**MINISTRY OF EDUCATION AND TRAINING**
**HCMC UNIVERSITY OF TECHNICAL EDUCATION**
----------

HCMUTE

# FINAL REPORT

PRACTICE OF MICROPROCESSORS


**Lecturer:** PhD. Huỳnh Quang Duy

**Student:** Hoàng Đoàn Tiến Phát

**Student ID:** 20146279

**Class:** Wed-Thu


*Ho Chi Minh City, July 2024*

# TABLE OF CONTENTS

# CHAPTER 1: GENERAL PURPOSE INPUT AND OUTPUT

## 1.1. Cấp xung hoạt động cho modue GPIOx

Được quản lý bởi module RCC (Reset and Clock Control). Mỗi một module GPIO nếu muốn hoạt động được thì cần phải được cấp xung (clock) từ nguồn xung của hệ thống.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reser-ved | OTGHSULPIEN | OTGHSEN | ETHMACPTPEN | ETHMACRXEN | ETHMACTXEN | ETHMACEN | Reserved | | DMA2EN | DMA1EN | Reserved | | BKPSRAMEN | Reserved | |
| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | CRCEN | Reserved | | | GPIOIEN | GPIOHEN | GPIOGEN | GPIOFEN | GPIOEEN | GPIODEN | GPIOCEN | GPIOBEN | GPIOAEN |
| | | | rw | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

*Figure 1. RCC_AHB1ENR register*

## 1.2 Thiết lập các chức năng cho module GPIO

### 1.2.1 Đối với Cortex-M4 Series

Đầu tiên, ta phải chọn chế độ GPIO cho chân vật lý cụ thể bằng cách thay đổi giá trị trong thanh ghi GPIOx_MODER dưới đây.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER15[1:0] | | MODER14[1:0] | | MODER13[1:0] | | MODER12[1:0] | | MODER11[1:0] | | MODER10[1:0] | | MODER9[1:0] | | MODER8[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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

*Figure 2. GPIOx_MODER*

Tiếp đến, ta chọn tùy chọn đầu ra của chân đó nếu chân đó là chân output với 2 tùy chọn bao gồm push-pull và open-drain bằng cách thay đổi giá trị của thanh ghi GPIOx_OTYPER:

| 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 |
| OT15 | OT14 | OT13 | OT12 | OT11 | OT10 | OT9 | OT8 | OT7 | OT6 | OT5 | OT4 | OT3 | OT2 | OT1 | OT0 |
| 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 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

*Figure 3. GPIOx_OTYPER*

Ta có thể tùy chọn tốc độ thay đổi trạng thái đầu ra của chân đó bằng cách chỉnh sửa giá trị thanh ghi GPIOx_OSPEEDR

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OSPEEDR15 [1:0] | | OSPEEDR14 [1:0] | | OSPEEDR13 [1:0] | | OSPEEDR12 [1:0] | | OSPEEDR11 [1:0] | | OSPEEDR10 [1:0] | | OSPEEDR9 [1:0] | | OSPEEDR8 [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 |
| OSPEEDR7[1:0] | | OSPEEDR6[1:0] | | OSPEEDR5[1:0] | | OSPEEDR4[1:0] | | OSPEEDR3[1:0] | | OSPEEDR2[1:0] | | OSPEEDR1 [1:0] | | OSPEEDR0 1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **OSPEEDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output speed.
00: Low speed
01: Medium speed
10: High speed
11: Very high speed

*Note: Refer to the product datasheets for the values of OSPEEDRy bits versus $V_{DD}$ range and external load.*

*Figure 4. GPIOx_OSPEEDR*

Cuối cùng, ta sẽ chọn chế độ treo điện trở cao hoặc thấp của chân đó với thanh ghi GPIOx_PUPDR

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PUPDR15[1:0] | | PUPDR14[1:0] | | PUPDR13[1:0] | | PUPDR12[1:0] | | PUPDR11[1:0] | | PUPDR10[1:0] | | PUPDR9[1:0] | | PUPDR8[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 |
| PUPDR7[1:0] | | PUPDR6[1:0] | | PUPDR5[1:0] | | PUPDR4[1:0] | | PUPDR3[1:0] | | PUPDR2[1:0] | | PUPDR1[1:0] | | PUPDR0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

*Figure 5. GPIOx_PUPDR*

### 1.2.2 Đối với Cortex-M3 series

Đối với các vi điều khiển dòng Cortex-M3, việc cấu hình có phần dễ dàng hơn khi chỉ phải thay đổi giá trị trên thanh GPIOx_CRL và GPIOx_CRH tương ứng với các chân Px0 tới Px15

| 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 |

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
**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

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)
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.

