1. Nêu khái niệm Embedded system (ES là gì)

Hệ thống nhúng là 1 hệ thống máy tính kết hợp các thiết bị điện tử hoặc hệ thống điện tử hoặc cơ khí nhằm thực hiện 1 task hoặc công việc có chức năng nhất định

Đi kèm với câu hỏi này thường là

Phần mềm nhúng là gì

Firmware là gì

2. Phân biệt kiến trúc Von NM và Harvard

Truyền thông trong này ko có chuyện 1 dây hay 2 dây mà nó là đường bus, ở đây thường sẽ có câu hỏi bus dữ liệu, bus địa chỉ là gì

Data bus và address bus trên PC của bạn đang là bao nhiêu, có band witdh (độ rộng) là bao nhiêu.

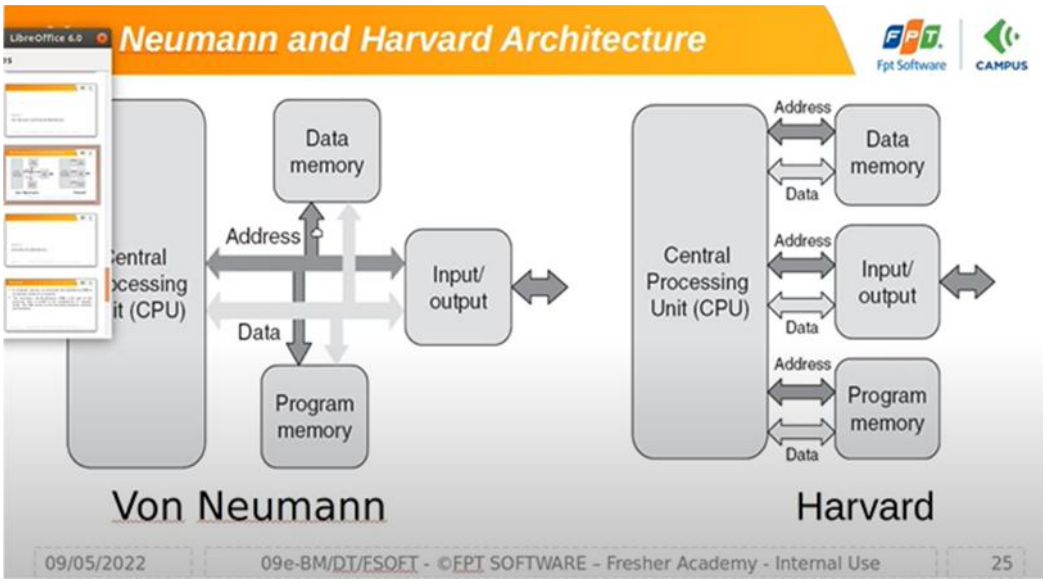
Bản chất trong 1 cái máy tính có 2 loại bus data và địa chỉ.

Bus địa chỉ dùng để giải mã dữ liệu giao tiếp với bộ nhớ từ CPU.

Bus data khi giải mã xong địa chỉ, đọc lại data từ memory theo đường data đấy lên.

Trong 2 kiến trúc này sẽ khác nhau 1 tý. 1 cái dùng chung đường địa chỉ để giải mã cả bộ nhớ chương trình, bộ nhớ dữ liệu, (program memory và data memory) trên cùng 1 đường bus địa chỉ.

1 loại nữa là trên đường bus dữ liệu từ bộ nhớ, hay còn gọi là đường bus của bộ nhớ chương trình riêng.



