

MODEL 4. POSITION CONTROL FOR DC SERVO MOTORS

4.1 Contents

- ✓ Empirical modeling of a single axis servo system.
- ✓ Using Matlab/Simulink to illustrate the system's performance.
- ✓ Programming using STM32F103 to read encoder signals and implement the PI/PD controller.
- ✓ Verify position control of a DC servo motor in a real model.

4.2. One axis servo system using dc servo motors

4.2.1 DC motor dynamics

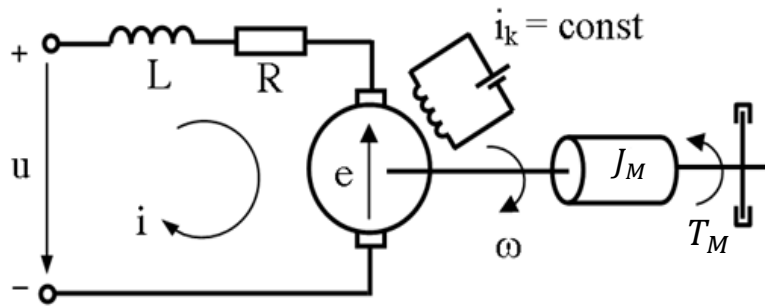


Fig. 4.1: The equivalent circuit of a DC servo motor

The electrical system: $u(t) = Ri + L \frac{di}{dt} + e$ (4.1)

where, $e = K_e \omega$

The mechanical system: $J_M \frac{d\omega}{dt} + b\omega = T_M$ (4.2)

where, $T_M = K_m i$

where,

u : voltage input [V]; ω : angular velocity [rad/s]

R : armature resistance [Ω]; L : armature inductance [H]

K_e : the back emf constant [V/rad/s]; K_m : torque constant [Nm/A]

J_M : inertial moment of motor shaft [kg.m²]

T_M : torque of motor [Nm]

b : viscous damping [Nm.s]

The block diagram of a DC servo motor:

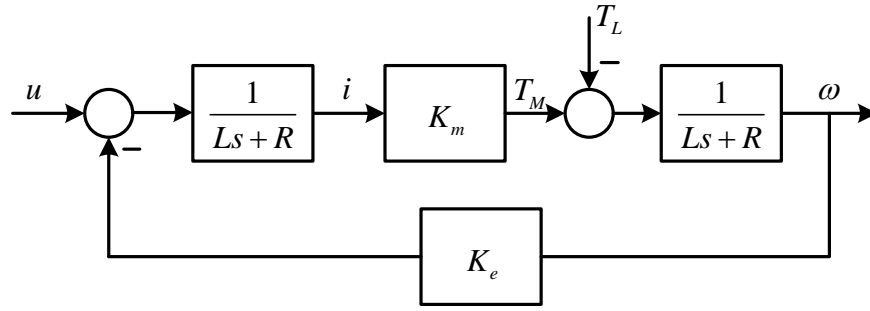


Fig. 4.2: The block diagram of a DC servo motor

The transfer function of a motor is obtained:

$$G(s) = \frac{\omega(s)}{U(s)} = \frac{K_m}{(Ls + R)(Js + b) + K_m K_e} \quad (4.3)$$

$$G(s) = \frac{\omega(s)}{U(s)} \approx \frac{K_m / Rb}{\left(\frac{L}{R}s + 1\right)\left(\frac{J}{b}s + 1\right)} = \frac{K}{(\tau_e s + 1)(\tau_m s + 1)} \quad (4.4)$$

where, $\tau_e = \frac{L}{R}$: electrical time constant (s)

$\tau_m = \frac{J}{b}$: mechanical time constant (s)

normally, $\tau_m \gg \tau_e$

Then we can approximate eq. (4) by a first order transfer function

$$G(s) = \frac{\omega(s)}{U(s)} \approx \frac{K}{\tau_m s + 1} \quad (4.5)$$

4.2.2. One axis servo system using DC servo motor

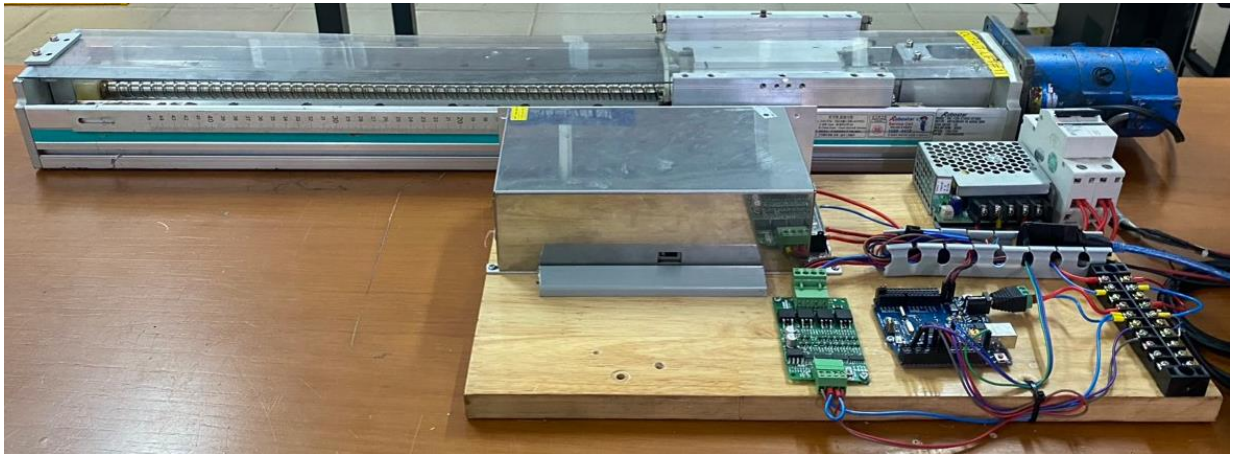


Fig. 4.3: The model of one-axis servo systems

4.2.3 The general diagram

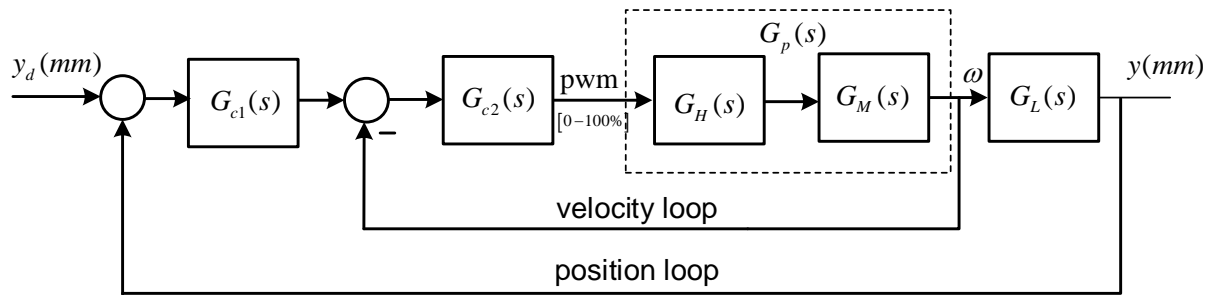


Fig. 4.4: The general diagram of the controlled systems

where, G_{c1} : P/PD controller; G_{c2} : PI controller

G_H : The transfer function of the H-bridge

G_M : The transfer function of the motor

G_L : The transfer function of the load.

