MỤC LỤC

Lời Mở Đầu

Ngày nay với sự phát triển của ngành điện tử và ứng dụng điện tử đã giúp sự sáng tạo của con người trở thành hiện thực. Các lĩnh vực của cuộc sống đều áp dụng những thiết bị điện tử và dường như nhìn đâu trong gia đình chúng ta cũng có thiết bị điện tử. Ngành điện tử và ứng dụng điện tử đã tạo chỗ đứng và khẳng định được tầm quan trọng của mình đối với nhu cầu của con người.

Với những ứng dụng cho các hệ thống nhúng ngày càng trở nên phổ biến: từ những ứng dụng đơn giản như điều khiển một chốt đèn giao thông định thời, đếm sản phẩm trong một dây chuyền sản xuất, điều khiển tốc độ động cơ điện một chiều, thiết kế một biển quảng cáo dùng Led ma trận, một đồng hồ thời gian thực .Đến các ứng dụng phức tạp như hệ thống điều khiển robot, bộ kiểm soát trong nhà máy hoặc hệ thống kiểm soát các máy năng lượng hạt nhân. Các hệ thống tự động trước đây sử dụng nhiều công nghệ khác nhau như các hệ thống tự động hoạt động bằng nguyên lý khí nén, thủy lực, rơle cơ điện, mạch điện tử số, các thiết bị máy móc tự động bằng các cam chốt cơ khí. Các thiết bị, hệ thống này có chức năng xử lý và mức độ tự động thấp so với các hệ thống tự động hiện đại được xây dựng trên nền tảng của các hệ thống nhúng.

Trong nhiều năm trước, các dòng vi điều khiển 8051 được sinh viên dùng nhiều với tính năng đơn giản, dễ sử dụng; AVR được sử dụng nhiều trong các cuộc thi Robocon nhờ tốc độ sử lý khá cao, ổn định; PIC với ưu thế tốc độ cao, chi phí thấp hơn cũng được nghiên cứu, sử dụng nhiều, đặc biệt trong các cuộc thi lập trình tay nghề khu vực và thế giới. Nhưng trong một vài năm trở lại đây, có một dòng vi điều khiển mới, càng ngày càng nắm vị trí quan trọng trong các lĩnh vực đòi hỏi tốc độ xử lý cao như điện tử viễn thông, sản xuất các dòng diện thoại di động smartphone, giám sát, an ninh… Đó là họ vi điều khiển ARM. Với rất nhiều thế hệ ra đời, với nhiều tính năng , công dụng khác nhau.

Với nhiều tính năng vượt trội của ARM và xu thế lựa chọn dòng vi điều khiển mới ở Việt Nam nên trong đề tài nghiên cứu khoa học này, dưới sự giúp đỡ của Thầy Nguyễn Huy Dũng, em thực hiện đề tài nghiên cứu Ứng dụng lập trình điều khiển động cơ bước sử dụng chip ARM Cortex M3 STM32F103RC.

Chương 1

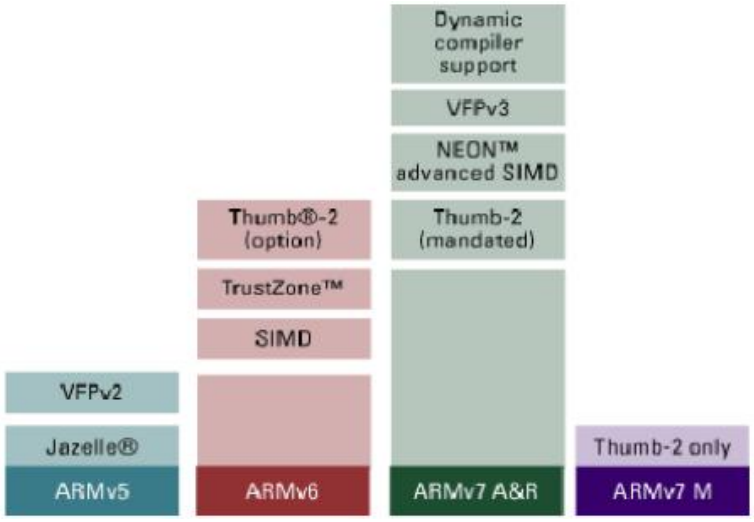
TỔNG QUAN VỀ CORTEX

Bộ xử lý Cortex là thế hệ lõi nhúng kế tiếp từ ARM. Cortex thừa kế các ưu điểm từ các bộ xử lí ARM trước đó, nó là một lõi xử lý hoàn chỉnh, bao gồm bộ xử lí trung tâm Cortex và một hệ thống các thiết bị ngoại vi xung quanh, Cortex cung cấp phần xử lí trung tâm của một hệ thống nhúng. Để đáp ứng yêu cầu khắt 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.

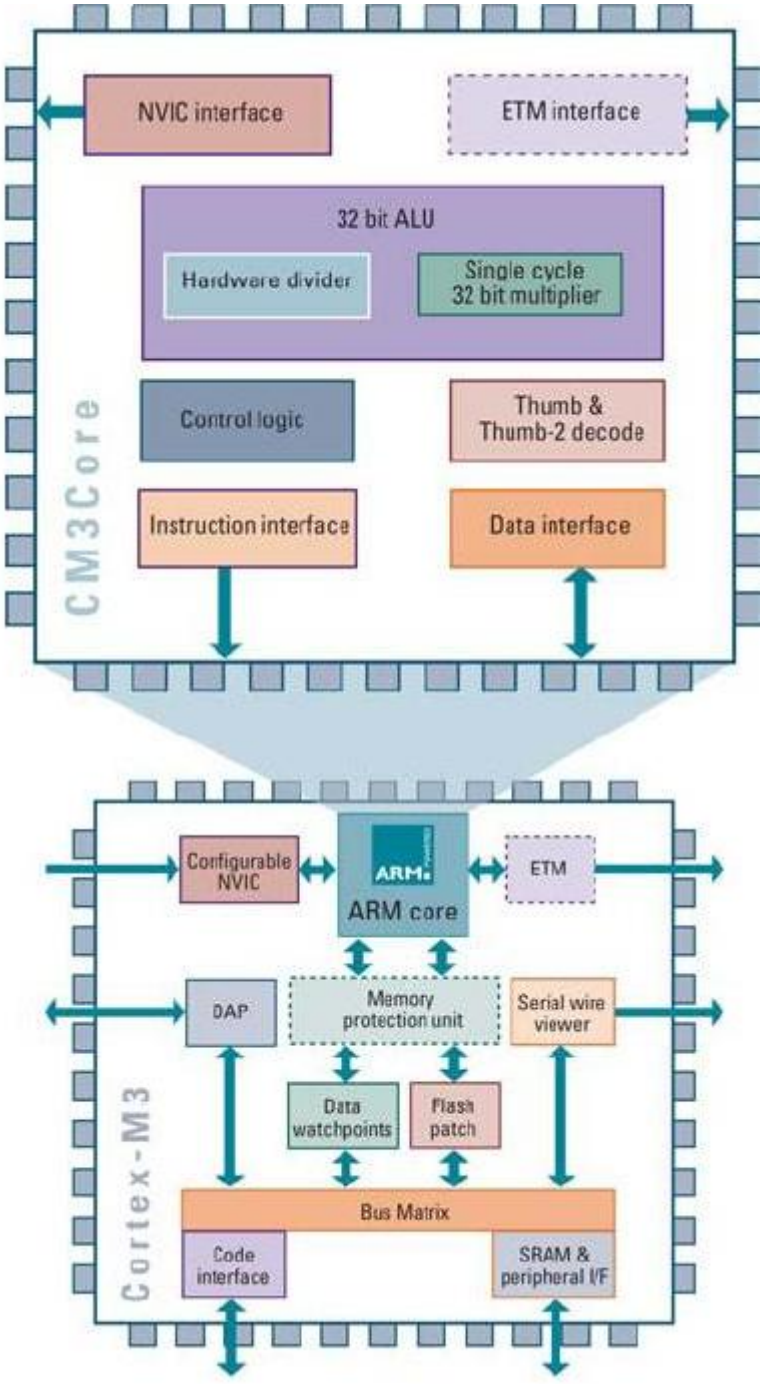
1.1. Các phiên bản kiến trúc ARM



Hinh 1.1.Các phiên bản kiến trúc của lõi 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.

1.2 Bộ xử lí Cortex và đơn vị xử lí trung tâm Cortex



Hình 1.2. Kiến trúc vi xử lí ARM Cortex-M3

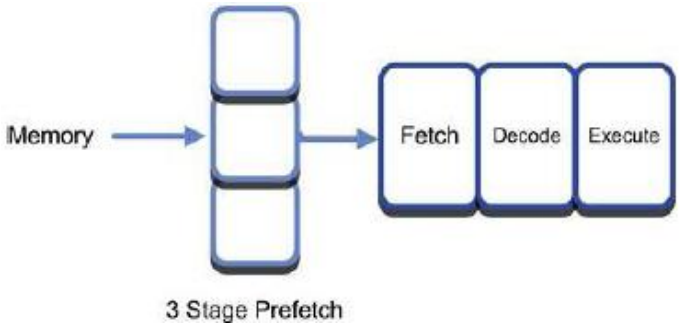
Thuật ngữ bộ xử lí Cortex (Cortex processor) và đơn vị xử lí trung tâm Cortex (Cortex CPU) sẽ được sử dụng để phân biệt giữa nhúng lõi Cortex hoàn chỉnh và bộ xử lí trung tâm RISC nội (internal RISC CPU).

1.3 Đơn vị xử lí 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.

1.3.1 Kiến trúc đường ống (Pipline)

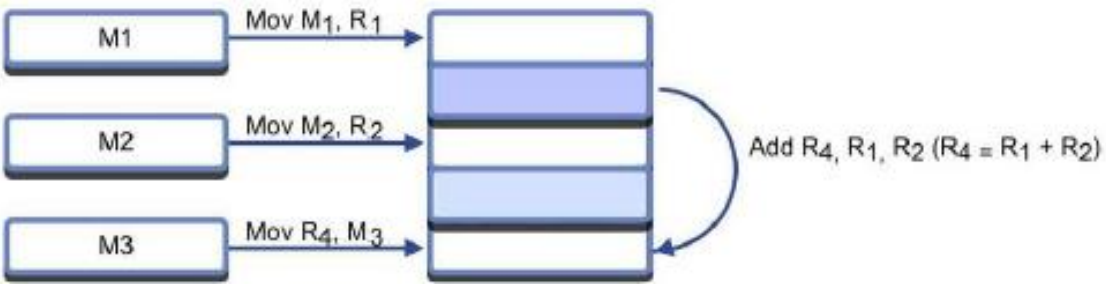
CPU Cortex có thể thực thi hầu hết các lệnh trong một chu kì đơn. Giống như CPU của ARM7 và ARM9, việc thực thi này đạt được với một đường ống ba tầng. Tuy nhiên Cortex-M3 khả năng dự đoán việc rẽ nhánh để giảm thiểu số lần làm rỗng (flush) đường ống.



Hinh 1.3. Kiến trúc đường ống của ARM Cortex-M3

1.3.2 Mô hình lập trình (Programmer's model)

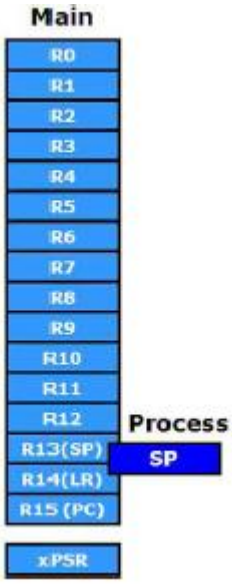
CPU Cortex là bộ xử lý dựa trên kiến trúc RISC, do đó hỗ trợ kiến trúc nạp và lưu trữ (load and store architecture). Để thực hiện lệnh xử lý dữ liệu, các toán hạng phải được nạp vào một tập thanh ghi trung tâm, các phép tính dữ liệu phải được thực hiện trên các thanh ghi này và kết quả sau đó được lưu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



Hinh 1.4. Kiến trúc load và store của ARM Cortex-M3

Tập thanh ghi này bao gồm mười sáu thanh ghi 32-bit.

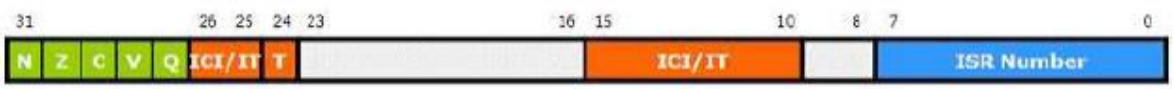
- Các thanh ghi R0-R12 là các thanh ghi đơn giản, có thể được dùng để chứa các biến của chương trình.
- Thanh ghi R13 được dùng như là con trỏ ngăn xếp (stack pointer). Trong CPU Cortex có hai ngăn xếp được gọi là main stack và process stack.
- Thanh ghi R14 tiếp theo được gọi là thanh ghi liên kết (link register). Thanh ghi này được sử dụng để lưu trữ các địa chỉ trở về khi một cuộc gọi thủ tục (call a procedure) được thực hiện. Điều này cho phép CPU Cortex thực hiện rất nhanh việc nhập và thoát khỏi một thủ tục (fast entry and exit to a procedure).
- Thanh ghi R15là bộđếm chương trình (Program Counter)



Hinh 1.5. Mô hình lập trình của ARM Cortex-M3

1.3.2.1 Thanh ghi XPSR

Ngoài tập thanh ghi trung tâm còn có một thanh ghi riêng biệt được gọi là thanh ghi trạng thái chương trình (Program Status Register). XPSR chứa một số các vùng chức năng quan trọng ảnh hưởng đến việc thực thi của CPU Cortex.

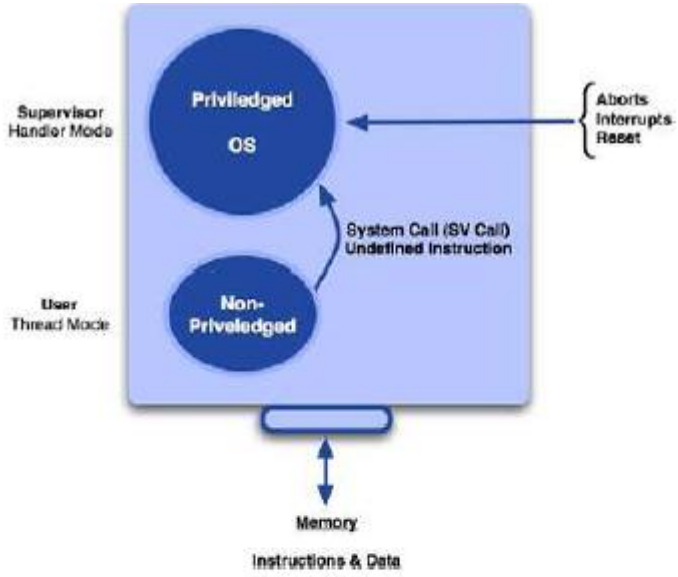


Hinh 1.6. Thanh ghi trạng thái chương trình của CPU Cortex

- Năm bit đầu là những cờ mã điều kiện và được gán biệt hiệu (aliased) như thanh ghi trạng thái chương trình ứng dụng. Bốn cờ N, Z, C, V (Negative, Zero, Carry và Overflow) sẽ được thiết lập và xóa tùy thuộc vào kết quả của một lệnh xử lýdữ liệu. Bit Q là được sử dụng bởi các lệnh toán học DPS để chỉ ra rằng một biến đã đạt giá trị tối đa hoặc giá trị tối thiểu của nó.
- Giống như tập lệnh ARM32-bit, các lệnh Thumb-2 chỉ được thực hiện nếu mã điều kiện của lệnh phù hợp với trạng thái của các cờ trong thanh ghi trạng thái chương trình ứngdụng (Application Program Status Register). Nếu mã điều kiện của lệnh không phù hợp, thì lệnh đi ngang qua đường ống như là một lệnh NOP (lệnh này không làm gì cả). Điều này đảm bảo rằng các lệnh đi qua đường ống một cách trơn tru và giảm thiểu làm rỗng đường ống.

1.3.3 Các chế độ hoạt động của CPU

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). Trong chế độ đặc quyền, CPU có quyền truy cập tất cả các lệnh. Trong chế độ không co đặc quyền, một số lệnh bị cấm truy cập (như lệnh MRS và MSR cho phép truy cập vào xPSR và các trường của nó). Ngoài ra, việc cập các thanh ghi điều khiển hệ thống trong bộ vi xử lý Cortex cũng bị cấm. Cách sử dụng ngăn xếp (stack) cũng có thể được cấu hình. Ngăn xếp chính (main stack-R13) có thể được sử dụng bởi cả hai chế độ Thread và Handler. Chế độ Handler có thể được cấu hình để sử dụng ngăn xếp quá trình (process stack-R13 banked register).



Modes (Thread out of reset)		Operations (privilege out of reset)	Stacks (Main out of reset)
	Handler - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

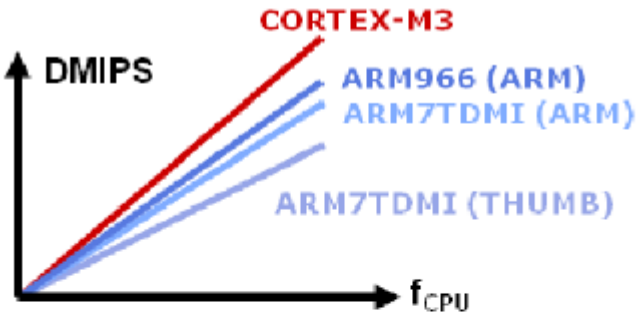
Hình 1.7.Mô hình hoạt động của chế độ Thread và Handler

Sau khi reset, bộ xử lý Cortex sẽ chạy trong cấu hình phẳng (flat configuration). Cả hai chế độ Thread và Handler được thực thi trong chế độ đặc quyền (privileged mode), do đó, không có sự giới hạn nào về quyền truy cập vào bất kỳ tài nguyên của bộ xử lý. Cả hai chế độ Thread và Handler đều sử dụng ngăn xếp chính.

1.3.4 Tập lệnh Thumb-2

Các CPU ARM7 và ARM9 có thể thực thi hai tập lệnh: ARM 32-bit và Thumb 16-bit. Điều này cho phép người phát triển để tối ưu hoá chương trình của mình bằng cách lựa chọn tập lệnh nào được sử dụng cho thủ tục khác nhau: lệnh 32-bit để tăng tốc độ xử lí và lệnh 16-bit để nén mã chương trình. CPU Cortex được thiết kế để thực thi tập lệnh Thumb-2, là một sự pha trộn của lệnh 16-bit và 32-bit. Tập lệnh thumb-2 cải tiến 26% mật độ mã so với tập lệnh ARM 32-bit và 25% hiệu suất so với tập lệnh Thumb 16-bit. Tập

lệnh Thumb2 có một số lệnh nhân được cải tiến, có thể thực hiện trong một chu kì đơn và khả năng thực hiện phép chia bằng phần cứng và chỉ mất từ 2-7 chu kỳ.



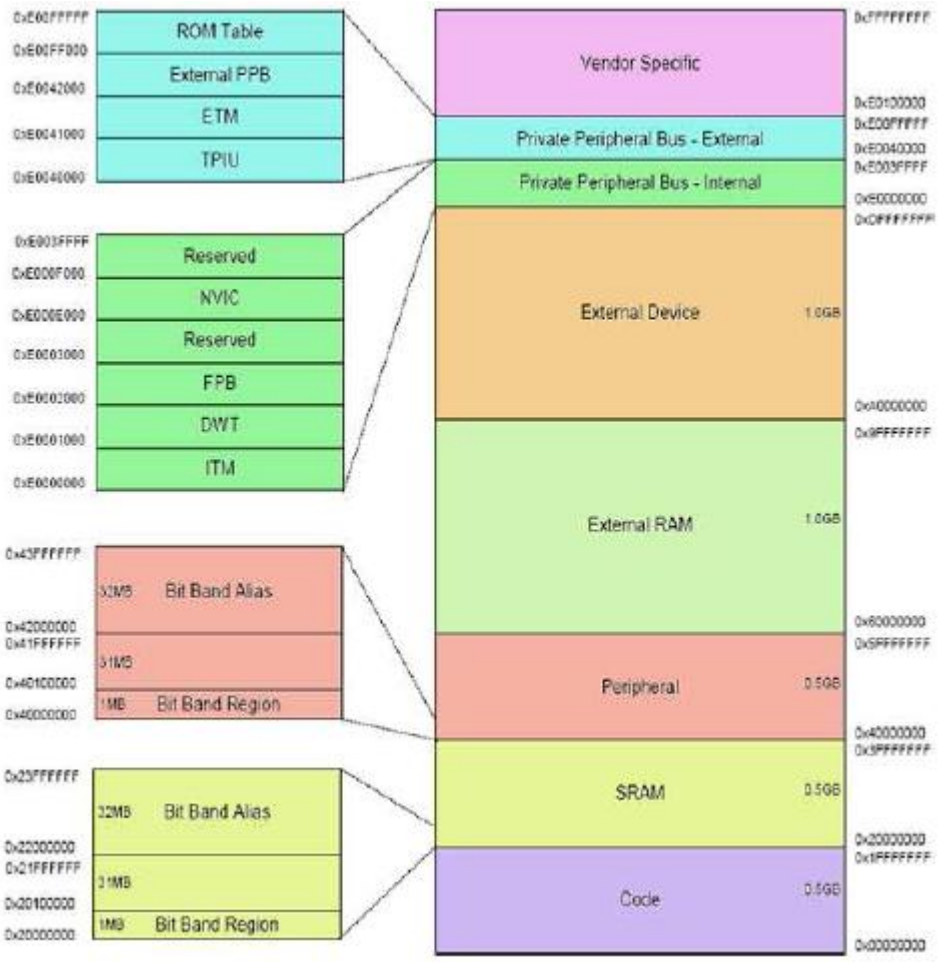
Source	Destination	Cycles
16b x 16b	32b	1
32b x 16b	32b	1
32b x 32b	32b	1
32b x 32b	64b	3-7*

Hình 1.8.Đồ thị biểu diễn hiệu năng của bộ xử lý Cortex

Điểm chuẩn bộ xử lý Cortex (Cortex processor benchmark) cho một mức độ thực hiện là 1,25 DMIPS/MHz, cao hơn so với ARM7 (0.95 DMIPS/MHz với tập lệnh ARM và 0.74 DMIPS/MHz với tập lệnh Thumb) và ARM9

1.3.5 Bản đồ bộ nhớ (Memory Map)

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.



