TAPiT

KHÓA HỌC
LẬP TRÌNH VI ĐIỀU KHIỂN

Giảng viên
NGUYỄN HUỲNH NHẬT THƯƠNG

LỊCH HỌC:
Tại Đà Nẵng: 19h30 - 22h00 thứ 2 và thứ 6

ĐỊA ĐIỂM:
Online qua nền tảng Zoom/Google Meet

MODULE 2

- KHỐI GPIO - CHỨC NĂNG INPUT, OUTPUT TÍN HIỆU SỐ
- LẬP TRÌNH STM32: THANH GHI, THƯ VIỆN
- THỰC HÀNH, DEBUG
- PHÂN TÍCH DỰ ÁN THỰC TẾ

GPIO - GENERAL PURPOSE INPUT OUPUT

TAPiT

- Là các chân Input/ Ouput của vi điều khiển có thể được sử dụng với **nhiều mục đích khác nhau**
- Giúp vi điều khiển có thể giao tiếp với thế giới bên ngoài.

Các tài liệu cần sử dụng để tìm hiểu tính năng GPIO hay bất kỳ tính năng ngoại vi nào khác của vi điều khiển:

- Tài liệu Reference Manual: Hiểu chức năng.
- Tài liệu HAL Description: Hiểu thư viện, hiểu cách dùng
- Tài liệu Project mẫu: Dùng

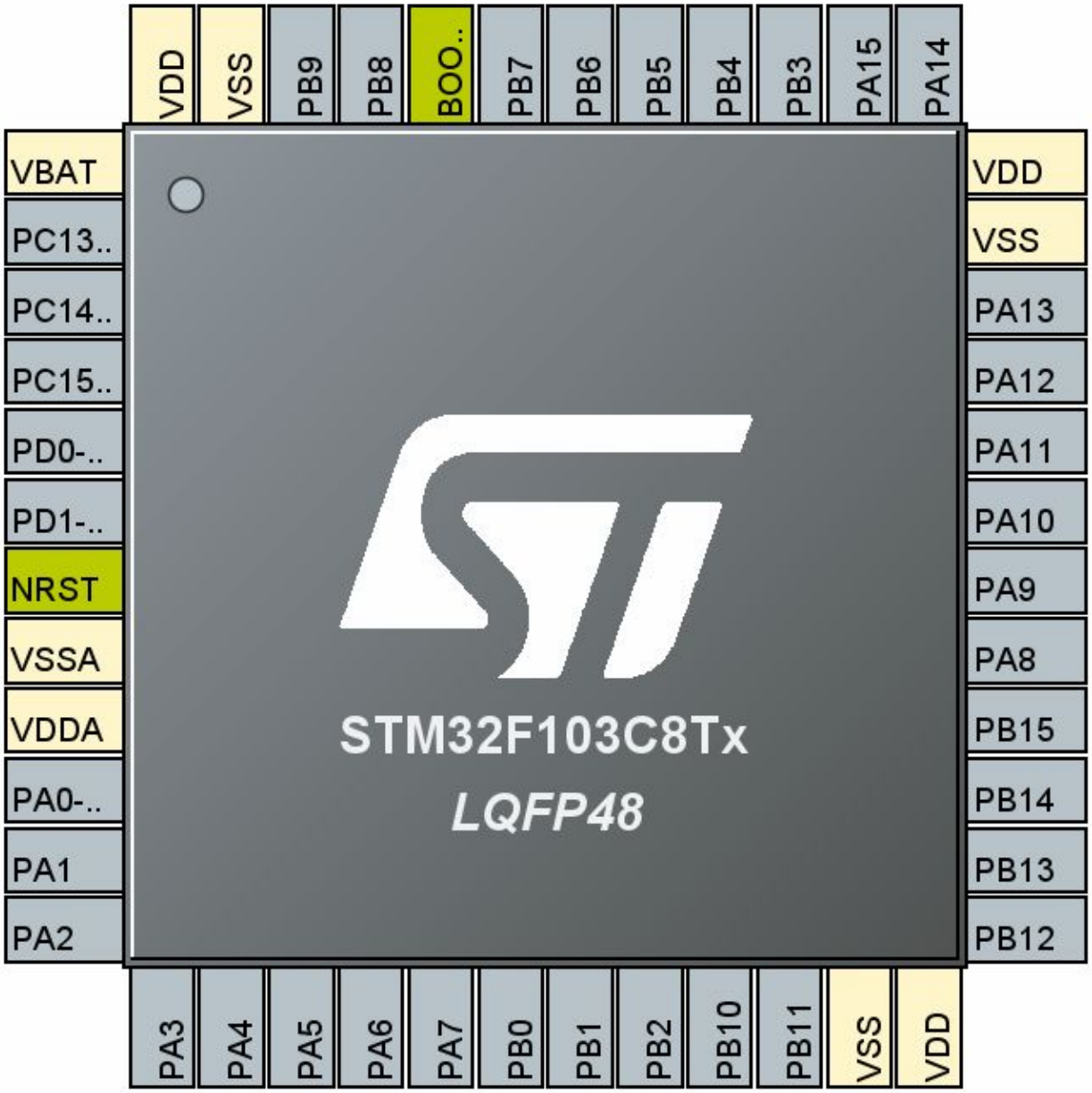
GPIO - ĐẶT TÊN VÀ TỔ CHỨC QUẢN LÝ

- Các khối GPIO được đặt tên theo các chữ cái A,B,C,D… GPIOA (PORTA)
- Số lượng khối GPIO phụ thuộc vào thiết kế của vi điều khiển có nhiều hay ít chân I/O ~32, 48,.., …
- Mỗi khối GPIO có thể quản lý tối đa được 16 I/O Pin
- Một khối GPIO còn có thể gọi là một PORT (PORTA ~ GPIOA, PORTB ~ GPIOB…)

Thực hành xác định:

Tên vi điều khiển STM32….

1. Vi điều khiển của bạn có tổng cộng bao nhiêu Pin?
2. Có bao nhiêu I/O Pin?
3. Được chia làm bao nhiêu Port?
4. Mỗi Port có bao nhiêu Pin?
5. Liệt kê những Pin còn lại?


VDD
VSS
PB9
PB8
BOO..
PB7
PB6
PB5
PB4
PB3
PA15
PA14
VBAT
PC13..
PC14..
PC15..
PD0-..
PD1-..
NRST
VSSA
VDDA
PA0-..
PA1
PA2
VDD
VSS
PA13
PA12
PA11
PA10
PA9
PA8
PB15
PB14
PB13
PB12
STM32F103C8Tx
LQFP48
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB10
PB11
VSS
VDD

GPIO Introduction

Each general-purpose **I/O port** has:

- **Four 32-bit configuration registers**: GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR
- **Two** 32-bit **data registers**:GPIOx_**I**DR, GPIOx_**O**DR
- A 32-bit set/reset register: GPIOx_BSRR
- A 32-bit locking register:GPIOx_LCKR
- Two 32-bit alternate function selection registers GPIOx_AFRH and GPIOx_AFRL

11 - General-purpose I/Os:
https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xb-c-d-e-stm32f303x6-8-stm32f328x8-stm32f358xc-stm32f398xe-advanced-arm-based-mcus-stmicroelectronics.pdf

CÁC NHÓM THANH GHI CỦA MỘT NGOẠI VI

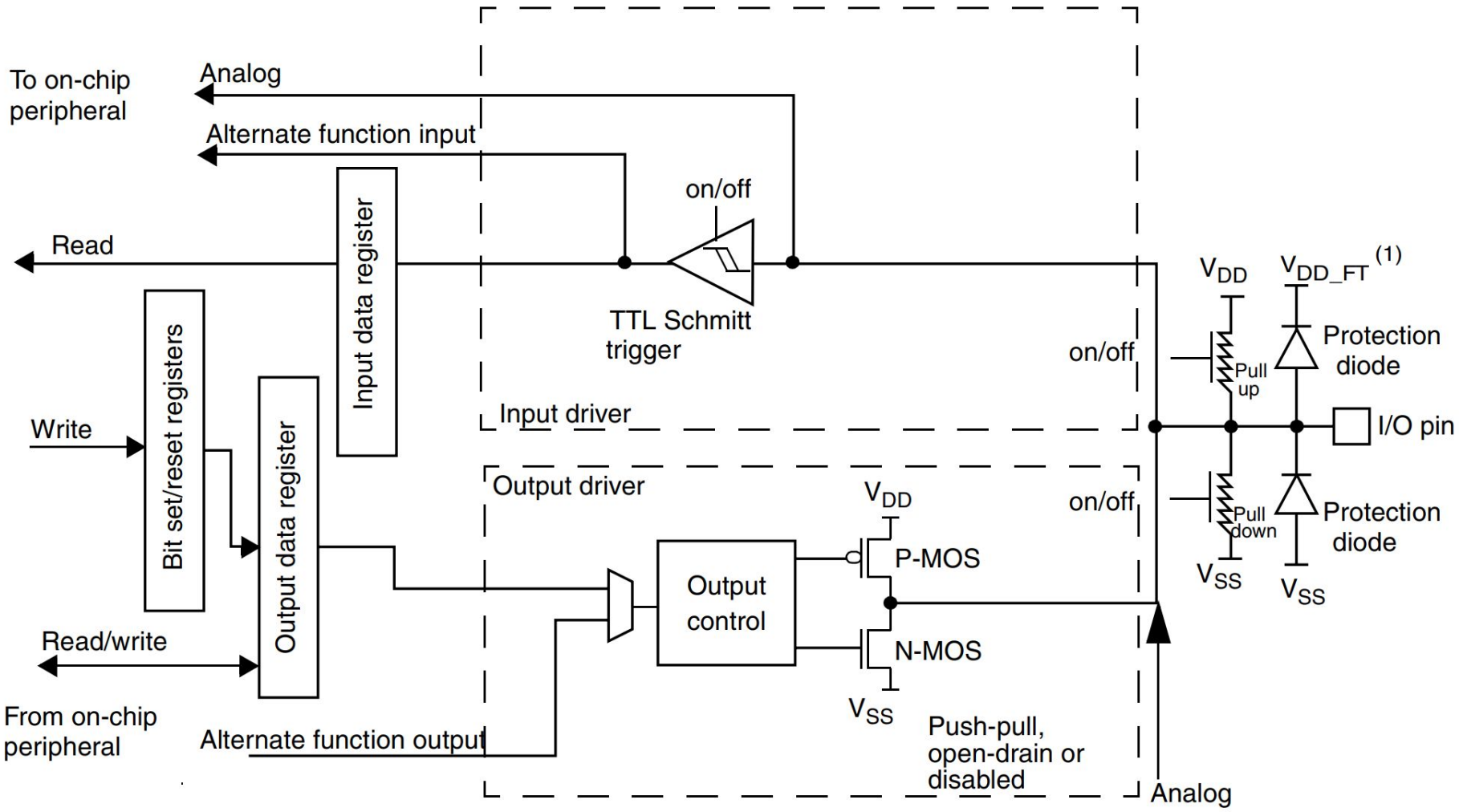
- Thanh ghi cấu hình
- Thanh ghi dữ liệu
- Thanh ghi trạng thái

Bản chất của lập trình các ngoại vi vi điều khiển là làm việc với các thanh ghi:

