MODEL 3. VELOCITY CONTROL FOR DC SERVO MOTORS

3.1 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 controller.
- ✓ Verify velocity control of a DC servo motor in a real model.

3.2. One axis servo system using dc servo motors

3.2.1 DC motor dynamics

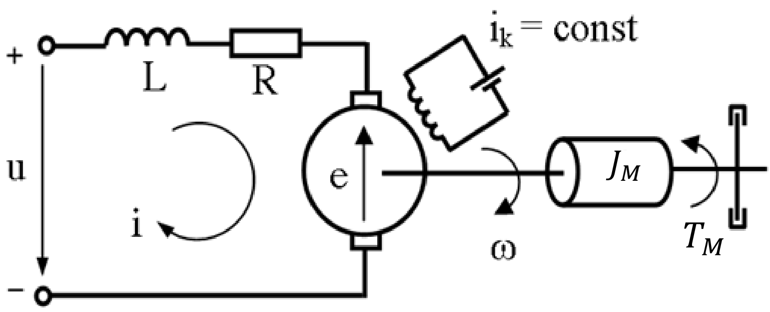


Fig. 3.1: The equivalent circuit of a DC servo motor

The electrical system: $u(t) = Ri + L\frac{di}{dt} + e$ (3.1)

where, $e = K_e\omega$

The mechanical system: $J_M\frac{d\omega}{dt} + b\omega = T_M$ (3.2)

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.m^2]

T_M**:** torque of motor [Nm]

b: viscous damping [Nm.s]

The block diagram of a DC servo motor:

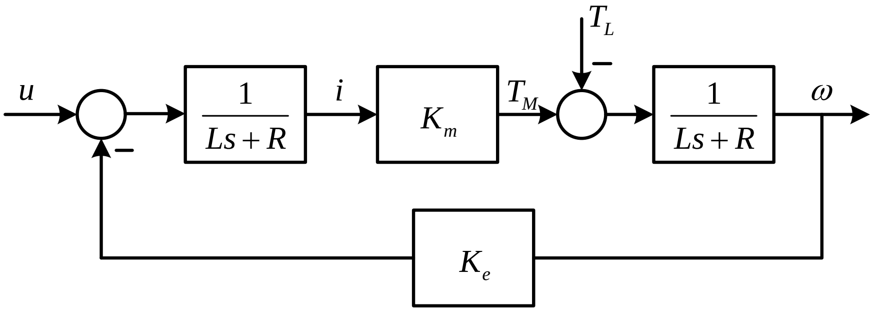


Fig. 3.2: The block diagram of a DC servo motor

The transfer function of a motor is obtained:

$$G(s) = \frac{\omega(s)}{U(s)} = \frac{K_m}{(Ls+R)(\mathrm{J}s+b)+K_m K_e} \tag{3.3}$$

$$G(s) = \frac{\omega(s)}{U(s)} \approx \frac{K_m / Rb}{\left(\frac{L}{R}s+1\right)\left(\frac{J}{b}s+1\right)} = \frac{K}{(\tau_e s+1)(\tau_m s+1)} \tag{3.4}$$

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

$$G(s) = \frac{\omega(s)}{U(s)} \approx \frac{K}{\tau_m s+1} \tag{3.5}$$

3.2.2. One axis servo system using DC servo motor

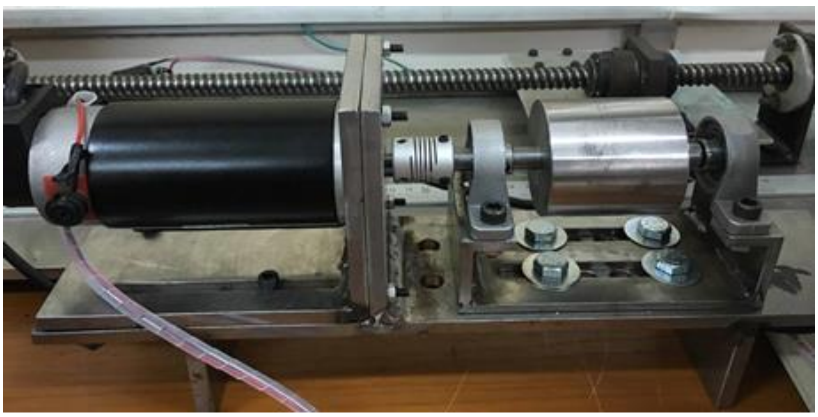


Fig. 3.3: The model of one-axis servo systems

Table 4.1. The characteristics of the DC servo motor

No.	Parameters	Symbol	Units	DCM50205
1	Continuous Torque (Max)	T_C	N.m	0.25
2	Peak Torque (Stall)	T_{PK}	N.m	1.59
3	Rated Speed	S_R	rpm	3400
4	Rotor Inertia	J_M	kg.m^2	3.11 x 10^{-5}
5	Rated Voltage	E	V	24
6	Rated Current	I	A	2.95
7	Torque Constant	K_T	N.m/A	52 x 10^{-3}
8	Resistance	R_T	Ω	0.8
9	Peak Current (Stall)	I_P	A	21.6
10	Encoder Resolution	-	Steps/rev.	1000

The rotary load with the parameters: weight (m = 0.81774 kg); material: Steel CT3; radius: R = 0.025 m. Therefore, the inertia moment of load is calculated:

$$J_L = \frac{1}{2}mr^2 = \frac{1}{2}0.81774 \times 0.025^2 = 2.56 \times 10^{-4} \text{ [kg.m}^2\text{]}$$

3.2.3 The general diagram

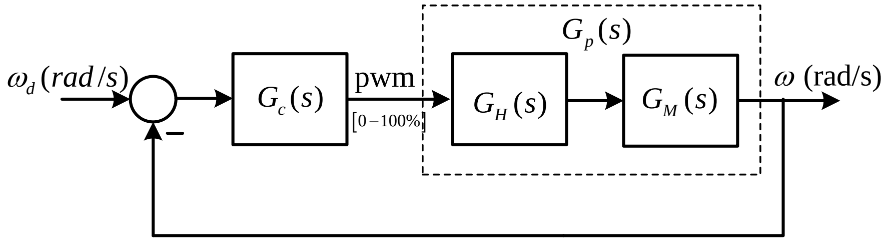


Fig. 3.4: The general diagram of the controlled systems

where, G_c: The PID controller

G_H: The transfer function of the H-bridge

G_M: The transfer function of the motor and load.

From Fig. 3.4, the transfer function of H-bridge and the motor has the following form:

$$G_p(s) = G_H(s)G_M(s) = \frac{K_p}{\tau s + 1} \quad (3.6)$$

Using IMC method, the controller G_c is obtained (PI controller):

$$G_c(s) = K_c(1 + \frac{1}{\tau_I s}) \quad (3.7)$$

where $K_c = \dfrac{\tau}{K\tau_c}$; $\tau_I = \tau$

τ_c is a tuning parameter

3.3. Quadrature encoder

3.3.1 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.

