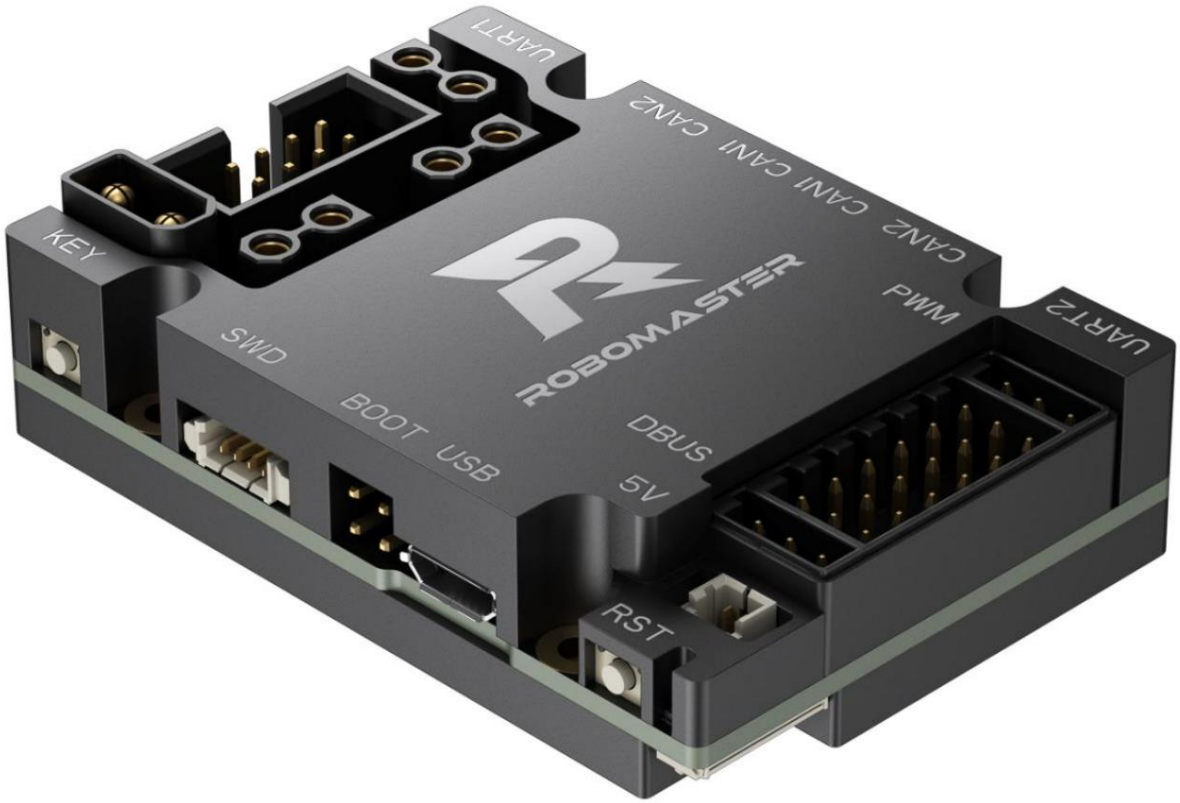
RoboMaster Development Board Type-C Embedded Software

Tutorial Documentation

v1.0 2020.01



Pre-reference reading

1. "RoboMaster Development Board C Type User Manual"
2. C language related books
3. The Definitive Guide to ARM Cortex-M3 and Cortex-M4
4. STM32F407IG related data sheet
5. FreeRTOS official website related documents

It is recommended that users first read the "RoboMaster Development Board Type C User Manual" to understand the RoboMaster Development Board Type C (hereinafter referred to as the Board C type) related functions and use methods, use the relevant interfaces of the development board C type correctly, and avoid the wrong use method causing the development board C

The programming language used by the C-type routines of the development board is the C language. It is recommended that the user learn the basic syntax of the C language. This document does not There are systematic explanations for C language; "ARM Cortex-M3 and Cortex-M4 Authoritative Guide", STM32F407IG related data

The manual and related documents on the freeRTOS official website are used as reference materials and can be consulted when necessary.

change log

date	Version revision record
2020.01.09	V1.0 is released for the first time.

content

0. Development board type C , getting started with cubeMX and keil

0.1 Knowledge points

ÿ Function introduction of C-type factory program of development board

ÿ cubeMX from new ioc project to generating keil project

ÿ Introduction to keil project settings

ÿ keil software debugging function introduction

ÿ Introduction to the overall functions of the RoboMaster robot

0.2 Course content

In this course, we first introduce the factory program function of RoboMaster development board C type (hereinafter referred to as development board C type);

After learning how to use cubeMX to generate keil project, learn the common settings of stm32 keil project, learn keil software

how to enter the debugging mode; finally, as the beginning of the tutorial, get an overview of the common functions of the RoboMaster robot and

Corresponding to the peripheral functions of stm32, guide the subsequent learning.

0.3 Basic Learning

RoboMaster development board C type adopts high-performance stm32 main control chip, supports wide voltage input, and integrates special expansion

extension interface, communication interface and high-precision IMU sensor, which can be used with RoboMaster products or other accessories.

The development board type C has the following peripherals: user-defined LED, 5V interface, BOOT configuration interface, micro USB interface

port, SWD interface, button, configurable I/O interface, UART interface, CAN bus interface, PWM interface, DBUS

interface, digital camera FPC interface, buzzer, voltage detection ADC, six-axis inertial measurement unit and magnetometer.

The development board C type has been programmed in the factory, you can connect the PC through the micro USB cable, and use the serial port tool to connect the development board C type.

The commonly used peripherals to operate, the operation is as follows:

ÿ On the first-level menu selection interface, you can input numbers 1-9 through the serial port tool to select the corresponding second-level display interface;

In the secondary display interface, input the letter q or Q through the serial port tool to exit the secondary display interface. One of the first selection interface

as the picture shows.


```
********************************
1.LED TEST
2.BUZZER TEST
3.LASER TEST
4.KEY TEST
5.ADC TEST
6.IMU TEST
7.DBUS TEST
8.PWM TEST
9.CAN TEST
********************************
```

ÿ The normal state of the LEDs is that the three-color LEDs light up in sequence. Display the current LED status on the LED display interface: "LED ON"
and "LED OFF". LED ON means that the three-color LEDs are all lit and emit white light; LED OFF means that the three-color LEDs are not
All light up. You can use the serial tool to input ON or OFF to switch the LED status. The LED display interface is as follows
as shown below.

```
********************************
LED :   OFF
********************************
********************************
LED :   ON
********************************
```

ÿ The buzzer will sound a boot sound when the development board Type C is powered on. Display the current buzzer status on the buzzer display interface:
"BUZZER OFF" and "BUZZER ON". BUZZER OFF means the buzzer does not sound; BUZZER ON
The theme song "You" of "Mecha Master" sounded for the buzzer. You can use the serial tool to input ON or OFF
The buzzer status is switched, and the buzzer display interface is shown in the figure below.

```
.............................
********************************
BUZZER:          OFF
********************************
********************************
BUZZER:          ON
********************************
```

ÿ Display the current 5V interface status on the 5V interface display interface: "LASER OFF" and "LASER ON". LASER
OFF means that the 5V interface does not output; LASER ON means LASER. You can use the serial tool to input ON or OFF
Switch the 5V interface status, the 5V interface display interface is shown in the figure below.

```
********************************
LASER:  OFF
********************************
********************************
LASER:  ON
********************************
```

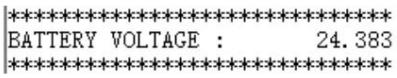
ÿ The current key state is displayed on the key display interface: "KEY OFF" and "KEY ON". KEY OFF means that the key is not in the
Pressed state; KEY ON means that the key is in the pressed state, and the key display interface is shown in the figure below.

```
.............................
********************************
KEY :   OFF
********************************
********************************
KEY :   ON
********************************
```

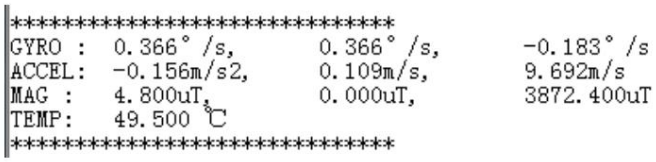
ÿ The current power supply voltage is displayed on the power supply voltage display interface: "BATTERY VOLTAGE: 24.000V". in

24.000V is the current power supply voltage, the actual value is subject to measurement, the LED display interface is like the power supply voltage display interface

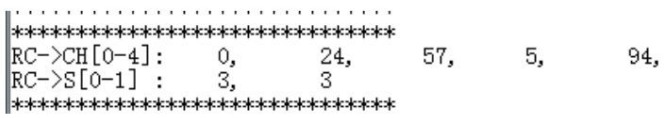
As shown below.



ÿ The gyroscope, accelerometer and magnetometer data are displayed on the IMU display interface. The IMU display interface is shown in the figure below.



ÿ The current remote control data status is displayed on the remote control display interface. The remote control data display interface is shown in the figure below.



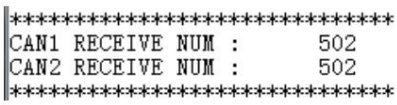
ÿ Display the current PWM output high level time status on the PWM display interface: "PWM: 1000". 1000 of them

It means that the high level time is 1000ms, and the PWM display interface is shown in the figure below.

ÿ Display the current number of data packets received by CAN on the CAN display interface: "CAN1 RECEIVE NUM: 1000",

"CAN2 RECEIVE NUM: 1000". Among them, 1000 means that the number of data packets received in 1 second is 1000,

The CAN display interface is shown in the figure below.



0.4 Program learning

0.4.1 Software Environment

Toolchain/IDE : MDK-ARM V5

STM32F4xx_DFP Packs:2.13.0

STM32CubeMx:5.2.1

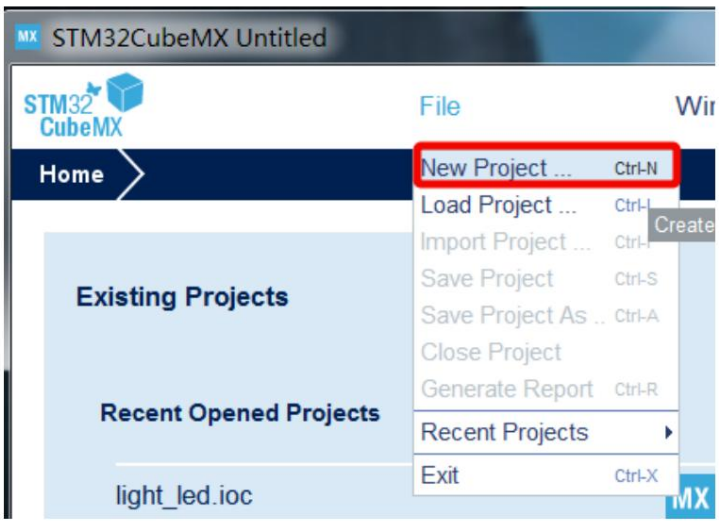
package version: STM32Cube FW_F4 V1.21.1