Data bus và address bus chứ ko phải là **1 DÂY HAY 2 DÂY.**

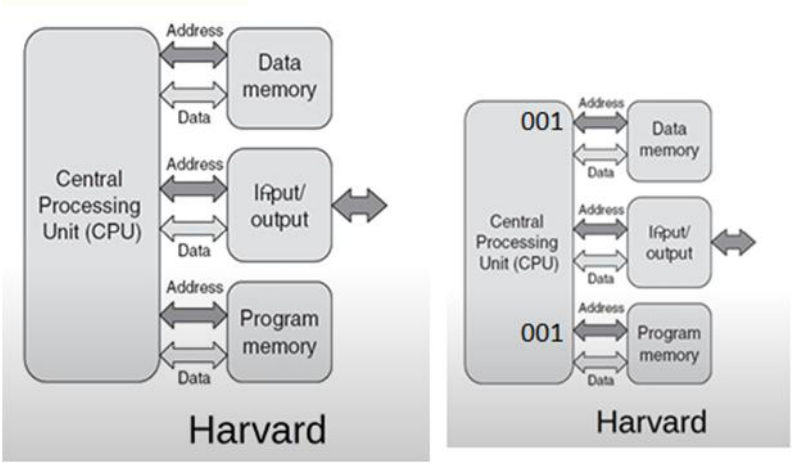
Giả sử muốn đọc 1 vùng nhớ ở data memory thì đẩy 1 địa chỉ lên address này cộng thêm lệnh đọc ghi (enable cái đọc ghi) thì lúc đấy cái data tại ô nhớ đó sẽ đẩy lên trên đường bus này và CPU sẽ đọc/ check được đường bus này và data đang trên đường bus song song nó như thế nào.

Đường bus này có thể là 16 bit/ 8 bit/ 32 bit hay 64 bit nó phụ thuộc vào hệ thống kiến trúc của mình đang là kiến trúc 64 bit hay 32 bit.

Để ý data memory và program memory là 2 thằng sử dụng chung trên 1 đường nối với nhau cho nên là cả 2 thằng này nằm trên cùng 1 dải địa chỉ.

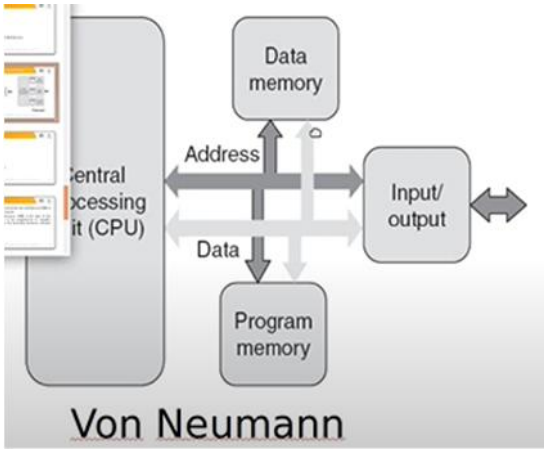
Ví dụ sau này học data memory (SRAM/DRAM) nó sẽ có cùng 1 dải địa chỉ cùng với FLASH Memory

Ví dụ con này (chỉ vào program memory) địa chỉ từ 0 đến 10 thì data memory có địa chỉ từ 11 đến 20. Khi tất cả các thiết bị input, output, peripheral trên cùng 1 memory map hết

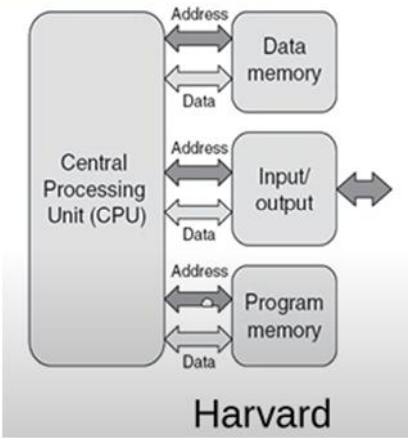


Với thằng kiến trúc harvard này thì data memory riêng, program memory riêng (bus địa chỉ và bus dữ liệu riêng) nên suy ra có thể đẩy 001 vào đường này, đọc được địa chỉ của data memory, nếu đẩy 001 vào program memory thì nó lại đọc trên program memory <u>dù 2 địa chỉ cùng là 001 nhưng đưa vào 2 đường bus address khác nhau</u>

➔ Đọc giá trị khác nhau → vừa đọc chương trình ở 1 nơi, vừa đọc data ở 1 nơi bởi vì 2 địa chỉ này khác nhau

