Kiến trúc cơ bản của STM32_ARM Cortex M3

ARM Vietnam

3/18/2010

Mục Lục

Chương 1

GIỚI THIỆU

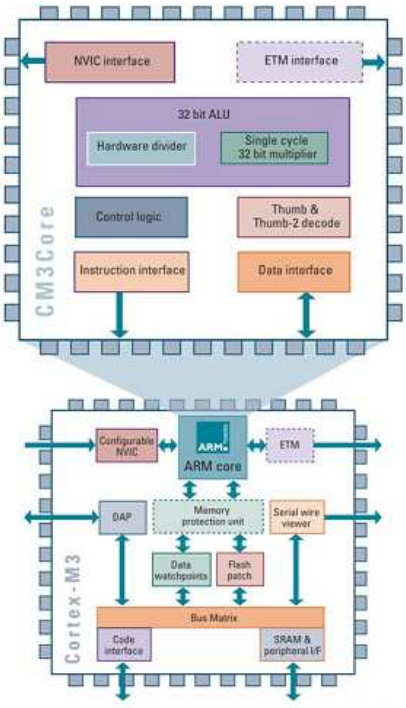
Trong vài năm trở lại đây, một trong những xu hướng chủ yếu trong các thiết kế với vi điều khiển là sử dụng các chip ARM7 và ARM9 như một vi điều khiển đa dụng. Ngày nay các nhà sản xuất IC đưa ra thị trường hơn 240 dòng vi điều khiển sử dụng lõi ARM. Tập đoàn ST Microelectronic vừa cho ra mắt dòng STM32, vi điều khiển đầu tiên dựa trên nền lõi ARM Cortex-M3 thế hệ mới do hãng ARM thiết kế, lõi ARM Cortex-M3 là sự cải tiến của lõi ARM7 truyền thống, từng mang lại sự thành công vang dội cho công ty ARM. Dòng STM32 thiết lập các tiêu chuẩn mới về hiệu suất, chi phí, cũng như khả năng đáp ứng các ứng dụng tiêu thụ năng lượng thấp và tính điều khiển thời gian thực khắc khe.

1.1 Cortex là gì?

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

Các chip ARM7 và ARM9 được các nhà sản xuất bán dẫn thiết kế với giải pháp riêng của mình, đặc biệt là phần xử lí các các ngắt đặc biệt (exception) và các ngắt thông thường (interrupt). Cortex-M3 đưa ra một lõi vi điều khiển chuẩn nhằm cung cấp phần tổng quát, quan trọng nhất của một vi điều khiển,

bao gồm hệ thống ngắt (interrupt system), SysTick timer (được thiết kế cho hệ điều hành thời gian thực), hệ thống kiểm lỗi (debug system) và memory map. Không gian địa chỉ 4Gbyte của Cortex-M3 được chia thành các vùng cho mã chương trình, SRAM, ngoại vi và ngoại vi hệ thống. Không giống với ARM7 được thiết kế theo kiến trúc Von Neumann (bộ nhớ chương trình và bộ nhớ dữ liệu chung với nhau), Cortex-M3 được thiết kế dựa theo kiến trúc Harvard (bộ nhớ chương trình và bộ nhớ dữ liệu tách biệt với nhau), và có nhiều bus cho phép thực hiện các thao tác song song với nhau, do đó làm tăng hiệu suất của chip. Không giống với các kiến trúc ARM trước đó, dòng Cortex cho phép truy cập dữ liệu không xếp hàng (unaligned data, vì chip ARM là kiến trúc 32bit, do đó tất cả các dữ liệu hoặc mã chương trình đều được sắp sếp khít với vùng bộ nhớ là bội số của 4byte). Đặc điểm này cho phép sử dụng hiệu quả SRAM nội. Dòng Cortex còn hỗ trợ việc đặt và xoá các bit bên trong hai vùng 1Mbyte của bộ nhớ bằng phương pháp gọi là bit banding. Đặc điểm này cho phép truy cập hiệu quả tới các thanh ghi ngoại vi và các cờ được dùng trên bộ nhớ SRAM mà không cần một bộ xử lí luận lí (Boolean processor).



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

Khối trung tâm của STM32 là bộ xử lí Cortex-M3. Bộ xử lí Cortex-M3 là một vi điều khiển được tiêu chuẩn hoá gồm một CPU 32bit, cấu trúc bus (bus structure), đơn vị xử lí ngắt có hỗ trợ tính năng lồng ngắt vào nhau (nested interrupt unit), hệ thống kiểm lỗi (debug system) và tiêu chuẩn bố trí bộ nhớ (standard memory layout).

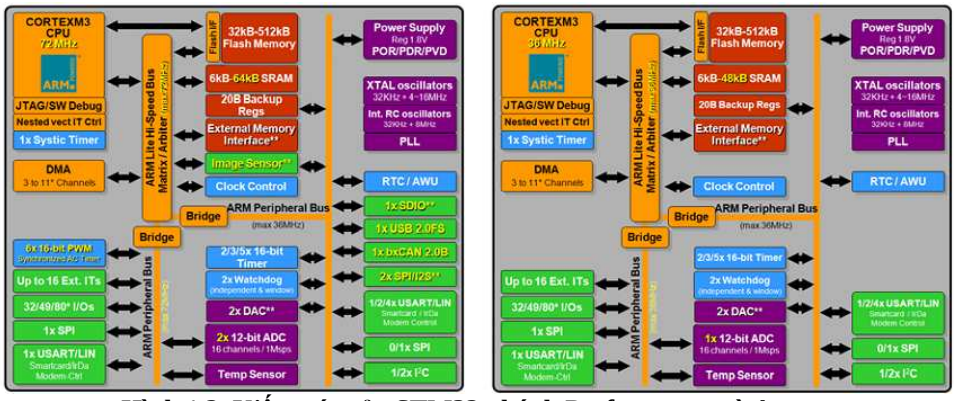
Một trong những thành phần chính của lõi Cortex-M3 là NVIC (Nested Vector Interrupt Controller). NVIC cung cấp một cấu trúc ngắt chuẩn cho tất cả các vi điều khiển được thiết kế dựa trên lõi Cortex và cách xử lí các ngắt đặc biệt (exceptional interrupt). NVIC cung cấp các vector ngắt chuyên dụng lên tới 240 nguồn ngắt từ ngoại vi, mỗi nguồn ngắt đó có thể được ưu tiên hoá với các mức riêng biệt. NVIC được thiết kế để xử lí các ngắt đòi hỏi thời gian đáp ứng cực kì nhanh (extremely fast interrupt). Thời gian từ lúc nhận một tín hiệu ngắt cho tới khi thực thi dòng lệnh đầu tiên trong trình phục vụ ngắt chỉ là 12 chu kì xung nhịp. Công việc này được thực hiện tự động bởi một vi chuong trình (microcode) được cài sẵn trong CPU. Trong trường hợp xuất hiện các interrupt lồng nhau (tức là xảy ra ngắt khi đang xử lí ngắt trước đó), NVIC sử dụng một phương thức gọi là “tail chain” cho phép ngắt liên tiếp được phục vụ với độ trễ chỉ có 6 chu kì xung nhịp. Trong suốt giai đoạn lưu trữ dữ liệu lên vùng nhớ stack để bắt đầu thực thi chương trình phục vụ ngắt, một ngắt có mức ưu tiên cao hơn ngắt hiện tại có thể cạnh tranh với (pre-empt) ngắt hiện tại mà không chịu bất kì trì hoãn nào. Cấu trúc ngắt cũng đi kèm với chế độ tiết kiệm năng lượng của trong lõi Cortex-M3. CPU có thể được cấu hình tự động vào chế độ tiết kiệm năng lượng sau khi thoát khỏi ngắt. Sau đó lõi tiếp tục ngủ cho đến khi một exception (ngắt đặc biệt) xuất hiện.

Mặc dù Cortex-M3 được thiết kế như là một lõi chi phí thấp (low cost core), nhưng nó vẫn là một CPU 32-bit và vẫn hỗ trợ hai chế độ hoạt động: Thread và Handler, mỗi chế độ có thể được cấu hình với mỗi vùng stack riêng biệt của nó, điều này cho phép thiết kế các phần mềm phức tạp và hỗ trợ các hệ diều

hành thời gian thực. Lõi Cortex có hỗ trợ một timer 24-bit tự động nạp lại giá trị, nó sẽ cung cấp một ngắt timer đều đặn cho một nhận RTOS (Real Time Operating System). Các chip ARM7 vả ARM9 có hai tập lệnh (tập lệnh ARM 32-bit và tập lệnh Thumb 16-bit), trong khi đó dòng Cortex được thiết kế hỗ trợ tập lệnh ARM Thumb-2, tập lệnh này được pha trộn giữa tập lệnh 16 và 32-bit, nhằm đạt được hiệu suất cao của của tập lệnh ARM 32-bit với mật độ mã chương trình tối ưu của tập lệnh Thumb 16-bit. Tập lệnh Thumb-2 được thiết kế đặc biệt dành cho trình biên dịch C/C++, tức là các ứng dụng dựa trên nền Cortex hoàn toàn có thể được viết bằng ngôn ngữ C mà không cần đến chương trình khởi động viết bằng assembler như ARM7 và ARM9.

1.2 Một vài đặc điểm nổi bật của STM32

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



Hình 1.2. Kiến trúc của STM32 nhánh Performance và Access

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

1.2.1 Sự tinh vi

Thoạt nhìn thì các ngoại vi của STM32 cũng giống như những vi điều khiển khác, như hai bộ chuyển đổi ADC, timer, I2C, SPI, CAN, USB và RTC. Tuy nhiên mỗi ngoại vi trên đều có rất nhiều đặc điểm thú vị. Ví dụ như bộ ADC 12-bit có tích hợp một cảm biến nhiệt độ để tự động hiệu chỉnh khi nhiệt độ thay đổi và hỗ trợ nhiều mode chuyển đổi. Mỗi bộ timer có 4 khối capture compare, mỗi khối timer có thể liên kết với các khối timer khác để tạo ra một mảng các timer tinh vi. Một timer cao cấp chuyên hỗ trợ điều khiển động cơ, với 6 đầu ra PWM với dead time lập trình được và một đường break input sẽ buộc tín hiệu PWM sang một trạng thái an toàn đã được cài sẵn. Ngoại vi nối tiếp SPI có một khối kiểm tổng CRC bằng phần cứng cho 8 và 16 word hỗ trợ tích cực cho giao tiếp thẻ nhớ SD hoặc MMC.

STM32 có hỗ trợ thêm 7 kênh DMA (Direct Memory Access). Mỗi kênh có thể được dùng để truyền dữ liệu đến các thanh ghi ngoại vi hoặc từ các thanh ghi ngoại vi đi với kích thước từ (word) dữ liệu truyền đi có thể là 8/16 hoặc 32-bit. Mỗi ngoại vi có thể có một bộ điều khiển DMA (DMA controller) đi kèm dùng để gửi hoặc đòi hỏi dữ liệu như yêu cầu. Một bộ phân xử bus nội (bus arbiter) và ma trận bus (bus matrix) tối thiểu hoá sự tranh chấp bus giữa truy cập dữ liệu thông qua CPU (CPU data access) và các kênh DMA. Điều đó cho phép các đơn vị DMA hoạt động linh hoạt, dễ dùng và tự động điều khiển các luồng dữ liệu bên trong vi điều khiển.

STM32 là một vi điều khiển tiêu thụ năng lượng thấp và đạt hiệu suất cao. Nó có thể hoạt động ở điện áp 2V, chạy ở tần số 72MHz và dòng tiêu thụ chỉ có 36mA với tất cả các khối bên trong vi điều khiển đều được hoạt động. Kết hợp với các chế độ tiết kiệm năng lượng của Cortex, STM32 chỉ tiêu thụ 2µA khi ở

chế độ standby. Một bộ dao động nội RC 8MHz cho phép chip nhanh chóng thoát khỏi chế độ tiết kiệm năng lượng trong khi bộ dao động ngoài đang khởi động. Khả năng nhanh đi vào và thoát khỏi các chế độ tiết kiệm năng lượng làm giảm nhiều sự tiêu thụ năng lượng tổng thể.

1.2.2 Sự an toàn

Ngày nay các ứng dụng hiện đại thường phải hoạt động trong môi trường khắc khe, đòi hỏi tính an toàn cao, cũng như đòi hỏi sức mạnh xử lý và càng nhiều thiết bị ngoại vi tinh vi. Để đáp ứng các yêu cầu khắc khe đó, STM32 cung cấp một số tính năng phần cứng hỗ trợ các ứng dụng một cách tốt nhất. Chúng bao gồm một bộ phát hiện điện áp thấp, một hệ thống bảo vệ xung clock và hai bộ watchdogs. Bộ đầu tiên là một watchdog cửa sổ. Watchdog này phải được làm tươi trong một khung thời gian xác định. Nếu nhấn nó quá sớm, hoặc quá muộn, thì watchdog sẽ kích hoạt. Bộ thứ hai là một watchdog độc lập, có bộ dao động bên ngoài tách biệt với xung nhịp hệ thống chính. Hệ thống bảo vệ xung nhịp có thể phát hiện lỗi của bộ dao động chính bên ngoài (thường là thạch anh) và tự động chuyển sang dùng bộ dao động nội RC 8MHz.

1.2.3 Tính bảo mật

Một trong những yêu cầu khắc khe khác của thiết kế hiện đại là nhu cầu bảo mật mã chương trình để ngăn chặn sao chép trái phép phần mềm. Bộ nhớ Flash của STM32 có thể được khóa để chống truy cập đọc Flash thông qua cổng debug. Khi tính năng bảo vệ đọc được kích hoạt, bộ nhớ Flash cũng được bảo vệ chống ghi để ngăn chặn mã không tin cậy được chèn vào bảng vector ngắt. Hơn nữa bảo vệ ghi có thể được cho phép trong phần còn lại của bộ nhớ Flash. STM32 cũng có một đồng hồ thời gian thực và một khu vực nhỏ dữ liệu trên SRAM được nuôi nhờ nguồn pin. Khu vực này có một đầu vào chống giả mạo, có thể kích hoạt một sự kiện ngắt khi có sự thay đổi trạng thái ở đầu vào này. Ngoài ra một sự kiện chống giả mạo sẽ tự động xóa dữ liệu được lưu trữ trên SRAM được nuôi bằng nguồn pin.

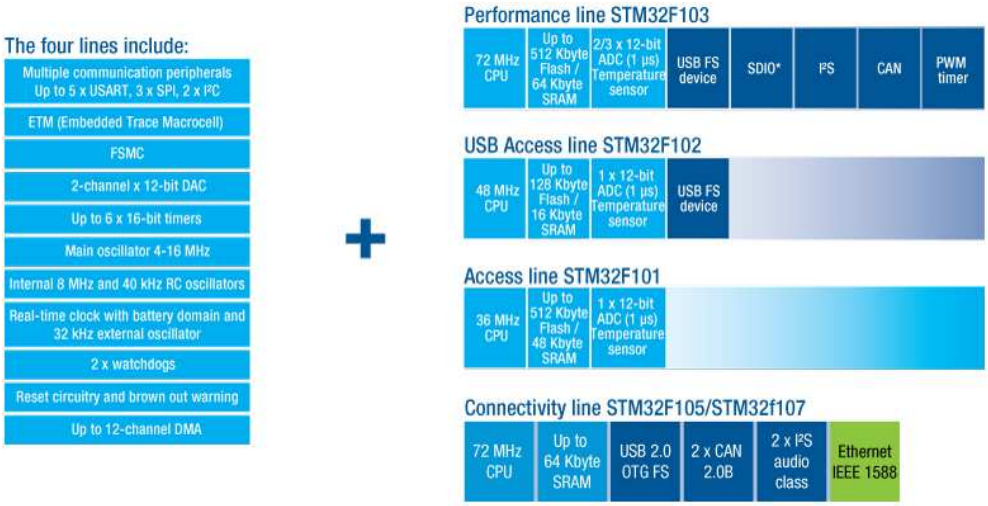
1.2.4 Phát triển phần mềm

Nếu bạn đã sử dụng một vi điều khiển dựa trên lõi ARM, các công cụ phát triển đã được hỗ trợ tập lệnh Thumb-2 và dòng Cortex. Ngoài ra ST cũng cung cấp một thư viện điều khiển thiết bị ngoại vi, một bộ thư viện phát triển USB như là một thư viện ANSI C và mã nguồn đó là tương thích với các thư viện trước đó được công bố cho vi điều khiển STR7 và STR9. Có rất nhiều RTOS mã nguồn mở và thương mại và middleware (TCP/IP, hệ thống tập tin, v.v.) hỗ trợ cho họ Cortex. Dòng Cortex-M3 cũng đi kèm với một hệ thống gỡ lỗi hoàn toàn mới gọi là CoreSight. Truy cập vào hệ thống CoreSight thông qua cổng truy cập Debug (Debug Access Port), cổng này hỗ trợ kết nối chuẩn JTAG hoặc giao diện 2 dây (serial wire-2 Pin), cũng như cung cấp trình điều khiển chạy gỡ lỗi, hệ thống CoreSight trên STM32 cung cấp một data watchpoint và một công cụ theo dõi (instrumentation trace). Công cụ này có thể gửi thông tin về ứng dụng được lựa chọn đến công cụ gỡ lỗi. Điều này có thể cung cấp thêm các thông tin gỡ lỗi và cũng có thể được sử dụng trong quá trình thử nghiệm phần mềm.

1.2.5 Dòng Performance và Access của STM32

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

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



Hình 1.3. Đặc điểm của bốn nhánh trong họ STM32

Chương 2

TỔNG QUAN VỀ CORTEX

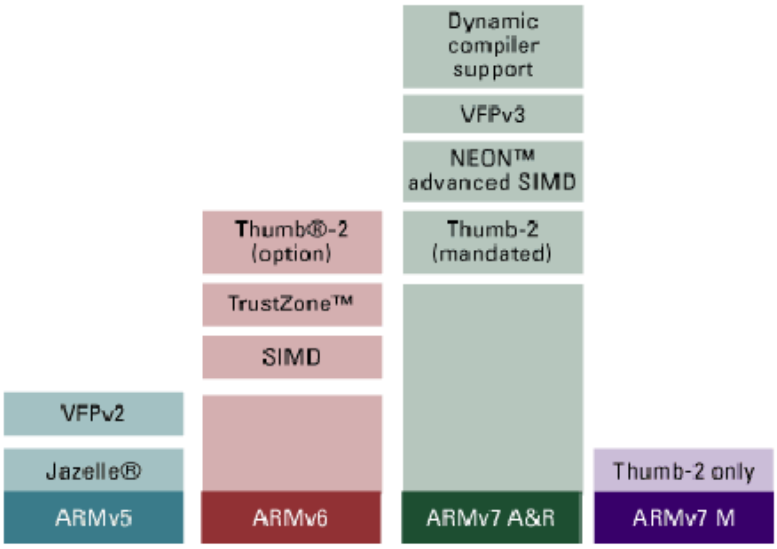
Như chúng ta đã thấy trong phần giới thiệu, bộ xử lý Cortex là thế hệ lõi nhúng kế tiếp từ ARM. Cortex thừa kế các ưu điểm từ các bộ xử lí ARM trước đó, nó là một lõi xử lý hoàn chỉnh, bao gồm bộ xử lí trung tâm Cortex và một hệ thống các thiết bị ngoại vi xung quanh, Cortex cung cấp phần xử lí trung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

- Cortex-A : bộ vi xử lý dành cho hệ điều hành và các ứng dụng của người dùng phức tạp. Hỗ trợ các tập lệnh ARM, Thumb và Thumb-2.
- Cortex-R : bộ xử lí dành cho các hệ thống đòi hỏi khắc khe về tính thời gian thực. Hỗ trợ các tập lệnh ARM, Thumb, và Thumb-2.
- Cortex-M : bộ xử lí dành cho dòng vi điều khiển, được tối ưu hóa cho các ứng dụng nhạy cảm về chi phí. Chỉ hỗ trợ tập lệnh Thumb-2.

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

2.1 Các phiên bản kiến trúc ARM

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



Hình 2.1. Các phiên bản kiến trúc của lõi ARM

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

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

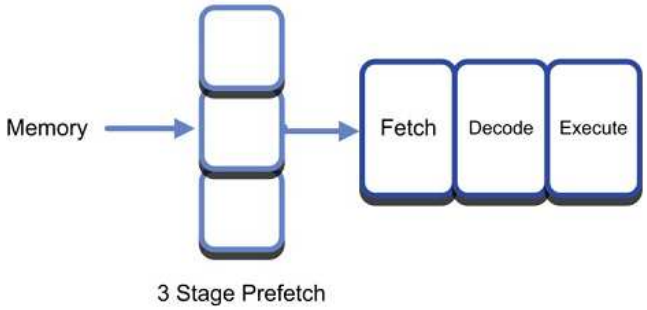
Trong suốt phần còn lại của tài liệu này, các thuật ngữ bộ xử lí Cortex (Cortex processor) và đơn vị xử lí trung tâm Cortex (Cortex CPU) sẽ được sử dụng để phân biệt giữa nhúng lõi Cortex hoàn chỉnh và bộ xử lí trung tâm RISC nội (internal RISC CPU). Trong phần tiếp theo chúng ta sẽ xem xét các đặc điểm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

2.3 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.

2.3.1 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.



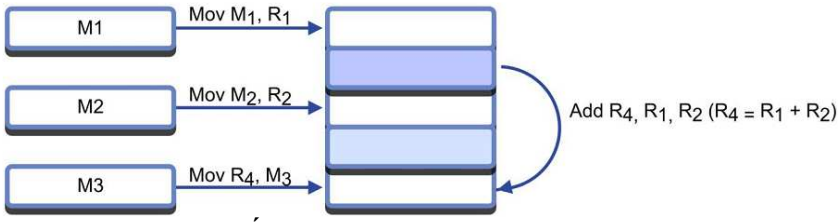
Hình 2.2. Kiến trúc đường ống của ARM Cortex-M3

Trong khi một lệnh đang được thực thi, thì lệnh tiếp theo sẽ được giải mã và lệnh tiếp theo nữa sẽ được lấy về từ bộ nhớ. Phương thức hoạt động này sẽ phát huy hiệu quả tối đa cho mã tuyến tính (linear code), nhưng khi gặp phải một rẽ nhánh (ví dụ cấu trúc lệnh if…else) thì các đường ống phải được làm rỗng (flush) và làm đầy (refill) trước khi mã có thể tiếp tục thực thi. Với CPU ARM7 và ARM9, việc rẽ nhánh là rất tốn kém về mặt hiệu suất mã (code performance). Trong CPU Cortex có đường ống ba tầng được tăng cường khả năng dự đoán rẽ nhánh, có nghĩa rằng khi một lệnh rẽ nhánh có điều kiện xuất hiện, một thao tác lấy lệnh dựa trên suy đoán được thực hiện, do đó lệnh rẽ nhánh có điều kiện sẵn sàng để thực hiện mà không cần chịu thêm một thao tác nào. Trường hợp xấu nhất khi gặp phải một rẽ nhánh gián tiếp, khi đó không thể thực hiện việc lấy lệnh dựa trên việc suy đoán, do đó phải làm rỗng đường ống dẫn. Kiến trúc đường ống là chìa khóa dẫn đến hiệu suất tổng thể của CPU Cortex, vì vậy không cần bất kì cân nhắc, xem xét đặc biệt nào được thực hiện trong mã ứng dụng.

2.3.2 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.



