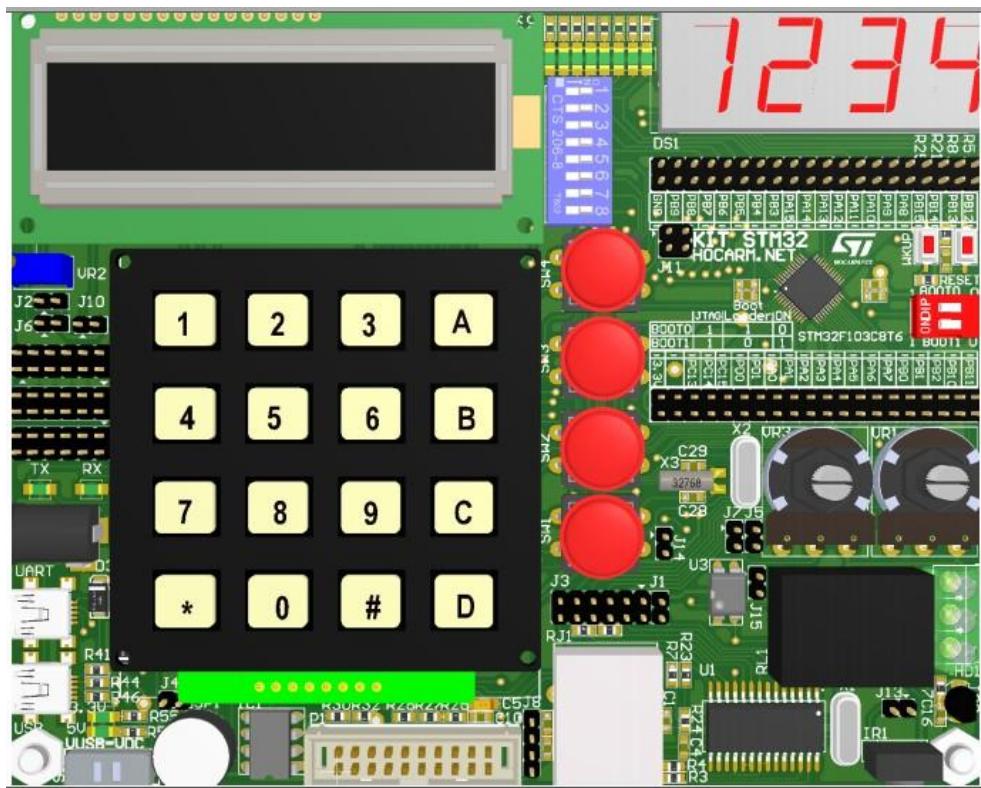
- PCB 3D lớp BOTTOM:



Hình 2.5 PCB 3 D Lớp BOTTOM

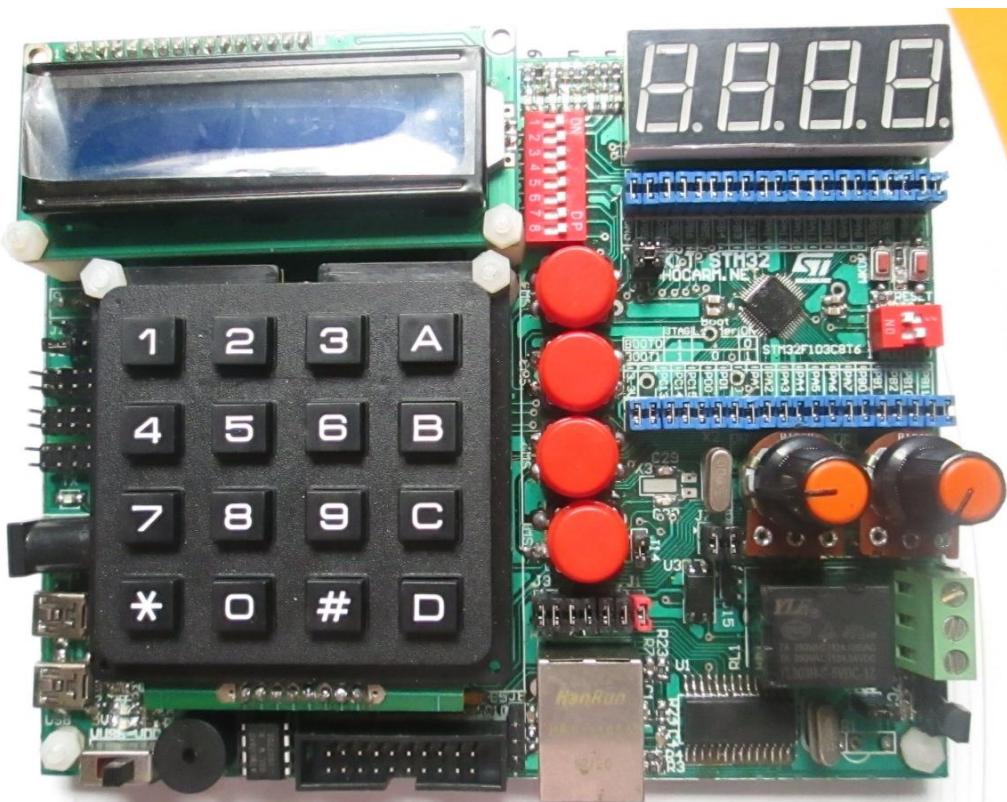
## CHƯƠNG II : THIẾT KẾ KIT THÍ NGHIỆM ARM STM32F103

- Hiển thị ở chế độ 3D



Hình 2.6 Mạch in 3D

- Mạch sau khi lắp ráp



Hình 2.7. Mạch sau khi lắp ráp

## CHƯƠNG III: LẬP TRÌNH ỨNG DỤNG CHO KIT STM32F103

### 3. 1 Hướng dẫn cơ bản cho một ứng dụng với KIT STM32F103

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 RS232 FT232RL** : sử dụng chip FT232 có đầy đủ chức năng các giao thức USB v2.0, driver miễn phí trên Windows, Linux, Mac ,...
- **Flash Bootloader for ARM** : Dùng nạp code cho VDK thông qua Bootloader
- **Keil C MDK** : Lập trình C cho dòng Vi điều khiển ARM
- **Thư viện CMIS** : chuẩn thư viện hỗ trợ cho dòng ARM STM32

#### 3.1.1 Các bước tạo một Project mới trên Keil C MDK

Download và giải nén thư viện CMSIS trên về ta có thư mục **STM32F10x\_StdPeriph\_Lib\_V3.5.0**,

Tạo 1 Foder mới để chứa toàn bộ dữ liệu cho project

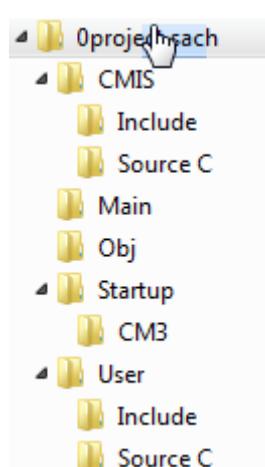
Các thư mục cần tạo trong Project này :

- **CMIS** : file thư viện CMIS

Copy các file trong theo đường dẫn :

..\\STM32F10x\_StdPeriph\_Lib\_V3.5.0\\Libraries\\STM32F10x\_StdPeriph\_Driver

- **Main**: chứa chương trình chính
- **Startup** : các file khởi động hệ thống .
- **User** : Là file driver người dùng hỗ trợ kết nối với KIT STM32F103C8T6
- **Obj** : Chứa file nạp cho chip



Hình 2.7 Các thư mục của một Project

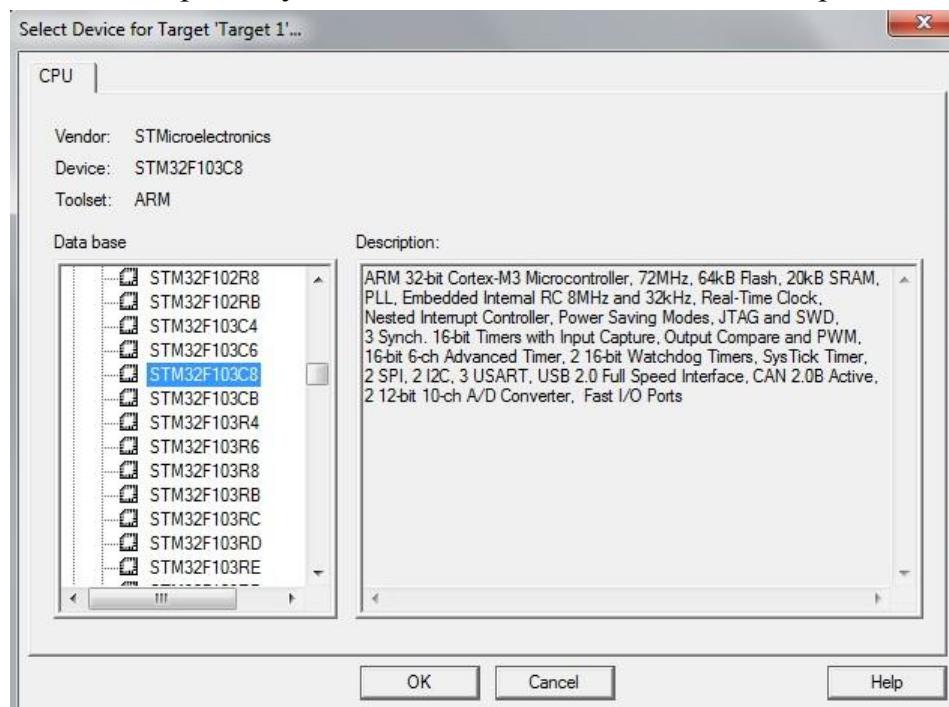
### CHƯƠNG III : LẬP TRÌNH ỨNG DỤNG CHO KIT STM32F103

Để tiện cho việc sử dụng thì chúng ta nên tạo một Project sạch, không viết chương trình gì cả, lúc cần lập trình cái nào thì copy và thêm các driver cần thiết vào để đỡ quá trình thao tác