*Figure 6. GPIOx_CRH*

### 1.3. Đọc và ghi tín hiệu logic

Để đọc giá trị logic đầu vào của chân cụ thể, ta có thể kiểm tra giá trị của bit tương ứng trên thanh ghi GPIOx_IDR. Với giá trị 1 tương ứng với mức logic HIGH và 0 tương ứng với mức logic LOW.

| 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 |

*Figure 7. GPIOx_IDR*

Còn để chỉnh giá trị logic trên một chân vật lý cụ thể, có 3 thanh ghi hỗ trợ thao tác này. Tuy nhiên, thanh ghi GPIOx_ODR dễ dàng thao tác và trực quan nhất. Chỉ cần thay đổi giá trị của bit tương ứng với chân vật lý đó, với giá trị 1 tương ứng mức logic HIGH và 0 tương ứng với mức logic LOW.

| 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 |

*Figure 8. GPIOx_ODR*

# CHAPTER 2: GENERAL TIMER

## 2.1. Lý thuyết về bộ định thời trên vi điều khiển

Bộ timer trên vi điều khiển có bản chất là một bộ counter với khoảng cách giữa hai lần đếm bằng với khoảng thời gian của một chu kì xung clock đầu vào của bộ timer đó. Khi đã đếm tới một mức đã cài trước (trong hình ảnh ví dụ dưới là 5) thì giá trị đếm sẽ được cài đặt lại về giá trị ban đầu, và bộ timer sẽ gửi yêu cầu ngắt tới vi điều khiển để thực hiện yêu cầu.

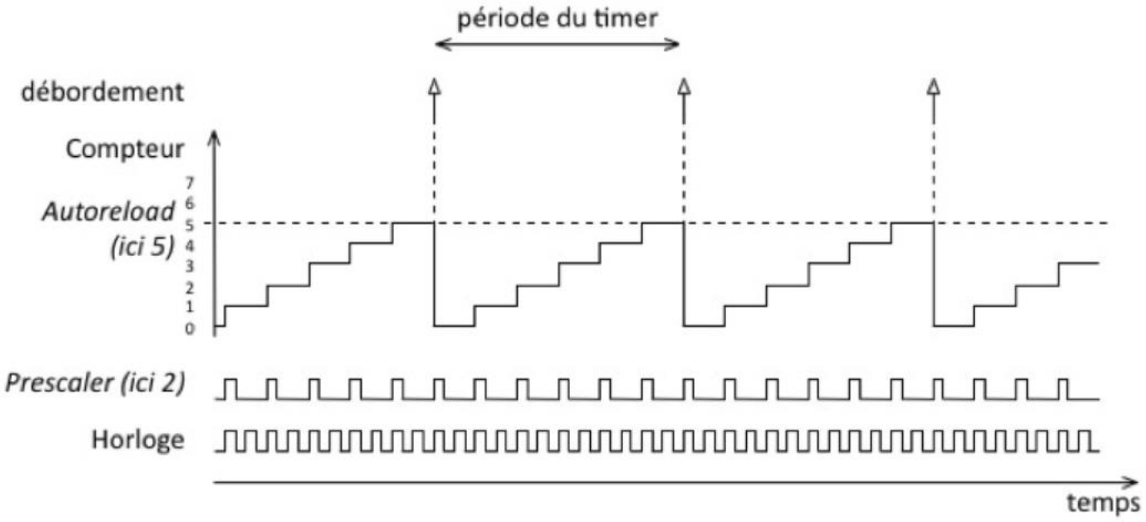


*Figure 9. Timer pipeline*

Chu kì của xung clock (khoảng thời gian giữa 2 lần đếm) được tính theo công thức như sau:

$$T = \frac{1}{f}$$

Với $T$ là chu kì giữa của xung clock và là khoảng thời gian giữa 2 lần đếm với đơn vị ms, và $f$ là tần số xung clock cấp vào bộ timer sau khi đã đi qua bộ chia của timer đó với đơn vị là MHz

## 2.2. Cấu hình cho bộ Timer chạy ở chế độ định thời

### 2.2.1. Cấp xung clock cho bộ Timer

Được quản lý bởi module RCC (Reset and Clock Control). Mỗi một module Timer nếu muốn hoạt động được thì cần phải được cấp xung (clock) từ nguồn xung của hệ thống.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | DAC EN | PWR EN | Reser-ved | CAN2 EN | CAN1 EN | Reser-ved | I2C3 EN | I2C2 EN | I2C1 EN | UART5 EN | UART4 EN | USART3 EN | USART2 EN | Reser-ved |
| | | 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 | | WWDG EN | 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 |

*Figure 10. RCC_APB1ENR*

### 2.2.2. Thiết lập bộ chia và giá trị đặt trước

Để thay đổi giá trị thời gian giữa 2 lần đếm của bộ timer, ta có thể sử dụng bộ chia của mỗi timer bằng cách thay đổi giá trị của thanh ghi TIMx_PSC (cần chú ý rằng giá trị bộ chia sẽ bằng giá trị của than h ghi TIMx_PSC trừ đi 1 đơn vị).