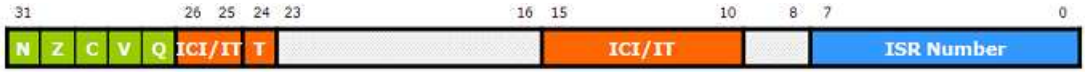
Hình 2.3. Kiến trúc load và store của ARM Cortex-M3

Do vậy tất cả các hoạt động của chương trình tập trung xung quanh tập thanh ghi của CPU. 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. Các thanh ghi R13-R15 có chức năng đặc biệt trong CPU Cortex. Thanh ghi R13 được dùng như là con trỏ ngăn xếp (stack pointer). Thanh ghi này được chia thành nhóm (banked), cho phép CPU Cortex có hai chế độ hoạt động, mỗi chế độ có không gian ngăn xếp riêng biệt. Đặc điểm này thường được hệ điều hành thời gian thực (Real Time Operating System) sử dụng để có thể chạy mã hệ thống của mình trong một chế độ bảo vệ. 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). Nếu chương trình của bạn gọi sâu vào nhiều lớp chương trình con, trình biên dịch sẽ tự động lưu R14 trên ngăn xếp (stack). Thanh ghi cuối cùng R15 là bộ đếm chương trình (Program Counter); nó là một phần của tập thanh ghi trung tâm, nó có thể được đọc và thao tác giống như bất kỳ thanh ghi khác.

Hình 2.3. Mô hình lập trình của ARM Cortex-M3

2.3.2.1 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). Nó không phải là một phần của tập thanh ghi chính và chỉ có thể truy cập thông qua hai lệnh chuyên dụng. XPSR chứa một số các vùng chức năng quan trọng ảnh hưởng đến việc thực thi của CPU Cortex.



Hình 2.4. Thanh ghi trạng thái chương trình của CPU Cortex

Thanh ghi xPSR cũng có thể được truy cập thông qua ba biệt hiệu đặc biệt (special alias names) cho phép truy cập vào các bit trong xPSR. 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 lý dữ 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 ARM 32-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 ứng dụng (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. Trong CPU Cortex, kỹ thuật này được mở rộng với thanh ghi trạng thái chương trình thực thi. Đây là một biệt hiệu của bit các bit từ 8-26 của xPSR. Nó gồm ba trường: trường "If then", trường "interrupt continuable instruction"và trường lệnh Thumb. Lệnh Thumb-2 có một phương pháp hiệu quả khi thực hiện các khối lệnh nhỏ 'if then'. Khi một kiểm tra điều kiện là đúng, nó có thể thiết lập một giá trị trong vùng IT, báo cho CPU thực thi lên bốn lệnh. Nếu việc kiểm tra điều kiện là sai, các lệnh này sẽ đi ngang qua đường ống như là một lệnh NOP. Vì vậy, một dòng lệnh C điển hình sẽ được mã hoá như sau:

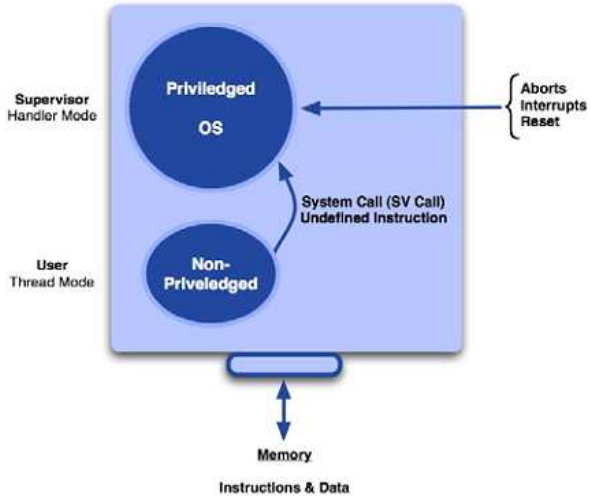
```
If (r0 ==0)
        CMP r0,#0           ;compare r0 to 0
        ITTEE EQ            ;if true execute the next two instructions
Then r0 = *r1 +2;
        LDR r0,[r1]         ;load contents of memory location into r0
        ADDr0,#2            ;add 2
```

Hầu hết các lệnh Thumb-2 thực thi trong một chu kì đơn, một số khác (như lệnh load và store) cần nhiều chu kỳ. Vì vậy, để CPU Cortex có thể có một thời gian đáp ứng ngắt xác định, các lệnh cần nhiều chu kì thực thi phải được ngắt. Khi một lệnh được chấm dứt sớm, vùng ICI (Interrupt Continuable Instruction) trong thanh ghi xPSR sẽ lưu lại số các thanh ghi tiếp theo được dùng trong lệnh load hoặc store nhiều dữ liệu cùng lúc. Vì vậy, một khi ngắt được phục vụ, lệnh load/store bị ngắt trước đó có thể tiếp tục được thực hiện. Trường Thumb cuối cùng được thừa hưởng từ phiên bản CPU ARM trước đó. Trường này chỉ ra nếu tập lệnh ARM hoặc Thumb đang được thực hiện bởi CPU. Trong Cortex-M3 bit này luôn luôn được thiết lập mức 1 (tức là tập lệnh đang được thực thi là tập lệnh Thumb). Cuối cùng, trường trạng thái ngắt chứa thông tin về yêu cầu ngắt đã được ưu tiên trước (pre-empted).

2.3.3 Các chế độ hoạt động của CPU

Bộ vi xử lý Cortex được thiết kế với mục tiêu giảm số bóng bán dẫn, nhanh chóng và dễ sử dụng lõi vi điều khiển, nó có được thiết kế để hỗ trợ việc sử

dụng hệ điều hành thực hành thời gian. Bộ xử lý Cortex có hai chế độ hoạt động: chế độ Thread và chế độ Handler. CPU sẽ chạy ở chế độ Thread trong khi nó đang thực thi ở chế độ nền không có ngắt xảy ra và sẽ chuyển sang chế độ Handler khi nó đang thực thi các ngắt đặc biệt (exceptions). Ngoài ra, CPU Cortex có thể thực thi mã trong chế độ đặc quyền hoặc không đặc quyền (privileged or non-privileged mode). Trong chế độ đặc quyền, CPU có quyền truy cập tất cả các lệnh. Trong chế độ không có 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).



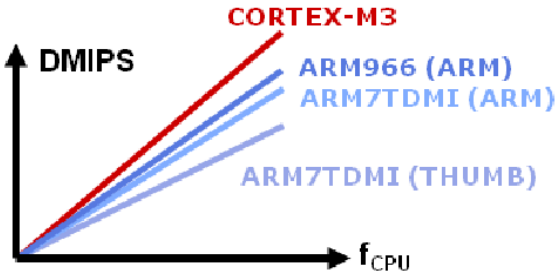
		Operations (privilege out of reset)	Stacks (Main out of reset)
Modes (Thread out of reset)	**Handler** - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged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

Hình 2.5. 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. Để bắt đầu thực hiện, bộ xử lý Cortex đơn giản chỉ cần vector reset và địa chỉ bắt đầu của ngăn xếp để được cấu hình trước khi bạn có thể bắt đầu thực thi chương trình ứng dụng C của bạn. Tuy nhiên, nếu bạn đang sử dụng một hệ điều hành thời gian thực (RTOS) hoặc đang phát triển một ứng dụng đòi hỏi khắc khe về độ an toàn, chip có thể được sử dụng trong chế độ cấu hình nâng cao, nơi chế độ Handler (exceptions và RTOS) chạy trong chế độ đặc quyền và sử dụng ngăn xếp chính (main stack), trong khi mã ứng dụng chạy trong chế độ Thread và không có đặc quyền truy cập và sử dụng ngăn xếp quá trình (process stack). Bằng cách này mã hệ thống và mã ứng dụng được phân vùng và các lỗi trong mã ứng dụng sẽ không làm cho RTOS sụp đổ.

2.3.4 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*

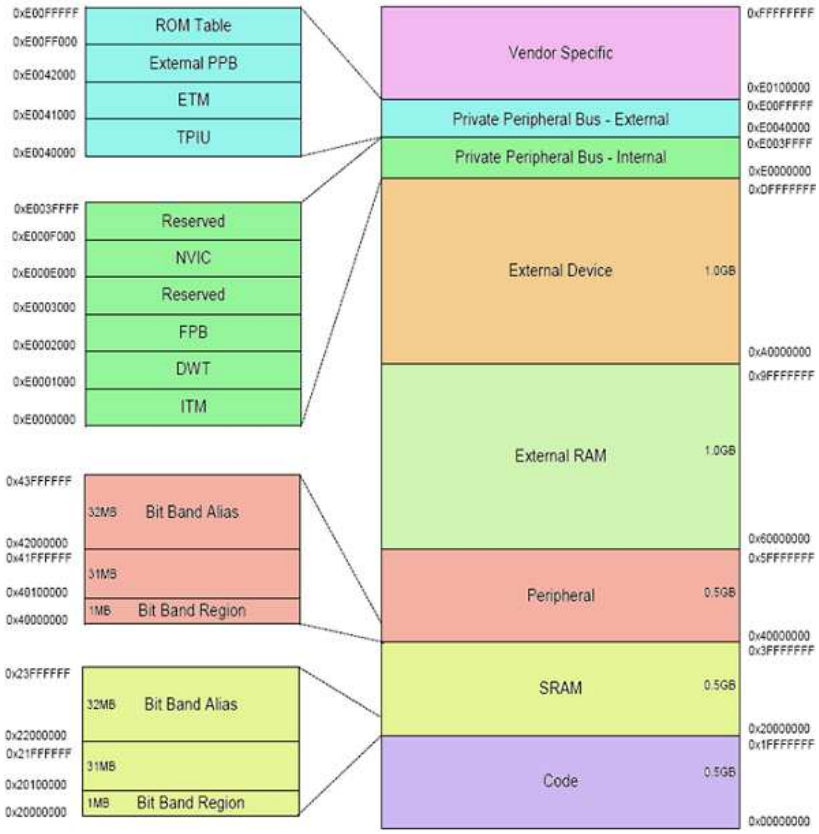
Hình 2.6. Đồ thị biểu diễn hiệu năng của bộ xử lí Cortex

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 ().

Tập lệnh Thumb-2 có: các lệnh rẽ nhánh được cải tiến bao gồm việc kiểm tra và so sánh, các khối thực thi có điều kiện **if/then**, thứ tự byte thao tác dữ liệu, các lệnh trích **byte** và **half word**. CPU Cortex có một tập lệnh phong phú được thiết kế đặc biệt cho trình biên dịch C. Một chương trình Cortex-M3 điển hình sẽ được viết hoàn toàn bằng ANSI C, với tối thiểu các từ khoá non-ANSI và chỉ có bảng véc tơ ngắt được viết bằng Assembler.

2.3.5 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.



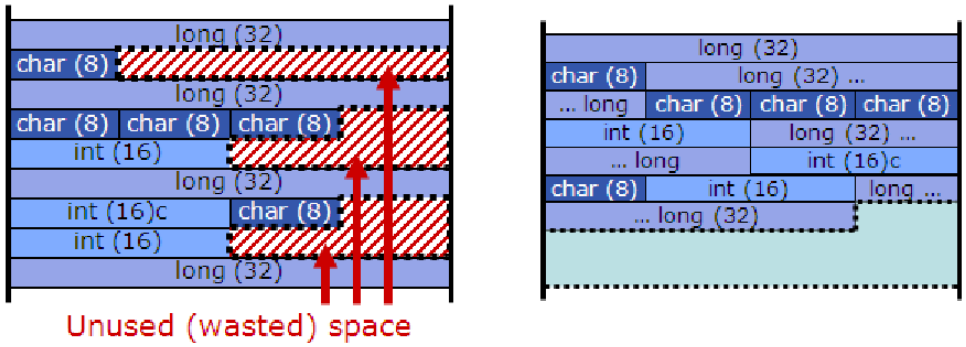
Hình 2.6. Bản đồ bộ nhớ tuyến tính 4Gbyte của bộ xử lí Cortex-M3

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.

2.3.6 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).



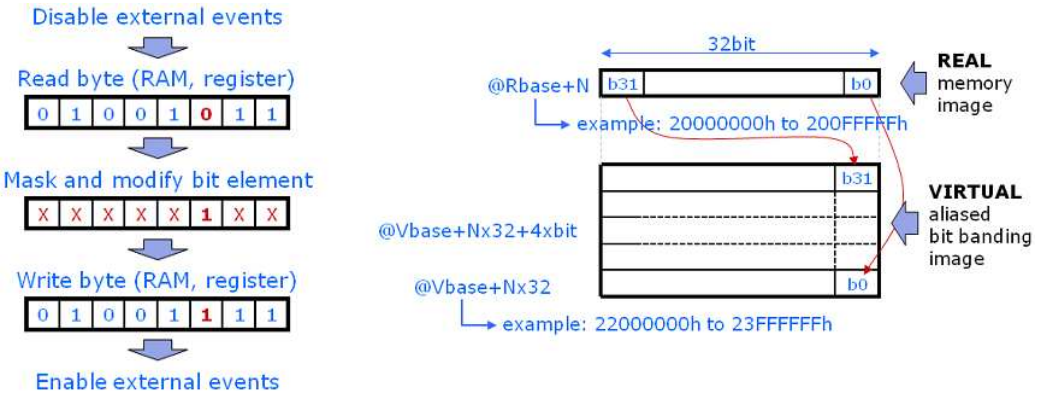
Hình 2.7. Khả năng truy cập bộ nhớ không xếp hàng của bộ xử lí Cortex-M3 so với các phiên bản CPU ARM trước đó

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.

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.

2.3.7 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.

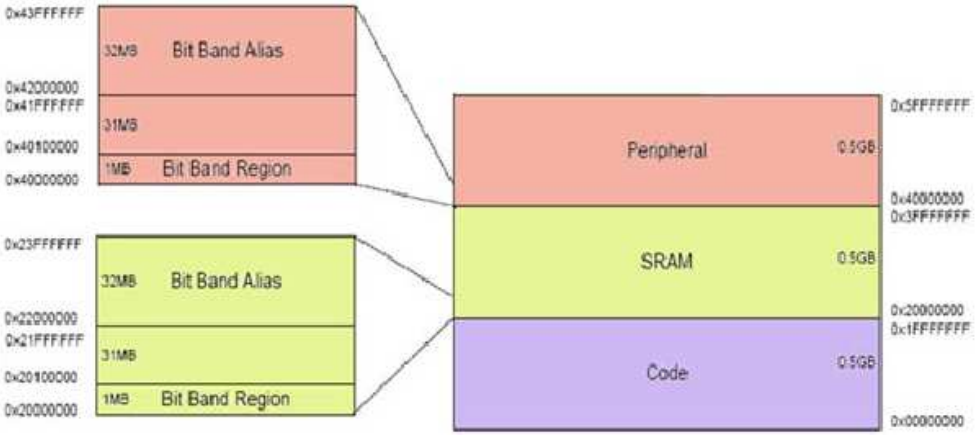


Hình 2.8. Thao tác đọc sửa đổi ghi của CPU ARM7 và ARM9 và kĩ thuật dải Bit của bộ xử lí Cortex-M3

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 2.9. 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

Cho một ví dụ thực tế, thanh ghi dữ liệu đầu ra GPIO (General Purpose I/O) được ghi vào để thiết lập và xoá các đường I/O riêng biệt. Địa chỉ vật lý của thanh ghi đầu ra của port B là 0x40010C0C. Trong ví dụ này, chúng ta muốn có thể thiết lập và xáo 8 bit của word này bằng cách sử dụng công thức trên.

Word address = 0x40010C0C
Peripheral bit band base = 0x40000000
Peripheral bit band Alias base = 0x42000000
Byte offset from bit band base = 0x40010c0c – 0x40000000 = 10c0c
Bit word offset = (0x10c0c x 0x20) +(8x4) = 0x2181A0
Bit Alias address = 0x42000000 + 0x2181A0 = 0x422181A0

Bây giờ chúng ta có thể tạo ra một con trỏ đến địa chỉ này bằng cách sử dụng các dòng lệnh C như sau :

```
#define  PB8      (*((volatile unsigned long*)0x422181A0))    // Port B bit 8
```

Con trỏ này có thể được sử dụng để thiết lập và xoá các bit của cổng I/O này:

```
PB8 = 1;      // led on
```

Mã trên được biên dịch ra ngôn ngữ assembly như sau:

```
MOVS      r0,#0x01

LDR       r1,[pc,#104]
STR       r0,[r1,#0x00]
```

Tắt LED:

```
PB8 = 0;      // led off
```

Tạo ra mã assembly sau đây:

```
MOVS      r0,#0x00
LDR       r1,[pc,#88]
STR       r0,[r1,#0x00]
```

Cả hai thao tác thiết lập và xoá mất ba lệnh 16-bit và với STM32 chạy ở tần số 72 MHz các lệnh này được thực hiện trong 80nsec. Bất kỳ một word trong khu vực bit band của thiết bị ngoại vi và SRAM có thể được định địa chỉ trực tiếp toàn word (word-wide), vì vậy có thể thực hiện việc thiết lập và xoá bằng cách sử dụng phương pháp truyền thống với các lệnh AND và OR:

```
GPIOB→ODR |= 0x00000100;        //LED on
        LDR         r0,[pc,#68]
        ADDS        r0,r0,#0x08
        LDR         r0,[r0,#0x00]
        ORR         r0,r0,#0x100
        LDR         r1,[pc,#64]
        STR         r0,[r1,#0xC0C]

GPIOB→ODR &= !0x00000100;       //LED off
        LDR         r0,[pc,#40]
        ADDS        r0,r0,#0x08
        LDR         r0,[r0,#0x00]
        MOVS        r0,#0x00
        LDR         r1,[pc,#40]
        STR         r0,[r1,#0xC0C]
```

Trường hợp này mỗi thao tác thiết lập và xoá sẽ lấy các phép toán hỗn hợp giữa 16 và 32-bit, điều này phải mất tối thiểu 14 byte cho từng phép toán và ở cùng một tần số 72MHz sẽ mất tối thiểu là 180 nSec. Nếu xem xét tác động của dải bit trên một ứng dụng nhúng điển hình thì việc thiết lập và xóa nhiều bit trong các thanh ghi ngoại vi và sử dụng semaphores (một dạng như cờ dùng để lập trình trong môi trường hệ điều hành) và cờ trong SRAM, rõ ràng kỹ thuật bit band sẽ tiết kiệm đáng kể kích thước mã và thời gian thực hiện.

2.4 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…

2.4.1 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.

2.4.2 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.

2.4.3 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 2.10. 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. Trong phần tiếp theo chúng ta sẽ xem xét cơ cấu ngắt của Cortex và sử dụng SysTick để tạo ra một ngắt ngoại lệ (exception) đầu tiên trên STM32.

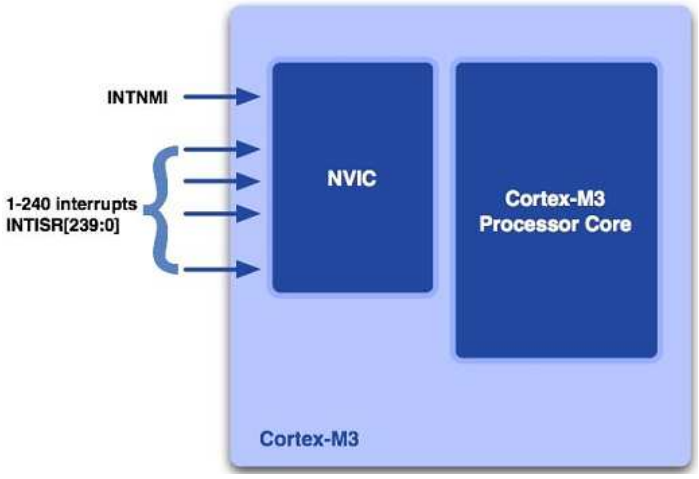
2.4.4 Xử lí ngắt (Interrupt Handling)

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). CPU ARM7 và ARM9 có hai đường ngắt: ngắt nhanh (fast interrupt-FIQ) và ngắt đa dụng (general purpose interrupt hay còn gọi là interrupt request-RIQ). Hai đường tín hiệu ngắt này phục vụ tất cả các nguồn ngắt bên trong một vi điều khiển, trong khi kỹ thuật được sử dụng là như nhau, nhưng việc thực hiện lại khác biệt giữa các nhà sản xuất chip. Cơ cấu ngắt của ARM7 và ARM9 gặp phải hai vấn đề. Trước hết nó không phải là xác định; thời gian để thực hiện việc chấm dứt hay hủy bỏ một lệnh đang thực thi khi xảy ra ngắt là không xác định. Điều này có thể không là vấn đề trở ngại cho nhiều ứng dụng, nhưng nó là một vấn đề lớn trong điều khiển thời gian thực. Thứ hai, cơ cấu ngắt của ARM7 và ARM9 không hỗ trợ ngắt lồng nhau (nested interrupts); cần có sự hỗ trợ của phần mềm: sử dụng macro Assembler hoặc một RTOS. Một trong những tiêu chí quan trọng của lõi Cortex là khắc phục những hạn chế này và cung cấp một cấu trúc ngắt chuẩn cực kỳ nhanh chóng và xác định (extremely fast and deterministic).

2.4.5 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... Vì vậy, mã ứng dụng và hệ điều hành có thể dễ dàng được chuyển từ vi điều khiển này sang vi điều khiển khác và lập trình viên khác không cần phải tìm hiểu một tập các thanh ghi hoàn toàn mới. NVIC cũng được thiết kế để có một độ trễ khi đáp ứng ngắt rất thấp. 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 trên STM32 có 16 cấp độ ưu tiên ngắt. Cấu trúc ngắt NVIC được thiết kế để hoàn toàn lập trình bằng ANSI C và không cần bất kỳ macro Assembler hoặc các chỉ dẫn (directives) non-ANSI.



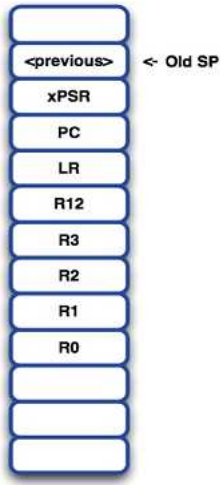
Hình 2.11. Cấu trúc của NVIC trong bộ xử lí Cortex

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).

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

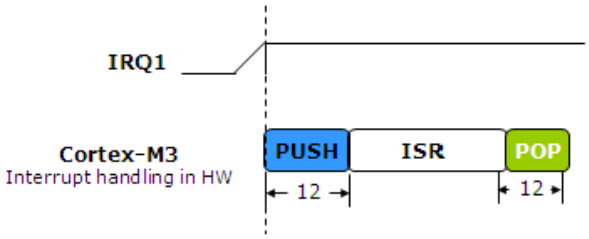
Khi một ngắt được sinh ra bởi một thiết bị ngoại vi, NVIC sẽ kích khởi CPU Cortex phục vụ ngắt. Khi CPU Cortex đi vào chế độ ngắt của nó, nó sẽ đẩy một tập các thanh ghi vào vùng ngăn xếp (stack). Thao tác này được thực hiện trong vi chương trình (microcode), vì vậy không cần viết thêm bất kì lệnh nào 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 2.12. Stack frame trong chế độ ngắt

Stack frame bao gồm thanh ghi trạng thái chương trình (Program Status Register), thanh ghi bộ đếm chương trình (program counter) và thanh ghi liên kết (link register). Stack frame dùng để lưu ngữ cảnh mà CPU Cortex đang chạy. Các thanh ghi từ R0 - R3 cũng được lưu. Trong chuẩn giao diện nhị phân ARM (ARM binary interface standard) các thanh ghi này được sử dụng để truyền tham số, do đó thao tác lưu trữ các thanh ghi này sẽ cung cấp cho chúng ta một bộ thanh ghi sẵn sàng được sử dụng bởi trình phục vụ ngắt (Interrupt

Service Routine-ISR). Thanh ghi cuối cùng cũng được lưu là R12; thanh ghi này được sử dụng bởi bất kỳ mã chương trình nào đang chạy khi một cuộc gọi hàm được thực hiện. Ví dụ, nếu bạn cho phép tính năng kiểm tra ngăn xếp (stack) trong trình biên dịch, mã chương trình được thêm vào khi biên dịch ra sẽ sử dụng R12 nếu nó cần một thanh ghi CPU. 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 2.13. Đáp ứng thời gian khi một ngắt bất kì xảy ra của Cortex-M3

2.4.5.2 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.

