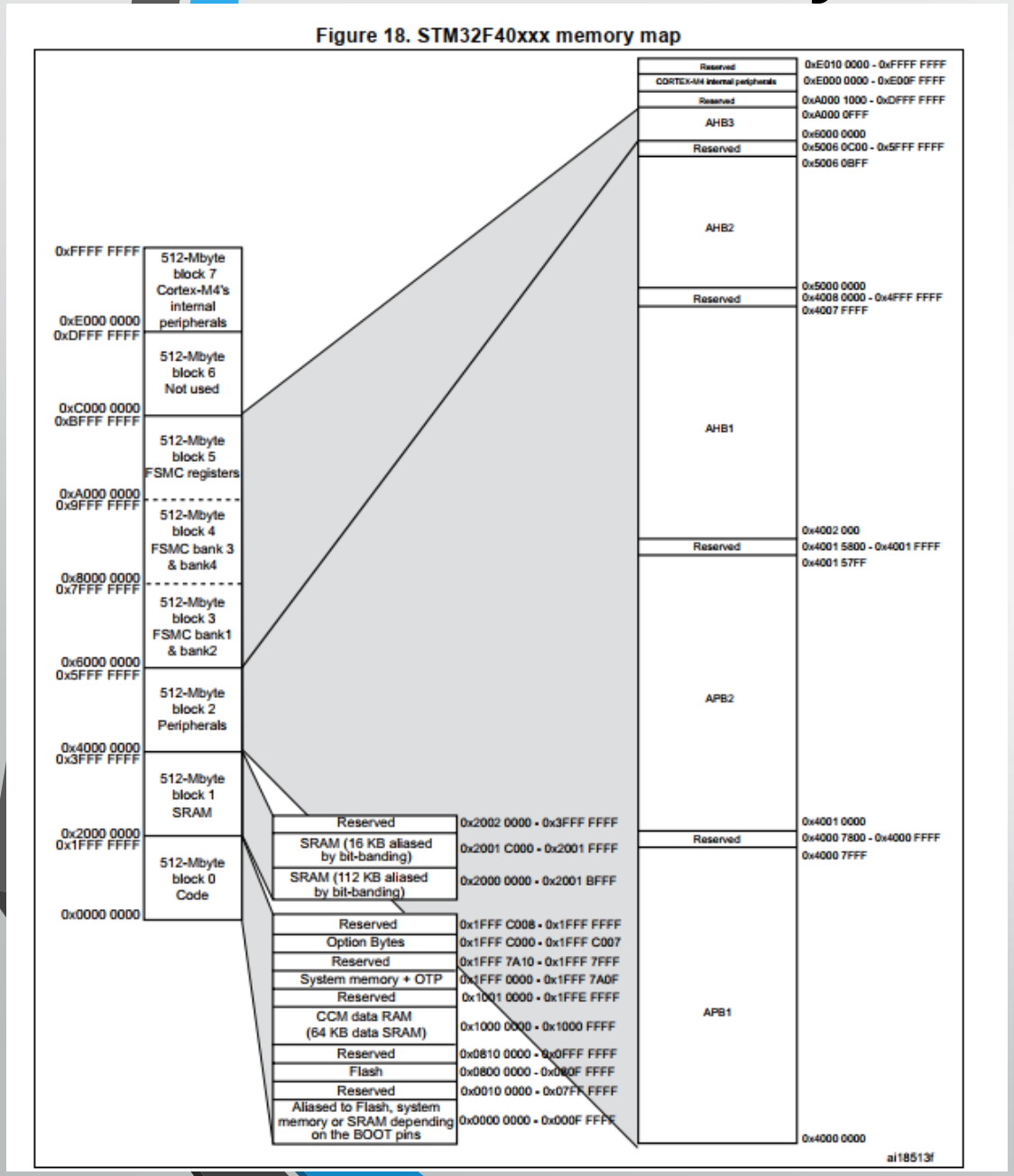
Tài liệu training STM32

07/2024 Tổng hợp các kiến thức cơ bản cần thiết

MỤC LỤC

1. Memory map trong STM32 MCU.



Với STM32F407ZET6 thuộc dòng chip ARM Cortex-M4.

Việc sở hữu system bus có độ rộng 32 bits:

- Vi điều khiển có thể xử lý dữ liệu 32-bit trong một chu kỳ bus. Điều này cho phép xử lý nhanh hơn so với bus có độ rộng nhỏ hơn
- Vi điều khiển có thể địa chỉ hóa đến 4GB bộ nhớ (2^32 = 4,294,967,296 địa chỉ). Ứng với địa chỉ bắt đầu từ 0x0000_0000 đến 0xFFFF_FFFF

1. Memory map trong STM32 MCU.

Table 10. register boundary addresses (continued)

Bus	Boundary address	Peripheral
AHB1	0x4004 0000 - 0x4007 FFFF	USB OTG HS
	0x4002 9400 - 0x4003 FFFF	Reserved
	0x4002 9000 - 0x4002 93FF	ETHERNET MAC
	0x4002 8C00 - 0x4002 8FFF	
	0x4002 8800 - 0x4002 8BFF	
	0x4002 8400 - 0x4002 87FF	
	0x4002 8000 - 0x4002 83FF	
	0x4002 6800 - 0x4002 7FFF	Reserved
	0x4002 6400 - 0x4002 67FF	DMA2
	0x4002 6000 - 0x4002 63FF	DMA1
	0x4002 5000 - 0x4002 5FFF	Reserved
	0x4002 4000 - 0x4002 4FFF	BKPSRAM
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0x4002 3400 - 0x4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2400 - 0x4002 2FFF	Reserved
	0x4002 2000 - 0x4002 23FF	GPIOI
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1800 - 0x4002 1BFF	GPIOG
	0x4002 1400 - 0x4002 17FF	GPIOF
	0x4002 1000 - 0x4002 13FF	GPIOE
	0x4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC

Với mỗi chức năng, mỗi ngoại vi được cung cấp địa chỉ để người sử dụng thao tác khác nhau.

Ta có thể tìm thấy bảng dữ liệu này trong tài liệu Datasheet của từng dòng chip cụ thể.

1. Memory map trong STM32 MCU.

Câu hỏi luyện tập: đối với STM32F407ZET6

1. Đâu là địa vùng địa chỉ của thanh ghi ngoại vi thuộc bus AHB1?
2. Đâu là địa chỉ cơ sở của các thanh ghi ngoại vi GPIOA?
3. Đâu là địa chỉ cơ sở của các thanh ghi RCC?
4. Đâu là địa chỉ cơ sở của thanh ghi ngoại vi thuộc bus APB1?
5. Đâu là địa chỉ cơ sở của Flash Memory?
6. Đâu là địa chỉ cơ sở của SRAM2?
7. Đâu là địa chỉ cơ sở của các thanh ghi ADC?

Câu hỏi luyện tập: đối với STM32F407ZET6

1. Đâu là địa vùng địa chỉ của thanh ghi ngoại vi thuộc bus AHB1?

    Địa chỉ bắt đầu: 0x4002 0000

    Địa chỉ kết thúc: 0x4007 FFFF

2. Đâu là địa chỉ cơ sở của các thanh ghi ngoại vi GPIOA?

    Địa chỉ: 0x4002 0000

3. Đâu là địa chỉ cơ sở của các thanh ghi RCC?

    Địa chỉ: 0x4002 3800

4. Đâu là địa chỉ cơ sở của thanh ghi ngoại vi thuộc bus APB1?

    Địa chỉ: 0x4000 0000

5. Đâu là địa chỉ cơ sở của Flash Memory?

    Địa chỉ: 0x0800 0000 – 0x080F FFFF

6. Đâu là địa chỉ cơ sở của SRAM2?

