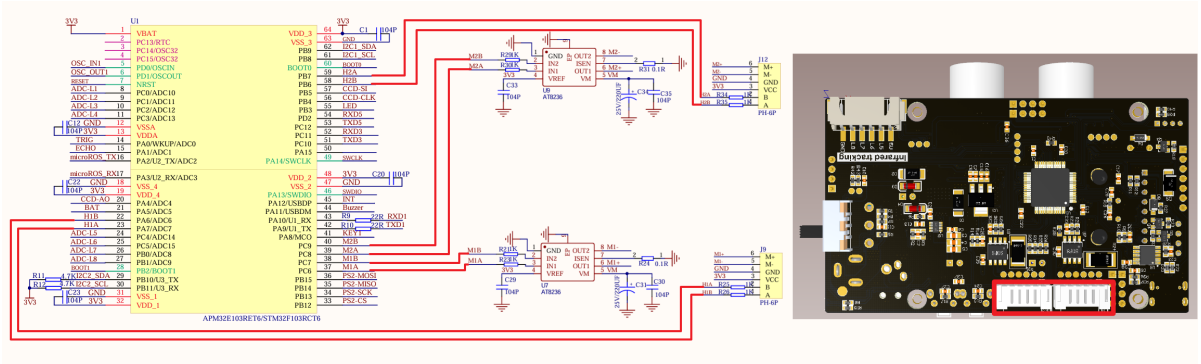
Motor drive + encoder (TIM)

Tutorial demonstrating motor driving and encoder reading.

```
TIM's PWM output controls the motor speed, and TIM's encoder interface function reads the motor encoder value
The tutorial only introduces the standard library project code
```

Hardware connection



Since we have configured a special connection line, we only need to install it to the corresponding interface::

Peripherals	Development Board	Description
Motor 1: M+ (M1A)	PC6	Development board PC6 pin controls the motor driver chip input pin, and the driver chip output pin controls motor 1: M+
Motor 1: M- (M1B)	PC7	Development board PC7 pin controls the motor driver chip input pin, and the driver chip output pin controls motor 1: M-
Motor 1: Encoder VCC	3.3V	Connect the development board 3.3V to power the encoder
Motor 1: Encoder GND	GND	Connect the development board GND to the encoder common ground
Motor 1: Encoder A phase (H1A)	PA7	Encoder A of motor 1 is connected to pin PA7 of the development board
Motor 1: Encoder B phase (H1B)	PA6	Encoder B of motor 1 is connected to pin PA6 of the development board
Motor 2: M+ (M2A)	PC8	The PC8 pin of the development board controls the input pin of the motor driver chip, and the output pin of the driver chip controls motor 2: M+
Motor 2: M- (M2B)	PC9	The PC9 pin of the development board controls the input pin of the motor driver chip, and the output pin of the driver chip controls motor 2: M-
Motor 2: Encoder VCC	3.3V	Connect the development board 3.3V to power the encoder
Motor 2: Encoder GND	GND	Connect the development board GNDV to the encoder common ground
Motor 2: Encoder A phase (H2A)	PB7	Encoder A of motor 2 is connected to pin PB7 of the development board
Motor 2: Encoder B phase (H2B)	PB6	Encoder B of motor 2 is connected to pin PB6 of the development board

Control principle

> **520 motor**

Motor model	520 motor
Motor rated voltage	12V
Encoder supply voltage	3.3-5V
Gear set reduction ratio	1:30
Magnetic ring line number	11 lines
Encoder type	AB phase incremental Hall encoder

> **Counting formula**

$$Maximumcountvalueofasingleturn = reductionratio * encoderlinenumber * encoderfrequencymultiplication = 30 * 11 * 4 = 13$$

```
According to this single turn maximum count value formula, the car speed can be calculated
```

> **Encoder reading**

The encoder interface mode of TIM3 and TIM4 of the general timer counts the encoder signal of the motor.

```
Get the count value by directly reading the timer CNT register value.
```

> **Speed and steering control**

The PWM output function of TIM8 of the advanced timer changes the duty cycle of the PWM signal.

> **Motor driver chip: AT8236**

The development board integrates 2 AT8236 single-channel brushed DC motor driver chips.

Input pins IN1 and IN2 control the output state of the H-bridge. The following table shows the logical relationship between the input and output:

IN1	IN2	OUT1	OUT2	Function
0	0	Z	Z	Coast, Sleep
1	0	H	L	Forward
0	1	L	H	Reverse
1	1	L	L	Brake

When using PWM control to achieve the speed regulation function, the H-bridge can operate in two different states: fast decay or slow decay.