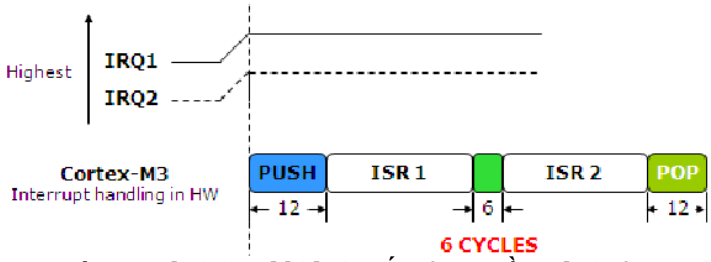
2.4.5.2.1 Quyền ưu tiên ngắt (Interrupt Pre-emption)

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. Trong trường hợp này ngắt đang chạy sẽ bị dừng và một khung ngăn xếp mới (new stack frame) được lưu lại, thao tác này chỉ mất 12 chu kỳ sau đó ngắt có mức ưu tiên cao hơn sẽ chạy. Khi ngắt có mức ưu tiên cao thực hiện xong, dữ liệu lưu trên

ngăn xếp trước đó sẽ được tự động lấy ra (automatically POPed) và ngắt ưu tiên thấp hơn có thể tiếp tục thực hiện.

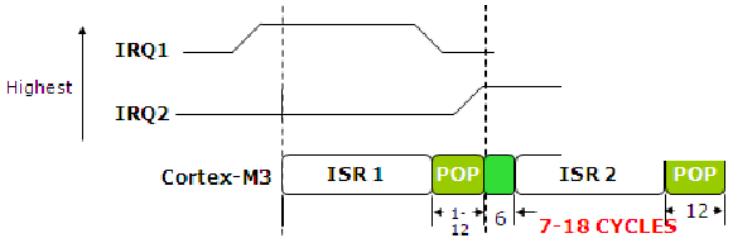
2.4.5.2.2 Kỹ thuật Tail Chaining trong NVIC

Nếu một ngắt có mức ưu tiên cao đang chạy và đồng thời một ngắt có mức ưu tiên thấp hơn cũgn 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.



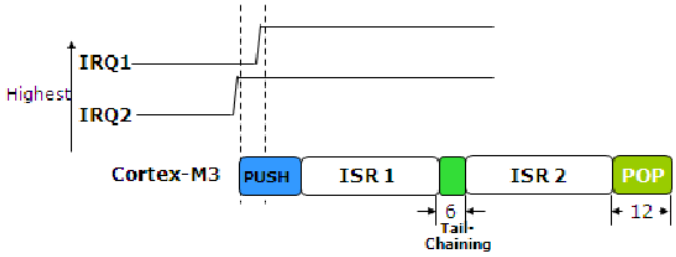
Hình 2.14. Đáp ứng thời gian khi hai ngắt xảy ra đồng thời của Cortex-M3

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 2.15. 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 2.16. 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.

2.4.5.3 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 mà bạn 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.

2.4.5.3.1 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 Interrupt
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 2.17. 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. Trong phần mềm, bảng vector thường được giữ trong chương trình khởi động bằng cách định vị các địa chỉ trình phục vụ ngắt tại địa chỉ nền của bộ nhớ.

```
                AREA        RESET, DATA, READONLY
                EXPORT      __Vectors

__Vectors       DCD     __initial_sp        ; Top of Stack
                DCD     Reset_Handler           ; Reset Handler
                DCD     NMI_Handler             ; NMI Handler
                DCD     HardFault_Handler       ; Hard Fault Handler
                DCD     MemManage_Handler       ; MPU Fault Handler
                DCD     BusFault_Handler        ; Bus Fault Handler
                DCD     UsageFault_Handler      ; Usage Fault Handler
                DCD     0                               ; Reserved
                DCD     0                               ; Reserved
                DCD     0                               ; Reserved
                DCD     0                               ; Reserved
```

```
DCD     SVC_Handler             ; SVCall Handler
DCD     DebugMon_Handler              ;Debug Monitor Handler
DCD     0                               ; Reserved
DCD     PendSV_Handler          ; PendSV Handler
DCD     SysTick_Handler         ; SysTick Handler
```

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

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

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 2.18. Các thanh ghi trạng thái và điều khiển của NVIC

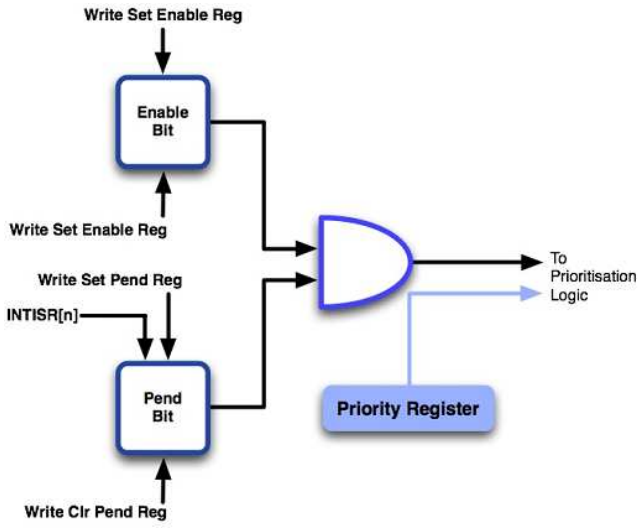
Các ngắt đặc biệt bên trong Cortex được cấu hình thông qua các thanh ghi điều khiển và thanh ghi cấu hình mức ưu tiên của hệ thống, trong khi đó các thiết bị ngoại vi người dùng được cấu hình bằng cách sử dụng các thanh ghi IRQ (Interrupt Request). Ngắt của SysTick là một ngắt đặc biệt bên trong Cortex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

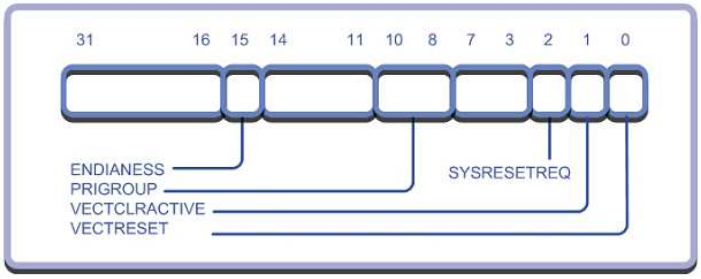
Mỗi thiết bị ngoại vi được điều khiển bởi các khối thanh ghi IRQ. Mỗi ngoại vi có một bit cho phép ngắt. Những bit nằm trên hai thanh ghi cho phép ngắt có chiều dài là 32-bit. Bên cạnh đó cũng có các thanh ghi tương ứng để cấm bất kì một nguồn ngắt. Ngoài ra NVIC cũng bao gồm các thanh ghi báo chờ (pending) và kích hoạt (active) cho phép xác định tình trạng hiện tại của một nguồn ngắt.



Hình 2.19. 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 2.20. Thanh ghi điều khiển reset và ngắt ở mức ứng dụng

PRIGROUP (3 Bits)	Binary Point (group.sub)		Preempting Priority (Group Priority)		Sub-Priority	
			Bits	Levels	Bits	Levels
011	4.0	gggg	4	16	0	0
100	3.1	ggga	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

Hình 2.21. 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;
```

2.5 Các chế độ năng lượng

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

2.5.1 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

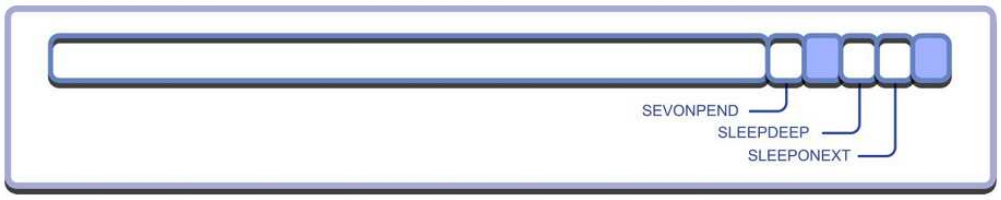
thực thi lệnh WFI, lõi Cortex sẽ tiếp tục thực hiện và phục vụ ngắt đang chờ xử lý. Khi trình phục vụ ngắt-ISR kết thúc, sẽ có hai khả năng xảy ra. Trước tiên, CPU Cortex có thể trở về từ ISR này và tiếp tục thực hiện chương trình ứng dụng nền như bình thường. Bằng cách đặt bit **SLEEPON EXIT** trong 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.

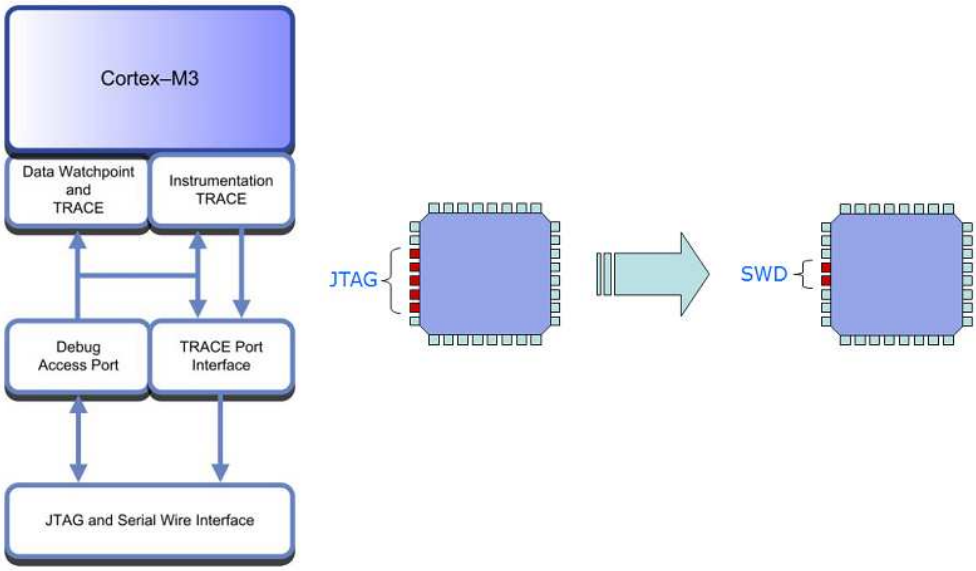


Hình 2.22. Thanh ghi điều khiển hệ thống dùng để cấu hình các chế độ ngủ của vi xử lí Cortex

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.

2.5.2 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.



Hình 2.23. Hệ thống gỡ lỗi CoreSight bên trong Cortex

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

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.

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 3

PHẦN CỨNG CƠ BẢN CHO MỘT THIẾT KẾ THỰC TẾ

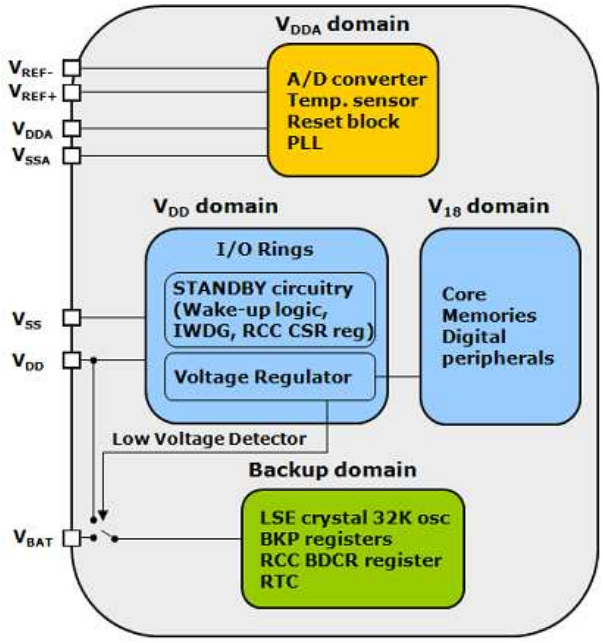
Một thiết kế cơ bản với STM32 cần rất ít các phụ kiện bên ngoài. Để STM32 có thể hoạt động chỉ cần một khối cấp nguồn. STM32 có sẵn một bộ dao động nội RC và một mạch reset nội. Phần này sẽ xem xét các phần cứng chính cần để xây dựng một thiết kế thực tế.

3.1 Kiểu đóng gói chip và kiểu chân linh kiện

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

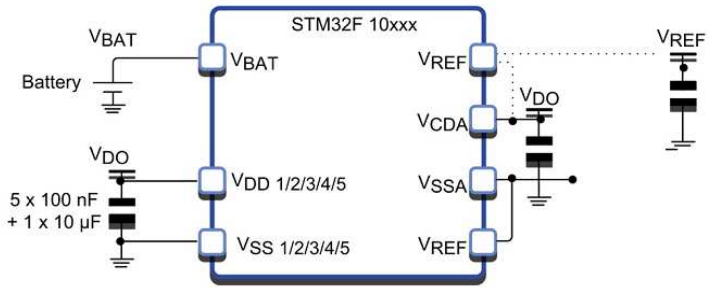
3.2 Nguồn cấp điện

STM32 đòi hỏi một nguồn cung cấp trong khoảng 2.0V đến 3.6V. Một bộ ổn áp nội được sử dụng để tạo ra nguồn cung cấp 1.8V cho lõi Cortex. STM32 có hai nguồn cung cấp năng lượng tùy chọn khác. Bộ RTC (Real Time Clock) và một vài thanh ghi được đặt trong một miền năng lượng riêng biệt, nó có thể được nuôi bởi pin để duy trì dữ liệu trong khi phần còn lại của STM32 đang trong trạng thái ngủ sâu (deep sleep mode). Nếu thiết kế không được sử dụng pin sao lưu, thì chân VBAT phải được kết nối với VDD.



Hình 3.1. Các miền năng lượng bên trong STM32

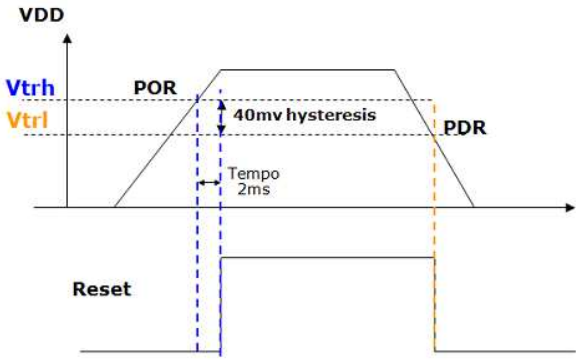
Tùy chọn cung cấp năng lượng thứ hai được sử dụng để cung cấp cho ADC. Nếu ADC được sử dụng, nguồn điện chính VDD được giới hạn trong phạm vi 2.4V đến 3.6V. Đối với chip đóng gói 100 chân, khối ADC có thêm chân điện áp tham khảo VREF+ và VREF-. Chân VREF- phải được kết nối với VDDA và VREF+ có thể thay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



Hình 3.2. Cách bố trí tụ chống nhiễu cho STM32

3.3 Mạch Reset

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

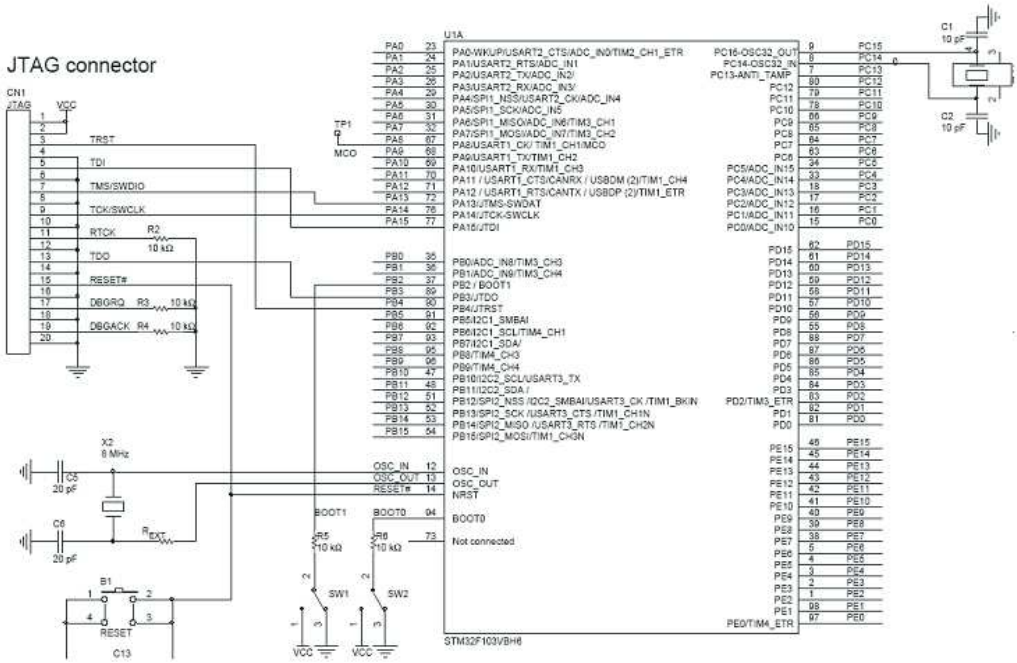


Hình 3.3. Đặc tính của mạch reset bên trong STM32

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

3.3.1 Sơ đồ mạch phần cứng cơ bản

Một mạch reset bên ngoài không cần thiết trong thiết kế của STM32. Tuy nhiên, trong quá trình phát triển chân nRST có thể được kết nối với một nút reset đơn giản, đồng thời chân nRST cũng được kết nối đến cổng JTAG, để công cụ phát triển có thể tạo ra tín hiệu reset vi điều khiển. STM32 có một số các nguồn reset nội có thể phát hiện các điều kiện vận hành bị lỗi và chúng ta sẽ xem xét kĩ hơn trong mục tính năng an toàn của STM32.

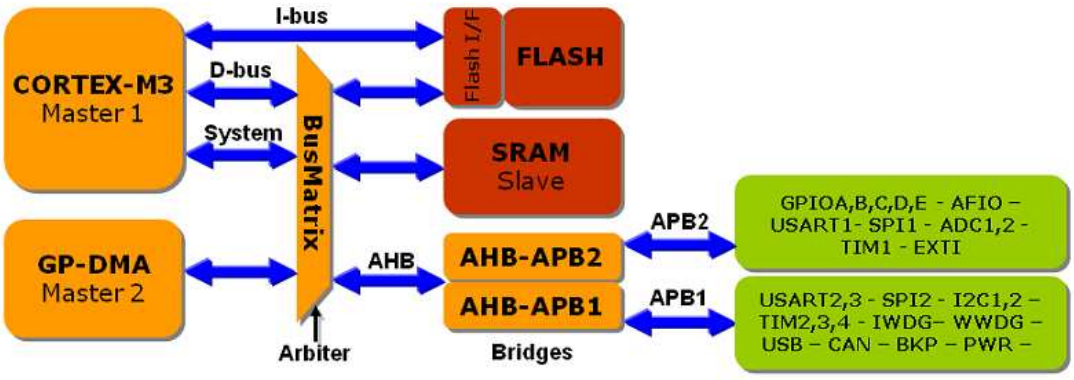


Hình 3.4. Phần cứng tối thiểu cho một thiết kế thực tế dựa trên STM32

Chương 4

KIẾN TRÚC HỆ THỐNG CỦA ARM CORTEX

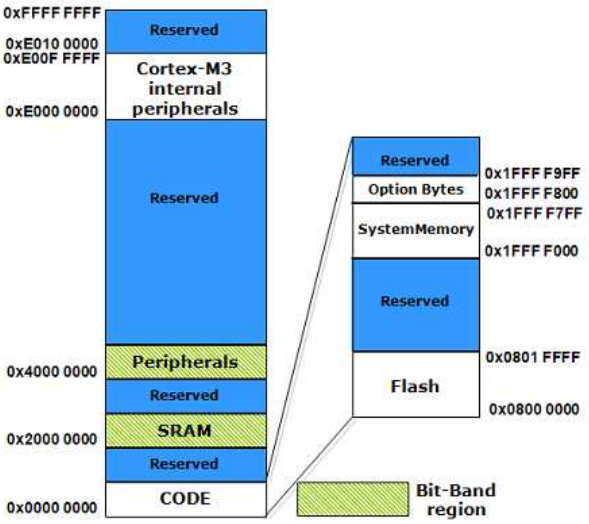
STM32 gồm nhân Cortex kết nối với bộ nhớ FLASH thông qua đường bus lệnh chuyên biệt. Các bus dữ liệu(Cortex Data busses) và hệ thống(Cortex System busses) được kết nối tới ma trận busses tốc độ cao(ARM Advanced High Speed Busses- AHB). SRAM nội kết nối với AHB và đóng vai trò là bộ DMA. Các thiết bị 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. Một điều quan trọng cần nhớ là APB2 và APB1 hoạt động ở tần số khác nhau, APB2 hoạt động ở 72MHz trong khi đó APB1 giới hạn hoạt động ở 36MHz. Cả nhân Cortex và bộ DMA đều đóng vai trò là bus master. Khi có tác vụ xử lý song song thì bộ phân xử sẽ quyết định nhân Cortex hay bộ DMA sẽ được quyền truy cập vào SRAM, APB1 và APB2. Tuy nhiên như đã trình bày ở mục DMA, bộ phân xử luôn đảm bảo 2/3 thời gian truy cập sẽ dành cho bộ điều khiển DMA and 1/3 dành cho nhân Cortex.



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

4.1 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.Vì STM32 là là nhân vi điều khiển dựa trên nền tảng Cortex nên phải tuân theo chuẩn phân vùng bộ nhớ của Cortex.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.



BOOT Mode Selection Pins		Boot Mode	Aliasing
BOOT1	BOOT0		
x	0	User Flash	User Flash is selected as boot space
0	1	SystemMemory	SystemMemory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

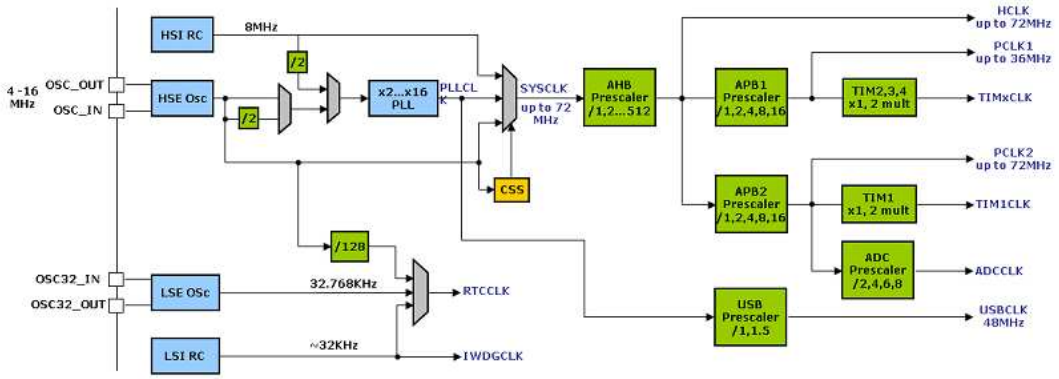
The STM32 memory map follows the Cortex standard. The first 2K of memory is mapped from FLASH, System Memory or SRAM depending on the condition of the boot pins.

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.

Để kích hoạt bootloader của STM32 người dùng phải thiết lập các chân BOOT0 và BOOT1 ở mức điện áp thấp và cao tương ứng. Khi đó sau khi STM32 được khởi động, chương trình sẽ đặt bootloader vào địa chỉ 0x00000000 và thực thi nó thay thi thực thi chương trình của người dùng ở User Flash. Để giao tiếp với bootloader, ST cung cấp một chương trình chạy ở PC, chương trình này có khả năng ghi, xóa vùng nhớ ở User Flash. Ngoài ra chúng ta có thể cấu hình các chân bootpins để ánh xạ SRAM nội vào địa chỉ 0x00000000, cho phép tải xuống và thực thi chương trình ngay tại SRAM. Điều này làm tăng tốc độ tải chương trình và hạn chế số lần ghi vào Flash.

4.2 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.



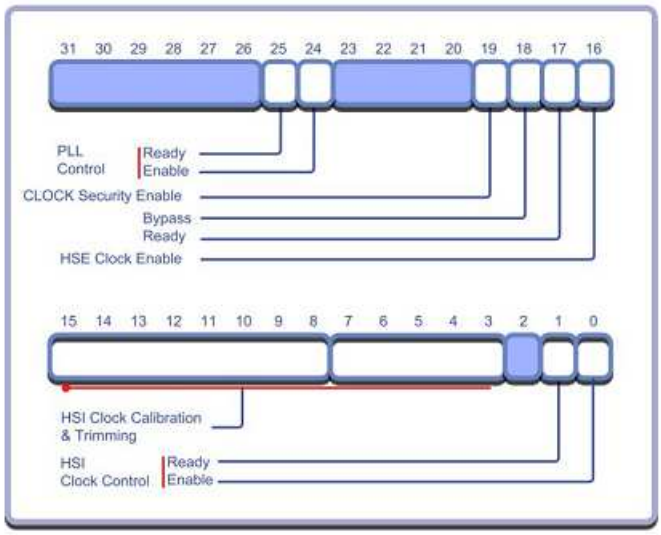
Hình 4.1 STM32 bao gồm 2 bộ tạo xung nhịp nội và 2 bộ tạo xung nhịp ngoại thêm vào đó là bộ vòng khóa pha(Phase Lock Loop-PLL).

Nhân Cortex có thể được cấp xung nhịp từ bộ tạo dao động nội và ngoại, đồng thời từ PLL nội. Như trên hình 4.1, 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).

4.2.1 Vòng Khóa Pha(Phase Lock Loop)

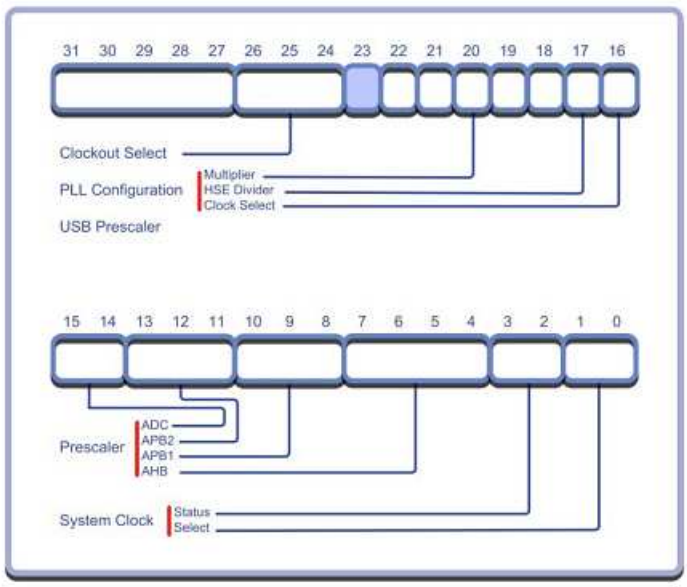
Sau khi hệ thống reset STM32 nhận xung nhịp từ bộ tạo dao động HIS. Tại thời điểm đó các bộ tạo dao động ngoại sẽ bị tắt. Bước đầu tiên để STM32 hoạt động ở mức xung nhịp cao nhất là bật bộ tạo dao động HSE và chờ cho đến khi đi vào hoạt động ổn định.



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;
```