FreeRTOS version: 10.0.1

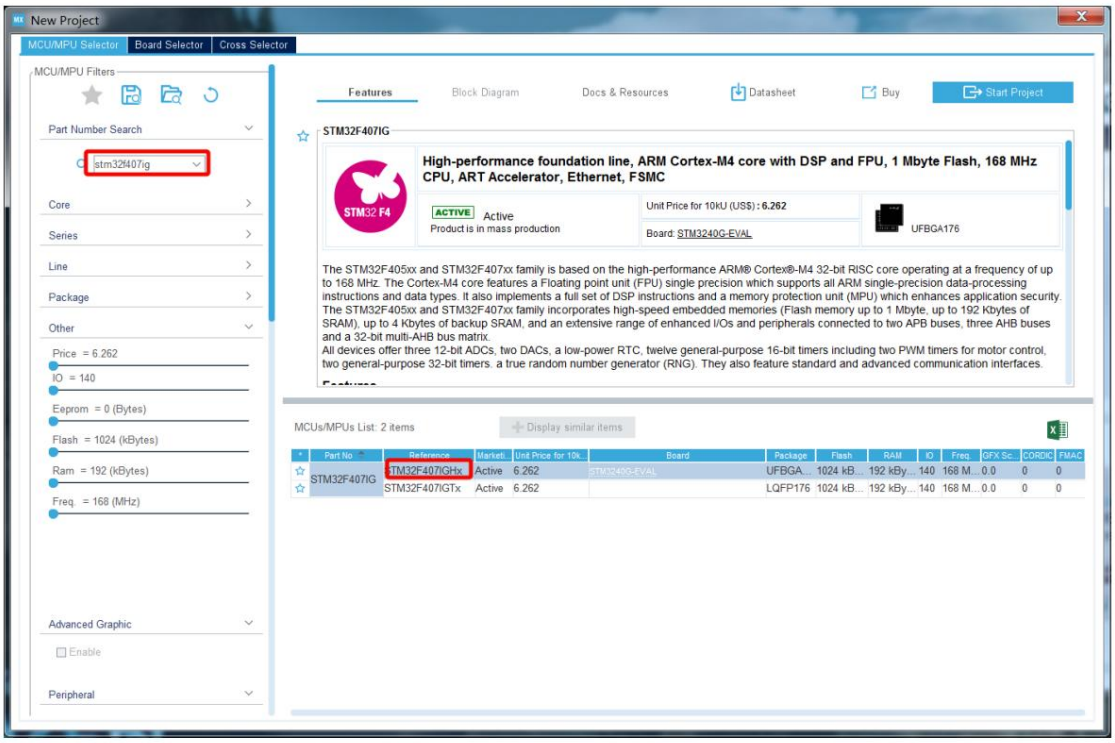
CMSIS-RTOS version: 1.02

0.4.2 cubeMX new project

1. Open cubeMX software and select "New Project" in the file option;

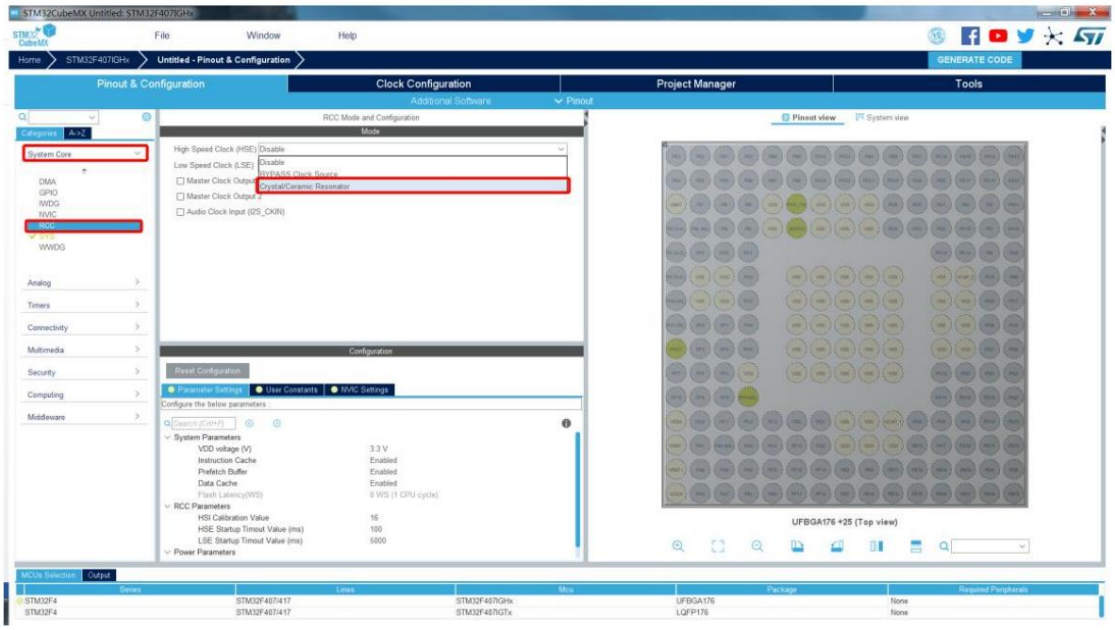


2. Search for stm32f407ig and select the STM32F407IGHx chip;

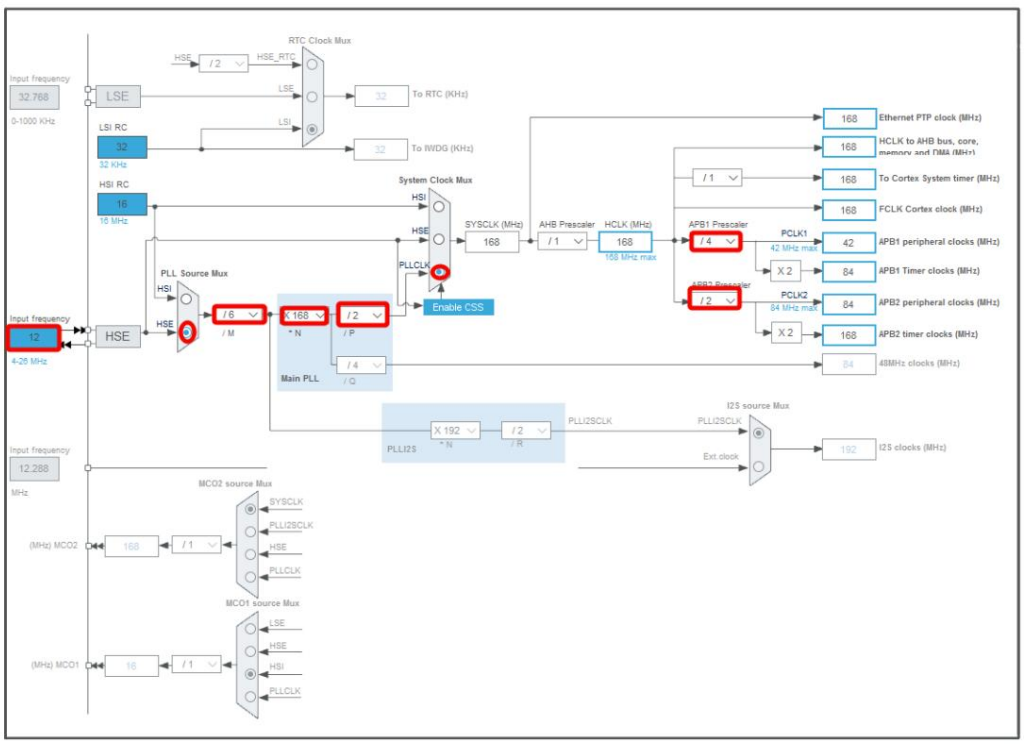


3. Select RCC option under System Core, High Speed in RCC mode and Configuration

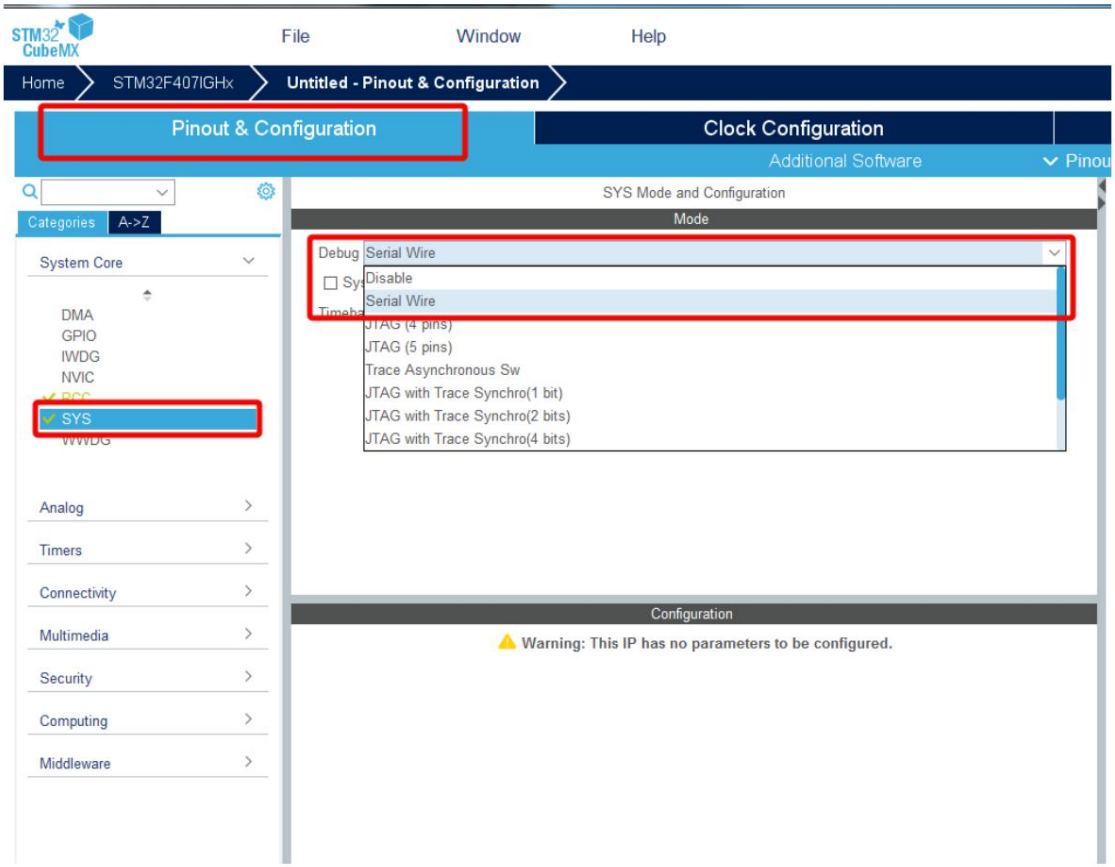
Select Crystal/Ceramic Resonator under Clock(HSE);



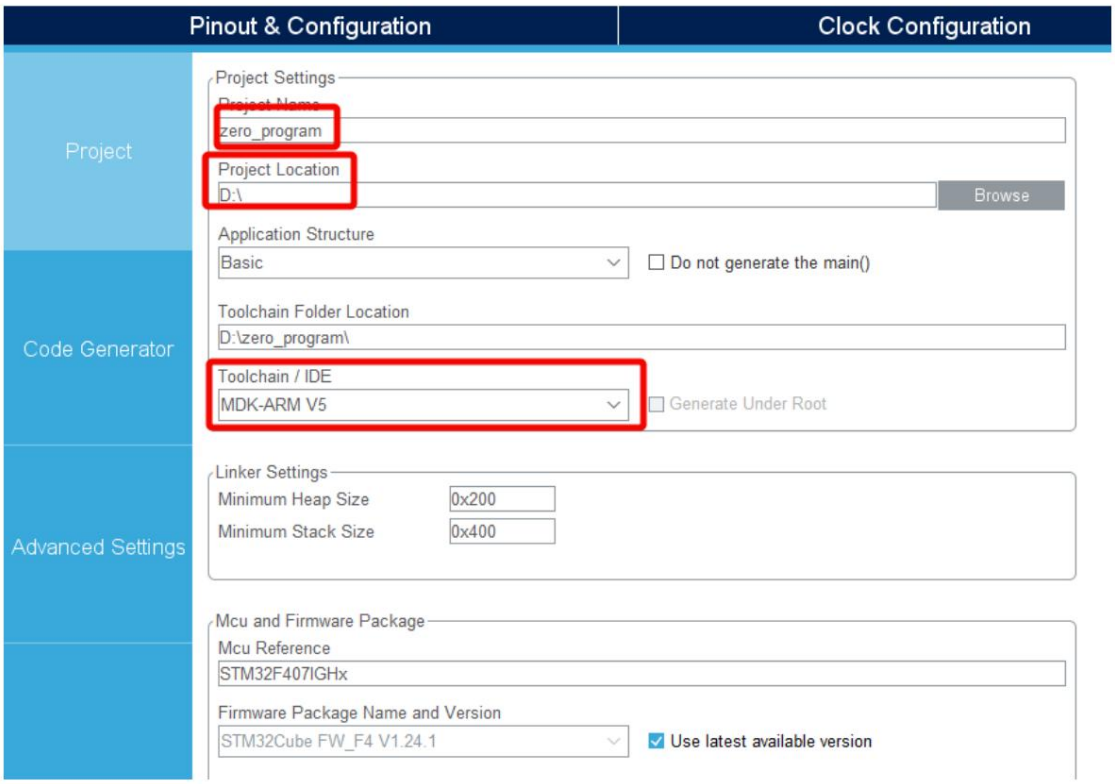
4. Click Clock Configuration at the top to configure the main frequency; set Input frequency to 12, click Next to the HSE round button, configure /M as /6, configure *N as X168, configure /P as /2, select PLLCLK round button, configure APB1 Prescaler as /4, and configure APB2 Prescaler as /2;



5. Click Pinout & Configuration at the top, select SYS, and select Serial Wire in the Debug drop-down box;

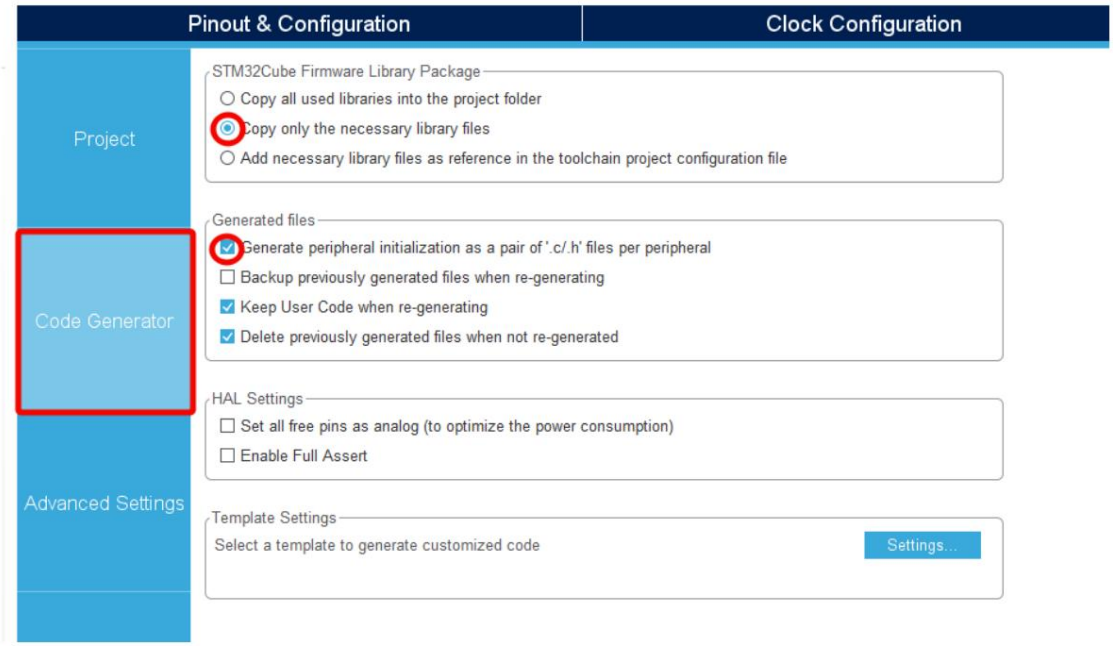


6. Click the Project Manager at the top, name the project, select the storage directory, and select MDK in Toolchain/IDE ARM V5 ÿ