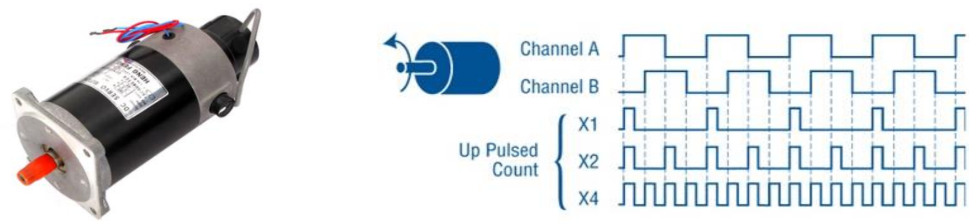


Fig. 3.5: Appearance of a DC servo motor and encoder output signals

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

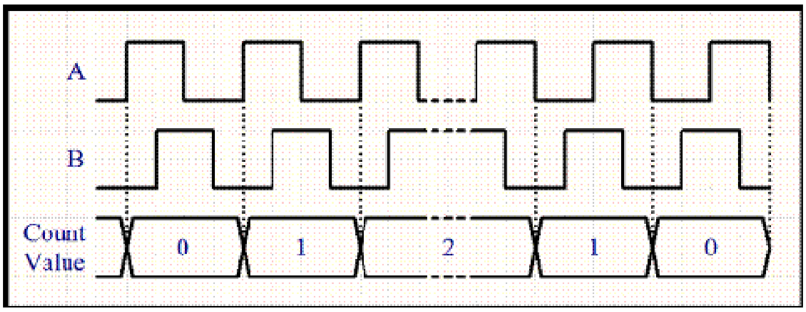


Fig. 3.6: Pulse diagram of 1x encoding

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

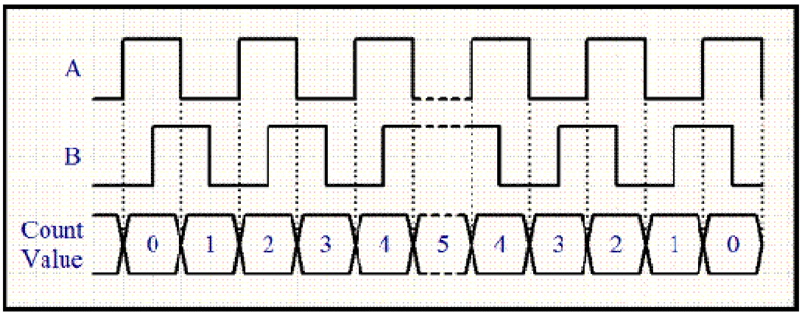
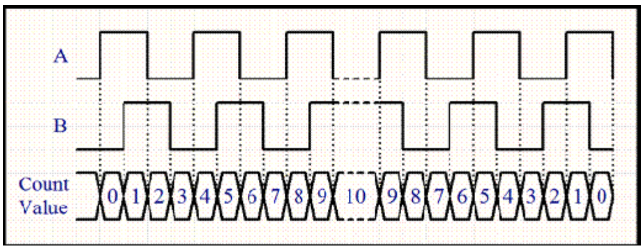


Fig. 3.7: Pulse diagram of 2x encoding

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



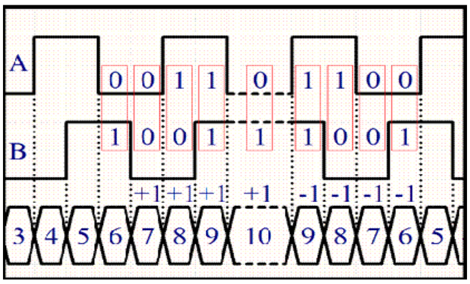


Fig. 3.8: Pulse diagram of 4x encoding

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

Table 3.3: 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:

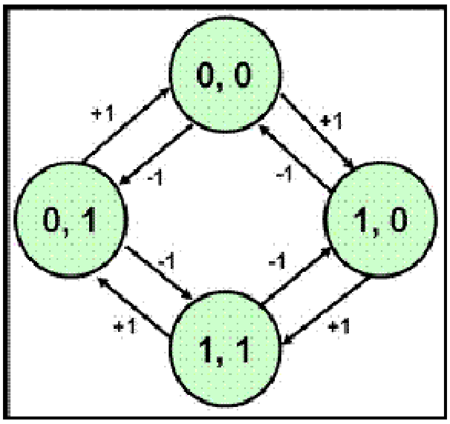


Fig. 3.9: The state transition diagram

The figure below (Fig. 3.10) 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

3.3.2 Calculating velocity/position

For simplicity, we use the following equation for estimating velocity of the motor:

$$\omega = \frac{60 \times CountValue}{T \times MaxCnt} \text{ (RPM)} \tag{3.8}$$

