Count flag chỉ dùng để read only, để thể hiện counter được đếm về 0 hay chưa kể từ thời điểm mình read only lần cuối cùng (tại thời điểm cuối cùng); như 1 cờ để báo counter (reference value) đã được đếm về 0.

Bits	TYPE	Name	Function
[31:17]	-	-	Reserved.
[16]	RO	COUNTFLAG	Indicates whether the counter has counted to 0 since the last read of this register: **0** timer has not counted to 0. **1** timer has counted to 0. COUNTFLAG is set to 1 by a count transition from 1 to 0. COUNTFLAG is cleared to 0 by a read of this register, and by any write to the Current Value register.

Cờ = 1 thì count sẽ từ 0 về 1 (set cờ lên)

Còn clear về 0 bằng cách read thanh ghi này và viết bất cứ thứ gì vào current value reg này. Và nó sẽ clear (như đọc ở trên là write vào thanh ghi này thì nó sẽ xoá flag này luôn).

[2]	RW	CLKSOURCE	Indicates the SysTick clock source: **0** SysTick uses the optional external reference clock. **1** SysTick uses the processor clock. If no external clock is provided, this bit reads as one and ignores writes.

Clock source, ở đây có bit clock source để chọn nguồn clock, nếu bằng 0 thì systick sử dụng 1 optional khác ở bên ngoài. Còn bằng 1 thì systick sử dụng processor clock luôn, là nó sử dụng clock của bộ vi xử lý

If no external clock is provided, thì bit này nó bằng 1 và bỏ qua việc write.

Bit này thường sẽ được

Figure B3-9 shows the SYST_CSR bit assignments.

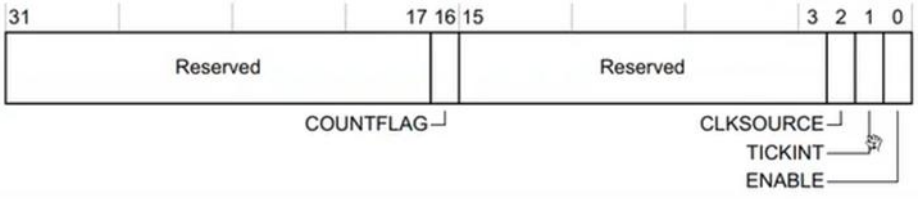
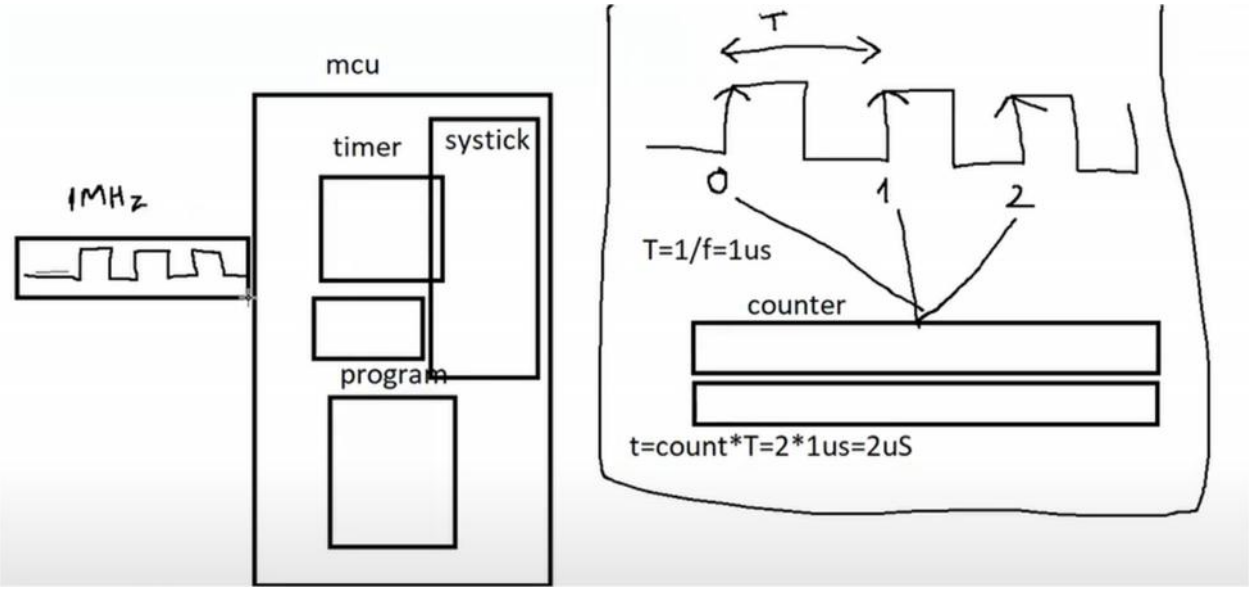


Figure B3-9 SYST_CSR bit assignments

Có giá trị bằng 1, ở đây có giá trị khi reset

0xE000E010	SYST_CSR	RW	0x00000000 or 0x00000004	*SysTick Control and Status Register, SYST_CSR* on page B3-239

Có 2 giá trị 000 và 004, khi reset có giá trị là 04 có nghĩa là bit thứ 3 bằng 1 (100B = 4D) thì systick sử dụng 1 processor clock



Thì có nghĩa là việc này nó chọn clock ở bên ngoài (1Mhz)

[1]	RW	TICKINT	Indicates whether counting to 0 causes the status of the SysTick exception to change to pending: **0** count to 0 does not affect the SysTick exception status. **1** count to 0 changes the SysTick exception status to pending. Changing the value of the counter to 0 by writing zero to the SysTick Current Value register to 0 never changes the status of the SysTick exception.

TICKINT, cái này được dùng để enable interrupt

Khi nó bằng 0 có nghĩa là khi đếm về 0 thì nó sẽ ko ảnh hưởng đến systick exception status

Còn bằng 1 thì khi đếm về 0 nó sẽ thay đổi exception status sang trạng thái chờ, và nó được xem là cái interrupt.

[0]	RW	ENABLE	Indicates the enabled status of the SysTick counter: **0** counter is disabled. **1** counter is operating.

Cái cuối cùng là enable để mình enable cái systick

Đó là thanh ghi đầu tiên

Giờ là thanh ghi tiếp theo RVR

B3.3.4 SysTick Reload Value Register, SYST_RVR

The SYST_RVR Register characteristics are:

Purpose	Sets or reads the reload value of the SYST_CVR register.
Usage constraints	There are no usage constraints.
Configurations	The register is only present if the optional system timer is implemented, otherwise the register is reserved.
Attributes	See Table B3-15.

Figure B3-10 shows the SYST_RVR bit assignments.

Figure B3-10 SYST_RVR bit assignments

Có 24 bit value từ 0 đến 23, giá trị max là 2^24 = 16tr.

V ghi vào thanh ghi reload này, reload value là giá trị để load vào trong value khi counter đọc về 0

Table B3-15 SYST_RVR bit assignments

Bits	Name	Function
[31:24]	-	Reserved. RAZ/WI.
[23:0]	RELOAD	The value to load into the SYST_CVR register when the counter reaches 0.

; nó sẽ lấy giá trị trong thanh ghi này ốp vào thanh ghi current value

SysTick Current Value Register, SYST_CVR

The SYST_CVR Register characteristics are:

Purpose	Reads or clears the current counter value.
Usage constraints	Any write to the register clears the register to 0. The counter does not provide read-modify-write protection. Unsupported bits are Read-As-Zero. See *SysTick Reload Value Register, SYST_RVR*.
Configurations	The register is only present if the optional system timer is implemented, otherwise the register is reserved.
Attributes	See Table B3-16 on page B3-241.

Figure B3-11 shows the SYST_CVR bit assignments.

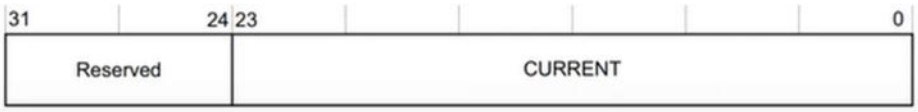


Figure B3-11 SYST_CVR bit assignments

Thì thanh ghi này với việc bất kỳ việc ghi cái gì vào thanh ghi này thì nó sẽ clear thanh ghi này về 0. Counter sẽ ko cung cấp …

Table B3-16 SYST_CVR bit assignments

Bits	Name	Function
[31:24]	-	Reserved. RAZ/WI.
[23:0]	CURRENT	Current counter value. This is the value of the counter at the time it is sampled.

Current value là 1 giá trị của counter tại một thời điểm là mình lấy mẫu

Figure B3-12 SYST_CALIB Register bit assignments

Thanh ghi hiệu chỉnh, với giá trị TENMS của nó thường là 1 thanh ghi read only thôi

mục đích của thanh ghi này là nó hiệu chỉnh để cho output của nó

Purpose	Reads the calibration value and parameters for SysTick.
Usage constraints	There are no usage constraints.
Configurations	The register is only present if the optional system timer is implemented, otherwise the register is reserved.
Attributes	See Table B3-17.

Để cho interrupt sinh ra 1 cái chu kỳ là đúng 10ms thì nó sẽ tự động hiệu chỉnh với điều kiện là mình đang sử dụng với clock là giá trị maximum cho phép.

Ví dụ clock đó cho phép sử dụng tới 48Mhz thì cần config ntn đó để clock đầu vào của nó là 48Mhz thì cái calibration này mới phát huy được tác dụng, nó mới ra được đúng thời gian là 10ms

Table B3-17 SYST_CALIB Register bit assignments

Bits	Name	Function
[31]	NOREF	Indicates whether the IMPLEMENTATION DEFINED reference clock is provided: **0** the reference clock is implemented. **1** the reference clock is not implemented. When this bit is 1, the CLKSOURCE bit of the SYST_CSR register is forced to 1 and cannot be cleared to 0.
[30]	SKEW	Indicates whether the 10ms calibration value is exact: **0** 10ms calibration value is exact. **1** 10ms calibration value is inexact, because of the clock frequency.
[29:24]	-	Reserved.
[23:0]	TENMS	Optionally, holds a reload value to be used for 10ms (100Hz) timing, subject to system clock skew errors. If this field is zero, the calibration value is not known.

Bit NOREF = 0 hay 1 để cho reference clock được hay ko được implemented.

Khi bit này bằng 1 thì … và ko thể clear về 0.

Nó sẽ phụ thuộc vào thanh ghi SYS_CSR; reference clock ko được implemented thì bắt buộc CLOCKSOURCE trong thanh ghi CSR này nó phải bằng 1 và ko thể clear về 0 được.

[2]	RW	CLKSOURCE	Indicates the SysTick clock source: **0** SysTick uses the optional external reference clock. **1** SysTick uses the processor clock. If no external clock is provided, this bit reads as one and ignores writes.

Bên cạnh đó cũng thấy, khi bit này bằng 1 (nếu ko có external clock được provided thì bit này bằng 1 và bỏ qua việc viết)

[30]	SKEW	Indicates whether the 10ms calibration value is exact: **0** 10ms calibration value is exact. **1** 10ms calibration value is inexact, because of the clock frequency.

Skew là để chỉ ra cái 10ms này có đúng hay ko

[23:0]	TENMS	Optionally, holds a reload value to be used for 10ms (100Hz) timing, subject to system clock skew errors. If this field is zero, the calibration value is not known.

Field cuối cùng này tương tự như thanh ghi reload value này nhưng mà nó tự động ghi vào chứ mình ko cần tính toán để write vào đây

Figure B3-12 SYST_CALIB Register bit assignments

Mà nó tự write vào 10ms, tự tính toán để ghi vào thanh ghi read load value này

Figure B3-10 shows the SYST_RVR bit assignments.

Figure B3-10 SYST_RVR bit assignments

RM của con chip nào cũng có bảng ntn

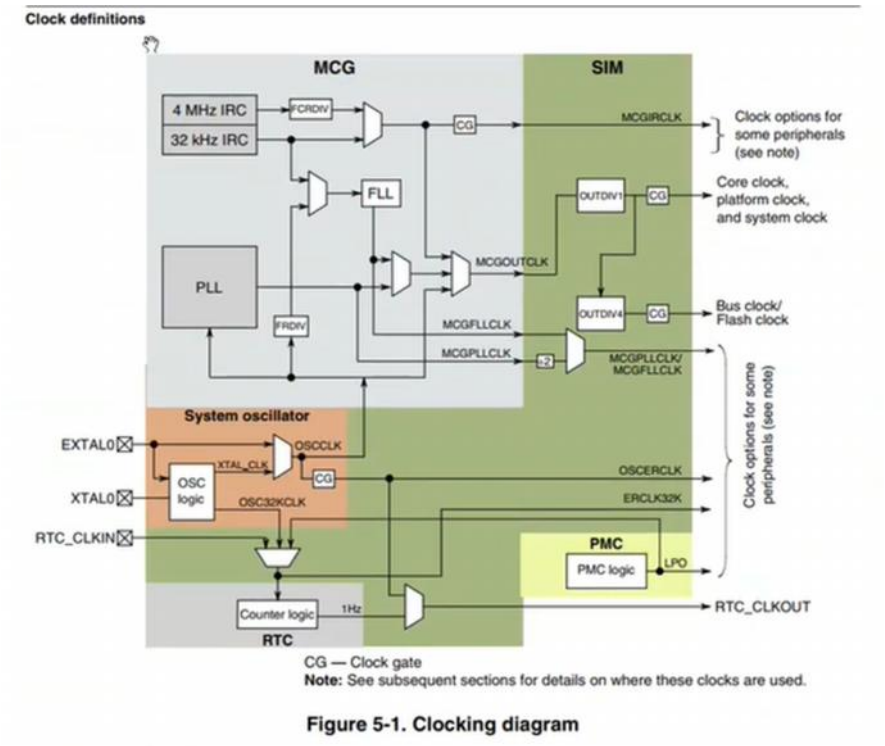
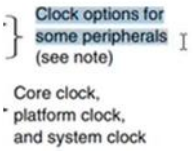


Figure 5-1. Clocking diagram

Đây là bảng phân phối clock, đường đi, các bộ nhân bộ chia, các bộ chọn clock thì mình thông qua diagram này để mình xác định được cái systick của mình nó sẽ lấy clock nào, hiện tại đang lấy clock nào và mình muốn điều chỉnh nó vào clock khác để thực hiện những cái gì

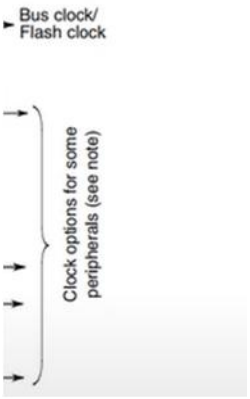
Trong khối này có SIM dùng để chọn clock, cũng có các bộ chia và 1 khối ở đây là MCG và khối system oscillator có đầu vào nối với thạch anh ngoài, hay thạch anh RTC.