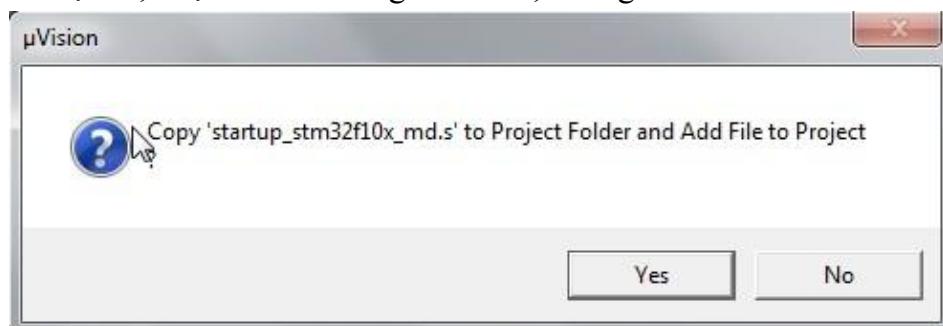
- Mở Keil C lên vào tạo một Project mới



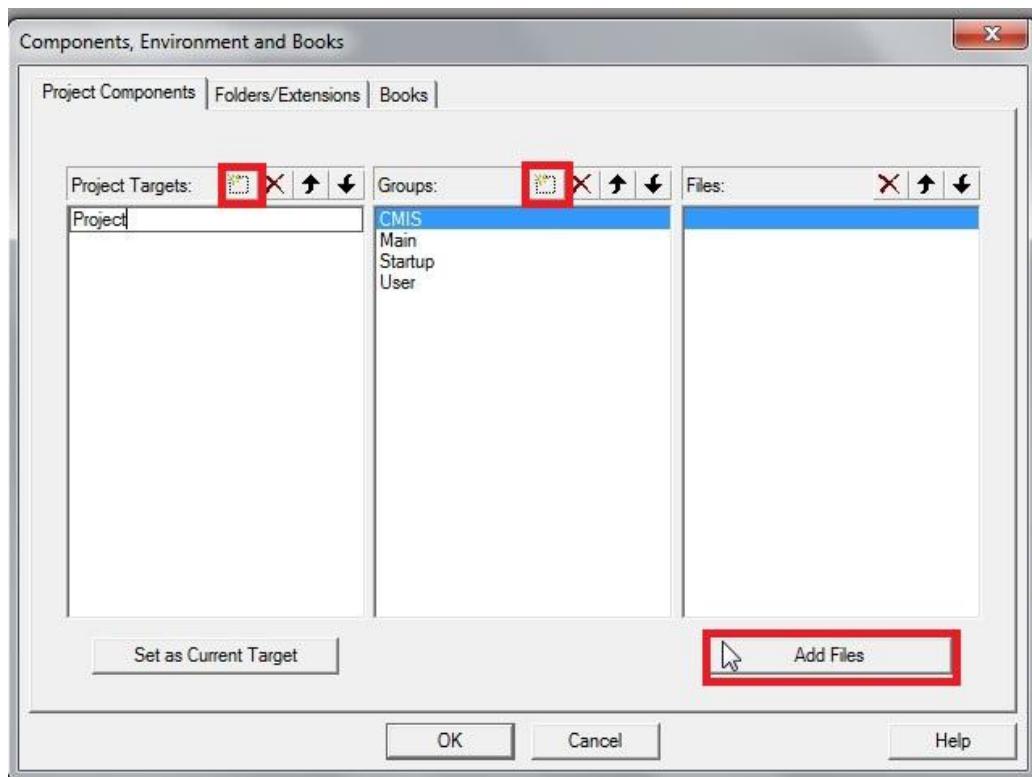
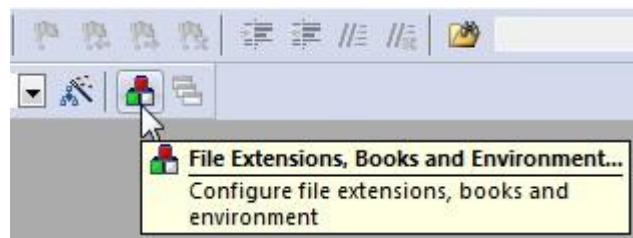
Hiện cửa sổ chọn Chip. Ở đây chọn **STMicroelectronics**. Chọn chip **STM32F103C8**



Cửa sổ mới hiện ra, chọn **No** vì không cần thiết, chúng ta sẽ add sau

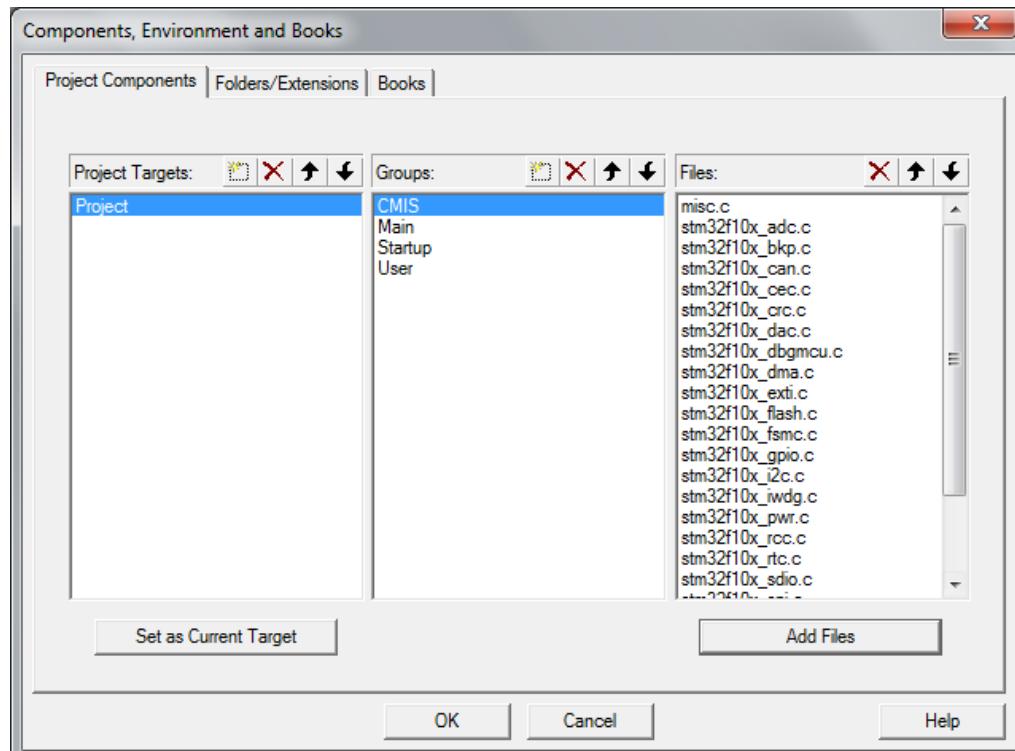


Trong Project mới , nhấp chuột vào **Target**

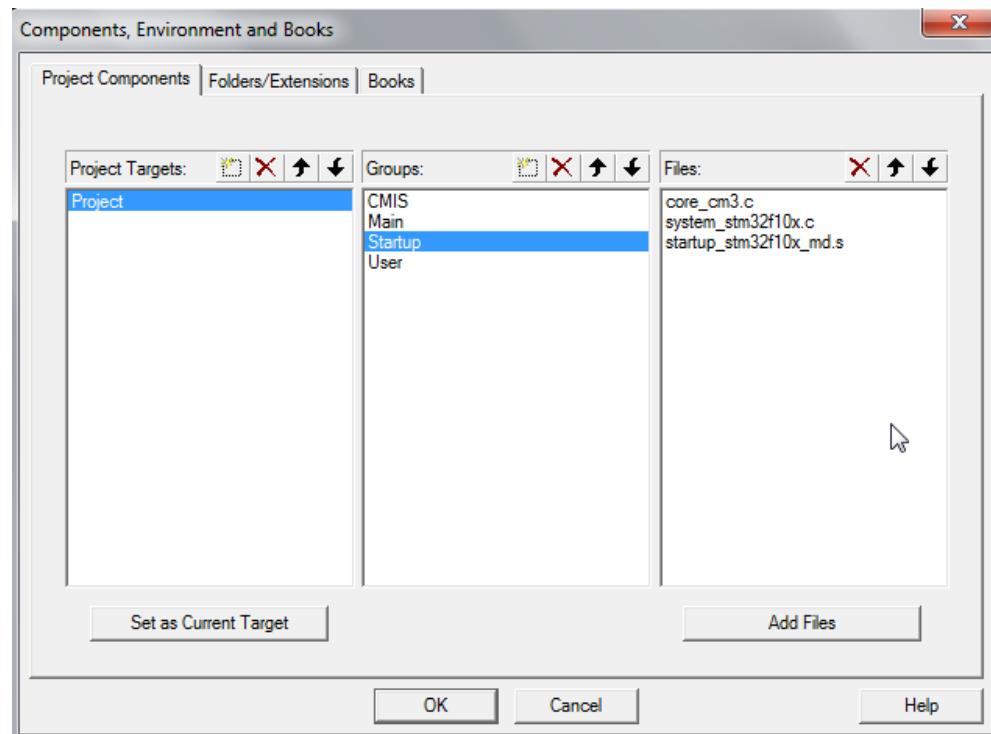


Nhấn vào ô vuông để tạo tên mới cho Project và ô vuông thứ hai để tạo các Group. Như trên là : **Startup**, **Main**, **Cmsis**, **User**. Chọn **add files** để add một số file vào group. Các file cần add đều nằm trong thư mục **Library**

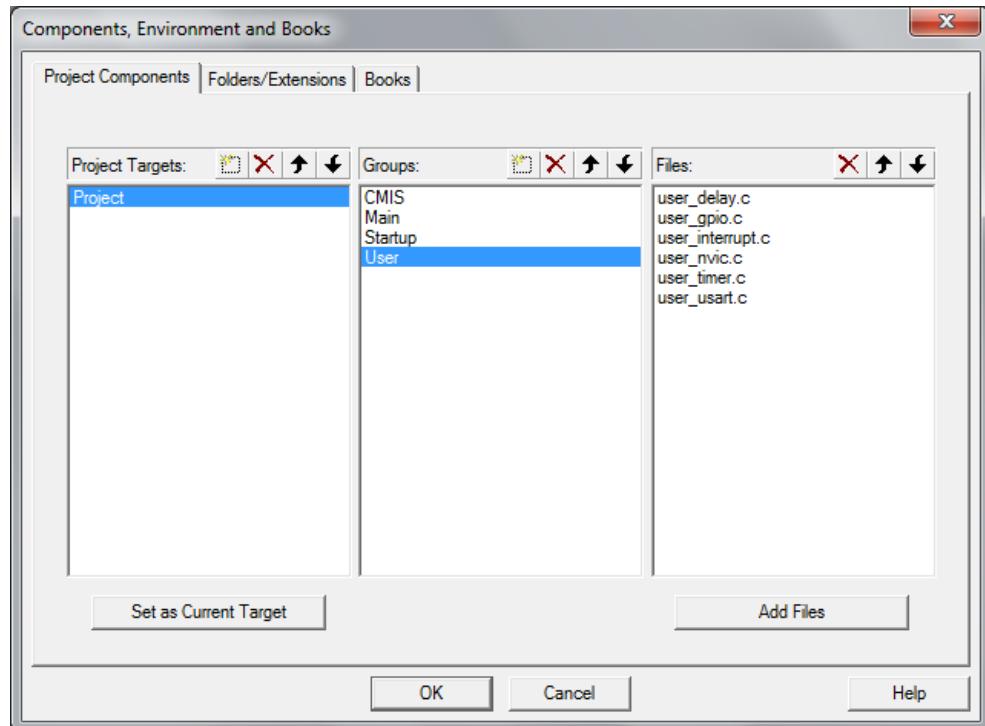
- Group **CMIS** : add các file trong mục **Source** nằm trong thư mục CMIS vừa tạo ở trên



- Groups Stratup : add file **starup\_stm32f103\_md.s** có sẵn trong thư mục Startup

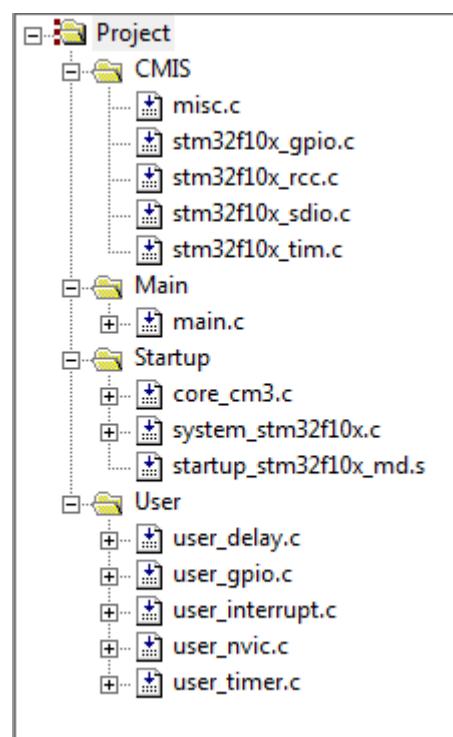


- Group **User** : add các file trong mục **User** vừa tạo ở trên nếu lập trình cho KIT STM32F103C8T6



- Tương tự với Group Main

Nhấn **OK** để hoàn thành. Project của chúng ta đãy đủ như hình dưới:

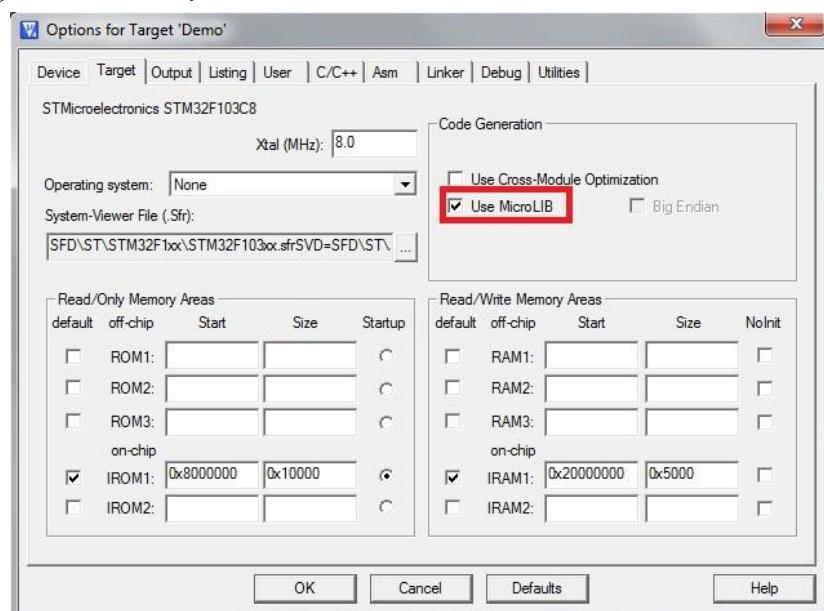


- Tiếp theo là cấu hình cho Project

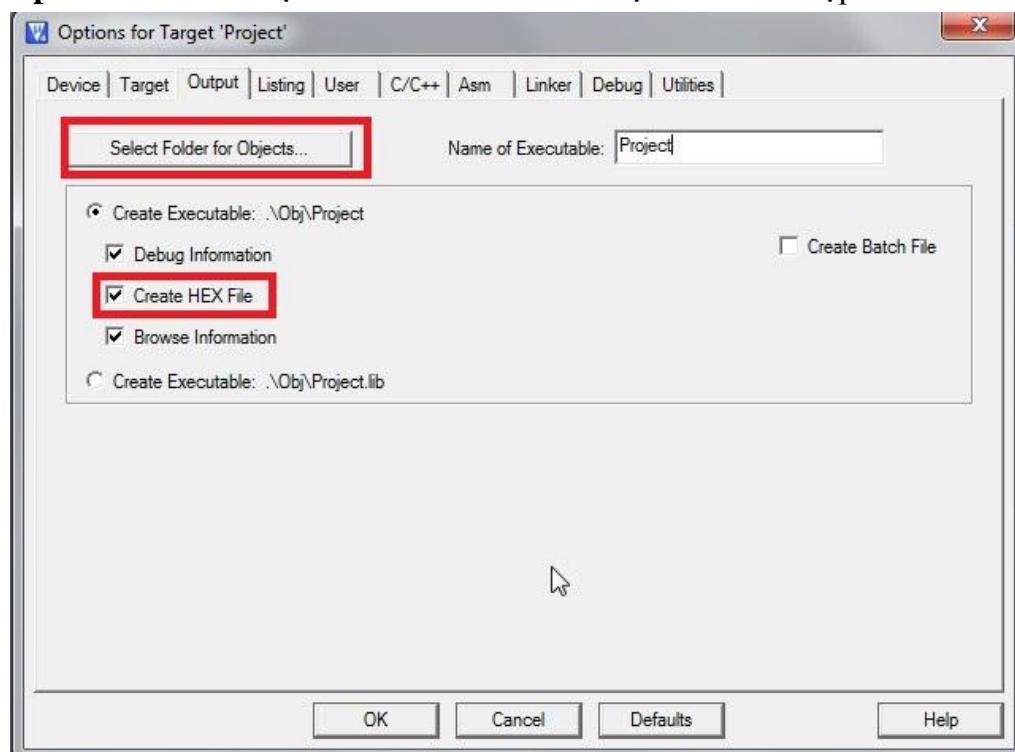
1. Chọn Target Options để cấu hình



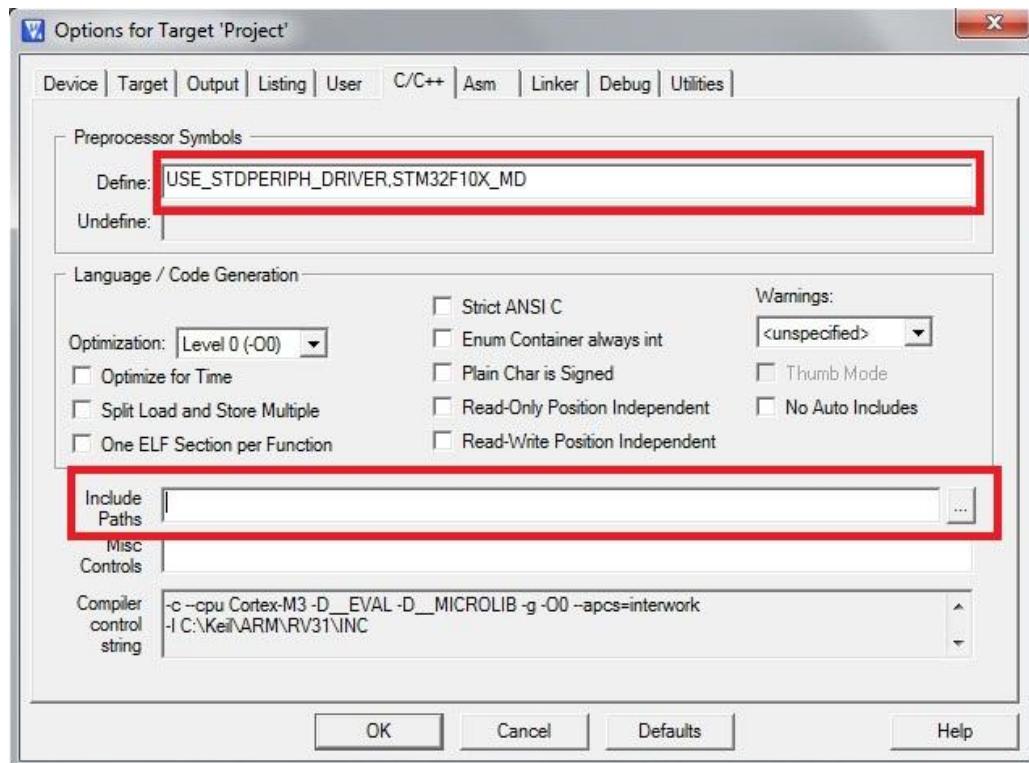
Ở tab target , đánh dấu chọn Use MicroLIB



Tab Output : đánh dấu chọn Create HEX File để tạo file HEX nạp cho VDK



Tab **C/C++:** tại dòng **Deline** gõ vào : **USE\_STDPERIPH\_DRIVER, STM32F10X\_MD**



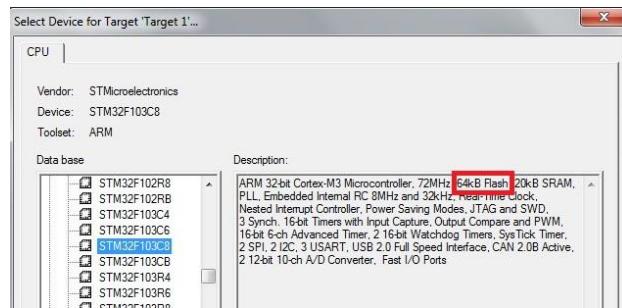
**USE\_STDPERIPH\_DRIVER :** Nằm trong stm32f10x.h, khai báo sử dụng thư viện bên ngoài

Chúng ta mở file stm32f10x.h lên và xem phần sau :

```
/* #define STM32F10X_LD */ /*!< STM32F10X_LD: STM32 Low density devices */
/* #define STM32F10X_LD_VL */ /*!< STM32F10X_LD_VL: STM32 Low density Value Line
devices */
/* #define STM32F10X_MD */ /*!< STM32F10X_MD: STM32 Medium density devices */
/* #define STM32F10X_MD_VL */ /*!< STM32F10X_MD_VL: STM32 Medium density Value Line
devices */
/* #define STM32F10X_HD */ /*!< STM32F10X_HD: STM32 High density devices */
/* #define STM32F10X_HD_VL */ /*!< STM32F10X_HD_VL: STM32 High density value line
devices */
/* #define STM32F10X_XL */ /*!< STM32F10X_XL: STM32 XL-density devices */
/* #define STM32F10X_CL */ /*!< STM32F10X_CL: STM32 Connectivity line devices */
```

Trên là hướng dẫn chọn **define**, **file startup** cho chương trình. Tùy theo chip tương ứng mà chúng ta cần khai báo cho đúng

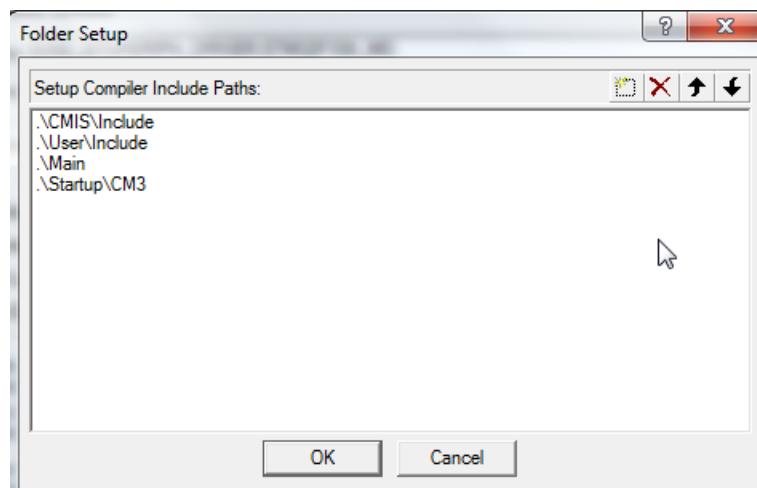
Muốn biết chip đang dùng thuộc loại nào thì khi khởi tạo project, lúc chọn chip có hiển thị thông tin chip, chúng ta xem **Flash** bao nhiêu để chọn **define** cho đúng



Ở đây chọn chip **STM32F103C8**, có 64kB Flash là chip **Medium-Density** nên để build được và nạp code cho VDK chạy phải chọn là **STM32F10X\_MD**

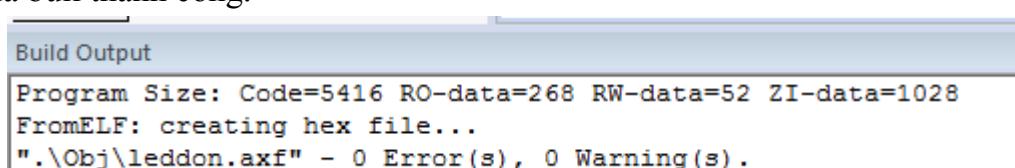
**STM32F10X\_MD** : Flash Memory

Nhập vào dòng **Include Paths** để cài đặt thư mục **Folder Setup** cho Project, ở bên dưới ô vuông đó là những thư chúng ta phải add vào. Mục đích là khai báo cho trình biên dịch biết được thư viện nằm ở đâu



**OK**, Nhấn **F7** để biên dịch chương trình

Kết quả build thành công:



### 3.1.2 Nạp chương trình vào vi điều khiển

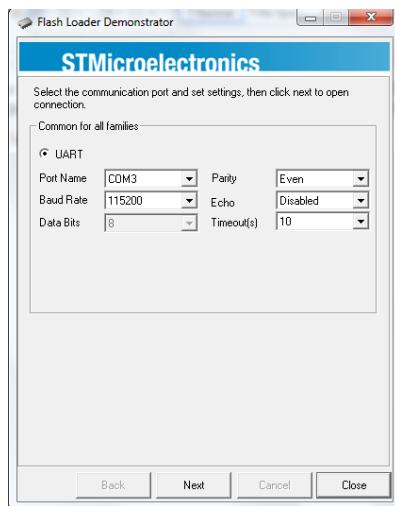
Việc nạp code cho **STM32** có nhiều cách

- **JTAG**: nạp và gỡ rối, việc dùng JTAG được thực hiện trên KeilC nên rất thuận tiện cho việc nạp code, debug , test sản phẩm,... nhược điểm là phần cứng rườm rà.
- **SWD** : chuẩn giao tiếp 2 dây, nhỏ gọn đơn giản và chi phí thấp hơn so với JTAG
- **Bootloader** : phần cứng đơn giản, dễ thực hiện,... nhưng chỉ dùng cho việc nạp code