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)
{
  /* USER CODE BEGIN EXTI4_IRQn 0 */
unsigned char State0;
  State0 = (State0<<1) | HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
  State0 = (State0<<1) | HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_6);
  State0 = State0&0x03;
  switch (State0) {
    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 = State0;
}
```

Fig. 3.10: Code example of channel A encoding in 4x mode

3.4. PID controller

3.4.1 PID calculation

The mathematic equation of the PID controller:

$$u(t) = K_P e(t) + K_I \int_0^t e(t)d(t) + K_D \frac{de(t)}{dt} \tag{3.9}$$

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:

$$u_k = u^P{}_k + u^D{}_k + u^I{}_k \tag{3.10}$$

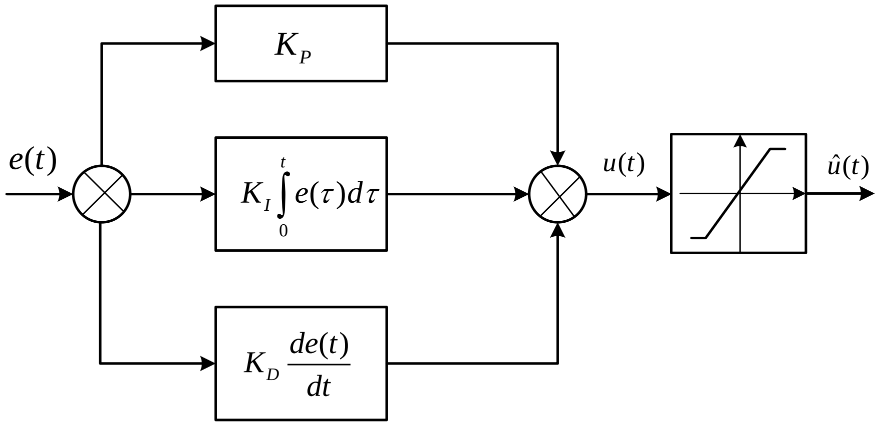


Fig 3.11: Block diagram of a PID controller

- **P Calculation**

$$u^P{}_k = K_P e_k \tag{3.11}$$

- **D calculation**

$$u^D{}_k = K_D \frac{e_k - e_{k-1}}{T} \tag{3.12}$$

