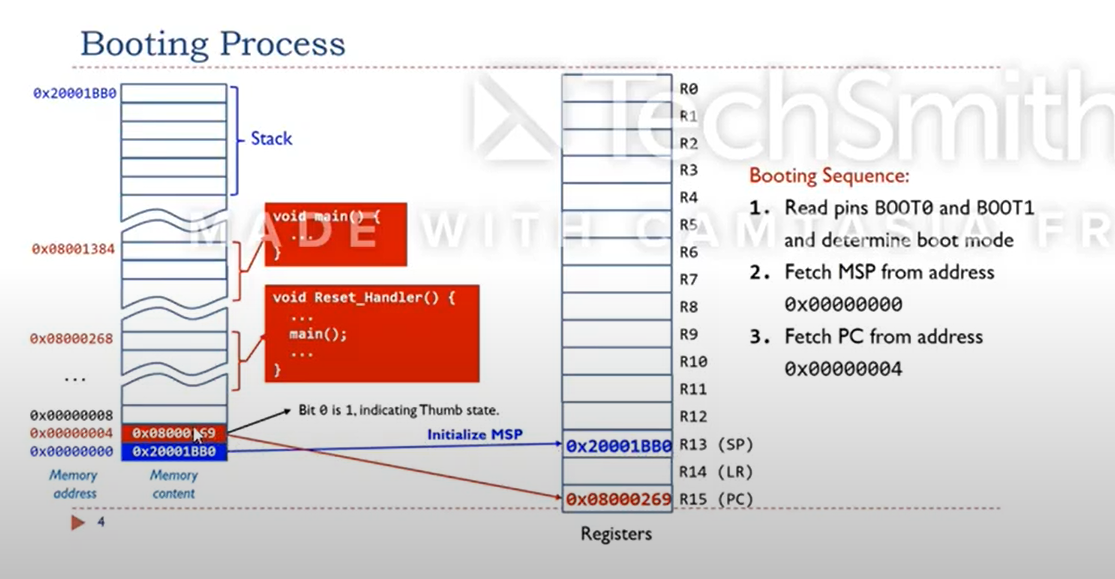
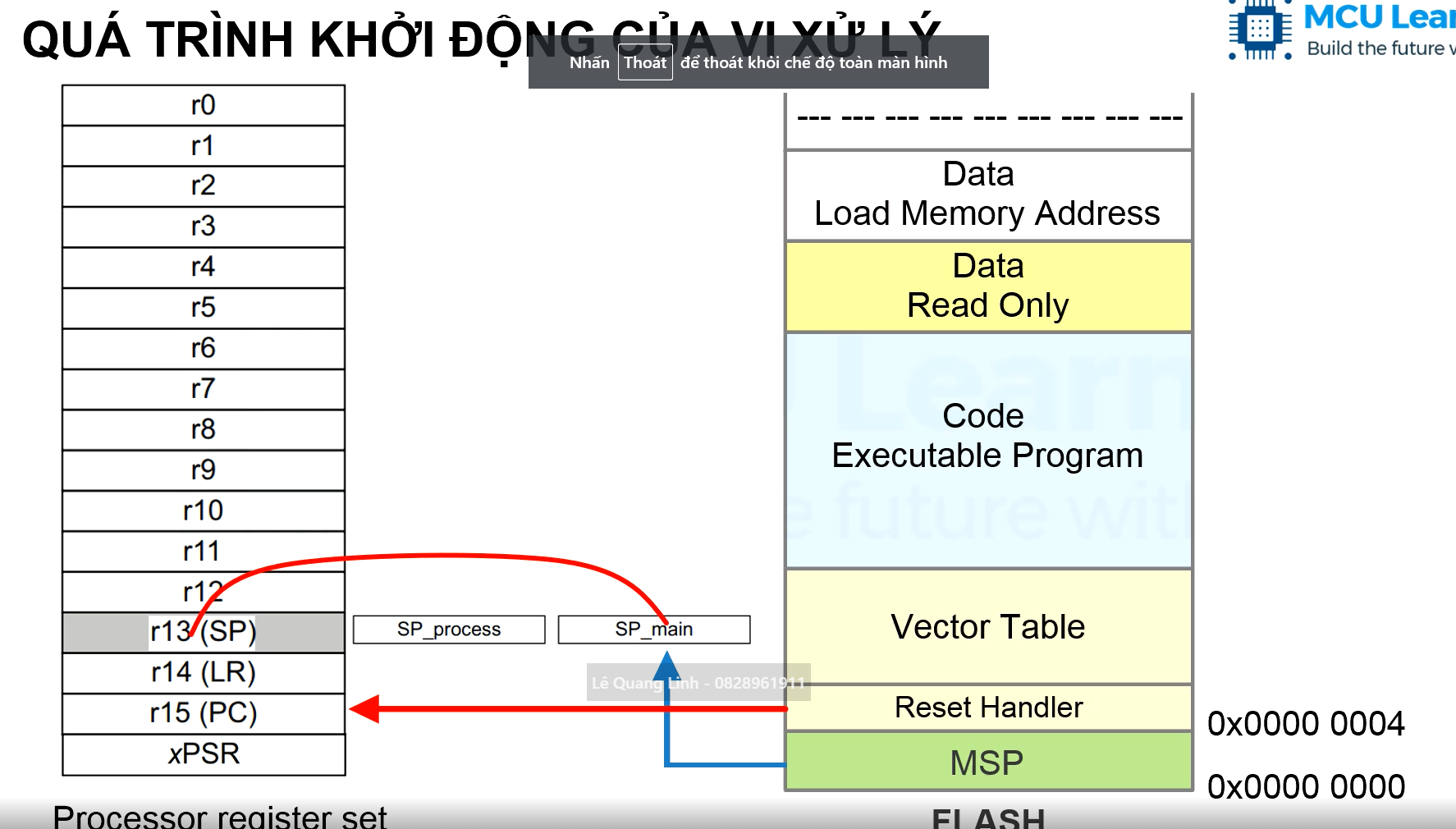
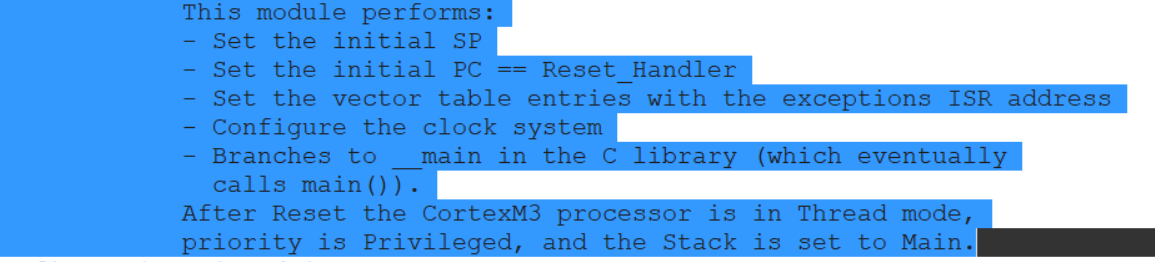
**Bài 2: BOOTING**





Sau khi reset vđk sẽ thực hiện theo thứ tự:

B1: Sau khi reset, sẽ nhảy vào địa chỉ 0x00 để thực hiện khởi tạo Stack pointer(value) nghĩa là nạp giá trị ở địa chỉ 0x00 vào thanh ghi SP

B2: Khởi tạo giá trị cho Program counter, nghĩa là lấy giá trị ở địa chỉ 0x04 gán vào thanh ghi PC. Địa chỉ này chính là địa chỉ của Reset Handler()

Địa chỉ: 0x0800 0001, số 1 ở cuối là chỉ kiểu THUMB, nhưng vẫn sẽ nhảy với địa chỉ 0x0800 0000

B3: Thanh ghi PC lưu địa chỉ câu lệnh tiếp theo nên vxl sẽ nhảy tới và thực hiện hàm Reset Handler()