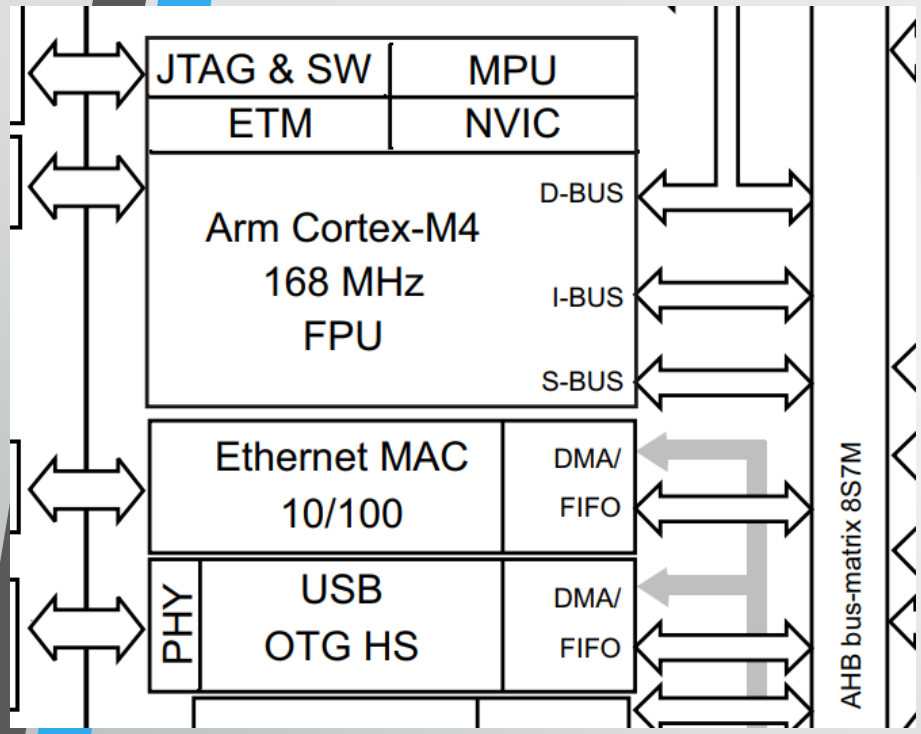
    SRAM1 bắt đầu từ 0x2000 0000 với kích thước SRAM1 là X Bytes

    => Địa chỉ cơ sở của SRAM2 = 0x2000 0000 + X

7. Đâu là địa chỉ cơ sở của các thanh ghi ADC?

    Địa chỉ: 0x4001 2000

2. MCU Bus trong STM32 MCU.



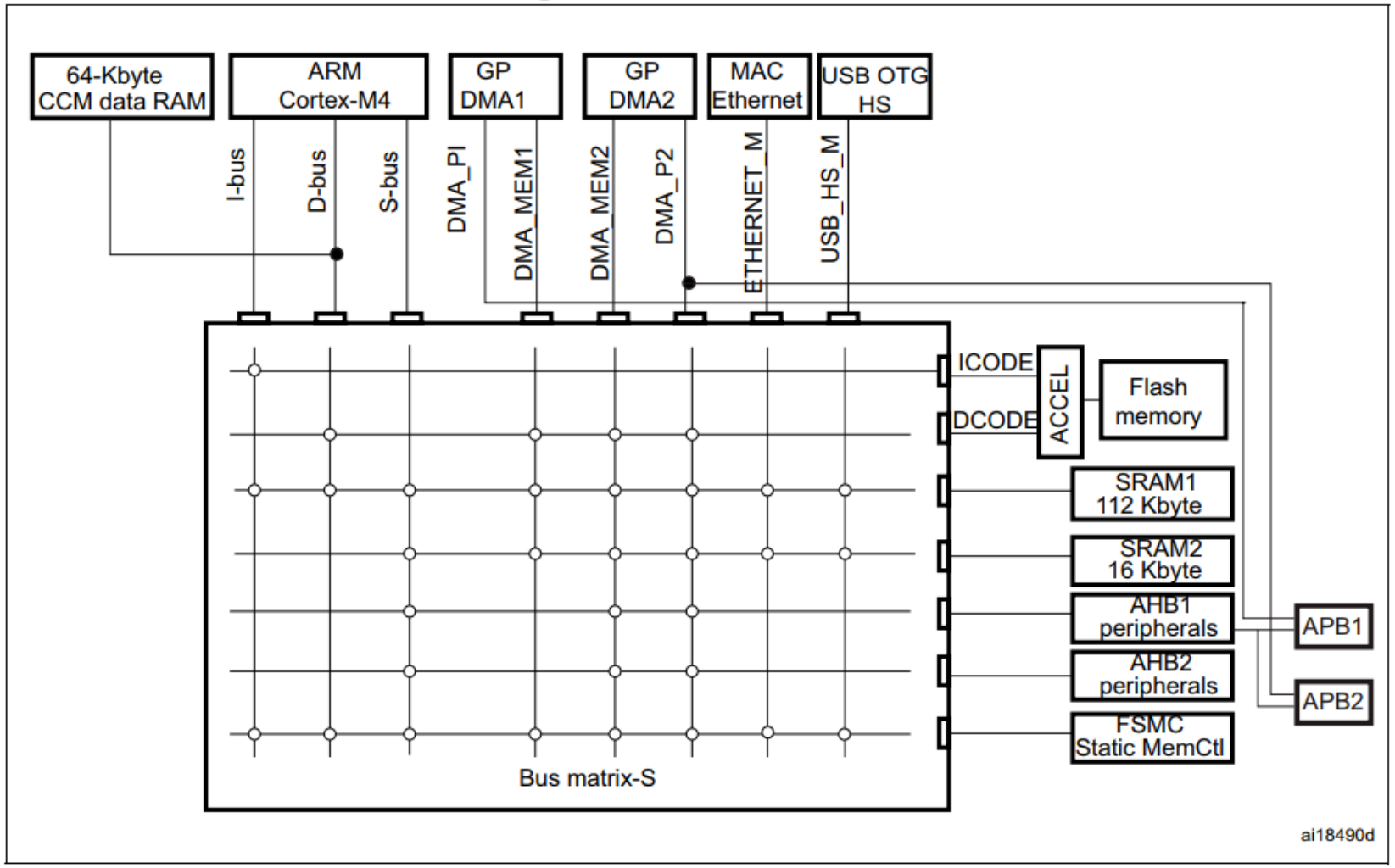
Lõi Arm Cortex-M4 giao tiếp với ngoại vi khác trong vi điều khiển thông qua các Bus.

Có 3 loại bus sau:

- I-Bus: Intruction bus: Bus lệnh được sử dụng để nạp lệnh từ bộ nhớ chương trình (Flash memory) đến bộ xử lý (CPU).
- D-Bus: Data bus: Bus dữ liệu được sử dụng để truyền dữ liệu giữa CPU và các bộ nhớ dữ liệu trên Flash hoặc các thiết bị ngoại vi khác.
- S-Bus: System bus: Bus hệ thống được sử dụng để truyền thông tin điều khiển và cấu hình giữa CPU và các thành phần hệ thống khác, như các thiết bị ngoại vi, SRAM, bộ điều khiển hệ thống, và các mô-đun khác.

2. MCU Bus trong STM32 MCU.

Figure 6. Multi-AHB matrix



3. MCU Clock tree

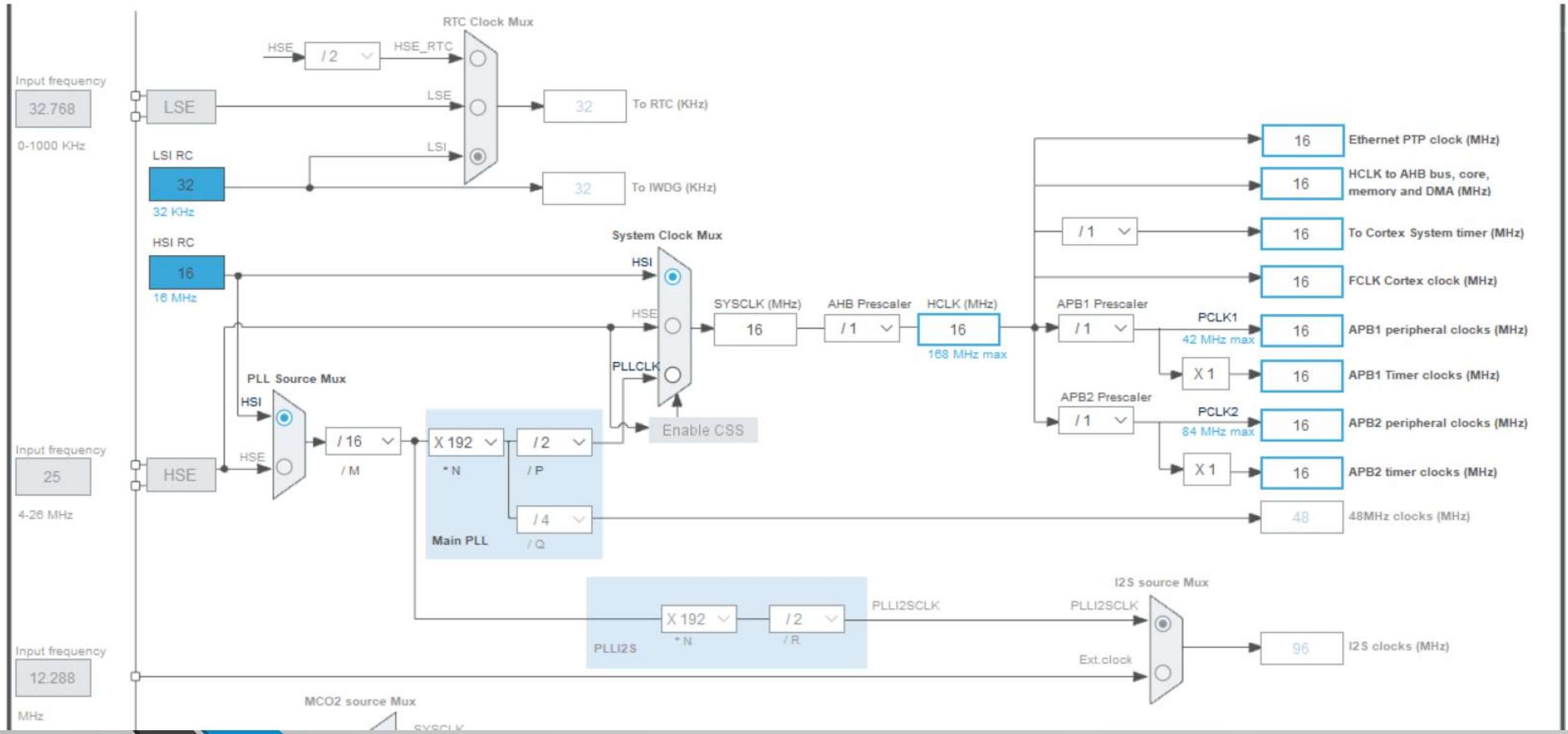
MCU Clocks là thành phần quan trọng, MCU sẽ không thể hoạt động nếu thiếu nó