- **I calculation**

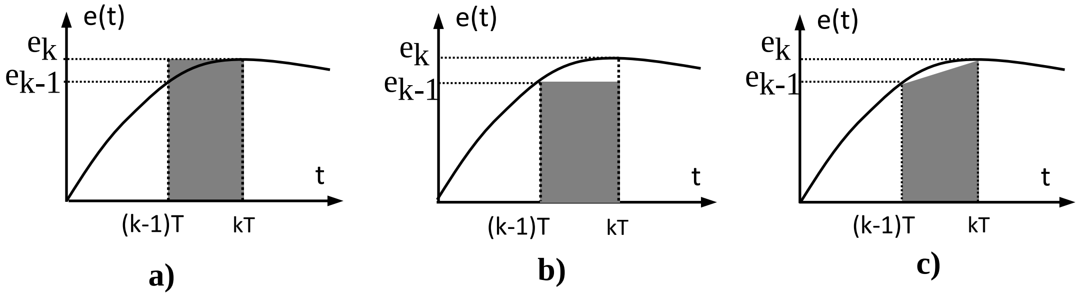


Fig 3.12: Integral approximation methods

a) Backward rectangular approximation (backward Euler):

$$u_k^I = K_I \sum_{i=1}^{k} Te_i = u_{k-1}^I + K_I Te_k \tag{3.13}$$

b) Forward rectangular approximation (forward Euler):

$$u_k^I = K_I \sum_{i=1}^{k} Te_{i-1} = u_{k-1}^I + K_I Te_{k-1} \tag{3.14}$$

c) Trapezoidal approximation:

$$u_k^I = K_I \sum_{i=1}^{k} T \frac{e_{i-1} + e_i}{2} = u_{k-1}^I + K_I T \frac{e_{k-1} + e_k}{2} \tag{3.15}$$

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 3.13: Code example of a simple PID algorithm

3.4.2 Low pass filter for D term:

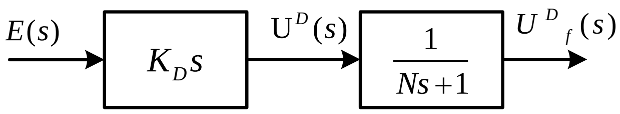


Fig 3.14: Low pass filter for D term

$$u^D{}_f(k) = \frac{N}{N+T} u^D{}_f(k-1) + \frac{T}{N+T} u^D(k) \tag{3.16}$$

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

From (4.27), we derive the equation of low-pass filter calculation:

$$\Rightarrow u^D{}_f(k) = (1-\alpha) u^D{}_f(k-1) + \alpha u^D(k) \tag{3.17}$$

3.4.3 Anti-windup

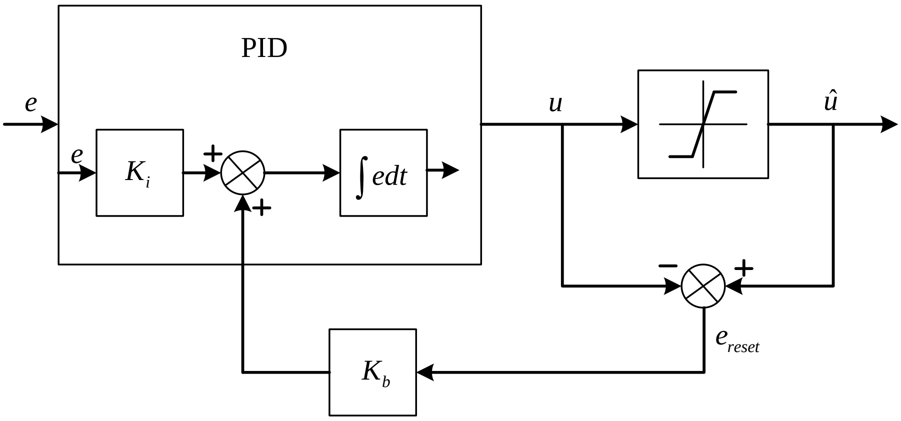


Fig 3.15: Block diagram of an anti-windup structure

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

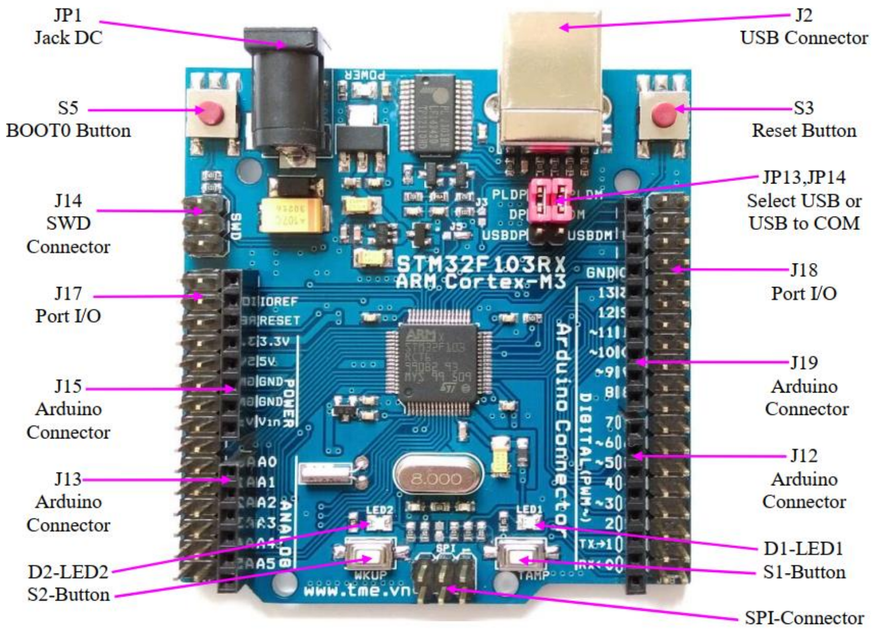
$$u^I(t) = \int_0^t [K_I e(\tau) + K_b e_{reset}(\tau)] d\tau$$

$$\Rightarrow u^I{}_k = u^I{}_{k-1} + K_I T e_k + K_b T e^{reset}{}_k \quad (3.18)$$

3.5. Experiments

3.5.1 Programming using STM32F103

- **Board STM32F103RX**



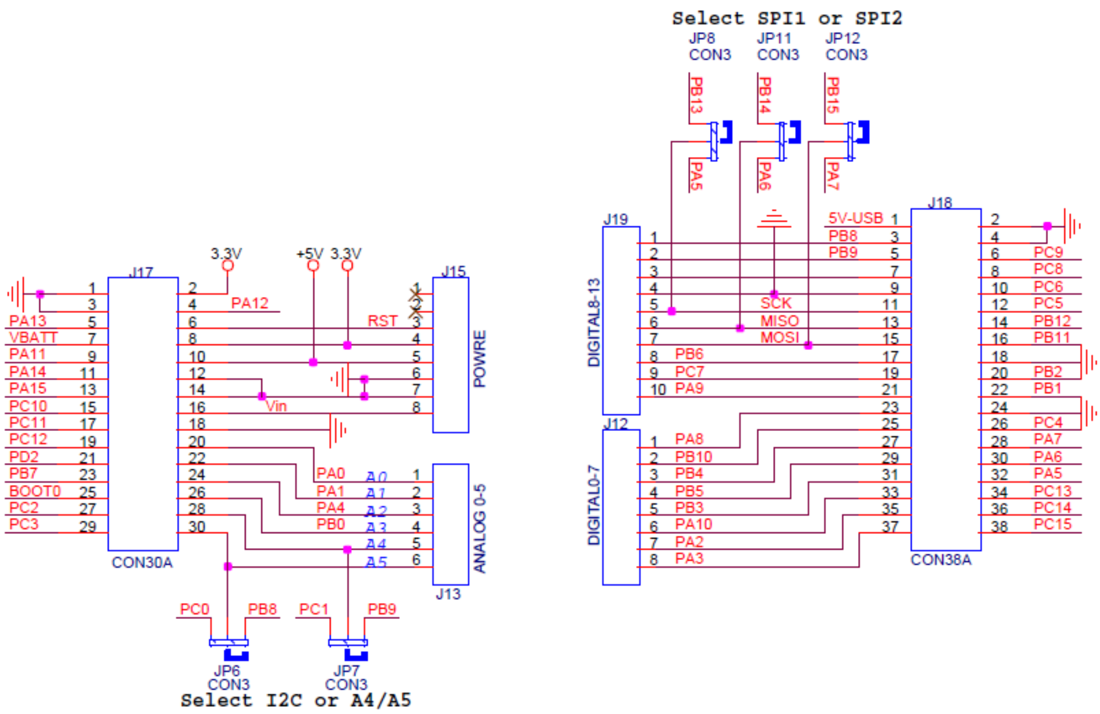


Fig 3.17: Schematic of all I/O pinouts

Programmer/Debugger

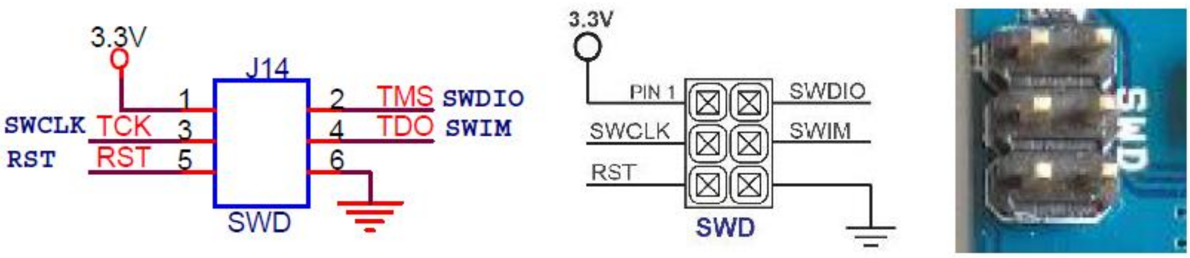
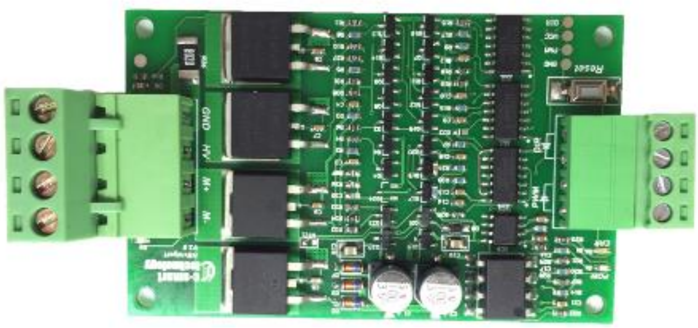


Fig 3.18: Schematic and pinouts of SWD

H-Bridge



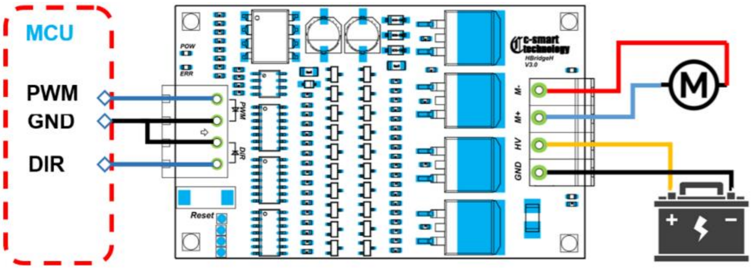


Fig 3.19: H-bridge pinouts

Specifications:

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

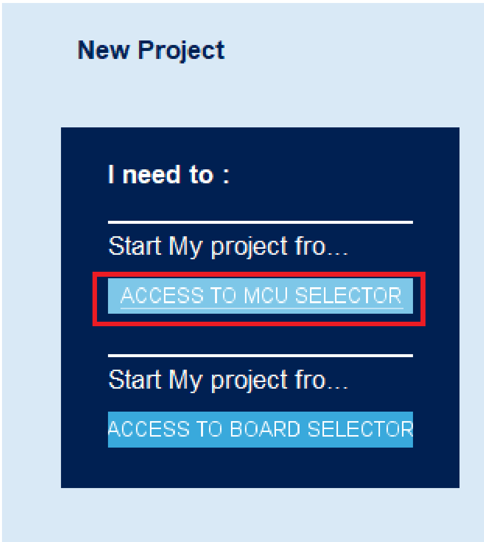