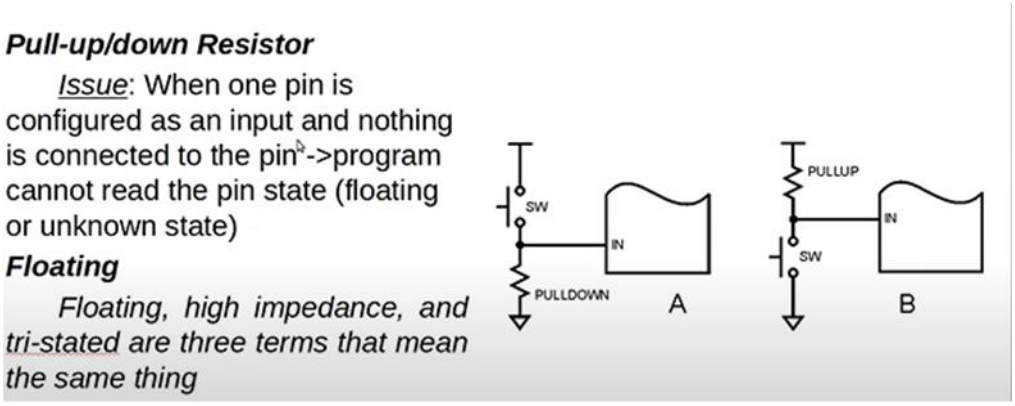


Còn thằng này lúc đọc chương trình, tại 1 thời điểm chỉ đọc được 1 địa chỉ thôi,



Còn thằng này tại 1 thời điểm đọc được nhiều địa chỉ bởi vì đang có nhiều bus thì thằng này tốc độ sẽ nhanh hơn nhưng kiến trúc và tập lệnh phức tạp hơn.
Có thể tính toán và xử lý trên trực tiếp các memory ko cần load and store.
Đường bus ko phải dây nối
3. getting started GPIO
3.1. Nêu các thuộc tính có thể cấu hình được của 1 pin bởi module port (có nghĩa là nêu các thuộc tính của 1 pin)
1 pin có thể chọn nhiều chức năng
Set hướng (input/output) (để ý hỏi là module port có các tính năng nào) → set hướng ko trong module port mà nằm trong module GPIO
Set up pull up hay pull down
Set up các tính năng slow rate (độ dốc sườn xung khi change trạng thái) từ thấp đến cao, từ cao xuống thấp (để xung vuông thì rất có hại).
Tăng cường độ dòng điện của 1 pin thông qua biến drive strength.
Lọc được tín hiệu input (filter).
Set 1 pin là 1 nguồn ngắt gồm có: set ngắt sườn lên sườn xuống, check cờ ngắt trong module port. Mỗi field là 1 feature (là 1 ngắt).
…

3.2 nêu tác dụng khi set up 1 trở treo (pull up)
Khi sử dụng các trở treo, bản thân của nó là xác định các mức logic khi ko có tín hiệu đầu vào của pin, khi nó được set là input
Cơ bản tất cả các pin của GPIO sẽ được set ở mode sử dụng đầu ra MOSFET là cực máng hở - open drain, là khi chúng ta set nó là input thì ngay tại 1 thời điểm thì ko thể xác định nó là mức điện áp dương hay âm. Nếu ko xác định được như vậy thì sẽ đưa pin của mình nối với nguồn hoặc xuống đất thông qua 1 con trở, con trở này có nhiệm vụ tiêu tốn mức năng lượng mà khi mức input được xác định