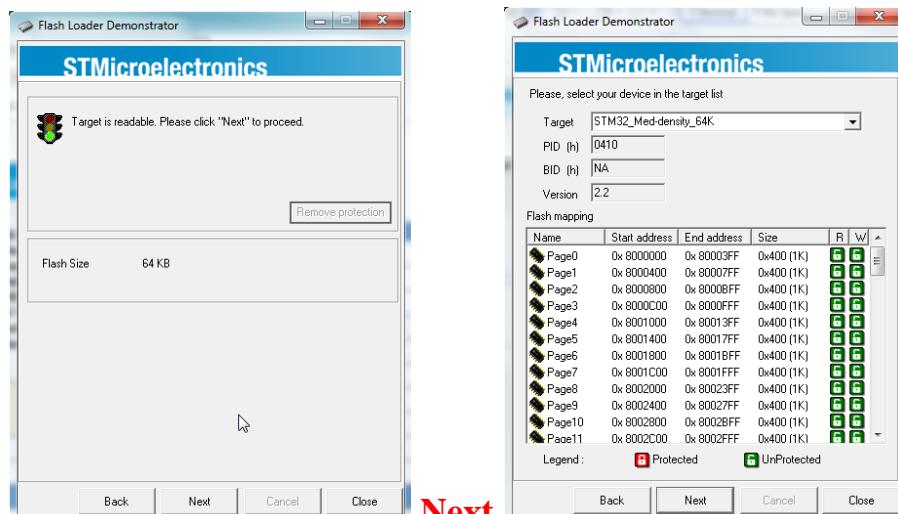
Ở đây, trên KIT STM32F103C8T6 hỗ trợ **JTAG** và **Bootloader** để đơn giản thì chúng ta sẽ nạp file HEX đã build ở trên vào VDK thông qua **Bootloader**

Để vào được chế độ **bootloader** cần cài đặt cho chân **BOOT0 =1** và chân **BOOT1=0**. Để chip tiếp tục chạy từ bộ nhớ Flash thì **BOOT0=0, BOOT1=1**

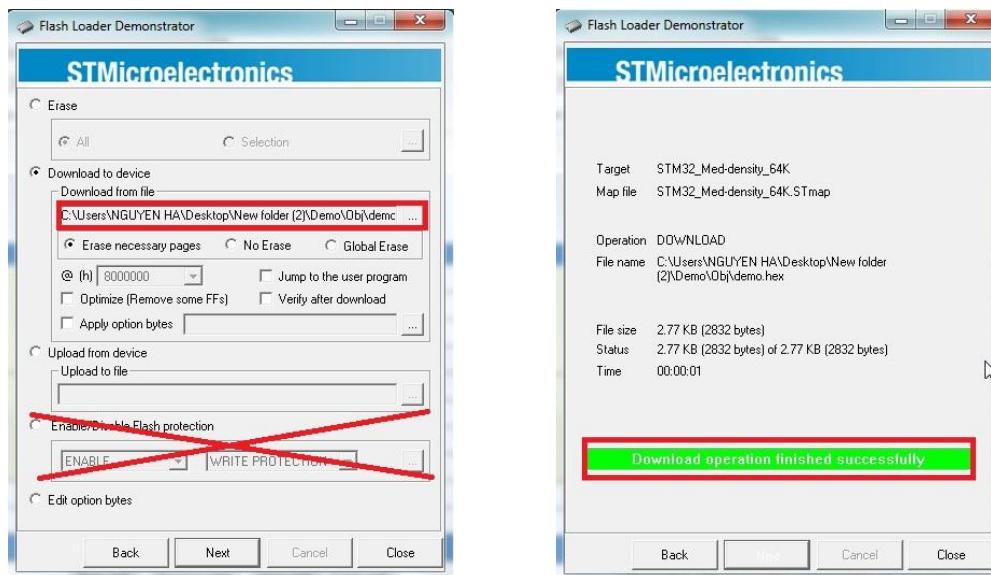
Khởi động **Flash Loader**, nếu có kết nối với KIT thì sẽ như hình:



Nhấn **Next** để tiếp tục, nếu không ở bootloader thì sẽ báo lỗi, còn không thì sẽ như hình:



Tiếp theo, tại **Download to device**, dẫn đến file HEX ở trên. Tuyệt đối không chọn vào **Enable/Divsble Flash protecion** nếu như không muốn khóa chip để bảo mật chương trình. Nhấn **Next** để thực hiện nạp code vào chip.



**Tắt bootloader và cài đặt lại 2 chân BOOT1, BOOT0 để chạy xem**

### 3.2 Lập trình ứng dụng

#### 3.2.1 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.

#### NGUỒN

#### CLOCK

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**).

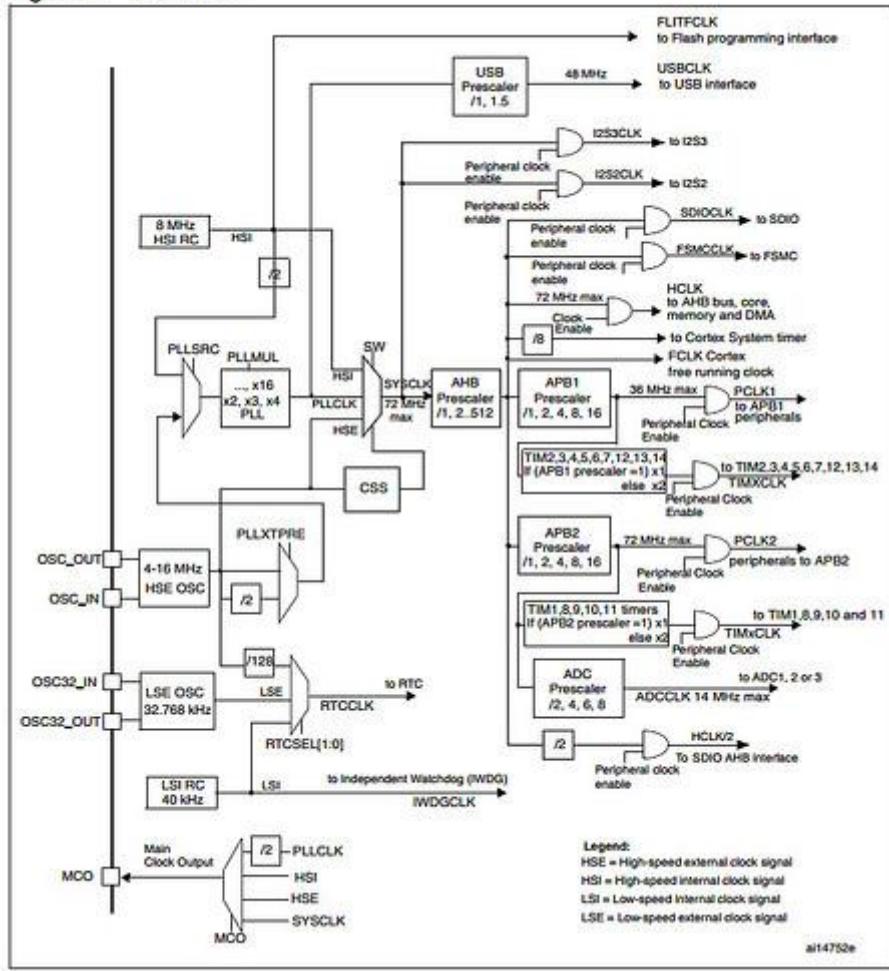
Ngoài ra, STM32 còn được hỗ trợ thêm 2 nguồn Clock phụ:

-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).

Xem chi tiết sơ đồ Clock sau (cho các dòng chíp):

Figure 8. Clock tree



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 .

### QUÁ TRÌNH THIẾT LẬP CLOCK CỦA HỆ THỐNG

Để cấu hình STM32 sử dụng ở tần số Clock cao nhất (72MHz) ta thực hiện theo các bước sau:

Cho phép bộ dao động **HSE** hoạt động.

Chờ cho bộ dao động **HSE** đi vào hoạt động ổn định (dùng làm đầu vào cho bộ **PLL**). Với tần số thạch anh **8MHz**, cấu hình bội số nhân cho bộ **PLL** là 9 để tạo ra tần số Clock ở đầu ra bộ **PLL = 72MHz**.

Cho phép bộ **PLL** hoạt động.

Chờ cho bộ **PLL** đi vào hoạt động ổn định.

Khi đã hoạt động ổn định, cấu hình xung Clock từ bộ **PLL** sử dụng để cung cấp cho hệ thống ànhệ thống hoạt động với tần số Clock max = 72MHz.

Tất cả được thiết lập trên thanh ghi **RCC (Reset – Clock Control)** ( tham khảo thêm Datasheet để tìm hiểu thêm về thanh ghi này).

Khi lập trình, ở hàm **main.c** đầu tiên phải sử dụng đến hàm **SystemInit()**. Trong hàm này sử dụng hàm **SetSysClock()**, hàm này có tác dụng khởi tạo Clock cho hệ thống.

Trong hàm **SetSysClock()**, xem đến hàm **SetSysClockTo72()** để thấy rõ các bước thiết lập ở trên.

### 3.2.2 Tạo thư viện delay sử dụng SYSTICK trong STM32

Trong lập trình vi điều khiển, tạo **Delay** là vấn đề chung và quan trọng khi tìm hiểu trên bất kỳ dòng vi điều khiển nào.Tùy vào từng dòng mà trình biên dịch sẽ hỗ trợ sẵn các hàm delay chuẩn. Tuy nhiên, khi lập trình với STM32 ta sẽ phải tự tạo hàm delay, trong nội dung bài viết này sẽ hướng dẫn tạo 2 hàm: **delay\_ms()**, **delay\_us()**.

Lõi ARM CortexM3cung cấp một bộ đếm thời gian nhằm tạo ra thời gian chuẩn cho tất cả các dòng vi điều khiển sử dụng Cortex: **SYSTICK**.

#### ĐẶC ĐIỂM CỦA SYSTICK

-Là một bộ đếm xuống 24 Bit.

-Giá trị đếm được tự động nạp lại khi bộ đếm đếm về 0.

-Có thể tạo ra 1 sự kiện ngắt khi bộ đếm đếm về 0.

-Nguồn Clock có thể chọn từ:

Nguồn Clock của hệ thống (AHB).

Nguồn Clock của hệ thống /8 (AHB/8).

Systick sẽ đếm từ giá trị nạp lại (được khởi tạo sẵn) về 0 và giá trị này tự động được nạp lại trong chu kỳ đếm tiếp theo.

#### CÁC THANH GHI TRẠNG THÁI VÀ ĐIỀU KHIỂN SYSTICK

##### Thanh ghi trạng thái và điều khiển SysTick (STK\_CTRL):

The SysTick CTRL register enables the SysTick features.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	COUNT FLAG rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved															CLKSOURCE rw   TICK INT rw   EN ABLE rw	
Reserved																

Trong đó (khi Reset, giá trị **STK\_CTRL** = **0x0000 0000**):

-Bit **COUNT FLAG**: Được set lên 1 khi bộ đếm đếm về 0.

-Bit **CLKSOURCE**: Lựa chọn nguồn Clock cho SysTick.

CLKSOURCE = 1: Sử dụng nguồn Clock hệ thống (AHB).

CLKSOURCE = 0: Sử dụng nguồn Clock hệ thống /8 (AHB/8).

### CHƯƠNG III : LẬP TRÌNH ỦNG DỤNG CHO KIT STM32F103

-Bit **TICKINIT**: Cho phép một yêu cầu ngoại lệ xảy ra khi bộ đếm đếm về 0.

TICKINIT = 1: Cho phép.

TICKINIT = 0: Không cho phép.

-Bit **ENABLE**:

ENABLE = 1: Cho phép bộ đếm STK hoạt động.

ENABLE = 0: Không cho phép bộ đếm STK hoạt động.

**Thanh ghi chứa giá trị Reload của SysTick (STK\_LOAD):**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														RELOAD[23:16]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOAD[15:0]														RELOAD[15:0]	
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Khi Reset, giá trị **STK\_LOAD** = **0x0000 0000**.

Để tạo ra một khoảng thời gian N chu kỳ xung nhịp thì cần nạp giá trị vào **STK\_LOAD** giá trị: **N-1**.

**Thanh ghi chứa giá trị đếm hiện tại của SysTick (STK\_VAL):**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														CURRENT[23:16]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CURRENT[15:0]														CURRENT[15:0]	
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Khi Reset, giá trị **STK\_VAL** = **0x0000 0000**.

STK\_VAL sẽ chứa giá trị đếm hiện tại của bộ đếm SysTick. Nếu ghi bất kỳ một giá trị nào vào thanh ghi này thì giá trị đếm của SysTick sẽ được xóa về 0 và COUNTFLAG cũng bị xóa về 0.