RCC – Reset and Clock Control

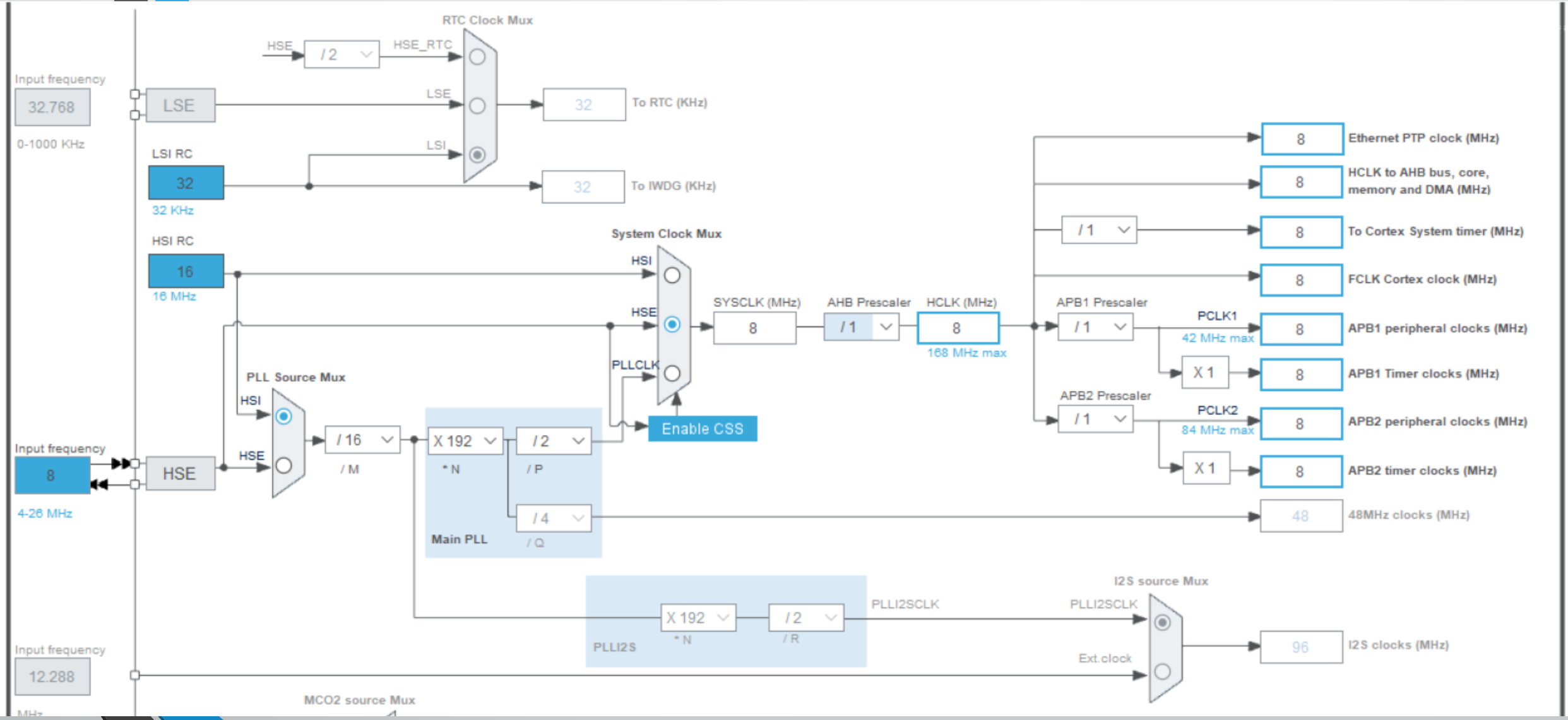
Clock: có 3 nguồn clock khác nhau được sử dụng cho hệ thống (Sysclk)

- HSI oscillator clock
- HSE oscillator clock
- Main PLL (PLL) clock

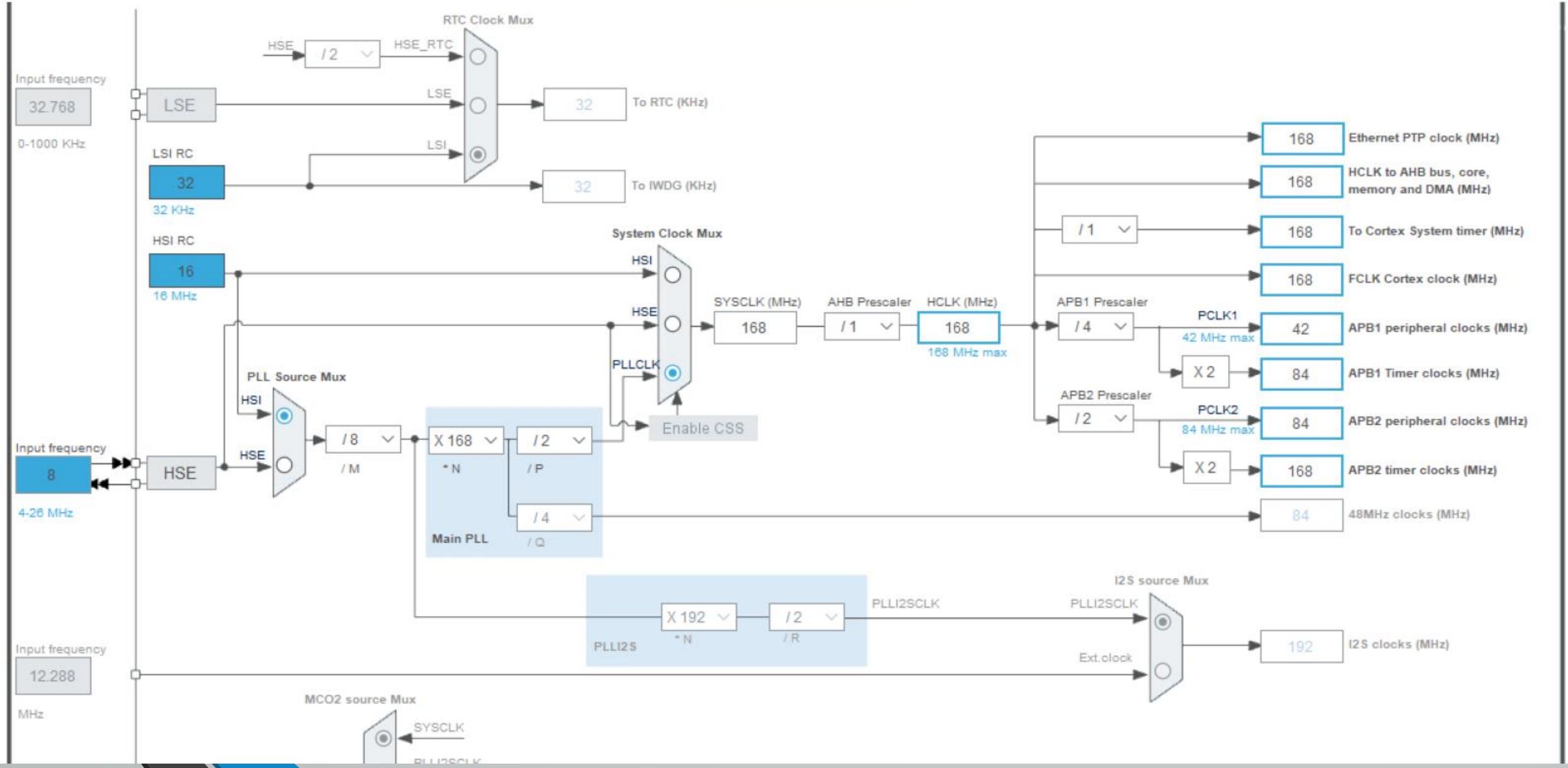
3. MCU Clock tree



3. MCU Clock tree



3. MCU Clock tree



4. Vector table và MCU interrupt design

Vector table là gì?