4.2.1.1 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;
```

4.2.2 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. Trong quá trình hoạt động bình thường chúng ta không cần quan tâm đến bộ đệm của Flash. Tuy nhiên chúng ta nên chắc chắn rằng các bộ đệm này phải được kích hoạt trước khi sử dụng PLL như là nguồn xung nhịp của hệ thống. 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

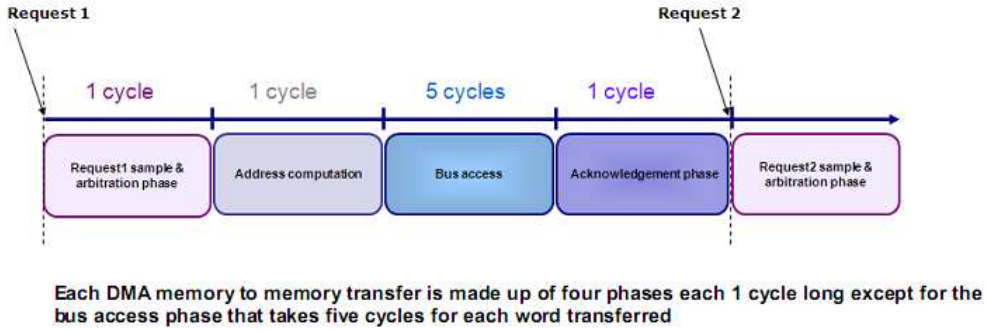
24< SYSCLK <48MHz 1 waitstate

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.

4.2.3 Direct Memory Access

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

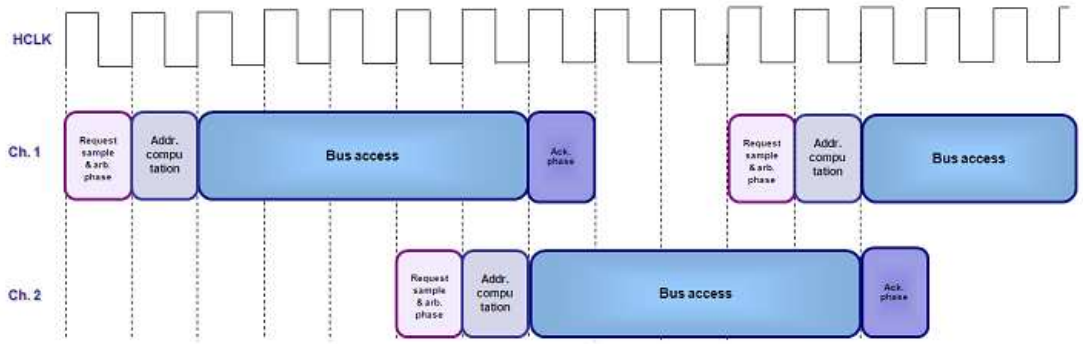


Each DMA memory to memory transfer is made up of four phases each 1 cycle long except for the bus access phase that takes five cycles for each word transferred

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

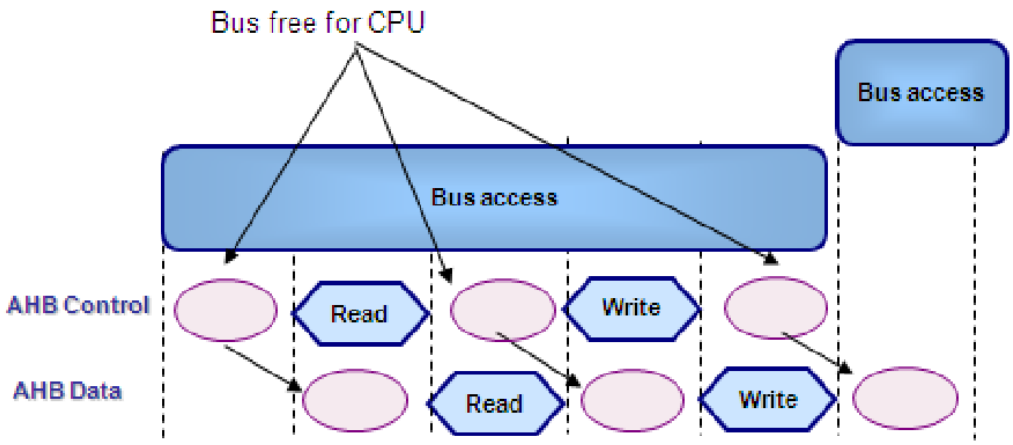
Quá trình truyền dữ liệu gồm 4 giai đoạn: lấy mẫu và phân xử, tính toán địa chỉ, truy cập đường truyền, và cuối cùng là hoàn tất. Mỗi giai đoạn thực hiện trong 1 chu kỳ lệnh, riêng truy cập đường truyền mất 5 chu kỳ lệnh. Ở giai

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.

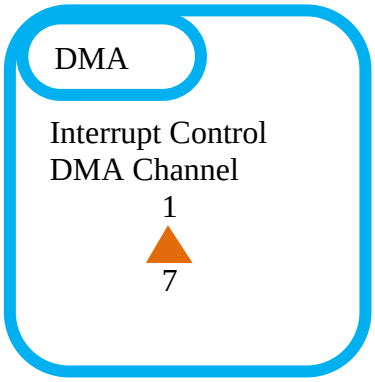
Trong trường hợp trao đổi dữ liệu từ vùng nhớ sang vùng nhớ mỗi kênh DMA chỉ sử dụng đường truyền dữ liệu ở giai đoạn Bus Access và 5 chu kỳ CPU để chuyển 2 bytes dữ liệu. Trong đó 1 chu kỳ để đọc và 1 chu kỳ để ghi, 3 chu kỳ còn lại được bố trí xen kẽ nhằm giải phóng đường dữ liệu cho nhân Cortex. Điều 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 dung 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.

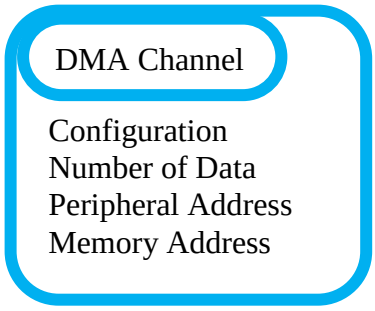


STM32 có 7 bộ DMA độc lập với nhau

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.



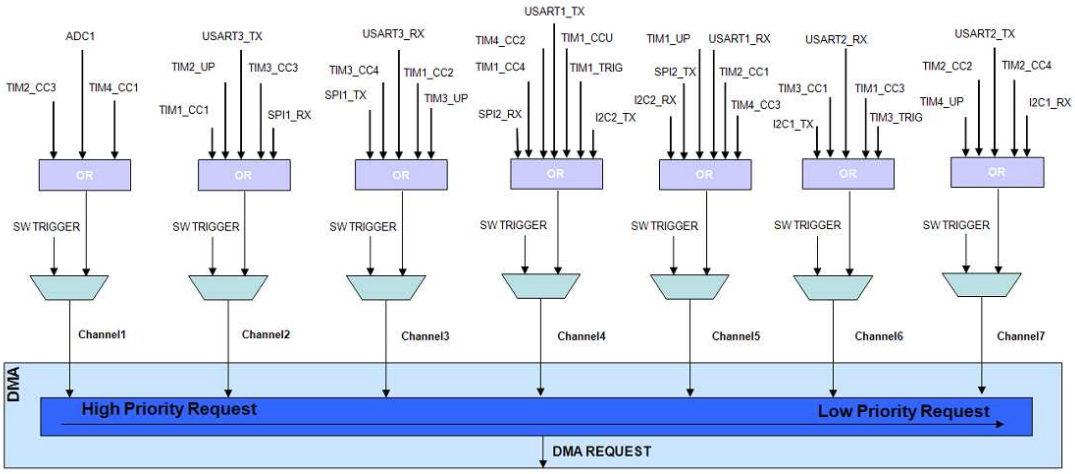
Mỗi bộ DMA có 4 thanh ghi điều khiển, 3 nguồn tín hiệu interrupt: hoàn tất, hoàn tất một nửa, lỗi.

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. Ví dụ với bộ nhớ kích cỡ đơn vị dữ liệu là 32-bit(mất 3 chu kỳ để truyền), chuyển đến ngoại vi UART là 4 đơn vị dữ liệu 8-bit mất 35 chu kỳ thay vì 64 chu kỳ nếu chuyển từng đơn vị dữ liệu 8-bit riêng rẽ. Chúng ta có thể tăng địa chỉ đích hoặc nguồn trong quá trình chuyển dữ liệu. Ví dụ khi

lấy dữ liệu từ bộ ADC, chúng ta có thể tăng địa chỉ vùng nhớ lên để lưu các giá trị từ ADC vào mảng dữ liệu. Trên thanh ghi điều khiển có Transfer Direction Bit cho phép ta cấu hình hướng dữ liệu từ ngoại vi vào vùng nhớ hay ngược lại. Để cấu hình chuyển dữ liệu từ vùng nhớ sang vùng nhớ trên SRAM, ta kích hoạt bit 14 trên thanh ghi điều khiển. 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.



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 5

NGOẠI VI

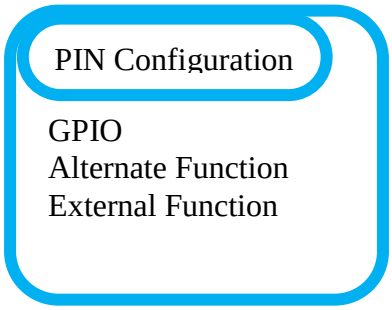
Chương này sẽ giới thiệu các thiết bị ngoại vi trên các phiên bản STM32. Để tiện theo dõi, chúng tôi chia ra thành 2 loại: ngoại vi đa dụng và ngoại vi giao tiếp. Tất cả ngoại vi trên STM32 được thiết kế và dựa trên bộ DMA. Mỗi ngoại vi đều có phần điều khiển mở rộng nhằm tiết kiệm thời gian xử lý của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

5.1 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”.

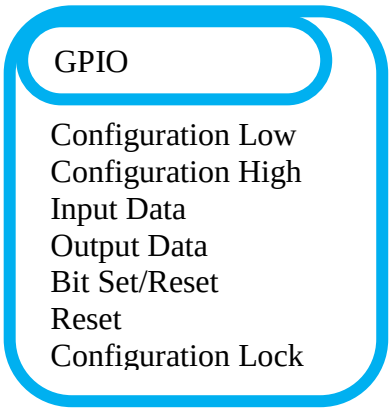
5.1.1 Các cổng I/O đa dụng

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



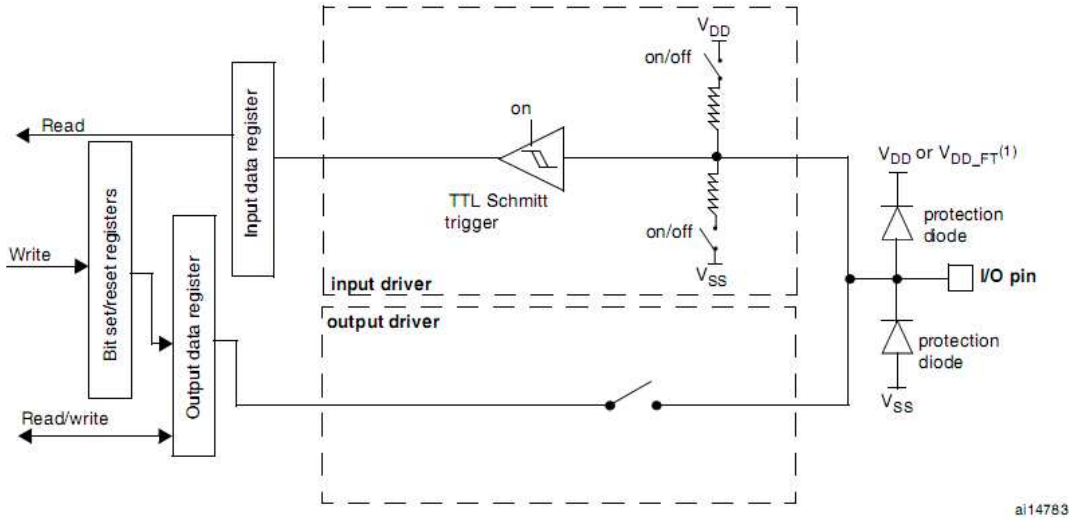
Mỗi chân điều khiển có thể cấu hình như là GPIO hoặc có chức năng thay thế khác. Hoặc mỗi chân có thể cùng lúc là nguồn ngắt ngoại.

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 Mode	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		



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
```

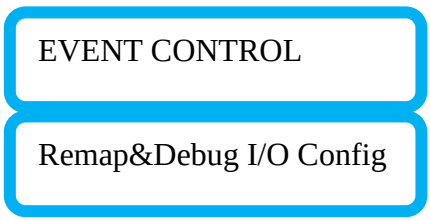
Sau đó đọc lại bit 16 liên tục 2 lần, nếu giá trị trả về lần lượt là 0 và 1 thì thiết lập khóa đã hoàn thành

```
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.

5.1.1.1 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.

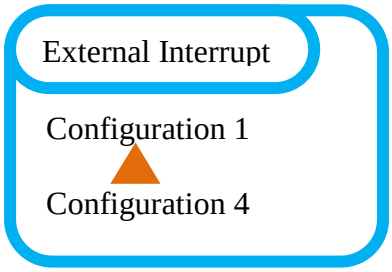
5.1.1.2 Event Out

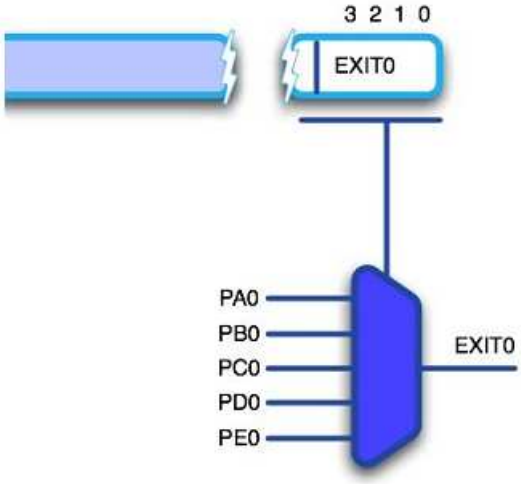
Nhân Cortex có khả năng tạo xung nhịp để "đánh thức" các khối vi điều khiển bên ngoài thoát khỏi trạng thái tiết kiệm năng lượng. Thông thường, xung nhịp này sẽ được nối với chân "Wake up" của vi xử lý STM32 khác. Lệnh SEV Thumb-2 khi được thực thi sẽ tạo ra xung nhịp "Wake up" này. Thanh ghi điều khiển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

5.1.2. 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.





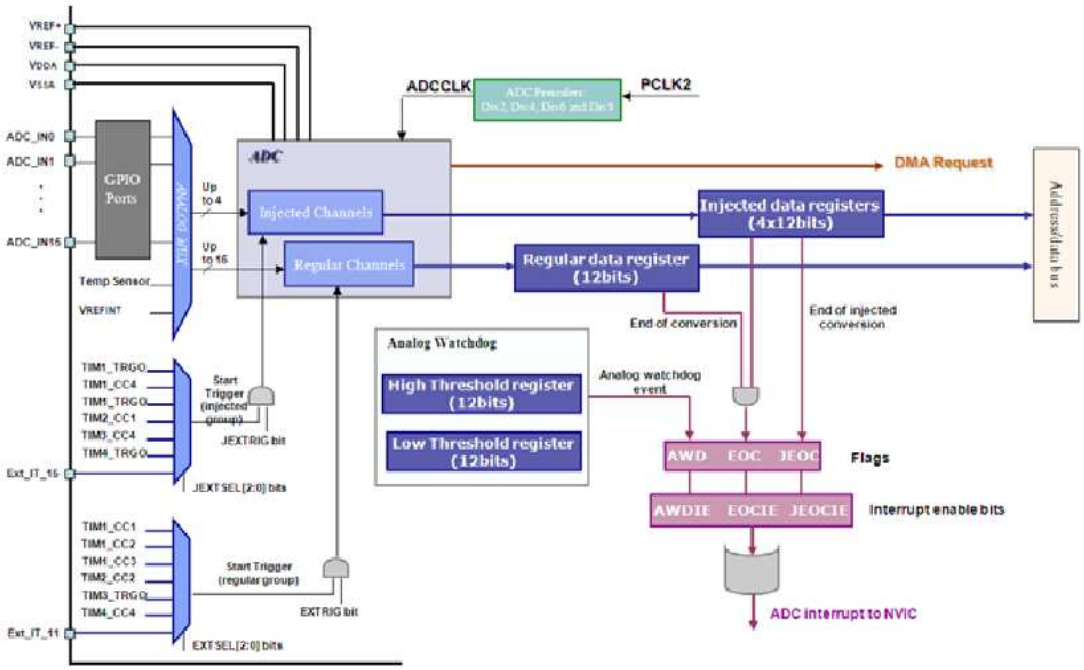
16 ngắt ngoại có thể được ánh xạ tới bất kỳ chân nào của vi xử lý thông qua 4 thanh ghi cấu hình điều khiển. Mỗi ngắt được điều khiển bởi trường 4 bit, đoạn mã sau mô tả cách cấu hình ngắt cho chân GPIO

```
//Map the external interrupts to port pins
AFIO->EXTICR[0] = 0x0000000;
//Enable external interrupt sources
EXTI->IMR  = 0x00000001;
//Enable wake up event
EXTI->EMR = 0x00000000;
```

