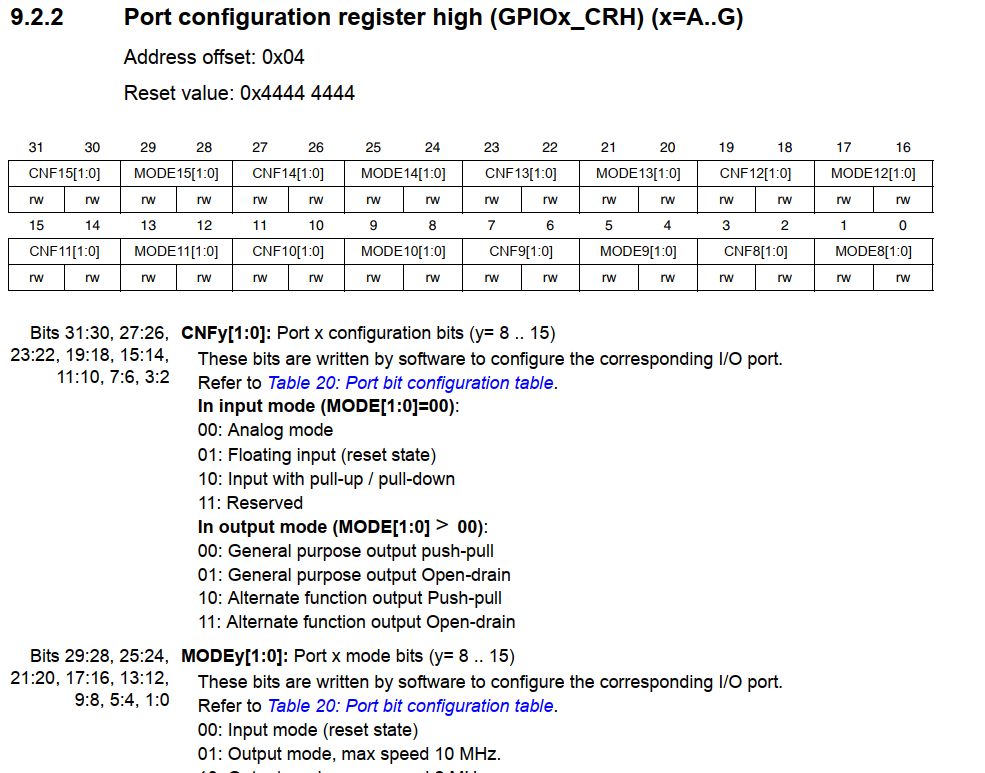
**Bài 11: GPIO**

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

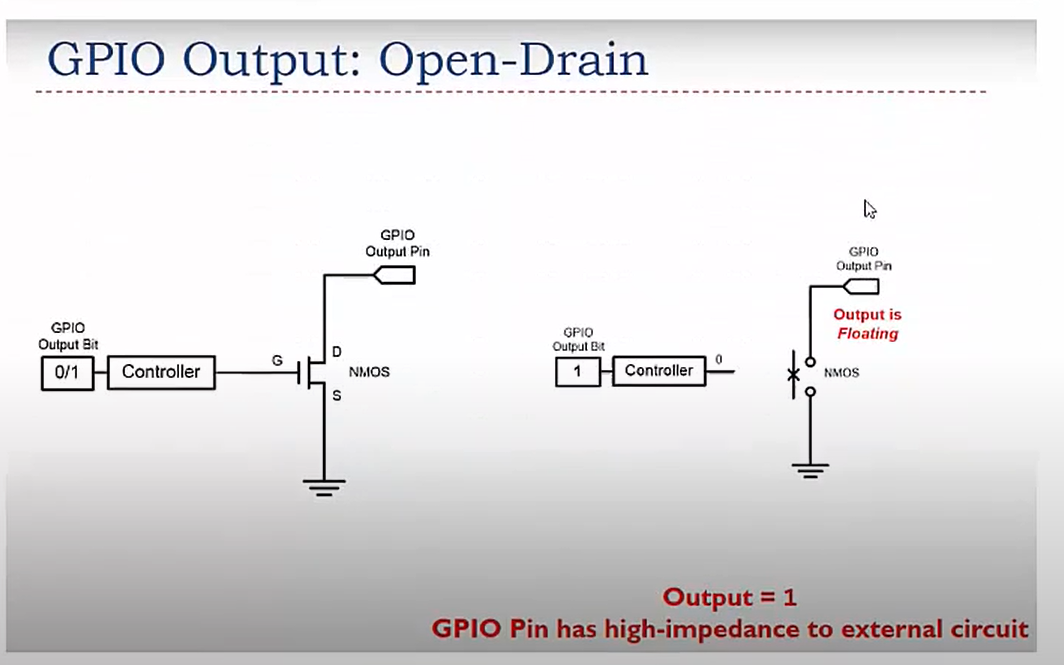
Vd:GPIOA,B,C.. có xung clock từ APB2, vào thanh ghi này để enable lên