Hardware layout and wiring diagram

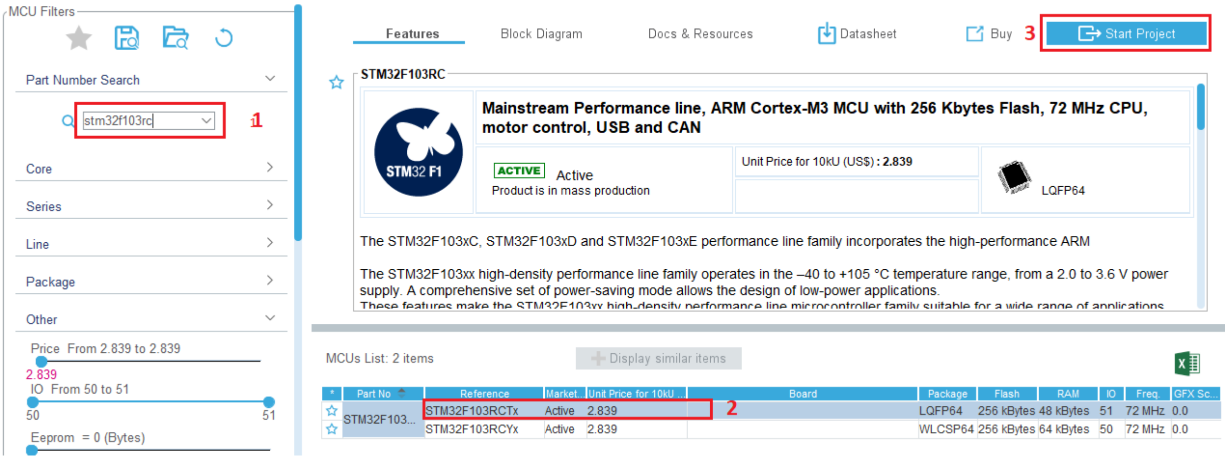
Investigating the components and hardware connection of the system, each group has to draw its schematic and include it in the report.

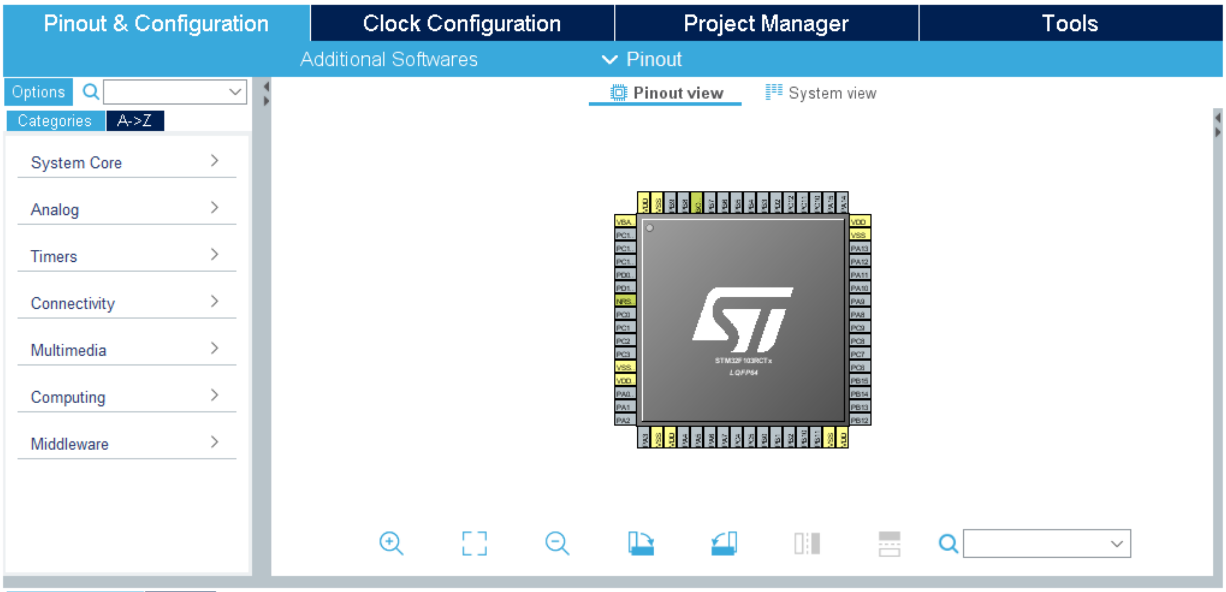
3.5.2 Microcontroller programming

Using STM32CUBEMX to create a project

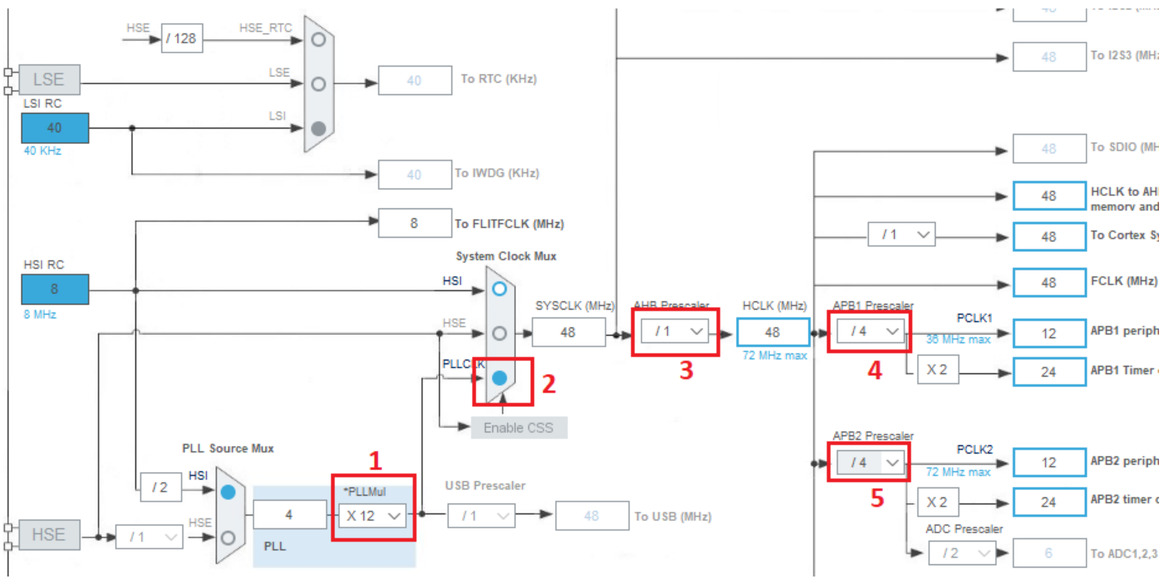
Step 1: ***Create a project using STM32F103RCT***



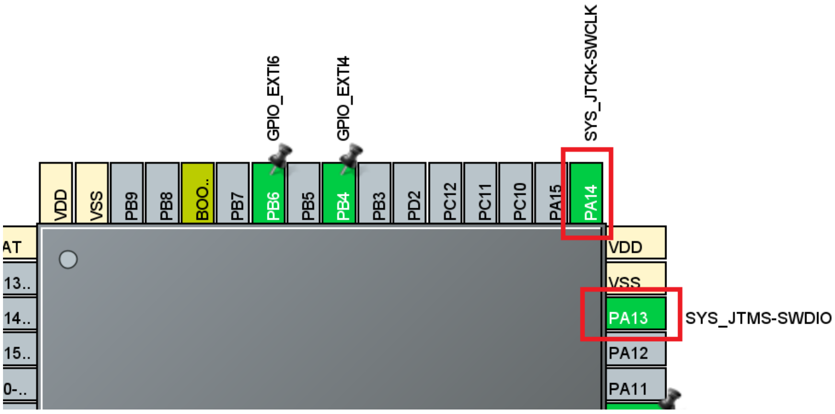




Step 2: ***Clock configuration***. Note that it is only an example, students can set up clocks with other parameters but in that case, from now on, all calculations related to clocks have to be modified.



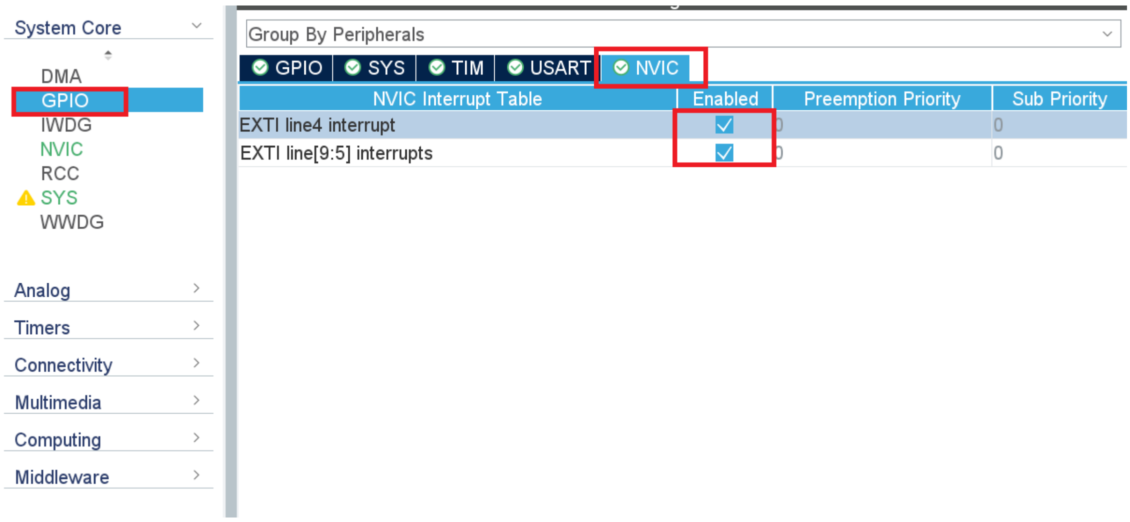
Step 3: ***SWD configuration for programming***



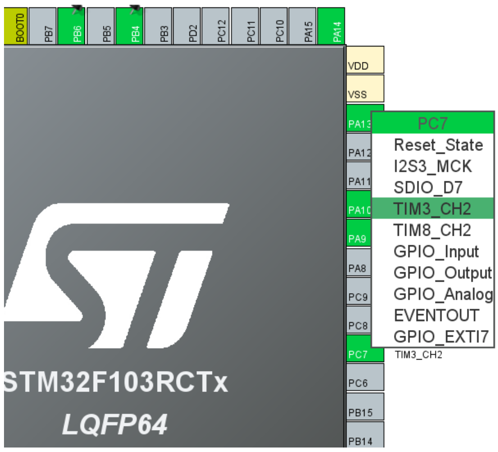
Step 4: ***Hardware declaration***

- 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.