Trước tiên cần biết systick của mình đang dùng clock nào thì mới điều chỉnh cho tần số của clock đó được. thường thì nó sẽ dùng clock của core



còn ở đây sẽ có 1 số option clock cho cái perip.

ở dưới thì có



Còn các clock cho perip này thì xem ở module clock

5.7 Module clocks

The following table summarizes the clocks associated with each module.

Table 5-2. Module clocks

Module	Bus interface clock	Internal clocks	I/O interface clocks
Core modules			
ARM Cortex-M0+ core	Platform clock	Core clock	—
NVIC	Platform clock	—	—
DAP	Platform clock	—	SWD_CLK
System modules			
DMA	System clock	—	—
DMA Mux	Bus clock	—	—
Port control	Bus clock	—	—
Crossbar Switch	Platform clock	—	—
Peripheral bridges	System clock	Bus clock	—
LLWU, PMC, SIM, RCM	Bus clock	LPO	—
Mode controller	Bus clock	—	—
MCM	Platform clock	—	—
Watchdog timer	Bus clock	LPO	—
Clocks			
MCG	Bus clock	MCGOUTCLK, MCGPLLCLK, MCGFLLCLK, MCGIRCLK, OSCERCLK	—
OSC	Bus clock	OSCERCLK	—

Table continues on the next page...

Nó sẽ có clock cho thằng COP thì

5.7.2 COP clocking

The COP may be clocked from two clock sources as shown in the following figure.

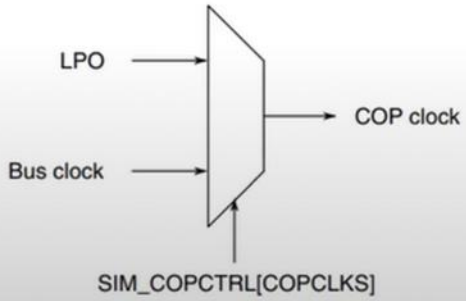


Figure 5-2. COP clock generation

Thì nó sẽ sử dụng clock gì, nó có thể dùng 2 clock LPO hoặc Busclock thông qua thanh ghi SIM_COPCTRL[COPCLKS] để chọn 1 trong 2 clock trên

5.7.3 RTC clocking

The RTC module can be clocked as shown in the following figure.

NOTE

The chosen clock must remain enabled if the RTC is to continue operating in all required low-power modes.

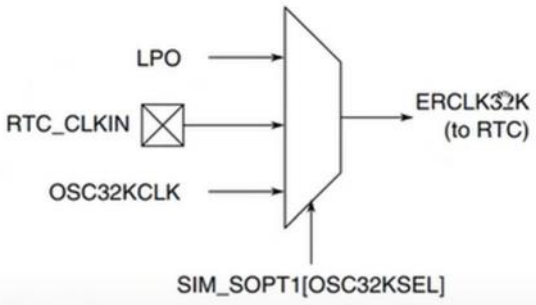


Figure 5-3. RTC clock generation

Mình có thể chọn 1 trong 3 input thông qua thanh ghi SIM_SOPT1[ÓC32SEL] tương tự cho các ngoại vi khác

5.7.9 UART clocking

The UART0 module has a selectable clock as shown in the following figure. UART1 and UART2 modules operate from the bus clock.

NOTE

The chosen clock must remain enabled if the UART0 is to continue operating in all required low-power modes.

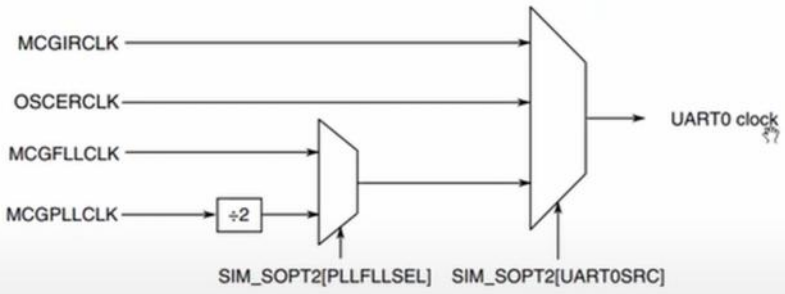


Figure 5-7. UART0 clock generation

Để chọn được clock cho UART thì cần chọn 2 cái bit này SIM_SOPT2[PLLFLLSEL] và SIM_SOPT2[UARTSRC].

Đây là clock cho ngoại vi, NHƯNG systick ko phải ngoại vi mà nó nằm trong core cho nên nó sẽ lấy clock của core

5.4 Clock definitions

The following table describes the clocks in the previous block diagram.

Clock name	Description
Core clock	MCGOUTCLK divided by OUTDIV1 Clocks the ARM Cortex-M0+ core.
Platform clock	MCGOUTCLK divided by OUTDIV1 Clocks the crossbar switch and NVIC.
System clock	MCGOUTCLK divided by OUTDIV1 Clocks the bus masters directly .

Table continues on the next page...

Clock name	Description
Bus clock	System clock divided by OUTDIV4. Clocks the bus slaves and peripherals.
Flash clock	Flash memory clock On this device, it is the same as Bus clock.
MCGIRCLK	MCG output of the slow or fast internal reference clock
MCGOUTCLK	MCG output of either IRC, MCGFLLCLK, MCGPLLCLK, or MCG's external reference clock that sources the core.

ở dưới này cũng có definitons cái clock name là gì và mô tả của nó. Thì core clock là clock cho arm coretex M0+ thì systick đang sử dụng cái này. ở đây cũng có mô tả các clock name

5.4.1 Device clock summary

The following table provides more information regarding the on-chip clocks.

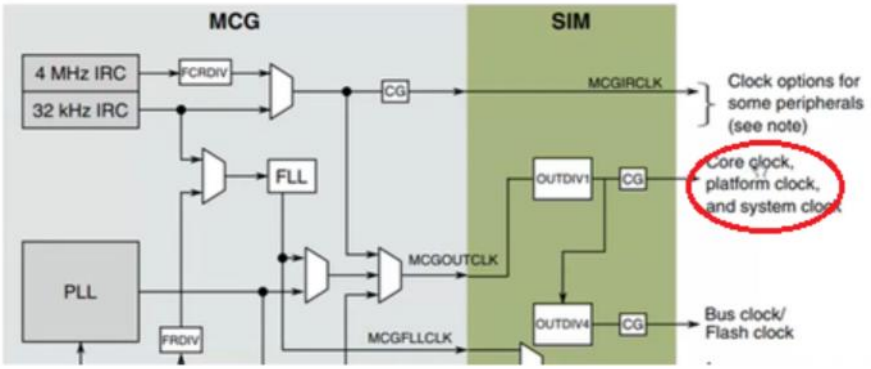
Table 5-1. Clock summary

Clock name	Run mode clock frequency	VLPR mode clock frequency	Clock source	Clock is disabled when...
MCGOUTCLK	Up to 100 MHz	Up to 4 MHz	MCG	In all stop modes except for partial stop modes and during PLL locking when MCGOUTCLK derived from PLL.
MCGFLLCLK	Up to 48 MHz	N/A	MCG	MCG clock controls are not enabled and in all stop modes

Table continues on the next page

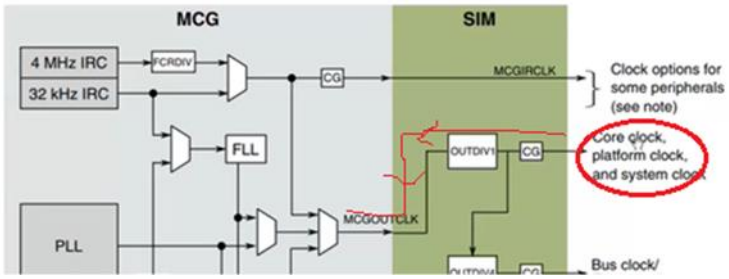
ở đây thì nó sẽ trình bày ra khi ở chế độ run mode thì clock này có thể được chọn maximum là bao nhiêu, nó cho phép max là bao nhiêu, tần số là bao nhiêu, còn ở chế độ VLPR là lowpower là tiết kiệm năng lượng thì tần số sẽ thấp hơn.

Clock source là nguồn clock từ đâu, thì quay lại bảng này để phân tích



Sys tick sẽ lấy từ core clock thì mình sẽ dò ngược lại

Chỗ này outdiv 1 có nghĩa là coi như nó ko chia



Đến đây dò tiếp MCGOUTCLK để xem nó chọn clock source cho clock nào (1 trong 3 đầu vào này)

24.3.1 MCG Control 1 Register (MCG_C1)

Address: 4006_4000h base + 0h offset = 4006_4000h

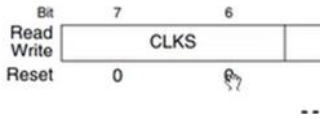
Bit	7	6	5	4	3	2	1	0
Read / Write	CLKS		FRDIV			IREFS	IRCLKEN	IREFSTEN
Reset	0	0	0	0	0	1	0	0

MCG_C1 field descriptions

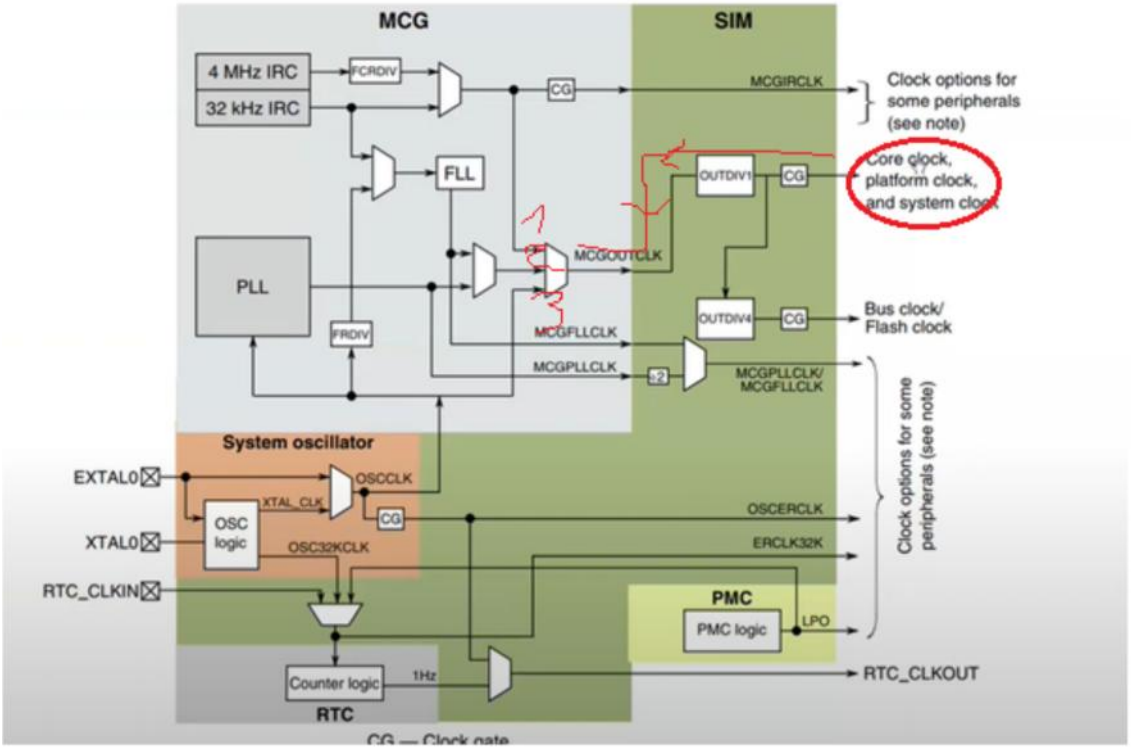
Field	Description
7–6 CLKS	Clock Source Select Selects the clock source for **MCGOUTCLK** . 00 Encoding 0 — Output of FLL or PLL is selected (depends on PLLS control bit). 01 Encoding 1 — Internal reference clock is selected. 10 Encoding 2 — External reference clock is selected. 11 Encoding 3 — Reserved.
5–3 FRDIV	FLL External Reference Divider

Table continues on the next page

Thì nó có 2 bit là bit 7 và bit 6. Bây giờ mình giả sử khi mình reset con chip thì mình sẽ xem giá trị khi mà nó reset là bao nhiêu để mình biết được là khi mình reset con chip thì nó chọn clock nào.