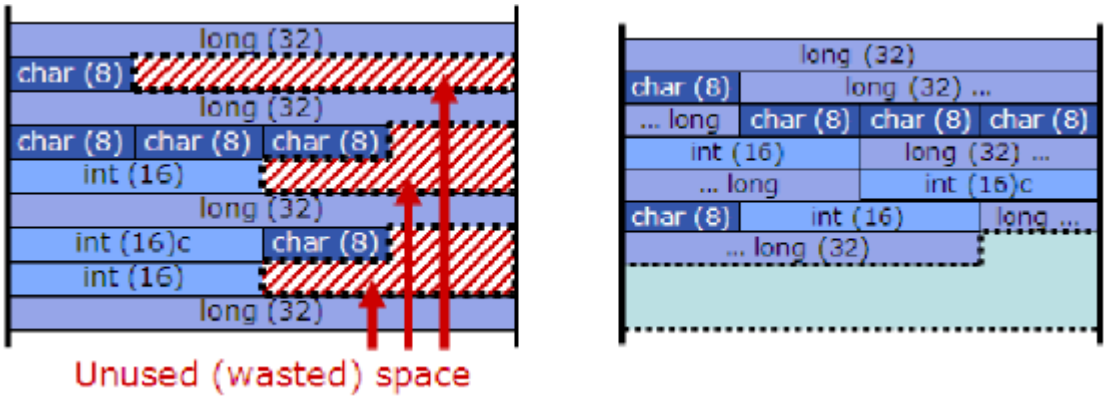
Hình 1.9.Bản đồ bộ nhớ tuyến tính 4Gbyte của bộ xử lý Cortex-M3

Một Gbyte bộ nhớ đầu tiên được chia đều cho một vùng mã (code region) và một vùng SRAM (SRAM region). Không gian mã được tối ưu hóa để thực thi từ bus I-Code. Tương tự, SRAM được nối đến bus D-Code. Mặc dù mã có thể được nạp và thực thi từ SRAM, các lệnh sẽ được lấy bằng cách sử dụng bus hệ thống, vì vậy phải chịu thêm một trạng thái chờ (an extra wait state). Tức là mã chạy trên SRAM sẽ chậm hơn so với từ bộ nhớ Flash trên chip (on-chip) nằm trong vùng mã. Vùng 0,5 Gbyte tiếp theo của bộ nhớ là vùng ngoại vi trên chip, tất cả thiết bị ngoại vi được cung cấp bởi nhà sản xuất vi điều khiển sẽ được đặt tại vùng này. Vùng 1 Mbyte đầu tiên gồm cả SRAM (màu vàng nhạt) và vùng ngoại vi (màu hồng nhạt) được định địa chỉ theo bit, sử dụng một kỹ thuật được gọi là dải bit (bit banding). Từ đó tất cả SRAM và các thiết bị ngoại vi người dùng (user peripherals) trên STM32 được đặt tại vùng này, và tất cả các vị trí bộ nhớ của những vùng này trên STM32

đều có thể được thao tác theo word-wide hoặc bitwise. Không gian địa chỉ 2 Gbyte tiếp theo được phân cho bộ nhớ ngoài- ánh xạ SRAM và thiết bị ngoại vi (external RAM và external Device). Vùng 0,5 Gbyte cuối cùng được phân cho các thiết bị ngoại vi bên trong của bộ xử lí Cortex và một khu vực dành cho các cải tiến trong tương lai của nhà sản xuất chip cho bộ xử lý Cortex. Tất cả các thanh ghi của bộ xử lý Cortex được đặt ở vị trí cố định cho tất cả vi điều khiển dựa trên lõi Cortex. Điều này cho phép mã chương trình dễ dàng được chuyển giữa các biến thể STM32 khác nhau và các vi điều khiển dựa trên lõi Cortex của các nhà sản xuất chip khác.

1.3.6 Truy cập bộ nhớ không xếp hàng (Unaligned Memory Accesses)

Tập lệnh ARM7 và ARM9 có khả năng truy cập các biến có dấu và không dấu có kích thước byte, half word (thường là 2byte) và word (thường là 4byte). Điều này cho phép CPU hỗ trợ các biến số nguyên mà không cần đến thư viện phần mềm hỗ trợ, thường được yêu cầu đối với vi điều khiển 8 và 16-bit. Tuy nhiên, các phiên bản CPU ARM trước đó gặp bất lợi ở chỗ, nó chỉ có thể truy cập dữ liệu kích thước là word hoặc half word. Điều này hạn chế khả năng của trình liên kết của trình biên dịch (compiler linker) trong việc đóng gói dữ liệu vào SRAM và như vậy một số SRAM sẽ bị lãng phí (Việc lãng phí này có thể lên đến 25% tùy thuộc vào sự kết hợp của các biến được sử dụng). Bộ xử lí Cortex-M3 có thể truy cập bộ nhớ không xếp hàng, việc đó đảm bảo rằng SRAM được sử dụng một cách hiệu quả.

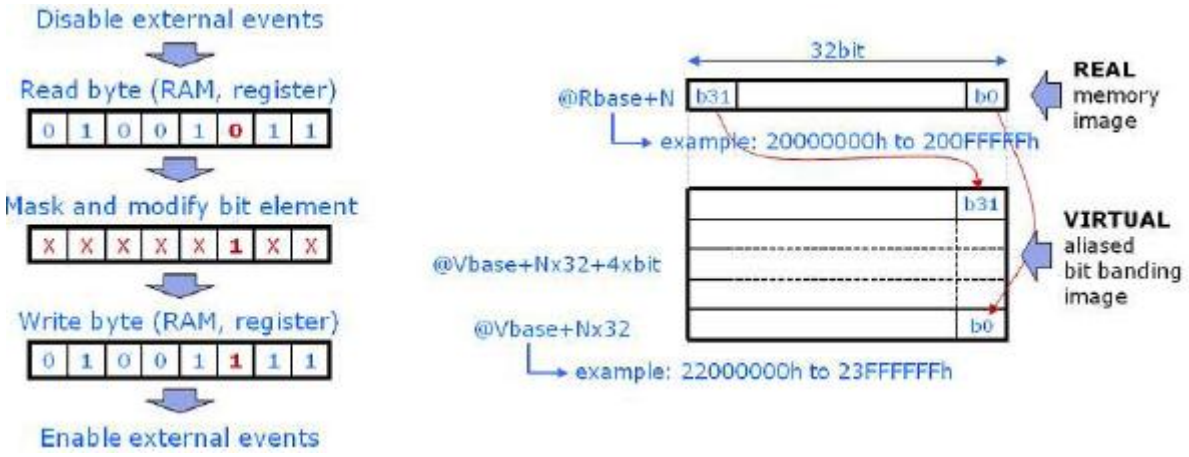


Hình 1.10.Khả năng truy cập bộ nhớ không xếp hàng của bộ xử lý Cortex-M3 so với các phiên bản CPU ARM trước đó

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.

1.3.7 Dải Bit (Bit Banding)

Các phiên bản CPU ARM7 và ARM9 trước đó chỉ có thể thực hiện thao tác bit trên bộ nhớ SRAM và vùng nhớ thiết bị ngoại vi bằng cách dùng các phép toán AND và OR. Điều này đòi hỏi thao tác đọc sửa đổi ghi (READ MODIFY WRITE operation), thao tác này sẽ tốn nhiều chu kì thực hiện để thiết lập và xoá các bit riêng biệt và cần nhiều không gian mã cho mỗi bit.

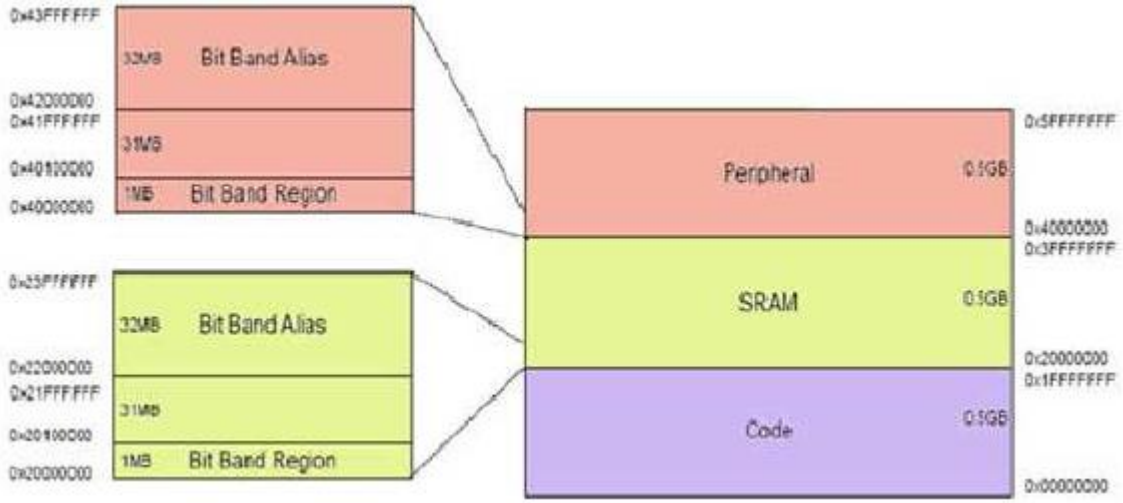


Hình 1.11.Thao tác đọc sửa đổi ghi của ARM7 và ARM9 và kỹ thuật dải bit của bộ xử lý Cortex-M3

Kỹ thuật dải Bit cho phép bộ xử lí Cortex-M3 thao tác các bit trong khi vẫn giữ được số lượng bóng bán dẫn ở mức tối thiểu.

Để khắc phục những hạn chế trong các thao tác bit ở CPU ARM7 và ARM9, có thể đưa ra các lệnh chuyên dụng để thiết lập hoặc xoá bit, hoặc một bộ xử lý Boolean đầy đủ, nhưng điều này sẽ làm tăng kích thước và sự phức tạp của CPU Cortex. Thay vào đó, một kỹ thuật gọi là dải bit cho phép thao tác bit trực tiếp trên các phần không gian bộ nhớ của các thiết bị ngoại vi và SRAM, mà không sự cần bất kỳ lệnh đặc biệt nào. Các khu vực định địa chỉ bit

của bản đồ bộ nhớ Cortex bao gồm vùng bit band (lên đến 1Mbyte bộ nhớ thực hoặc các thanh ghi ngoại vi) và vùng biệt hiệu bit band (bit band Alias region) chiếm đến 32Mbyte của bản đồ bộ nhớ. Dải Bit hoạt động bằng cách ánh xạ mỗi bit trong vùng bit band tới một địa chỉ word trong vùng Alias. Vì vậy, bằng cách thiết lập và xoá địa chỉ word được đặt biệt hiệu (aliased word address) chúng ta có thể thiết lập và xoá các bit trong bộ nhớ thực.



Hình 1.12.Dải bit của vùng bộ nhớ SRAM và các ngoại vi

Dải Bit được hỗ trợ trên 1Mb đầu tiên của khu vực SRAM và ngoại vi . Nó bao gồm tất cả các tài nguyên của STM32.

Kỹ thuật Bit Banding cho phép thực hiện thao tác bit riêng lẻ mà không cần bất kì lệnh đặc biệt nào, điều này giữ cho kích thước tổng thể của lõi Cortex nhỏ nhất có thể. Trong thực tế, chúng ta cần phải tính toán địa chỉ của các word nằm trong vùng Bit Band Alias cho một vị trí bộ nhớ nhất định trong không gian bộ nhớ của thiết bị ngoại vi hoặc SRAM. Công thức để tính toán alias address như sau:

- Địa chỉ trong khu vực Bit Band Alias = Bit band alias base address + bit word offset
- bit word offset = Byte offset from bit band base x 0x20 + bit number x 4

1.4 Bộ xử lí Cortex

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

1.4.1 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.

1.4.2 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. Tuy nhiên, trong STM32 khối DMA được thiết kế để làm việc hòa hợp với CPU Cortex.

1.4.3 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 kì 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. Bằng cách thiết lập bit CLKSOURCE, đồng hồ SysTick sẽ chạy ở tần số đúng bằng tần số hoạt động của CPU. Khi bit này được xóa, SysTick sẽ chạy ở tần số bằng 1/8 CPU.

Hình 1.13. Các thanh ghi trạng thái và điều khiển của SysTick

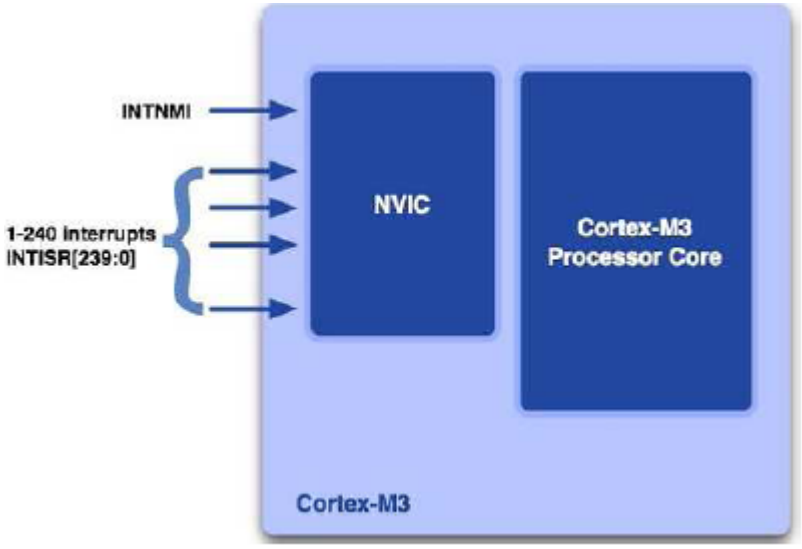
Đồng hồ SysTick có ba thanh ghi. Giá trị hiện tại và giá trị tải (current value và reload value) nên được khởi tạo với chu kì đếm. Thanh ghi trạng thái và điều khiển có một bit cho phép (ENABLE bit) để bắt đầu chạy bộ đếm thời gian và một bit TICKINT cho phép tín hiệu ngắt.

1.4.4 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).

1.4.5 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...



Hình 1.14. Cấu trúc của NVIC trong bộ xử lí Cortex

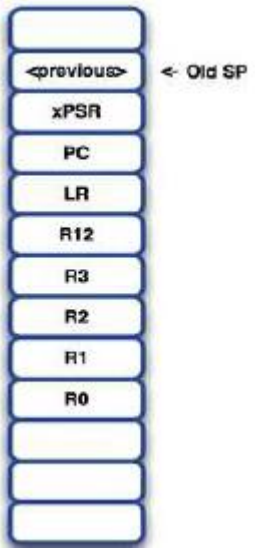
NVIC cũng được thiết kế để có một độ trễ khi đáp ứng ngắt rất thấp. Đây là một đặc điểm của chính bản thân bộ NVIC và của tập lệnh Thumb-2, nó cho phép thực thi các lệnh nhiều chu kì (multi-cycle instructions) như lệnh tải và lưu trữ nhiều dữ liệu (load and store multiple instruction) có thể được ngắt khi đang thực thi. Do đó độ trễ khi đáp ứng ngắt là xác định, với nhiều đặc điểm xử lí ngắt tiên tiến, nó hỗ trợ rất tốt cho các ứng dụng thời gian thực.

Như tên gọi của nó,NVIC được thiết kế để hỗ trợ các ngắt lồng nhau (nested interrupts) và trênSTM32 có16 cấp độ ưu tiên ngắt.

Mặc dù NVIC là một đơn vị đạt chuẩn bên trong lõi Cortex, để giữ cho số bóng bán dẫn ở mức tối thiểu, số đường tín hiệu ngắt đi vào NVIC có thể cấu hình khi vi điều khiển được thiết kế. NVIC có một ngắt không che mặt nạ (non-maskable interrupt) và hơn 240 đường tín hiệu ngắt bên ngoài và có thể được kết nối với ngoại vi người dùng. Ngoài ra còn có thêm 15 nguồn ngắt bên trong lõi Cortex, được sử dụng để xử lý các ngắt nội ngoại lệ trong lõi Cortex. Bộ NVIC của STM32 được tổng hợp với tối đa là 43 đường ngắt che mặt nạ (maskable interrupt lines).

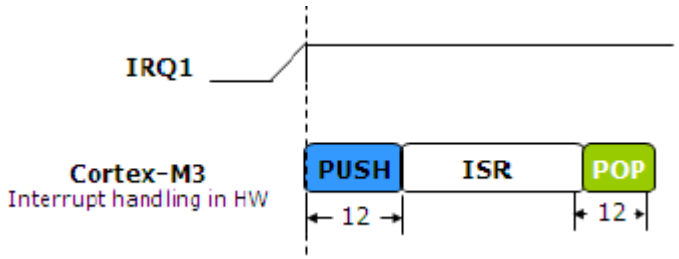
1.4.5.1 Nhập và thoát khỏi một ngoại lệ của NVIC (NVIC Operation Exception Entry And Exit)

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 kì 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 1.15. Stack frame trong chế độ ngắt

Khi kết thúc quá trình phục vụ ngắt, khung ngăn xếp được khôi phục tự động bởi vi chương trình (microcode), song song với thao tác đó thì địa chỉ trở về được lấy về, để chương trình nền có thể tiếp tục thực hiện chỉ sau 12 chu kỳ.



Hình 1.16. Đáp ứng thời gian khi một ngắt bất kì xảy ra của Cortex-M3

1.4.5.2 Các chế độ xử lí 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.

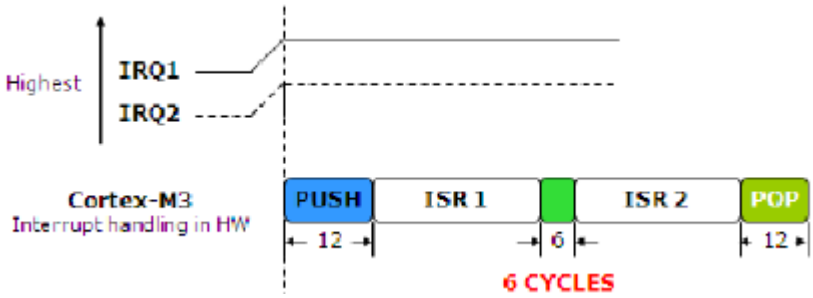
1.4.5.2.1 Quyền ưu tiên ngắt (Interrupt Pre-emption)

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.

1.4.5.2.2 Kỹ thuật Tail Chaining trong NVIC

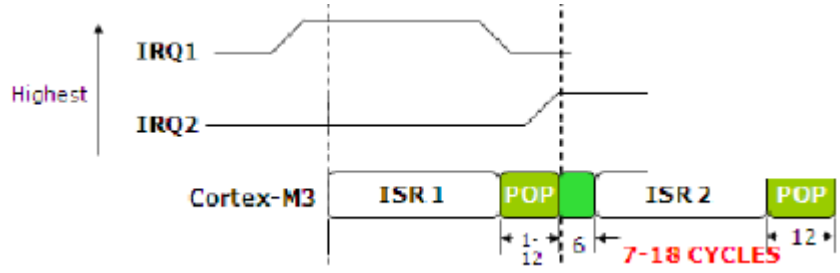
Nếu một ngắt có mức ưu tiên cao đang chạy và đồng thời một ngắt có mức

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. Tuy nhiên, khi đến cuối trình phục vụ ngắt CPU Cortex không trở về chương trình ứng dụng nền, vì vậy mà stack frame của ngắt này không được khôi phục, thay vào đó chỉ có địa chỉ của hàm phục vụ ngắt có mức ưu tiên cao nhất kế tiếp được lấy về.



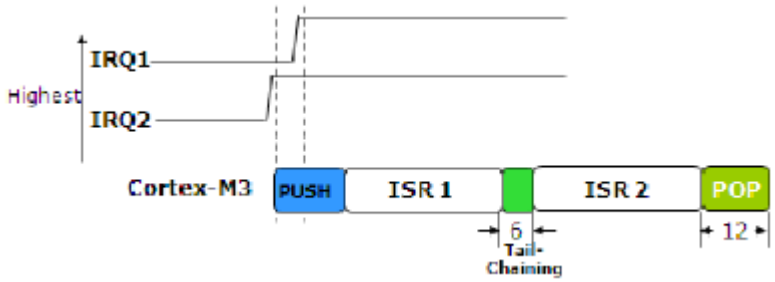
Hình 1.17. Đáp ứng thời gian khi hai ngắt xảy ra đồng thời của Cortex-M3

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 trở về được lấy, tiếp đó chương trình ứng dụng nền có thể bắt đầu thực thi chỉ trong 12 chu kỳ xung nhịp. Nếu một ngắt có mức ưu tiên thấp xuất hiện trong khi một ngắt khác đang thực thi chuẩn bị thoát khỏi trình phục vụ ngắt, thao tác POP (lấy dữ liệu từ ngăn xếp) sẽ bị bỏ qua và con trỏ stack sẽ được cuộn về giá trị ban đầu để có thể tiếp tục lưu trữ stack frame của ngắt mới xuất hiện, sẽ có một độ trễ 6 chu kỳ xung nhịp cho tới khi địa chỉ của ISR mới được lấy về. Điều này tạo ra một độ trễ từ 7-18 chu kỳ xung nhịp trước khi trình phục vụ ngắt mới có thể bắt đầu được thực hiện.



Hình 1.18. Đáp ứng thời gian khi hai ngắt xảy ra lần lượt của Cortex-M3

Trong một hệ thống thời gian thực thường xuất hiện tình huống, trong khi một ngắt có mức ưu tiên thấp đang được phục vụ, thì chỉ có một ngắt có mức ưu tiên cao hơn xuất hiện. Nếu tình huống này xảy ra trong quá trình PUSH dữ liệu lên ngăn xếp, NVIC sẽ chuyển sang phục vụ ngắt ưu tiên cao hơn. Việc PUSH dữ liệu lên ngăn xếp được tiếp tục và sẽ có tối thiểu 6 chu kỳ xung nhịp tại thời điểm ngắt ưu tiên cao hơn xuất hiện, cho tới khi địa chỉ của ISR mới được lấy về.



Hình 1.19. Đáp ứng thời gian khi ngắt ưu tiên cao đến sau của Cortex-M3

Sau khi ngắt ưu tiên cao hơn thực hiện xong, ngắt ưu tiên thấp ban đầu sẽ được nối đuôi (tail chain) và bắt đầu thực hiện sau 6 chu kỳ xung nhịp.

1.4.5.3 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 cần 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.

1.4.5.3.1 Bảng vector ngắt (Exception Vector Table)

Bảng vector ngắt của Cortex bắt đầu ở dưới cùng của bảng địa chỉ. Tuy nhiên bảng vector bắt đầu tại địa chỉ 0x00000004 thay vì là 0x00000000 như ARM7 và ARM9, bốn byte đầu tiên được sử dụng để lưu trữ địa chỉ bắt đầu của con trỏ ngăn xếp (stack pointer).