Khi chân input ko có tín hiệu đầu vào, con trở này ko nối với gnd thì khi ko tác động/ ấn thì input này ko xác định nó là mức 0 hay mức 1 bởi vì chân input được xác định ở mode tín hiệu người ta gọi là được thiết kế ở con mosfet gọi là cực máng hở hay tiếng anh gọi là open drain. Khi nối ntn thì cta sẽ thấy rằng nó sẽ ko biết được trạng thái của nó là mức 0 hay mức 1 nếu ko sử dụng 1 tín hiệu xác định cho nó cho nên ngta sẽ nối trực tiếp nó xuống đất hoặc nối với nguồn thông qua 1 con trở.
Con trở này khi ko có tín hiệu input thì nó sẽ xác định được giá trị input ở trên, nó sẽ là giá trị điện áp trên con trở là mức 0 hoặc nếu cta dùng pull up là mức 1.
Tác dụng thứ 2 là khi có tín hiệu đầu vào sẽ tránh trường hợp ngắn mạch mức tín hiệu dương đến mức tín hiệu âm mà nó vẫn tiêu tốn trên con trở này.
Tác dụng của con trở dùng để xác định giá trị đầu vào khi mà tín hiệu input ko xác định thì con trở này xác định input này là up hay là down (bằng 1 hay 0 tuỳ vào con trở này).

4. Nêu các bước cấu hình cho 1 nút bấm led

Cấu hình cho module sim → cấu hình module port → cấu hình chân module port.

Mục đích cấu hình module sim là **enable clock gate** hay open clock để có thể write được cho 1 module, ở đây là module port.

Module port thì set mux (enable gpio), pull up pull down

Anh K trả lời

Nếu cấu hình cho input thì phải nói là pull up hay pull down

Còn ouput thì ko cần

Ngoài ra còn?

Cấu hình hướng cho gpio là in/out put

Đọc trạng thái:

Nếu là nút nhấn (input) đọc trạng thái của chân

Output thì có thể ghi lên trạng thái hoặc toggle trạng thái của output.

Còn tín hiệu clock hay xung clock thì tần số của nó là bus clock (module clock thì tín hiệu này open ra thì sẽ là module của bus clock/ đặt freq … để về sau rồi nói)

5. giải thích hoạt động của từ khoá volatile trên góc độ của CPU/memory/ complier

Tại sao trong ngắt và memory mapped IO cần dùng volatile

Tại sao trong memory mapped IO cần dùng từ khoá volatile

Về góc độ compler, báo cho compler biết là tắt hoặc đưa mức optimize của complier cho biến đó (khi biên dịch các đoạn code đó) là về mức (level) thấp nhất, complier sẽ mặc định update cái đó.

Với góc độ memory, liên quan đến thanh ghi, các task liên quan đến thanh ghi: đọc, viết (confirmed)

Về tìm hiểu thêm

Mục đích của volatile là tránh các biến thay đổi bất thường.

Chuyển sang bài core M0+ và interrupt

1. Quy trình thực hiện 1 ngắt

Nêu tổ chức bảng vector ngắt hay vector table là gì

2 câu này tự tìm hiểu, tự đọc vì quá dễ, stacking/ unstacking ntn rồi core nó ra làm sau, .. liên quan đến việc optimization việc ngắt nữa

Bảng vector ngắt nằm ở trên, đầu vùng nhớ Flash bắt đầu từ địa chỉ 0x00000000

2. có thể thay đổi bảng vector ngắt sang 1 vùng nhớ khác được ko, có thể lập trình để thay đổi địa chỉ chứa bảng vector ngắt này.

3.Mục đích của việc di chuyển là này ? bảng vector ngắt nằm trên vùng nhớ Flash mà mình ko thay đổi được mà mình có thể đổi địa chỉ chứa bảng vector ngắt để mình có thể lập trình lại bảng đó, cho vector ngắt trỏ tới 1 vùng nhớ khác dễ dàng hơn.

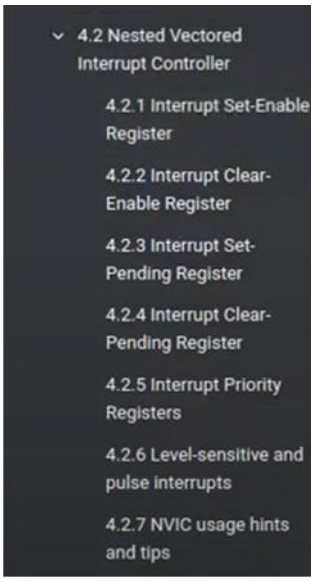
Tìm hiểu kỹ lại cho rõ hơn.