ở đây giá trị khi reset là 00 cho nên nó sẽ lấy output của fll hoặc pll



Nếu chọn được 3 thì nó sẽ đi theo 0sc clock như vậy là external reference clock

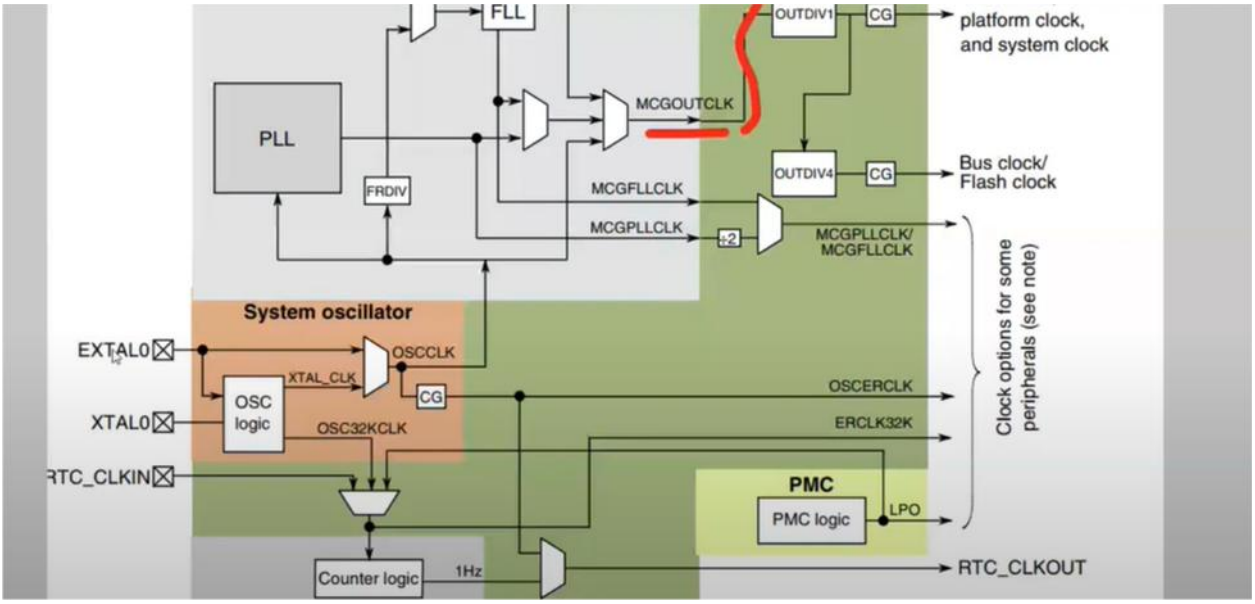
24.3.1 MCG Control 1 Register (MCG_C1)

Address: 4006_4000h base + 0h offset = 4006_4000h

Bit	7	6	5	4	3	2	1	0
Read / Write	CLKS		FRDIV			IREFS	IRCLKEN	IREFSTEN
Reset	0	0	0	0	0	1	0	0

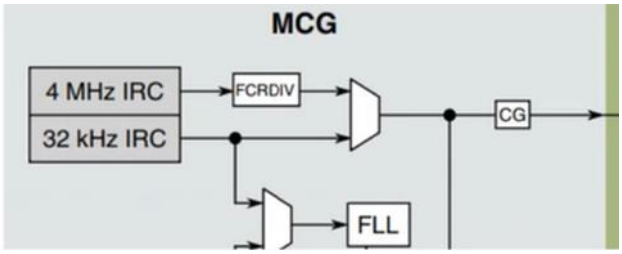
MCG_C1 field descriptions

Field	Description
7–6 CLKS	Clock Source Select Selects the clock source for MCGOUTCLK . 00 Encoding 0 — Output of FLL or PLL is selected (depends on PLLS control bit). 01 Encoding 1 — Internal reference clock is selected. 10 Encoding 2 — External reference clock is selected. 11 Encoding 3 — Reserved.
5–3 FRDIV	FLL External Reference Divider

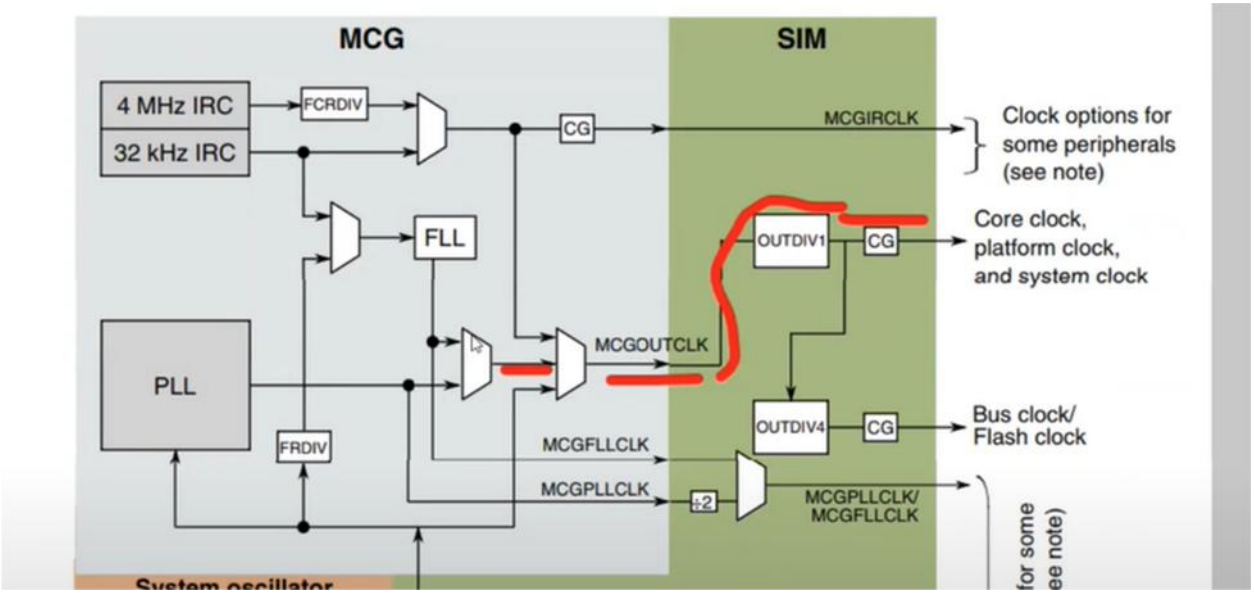


Từ EXTAL0 đi vào.

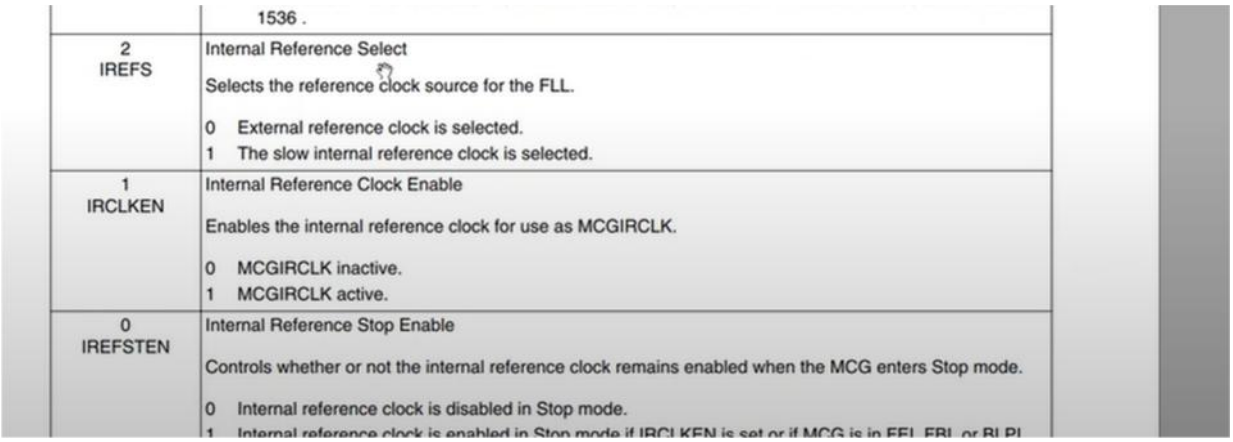
Nếu bằng 1 là internal clock



Nếu bằng 0 thì nó sẽ lấy giá trị output của fll hoặc pll để nó

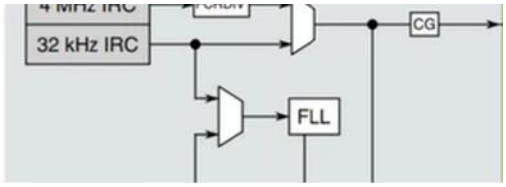


Chính là đường này, nó sẽ lấy hay dưới phụ thuộc vào 1 thanh ghi khác nữa để chọn xem là fll hay pll

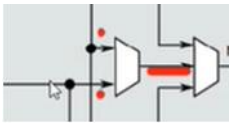


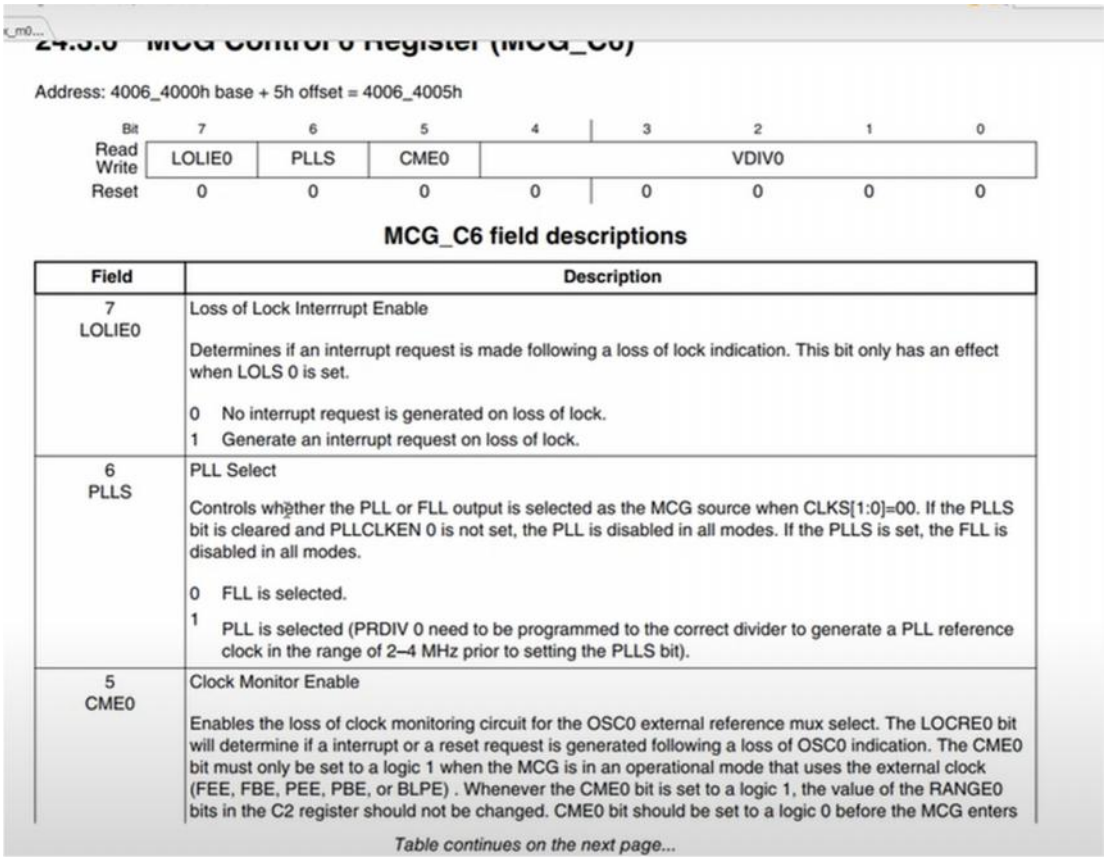
	1536 .
2 IREFS	Internal Reference Select Selects the reference clock source for the FLL. 0 External reference clock is selected. 1 The slow internal reference clock is selected.
1 IRCLKEN	Internal Reference Clock Enable Enables the internal reference clock for use as MCGIRCLK. 0 MCGIRCLK inactive. 1 MCGIRCLK active.
0 IREFSTEN	Internal Reference Stop Enable Controls whether or not the internal reference clock remains enabled when the MCG enters Stop mode. 0 Internal reference clock is disabled in Stop mode. 1 Internal reference clock is enabled in Stop mode if IRCLKEN is set or if MCG is in FEI, FBI, or BLPI

Thấy internal reference select, nó sẽ chọn clock source cho fll là thanh ghi này



Còn chỗ này để chọn fll hay pll





Address: 4006_4000h base + 5h offset = 4006_4005h

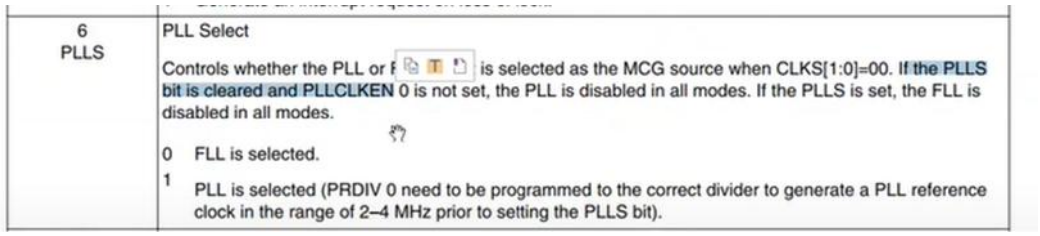
Bit	7	6	5	4	3	2	1	0
Read / Write	LOLIE0	PLLS	CME0	VDIV0				
Reset	0	0	0	0	0	0	0	0

MCG_C6 field descriptions

Field	Description
7 LOLIE0	Loss of Lock Interrrupt Enable Determines if an interrupt request is made following a loss of lock indication. This bit only has an effect when LOLS 0 is set. 0 No interrupt request is generated on loss of lock. 1 Generate an interrupt request on loss of lock.
6 PLLS	PLL Select Controls whether the PLL or FLL output is selected as the MCG source when CLKS[1:0]=00. If the PLLS bit is cleared and PLLCLKEN 0 is not set, the PLL is disabled in all modes. If the PLLS is set, the FLL is disabled in all modes. 0 FLL is selected. 1 PLL is selected (PRDIV 0 need to be programmed to the correct divider to generate a PLL reference clock in the range of 2–4 MHz prior to setting the PLLS bit).
5 CME0	Clock Monitor Enable Enables the loss of clock monitoring circuit for the OSC0 external reference mux select. The LOCRE0 bit will determine if a interrupt or a reset request is generated following a loss of OSC0 indication. The CME0 bit must only be set to a logic 1 when the MCG is in an operational mode that uses the external clock (FEE, FBE, PEE, PBE, or BLPE) . Whenever the CME0 bit is set to a logic 1, the value of the RANGE0 bits in the C2 register should not be changed. CME0 bit should be set to a logic 0 before the MCG enters

Table continues on the next page...