4.3. Quadrature encoder

4.3.1 Incremental encoder

Rotary encoder is a sensor attached to a rotating object (such as a shaft or motor) to measure rotation. By measuring rotation, we can determine any displacement, velocity, acceleration, or the angle of a rotating object.

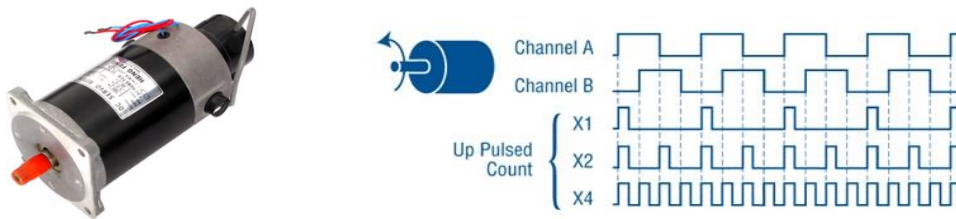


Fig. 4.5: Appearance of a DC servo motor and encoder output signals

➤ **1X Encoding: using 1 external interrupt for channel A or B**

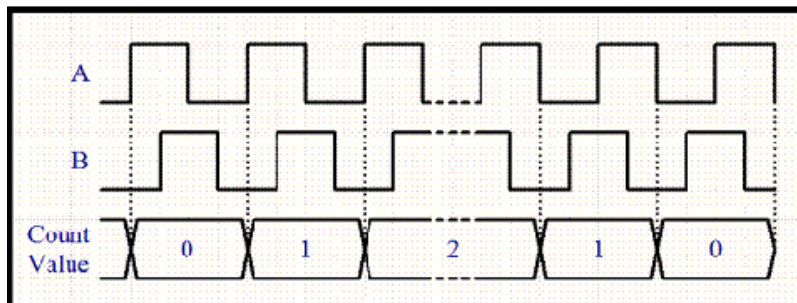


Fig. 4.6: Pulse diagram of 1x encoding

➤ **2X Encoding: using 1 external interrupt for channel A or B**

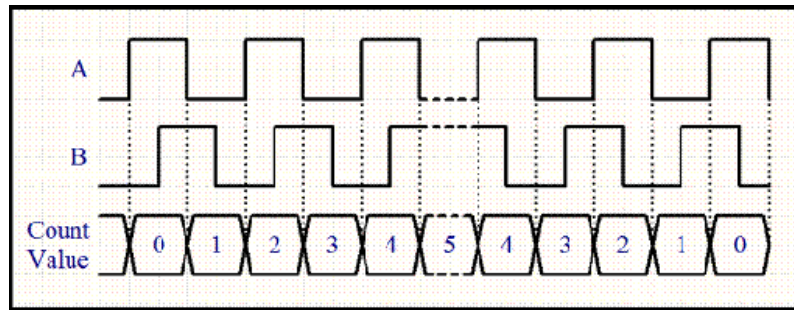


Fig. 4.7: Pulse diagram of 2x encoding

➤ **4X Encoding:** using 2 external interrupts for 2 channels A and B

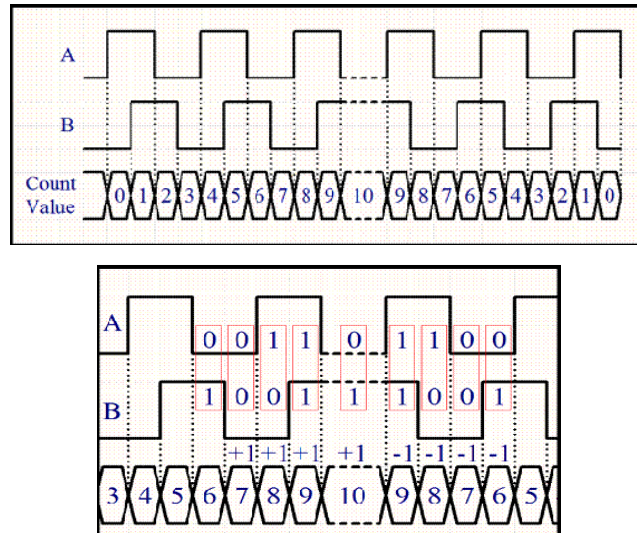


Fig. 4.8: Pulse diagram of 4x encoding

From the above pulses, we can draw out a table that describes the transition states of the two channels A and B:

Table 4.2: The transition states of channel A and B

State	Clockwise-Transition (CW)	Counter-Clockwise Transition (CCW)
0, 0	(0, 1) to (0, 0)	(1, 0) to (0, 0)
1, 0	(0, 0) to (1, 0)	(1, 1) to (1, 0)
1, 1	(1, 0) to (1, 1)	(0, 1) to (1, 1)
0, 1	(1, 1) to (0, 1)	(0, 0) to (0, 1)

The state transition diagram:

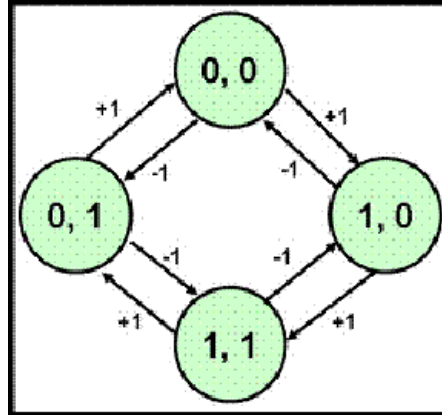


Fig. 4.9: The state transition diagram

The figure below (Fig. 4.10) is the code example of 4x encoder reading for channel A. To complete reading encoder in 4x mode, we have to add another external interrupt for channel B, the code for channel B is almost similar.