Vector là gì?

Vector khiến ta liên tưởng tới chiều, phương hướng.

Cụ thể hơn là con trỏ hay địa chỉ

⇒ Vector table là bảng chứa địa chỉ, chứa các pointers

⇒ Vector chứa địa chỉ của các hàm xử lý các ngoại lệ

4. Vector table và MCU interrupt design

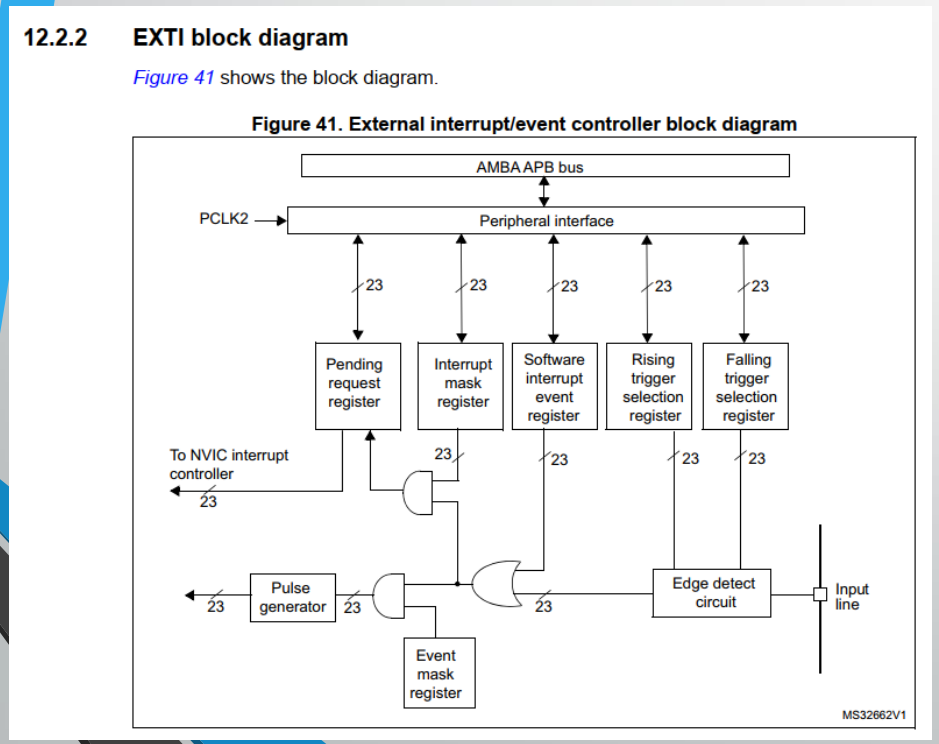
Table 62. Vector table for STM32F405xx/07xx and STM32F415xx/17xx

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-	-1	fixed	HardFault	All class of fault	0x0000 000C
-	0	settable	MemManage	Memory management	0x0000 0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C - 0x0000 002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000 002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000 0030

4. Vector table và MCU interrupt design

NVIC (Nested Vectored Interrupt Controller) là một thành phần quan trọng trong các vi điều khiển dòng STM32. NVIC quản lý và điều khiển các ngắt (interrupt) của vi điều khiển, cho phép xử lý ngắt một cách hiệu quả và ưu tiên các ngắt quan trọng hơn.

- Một số ngoại vi cho phép ngắt được quản lý trực tiếp bởi NVIC, một số khác như ngắt tại GPIOs cần quản lý qua EXTI trước khi vào NVIC.



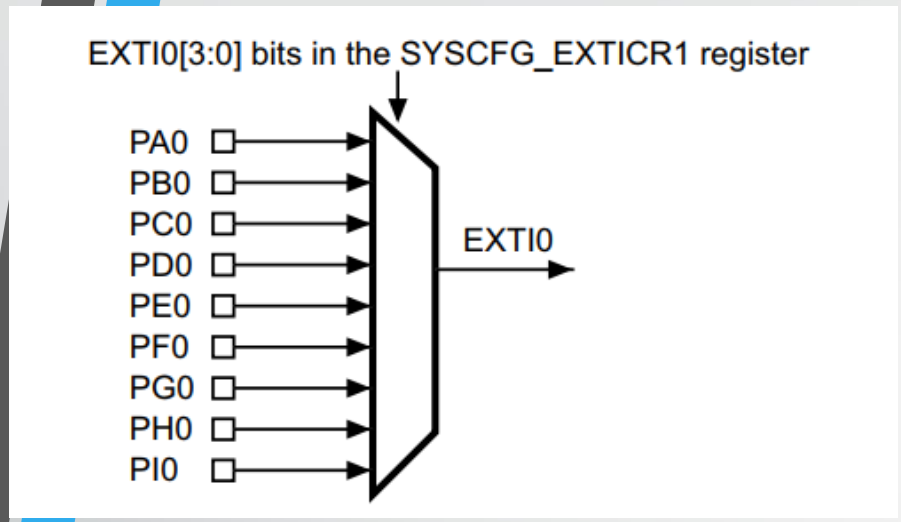
12.2.2 EXTI block diagram

Figure 41 shows the block diagram.

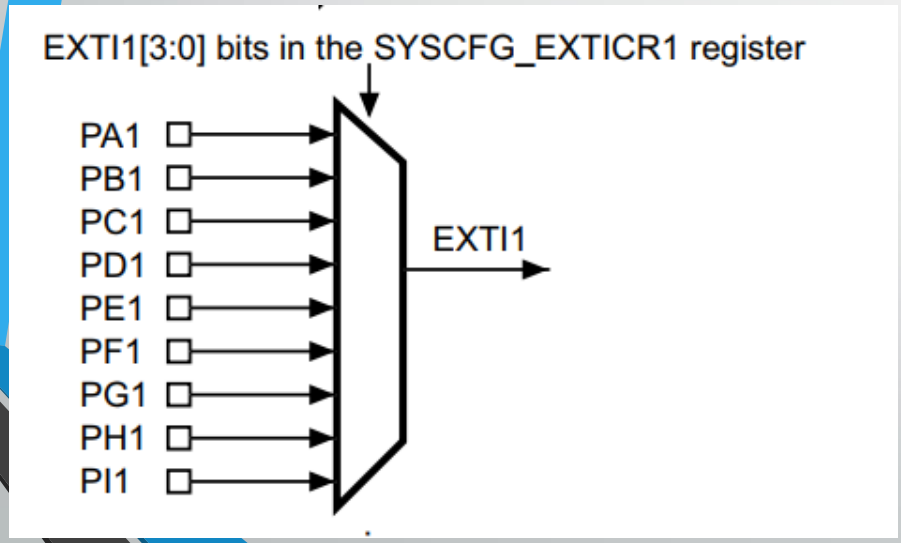
Figure 41. External interrupt/event controller block diagram

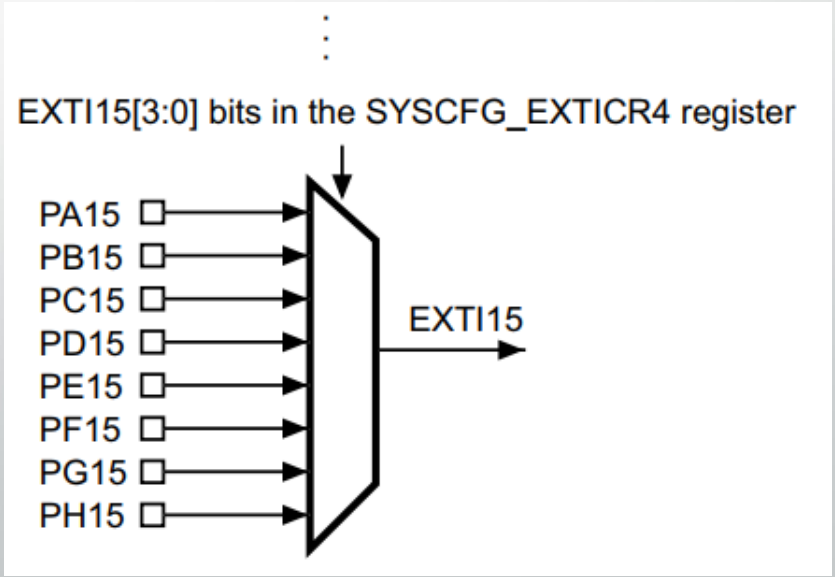
EXTI (External Interrupt/Event Controller) cho phép xử lý các ngắt (interrupts) hoặc sự kiện (events) từ các chân GPIOs (General Purpose Input/Output) bên ngoài.