- Đọc thanh ghi (r)
- Ghi thanh ghi (w)

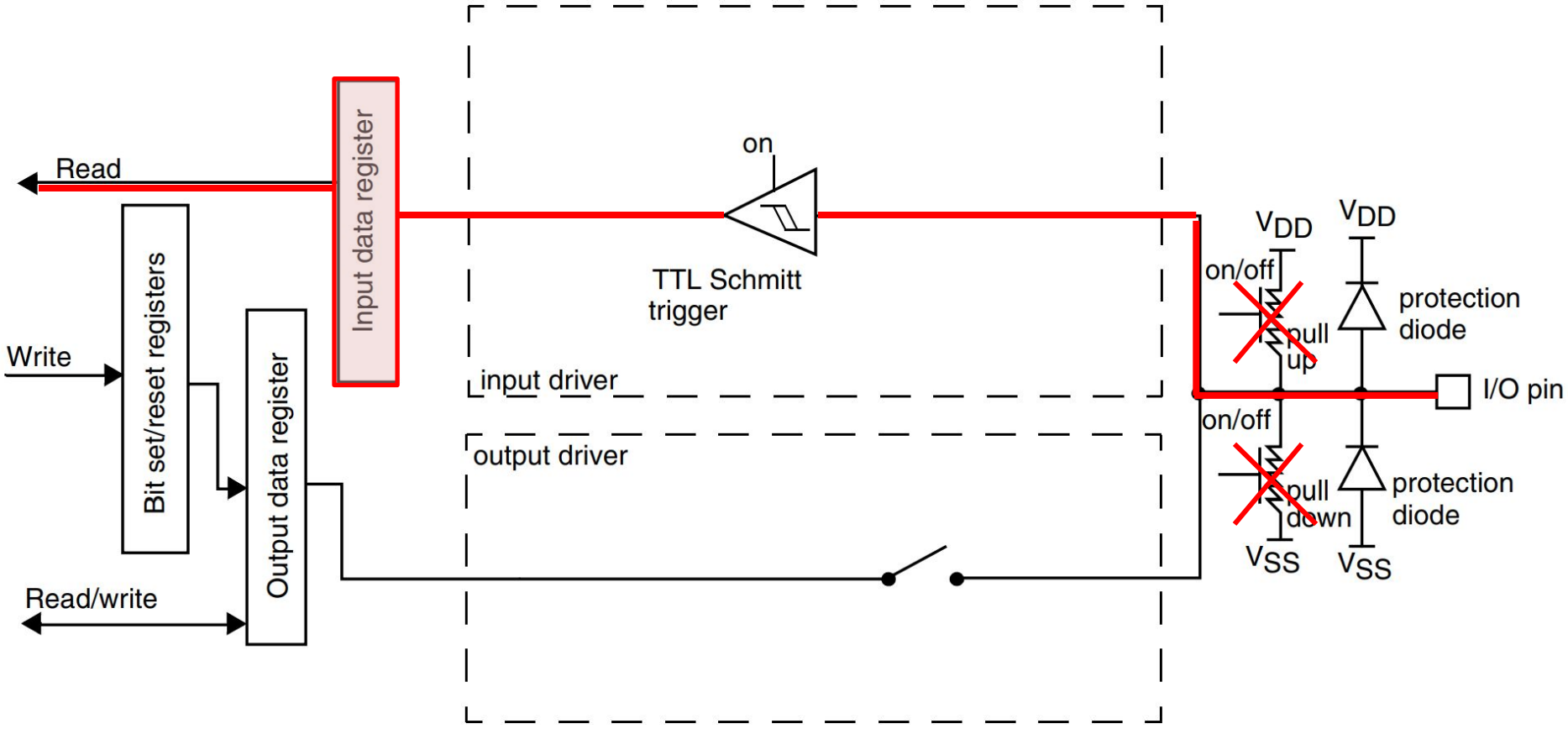
11 - General-purpose I/Os:
https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xb-c-d-e-stm32f303x6-8-stm32f328x8-stm32f358xc-stm32f398xe-advanced-arm-based-mcus-stmicroelectronics.pdf

Basic structure of a five-volt tolerant I/O port bit ~ pin



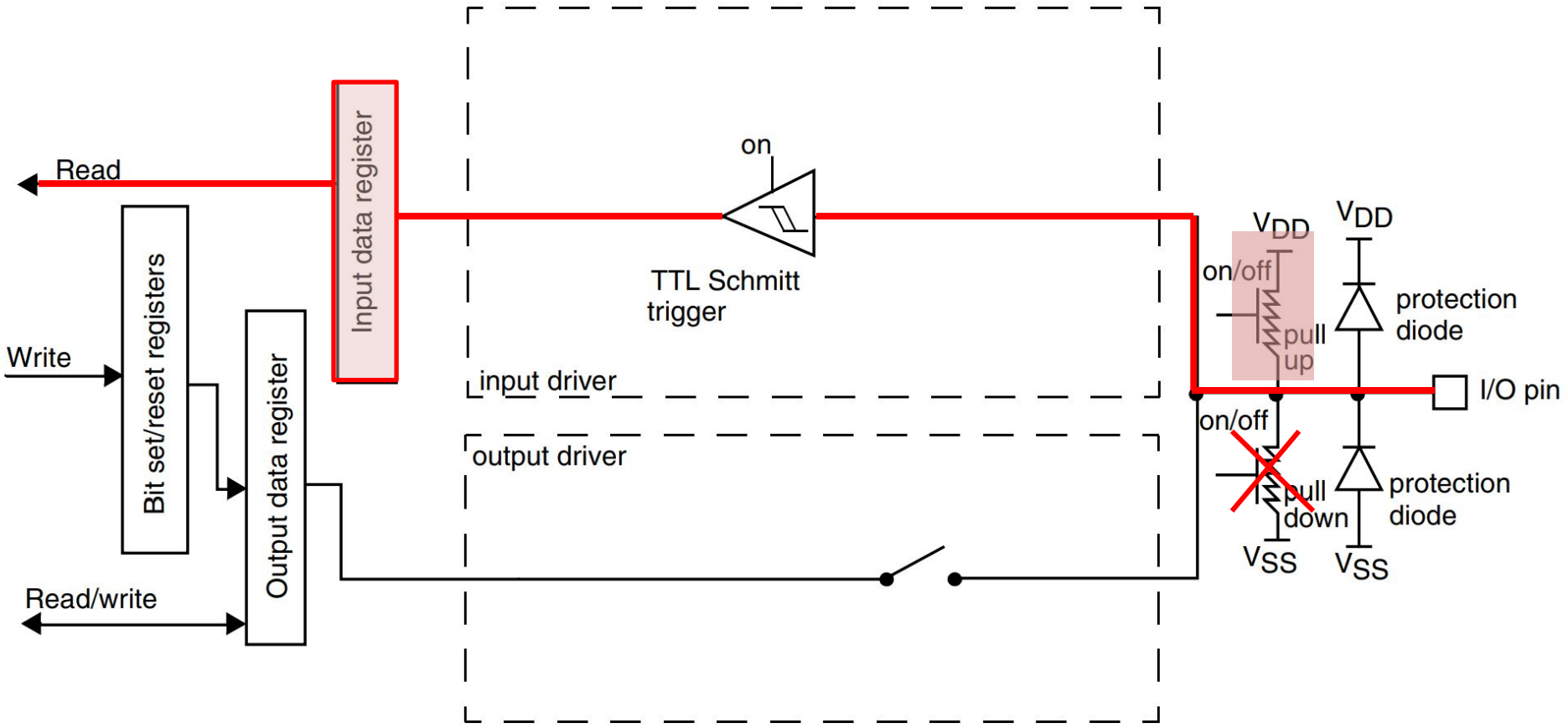
Input / Ouput / Analog/ Alternate function

GPIO: Input floating



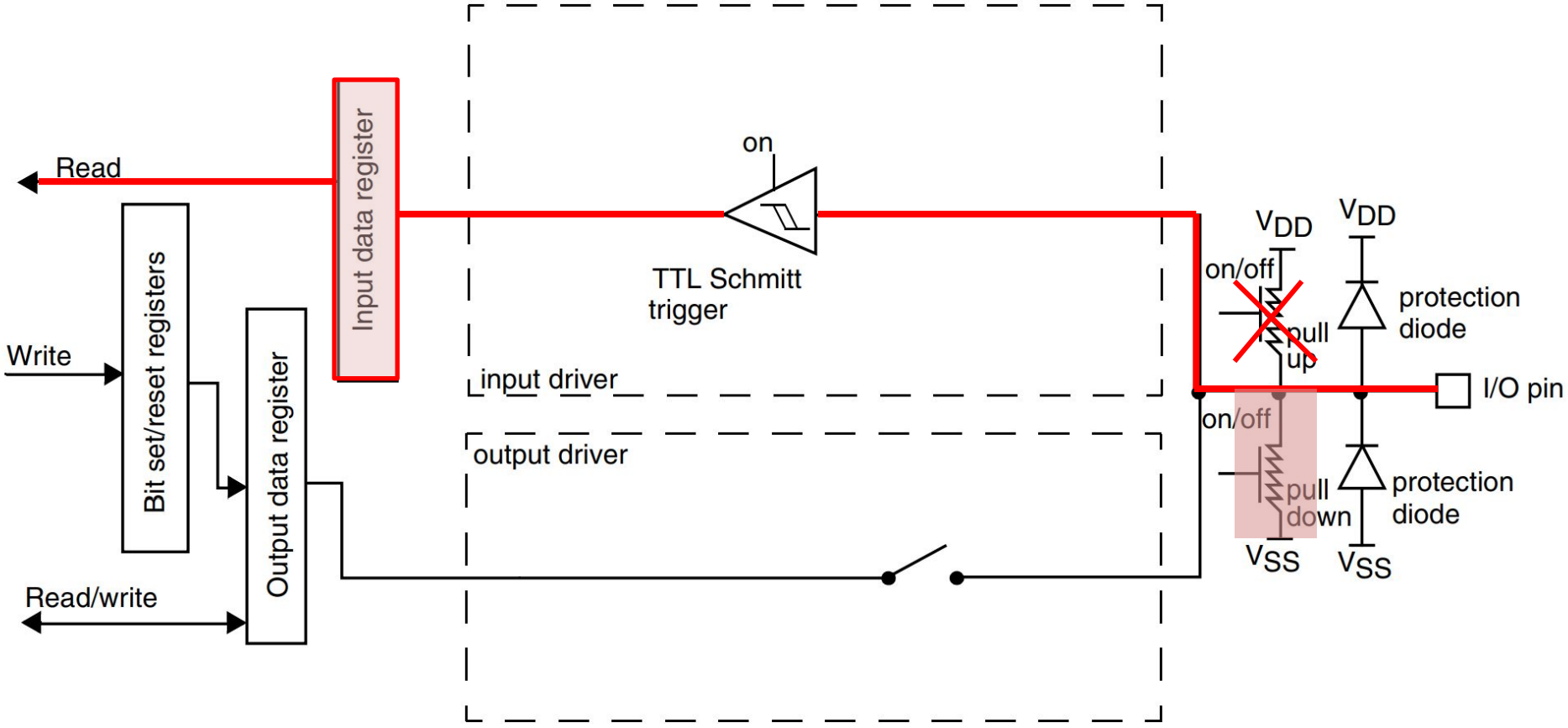
- The output buffer is disabled
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register provides the I/O state