```
//Select falling edge  trigger souces
EXTI->FTSR = 0x00000001;
//Select resing edge trigger sources
EXTI->RTSR = 000000000;
//Enable interrupt souces in NVIC
NVIC->Enable[0] = 0x00000040;
NVIC->Enable[1] = 0x00000000;
```

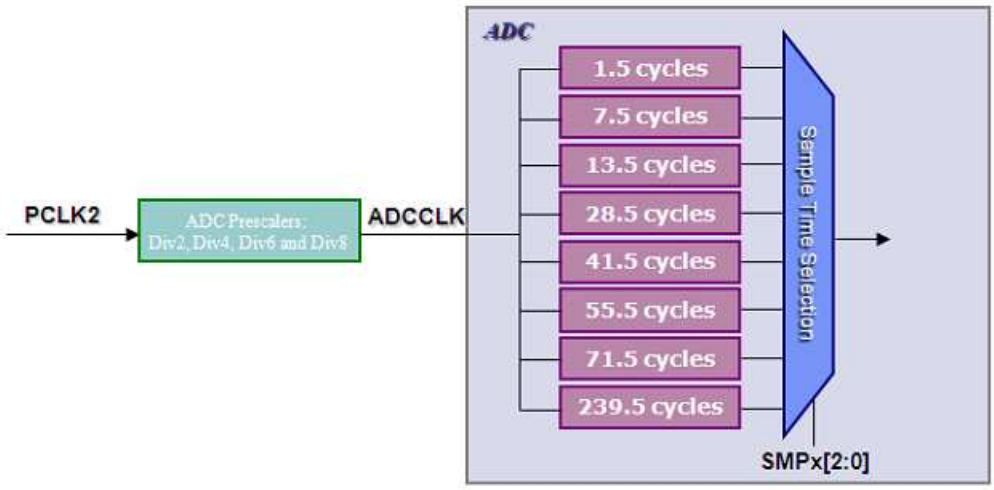
5.1.3 ADC

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

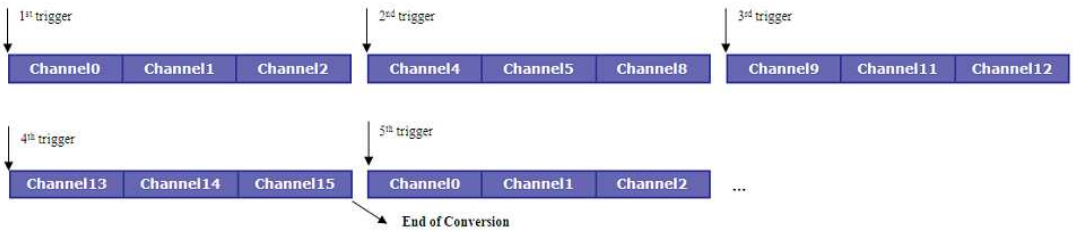


5.1.3.1 Thời gian chuyển đổi và nhóm chuyển đổi

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.

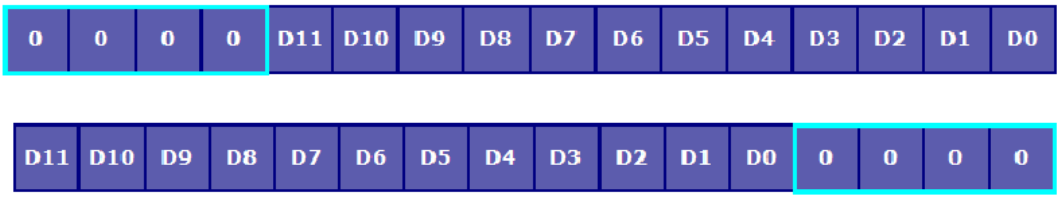


Mỗi bộ ADC có 2 chế độ chuyển đổi: thông thường(regular) và injected. Ở chế độ regular cho phép một hay một nhóm các kênh kết hợp với nhau thực thi tác vụ chuyển đổi. Một nhóm kênh tối đa có thể gồm 16 kênh. Thứ tự chuyển đổi trong nhóm có thể được cấu hình bởi phần mềm, và trong một chu kỳ chuyển đổi 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.



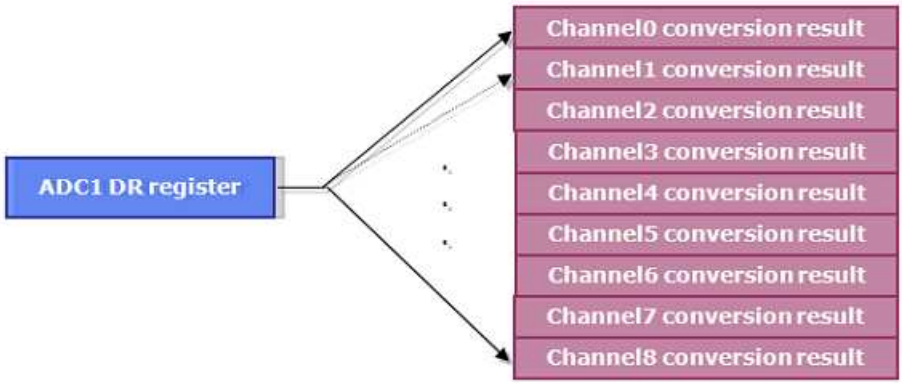
Một nhóm kênh hoạt động ở chế độ Regular có thể liên tục thực hiện quá trình chuyển đổi, hoặc chỉ chuyển đổi khi nhận tín hiệu kích hoạt.

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.



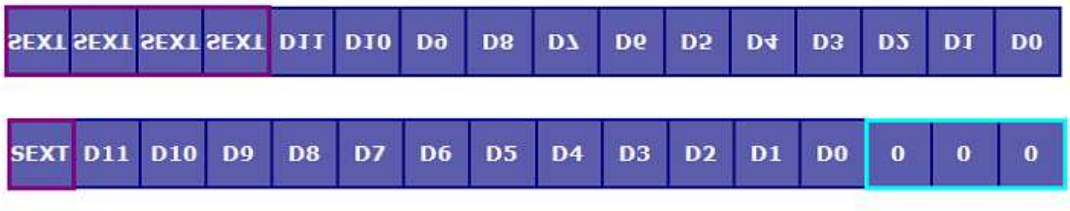
Dữ liệu có thể được canh lề trái hoặc phải trong thanh ghi kết quả

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

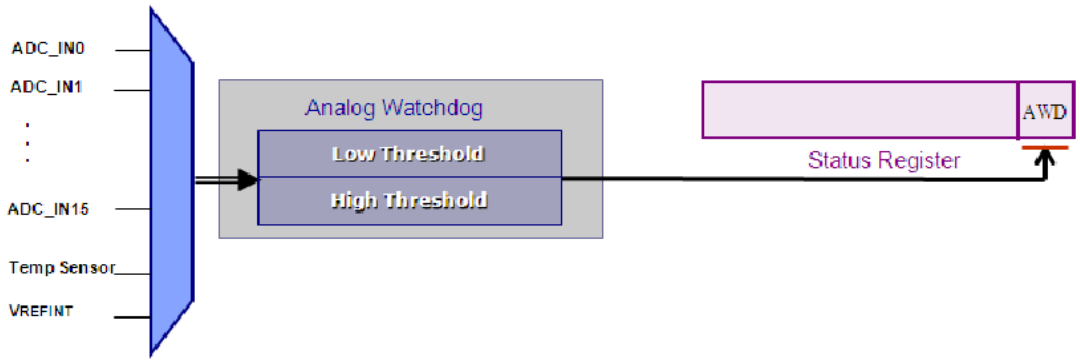
Loại ADC thứ 2 là Injected ADC. Injected ADC là dãy các kênh ADC, tối đa là 4 kênh. Injected ADC có thể được kích hoạt bằng phần mềm hoặc tín hiệu phần cứng. Khi được kích hoạt, Injected ADC với mức ưu tiên cao hơn sẽ tạm ngưng các kênh Regular ADC đang hoạt động. Các kênh Regular ADC chỉ tiếp tục hoạt 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

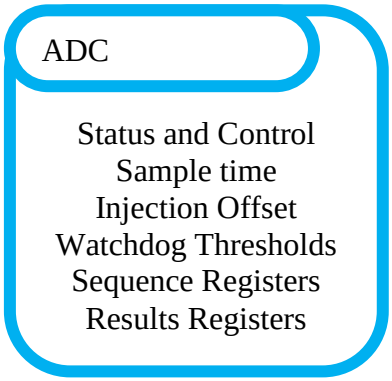
5.1.3.2 Analogue WatchDog

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.



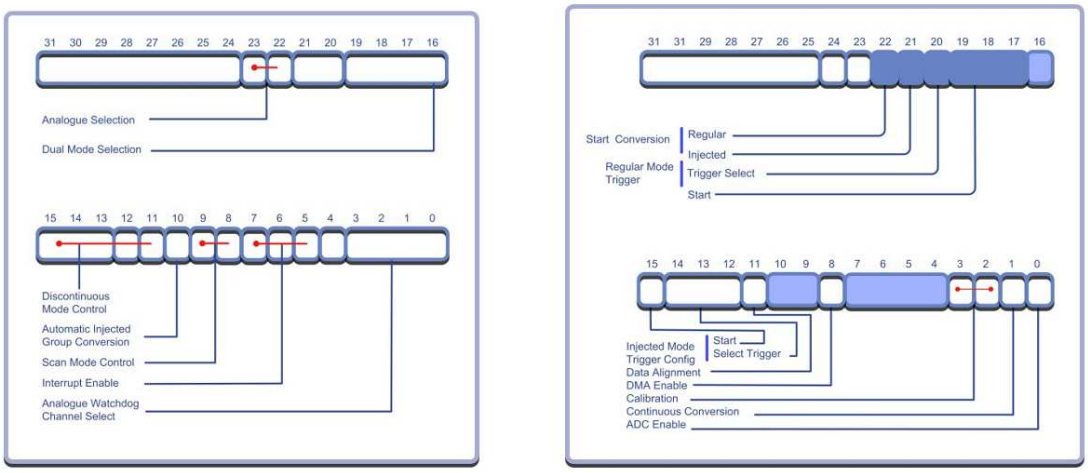
Analogue Watchdog có thể dùng giám sát một hay nhiều kênh ADC với vùng ngưỡng được cấu hình bởi người dùng

5.1.3.3 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.



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

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

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

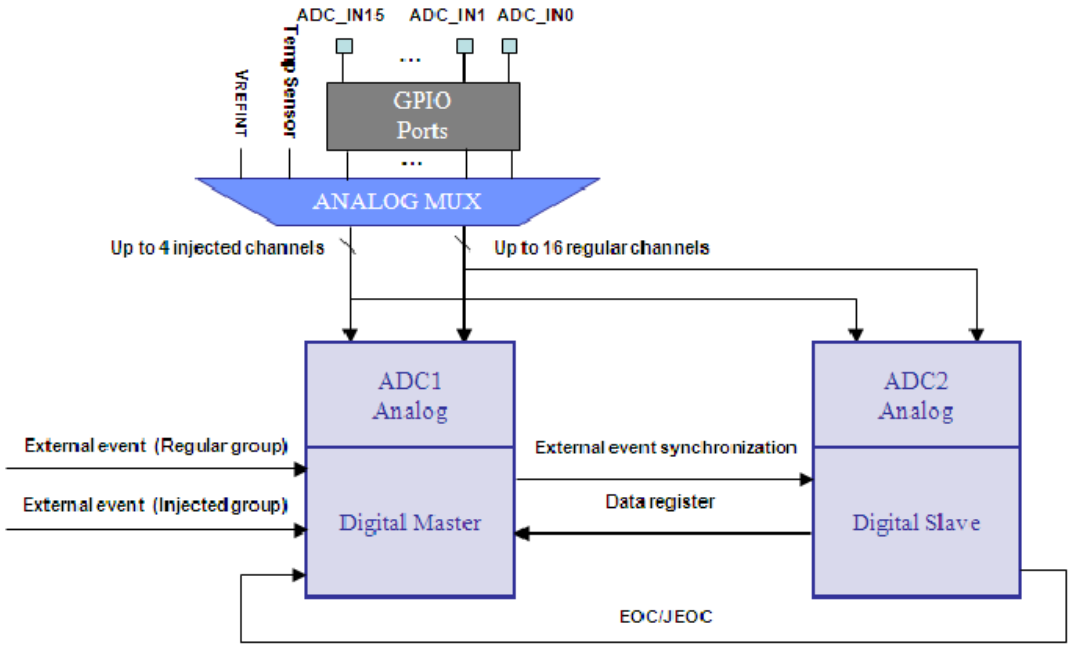
```
DMA_Channel1->CCR = 0x00003A28; //Circular mode, peripheral and memory increased disable
//Load destination address into peripheral register, GPIO port data register
DMA_Channel1->CMAR = (unsigned int) 0x04001244C;
DMA_Channel1->CNDTR = 0x0001; //number of words to transfer
DMA_Channel1->CCR = 0x00000001;//Enable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

Chúng ta kích hoạt chế độ DMA của khối ADC

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

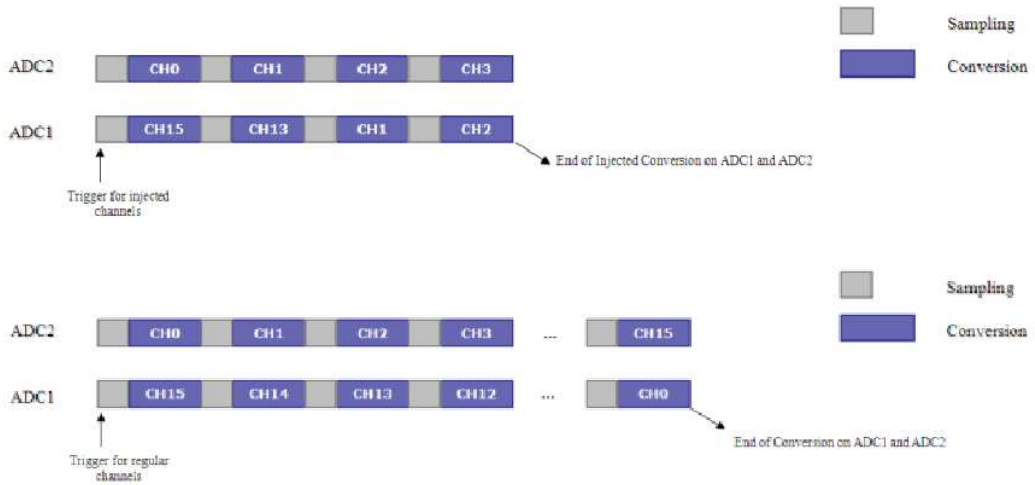
5.1.3.4. Dual mode

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



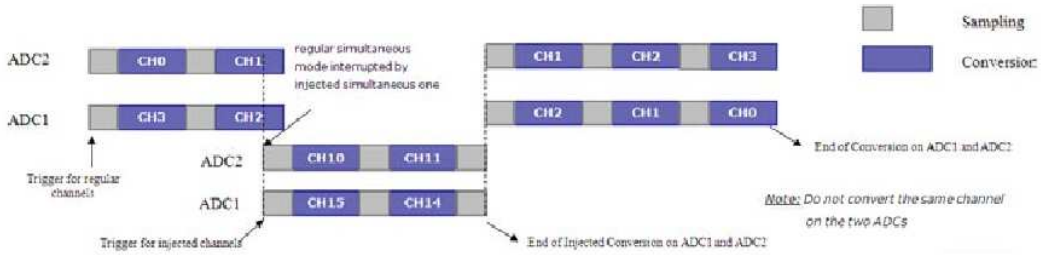
Khi hoạt động ở chế độ Dual, khối ADC2 đóng vai trò phụ đối với ADC1. Khi kết hợp ADC1 và ADC2, chúng ta sẽ có 8 chế độ hoạt động

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



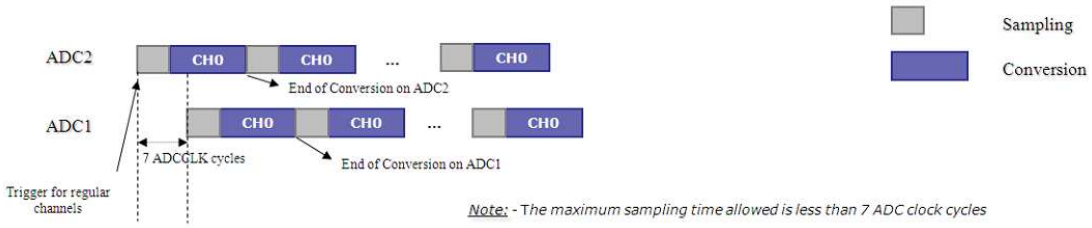
Khi hoạt động ở chế độ này, cùng lúc khối ADC1 và ADC2 sẽ chuyển đổi dữ liệu từ 2 kênh khác nhau. Ví dụ trong các ứng dụng cần theo dõi cùng lúc điện áp và cường độ dòng.

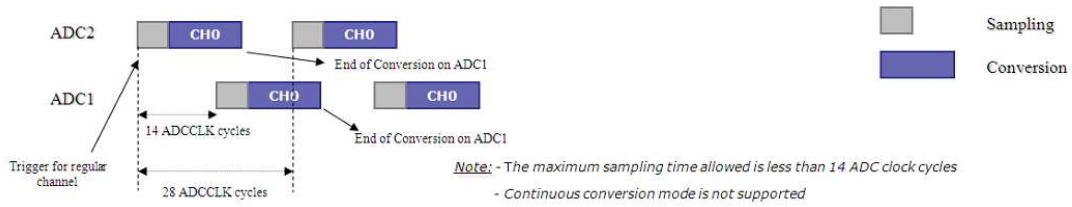
5.3.1.4.2. Cả hai khối cùng hoạt động ở 2 chế độ Regular và Injected xen kẽ



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.

5.3.1.4.3. Hoạt động xen kẽ nhanh và chậm Regular

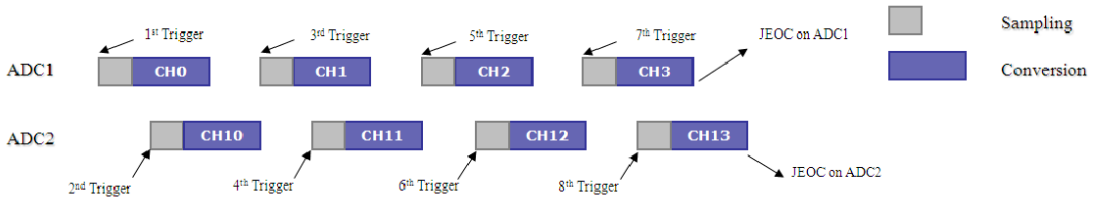




Ở chế độ xen kẽ nhanh, một kênh có thể liên tục chuyển đổi bởi hai khối ADC, thời gian nhỏ nhất để kích hoạt lần chuyển đổi kế tiếp là 7 chu kỳ xung nhịp của ADC. Ở chế độ xen kẽ chậm khoảng cách thời gian tối thiểu là 14 chu kỳ

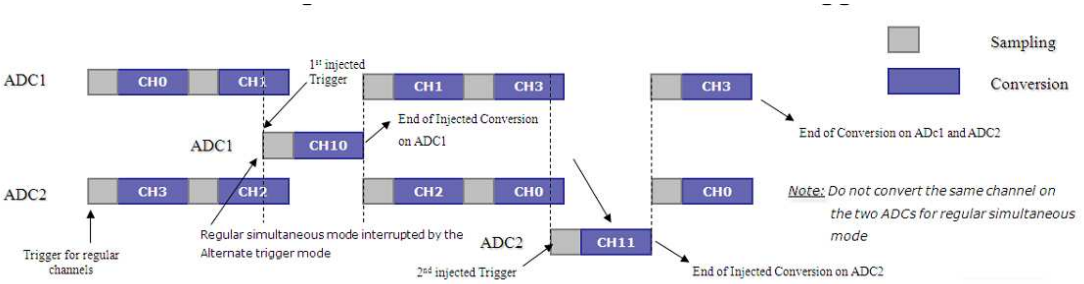
xung nhịp. Hai chế độ kết hợp này làm tăng hiệu suất chuyển đổi của khối ADC.

5.3.1.4.4. Chế độ kích hoạt thay thế



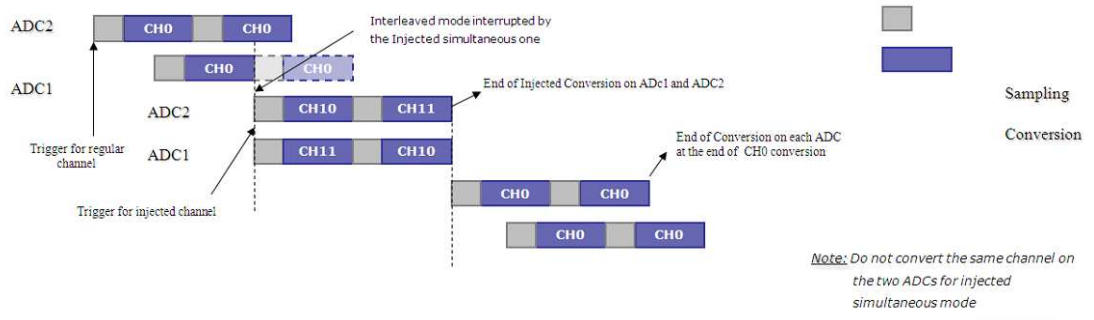
Ban đầu phần cứng sẽ kích hoạt kênh đầu tiên trong nhóm chuyển đổi Injected của khối ADC1, sau đó sẽ kích hoạt tiếp nhóm Injected của ADC2. Cứ như vậy liên tục và xen kẽ.

5.3.1.4.5. 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.

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



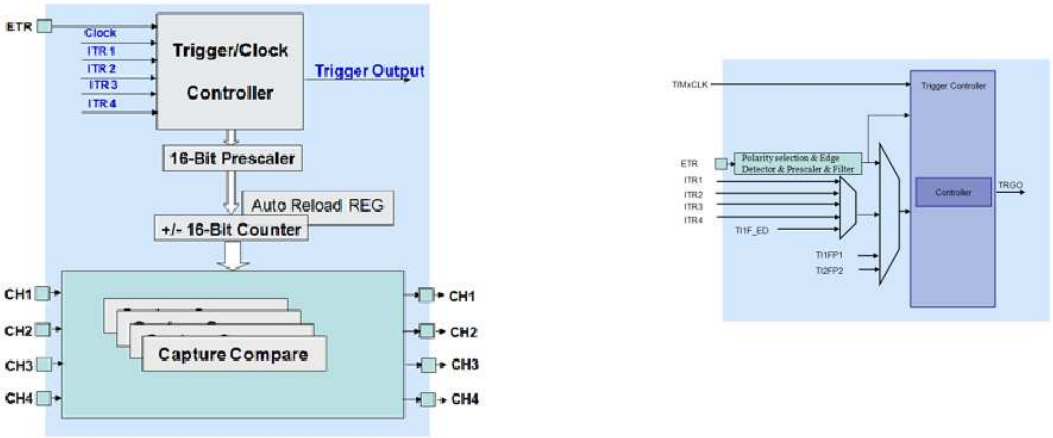
Hai khối ADC1 và ADC2 hoạt động ở chế độ Regular xen kẽ nhau thì được kích hoạt chuyển sang hoạt động ở chế độ đồng bộ Injected. Lưu ý là: khi ở chế độ xen kẽ Regular, cả hai kênh ADC1 và ADC2 có thể chuyển đổi chung trên cùng một kênh, tuy nhiên khi sang chế độ đồng bộ Injected, thì kênh được sử dụng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

5.1.4. 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.

5.1.4.1. 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.

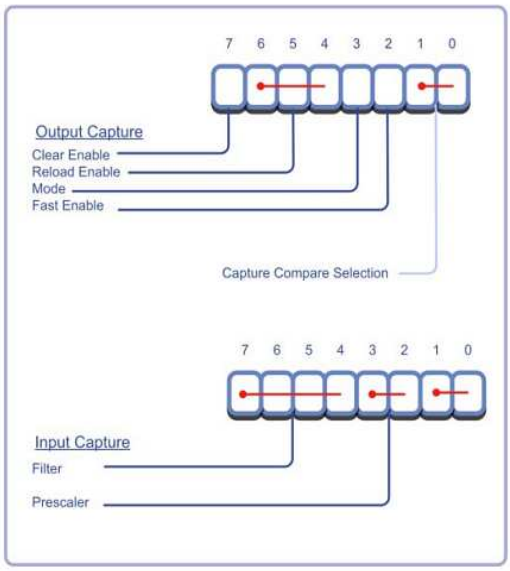


4 khối định thời với các thanh ghi 16-bit Prescaler, 16-bit Counter và Auto-reload. Xung nhịp hoạt động có thể lấy từ đồng hồ hệ thống, tín hiệu ngoại và các khối định thời khác

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

5.1.4.1.1 Khối Capture/Compare

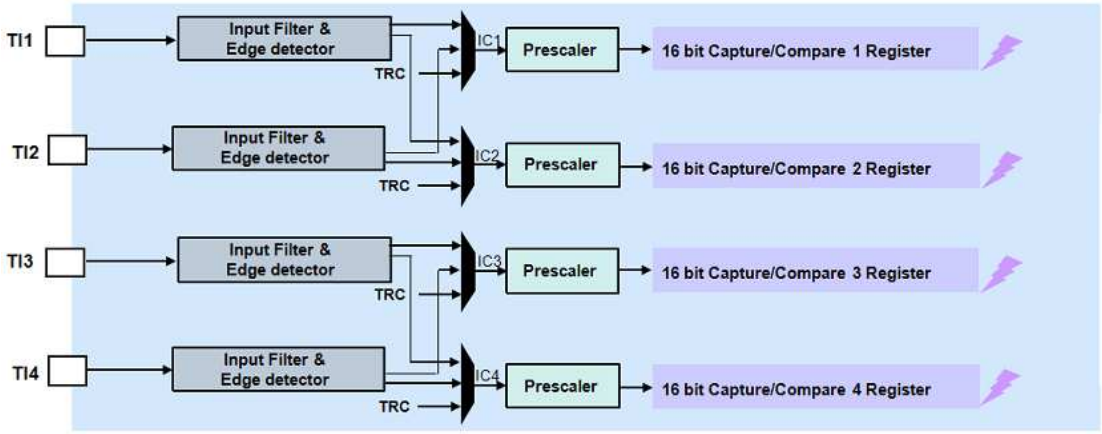
Mỗi kênh Capture/Compare được điều khiển bởi duy nhất một thanh ghi. Chức năng của thanh ghi này có thể thay đổi tùy thuộc cấu hình. Ở chế độ Capture, thanh ghi này có nhóm các bit đảm nhận thiết lập lọc dữ liệu đầu vào và chế độ đánh giá các ngõ PWM. Ở chế độ Compare, STM32 cung cấp hàm chuẩn so sánh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

5.1.4.1.2 Khối Capture

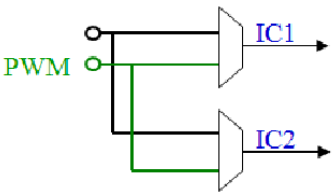
Một khối Capture cơ bản gồm có bốn kênh vào để cấu hình bộ phát hiện xung(Edge Detector). Khi một xung lên(rising edge) hay xung cạnh xuống(falling edge) được phát hiện, bộ đếm hiện thời của sẽ được cập nhật vào các thanh ghi 16-bit Capture/Compare. Khi sự kiện capture xảy ra bộ đếm có thể được khởi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

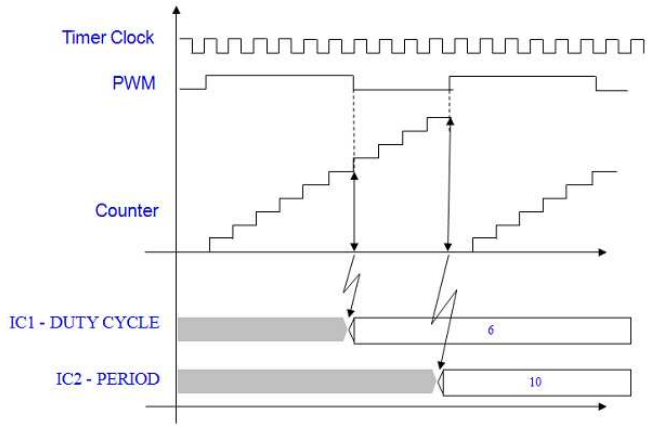


4 kênh vào của khối Capture có các bộ lọc dữ liệu và phát hiện xung cạnh riêng. Khi sự kiện capture được nó có thể được dùng để kích hoạt một sự kiện DMA khác.