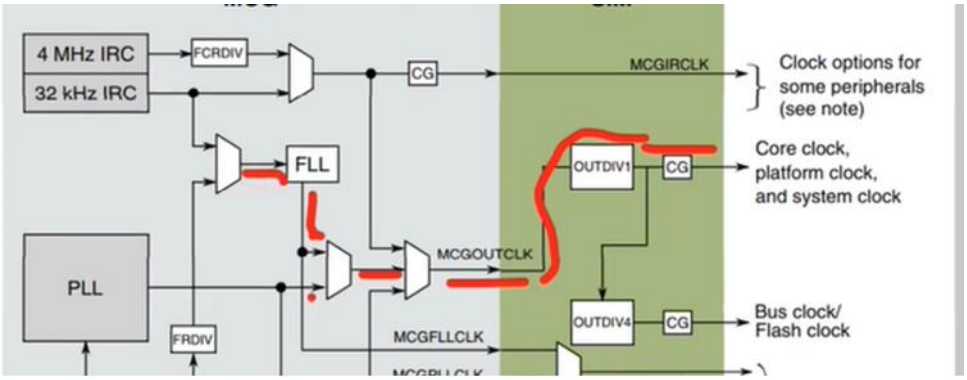
Bit PLLS thì nó sẽ: điều khiển PLL hoặc FLL output nó được chọn như là 1 MCG source



6 PLLS	PLL Select Controls whether the PLL or F is selected as the MCG source when CLKS[1:0]=00. If the PLLS bit is cleared and PLLCLKEN 0 is not set, the PLL is disabled in all modes. If the PLLS is set, the FLL is disabled in all modes. 0 FLL is selected. 1 PLL is selected (PRDIV 0 need to be programmed to the correct divider to generate a PLL reference clock in the range of 2–4 MHz prior to setting the PLLS bit).

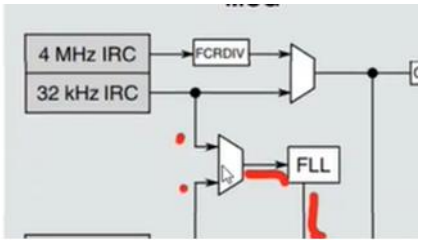
Nếu bit này bằng 0 thì FLL được chọn và bằng 1 thì PLL được chọn. tuy nhiên nếu PLL được chọn thì thêm 1 số điều kiện nữa.

Thì mình cần xem PLLS này đang có giá trị là bao nhiêu (khi reset đang có giá trị bằng 0). Như vậy nó chọn FLL chứ ko chọn PLL



	1536 .
2 IREFS	Internal Reference Select Selects the reference clock source for the FLL. 0 External reference clock is selected. 1 The slow internal reference clock is selected.
1 IRCLKEN	Internal Reference Clock Enable Enables the internal reference clock for use as MCGIRCLK. 0 MCGIRCLK inactive. 1 MCGIRCLK active.
0 IREFSTEN	Internal Reference Stop Enable Controls whether or not the internal reference clock remains enabled when the MCG enters Stop mode. 0 Internal reference clock is disabled in Stop mode. 1 Internal reference clock is enabled in Stop mode if IRCLKEN is set or if MCG is in FEI, FBI, or BLPI

Bit này sẽ chọn clock source cho FLL, bit này sẽ lấy internal clock, 1 nguồn clock nội rất thấp 32kHz hoặc nó lấy từ FRDIV. Bằng 0 thì nó sẽ chọn external thì nó lấy ở dưới, còn nếu bằng 1 thì nó sẽ lấy low internal reference clock là mức ở trên.

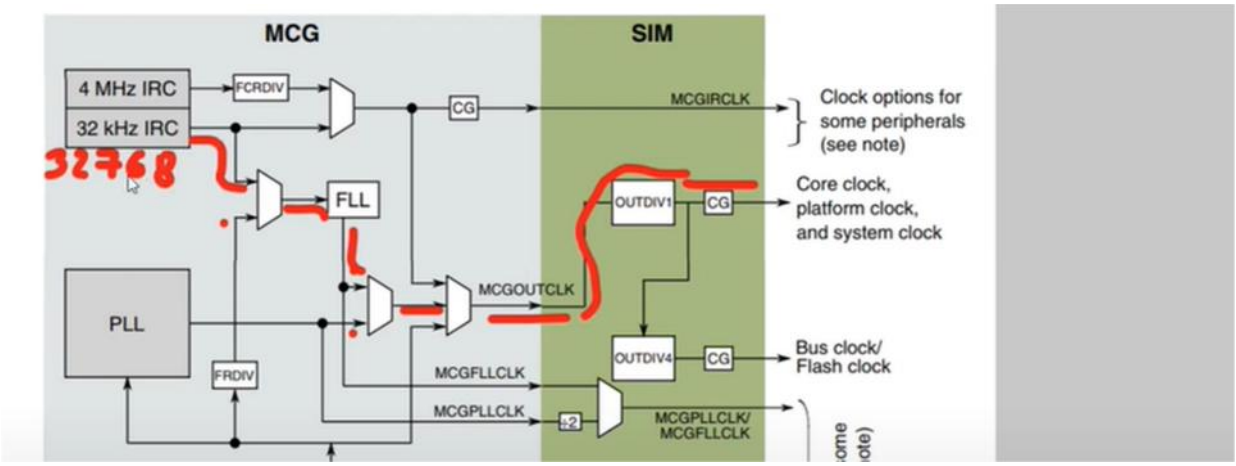


Như vậy giá trị khi reset là 1

Address: 4006_4000h base + 0h offset = 4006_4000h

Bit	7	6	5	4	3	2	1	0
Read Write	CLKS		FRDIV			IREFS	IRCLKEN	IREFSTEN
Reset	0	0	0	0	0	1	0	0

Nhìn vào bit thứ 2 IREFS thì nó sử dụng slow …



Như vậy core clock của mình ở đây là ? Bộ FLL ở đây là bộ nhân giá trị (32768Hz) với FLL Factor để ra giá trị DCO Range

Field	Description

The following table identifies settings for the DCO frequency range.

NOTE: The system clocks derived from this source should not exceed their specified maximums.

DRST_DRS	DMX32	Reference Range	FLL Factor	DCO Range
00	0	31.25–39.0625 kHz	640	20–25 MHz
	1	32.768 kHz	732	24 MHz
01	0	31.25–39.0625 kHz	1280	40–50 MHz
	1	32.768 kHz	1464	48 MHz
10	0	31.25–39.0625 kHz	1920	60–75 MHz
	1	32.768 kHz	2197	72 MHz
11	0	31.25–39.0625 kHz	2560	80–100 MHz
	1	32.768 kHz	2929	96 MHz

0 DCO has a default range of 25%.
1 DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

Vậy tần số qua FLL cao hơn gấp x lần. Bộ nhân này được config qua thanh ghi MCG_C4

24.3.4 MCG Control 4 Register (MCG_C4)

NOTE

Reset values for DRST and DMX32 bits are 0.

Address: 4006_4000h base + 3h offset = 4006_4003h

Bit	7	6	5	4	3	2	1	0
Read Write	DMX32	DRST_DRS		FCTRIM				SCFTRIM
Reset	0	0	0	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.
- A value for FCTRIM is loaded during reset from a factory programmed location. x = Undefined at reset.

MCG_C4 field descriptions

Field	Description
7 DMX32	DCO Maximum Frequency with 32.768 kHz Reference The DMX32 bit controls whether the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference.

Table continues on the next page...

ở đây có 2 cái cần quan tâm là bit DMX32 và field DRST_DRS.

DMX32 control bằng 0 thì có default range là 25 % và bằng 1 là …

MCG_C4 field descriptions (continued)

Field	Description
	The following table identifies settings for the DCO frequency range. **NOTE:** The system clocks derived from this source should not exceed their specified maximums.

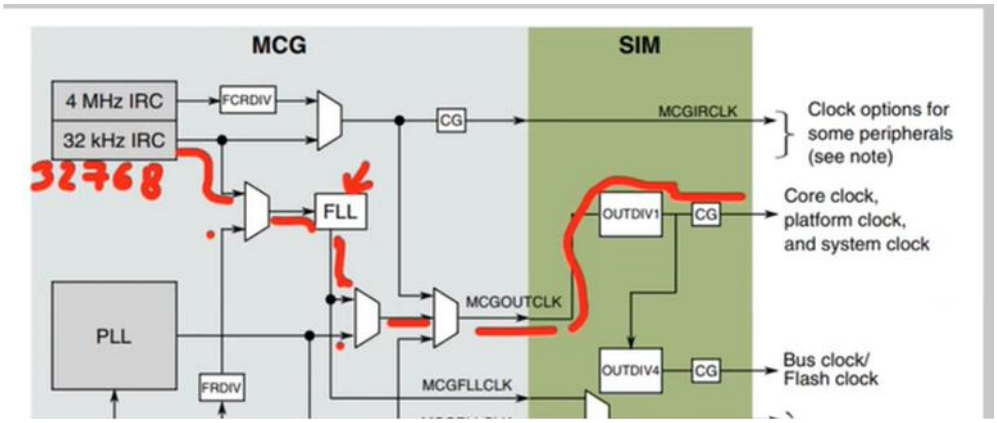
DRST_DRS	DMX32	Reference Range	FLL Factor	DCO Range
00	0	31.25–39.0625 kHz	640	20–25 MHz
	1	32.768 kHz	732	24 MHz
01	0	31.25–39.0625 kHz	1280	40–50 MHz
	1	32.768 kHz	1464	48 MHz
10	0	31.25–39.0625 kHz	1920	60–75 MHz
	1	32.768 kHz	2197	72 MHz
11	0	31.25–39.0625 kHz	2560	80–100 MHz
	1	32.768 kHz	2929	96 MHz

0 DCO has a default range of 25%.
1 DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

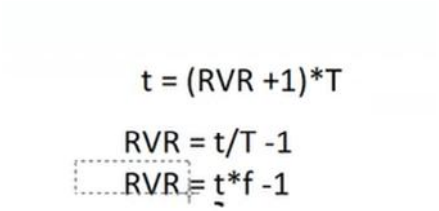
Đưa tần số chỗ này là 32768 Hz luôn.

Khi nó bằng 0 thì nó có khoảng ntn là từ 31.25–39.0625 kHz là vì khi NSX thiết kế thì người ta ko có chính xác được tần số cho nên họ có khoảng ntn để hiệu chỉnh, nó bao nhiêu Hz còn phụ thuộc vào TRIM value nữa để viết ra được TRIM value và từ đó tính ra được tần số hiện tại đang là bao nhiêu, thì ở đây mình có thể mặc định nó là 32768 luôn. Nhưng khi reset nó bằng 0, vậy tần số qua FLL là 32768x640 =

20.9 Mhz



vậy khi không config gì cả thì tần số mặc định là 20.9 Mhz



Áp dụng vào công thức này để tính ra được chu kỳ interrupt, giá trị RVR, mình sẽ ghi vào RVR thì ra được chu kỳ interrupt theo thời gian mong muốn.

ở kinetic thì hỗ trợ vài default tần số, còn việc config tần số thì nó tự tính toán ra miễn ra được tần số theo yêu cầu là được

ở kinematic thì nó define sẵn

```
#define MCG_MODE_FEI                0U
#define MCG_MODE_FBI                1U
#define MCG_MODE_BLPI               2U
#define MCG_MODE_FEE                3U
#define MCG_MODE_FBE                4U
#define MCG_MODE_BLPE               5U
#define MCG_MODE_PBE                6U
#define MCG_MODE_PEE                7U

/* Predefined clock setups
   0 ... Default part configuration
         Multipurpose Clock Generator (MCG) in FEI mode.
         Reference clock source for MCG module: Slow internal reference clock
         Core clock = 20.97152MHz
         Bus clock  = 20.97152MHz
   1 ... Maximum achievable clock frequency configuration
         Multipurpose Clock Generator (MCG) in PEE mode.
         Reference clock source for MCG module: System oscillator reference clock
         Core clock = 48MHz
         Bus clock  = 24MHz
   2 ... Chip internally clocked, ready for Very Low Power Run mode
         Multipurpose Clock Generator (MCG) in BLPI mode.
         Reference clock source for MCG module: Fast internal reference clock
         Core clock = 4MHz
         Bus clock  = 0.8MHz
   3 ... Chip externally clocked, ready for Very Low Power Run mode
         Multipurpose Clock Generator (MCG) in BLPE mode.
         Reference clock source for MCG module: System oscillator reference clock
         Core clock = 4MHz
         Bus clock  = 1MHz
   4 ... USB clock setup
         Multipurpose Clock Generator (MCG) in PEE mode.
         Reference clock source for MCG module: System oscillator reference clock
         Core clock = 48MHz
         Bus clock  = 24MHz
*/
```