GPIO: Input pull-up



- The output buffer is disabled
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register provides the I/O state

GPIO: Input pull-down



- The output buffer is disabled
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register provides the I/O state

VIL 0 -> 1,164 / 1.166 / 1.155 với VDD = 3.3V

VIH 2.145 -> 3,6

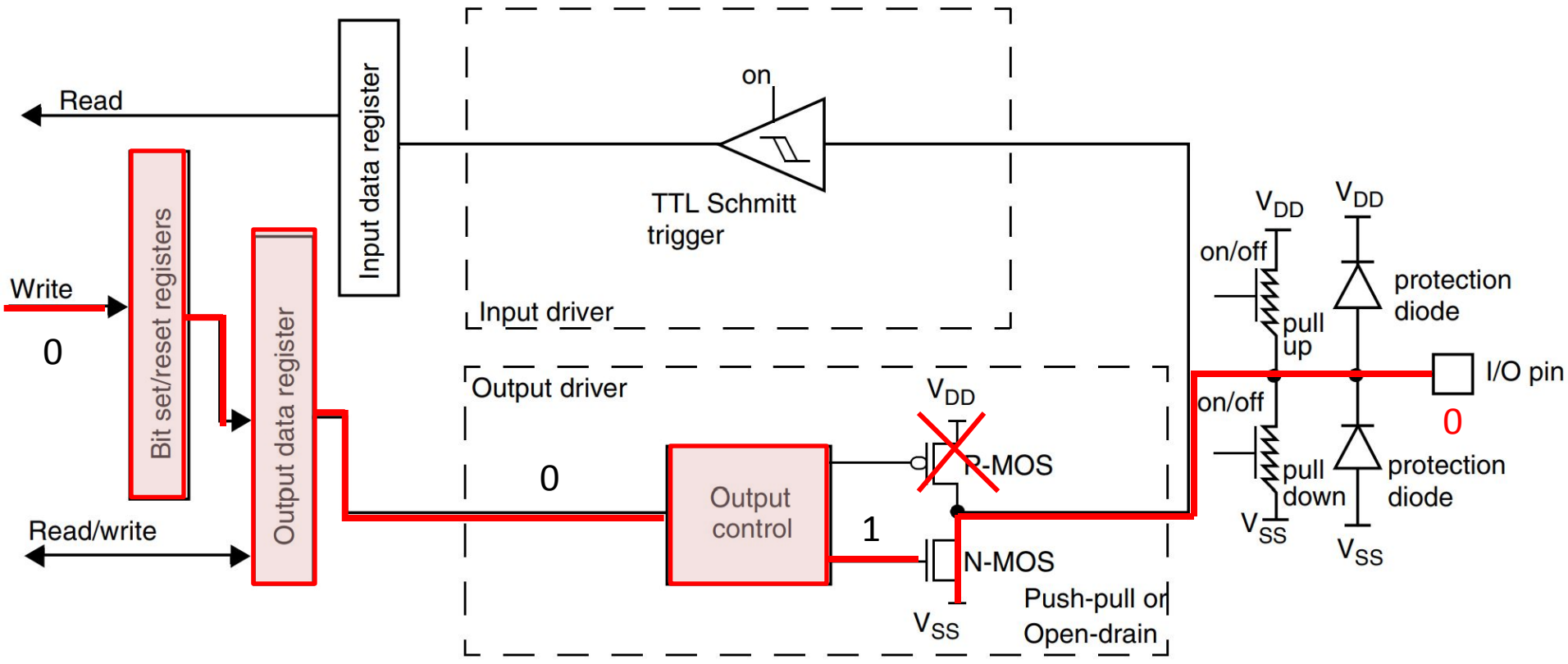
VOL 0V-> 0.4 (Lưu ý, VDD = 3.3v, và dòng <10mA)

VIL 2.4V/2.9 -> 3.3V

Mức 0 tiêu chuẩn là 0V.

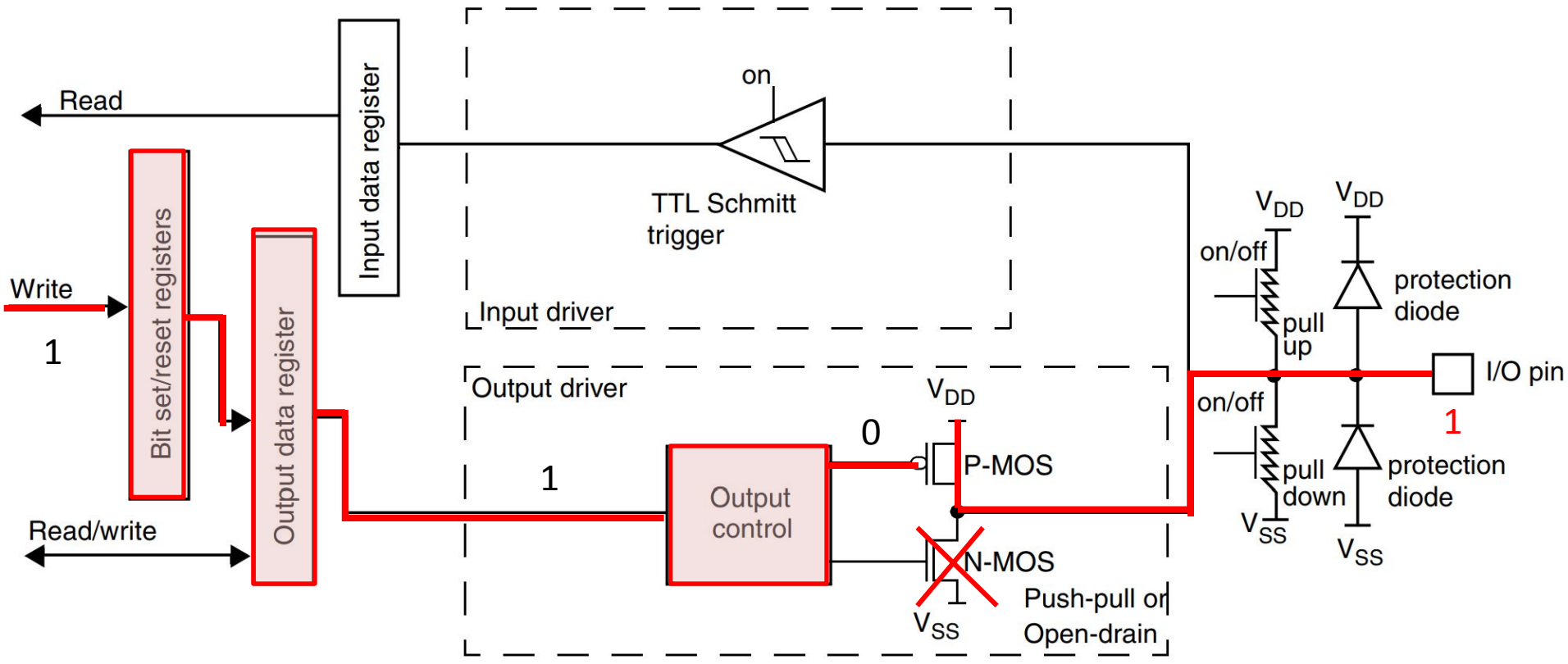
Vì sao cần 1 dải điện áp cho mức 0 INPUT như vậy?

GPIO: Output push-pull



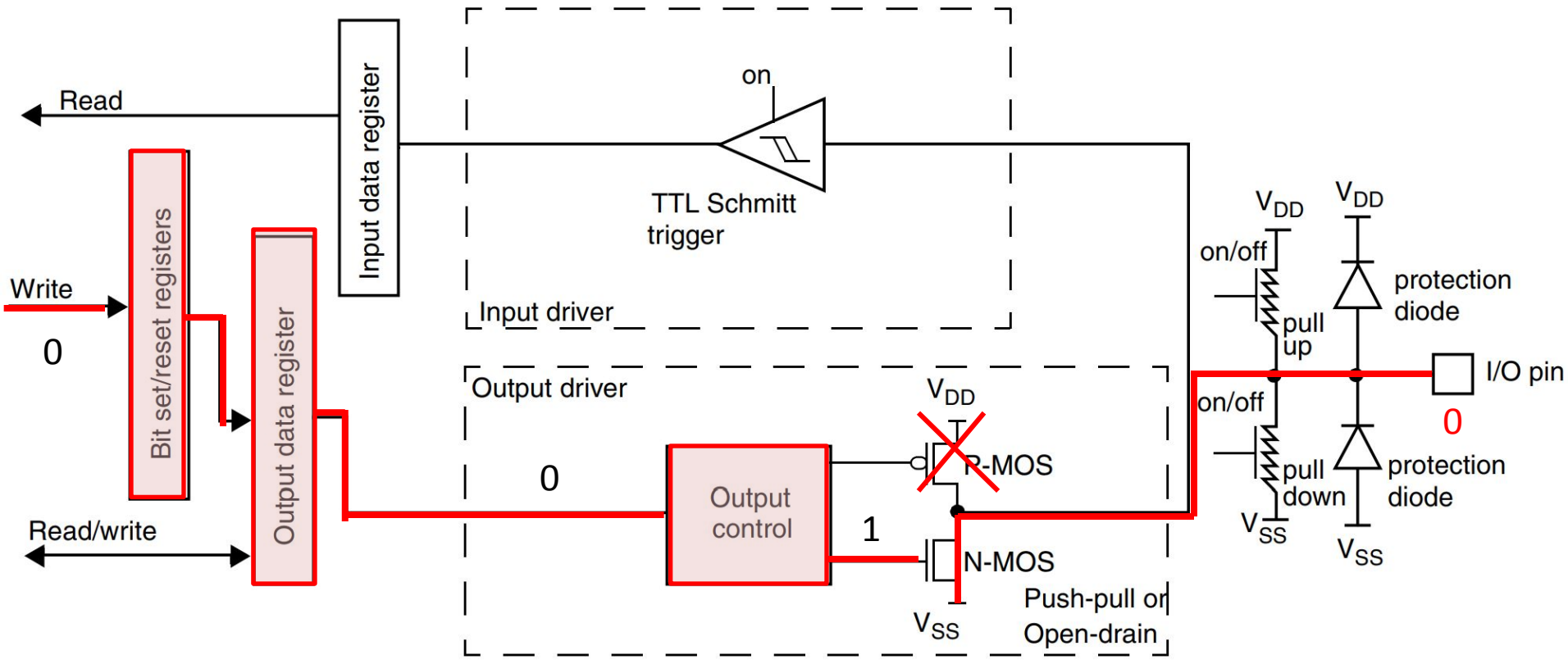
- The output buffer is enabled - Push-pull mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register activates the P-MOS
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value