No	Exception Type	Priority	Type of Priority	Descriptions
1	Reset	-3(Highest)	fixed	Reset
2	NMI	-2	fixed	Non-Maskable Intertupt
3	Hard Fault	-1	fixed	Default fault if other hander not implemented
4	MemManage Fault	0	settable	MPU violation or access to illegal locations
5	Bus Fault	1	settable	Fault if AHB interface receives error
6	Usage Fault	2	settable	Exceptions due to program errors
7-10	Reserved	N.A	N.A	
11	SVCall	3	settable	System Service call
12	Debug Monitor	4	settable	Break points watch points, external debug
13	Reserved	N.A	N.A	
14	PendSV	5	settable	Pendable request for System Device
15	SYSTICK	6	settable	System Tick Timer
16	Interrupt # 0	7	settable	External Interrupt # 0
….	…….	……	settable	……
256	Interrupt # 240	247	settable	External Interrupt # 240

Hình 1.20. Bảng vector ngắt của Cortex-M3

Mỗi vector ngắt có độ rộng là bốn byte và giữ địa chỉ bắt đầu của trình phục vụ ngắt tương ứng, 15 vector ngắt đầu tiên là các ngắt đặc biệt chỉ xảy ra trong lõi Cortex, bao gồm reset vector, non-maskable interrupt, quản lý fault và error, debug exceptions và ngắt timer của SysTick. Tập lệnh Thumb-2 cũng bao

gồm lệnh gọi dịch vụ hệ thống (system service call), khi được gọi, nó sẽ tạo ra một ngắt đặc biệt. Các ngắt ngoại vi người dùng bắt đầu từ vector 16, được định nghĩa bởi nhà sản xuất và được liên kết đến thiết bị ngoại vi.

Sau khi cấu hình xong bảng vector ngắt và định nghĩa các ISR (Interrupt Service Routine), chúng ta có thể cấu hình NVIC để xử lý ngắt của timer SysTick qua hai bước: thiết lập mức ưu tiên ngắt và sau đó cho phép ngắt nguồn. Các thanh ghi NVIC nằm trong vùng điều khiển hệ thống của Cortex-M3 và chỉ có thể truy cập khi CPU đang chạy ở chế độ đặc quyền (privileged mode).

Hình 1.21. Các thanh ghi trạng thái và điều khiển của NVIC

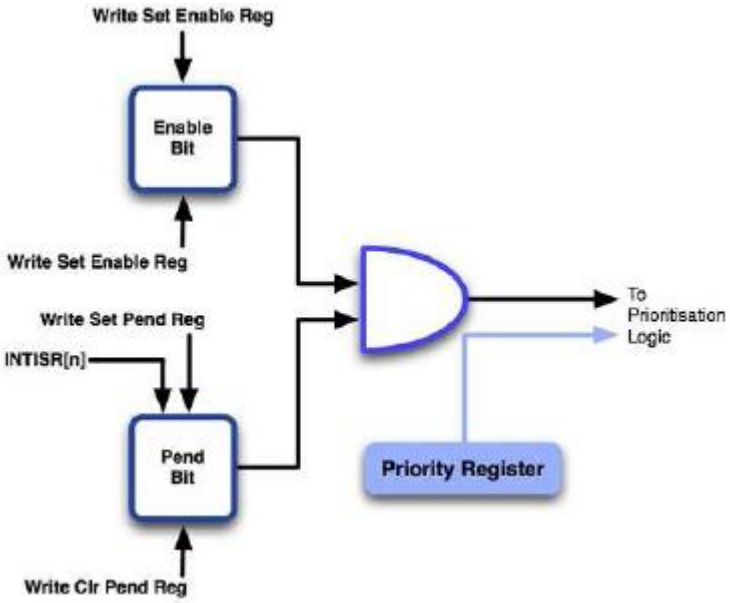
Các ngắt đặc biệt bên trong Cortex được cấu hình thông qua các thanh ghi điều khiển và thanh ghi cấu hình mức ưu tiên của hệ thống, trong khi đó các thiết bị ngoại vi người dùng được cấu hình bằng cách sử dụng các thanh ghi IRQ (Interrupt Request). Ngắt của SysTick là một ngắt đặc biệt bên trong Cortex 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. Để cấu hình ngắt cho SysTick, chúng ta cần phải cấu hình cho SysTick chạy và cho phép ngắt bên trong SysTick:

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

Mức ưu tiên của mỗi exception (ngắt đặc biệt) bên trong Cortex có thể

được cài đặt thông qua các thanh ghi cấu hình mức độ ưu tiên trong hệ thống. Mức độ ưu tiên của các exception như Reset, NMI và hard fault được cố định để đảm bảo rằng lõi Cortex sẽ luôn luôn sẵn sàng cho một exception được biết trước. Mỗi exception có một trường 8-bit nằm trong ba thanh ghi về mức độ ưu tiên của hệ thống. Tuy nhiên STM32 chỉ thực hiện 16 mức độ ưu tiên, như vậy chỉ có bốn bit của trường này được dùng. Một điều quan trọng cần lưu ý là mức ưu tiên được thiết lập bởi bốn bit có trọng số cao nhất.

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. Bên cạnh đó cũng có các thanh ghi tương ứng để cấm bất kì một nguồn ngắt. Ngoài ra NVIC cũng bao gồm các thanh ghi báo chờ (pending) và kích hoạt (active) cho phép xác định tình trạng hiện tại của một nguồn ngắt.

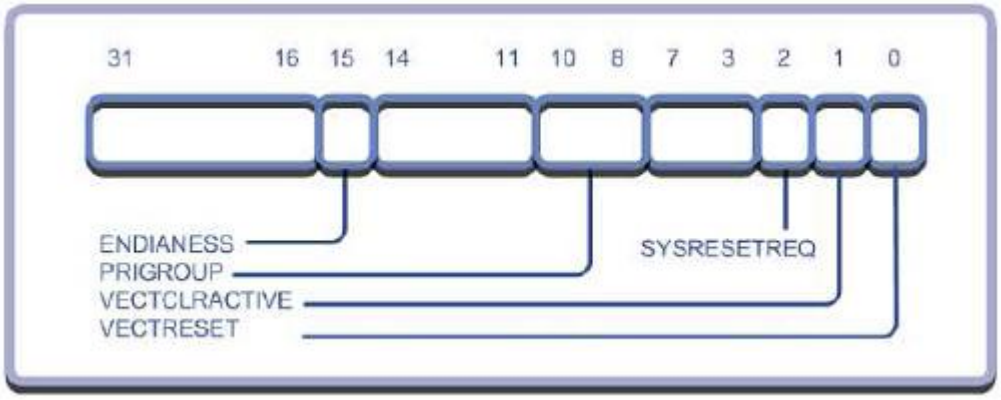


Hình 1.22. Cấu hình ngắt cho thiết bị ngoại vi

Chú ý: Mỗi nguồn ngắt có một bit cho phép bên trong NVIC và khối ngoại vi tương ứng.

Có 16 thanh ghi cài đặt mức ưu tiên ngắt. Mỗi thanh ghi được chia thành bốn trường có độ rộng là 8-bit để cấu hình mức ưu tiên, mỗi trường đó được chỉ định cho một vector ngắt nhất định. STM32 chỉ sử dụng một nửa

của trường này (4-bit có trọng số cao nhất) để thực hiện 16 mức ưu tiên ngắt. Mặc định các trường này xác định 16 mức độ ưu tiên với mức độ 0 là cao nhất và 15 là thấp nhất. Ngoài ra có thể sắp sếp các trường ưu tiên thành các nhóm (group) và nhóm con (subgroup). Điều này không tạo thêm bất kì mức ưu tiên nào, nhưng giúp chúng ta dễ quản lý các mức ưu tiên khi chương trình ứng dụng có một số lượng lớn các ngắt bằng cách lập trình trường PRIGROUP trong thanh ghi điều khiển reset và ngắt ở mức ứng dụng.



Hình 1.23. Thanh ghi điều khiển reset và ngắt ở mức ứng dụng

PRIGROUP (3 Bits)	Binary Point (group.sub)		Preemting Priority (Group Priority)		Sub-Priority	
			Bits	Levels	Bits	Levels
011	4.0	Gggg	4	16	0	0
100	3.1	Gggs	3	8	1	2
101	2.2	Ggss	2	4	2	4
110	1.3	Gsss	1	2	3	8
111	0.4	Ssss	0	0	4	16

Hình1.24. Cấu hình mức ưu tiên thành các group và subgroup

Trường PRIGROUP gồm 3-bit cho phép chia trường 4-bit trong các thanh ghi cài đặt mức ưu tiên thành các nhóm và nhóm con. Ví dụ, trị giá của PRIGROUP là 5 sẽ tạo ra hai nhóm, mỗi nhóm với 4 mức độ ưu tiên.

Trong chương trình ứng dụng , chúng ta có thể xác định một nhóm các ngắt có mức ưu tiên cao và một nhóm có mức ưu tiên thấp. Bên trong mỗi nhóm chúng ta có thể xác định các mức cho nhóm con như mức thấp, trung bình, cao và rất cao. Như đã đề cập ở trên việc phân nhóm sẽ không tạo ra thêm mức ưu tiên nào nhưng cung cấp một cái nhìn trừu tượng về cấu trúc ngắt, điều này hữu ích cho người lập trình khi quản lý một số lượng lớn các ngắt. 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:

```
DCD 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;
```

1.5 Các chế độ năng lượng

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.

1.5.1 Cách đi vào chế độ năng lượng thấp của CPU Cortex

Lõi Cortex có thể được đặt vào chế độ sleep của mình 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. Điều này cho phép một ứng dụng năng lượng thấp (trạng thái hệ thống luôn ở chế độ sleep khi không có sự kiện nào xảy ra) sẽ hoàn toàn được điều khiển bằng ngắt, để lõi Cortex sẽ được đánh thức bởi một sự kiện (từ ngắt bên trong hoặc bên ngoài CPU Cortex), chỉ cần thực thi một đoạn mã thích hợp và sau đó lại đi vào chế độ sleep, như vậy với một mã chương trình tối thiểu chúng ta có thể quản lý hiệu quả năng lượng của hệ thống.

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. Điều này cho phép một thiết bị ngoại vi có thể báo để đáng thức lõi Cortex và tiếp tục thực thi chương trình ứng dụng mà không cần một trình phục vụ ngắt nào. Các lệnh WFI và WFE không thể gọi trực tiếp từ ngôn ngữ C, tuy nhiên thuận lợi là trình biên dịch cho tập lệnh Thumb-2 cung cấp sẵn các macro để có thể được sử dụng như một lệnh C chuẩn (inline C command):

__WFI

__WFE

Ngoài các chế độ năng lượng thấp **SLEEPNOW** và **SLEEPONEXIT**, lõi Cortex có thể phát ra một tín hiệu **SLEEPDEEP** cho phần còn lại của hệ thống vi điều khiển.

Hình 1.25. Thanh ghi điều khiển hệ thống dùng để cấu hình
các chế độ ngủ của vi xử lí Cortex

Điều này cho phép các khối chức năng như PLL (Phase Loop Lock) và thiết bị ngoại vi có thể ngừng hoạt động, để STM32 có thể đi vào chế độ năng

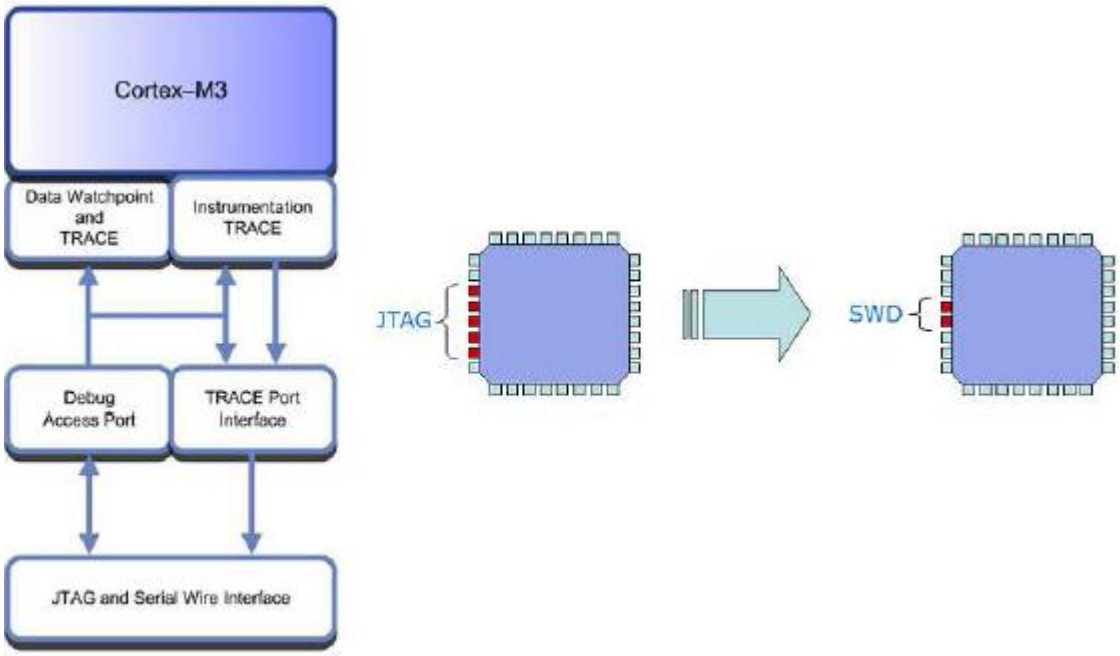
lượng thấp nhất của nó.

1.5.2 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 bản (thiết lập chạy từng bước và các breakpoint v.v…) cũng như có thể xem nội dung của các vị trí trong bộ nhớ. Ngoài ra CPU ARM7 và ARM9 còn có thể cung cấp một bộ theo dõi thời gian thực (real-time trace) thông qua một thiết bị ngoại vi gỡ lỗi được gọi là ETM (embedded trace macro cell). Trong khi hệ thống gỡ lỗi này hoạt động tốt, thì nó bộc lộ một số hạn chế. JTAG chỉ có thể cung cấp thông tin gỡ lỗi cho công cụ phát triển (như Keil, IAR…) khi CPU ARM dừng lại, do đó không có khả năng cập nhật thời gian thực. Ngoài ra, số lượng của breakpoints phần cứng được giới hạn tới hai điểm, mặc dù tập lệnh ARM7 và ARM9 hỗ trợ một lệnh breakpoint, có thể được chèn vào mã chương trình bằng công cụ phát triển (gọi là soft breakpoints). Tương tự vời JTAG, bộ theo dõi thời gian thực-ETM phải được trang bị bởi các nhà sản xuất với chi phí bổ sung. Do vậy ETM không phải lúc nào cũng được hỗ trợ. Với lõi Cortex mới, toàn bộ hệ thống gỡ lỗi gọi là CoreSight đã được giới thiệu.

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.

Hệ thống gỡ lỗi CoreSight có một cổng truy cập gỡ lỗi cho phép kết nối với vi điều khiển bằng công cụ JTAG. Công cụ gỡ lỗi có thể kết nối bằng cách sử dụng chuẩn giao diện JTAG 5 chân hoặc giao diện 2 dây nối tiếp. Ngoài các tính năng gỡ lỗi của JTAG, CoreSight có chứa một theo dõi dữ liệu và một ETM.



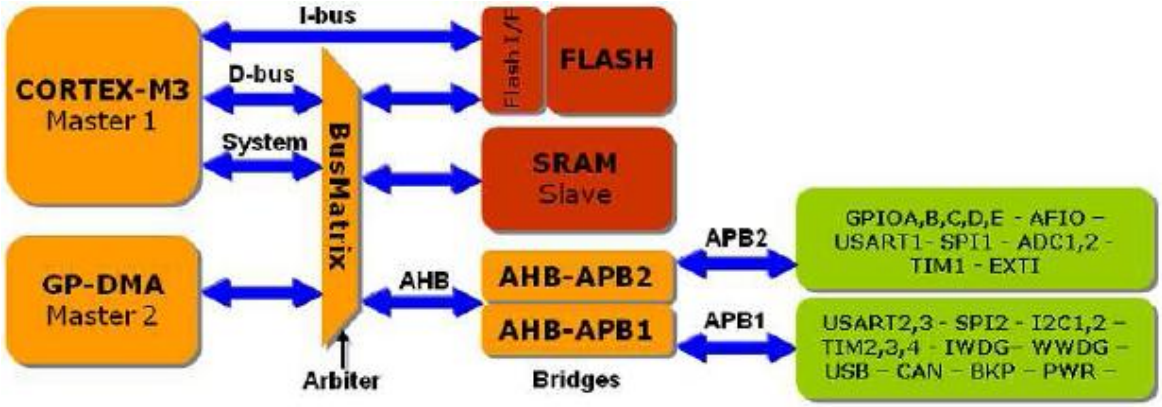
Hình 1.26. Hệ thống gỡ lỗi CoreSight bên trong Cortex

Trong thực tế, cơ cấu gỡ lỗi CoreSight trên STM32 cung cấp một phiên bản thời gian thực được cải tiến của chuẩn gỡ lỗi JTAG. Hệ thống gỡ lỗi STM32 CoreSight cung cấp 8 breakpoints phần cứng có thể được đặt và xóa trong khi CPU Cortex đang chạy. Ngoài ra bộ theo dõi Data Watch cho phép bạn xem các nội dung của các vị trí nhớ trong khi CPU Cortex đang chạy. Hệ thống CoreSight có thể duy trì ở trạng thái hoạt động khi lõi Cortex đi vào chế độ ngủ. Ngoài ra các timer của STM32 có thể được tạm dừng khi hệ thống CoreSight tạm dừng CPU. Điều này cho phép chúng ta thực thi từng bước mã chương trình và giữ cho timer đồng bộ với hệ thống. Với các lệnh thực thi trên CPU Cortex, CoreSight cải thiện đáng kể khả năng gỡ lỗi thời gian thực của STM32 so với CPU ARM7 và ARM9 trước kia, trong khi vẫn sử dụng cùng một phần cứng chi phí thấp.

Chương 2

KIẾN TRÚC HỆ THỐNG CỦA ARM CORTEX

ARM Cortex 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). SRAM nội kết nối với AHB và đóng vai trò là bộ DMA. Các thiết bị ngoại vi được kết nối bằng 2 hệ thống bus ngoại vi tốc độ cao (APB-ARM Advanced Peripheral Busses). Các bus APBs thông qua các bus cầu nối AHB-APBs kết nối vào hệ thống AHB. Ma trận bus AHB sử dụng xung nhịp đồng hồ bằng với xung nhịp của nhân Cortex. Tuy nhiên thông qua bộ chia tần số AHB có thể hoạt động ở tần số thấp hơn nhằm tiết kiệm năng lượng.

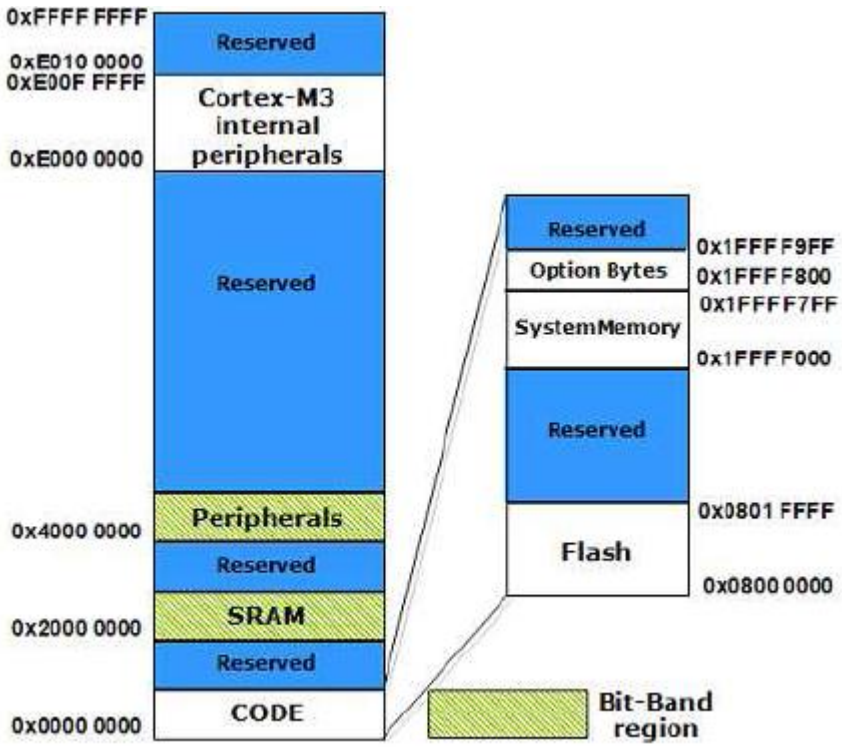


Hình 2.1 Hệ thống Bus nội

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 và bộ điều khiển DMA truy cập tài nguyên trên vi xử lý.

2.1 Cấu trúc bộ nhớ

Bên cạnh hệ thống bus nội đa dạng STM32 còn cung cấp 4Gbytes không gian bộ nhớ liên tục dành cho lập trình. Bộ nhớ được bắt đầu từ địa chỉ 0x00000000 .On-chip SRAM bắt đầu từ địa chỉ 0x20000000 và tất cả SRAM nội đều được bố trí ở điểm bắt đầu vùng bit band. Vùng nhớ thiết bị ngoại vi được ánh xạ từ địa chỉ 0x40000000 và ở vùng bit band. Các thanh ghi điều khiển của nhân Cortex được ánh xạ từ địa chỉ 0xE0000000.

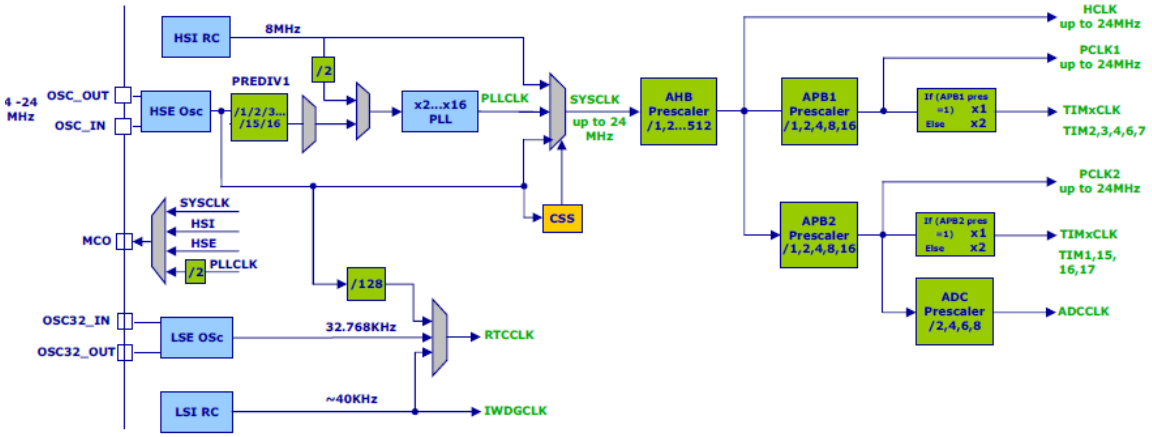


Hình 2.2 Cấu trúc bộ nhớ

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 0x1FFFFF80 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.

2.2 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. Bộ xung nhịp tốc độ thấp này thường được dùng cho đồng hồ thời gian thực và watchdog.