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSC[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK\_PSC}$ / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

*Figure 11. TIMx_PSC*

Ngoài ra, ra có thể thay đổi giá trị đếm tối đa bằng cách thay đổi giá trị của thanh ghi TIMx_ARR.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ARR[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.
Refer to *Section 17.3.1: Time-base unit* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

*Figure 12. TIMx_ARR*

### 2.2.3. Kích hoạt Timer

Sau khi các thông số cần thiết cho timer đã hoàn thành, tiến hành kích hoạt cho timer để có thể bắt đầu đếm với bit0 của thanh ghi TIMx_CR1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | CKD[1:0] | | ARPE | CMS | | DIR | OPM | URS | UDIS | CEN |
| | | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

*Figure 13. TIMx_CR1*

### 2.2.4. Thiết lập Timer interrupt

Mỗi khi bộ đếm của Timer bị tràn thì Timer sẽ cho phép 1 sự kiện ngắt (nếu như nó được cấu hình). Để kích hoạt ngắt cho timer đó, ta đầu tiên cần phải bật ngắt của chính nó bằng cách cho giá trị của bit0 trên thanh ghi TIMx_DIER.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Res. | TDE | Res | CC4DE | CC3DE | CC2DE | CC1DE | UDE | Res. | TIE | Res | CC4IE | CC3IE | CC2IE | CC1IE | UIE |
| | rw | | rw | rw | rw | rw | rw | | rw | | rw | rw | rw | rw | rw |

*Figure 14. TIMx_DIER*

Sau đó, ta sẽ kích hoạt ngắt toàn cục của timer đó trên thanh ghi NVIC_ISER. Bên dưới là ví dụ để kích hoạt ngắt toàn cục của TIM2

```
NVIC->ISER[0] |= (1<<28);   // enable global interrupt timer 2
```

Mỗi khi bộ đếm của Timer bị tràn thì con trỏ chương trình nhảy vào vector ngắt của timer để thực hiện. Một vector ngắt sẽ có cấu trúc như sau.

```
/// timerx interrupt vector
void TIMx_IRQHandler (void)       // x = 1, 2, 3, 4,…..
{
        TIMx->SR &= ~(1<< 0);     //clear timerx interrupt flag
        //placed code begin

        //placed code end
}
```

*Figure 15. Hàm để thiết lập vector ngắt cho Timer*

# CHAPTER 3: PULSE-WIDTH MODULATION

## 3.1. Lý thuyết về điều chế độ rộng xung

Ngoài việc định thời, timer trên vi điều khiển còn có chức năng điều chế độ rộng xung. Chức năng này cho phép điều chỉnh điện áp tải, hay hiểu đơn giản hơn đây là phương pháp điều chỉnh, thay đổi điện áp tải ra bằng việc thay đổi độ rộng của chuỗi xung vuông, từ đó có sự thay đổi điện áp.

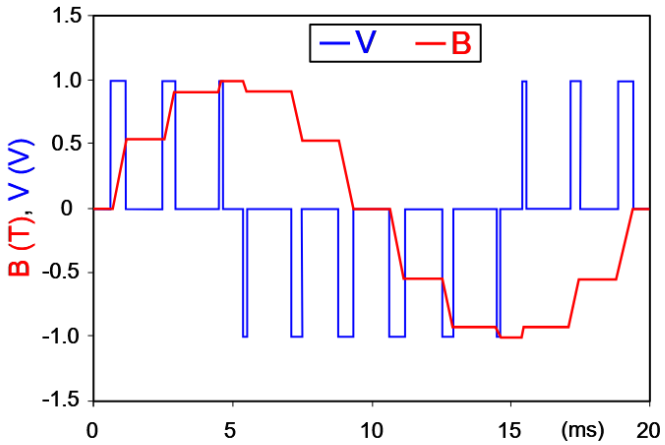


*Figure 16. Điều chế độ rộng xung*

Trên vi điều khiển, ta có thể sử dụng bộ timer để điều chế độ rộng xung bằng cách đặt ra một mức duty cycle (đường ngang màu đỏ trong hình dưới). Mỗi khi timer đếm đếm giá trị đó hoặc timer đếm đến giá trị đếm tối đa ARR, mức logic đầu ra sẽ được thay đổi.

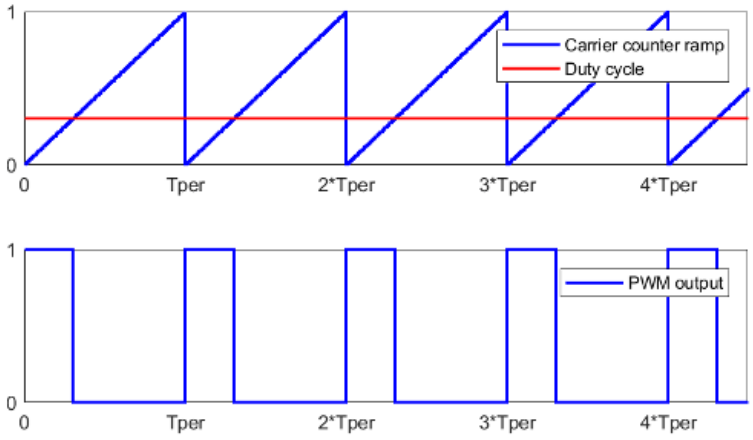


*Figure 17. Sự thay đổi của Duty Cycle*

### 3.2. Cấu hình cho bộ Timer chạy ở chế độ PWM

#### 3.2.1. Cấp xung cho bộ Timer

Được quản lý bởi module RCC (Reset and Clock Control). Mỗi một module Timer nếu muốn hoạt động được thì cần phải được cấp xung (clock) từ nguồn xung của hệ thống.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | DAC EN | PWR EN | Reser-ved | CAN2 EN | CAN1 EN | Reser-ved | I2C3 EN | I2C2 EN | I2C1 EN | UART5 EN | UART4 EN | USART3 EN | USART2 EN | Reser-ved |
| | | 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 | | WWDG EN | 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 |