5.1.4.1.3 Chế độ PWM Input

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





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

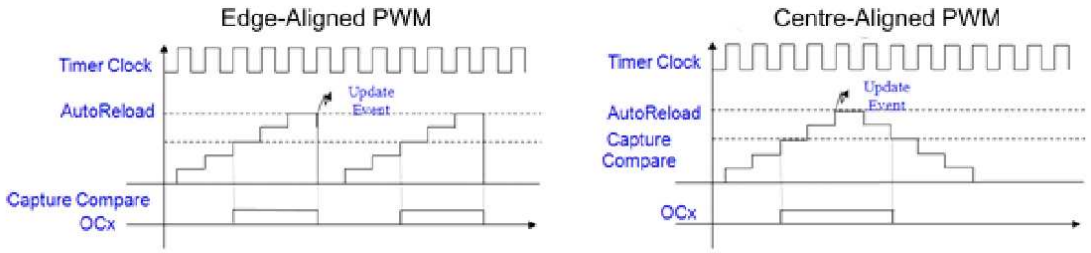
```
M3->CR1     =     0x00000000;//default
TIM3->PSC   =     0x000000FF;//set max prescale
TIM3->ARR   =     0x00000FFF;//set max reload count
```

```
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.

5.1.4.1.4 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.

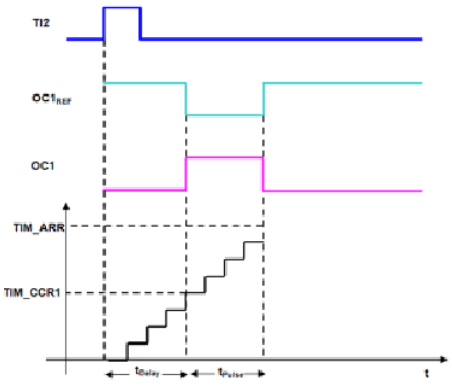


Mỗi khối Timer đều có khả năng tạo ra các xung PWM với độ lệch chu kỳ có thể được cấu hình edge-aligned hoặc centre-aligned tính theo thời điểm cập nhật giá trị của thanh ghi Reload.