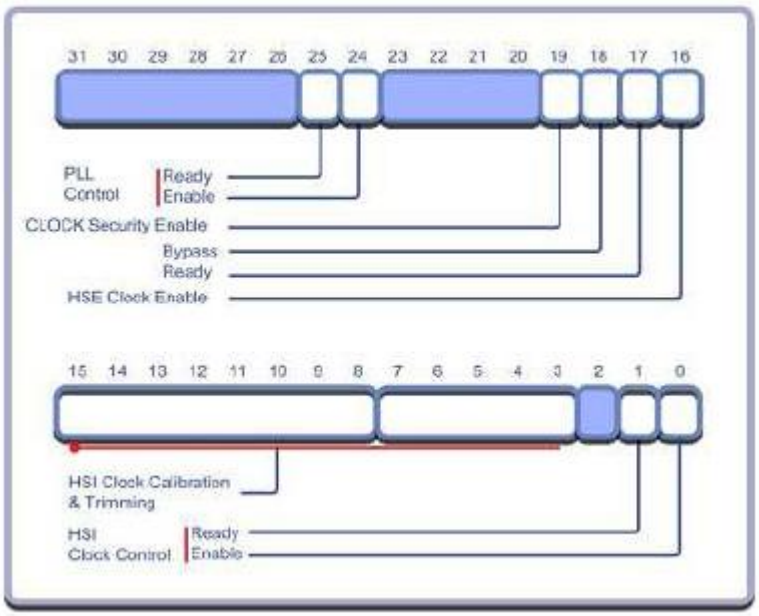


Hình 2.3 STM32 bao gồm 2 bộ tạo xung nhịp nội và 2 bộ tạo xung nhịp ngoại thêm vào đó là bộ vòng khóa pha(Phase Lock Loop-PLL).

Nhân Cortex có thể được cấp xung nhịp từ bộ tạo dao động nội và ngoại, đồng thời từ PLL nội. Như trên hình 2.3, PLL có thể lây dao động từ bộ tạo dao động tốc độ cao nội và ngoại. 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 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).

2.2.1 Vòng Khóa Pha (Phase Lock Loop)

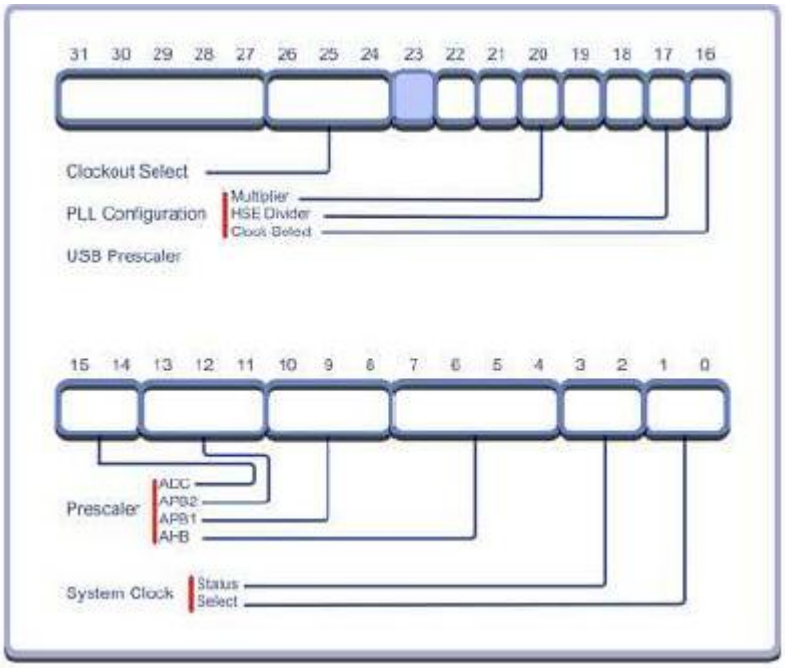
Sau khi hệ thống reset STM32 nhận xung nhịp từ bộ tạo dao động HIS. Tại thời điểm đó các bộ tạo dao động ngoại sẽ bị tắt. Bước đầu tiên để STM32 hoạt động ở mức xung nhịp cao nhất là bật bộ tạo dao động HSE và chờ cho đến khi đi vào hoạt động ổn định.



Đoạn mã sau mô tả cách cấu hình để CPU của STM32 hoạt động ở mức xung nhịp cao nhất

```
RCC->CR |= 0x10000; //HSE on
//wait until HSE stable
While(!(RCC->CR & 0x0020000))
{};
```

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 đinh, 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. Khi bộ tạo dao động ngoại và PLL hoạt động ổn định, bit điều khiển trạng thái sẽ bật lên, khi đó dao động được tạo bởi PLL sẽ được cấp cho nhân CPU Cortex của STM32.



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;
```

2.2.2 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;
```

2.2.3 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. Tuy nhiên khi PLL được kích hoạt và sử dụng để tạo dao động cho CPU, thời gian truy cập vào Flash rất chậm khoảng 35ns, điều này làm giảm hiệu năng của hệ thống. Để Cortex CPU hoạt động ở xung nhịp cao nhất 72MHz với thời gian ở trạng thái chờ là 0 bộ nhớ Flash được trang bị bộ 2 nhớ đệm 64-bit. Hai bộ nhớ đệm này có thể thực thi các lệnh đọc ghi dữ liệu 64-bit trên Flash và chuyển các lệnh 16 hay 32 bit cho nhân Cortex để thực thi. Kỹ thuật này hoạt động tốt đối với các lệnh thuộc tập lệnh Thumb-2 và các tập lệnh có khả năng dự báo chỉ dẫn(Branch Prediction) của Cortex pipeline. Hệ thống bộ đệm Flash được quản

lý bởi các thanh ghi cấu hình Flash. Cùng với việc kích hoạt bộ đệm tiền xử lý,chúng ta phải điều chỉnh số trạng thái chờ khi Flash đọc 8 bytes lệnh từ bộ nhớ Flash. Độ trễ được thiết lập như sau:

0< SYSCLK <24MHz 0 waitstate

24< SYSCLK <48MHz 1 waitstate

48<SYSCLK <72MHz 2 waitstate

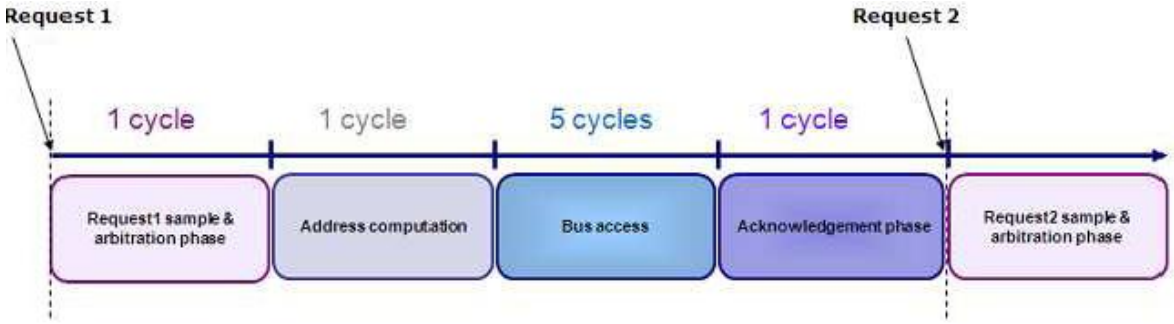
Thời gian trạng thái chờ này giữa bộ đệm tiền xử lý với bộ nhớ Flash không tác

động đến nhân Cortex CPU. Khi CPU đang thực thi các lệnh ở nửa đầu của bộ

đệm thì các lệnh ở nửa sau của bộ đệm sẽ được tiền xử lý và tải lên nhân để sử lý ngay tiếp theo, điều này làm tối ưu hóa hiệu năng xử lý của Cortex CPU.

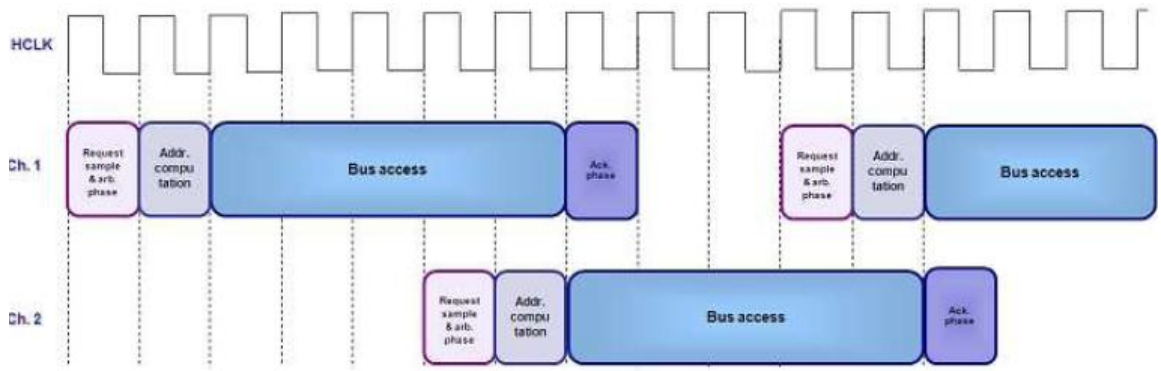
2.2.4 Direct Memory Access

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. Trong trường hợp trao đổi dữ liệu giữa bộ nhớ và bộ nhớ, tốc độ dữ liệu phụ thuộc tốc độ của kênh DMA quản lý nó. Còn với giao tiếp dữ liệu với ngoại vi, thì tốc độ phụ thuộc vào bộ điều khiển của ngoại vi đó và hướng dữ liệu di chuyển. Cùng với chuyển dữ liệu theo luồng, bộ DMA của STM32 còn hỗ trợ bộ đệm vòng. Vì hầu hết các ngoại vi hiện nay không có bộ nhớ FIFO, mỗi bộ DMA sẽ lưu dữ liệu vào trong bộ nhớ SRAM. Bộ DMA của STM32 được thiết kế dành cho truyền các loại dữ liệu tốc độ cao và nhỏ.



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

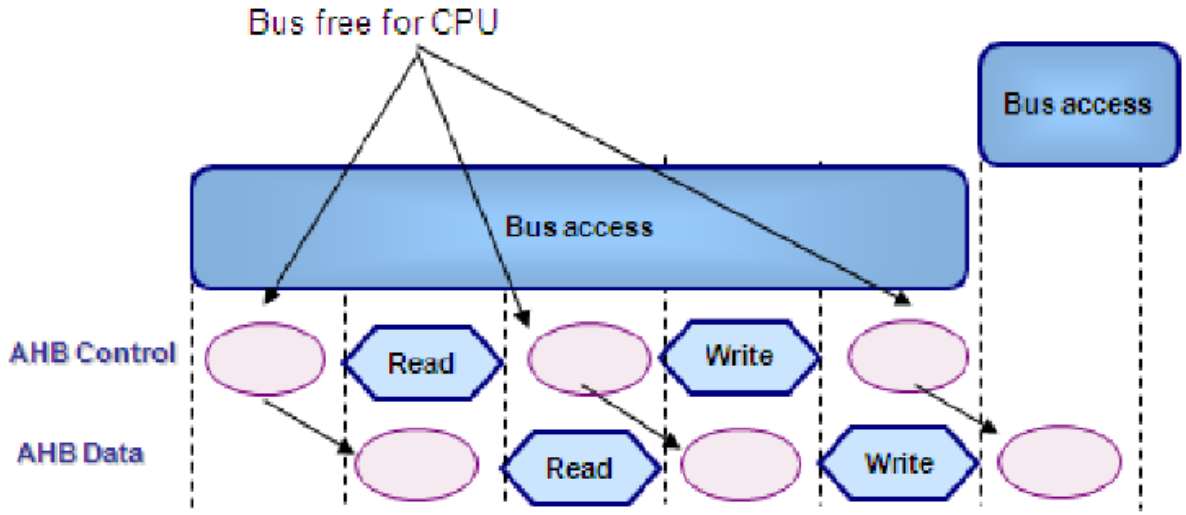
Quá trình truyền dữ liệu gồm 4 giai đoạn: lấy mẫu và phân xử, tính toán địa chỉ, truy cập đường truyền, và cuối cùng là hoàn tất. Mỗi giai đoạn thực hiện trong 1 chu kỳ lệnh, riêng truy cập đường truyền mất 5 chu kỳ lệnh. Ở giai đoạn truy cập đường truyền thực chất là giai đoan dữ liệu được truyền, mỗi từ (word) sẽ mất 3 chu kỳ lệnh. Bộ DMA và CPU đươc thiết kế để cùng lúc có thể hoạt động mà không tranh chấp tài nguyên lẫn nhau. Giữa 2 kênh DMA khác nhau, sẽ có sự ưu tiên mức hoạt động, dựa trên đó bộ phân xử sẽ quyết định kênh DMA có mức ưu tiên cao hơn sẽ được lấy tài nguyên trước. Nếu 2 kênh DMA có cùng mức ưu tiên, lại đang ở trạng thái chờ để truy cập tài nguyên, thì kênh DMA có số thứ tự nhỏ hơn sẽ được sử dụng tài nguyên trước.



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 hiên việc phân xử tài nguyên và tính toán địa chỉ trong khi bộ DMA khác đang ở giai đoạn truy cập đường truyên 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 truyên dữ liệu. Điều này vừa làm tăng tốc độ truyền dữ liệu, tối đa hóa viêc sử dụng tài nguyên.



Ở giai đoạn Bus Access CPU sẽ có 3 chu kỳ rảnh. Khi chuyển dữ liệu từ vùng nhớ sang vùng nhớ điều này sẽ đảm bảo nhân Cortex-M3 sử dụng 60% dung lượng của đường truyền dữ liệu cho dù bộ DMA vẫn hoạt động liên tục.

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

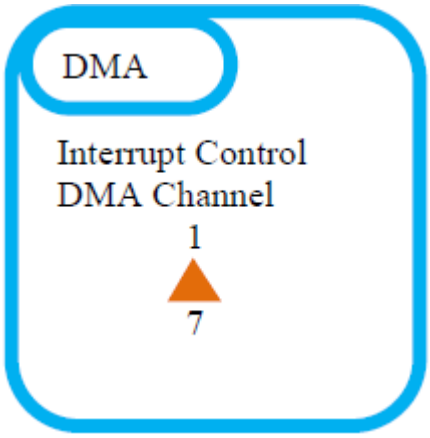
CPU để chuyển 2 bytes dữ liệu. Trong đó 1 chu kỳ để đọc và 1 chu kỳ để ghi, 3 chu kỳ còn lại được bố trí xen kẽ nhằm giải phóng đường dữ liệu cho nhân Cortex.

Điều đó có nghĩa là bộ DMA chỉ sử dụng tối đa 40% băng thông của đường dữ liệu. Tuy nhiên giai đoạn Bus Access hơi phức tạp ở trường hợp dữ liệu truyền giữa thiết bị ngoại vi hoặc giữa ngoại vi và bộ nhớ do liên quan đến AHB và APB. Trao đổi trên bus AHB sử dụng 2 chu kỳ xung nhịp của AHB, trên bus APB sẽ sử dụng 2 chu kỳ xung nhịp của APB cộng thêm 2 chu kỳ xung nhịp của AHB. Mỗi lần trao đổi dữ liệu, bộ DMA sẽ sử dụng bus AHB, bus APB và 1 chu kỳ xung nhịp AHB. Ví dụ để chuyển dữ liệu từ bus SPI tới SRAM chúng ta sẽ sử dụng:

SPI đến SRAM sử dụng DMA = SPI transfer(APB) + SRAM transfer(AHB) + free cycle(AHB)

= (2 APB cycles + 2 AHB cycles) + (2 AHB cycles) + (1 AHB cycle) = (2 APB cycles) + (5 AHB cycles)

* Lưu ý: Quá trình trên chỉ áp dụng cho các nhân Cortex sử dụng đường I-bus để nạp lệnh cho nhân xử lý.



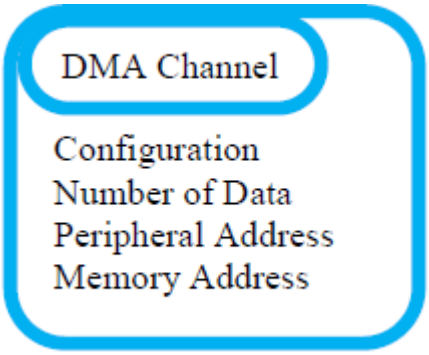
STM32 có 7 bộ DMA độc lập với nhau

Việc sử dụng DMA rất đơn giản. Đầu tiên là kích hoạt đồng hồ xung nhịp

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

Một khi được cấp nguồn khối DMA sẽ được điều khiển bởi 4 thanh ghi điền khiển. 2 thanh ghi điều khiển địa chỉ đích và nguồn của ngoại vi và vùng

nhớ. Kích thước dữ liệu truyền và cấu hình tổng quan DMA được lưu trong 2 thanh ghi còn lại.



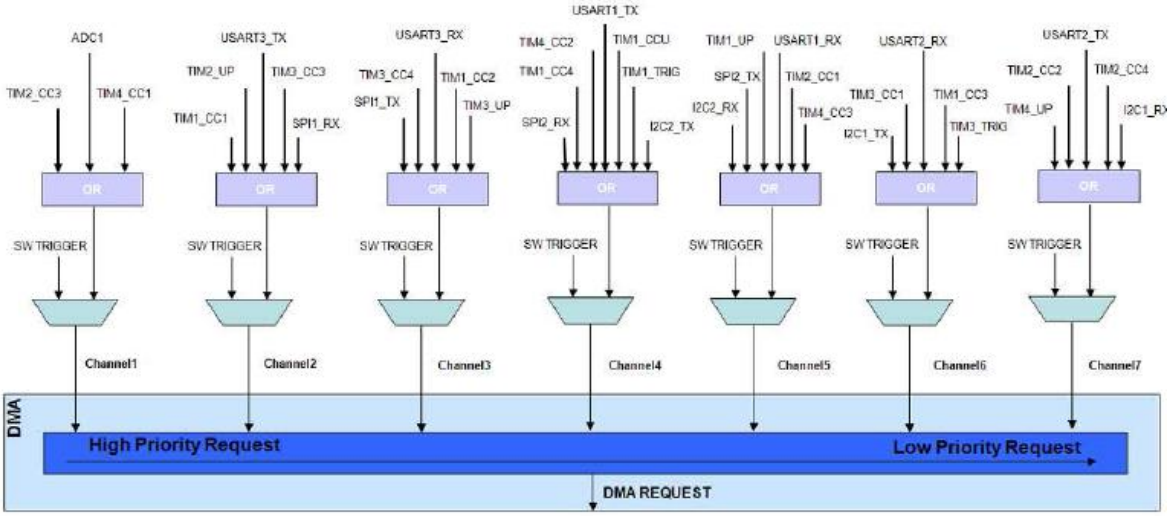
Mỗi bộ DMA có 4 thanh ghi điều khiển, 3 nguồn tín hiệu interrupt: hoàn tất, hoàn tất một nửa, lỗi.

Mỗi kênh DMA có thể được gắn với một mức ưu tiên: rất cao, cao, trung bình
và thấp. Kích cỡ của dữ liệu được truyền có thể điều chỉnh để phù hợp cho ngoại vi và vùng nhớ. Ngoài việc sử dụng DMA với chế độ vòng lặp chờ, chúng ta có thể dùng ngắt để theo dõi quá trình chuyển dữ liệu. 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. Ví dụ sau mô tả quá trình chuyển dữ liệu giữa 2 vùng nhớ trên SRAM:


```
DMA_Channel1->CCR = 0x00007AC0; //cấu hình mem to mem
DMA_Channel1->CPAR = (unsigned int)src_array;//địa chỉ nguồn
DMA_Channel1->CMAR=(unsigned int)dst_array;//địa chỉ đích
DMA_Channel1->CNDTR=0x000A;//số lượng dữ liệu
TIM2->CR1 = 1;              //khởi tạo thời gian
DMA_Channel1->CCR |= 0x00000001;//kích hoạt quá trình chuyển dữ liệu
While( !(DMA->ISR & 0x00000001); //chờ cho đến khi hoàn thành
TIM2->CR1 = 0; //ngưng đếm
TIM2->CNT = 0; // thiết lập giá trị đếm bằng 0
TIM2->CR1 = 1; //bắt đầu đếm lại
for(index = 0;index < 0x000A;index++)
        dst_array[index]=src_array[index];
TIM2-CR1 = 0;
```

Ở đoạn mã trên, ta sử dụng TIM2 để đo thời gian (tính theo chu kỳ) chuyển dữ liệu từ 2 vùng nhớ kích thước 10 word. Với DMA quá trình chuyển tiêu tốn

220 chu kỳ, với cách sử dụng CPU tiêu tốn 536 chu kỳ.



Hình 3.4 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.

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. Để sử dụng DMA, đầu tiên ta khởi tạo thiết bị ngoại vi và kích hoạt chế độ DMA trên thiết bị ngoại vi đó, sau đó khởi tạo kênh DMA tương ứng.

Chương 3

NGOẠI VI

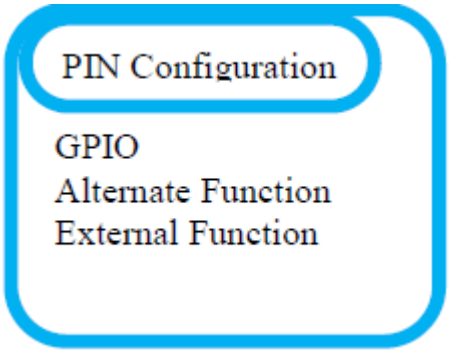
Chương này sẽ giới thiệu các thiết bị ngoại vi trên các phiên bản ARM Cortex STM32. Gồm 2 loại: ngoại vi đa dụng và ngoại vi giao tiếp. Tất cả ngoại vi trên STM32 được thiết kế và dựa trên bộ DMA. Mỗi ngoại vi đều có phần điều khiển mở rộng nhằm tiết kiệm thời gian xử lý của CPU.

3.1 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".