*Figure 17. RCC_APB1ENR*

#### 3.2.2. Thiết lập bộ chia và giá trị đặt trước

Để thay đổi giá trị thời gian giữa 2 lần đếm của bộ timer, ta có thể sử dụng bộ chia của mỗi timer bằng cách thay đổi giá trị của thanh ghi TIMx_PSC (cần chú ý rằng giá trị bộ chia sẽ bằng giá trị của than h ghi TIMx_PSC trừ đi 1 đơn vị).

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSC[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK\_PSC}$ / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

*Figure 18. TIMx_PSC*

Ngoài ra, ra có thể thay đổi giá trị đếm tối đa bằng cách thay đổi giá trị của thanh ghi TIMx_ARR.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ARR[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.
Refer to *Section 17.3.1: Time-base unit* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

*Figure 19. TIMx_ARR*

### 3.2.3. Kích hoạt Timer

Sau khi các thông số cần thiết cho timer đã hoàn thành, tiến hành kích hoạt cho timer để có thể bắt đầu đếm với bit0 của thanh ghi TIMx_CR1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | CKD[1:0] | | ARPE | CMS | | DIR | OPM | URS | UDIS | CEN |
| | | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

*Figure 20. TIMx_CR1*

### 3.2.4. Cài đặt giá trị Duty Cycle khởi đầu

Mỗi timer sẽ có 4 kênh PWM bao gồm TIMx_CH1, TIMx_CH2, TIMx_CH3 và TIMx_CH4. Để chỉnh giá trị Duty Cycle cho các kênh đó, ta có thể thay đổi giá trị của các thanh ghi TIMx_CCR1, TIMx_CCR2, TIMx_CCR3 và TIMx_CCR4, với giá trị từ 0 tới bằng với giá trị của thanh TIMx_ARR (tương ứng với 0% tới 100%).

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CCR1[15:0] | | | | | | | | | | | | | | | |
| rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro |

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output**:
CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).
It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.
The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC1 output.

**If channel CC1 is configured as input**:
CCR1 is the counter value transferred by the last input capture 1 event (IC1). The TIMx_CCR1 register is read-only and cannot be programmed.

*Figure 21. TIMx_CCR1*

### 3.2.5. Tùy chỉnh chế độ PWM

Ở trong môn học này, ta sẽ chỉ quan tâm tới 2 chế độ phổ biến nhất của PWM là mode 1 và mode 2. Với mode 1, mức logic đầu ra sẽ là active HIGH, nghĩa là khi đang ở Duty Cycle, mức logic trên chân đầu ra sẽ là HIGH. Còn đối với mode 2 thì ngược lại với mode 1. Khi đang ở Duty Cycle thì mức logic trên chân đầu ra sẽ là LOW. Hình dưới mô tả 2 chế độ PWM kể trên với đường màu đỏ bên trên là PWM mode 1 còn đường màu đỏ bên dưới là PWM mode 2.

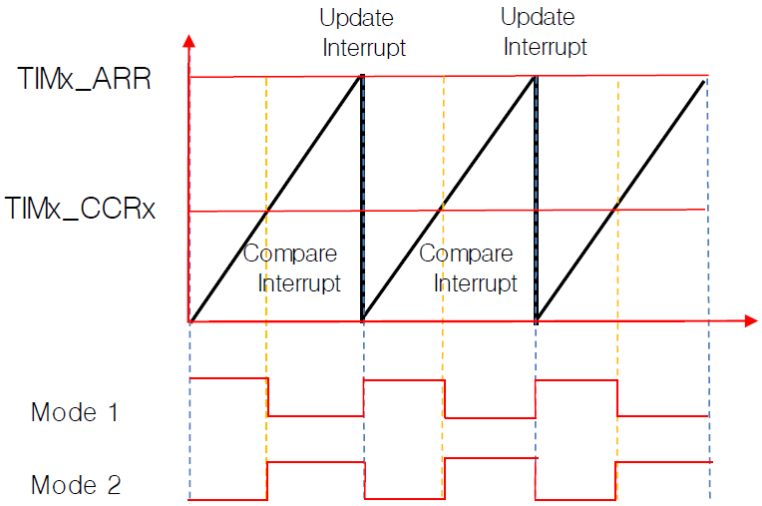


*Figure 22. PWM mode 1 and mode 2*

Để cài đặt chế độ PWM cho các kênh cụ thể, ta có thể sử dụng các bit4 tới bit6 trên 2 thanh ghi TIMx_CCMR1 và TIMx_CCMR2 tương ứng với các kênh TIMx_CH1, TIMx_CH2, TIMx_CH3 và TIMx_CH4