GPIO: Output push-pull



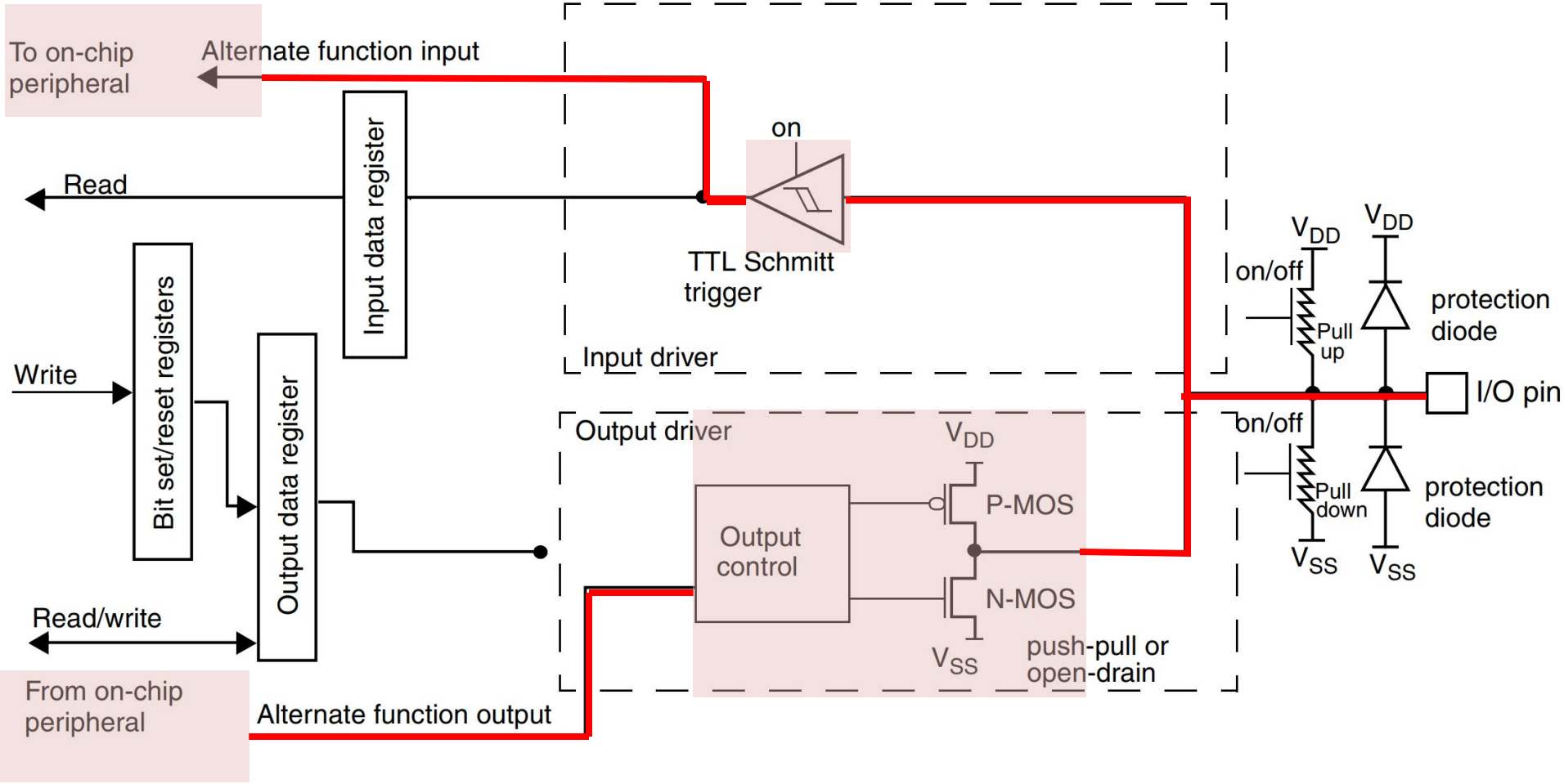
- The output buffer is enabled - Push-pull mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register activates the P-MOS
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value

GPIO: Open-drain

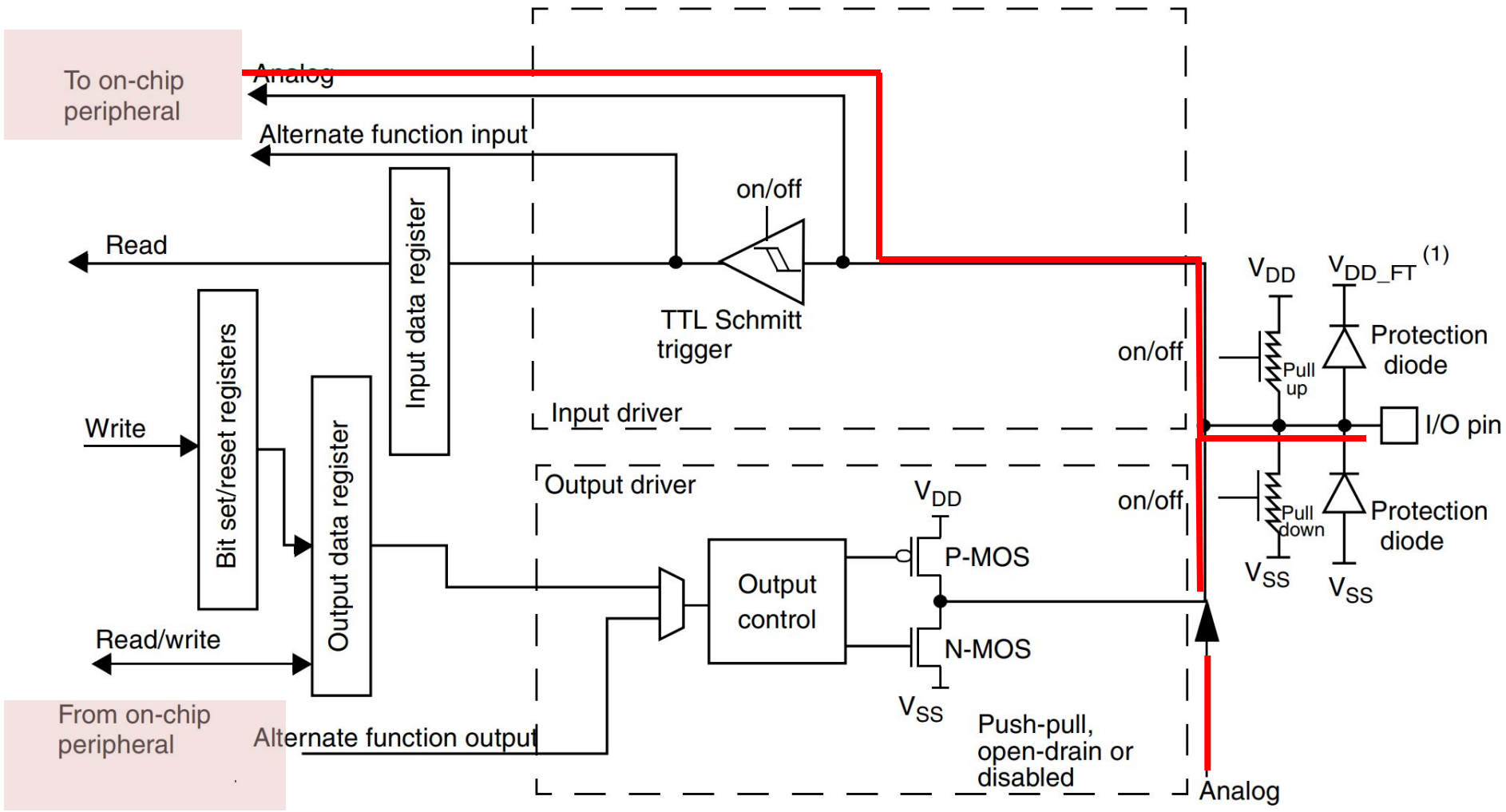


• The output buffer is enabled - Open drain mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register leaves the port in Hi-Z (the P-MOS is never activated)
• The Schmitt trigger input is activated
• The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
• The data present on the I/O pin are sampled into the input data register every AHB clock cycle • A read access to the input data register gets the I/O state
• A read access to the output data register gets the last written value

GPIO: Alternate function configuration



GPIO: Analog configuration



GPIO: HỖ TRỢ CỦA HÃNG ST

- Cấu hình và sinh mã qua giao diện CubeMX
- Lập trình sử dụng các hàm thư viện HAL

-> Tuy nhiên, cần nắm các thanh ghi để hiểu, kiểm chứng, debug trong quá trình phát triển phần mềm và gỡ lỗi.

THƯ VIỆN HAL - Hardware Abstraction Layer

- Được phát triển bởi hãng ST //Chứ không phải cộng đồng
- Sử dụng ngôn ngữ lập trình C
- Đã được STM32CubeIDE tải về khi khởi tạo project với một vi điều khiển mới
- Có đầy đủ các hàm hỗ trợ làm việc ở các chế độ: Polling, Interrupt, DMA, RTOS
- Hỗ trợ kỹ thuật timeout, quản lý lỗi
- Đảm bảo khả năng kế thừa, sử dụng lại mã chương trình, khi thay đổi, nâng cấp phần cứng vi điều khiển
- Có phân loại các thư viện dùng chung và thư viện riêng cho mỗi dòng