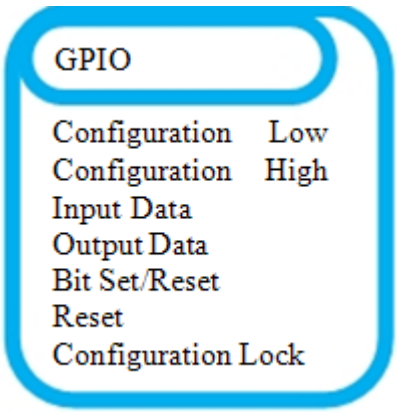
3.1.1 Các cổng I/O đa dụng

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



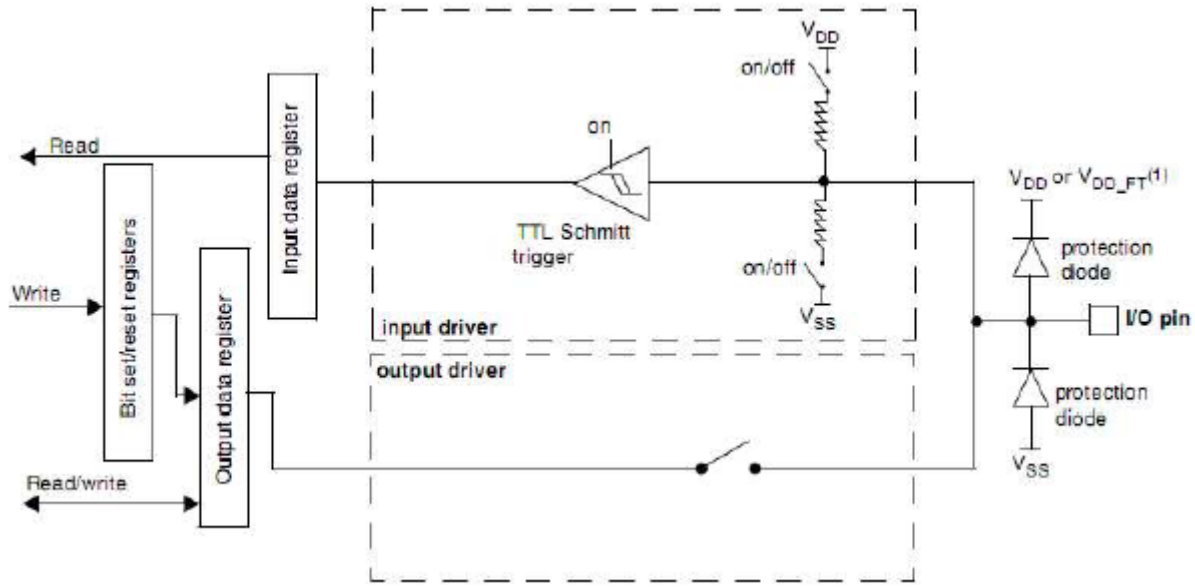
Mỗi chân điều khiển có thể cấu hình như là GPIO hoặc có chức năng thay thế khác. Hoặc mỗi chân có thể cùng lúc là nguồn ngắt ngoại.

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.

Configuration	CNF1	CNF0	MOD1	MOD0
Analog Input	0	0	00	
Input Floating(Reset state)	0	1		
Input Pull-up	1	0		
Input Pull-down	1	0		
Output Push-Pull	0	0	00:Reserved 01:10Mhz 10:2Mhz 11:50Mhz	
Output Open-drain	0	1		
AF Push-Pull	1	0		
AF Open-drain	1	1		



Hình 3.1 Cấu trúc cổng I/O

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

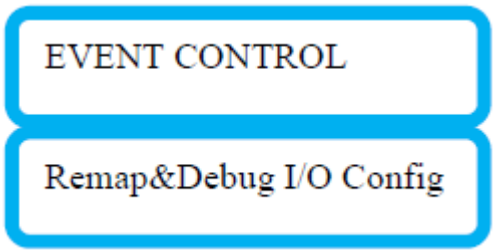
```
tmp = GPIOx->LCKR;
tmp = GPIOx->LCKR;
```

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.

3.1.1.1 Chức năng thay thế (Alternate Function)

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.

3.1.1.2 Event Out

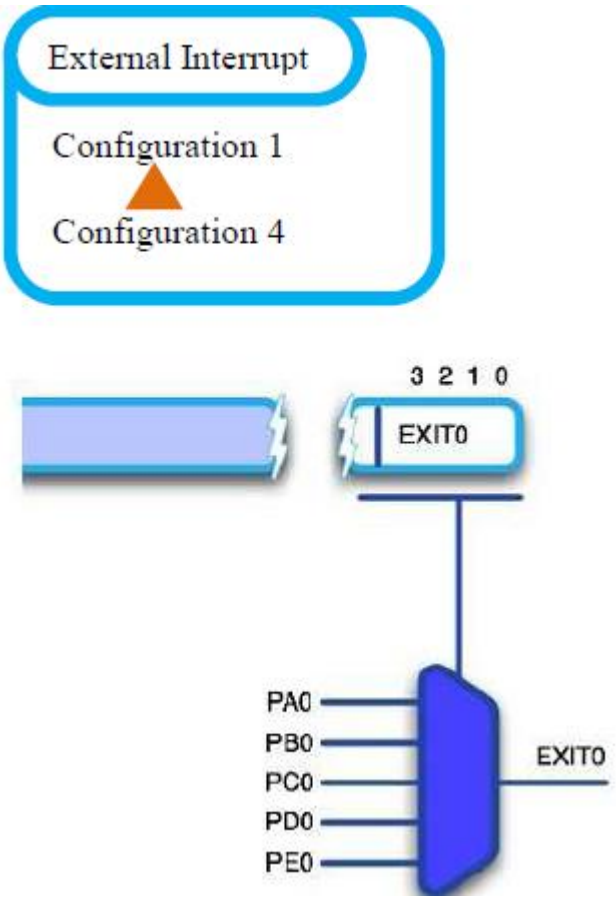
Nhân Cortex có khả năng tạo xung nhịp để "đánh thức" các khối vi điều khiển bên ngoài thoát khỏi trạng thái tiết kiệm năng lượng. Thông thường, xung nhịp này sẽ được nối với chân "Wake up" của vi xử lý STM32 khác. Lệnh SEV Thumb-2 khi được thực thi sẽ tạo ra xung nhịp "Wake up" này. Thanh ghi điều khiển sự kiện của STM32 cấu hình chân GPI nào sẽ xuất xung nhịp "Wake up".

3.1.2. Ngắt ngoại (EXTI)

Bộ điều khiển ngắt ngoại có 19 ngắt và kết nối vào bảng vector ngắt thông qua bộ NVIC. 16 ngắt được kết nối thông qua các chân của cổng GPIO và tạo ngắt khi phát khi có xung lên(rasing) hoặc xuống (falling) hoặc cả hai. 3 ngắt còn lại được nối với "RTC alarm", "USB wake up" và "Power voltage detect".

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. EXTI có thể tạo ra các ngắt để thoát ra khỏi sự kiện Wait của chế độ Interrupt và thoát khỏi sự kiện Wait của chế độ Event.

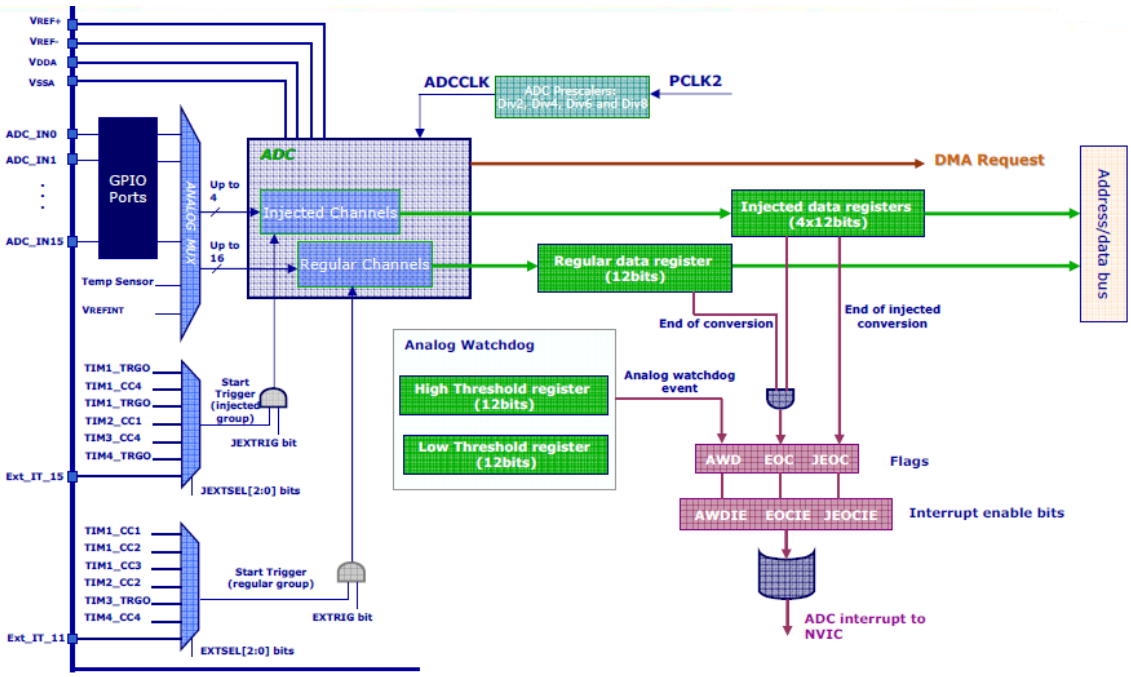


Hình 3.2 Ngắt ngoại

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.

3.1.3 ADC

STM32 có thể có 2 bộ chuyển đổi tín hiệu tương tự sang tín hiệu số tùy vào các phiên bản.

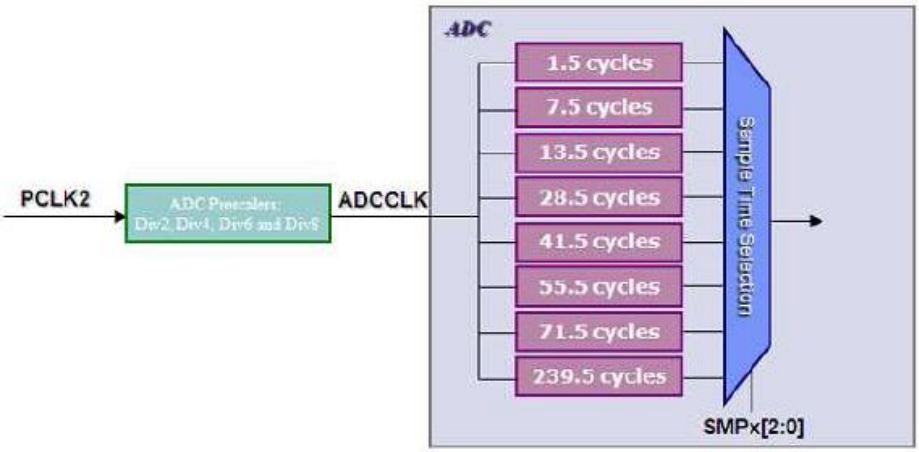


Hình 3.3 Mạch ADC trong STM32

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.

3.1.3.1 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.

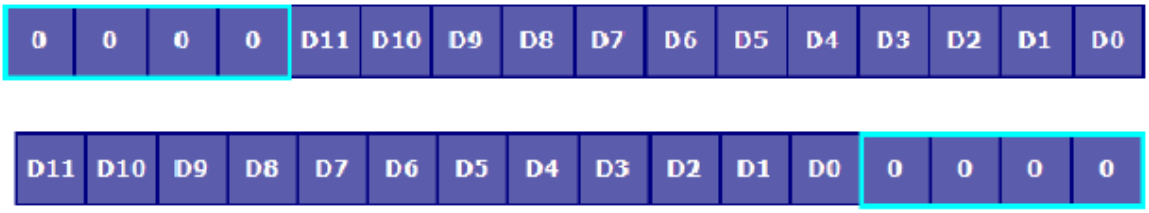


Hình 3.4 Có 8 mức thời gian chuyển đổi

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. Một khi được kích hoạt, chế độ Regular có thực thi chuyển đổi liên tục(continuos convertion) hoặc không liên tục.

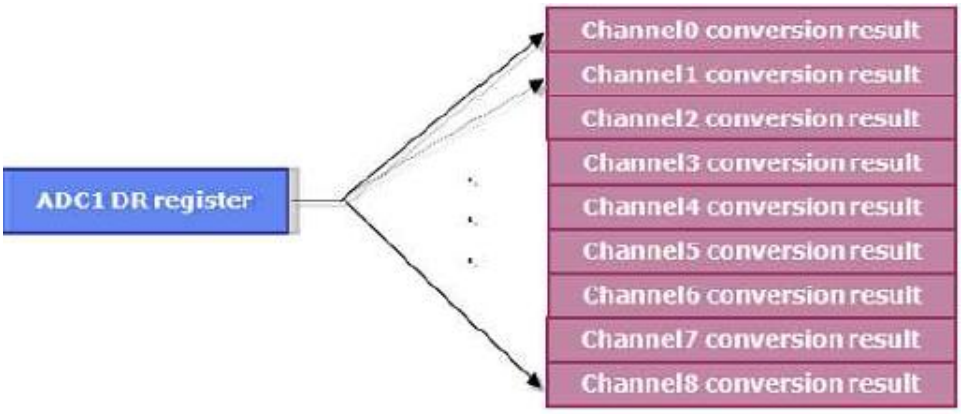
Một nhóm kênh hoạt động ở chế độ Regular có thể liên tục thực hiện quá trình chuyển đổi, hoặc chỉ chuyển đổi khi nhận tín hiệu kích hoạt.

Khi một nhóm các kênh hoàn thành việc chuyển đổi, kết quả được lưu vào thanh ghi kết quả và tín hiệu ngắt được tạo. Vì bộ ADC có độ phân giải là 12 bit và được lưu trong thanh ghi 16 bit do đó dữ liệu có thể được "canh lề" trái hoặc phải.



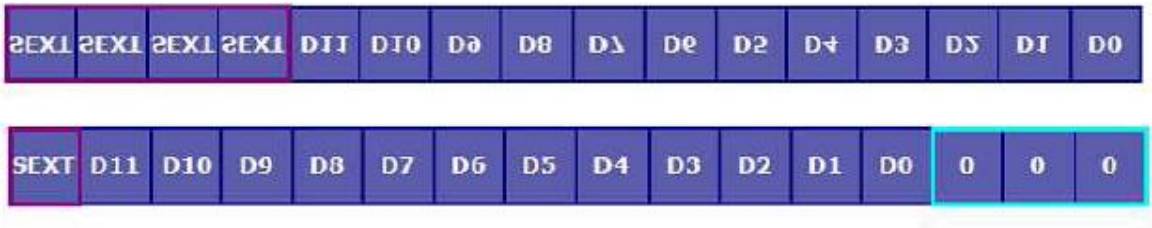
Dữ liệu có thể được canh lề trái hoặc phải trong thanh ghi kết quả

Bộ ADC1 có riêng kênh DMA để chuyển dữ liệu từ thanh ghi kết quả sang vùng nhớ. Với phương pháp này, dữ liệu từ kết quả chuyển đổi của một nhóm các kênh ADC sẽ được chuyển toàn bộ lên vùng nhớ ngay trước khi ngắt được phát sinh.



ADC1 sử dụng DMA chuyển dữ liệu kết quả của một nhóm các kênh vào vùng nhớ được khởi tạo trên SRAM

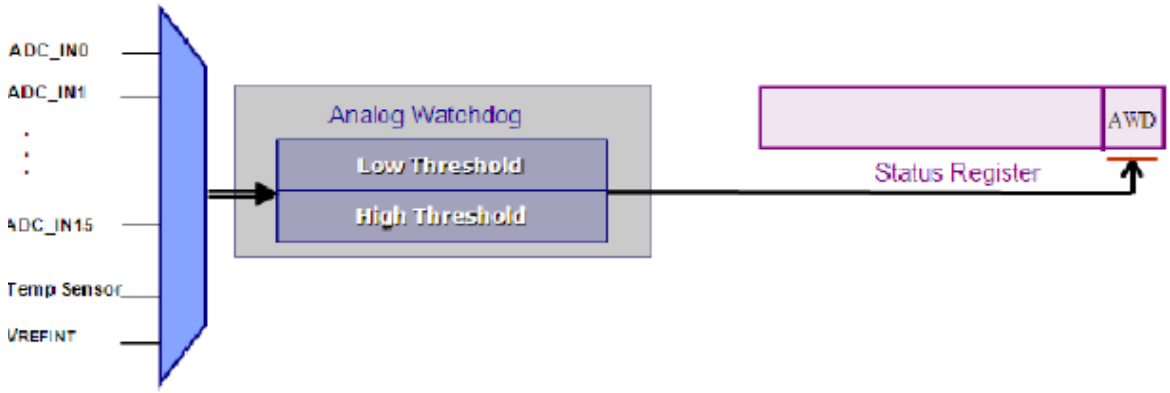
Loại ADC thứ 2 là Injected ADC. Injected ADC là dãy các kênh ADC, tối đa là 4 kênh. Injected ADC có thể được kích hoạt bằng phần mềm hoặc tín hiệu phần cứng. Khi được kích hoạt, Injected ADC với mức ưu tiên cao hơn sẽ tạm ngưng các kênh Regular ADC đang hoạt động. Các kênh Regular ADC chỉ tiếp tục hoạt động sau khi Injected ADC thực thi xong. Về cấu hình hoạt động của Injected tương tự như của Regular, tuy nhiên mỗi kênh chuyển đổi của Injected có thanh ghi dữ liệu ADC_JDRx tương ứng.



Tương tự như Regular ADC, dữ liệu ở thanh ghi ADC_JDRx có thể được canh lề trái hoặc phải, kèm theo đó là dấu nếu dữ liệu âm

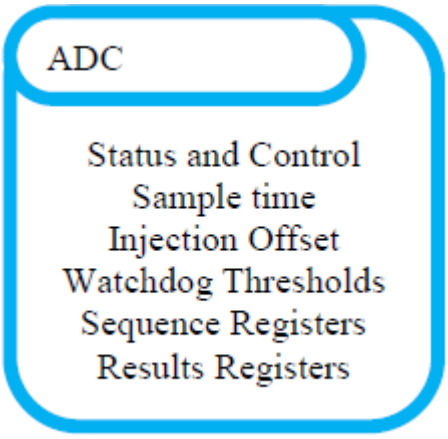
3.1.3.2 Analogue WatchDog

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. Ngoài việc giám sát tín hiệu điện áp thông thường, Analogue Watchdog có thể được dùng để phát hiện điện áp khác 0 V.



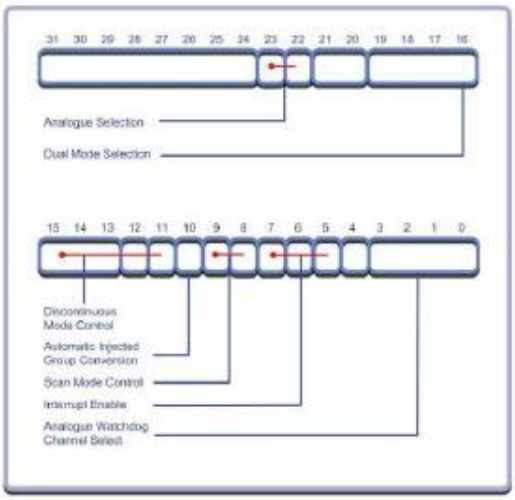
Hình 3.5 Analogue Watchdog có thể dùng giám sát một hay nhiều kênh ADC với vùng ngưỡng được cấu hình bởi người dùng

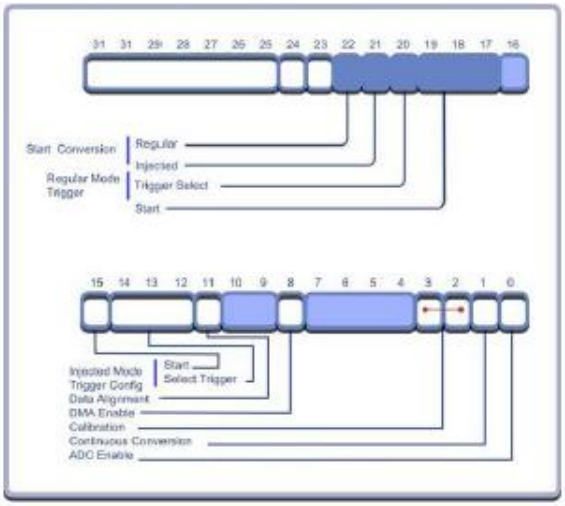
3.1.3.3 Cấu hình ADC



Các thanh ghi của khối ADC được tách ra thành 6 nhóm thanh ghi, trong đó các thanh ghi Status và Control xác định chế độ hoạt động của ADC.