Note that: - Channel A is connected to GPIOB, PIN 4
 - Channel B is connected to GPIOB, PIN 6

4.3.2. Calculating velocity/position

We use the following equation for estimating velocity of the motor:

$$\omega = \frac{60 \times CountValue}{T \times MaxCnt} \text{ (RPM)} \quad (4.6)$$

where, *CountValue* is the number of pulses counted in *T* (s). *T* is also timer interrupt for reading encoder, and in this experiment, *T* is configured as 0.005 (s). *MaxCnt* equals to encoder resolution multiplied by its mode (x1, x2 or x4).

```

void EXTI4_IRQHandler(void) // doc encoder
{
    /* USER CODE BEGIN EXTI4_IRQn 0 */
    // CHANNEL A
    unsigned char Statel;
    Statel = (Statel<<1) | HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
    Statel = (Statel<<1) | HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_6);
    Statel = Statel&0x03;
    switch (Statel) {
        case 0:
            if(PreviousState==1) CountValue++;
            else CountValue--;
            break;
        case 1:
            if(PreviousState==3) CountValue++;
            else CountValue--;
            break;
        case 2:
            if(PreviousState==0) CountValue++;
            else CountValue--;
            break;
        case 3:
            if(PreviousState==2) CountValue++;
            else CountValue--;
            break;
    }
    PreviousState = Statel;
    CntVel++;
    if (CountValue>=4000) {
        CountValue = 0;
        PosCnt++;
    }
    else if (CountValue<=-4000) {
        CountValue = 0;
        PosCnt--;
    }
    /* USER CODE END EXTI3_IRQn 0 */
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
    /* USER CODE BEGIN EXTI3_IRQn 1 */
}

```

Fig. 4.10: Code example of channel A encoding in 4x mode

From Fig. (4.10), we can see that **PosCnt** is the number of rounds of the motor at the moment. **CountValue** is the number of pulses of the present rotation, CountValue will be reset when the rotation is completed. Therefore, we can calculate the position of the motor as follow:

$$pos = PosCnt \times 2\pi + \frac{CountValue}{MaxCnt} \times 2\pi \quad (rad) \quad (4.7)$$

Based on the rotation of the motor (rad), Eq. (4.7), students have to calculate the translation of the load (mm).

4.4. PID controller

4.4.1 PID calculation

The mathematic equation of the PID controller:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt} \quad (4.8)$$

When implementing the PID controller in practice, the input variable (error) is obtained by sampling the plant's output at the sample rate. Then, the PID algorithm is also calculated at the same rate. At the step k^{th} , we have:

$$u_k = u_k^P + u_k^D + u_k^I \quad (4.9)$$

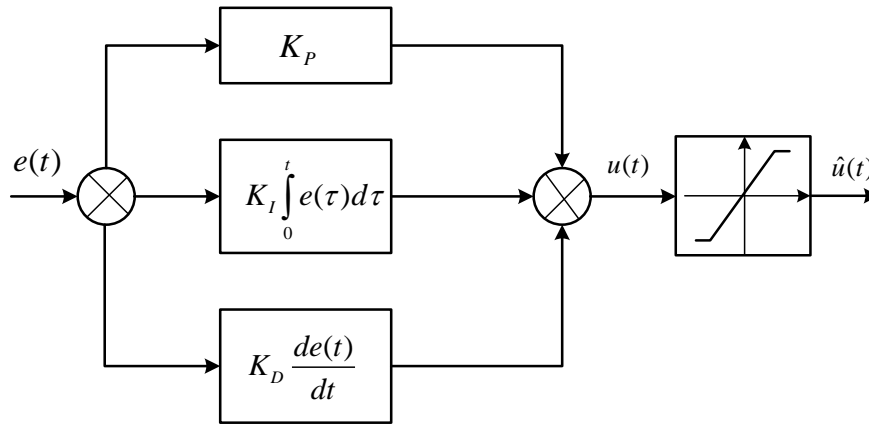


Fig 4.11: Block diagram of a PID controller

➤ P Calculation

$$u_k^P = K_P e_k \quad (4.10)$$

➤ D calculation

$$u_k^D = K_D \frac{e_k - e_{k-1}}{T} \quad (4.11)$$

➤ I calculation

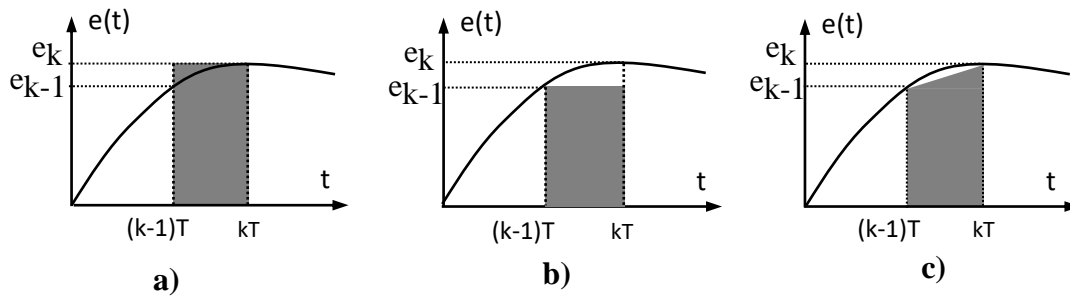


Fig 4.12: Integral approximation methods

a) Backward rectangular approximation (backward Euler):

$$u_k^I = K_I \sum_{i=1}^k T e_i = u_{k-1}^I + K_I T e_k \quad (4.12)$$

b) Forward rectangular approximation (forward Euler):

$$u_k^I = K_I \sum_{i=1}^k T e_{i-1} = u_{k-1}^I + K_I T e_{k-1} \quad (4.13)$$

c) Trapezoidal approximation:

$$u_k^I = K_I \sum_{i=1}^k T \frac{e_{i-1} + e_i}{2} = u_{k-1}^I + K_I T \frac{e_{k-1} + e_k}{2} \quad (4.14)$$

Code example:

```
int PIDVel(float DesiredValue, float CurrentValue)
{
    static float err_p=0;
    static float ui_p=0;
    float err, up, ud, ui;
    int uout;

    err = DesiredValue-CurrentValue;

    up = Kp*err;
    ud = Kd*(err-err_p)/sampletime;
    ui = ui_p+Ki*err*sampletime;

    err_p = err;
    ui_p = ui;

    uout = (int) (up+ud+ui);
    if (uout>HILIM)
        uout=HILIM;
    else if (uout<LOLIM)
        uout=LOLIM;
    return uout;
}
```

Fig 4.13: Code example of a simple PID algorithm

4.4.2 Low pass filter for D term:

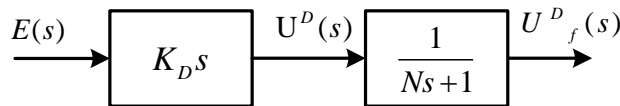


Fig 3.14: Low pass filter for D term

$$u_f^D(k) = \frac{N}{N+T} u_f^D(k-1) + \frac{T}{N+T} u^D(k) \quad (4.15)$$

where, $\alpha = \frac{T}{N+T}$ ($0 < \alpha \leq 1$): coefficient of low pass filter