- ***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:

$$\text{Timer\_tick} = \frac{\text{Timer\_clock (APB1)}}{\text{Prescaller} + 1} \qquad (3.19)$$

$$f_{PWM} = \frac{\text{Timer\_tick}}{\text{Counter Period} + 1} \qquad (3.20)$$

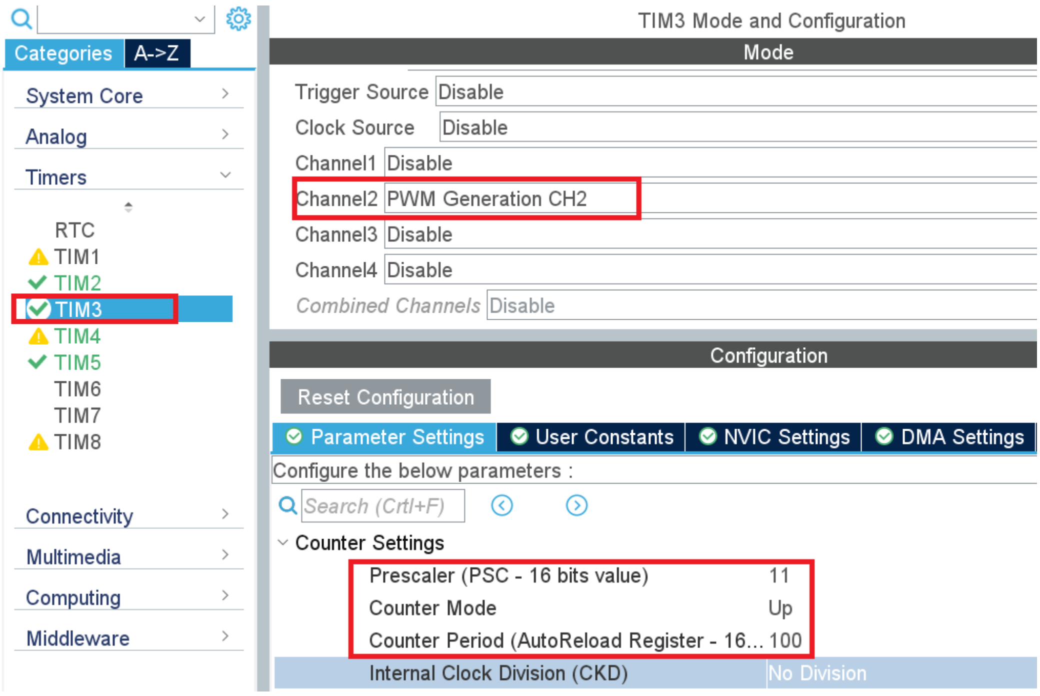
Example: Timer_clock (APB1) = 24 Mhz (Clock configuration)

Prescaller = 11; Counter Period =100

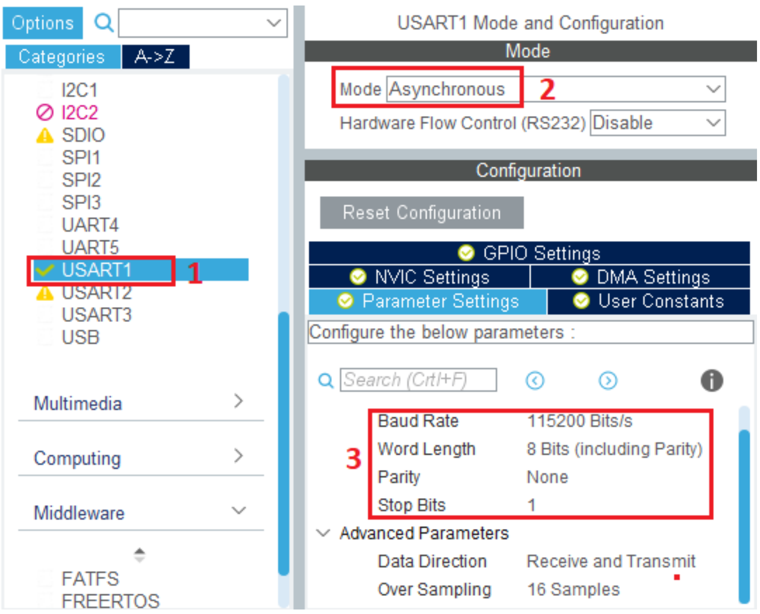
$$\text{Timer\_tick} = \frac{24\times10^{6}}{11+1} = 2\times10^{6}(\text{Hz})$$

$$\Rightarrow f_{PWM} = \frac{2\times10^{6}}{100+1} \approx 19.8\ (\text{KHz})$$

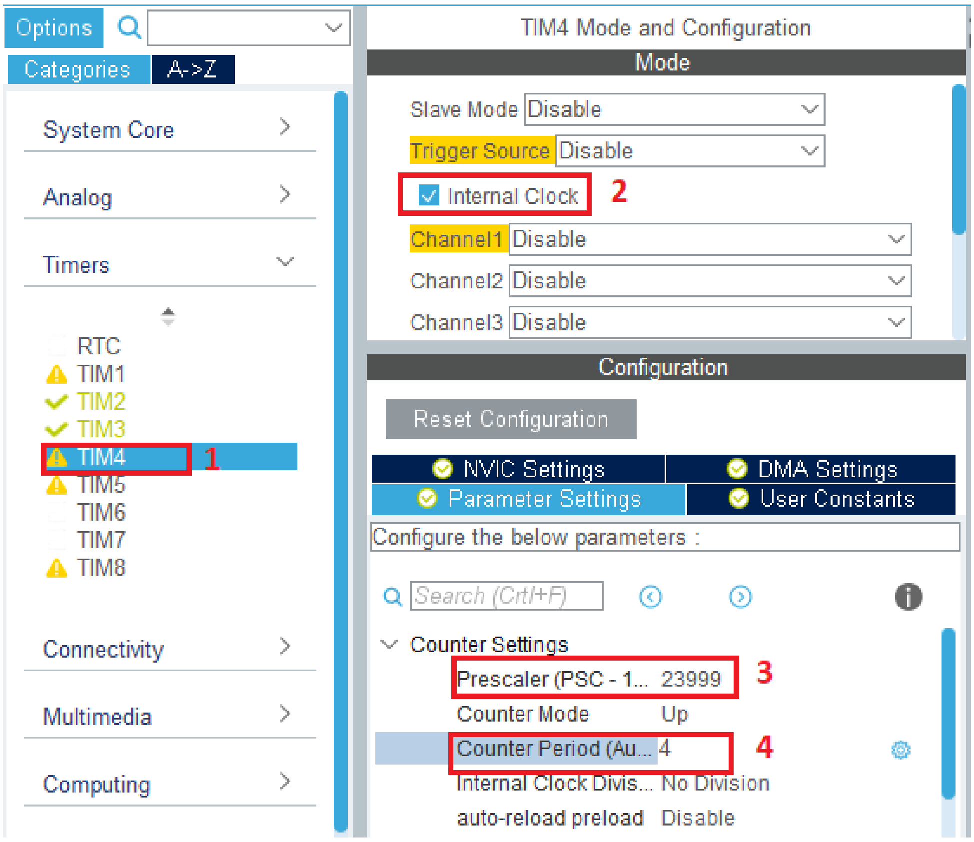
TIM3_CH2 is configured as follows:



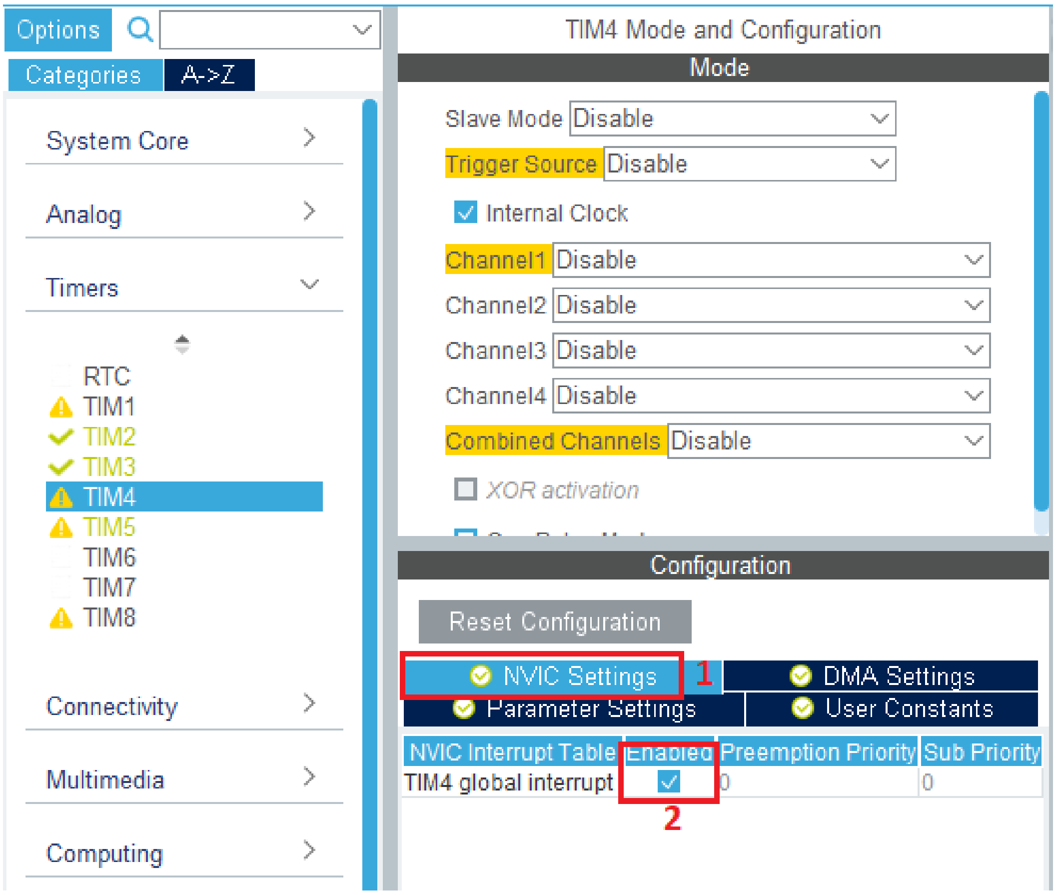
USART settings: Using UART to communicate with computer



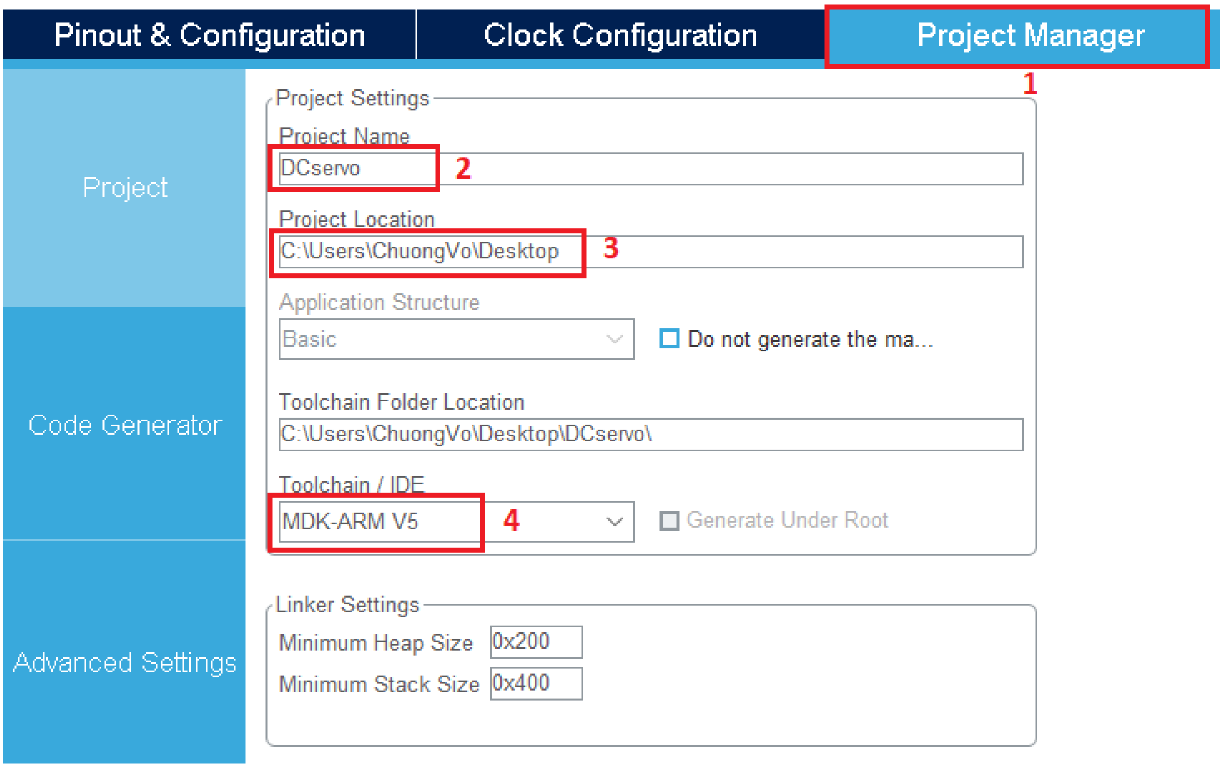
- ***Timer Interrupt***: Timer 4 is adopted to configure a cyclic interrupt 5 (ms). Using Eq. 3.17 to calculate the setting parameters as follow