4. Vector table và MCU interrupt design

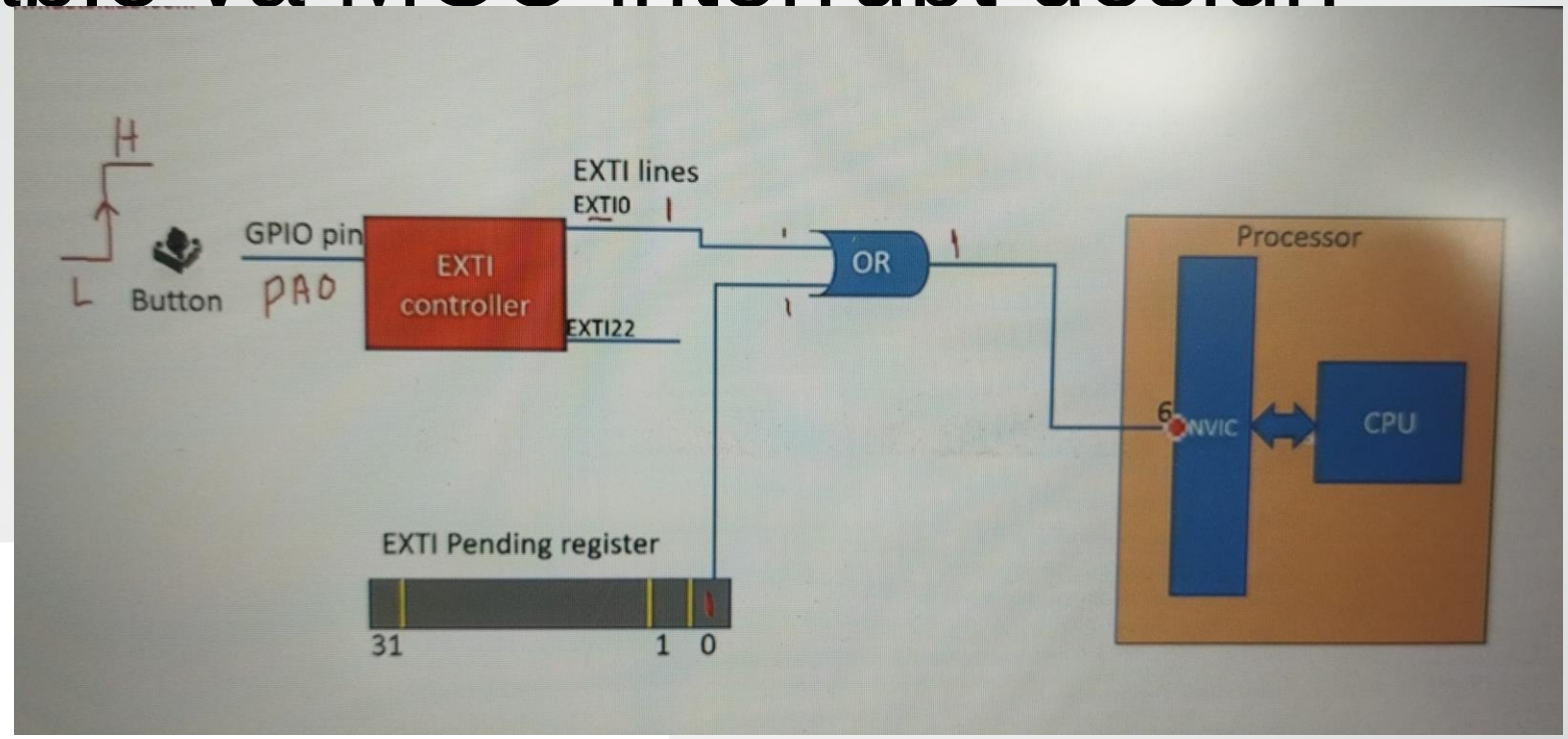


=> Tại 1 thời điểm chỉ có 1 xử lý ngắt In/Out được xử lý trên một trong các PA0 hoặc PB0 hoặc PC0,… Bởi chúng đều được quản lý trên 1 line EXTI0.





4. Vector table và MCU interrupt design



12.3.6 Pending register (EXTI_PR)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PR22	PR21	PR20	PR19	PR18	PR17	PR16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **PRx:** Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line.

This bit is cleared by programming it to ‘1’.

5. GPIOs

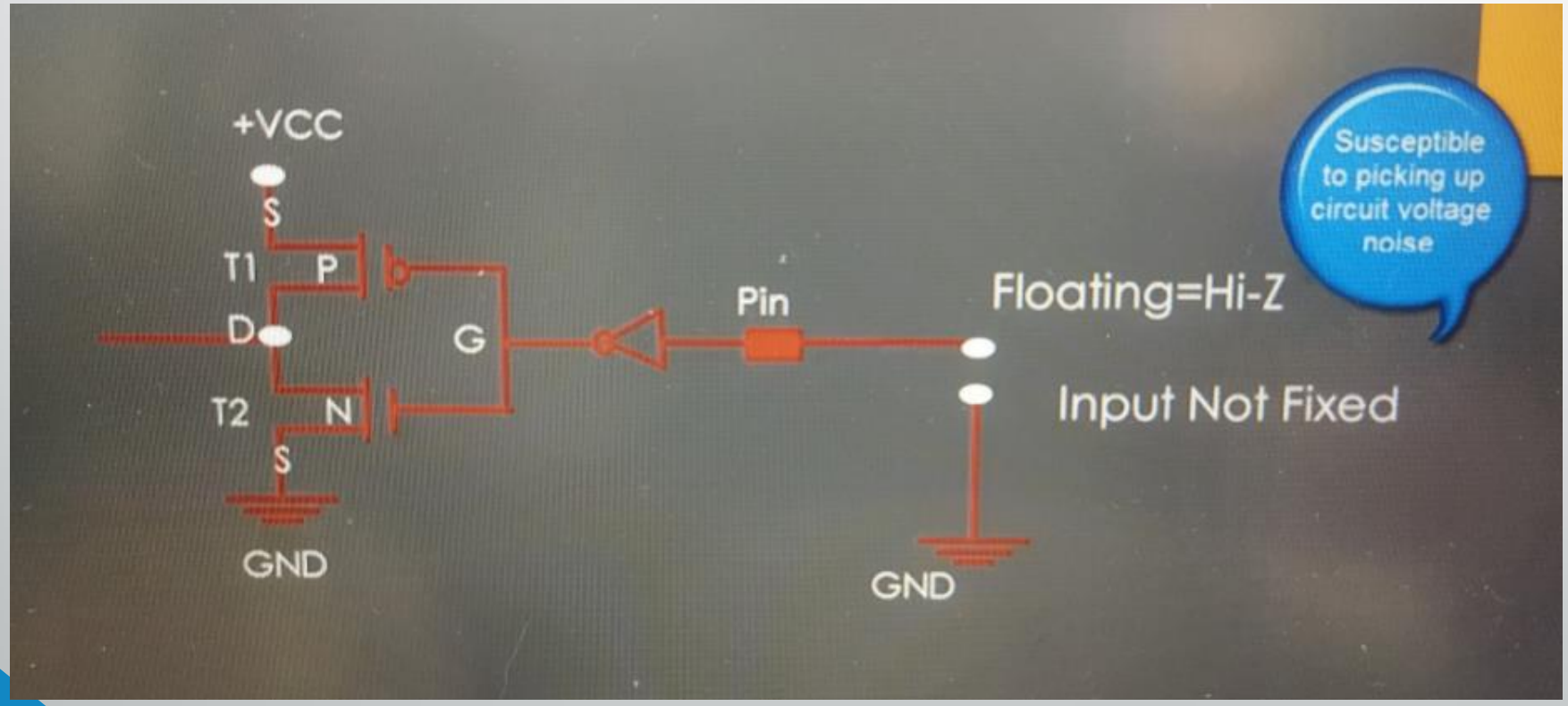
Tổng quan: STM32F407ZET6 có tổng cộng 8 port GPIO từ GPIOA đến GPIOH, mỗi port có thể có đến 16 chân, tổng cộng có thể lên tới 112 chân GPIO khả dụng.