```
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
```

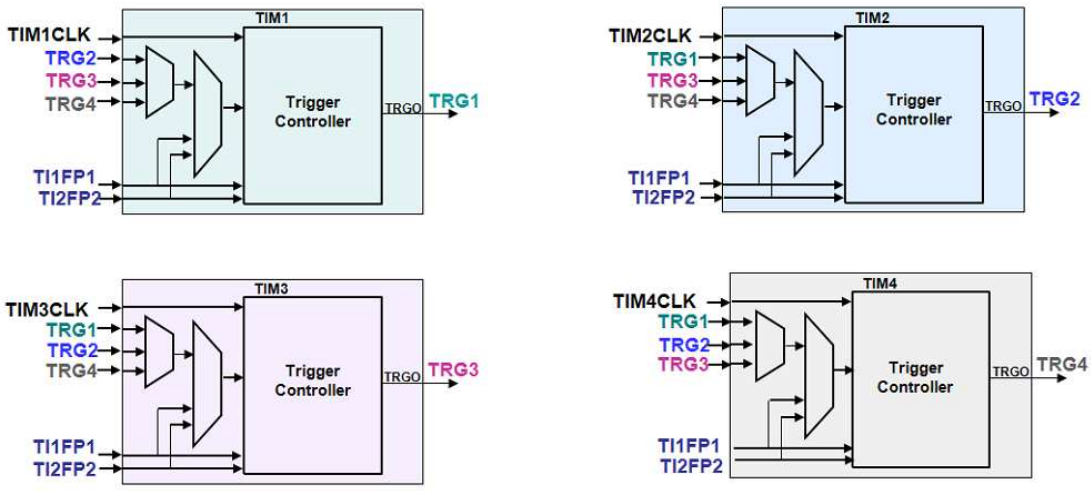
5.1.4.1.5 Chế độ One Pulse

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.



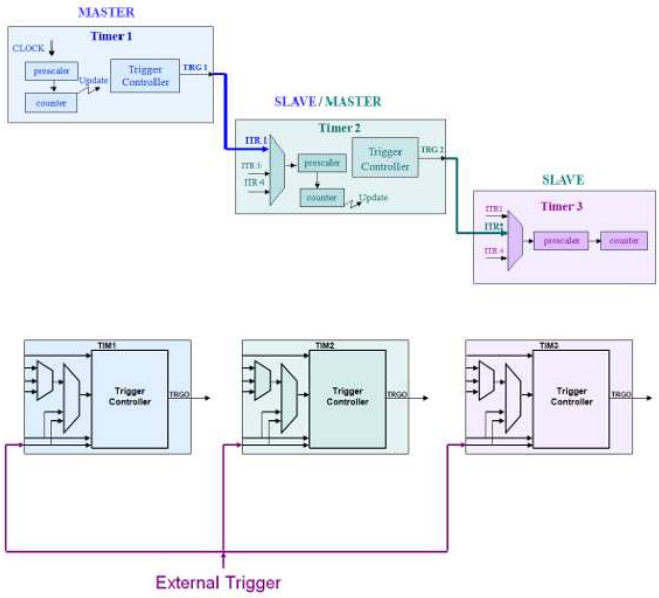
5.1.4.3 Đồng bộ hoá các bộ định thời

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.



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

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



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

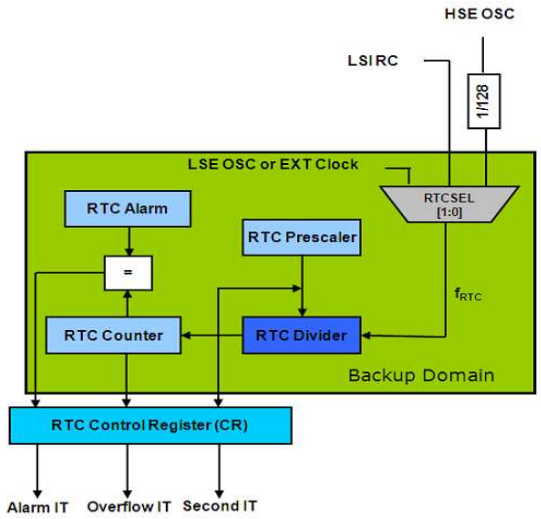
Ở mô hình tạo thành một mảng Timer, một Timer đóng vai trò Master, các Timer còn lại đóng vai trò là Slave.

5.1.5 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.



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.

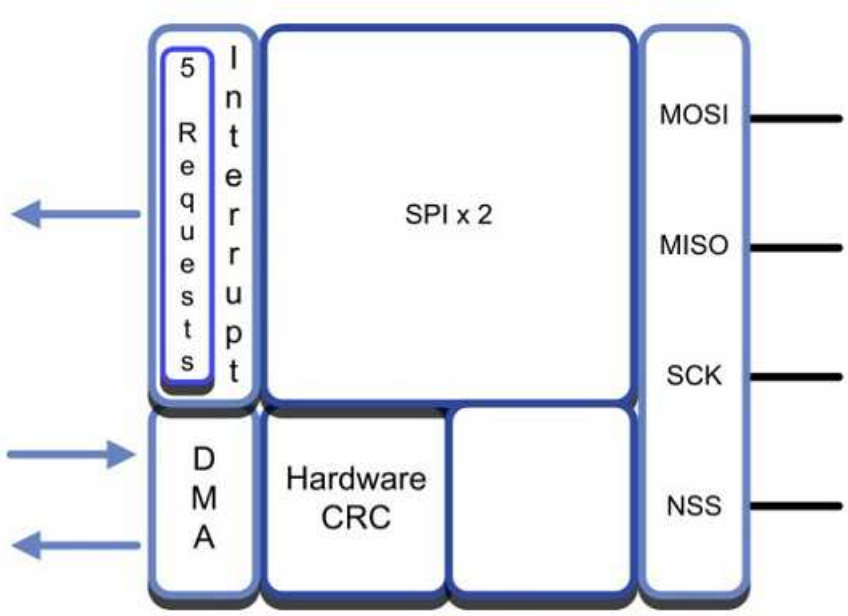
5.2 Kết nối với các giao tiếp khác

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

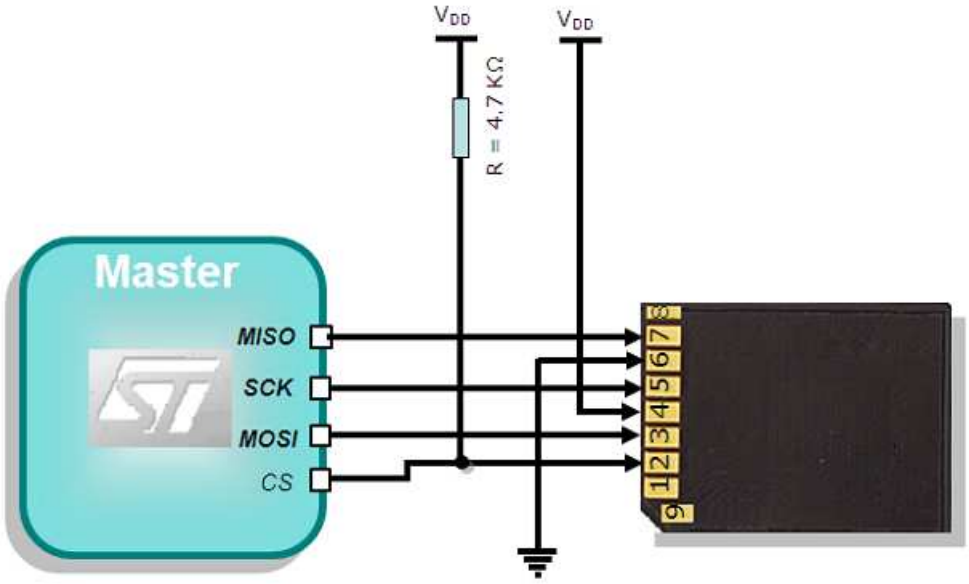
5.2.1 SPI

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.

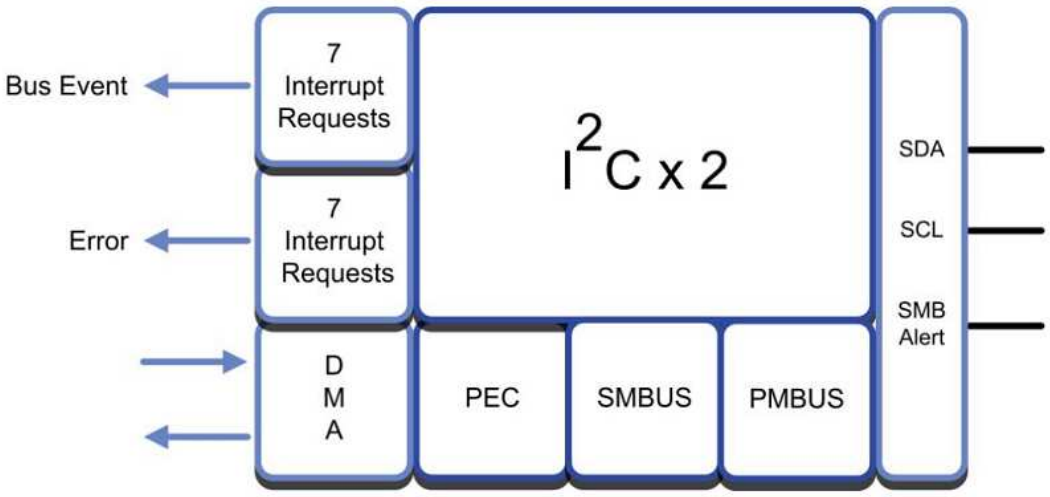


Để hỗ trợ truyền dữ liệu tốc độ cao, mỗi khối SPI có 2 kênh DMA dành cho gửi và nhận dữ liệu. Thêm vào đó là khối CRC dành cho cả truyền và nhận dữ liệu. Khối CRC đều có thể hỗ trợ kiểm tra CRC8 và CRC16. Các đặc tính này rất cần thiết khi sử dụng SPI để giao tiếp với MMC/SD card.

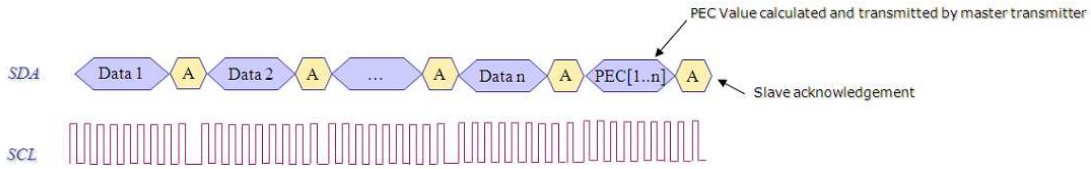


5.2.2 I2C

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

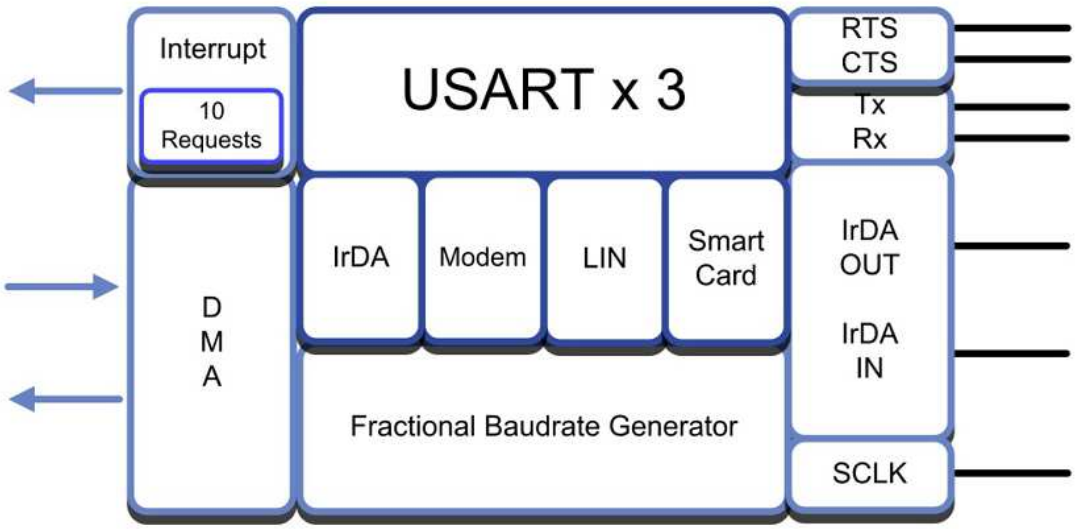


Thêm nữa để đảm bảo tính chính xác dữ liệu truyền, khối kiểm tra lỗi dữ liệu(PAC – packet error checking) được tích hợp thêm vào giao diện I2C cho phép kiểm tra mã CRC-8 bit. Thao tác này được thực hiện hoàn toàn tự động bởi phần cứng.



5.2.3 USART

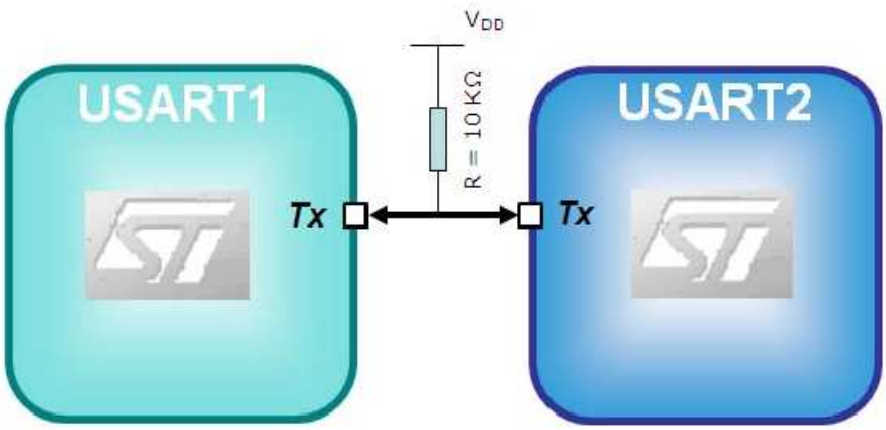
Mặc dù các giao diện trao đổi dữ liệu dạng nối tiếp dần dần không còn được hỗ trợ trên máy tính, chúng vẫn còn được sử dụng rất nhiều trong lĩnh vực nhúng bởi sự tiện ích và tính đơn giản. STM32 có đến 3 khối USART, mỗi khối có khả năng hoạt động đến tốc độ 4.5Mbps. Một khối USART nằm trên APB1 với xung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



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

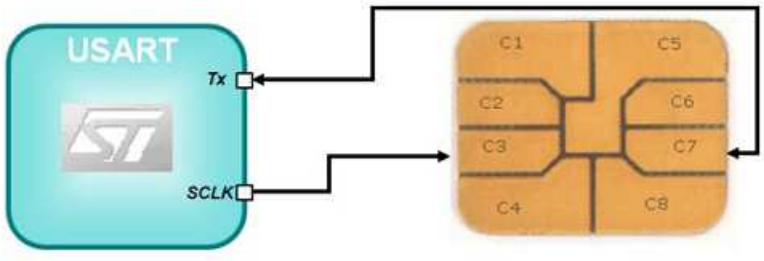
Với mạch tích hợp cho phép chia nhỏ tốc độ BAUD chuẩn thành nhiều tốc độ khác nhau thích hợp với nhiều kiểu trao đổi dữ liệu khác nhau. Mỗi khối USART có hai kênh DMA dành cho truyền và nhận dữ liệu. Khi hỗ trợ giao

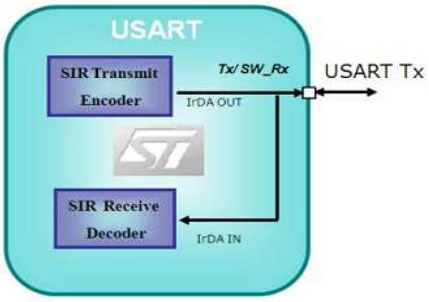
tiếp dạng UART, USART cung cấp nhiều chế độ giao tiếp. Có thể trao đổi dữ liệu theo kiểu chế độ hafl-duplex trên đường truyền Tx. Khi hỗ trợ giao tiếp modem và giao tiếp có sử dụng điều khiển luồng (hardware flow control) USART cung cấp thêm các tín hiệu điều khiển CTS và RTS.



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

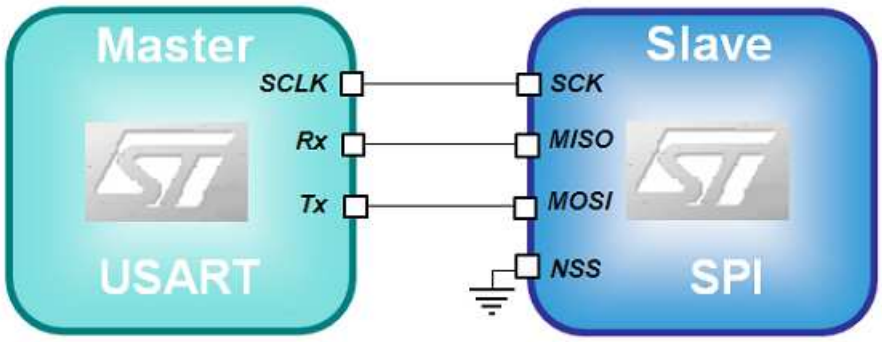
Ngoài ra USART còn có thể dùng để tạo các giao tiếp nội (local interconnect bus). Đây là mô hình cho phép nhiều vi xử lý trao đổi dữ liệu lẫn nhau. USART còn có khối encoder/decoder dùng cho giao tiếp hồng ngoại với tốc độ hỗ trợ có thể đạt đến 1115200bps, hoạt động ở chế độ hafl-duplex NRZ khi xung 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.





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.

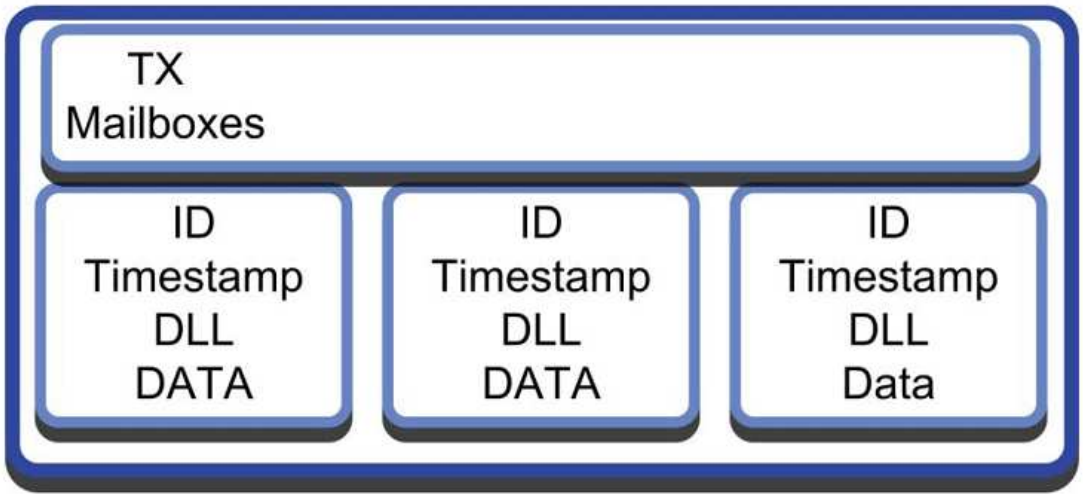


Hỗ trợ giao tiếp đồng bộ SPI

5.2.4 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).

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



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

Một điểm quan trọng nữa của CAN là lọc gói tin nhận(receive message filter). Vì giao thức CAN truyền dữ liệu dựa trên địa chỉ đích nhận, do đó gói tin sẽ được phát trên toàn bộ mạng, chỉ có điểm nào có địa chỉ giống như địa chỉ nhận trên gói tin sẽ dùng gói tin đó. Lọc gói tin giúp các điểm trên mạng CAN tránh xử lý các gói tin không 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.

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.

5.2.5 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 6

CHẾ ĐỘ TIÊU THỤ NĂNG LƯỢNG THẤP

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

6.1 Chế độ bình thường - RUN mode

RUN mode là chế độ STM32 thực hiện các lệnh chương trình và nó tiêu thụ công suất ở mức cao nhất. Trong phần này sẽ chỉ ra nhiều cách để giảm công suất tiêu thụ tổng thể của hệ thống trong chương trình thực thi. Điểm cốt lõi là tất cả các chức năng của hệ thống nên được dùng một cách linh hoạt khi thực thi mã chương trình. Nghĩa là khi có thể, mã nên được chạy ở chế độ công suất thấp-hiệu năng xử lý thấp, và chuyển sang chế độ công suất cao-cấu hình hiệu năng cao để đáp ứng ngắt và chương trình sự kiện (program event).

Trong hoạt động bình thường bộ xử lý Cortex và hầu hết STM32 có thể chạy ở 72MHz. Khi chạy ở tốc độ tối đa, STM32 tiêu thụ hơn 30mA. Tránh tiêu tốn năng lượng ở các ngoại vi không sử dụng bằng cách chặn xung clock của chúng. Khi đó năng lượng tiêu thụ trên toàn hệ thống sẽ giảm. Các xung clock ngoại vi có thể chuyển đổi on/off một cách linh động thông qua khối điều

khiển Clock Reset (Reset clock control module). Bên cạnh đó, có thể giảm công suất tiêu thụ bằng cách giảm xung nhịp hoạt động của hệ thống. Nếu hệ thống không cần hoạt động ở tốc độ cao, có thể tắt bộ nhân tần số PLL và STM32 có thể dùng xung nhịp trực tiếp từ bộ dao động ngoài HSE. Hơn nữa, có thể tắt luôn HSE và dùng xung nhịp từ bộ dao động nội HSI. Điều này có một bất lợi là nguồn xung clock từ HSI không chính xác bằng HSE. Tương tự, nếu không sử dụng khối Windowed Watchdog và đồng hồ thời gian thực(realtime clock) không dùng tới, thì tắt bộ dao động LSI nhằm tiết kiệm hơn nữa công suất tiêu thụ.

6.1.1 Chế độ Half-cycle và Prefetch-buffer

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

APB1	APB2	Peripheral	Frequency	Prefetch	Half Cycle	WFI	Oscillator	Typical consumption at 25 °C in mA
DIV4	DIV2	ALL_ON	72 MHz	ON	OFF	OFF	HSE	33.15
DIV 8	DIV 8	ALL_ON	72 MHz	ON	OFF	OFF	HSE	27.75
DIV 8	DIV 8	USART	72 MHz	ON	OFF	OFF	HSE	23.65
DIV4	DIV2	USART	8 MHz	ON	OFF	OFF	HSE	8.65
DIV4	DIV2	USART	8 MHz	OFF	OFF	OFF	HSE	8.48
DIV4	DIV2	USART	8 MHz	OFF	OFF	ON	HSE	1.68
DIV4	DIV2	USART	8 MHz	OFF	OFF	ON	HSI	0.9

Công suất tiêu thụ ở tốc độ tối đa khoảng 34mA, nhưng ở 8MHz (9.6 DMIPS) công suất tiêu thụ dưới 1mA.

6.2. Các chế độ sử dụng công suất tiêu thụ thấp

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

6.2.1. SLEEP

Mức đầu tiên của hoạt động công suất thấp là chế độ SLEEP mode. Mặc định, khi một lệnh WFE hoặc WFI được thực hiện bộ xử lý Cortex sẽ tạm dừng các đồng hồ xung nhịp nội và dừng thực thi mã ứng dụng. Trong chế độ SLEEP các phần còn lại của STM32 vẫn tiếp tục hoạt động. STM32 sẽ thoát khỏi SLEEP mode khi một ngoại vi nào đó phát sinh ngắt. Khi STM32 vào SLEEP mode cùng tất cả ngoại vi đang hoạt động và nó chạy ở 72MHz từ HSE thông qua bộ nhân PLL, công suất tiêu thụ của nó vào khoảng 14.4mA. Tuy nhiên, có thể được điều chỉnh lại STM32 cho các ứng dụng công suất thấp thông qua 2 bước: thứ nhất tắt tất cả đồng hồ tạo xung nhịp cho các ngoại vi (trừ ngoại vi đánh thức bộ xử lý Cortex); thứ hai chuyển qua dùng bộ dao động nội HSI (có thể chia tần số hoạt động đến mức dưới 1MHz). Khi đó công suất tiêu thụ vào khoảng 0.5mA.

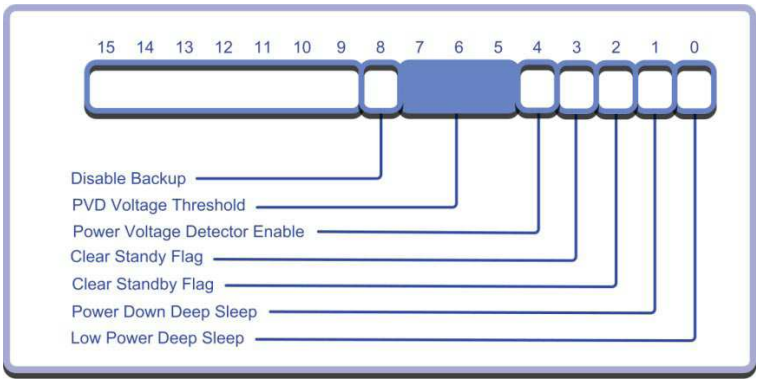
Conditions	f_{HCLK}	All APB peripherals enabled	All peripherals disabled	Unit
Running on HSE, AHB prescaler used to reduce the frequency	72 MHz	14.4	5.5	mA
	48 MHz	9.9	3.9	
	36 MHz	7.6	3.1	
	24 MHz	5.3	2.3	
	16 MHz	3.8	1.8	
	8 MHz	2.1	1.2	
	4 MHz	1.6	1.1	
	2 MHz	1.3	1	
	1 MHz	1.11	0.98	
	500 kHz	1.04	0.96	
	125 kHz	0.98	0.95	
Running on high speed internal RC (HSI), AHB prescaler used to reduce the frequency	64 MHz	12.3	4.4	
	48 MHZ	9.3	3.3	
	36 MHz	7	2.5	
	24 MHz	4.8	1.8	
	16 MHz	3.2	1.2	
	8 MHz	1.6	0.6	
	4 MHz	1	0.5	
	2 MHz	0.72	0.47	
	1 MHz	0.56	0.44	
	500 kHz	0.49	0.42	
	125 kHz	0.43	0.41	

Công suất tiêu thụ chế độ SLEEP mode có thể thấp đến 0.14mA

Trong ứng dụng công suất thấp bạn nên sử dụng chế độ SLEEP mode mỗi khi có thể, nhằm tiêu thụ tối thiểu công suất. Vấn đề tiếp theo là STM32 mất bao lâu để thoát khỏi chế độ công suất thấp và tiếp tục xử lý. Các hình dưới đây cho thấy thời gian đánh thức Cortex CPU để khôi phục xử lý đang chạy bằng bộ dao động RC nội HSI.

6.2.2 STOP Mode

STM32 có thể được cấu hình để vào chế độ công suất thấp STOP Mode bằng cách thiết lập bit SLEEPDEEP trong thanh ghi điều khiển công suất Cortex (the Cortex power control register) và xóa bit Power Down Deep Sleep (PDDS) trong thanh ghi điều khiển công suất STM32.



Khi cấu hình cho chế độ STOP mode, việc thực hiện lệnh WFI hoặc WFE sẽ tạm dừng bộ vi xử lý Cortex và tắt cả bộ tạo dao động nội HIS, lẫn ngoại HSE. Các Flash, SRAM và thiết bị ngoại vi vẫn còn được cung cấp nguồn, do đó, trạng thái của STM32 vẫn được bảo toàn. Cũng giống như chế độ SLEEP, có thể thoát khỏi chế độ STOP bằng ngắt được phát ra từ một ngoại vi STM32. Tuy nhiên, trong chế độ STOP tất cả các xung clock ngoại vi được tạm dừng, ngoại trừ ngoại vi ngắt ngoài(External Interrupt). Việc sử dụng các thiết bị ngoại vi EXTI cho phép STM32 thoát khỏi chế độ STOP mode khi có sự thay đổi trạng thái trên bất kỳ pin GPIO. Ngoài ra, EXTI có một đường tín hiệu điều khiển mà có thể yêu cầu ngắt và tạo ra ngắt từ sự kiện báo thức (Alarm) của đồng hồ thời gian thực. Vì đồng hồ thời gian thực có bộ dao động chuyên dụng của riêng nó (hoặc là LSI hoặc LSE) nó có thể cung cấp ngắt định kỳ để đánh thức STM32 từ chế độ STOP.

Một khi STM32 đã vào chế độ STOP, tiêu thụ điện năng của nó giảm mạnh xuống khoảng 24 uA thay vì hàng mA ở chế độ RUN. Có thể tiết kiệm thêm điện năng bằng cách đặt các bộ điều áp nội trong một chế độ năng lượng thấp đặc biệt, khi nó vào chế độ STOP. Các chế độ năng lượng thấp cho các bộ điều áp được chọn bằng cách thiết lập bit LPDS (Low Power Sleep Deep) trong

thanh ghi điều khiển công suất STM32. Với thiết lập này khi STM32 vào chế độ STOP, tiêu thụ điện năng của nó sẽ giảm xuống 14uA. Nếu RTC đang được sử dụng, sẽ tiêu thụ thêm 1,4 uA.

Conditions	V_{DD}/V_{BAT} = 2.4 V	V_{DD}/V_{BAT} = 3.3 V	Unit
Regulator in Run mode, low-speed and high speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog)	NA	24	µA
Regulator in Low Power mode, low speed and high speed internal RC oscillators and high speed oscillator OFF (no independant watchdog)	NA	14	

Symbol	Parameter	Conditions	Type	Unit
t_{WUSTOP}	Wakeup from Stop mode (regulator in run mode)	Wakeup on HSI RC clock	3.52	µs
	Wakeup from Stop mode (regulator in run mode + WFI)		5.42	
	Wakeup from Stop mode (regulator in Low power mode + WFE)		5.32	
	Wakeup from Stop mode (regulator in Low power mode + WFI)		7.21	

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

6.3 Standby

STM32 có thể được cấu hình để vào chế độ Standby bằng cách thiết lập bit SLEEPDEEP trong thanh ghi điều khiển công suất Cortex và thiết lập bit

Power Down Deep Sleep (PDDS) trong thanh ghi điều khiển công suất của STM32. Bây giờ, khi lệnh WFI hoặc WFE được thực hiện, STM32 sẽ vào chế độ năng lượng thấp nhất của nó. Trong chế độ Standby, STM32 thực sự tắt. Các điều điện áp nội bộ được tắt các bộ tạo dao động HSE và HIS cũng tắt. Trong chế độ này STM32 chỉ chiếm 2uA.

Conditions	V_{DD}/V_{BAT} = 2.4 V	V_{DD}/V_{BAT} = 3.3 V	Unit
Low-speed internal oscillator and independent watchdog OFF, low speed oscillator and RTC OFF	NA	2	µA
Low-speed oscillator and RTC ON	1.08	1.4	

Symbol	Parameter	Conditions	Type	Unit
$t_{WUSTDBY}$	Wakeup from Standby mode	Wakeup on HSI RC clock	50	µs

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

Chúng ta có thể thoát khỏi chế độ Standby bằng cách sử dụng tín hiệu ngắt của khối RTC tương tự như trong chế độ STOP. Ngoài ra ta có thể sử dụng chân Reset ngoại hay chân Reset từ Watchdog độc lập. Chế độ Standby có thể kết thúc khi có xung nhịp cạnh lên(rising edge) ở chân 0 của Port A nếu chân này được cấu hình như là chân có tính năng WKUP bằng cách thiết lập bi EWUP ở thanh ghi Power và Status.

Standby là chế độ tiêu thụ ít năng lượng nhất cho nên sẽ mất nhiều thời gian để hệ thống phục hồi trở lại trạng thái hoạt động bình thường. Mất khoảng 50usec

trước khi nhân Cortex khởi động lại tiến trình xử lý lệnh. Một khi vào chế độ Standby, toàn bộ dữ liệu trên SRAM, nhân Cortex và các thanh ghi đều bị mất. Khi đó phục hồi trạng thái hoạt động từ chế độ Stanby tương tự như ta khởi động lại hệ thống.

6.4. Sự tiêu thụ công suất của nguồn dự phòng (Backup Region Power Consumption)

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

6.5 Hỗ trợ Debug (Debug Support)

Trên hệ thống vi điều khiển truyền thống, gỡ lỗi một ứng dụng mà sử dụng chế độ công suất thấp có thể mất rất nhiều công sức. Ngay sau khi vi điều khiển vào các chế độ công suất thấp nó dừng đáp ứng các trình gỡ lỗi, điều này sẽ tạo một lỗi hoặc ngừng làm việc. Trong STM32 có thể cấu hình các chế độ công suất thấp để giữ bộ dao động nội HSI chạy trong các chế độ năng lượng thấp tương ứng, cung cấp một đường xung xung nhịp dành cho khối gỡ lỗi CoreSight. Điều này có nghĩa là bạn có thể gỡ lỗi đầy đủ các ứng dụng năng lượng thấp mà không cần phải gỡ bỏ chế độ năng lượng thấp. Điều này giúp loại bỏ các vấn đề gỡ lỗi timeout. Việc tăng cường tính năng gỡ lỗi STM32 được cấu hình với thanh ghi DBG_MCU.

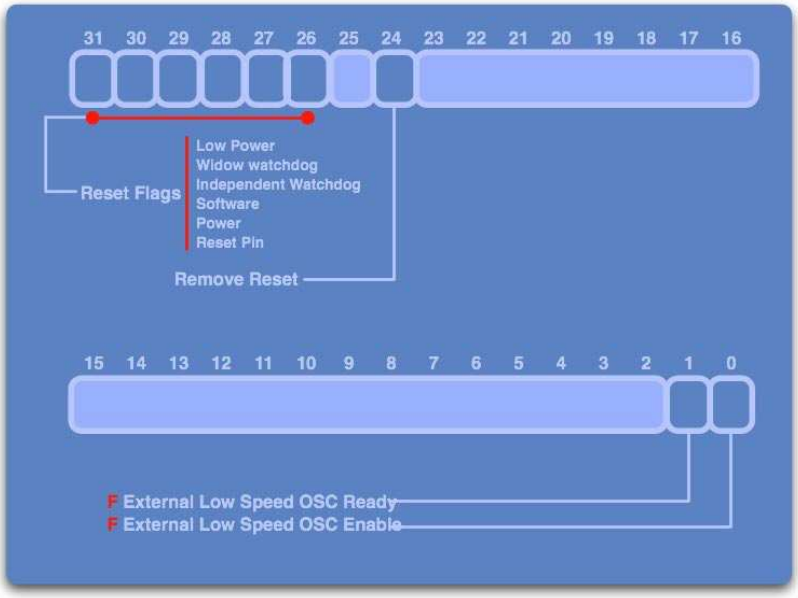
Chương 7

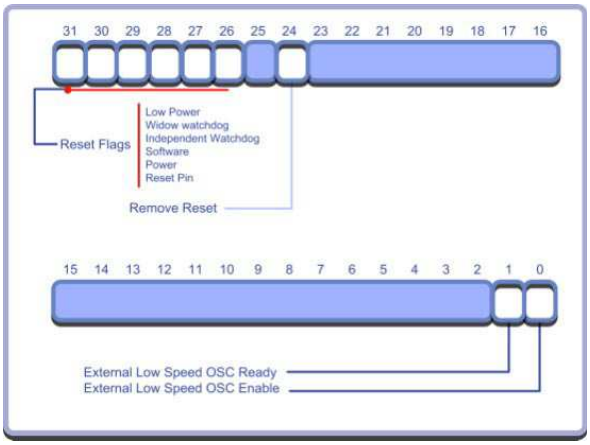
TÍNH AN TOÀN

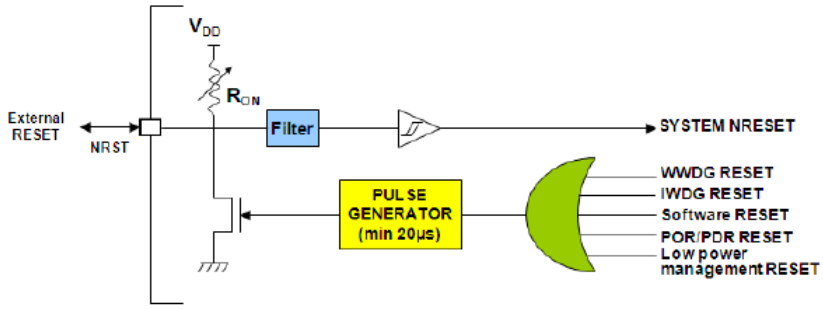
STM32 cũng đã được thiết kế với một số tính năng vốn có mà sẽ phát hiện các hoạt động không chính xác của các mã ứng dụng hoặc của chính STM32. Để đảm bảo rằng có một nguồn cung cấp năng lượng đáng tin cậy, STM32 có một bộ reset nội thực hiện chức năng reset chip nếu các nguồn cung cấp điện áp dưới mức tối thiểu VDD. Ngoài ra còn có một mạch kiểm tra điện áp nguồn khả trình có thể được sử dụng để phát hiện việc mất nguồn đột ngột. Sau khi phát hiện mạch kiểm tra sẽ tạo ra một ngắt để đặt chip vào trạng thái an toàn. Hệ thống giám sát tính an toàn của xung nhịp hệ thống (clock security system) theo dõi theo dõi bộ dao động ngoại HSE. Nếu xung nhịp ngoại bị gặp sự cố đột ngột, CSS điều chỉnh hệ thống hoạt động bằng bộ dao động nội. Một chương trình thực thi sẽ được theo dõi bởi hai bộ Watchdog nội. Bộ thứ nhất, là một Windowed Watchdog với thời gian theo dõi được cấu hình cụ thể. Bộ thứ hai là một Watchdog độc lập được cấp xung nhịp từ một bộ dao động độc lập với hệ thống chính. Ngoài ra, bộ nhớ Flash trên-chip có được khả năng lưu dữ liệu đến 30 năm ở 85 độ C. Đây là sự lưu dữ liệu tốt nhất cho một vi điều khiển đa dụng.

7.1 Reset Control

STM32 có nhiều nguồn Reset khác ngoài đường Reset bên ngoài. STM32 có thể bị buộc khởi động lại từ: các Watchdogs nội, một Reset mềm thông qua NVIC, bộ Reset mở/tắt nguồn nội và mạch phát hiện điện áp nguồn thấp. Nếu một hiệu lệnh Reset xuất hiện, một bộ cờ trong thanh ghi kiểm soát và trạng thái RCC có thể được đọc để xác định nguyên nhân gây ra Reset. Trạng thái của những cờ này sẽ vẫn tồn tại cho khi hệ thống được cấp nguồn trở lại hoặc cho đến khi người dùng thiết lập bit Remove Reset.



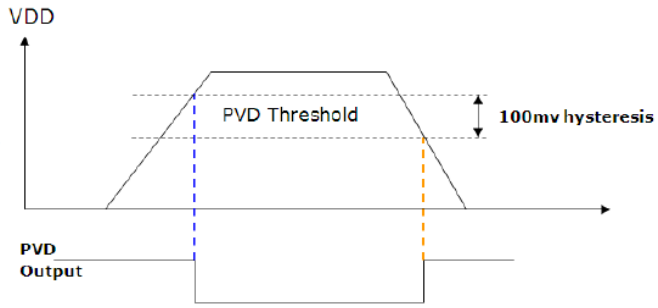




STM32 có thể có nhiều tín hiệu điều khiển để đưa hệ thống về trạng thái Reset. Trạng thái hệ thống sẽ được lưu lại trong thanh ghi RCC

7.2 Kiểm tra điện áp nguồn

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

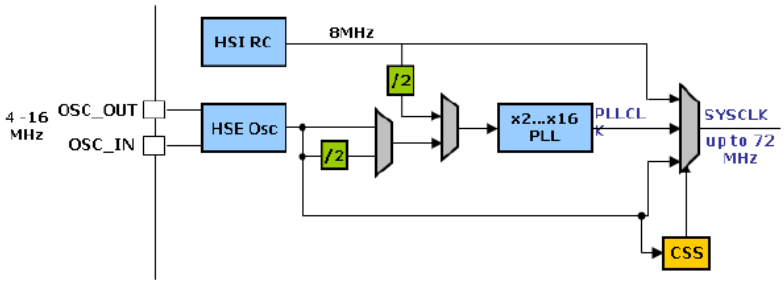


The power supply is monitored by the power voltage detect unit, which can generate an interrupt if the supply dips below a configured threshold.

Đầu ra của PVD được kết nối với 16 đường của bộ ngắt ngoài. Do các đường ngắt ngoài EXTI có thể được cấu hình để tạo ra ngắt từ tín hiệu cạnh lên hoặc cạnh xuống, hoặc cả hai, đơn vị PVD có thể được sử dụng để tạo ra một ngắt cho cả thấp áp và quá áp.

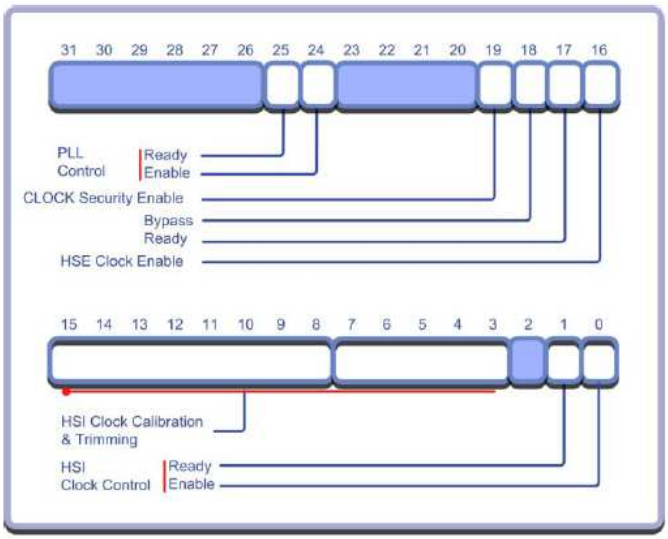
7.3 Hệ thống an toàn xung nhịp (Clock Security System - CSS)

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



The Clock Security System generates an interrupt if the external oscillator fails and switches to the internal RC oscillator.

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



The Clock Security System is enabled by setting the CSS enable bit in the RCC control register.