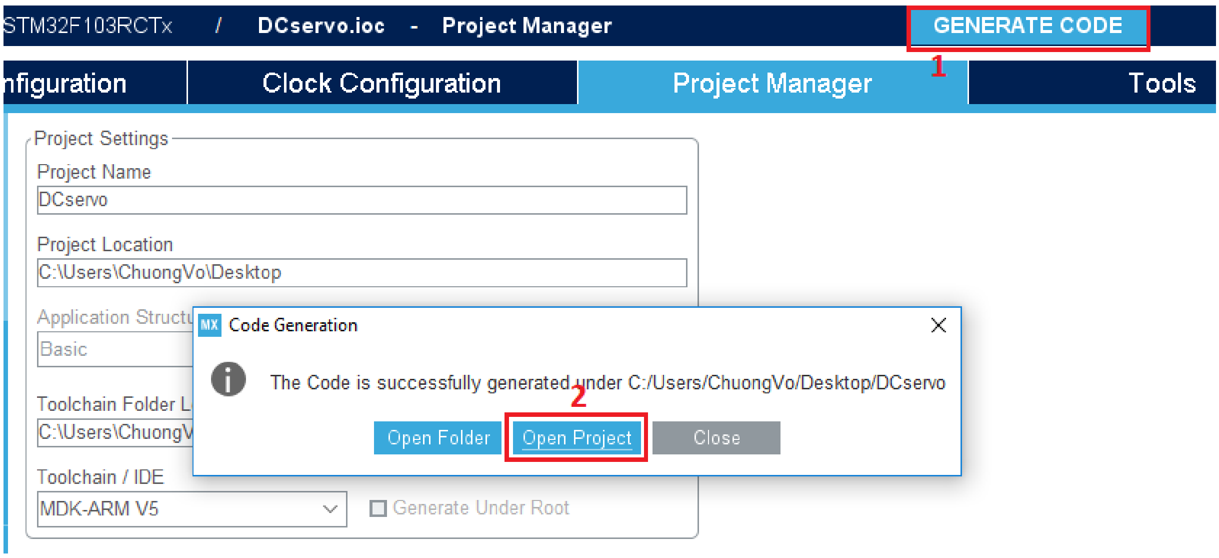


Enable TIM4 interrupt



Step 4: ***Code generation***





3.5.3 Velocity 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. 3.10
- ✓ Velocity estimation in TIM4_IRQHandler using Eq. (3.8)

- ✓ **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 function*** of the H-bridge and the motor by applying the pwm signal 100% and plotting the velocity response. From the response, calculating the parameters of the transfer function including gain (K) and time constant (τ)
- ✓ Write a ***PI algorithm with anti-windup*** based on the sample code in Fig. 3.13. However, students must improve the algorithm by adding an anti-windup for I-term (Eq. 3.18)

3.6. Report

Student have to show all their results including the hardware (schematic), obtained transfer function, calculations for PI parameters, the code and the controlled velocity 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);
     }
}
```