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)

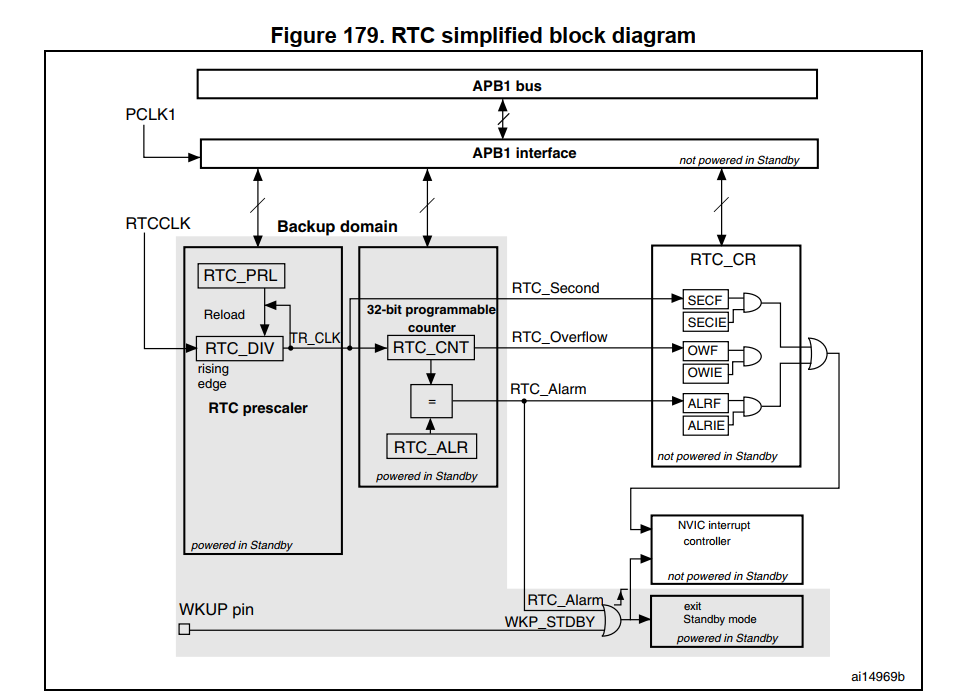
{

Cau lenh;