4.Thanh ghi chứa giá trị hiệu chỉnh của SysTick (STK\_CALIB):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
NO REF	SKEW	Reserved														TENMS[23:16]
		f	f	TENMS[15:0]												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TENMS[15:0]														TENMS[15:0]		
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	

### TẠO HÀM DELAY SỬ DỤNG SYSTICK

Dựa vào tính chất và các thanh ghi của bộ SysTick, có 3 hàm sử dụng để tạo thời gian Delay sau

-**void delay\_init(u8 SYSCLK)**: Hàm khởi tạo, trong đó **SYSCLK** là Clock của hệ thống.

```
void delay_init(u8 SYSCLK)
{
    SysTick->CTRL&=0xffffffff;
    fac_us=SYSCLK/8;
    fac_ms=(u16)fac_us*1000;
}
```

**-void delay\_ms(u16 nms):** Hàm tạo thời gian Delay ms.

```
void delay_ms(u16 nms)
{
    u32 temp;
    SysTick->LOAD=(u32)nms*fac_ms;
    SysTick->VAL =0x00;
    SysTick->CTRL=0x01 ;
    do
    {
        temp=SysTick->CTRL;
    }
    while(temp&0x01&&!(temp&(1<<16)));
    SysTick->CTRL=0x00;
    SysTick->VAL =0X00;
}
```

**-void delay\_us(u32 nus):** Hàm tạo thời gian Delay us.

```
void delay_us(u32 nus)
{
    u32 temp;
    SysTick->LOAD=nus*fac_us;
    SysTick->VAL=0x00;
    SysTick->CTRL=0x01 ;
    do
    {
        temp=SysTick->CTRL;
    }
    while(temp&0x01&&!(temp&(1<<16)));
    SysTick->CTRL=0x00;
    SysTick->VAL =0X00;
}
```

Như vậy ta có các hàm Delay trong **thư viện user\_delay.c** ở các code chạy trên KIT STM32F103C8T6.

### 3.2.3 Lập trình GPIO điều khiển led đơn và thư viện GPIO

MCU STM32F10x có nhiều loại với số lượng IO khác nhau. Mỗi port IO được cấu hình bởi 2 thanh ghi 32bit (**GPIOx\_CRL&GPIOx\_CRH**)

- **GPIOx\_CRL** : cấu hình các pin từ 0→7

- **GPIO\_CRH** : cấu hình các pin từ 8→15

Có 8 chế độ IO có thể lập trình cho từng pin

- Input floating

- Input pull-up

- Input pull-down

- Analog input

- Output open-drain

- Output push-pull

- Alternate function push-pull

- Alternate function open-drain

Các bit mode[1:0] cấu hình chế độ input hoặc output

#### Mode info

00 input( mặc định khi reset)

01 output max 10MHz

10 output Max 2Mhz

11 output Max 50 MHz

Các bit CNF[1:0] có ý nghĩa phụ thuộc vào trạng thái pin là input hay output

#### Input Mode :

CNF[1:0] info

00 analog input

01 floating input(digital)

10 input với pullup/pulldown.

11 reserver

#### Output Mode :

CNF[1:0] info

00 output push/pull

01 output open drain

10 alternate output push/pull

11 alternate output open drain

**Chú ý :** chế độ input pullup/pulldown sẽ do giá trị bit tương ứng trên thanh ghi ODR quyết định. Nếu sử dụng hàm chuẩn trong thư viện của ST thì có thể không cần biết cũng làm được.Nhưng tốt hơn nên biết xem hàm dưới đây nó tác động

vào thanh ghi nào để lúc cần thì có thể gán trực tiếp cho nhanh Các pin IO đều có dạng 5V tolerant (ngoài 2 pin chung chức năng với thạch anh đồng hồ thời gian thực) tức là có thể nối với các thiết bị dùng chuẩn 5V. Thường thì nối thêm con trỏ nhỏ nối tiếp với chân IO nếu nó là chế độ input (để phân điện áp dư rọi trên đó tránh gây hỏng pin IO).

Sơ đồ các pin các có thể tham khảo trong datasheet.Các thanh ghi quan trọng.

- Input data register **GPIOx\_IDR**
- Output data register **GPIOx\_ODR**
- Bit Set/Reset register **GPIOx\_BSRR**
- Bit Reset register **GPIOx\_BRR**
- lock mechanism register **GPIOx\_LCKR**

Ngoài ra còn có thanh ghi remap các chân vào ra của ngoại vi. Có thể xem trong datasheet để hiểu rõ hơn.

Trong thư viện “*stm32f10x\_gpio.h*” các pin tương ứng đã được định nghĩa sẵn để người dùng dễ sử dụng : **GPIO\_Pin\_x**

Các port được định nghĩa bằng tên **GPIOx** trong đó x: A,B,C,...G. Thực chất **GPIOx** có dạng con trỏ trỏ tới địa gốc của port tương ứng.

Lệnh dùng khi **Set Bit x** của **port y** : **GPIOx→BSRR = GPIO\_Pin\_y**

Lệnh dùng khi **Reset bit x** của **Port y** : **GPIOx→BRR = GPIO\_Pin\_y**

Hoặc dùng lệnh :**GPIOx→BSRR = GPIO\_Pin\_y <<16**

Thư viện chuẩn của ST, để bật tắt các bit, ta sử dụng hàm **GPIO\_SetBit()** và **GPIO\_ReSetBit()**.

Và để dễ cho người dùng thì các hàm này được viết trong thư viện **user\_gpio.c**

Ví dụ : ta cấu hình GPIO sử dụng USART1:

- Enable clock GPIOA :

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
```

- Enable clock AFIO :

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
```

Cấu hình PIN TX :

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

### CHƯƠNG III : LẬP TRÌNH ỦNG DỤNG CHO KIT STM32F103

Cấu hình PIN RX :

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Cấu hình GPIO PINB.5 là cổng ra

Enable clock GPIOB :

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
```

Cấu hình cổng ra :

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

File user\_gpio.c như sau:

```
#include "user_gpio.h"
void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    //Enable clock AFIO
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    GPIO.PinRemapConfig(GPIO_Remap_SWJ_Disable, ENABLE);
    /*=====PORTA=====*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    // _____ OUTPUT _____
    GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO
    _Pin_7|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

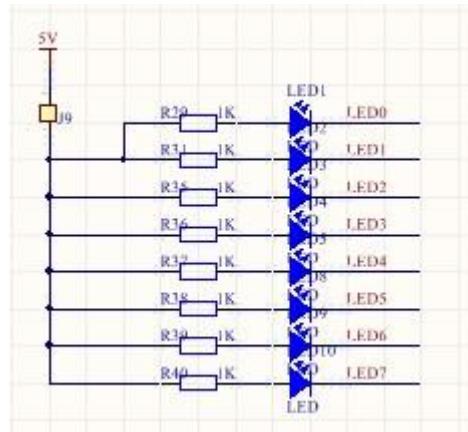
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9; //USART1 TX
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    // _____ INPUT _____
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10|GPIO_Pin_8; //USART1 RX
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    // _____ INPUT _____
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_14|GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /*=====PORTB=====*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
```

### CHƯƠNG III : LẬP TRÌNH ỦNG DỤNG CHO KIT STM32F103

```
// _____OUTPUT_____
GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_
_Pin_7|GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_
Pin_14|GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

Đến đây thì việc lập trình GPIO điều khiển led đơn khá dễ dàng:

Trên KIT STM32F103C8T6 có hỗ trợ khối Led đơn gồm 8 led măc Anode chung có sơ đồ :



Hình 3.1 Khối hiển thị led đơn

Jumper J9 để cấp nguồn cho Led

LED Tắt = 1

LED sáng = 0

Các chân **LED[0-7]** được kết nối theo thứ tự với **PORTB[0-7]** của STM32F103C8T6

#### - Lập trình phần mềm

Khởi tạo một Project mới, Trong Project này add thêm các file .c sau :

Main: main.c

Starup : core\_cm3.c, Startup\_stm32f10x\_md.s, system\_stm32f10x.c

User: user\_delay.c. user\_gpio.c

Cmis: misc.c, stm32f10x\_gpio.c

Trong hàm main. C cần sử dụng:

+ hàm **SystemInt()** : hàm khởi tạo nguồn Clock cho vi điều khiển

+ hàm **GPIO\_Configuration()**: hàm khởi tạo và cấu hình chức năng GPIO . Ở phần này, cần cấu hình các chân PB0-PB7 là chân Output Push Pull, tốc độ trung bình 50MHz

Xem file **user\_gpio.c**

Trong hàm này cần chú ý :

+ Do cấu tạo phần cứng khối led có sử dụng 2 chân **PB3, PB4** là chân JTAG, nên muốn sử dụng 2 chân này với chức năng I/O thông thường thì cần vô hiệu hóa chức năng JTAG bằng lệnh:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
GPIO_PinRemapConfig(GPIO_Remap_SWJ_Disable, ENABLE);
```

- + Cần phải cung cấp Clock cho Port chứa các chân I/O sử dụng. Cụ thể là PORTB:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
```

Cấu hình chức năng I/O để điều khiển led :

```
GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

- + hàm delay\_int(72): Đây là hàm khởi tạo để sử dụng các hàm delay\_ms(), delay\_us() (nằm trong thư viện user\_delay.c). 72 là tần số nguồn clock sử dụng

Sau khi khai báo các hàm ở trên ta có thể bắt đầu viết những hiệu ứng cho khôi led đơn:

```
while(1)
{
    TempPort=0x0001; PORTB|=0x00ff;for(i=0;i<8;i++)
    {
        PORTB&=~TempPort; delay_ms(200);
        PORTB|=TempPort; delay_ms(200); TempPort<<=1;
    }
    TempPort=0x0080; PORTB|=0x00ff; for(i=0;i<8;i++)
    {
        PORTB&=~TempPort; delay_ms(200);
        PORTB|=TempPort; delay_ms(200); TempPort>>=1;
    }
    Temp1=0x08; Temp2=0x10; for(i=0;i<4;i++)
    {
        PORTB=~(Temp1|Temp2); delay_ms(200); Temp1>>=1; Temp2<<=1;
    }
    Temp1=0x01; Temp2=0x80; for(i=0;i<4;i++)
    {
        PORTB=~(Temp1|Temp2); delay_ms(200); Temp1<<=1; Temp2>>=1;
    }
    Temp1=0x08; Temp2=0x10; for(i=0;i<4;i++)
    {
        Temp|=Temp1|Temp2; delay_ms(200);
        Temp1>>=1; Temp2<<=1; PORTB=~Temp;
    }
    Temp1=0x01;
    Temp2=0x80;
    Temp=0x00;
```

```

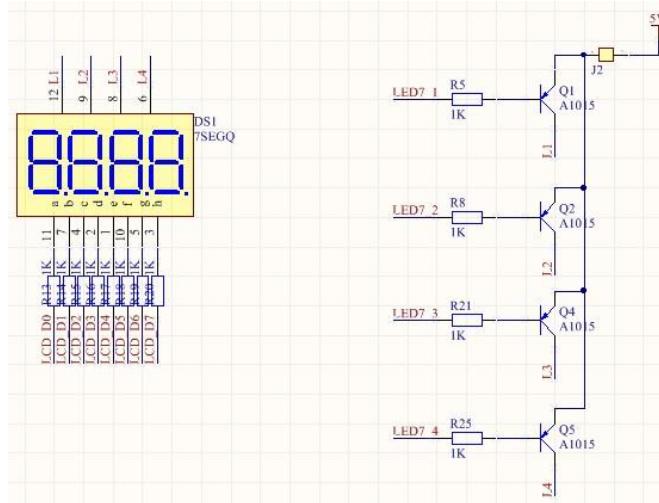
for(i=0;i<4;i++)
{
    Temp|=(Temp1|Temp2); delay_ms(200);
    Temp1<<=1; Temp2>>=1; PORTB=~Temp;
}
delay_ms(500);
}

```

Biên dịch chương trình và nạp chạy trên KIT STM32F103C8T6

### 3.2.5 Lập trình hiển thị Led 7 đoạn

Sơ đồ phần cứng trên KIT STM32 có khối Led 7 thanh 4 số Anode chung có sơ đồ như sau:



Hình 3.2 Khối hiển thị LED 7 đoạn

Trong đó, các chân kết nối với STM32F103C8T:

Các chân điều khiển cấp nguồn **LED7\_[1 – 4]** kết nối lần lượt với **PORTB[12 – 15]**.

Các chân dữ liệu **A, B, C, D, E, F, G, Dp** kết nối lần lượt với **PORTA[0 – 7]**.

Ta có bảng mã hiển thị các số từ 0 – 9 trên Led 7 thanh:

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

Hình 3.3 Mã hiển thị Led 7 đoạn

## Lập trình phần mềm

### Phương pháp quét Led 7 thanh

Kit STM32 START hỗ trợ phần cứng Led 7 thanh 4 số Anode chung vì vậy cần sử dụng phương pháp quét Led để điều khiển hiển thị.

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 thì giá trị hiển thị trên Led giống như luôn sáng.

Để hiển thị lên màn hình Led 7 thanh 4 số, ta lần lượt cấp nguồn cho từng con Led và bắn dữ liệu vào các chân data.

Gọi thời gian giữa 2 lần cấp nguồn là **T**.

Vậy khoảng thời gian 1 con Led sáng – tắt là **4T < 25ms à T < 6ms**.

Sau khi khởi tạo thành công Project, chúng ta thêm các File.c cần thiết vào các Folder, cụ thể

- CMIS:** stm32f10x\_gpio.c, stm32f10x\_rcc.c, stm32f10x\_tim.c, misc.c
- Main:** main.c
- User:** user\_delay.c, user\_gpio.c
- StartUp:** core\_cm3.c, startup\_stm32f10x\_md.s, system\_stm32f10x.c

Trong hàm **main.c** cần sử dụng:

-Hàm **SystemInit()**:

-Hàm **GPIO\_Configuration()**

các chân **PA0 – PA7** là chân **Output Push Pull**, tốc độ trung bình **50MHz**.

các chân **PB12 – PB15** là chân **Output Push Pull**, tốc độ trung bình **50MHz**

-Hàm **delay\_init(72)**

Viết các chương trình con phục vụ quét led và hiển thị:

```

void LED7_InBuffer(uint8_t Point)
{
    BufferLed7[3] = CHU_SO[(Point++)%10]; BufferLed7[2] = CHU_SO[(Point++)%10];
    BufferLed7[1] = CHU_SO[(Point++)%10]; BufferLed7[0] = CHU_SO[(Point++)%10];
}
void LED7_CacuCode(uint16_t Number)
{
int8_t i;
for(i=0;i<4;i++)
{
    BufferLed7[i]=CHU_SO[Number%10]; Number=Number/10;
}
for(i=3;i>=0;i--)
{
    if(BufferLed7[i]!=CHU_SO[0])
break;
    BufferLed7[i]=0xff;
}
}
void Led7OnOneLed(uint8_t Point)
{
    uint16_t TempPoint=0x8000; TempPoint>>=Point;
}

```

```

LED7_ADDR_PORT/=0xf000; LED7_DATA_PORT/=0x00ff;
LED7_ADDR_PORT&=(~TempPoint);
LED7_DATA_PORT&=(0xff00/BufferLed7[Point]);
}
void LED7_Display(uint16_t Value)
{
    int8_t i;
    for(i=0;i<4;i++)
    {
        Led7OnOneLed(i);
        delay_ms(1);
    }
}

```

Trong chương trình chính :

```

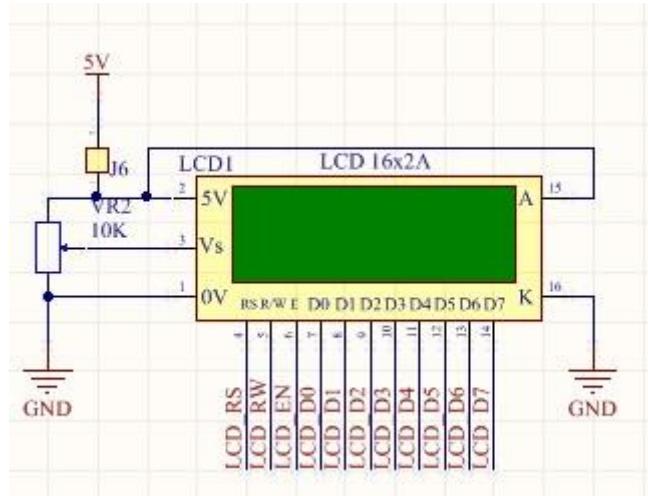
while(1)
{
    for(i=0;i<10;i++)
    {
        LED7_InBuffer(i);
        for(j=0;j<100;j++)
        {
            LED7_Display(0);
        }
    }
}

```

Biên dịch chương trình và nạp chạy demo trên KIT STM32

### 3.2.6 Lập trình hiển thị LCD 1602

#### Sơ đồ phần cứng trên KIT



Hình 3.4 Khối hiển thị LCD16x2

Trong đó, các chân kết nối với STM32F103C8T

- Các chân điều khiển LCD16x2 **LCD\_RS**, **LCD\_RW**, **LCD\_EN** kết nối thứ tự với các chân **PORTB[12 – 14]**.

- Các chân dữ liệu của LCD16x2 **LCD\_D[4 – 7]** kết nối với các chân **PORTA[4 – 7]**.

#### Lập trình phần mềm:

Sau khi khởi tạo thành công Project, chúng ta thêm các File.c cần thiết vào :

- CMIS:** stm32f10x\_gpio.c, stm32f10x\_rcc.c, stm32f10x\_tim.c, misc.c
- Main:** main.c
- User:** user\_delay.c, user\_gpio.c, lcd16x2.c
- StartUp:** core\_cm3.c, startup\_stm32f10x\_md.s, system\_stm32f10x.c

Các hàm cần thiết trong main.c

-Hàm **SystemInit()**:

-Hàm **GPIO\_Configuration()**

các chân **PA4 – PA7** là chân **Output Push Pull**, tốc độ trung bình **50MHz**.

các chân **PB12 – PB14** là chân **Output Push Pull**, tốc độ trung bình **50MHz**

-Hàm **delay\_init(72)**

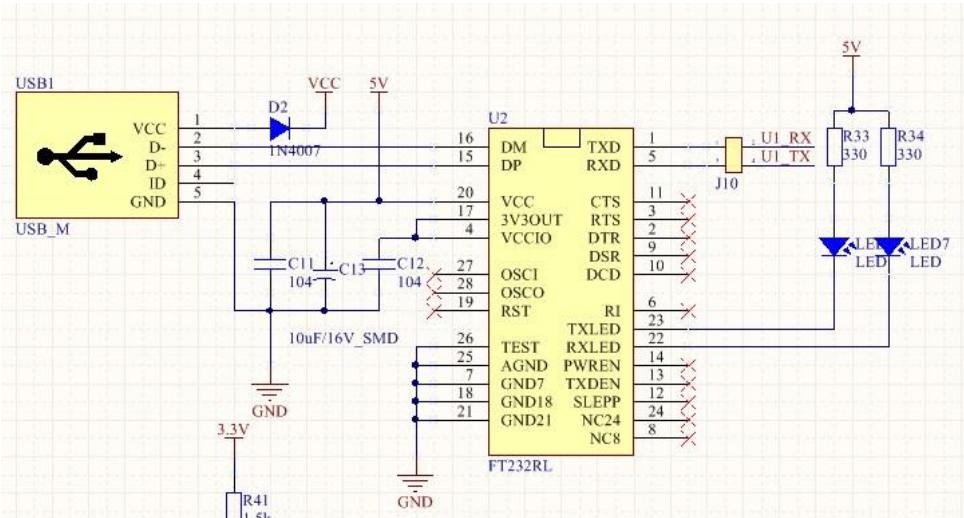
- **LCDInt()** : Khởi tạo LCD1602 và lập trình hiển thị chuỗi ký tự

```
LCD_Puts("wWw.HocARM.net");
LCD_Gotoxy(0,1); LCD_Puts("STM32F103C8T6");
```

### 3.2.7 Giao tiếp USART với KIT STM32F103C8T6

STM32 hỗ trợ tới 3 giao tiếp USART, trên KIT có thiết kế sẵn giao tiếp UART 1 sử dụng chip chuyên dụng FT232RL

**Sơ đồ phần cứng :**



Hình 3.5 Khối giao tiếp USART

Trong đó :

**U1\_TX -> PORTA9, U1\_RX -> PORTA10**

**Lập trình phần mềm**

Đối với bài toán giao tiếp USART chúng ta sẽ thực hiện hai công việc sau:

-Truyền một chuỗi ký tự từ KIT STM32 START lên PC.

-Truyền các ký tự từ PC xuống KIT STM32 START

Sau khi khởi tạo thành công Project, thêm các File.c cần thiết vào các Folder, cụ thể:

- CMIS:** stm32f10x\_gpio.c, stm32f10x\_rcc.c, stm32f10x\_tim.c, misc.c, stm32f10x\_usart.c
- Main:** main.c
- User:** user\_delay.c, user\_gpio.c, lcd16x2.c, user\_usart.c
- StartUp:** core\_cm3.c, startup\_stm32f10x\_md.s, system\_stm32f10x.c

Trong hàm Main.c :

-Hàm **SystemInit()**:

-Hàm **GPIO\_Configuration()**

Cấu hình các chân PA9, PA10 là USART

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9; // TX - USART1
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10; // RX - USART1
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

-Hàm **USARTx\_Configuration(USART1)**: Đây là hàm khởi tạo cho USART1(nằm trong File user\_usart), cách cấu hình các thông số cho USART1 trong hàm này:

```
void USARTx_Configuration(USART_TypeDef* USARTx)
{
    USART_InitTypeDef USART_InitStructure;
    if(USARTx==USART1)
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    else if(USARTx==USART2)
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    else if(USARTx==USART3)
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE); USART_DeInit(USARTx);
    USART_InitStructure.USART_BaudRate = 9600; // Cau hinh BaudRate
    USART_InitStructure.USART_WordLength = USART_WordLength_8b; // Cau hinh so Bit du lieu
    USART_InitStructure.USART_StopBits = USART_StopBits_1; // Cau hinh so Bit STOP trong khung truyen
    USART_InitStructure.USART_Parity = USART_Parity_No; // Cau hinh su dung che do Parity
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None; // Cau hinh che do dieu khien theo luong
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; // Cau hinh che do truyen nhan
    USART_Init(USARTx, &USART_InitStructure); // Cau hinh USART1
    USART_ITConfig(USARTx, USART_IT_RXNE, ENABLE); USART_ITConfig(USARTx,
    USART_IT_TXE, ENABLE); USART_Cmd(USARTx, ENABLE); // Kich hoat USART1
}
```

- Hàm **delay\_init(72)**:
  - Sử dụng các hàm trong File **user\_usart.c** để giao tiếp Usart:
- Hàm truyền 1 chuỗi ký tự lên PC:

```
USART_SendString(USART1,str);
```

Hàm nhận 1 ký tự từ PC

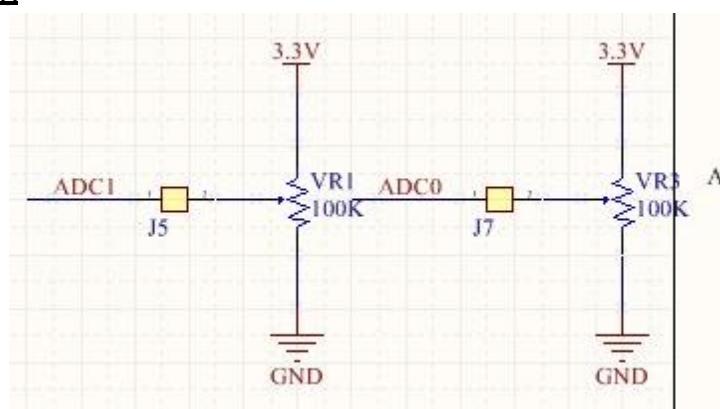
```
USART_ReceiveData(USART1);
```

Chương trình chính gửi một chuỗi ký tự lên PC và nhận ký tự từ PC gửi xuống để điều khiển LED đơn