From (4.15), we derive the equation of low-pass filter calculation:

$$\Rightarrow u_f^D(k) = (1 - \alpha)u_f^D(k-1) + \alpha u^D(k) \quad (4.16)$$

4.4.3 Anti-windup

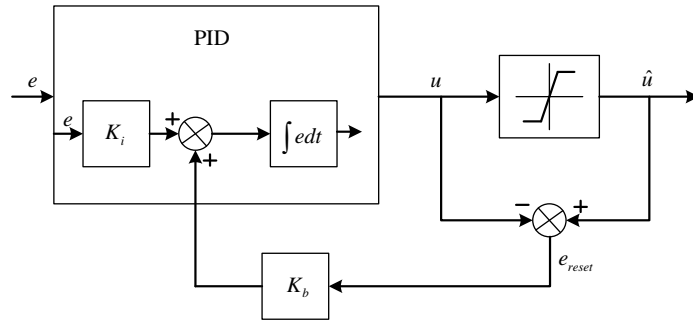


Fig 4.15: Block diagram of an anti-windup structure

From the block, we draw out the equation to calculate the anti-windup for I term

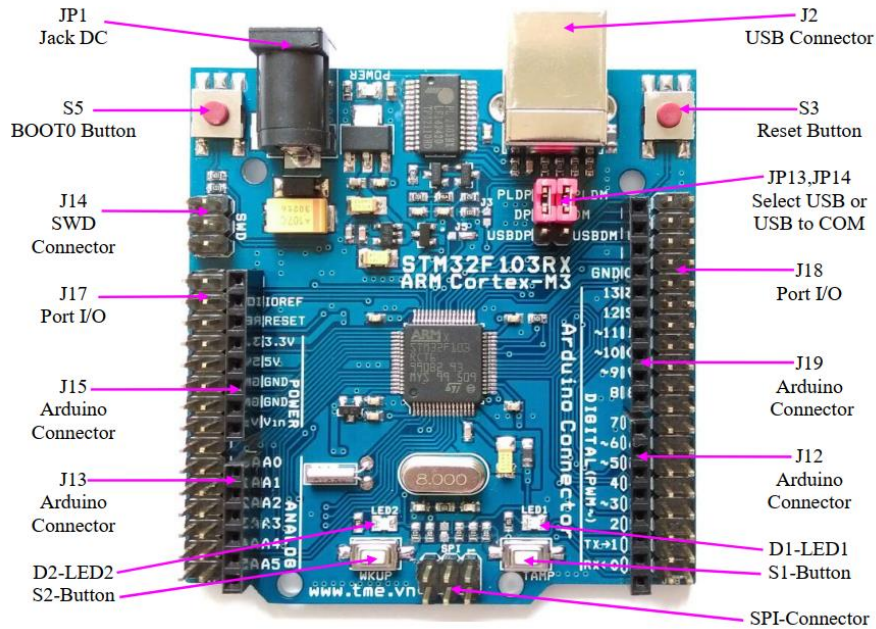
$$u^I(t) = \int_0^t [K_I e(\tau) + K_b e_{\text{reset}}(\tau)] d\tau$$

$$\Rightarrow u_k^I = u_{k-1}^I + K_I T e_k + K_b T e_k^{\text{reset}} \quad (4.17)$$

4.5. Experiments

4.5.1 Programming using STM32F103

Board STM32F103RX



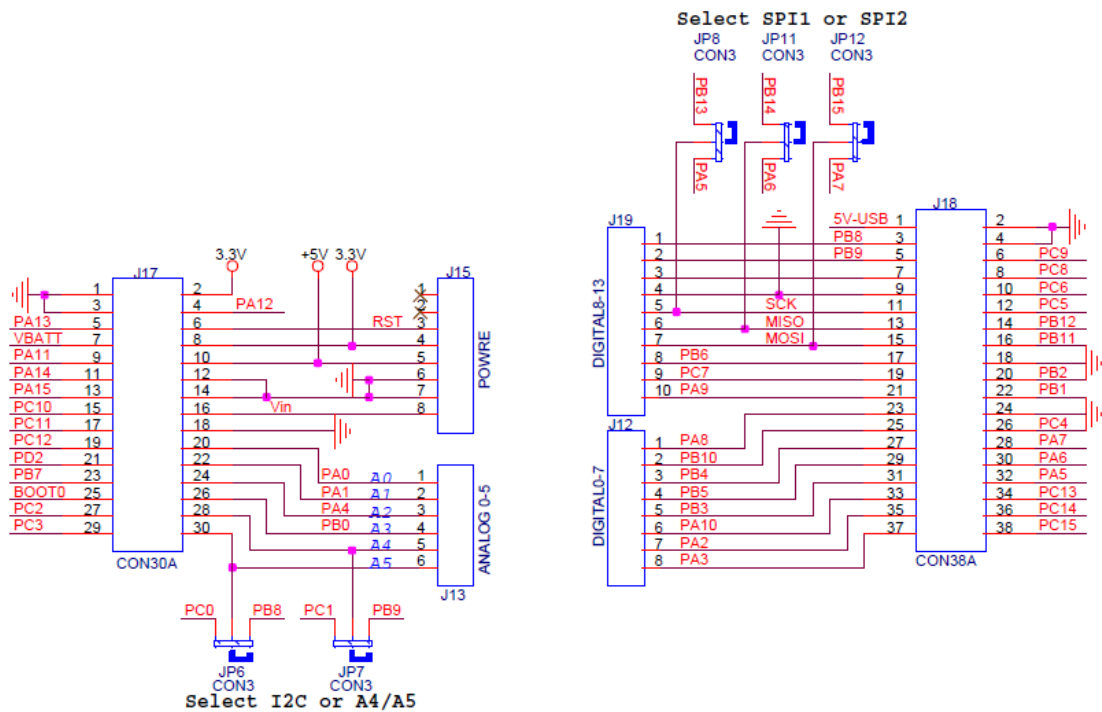


Fig 4.17: Schematic of all I/O pinouts

Programmer/Debugger

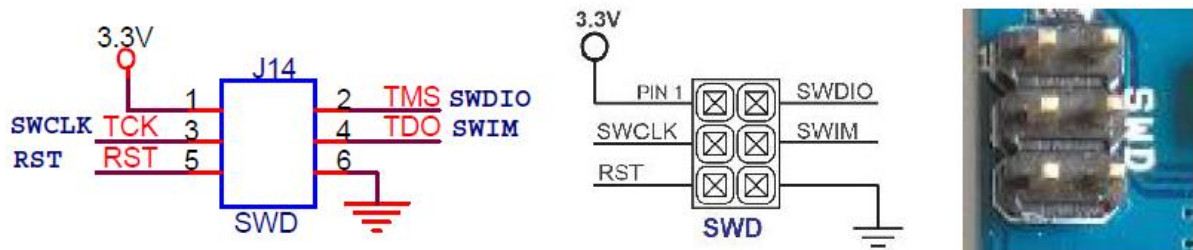


Fig 4.18: Schematic and pinouts of SWD

H-Bridge



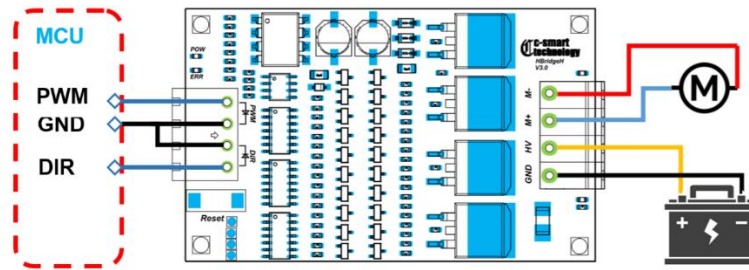


Fig 4.19: H-bridge pinouts

Specifications:

- ✓ Input voltage: 8V-32V; Continuous current: 16A
- ✓ Input level: 3.3-5V
- ✓ The PWM frequency max: 20 KHz
- ✓ Duty cycle: 0-100%

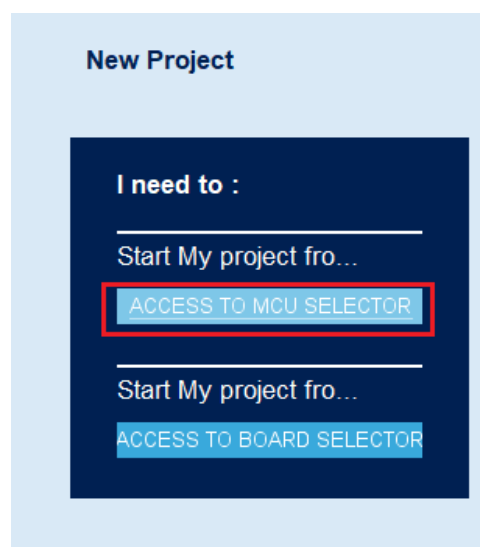
✚ Hardware layout and wiring diagram

Investigating the components and hardware connection of the system, each group has to draw its schematic and include it in the report.

4.5.2 Microcontroller programming

✚ Using STM32CUBEMX to create a project

Step 1: Create a project using STM32F103RCT



MCU Filters

Part Number Search

Core

Series

Line

Package

Other

Price From 2.839 to 2.839

IO From 50 to 51

Eeprom = 0 (Bytes)

Features

Block Diagram

Docs & Resources

Datasheet

Buy 3

Start Project

STM32F103RC

Mainstream Performance line, ARM Cortex-M3 MCU with 256 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN

ACTIVE Active Product is in mass production

Unit Price for 10kU (US\$): 2.839

LQFP64

The STM32F103xx, STM32F103xD and STM32F103xE performance line family incorporates the high-performance ARM

The STM32F103xx high-density performance line family operates in the -40 to +105 °C temperature range, from a 2.0 to 3.6 V power supply. A comprehensive set of power-saving mode allows the design of low-power applications. These features make the STM32F103xx high-density performance line microcontroller family suitable for a wide range of applications.

MCUs List: 2 items

Display similar items

Part No	Reference	Market	Unit Price for 10kU	Board	Package	Flash	RAM	IO	Freq	GFX Sc.
STM32F103RC	STM32F103RCTx	Active	2.839		LQFP64	256 kBytes	48 kBytes	51	72 MHz	0.0
STM32F103...	STM32F103RCYx	Active	2.839		WLCSP64	256 kBytes	64 kBytes	50	72 MHz	0.0

Pinout & Configuration

Clock Configuration

Project Manager

Tools

Additional Softwares

Pinout

Options

Categories A-Z

System Core

Analog

Timers

Connectivity

Multimedia

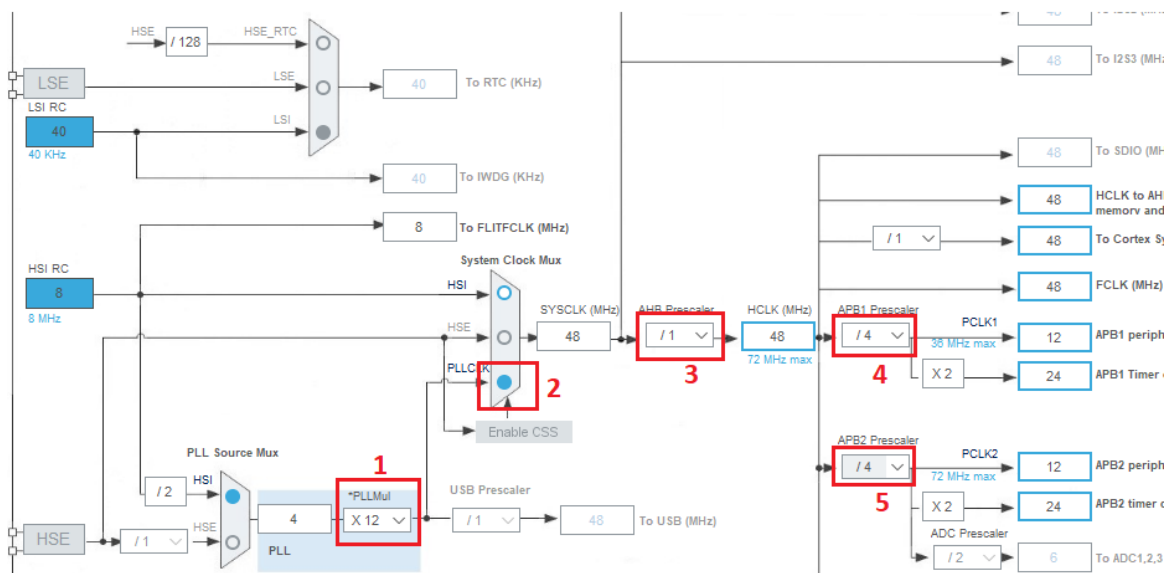
Computing

Middleware

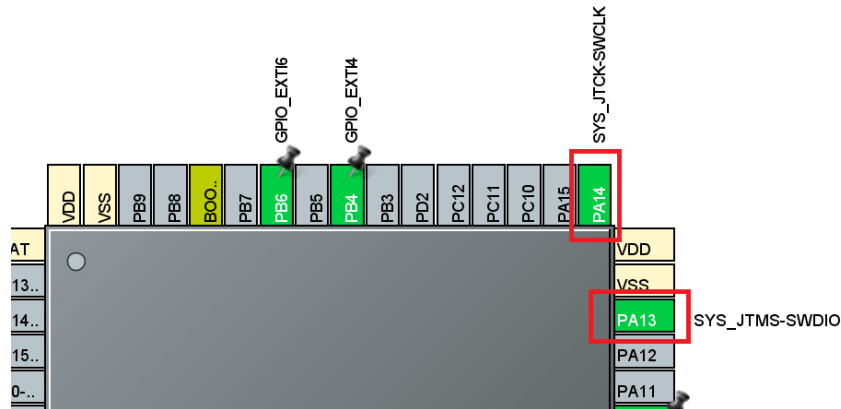
Pinout view

System view

Step 2: Clock configuration. Note that it is only an example, students can set up clocks with other parameters but in that case, from now on, all calculations related to clocks have to be modified.