Mỗi chân GPIO có thể được cấu hình để hoạt động ở các chế độ khác nhau:

- Input: Đầu vào (Input floating, Input pull-up, Input pull-down).
- Output: Đầu ra (Output push-pull, Output open-drain).
- Alternate Function (AF): Để sử dụng chân GPIO làm chức năng thay thế như UART, SPI, I2C, v.v.
- Analog: Để kết nối với các bộ chuyển đổi tín hiệu analog như ADC.

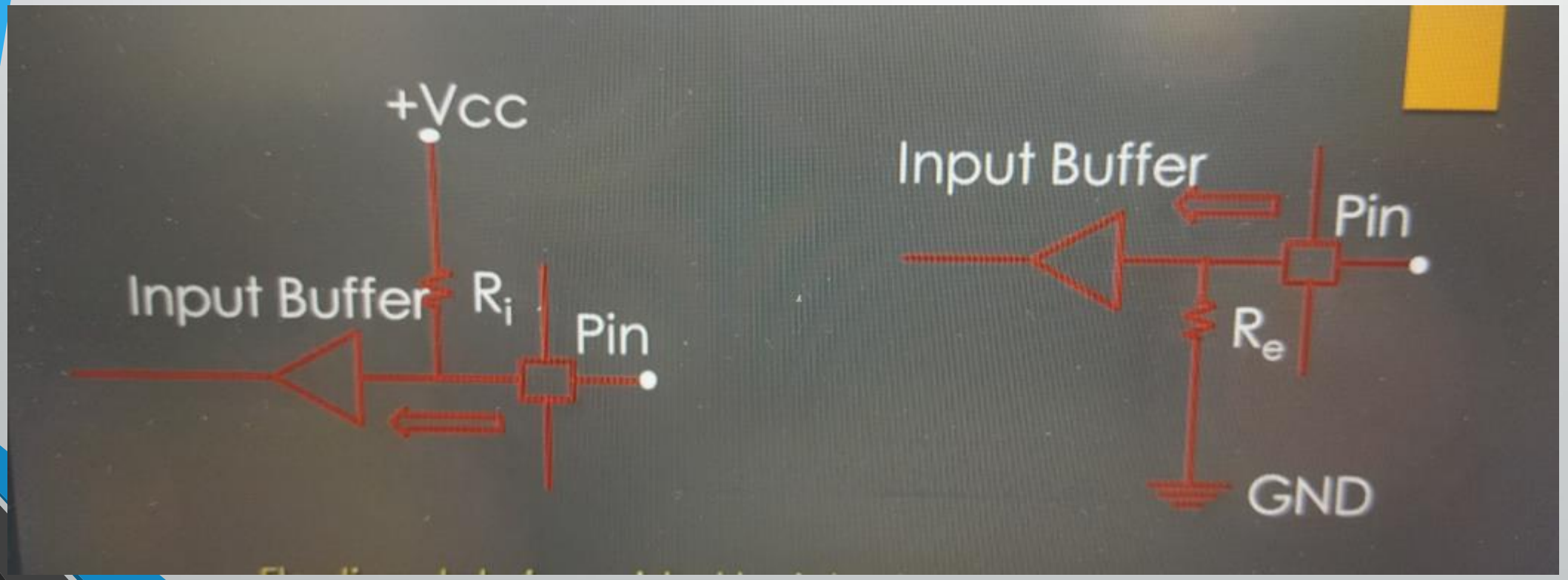
5. GPIOs

INPUT: tín hiệu đầu vào của GPIO bao gồm 3 trạng thái HIGH, LOW và FLOAT



5. GPIOs

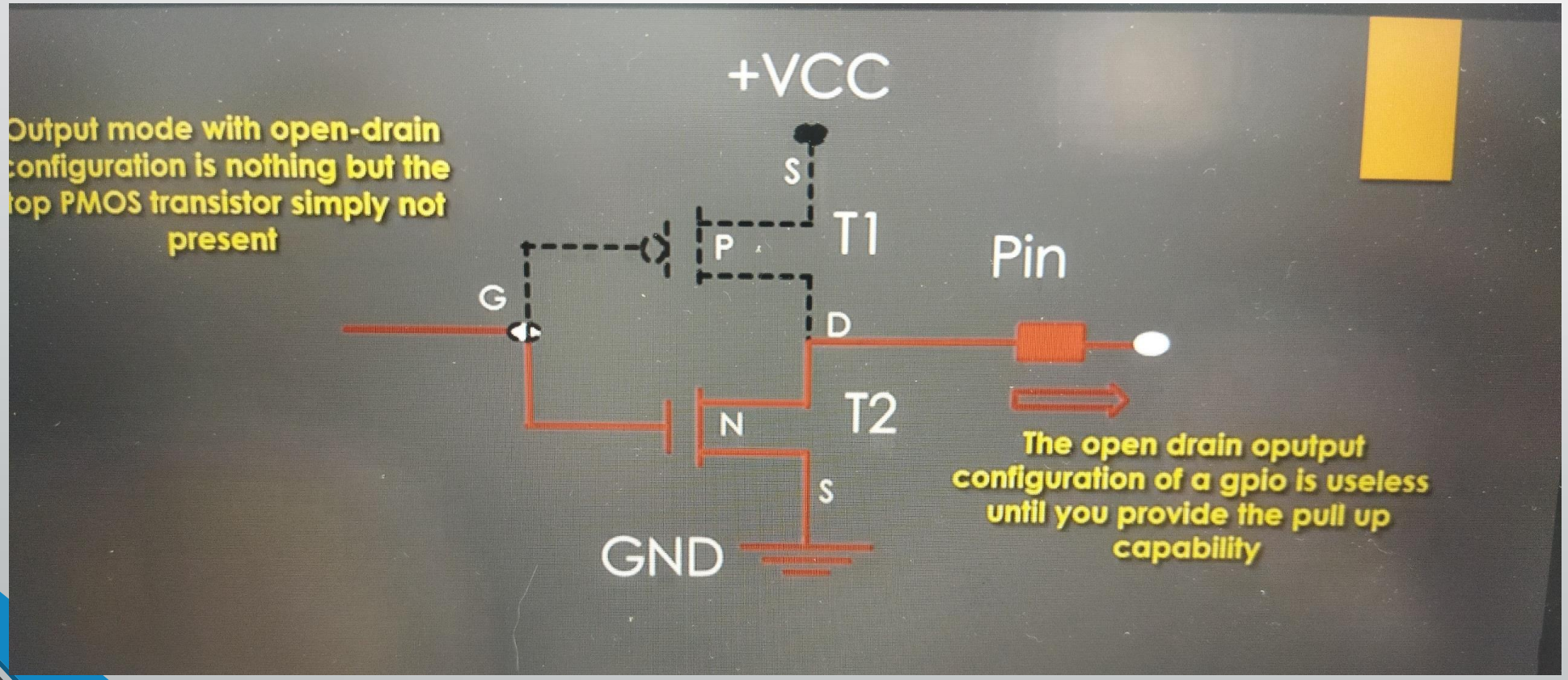
Để tránh trường hợp trạng thái bị thả nổi, người ta sẽ sử dụng trở kéo nội bên trong GPIOs hoặc dùng trở kéo ngoại bên ngoài