Trong hàm Reset Handler() thực hiện

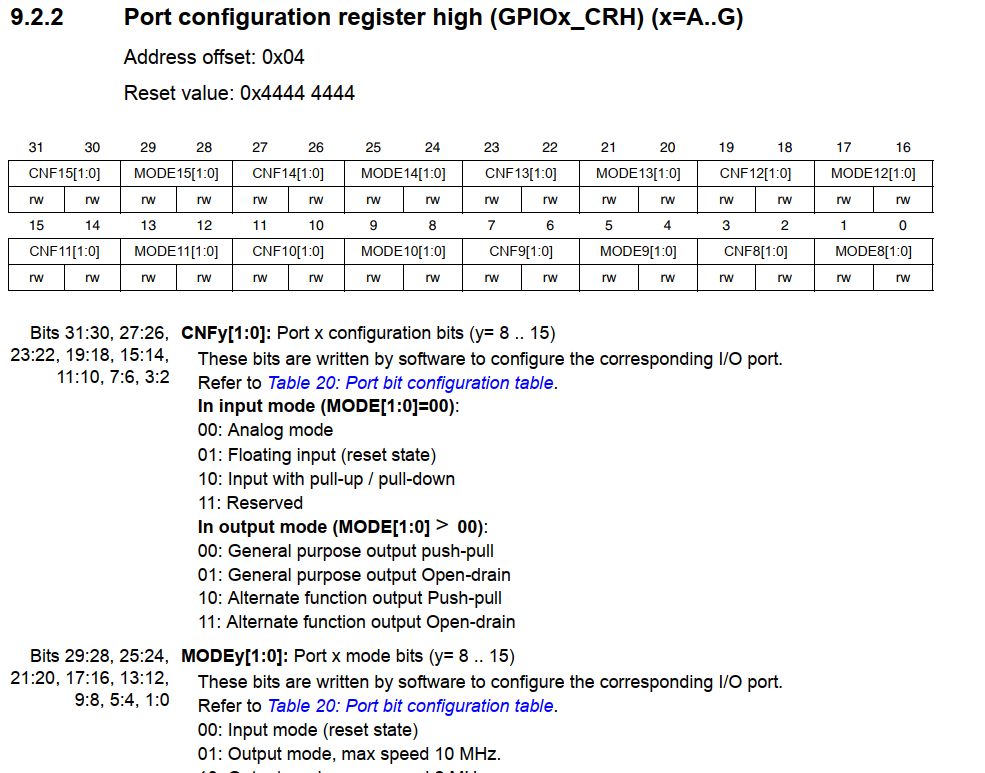
* KHởi tạo System Init: khởi tạo clock, gán giá trị thanh ghi VTOR = flash\_base | vect\_tab\_offset
* Nhảy vào hàm main

Mỗi ngắt sẽ có hàm thực hiện ngắt cố định ở địa chỉ đã được đăng kí sẵn trong bảng vector table

**Bài 11: GPIO**

B1.Đầu tiên phải cấp xung clock cho PORT: xem ở BLOCK DIAGRAM ở datasheet

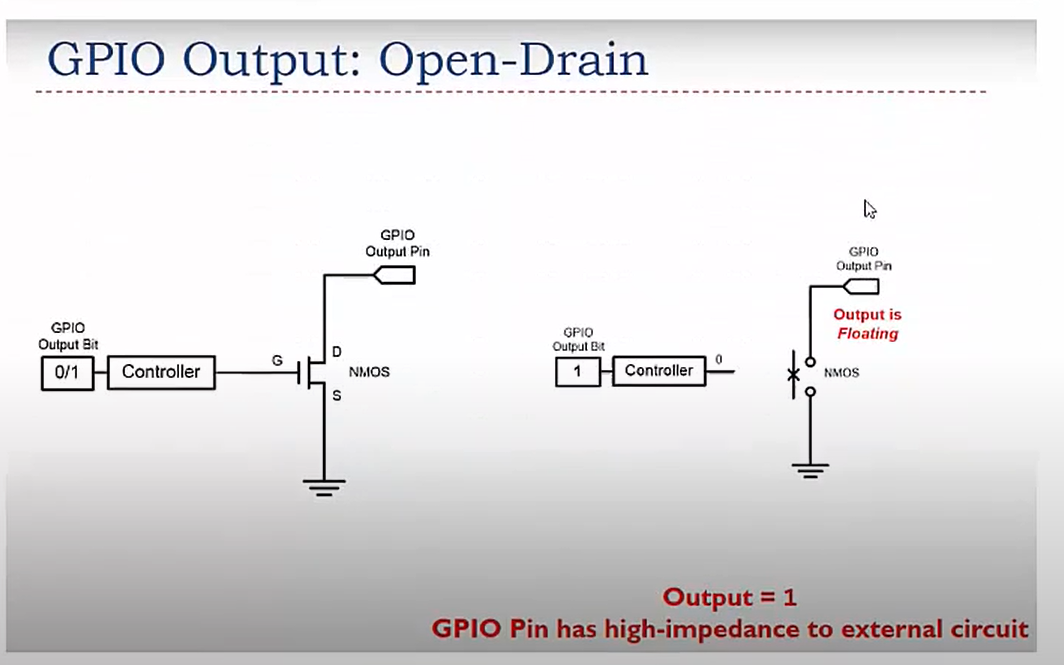
Vd:GPIOA,B,C.. có xung clock từ APB2, vào thanh ghi này để enable lên

B2. 

Thanh ghi CRL(pin 0-7) va thanh ghi CRH(pin 8-15) chứa các bit cấu hình cho PORT, các thanh ghi có giá trị ban đầu là 0x4444 4444 (4 = 0100) nghĩa là ban đầu các chân ở chế độ Input và floating input

Set MODEy > 00 để chọn mode output cho chân y, ban đầu sẽ ouput với open drain

-OPEN DRAIN Là chế độ khi =0 thì low, Khi =1 thì sẽ thả nổi (floating), không cao không thấp, phụ thuộc mạch ngoài



4

**Giải thích hàm Init:**

Vd: GPIO\_Init(GPIOA, &GPIO\_InitStructure);

* 2 giá trị truyền vào là địa chỉ của GPIOA và địa chỉ của struct GPIO\_InitStructure vừa mới được gán các giá trị
* Trong hàm Init, sẽ có 1 biến tmpreg 32 bit( bằng với độ dài thanh ghi), các giá trị của GPIO\_InitStructure sẽ được gán lần lượt hoặc sử dụng phép | để gán nhiều cái lại với nhau
  + Lần lượt: tmpreg &= ~pinmask;

tmpreg |= (currentmode << pos);

* + Hoặc tmpreg = mode1 | mode2 | enable<<position;

* Sau đó sẽ gán giá trị này cho thanh ghi tương ứng: GPIOx->CRL = tmpreg

**BÀI 19 Interrupt and Polling**

Trong hàm ngắt cần kiểm tra CỜ NGẮT và BIT ENABLE NGẮT để chắc chắn có ngắt xảy ra, vì có thể có trường hợp chưa eable ngắt mà cờ ngắt đã bật

If( cờ\_ngắt && bit\_en\_ngắt)

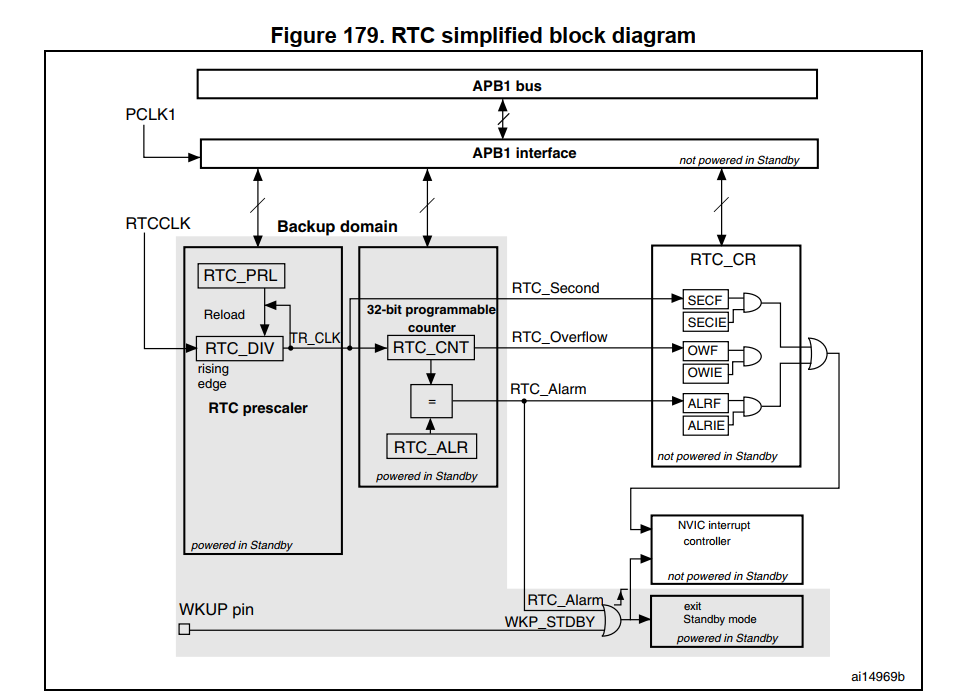
{

Cau lenh;

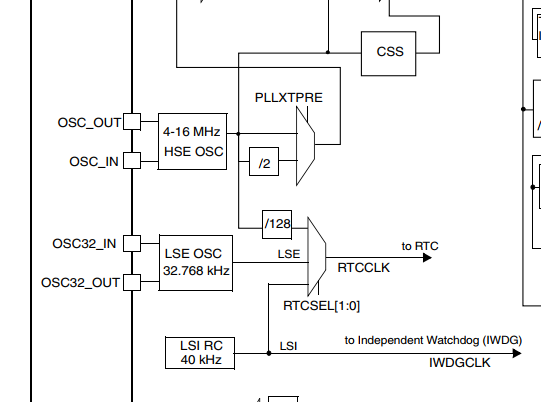
}

**BÀI 21:** RTC

Tần số đầu vào RTCCLK, qua xử lý ta được tần số bộ RTC fTR\_CLK, cứ sau 1 khoảng thời gian bằng 1 chu kỳ thì bit SECF được lên 1



B1:18.1 To enable access to the Backup  
registers and the RTC, proceed as follows:  
• enable the power and backup interface clocks by setting the PWREN and BKPEN bits  
in the RCC\_APB1ENR register  
• set the DBP bit the Power Control Register (PWR\_CR) to enable access to the Backup  
registers and RTC.