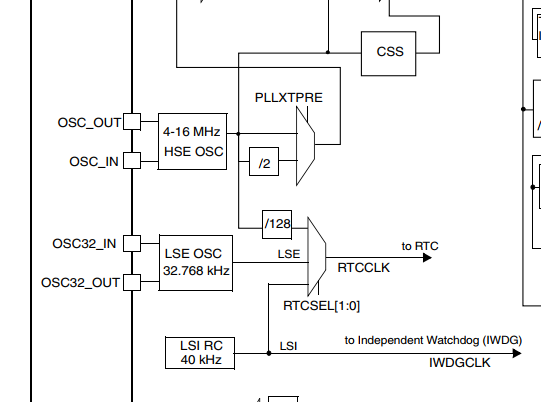
}

**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, vđk vẫn còn chạy

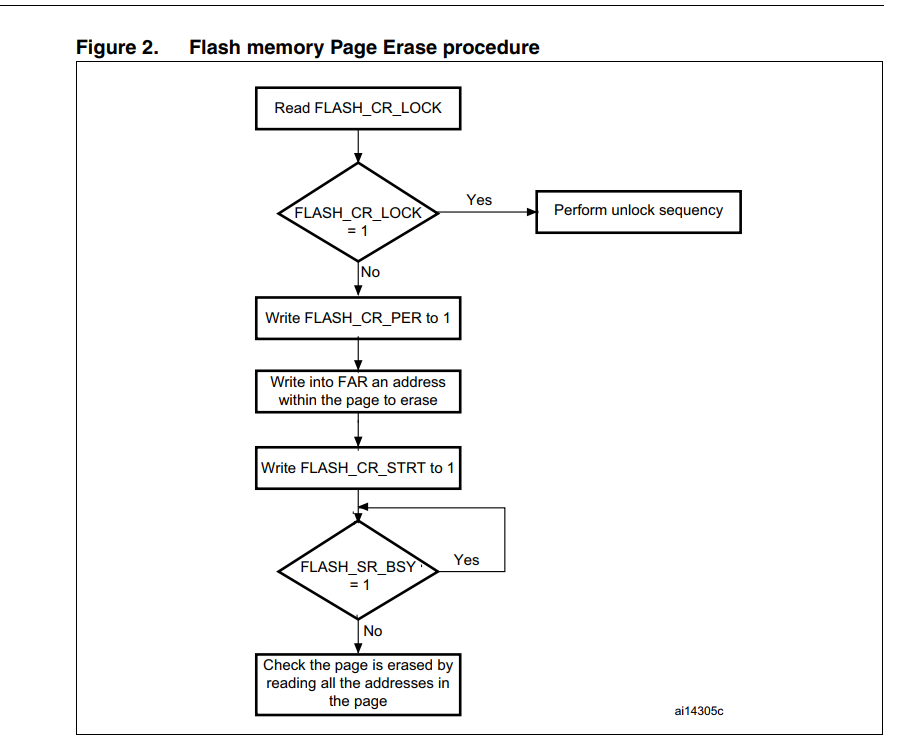
-Làm flash phải quan tâm tới Size và từng page của flash

-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: 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;

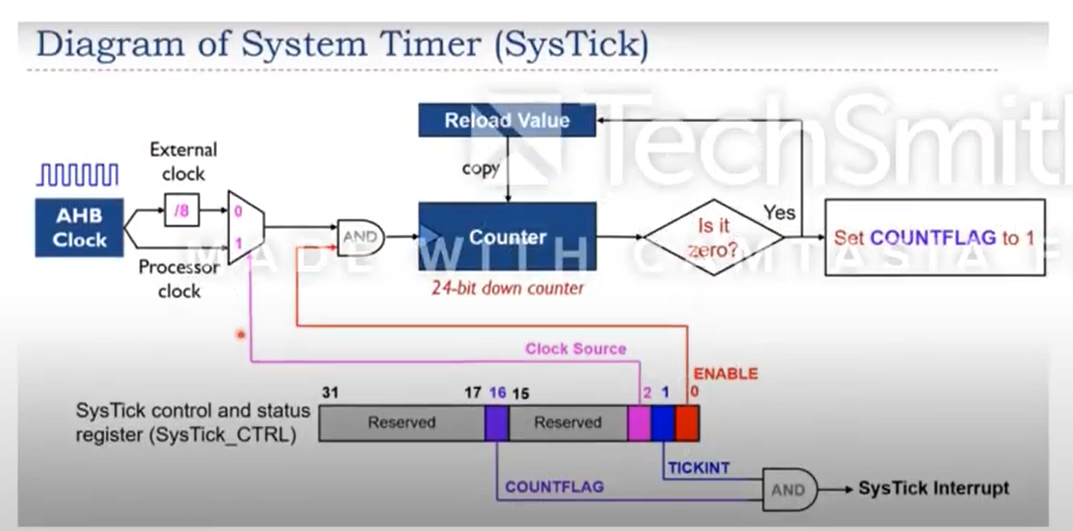
Muốn kiểm tra bit đó có bằng 1 không:

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

**Bài 26: Systick**

Systick 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**

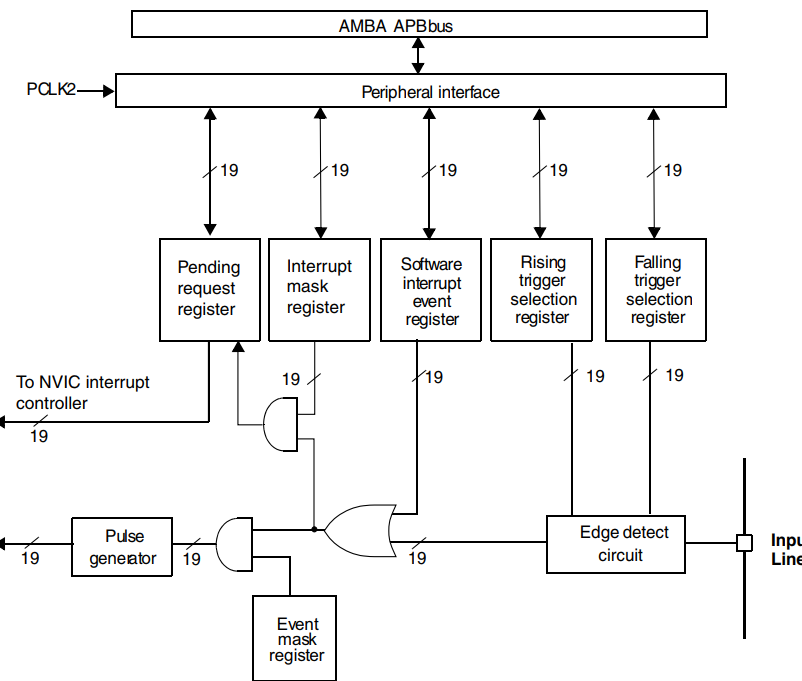
B1. GPIO input config

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

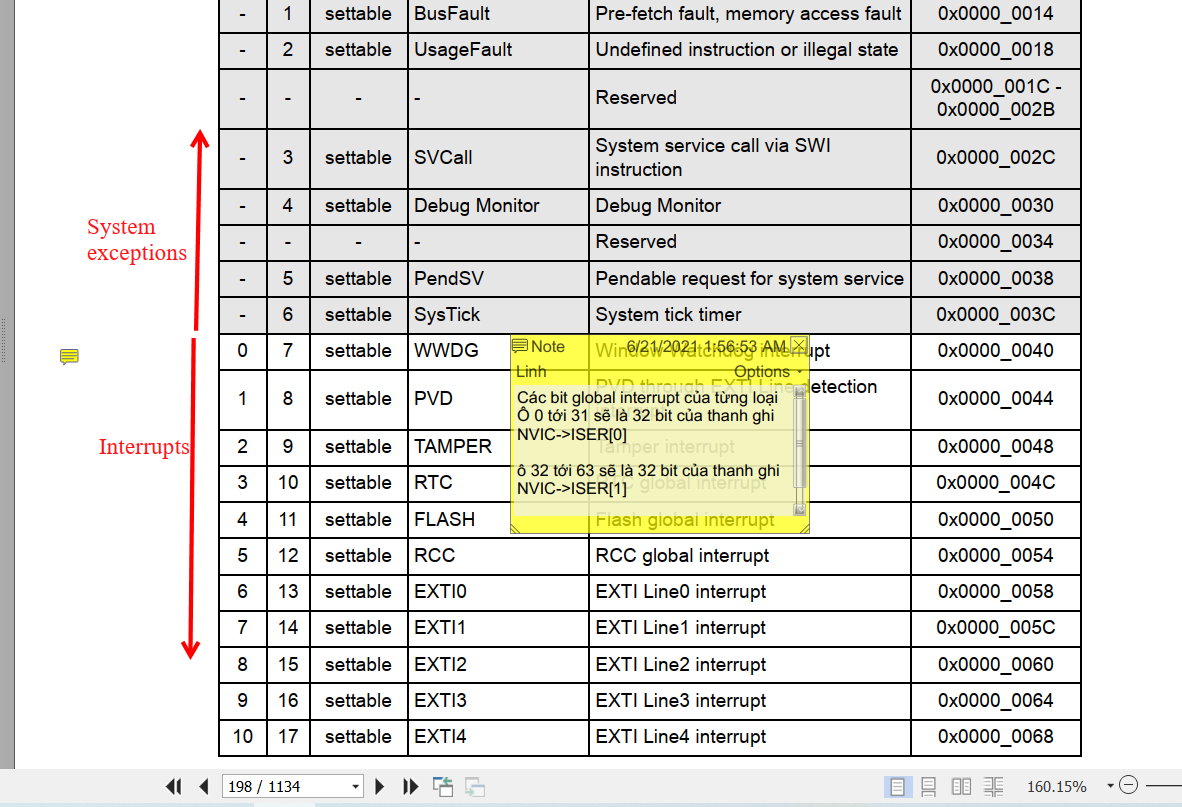
B5. Khi có ngắt xảy ra thì pending request register sẽ lên 1, nên phải clear bit đó trước(cờ ngắt)