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OC4 CE | OC4M[2:0] | | | OC4 PE | OC4 FE | CC4S[1:0] | | OC3 CE. | OC3M[2:0] | | | OC3 PE | OC3 FE | CC3S[1:0] | |
| IC4F[3:0] | | | | IC4PSC[1:0] | | | | IC3F[3:0] | | | | IC3PSC[1:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.
000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).
001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.
100: Force inactive level - OC1REF is forced low.
101: Force active level - OC1REF is forced high.
110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF='1').
111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

*Figure 23. TIMx_CCMR1*

### 3.2.6. Kích hoạt các kênh PWM

Sau khi đã hoàn tất thiết lập các kênh, ta chỉ cần bật các kênh đó bằng cách thay đổi giá trị của các bit0, bit4, bit8 và bit12 trên thanh ghi TIMx_CCER để có thể kích hoạt các kênh TIMx_CH1, TIMx_CH2, TIMx_CH3 và TIMx_CH4.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CC4NP | Res. | CC4P | CC4E | CC3NP | CC3NE | CC3P | CC3E | CC2NP | CC2NE | CC2P | CC2E | CC1NP | CC1NE | CC1P | CC1E |
| rw | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

*Figure 24. TIMx_CCER*

### 3.3. Cấu hình cho các chân vật lý để chạy ở chế độ PWM

#### 3.3.1. Đối với Cortex-M4 Series

Đầu tiên ta sẽ kích hoạt các chân vật lý bằng cách bật nguồn clock cho các module của các port thông qua việc thay đổi giá trị các bit0 tới bit8 tương ứng với GPIOA tới GPIOE trên thanh ghi RCC_AHB1EN

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reser-ved | OTGHS ULPIEN | OTGHSEN | ETHMACPTPEN | ETHMACRXEN | ETHMACTXEN | ETHMACEN | Reserved | | DMA2EN | DMA1EN | CCMDATARAMEN | Res. | BKPSRAMEN | Reserved | |
| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | CRCEN | Reserved | | | GPIOIEN | GPIOHEN | GPIOGEN | GPIOFEN | GPIOEEN | GPIODEN | GPIOCEN | GPIOBEN | GPIOAEN |
| | | | rw | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

*Figure 24. RCC_AHB1EN*

Sau đó, ta sẽ cài đặt cho các chân chạy ở chế độ Alternate Function bằng cách ghi giá trị 0b10 lên 2 bit tương ứng với chân mà ta muốn cài đặt trên thanh ghi GPIOx_MODER.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER15[1:0] | | MODER14[1:0] | | MODER13[1:0] | | MODER12[1:0] | | MODER11[1:0] | | MODER10[1:0] | | MODER9[1:0] | | MODER8[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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

*Figure 25. GPIOx_MODER*

Cuối cùng, ta chỉ cần chọn đúng chế độ Alternate Function cần thiết bằng cách ghi giá trị lên thanh ghi GPIOx_AFRL và GPIOx_AFRH.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AFRL7[3:0] | | | | AFRL6[3:0] | | | | AFRL5[3:0] | | | | AFRL4[3:0] | | | |
| 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 |
| AFRL3[3:0] | | | | AFRL2[3:0] | | | | AFRL1[3:0] | | | | AFRL0[3:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AFRH15[3:0] | | | | AFRH14[3:0] | | | | AFRH13[3:0] | | | | AFRH12[3:0] | | | |
| 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 |
| AFRH11[3:0] | | | | AFRH10[3:0] | | | | AFRH9[3:0] | | | | AFRH8[3:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:0 **AFRHy:** Alternate function selection for port x bit y (y = 8..15)
These bits are written by software to configure alternate function I/Os

AFRHy selection:

| | |
|---|---|
| 0000: AF0 | 1000: AF8 |
| 0001: AF1 | 1001: AF9 |
| 0010: AF2 | 1010: AF10 |
| 0011: AF3 | 1011: AF11 |
| 0100: AF4 | 1100: AF12 |
| 0101: AF5 | 1101: AF13 |
| 0110: AF6 | 1110: AF14 |
| 0111: AF7 | 1111: AF15 |

*Figure 26. GPIOx_AFRL và GPIOx_ AFRH*

Để có thể xác định được cần phải ghi giá trị gì vào thanh ghi đó, ta có thể xem qua hai hình dưới đây.

