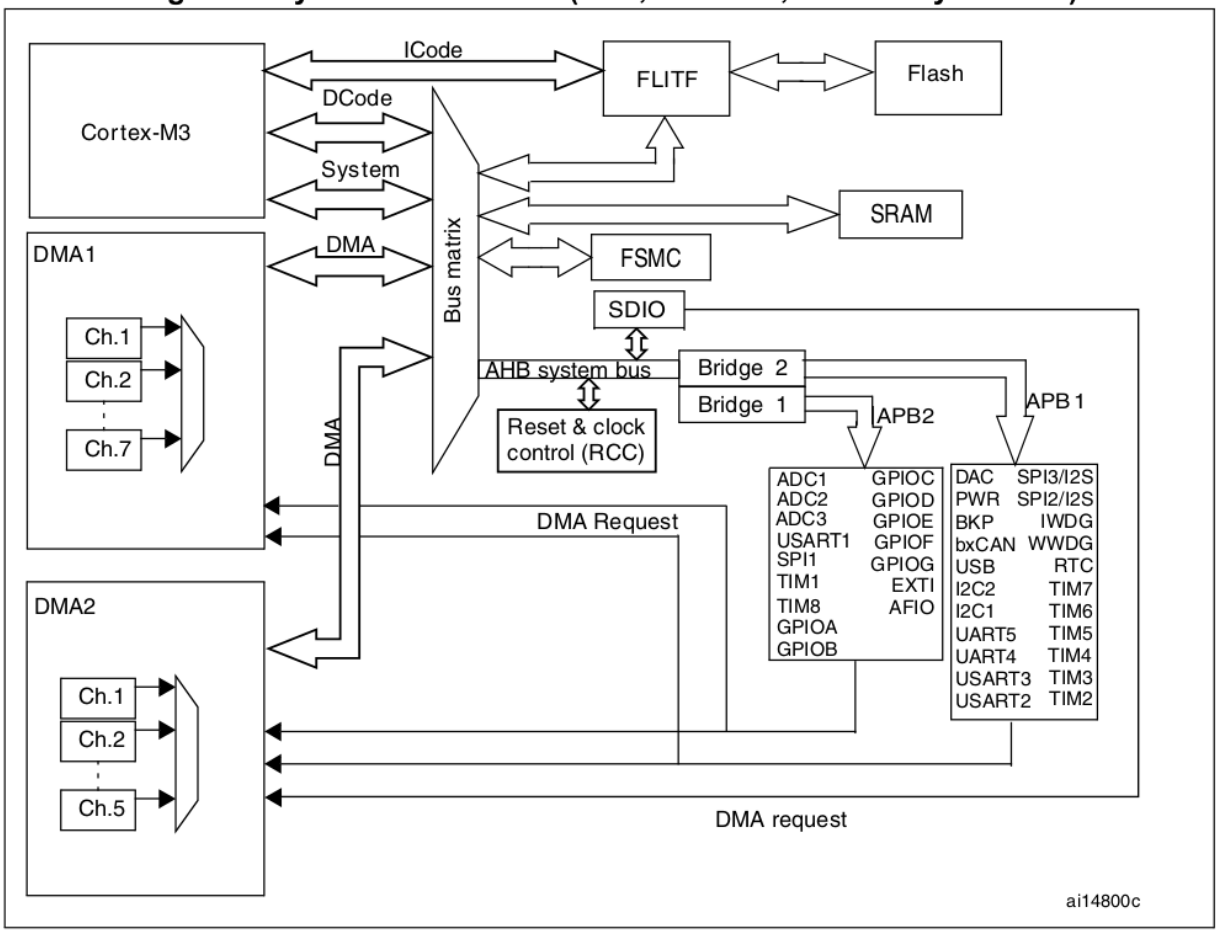
STM32 - Code thanh ghi với thư viện CMSIS

Kiến trúc của dòng vđk STM32F103:

Figure 1. System architecture (low-, medium-, XL-density devices)



Các thành phần:

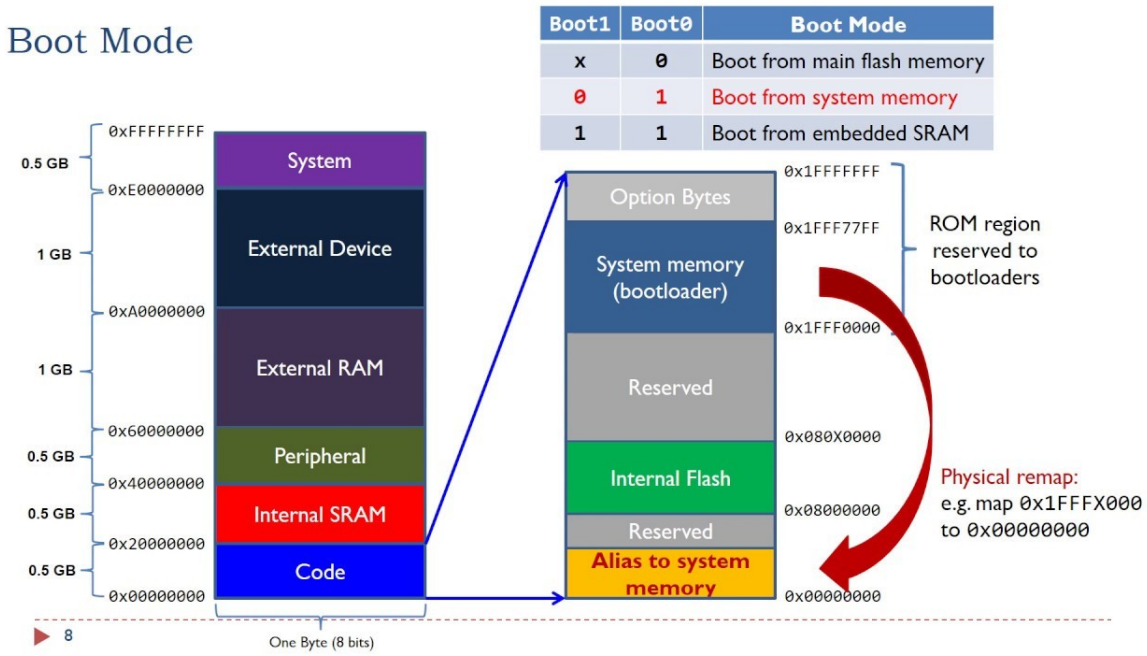
+Lõi Arm Cortex M3
+Các ngoại vi
+Bộ DMA
+Các đường bus.
+Bộ nhớ Flash
+Bộ nhớ RAM
Các đường bus gồm có:
+AHB bus (Advance High-performance Bus) là bus tốc độ cao để kết nối hệ thống với nhau, fmax = 72 MHz
+ APB bus (Advance Peripheral Bus) là bus kết nối ngoại vi, APB1 có tần số max là 36 Mhz, APB2 có tần số max là 72 Mhz
+ AHB và APB được nối với nhau bằng các cầu liên kết có chức năng đồng bộ kết nối giữa 2 bus APB và AHB.
+ Đường Icode để kết nối từ lõi Cortex ra Flash Instruction Interface, Flash chính là thành phần bộ nhớ lưu chương trình.

Các dòng STM32F1 sử dụng lõi ARM Cortex M3 là kiến trúc Harvard, với đường bus dữ liệu và bus fetch instruction khác nhau, nhưng hệ thống địa chỉ của code và data chung nhau.

Memory map và các bootmap

STM32 có các vùng bộ nhớ sau:
+ Vùng Code, địa chỉ từ 0x00000000 đến 0x20000000, vùng này chịu trách nhiệm lưu code của vđk
Vùng này được chia thành các vùng:
+ System Memory
+ Internal Flash
+ System Memory
+ Option Byte

Và các vùng quản lý còn lại:
+ Vùng SRAM từ 0x20000000 đến 0x40000000
+ Vùng External RAM
+ Vùng External Device
+ Vùng System



Chương trình có thể được lưu trong SRAM, FLASH hoặc là system memory.

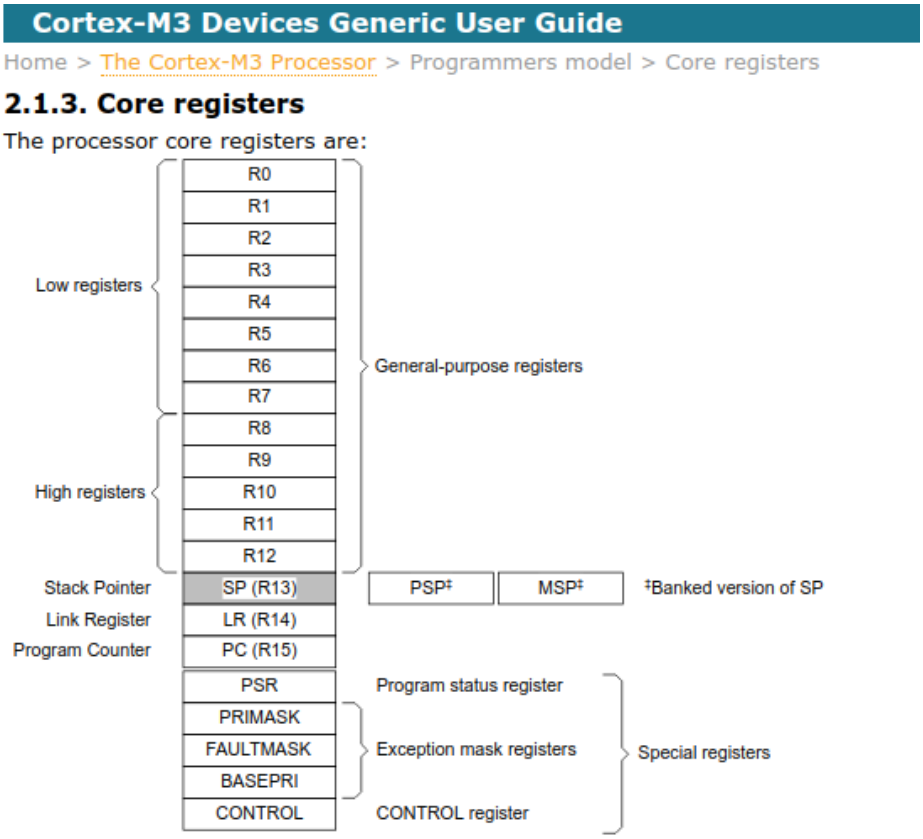
Instruction và Interrupt vector table

Để vđk chạy và thực hiện các chức năng thì các Core register được sử dụng để thực hiện các chương trình, việc hiểu rõ các thanh ghi này không cần thiết nhưng có thể có ích khi debug.