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

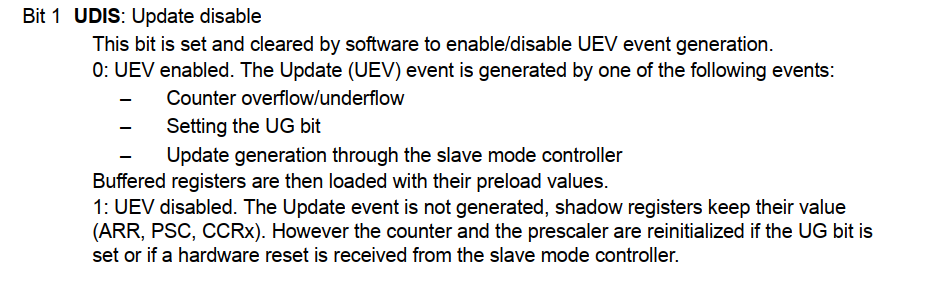
Select auto reload value: tim2->arr

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%

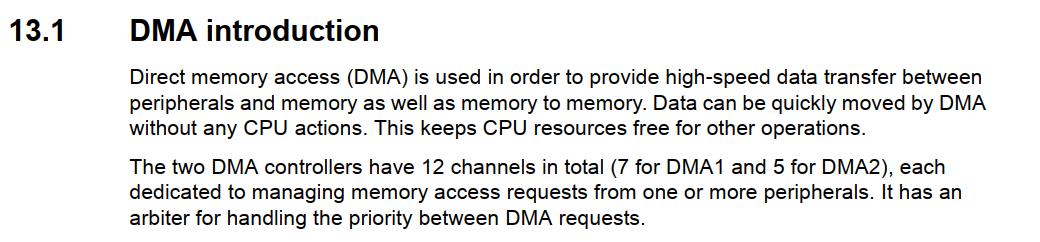
CC1P = 0: OC1 active high : 10% high và 90% low  
CC1P = 1: OC1 active low : 10% low và 90% high