7. Click the Code Generator next to it and check Copy only the necessary library files and Generate

peripheral initialization as a pair of '.c/.h' files per peripheraly

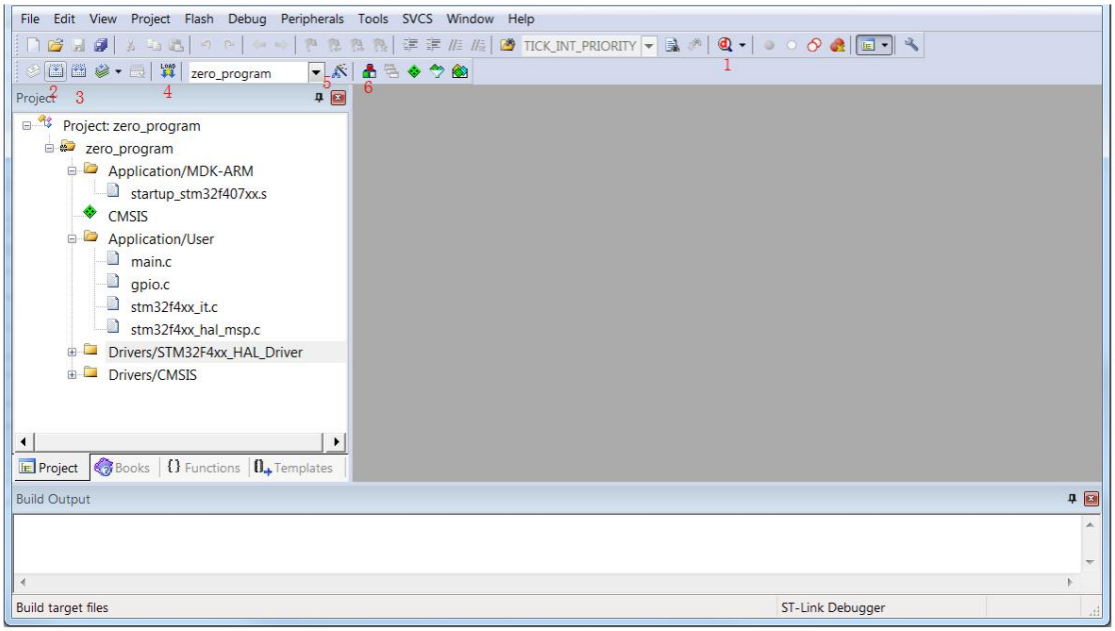


8. Click GENERATE CODE at the top, wait for the code to be generated, and open the project.

0.4.3 Simple introduction to keil software

1. Open the generated project, the keil interface is as shown in the figure, where 1 is the debug mode, 2 is the file changed by compilation, and 3 is the editor

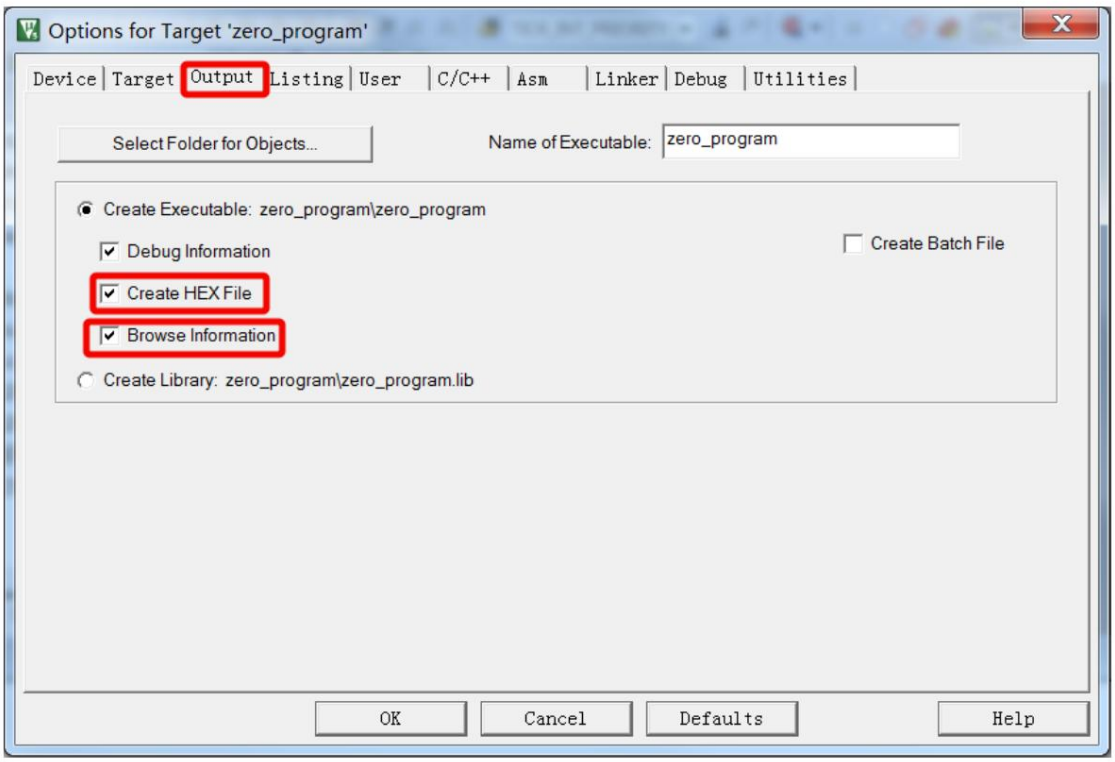
Translate all files, 4 is the download button, 5 is the project setting option, and 6 is the project directory;



2. Click 5 Project Settings, make project-related settings, select Output, where Create HEX file is whether to generate

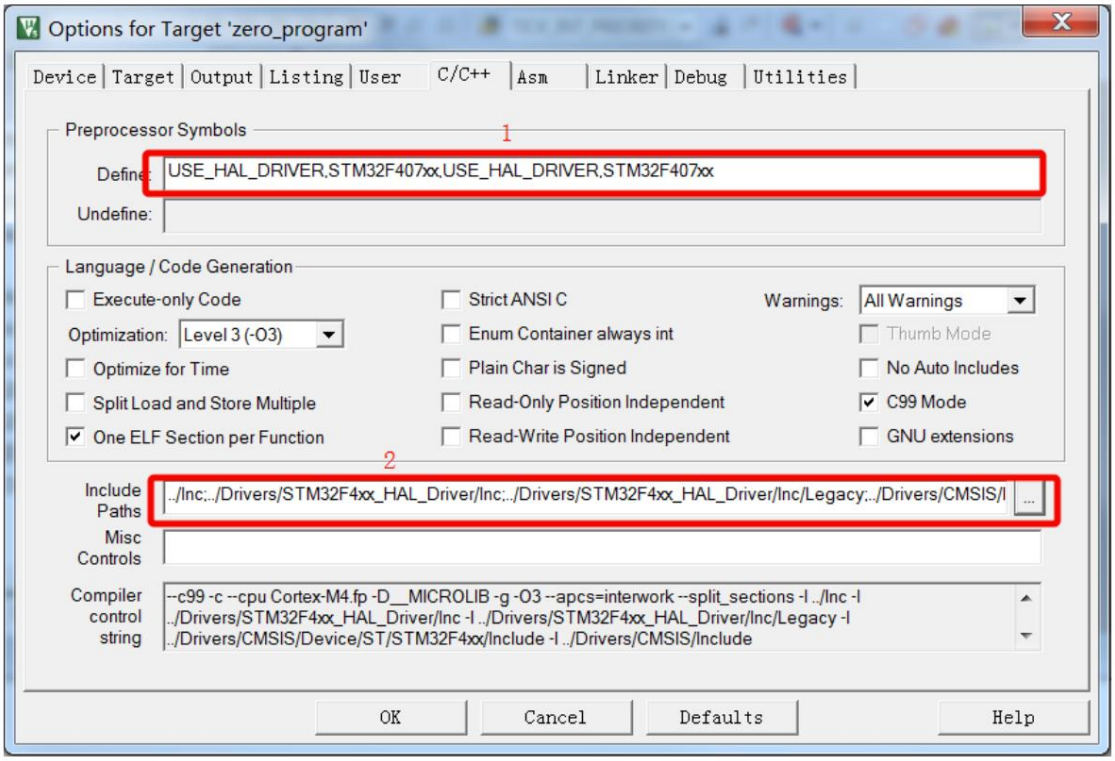
HEX file, Browse Information is whether to add browsing information, select Yes, you can use the right mouse button to click the function

Perform a jump operation, but it will increase the compilation time;

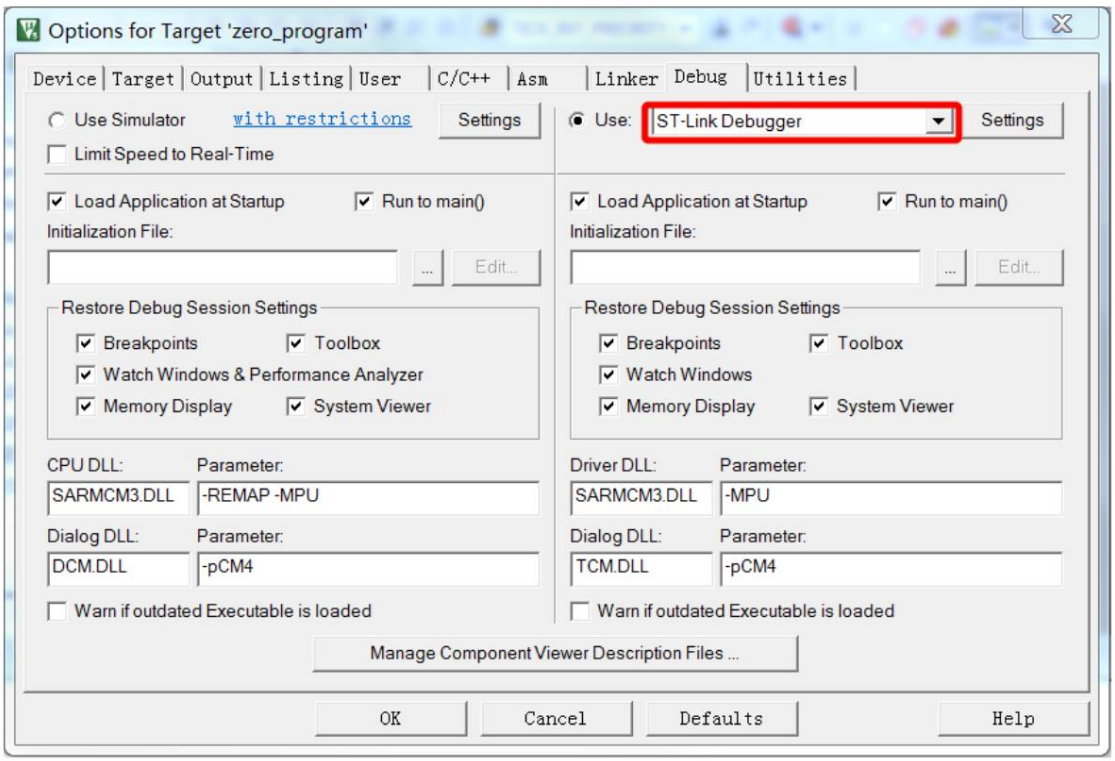


3. Click C/C++, in which box 1 is the project macro definition setting, you can add macro definition here; box 2 is the header file reference object.

For the h file created by the project itself, you need to enter the directory here;

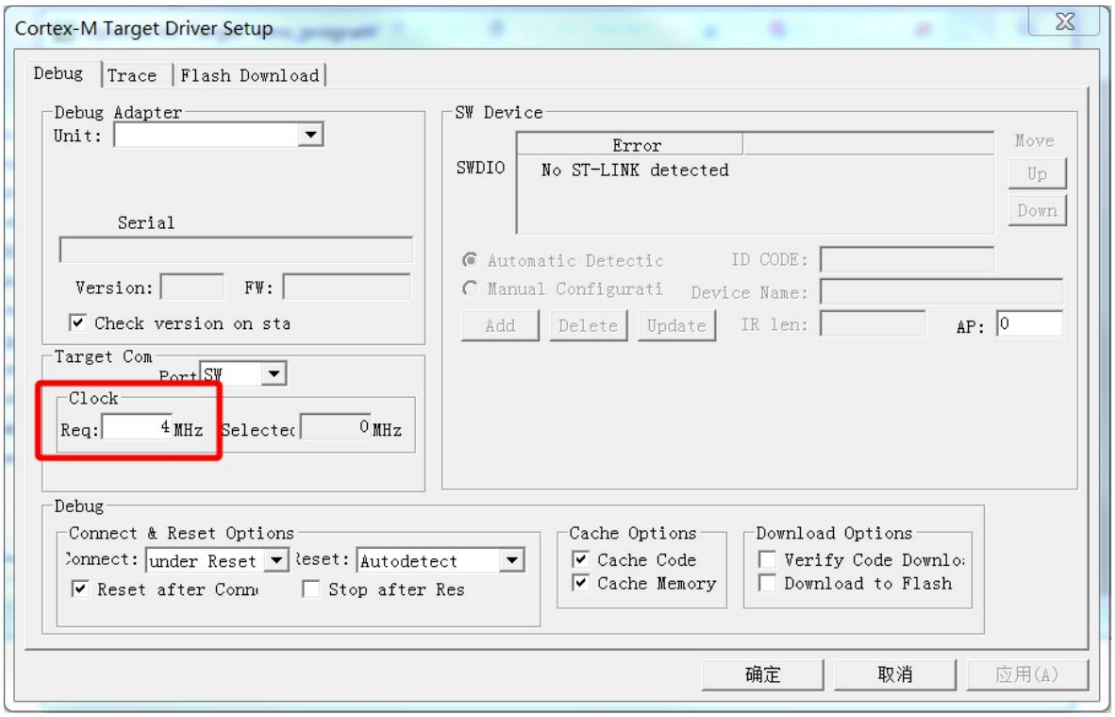


4. Click the Debug option to set the corresponding downloader;



5. Click the Settings option next to the downloader to set the downloader-related settings, in which the frequency of the downloader is set in the Clock.