4. nếu chức năng của module NVIC

Là 1 khối dùng để quản lý ngắt trên 1 vi điều khiển của arm. Khi có 1 sự kiện ngắt xảy ra thì đưa vào NVIC rồi dựa vào mức độ ưu tiên để xem thử cái nào thực hiện ngắt trước nếu có mức độ ưu tiên mà nó thấp mà thực hiện ngắt đó

Nếu ko enable thì NVIC ko thực hiện ngắt đó ko?

Trong module NVIC thì có các thanh ghi và feature này



set interrupt enable reg

Khi 1 thanh ghi được set 1 bit ở trên đây lên thì chỉ số ngắt của ngắt mà có chỉ số tương ứng với bit trên này thì sẽ được enable lên

Nếu ko thì interrupt thì sẽ là disable

This section describes the *Nested Vectored Interrupt Controller* (NVIC) and the registers it uses. The NVIC supports:

- 0 to up to 32 interrupts.
- A programmable priority level of 0-192 in steps of 64 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest programmable interrupt priority.
- Level and pulse detection of interrupt signals.
- Interrupt tail-chaining.
- An external *Non-Maskable Interrupt* (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

Module NVIC dùng để set 1 ngắt trong thanh ghi interrput set enable register

Address	Name	Type	Reset value	Description
0xE000E100	NVIC_ISER	RW	0x00000000	*Interrupt Set-Enable Register*.
0xE000E180	NVIC_ICER	RW	0x00000000	*Interrupt Clear-Enable Register* on page 4-4.
0xE000E200	NVIC_ISPR	RW	0x00000000	*Interrupt Set-Pending Register* on page 4-4.
0xE000E280	NVIC_ICPR	RW	0x00000000	*Interrupt Clear-Pending Register* on page 4-4.
0xE000E400-0xE000E4EF	NVIC_IPR0-7	RW	0x00000000	*Interrupt Priority Registers* on page 4-5.

Thứ 2 là clear enable đi bằng cách write 1 bit tương ứng với chỉ số ngắt interrupt clear enable register để disable ngắt đi

Tiếp theo là set pending cho 1 ngắt, thứ 3 là clear pending cho 1 ngắt. ngoài ra có tất cả 8 thanh ghi dùng để set priority cho 32 ngắt.

The NVIC_ISER enables interrupts, and shows which interrupts are enabled. See the register summary in Table 4-2 for the register attributes.

The bit assignments are:

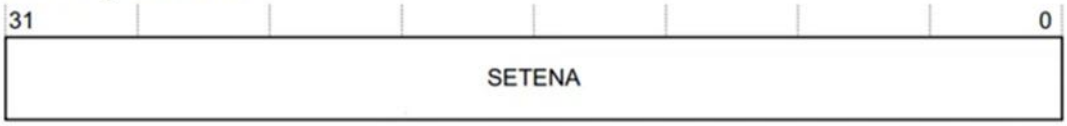


Table 4-3 NVIC_ISER bit assignments

Bits	Name	Function
[31:0]	SETENA	Interrupt set-enable bits. Write: 0 = no effect. 1 = enable interrupt. Read: 0 = interrupt disabled. 1 = interrupt enabled.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

0 là no effect mà 1 là enable

4.2.3 Interrupt Set-Pending Register

The NVIC_ISPR forces interrupts into the pending state, and shows which interrupts are pending. See the register summary in Table 4-2 on page 4-3 for the register attributes.

The bit assignments are:

Table 4-5 NVIC_ISPR bit assignments

Bits	Name	Function
[31:0]	SETPEND	Interrupt set-pending bits. Write: 0 = no effect. 1 = changes interrupt state to pending. Read: 0 = interrupt is not pending. 1 = interrupt is pending.

Note

Writing 1 to the NVIC_ISPR bit corresponding to:

- An interrupt that is pending has no effect.
- A disabled interrupt sets the state of that interrupt to pending.