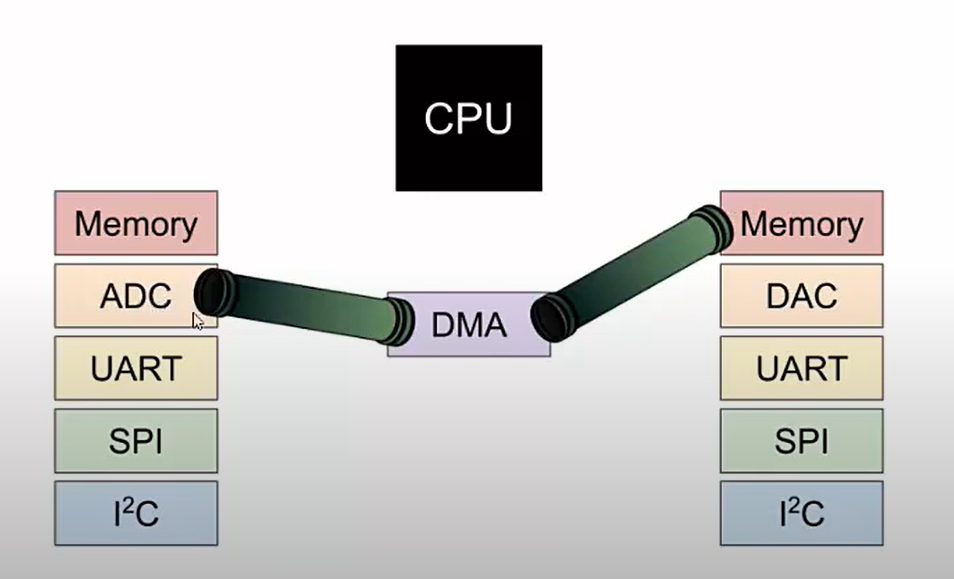
**Bài 36 : DMA**

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

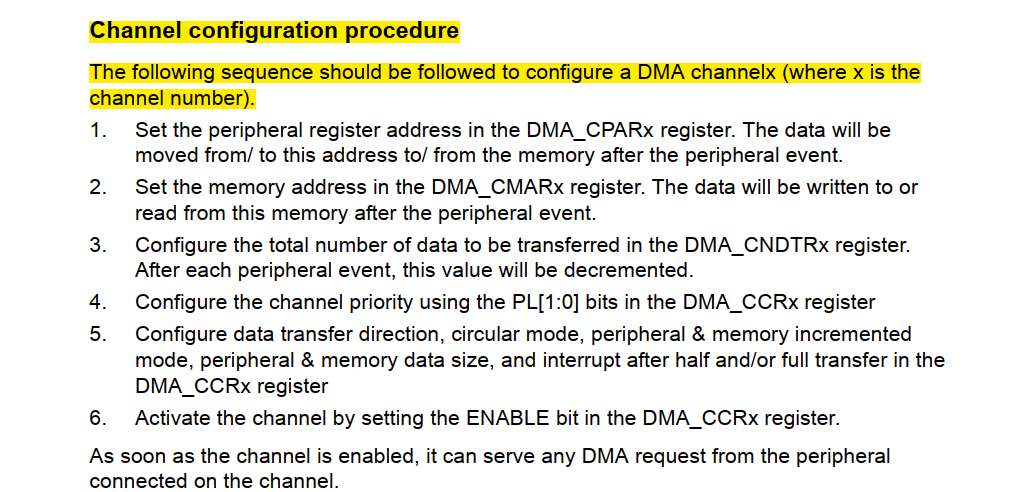
DMA : Direct memory access

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



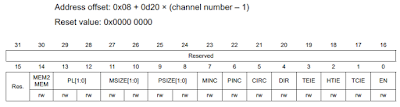


\*\* Các bước làm DMA:



**II. Thanh ghi quan trọng :**

1. 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.

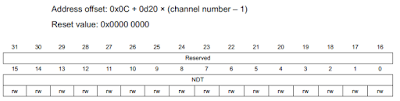
Số bit của data bộ nhớ

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

Số bit của data ngoại vi

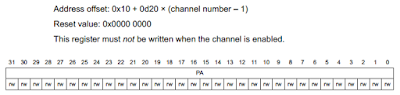
* + MINC: có cho phép mode tăng địa chỉ bộ nhớ hay không.
  + PINC: có cho phép mode tăng địa chỉ ngoại vi hay không.
  + CIRC: có cho phép việc chuyển đổi được diễn ra liên tục hay không.
  + TEIE: cho phép ngắt khi có lỗi trong quá trình truyền hay không.
  + HTIE: cho phép ngắt khi truyền xong data 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.

1. 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)

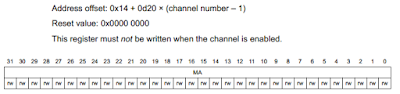
##### Thanh ghi này có giá trị là 16 bit tương ứng với 65535, chứa số lượng data truyền.

1. 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.

1. 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,…, uart, i2c,..
* 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.

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

**1 số thuật ngữ:**

* Trigger : kích hoạt