Các câu lệnh của vi điều khiển được gọi là instruction, các lệnh này được lấy (fetch) từ bộ nhớ vào để CPU thực hiện và đưa ra kết quả qua các đường bus, kiến trúc Harvard của ARM Cortex gồm có các đường instruction và data riêng. Có 2 chế độ hoạt động của chip ARM Cortex là Thumb state và ARM state, trong đó Thumb state sử dụng instruction 16 bit, nhẹ và đơn giản hơn trong khi ARM state sử

dụng instruction 32 bit. Với dòng vđk nhỏ ta thường thấy Thumb state (ví dụ như dòng ARM Cortex M3).

Giống dòng vđk 8051 để thực thi chương trình ARM Cortex M3 cũng sử dụng các thanh ghi hệ thống để thực hiện các instruction, việc code Assembly cho ARM là không cần thiết.



Trên đây là các Core register của ARM Cortex M3.

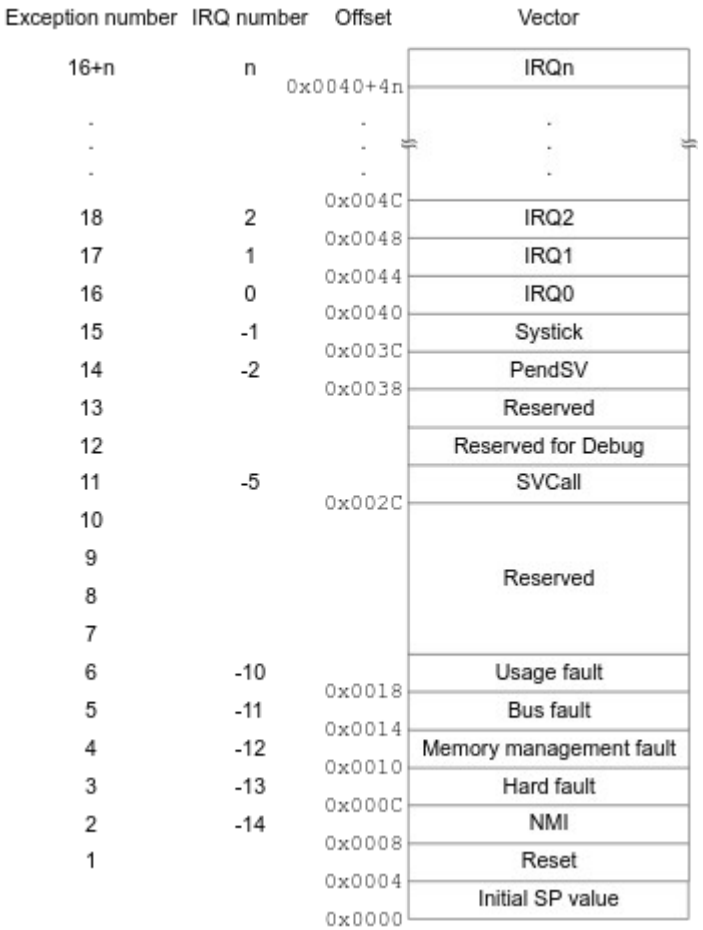
Các thanh ghi Stack Pointer, Program Counter, General Register cũng có chức năng như là các thanh ghi của 8051, ngoài ra còn có các thanh ghi Main Stack Pointer xử lý các tác vụ của đơn nhiệm của chương trình chính, trong khi PSP (Processor Stack Pointer) là con trỏ cho stack của các thread, giúp việc xử lý các
tác vụ đa nhiệm (OS) đơn giản hơn. Ngoài ra các thanh ghi đặc biệt có thể tìm hiểu thêm trong tài liệu của ARM

Quá trình boot của STM32

Khi chân NRST được bật, qúa trình thực hiện chương trình (boot) bắt đầu, bit BOOT0 và BOOT1 (mà trong kit bluepill đã nối ra 2 header) được kiểm tra, việc cấu hình 2 bit này sẽ quyết định quá trình boot như hình trên.

Sau khi đã đọc 2 bit BOOT0 và BOOT1, ngắt Reset sẽ xảy ra, CPU sẽ fetch giá trị tại vị trí 0x00000000 vào thanh ghi MSP, giá trị tại điểm 0x00000004 sẽ được fetch vào thanh ghi PC

CPU sau đó sẽ thực hiện phần Reset Handler, Reset Handler sẽ xử lý việc cấu hình bộ nhớ để thực hiện chương trình, sau đó Reset Handler gọi hàm main để thực hiện chương trình.

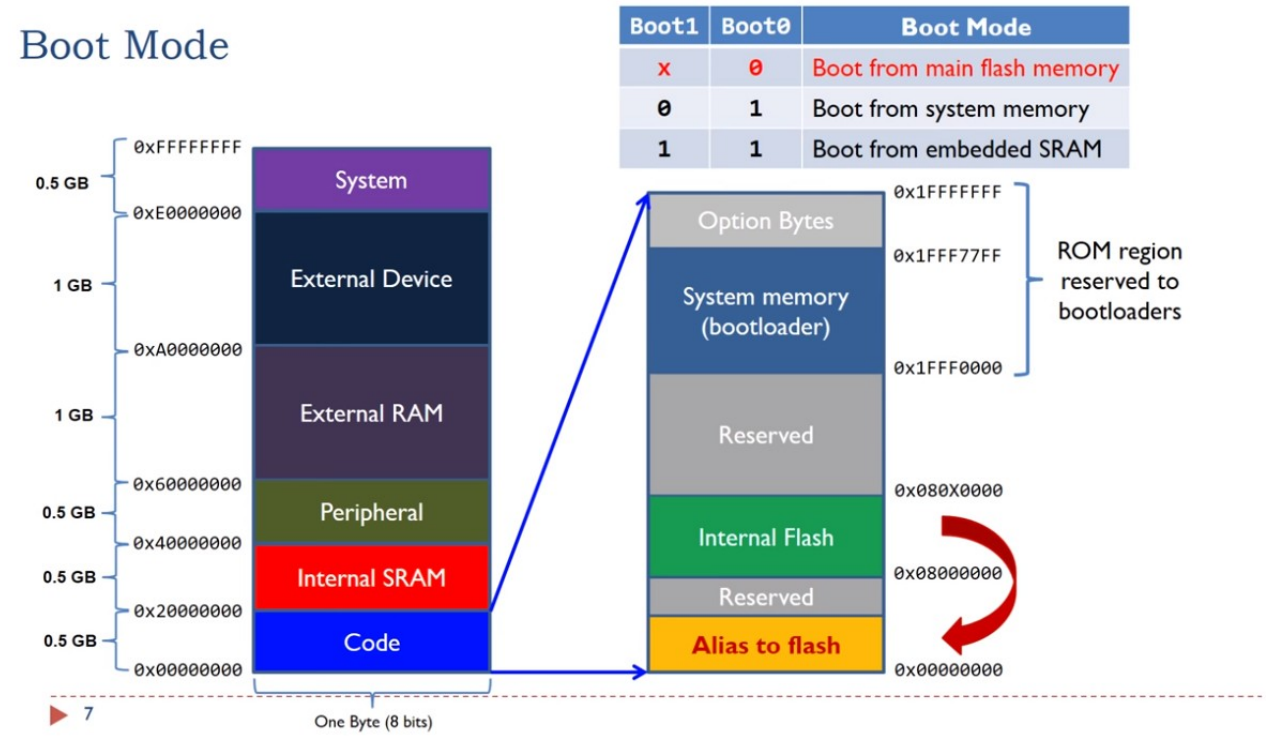


Giá trị của data ở 0x00000000 còn quy định CPU đang hoạt động ở chế độ Thumb state hay ARM state, nếu là ở Thumb state bit LSB sẽ được set thành 1, thể hiện là Thumb state
Ví dụ: Giá trị ở 0x00000000 là 0x08000269 → Reset handler sẽ ở vị trí 0x08000268, bit LSB = 1 quy định Thumb state. Tuy nhiên khi CPU đọc 2 bit BOOT và boot thì giá trị 0x08000269 được ghi vào thanh ghi PC.

Giá trị ở 0x00000004 là 0x02001BB0 → giá trị này quy định vị trí đầu tiên của Stack, sau quá trình boot sẽ được lưu vào thanh ghi Stack Pointer.