Có hai thanh ghi điều khiển ADC_CR1 và ADC_CR2 để 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;
```

Ở hàm xử lý ngắt 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 ngắt

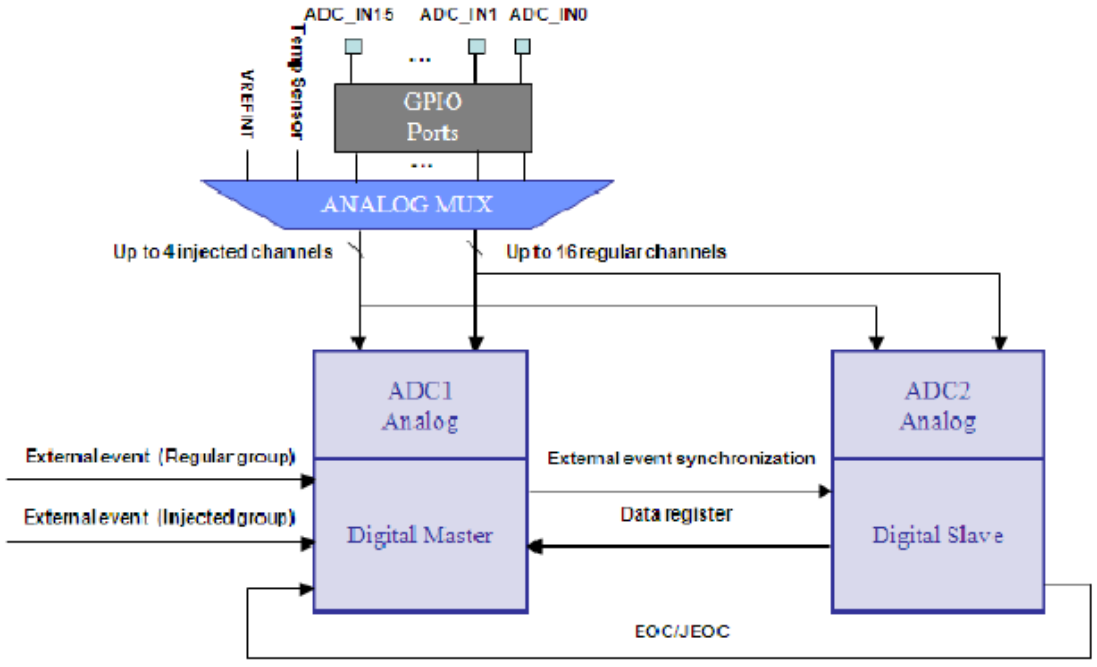
```
DMA_Channel1->CCR = 0x00003A28; //Circular mode, peripheral and memory increased disable
//Load destination address into peripheral register, GPIO port data register
DMA_Channel1->CMAR = (unsigned int) 0x04001244C;
DMA_Channel1->CNDTR = 0x0001; //number of words to transfer
DMA_Channel1->CCR = 0x00000001;//Enable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

Chúng ta kích hoạt chế độ DMA của khối ADC

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

3.1.3.4. Dual mode

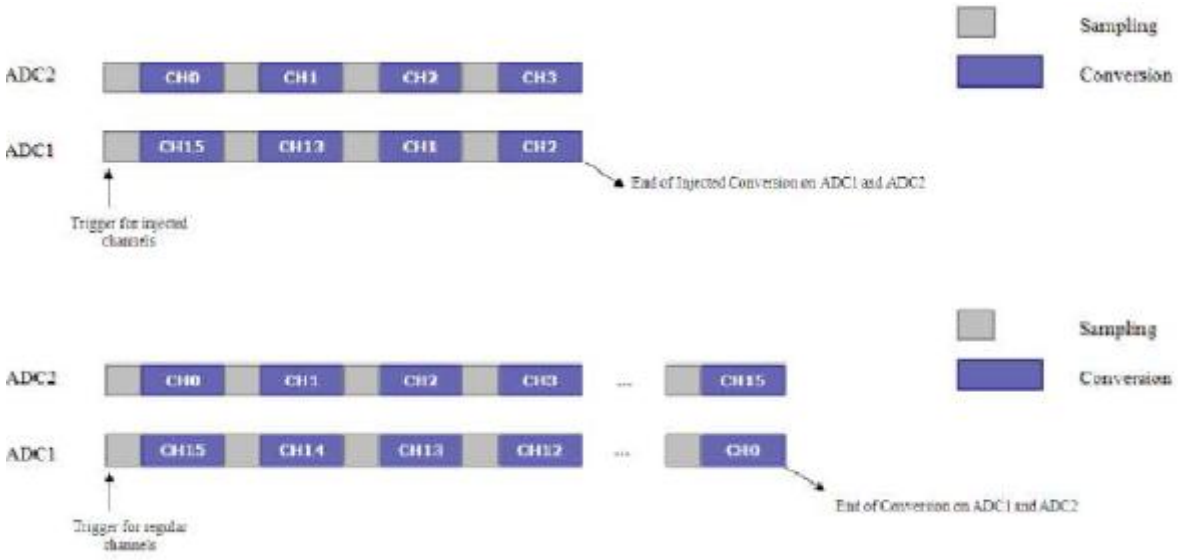
Ở một số phiên bản, ST cung cấp 2 khối ADC nhằm đáp ứng các tác vụ phức tạp hơn



Hình 3.6 Phiên bản có 2 khối ADC

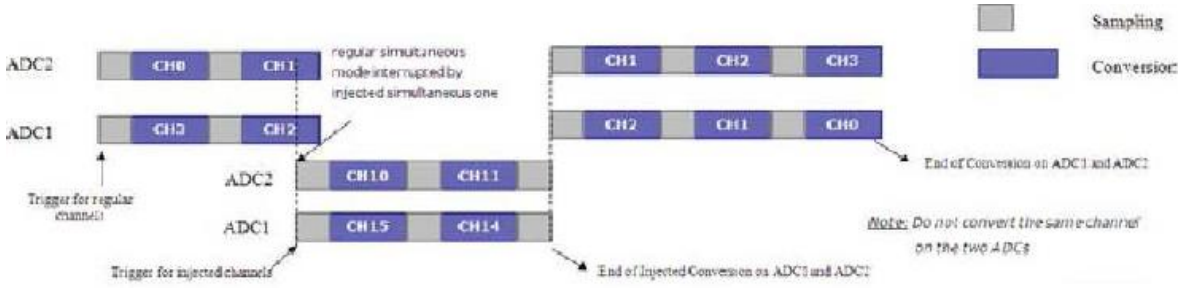
Khi hoạt động ở chế độ Dual, khối ADC2 đóng vai trò phụ đối với ADC1. Khi kết hợp ADC1 và ADC2, chúng ta sẽ có 8 chế độ hoạt động

3.1.4.1. 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.

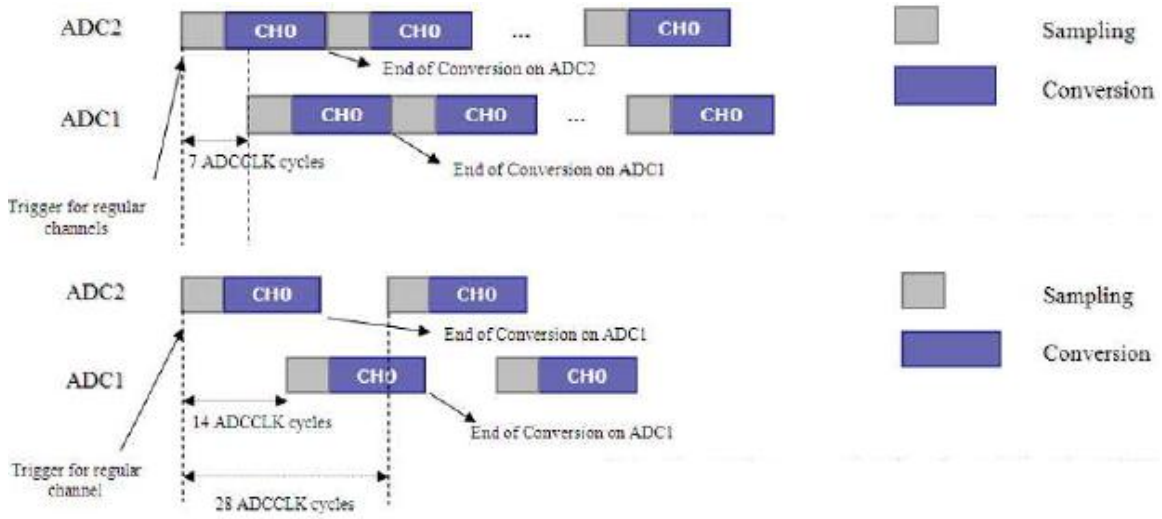
3.1.4.2. Cả hai khối cùng hoạt động ở 2 chế độ Regular và Injected xen kẽ



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.

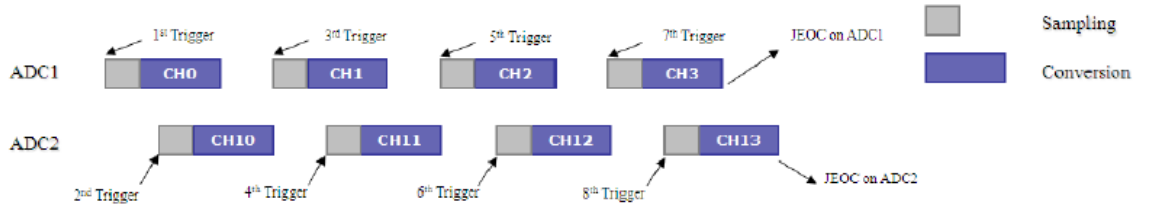
3.1.4.3. Hoạt động xen kẽ nhanh và chậm Regular



Ở chế độ xen kẽ nhanh, một kênh có thể liên tục chuyển đổi bởi hai khối ADC,

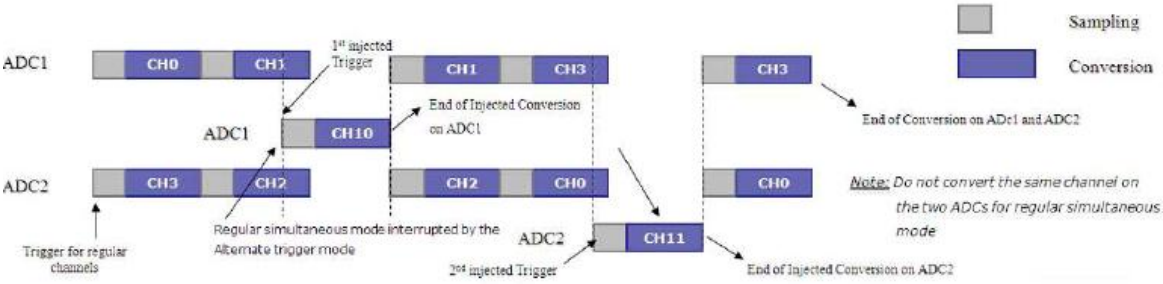
thời gian nhỏ nhất để kích hoạt lần chuyển đổi kế tiếp là 7 chu kỳ xung nhịp của ADC. Ở chế độ xen kẽ chậm khoảng cách thời gian tối thiểu là 14 chu kỳ xung nhịp. Hai chế độ kết hợp này làm tăng hiệu suất chuyển đổi của khối ADC.

3.1.4.4. Chế độ kích hoạt thay thế



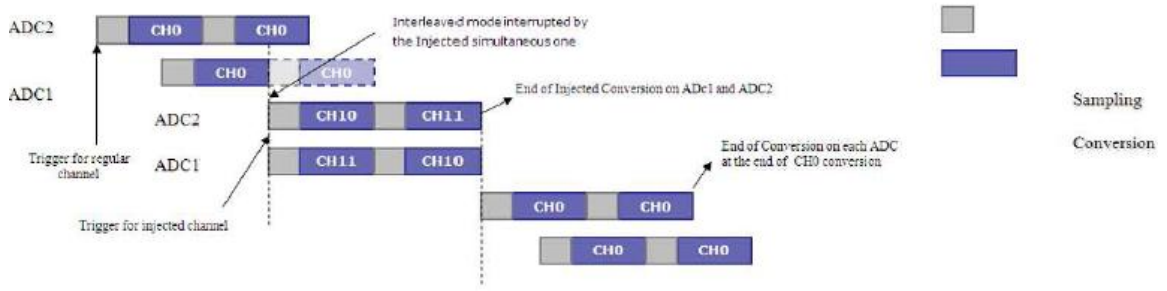
Ban đầu phần cứng sẽ kích hoạt kênh đầu tiên trong nhóm chuyển đổi Injected của khối ADC1, sau đó sẽ kích hoạt tiếp nhóm Injected của ADC2. Cứ như vậy liên tục và xen kẽ.

3.1.4.5. Kết hợp đồng bộ hóa Regular và kích hoạt thay thế



Như ta thấy ở trên, việc chuyển đổi ở chế độ Regular được cả hai khối ADC1 và ADC2 thực thi đồng thời, đồng bộ. Khi có kích hoạt bởi hardware, nhóm Injected của khối ADC1 được thực thi, chế độ Regular tạm thời ngưng và hoạt động trở lại khi tác vụ thuộc nhóm Injected hoàn tất.

3.1.4.6. Kết hợp đồng bộ hóa Injected và xen kẽ Regular



Hai khối ADC1 và ADC2 hoạt động ở chế độ Regular xen kẽ nhau thì được kích hoạt chuyển sang hoạt động ở chế độ đồng bộ Injected. Lưu ý là: khi ở chế độ xen kẽ Regular, cả hai kênh ADC1 và ADC2 có thể chuyển đổi chung trên cùng một kênh, tuy nhiên khi sang chế độ đồng bộ Injected, thì kênh được sử dụng của ADC1 và ADC2 phải khác nhau.

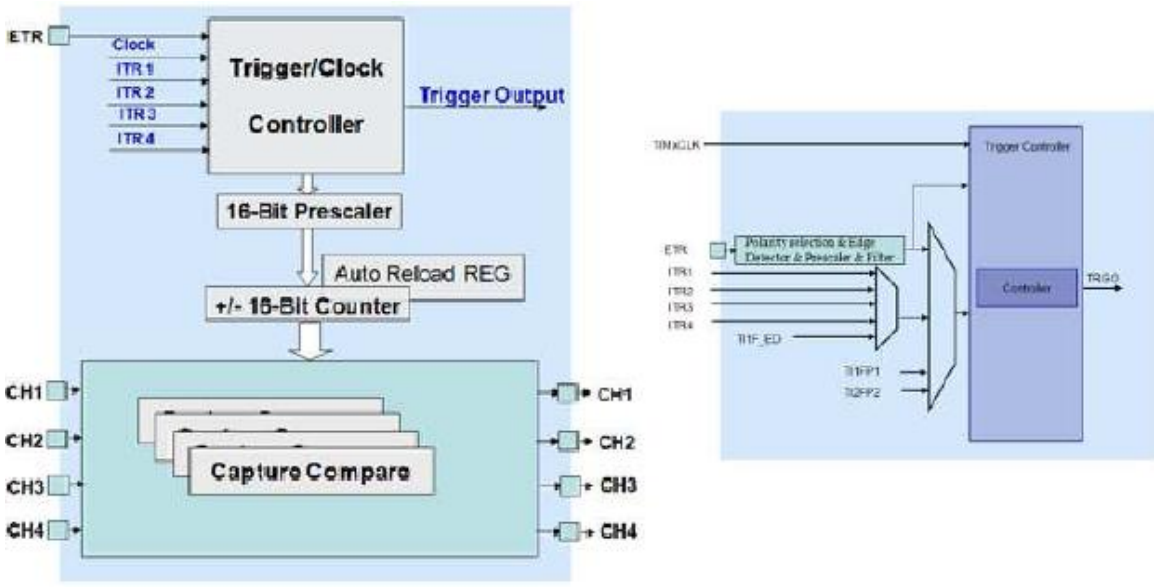
3.1.5. Bộ định thời đa nhiệm và nâng cao

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.

3.1.4.4. 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. Khối định thời sử dụng cổng chọn để lấy xung nhịp đầu vào thích hợp, người dùng có thể sử dụng chân ETR để điều khiển cổng chọn này.

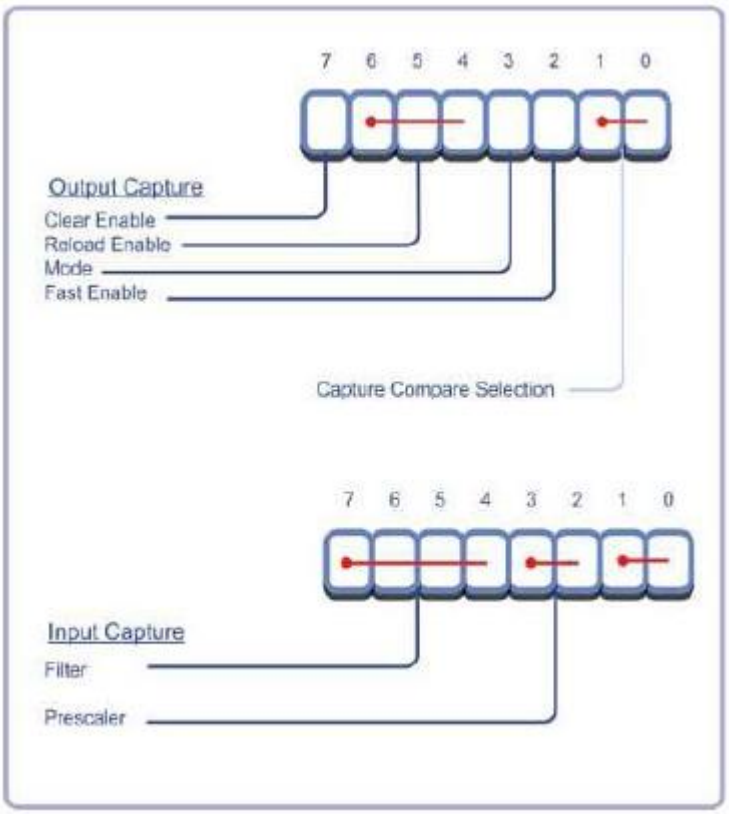


Hình 3.7 4 khối định thời với các thanh ghi 16-bit Prescaler, 16-bit Counter và Auto-reload. Xung nhịp hoạt động có thể lấy từ đồng hồ hệ thống, tín hiệu ngoại và các khối định thời khác

Mỗi khối định thời được cung cấp thêm 4 kênh Capture/Compare. Mỗi khối định thời còn được hỗ trợ ngắt và DMA.

3.2.1. Khối Capture/Compare

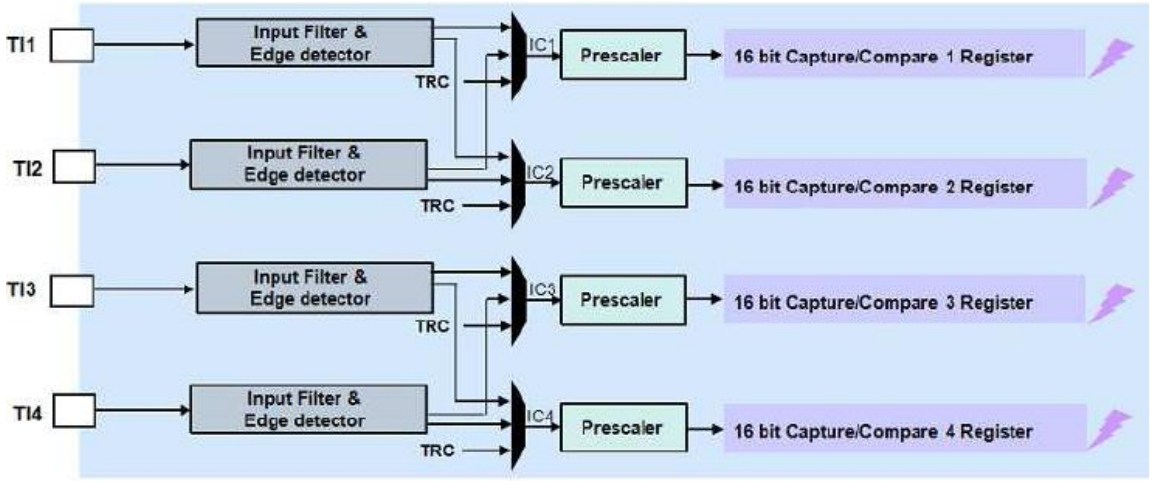
Mỗi kênh Capture/Compare được điều khiển bởi duy nhất một thanh ghi. Chức năng của thanh ghi này có thể thay đổi tùy thuộc cấu hình. Ở chế độ Capture, thanh ghi này có nhóm các bit đảm nhận thiết lập lọc dữ liệu đầu vào và chế độ đánh giá các ngõ PWM. Ở chế độ Compare, STM32 cung cấp hàm chuẩn so sánh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



Mỗi một kênh Capture/Compare đều có một thanh ghi đơn cấu hình chế độ hoạt động. Bit Capture Compare Selection dùng để chọn chế độ.

3.2.2 Khối Capture

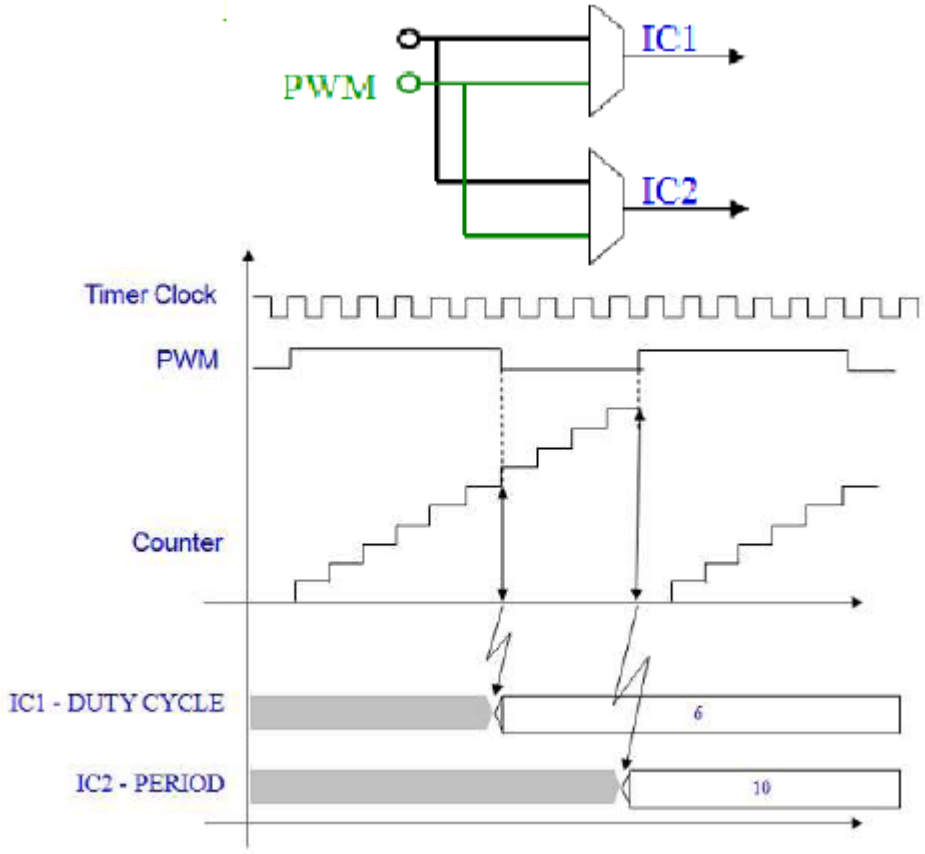
Một khối Capture cơ bản gồm có bốn kênh vào để cấu hình bộ phát hiện xung(Edge Detector). Khi một xung lên(rising edge) hay xung cạnh xuống(falling edge) được phát hiện, bộ đếm hiện thời của sẽ được cập nhật vào các thanh ghi 16-bit Capture/Compare. Khi sự kiện capture xảy ra bộ đếm có thể được khởi động lại hoặc tạm ngưng. Một ngắt DMA có thể được sử dụng ở trường hợp này.



Hình 3.8 4 kênh vào của khối Capture có các bộ lọc dữ liệu và phát hiện xung cạnh riêng. Khi sự kiện capture được nó có thể được dùng để kích hoạt một sự kiện DMA khác.

3.2.3 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 3.9 Chế độ PWM Input

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

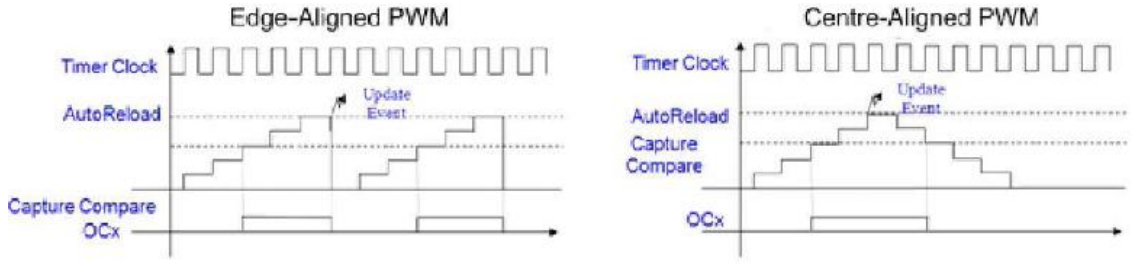
```
M3->CR1     =     0x00000000;//default
TIM3->PSC   =     0x000000FF;//set max prescale
TIM3->ARR   =     0x00000FFF;//set max reload count
```

```
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
                             //reset counter
TIM3->CCER   = 0x00000011;//enable capture channel
TIM3->CR1              = 0x00000001;//enable timer
```

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.

3.2.4 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.