IN1	IN2	Function
PWM	0	Forward PWM, fast decay
1	PWM	Forward PWM, slow decay
0	PWM	Reverse PWM, fast decay
PWM	1	Reverse PWM, slow decay

Pin definition

Main control chip	Pin	Main function (after reset)	Default multiplexing function	Redefine function
STM32F103RCT6	PA6	PA6	SPI1_MISO/ ADC12_IN6/TIM3_CH1	TIM1_BKIN
STM32F103RCT6	PA7	PA7	SPI1_MOSI/ ADC12_IN7/TIM3_CH2	TIM1_CH1N

Software code

Since the default function of the pin is the normal IO pin function, we need to use the multiplexing function.

```
Product supporting materials source code path: Attachment → Source code summary → 2. Extended_Course → 3.
Motor
```

Control function

The tutorial only briefly introduces the code, you can open the project source code to read the details.

> **Encoder_Init_TIM3**

```
void Encoder_Init_TIM3(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_ICInitTypeDef TIM_ICInitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); //Enable the clock of timer 3
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //Enable the PA port clock

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7; //Port configuration
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //Floating input
    GPIO_Init(GPIOA, &GPIO_InitStructure); //Initialize GPIOA according to the set parameters
    TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Prescaler = 0; // Prescaler
    TIM_TimeBaseStructure.TIM_Period = ENCODER_TIM_PERIOD; // Set counter auto-reload value
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; // Select clock division: no division
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; /////TIM counts up
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
    TIM_EncoderInterfaceConfig(TIM3, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising); //
Use encoder mode 3
    TIM_ICStructInit(&TIM_ICInitStructure);
    TIM_ICInitStructure.TIM_ICFilter = 10; //Filter 10
    TIM_ICInit(TIM3, &TIM_ICInitStructure);
    TIM_ClearFlag(TIM3, TIM_FLAG_Update);//Clear TIM update flag
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
    //Reset counter
    TIM_SetCounter(TIM3,0);
    TIM_Cmd(TIM3, ENABLE);
}
```

> **Encoder_Init_TIM4**

```
void Encoder_Init_TIM4(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_ICInitTypeDef TIM_ICInitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE); //Enable the clock of timer 4
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //Enable the clock of PB port

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7; //Port configuration
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //Floating input
    GPIO_Init(GPIOB, &GPIO_InitStructure); //Initialize GPIOB according to the set parameters
    TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Prescaler = 0x0; // Prescaler
    TIM_TimeBaseStructure.TIM_Period = ENCODER_TIM_PERIOD; // Set counter auto-reload value
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; // Select clock division: no division
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; /////TIM counts up
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
    TIM_EncoderInterfaceConfig(TIM4, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising); //
Use encoder mode 3
    TIM_ICStructInit(&TIM_ICInitStructure); TIM_ICInitStructure.TIM_ICFilter = 10; TIM_ICInit(TIM4,
&TIM_ICInitStructure);
    TIM_ClearFlag(TIM4, TIM_FLAG_Update);//Clear the TIM update flag TIM_ITConfig(TIM4, TIM_IT_Update,
ENABLE); //Re set counter
    TIM_SetCounter(TIM4,0); TIM_Cmd(TIM4, ENABLE);
}
```

> **Read_Encoder**

```
int Read_Encoder(Motor_ID MYTIMX)
{
   int Encoder_TIM;
   switch(MYTIMX)
     {
         case MOTOR_ID_ML:  Encoder_TIM= (short)TIM3 -> CNT;  TIM3 -> CNT=0;break;
         case MOTOR_ID_MR:  Encoder_TIM= (short)TIM4 -> CNT;  TIM4 -> CNT=0;break;
         default: Encoder_TIM=0;
     }
        return Encoder_TIM;
}
```

> **TIM3_IRQHandler**

```
void TIM3_IRQHandler(void)
{
    if(TIM3->SR&0X0001)
    {
    }
    TIM3->SR&=~(1<<0);
}
```

> **TIM4_IRQHandler**

```
void TIM4_IRQHandler(void)
{
    if(TIM4->SR&0X0001)
    {
    }
    TIM4->SR&=~(1<<0);
}
```