B2: chọn clock đầu vào cho RTCCLK: 3 loại, chọn loại nào thì phải enable loại đó lên rồi, rồi chọn clock cho RTC clock ở bit 8:9 thanh ghi RCC->BDCR ( mục 7.3.9) 

* HSE
* LSE
* LSI

B3: Enable RTC ( mục 18.3.4)

1. Poll RTOFF, wait until its value goes to ‘1’

2. Set the CNF bit to enter configuration mode

B4: Chọn bộ chia trước cho RTC

Giá trị bộ chia trước PRL[19:0] = PRLH[3:0] + PRLL[15:0]

* Tần số bộ RTC = fTR\_CLK = fRTCCLK/(PRL[19:0]+1)

B5: 4. Clear the CNF bit to exit configuration mode  
5. Poll RTOFF, wait until its value goes to ‘1’ to check the end of the write operation.

**Bài 23: Đồng hồ**

Cấu hình RTC 1 giây  
 Cứ sau 1 giây thì nhảy vào chương trình ngắt để tăng thời gian

Hiển thị LCD, dùng systick để tạo hàm delay\_ms()

**Bài 24: FLASH**

-Khi mất điện: - Ram: mất dữ liệu

-Flash: dữ liệu không mất đi, khi có điện lại thì không phải nạp code lại

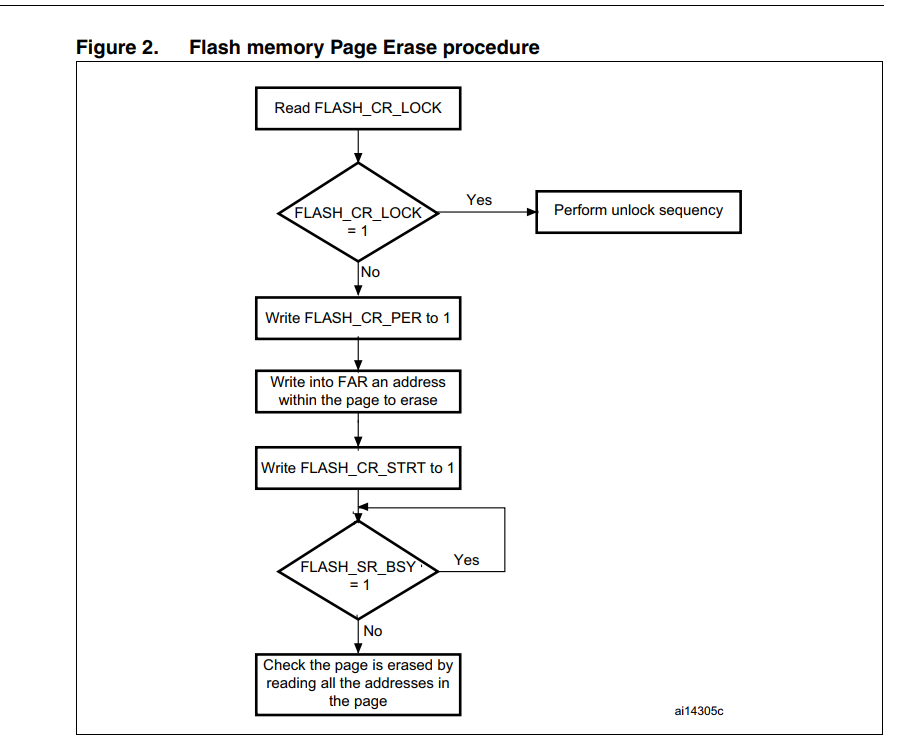
-Làm flash phải quan tâm tới Size và từng page của flash

-3 funtion quan trọng của FLASH : ghi, đọc, xóa

+Ghi : không thể ghi đè lên được nên trước khi ghi cần xóa 1 page trước

+ Xóa:

* Xóa theo Sector(xóa 1 phần/ 1 page) hoặc Mass(xóa toàn bộ).
* không nên xóa những page đầu vì những page này lưu code đang chạy
* B1: Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH\_CR register
* B2: thuc hien nhu hinh



* Trong đó unlock senquency nghĩa là (2.3.2)

After reset, the FPEC block is protected. The FLASH\_CR register is not accessible in write  
mode. An unlocking sequence should be written to the FLASH\_KEYR register to open up  
the FPEC block. This sequence consists of two write cycles, where two key values (KEY1  
and KEY2) are written to the FLASH\_KEYR address (refer to Section 2.3.1 for key values).  
Any wrong sequence locks up the FPEC block and FLASH\_CR register until the next reset

\*chú ý: trong thư viện người ta có thể define giá trị để dùng cho việc thay đổi giá trị thanh ghi

Vd: define FLASH\_SR\_BSY = 0x01;

Muốn cho bit đó lên 1 thực hiện: FLASH->SR |= FLASH\_SR\_BSY;

Muốn kiểm tra bit đó có bằng 1 không:

FLASH->SR & FLASH\_SR\_BSY == FLASH\_SR\_BSY;

**Bài 25: Từ khóa RWW trong Flash**

**RWW : Read while Write**

\*\* không thể vừa read và write trên cùng 1 Bank được( vì cùng 1 đường bus)

VD: đoạn code được lưu trong SECTOR 0(Bank 1), muốn write dữ liệu vào SECTOR 5(cũng bank 1) thì không thể làm trực tiếp được vì ta đang read code từ Bank 1(sector 0) và write vào Bank 1(sector 5)

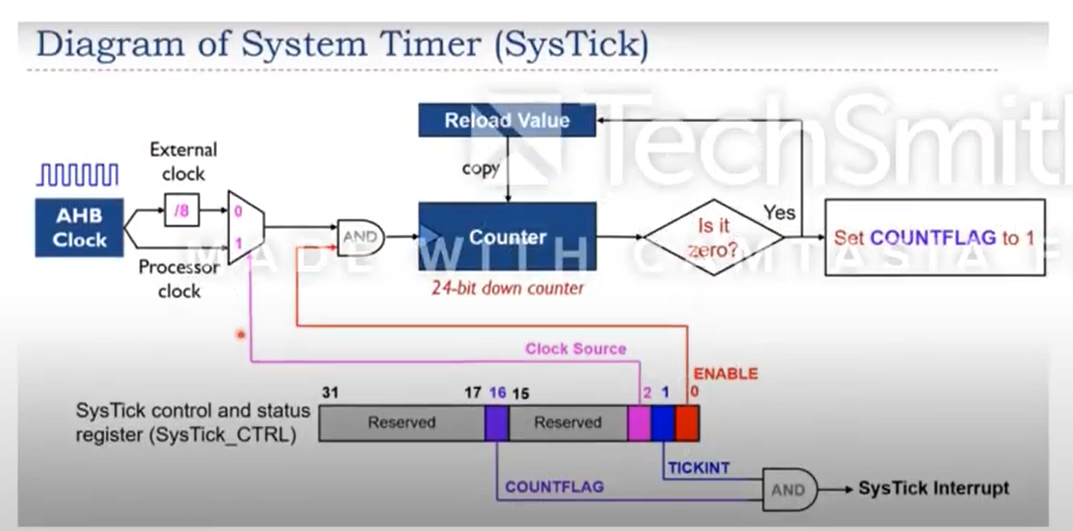
Giải pháp:

* Move đoạn code lên ram, khi đó việc run sẽ read code ở ram và write dữ liệu vào Sector 5 được

**Bài 26: Systick**

Systick là 1 bộ đếm, nghĩa là 1 tick trong hệ thống sẽ mất khoảng t/g

t =1/Sys\_clock

****

* Như 1 counter đếm xuống
* Có thể ứng dụng làm hàm delay
* Khi con chip được nạp, thì sẽ vào hàm SysInit trước, rồi vài main sau  
  Trong SysInit đã nạp SysClock = 72Mhz ban đầu (có thể thay đổi được)

**Bài 27: Systick(interrupt), đo t/g 1 đoạn code**

**Chưa học**

**Bài 28 Interrupt**

**1.Interrupt\_Config**

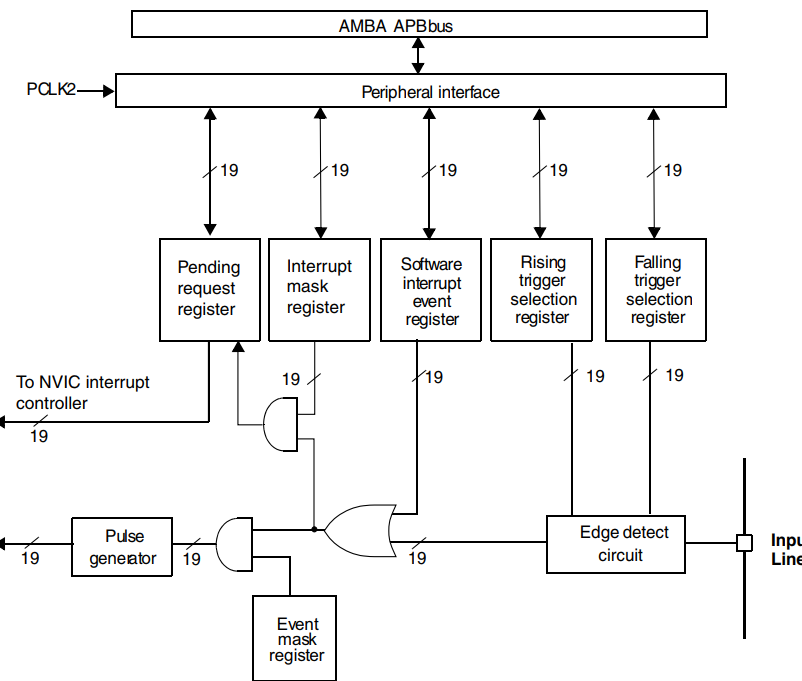
B1. GPIO input config

Alternate Function IO clock enabled (AFIOEN)