Quá trình khởi tạo SP và PC là như trên, tuy nhiên trước khi thực hiện quá trình này, tùy thuộc vào từng mode boot mà SP và PC lại được load từ các phần bộ nhớ khác nhau:

Ví dụ với boot từ Flash:

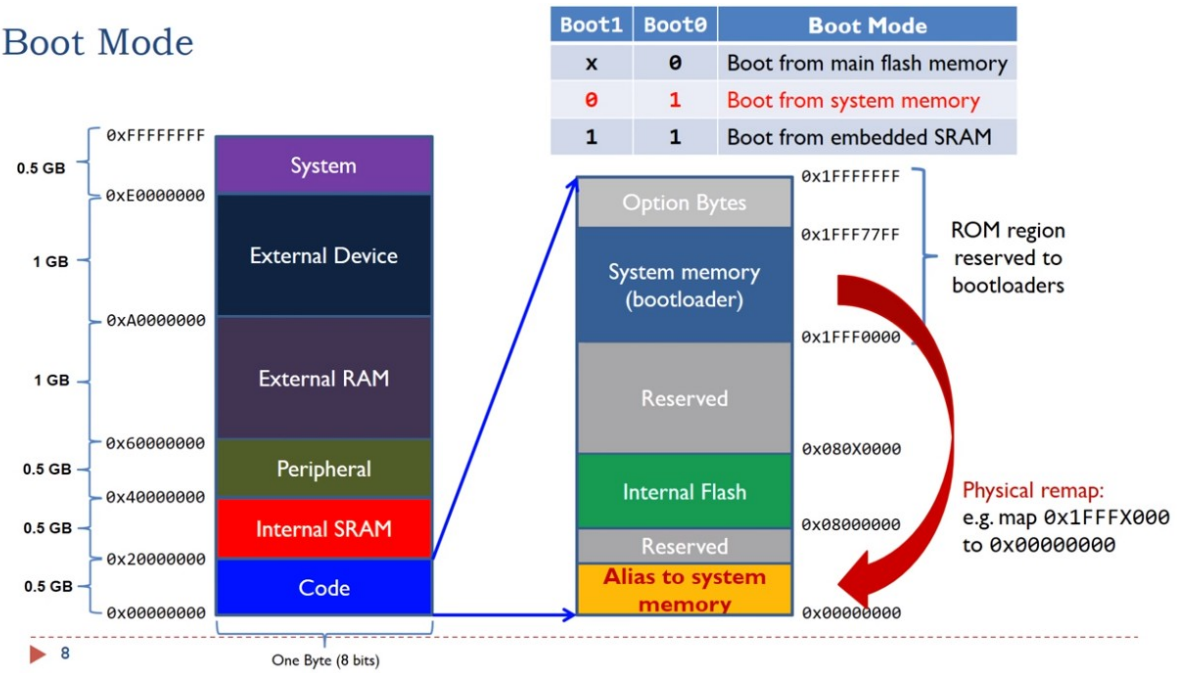


Ví dụ: Flash bắt đầu ở 0x08000000, khi chế độ boot Flash diễn ra, CPU sẽ tham chiếu địa chỉ 0x00000000 đến 0x08000000 và thực hiện ở đây.

Lúc này giá trị 0x00000000 và 0x00000004 được tham chiếu đến sẽ có giá trị của 0x08000000 và 0x08000004, do đó thực chất là SP và PC được cấu hình để chạy chương trình trong vùng bộ nhớ FLASH

Boot mode từ System Memory (Bootloader):

Ở phần này thì địa chỉ bộ nhớ System Memory và Option Bytes được tham chiếu vào vùng boot: Phần này là bộ nhớ hệ thống lưu trong ROM, người dùng có thể lập trình để nạp lại Firmware cho FLASH,….

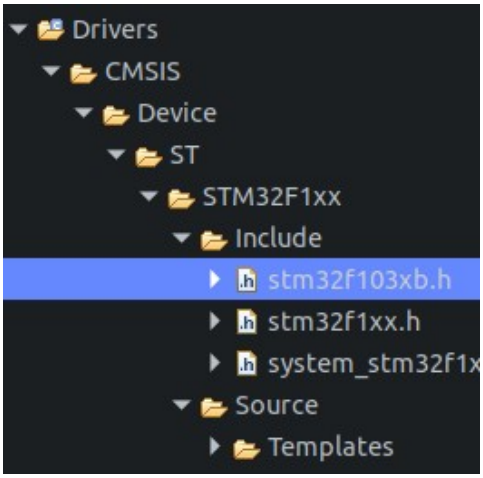


Tham khảo thêm: https://www.youtube.com/watch?v=3brOzLJmeek

Special Function Register

Các chức năng trong STM32 được quản lý bằng hệ thống các thanh ghi chức năng đặc biệt (SFR), các thanh ghi này được quy định trong bộ nhớ Flash của vđk, và việc truy cập vào chúng có thể thực hiện như sau:

Để truy cập và sử dụng các thanh ghi này, chúng ta cần tham khảo file header stm32f103xb.h trong thư mục CMSIS:



Các thanh ghi này được gọi dựa trên địa chỉ của chúng, và chúng được define trong header này dưới dạng các kiểu __IO uint32_t, trong đó __IO thể hiện biến này tham chiếu đến bộ nhớ của vđk, và uint32_t là 32bit dữ liệu của thanh ghi, với các thanh ghi liên hệ với nhau được khai báo trong một struct, (một kiểu dữ liệu của C, trong đó có một struct có chứa nhiều biến có thể có kiểu và giá trị khác nhau). Các thanh ghi có liên hệ với nhay được tổ chức thành một struct, việc này làm việc tra cứu thanh ghi thuận tiện hơn.

```
420 /**
421   * @brief Reset and Clock Control
422   */
423
424 typedef struct
425 {
426   __IO uint32_t CR;
427   __IO uint32_t CFGR;
428   __IO uint32_t CIR;
429   __IO uint32_t APB2RSTR;
430   __IO uint32_t APB1RSTR;
431   __IO uint32_t AHBENR;
432   __IO uint32_t APB2ENR;
433   __IO uint32_t APB1ENR;
434   __IO uint32_t BDCR;
435   __IO uint32_t CSR;
436
437
438 } RCC_TypeDef;
```

Để truy cập một thanh ghi, ta cần include header này vào và thực hiện truy cập giá trị thanh ghi với cú pháp sau:

Tên struct - > Tên thanh ghi

Cả thư viện HAL (Hardware Abstraction Layer) và LL (Low Layer) đều được xây dựng dựa trên việc sử dụng header này.

Ví dụ: Để đọc thanh ghi RCC_CR trong struct RCC vào biến temp ta cần thực hiện câu lệnh:

```
uint32_t temp = RCC->CR;
```

* Các thao tác trên thanh ghi sẽ được thực hiện như sau:
+ Gán giá trị cho thanh ghi:

```
FLASH->ACR = 0x00000001;
```

Giá trị được gán cho thanh ghi phải là một số uint32_t (số nguyên 32 bit không dấu).
+ Thay đổi giá trị (Modify thanh ghi):

```
FLASH->ACR = (FLASH->ACR) | LL_FLASH_LATENCY_2;
```

Ở ví dụ này ta cho thanh ghi thực hiện phép or bit với một mặt nạ (tức giá trị bit ta muốn dịch chuyển tại vị trí của nó, các vị trí còn lại là 0) nên khi thực hiện phép or sẽ chỉ thay đổi giá trị bit ta cần thay đổi, còn lại không thay đổi.
Như ở ví dụ trên, LL_FLASH_LATENCY_2 thực chất là các bit ở vị trí từ 0 đến 2 của thanh ghi FLASH_ACR, giá trị của 3 bit này do đó chạy từ 000 → 111, chính vì vậy mà ta có thể tạo ra mặt nạ cho 3 bit này như sau:

```
#define FLASH_ACR_LATENCY_Pos              (0U)
#define FLASH_ACR_LATENCY_1 (0x2UL << FLASH_ACR_LATENCY_Pos) /*!< 0x00000002 */
```

Vị trí bắt đầu của các bit latency được định nghĩa nhờ `FLASH_ACR_LATENCY_Pos` , phép dịch trái `FLASH_ACR_LATENCY_Pos` vị trí sẽ đẩy giá trị cần thay đổi tới vị trí của nó, do đó ta thu được mask cần tìm để thay đổi giá trị thanh ghi.
Câu lệnh tương đương:

```
FLASH->ACR |= LL_FLASH_LATENCY_2;
```

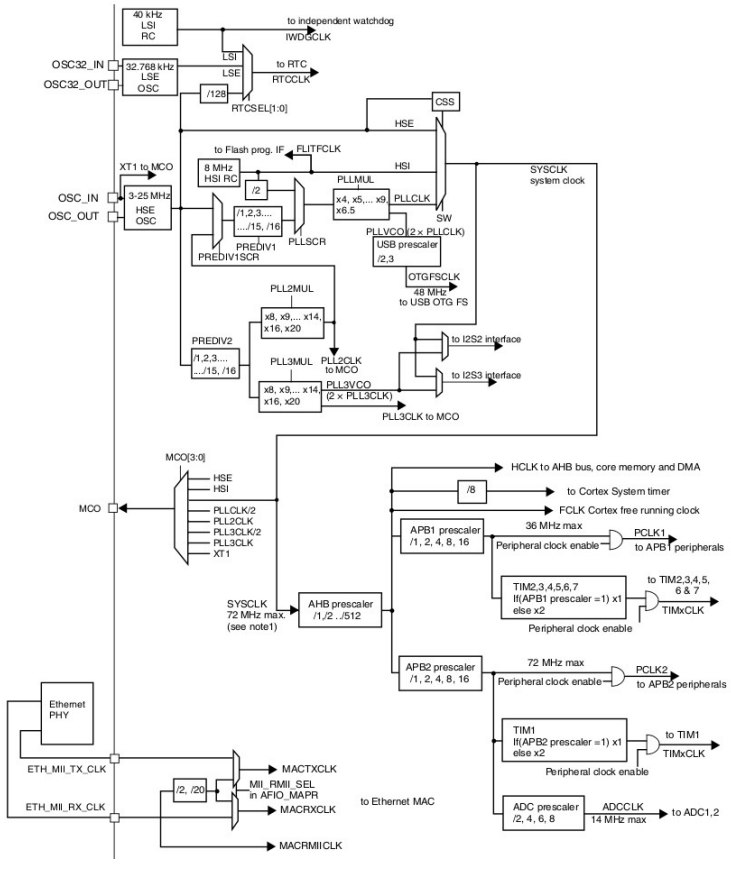
*Phần lớn các mask này đã được định nghĩa trong header và thư viện LL nên việc code thanh ghi sẽ không quá khó khăn.