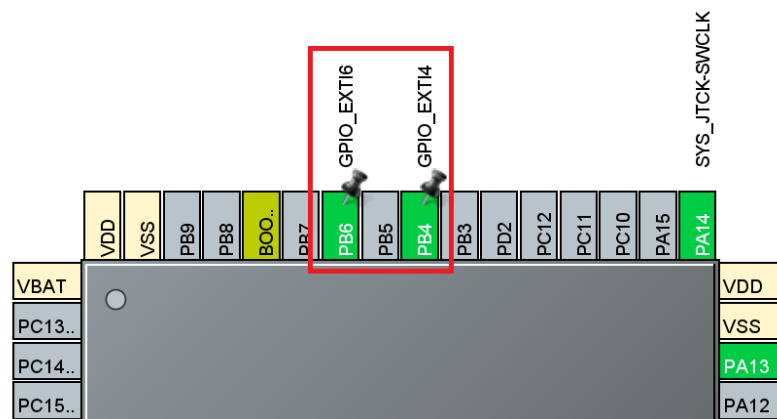


Step 3: SWD configuration for programming



Step 4: Hardware declaration

✚ We use *two external interrupts* to read encoder signals (Ch. A, Ch. B). In this example, PB4 and PB6 will be connected and configured as the below figure.



System Core

- DMA
- GPIO**
- IWDG
- NVIC
- RCC
- ▲ SYS
- WWDG

Analog >

Timers >

Connectivity >

Multimedia >

Computing >

Middleware >

Group By Peripherals

GPIO SYS TIM USART NVIC

Search Signals

Search (Ctrl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO outp...	GPIO mode	GPIO P...	Maximu...	User L...	Modified
PB4	n/a	n/a	External Interrupt Mode...	Pull-up	n/a		✓
PB6	n/a	n/a	External Interrupt Mode...	Pull-up	n/a		✓
PC3	n/a	Low	Output Push Pull	No pull-...	Low		✓

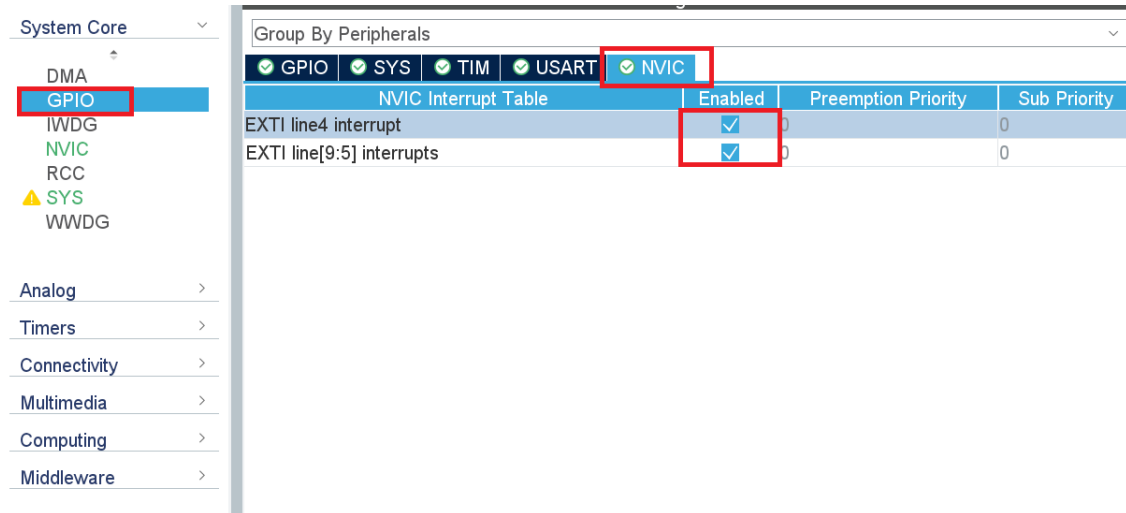
PB4 Configuration :

GPIO mode: External Interrupt Mode with Rising/Falling edge trigger detection

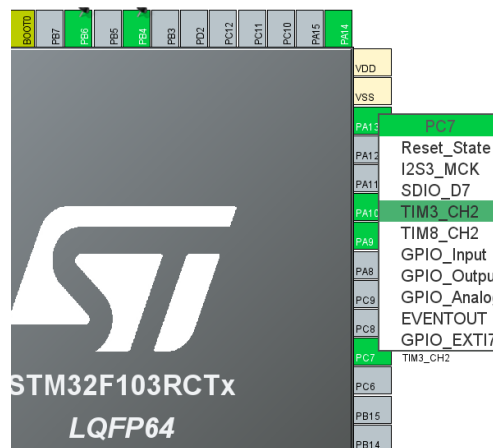
GPIO Pull-up/Pull-down: Pull-up

User Label:

✚ *Enable external interrupt* to read encoder



✚ **PWM settings:** A PWM channel is used for the H-bridge. In this case, TIM3_CH2 (PC7) is configured as the following figures:



The PWM frequency is calculated:

$$\text{Timer_tick} = \frac{\text{Timer_clock (APB1)}}{\text{Prescaler} + 1} \quad (3.17)$$

$$f_{\text{PWM}} = \frac{\text{Timer_tick}}{\text{Counter Period} + 1} \quad (3.18)$$

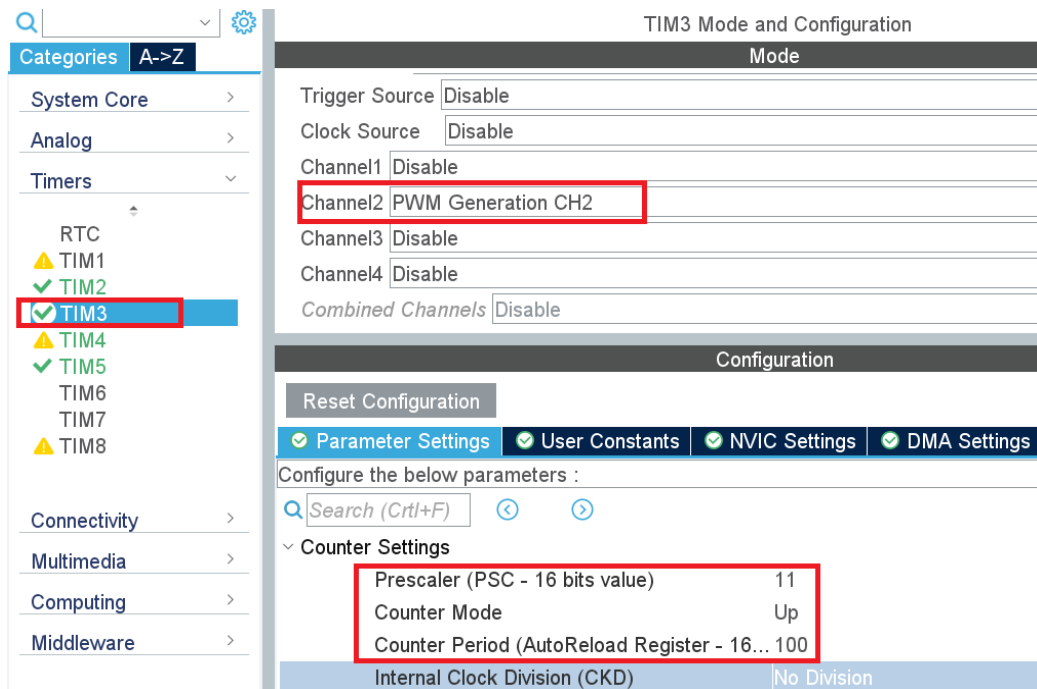
Example: Timer_clock (APB1) = 24 Mhz (Clock configuration)