B2. chọn EXTIx cho chân cần ngắt trong 9.4.3 AFIO\_EXTICR1

Vd: exti0 cho chân PA) thì

B3. Chọn sườn lên/xuống cho ngắt ngoài VÀ event nếu là ngắt nội



B4. Enable interrupt mask

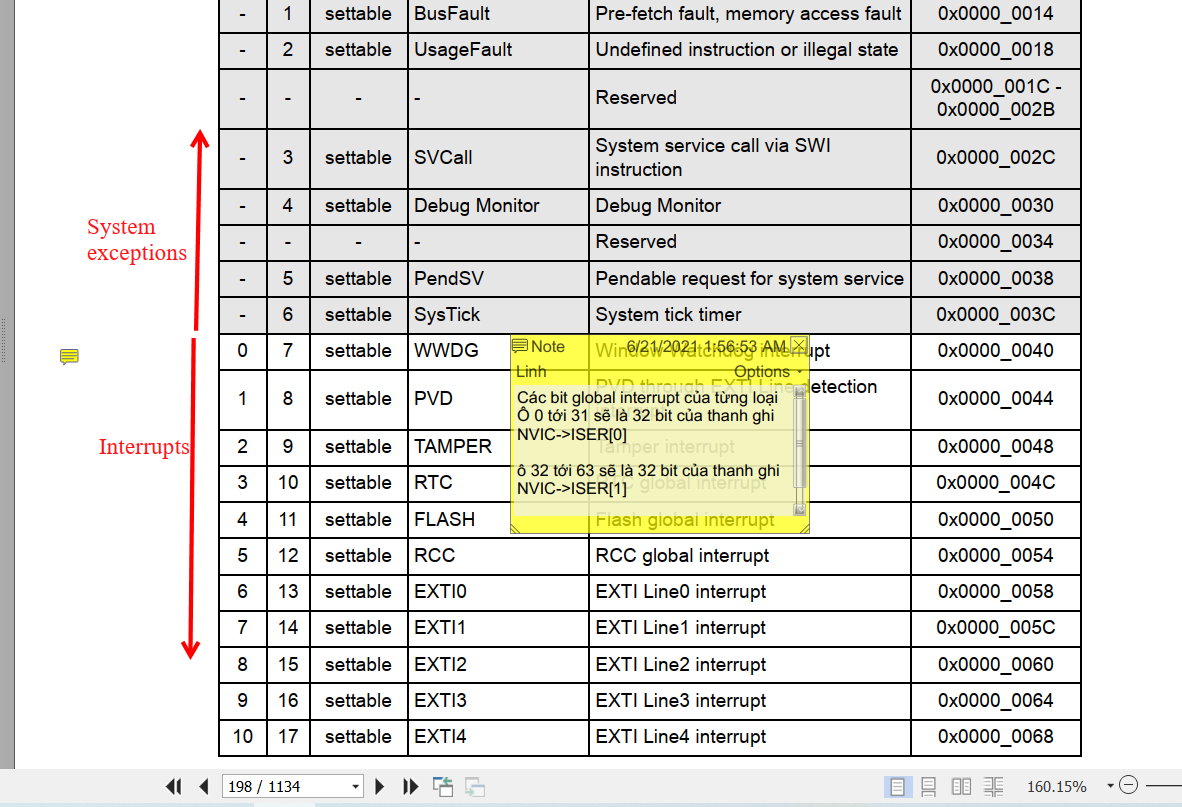
B5. Khi có ngắt xảy ra thì pending request register sẽ lên 1, nên phải clear bit đó trước(cờ ngắt)

B6. Enable ngắt trong tài liệu Core, thanh ghi NVIC  
Xem bảng vector table(10.1.2 của stm32f103 rm) để biết cần enable bit nào trong thanh ghi NVIC  
NVIC->ISER (interrupt select enable register)

Các bit global interrupt của từng loại

Ô 0 tới 31 sẽ là 32 bit của thanh ghi NVIC->ISER[0]

ô 32 tới 63 sẽ là 32 bit của thanh ghi NVIC->ISER[1]



**2.Interrupt Function**

Đặt tên hàm ngắt theo quy tắc

B1. Kiểm tra cờ ngắt (pending register) && bit enable ngắt

B2. Clear cờ ngắt, để dùng cho lần ngắt tiếp theo  
Thực hiện việc cần làm

**Bài 30 Timer**

Cứ 1 lần counter đếm lên 1 số tương ứng 1 tick = fTIM (f đầu vào của timer)

fTIM: f đầu vào của timer, tần số của 1 tick

T = (1/fTIM)/(auto reload value + 1): t/g đếm được 1 tick

prescal value: giá trị khi bắt đầu đếm Timer

auto reload value : là giá trị khi bộ đếm đạt được thì sẽ reload về 0

Cần tạo event update để các giá trị trong shadow register được nào vào thanh ghi gốc

1. **General-purpose timers (TIM2 to TIM5)**

The general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

The counter can count up, down or both up and down. The counter clock can be divided by a prescaler

**1.1 Timer 2 config**: mode time base uint

B1: Enable clock for timer 2: RCC->APB1ENR

B2: Select prescaller value: TimerX\_PSC

Tần số của 1 tick timer: fTIM = fCK\_PSC / (PSC[15:0] + 1)

B3: select upcounter or downcounter: TIM2->CR1

B4:

Select prescal value: tim2->cnt

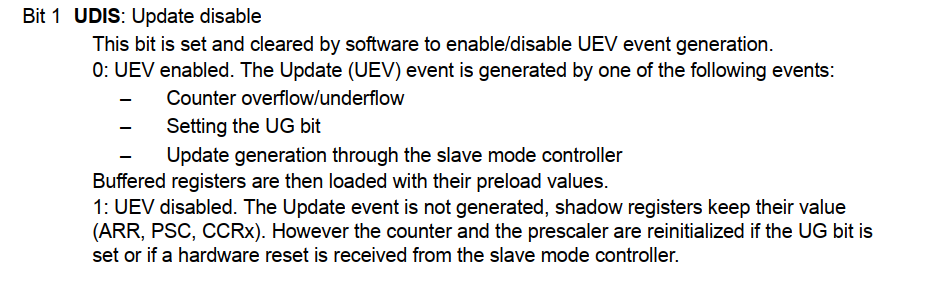
Select auto reload value: tim2->arr

B5: enable timer2 counter

B6: Bit 0 UG TIM2->EGR: Update generation

**\*\*Why need to generate a update event??**

* Because when the Update event is not generated, shadow registers (thanh ghi ảo) keep their value (ARR, PSC, CCRx).
* Sau khi có update event  
  This bit is set and cleared by software to enable/disable UEV event generation.  
  0: UEV enabled. The Update (UEV) event is generated by one of the following events:  
  – Counter overflow/underflow  
  – Setting the UG bit  
  – Update generation through the slave mode controller  
  Buffered registers are then loaded with their preload values.