5. GPIOs

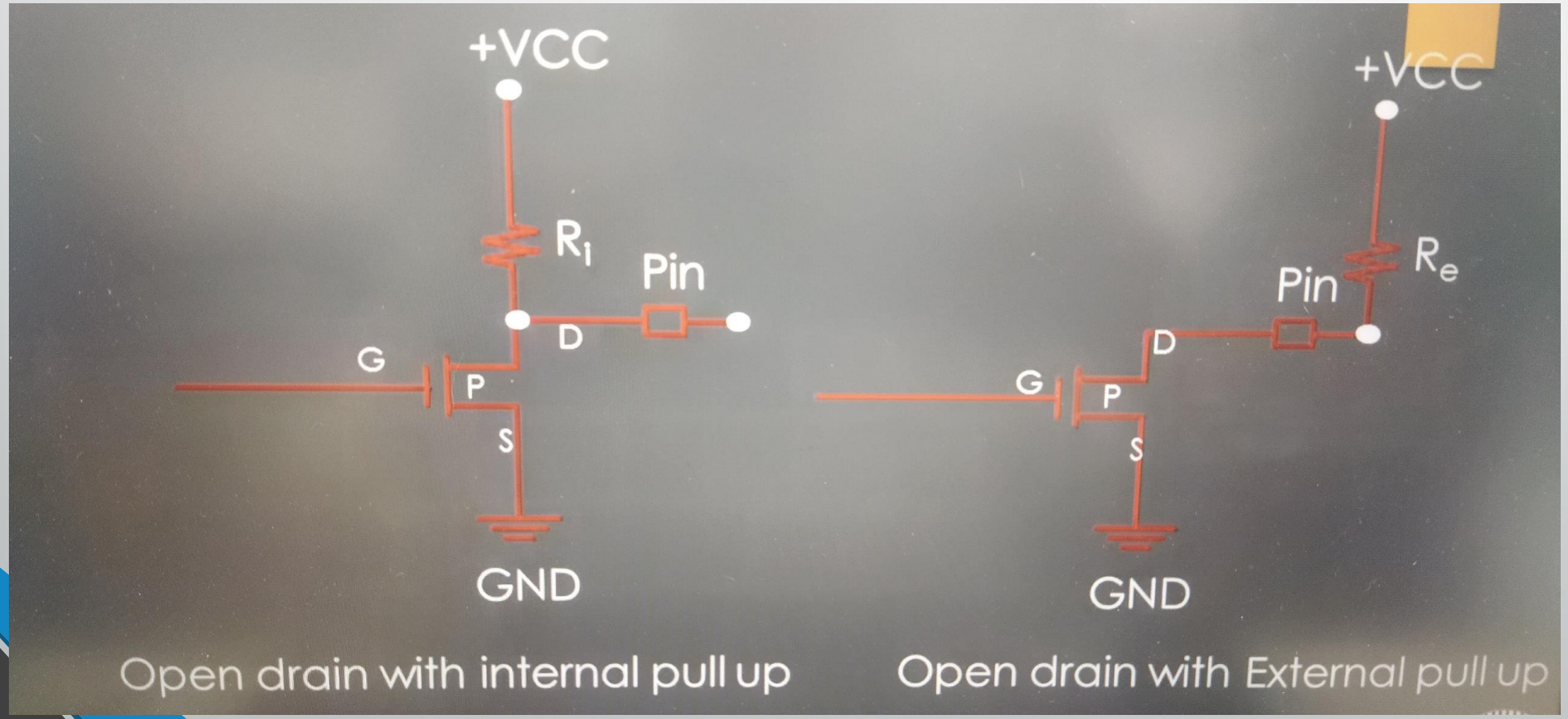
OUTPUT: Chia làm 2 chế độ cấu hình Open drain và Push-pull

Ở open-drain, GPIOs chỉ có thể kéo về LOW ở mức logic 0 và thả nổi (Float) ở mức logic 1

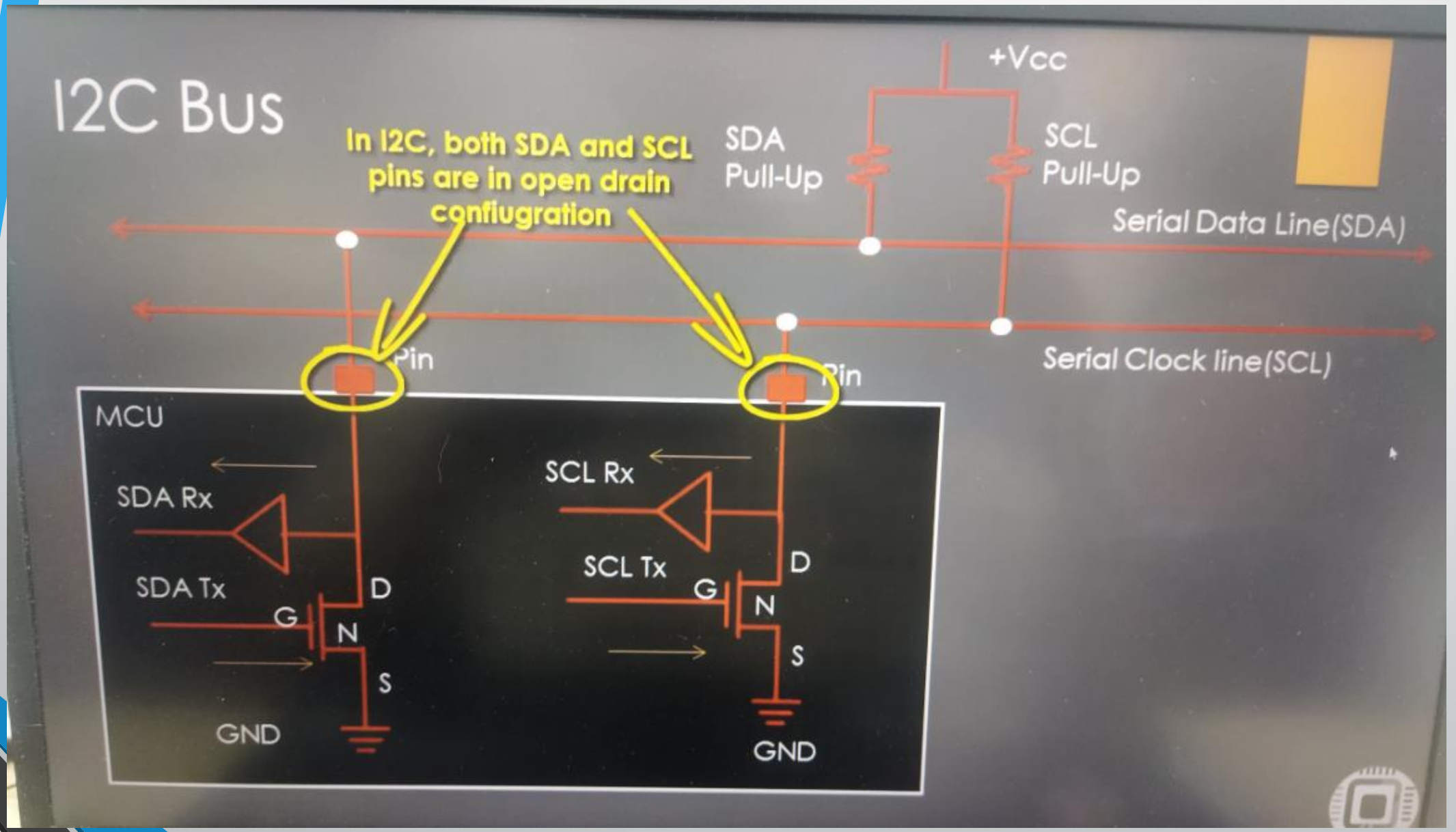


5. GPIOs

Ở chế độ Output open-drain, để tránh trạng thái output thả nổi, ta sẽ sử dụng trở kéo nội hoặc ngoại với đầu ra của GPIOs.