Hệ thống CSS có một đường ngắt được kết nối với ngắt gẫy (break interrupt) của khối định thời nâng cao Timer1, mà ở đó ngắt được ánh xạ vào bảng vector điều khiển ngắt Cortex NVIC non-maskable. Điều này đảm bảo nếu dao động chính không hoạt động nữa, các ngõ ra PWM của khối định thời nâng cao sẽ ngay lập tức được đặt trong trạng thái an toàn tiền lập trình(pre-programmed safe state) bởi phần cứng. Điều này đảm bảo rằng bất kỳ phần cứng nào được điều khiển bởi các đầu ra PWM của khối định thời tiên tiến sẽ không được phép hoạt động khi không có sự kiểm soát của bộ vi xử lý Cortex. Nó đặc biệt quan trọng cho các ứng dụng điều khiển động cơ.

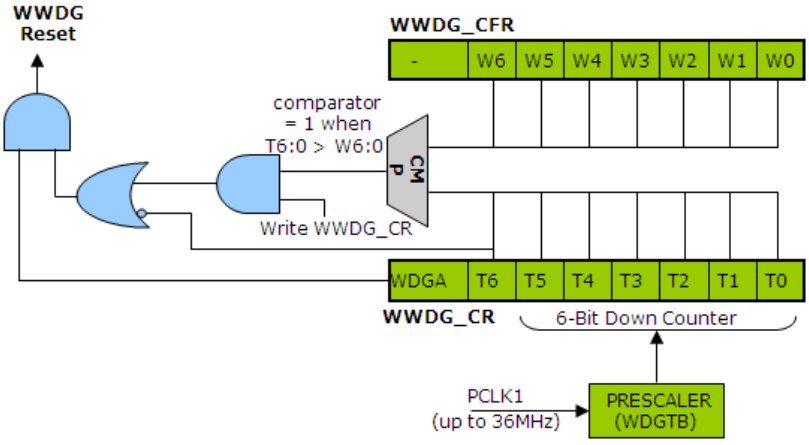
7.4 Watchdogs

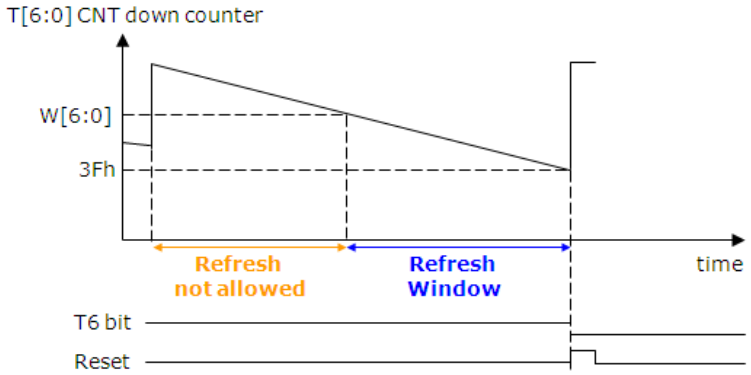
STM32 chứa hai Watchdogs riêng biệt. Watchdog độc lập(independent watchdog) là hoàn toàn tách biệt với hệ thống STM32 chính. Nó được đặt trong miền nguồn điện dự phòng và xung nhịp của nó bắt nguồn từ dao động nội tốc độ thấp (LSI- Low Speed Oscillator). Windowed Watchdog là một phần của hệ thống STM32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

truyền xung nhịp ngoại vi số 1(PCLK1). Cả hai Watchdogs phải được kích hoạt riêng biệt và có thể được sử dụng đồng thời.

7.4.1 Windowed Watchdog

Windowed Watchdog là một phiên bản nâng cao hơn của Watchdog truyền thống trên chip. Sau khi kích hoạt, Watchdog sẽ đếm ngược và sẽ tạo ra tín hiệu Reset khi giá trị của thanh ghi thay đổi từ 0x40 sang 0x3F tức là khi bit T6 bị xóa. Giá trị đếm ngưỡng trên được lưu trong thanh ghi cấu hình Windowed Watchdog. Nếu phần mềm ứng dụng làm tươi bộ đếm Watchdog trong khi giá trị thực của bộ đếm Watchdog lớn hơn giá trị ngưỡng đếm trên một tín hiệu Reset cũng sẽ được phát ra. Windowed Watchdog cung cấp một khung làm tươi khả trình mà khi đó Watchdog mới có thể được ghi vào theo lịch trình thời gian đã định. Điều này cho phép người dùng đảm bảo ứng dụng vẫn đang trong chế độ hoạt động bình thường đã xác định.





Windowed Watchdog là bộ đếm xuống 6 bit, được cung cấp xung clock từ PCLK1 thông qua bộ tiền chia 12 bit – chia PCKL1 xuống bởi 4096. Bộ tiền chia có thêm 4 bit khả lập trình cho phép người dùng chia thêm 1,2,4 hoặc 8. Các bít của bộ tiền chia được chứa trong bít 6,7 của thanh ghi điều khiển.

Do đó thời gian lặp của windowed watchdog được cho bởi:

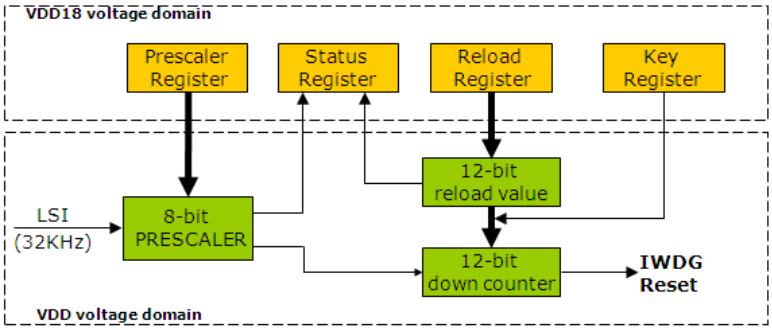
Twwdg = Tpclk1x4096x2POW (WDGTB) x (reload value+1)

Với Pclk1 chạy ở tốc độ tối đa 36MHz, thời gian timeout nhỏ nhất của windowed watchdog là 910uSec và lớn nhất là 58,25mSec.

Sau khi windowed watchdog đã được cấu hình, nó có thể được kích hoạt bằng cách thiết lập bit Watchdog Activation trong trong thanh ghi điều khiển. Sau khi Windowed Watchdog đã được kích hoạt bằng phần mềm nó không thể bị vô hiệu, ngoại trừ hệ thống tái khởi động.

7.4.2 Independent Watchdog

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



Independent Watchdog là một định thời đếm xuống 12 bit, và sẽ phát sinh hiệu lệnh Reset cho hệ thống khi giá trị đạt dưới ngưỡng. Nó được cấp xung nhịp từ dao động nội tốc độ thấp thông qua một bộ chia 8 bit. Bộ tạo dao động LSI có một tần số danh nghĩa là 32kHz, nhưng trong thực tế nó có thể thay đổi giữa 30 KHz đến 60 KHz. Independent Watchdog được khởi động bằng cách, đầu tiên thiết lập thanh ghi Prescaler, khi qua bộ chia Prescaler dao động sẽ có giá trị trong khoảng từ 4 đến 256. Khoảng thời gian chờ tối thiểu cho Independent Watchdog là 0mSec và tối đa chỉ trên 26 giây. Thời gian chờ được lập trình trực tiếp vào thanh ghi nạp lại (reload register).

The independent watchdog is a countdown watchdog with its own oscillator. It is also located in the backup domain so that it can remain active during Stop and Standby modes.

Các byte tùy chọn trong bộ nhớ Flash khối thông tin nhỏ có thể được dùng để cấu hình Independent Watchdog để bắt đầu sau khi reset, hoặc bằng lệnh phần mềm. Nếu theo phần mềm kiểm soát, Independent Watchdog có thể được bắt đầu bằng cách ghi 0xCCCC vào thanh ghi khóa. Independent Watchdog sẽ đếm xuống từ một giá trị ban đầu 0xFFF. Giá tri 0xAAAA phải được ghi vào thanh ghi khóa để làm tươi watchdog. Điều này làm cho giá trị nạp lại được nạp vào thanh ghi đếm xuống, làm mới giá trị đếm.

Rất khó để gỡ lỗi chương trình ở các dòng vi điều khiển nhỏ nếu Watchdog đã được kích hoạt. Ngay sau khi CPU ngưng hoạt động, watchdog không thể được cập nhật. Nó sẽ chờ thời gian tới hạn(timeout) và sau đó sẽ phát sinh ra ngắt, khi đó toàn bộ hệ thống khởi động lại xóa đi trạng thái lỗi cần mà người dùng cần gỡ. Bình thường, một Watchdog phải được vô hiệu hoá để nó không gây rối loạn các trình gỡ lỗi. Nếu vậy thì lại khó khăn trong việc đánh giá cấu hình thời gian kiểm tra của Watchdog là tối ưu hay chưa. Trong thanh ghi MCUDBG STM32 có thể cấu hình cả hai Independent Watchdog và Window Watchdog dừng hoạt động khi Cortex-M3 CPU nằm dưới sự kiểm soát của hệ thống gỡ lỗi CoreSight. Điều này cho phép người dùng gỡ lỗi ngay cả khi hai khối Watchdogs được kích hoạt.

7.5 Tính năng ngoại vi

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

7.5.1 GPIO Port Locking (khóa port GPIO)

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

7.5.2 Analog Watchdog

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

7.5.3 Break Input

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

Chương 8:

FLASH

Bộ nhớ FLASH on-chip của STM32 được bố trí trong 3 vùng chính. Đầu tiên, vùng nhớ FLASH chính được thiết kế để chứa các lệnh chương trình. Đây là vùng nhớ 64 bit, để cung cấp sự truy cập bộ nhớ hiệu quả với bộ đệm lấy lệnh. Vùng nhớ này được chia thành từng trang 4K đối với hoạt động xóa và ghi chương trình flash. Bộ nhớ có độ bền 10.000 chu kỳ, duy trì dữ liệu 30 năm ở 85^{o}C. Hầu hết các bộ nhớ FLASH vi điều khiển chỉ duy trì dữ liệu tốt ở 25 ^{0}C, bộ nhớ FLASH của STM32 là một ngoại lệ. Bên canh vùng nhớ chương trình chính, là 2 vùng nhớ nhỏ hơn: khối thông tin lớn và khối thông tin nhỏ. Khối thông tin lớn là 2k bộ nhớ FLASH chứa chương trình **bootloader** được cung cấp bởi nhà sản xuất, là chương trình được thiết kế sẵn để tải code về qua cổng USART1. Khối thông tin nhỏ chứa 6 byte cấu hình, là các byte được dùng để định nghĩa các thuộc tính Reset của STM32 và bảo vệ bộ nhớ.

8.1 Lập trình và đảm bảo an toàn cho FLASH nội

Bộ nhó FLASH nội có thể được cập nhật bởi bootloader, bằng JTAG, hoặc bằng chương trình ứng dụng bên trong thông qua một bộ thanh ghi chuyên dụng được gọi là chương trình FLASH và xóa điều khiển FPEC. FPEC cũng được dùng để lập trình các byte tùy chọn trong khối thông tin nhỏ.

Khối FPEC được dùng để cho phép trong ứng dụng lập trình của bộ nhớ FLASH. Bộ nhơ FLASH cũng có thể được bảo vệ đọc khỏi các công cụ debug và được bảo vệ ghi.

8.2 Hoạt động xóa và ghi

Sau khi reset, các thanh ghi FPEC được bảo vệ và phải được mở khóa bằng cách ghi một chuỗi đặc biệt vào thanh ghi khóa (key register). Để mở khóa FPEC bạn phải ghi 0x45670123, theo sau bởi 0xCDEF89AB. Nếu có lỗi xuất hiện trong chuỗi này, FPEC sẽ ở trạng thái bị khóa cho đến lần reset kế tiếp. Một khi FPEC đã được mở khóa, nó có thể thực hiện hoạt động xóa và ghi trên bộ nhớ FLASH chính. Trong khối bộ nhớ FLASH chính nó có thể thực hiện xóa theo khối hoặc xóa một trang 4k được chọn. Hoạt động xóa khối được thực hiện bằng cách thiết lập đơn giản xóa khối và khởi động các bit trong thanh ghi điều khiển. Khi bit Busy trong cùng thanh ghi được reset, mỗi vị trí trong bộ nhớ FLASH chính sẽ được reset về 0xFFFF. Xóa một trang thì cũng đơn giản như vậy. Đầu tiên, bạn phải lập trình địa chỉ bắt đầu của một trang FLASH trong thanh ghi địa chỉ, sau đó thiết lập xóa trang và khởi động các bit trong thanh ghi điều khiển. Một lần nữa, khi bit Busy bị xóa, trang được thiết lập sẽ bị xóa. Dữ liệu mới có thể được ghi vào ô nhớ FLASH chỉ sau khi nó được xóa rồi. Một hoạt động ghi (WRITE) được thực hiện bằng cách thiết lập bit Program trong thanh ghi điều khiển và sau đó thực hiện ghi một haft-word (2

byte) vào vị trí mong muốn. Nếu vị trí FLASH được xóa và không được bảo vệ ghi, FPEC sẽ lập trình giá trị mới vào trong ô nhớ FLASH.

8.3 Các byte Option (Option Bytes)

Khối thông tin nhỏ có 8 byte Option cấu hình dành cho người dùng. Trong đó 4 byte được dùng để thiết lập hoạt động bảo vệ ghi trên bộ nhớ FLASH chính. Byte thứ năm dùng để thiết lập bảo vệ đọc để tránh sự truy nhập vào các vùng nhớ khi chip ở trong chế độ debug. Byte thứ sáu được dùng để cấu hình công suất thấp và hoạt động reset. Hai byte cuối cùng là những ô bộ nhớ FLASH đơn giản, sẵn có cho người dùng tùy chọn. trước khi các byte Option có thể được ghi vào, FPEC phải được mở khóa như mô tả ở trên. Kế tiếp các byte tùy chọn phải được mở khóa bằng cách ghi giống 2 khóa vào thành ghi khóa option. Các byte Option có một chương trình riêng biệt và thủ tục xóa bộ nhớ FLASH chính. Khối thông tin nhỏ được xóa băng cách thiết lập bit OPTER trong thanh ghi điều khiển kế tiếp là bit STRT. Một khi bit BSY được reset, khối thông tin nhỏ được xóa. Để lập trình một byte Option, set bit OPTPG trong thanh ghi điều khiển FLASH và thực hiện ghi half-word vào byte Option. Mỗi byte Option được lưu như một half-word. Byte Option được lưu trong byte thấp của halfword và giá trị bù của nó được lưu trong phần cao của halfword. Bạn phải ghi một giá trị phù hợp vào phần thấp của halfword và FPEC sẽ tính giá trị bù một cách tự động.

8.3.1 Bảo vệ ghi

Khi nó được thiết lập, mỗi bit trong các byte Option bảo vệ ghi cho phép việc bảo vệ trên một trang FLASH. Việc bảo vệ ghi có thể không được cho phép bằng cách xóa khối thông tin nhỏ.

8.3.2 Bảo vệ đọc

Khi bảo vệ đọc được thiết lập, tất cả các truy cập đọc vào bộ nhớ FLASH đề không được phép khi thiết bị vào chế độ debug. Truy cập vào SRAM thì vẫn

được và code có thể được download và thực thi trong vùng này. Nên có thể không cho phép việc bảo vệ đọc khi chạy một chương trình ngoài vùng SRAM. Tuy nhiên, khi bảo vệ đọc bị tắt lúc đó việc xóa khối của FLASH nội sẽ được thực hiện nhằm tránh các chương trình ăn cắp phần mềm. Khi chế độ bảo vệ đọc được cho phép, bộ nhớ FLASH cũng được bảo vệ ghi để tránh một chương trình hiểm độc (malicious) khỏi bị chèn vào vùng nhớ chứa bảng vector. Bộ nhớ STM FLASH được bảo vệ nếu byte bảo vệ đọc và phần bù của nó được set lên 0xFF. Bộ nhớ có thể được vô hiệu bảo vệ bằng cách ghi 0xFA và phần bù của nó như một half-word vào byte Option bảo vệ đọc.

8.3.3 Byte Cấu hình

Byte Option cấu hình chứa 3 bit active. Trong đó 2 bit quản lý STM32 vào chế độ Standby và Stop như thế nào. Một trong hai chế độ có thể được cấu hình để phát một tín hiệu reset khi vào. Điều này sẽ cấu hình các chân IO số như ngõ vào, giảm công suất tiêu thụ toàn bộ của STM32. PLL và bộ dao động ngoài cũng sẽ bị vô hiệu hóa và chip sẽ chuyển qua dùng bộ dao động tốc độ cao RC nội như là nguồn xung chính. Bit cuối cùng trong byte Option cấu hình hoạt động của Independent Watchdog. Watchdog này có một chế độ watchdog cứng ở đó nó sẽ bắt đầu ngay lập tức sau khi bộ xử lý reset, hoặc chế độ watchdog mềm ở đó nó phải được bắt đầu dưới sự điều khiển của phần mềm.

Chương 9:

CÔNG CỤ PHÁT TRIỂN

Việc chấp nhận ARM7 và ARM9 là vi điều khiển tiêu chuẩn đã dẫn tới một sự bùng nổ trong công cụ phát triển hỗ trợ cho các CPU này. Tất cả các nhà phát triển trình biên dịch lớn như GCC, Greenhills, Keil, IAR và Tasking đều cung cấp các công cụ phát triển ARM . Với việc giới thiệu các bộ vi xử lý Cortex, tất cả các công cụ phát triển đã được mở rộng để hỗ trợ các tập lệnh Thumb-2. Nếu bạn đã sử dụng một vi điều khiển ARM-based, thì trình biên dịch các bạn đang dùng sẽ hỗ trợ biên dịch mã cho STM32. Nếu phiên bản phần mềm đang dùng quá cũ thì chúng ta phải nâng cấp chúng.

Nếu đây là dự án đầu tiên của bạn với một vi điều khiển ARM-based, bạn sẽ có thể chọn một bộ công cụ (toolset) từ nhà sản xuất quen thuộc của bạn. Thứ nhất là trình biên dịch "GCC" hoặc "GNU" là một công cụ mã nguồn mở có thể được tải về và sử dụng miễn phí. Trình biên dịch GCC đã được tích hợp vào một số IDE thương mại và gỡ rối để làm cho thấp chi phí công cụ phát triển và kit đánh giá. Trong khi các trình biên dịch GCC là một trình biên dịch đáng tin cậy và ổn định. Các bạn có thể sử dụng Sourcery phiên bản miễn phí để biên dịch, tuy nhiên mã chương trình sẽ không tối ưa bằng các phần mềm thương mại khác. Giống như các phần mềm nguồn mở khác, hỗ trợ sử dụng của GCC rất hạn chế, nếu bạn là người mới vào nghề thì không nên tiếp cận GCC khi làm dự án.

Trình biên dịch ARM RealView là trình biên dịch C bản gốc và hầu hết được chọn lọc và được phát triển bởi ARM để sử dụng với CPU của họ. Trình biên dịch RealView có sẵn như là một phần của toolset ARM RealView. Toolset này là nhằm vào các nhà phát triển system-on-chip và không thực sự phù hợp cho các dự án vi điều khiển. Tuy nhiên kể từ tháng 1 năm 2006 trình biên dịch RealView đã được tích hợp vào Keil Microcontroller Development Kit (MDK-

ARM). Như tên gọi của nó ngụ ý, MDK-ARM là một chuỗi công cụ hoàn chỉnh thiết kế riêng cho vi điều khiển dựa trên ARM-based. Ở nước ta, hầu hết các kỹ sư đều quen thuộc với Keil khi lập trình với 8051, do đó sử dụng Keil để lập trình cho ARM cũng là điều thuận lợi.

Nếu bạn đang làm cho một quyết định giữa việc sử dụng các trình biên dịch GCC và một trình biên dịch thương mại, bạn sẽ bị chi phối một phần bởi ngân sách dự án. Một dự án 'đơn giản' dường như không có ngân sách để chi cho một toolset thương mại. Tuy nhiên, nếu bạn có kế hoạch chuẩn hóa vi điều khiển ARM- based, sau đó là một toolset 'xịn' sẽ sớm chi cho bản thân dự án cả về rút ngắn thời gian phát triển và một mã chương trình nhỏ gọn và tối ưu hơn.

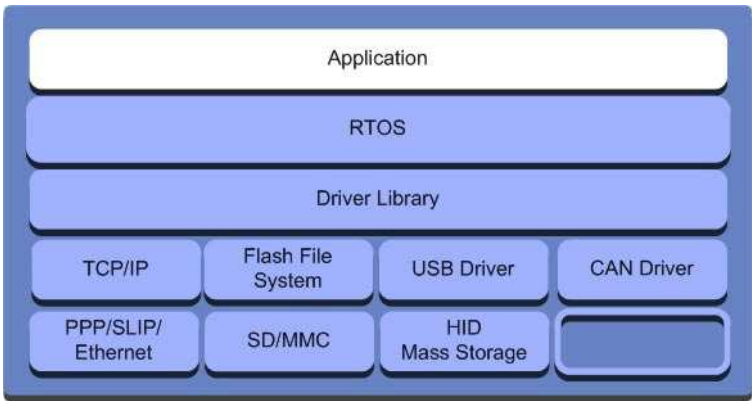
9.1 Evaluation Tools

Hầu hết các nhà cung cấp trình biên dịch cũng sẽ cung cấp một kit đánh giá hoặc kit starter (evaluation kit or starter kit). Đây là một board mạch phần cứng truyền thống và một phiên bản cắt giảm xuống hoặc giới hạn thời gian của toolset của họ. Chúng tôi giới thiệu Kit GE-M3M-V2. Chi phí khoảng 899000VND, GE-M3M-V2 là công cụ hoàn hảo để các kỹ sư tiếp cận đến vi xử lý STM32.

9.2 Các thư viện và giao thức

Để hỗ trợ phát triển mã nhanh chóng, ST đã cung cấp một thư viện phần mềm STM32 firmware như là một download miễn phí từ trang web của họ. Thư viện phần mềm cung cấp chức năng điều khiển cấp thấp cho tất cả các thiết bị ngoại vi on-chip. Điều này cho phép bạn xây dựng một số khối cơ bản mà trên đó bắt đầu xây dựng ứng dụng của bạn. Các thiết bị ngoại vi phức tạp nhất trên các biến thể STM32 hiện nay là bộ điều khiển thiết bị USB. Để giúp bạn xây dựng các thiết bị lớp USB phổ biến, ST cũng cung cấp một bộ kit phát triển USB miễn phí. Cũng giống như các thư viện phần mềm, kit phát triển của USB có

thể được tải về từ trang web của ST. Kit phát triển USB cung cấp một thư viện USB và các ứng dụng trình diễn cho HID, Mass Storage, âm thanh và thiết Device Field Upgrade.



Với sự gia tăng độ phức tạp của các ngoại vi Vi điều khiển, thì điều quan trọng là chọn chuỗi công cụ cái mà được hỗ trợ tốt với giao thức và các phần mềm ứng dụng ngăn xếp.

Như các biến thể mới của các STM32 được phát hành, chúng sẽ có nhiều thiết bị ngoại vi càng phức tạp hơn (Ethernet MAC, giao diện màn hình TFT, vv). Như vậy làm tăng độ phức tạp, nó sẽ trở thành là chỉ không thể phát triển tất cả các mã ứng dụng của chính bạn. Vì vậy, khi lựa chọn công cụ phát triển nó cũng rất quan trọng để xem xét tính sẵn có của giao thức ngăn xếp, chẳng hạn như ngăn xếp TCP / IP và phần mềm ứng dụng khác, chẳng hạn như GUI, có thể được yêu cầu trong các dự án tương lai. Lý tưởng những thứ này nên từ cùng nhà cung cấp và tích hợp tốt vào toolset bạn đã chọn.

9.3 Hệ điều hành thời gian thực

Nếu bạn đang chuyển từ một vi điều khiển tám hoặc mười sáu bit, thì cơ hội được rằng hiện tại bạn không sử dụng một RTOS. Như chúng ta đã thấy, các Cortex-M3 cung cấp cho bạn sức mạnh xử lý đặc biệt nhiều hơn so sánh giá thành các vi điều khiển và được thiết kế để hỗ trợ RTOS footprint nhỏ. Vì vậy, nếu bạn không sử dụng một RTOS nó là giá trị để xem xét khi bạn bắt đầu làm việc với các STM32. Việc sử dụng một RTOS cung cấp cho bạn những lợi thế về phát triển thêm mã rút gọn, nâng cao tái sử dụng mã, quản lý dự án dễ dàng hơn và tăng cường gỡ lỗi. Việc sử dụng một RTOS cũng cung cấp một cấu trúc

cho mã của bạn, điều mà buộc bạn phải lên kế hoạch cho ứng dụng trước khi bạn nhảy vào và bắt đầu viết. Có nhiều RTOS có sẵn hơn cho ARM và Cortex hơn cho hầu hết các CPU nhúng. Nhiều nhà cung cấp trình biên dịch sẽ cung cấp hệ điều hành của chính họ và hoặc một RTOS ngoài, một trong những hệ điều hành mã nguồn mở phổ biến nhất là "FreeRTOS", có sẵn từ www.freertos.org. Một phiên bản thương mại của FreeRTOS được gọi là "SafeRTOS", mà đã được thử nghiệm để đáp ứng các tiêu chuẩn an toàn IEC 61508 và cũng có sẵn từ cùng một trang.