11 - General-purpose I/Os:
https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xb-c-d-e-stm32f303x6-8-stm32f328x8-stm32f358xc-stm32f398xe-advanced-arm-based-mcus-stmicroelectronics.pdf

```c
#include "main.h"


void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{

  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  while (1)
  {

  }
}
```

```c
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* USER CODE BEGIN PV */
/* Private variables --------

/* USER CODE END PV */

/* USER CODE BEGIN PFP */
/* Private function prototypes ----

/* USER CODE END PFP */

/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

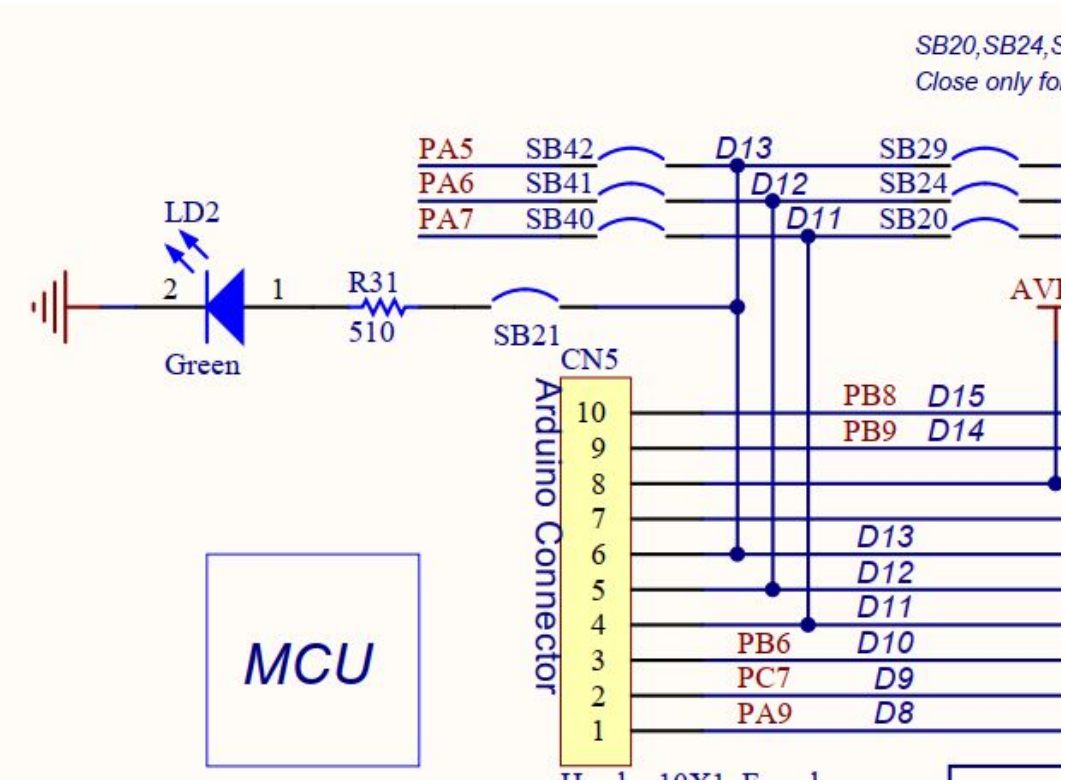
GPIO Lab

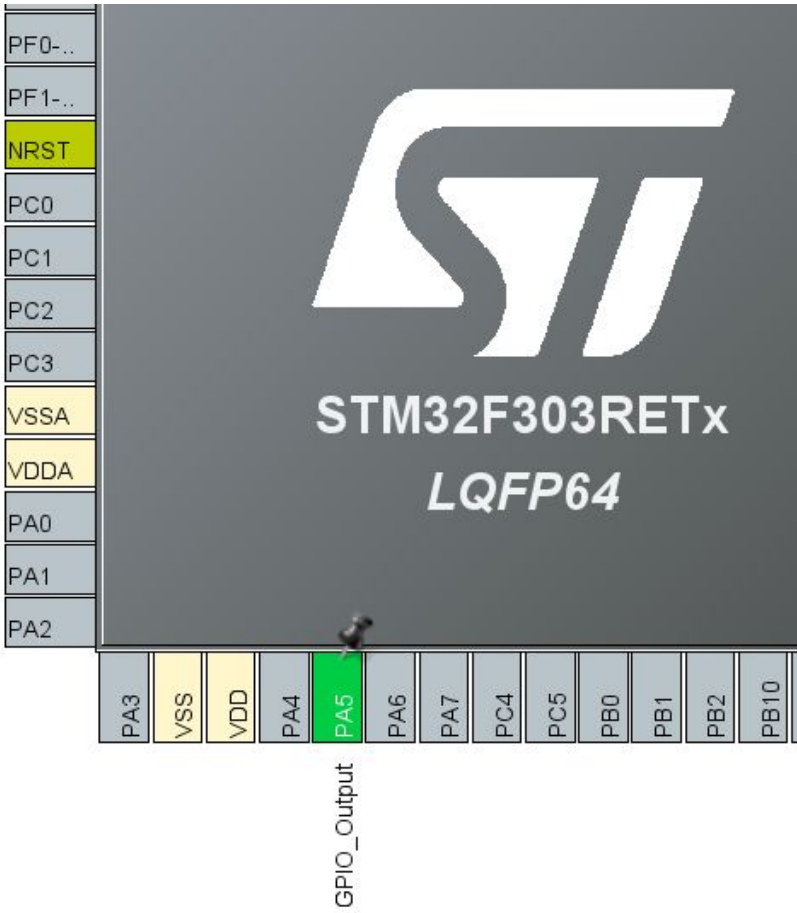
- Objective
  - Learn how to setup pin and GPIO port in CubeMX
  - How to Generate Code in CubeMX and use HAL functions
- Goal
  - Configure GPIO pin in CubeMX and Generate Code
  - Add in to project HAL_Delay function and HAL_GPIO_Toggle function
  - Verify the correct functionality on toggling LED

Create project in STM32CubeIDE

- Menu > File > STM32 Project
- Select STM32F303RE > Project Name: “ToggleLED” > Finish
- Select System Core > SYS > Debug: Serial Wire

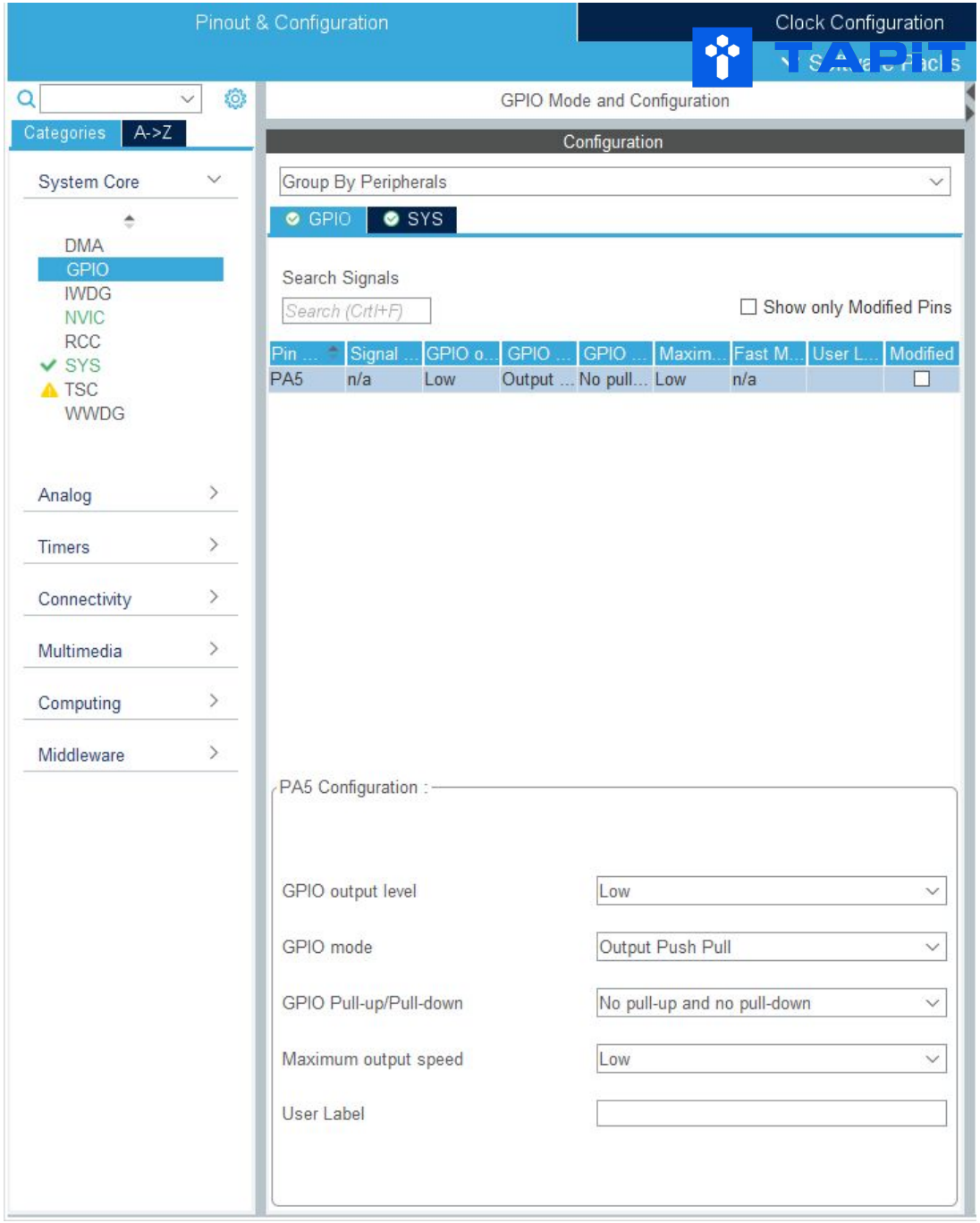
Configure LED pin as GPIO_Output





GPIO(Pin) Configuration

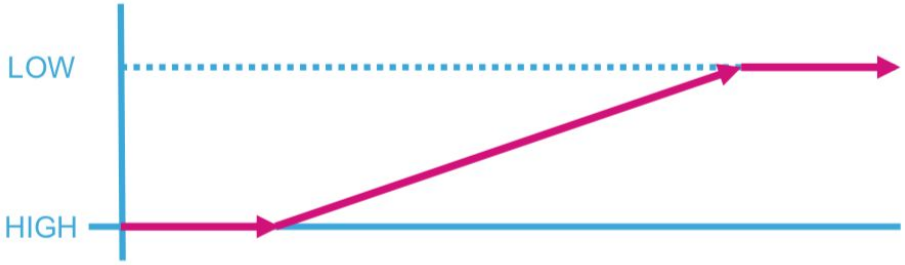
- Select Push Pull Mode
- No pull-up and pull-down
- Output speed to LOW as default



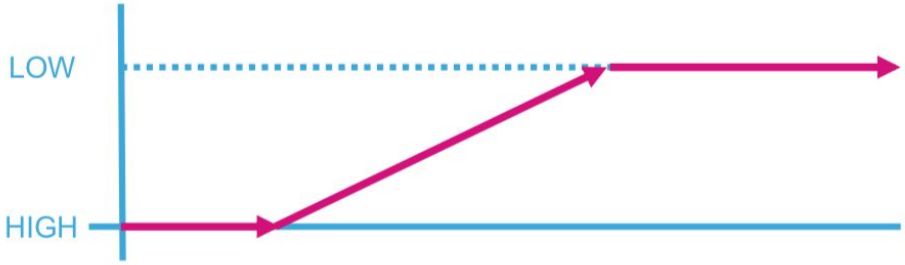
GPIO(Pin) output speed configuration

- Change the rising and falling edge when pin change state from high to low or low to high
- **Higher** GPIO speed increase **EMI noise** from STM32 and increase STM32 **consumption**
- It is good to adapt GPIO speed with periphery speed. Ex.: Toggling GPIO on 1Hz is LOW optimal settings, but SPI on 45MHz the HIGH must be set

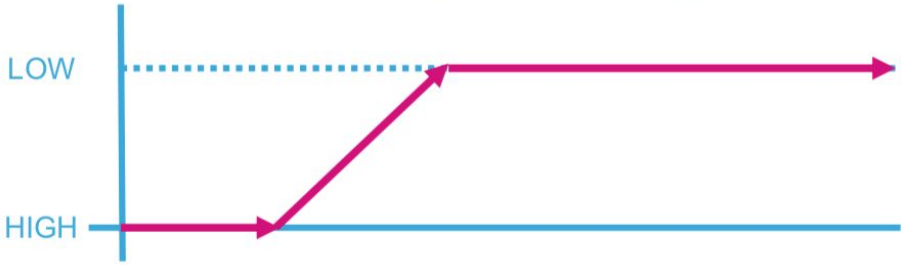
GPIO output LOW speed



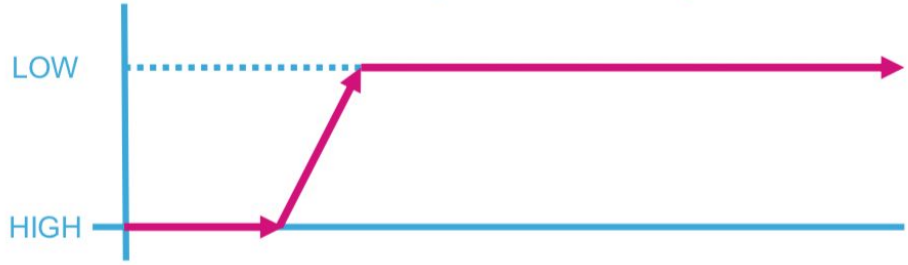
GPIO output MEDIUM speed



GPIO output FAST speed



GPIO output HIGH speed



Now we can Generate Code

Menu > Project > Generate Code or click Generate icon

Now we open the project in our IDE

The functions we want to put into **main.c**
Between /* USER CODE BEGIN WHILE */ and /* USER CODE END WHILE */ tags
Into infinite loop while(1){ }

For toggling we need to use this functions

HAL_Delay: which create specific delay
HAL_GPIO_WritePin or HAL_GPIO_TogglePin