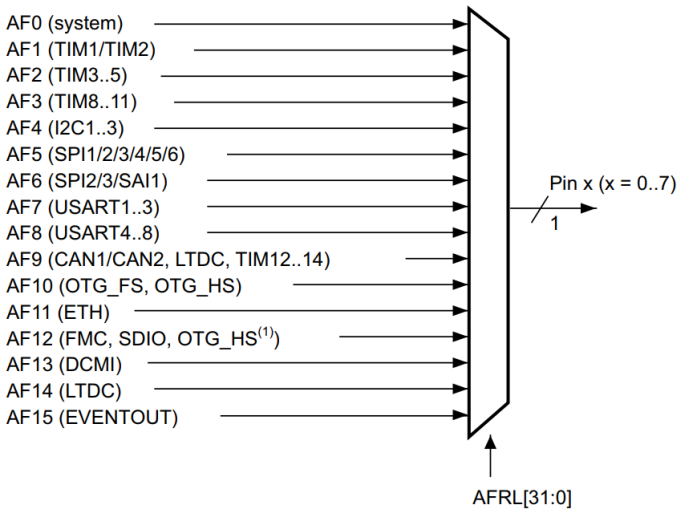


*Figure 27. ý nghĩa giá trị của GPIOx_AFRL*

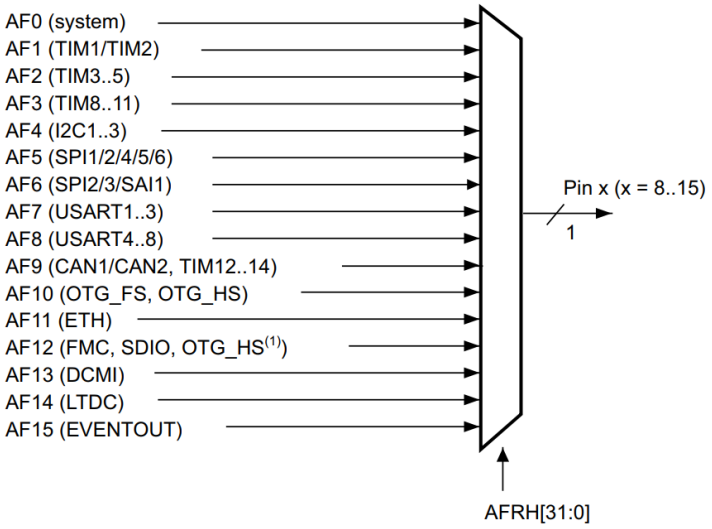


*Figure 28. ý nghĩa giá trị của GPIOx_AFRH*

# CHAPTER 4: UNIVERSAL ASYNCHRONOUS RECEIVER AND TRANSMITTER

## 4.1. Lý thuyết về giao thức truyền thông UART

UART là một chuẩn giao tiếp có dây không đồng bộ giúp truyền dữ liệu serial qua lại giữa 2 thiết bị. Chuẩn giao tiếp này yêu cầu 3 chân vật lý bao gồm chân GND được nối với nhau và hai chân dữ liệu Rx và Tx được nối bắt chéo nhau như hình dưới.

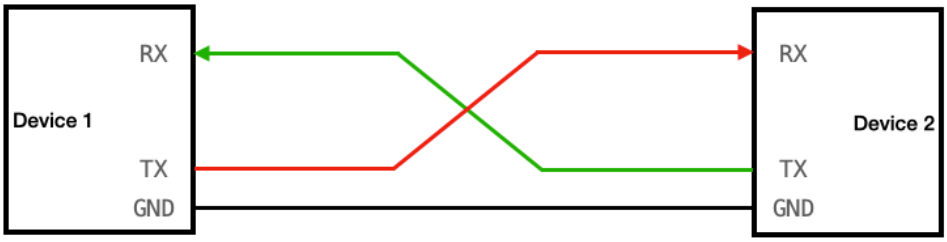


*Figure 29. Giao thức truyền thông UART*

Khi ở trạng thái chờ, dây tín hiệu sẽ có mức logic HIGH. Khi truyền dữ liệu, UART sẽ gửi một frame dữ liệu. Một frame dữ liệu cơ bản của UART bao gồm các bit với các chức năng như sau:

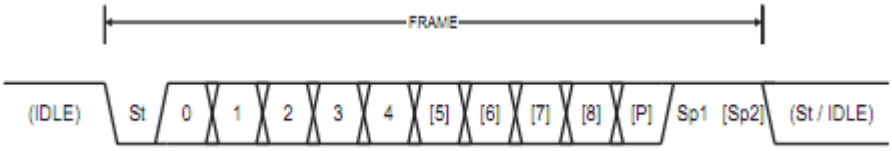


*Figure 30. Data frame*

- **Start bit**: luôn có giá trị LOW, giúp chỉ định điểm bắt đầu của một khung dữ liệu
- **Data frame**: chứa dữ liệu muốn truyền nhận giữa hai thiết bị. Một data frame có thể có 5 tới 8 bit và đôi khi có thể lên tới 9 bit dữ liệu, Nhưng thông thường sẽ chỉ có 8 bit trong một data frame.
- **Parity bit**: (không bắt buộc) dùng để nhận biết liệu có bất kỳ dữ liệu nào đã thay đổi trong quá trình truyền hay không. Bit có thể bị thay đổi bởi nhiễu tín hiệu, tốc độ truyền không khớp hoặc truyền dữ liệu khoảng cách xa.
- **Stop bit**: báo hiệu sự kết thúc của gói dữ liệu, UART gửi sẽ điều khiển đường truyền dữ liệu từ điện áp thấp đến điện áp cao trong ít nhất khoảng 2 bit.

UART là giao thức không đồng bộ, do đó không có đường clock nào điều chỉnh tốc độ truyền dữ liệu. Người dùng phải đặt cả hai thiết bị để giao tiếp ở cùng tốc độ. Tốc độ này được gọi là tốc độ truyền, được biểu thị bằng bit trên giây hoặc bps. Tốc độ

truyền thay đổi đáng kể, từ 9600 baud đến 115200 và hơn nữa. Tốc độ truyền giữa UART truyền và nhận chỉ có thể chênh lệch khoảng 10% trước khi thời gian của các bit bị lệch quá xa.