Prescaler = 11; Counter Period = 100

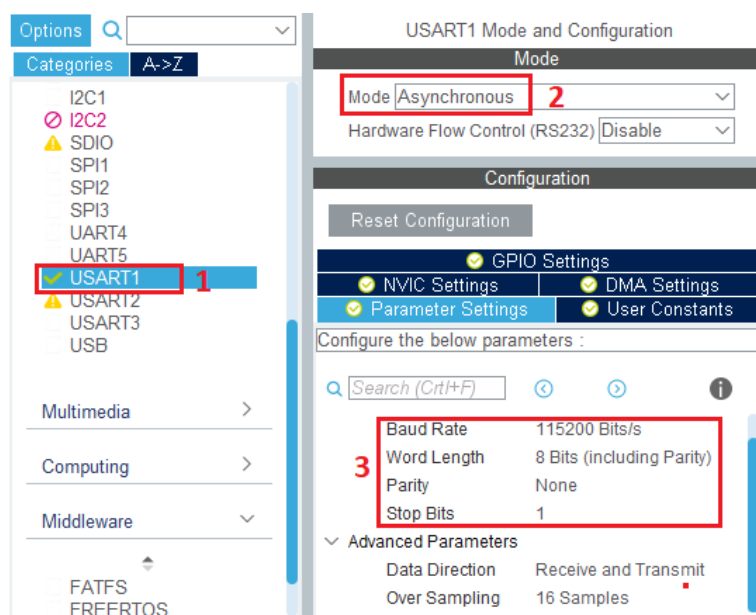
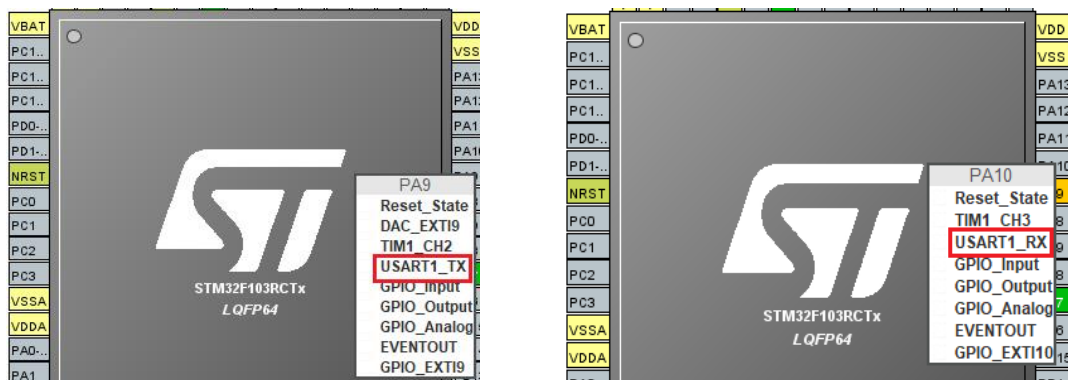
$$\text{Timer_tick} = \frac{24 \times 10^6}{11 + 1} = 2 \times 10^6 (\text{Hz})$$

$$\Rightarrow f_{\text{PWM}} = \frac{2 \times 10^6}{100 + 1} \approx 19.8 (\text{KHz})$$

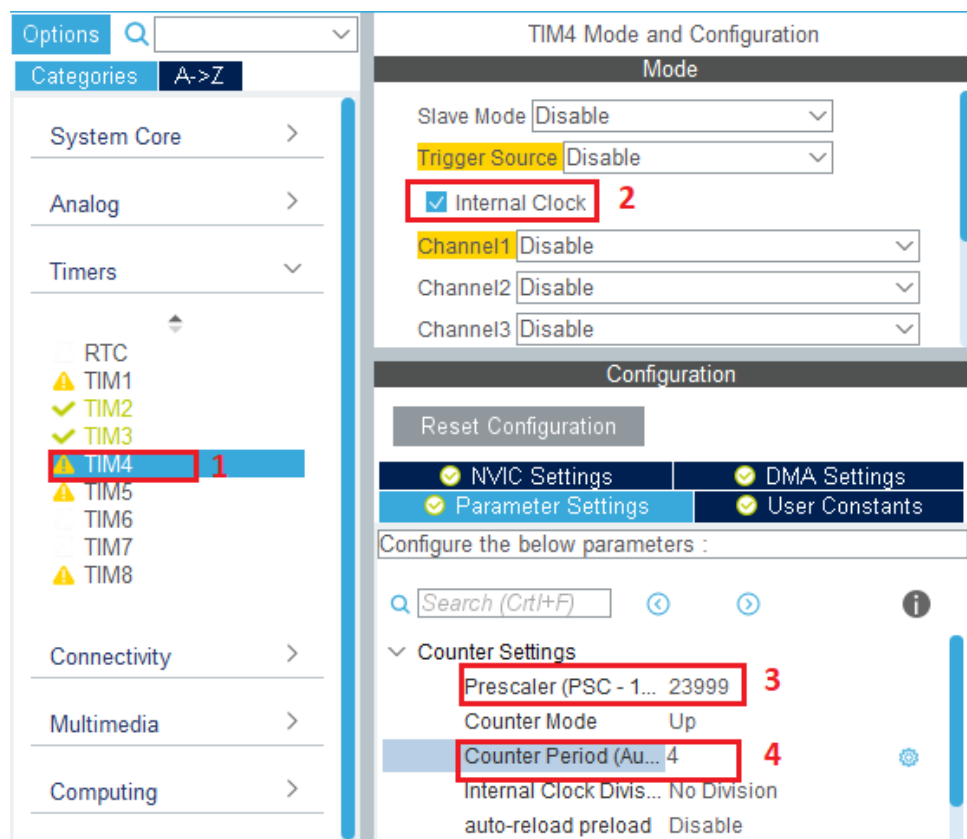
TIM3_CH2 is configured as follows:



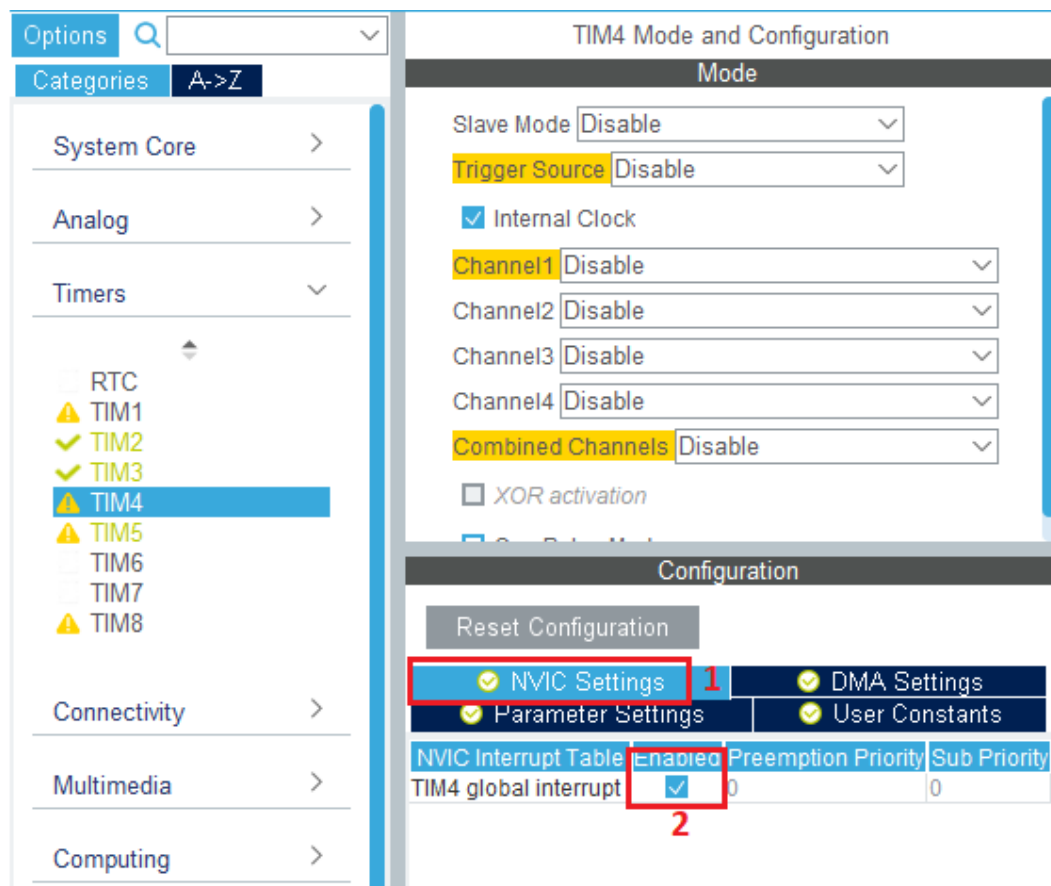
✚ **USART settings:** Using UART to communicate with computer



✚ **Timer Interrupt:** Timer 4 is adopted to configure a cyclic interrupt 5 (ms). Using Eq. 3.17 to calculate the setting parameters as follow



Enable TIM4 interrupt



Step 4: Code generation

Pinout & Configuration | Clock Configuration | **Project Manager**

Project

Code Generator

Advanced Settings

Project Settings

Project Name: **DCservo**

Project Location: **C:\Users\ChuongVo\Desktop**

Application Structure: Basic ☐ Do not generate the ma...

Toolchain Folder Location: C:\Users\ChuongVo\Desktop\DCservo\

Toolchain / IDE: **MDK-ARM V5** ☐ Generate Under Root

Linker Settings

Minimum Heap Size: 0x200

Minimum Stack Size: 0x400

STM32F103RCTx / DCservo.ioc - Project Manager

GENERATE CODE

Configuration | Clock Configuration | **Project Manager** | Tools

Project Settings

Project Name: DCservo

Project Location: C:\Users\ChuongVo\Desktop

Application Structure: Basic

Toolchain Folder Location: C:\Users\ChuongVo\Desktop\DCservo\

Toolchain / IDE: MDK-ARM V5 ☐ Generate Under Root

Code Generation

The Code is successfully generated under C:/Users/ChuongVo/Desktop/DCservo

Open Folder **Open Project** Close

4.5.3 Position control

Students have to modify the given code including:

- ✓ 2 external interrupts for encoder reading (x4 mode): **EXTI4_IRQHandler**, **EXTI9_5_IRQHandler**. Using sample code in Fig. 4.10.
- ✓ Position in TIM4_IRQHandler using Eq. (4.7).

- ✓ **HAL_UART_RxCpltCallback**: based on the sample code to communicate with Visual C# interface (appendix).
- ✓ Open the Visual studio program, choose parameters for RS232 connection. Note that the BaudRate has to be compatible with the one already configured in USART1.
- ✓ *Identify the transfer functions* of the system including H-bridge and the motor (similar to the experiment of velocity control) and the load.
- ✓ Write a *PI algorithm with anti-windup* (similar to the experiment of velocity control) for the inner loop (velocity control loop) and *PD controller with low pass filter* for the outer loop (position control loop).

3.6. Report

Student have to show all their results including the hardware (schematic), obtained transfer functions, calculations for PI/PD parameters, the code and the system responses.

APPENDIX

USART1 COMUNICATION

```
int16_t DesiredSpeed;
char Rx_indx, Rx_Buffer[20], Rx_data[2];
float DesiredPos;
#ifdef __GNUC__
    #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
    #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
    #define GETCHAR_PROTOTYPE int fgetc(FILE *f)
#endif

PUTCHAR_PROTOTYPE
{
    HAL_UART_Transmit(&huart1, (uint8_t*)&ch, 1, 100);
    return ch;
}

// Ham ngat Uart
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    uint8_t i;
    if(huart->Instance == USART1) { //uart1

        if(Rx_indx==0) { for (i=0;i<20;i++) Rx_Buffer[i] = 0;}
        switch(Rx_data[0]) {
            /* dung dong co */
            case 'e':
                run =false;
                break;
            /* dong co chay */
            case 'r':
                run = true;
                break;
            case 'b':
```

```

// reset();
    break;
case 's':
    DesiredPos = atoi(Rx_Buffer);
    memset(Rx_Buffer, 0, sizeof(Rx_Buffer));
    Rx_indx = 0;
    break;
case 'v':
    DesiredSpeed = atoi(Rx_Buffer);
    memset(Rx_Buffer, 0, sizeof(Rx_Buffer));
    Rx_indx = 0;
    break;
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
case '.':
case '-':
    Rx_Buffer[Rx_indx++] |= Rx_data[0];
    break;
default:
    break;
}
HAL_UART_Receive_IT(&huart1,(uint8_t*)Rx_data,1);
}
}

```