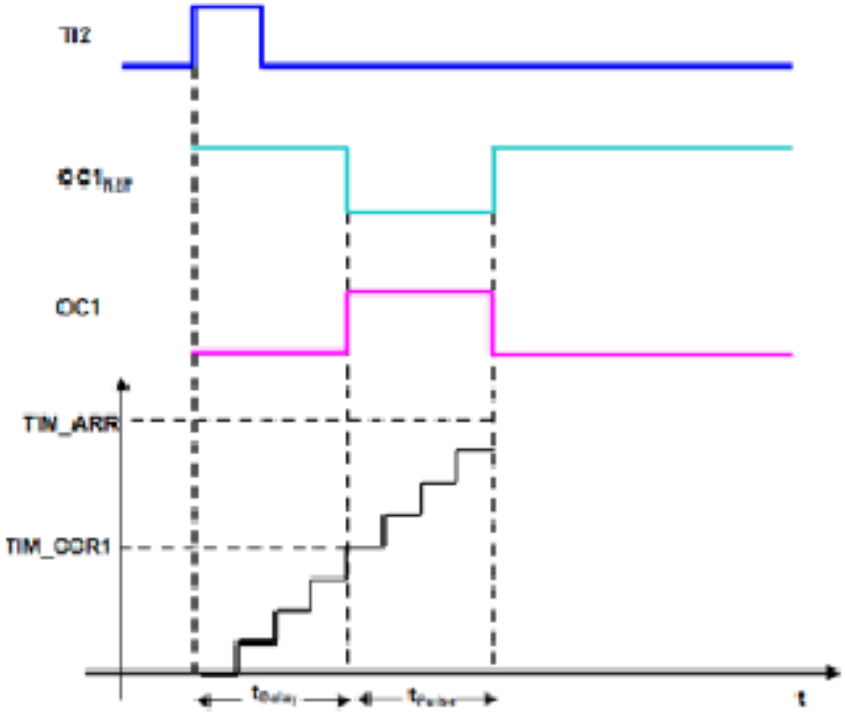


Mỗi khối Timer đều có khả năng tạo ra các xung PWM với độ lệch chu kỳ có thể được cấuhình edge-aligned hoặc centre-aligned tính theo thời điểm cập nhật giá trị của thanh ghi Reload.

```
TIM2->CR1       = 0x00000000;//default
TIM2->PSC       = 0x000000FF;//set max prescaler
TIM2->ARR       = 0x00000FFF;//set max reload count
TIM2->CCMR1     = 0x00000068;//set PWM mode
TIM2->CCR1      = 0x000000FF;//Set PWM start value
TIM2->CCER      = 0x00000101;//Enable CH1 output
TIM2->DIER      = 0x00000000;//enable update interrupt
TIM2->EGR       = 0x00000001;//enable update
TIM2->CR1       = 0x00000001;//enable timer
```

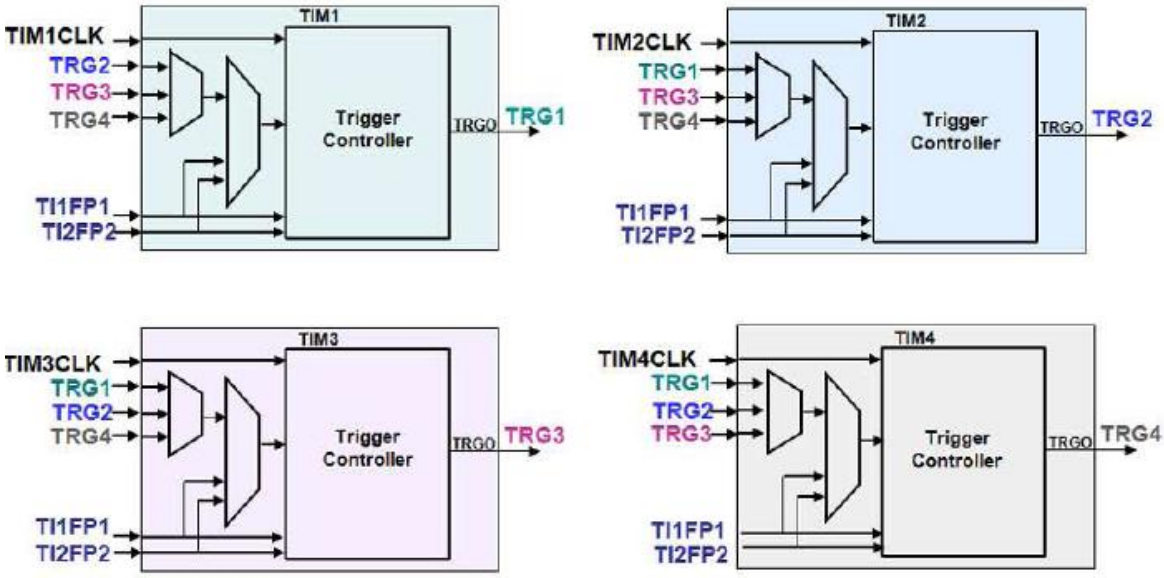
3.2.5 Chế độ One Pulse

Ở 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, bề rộng xung cùng với thời gian trễ có khả năng được cấu hình một cách linh động.



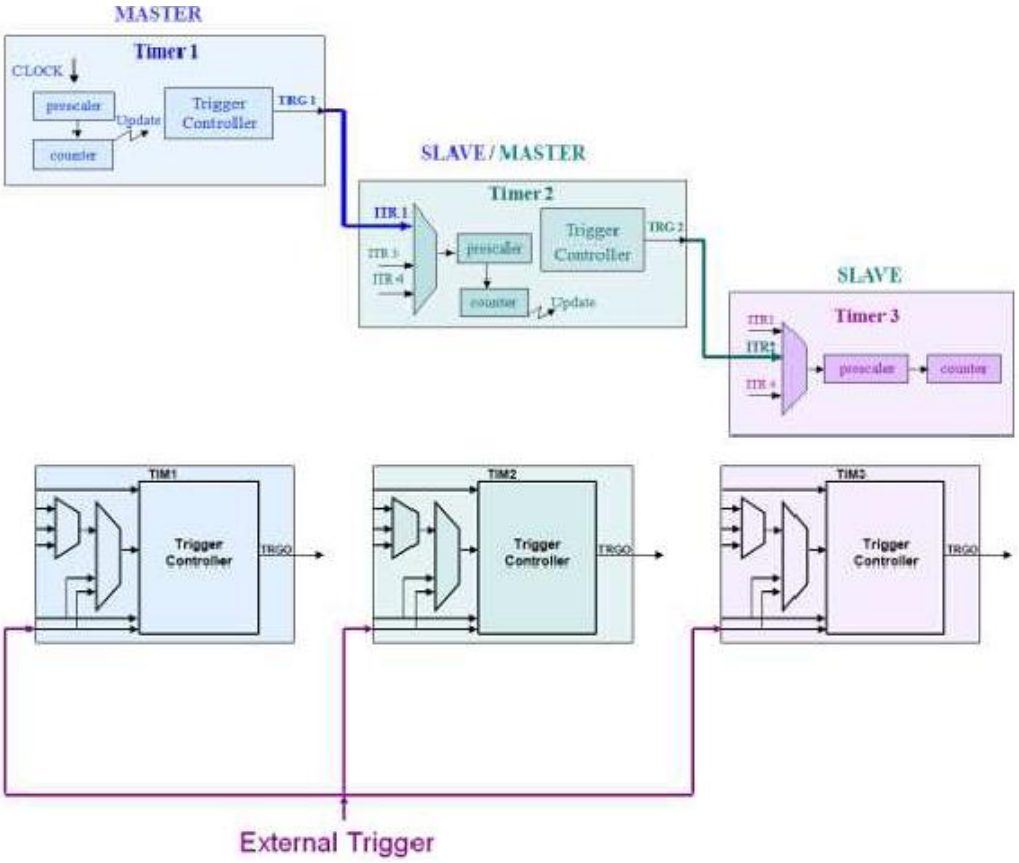
3.3 Đồng bộ hoá các bộ định thời

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 3.10 Mỗi khối Timer có đầu vào là các xung sự kiện
từ các khối Timers khác.

Mỗi khôi 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 khối điều khiển sự kiện của mỗi Timer.



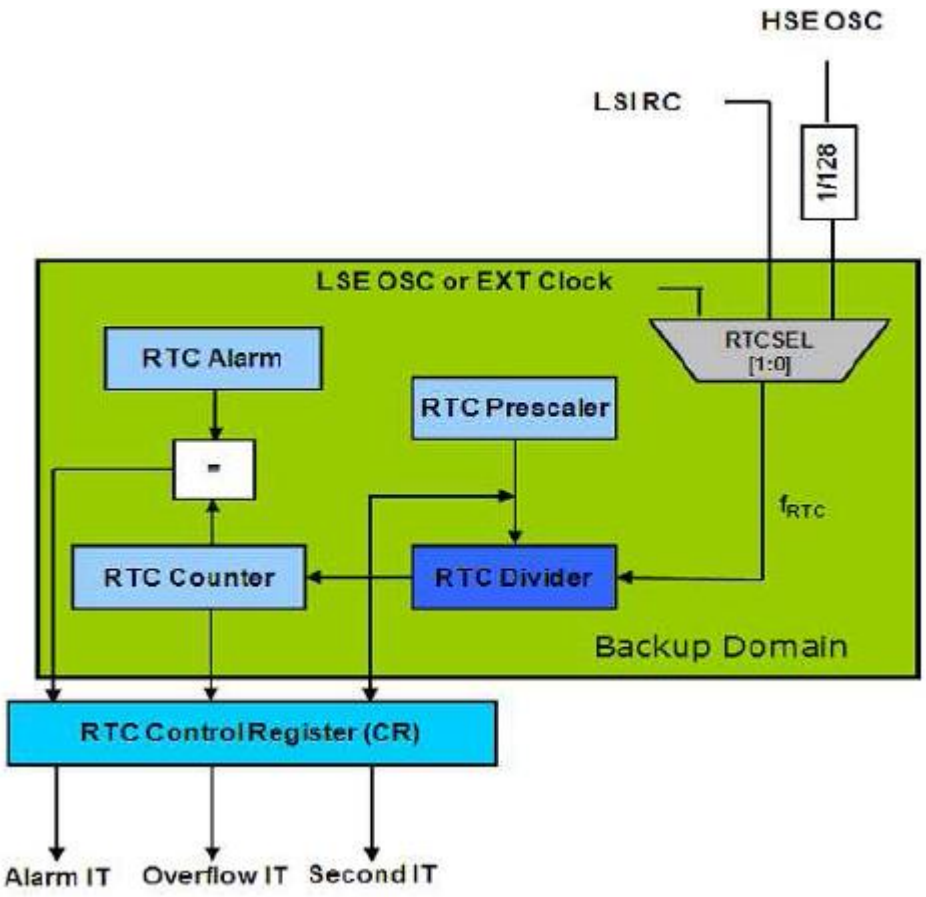
Hình 3.11 Cấu hình các khối Timer kết hợp lại tạo thành mảng các Timer

Ở mô hình tạo thành một mảng Timer, một Timer đóng vai trò Master, các Timer còn lại đóng vai trò là Slave.

3.4 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. Bộ đếm RTC có thể tạo được 3 sự kiện: tăng giá trị đếm, bộ đếm tràn và ngắt báo động. Ngắt báo động khi giá trị bộ đếm trùng với giá trị được cấu hình trong thanh ghi Alarm.



Hình 3.12 Khối RTC có thể lấy nguồn xung nhịp từ LSI, LSE và HSE.

RTC được đặt trong khối dự phòng với nguồn cung Vbat và tín hiệu ngắt Alarm được kết nối với chân nhận xung EXTI17. Điều đó có nghĩa khi hệ thống vào trạng thái hoạt động của mức năng lượng thấp, RTC vẫn hoạt động. Và thông qua sự kiện Alarm, toàn bộ hệ thống có thể được kích hoạt để hoạt động trở lại ở chế độ bình thường.

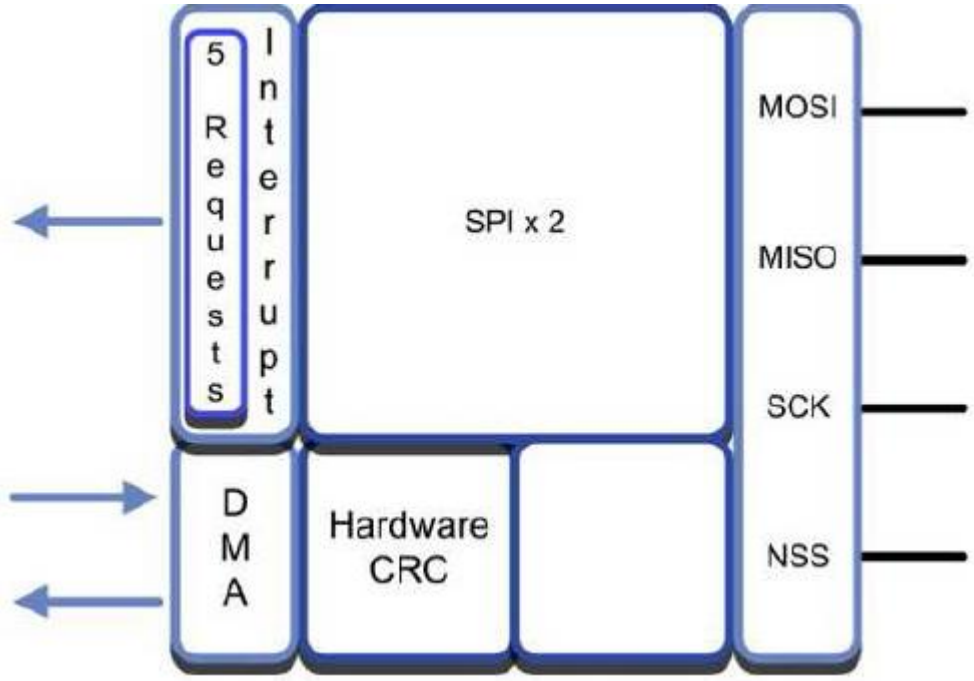
3.5 Kết nối với các giao tiếp khác

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.

3.5.1 SPI

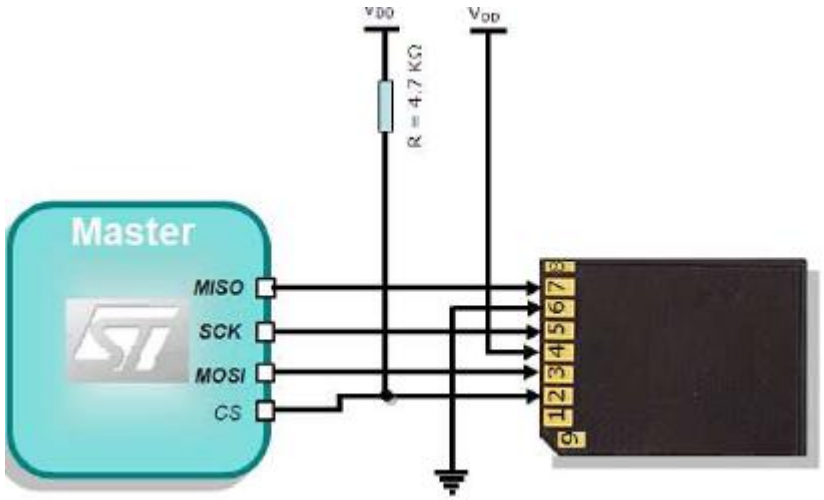
Hỗ trợ giao tiếp tốc độ cao với các mạch tích hợp khác, STM cung cấp 2 khối điều khiển SPI có khả năng chạy ở chế độ song công(Full duplex) với tốc độ truyền dữ liệu lên tới 18MHz. Khối SPI tốc độ cao nằm trên APB2, khối SPI tốc độ thấp nằm trên APB1.Mỗi khối SPI có hệ thống thanh ghi cấu hình độc lập, 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 3.13 Khối 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 thể 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.

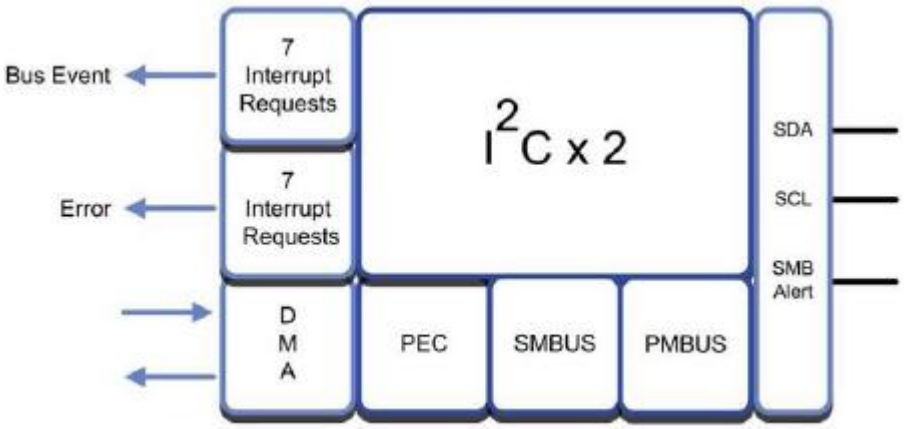


Hình 3.14 Sử dụng SPI để giao tiếp với MMC/SD card.

3.5.2 I2C

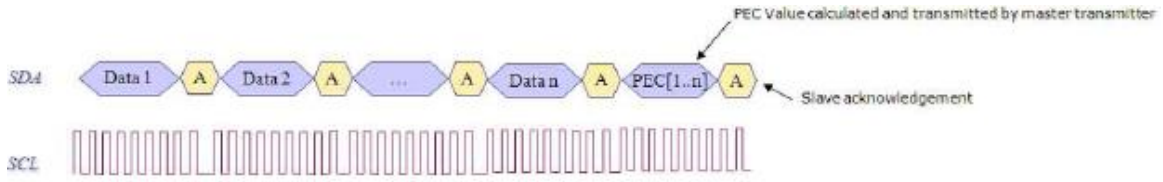
Tương tự như SPI, chuẩn I2C cũng được STM32 hỗ trợ nhằm giao tiếp với các mạch tích hợp ngoài. Giao diện I2C có thể được cấu hình hoạt động ở

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



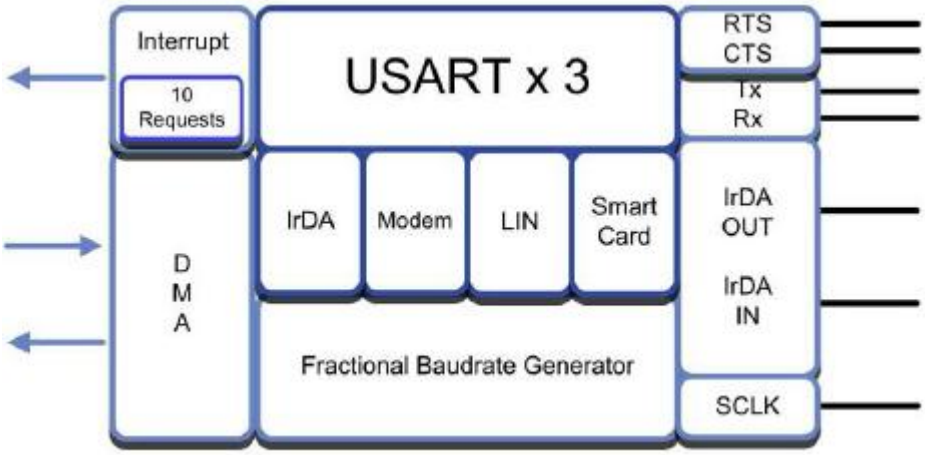
Hình 3.15 Khối I2C

Thêm nữa để đảm bảo tính chính xác dữ liệu truyền, khối 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.



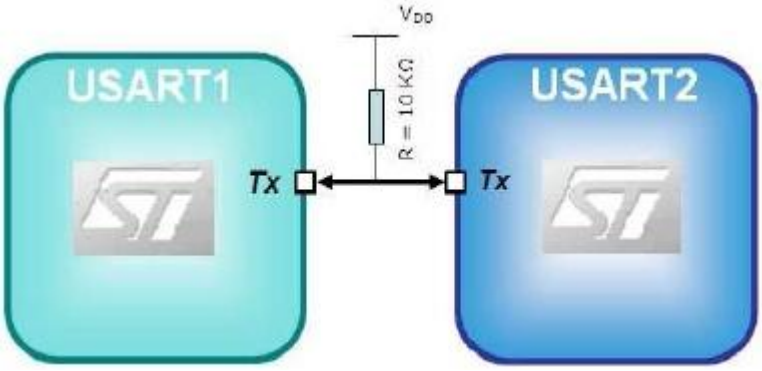
3.5.3 USART

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 khối USART, mỗi khối có khả năng hoạt động đến tốc độ 4.5Mbps. Một khối USART nằm trên APB1 với xung nhịp hoạt động 72MHz, các khối còn lại nằm trên APB2 hoạt động ở xung nhịp 36MHz.



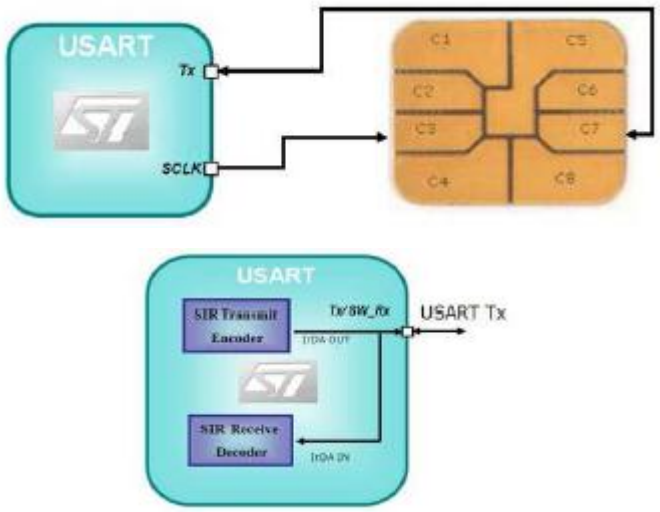
Hình 3.16 Giao diện USART có khả năng hỗ trợ giao tiếp không đồng bộ UARTS, modem cũng như giao tiếp hồng ngoại và Smartcard.

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 hafl-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.