## 4.2. Cấu hình cho bộ UART

### 4.2.1. Cấp xung clock đầu vào cho bộ UART

Mỗi một module UART nếu muốn hoạt động được thì cần phải được cấp xung clock từ nguồn xung của hệ thống thông qua việc thay đổi giá trị trên thanh ghi RCC_ APB2ENR.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | TIM11 EN | TIM10 EN | TIM9 EN |
| | | | | | | | | | | | | | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reser-ved | SYSCFG EN | Reser-ved | SPI1 EN | SDIO EN | ADC3 EN | ADC2 EN | ADC1 EN | Reserved | | USART6 EN | USART1 EN | Reserved | | TIM8 EN | TIM1 EN |
| | rw | | rw | rw | rw | rw | rw | | | rw | rw | | | rw | rw |

*Figure 31. RCC_APB2ENR*

### 4.2.2. Thiết lập baudrate

Để thiết lập tốc độ truyền dữ liệu của UART, đầu tiên ta cần xác định giá trị cần thiết của bộ chia của mô đun UART bằng công thức bên dưới. với Baudrate là tốc độ truyền mong muốn (thông thường là 9600bps), Clock Frequency là tần số của xung clock đầu vào, và UART Div là giá trị của bộ chia của mô đun UART.

❖ ***Chế độ 16 bit (mặc định)*** $\text{Baudrate} = \frac{\text{Clock Frequency}}{(8 * 2)(\text{USART Div})}$ ***If OVER8 = 0.***

❖ ***Chế độ 8 bit*** $\text{Baudrate} = \frac{\text{Clock Frequency}}{(8)(\text{USART Div})}$ ***If OVER8 = 1***

Sau khi đã tính ra được giá trị bộ chia của mô đun UART là một số thực, ta sẽ chỉnh bộ chia bằng cách thay đổi giá trị của thanh ghi UARTx_BRR. Thanh ghi này gồm 2 phần, bit4 tới bit15 dùng để ghi giá trị phần nguyên của bộ chia, còn bit0 tới bit3 dùng để ghi giá trị của phần dư của bộ chia đã tính nhân với 16.

| 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 |
| DIV_Mantissa[11:0] | | | | | | | | | | | | DIV_Fraction[3:0] | | | |
| 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 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

*Figure 32. UARTx_BRR*

Dưới đây là một ví dụ để tính và cài đặt giá trị của bộ chia.

Example: Tốc độ truyền 9600 bit/s, Clock Freq = 16Mhz.

$$\text{Baund Rate} = \frac{\text{Clock Frequency}}{(8 * 2)(\text{USART Div})}$$

=> USART Div = 104.17

- Mantissa = 104
- Fraction = 0.17*16 = 2.72 => Fraction = 3

### 4.2.3. Cài đặt khung dữ liệu

Như đã nói ở phần trước, một khung dữ liệu cơ bản của UART bao gồm: Start bit, Data bits, Parity bit và Stop bit. Để cấu hình khung dữ liệu của UART, ta sẽ sử dụng hai thanh ghi UARTx_CR1 và UARTx_CR2. Đầu tiên ta sẽ chọn khung dữ liệu mong muốn bằng cách thay đổi giá trị của bit10 và bit12 trong thanh UARTx_CR1.

| 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 |
| OVER8 | Reserved | UE | M | WAKE | PCE | PS | PEIE | TXEIE | TCIE | RXNEIE | IDLEIE | TE | RE | RWU | SBK |
| rw | Res. | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bit 12 **M**: Word length
This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 Data bits, n Stop bit
1: 1 Start bit, 9 Data bits, n Stop bit
*Note: The M bit must not be modified during a data transfer (both transmission and reception)*

Bit 10 **PCE**: Parity control enable
This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).
0: Parity control disabled
1: Parity control enabled

Sau đó ta sẽ chọn số lượng Stop bit bằng cách thay đổi giá trị bit12 và bit13 trên thanh ghi UARTx_CR2.

| 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 |
| Res. | LINEN | STOP[1:0] | | CLKEN | CPOL | CPHA | LBCL | Res. | LBDIE | LBDL | Res. | ADD[3:0] | | | |
| | rw | rw | rw | rw | rw | rw | rw | | rw | rw | rw | rw | rw | rw | rw |

Bits 13:12 **STOP**: STOP bits

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

*Note: The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.*

*Figure 33. UARTx_CR2*

### 4.2.4. Kích hoạt mô đun UART

Ở phần này, ta sẽ kích hoạt chế độ gửi và nhận cùng với kích hoạt UART để bắt đầu hoạt động với việc thay đổi giá trị các bit2, bit3 và bit13 của thanh ghi UARTx_CR1

| 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 |
| OVER8 | Reserved | UE | M | WAKE | PCE | PS | PEIE | TXEIE | TCIE | RXNEIE | IDLEIE | TE | RE | RWU | SBK |
| rw | Res. | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