Còn mình chỉ cần define clock setup

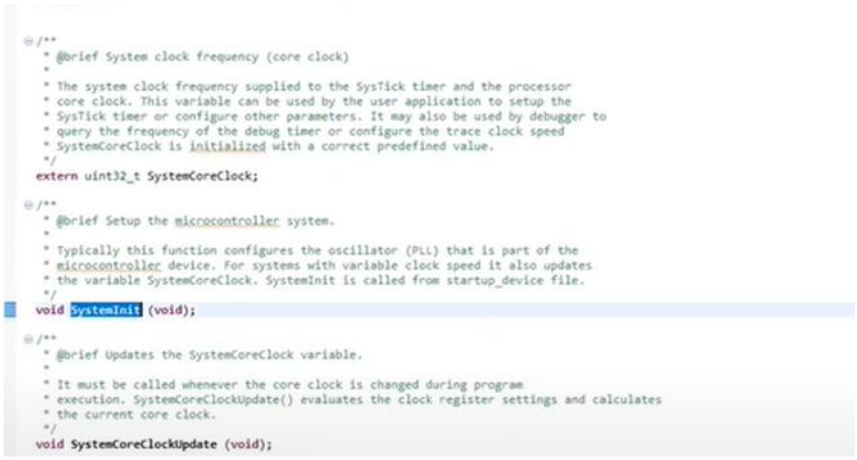
```
/* Define clock source values */

#define CPU_XTAL_CLK_HZ                 8000000U            /* Value of the external crystal or oscillator clock frequency of the system oscillator (OSC) in Hz */
#define CPU_INT_SLOW_CLK_HZ             32768U              /* Value of the slow internal oscillator clock frequency in Hz */
#define CPU_INT_FAST_CLK_HZ             4000000U            /* Value of the fast internal oscillator clock frequency in Hz */

/* RTC oscillator setting */

/* Low power mode enable */
/* SMC_PMPROT: AVLP=1,ALLS=1,AVLLS=1 */
#define SYSTEM_SMC_PMPROT_VALUE         0x2AU               /* SMC_PMPROT */

/* Internal reference clock trim */
/* #undef SLOW_TRIM_ADDRESS */                              /* Slow oscillator not trimmed. Commented out for MISRA compliance. */
/* #undef SLOW_FINE_TRIM_ADDRESS */                         /* Slow oscillator not trimmed. Commented out for MISRA compliance. */
/* #undef FAST_TRIM_ADDRESS */                              /* Fast oscillator not trimmed. Commented out for MISRA compliance. */
/* #undef FAST_FINE_TRIM_ADDRESS */                         /* Fast oscillator not trimmed. Commented out for MISRA compliance. */

#define CLOCK_SETUP 4
```

Bằng 2 thì nó sẽ ra 4Mhz

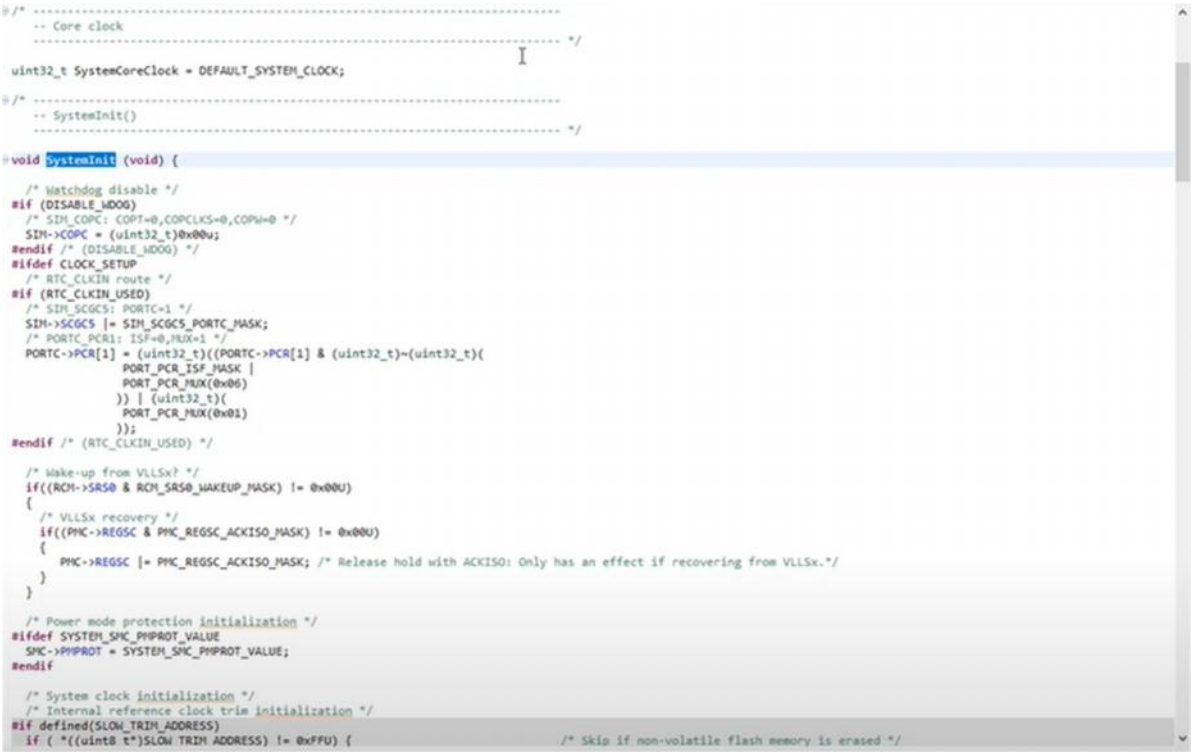
```
#elif (CLOCK_SETUP == 2)
  #define DEFAULT_SYSTEM_CLOCK          4000000U            /* Default System clock value */
  #define MCG_MODE                      MCG_MODE_BLPI /* Clock generator mode */
  /* MCG_C1: CLKS=1,FRDIV=0,IREFS=1,IRCLKEN=1,IREFSTEN=0 */
  #define SYSTEM_MCG_C1_VALUE           0x46U               /* MCG_C1 */
  /* MCG_C2: LOCRE0=0,FCFTRIM=0,RANGE0=2,HGO0=0,EREFS0=1,LP=1,IRCS=1 */
  #define SYSTEM_MCG_C2_VALUE           0x27U               /* MCG_C2 */
  /* MCG_C4: DMX32=0,DRST_DRS=0,FCTRIM=0,SCFTRIM=0 */
  #define SYSTEM_MCG_C4_VALUE           0x00U               /* MCG_C4 */
  /* MCG_SC: ATME=0,ATMS=0,ATMF=0,FLTPRSRV=0,FCRDIV=0,LOCS0=0 */
  #define SYSTEM_MCG_SC_VALUE           0x00U               /* MCG_SC */
  /* MCG_C5: PLLCLKEN0=0,PLLSTEN0=0,PRDIV0=0 */
  #define SYSTEM_MCG_C5_VALUE           0x00U               /* MCG_C5 */
  /* MCG_C6: LOLIE0=0,PLLS=0,CME0=0,VDIV0=0 */
  #define SYSTEM_MCG_C6_VALUE           0x00U               /* MCG_C6 */
  /* OSC0_CR: ERCLKEN=1,EREFSTEN=0,SC2P=0,SC4P=0,SC8P=0,SC16P=0 */
  #define SYSTEM_OSC0_CR_VALUE          0x80U               /* OSC0_CR */
  /* SMC_PMCTRL: RUNM=0,STOPA=0,STOPM=0 */
  #define SYSTEM_SMC_PMCTRL_VALUE       0x00U               /* SMC_PMCTRL */
  /* SIM_CLKDIV1: OUTDIV1=0,OUTDIV4=4 */
  #define SYSTEM_SIM_CLKDIV1_VALUE      0x00040000U         /* SIM_CLKDIV1 */
  /* SIM_SOPT1: USBREGEN=0,USBSSTBY=0,USBVSTBY=0,OSC32KSEL=3 */
  #define SYSTEM_SIM_SOPT1_VALUE        0x000C0000U         /* SIM_SOPT1 */
  /* SIM_SOPT2: UART0SRC=0,TPMSRC=2,USBSRC=0,PLLFLLSEL=0,CLKOUTSEL=0,RTCCLKOUTSEL=0 */
  #define SYSTEM_SIM_SOPT2_VALUE        0x02000000U         /* SIM_SOPT2 */
#elif (CLOCK_SETUP == 3)
  #define DEFAULT_SYSTEM_CLOCK          4000000U            /* Default System clock value */
```

Với core clock bằng 2 thì nó define các giá trị của các thanh ghi MCG để làm sao mà cái core clock output bằng 4Mhz và bus clock là 0.8 Mhz

Và mình chỉ cần gọi hàm SysInit

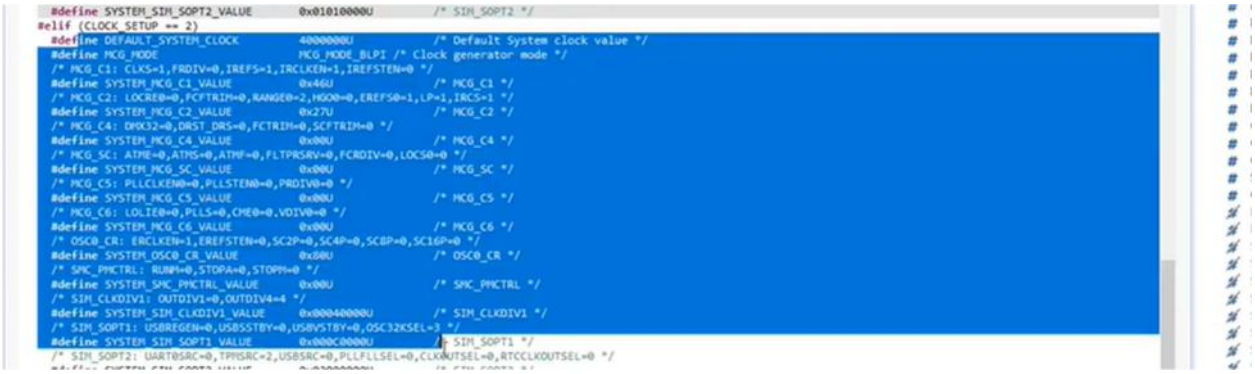


```
/**
 * @brief System clock frequency (core clock)
 *
 * The system clock frequency supplied to the SysTick timer and the processor
 * core clock. This variable can be used by the user application to setup the
 * SysTick timer or configure other parameters. It may also be used by debugger to
 * query the frequency of the debug timer or configure the trace clock speed
 * SystemCoreClock is initialized with a correct predefined value.
 */
extern uint32_t SystemCoreClock;

/**
 * @brief Setup the microcontroller system.
 *
 * Typically this function configures the oscillator (PLL) that is part of the
 * microcontroller device. For systems with variable clock speed it also updates
 * the variable SystemCoreClock. SystemInit is called from startup_device file.
 */
void SystemInit (void);

/**
 * @brief Updates the SystemCoreClock variable.
 *
 * It must be called whenever the core clock is changed during program
 * execution. SystemCoreClockUpdate() evaluates the clock register settings and calculates
 * the current core clock.
 */
void SystemCoreClockUpdate (void);
```



```
/* ----------------------------------------------------------------------------
   -- Core clock
   ---------------------------------------------------------------------------- */

uint32_t SystemCoreClock = DEFAULT_SYSTEM_CLOCK;

/* ----------------------------------------------------------------------------
   -- SystemInit()
   ---------------------------------------------------------------------------- */

void SystemInit (void) {

  /* Watchdog disable */
#if (DISABLE_WDOG)
  /* SIM_COPC: COPT=0,COPCLKS=0,COPW=0 */
  SIM->COPC = (uint32_t)0x00u;
#endif /* (DISABLE_WDOG) */
#ifdef CLOCK_SETUP
  /* RTC_CLKIN route */
#if (RTC_CLKIN_USED)
  /* SIM_SCGC5: PORTC=1 */
  SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;
  /* PORTC_PCR1: ISF=0,MUX=1 */
  PORTC->PCR[1] = (uint32_t)((PORTC->PCR[1] & (uint32_t)~(uint32_t)(
                  PORT_PCR_ISF_MASK |
                  PORT_PCR_MUX(0x06)
                 )) | (uint32_t)(
                  PORT_PCR_MUX(0x01)
                 ));
#endif /* (RTC_CLKIN_USED) */

  /* Wake-up from VLLSx? */
  if((RCM->SRS0 & RCM_SRS0_WAKEUP_MASK) != 0x00U)
  {
    /* VLLSx recovery */
    if((PMC->REGSC & PMC_REGSC_ACKISO_MASK) != 0x00U)
    {
      PMC->REGSC |= PMC_REGSC_ACKISO_MASK; /* Release hold with ACKISO: Only has an effect if recovering from VLLSx.*/
    }
  }

  /* Power mode protection initialization */
#ifdef SYSTEM_SMC_PMPROT_VALUE
  SMC->PMPROT = SYSTEM_SMC_PMPROT_VALUE;
#endif

  /* System clock initialization */
  /* Internal reference clock trim initialization */
#if defined(SLOW_TRIM_ADDRESS)
  if ( *((uint8_t*)SLOW_TRIM_ADDRESS) != 0xFFU) {                /* Skip if non-volatile flash memory is erased */
```