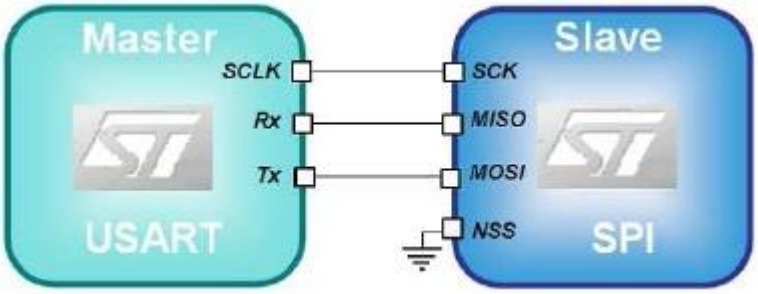
Hình 3.17 Hỗ trợ giao tiếp ở chế độ hafl-duplex dựa trên một đường truyền

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 hafl-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 3.18 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 3.19 Hỗ trợ giao tiếp đồng bộ SPI

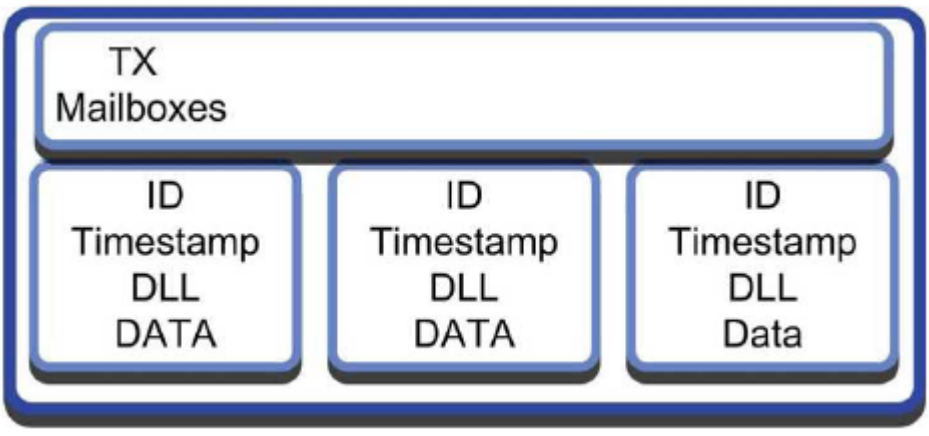
3.5.4 CAN

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 thẻ thời gian Time-trigger CAN(TTCAN).

Hình 3.20 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

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 phải của mình. 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.



Hình 3.21 Khối CAN có 3 mailbox cho truyền dữ liệu với đánh nhãn thời gian tự động cho chuẩn TTCAN

Mỗi bộ lọc có thể được cấu hình hoạt động ở 4 chế độ lọc được đưa vào 2 nhóm chính là lọc theo ID hoặc theo nhóm ID. Chế độ thứ nhất là lọc dựa trên ID của gói tin, nếu các gói tin nào không có ID giống hoặc không giống như ID được cấu hình trong bộ lọc, nó sẽ bị bỏ qua. Chế độ thứ hai cho phép nhận gói tin trong cùng một nhóm. Thanh ghi thứ nhất chứa ID của gói tin, thanh ghi thứ hai chứa "mặt nạ",quy định các thành phần trên vùng ID của thanh ghi thứ nhất mà bộ lọc dựa trên đó để so sánh lọc hay không lọc gói tin.

CAN hoạt động ở hai chế độ: bình thường để truyền nhận dữ liệu và chế độ khởi tạo để cấu hình thông số mạng. Thêm vào đó khối CAN có thể sử dụng chế độ tiết kiệm năng lượng Sleep Mode. Khi ở chế độ Sleep Mode, đồng hồ xung nhịp cấp cho CAN ngưng hoạt động, tuy nhiên thanh ghi mailbox vẫn hoạt động. Điều này cho phép CAN được kích hoạt dựa trên các hoạt động mạng. Có hai chế độ phụ khi CAN hoạt động ở chế độ truyền nhận dữ liệu thông thường. Chế độ Silent, khối CAN chỉ nhận dữ liệu không thể truyền dữ liệu, người ta hay sử dụng chế độ này để theo dõi mạng và các gói tin truyền trong mạng. Chế độ Loopback cho phép toàn bộ các gói tin chuyển được đưa vào ngay chính bộ đệm nhận của khối CAN đó. Chế độ này dùng để tự kiểm tra hoạt động của phần cứng CAN và phần mềm điều khiển.

3.5.5 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(phisical 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.

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.

Chương 4

LẬP TRÌNH ĐIỀU KHIỂN ĐỘNG CƠ BƯỚC SỬ DỤNG ARM-STM32F103

4.1 Giới thiệu Kit STM32 STM32F103

Đặc tính của Kit:

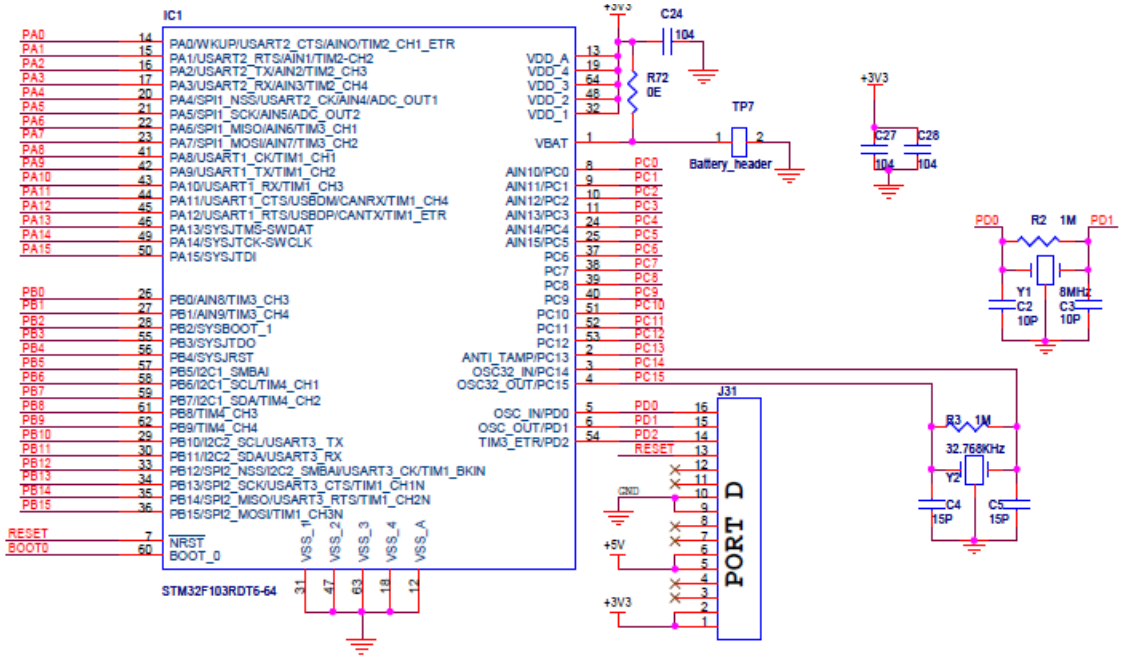
1. MCU: STM32F103 ARM 32 bit CORTEX M3™ with 384K
2. Program Flash, 64K Bytes RAM, USB, CAN, x2 I2C, x16 ADC, x2 DAC
3. x5 UART, x2 SPI, x12 TIMERS, up to 72Mhz operation
4. JTAG connector tiêu chuan với ARM 2x10 pin dành cho viec lap trình và ghỡ rối
5. USB connector
6. SD-MMC card, Audio, Microphone
7. user buttions x3
8. user leds x3
9. RS-232 connector
10. RESET button
11. status LED
12. 8 Mhz crystal oscillator
13. 32768 Hz crystal and RTC backup battery
14. extension headers for all uC ports

Đặc tính STM32F103RDT6:

- CPU clock up to 72Mhz
- FLASH 384KB
- RAM 64KB
- DMA x12 channels
- RTC
- WDT
- Timers x11+1
- SPI x2
- I2C x2

- USART x5
- USB x1
- CAN x1 (multiplexed with USB so both can't be used in same time)
- GPIO up to 51 (multiplexed with peripherials)
- 16 kênh ADC 12-bit, DAC x2
- operating voltage 2.0-3.6V
- temperature -40C +85C

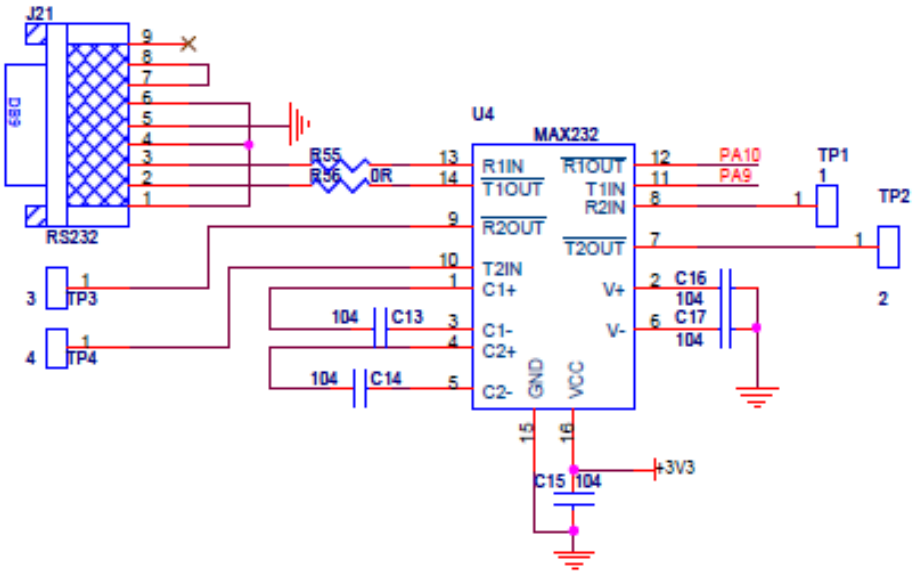
4.1.1 Mạch CPU



Hình 4.1Mạch CPU

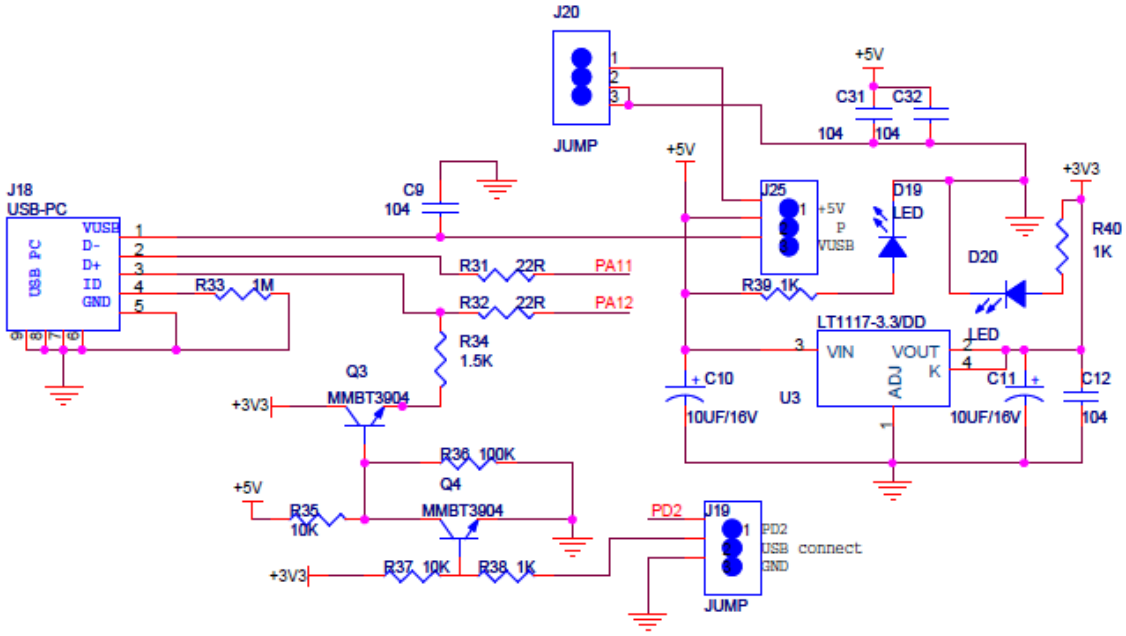
- Thạch anh 8 MHz chân 8-9 tạo xung đồng hồ cho các hoạt động của hệ thống.
- Thạch anh 32.768 KHz chân 3-4 tạo xung dùng cho đồng hồ thời gian thực và watchdog.

4.1.2 Mạch giao tiếp RS232 qua USART1



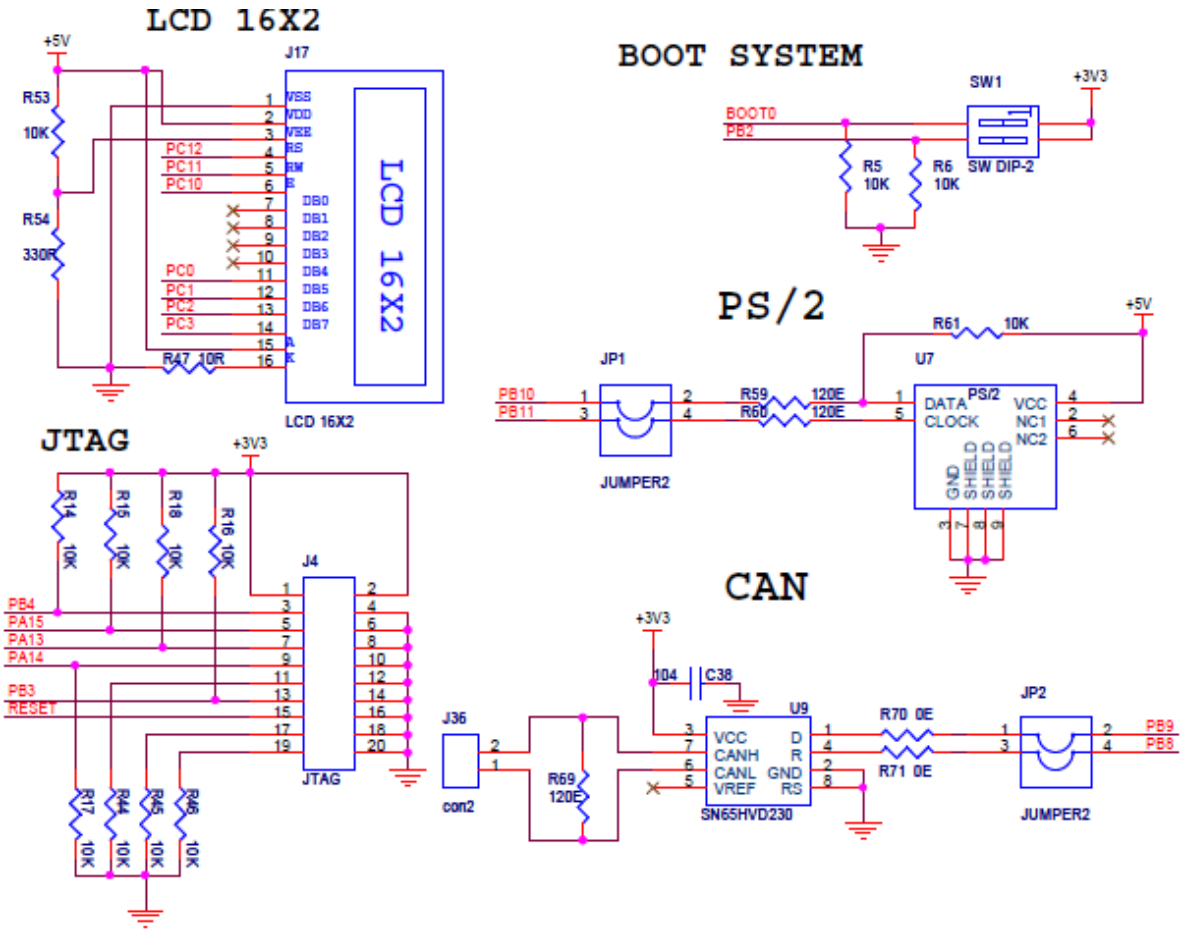
Hình 4.2 Giao tiếp RS232

4.1.3 Mạch cấp nguồn và USB



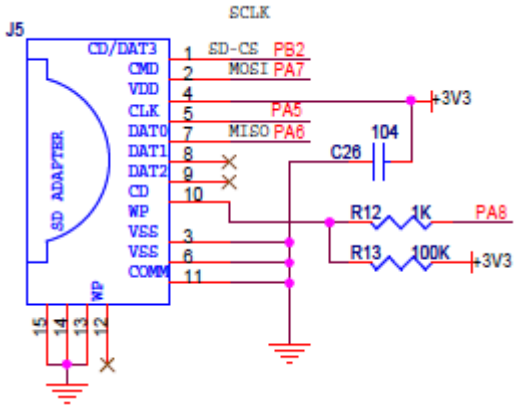
Hình 4.3 Mạch cấp nguồn và USB

4.1.4 Mạch giao tiếp với LCD, nạp và gỡ nỗi chương trình qua JTAG, các mạch giao tiếp CAN/ PS2



Hình 4.4 Giao tiếp LCD, JTAG, PS2, CAN

4.1.5 Mạch thẻ nhớ SD/MMC qua giao tiếp SPI



Hình 4.5 Giao tiếp với thẻ nhớ SD/MMC

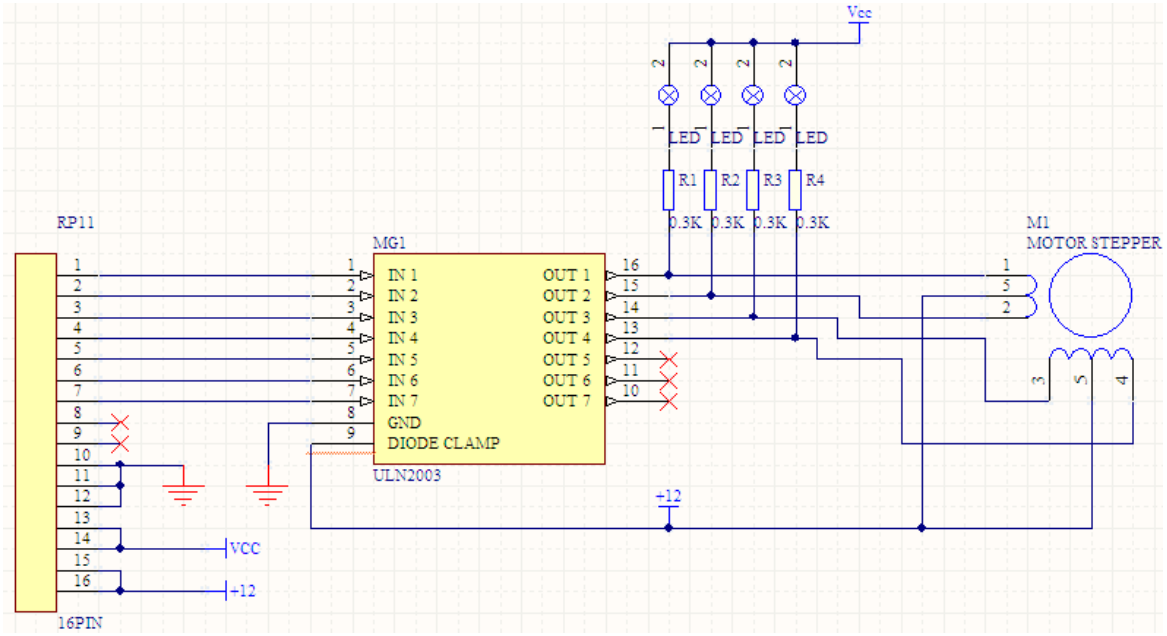
4.2 Điều khiển động cơ bước với Kit STM32 STM32F103

4.2.1.Thiết kế mạch Motor Driver:

- Sử dụng Step Motor đơn cực- 6 dây có góc bước $1,8^0$/ nguồn cấp 12V.
- Với loại motor này có thể đệm dòng bằng IC- ULN 2003.

- Mạch Motor Driver ghép nối với Kit qua cổng PB (chân PB.12, PB.13, PB.14, PB.15)

Sơ đồ Motor Driver như hình 4.6:



Hình 4.6. Mạch Motor Driver

4.2.2. Chương trình điều khiển Step Motor:

Chương trình được viết trên Keil v4.2, sử dụng bộ thư viện chuẩn CMSIS của dòng ARM Cortex-M3

```
#include "main.h"
GPIO_InitTypeDef GPIO_InitStructure;
/**
* @brief Configures the different system clocks.
* @param None
* @retval None
*/
void RCC_Configuration(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
}
/**
* @brief Inserts a delay time with resolution is 10 milisecond..
```

```
* @param nCount: specifies the delay time length.
* @retval None
*/
void delay_ms(__IO uint32_t num)
{
	__IO uint32_t index = 0;
	/* xung dong ho he thong mac dinh la 72MHz */
	for(index = (720000 * num); index != 0; index--)
	{
	}
}
/**
* @brief Main program.
* @param None
* @retval None
*/
int main(void)
{
	/* cau hinh dong ho he thong */
	RCC_Configuration();
	/* cau hinh cac chan xuat */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15 | GPIO_Pin_14 |
GPIO_Pin_13 |
	GPIO_Pin_12;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_Init(GPIOB, &GPIO_InitStructure);
	while (1)
	{
		GPIO_Write(GPIOB,0xC000);
```

```
            delay_ms(3);
            GPIO_Write(GPIOB,0x6000);
            delay_ms(3);
            GPIO_Write(GPIOB,0x3000);
           delay_ms(3);
           GPIO_Write(GPIOB,0x9000);
           delay_ms(3);
       }
}
#ifdef USE_FULL_ASSERT
/**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None
*/
void assert_failed(uint8_t* file, uint32_t line)
{
      /* User can add his own implementation to report the file name and line
number,
      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

      /* vong lap vo han */
      while (1)
      {
      }
}
#endif
```

Kết Luận

Nghiên cứu này ban đầu đã cho thấy được kết quả khả quan, tạo tiền đề cho phát triển các ứng dụng với ARM Cortex M3. Để phát triển đề tài này, tôi xin đưa ra một số ưu nhược điểm như sau:

❖ **Ưu, nhược điểm**:

- **Ưu điểm**:

Giá thành chip rẻ so với các dòng chip khác với cùng số tài nguyên như ARM.

Tốc độ xử lý cao, ổn định.

Tiết kiệm năng lượng

Số lượng tài nguyên lớn, phù hợp với nhiều ứng dụng khác nhau

- Nhược điểm

Nhiều thanh ghi, câu lệnh khá dài, gây khó nhớ cho người dùng, dễ nhầm lẫn.

Thị trường ARM ở Việt Nam chưa rộng, gây khó trong việc tìm kiếm tài liệu và khó khăn trong việc đặt mua chip, do vậy việc nghiên cứu chưa được sâu.

❖ **Hướng phát triển*:***

Đặt mua KIT tạo điều kiện nghiên cứu thực tế trên module.

Tạo các module thực tế để tạo điều kiện thuận lợi cho sinh viên nghiên cứu thực hành với các ứng dụng thực tế, dễ hình dung.

Tài liệu tham khảo:

1/ ARM7TDMI (Rev 3)Technical Reference Manual. Copyright © 1994-2001. All rights reserved. ARM DDI 0029G

2/ The Defi nitive Guide to the ARM Cortex-M3.

http://www.arm.com/