```
#include "main.h"
#include "var.h"
int main(void)
{
char str[50],i;
SystemInit();
NVIC_Configuration();
GPIO_Configuration();
TIMER1_Configuration();
USART1_Configuration();
delay_init(72);
USART_SendString(USART1," wWw.HocARM.net - USART1 STM32 START\r\n");
sprintf(str,"%\rDEMO USART%\r");
USART_SendString(USART1,str);
while(1)
{
    unsigned char c; c=USART_ReceiveData(USART1);
    if(c=='1'){ PORTB=0x00;} if(c=='2'){ PORTB=0xFF;}
}
}
```

### 3.2.8 Đo giá trị ADC và hiển thị LCD

#### Sơ đồ phần cứng



Hình 3.6 Khối giao tiếp ADC

Trên KIT STM32F103 có hỗ trợ 2 chân ADC đọc giá trị từ biến trở

Trong đó, chân đầu vào các bộ **ADC1**, **ADC2** kết nối lần lượt với các chân **PORATA0**, **PORATA1** của STM32F103C8T6

#### Lập trình phần mềm

Khởi tạo một Project mới, thêm các file cần thiết vào chương trình

- CMIS:** stm32f10x\_gpio.c, stm32f10x\_rcc.c, stm32f10x\_tim.c, misc.c, stm32f10x\_adc.c
- Main:** main.c
- User:** user\_delay.c, user\_gpio.c, lcd16x2.c, user\_adc.c
- StartUp:** core\_cm3.c, startup\_stm32f10x\_md.s, system\_stm32f10x.c

Khai báo trong hàm main.c

-Hàm **SystemInit()**

-Hàm **GPIO\_Configuration()**

Các chân dữ liệu **LCD16x2 PA[4-7]** là chân Output Push Pull tốc độ xung 50MHz

Các chân điều khiển **LCD16x2 PB[12-14]** là chân Output Push Pull tốc độ xung

50MHz

Chân đầu vào **ADC0, PORTA0** là chân **Analog Input**, tốc độ trung bình 50MHz

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

- Hàm **ADC1\_Configuration()**: Đây là hàm khởi tạo bộ ADC1, cấu hình tham số cho bộ ADC1 trong hàm này

```
#include "user_adc.h"
void ADC1_Configuration(void)
{
    ADC_InitTypeDef ADC_InitStructure; //Enable Clock
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; // Chon Che Do ADC
    ADC_InitStructure.ADC_ScanConvMode = DISABLE; //Enable - Disable che do Scan
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; //Enable - Disable che do lien tuc
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //Dinh nghia Trigger ben ngoai de bat dau 1 chuyen doi ADC kenh Regular
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//ADC_DataAlign_Left;
    //Chon Kieu Luu Tru Du Lieu
    ADC_InitStructure.ADC_NbrOfChannel =1; //Chon so luong kenh regular ADC su dung
    ADC_ExternalTrigConvCmd(ADC1, ENABLE); // Enable kich thich chuyen doi tu ben ngoai
    ADC_Init(ADC1, &ADC_InitStructure); // Cau hinh ADC
    ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);
    //++++++ ADC1 WatchDog configuration+++++++
    ADC_AnalogWatchdogCmd(ADC1, ADC_AnalogWatchdog_SingleRegEnable);
    ADC_TempSensorVrefintCmd(ENABLE); //Enable ADC Temsensor - Vref
    ADC_DMACmd(ADC1, ENABLE); //Enable DMA
    ADC_Cmd(ADC1, ENABLE); //Kich Hoat ADC1
    ADC_ResetCalibration(ADC1); //RESET Hieu Chuan ADC1
    ADC_StartCalibration(ADC1); //Bat Dau hieu chuan ADC1 }
    uint16_t read_adc(ADC_TypeDef* ADCx,u8 ADC_Channel)
    {
        uint16_t data;
        assert_param(IS_ADC_CHANNEL(ADC_Channel));
    }
}
```

```

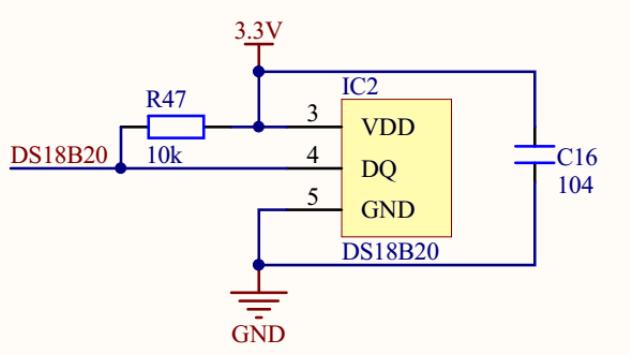
ADC_RegularChannelConfig(ADCx, ADC_Channel, 1, ADC_SampleTime_55Cycles5);
ADC_ResetCalibration(ADCx);
while(ADC_GetResetCalibrationStatus(ADCx)); ADC_StartCalibration(ADCx);
while(ADC_GetCalibrationStatus(ADCx));ADC_SoftwareStartConvCmd(ADCx, ENABLE);
while(!(ADC_GetFlagStatus(ADCx, ADC_FLAG_EOC)));ADC_ClearFlag(ADCx,
ADC_FLAG_EOC); ADC_SoftwareStartConvCmd(ADCx, DISABLE);
data=ADC_GetConversionValue(ADCx);
return data;
}

```

- Hàm **delay\_init(72)**
- Trong chương trình chính dùng hàm **read\_adc(ADC1,0)** để đọc giá trị ADC1

### 3.2.9 Giao tiếp cảm biến nhiệt độ DS18B20

#### Sơ đồ phần cứng



Hình 3.7 Khối giao tiếp cảm biến nhiệt độ DS18B20

Chân dữ liệu của DS18B20 được nối với PORTB\_9 của STM32F103C8T6

Đặc điểm của DS18B20

- Sử dụng giao tiếp chuẩn 1wire
- Nhiều cảm biến có thể kết nối vào cùng một dây dữ liệu
- Không hạn chế số lượng cảm biến
- Có thể cấp nguồn bằng đường dữ liệu, điện áp nguồn nuôi trong khoảng 3.0V – 5.5V
- Dải nhiệt độ đo được -55<sup>0</sup>C – 125<sup>0</sup>C
- Có thể cài đặt độ phân giải khi đo bằng phần mềm : 9bit, 10bit, 11bit, 12bit tương ứng với độ chính xác 0.5, 0.25, 0.125, 0.0625
- Có cờ báo khi nhiệt độ đo được nằm ngoài khoảng giá trị nhiệt độ cài đặt
- Mỗi cảm biến có mỗi mã định danh 64bit duy nhất chứa trong bộ nhớ ROM

#### Lập trình phần mềm:

Khởi tạo một Project mới, thêm các file cần thiết vào chương trình

<b>-CMIS:</b> stm32f10x_gpio.c, stm32f10x_rcc.c, misc.c <b>-Main:</b> main.c <b>-User:</b> user_delay.c, user_gpio.c, lcd16x2.c, ds18b20.c, onewire.c <b>-StartUp:</b> core_cm3.c, startup_stm32f10x_md.s, system_stm32f10x.c
--

Khai báo trong hàm main.c

-Hàm **SystemInit()**

-Hàm **GPIO\_Configuration()**

Khởi tạo LCD như các bài trước, ngoài ra còn có :

```
void GPIO_SetState(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin,GPIO_Mode_TypeDef GPIO_Mode);
```

Hàm này có nhiệm vụ cấu hình Input, output cho chân dữ liệu DS18B20 trong quá trình giao tiếp

Các hàm làm việc với DS18B20 được viết trong thư viện **ds18b20.c**

Các hàm giao tiếp theo chuẩn 1wire được viết trong thư viện **onewire.c**

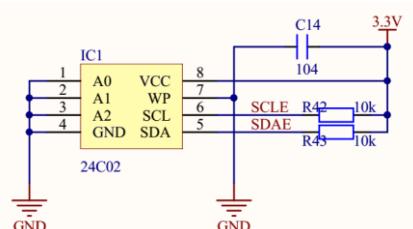
- **Hàm delay\_init(72)**

- **Hàm LCD16x2\_init()**

Sử dụng hàm **ds18b20\_read()** để đọc giá trị nhiệt độ từ DS18B20

### 3.2.10 Giao tiếp I2C với IC EEPROM 24C02

#### Sơ đồ phần cứng



Hình 3.8 Khởi giao tiếp I2C với EEPROM

Trong đó : **SCLE** -> **PORTB\_11**; **SDAE** -> **PORTB\_10** của STM32F103C8T6

Thông số kỹ thuật:

- Điện áp hoạt động : 1.8V – 5.5V
- có 256Byte EEPROM nội
- Giao tiếp chuẩn I2C
- Dữ liệu được lưu lại trong 100 Năm
- Có thể ghi tối đa 1 triệu lần
- Hỗ trợ phần cứng bảo vệ dữ liệu ghi vào

#### Lập trình phần mềm

Về giao tiếp I2C

Khởi tạo I2C

```
void I2C_Init(void)
{
    GPIO_SetState(GPIOB,GPIO_Pin_11,GPIO_Mode_Out_OD);
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_Out_OD);
    SCL=1;delay_us(5);
    SDA_OUT=1;delay_us(5);
}
```

### CHƯƠNG III : LẬP TRÌNH ỨNG DỤNG CHO KIT STM32F103

Lệnh start I2C :

```
void I2C_Start(void)
{
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_Out_OD);
    SDA_OUT=1; SCL=1; delay_us(5);
    SDA_OUT=0; delay_us(5);
    SCL=0; delay_us(5);
}
```

Lệnh stop I2C :

```
void I2C_Stop(void)
{
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_Out_OD);
    SDA_OUT=0; SCL=1; delay_us(5);
    SDA_OUT=1; delay_us(5);
    SCL=0; delay_us(5);
}
```

Lệnh ghi dữ liệu I2C :

```
uint8_t I2C_Write(uint8_t Data)
{
    uint8_t i;
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_Out_OD);
    for(i=0;i<8;i++)
    { if(Data&0x80)      SDA_OUT=1;
      else            SDA_OUT=0; Data<<=1;      delay_us(5);
      SCL=1; delay_us(5); SCL=0; delay_us(5); }
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_IN_FLOATING);
    SCL=1; delay_us(5); i=SDA_IN; delay_us(5); SCL=0; delay_us(5);
    return i;
}
```

Lệnh đọc dữ liệu I2C :

```
uint8_t I2C_Read(uint8_t Ack)
{
    uint8_t I2C_data=0,i;
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_IN_FLOATING);
    for(i=0;i<8;i++)
    {
        SCL=1; delay_us(10);
        I2C_data<<=1;
        if(SDA_IN)
            I2C_data|=1; delay_us(10); SCL=0;
    }
    GPIO_SetState(GPIOB,GPIO_Pin_10,GPIO_Mode_Out_OD);
    SCL=1; delay_us(5);
    SDA_OUT=Ack; delay_us(5);
    SCL=0; return I2C_data;
}
```

Về giao tiếp với EEPROM 24C02

Khởi tạo một Project mới, thêm các file cần thiết vào chương trình

- CMIS:** stm32f10x\_gpio.c, stm32f10x\_rcc.c, misc.c
- Main:** main.c
- User:** user\_delay.c, user\_gpio.c, lcd16x2.c, 24cxx.c, i2c.c
- StartUp:** core\_cm3.c, startup\_stm32f10x\_md.s, system\_stm32f10x.c

Khai báo trong hàm main.c

-Hàm **SystemInit()**

-Hàm **GPIO\_Configuration()**

**GPIO\_SetState(GPIO\_TypeDef\* GPIOx, uint16\_t**

**GPIO\_Pin, GPIO\_Mode\_TypeDef GPIO\_Mode):** Hàm này có nhiệm vụ cấu hình Input, Output cho chân dữ liệu và chân Clock của 24C02 trong quá trình giao tiếp

```
void GPIO_SetState(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_Mode_TypeDef GPIO_Mode)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOx, &GPIO_InitStructure);
}
```

- Hàm **delay\_init(72)**
- Hàm **LCD16x2\_init()**
- Hàm **I2C\_Init()** : hàm giao tiếp khởi tạo chuẩn I2C

Hàm ghi dữ liệu vào EEPROM

```
void EEP_24CXX_Write(uint8_t address, uint8_t Data)
{
    I2C_Start();
    I2C_Write(0xa0); I2C_Write(address); I2C_Write(Data);
    I2C_Stop(); delay_ms(10);
}
void EEP_24CXX_WriteS(uint8_t address, uint8_t *s)
{
    while(*s)
    {
        EEP_24CXX_Write(address++, *s); s++;
    }
}
```

Hàm đọc dữ liệu từ EEPROM