```
/*==========================================
    ##### IO operation functions #####
==========================================*/
/**
  * @note   This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify
  *         accesses. In this way, there is no risk of an IRQ occurring between
  *         the read and the modify access.
  *
  * @param  GPIOx where x can be (A..F) to select the GPIO peripheral for STM32F3 family
  * @param  GPIO_Pin specifies the port bit to be written.
  *         This parameter can be one of GPIO_PIN_x where x can be (0..15).
  * @param  PinState specifies the value to be written to the selected bit.
  *         This parameter can be one of the GPIO_PinState enum values:
  *            @arg GPIO_PIN_RESET: to clear the port pin
  *            @arg GPIO_PIN_SET: to set the port pin
  * @retval None
  */
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  assert_param(IS_GPIO_PIN_ACTION(PinState));

  if(PinState != GPIO_PIN_RESET)
  {
    GPIOx->BSRR = (uint32_t)GPIO_Pin;
  }
  else
  {
    GPIOx->BRR = (uint32_t)GPIO_Pin;
  }
}
```

GPIOx : **GPIOA, GPIOB, GPIOC, GPIOD, GPIOF**
GPIO_Pin: **GPIO_PIN_0 -> GPIO_PIN_15**
PinState: **GPIO_PIN_SET (1)**
GPIO_PIN_RESET (0)

```
==========================================
    ##### IO operation functions #####
==========================================
```

```
  * @brief  Toggle the specified GPIO pin.
  * @param  GPIOx where x can be (A..F) to select the GPIO peripheral for STM32F3 family
  * @param  GPIO_Pin specifies the pin to be toggled.
  * @retval None
  */
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  uint32_t odr;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  /* get current Ouput Data Register value */
  odr = GPIOx->ODR;

  /* Set selected pins that were at low level, and reset ones that were high */
  GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}
```

GPIOx : **GPIOA, GPIOB, GPIOC, GPIOD, GPIOF**

GPIO_Pin: **GPIO_PIN_0 -> GPIO_PIN_15**

```
/*==============================================*/
    ##### IO operation functions #####
/*==============================================*/

/**
  * @brief  Read the specified input port pin.
  * @param  GPIOx where x can be (A..F) to select the GPIO peripheral for STM32F3 family
  * @param  GPIO_Pin specifies the port bit to read.
  *         This parameter can be GPIO_PIN_x where x can be (0..15).
  * @retval The input port pin value.
  */
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  GPIO_PinState bitstatus;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
  {
    bitstatus = GPIO_PIN_SET;
  }
  else
  {
    bitstatus = GPIO_PIN_RESET;
  }
  return bitstatus;
}
```

GPIOx : **GPIOA, GPIOB, GPIOC, GPIOD, GPIOF**
GPIO_Pin: **GPIO_PIN_0 -> GPIO_PIN_15**

Return: **GPIO_PIN_SET (1)**
GPIO_PIN_RESET (0)

BÀI TẬP THỰC HÀNH

NẾU NÚT NHẤN ĐƯỢC NHẤT GIỮ THÌ LED SÁNG

- Xác định chân nào nối với nút nhấn, khi nhấn nút thì mức logic là 0 hay 1 và khi không nhấn nút thì sao?
- Xác định chân nào nối với đèn LED, muốn đèn sáng thì cần mức logic 0 hay 1?
- Khai báo sử dụng giao diện CubeMX
- Viết trong while 1 của main:
- Trong khi đang nhấn nút, đèn sáng, sử dụng
  - while
  - HAL_GPIO_ReadPin
  - HAL_GPIO_WritePin

```
==========================================
    ##### HAL Control functions #####
==========================================
```

TAPiT

```
/**
  * @brief  This function provides accurate delay (in milliseconds) based
  *         on variable incremented.
  * @note   In the default implementation , SysTick timer is the source of time base.
  *         It is used to generate interrupts at regular time intervals where uwTick
  *         is incremented.
  *         The function is declared as __Weak  to be overwritten  in case of other
  *         implementations  in user file.
  * @param  Delay specifies the delay time length, in milliseconds.
  * @retval None
  */
__weak void HAL_Delay(uint32_t Delay)
{
  uint32_t tickstart = HAL_GetTick();
  uint32_t wait = Delay;

  /* Add freq to guarantee minimum wait */
  if (wait < HAL_MAX_DELAY)
  {
    wait += (uint32_t)(uwTickFreq);
  }