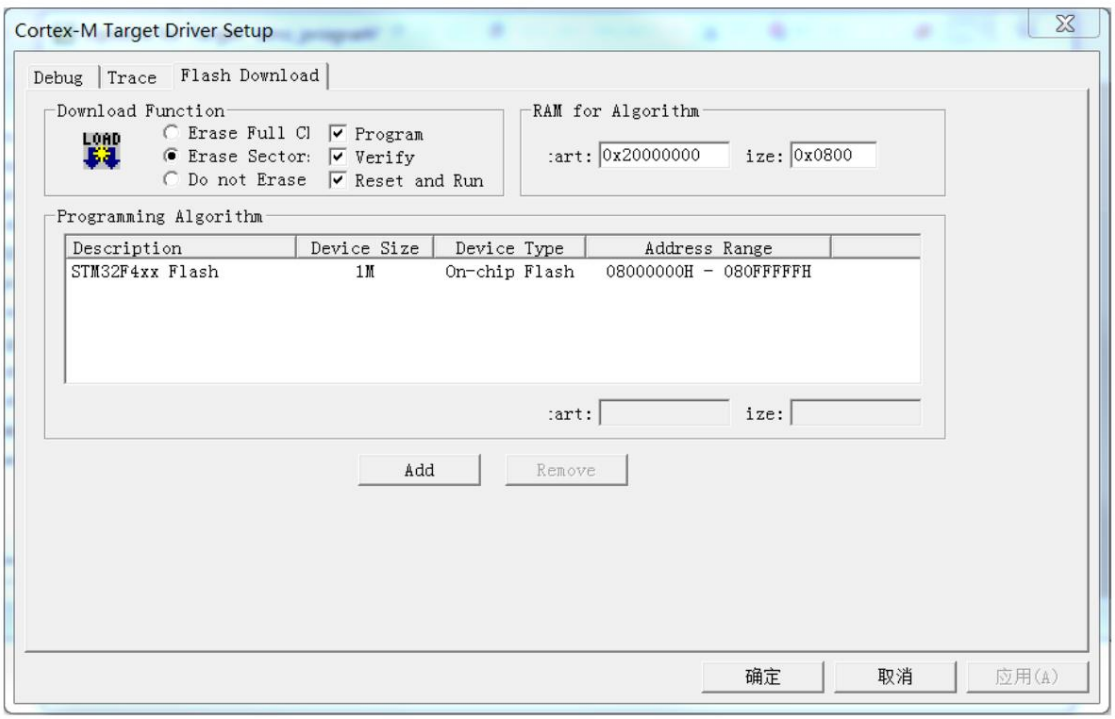
The higher the frequency, the faster the download speed, but it is easy to be interfered;



6. Click Flash Download, where Erase Full Chip represents the flash that erases all pages in the chip when downloading.

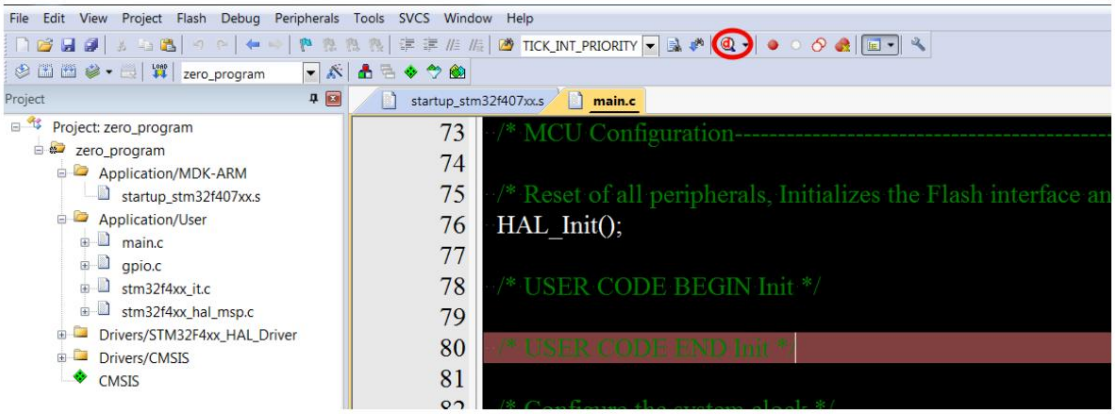
Erase Sectors means to erase the flash of some pages when downloading, Do not Erase means not to erase the flash when downloading ,

Click Reset and Run to run the program immediately after downloading the program.

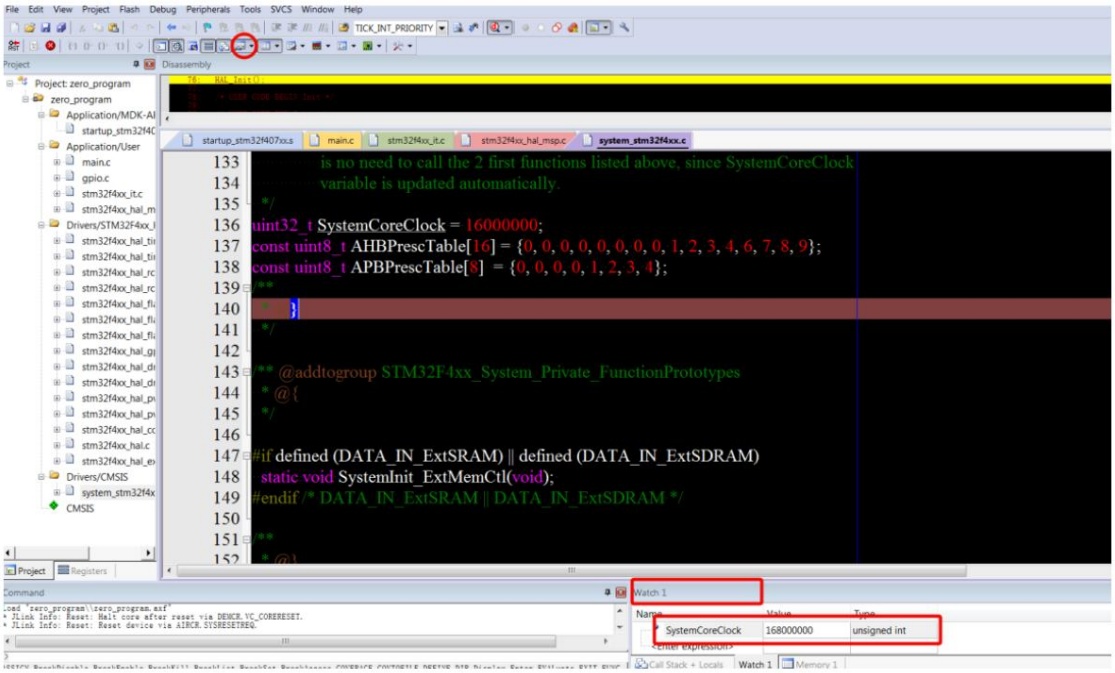


0.4.4 Keil 's debug mode

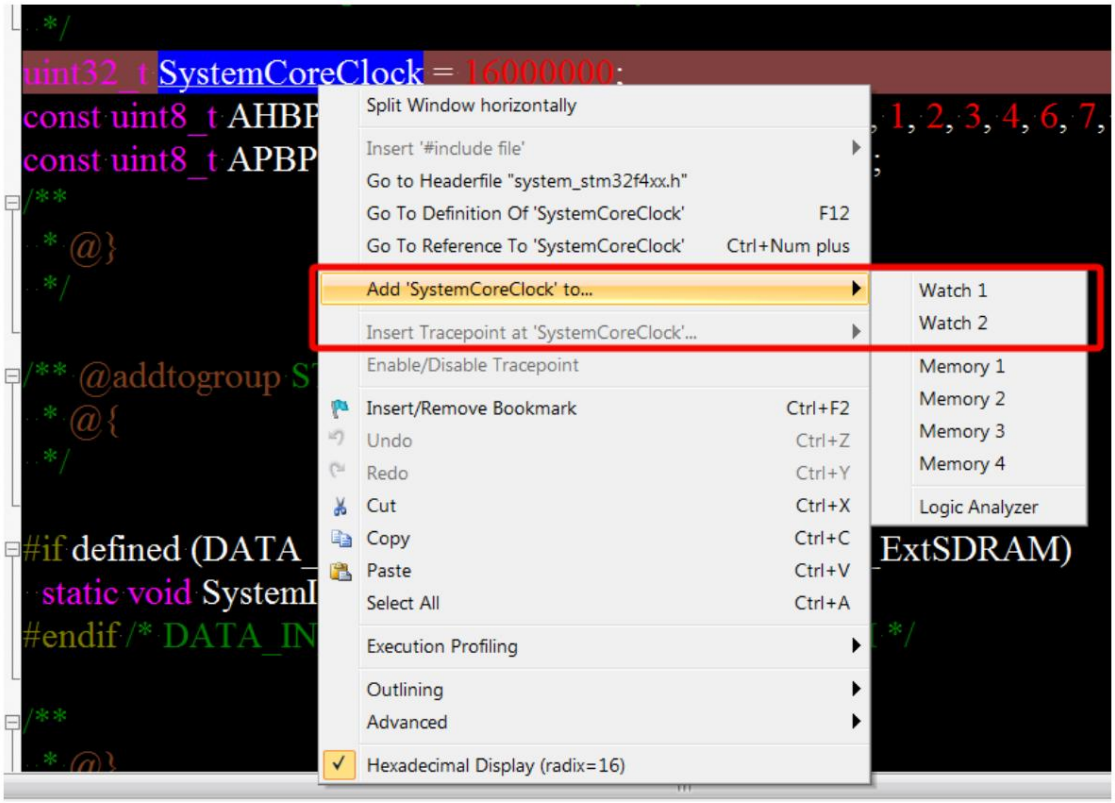
1. After compiling in keil, click the debug button to enter the debug mode;



2. After entering the debug mode, you can select the watch window, and the watch window can refer to the numerical value of the variable;

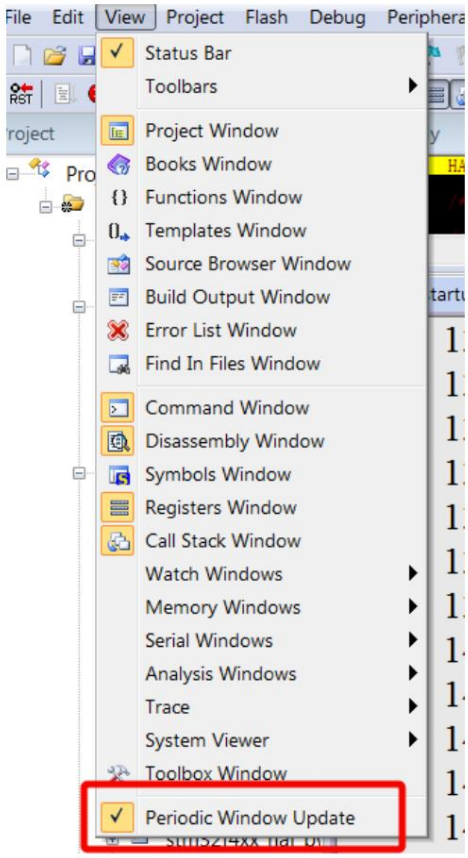


3. You can right-click the variable you want to see and add it to the watch window;



4. If the data does not change during program operation, in addition to the case where the data itself does not change, it may also be due to

PeriodicWindows Updateÿ



0.5 RoboMaster Robot Function Introduction

Common functions of RoboMaster robots are as follows:

1. Calibration function: provide the calibration function of the robot, and use the flash read and write function of stm32;

2. Chassis control function: complete the McLun motion control of the chassis, and use the CAN bus function of stm32.

3. Offline judgment function: to judge whether the device is offline, use the buzzer of type C of the development board;

4. PTZ control function: complete the angle control of the PTZ, and use the CAN bus function of stm32;

5. Attitude calculation function: complete the angle fusion of the gyroscope accelerometer, solve the Euler angle, use the SPI and stm32
The I2C bus reads the relevant data, uses the external interrupt to stm32 as the flag of data update, and uses the external interrupt to stm32
PWM controls the heating resistance;

6. RGB switching of LED: use three-color LED to complete RGB display, breathing light effect, use the LED of type C of the development board
lamp;

7. OLED display function: display the information and use the I2C bus of stm32;

8. Referee system data parsing function: use single-byte parsing of the referee system data, use the stm32 serial port and DMA
Function;

9. Remote control data parsing function: parsing the data sent by the receiver, using the serial port and DMA function of the stm32;

10. Servo control function: Control the servo through the buttons and use the PWM function of stm32.

11. Shooting control: control the bomb feeder, complete the launch logic, use the GPIO reading to stm32 and the CAN general
String.

12. Power sampling function: sample the power supply voltage and estimate the current battery power, using the ADC sampling function of stm32.

0.6 Course Summary

This course is a basic introductory course, learn how to use cubeMX and keil software to create, compile, download,
Debugging and other functions; introduced the program functions that come with the development board C type and the common functions of the RoboMaster robot,
As a guide for learning, to pave the way for subsequent learning.

1. Light up the **LED**

1.1 Knowledge points

ÿ Basic knowledge of LED lights

ÿ Understand the on-off characteristics of triodes

ÿ Pull-up and pull-down resistors on the hardware schematic

ÿ Basic operation of configuring GPIO in cubeMX

1.2 Course content

In this course, you will learn how to light LED lights, understand the basic principles of LEDs, and learn about simple hardware. by using

The cubeMX software completes the configuration of the pins, and then writes the program to make the output of the corresponding pin a high level, which is connected through the triode.

Off action, the current will pass through the LED, thus emitting light.