> **Balance_Motor_Init**

```
void Balance_Motor_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //Enable PC port clock
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_9; //Port configuration
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; //Push-pull output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //50M
    GPIO_Init(GPIOC, &GPIO_InitStructure); //Initialize GPIOC according to the set parameters
}
```

> **Balance_PWM_Init**

```
void Balance_PWM_Init(u16 arr,u16 psc)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
    TIM_DeInit(TIM8);
    TIM_TimeBaseStructure.TIM_Period = arr - 1; //Set the value of the auto-reload register period to load the
activity at the next update event
    TIM_TimeBaseStructure.TIM_Prescaler = psc; //Set the prescaler value used as the divisor of the TIMx clock
frequency. No division
    TIM_TimeBaseStructure.TIM_ClockDivision = 0; //Set the clock division: TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM up counting mode
    TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure); //Initialize the time base unit of TIMx according to the
parameters specified in TIM_TimeBaseInitStruct

    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //Select timer mode: TIM pulse width modulation mode 1
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //Comparison output enable
    TIM_OCInitStructure.TIM_Pulse = 0; //Set the pulse value to be loaded into the capture comparison register
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //Output polarity: TIM output comparison
polarity high

    TIM_OC1Init(TIM8, &TIM_OCInitStructure); //Initialize peripheral TIMx according to the parameters
specified in TIM_OCInitStruct
    TIM_OC2Init(TIM8, &TIM_OCInitStructure); //Initialize peripheral TIMx according to the parameters
specified in TIM_OCInitStruct
```

```
    TIM_OC3Init(TIM8, &TIM_OCInitStructure); //Initialize peripheral TIMx according to the parameters
specified in TIM_OCInitStruct
    TIM_OC4Init(TIM8, &TIM_OCInitStructure); //Initialize peripheral TIMx according to the parameters
specified in TIM_OCInitStruct

    TIM_CtrlPWMOutputs(TIM8,ENABLE); //MOE main output enable

    TIM_OC1PreloadConfig(TIM8, TIM_OCPreload_Enable); //CH1 preload enable
    TIM_OC2PreloadConfig(TIM8, TIM_OCPreload_Enable); //CH2 preload enable
    TIM_OC3PreloadConfig(TIM8, TIM_OCPreload_Enable); //CH3 preload enable
    TIM_OC4PreloadConfig(TIM8, TIM_OCPreload_Enable); //CH4 preload enable
    TIM_ARRPreloadConfig(TIM8, ENABLE); //Enable TIMx preload register on ARR
    /* TIM8 enable counter */
    TIM_Cmd(TIM8, ENABLE); //Enable timer 8
}
```

> **Set_Pwm**

```
void Set_Pwm(int motor_left,int motor_right)
{
    if(motor_left == 0)
    {
        L_PWMA = 0;
        L_PWMB = 0;
    }
    if(motor_right == 0)
    {
        R_PWMA = 0;
        R_PWMB = 0;
    }
  if(motor_left>0)
    {
        L_PWMB = myabs(motor_left);
        L_PWMA = 0;
    }
    else
    {
        L_PWMB = 0;
        L_PWMA = myabs(motor_left);
    }
    if(motor_right>0)
    {
        R_PWMA = myabs(motor_right);
        R_PWMB = 0;
    }
    else
    {
        R_PWMA = 0;
        R_PWMB = myabs(motor_right);
    }
}
```