**Bài 33: PWM**

Thanh ghi tim->psc (prescaller) lưu giá trị bộ chia trước, tần số 1 tick timer fTIM = fCK\_PSC / (PSC[15:0] + 1)

Thanh ghi tim->arr (auto reload register) lưu số tick+1 mà bộ timer đếm trong 1 chu kì của PWM => chu ki TPWM = (tim->arr + 1) \* (1/fTIM) = t/g 1 vòng đếm của bộ timer

Thanh ghi tim-CCRx (tương ứng với CHx) lưu giá trị của số tick mà PWM ở mức high (or low), còn tùy thuộc vào bit CCxP trong thanh ghi tim->CCER hoặc bit OCxM trong thanh ghi tim->CCMR1,2

Bit OCxM trong thanh ghi tim->CCMR1,2 là chọn chế độ cho đầu ra ở CHx của timer, xem trong 15.4.7

Bit CCxP trong thanh ghi tim>CCER quyết định

Vd. Chọn duty = 10%

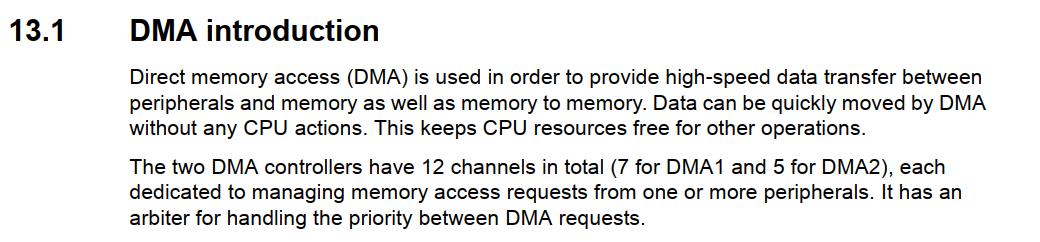
CC1P = 0: OC1 active high : 10% high và 90% low  
CC1P = 1: OC1 active low : 10% low và 90% high

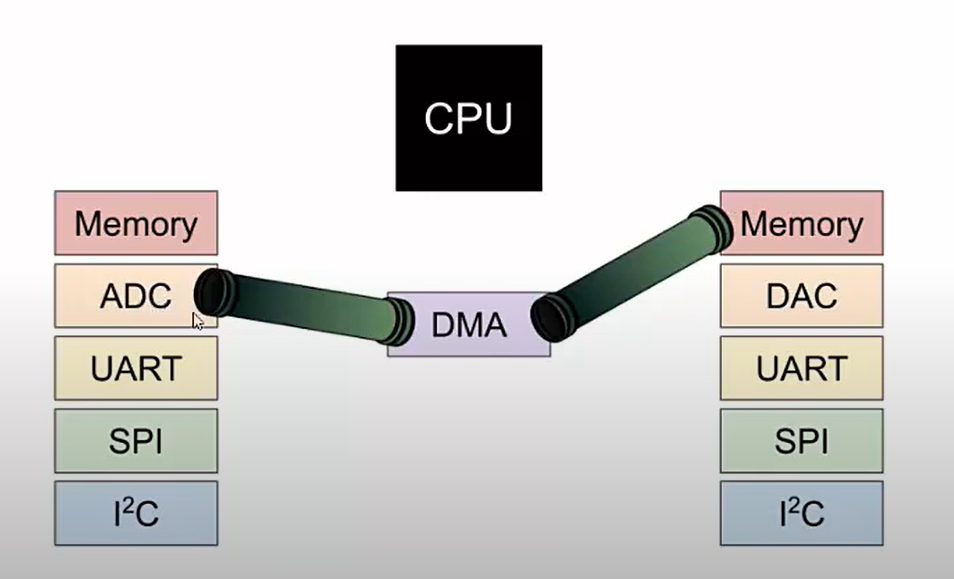
**Bài 36 : DMA**

Tham khảo: http://laptrinharmst.blogspot.com/2018/04/bai-10-dma-voi-stm32f103.html

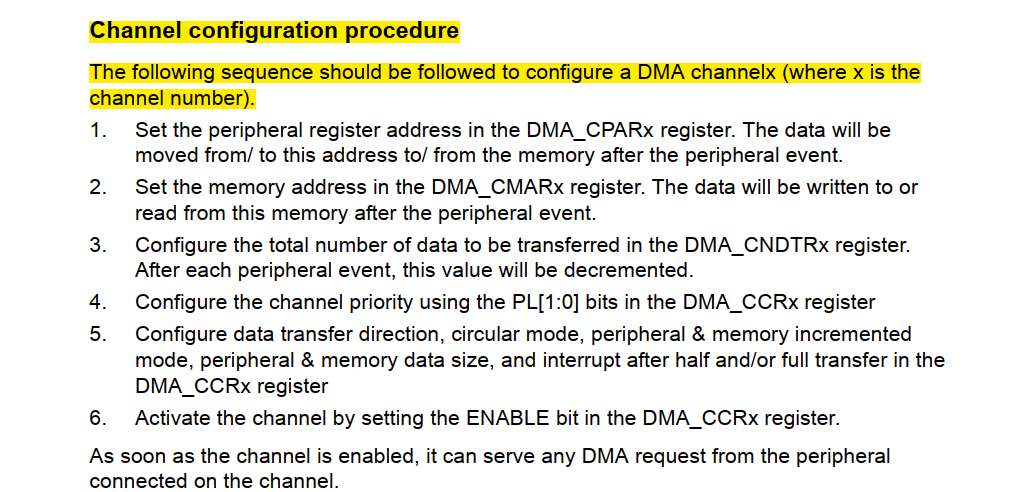
DMA : Direct memory access

Có thể làm những công việc như đọc dữ liệu ADC, chuyển dữ liệu trong các mảng dữ liệu mà không cần tới CPU, trong lúc đó CPU có thể làm việc khác



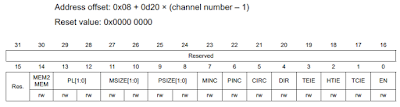


\*\* Các bước làm DMA:



**II. Thanh ghi quan trọng :**

* DMA\_CCRx – DMA channel x configuration register.

[](https://2.bp.blogspot.com/-SxTtShlT5Rg/WsSc1K-YRsI/AAAAAAAAAr0/XIhSh8RS0_EJPdof0zijJQxvPPe0gigMACKgBGAs/s1600/10_04.png)

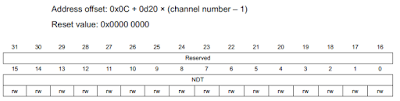
* + MEM2MEM: cho phép truyền DMA từ bộ nhớ đến bộ nhớ.
  + PL[1:0] : mức ưu tiên của channel tương ứng 00: thấp nhất, 11 cao nhất.
  + MSIZE[1:0] : kích thước của bộ nhớ 00,01,10 tương ứng 8 bit, 16 bit, 32 bit.

Số bit của data bộ nhớ

* + PSIZE[1:0] : kích thước ngoại vi 00,01,10 tương ứng 8 bit, 16 bit, 32 bit.

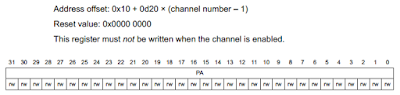
Số bit của data ngoại vi

* + MINC: có cho phép mode tăng địa chỉ bộ nhớ hay không.
  + PINC: có cho phép mode tăng địa chỉ ngoại vi hay không.
  + CIRC: có cho phép việc chuyển đổi được diễn ra liên tục hay không.
  + TEIE: cho phép ngắt khi có lỗi trong quá trình truyền hay không.
  + HTIE: cho phép ngắt khi truyền xong data ở chế độ half word.
  + TCIE: cho phép ngắt khi truyền xong data ở chế độ word.
  + EN : cho phép bộ DMA hoạt động hay không.
* DMA\_CNDTRx – DMA channel x number of data register.

[](https://2.bp.blogspot.com/-mH_wEEQo2pY/WsSc5wP_cxI/AAAAAAAAAr8/YiVKsnkWlfoxh-QRw_fZFlZUsDY_sF4kwCKgBGAs/s1600/10_05.png)

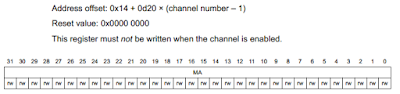
##### Thanh ghi này có giá trị là 16 bit tương ứng với 65535, chứa số lượng data truyền.

* DMA\_CPARx – DMA channel x pheripheral address register.

[](https://2.bp.blogspot.com/-RPklb7hJMYY/WsSdCqhzF3I/AAAAAAAAAsA/pIRh8VJdXPIJ1q6F5a8AfCCxjsFkFcSHwCKgBGAs/s1600/10_06.png)

##### Thanh ghi này là thanh ghi 32 bit chứa địa chỉ của ngoại vi.

* DMA\_CMARx – DMA channel x memory address register.

[](https://3.bp.blogspot.com/-uJNL1T7KmJg/WsSdGDzroLI/AAAAAAAAAsE/3JlN-pRyU2I10LX6CG5825OS4uxqiEVWwCKgBGAs/s1600/10_07.png)