```
void EEPROM_24CXX_ReadS(uint8_t address, uint8_t lenght, uint8_t *s)
{
    uint8_t i=0;
    while(lenght)
    {
        s[i++]=EEP_24CXX_Read(address++); lenght--; delay_ms(2);
        s[i]=0;
    }
}
```

Trong chương trình chính chúng ta dùng lệnh

**EEP\_24CXX\_WriteS(0x01,(uint8\_t\*)"str");** : để ghi chuỗi ký tự và

**EEPROM\_24CXX\_ReadS(0x01,15,(uint8\_t \*)str);** : để đọc dữ liệu

## CHƯƠNG IV : THIẾT KẾ WEB HOCARM.NET

Nhằm phục vụ nhu cầu của người học Vi điều khiển ARM Cortex M3 STM32F103 được dễ dàng và thuận tiện, em đã xây dựng website [www.hocarm.net](http://www.hocarm.net). Trang web sẽ cung cấp kiến thức lý thuyết đầy đủ về ARM STM32 cũng như các bài học lập trình, hướng dẫn chi tiết về con Vi điều khiển này

Website được xây dựng dựa trên mã nguồn mở Nukeviet – một trong những mã nguồn mở khá phổ biến trong cộng đồng web design Việt Nam hiện nay

### 4.1 Giới thiệu về mã nguồn mở Nukeviet

NukeViet là một ứng dụng trên nền web có thể sử dụng vào nhiều mục đích khác nhau. Phiên bản đang được phát hành theo giấy phép phần mềm tự do nguồn mở có tên gọi đầy đủ là NukeViet CMS gồm 2 phần chính là phần nhân (core) của hệ thống NukeViet và nhóm chức năng quản trị nội dung của CMS thường được sử dụng để xây dựng các website tin tức do đó người dùng thường nghĩ rằng NukeViet mạnh về hệ thống tin tức. Tuy nhiên, đội ngũ phát triển NukeViet đã phát triển nhiều hệ thống khác nhau cho NukeViet, nổi bật nhất phải kể đến NukeViet Portal (Cổng thông tin hai chiều dùng cho doanh nghiệp), NukeViet Edu Gate (Cổng thông tin tích hợp nhiều website, sử dụng cho phòng giáo dục, sở giáo dục) và NukeViet Tòa Soạn Điện Tử (Sử dụng cho các tòa soạn báo điện tử, trang tin điện tử).

#### Yêu cầu:

Hệ điều hành: Unix (Linux, Ubuntu, Fedore...) hoặc Windows

PHP: PHP 5.2 hoặc phiên bản mới nhất.

MySQL: MySQL 5.02 hoặc phiên bản mới nhất

Tùy chọn bổ sung

Tính năng mở rộng của Máy chủ: Máy chủ Apache cần hỗ trợ mod mod\_rewrite; Hoặc máy chủ IIS 7.0 hoặc IIS 7.5 cần cài thêm module rewrite Môi trường PHP mở rộng: Các thư viện PHP cần có: file\_uploads, session, mbstring, curl, gd2, zlib, soap, sockets, tidy, php\_zip.

**Ghi chú:** Những yêu cầu trên không có nghĩa là NukeViet 3.0 không làm việc trên các hệ thống khác, điều quan trọng là cần thiết lập môi trường làm việc phù hợp. Với những website sử dụng hosting, NukeViet 3.0 làm việc tốt nhất trên các hosting Linux cài sẵn Apache 2.2, PHP 5, MySQL 5, DirectAdmin hoặc Cpanel.

**Máy tính người truy cập:** NukeViet 3.0 cho kết quả là chuẩn xHTML 1.0 và CSS 2.1, đây là định dạng chuẩn mà hầu hết các trình duyệt hiện nay đang theo đuổi. Chính vì vậy các website làm trên nền NukeViet 3.0 có thể truy cập tốt trên các phiên bản mới nhất của trình duyệt FireFox, Internet Explorer, Google Chrome, Opera...

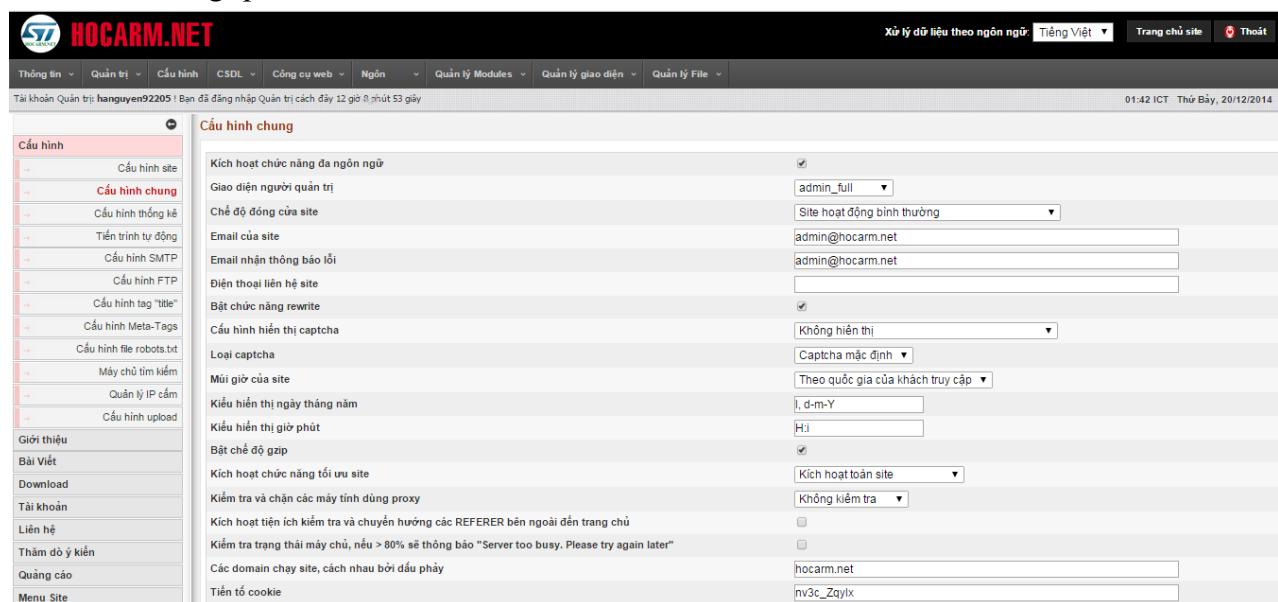
Việc cài đặt Nukeviet chúng ta có thể tìm hiểu thêm trên trang chủ [nukeviet.vn](http://nukeviet.vn)

## CHƯƠNG IV : THIẾT KẾ WEB HOCARM.NET

### 4.2 Thiết kế giao diện website

Sau khi cài đặt Nukeviet, cần phải thiết kế giao diện cho website phù hợp với mục đích sử dụng cũng như xây dựng nội dung cho trang web

Giao diện trang quản trị



Hình 4.1 Giao diện trang quản trị

Giao diện trang chủ website HocARM.net:



Hình 4.2 Trang chủ website hocarm.net

## CHƯƠNG IV : THIẾT KẾ WEB HOCARM.NET

Giao diện trang bài viết gồm các bài học về vi điều khiển STM32:

- Hướng dẫn: các hướng dẫn về cài đặt phần mềm, hướng dẫn tạo project, nạp chương trình ,...
- Bài học: kiến thức lý thuyết về STM32
- Ứng dụng : hướng dẫn lập trình cho các ứng dụng với vi xử lý STM32

The screenshot shows the HOCARM.NET website interface. At the top, there is a logo for ST HOCARM.NET and a main title "HOCARM.NET". Below the title, there is a navigation bar with links: Trang nhất, Bài Viết, Download, Thành viên, Liên hệ, and RSS feed. A sidebar on the left shows a breadcrumb trail: » Hướng dẫn » Bài Viết » Bài Học » Ứng Dụng. The main content area is divided into several sections:

- HƯỚNG DẪN**: Includes a sub-section "Giới thiệu về KIT STM32F103C8T6" with a thumbnail image of the kit and some text.
- BÀI HỌC**: Includes a sub-section "[BÀI HỌC 11] Công cụ phát triển STM-STUDIO" with a thumbnail image of the software and some text.
- ỨNG DỤNG**: Includes a sub-section "[ỨNG DỤNG 10] Giao tiếp I2C với EEPROM 24C02" with a thumbnail image of the circuit board and some text.
- Thành viên**: Shows a user profile for "hanguyen92205" with options to đổi mật khẩu and tài khoản.

On the right side of the main content area, there are three small images with their respective titles:

- [HƯỚNG DẪN 3] Nạp chương trình cho STM32 thông qua BootLoader
- [HƯỚNG DẪN 2] TẠO PROJECT CHO STM32 SỬ DỤNG KEILC
- [HƯỚNG DẪN 1] CÀI ĐẶT + CRACK KEIL C MDK CHO ARM

Hình 4.3 Giao diện trang bài viết

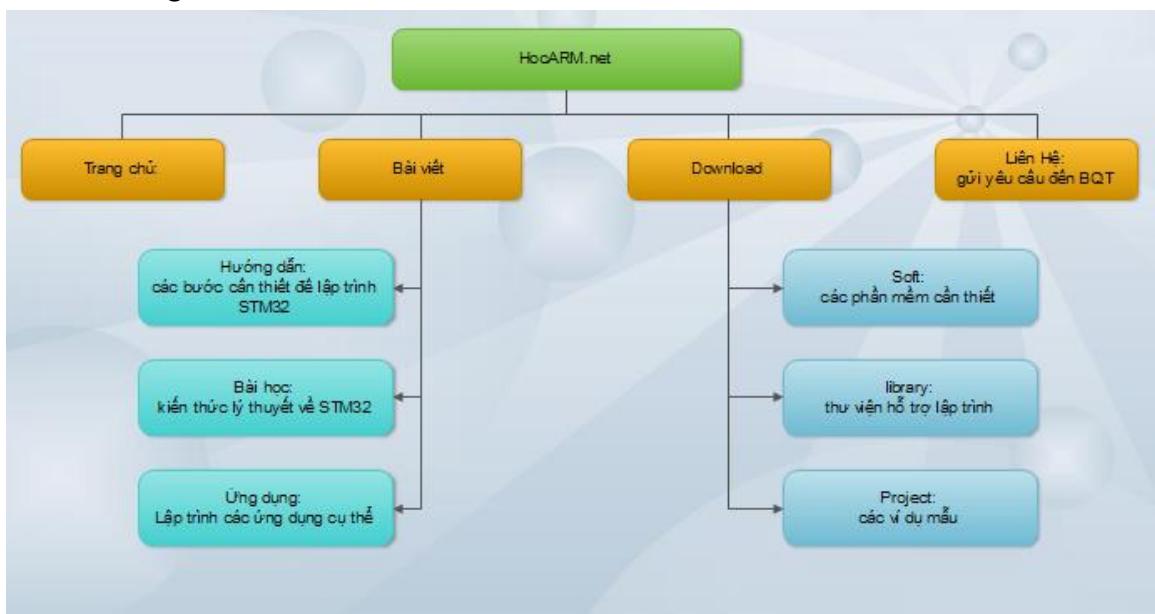
Giao diện trang download

- Soft: link download các phần mềm cần thiết cho việc lập trình vi xử lý STM32
- Library : thư viện hỗ trợ lập trình và giao tiếp người dùng cho vi xử lý STM32
- Project : Source code các chương trình lập trình trên KIT STM32



Hình 4.4 Giao diện trang download

Bố cục trang web :



Hình 4.5 Bố cục trang web

Trang web cũng hỗ trợ cho phép đăng ký thành viên và gửi bài về web, cũng có và xây dựng nội dung phong phú cho website. Bên cạnh đó website cũng hỗ trợ giao diện cho các thiết bị di động như: smartphone, tablet,... giúp cho người học dễ dàng tiếp cận hơn. Với sự bùng nổ và phát triển của internet thì trang web sẽ là nơi cung cấp nhiều thông tin, và cập nhật liên tục các nội dung, bài giảng về ARM Cortex M3 STM32 cho người học.

## TÀI LIỆU THAM KHẢO

- [1] STMicrocontroller, *datasheet STM32F103C8*
- [2] STMicrocontroller, *Reference manual advanced ARM-based 32-bit MCUs, July 2011*
- [3] website [www.st.com](http://www.st.com)
- [4] website [www.arm.vn](http://www.arm.vn)
- [5] VINADES.,JSC, *Hướng dẫn sử dụng Nukeviet 3*