  while((HAL_GetTick() - tickstart) < wait)
  {
  }
}
```

DEBUG

https://youtu.be/50cc5TmiF78

PHÂN TÍCH DỰ ÁN

KTTV - DATALOGGER

Yêu cầu dự án

- Cứ tròn 10 phút thiết bị sẽ thu thập và tổng hợp các số liệu sau và gửi lên Server:
  - Giá trị mực nước đo được từ cảm biến siêu âm, tín hiệu analog current
  - Giá trị lượng mưa đo được từ cảm biến đo mưa kiểu chao lật, tín hiệu xung
  - Giá trị điện áp cung cấp cho hệ thống, tín hiệu analog voltage
  - Trạng thái của cảm biến: OK hoặc ERROR

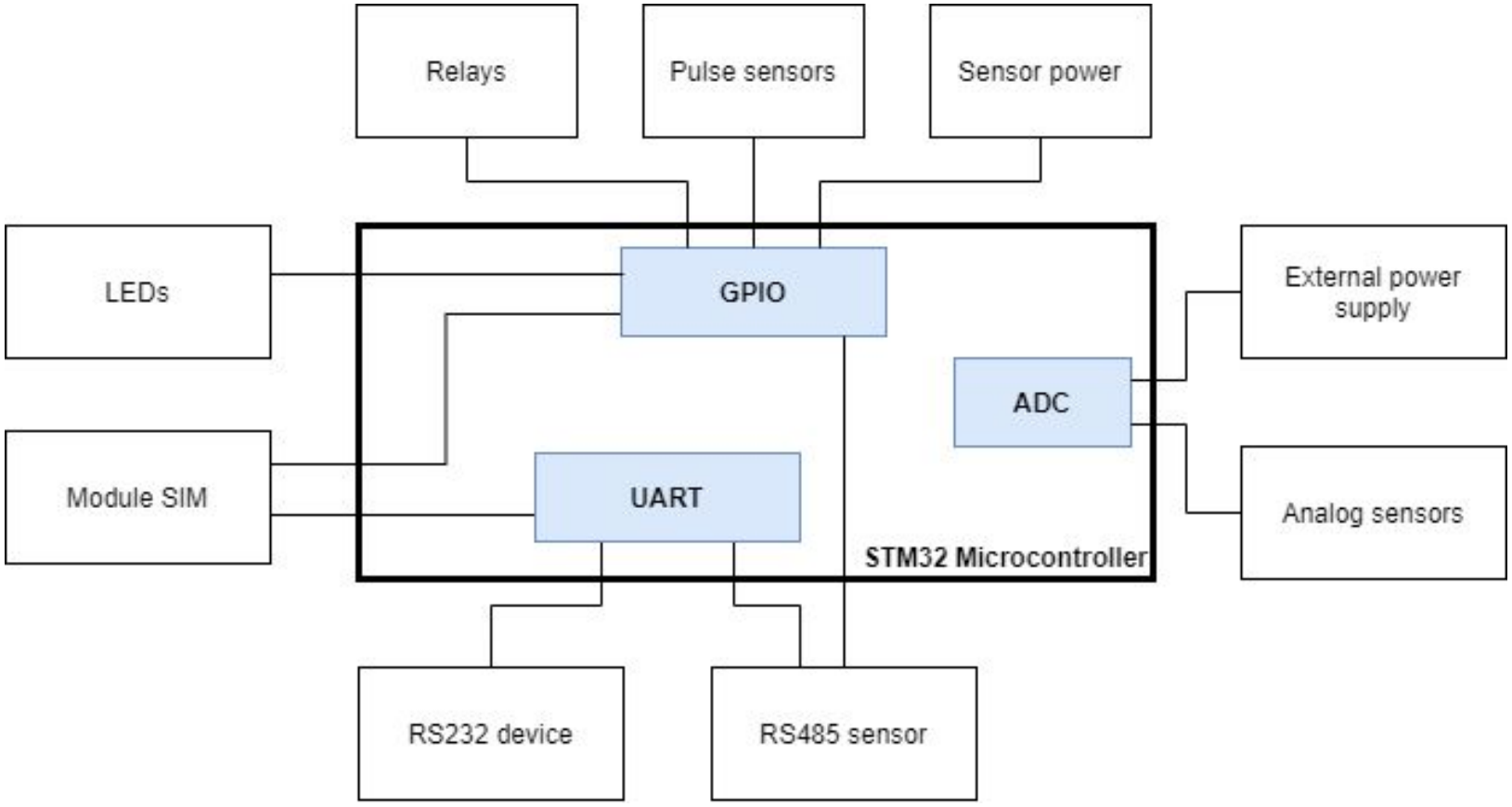
- Báo hiệu trạng thái của hệ thống thông qua LED STATUS trên thiết bị
  - LED nhấp nháy 10 giây 1 lần: Hệ thống hoạt động bình thường
  - LED nhấp nháy 3 giây 1 lần: Thiết bị không kết nối được Internet
  - LED nhấp nháy 1 giây 1 lần: Lỗi cảm biến

- Khởi động lại thiết bị từ xa thông qua tin nhắn SMS

- Khởi động lại thiết bị vào lúc 0h4p hằng ngày

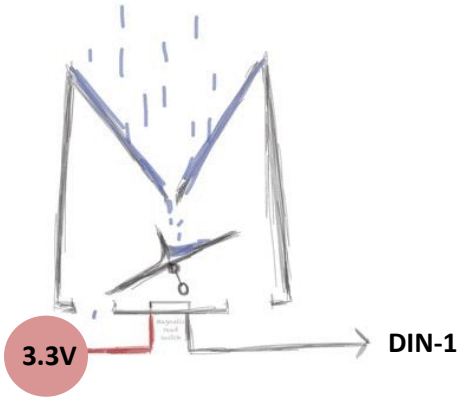
- Tiết kiệm năng lượng tiêu thụ của hệ thống:
  - Đưa Module SIM về trạng thái Sleep Mode
  - Tắt nguồn cung cấp cho các cảm biến (Relay, FET)

Hardware Block Diagram



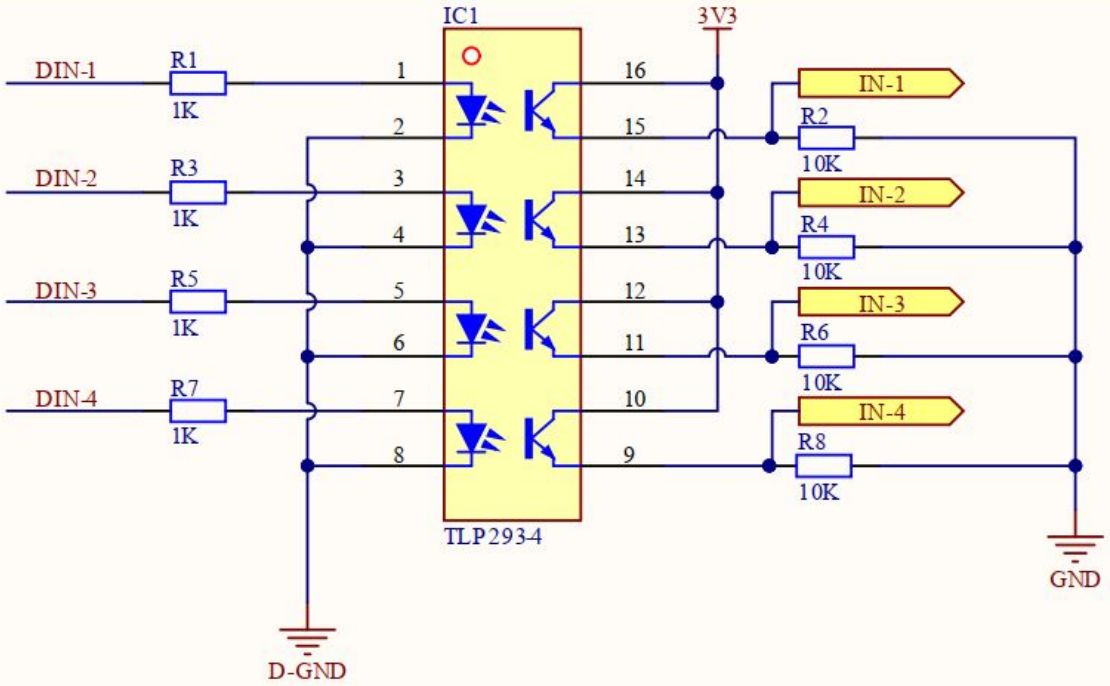
NHỮNG TÍNH NĂNG LIÊN QUAN ĐẾN NGOẠI VI GPIO

Đọc giá trị lượng mưa



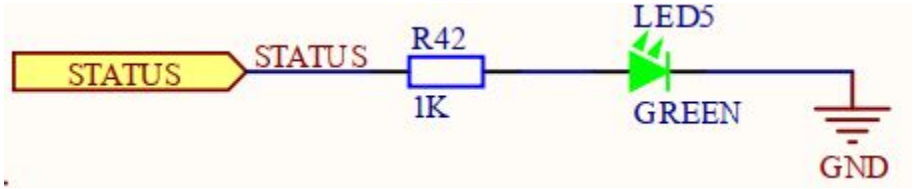
DIGITAL INPUT

I_F_max=50mA V_F=1,4V,
CTR_min=100%
V_DIN=(0V-50V)
LOGIC 0: 0V-1V
LOGIC 1: 3V-50V



Báo hiệu trạng thái của hệ thống

- LED nhấp nháy 10 giây 1 lần: Hệ thống hoạt động bình thường
- LED nhấp nháy 3 giây 1 lần: Thiết bị không kết nối được Internet
- LED nhấp nháy 1 giây 1 lần: Lỗi cảm biến



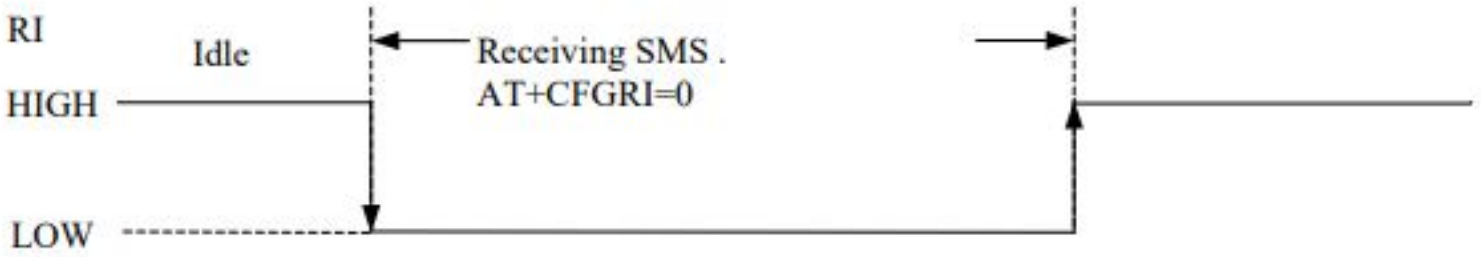
Sleep Mode - ModuleSIM

- Thiết bị Host (vi điều khiển STM32) phải gửi lệnh AT+CSCLK=1 tới ModuleSIM để bật chức năng SleepMode. Sử dụng chân DTR của ModuleSIM làm để điều khiển Module vào hoặc thoát khỏi chế độ Sleep.
- DTR pulled up: ModuleSIM **vào** SleepMode
- DTR pulled down: ModuleSIM **thoát** SleepMode

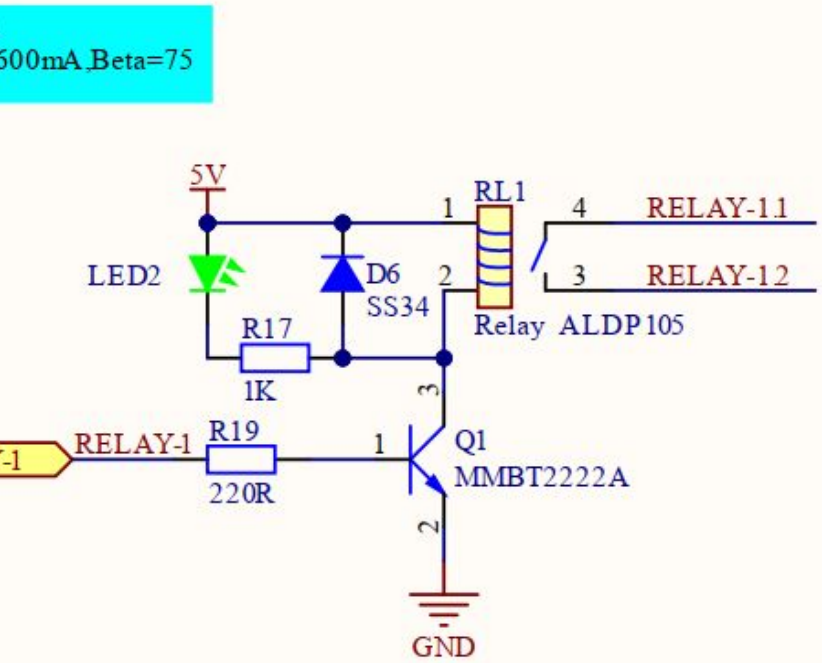
SMS – ModuleSIM

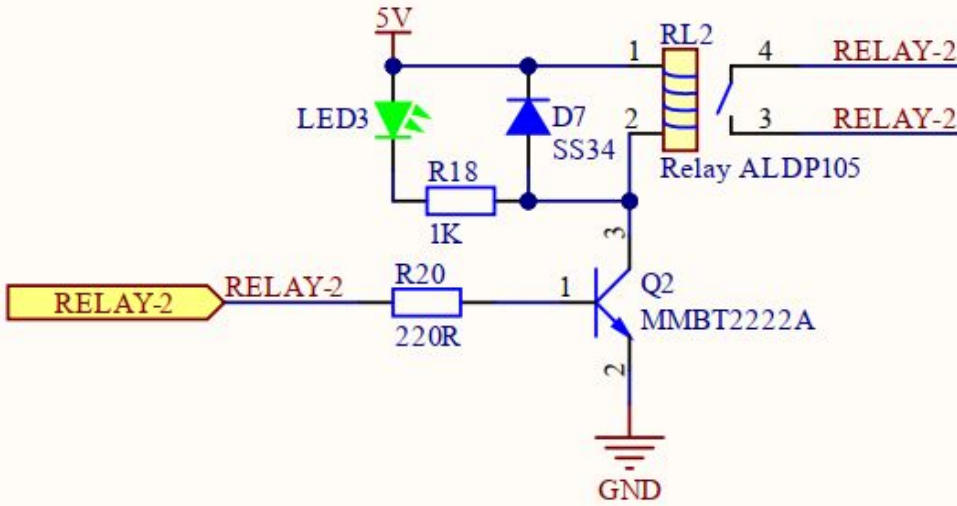
- Chân RI của ModuleSIM được sử dụng để thông báo đến Host khi ModuleSIM có 1 tin nhắn đến.
- Sự thay đổi mức logic khi vừa có SMS:

  HIGH -> LOW -> HIGH



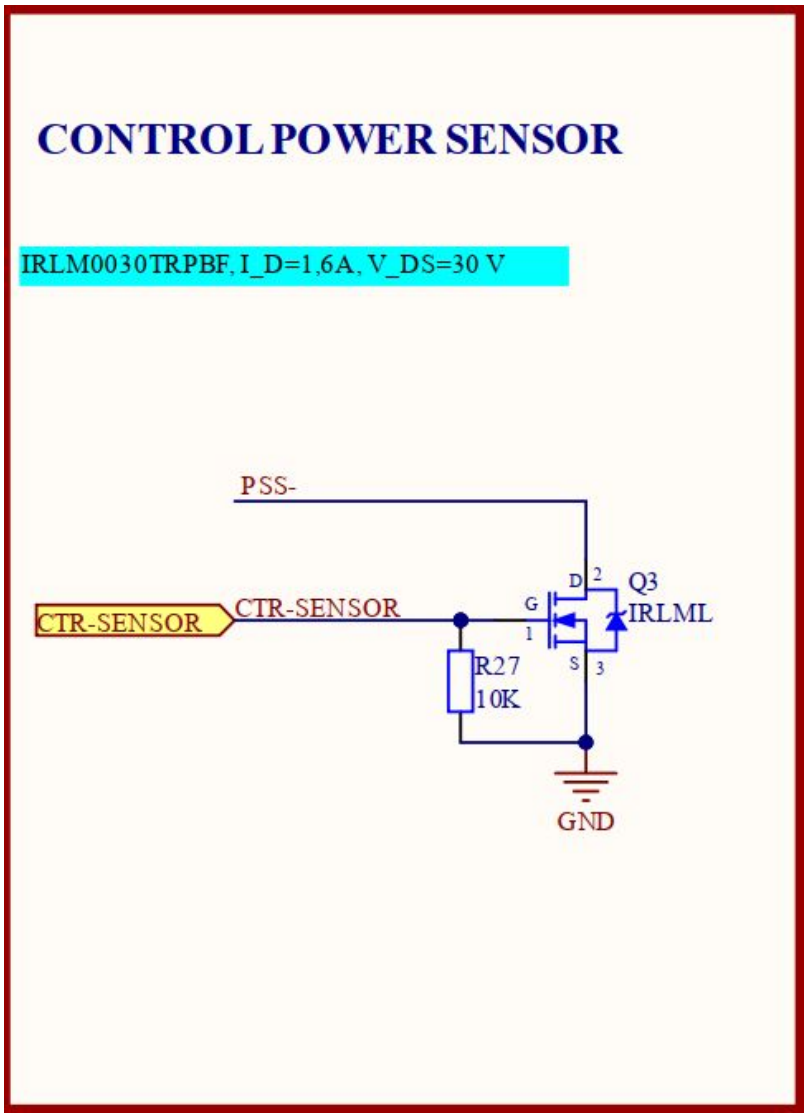
Điều khiển nguồn cảm biến - RELAY





Điều khiển nguồn cảm biến - FET

- Dây **nguồn dương** của cảm biến nối Vin của thiết bị
- Dây **nguồn âm** của cảm biến nối với PSS-



Bài tập1 làm luôn:

Nếu nhấn nút (nhấn thả) thì LED thay đổi trạng thái

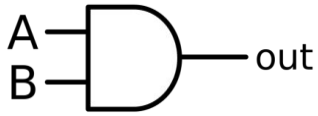
Nếu trước đó tắt thì sau khi nhấn thả sẽ sáng và ngược lại.

Bài tập 2:

Nhấn nút (nhấn thả) 3 lần thì LED đảo trạng thái.

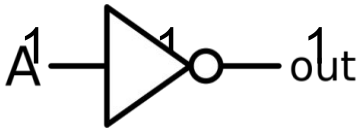
CLEAR BIT (Masking bits to 0**)**

This operation uses the principle that **X AND 1 = X** and **X AND 0 = 0**.



AND Truth Table

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1



NOT Truth Table

A	~A
0	1
1	0

Example here:

```
0bXXXXXXXX &      0bXXXXXXXX &      0bXXXXXXXX &
0b11111110        0b11101111        0b11100111
------------      ------------      ------------
0bXXXXXXX0        0bXXX0XXXX        0bXXX00XXX
```

Which shift left **<<** and the **ones' complement** operators we can perform the same operations in a different way.

```
0bXXXXXXXX &      0bXXXXXXXX &      0bXXXXXXXX &
 ~(1 << 0)         ~(1 << 4)       ~(0b11 << 3)
------------      ------------      ------------
0bXXXXXXX0        0bXXX0XXXX        0bXXX00XXX
```

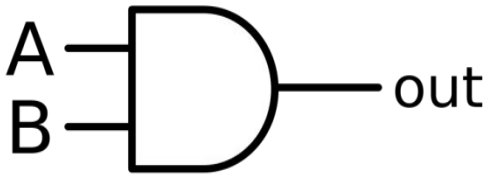
Concluding the operation **word & ~(1 << n)** masks the n-th bit of "**word**" to "**0**".

Bitwise

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise Complement (NOT)
<<	Shift left
>>	Shift right

Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1.
If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

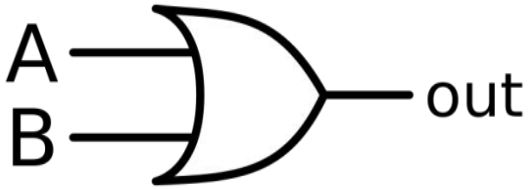


AND Truth Table

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.
In C Programming, bitwise OR operator is denoted by |.

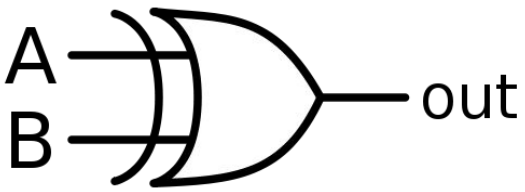


OR Truth Table

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

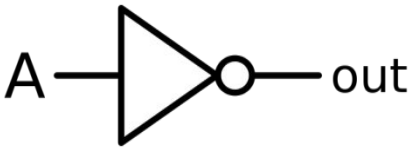


XOR Truth Table

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand).
It changes 1 to 0 and 0 to 1. It is denoted by ~.



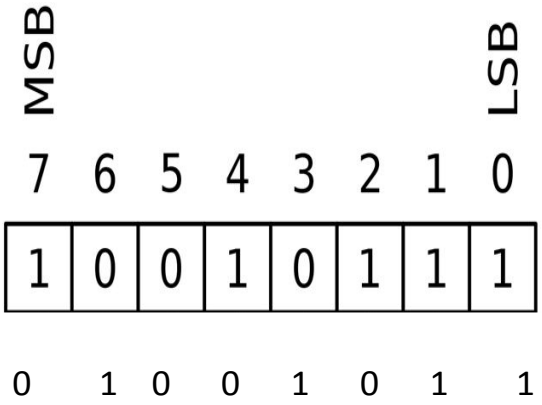
NOT Truth Table

A	~A
0	1
1	0

Shift Operators

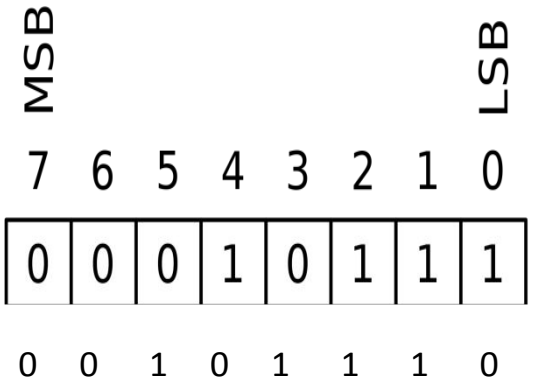
Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits.
It is denoted by >>.



Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits.
It is denoted by <<.

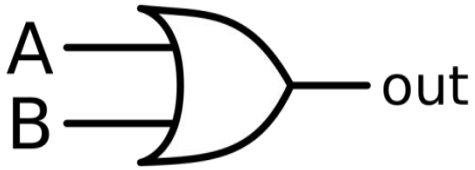


Registers and bit masks

Using certain operand in a single **bitwise** operation we can set to ‘1’, to ‘0’ or toggle multiple bits in a word. This operand is commonly known as mask or **bit-mask** and the operation is called masking or **bit-masking**.

Operators	Meaning of operators
|=	Set bit
&=~	Clear bit
^=	Toggle bit
&	Check bit value

SET BIT (Masking bits to 1**)**
This operation uses the principle that **X OR 1 = 1** and **X OR 0 = X**



OR Truth Table

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Example here:

```
0bXXXXXXXX |      0bXXXXXXXX |      0bXXXXXXXX |
0b00000001       0b00010000       0b00011000
------------     ------------     ------------
0bXXXXXXX1       0bXXX1XXXX       0bXXX11XXX
```

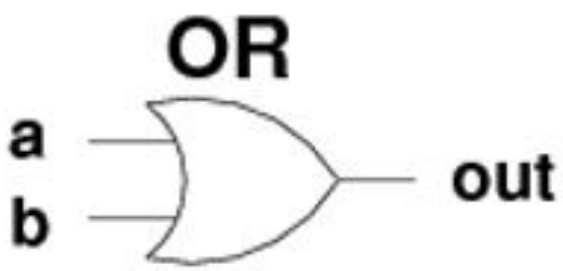
Which shift left operator **<<** we can perform the same operations in a different way.

```
0bXXXXXXXX |      0bXXXXXXXX |      0bXXXXXXXX |
  (1 << 0)         (1 << 4)        (0b11 << 3)
------------     ------------     ------------
0bXXXXXXX1       0bXXX1XXXX       0bXXX11XXX
```

Concluding the operation **word | (1 << n)** masks the n-th bit of "**word**" to "**1**".

PHÉP SET BIT: |=

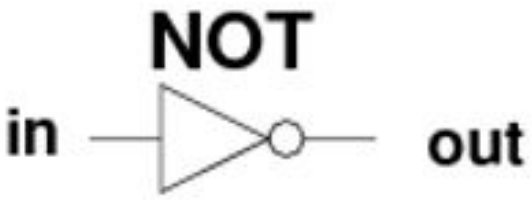
Phép SET BIT là phép cài đặt 1 bit mong muốn trong thanh ghi cho nó có giá trị **logic 1** và không làm thay đổi giá trị các bit còn lại của thanh ghi đó.



a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

PHÉP CLEAR BIT: &= ~

Phép CLEAR BIT là phép cài đặt 1 bit mong muốn trong thanh ghi cho nó giá trị **logic 0** và không làm thay đổi giá trị các bit còn lại của thanh ghi đó.



in	out
0	1
1	0

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

11.4.5 GPIO port input data register (GPIOx_IDR) (x = A..H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
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 bit (y = 0..15)
These bits are read-only. They contain the input value of the corresponding I/O port.

11.4.6 GPIO port output data register (GPIOx_ODR) (x = A..H)

Address offset: 0x14

Reset value: 0x0000 0000

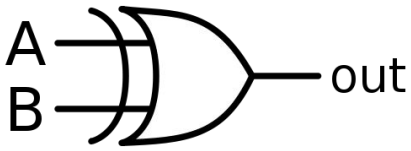
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:0 **ODRy:** Port output data bit (y = 0..15)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR or GPIOx_BRR registers (x = A..F).

TOGGLE BIT

This operation uses the principle that **X XOR 1 = ~X** and **X XOR 0 = X**.



Example here:

```
0b10101010 ^     0b10101010 ^     0b10101010 ^
0b00000001       0b00010000       0b1000011
-----------      -----------      -----------
0b10101011       0b10111010       0b00101001
```

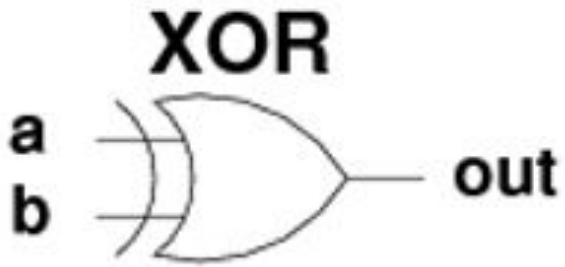
XOR Truth Table

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Concluding the operation **word ^ (1 << n)** toggle the n-th bit of "**word**".

PHÉP TOGGLE: ^=

Phép TOGGLE BIT là phép cài đặt 1 bit mong muốn trong thanh ghi cho nó giá trị **logic 0** nếu trước đó nó có giá trị logic 1 và ngược lại, đồng thời không làm thay đổi giá trị các bit còn lại của thanh ghi đó.

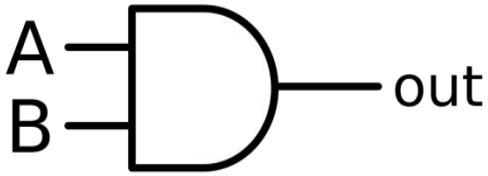


a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

How to check whether a bit is set or not in C language?

CHECK BIT VALUE

When we need to select a certain bit from a word we can still use **bit-masks**.



Example here:

AND Truth Table

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

```
0bXXXXXXXX &
0b00000001
------------
0b0000000X
```

```
0bXXXXXXXX &
0b00010000
------------
0b000X0000
```

```
0bXXXXXXXX &
0b00011000
------------
0b000XX000
```

Instructor

Eng. Nguyen Huynh Nhat Thuong