Thì hàm này nó sẽ dựa theo những macro mà mình đã define ở trên

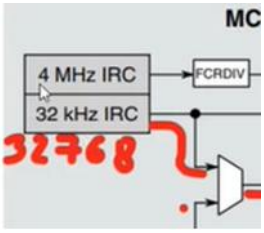


Để nó config

Các thanh ghi để cuối cùng cho ra được tần số là 4Mhz. để config được clock thì control nhiều thanh ghi

```
      case 0x20U:
        MCGOUTClock *= 1280U;
        break;
      case 0x40U:
        MCGOUTClock *= 1920U;
        break;
      case 0x60U:
        MCGOUTClock *= 2560U;
        break;
      case 0x80U:
        MCGOUTClock *= 732U;
        break;
      case 0xA0U:
        MCGOUTClock *= 1464U;
        break;
      case 0xC0U:
        MCGOUTClock *= 2197U;
        break;
      case 0xE0U:
        MCGOUTClock *= 2929U;
        break;
      default:
        break;
    }
  } else { /* (!((MCG->C6 & MCG_C6_PLLS_MASK) == 0x00U)) */
    /* PLL is selected */
    Divider = (((uint16_t)MCG->C5 & MCG_C5_PRDIV0_MASK) + 0x01U);
    MCGOUTClock = (uint32_t)(CPU_XTAL_CLK_HZ / Divider); /* Calculate the PLL reference clock */
    Divider = (((uint16_t)MCG->C6 & MCG_C6_VDIV0_MASK) + 24U);
    MCGOUTClock *= Divider;          /* Calculate the MCG output clock */
  } /* (!((MCG->C6 & MCG_C6_PLLS_MASK) == 0x00U)) */
} else if ((MCG->C1 & MCG_C1_CLKS_MASK) == 0x40U) {
  /* Internal reference clock is selected */
  if ((MCG->C2 & MCG_C2_IRCS_MASK) == 0x00U) {
    MCGOUTClock = CPU_INT_SLOW_CLK_HZ; /* Slow internal reference clock selected */
  } else { /* (!((MCG->C2 & MCG_C2_IRCS_MASK) == 0x00U)) */
    Divider = (uint16_t)(0x01LU << ((MCG->SC & MCG_SC_FCRDIV_MASK) >> MCG_SC_FCRDIV_SHIFT));
    MCGOUTClock = (uint32_t) (CPU_INT_FAST_CLK_HZ / Divider); /* Fast internal reference clock selected */
  } /* (!((MCG->C2 & MCG_C2_IRCS_MASK) == 0x00U)) */
} else if ((MCG->C1 & MCG_C1_CLKS_MASK) == 0x80U) {
  /* External reference clock is selected */
  MCGOUTClock = CPU_XTAL_CLK_HZ;     /* System oscillator drives MCG clock */
} else { /* (!((MCG->C1 & MCG_C1_CLKS_MASK) == 0x80U)) */
  /* Reserved value */
  return;
} /* (!((MCG->C1 & MCG_C1_CLKS_MASK) == 0x80U)) */
SystemCoreClock = (MCGOUTClock / (0x01U + ((SIM->CLKDIV1 & SIM_CLKDIV1_OUTDIV1_MASK) >> SIM_CLKDIV1_OUTDIV1_SHIFT)));
```

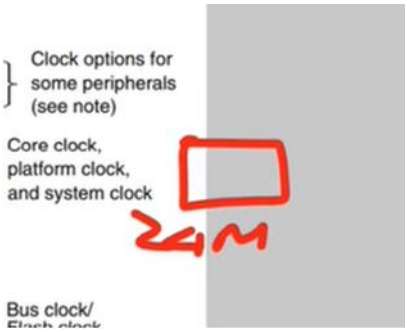
Hàm SysCoreClockUpdate có mục đích là sẽ thay đổi giá trị SysCoreClock tại vì khi mình config bộ chia, mình chỉ biết được tần số ở đây là bao nhiêu



Và mình config những bộ nhân, chia xong rồi nó ra tần số chỗ kia là bao nhiêu, cái này là do mình tự tính ra được thôi

Còn chương trình chỉ biết control các thanh ghi nhân chia này bao nhiêu, chọn nguồn nào còn cuối cùng đáp số là tần số bao nhiêu thì nó ko biết.

Giả sử mình tính ra ở đây là 24Mhz



Làm sao để biết ở đây là đúng 24Mhz, thường thì để verify được đúng 24 Mhz hay ko thì mình code 1 vài ứng dụng để tạo ra 1 xung và mình đo xem nó có đúng xung như mình mong muốn ko, nếu nó bị lệch thì chắc chắn do cái nguồn clock của mình ko đúng. Nguồn clock đang tính toán ko đúng. Mình sẽ lấy 24Mhz để tính toán nhưng thực tế output ra ko phải 24Mhz thì nó cho ra 1 xung ko đúng.

Còn để config như thế này thì có 1 hàm hỗ trợ cho mình giá trị systemcoreclock

Nếu 1 người nào đó config bộ chia bộ nhân thì mình gọi hàm system core clock update thì mình có được giá trị biến tần số của nó ở trong này luôn

```
} /* (!((MCG->C1 & MCG_C1_CLKS_MASK) == 0x00U)) */
SystemCoreClock = (MCGOUTClock / (0x01U + ((SIM->CLKDIV1 & SIM_CLKDIV1_OUTDIV1_MASK) >> SIM_CLKDIV1_OUTDIV1_SHIFT)));
```

, đem biến này đi tính toán.

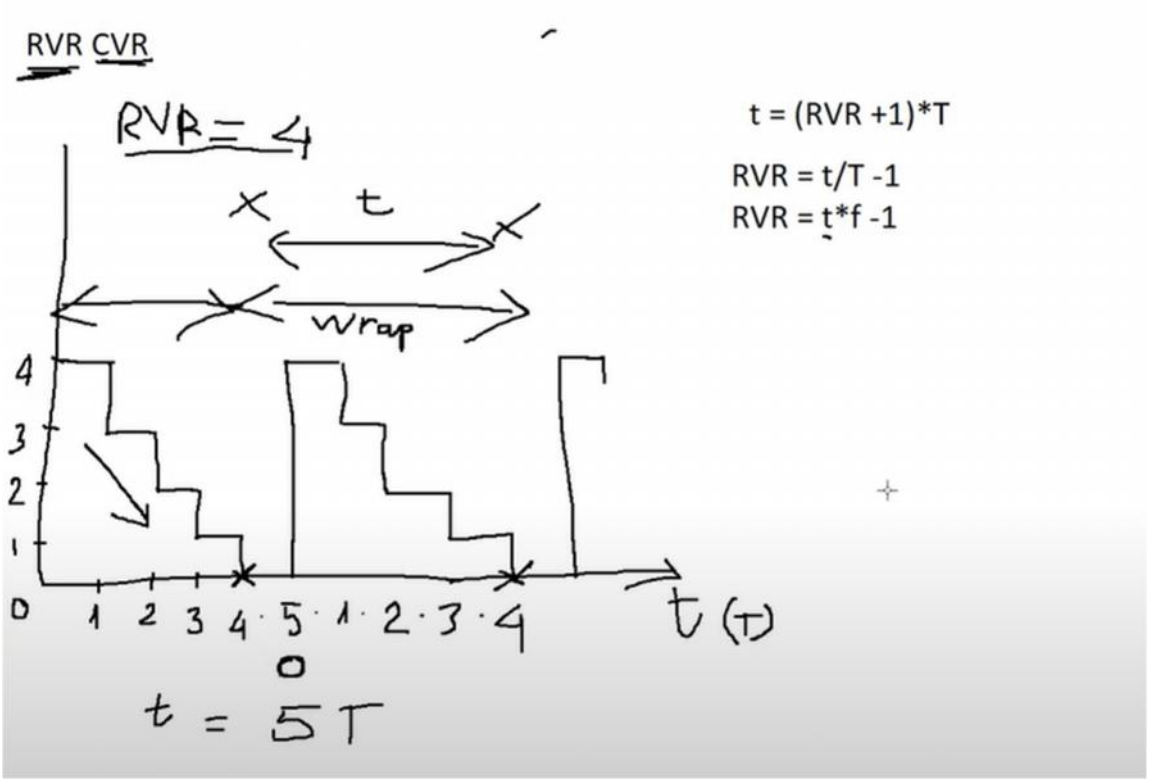
Nó ko sp việc mình thay đổi, config, muốn thay đổi clock thì phải tự code.

Ngoài ra còn có những control mcg khác nhau nữa

Viết 1 example sử dụng default clock (20.9 Mhz) để mình tính toán

```
}
void sys_init()
{
    SysTick->CTRL = (SysTick_CTRL_CLKSOURCE_Msk|SysTick_CTRL_ENABLE_Msk);
}
void delay_ms(uint32_t t_ms)
{
```

Cần set up reload value trước vì nếu set up enable clock source trước thì nó chạy mất rồi.

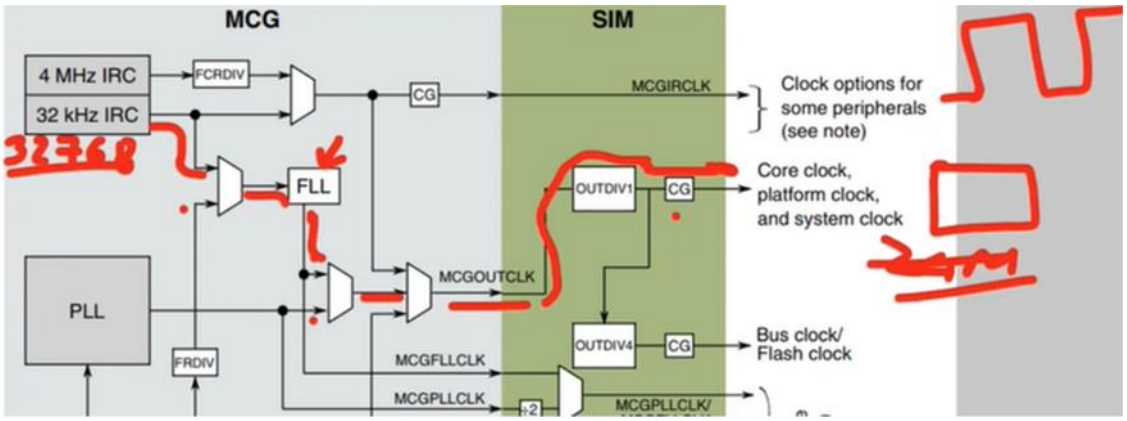


Mình muốn chu kỳ wrap này là 1 ms, mình muốn delay 1ms thôi để cái flag nó set lên

RVR = 1*, mà cái này đang tính theo đơn vị ms nên

RVR = 20971520*1/1000-1=

Cái này là số lẻ nên nó gây ra sai số, mình cần giảm thiểu tần số bằng cách config lại tần số ở đây



Để nó cho ra tần số chẵn thì ít sai số hơn. Thì đây, giá trị của mình là 20970

t = (RVR +1)*T

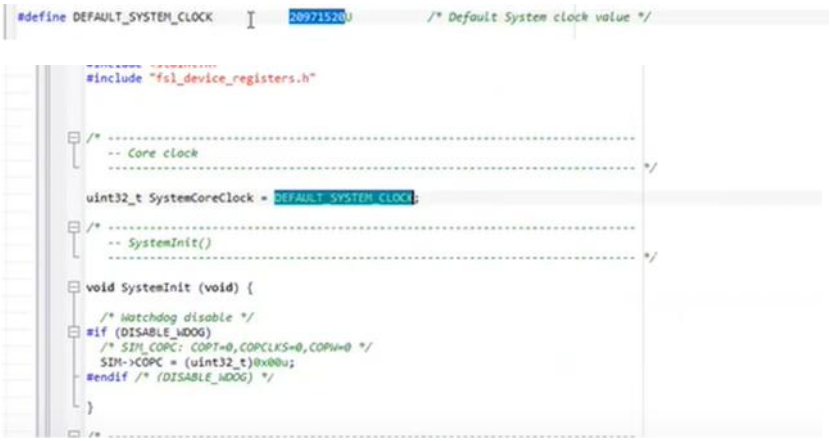
RVR = t/T -1

RVR = t*f -1

RVR = 20971520*1/1000-1=20970

Ghi vào thanh ghi RVR này, vì nó là 24 bit nên mình cần tính toán giá trị sao cho nó thuộc 0 đến 2^24 -1 thôi.

Lưu ý là mình đang config cho nó đúng 1ms, lưu ý những số này là 32 bit



Mình cần tính toán chỗ này nằm trong phạm vi 32 bit

20971520*1/

Mình nên chia trước rồi nhân sau

20971520*1000/ (vượt ngoài uint32_t)

Khi mình gọi systick thì nó bắt đầu đếm rồi và current value bắt đầu load cái giá trị của reload value vào và sau đó bắt đầu count down và khi nó về 0 thì nó sẽ set flag lên thì mình đã biết là nó đã trải qua được 1ms.

```
}
void delay_ms(uint32_t t_ms)
{

}
int main(void)
{
    initPin();
    sys_init();
    while (1)
    {
      GPIO_DRV_Toggle_Bit(RED_LED_PIN);
      delay_ms(500);
    }
}
```

Vậy làm sao để delay đúng tham số mình đưa vào?

```
void delay_ms(uint32_t t_ms)
{
  SysTick->
}
int main(vo
{
    initPin
    sys_ini
    while (1)
    {
      GPIO_DRV_Toggle_Bit(RED_LED_PIN);
      delay_ms(500);
    }
}
```
CALIB const volatile uint32_t
CTRL volatile uint32_t
LOAD volatile uint32_t
VAL volatile uint32_t

Chỗ này thì mình sẽ write vào current value, cái val này là current value, mình sẽ write vào 0.

```
}
void delay_ms(uint32_t t_ms)
{
  SysTick->VAL = 0;
}
```

khi mình write vào current value thì cái current value reset về 0 và cái cờ count flag sẽ được clear luôn. Khi mình write thế này thì nó sẽ bắt đầu đếm lại từ đầu thì mình sẽ đợi (while())

Mình sẽ có 2 vòng while, đầu tiên kiểm tra t_ms lớn hơn ko hay ko và vòng while đợi count flag set lên thì count flag thì mình sẽ lấy thông qua thanh ghi control và mình and với systick control count flag, sau khi thằng này nó bằng 0, tức là sau khi cờ count flag này chưa được set lên và nó sẽ đợi trong này, khi count flag set lên thì đủ 1ms

```
}
void delay_ms(uint32_t t_ms)
{
  SysTick->VAL = 0;
  while ( t_ms >0)
  {
    while((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) == 0)
    {

    }
  }
```

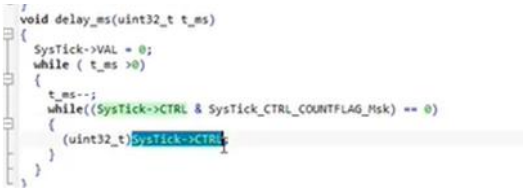
Thì mình sẽ trừ trừ thằng t_ms này

```
}
void delay_ms(uint32_t t_ms)
{
  SysTick->VAL = 0;
  while ( t_ms >0)
  {
    t_ms--;
    while((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) == 0)
    {

    }
  }
}
int main(void)
```

nó sẽ trừ đi 500 lần, t_ms về 0 thì nó sẽ out ra.