##### Thanh ghi này là thanh ghi 32 bit chứa địa chỉ của bộ nhớ.

**III. Nguyên lý hoạt động : quang trọng**

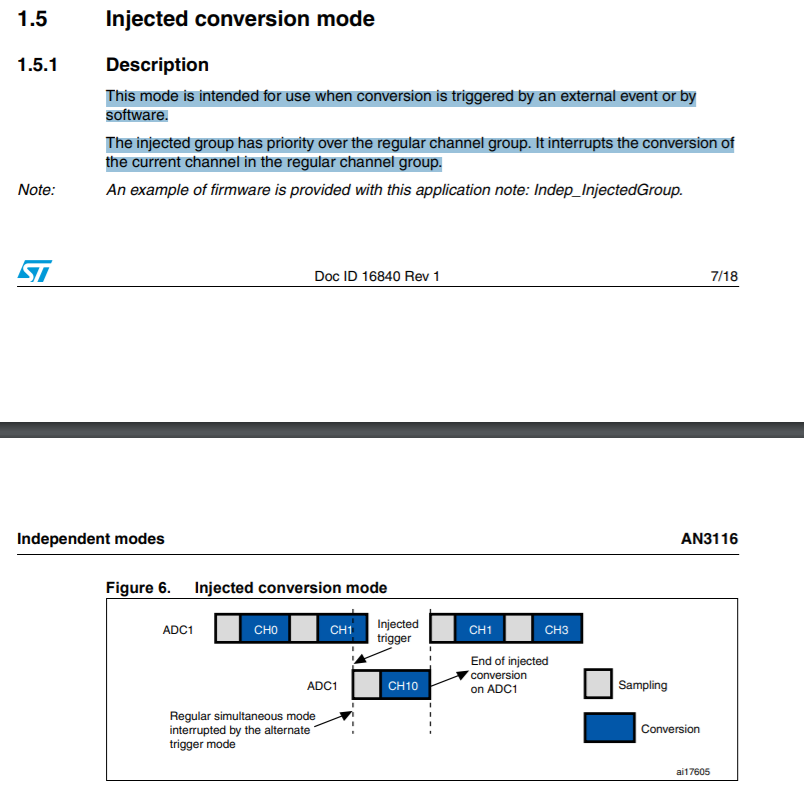
* DMA giúp truyền giữ liệu mà không cần tới CPU, nên sẽ là nhanh nhất trong việc truyền bộ nhớ và trong lúc đó CPU có thể làm việc khác
* Có thể truyền từ :
  + Mem to mem
  + Không phải từ mem to mem, ví dụ mem to peripheral, peripheral to mem,peripheral to peripheral
* Khi bit MINC trong thanh ghi CCR = 1, thì sau mỗi lần truyền dữ liệu, địa chỉ của ĐÍCH sẽ tăng lên (nếu đích là memory, còn nếu đích là ngoại vi thì chưa biết, vì chưa test)
* Khi bit PINC trong thanh ghi CCR = 1, thi sau mỗi lần truyền, địa chỉ của START\_ADDRESS sẽ tăng lên( nếu Start\_Address là memory và bit Mem2Mem = 1, còn nếu là ngoại vi thì chưa biết, chưa test)

**Bài 38,39: ADC**

ADC Analog to Digital Convert là bộ chuyển đôi tương tự sang số. Đại lượng tương tự là Điện áp Vin được so sánh với điện áp mẫu Vref (giá trị lớn nhất), sau đó được chuyển đổi thành số lưu vào thanh ghi DATA của bộ chuyển đổi đó.

ADC có regulả group channel vs injected group channel, 2 cái này khác nhau: sự khác nhau tuân theo hình dưới

* Regular group: là chuyển đổi ADC thông thường
* Injected group:
  + Chế độ này được thiết kế để sử dụng khi chuyển đổi được kích hoạt bởi một sự kiện bên ngoài hoặc bởi phần mềm.
  + Injected group có quyền ưu tiên hơn Regular group. Nó làm gián đoạn việc chuyển đổi kênh hiện tại trong nhóm kênh thông thường.



Chân ngoại vi kết nối ADC phải ở chế độ Analog Input

Có 2 tham số quan trọng của bộ ADC cần lưu ý:

* Tốc độ lấy mẫu (sampling) được tính theo số chu kì chuyển đổi
* Độ phân giải: Tính theo Bit bộ ADC có độ phân giải 10 Bit sẽ có 2^10 = 1024 giá trị

**Các chức năng chính của ADC trong STM32**

* Độ phân giải 12Bit
* Sinh ra ngắt tại các sự kiện End of convert, End of Injected, Analog Watchdog
* Chế độ Single hoặc Continuous
* Chế độ Scan tự động quét từ Kênh 0 đến Kênh n (mỗi bộ có 10 kênh tối đa)
* Có cơ chế cân chỉnh tay
* Data Alignment (Căn chỉnh Data) căn trái hoặc căn phải
* Cài đặt thời gian chuyển đổi đến từng Kênh
* Có thể kích hoạt bằng xung bên ngoài
* Chế độ Dual mode sử dụng cùng lúc 2 hoặc nhiều bộ ADC
* Hỗ trợ DMA

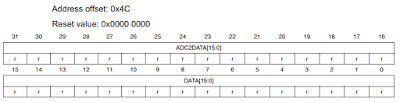
Kết quả convert được lưu vào thanh ghi DR

Bắt đầu convert bằng cách set bit ADON lên 1

**II. Một số thanh ghi quan trọng.**

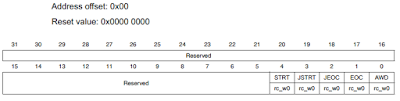
* + 1. **Các mode cấu hình**
* **Scan Conversion Mode**: Mỗi bộ ADC tại 1 thời điểm chỉ lấy mẫu được trên 1 kênh vđk,vì vậy khi sử dụng 1 bộ ADC nhưng cần lấy mẫu trên nhiều kênh, chúng ta cần Enable cho chế độ này để quét liên tục các kênh
* **Continuous Conversion Mode**: cho phép lấy mẫu liên tục, nếu không cho phép thì sau mỗi lan chuyen doi, ta se phai gui lai lenh doc giá tri ADC de bat dau quá trình chuyen doi moi
* **DMA Continuous Requests**: ở bài này chúng ta sử dụng DMA nên sẽ phải Enable chức năng này.
* **Number Of Conversion**: số chuyển đổi được thực hiện, ở đây mình sẽ điền “3” vì sử dụng 3 kênh ADC.
* **Rank**: thiết lập mức ưu tiên cho các kênh ADC, kênh có rank nhỏ hơn thì độ ưu tiên cao hơn.
* **Samping Time**: 239.5 Cycles.Thời gian chuyển đổi tính theo chu kì ADC, Sampling time càng cao thì chuyển đổi càng chính xác

1. ADC\_DR – ADC regular data register.

[](https://3.bp.blogspot.com/-UBDs-VG190o/Wpu9C1KA5WI/AAAAAAAAAho/lOyRC6qieyIrlQP0bUkkzGGs0ZbyMD18ACKgBGAs/s1600/09_08.png)

Thanh ghi này chứa giá trị ADC đọc về, nó là thanh ghi 32 bit với 16 bit data của bộ ADC1 và 16 bit data của bộ ADC2.

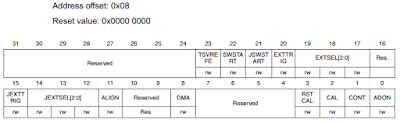
1. ADC\_SR – ADC status register.

[](https://4.bp.blogspot.com/-iq6bpY51Las/Wpu9GYiZFkI/AAAAAAAAAhs/OUe3f2VSifkCu4-Xi0wkdaik_u5GuHxcACKgBGAs/s1600/09_10.png)

Thanh ghi này chứa các cờ báo trạng thái như:

* 1. STRT : báo channel đã bắt đầu chuyển đổi giá trị ADC hay chưa.
  2. JSTRT : báo channel đã bắt đầu chuyển đổi khi có tín hiệu bên ngoài điều khiển hay chưa.
  3. JEOC: báo kết thúc quá trình chuyển đổi khi có tín hiệu bên ngoài điều khiển hay chưa.
  4. EOC: báo kết thúc quá trình chuyển đổi ADC.
  5. AWD: báo có sự kiện Analog Watchdog có xảy ra hay không.

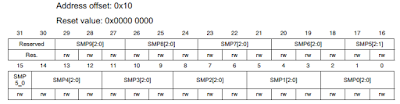
1. ADC\_CR2 – ADC Control register 2.

[](https://1.bp.blogspot.com/-4r4tDIw5PaM/Wpu9KXAJmZI/AAAAAAAAAhw/G2v-qwo6MiIQycFRBvs0XOLhr3AKx3Y1QCKgBGAs/s1600/09_11.png)

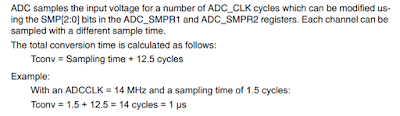
Thanh ghi này điều khiển các quá trình chuyển đổi ADC như:

* 1. TSVREFE: bật hay tắt cảm biến nhiệt độ và Vrefint.
  2. SWSTART : bật hay reset trạng thái bộ chuyển đổi liên tục.
  3. JSWSTART: bật hay reset trạng thái bộ chuyển đổi liên tục được điều khiển từ bên ngoài bộ ADC.
  4. EXTTRIG: cho phép hay không cho phép bắt đầu bộ chuyển đổi liên tục từ xung trigger bên ngoài.
  5. EXTSEL[2:0] : bit chọn lựa xung trigger bên ngoài từ nguồn nào.
  6. ALIGN : chọn 12 bit lưu giá trị là trái hay phải trong 16 bit
  7. DMA: có sử dụng bộ DMA hay không.
  8. RSTCAL: reset lại thanh ghi calib hay không.
  9. CAL: cho phép hay báo là đã calib xong.
  10. CONT: lựa chọn mode chuyển đổi liên tục hay chuyển đổi đơn.
  11. ADON: bật hay tắt bộ chuyển đổi ADC.