> **myabs**

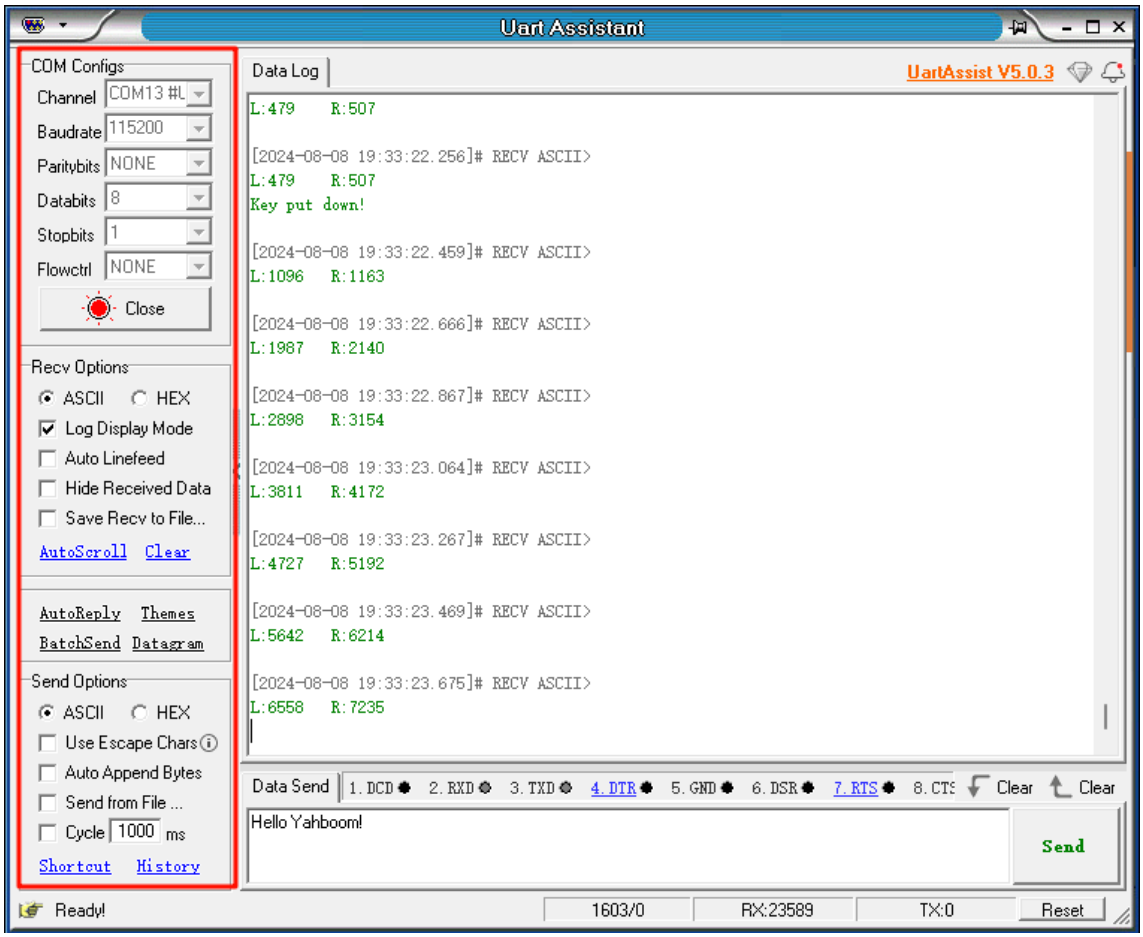
```
int myabs(int a)
{
    int temp;
    if(a<0)  temp=-a;
    else temp=a;
    return temp;
}
```

Experimental phenomenon

The Motor.hex file generated by the project compilation is located in the OBJ folder of the Motor project. Find the Motor.hex file corresponding to the project and use the FlyMcu software to download the program to the development board.

After the program is successfully downloaded: the serial port will print the encoder count value, and the button can control the motor to stop, forward, and reverse.

```
When using the serial port debugging assistant, you need to pay attention to the serial port settings. If the
settings are wrong, the phenomenon may be inconsistent.
```


Uart Assistant
COM Configs
Channel COM13 #L
Baudrate 115200
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE
Close
Recv Options
ASCII HEX
Log Display Mode
Auto Linefeed
Hide Received Data
Save Recv to File...
AutoScroll Clear
AutoReply Themes
BatchSend Datagram
Send Options
ASCII HEX
Use Escape Chars
Auto Append Bytes
Send from File ...
Cycle 1000 ms
Shortcut History
Data Log
UartAssist V5.0.3
L:479 R:507
[2024-08-08 19:33:22.256]# RECV ASCII>
L:479 R:507
Key put down!
[2024-08-08 19:33:22.459]# RECV ASCII>
L:1096 R:1163
[2024-08-08 19:33:22.666]# RECV ASCII>
L:1987 R:2140
[2024-08-08 19:33:22.867]# RECV ASCII>
L:2898 R:3154
[2024-08-08 19:33:23.064]# RECV ASCII>
L:3811 R:4172
[2024-08-08 19:33:23.267]# RECV ASCII>
L:4727 R:5192
[2024-08-08 19:33:23.469]# RECV ASCII>
L:5642 R:6214
[2024-08-08 19:33:23.675]# RECV ASCII>
L:6558 R:7235
Data Send
1. DCD 2. RXD 3. TXD 4. DTR 5. GND 6. DSR 7. RTS 8. CTS
Clear Clear
Hello Yahboom!
Send
Ready!
1603/0
RX:23589
TX:0
Reset