*Note: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.*

*When TE is set, there is a 1 bit-time delay before the transmission starts.*

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

Bit 13 **UE**: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: USART prescaler and outputs disabled

1: USART enabled

*Figure 34. UARTx_CR1*

### 4.2.5. Thiết lập ngắt của UART

Cuối cùng, ta sẽ kích hoạt ngắt cho Tx và Rx. Phần này chỉ là tùy chọn thêm. Tuy nhiên, thông thường thì việc gửi dữ liệu đi sẽ được vi điều khiển chủ động chọn thời điểm nên việc thiết lập ngắt Tx hầu như là không cần thiết trừ khi có ý đồ cụ thể. Còn việc nhận dữ liệu sẽ là hành động bị động, nên ta sẽ cần phải thiết lập ngắt cho vi điều khiển để có thể hành động bất kì khi nào có dữ liệu Rx nhận về. Để kích hoạt ngắt Rx, ta đơn giản ghi giá trị bit5 của thanh ghi UARTx_CR1 lên mức high.

| 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 |
| OVER8 | Reserved | UE | M | WAKE | PCE | PS | PEIE | TXEIE | TCIE | RXNEIE | IDLEIE | TE | RE | RWU | SBK |
| rw | Res. | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bit 5 **RXNEIE**: RXNE interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_SR register

*Figure 35. UARTx_CR1*

Sau đó ta sẽ kích hoạt vector ngắt toàn cục của mô đun UART đó trên thanh ghi NVIC_ISER.

### 4.3. Cấu hình các chân Tx Rx

Khai báo chế độ hoạt động cho vi điều khiển tương ứng với module USART. Cần chú ý là một ngoại vi nên chủ yếu khai báo các chân ở chế độ Alternate Function (nếu có).

| | USART1 | USART2 | USART3 |
|---|---|---|---|
| **TX pin** | **PA9** | **PA2** | **PB10** |
| **RX pin** | **PA10** | **PA3** | **PB11** |

*Figure 35. Các chân Tx Rx trên vi điều khiển*

Đối với các dòng vi điều khiển sử dụng kiến trúc Cortex-M3 như F1 series, ta sẽ cấu hình chân Tx là Output Alternate Function Mode và chân Rx là Input Floating Mode với các thanh ghi GPIOx_CRL và GPIOx_CRH.

Đối với các dòng vi điều khiển sử dụng kiến trúc Cortex-M4 như F2,3,4 series, ta sẽ cấu hình chân Tx và Rx là Alternate Function Mode với thanh ghi GPIOx_MODER. Sau đó sẽ chọn AF Mode cụ thể với thanh ghi GPIOx_AFRL và GPIOx_AFRH.

### 4.4. Truyền dữ liệu

Để truyền một khung dữ liệu Tx, ta sẽ làm theo hai bước:

- Ghi 8 bit dữ liệu vào thanh ghi UARTx_DR để truyền đi
- Kiểm tra xem đã truyền thành công chưa với bit7 của thanh ghi UARTx_SR

Đối với việc truyền một chuỗi các byte dữ liệu, ta chỉ đơn giản là lặp lại 2 hành động trên với lần lượt từng byte một trong chuỗi đó.

| 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 | | | | | | CTS | LBD | TXE | TC | RXNE | IDLE | ORE | NF | FE | PE |
| | | | | | | rc_w0 | rc_w0 | r | rc_w0 | rc_w0 | r | r | r | r | r |

Bit 7 **TXE**: Transmit data register empty
This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by a write to the USART_DR register.
0: Data is not transferred to the shift register
1: Data is transferred to the shift register)
*Note: This bit is used during single buffer transmission.*

*Figure 36. UARTx_SR*

### 4.5. Nhận dữ liệu

Việc nhận chuỗi dữ liệu Rx sẽ có phần phức tạp hơn, khi ta sẽ phải dùng tới ngắt Rx của UART để phát hiện có dữ liệu nhận về. Mỗi khi nhận về một byte dữ liệu, chương trình sẽ bị ngắt và thực hiện chương trình trong vector ngắt.

```
void USART1_IRQHandler (void)
{
    // Clear Rx interupt flag
    USART1->SR &= ~(1UL<<5);
    // Get a string of data
    if (USART1->DR != ';'){
        Rx_Buffer[Rx_count] = USART1->DR;
        Rx_count++;
    } else {
        Rx_Buffer[Rx_count] = '\0';
        Rx_count = 0;
        // Estimate Rx data
        sscanf(Rx_Buffer, "red %d green %d", &red_value, &green_value);
    }
}
```

Figure 37. Ví dụ nhận chuỗi Rx

Một cách để xử lý việc đọc chuỗi Rx đó là sử dụng kí tự kết thúc chuỗi. trong ví dụ bên trên, mỗi khi hàm ngắt được chạy thì nó sẽ nhận và lưu giá trị của một byte dữ liệu nhận được vào một Buffer. Chỉ khi byte nhận được là kí tự kết thúc chuỗi (trong ví dụ là dấu chấm phẩy) thì mới reset lại biến đếm và xử lý chuỗi Rx đó.