The bit assignments are:

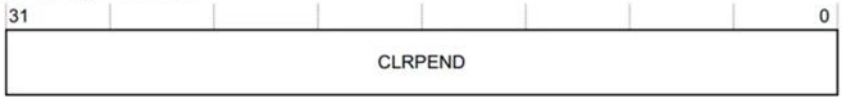


Table 4-6 NVIC_ICPR bit assignments

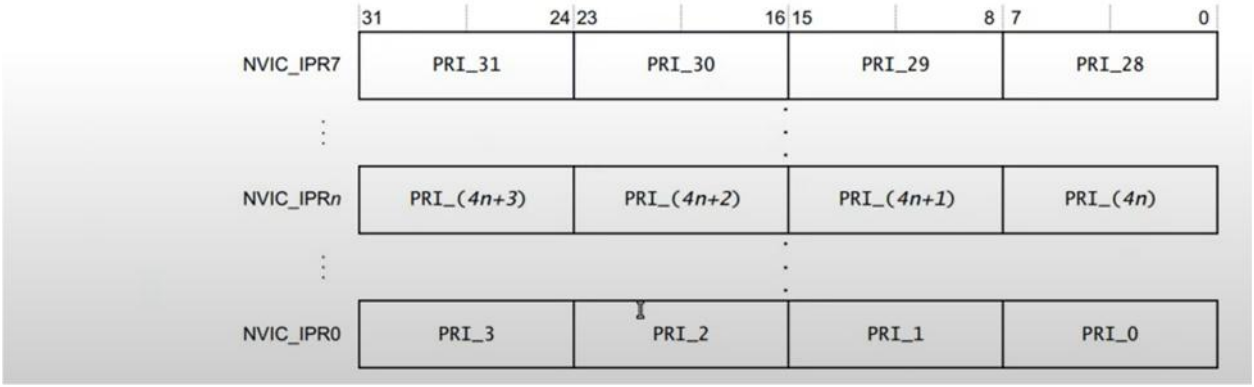
Bits	Name	Function
[31:0]	CLRPEND	Interrupt clear-pending bits. Write: 0 = no effect. 1 = removes pending state from interrupt. Read: 0 = interrupt is not pending. 1 = interrupt is pending.

Note

Writing 1 to an NVIC_ICPR bit does not affect the active state of the corresponding interrupt.

4.2.5 Interrupt Priority Registers

The NVIC_IPR0-NVIC_IPR7 registers provide an 8-bit priority field for each interrupt. These registers are only word-accessible. See the register summary in Table 4-2 on page 4-3 for their attributes. Each register holds four priority fields as shown:



Các mức ưu tiên thì chúng ta dùng 8 bit để set up mức priority nhưng giả sử nó chỉ có 2 bit thôi, 2 bit này là 2 bit 6 và 7, tức là chỉ sử dụng bit 6 và 7 thôi

Bits	Name	Function
[31:24]	Priority, byte offset 3	Each priority field holds a priority value, 0-192. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:6] of each field, bits [5:0] read as zero and ignore writes. This means writing 255 to a priority register saves value 192 to the register.
[23:16]	Priority, byte offset 2	
[15:8]	Priority, byte offset 1	
[7:0]	Priority, byte offset 0	

See *NVIC usage hints and tips* on page 4-7 for more information about the access to the interrupt

Còn bit 0 đến 5 là để read như là zero cho nên các mức ưu tiên của nó sẽ là, nếu bit 6 7 để 00 thì nó sẽ là mức priority là 0. Giả sử là 0 1 thì có mức ưu tiên là 64, 10 là 128 và 11 là 192. Chức năng của vector ngắt là làm các việc đó

Find the NVIC_IPR number and byte offset for interrupt *M* as follows:

- the corresponding NVIC_IPR number, *N*, is given by *N* = *N* DIV 4
- the byte offset of the required Priority field in this register is *M* MOD 4, where:
  - — Byte offset 0 refers to register bits[7:0].
  - — Byte offset 1 refers to register bits[15:8].
  - — Byte offset 2 refers to register bits[23:16].
  - — Byte offset 3 refers to register bits[31:24].