Tổ chức Clock STM32F103:

Trong vi điều khiển thì clock cũng như là trái tim để vđk có thể chạy được, chúng ta sẽ tìm hiểu
+ Clock của Stm32f103 có các clock như thế nào
+ Cách cấp xung clock cho Stm32F103
+ Cách setup clock cho STM32F103.

1. Các clock của STM32:

Sơ đồ clock



Trong đó clock để thực hiện chức năng của lõi Arm Cortex M3 được gọi là SYSCLK, clock này có thể được cung cấp từ các nguồn như sau:
HSE: nguồn dao động tần số cao ngoại, tần số từ 4 - 25Mhz
HSI: nguồn dao động tần số thấp nội tần số 8Mhz
PLL: bộ nhân tần, có chức năng chỉnh sửa tần số dao động sao cho phù hợp với yêu cầu của hệ thống.

*Nguồn clock cấp vào có thể là sóng sin, xung tam giác hoặc sóng vuông, tuy nhiên phải thỏa mãn điều kiện là thời gian mức cao bằng off = 50%.

Ngoài ra còn có các nguồn xung cho bộ Real Time Clock như:

LSI: Mạch dao động LC tần số thấp, có xung nhịp mặc định 40kHz.
LSE: Nguồn dao động RTC ngoại, có thể là thạch anh với tần số mặc định là 36.768 kHz. (2^15)

Các nguồn clock có thể được thay đổi tùy theo yêu cầu và chế độ hoạt động của chip
Trong đó các chế độ hoạt động của clock STM32 như sau:

+ Sử dụng HSI (mạch dao động nội)
+ Sử dụng HSE (Thạch anh hoặc nguồn xung tương đương)
+ Sử dụng HSI và HSE phối hợp với bộ chia tần và bộ nhân tần, theo thứ tự từ nguồn dao động → bộ chia tần (Prediv) → bộ nhân tần(PLL)

Các nguồn dao động trên sẽ được đi qua một bộ MUX để chọn ra nguồn dao động hệ thống (Sysclk)

Từ nguồn Sysclk này sẽ được sử dụng cho các ứng dụng như sau:

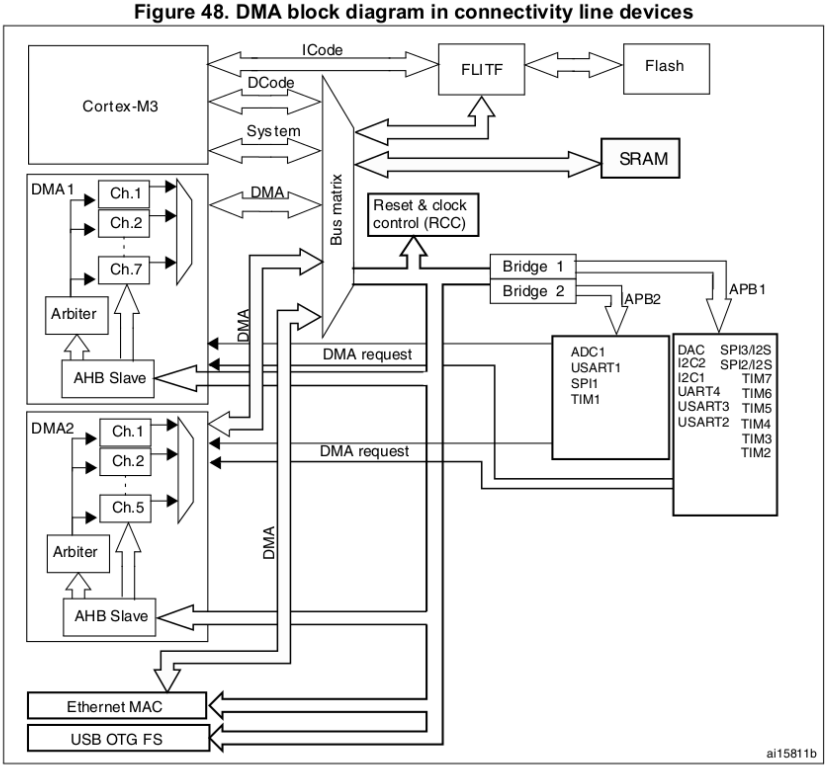
+ Đưa vào bộ nhân tần PLL từ 1 → 3 (Chú ý là không liên quan đến bộ nhân tần PLL của nguồn dao động vào), liên hệ với đầu ra tần số của chip (MCO = Microcontroller Clock Output)

+ Đưa vào bộ chia để vào AHB - bus kết nối tốc độ cao, đường bus AHB này n. Từ đây nguồn clock được chia ra các đường như sau:

1. HCLK vào bus AHB kết nối với các thành phần trong chip và DMA
2. /8 để đưa vào timer hệ thống (Systick). Systick là một timer 24 bit, đếm xuống và auto reload, ngắt được tạo khi giá trị đếm bằng 0, được bulid sẵn trong lõi ARM mục đích sử dụng chính là để tạo delay với độ chính xác cao, Systick cũng được sử dụng trong RTOS như là timer mặc định
3. FCLK: nguồn clock cho việc debug, bằng việc sử dụng clock này, ngắt debug hoặc khi chip vào trạng thái sleep debugger vẫn biết được.
4. Bus APB1: clock đi vào bus này sẽ qua một bộ chia, và clock lớn nhất của bus này là 36Mhz, clock từ bus này đi vào các ngoại vi và Timer từ 2 → 7 (Tùy vào dòng vđk trong họ f1 mà sẽ có các timer khác nhau, tham khảo thêm trong các reference manual)
5. Bus APB2: clock này đi vào bộ chia như APB1, với tốc độ tối đa 72Mhz và vào Timer 1 (Advance control timer)

* Ngoài ra AHB bus còn là đường truyền cho Icode, FLITF để giao tiếp flash và lấy instruction từ flash vào chip, đây cũng là đường truyền DMA của chip.

Figure 48. DMA block diagram in connectivity line devices



Để sử dụng được STM32, chúng ta phải thực hiện được việc set clock cho chip, với "hệ sinh thái STM32" mà STMicroelectronic đang xây dựng với hạt nhân là STM32CubeIDE hỗ trợ gen code rất nhanh và tiện thì hầu như ta không quan tâm đến clock nữa, nhưng tìm hiểu qua nó cũng không phải là vô ích. Thực chất các chuyên gia của STM khuyên rằng việc gen code chỉ nên thực hiện khi người dùng đã có hiểu biết về chip.

Chúng ta sẽ tìm hiểu về các thanh ghi Clock của STM32:
Như ví dụ ở trên ta là các thanh ghi quản lý việc Reset và Clock của STM32.

Ta sẽ xét đến các thanh ghi sau:

Thanh ghi điều khiển clock RCC_CR
Thanh ghi điều cấu hình clock RCC_CFGR
Thanh ghi cấu hình ngắt clock RCC_CIR
Thanh ghi cho phép ngoại vi APB1 RCC_APB1ENR
Thanh ghi cho phép ngoại vi APB1 RCC_APB2ENR

```
420 /**
421   * @brief Reset and Clock Control
422   */
423
424 typedef struct
425 {
426   __IO uint32_t CR;
427   __IO uint32_t CFGR;
428   __IO uint32_t CIR;
429   __IO uint32_t APB2RSTR;
430   __IO uint32_t APB1RSTR;
431   __IO uint32_t AHBENR;
432   __IO uint32_t APB2ENR;
433   __IO uint32_t APB1ENR;
434   __IO uint32_t BDCR;
435   __IO uint32_t CSR;
436
437
438 } RCC_TypeDef;
```

1. Thanh ghi RCC_CR: Thanh ghi điều khiển clock

Địa chỉ và các bit:

8.3.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		PLL3 RDY	PLL3 ON	PLL2 RDY	PLL2 ON	PLLRD Y	PLLON	Reserved				CSSON	HSEBY P	HSERDY	HSEON
		r	rw	r	rw	r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Các bit này đại diện cho các chức năng:
+ Bật tắt các bộ PLL (Các bộ PLL để đưa xung ra (MCO) hoặc là I2S)
+ Kiểm tra các bộ PLL đã sẵn sàng chưa
+ Bật bộ HSE, HSI
+ Kiểm tra bộ HSE, HSI đã hoạt động hay chưa
+ Calibrate bộ HSI
+ …. các chức năng còn lại có thể xem ở reference manual.
Về cơ bản ta cần biết các chức năng trên

Thanh ghi RCC_CFGR: Clock Configuration Register

8.3.2 Clock configuration register (RCC_CFGR)

Các bit cần

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				MCO[3:0]				Res.	OTGFS PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

quan tâm:
+ PLLMUL[3:0] : Số lần nhân của bộ lặp tần PLL, bộ PLL này là bộ PLL đưa vào Sysclk. Các gía trị nguyên thay đổi từ 2 → 16
+ PLLSRC : Nguồn source clock đưa vào PLL nếu giá trị 0 là HSI/2, nếu là 1 thì là PREDIV1. (Lý của hai lựa chọn này là do clock từ HSE vào phải qua PREDIV1, PREDIV1 được quyết định bởi bit PLLXTPRE)
+ PLLXTPRE : Chọn sử dụng bộ nhân chia HSE hoặc HSI trước khi vào bộ PLL, giá trị của bộ này là không chia hoặc chia 2
+ ADC_PRE[1:0] : Điều khiển số chia nguồn clock cho bộ ADC.
+ PPRE2[2:0] : Số chia clock cho bộ APB2, số chia này dao động thuộc tập {2,4,8,16}, chi tiết về set bit tham khảo trong reference manual
+ PPRE1[2:0] : Số chia clock cho bộ APB1, số chia thuộc tập {0,2,4,8,16}
+ HPRE : Bộ chia từ Sysclk sang AHB Clock, độ chia từ 0 → 512.
+ SW[1:0] : Các bit chọn nguồn clock, có thể chọn HSI, HSE hoặc PLL làm nguồn cho sysclk.