Như vậy thì nó sẽ delay được 500ms

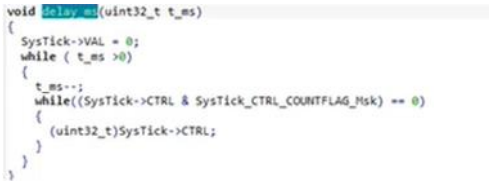
ở trong vòng while thứ 2 thì mình thường sẽ read thanh ghi này 1 lần nữa, mục đích là để nó clear COUNT FLAG này



Cái hoạt động read thì nó sẽ tự động clear cờ này đi

Như vậy đã được implement xong phần delay từng này ms

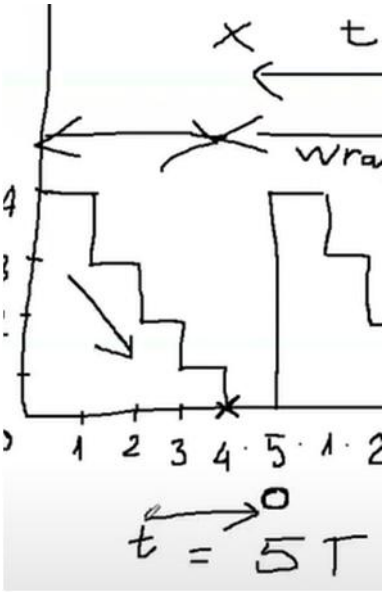
Lưu ý chỗ này khi reset thì nó bằng 0, còn chỗ delay này



Thì mục đích của nó là

Tại vì nếu mình ko clear cái current thì ví dụ current lúc này đang nhảy vào thằng delay vì nó đã được tạo trên này rồi

cho nên là nó đang đếm xuống. nếu mà mình nhảy vào hàm delay lúc đó current value không còn ở vị trí đầu mà nó nằm ở giữa chẳng hạn thì nó sẽ sai số 1 đoạn ngay chỗ này



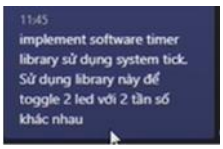
Nó ko phải 1 wrap duration mà chỉ là 1 phần của 1 wrap duration mà thôi.

Clear current value về 0 thì cũng clear flag luôn và sau đó thì nó cũng load lại luôn giá trị current value này bằng giá trị LOAD này

```
SysTick->LOAD = SystemCoreClock*1/1000 -1 ;
SysTick->CTRL = (SysTick_CTRL_CLKSOURCE_Msk|SysTick_CTRL_ENABLE_Msk);
```

Và nó được nạp lại ở đây

```
SysTick->VAL = 0;
while ( t_ms >0)
```



Bài tập là toggle 2 led với 2 tần số khác nhau

Cái software timer thì như cái tên của nó là timer phần mềm, nó chỉ dựa trên 1 cái hardware (chính là systick) còn Swtimer là phần mềm do mình code, do mình implement dựa trên 1 cái HW là systick để mình tạo ra nhiều bộ timer nhưng thực tế trên HW nó lại chỉ có 1 timer thôi. Những bộ timer đó dùng để toggle 2 led với 2 tần số khác nhau.

Bài trên (systick) mục đích là để nghiên cứu về clock, thay đổi clock thì config như thế nào để sau này những bài sau học về timer thì phải config timer về chẵn số để mình tính, còn đây là lẻ số nên mình khó tính và sinh ra sai số

Bây giờ chuyển sang ngắt thì mình enable thêm ngắt

```
}
void initSysTick()
{
  SysTick->LOAD = SystemCoreClock*1/1000-1;
  SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk | SysTick_CTRL_TICKINT_Msk ;
}
```

Thì khi count đếm về 0 và nó sinh ra ngắt, nó sẽ nhảy đến 1 handler và mình

```
        PUBWEAK SysTick_Handler
        SECTION .text:CODE:REORDER:NOROOT(1)
SysTick_Handler
        B .

        PUBWEAK DMA0_IRQHandler
        PUBWEAK DMA0_DriverIRQHandler
        SECTION .text:CODE:REORDER:NOROOT(2)
```

Hàm này khi có interrupt xảy ra thì có nghĩa là cờ này được set lên

```
& SysTick_CTRL_COUNTFLAG_Msk) =
```

Thì nó sẽ nhảy đến hàm interrupt này (symbol của nó có trong asm) là SysTick_Handler

```
}
void SysTick_Handler()
{

}
```

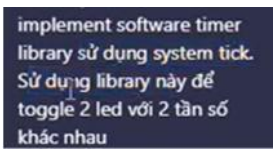
```
uint32_t count=0;
```

ở đây ví dụ mình khai báo 1 biến.

```
void SysTick_Handler()
{
  count++;
  (uint32_t)SysTick->CTRL;
}
```

Dòng thứ 2 là read, tức là clear cờ đó đi.

Bây giờ ko sử dụng hàm delay nữa



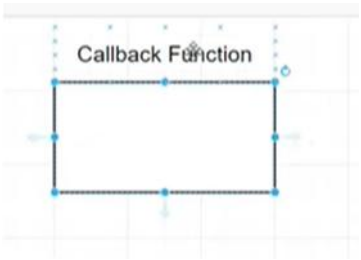
Mình sử dụng lib mà mình đã viết ra rồi và để toggle 2 led với 2 tần số khác nhau.

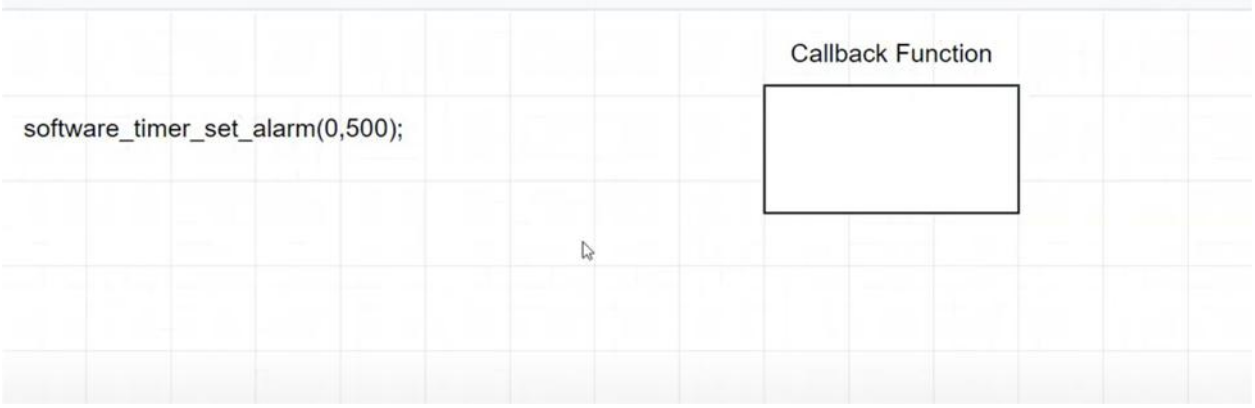
Mục đích là tầng ứng dụng ngta sử dụng như thế này:

Người ta muốn đưa vào thông số có đơn vị là ms. Mình muốn bộ software nào, ví dụ bộ softwar timer 0 chẳng hạn

bộ sw timer sinh ra callback mỗi 500ms, mỗi 500ms thì gọi 1 function callback (callbackfunction do mình tự code)

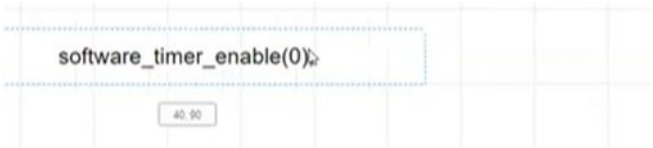
Ko phải handler như thế này từ vector ngắt trỏ ra rồi, đó là ngắt do phần cứng, ở đây mình implement sw timer do mình tự thiết kế





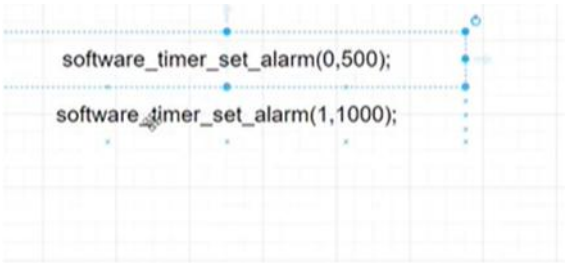
Cứ mỗi 500ms thì cái call back function được gọi

Mình sẽ cần thêm function enable nữa



Để cho bộ sw này enable

Ví dụ ở đây mình set up 1 bộ sw timer khác, 1000 này sẽ gọi đến 1 hàm khác

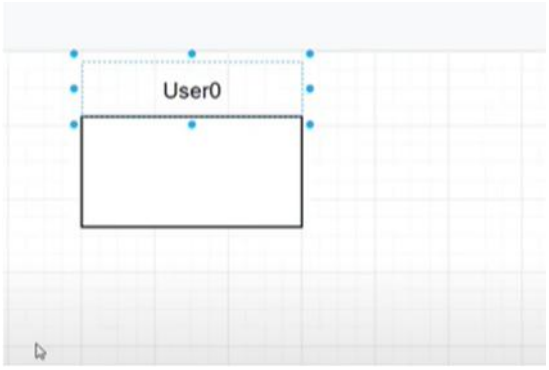


Mình sẽ có thêm 1 cái nữa là sw reg callback, là đăng ký 1 function callback

Cái này thì ở bài C cũng có rồi, ở đây sử dụng function pointer, mình sẽ đưa vào 1 tên function

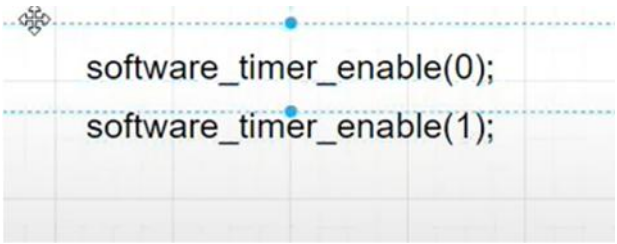
Ví dụ ở đây mình config 2 tham số, tham số thứ nhất là mình config cho bộ timer nào và cái thứ 2 là tên function

Như thế này có nghĩa là khi mà mình config cho bộ timer 0, cứ mỗi 500ms thì nó lại gọi 1 cái call back function, và cái callback function đó được define là user 0



Mỗi 500ms thì lại gọi đến user 0

Tương tự với user 1



Sau đó là enable. Đây là những cái mà người dùng sử dụng

Khi gọi những cái api như thế này thì timer sẽ đếm và khi đủ 500 thì gọi đến user 0 và đủ 1000ms thì gọi đến user1.

Để làm dc cái này thì sử dụng systick

Đây là 1 cái hw timer

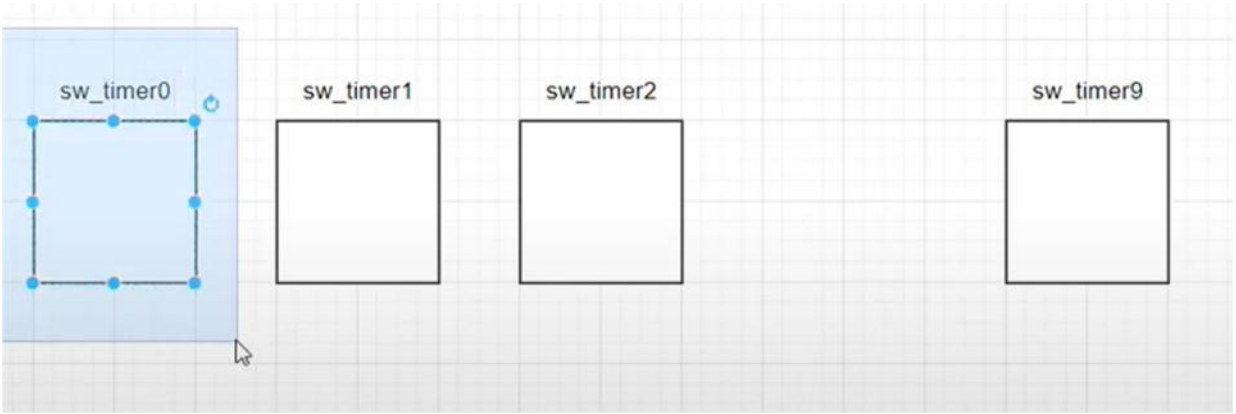
Còn sw timer là do mình implement

Ý tưởng ở đây là sử dụng systick sinh ra 1 ngắt với đơn vị là 1ms

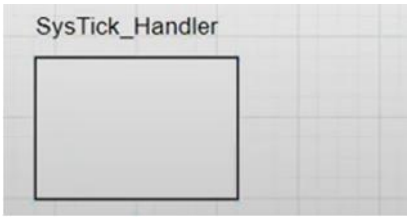
```
}
void SysTick_Handler()
{
  count++;
  (uint32_t)SysTick->CTRL;
}
int main(void)
{
```