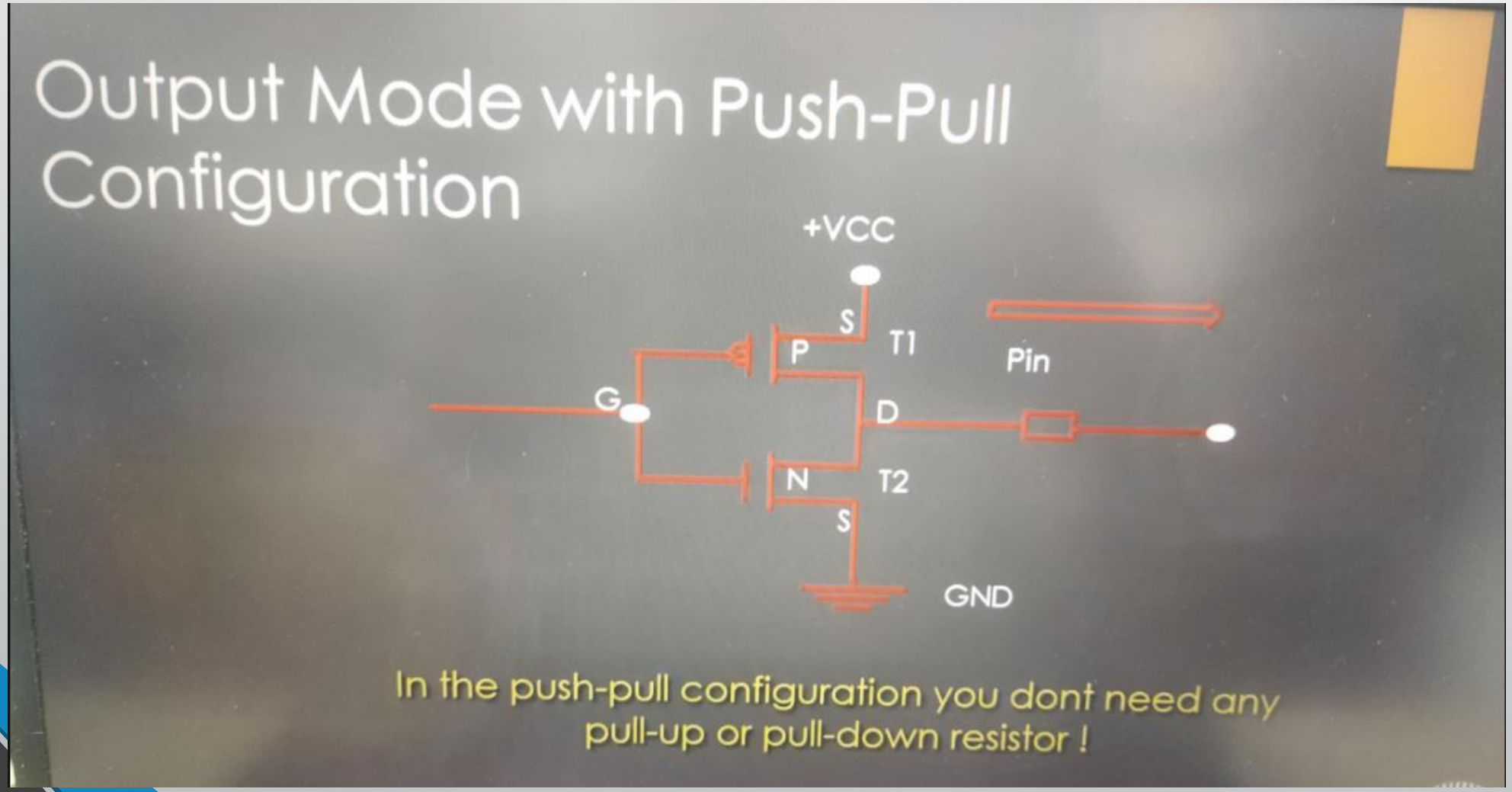


5. GPIOs


I2C Bus
In I2C, both SDA and SCL pins are in open drain configuration
+Vcc
SDA Pull-Up
SCL Pull-Up
Serial Data Line(SDA)
Serial Clock line(SCL)
Pin
Pin
MCU
SDA Rx
SDA Tx
G
D
N
S
GND
SCL Rx
SCL Tx
G
D
N
S
GND

5. GPIOs

Ở chế độ Output-Push Pull, ứng với mức logic 0 hay 1 thì điện áp đầu ra sẽ được nối đến GND hoặc VCC.



6. Lập trình điều khiển GPIO với thanh ghi

Một ví dụ về cách điều khiển chân GPIOA1 ở trạng thái output push-pull sử dụng thanh ghi trên STM32F407ZET6

```
// Định nghĩa địa chỉ thanh ghi cần thiết
#define RCC_AHB1ENR  (*(volatile uint32_t*)0x40023830)
#define GPIOA_MODER  (*(volatile uint32_t*)0x40020000)
#define GPIOA_OTYPER (*(volatile uint32_t*)0x40020004)
#define GPIOA_OSPEEDR (*(volatile uint32_t*)0x40020008)
#define GPIOA_PUPDR  (*(volatile uint32_t*)0x4002000C)
#define GPIOA_ODR    (*(volatile uint32_t*)0x40020014)
```

6. Lập trình điều khiển GPIO với thanh ghi

Một ví dụ về cách điều khiển chân GPIOA1 ở trạng thái output push-pull sử dụng thanh ghi trên STM32F407ZET6

```
int main(void) {
    // Bật clock cho GPIOA
    RCC_AHB1ENR |= (1 << 0);  // Bit 0 của thanh ghi RCC_AHB1ENR để bật clock cho GPIOA

    // Cấu hình PA1 làm chế độ output
    GPIOA_MODER &= ~(0x3 << (1 * 2));  // Clear bits 2 và 3 của PA1
    GPIOA_MODER |= (0x1 << (1 * 2));   // Set bits 2 và 3 của PA1 thành 01 (Output mode)

    // Cấu hình PA1 là push-pull
    GPIOA_OTYPER &= ~(0x1 << 1);  // Clear bit 1 của PA1 (Push-pull)

    // Cấu hình tốc độ cho PA1 (Low speed)
    GPIOA_OSPEEDR &= ~(0x3 << (1 * 2)); // Clear bits 2 và 3 của PA1
    GPIOA_OSPEEDR |= (0x1 << (1 * 2));  // Set bits 2 và 3 của PA1 thành 01 (Low speed)

    // Không sử dụng pull-up/pull-down cho PA1
    GPIOA_PUPDR &= ~(0x3 << (1 * 2));  // Clear bits 2 và 3 của PA1
```

6. Lập trình điều khiển GPIO với thanh ghi

Một ví dụ về cách điều khiển chân GPIOA1 ở trạng thái output push-pull sử dụng thanh ghi trên STM32F407ZET6

```
while (1) {
     // Bật PA1
    GPIOA_ODR |= (1 << 1);  // Set bit 1 của ODR (Output Data Register)
    for (int i = 0; i < 1000000; i++);  // Delay đơn giản
    // Tắt PA1
    GPIOA_ODR &= ~(1 << 1);  // Clear bit 1 của ODR
    for (int i = 0; i < 1000000; i++);  // Delay đơn giản
}
```

6. Lập trình điều khiển GPIO với thanh ghi

Một ví dụ về cách điều khiển chân GPIOA1 ở trạng thái output push-pull sử dụng thư viện SPL (standard peripheral Library) trên STM32F407ZET6

```
#include "stm32f4xx.h"  // Thư viện SPL cho STM32F4xx series
//Cấu hình chân GPIO
int main(void) {
        // 1. Bật clock cho GPIOA
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
        // 2. Cấu hình PA1 làm chế độ output
        GPIO_InitTypeDef GPIO_InitStructure;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;  // Chọn chân PA1
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  // Chế độ output
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;  // Tốc độ thấp, tốc độ 50 MHz
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  // Chế độ push-pull
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  // Không sử dụng pull-up/pull-down
        GPIO_Init(GPIOA, &GPIO_InitStructure);  // Áp dụng cấu hình
```

6. Lập trình điều khiển GPIO với thanh ghi

Một ví dụ về cách điều khiển chân GPIOA1 ở trạng thái output push-pull sử dụng thư viện SPL (standard peripheral Library) trên STM32F407ZET6

```
while (1) {
		// Bật PA1
		GPIO_SetBits(GPIOA, GPIO_Pin_1);  // Set bit PA1
		delay(1000000);  // Delay đơn giản
		// Tắt PA1	GPIO_ResetBits(GPIOA, GPIO_Pin_1);  // Clear bit PA1
		delay(1000000);  // Delay đơn giản
}
```

6. Lập trình ngoại vi với SPL

Thông thường việc cấu hình cho các ngoại vi sử dụng thư viện SPL sẽ theo một format chung

1. Khai báo các thư viện cần thiết.
2. Khai Báo Init struct và Các Hàm Thực Hiện

   // Khai báo cấu trúc

   GPIO InitGPIO_InitTypeDef GPIO_InitStructure;
3. Khởi tạo cấu hình cho ngoại vi

   Cấu hình RCC (bật clock)

   Gán thông số cho các trường thông tin trong struct init
4. Kích hoạt chức năng và sử dụng ngoại vi

   Gọi hàm GPIO_Init hoặc Cmd để kích hoạt chức năng

6. Lập trình ngoại vi với SPL

Thông thường việc cấu hình cho các ngoại vi sử dụng thư viện SPL sẽ theo một format chung

Ví dụ: Sử dụng Timer

c Copy code

```c
// Khai báo cấu trúc Timer Init
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

// Cấu hình Timer
void Timer_Configuration(void) {
    // Bật Clock cho Timer2
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    // Cấu hình Timer
    TIM_TimeBaseStructure.TIM_Period = 999;  // Thay đổi theo nhu cầu
    TIM_TimeBaseStructure.TIM_Prescaler = 83;  // Thay đổi theo nhu cầu
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);  // Gọi hàm Init để cấu hình Timer

    // Bật Timer
    TIM_Cmd(TIM2, ENABLE);
}
```

6. Lập trình ngoại vi với SPL

Thông thường việc cấu hình cho các ngoại vi sử dụng thư viện SPL sẽ theo một format chung

c Copy code

```c
// Khai báo cấu trúc UART Init
USART_InitTypeDef USART_InitStructure;

// Cấu hình UART
void UART_Configuration(void) {
    // Bật Clock cho UART2
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // Cấu hình UART
    USART_InitStructure.USART_BaudRate = 9600;  // Tốc độ baud rate
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;  // Độ dài dữ liệu
    USART_InitStructure.USART_StopBits = USART_StopBits_1;  // Bit dừng
    USART_InitStructure.USART_Parity = USART_Parity_No;  // Parity bit
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;  // Fl
    USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;  // Chế độ Tx và Rx
    USART_Init(USART2, &USART_InitStructure);  // Gọi hàm Init để cấu hình UART

    // Bật UART
    USART_Cmd(USART2, ENABLE);
}
```