1.3 Basic Learning

1.3.1 Basic knowledge of LED lights

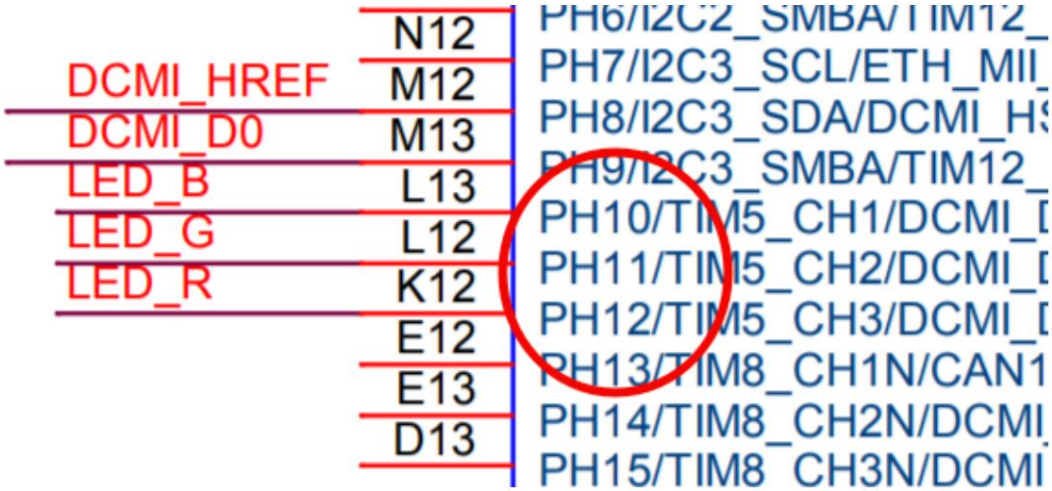
LED is a light-emitting diode. When there is current in the LED, it will emit light. Within the safe current range, the greater the current, the brighter the light.

brighter. The actual LED light is shown in the figure:

1.4 Program Learning

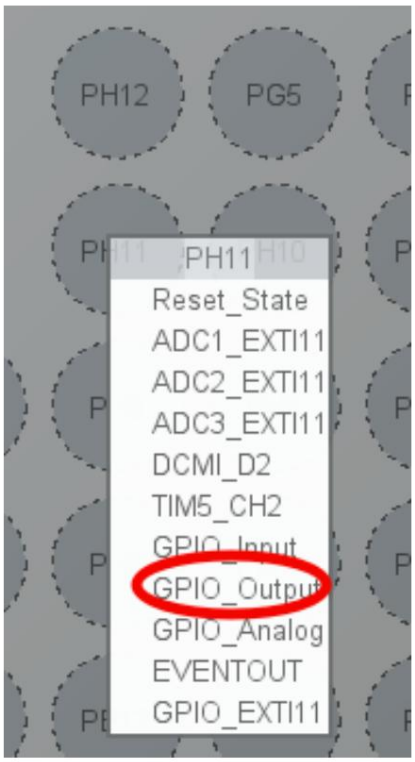
1.4.1 Basic operation of configuring **GPIO** in cubeMX

1. It can be seen from the schematic diagram that the pins of the three LED lights are PH10, PH11, PH12, as shown in the figure:



2. Configure GPIO as output mode in cubeMX, find the corresponding pin in cubeMX and configure it as GPIO_Output

model.

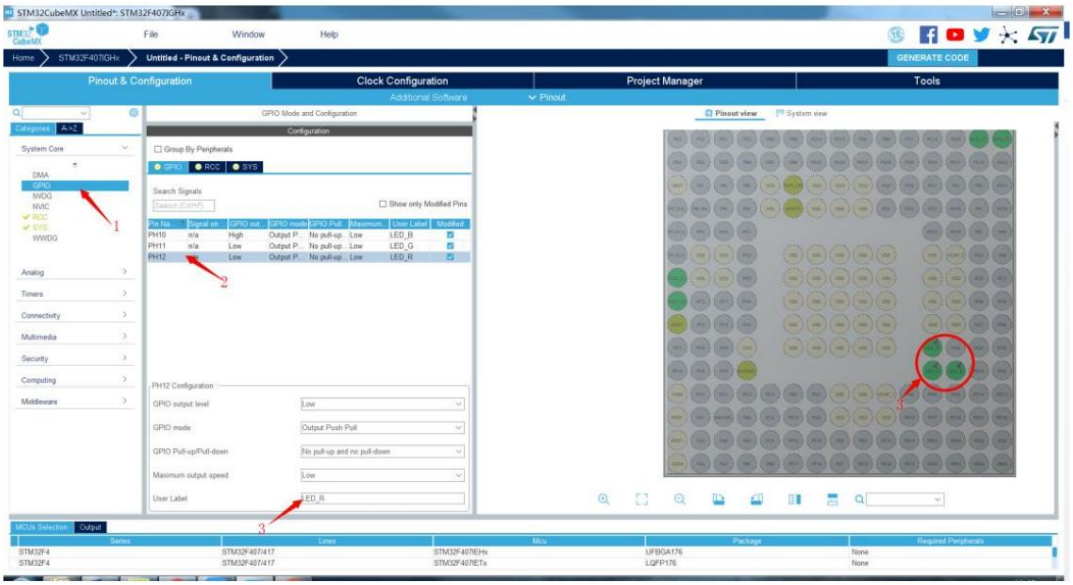


3. Modify the name of the corresponding pin in cubeMX.

1) Find System core->GPIO on the left;

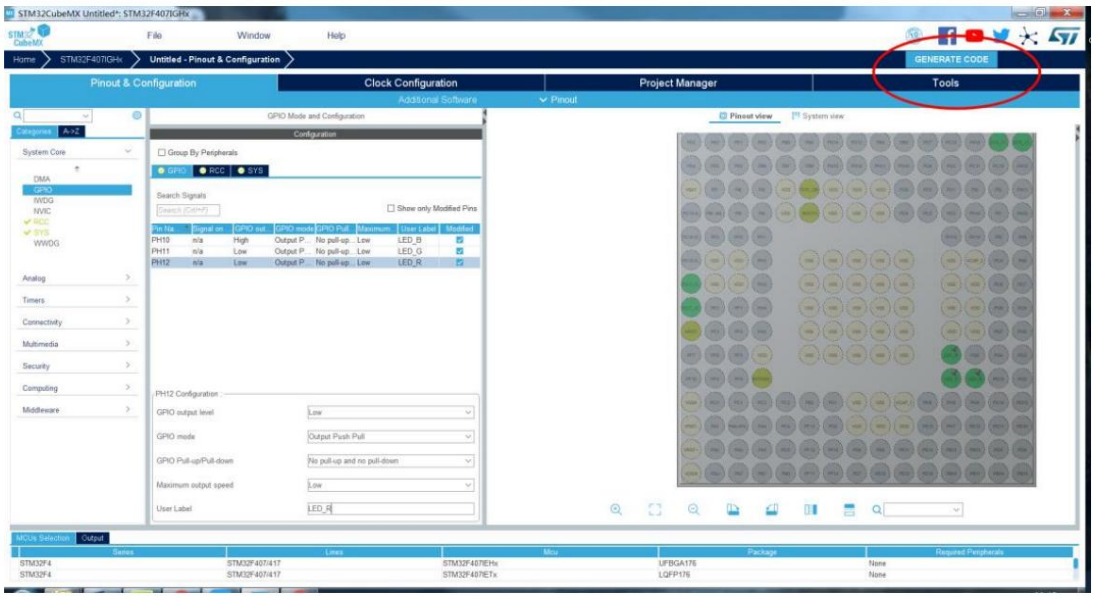
2) Find the corresponding GPIO, such as PH12;

3) Fill in the name of the user label in the configuration list below, and it will be updated in the chip thumbnail after filling in.



4. Generate code

Click the GENERATE CODE button.



1.4.2 HAL_GPIO_WritePin function explanation

The HAL library provides a function to operate the GPIO level: HAL_GPIO_WritePin function

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
```


Function name	HAL_GPIO_WritePin
function	Make the corresponding pin output high level or low level
return value	Void
Parameter 1: GPIOx	Corresponds to a GPIO bus, where x can be A…I. For example, PH10, enter GPIOH
Parameter 2: GPIO_Pin	Corresponds to the number of pins. Can be 0-15. For example, PH10, enter GPIO_PIN_10
Parameter 3: PinState	GPIO_PIN_RESET: output low level GPIO_PIN_SET: output high level

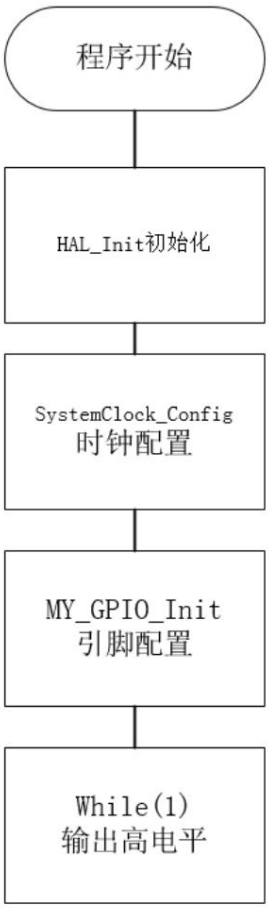
1.4.3 Program Flow

The program is initialized by HAL_Init, GPIO is initialized, and enters the main loop. In the main loop, all three LED pins are output.

output high level to light up the LED light. The main loop code is as follows:

```
//set GPIO output high level
//Set GPIO output high level
HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_SET);
```

The program flow chart is shown in the figure:



1.4.4 Effect display

Since the three-color LEDs are all lit, red, green, and blue are the three primary colors, and white light is synthesized, as shown in the figure.

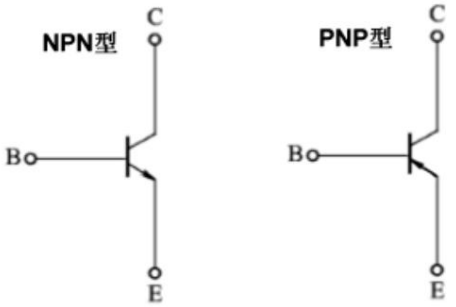
1.5 Advanced Learning

1.5.1 On-off characteristics of triode

From the schematic diagram of the LED, it can be seen that the connection of the LED is not directly connected to the stm32 pin, this is because for the stm32

In other words, the output current capability of the pin is limited, and the LED light needs to be turned on and off through a triode. Common

The triode is divided into NPN type and PNP type, as shown in the figure.



	common ground	difference
NPN type	ÿ Both have three connection points ÿ Both have the effect of current amplification ÿ The control methods are all controlled by the level of point B: ÿ Point B is at high level: CE line is on ÿ Point B is at low level: CE line is not conducting	When the voltage at point B is higher than the voltage at point E, three The diode is turned on, and the current direction is from point C to Point E
PNP type		When the voltage at point B is lower than the voltage at point E, three The pole tube is turned on, and the current direction is from point E to point C

In this design, the triode works in a saturated state when it is turned on, and works in a cut-off state when it is not turned on; it is equivalent to a switch,

While point B corresponds to the hand pressing the switch, point CE corresponds to the line to which the switch is connected. When point B is high, the switch is closed

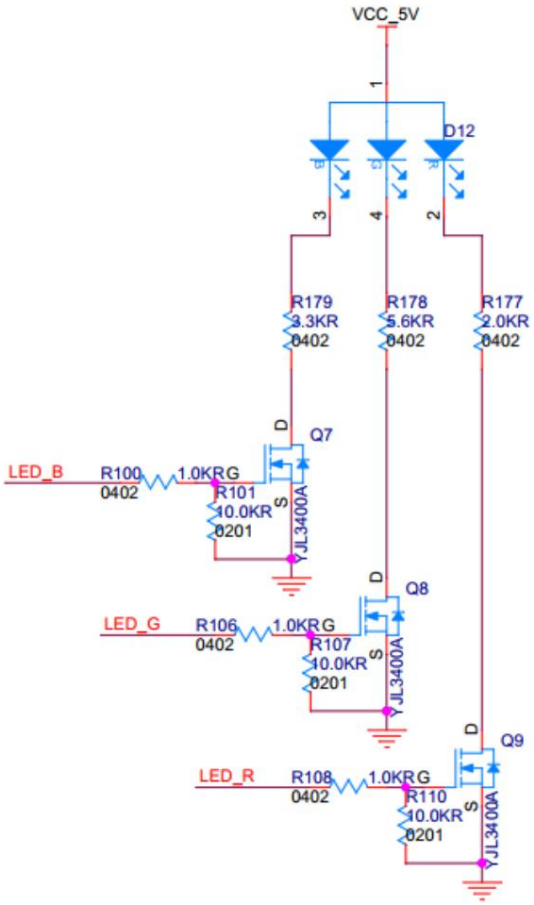
When the switch is closed, the CE line is connected; when the B point is at a low level, the switch is disconnected and the CE line is disconnected.

1.5.2 Pull-down resistor of LED

The voltage at point B has a control effect on the triode. When the program does not have control behavior, the voltage needs to be controlled to a low level.

To ensure that the device is not accidentally triggered, it is mainly a pull-down resistor for the triode, and there is also a pull-up resistor, which can be used from the LED.

Schematic to find the pull-down resistor.



LED

From the schematic diagram of the LED light, when the three pins of LED_B, LED_G and LED_R are not in the high-level output state, then

Then the control end of the triode will be pulled low by the pull-down resistor in the red circle, when LED_B, LED_G and LED_R

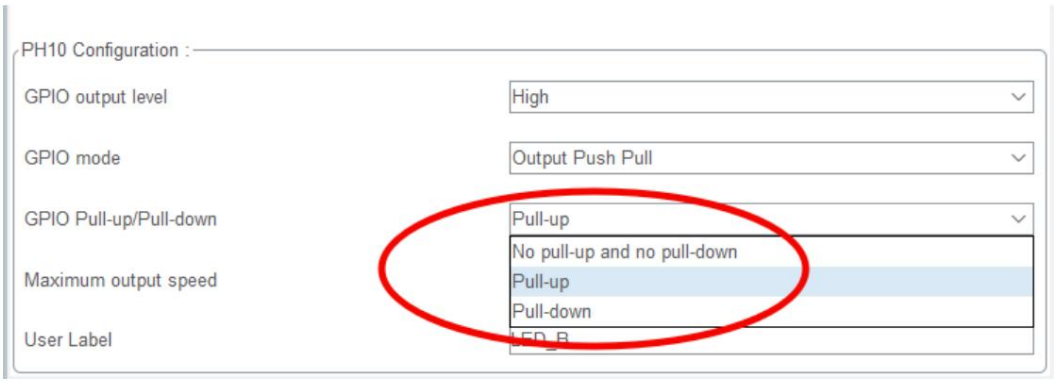
In the high-level output state, after the voltage divider between 1kÿ and 10kÿ in the schematic diagram, the base control terminal of the triode will become

high level, so the control terminal voltage of the triode will become high level.

To configure the pins of stm32 in cubeMX, you can choose to configure a pull-up resistor or a pull-down resistor, as shown in the figure. Pull

up is to configure the pin as a pull-up resistor, Pull-down is to configure the pin as a pull-down resistor, No pull-up and no pull-down

It is the configuration pin that is neither pulled up nor pulled down.

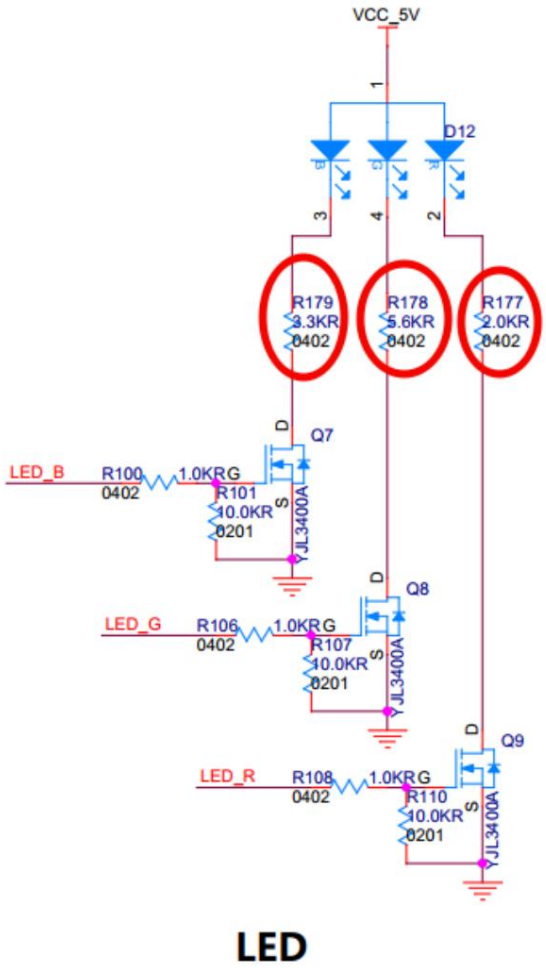


1.5.3 Current limiting resistor on the hardware schematic

LED lights need to work within a suitable current range, so a current limiting resistor is needed to limit its current, as shown in the figure below

of R177, R178, R179. The current limiting resistor ensures that the current through the LED in the circuit does not exceed its rated current. also,

Can also be used to adjust the brightness of LEDs.



LED

As shown above, on the schematic of the LED, there are three current limiting resistors. Among them, the green light has a larger current-limiting resistance, and the blue light has a larger current-limiting resistance.

The current-limiting resistance is small, because the human eye is sensitive to different wavelengths of light, that is, people are sensitive to different wavelengths of the same light intensity.

The light feels different brightness, green light is the most sensitive, green light needs to pass a small current, red light needs to pass a normal current,

The blue light needs to pass a large current to ensure that the human eye sees the same brightness of the three LED lights.

1.6 Course Summary

The GPIO output operation is the most basic operation in stm32, and a switch-like control is completed through high and low level control, while

The high level corresponds to the number 1 in the computer, and the low level corresponds to the number 0 in the computer. The high and low levels are constantly changing.

Corresponding to the digital changes inside the computer, stm32 outputs signals to the outside world in this way. In addition to this lesson, we also learn about LEDs

The use of lamps, triodes, and resistors are commonly used devices in robots. LEDs are presented in the form of lighting effects for the teams

Team reminder.

2. Blink LED

2.1 Knowledge points

ÿ Three delay methods

ÿ HAL_Init function

ÿ Tick timer

2.2 Course content

In this course, you will learn how to control the level change of stm32, learn three delay methods in stm32, and learn about Hal_Init
The function of the function, understand the principle and function of the tick timer. Use cubeMX software to complete the pin configuration, then write
The program, through the pin level inversion flip function HAL_GPIO_TogglePin and the delay function, realizes the blinking of the LED light
Effect.

2.3 Basic Learning

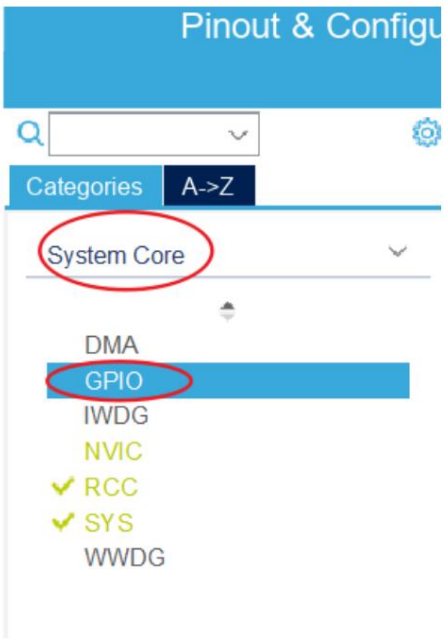
2.3.1 GPIO flip speed

In this section, you will learn how to configure the toggle speed of GPIO through cubeMX. In different cases, GPIO pins are required
The pins will output square waves of different frequencies. For example, when the led is flashing, it only needs a frequency of 1Hz to see obvious flashing
effect, but when the software simulates I2C, it needs hundreds of KHz frequency to complete normal communication. In cubeMX you can
According to different output frequency requirements, set different GPIO flip speeds.

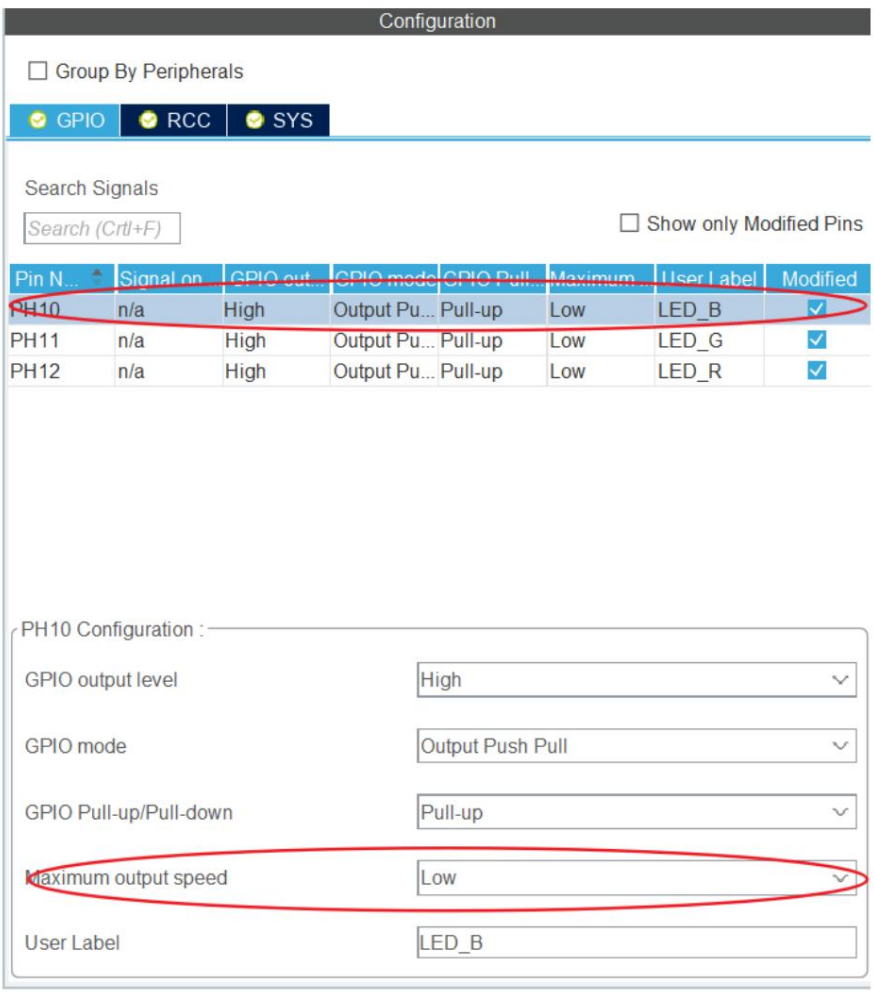
1. Use the method described in the previous lecture to enable the output function of the PH10, PH11, and PH12 pins.

2. There is a GPIO option under System Core in the left sidebar of cubeMX.

The configuration information of the enabled pins is shown in the figure:



3. Under this tab, you can select the GPIO that needs to be configured and view its detailed status, among which the Maximum output speed is the selectable flip speed mode, as shown in the figure, the corresponding flip mode is Low, which is Low fast output mode.

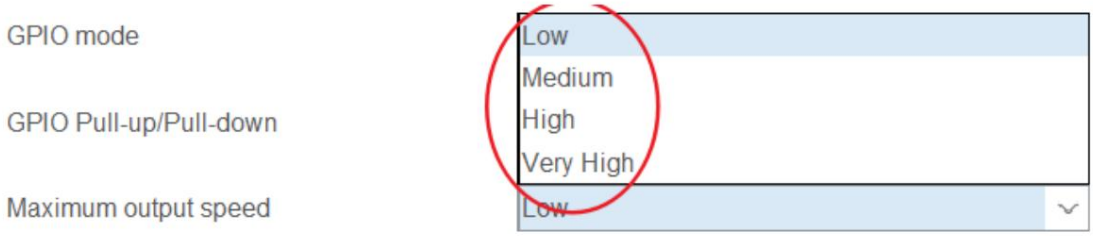


Click the small arrow at the right end, the optional output speed is divided into four gears: Low, Medium, High, Very High, as shown in the figure.

Generally, when using GPIO output to drive LED and other functions, select the Low gear flip speed, and the GPIO generally used for communication

It needs to be set to High or Very High. The specific setting can be adjusted according to the relevant communication protocol to the flip speed of GPIO.

Setup is required.



2.4 Program learning

2.4.1 Introduction to counting delay

It takes time to execute any instruction in stm32. Achieving a delay effect allows stm32 to continuously execute a

Segment count loop until it counts to the set number, and then let stm32 exit the loop.

In the program, the microsecond-level delay function is completed through the user_delay_us function. The implementation of the function is as follows:

```
void user_delay_us(uint16_t us)
{
    for(; us > 0; us--)
    {
        for(uint8_t i = 50; i > 0; i--)
        {
            ;
        }
    }
}
```

Initialize the inner loop variable i to 50, and the assignment of the outer loop variable is determined by the entry parameter of the function.

The number corresponds to the number of outer loops, and then the corresponding time delay is implemented accordingly.

Similar to the implementation of microsecond-level delay, adding a layer of external loop can achieve millisecond-level delay, and the corresponding function is

user_delay_ms, its specific implementation is as follows. This function calls the user_delay_us function and assigns its parameter

With a value of 1000, 1 millisecond is equal to 1000 microseconds. The delay time is converted to millisecond level by means of cyclic calling.


```
void user_delay_ms(uint16_t ms)
{
    for(; ms > 0; ms--)
    {
        user_delay_us(1000);
    }
}
```

2.4.2 **Nop** Delay Introduction

Using the nop function is the second delay method, and its principle is similar to the timing delay. Also by repeatedly executing instructions until Consume time before proceeding to the next step. But unlike the count delay, the nop delay is passed through the no-op instruction __nop() function to implement the delay. When stm32 executes to __nop(), it can be understood that in the current instruction cycle, stm32 does not do any work.

In the same way, microsecond-level delays nop_delay_us and nop_delay_ms can be completed in the same way

```
void nop_delay_us(uint16_t us)
{
    for(; us > 0; us--)
    {
        for(uint8_t i = 10; i > 0; i--)
        {
            __NOP();
            __NOP();
            __NOP();
            __NOP();
            __NOP();
            __NOP();
            __NOP();
            __NOP();
            __NOP();
            __NOP();
```

```
                __NOP();
                __NOP();
                __NOP();
                __NOP();
                __NOP();
            }
        }
}
void nop_delay_ms(uint16_t ms)
{
        for(; ms > 0; ms--)
        {
            nop_delay_us(1000);
        }
}
```

2.4.3 Introduction of tick timer and **HAL_Init** initialization

Before introducing the third delay method, the tick timer and its initialization are introduced first.

The tick timer, also known as SysTick, is a built-in countdown timer in stm32, which is triggered every time it counts to 0

SysTick interrupts and reloads register values. The initialization of the tick timer is done in the HAL_Init function, which is configured as 1ms

interruption. The contents of HAL_Init are as follows, as you can see from the comments: In HAL_Init, SysTick and bottom

Initialization of layer hardware.

```
HAL_StatusTypeDef HAL_Init(void)
{
   /* Configure Flash prefetch, Instruction cache, Data cache */
#if (INSTRUCTION_CACHE_ENABLE != 0U)
   __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
#endif /* INSTRUCTION_CACHE_ENABLE */

#if (DATA_CACHE_ENABLE != 0U)
```



```
    __HAL_FLASH_DATA_CACHE_ENABLE();
#endif /* DATA_CACHE_ENABLE */

#if (PREFETCH_ENABLE != 0U)
    __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
#endif /* PREFETCH_ENABLE */

    /* Set Interrupt Group Priority */
    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

    /* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) */
    HAL_InitTick(TICK_INT_PRIORITY);

    /* Init the low level hardware */
    HAL_MspInit();
    /* Return function status */
    return HAL_OK;
}
```

In the implementation of HAL_Init, pay attention to the tick timer initialization function.