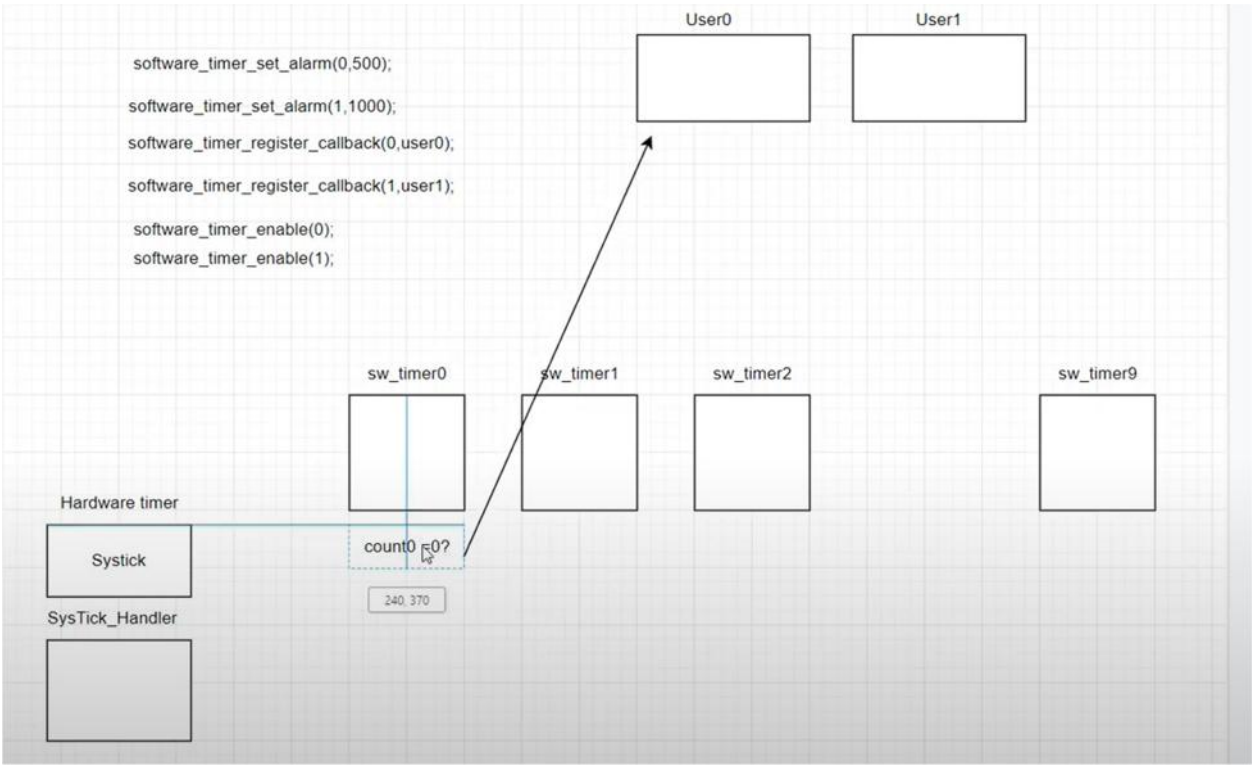
Cứ 1ms thì sinh ra 1 interrupt

Không có giới hạn cho việc tạo n bộ sw timer, ko bị giới hạn ở phần cứng.



Sw timer này phải giữ được 1 giá trị 500, sau đó thì cứ mỗi lần interrupt thì ở đây có 1 chương trình code là systick handler, mỗi lần interrupt ntn là trải qua 1ms thì mình sẽ trừ đi giá trị count

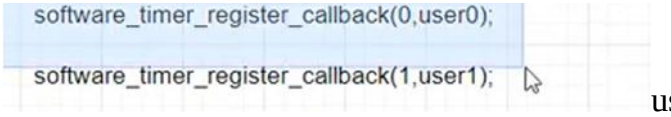




Nếu thg này bằng 0 thì gọi user1

Bình thường hw tự động

Còn sw thì tự trừ, mỗi lần có interrupt thì tự trừ

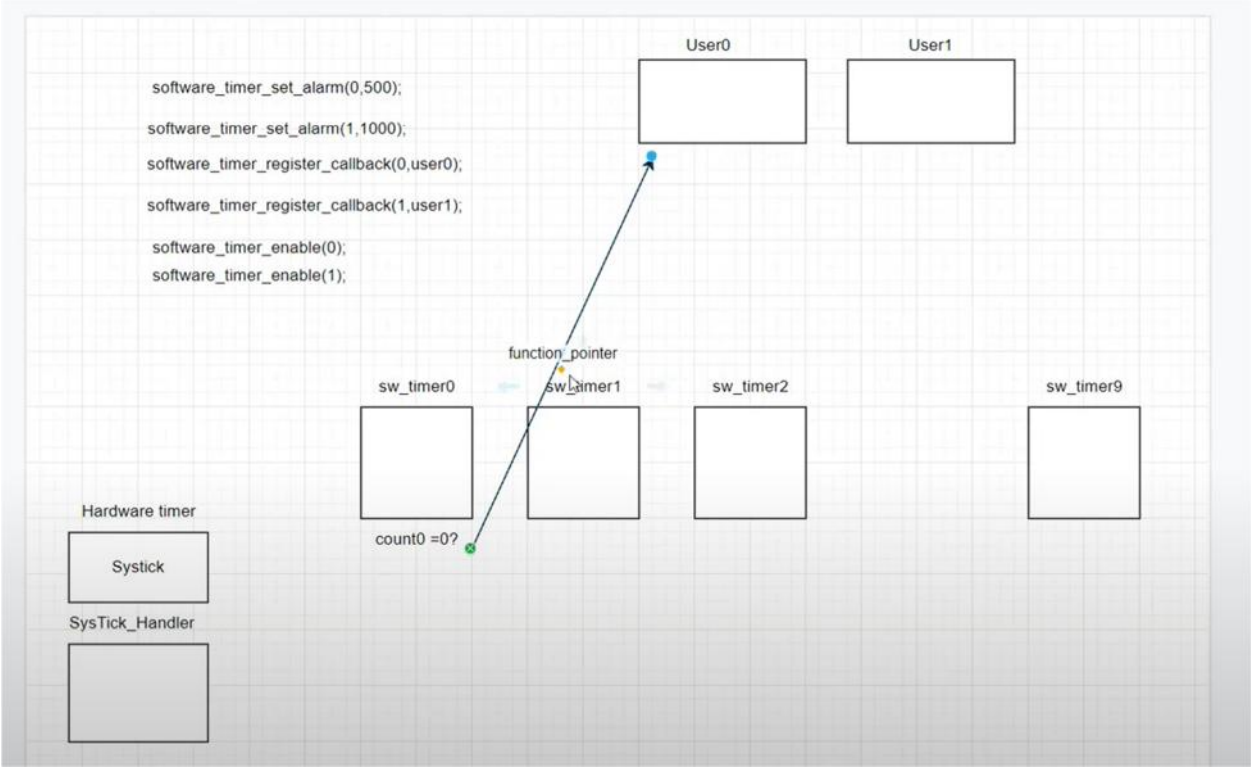


user function thì mình biết được thông qua hàm này register callback để đưa xuống lib địa chỉ nào của hàm này.

Thì đây là ví dụ cho sw timer0

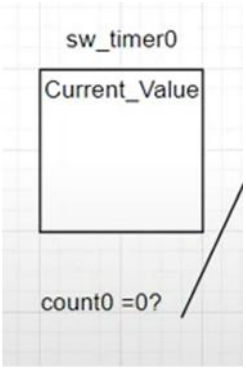
Những bộ sw này đều phải được xử lý khi 1 sự kiện ngắt của systick xảy ra thì mình phải xử lý hết tất cả bộ timer này.

Mình có thể làm rất nhiều bộ sw timer khác nhau với chu kỳ ngắt khác nhau nhưng thực tế chỉ có 1 bộ hw timer, nó chỉ tiêu tốn tài nguyên của 1 systick như thế này thôi, còn những sw timer kia là do thư viện.



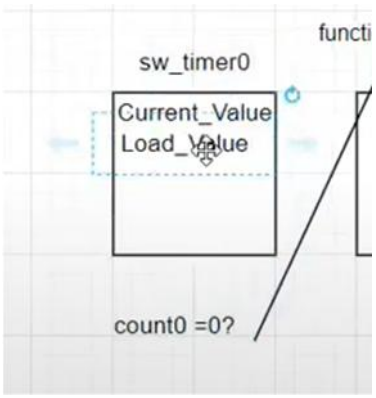
Function pointer này khi mình làm dự án, nó sẽ gọi qua các layer với nhau bằng các function pointer

Thay vì đặt là count thì đặt là current value



Cho người dùng setup như thế này rồi nó trừ trừ xuống

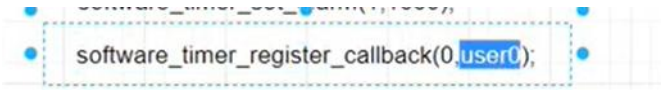
Mình sẽ làm 1 bộ sw timer count down cho giống systick, mình sẽ có 1 cái load value



là thông số người dùng sử dụng thông qua API này (500)

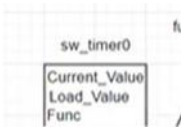
Mình sẽ lấy thông số này để điền vào load value

Thông số thứ 3 mà mình cần là function

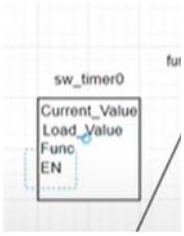


Nó sẽ chứa địa chỉ của người dùng họ đăng ký call back này

Ví dụ họ muốn gọi đến user 0 thì sw này cũng phải có 1 biến để lưu được địa chỉ của user 0 để khi mà current value này về 0 thì nó sẽ gọi đến Func

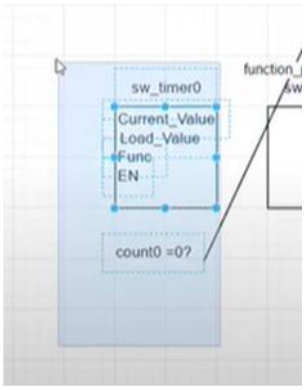


1 cái cuối cùng là mình cần enable và cũng cần phải có disable

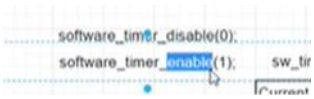


Bằng 0 thì disable (EN có kiểu dữ liệu là boolean là 0 or 1 thôi)

Khi nó enable lên nó thì count thằng current value xuống. còn khi EN bằng 0 thì mình disable đi thì thằng current value = 0, có được count hay ko và dựa theo thằng EN bằng 1 hay ko thì cái đó



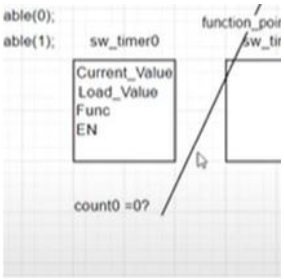
Tất cả những thằng này đều phải sw hết chứ ko như HW



Khi mình gọi hàm này thì nó sẽ tác động đến biến EN của timer 1

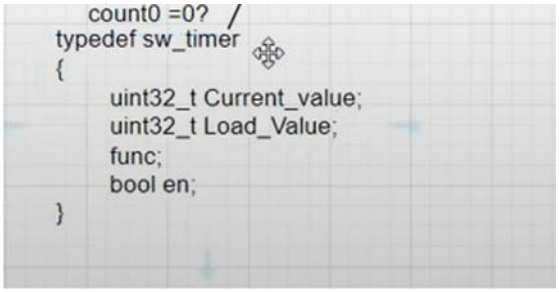
Còn disable 0 vào timer 0 thì nó sẽ tác động đến EN của timer0

Của sw timer nào thì chỉ tác động đến phần tử của sw đó thôi. Mình sẽ cố định 10 bộ sw ntn, có thể khai báo nó như là 1 mảng cũng được, ko cần phải cấp phát động

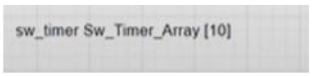


Mỗi phần tử đều có 1 tham số/ phần tử như thế này thì mình sẽ ứng dụng structure để thay thế để mình sử dụng type define.

Tránh ghi nhầm với systick



Khi mình define ntn thì mình define 1 mảng có type ntn:



Ngoài ra những hàm về systick ví dụ như mình cần init systick rồi các thứ, mình cần init cái tạo ra systick là bao nhiêu nữa bởi vì cái systick mình đang mặc định mỗi lần interrupt là trải qua 1 ms

còn cái cấu thành cho nó có đủ 1ms hay ko và mình cần có 1 API để người dùng gọi từ trên này xuống.

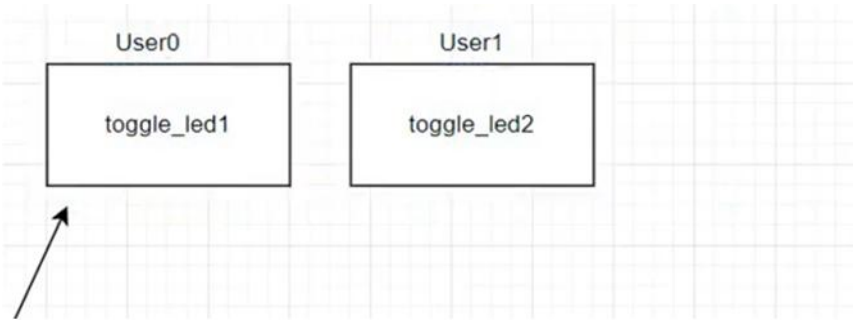
Họ muốn cấu thành Systick là 1 API

Nếu mình set up systick này có chu kỳ lớn hơn 1ms thì nó ra được chu kỳ chính xác call back được đến function này

Phần này có thể gọi xuống systick như hôm trước đã implement phần HAL và phần driver

Thì thằng này sẽ nằm ở layer thấp nhất. phần systick này có thể để ở phía dưới. những bộ software timer sẽ thuộc phần driver thì mình sẽ bỏ vào trong layer driver.

Mình define các hàm user 0 và user rồi sau đó toggle led trong này


User0
User1
toggle_led1
toggle_led2