Thanh ghi RCC_AHBENR: thanh ghi quản lý nguồn clock của các chức năng AHB thực hiện:

7.3.6 AHB peripheral clock enable register (RCC_AHBENR)

Address offset: 0x14

Reset value: 0x0000 0014

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					SDIO EN	Res.	FSMC EN	Res.	CRCE N	Res.	FLITF EN	Res.	SRAM EN	DMA2 EN	DMA1 EN
					rw		rw		rw		rw		rw	rw	rw

Các chức năng gồm:
+ SDIOEN để điều khiển đọc ghi khi debug.
+ FSMC điều khiển ngoại vi FSMC cho phép thực hiện đọc ghi bộ nhớ ngoại của chip.
+ CRCEN bật chức năng tự động tính CRC
+ FLITF thực hiện bật clock cho bộ FLITF (Flash interface memory) hay bộ đệm lưu instruction từ flash trước khi đưa vào CPU.
+ SDRAMEN bật clock từ cho bus AHB vào SDRAM, STM32F1 có khả năng thực hiện boot từ flash, SDRAM và System Memory.
+ DMA2EN và DMA1EN bật clock cho các bộ DMA chạy.

Thanh ghi RCC_APB2ENR : bật tắt clock cho các ngoại vi được quản lý bởi bus APB2:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Trong đó + TIMxEN bật nguồn clock cho timer thứ x
+ ADCxEN bật nguồn clock cho bộ ADC thứ x
+ SPIxEN bật nguồn clock cho bộ SPI thứ x
+ IOPxEN bật nguồn clock cho PORT GPIO thứ x
+ AFIOEN bật nguồn clock cho các chức năng alternative của GPIO (có thể là debug,...)
Các bit này được sử dụng để bật clock vào trong các ngoại vi được APB1 quản lý.

Tương tự là thanh ghi RCC_APB1EN:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
		rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Các ngoại vi: DACEN bộ DAC
PWREN bật nguồn clock
CANEN ngoại vi CAN
USBEN ngoại vi USB
UARTxEN ngoại vi UART thứ x
SPIxRN ngoại vi SPI thứ x
WWDGEN bật nguồn clock cho Watch Dog Timer

May mắn là thư viện CMSIS đã định nghĩa sẵn các bit này giúp chúng ta không quá mất công để tra cứu bit.

Để setup clock, STM32CubeMX cho phép setup clock bằng GUI, quy trình setup clock như sau:

Các bước setup clock STM32

Bước 1. Setup Flash latency, do Flash và MCU có clock tối đa khác nhau nên nếu chip chạy ở tốc độ cao cần phải set up thời gian trễ của Flash so với MCU, các instruction không kịp sẽ được bộ FLITF lưu lại.

```
FLASH->ACR = (FLASH->ACR) | FLASH_ACR_LATENCY_1;
```

*Các giá trị define này có thể tìm thấy ở header stm32f103xb.h, hầu hết đều có tên như trong manual nên Ctrl+C và Ctrl+F khá dễ.
Ví dụ đây là các bit của thanh ghi RCC_CFGR:

```
/******************  Bit definition for RCC_CFGR register  ******************/
/*!< SW configuration */
#define RCC_CFGR_SW_Pos                    (0U)
#define RCC_CFGR_SW_Msk                    (0x3UL << RCC_CFGR_SW_Pos)         /*!< 0x00000003 */
#define RCC_CFGR_SW                        RCC_CFGR_SW_Msk                   /*!< SW[1:0] bits (System clock Switch) */
#define RCC_CFGR_SW_0                      (0x1UL << RCC_CFGR_SW_Pos)         /*!< 0x00000001 */
#define RCC_CFGR_SW_1                      (0x2UL << RCC_CFGR_SW_Pos)         /*!< 0x00000002 */

#define RCC_CFGR_SW_HSI                    0x00000000U                       /*!< HSI selected as system clock */
#define RCC_CFGR_SW_HSE                    0x00000001U                       /*!< HSE selected as system clock */
#define RCC_CFGR_SW_PLL                    0x00000002U                       /*!< PLL selected as system clock */

/*!< SWS configuration */
#define RCC_CFGR_SWS_Pos                   (2U)
#define RCC_CFGR_SWS_Msk                   (0x3UL << RCC_CFGR_SWS_Pos)        /*!< 0x0000000C */
#define RCC_CFGR_SWS                       RCC_CFGR_SWS_Msk                  /*!< SWS[1:0] bits (System Clock Switch Status) */
#define RCC_CFGR_SWS_0                     (0x1UL << RCC_CFGR_SWS_Pos)        /*!< 0x00000004 */
#define RCC_CFGR_SWS_1                     (0x2UL << RCC_CFGR_SWS_Pos)        /*!< 0x00000008 */

#define RCC_CFGR_SWS_HSI                   0x00000000U                       /*!< HSI oscillator used as system clock */
#define RCC_CFGR_SWS_HSE                   0x00000004U                       /*!< HSE oscillator used as system clock */
#define RCC_CFGR_SWS_PLL                   0x00000008U                       /*!< PLL used as system clock */

/*!< HPRE configuration */
#define RCC_CFGR_HPRE_Pos                  (4U)
#define RCC_CFGR_HPRE_Msk                  (0xFUL << RCC_CFGR_HPRE_Pos)       /*!< 0x000000F0 */
#define RCC_CFGR_HPRE                      RCC_CFGR_HPRE_Msk                 /*!< HPRE[3:0] bits (AHB prescaler) */
#define RCC_CFGR_HPRE_0                    (0x1UL << RCC_CFGR_HPRE_Pos)       /*!< 0x00000010 */
#define RCC_CFGR_HPRE_1                    (0x2UL << RCC_CFGR_HPRE_Pos)       /*!< 0x00000020 */
#define RCC_CFGR_HPRE_2                    (0x4UL << RCC_CFGR_HPRE_Pos)       /*!< 0x00000040 */
#define RCC_CFGR_HPRE_3                    (0x8UL << RCC_CFGR_HPRE_Pos)       /*!< 0x00000080 */
```

Bước 2. Bật nguồn clock source bằng thanh ghi RCC_CR:
Set bit HSEON sẽ cho phép nguồn source tần số cao chạy `RCC->CR |= RCC_CR_HSEON;`

Bước 3. Setup nguồn clock:
Ví dụ khi sử dụng bộ PLL thì ta cần phải thực hiện bước setup prediv và mul của PLL để cho ra tần số cần thiết, việc này được thực hiện bằng cách setup thanh ghi RCC_CFGR như sau:

Ví dụ trên kit stm32f103 có thạch anh là 8Mhz, ta cần tần số là 72MHz, vì vậy ta có thể setup PREDIV = 1, PLL_MUL = 9:

```
RCC->CFGR |= (RCC_CFGR_PLLSRC | 0x00000000U);// Setup PLL source without prescale
RCC->CFGR |=  RCC_CFGR_PLLMULL9;          // Setup PLL multiplication factor
```

Nếu chỉ sử dụng clock source HSE hoặc HSI thì ta chỉ cần set bit HSEON hoặc HSION của thanh ghi RCC_CR.

Để sử dụng clock từ bộ PLL, ta cần bật bit PLLON:
Để bật bộ PLL: `RCC->CR |= RCC_CR_PLLON;`

Bước 4. Setup Sysclk:
Để chọn source clock đưa vào Sysclk chạy ta set các bit System clock switch trong thanh ghi RCC_CFGR:

```
RCC->CFGR |= RCC_CFGR_SW_PLL;
```

Trong trường hợp lấy clock từ HSE trực tiếp:

Bước 5. Setup bộ chia clock cho APB1 và APB2:
Clock của APB1 và APB2 lấy từ Sysclk, với chú ý là APB1 có giới hạn là 36MHz (thực ra cũng có thể overclock). Ở đây Sysclk lấy từ PLL là 72MHz nên APB1 phải chia đôi, còn APB2 không có giới hạn như APB1 nên có thể chọn chia 1:

```
RCC->CFGR |= RCC_CFGR_PPRE1_DIV2 | RCC_CFGR_PPRE2_DIV1;
```

Tổng hợp lại ta có các hàm setup clock như sau:

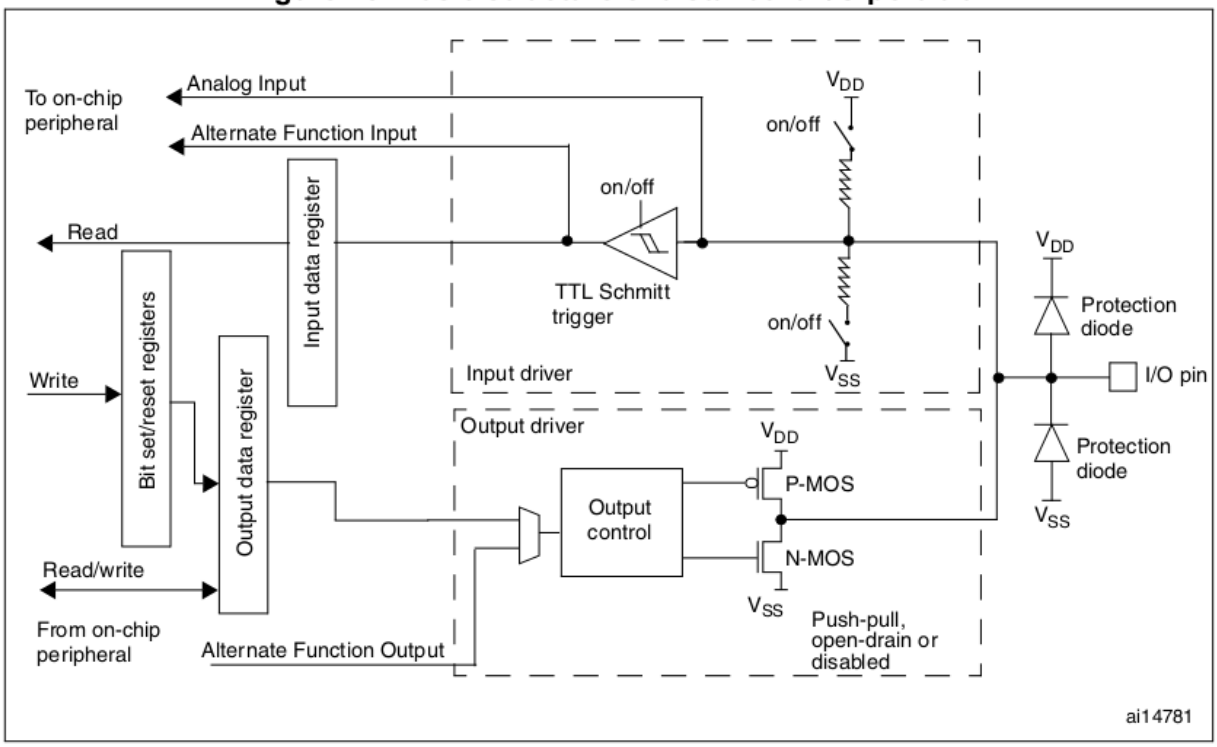
```
void SystemClock_Config(void)
{

  FLASH->ACR = (FLASH->ACR) | LL_FLASH_LATENCY_2;
  RCC->CR |= RCC_CR_HSEON;
  RCC->CFGR |= (RCC_CFGR_PLLSRC | 0x00000000U);// Setup PLL source without prescale
  RCC->CFGR |=  RCC_CFGR_PLLMULL9;            // Setup PLL multiplication factor
  RCC->CR |= RCC_CR_PLLON;
  RCC->CFGR |= RCC_CFGR_SW_PLL;
  RCC->CFGR |= RCC_CFGR_PPRE1_DIV2 | RCC_CFGR_PPRE2_DIV1;
  LL_Init1msTick(72000000);
  SysTick->CTRL |= LL_SYSTICK_CLKSOURCE_HCLK;
  LL_SetSystemCoreClock(72000000);
}
```

3 câu lệnh cuối là 3 câu lệnh tạo delay của thư viện Low Layer.

GPIO của STM32:

Cấu hình GPIO của STM32 khá là tiêu chuẩn.

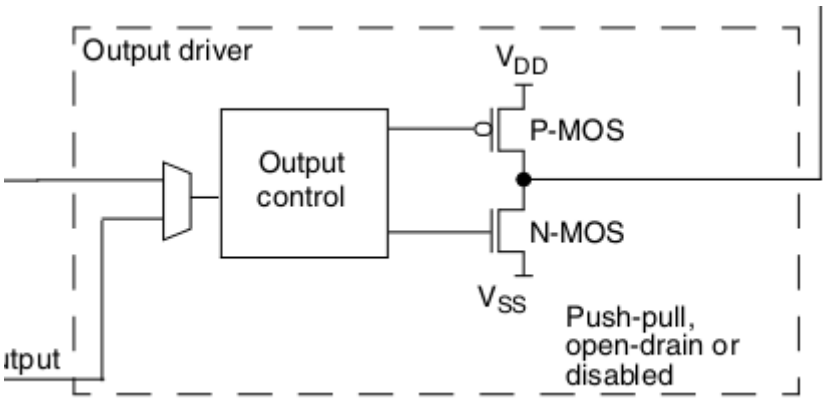
Figure 13. Basic structure of a standard I/O port bit



Việc đọc ghi được thực hiện qua các thanh ghi đọc ghi, thanh ghi output và thanh ghi BSRR
Các chế độ hoạt động của GPIO:

- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain
- Output push-pull
- Alternate function push-pull
- Alternate function open-drain

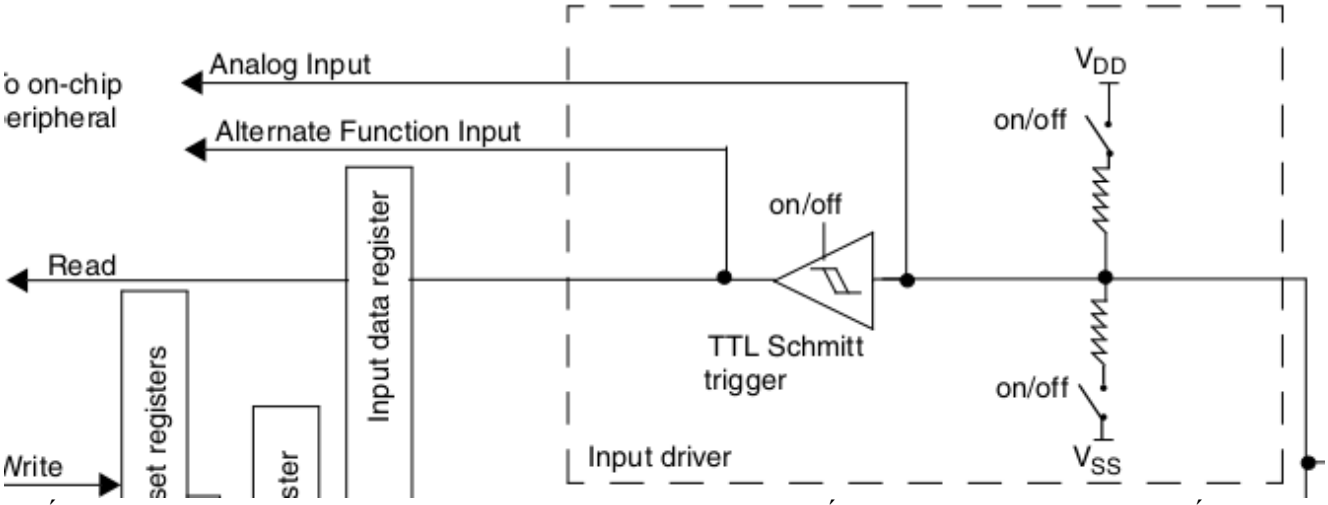
Về chức năng của từng khối, với bộ output kéo đẩy (push-pull):



Khi output control ở mức thấp, P-MOS hoạt động nối output với VDD từ đó lên mức cao, N-MOS không hoạt động nên không nối với đất
Ngược lại thì khi output control ở mức cao thì chân ra nối VSS, do đó ở mức thấp.

Việc sử dụng mạch Push Pull này giúp cho output của GPIO không bị thả nổi “float”

Khối Smith Trigger:



Khối smith trigger này có tác dụng kích lên mức 1 khi điện áp đến một ngưỡng nào đó và xuống 0 khi điện áp thấp hơn một mức khác, từ đó ta có thể đọc được mức logic trên chân GPIO.

Các thanh ghi GPIO:

Tùy thuộc vào dòng chip mà sẽ có số lượng GPIO khác nhau, các GPIO này được ký hiệu bằng các kí tự như A, B, C,…. Khi nhắc đến GPIOx thì x là một trong các kí tự đó. Ta gọi một nhóm GPIO như vậy là một PORT (cổng)

1 PORT kiểm soát 16 chân GPIO.

Thanh ghi cấu hình GPIO: GPIOx_CRL

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Thanh ghi này có chức năng chọn lựa chức năng và tốc độ hoạt động của 8 chân GPIO từ 7→ 0 của một PORT GPIO bất kì.

2 bit CNFx [1:0] với x là số thứ tự chân GPIO thực hiện setup chức năng của chân đó
Các giá trị của CNF như sau:

In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Input Mode hay Output mode được set up bởi thanh ghi GPIOx_IDR

2 bit MODEx [1:0] với x là số thứ tự chân GPIO setup chân GPIO này là Output hoặc Input. Nếu giá trị khác 00 thì sẽ quy định tốc độ output của chân GPIO này.
. Tốc độ càng cao càng dễ nhiễu hơn.

MODEy[1:0]: Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Khi được set ở chế độ Input, GPIO sẽ có tính chất sau:

When the I/O Port is programmed as Input:
- The Output Buffer is disabled
- The Schmitt Trigger Input is activated
- The weak pull-up and pull-down resistors are activated or not depending on input configuration (pull-up, pull-down or floating):
- The data present on the I/O pin is sampled into the Input Data Register every APB2 clock cycle
- A read access to the Input Data Register obtains the I/O State.

Các tính chất khi được set ở chế độ Output:

9.1.8 Output configuration

When the I/O Port is programmed as Output:
- The Output Buffer is enabled:
  - Open Drain Mode: A “0” in the Output register activates the N-MOS while a “1” in the Output register leaves the port in Hi-Z (the P-MOS is never activated)
  - Push-Pull Mode: A “0” in the Output register activates the N-MOS while a “1” in the Output register activates the P-MOS
- The Schmitt Trigger Input is activated.
- The weak pull-up and pull-down resistors are disabled.
- The data present on the I/O pin is sampled into the Input Data Register every APB2 clock cycle
- A read access to the Input Data Register gets the I/O state in open drain mode
- A read access to the Output Data register gets the last written value in Push-Pull mode

Thanh ghi GPIOx_CRH: Tương tự thanh ghi trên nhưng cho bit từ 15→8

9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Thanh ghi GPIOx_IDR: Data đọc được ghi ở đây

9.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Address offset: 0x08h

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy:** Port input data (y= 0 .. 15)

These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

Để điều khiển chân output GPIO, ta có 2 cách là setup thanh ghi GPIOx_ODR (GPIO output data register), thanh ghi này kiểm soát output của chân GPIO có đang là kéo cao không hoặc GPIOx_BSRR (GPIO Bit set reset register) là thanh ghi kiểm soát cả việc reset hay không của chân GPIO.