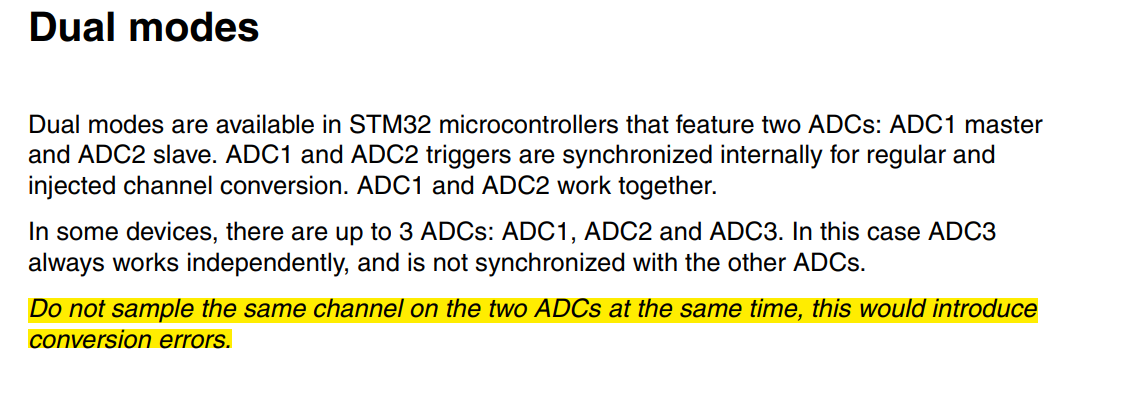
1. ADC\_SMPR2 – ADC sample time register.

[](https://1.bp.blogspot.com/-A5ptdBFj5CA/Wpu9QrxCAFI/AAAAAAAAAh0/u9NdiQLGaZE67FfeuVj8IBkMWSQHcjaAwCKgBGAs/s1600/09_12.png)

Thanh ghi này thiết lập thời gian lấy mẫu nhanh hay chậm và được cài đặt bởi lập trình.SMPx[2:0] tương ứng giá trị nhị phân từ 0->7 sẽ tương ứng với thời gian lấy mầu là: 1.5 - 7.5 - 13.5 - 28.5 - 41.5 - 55.5 - 71.5 - 239.5 cycles. Cách tính thời gian dựa theo hình sau:

[](https://2.bp.blogspot.com/-2obN0xHRYEg/Wpu9T2g_HWI/AAAAAAAAAh8/TxfFhRoi_KY1N55oJ5ti8OSwrsp9IFVwwCKgBGAs/s1600/09_13.png)

**III. Dual Mode**



**Bài 40: ADC Interrupt**

Khong dungf chung được với DMA

**Bài 45: IDWG- Independent Watchdog**

**Watchdog timer**là bộ đếm thời gian hoạt động liên tục nhằm tự động thực hiện một nhiệm vụ nào đó. Sau một khoảng thời gian được định trước nếu bộ đếm không được dừng hoặc refresh nó sẽ kích hoạt tác vụ đã được định sẵn ví dụ như reset thiết bị, tắt màn hình, phát tín hiệu cảnh báo, ...

1.Introductory

IWDG là bộ đếm thời gian, khi giá trị đếm về 0 thì sẽ reset lại vđk, ví dụ như chương trình bị lỗi hay bị đứng ở chỗ nào đó trong software.

2.Main Features

3.**functional description**

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (LSI- 40KHz) and thus stays active even if the main clock fails.

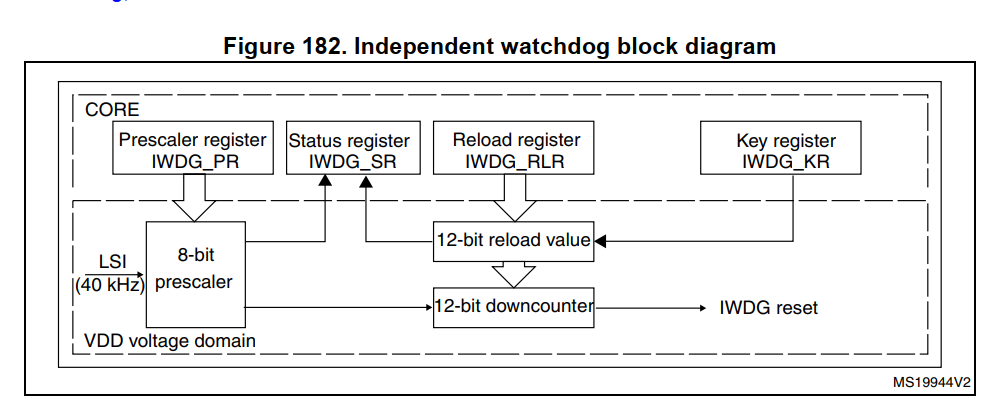
B1. Viết vào Key Register(KR) giá trị 0x5555 để cho phép ghi giá trị vào 2 thanh ghi PR và RLR

B2.ghi giá trị vào 2 thanh gi PR là RLR

* PR: bộ chia trước
* RLR: reload value
* =>time out = (1 / (40000/prescaller)) \* (RLR\_vale +1)

B3. Gán IWDG->KR = 0xaaaa để không cho phép ghi vào 2 thanh ghi PR là RLR nữa

B4. Gán IWDG->KR = 0xcccc để start watchdog timer



Mô tả hình: LSI đi qua prescaller ra tần số của watchdog + iwdg\_rlr = 12 bit đếm xuống, tới khi bằng 0 thì sẽ có tín hiệu reset

**Bài 48: CRC- cyclic redundancy check**

Khi truyền dữ liệu (SPI,DMA,…), khi truyền xong sẽ không biết được nó truyền đúng hay sai, có truyền đúng không, trên đường truyền có lỗi hay không

CRC có chức năng mã hóa DỮ LIỆU TRUYỀN và DỮ LIỆU NHẬN ĐƯỢC, nếu 2 mã này giống nhau thì truyền đúng, nếu khác thì truyền sai

Việc mã hóa sẽ do phần cứng thực hiện theo 1 quy tắc trong file CRC của STM32 có hướng dẫn

CRC-32 (Ethernet) polynomial : POLY là giá trị cố định được dùng khi mã hóa dữ liệu

Quá trình thực hiện:

* B1:Resets the CRC calculation unit và gán cho thanh ghi Data register CRC\_DR = 0xffffffff.
* B2: ghi giá trị cần mã hóa vào CRC\_DR, nếu là 1 mảng thì ghi lần lượt
  + Sau khi vào, hardware sẽ sử dụng giá trị cũ của thanh ghi CRC\_DR và giá trị vừa ghi vào để tạo ra 1 mã và cũng được lưu trên chính thanh ghi CRC\_DR
* B3: ghi giá trị tiếp theo nếu là mảng vào thì lại được 1 mã khác cho đến khi kết thúc
* B4. Lưu giá trị cuối cùng vào 1 biến, sau đó tiếp tực với mảng khác(mảng nhận được) rồi so sánh 2 mã cuối cùng với nhau. Nếu bằng nhau thì truyền đúng, nếu sai thì truyền bị lỗi

Các thanh ghi:

* CRC\_DR: để ghi giá trị vào và tạo ra mã

VD:

Truyền a[2] = {0x112; 0x43} được b[2]

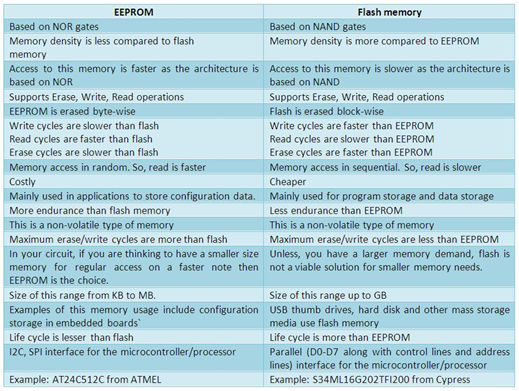
B1: CRC\_DR = 0, ghi vào 0x112 => mã a

B2:CRC\_DR = a, ghi vào 0x42 => được mã b

B3.tương tự với mảng b[2], nếu giống nhau thì sẽ truyền đúng

## Bài 49: EEPROM

EEPROM là một bộ nhớ khi mất điện thì dữ liệu không mất đi, bật lên lại vẫn còn, tương tự FLASH



|  |  |  |
| --- | --- | --- |
|  | **EEPROM** | **FLASH** |
| Xóa | Theo từng byte 1 được | Theo từng sector(vùng nhớ) |
| Tốc độ | -Write slower  - Read faster  - Eraser slower | - Write Faster  - Read Slower  - Eraser Faster |
| Giới hạn ghi xóa | Ít hơn | Nhiều hơn |
| Giá | Đắt | Rẻ hơn |

**KHông hiểu @**

**Bài 50: Các bộ lọc bằng Code**

## Bài 51: Encoder

TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S  
bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00,  
the channel is configured in input and the TIMx\_CCR1 register becomes read-only.

Then write IC1F bits to 0011 in  
the TIMx\_CCMR1 register.

Select the edge of the active transition on the TI1 channel by writing CC1P bit to 0 in  
the TIMx\_CCER register (rising edge in this case)

(write IC1PS bits to ‘00’ in the  
TIMx\_CCMR1 register).