Muốn sử dụng các ngắt đó thì mình phải set mức độ ưu tiên trước. nếu ko set mức ưu tiên thì giá trị mặc định của nó là 00

Thực chất trong module core đã có 1 function giúp chúng ta có thể write/ thay đổi mức ưu tiên này rồi, chỉ cần truyền chỉ số ngắt vào mức ưu tiên để nó có thể thay đổi. nhưng

cần phải hiểu module NVIC nó làm việc thay đổi đó thông qua các thanh ghi kia. Đó chính là các chức năng của module NVIC.

5. nêu các bước cấu hình để sử dụng system tick và nêu ý nghĩa các giá trị mà mình muốn cấu hình

Module systick thì đầu tiên phải cấu hình ở trên/ để xung trong thanh ghi mcg và các bộ chia clock của nó để đưa ra internal clock của nó là 32k Hz hay là 4 Mhz (internal) và thạch anh ngoài là 8Mhz hay cấu hình trong các thanh ghi SIM ví dụ như cấu hình các thanh ghi outdiv1, outdiv 4 cho core hoặc bus,.. còn nhiều cái khác nữa

Tiếp theo cấu hình các thanh ghi systick và reload, current value, control trạng thái và để set systick mình muốn chọn, có sử dụng interrupt hay ko, các giá trị mà để set đầu tiên như là reload với là current.

Các giá trị reload và các giá trị current đó sẽ được tính như thế nào, phụ thuộc vào yếu tố nào. Giá trị reload là giá trị mà mình muốn 1 chu kỳ, nó sẽ đếm từ giá trị reload đếm xuống 0, cứ đếm 1 lượt thì xảy ra 1 ngắt ở trong core để ..

Cứ nhiều ngắt như vậy sẽ tính ra delay của nó dựa vào chu kỳ thạch anh và config.

Thời gian mình muốn sẽ bằng thanh ghi reload đó + 1 nhân với chu kỳ thạch anh 1/f là 1/8Mhz hoặc là 20 hay 24 .. (ko quan trọng)

ở đây chỉ cần biết là systick được sử dụng để đếm, đây là 1 bộ đếm/ timer systick, bản chất là 1 bộ đếm lùi và clock mà nó đếm thì chính là core clock. Như vậy sẽ xác định được core clock thông qua các module set up clock và lúc này khi đếm thì sẽ đếm giá trị từ thanh ghi current về giá trị 0.

Khi về giá trị 0 thì ngta gọi là bộ timer expire hay bộ đếm expire. Việc giá trị này thì cta có thể xác định bộ đếm là, system tick này cta có thể xác định được là đếm theo kiểu oneshot (đếm 1 lần xong dừng lại) hay là set up preodic (đếm tuần hoàn), sau mỗi lần expire thì giá trị mới lại được nạp từ giá trị thanh ghi reload vào.

Và các giá trị này sẽ phụ thuộc vào tần số clock và thời gian mình xác định cho bộ timer expire

6. liệt kê và nêu các hoạt động, chức năg của các thanh ghi general purposal trong core m0+. Nêu các chức năng của các thanh ghi general purposal là gì, mixture purposal là gì,

7. nêu hoạt động của các thanh ghi khi hàm mẹ gọi hàm con rồi hàm con return về hàm mẹ. ví dụ giả sử các thanh ghi này có gì, thanh ghi general có chức năng gì, LR sẽ làm gì, CPU counter sẽ làm gì, mainstack pointer ntn

8. phân biệt chương trình chuyển sang ngắt, chương trình ngắt rồi chương trình chính khác nhau so với chương trình mẹ gọi hàm con ntn hay đơn giản câu hỏi có nghĩa là chuyển từ thread mode trong thread model trong core sang interrupt model trong core ntn

Nhớ đọc lại thanh ghi, check lại trường hợp đọc của các thanh ghi khi mà sang các ct con pvu ngắt hay các câu lệnh LR đó.

Cần bổ sung câu trả lời cho các câu hỏi