9.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

Ví dụ setup chân GPIO:

Ví dụ ta setup chân PC13 trên kit STM32F103 bluepill.
Để chạy GPIO, đầu tiên ta cần cấp xung vào cho PORTC:

PORTC này được kiểm soát bởi APB2, cụ thể là thanh ghi APB2ENR, nên ta bật clock cho ngoại vi này

```
RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;
```

Cũng như phần clock, ta setup thanh ghi GPIOC_CRH sử dụng các bit đã được define sẵn, ta hoàn toàn cũng có thể dùng "magic hex" để set thanh ghi, nhưng mà thế thì chả ai đọc nổi code :>
Các số định nghĩa của thanh ghi này như sau:

Ví dụ ở đây là các define cho chân 13.
Trong đó GPIO_CRH_MODE13 có giá trị MODE[1:0] là 3 tức 11 nhị phân, sẽ là speed 50MHz.
GPIO_CRH_MODE13_0 có giá trị MODE[1:0] là 1 tức 01, sẽ là speed 10MHz.
GPIO_CRH_MODE13_1 có giá trị MODE[1:0] là 2 tức 10, sẽ là speed 2MHz.

```
#define GPIO_CRH_MODE13_Pos                 (20U)
#define GPIO_CRH_MODE13_Msk                 (0x3UL << GPIO_CRH_MODE13_Pos)     /*!< 0x00300000 */
#define GPIO_CRH_MODE13                     GPIO_CRH_MODE13_Msk                /*!< MODE13[1:0] bits (Port x mode bits, pin 13) */
#define GPIO_CRH_MODE13_0                   (0x1UL << GPIO_CRH_MODE13_Pos)     /*!< 0x00100000 */
#define GPIO_CRH_MODE13_1                   (0x2UL << GPIO_CRH_MODE13_Pos)     /*!< 0x00200000 */
```

Nếu không set bit này thì coi như ở chế độ Input nên không cần định nghĩa :3
Tương tự với các bit CNF

```
#define GPIO_CRH_CNF13_Pos                  (22U)
#define GPIO_CRH_CNF13_Msk                  (0x3UL << GPIO_CRH_CNF13_Pos)      /*!< 0x00C00000 */
#define GPIO_CRH_CNF13                      GPIO_CRH_CNF13_Msk                 /*!< CNF13[1:0] bits (Port x configuration bits, pin 13) */
#define GPIO_CRH_CNF13_0                    (0x1UL << GPIO_CRH_CNF13_Pos)      /*!< 0x00400000 */
#define GPIO_CRH_CNF13_1                    (0x2UL << GPIO_CRH_CNF13_Pos)      /*!< 0x00800000 */
```

Trong đó CNF13 có giá trị 3, CNF13_0 là 4, tương đương 2 bit CNF là 2,… với bài này ta sẽ set General push-pull là 0 nên không cần set bit này
Vậy đây là câu lệnh setup của chân 13 (Mode 10Mhz):

```
GPIOC->CRH |= GPIO_CRH_MODE13_0;
```

So sánh với code setup do HAL sinh ra:

```
/*Configure GPIO pin : PC13 */
GPIO_InitStruct.Pin = GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

Dĩ nhiên ngắn gọn của dòng setup bằng thanh ghi đi kèm với hiểu biết về hệ thống, sinh code tất nhiên là dễ (và chậm hơn).

Vậy hàm setup GPIO sẽ như sau:

```
static void MX_GPIO_Init(void)
{
  /* GPIO Ports Clock Enable */
  RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;
  GPIOC->CRH |= GPIO_CRH_MODE13_0;
}
```

Để set bit thanh ghi PC13, ta có thể sử dụng 2 phương pháp sau:

Dùng ODR:

Để thay đổi Output lên cao ta set bit của thanh ghi GPIOC_ODR lên cao như sau:

```
GPIOC->ODR |= GPIO_ODR_ODR13;
```

Để set bit này xuống thấp:

```
GPIOC->ODR &= ~GPIO_ODR_ODR13;
```

Ta thực hiện phép đảo bit thanh ghi lại và thực hiện phép AND để set bit này trở lại thành 0.

Với thanh ghi BSRR thì ta sẽ set bit set hoặc reset chứ không cần phải đảo bit:
Đây là quá trình set bit

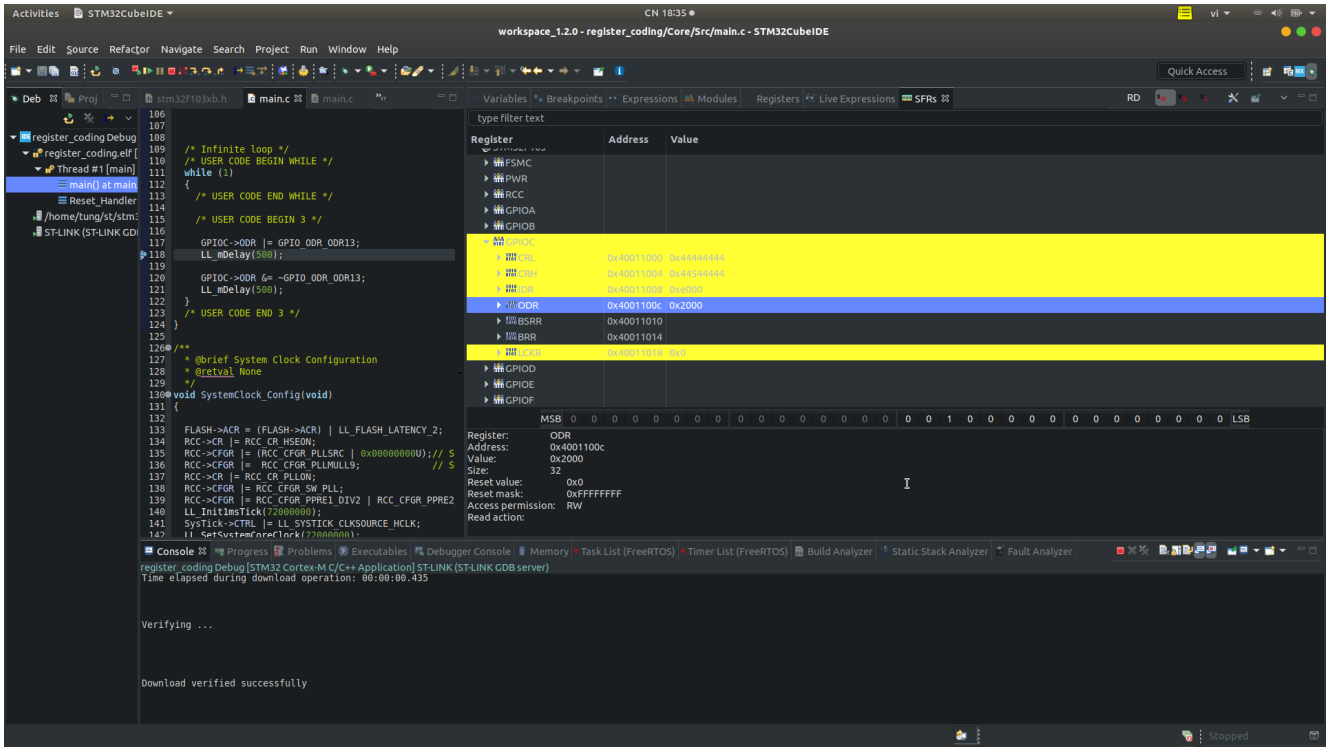
```
GPIOC->BSRR |= GPIO_BSRR_BS13;
```

Còn về quá trình reset bit:

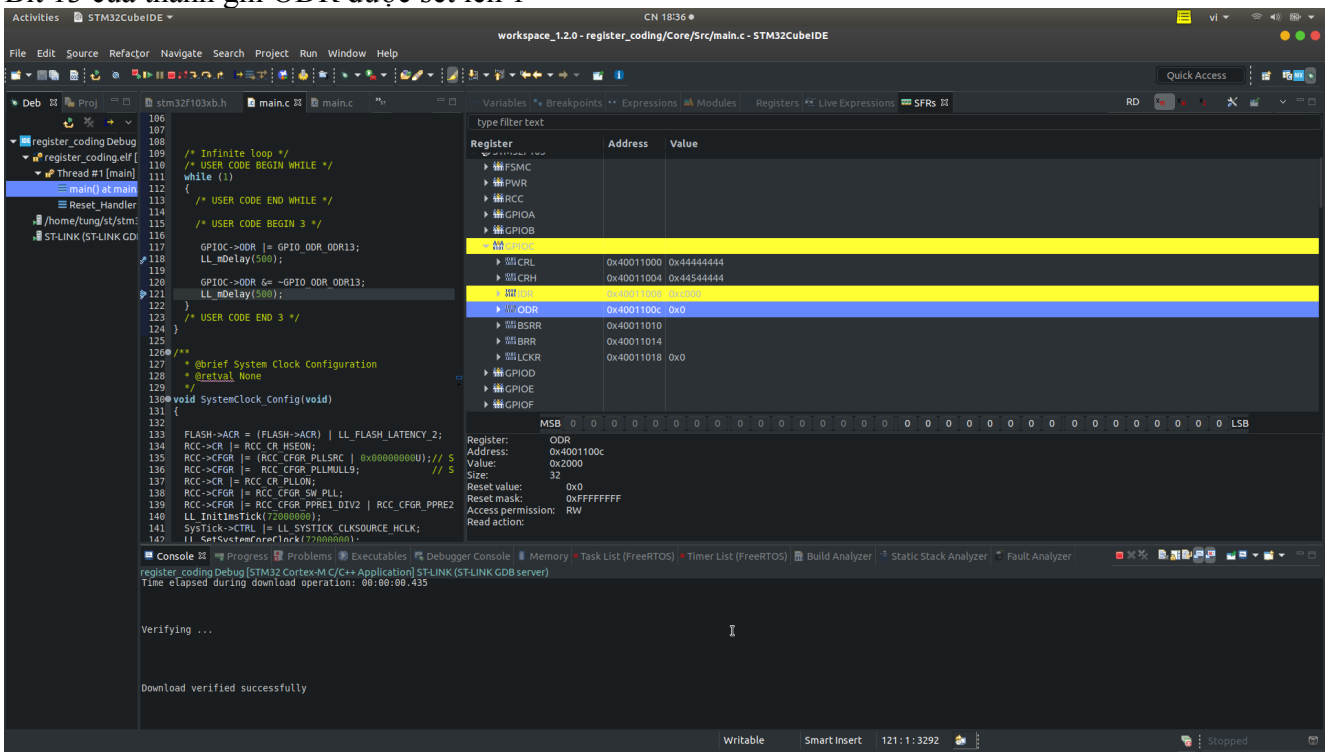
```
GPIOC->BSRR |= GPIO_BSRR_BR13;
```

Để theo dõi các bit SFR này thực hiện, ta có thể dùng công cụ SFR của STM32CubeIDE:

Ví dụ như đây là sự thay đổi của thanh ghi ODR:

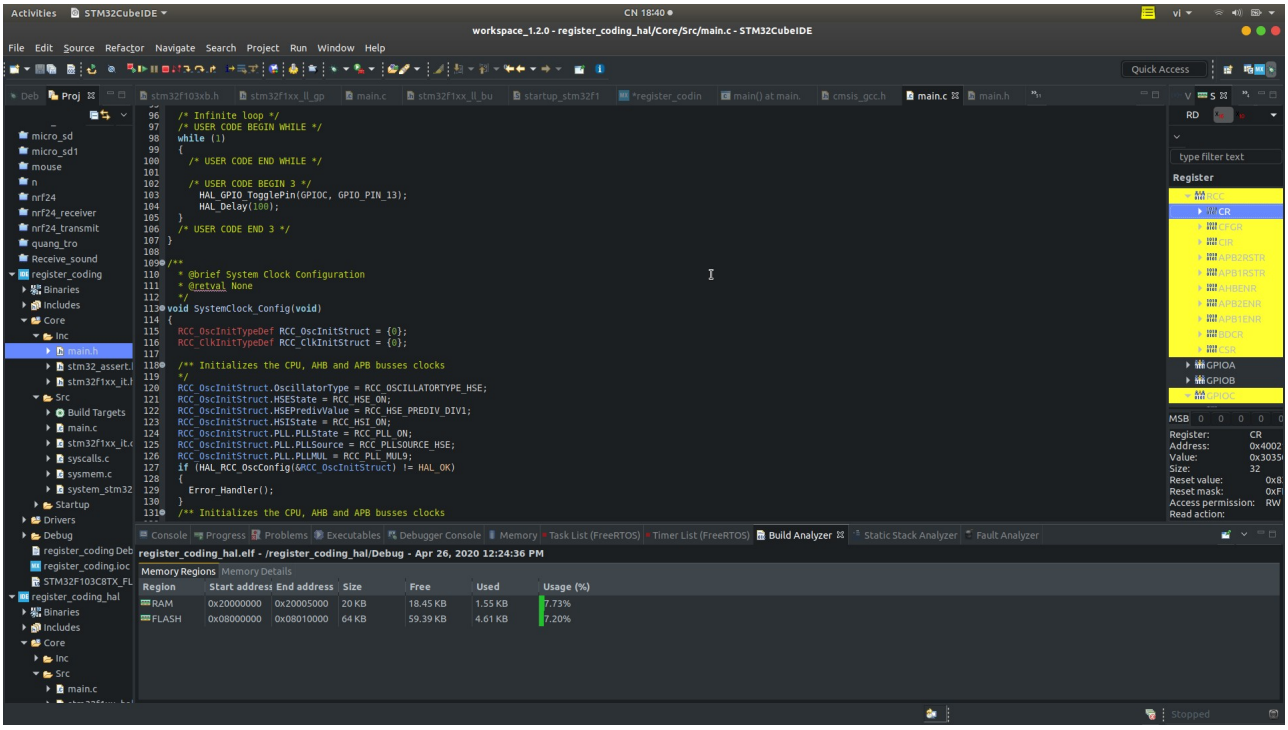


Bit 13 của thanh ghi ODR được set lên 1

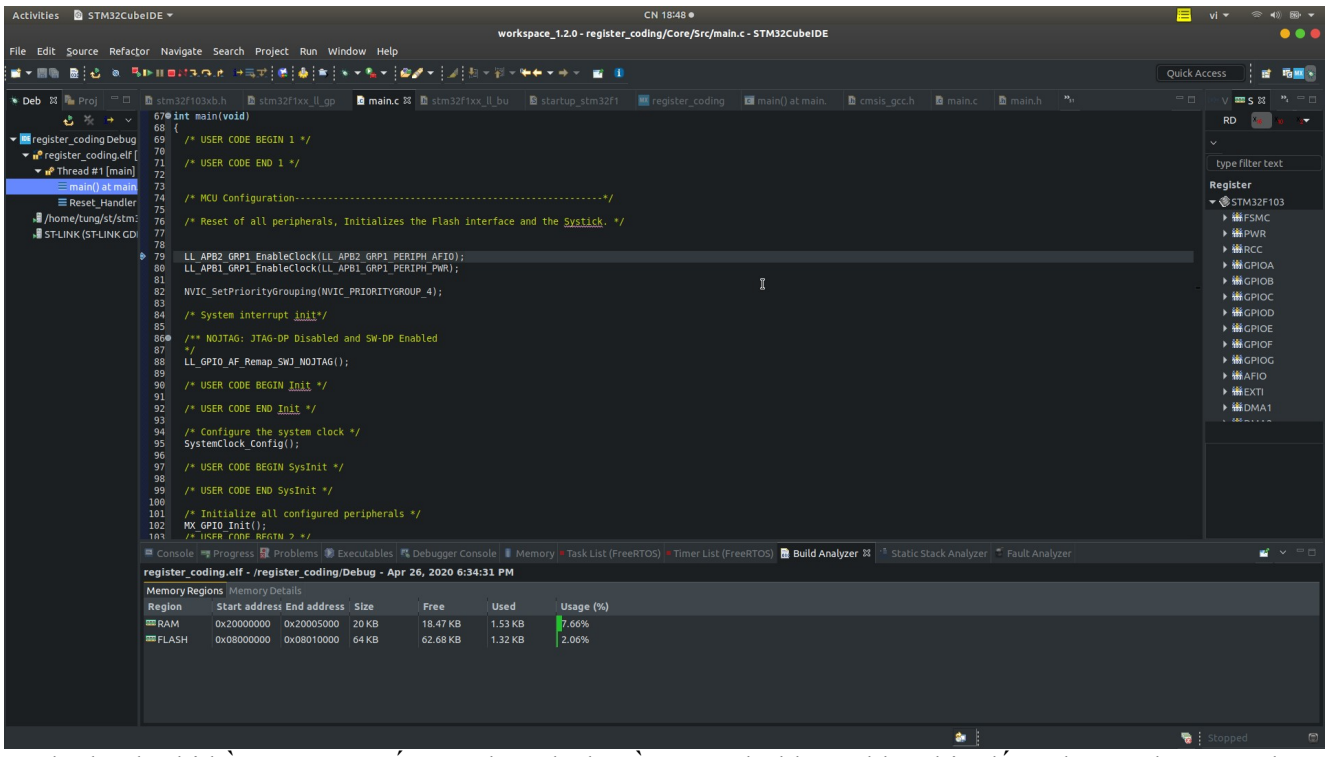


Bit 13 của thanh ghi ODR được set xuống 0.

Vậy là cuối cùng ta đã có thể tự tin làm nháy led cho STM32 bằng thanh ghi, vậy lý do gì ta phải code thanh ghi. Để trả lời câu hỏi này ta chỉ cần thực hiện một tác vụ nháy led như vậy bằng HAL:



Code HAL tốn 4.61kB Flash



Code thanh ghi bằng CMSIS tốn 1.32kB Flash Về RAM thì không khác biệt lắm, dù sao thì cũng chỉ là so sánh 2 tác vụ nháy led nên khác biệt về RAM không thể hiện được, tuy nhiên ta đã tiết kiệm được 4.61/1.32 = 3.4924242242 lần bộ nhớ FLASH, về tốc độ có lẽ khó so sánh nhưng code thanh ghi khá ngắn so với HAL nên có lẽ là nhanh hơn chút. Trong thời đại bộ nhớ rẻ như cho, các hãng tăng tính năng chip thì vài kB có thể không lớn, nhưng trước đây con người lên mặt trăng được bằng 72kBs ROM thì việc tối ưu code một chút có lẽ cũng không tệ

Tham khảo

STM32 Reference Manual: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

https://electronics.stackexchange.com/questions/125136/whats-the-concept-of-these-three-options-for-configuring-the-memory-of-stm32

https://tapit.vn/stm32f411-tim-hieu-cau-truc-va-lap-trinh-nhap-xuat-gpio-co-ban/

https://emsyfs.blogspot.com/2016/06/stm32-microcontroller-2-setting-clock.html

thtp://embedded-lab.com/blog/stm32-internals/

Kiến trúc Harvard của ARM Cortex M3: https://www.quora.com/Are-the-ARM-Cortex-Mx-processors-Harvard-or-Von-Neumann-architecture

Memory model của chip Cortex M3: https://developer.arm.com/docs/dui0552/a/the-cortex-m3-processor/memory-model

Mở rộng thêm về code thanh ghi với STM32:
https://www.vishnumaiea.in/articles/electronics/understanding-stm32-arm-microcontroller-gpios

Cấu hình Apollo Guidance Computer:
https://www.realclearscience.com/articles/2019/07/02/your_mobile_phone_vs_apollo_11s_guidance_computer_111026.html