Enable capture from the counter into the capture register by setting the CC1E bit in the  
TIMx\_CCER register

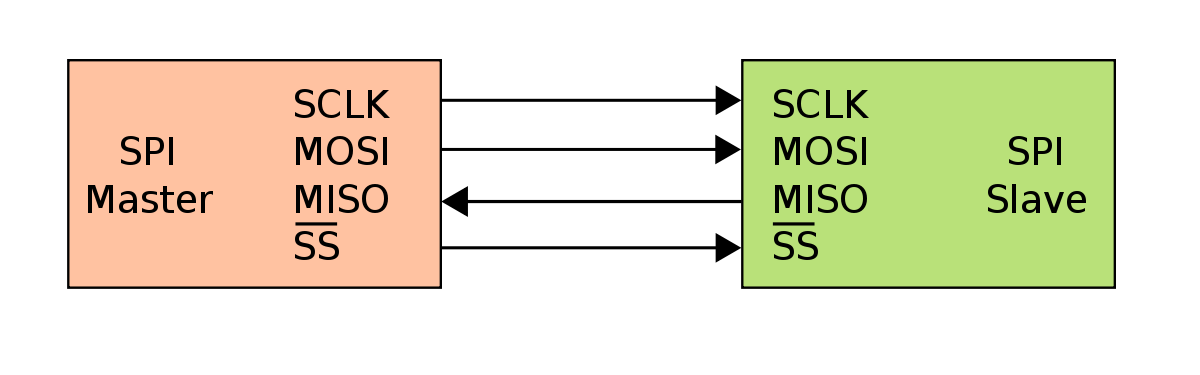
**Bài 59: SPI**

I. Tổng quan

<https://tapit.vn/chuan-giao-tiep-spi-tren-stm32f4/>

SPI (Serial Peripheral Interface) – Hay còn được gọi là Four wire – Chuẩn giao tiếp 4 dây. Kênh truyền nối tiếp song công (full duplex – có 2 đường truyền dữ liệu riêng biệt, một đường truyền và một đường nhận, cho phép dữ liệu truyền nhận theo 2 hướng cùng lúc).

* SPI là một giao thức kiểu Master – Slave. Master là thiết bị điều khiển (thường là vi điều khiển), còn Slave (thường là các cảm biến, màn hình LCD hoặc chip nhớ) sẽ nhận lệnh từ Master.
* Một hệ thống SPI chỉ gồm một thiết bị Master, nhưng lại có thể có một hoặc nhiều thiết bị Slave.
* Khối SPI của thiết bị Master sẽ kết nối với khối SPI của thiết bị Slave qua 4 chân tín hiệu:



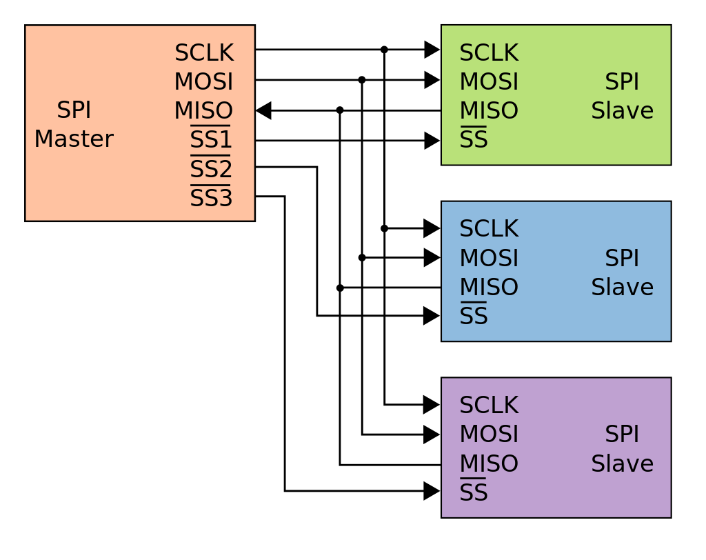
MOSI (Master Output/Slave Input) – Master gửi dữ liệu đến Slave.

MISO (Master Input/Slave Output) – Slave gửi dữ liệu cho Master.

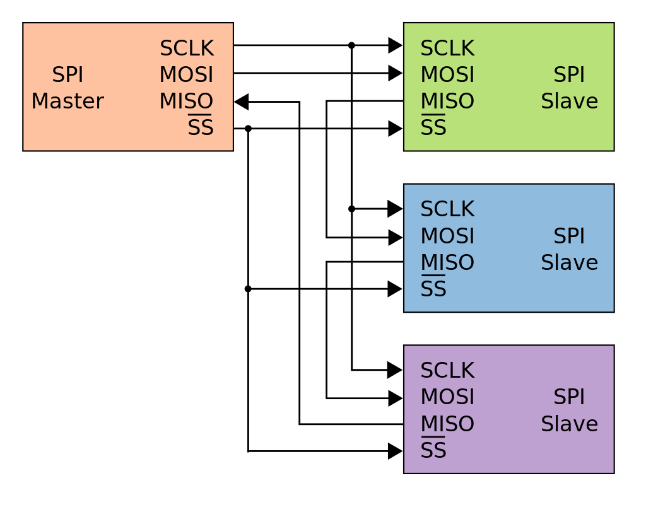
SCLK (Serial Clock) – xung giữ nhịp Clock.

SS/CS (Slave Select/Chip Select) – Master lựa chọn Slave để giao tiếp. Cụ thể trong trường hợp hệ thống gồm một Master giao tiếp với nhiều Slave, Master điều khiển tín hiệu mức logic trên chân SS/CS của Slave nào xuống mức LOW (mức 0), nghĩa là Master đang giao tiếp với Slave đó.

CÁCH MẮC NHIỀU SLAVE VS 1 MASTER



OR



Ưu điểm:

* Dữ liệu được truyền liên tục và không gián đoạn, vì không có bit Start và bit Stop.
* Hệ thống định địa chỉ đơn giản.
* Dây MOSI và MISO riêng biệt nên có thể đồng thời nhận và truyền dữ liệu.
* Tốc độ truyền dữ liệu nhanh và tiết kiệm năng lượng.

Nhược điểm:

* Sử dụng nhiều dây để kết nối với ngoại vi.
* Không có dấu hiệu nhận biết dữ liệu đã được truyền thành công.

Không có hình thức kiểm tra lỗi (như bit chẵn lẻ của UART). Có thể kiểm tra bằng CRC

* Khoảng cách các thiết bị truyền nhận bằng SPI rất ngắn.

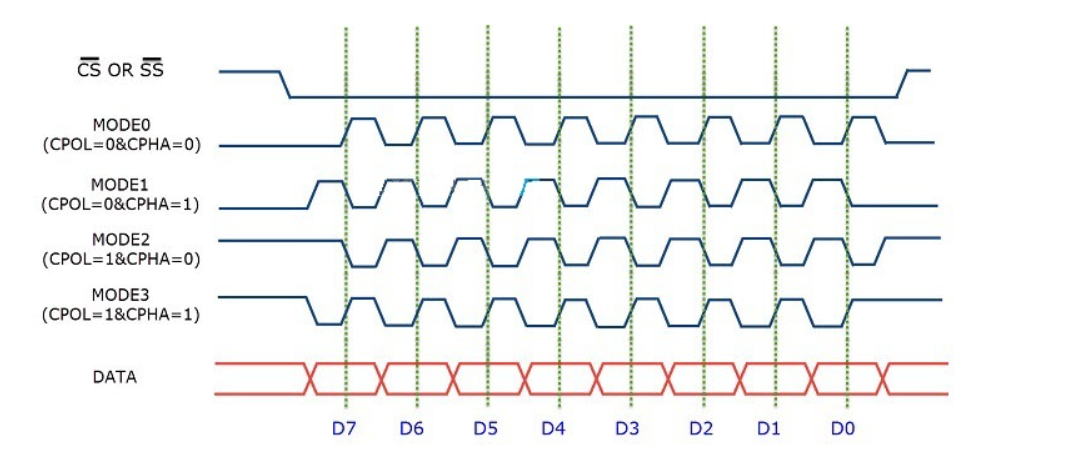
II.Hoạt động

Có 4 chế độ hoạt động của SPI dựa trên hai Bit **CPOL** và **CPHA**

CPOL: Clock Polarity xác định mức tín hiệu tại Clock lúc nhàn rỗi (Idle), Nếu CPOL = 0, lúc không gửi dữ liệu (nhàn rỗi) Clock sẽ ở mức thấp CPOL = 1 sẽ ngược lại

CPHA: Phase Clock xác định quá trình truyền dữ liệu tại sườn tín hiệu thứ nhất hay thứ 2. Nếu CHPA = 0 thì truyền tín hiệu tại sườn thứ nhất, nếu =1 thì truyền tín hiệu tại sườn thứ 2. Tham khảo hình

Chấn CSorSS là tiêu chuẩn gửi nhận phụ thuộc vào slave hoặc master, ví dụ như hình dưới thì đưa chân SS = 0 để bắt đầu truyền, và đưa chân SS =1 khi kết thúc truyền



**1 số thuật ngữ:**

Flash:

* Bank : nhiều sector
* Sector(khu vực)
* Trigger : kích hoạt
* Sequence : trình tự, sự nối tiếp
* Successive = consecutive : liên tiếp
* Synchronized : đồng bộ
* Simultaneous : đồng thời