```
__weak HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)
```

The HAL_InitTick function will set the timing period of SysTick to 1ms, that is, the frequency is 1000Hz, and make SysTick open start working.

Whenever the tick timer decrements to 0, an interrupt is triggered, causing the program to enter the SysTick interrupt handler

```
void SysTick_Handler(void)
{
    /* USER CODE BEGIN SysTick_IRQn 0 */

    /* USER CODE END SysTick_IRQn 0 */
    HAL_IncTick();
    /* USER CODE BEGIN SysTick_IRQn 1 */
```

```
    /* USER CODE END SysTick_IRQn 1 */
}
```

In it, the HAL_IncTick function is called, which is implemented as follows

```
__weak void HAL_IncTick(void)
{
    uwTick += uwTickFreq;
}
```

Stored in the uwTick variable is the elapsed time (ms) since the initialization of the Systick of stm32, the uwTick's

Existence is equivalent to providing an absolute time base for the entire program, and the HAL_Delay function delay function is passed through

The value of uwTick is done.

To get the current uwTick value, you can use the function HAL_GetTick provided by the HAL library

```
uint32_t HAL_GetTick(void)
```

Function name	HAL_GetTick
function	Returns the uwTick value at the current moment
return value	uwTick value at the current moment

2.4.4 Introduction to HAL_Delay

The HAL library provides functions for millisecond delays, the HAL_Delay function (using the _weak modifier to indicate that the function is

user-redefinable).

```
__weak void HAL_Delay(uint32_t Delay)
```

Function name	HAL_Delay
function	Make the system delay the corresponding millisecond time
return value	void
parameter	Delay, the corresponding delay in milliseconds, for example, a delay of 1 second is 1000

In the implementation of HAL_Delay, the HAL_GetTick function is called to obtain the reference time uwTick, indicating that HAL_Delay

The implementation of the function is based on the tick timer (Systick).

```
__weak void HAL_Delay(uint32_t Delay)
{
  uint32_t tickstart = HAL_GetTick();
  uint32_t wait = Delay;

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

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

The above three delay methods each have their own characteristics. Both count delay and nop delay require users to write functions to implement them.

Now, it is more troublesome, and it is more convenient to directly call the HAL_Delay function provided by the HAL library. but HAL_Delay

Only millisecond-level delay can be achieved. If a shorter delay function is required, a user-written delay function must be used.

2.4.5 Introduction to HAL_GPIO_TogglePin

The cubeMX configuration used in this lab is the same as the previous one.

In the previous lecture, the function to operate the GPIO level was introduced: HAL_GPIO_WritePin, through which you can set

GPIO outputs high or low level. In order to realize the flickering of the LED light, it is only necessary to output the high and low levels alternately.

This can be achieved through two sentences of HAL_GPIO_WritePin, respectively setting the pin to a high level and a low level.

Flashing function now.

But in addition to the above methods, there are easier functions to implement pin level inversion -- provided by the HAL library

HAL_GPIO_TogglePin function.

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Function name	HAL_GPIO_TogglePin
function	Flip the level of the corresponding pin
return value	Void
Parameter 1: GPIOx	Corresponds to a GPIO bus, where x can be A…I. For example, PH10, enter GPIOH
Parameter 2: GPIO_Pin	Corresponds to the number of pins. Can be 0-15. For example, PH10, enter GPIO_PIN_10

2.4.6 Program Flow

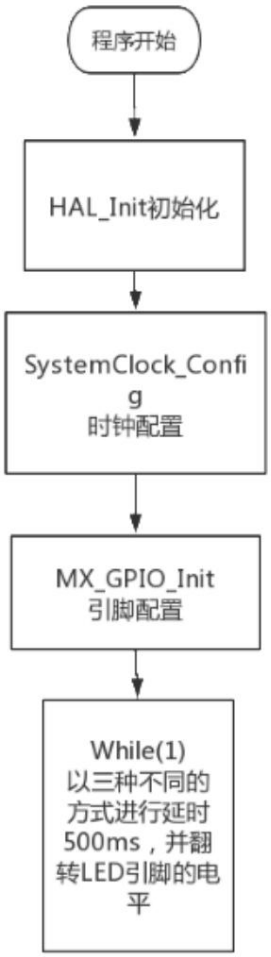
The program is initialized by HAL_Init, SystemClock is initialized, and GPIO is initialized, and then it enters the main loop.

Use the three methods described before to delay respectively, and then flip the level of the LED, so that the LED will follow the

On and off at a fixed frequency of 500 milliseconds, the main loop code is as follows.

```
bsp_led_toggle();

//nop delay
nop_delay_ms(500);

bsp_led_toggle();

//cycly count delay
user_delay_ms(500);

bsp_led_toggle();

//systick delay
HAL_Delay(500);
```


The program flow chart is as follows:



2.4.7 Effect display

The LED light on the C-type development board will jump once in about 500ms, showing a flickering effect. As shown in the figure, the open Hair board Type C LED on and off effect.

LED lighting diagram

LED off graph

2.5 Course Summary

Delay operation is also the basic operation in stm32. The three delay methods introduced in this lesson have their own characteristics.

The provided HAL_Delay function is the most convenient, but it cannot complete the microsecond-level delay.

Time or count delay method to implement.

3. Timer blinking LED

3.1 Knowledge points

ÿ Basic functions of timer

ÿ Timer configuration

ÿ Timer interrupt and interrupt priority explanation

ÿ Interrupt management in cubeMX

3.2 Course content

This lesson will learn about the basic functions of timers and how to configure them, and also touch on one of the most important concepts in stm32 - medium

Interrupts, introduce how to set interrupts in cubeMX, how to enable interrupts and configure interrupt priorities, etc., and finally

The timer interrupt triggered by the timer will be implemented to control the blinking of the LED light.

3.3 Basic Learning

3.3.1 Timer explanation

The basic function of the timer is the timing function, just like an alarm, after setting the corresponding time, it will sound at the set time.

ring. For example, the tick timer in the previous chapter, set to 1ms timing time, will cause an interrupt function every 1ms.

When using a timer, three very important concepts are involved - frequency division, counting, and reloading. These three concepts can be combined

It is understood in conjunction with the clocks used in life.

ÿ Frequency division: Different hands on the clock need to have different speeds, that is, different frequencies, so as to accurately represent the time,

For example, the second hand, the minute hand, and the hour hand, the ratio of the adjacent frequencies of these three is 60:1, that is, the minute hand rotates every 60 divisions of the second hand.

Move 1 division, the minute hand rotates 60 divisions and the hour hand rotates 1 division, so the division of the minute hand to the second hand is 60.

ÿ Counting: The value corresponding to the clock is proportional to the working time. For example, if the second hand rotates 10 divisions, it means that 10

Second, the count in the timer is also a value proportional to the count time. The higher the frequency, the faster the growth rate.

ÿ Overload: The scales of hours, minutes and seconds have upper limits. A dial can record up to 12 hours, 60 minutes, 60 seconds,

If it continues to increase, it will return to 0. The same also needs to be reloaded in the timer, when the count value in the timer reaches

When the reload value is reached, the count value is cleared.

Three important registers related to timers are now introduced.

ÿ Prescaler register TIMx_PSC

ÿ Counter register TIMx_CNT

ÿ Automatic reload register TIMx_ARR

The clock signal at the clock source passes through the prescaler register and is divided according to the internal value of the prescaler register. such as clock source
The frequency is 16MHz, and the value set in the prescaler register is 16:1, then enter the timer's
The clock frequency drops to 1MHz.

Driven by the divided timer clock, TIMx_CNT counts up according to the frequency of the clock until
When the value of TIMx_CNT increases to be equal to the set automatic reload register TIMx_ARR, TIMx_CNT is cleared,
And start counting up from 0 again, TIMx_CNT grows to the value in TIMx_ARR and is cleared to generate a timing
Interrupt trigger signal. In summary, the time when the timer triggers the interrupt is determined by the frequency division ratio in the set TIMx_PSC and the
Determined jointly with the auto-reload value in TIMx_ARR.

The timer is a very important peripheral in stm32. In most application scenarios, some tasks need to be executed periodically.
For example, the LED flashing mentioned in the previous lecture, this function can be realized by the timer, in addition, the timer of stm32
It can also provide PWM output, input capture, output comparison and other functions.

3.3.2 Interrupt explanation

Robots on the RM field often need to process a variety of signal inputs, such as remote control signals, signals sent by visual PCs,
Various sensor signals, etc., in addition to the need to output a variety of signals, such as the CAN signal to control the motor, control
The PWM of the steering gear, etc., so how does the STM32 arrange these tasks in an orderly manner? is dependent on the pre-interrupt composition
background mechanism.

In STM32, the processing of signals can be divided into polling mode and interrupt mode. The polling mode is to continuously access a
The port of the signal, to see if there is a signal entering, if there is, it will be processed, and the interrupt method will be generated when the input is generated.
A trigger signal tells STM32 that an input signal is coming in and needs to be processed.

For example, boiling water is boiling in the kitchen, and the owner is watching TV in the living room. In order to prevent the boiling water from drying out, he has two ways, the first is
Go to the kitchen every 10 minutes to take a look, the other is to wait for the kettle to boil and start to make a noise before dealing with it. The former is
The polling method, the latter is the interrupt method.

Each interrupt has a corresponding interrupt function. When an interrupt occurs, the program will automatically jump to the processing function to run, instead of
It needs to be called manually.

As in the first section, when the count value of the timer increases to the reload value, while clearing the count value, the timer will be triggered once
Interrupt, that is, the timer update interrupt. As long as the reload value of the timer is set, the timer interrupt can be guaranteed at a fixed frequency.
rate is triggered. In the RM competition, tasks that need to be executed regularly such as chassis control tasks and PTZ control tasks can be placed.
Executed in the timer update interrupt.

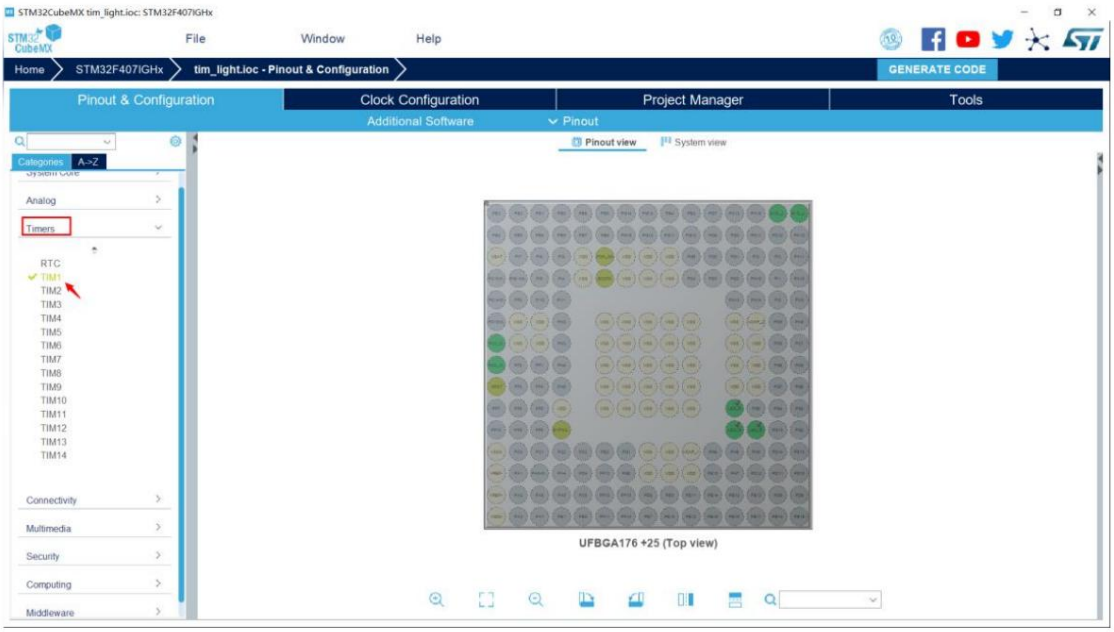
3.4 Program Learning

3.4.1 The timer is configured in cubeMX

The GPIO configuration for driving the LEDs in this chapter is the same as the previous two chapters, except that the timer needs to be set in cubeMX. operate as

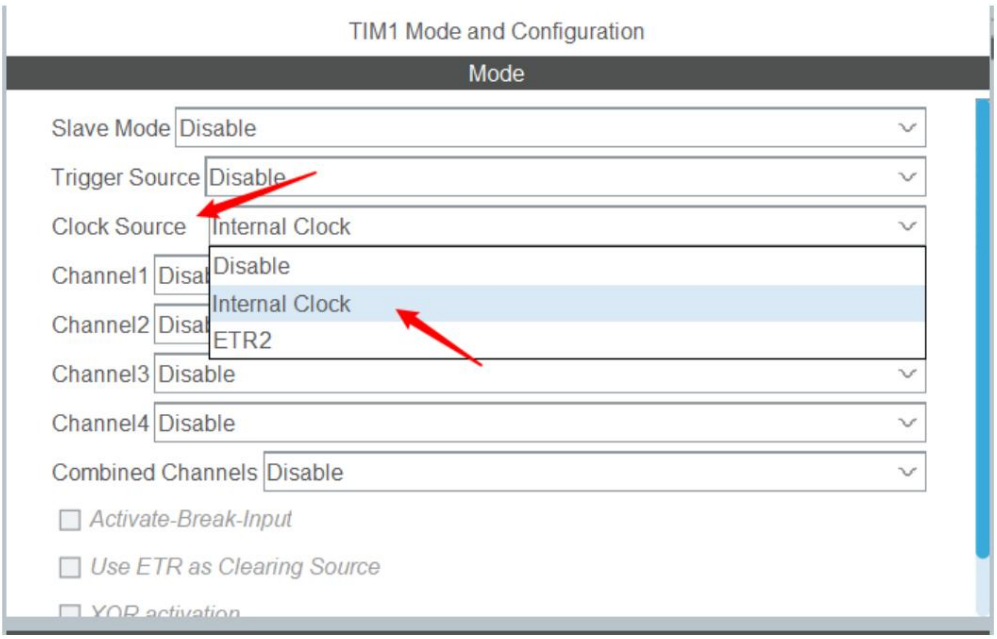
Down:

1. Select Timer in the left tab, and click TIM1 under the tab.



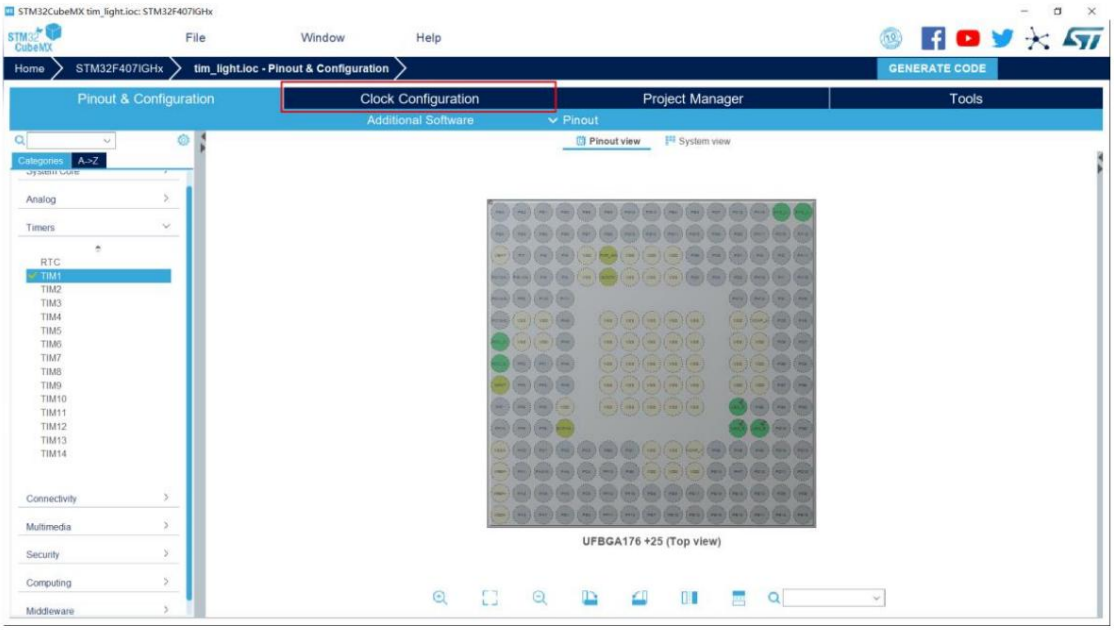
2. In the pop-up TIM1 Mode and Configuration, select it from the drop-down menu on the right side of ClockSouce

Internal Clockÿ



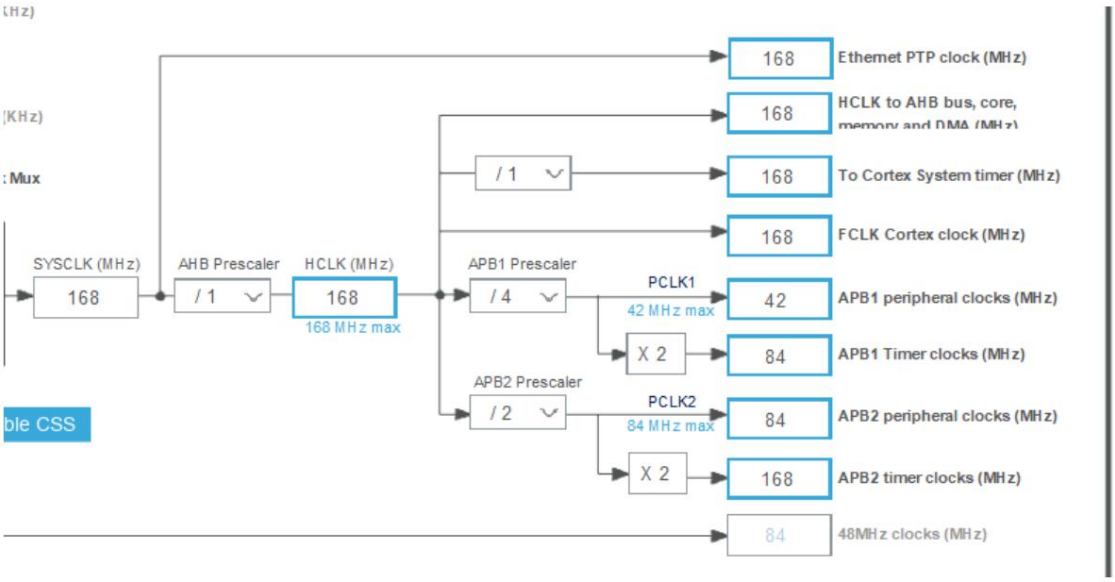
3. At this point, TIM1 is enabled, and then you need to configure the operating cycle of TIM1. Need to open Clock Configuration

Bookmark page.

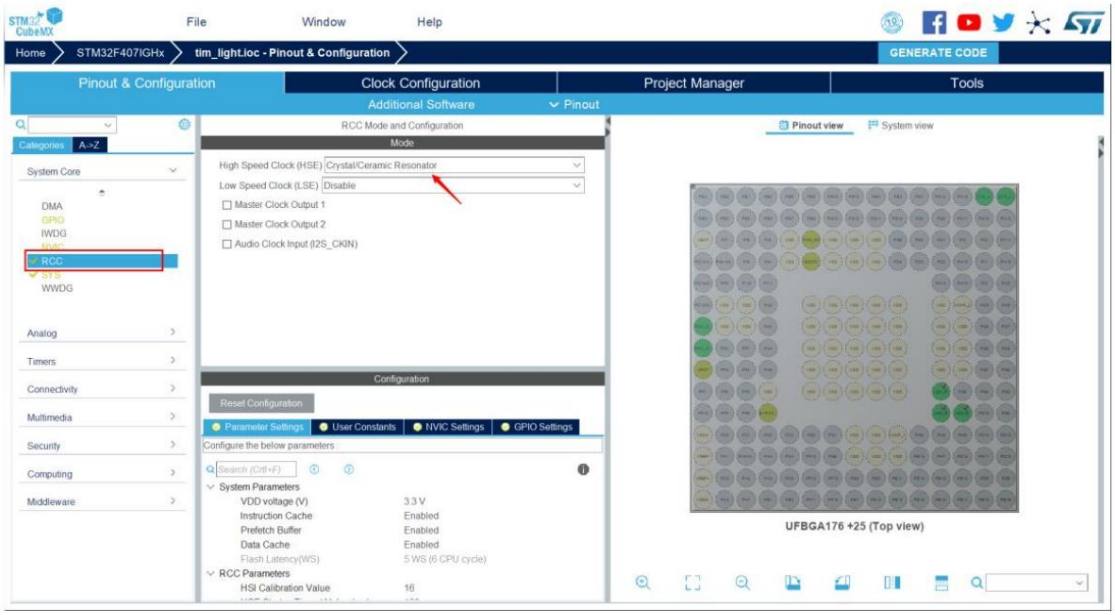


The following figure shows the clock tree structure of STM32. By configuring the frequency division/multiplier ratio everywhere in this structure, the final output can be controlled

clock to each peripheral.



If the HSE is gray, go to the Pinout&Configuration tab first to determine whether to enable the external crystal oscillator.



By consulting the data sheet, you can know that the clock source of TIM1 comes from the APB2 bus.

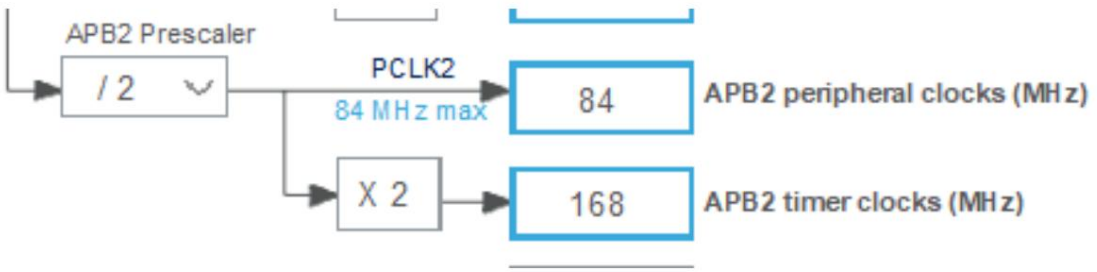
_HAL_RCC_TIM1_CLK_ENABLE is found in the definition, you can see APB2 in the code, indicating that TIM1 is hanging

Loaded on APB2.

```
#define __HAL_RCC_TIM1_CLK_ENABLE()             do { \
                                                  __IO uint32_t tmpreg = 0x00U; \
                                                  SET_BIT(RCC->APB2ENR, RCC_APB2ENR_TIM1EN);\
                                                  /* Delay after an RCC peripheral clock enabling */ \
                                                  tmpreg = READ_BIT(RCC->APB2ENR,
RCC_APB2ENR_TIM1EN);\
                                                  UNUSED(tmpreg); \
                                                    } while(0U)
```

Note that the APB2 Timer clocks (MHz) under the clock tree configuration page is 168MHz, which means that the

The frequency of the TIM1 prescaler register is 168MHz.



Next, control the period of the timer by setting the frequency division ratio and reload value. For detailed calculation steps, you can view the advanced learning section.

point.

If you want to get a timer with a period of 500 milliseconds, you can use the formula introduced in advanced learning to divide the frequency division value and the repetition rate.

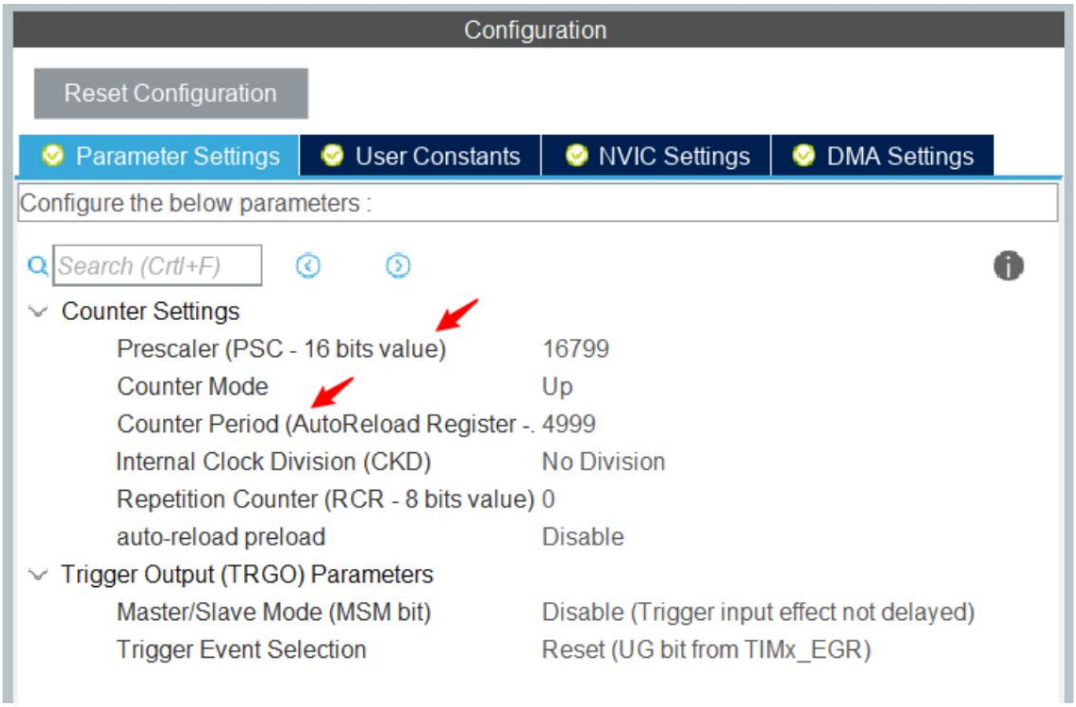
load value is set. Go back to the Pinout&Configuration tab, corresponding to the Prescaler of the TIMx_PSC register

entry and the corresponding Counter Period entry in the TIMx_ARR register. The frequency corresponding to 500ms is 2Hz, in order to get

For a frequency of 2Hz, the frequency division value can be set to 16799, and the reload value can be set to 4999, then the timer trigger frequency can be calculated.

for

$$= \frac{168000000}{(16799 + 1) \text{ ÿ } (4999 + 1)} = 2$$



3.4.2 Interrupt priority explanation

Recalling the concept of interrupts mentioned earlier, the controller dedicated to processing interrupts in STM32 is called NVIC, which is nested vector

Interrupt Controller (Nested Vectored Interrupt Controller).

NVIC is very powerful and supports interrupt priority and interrupt nesting functions. Interrupt priority is to assign different interrupts.

There are different response levels. If multiple interrupts are generated at the same time, the STM32 will give priority to the high-priority interrupts.

Interrupt nesting means that when an interrupt is processed, if an interrupt with a higher priority is generated, the current interrupt will be suspended and processed first.

The generated high-priority interrupt is processed and then restored to the original interrupt to continue processing.

This process is understood as in the situation above, where the owner heard the water boiling and was about to go to the kitchen when he heard the door

There was a quick knock on the door, and the owner would first perform the door opening operation, and then go to the kitchen to process the boiling water.

In order to achieve richer interrupt priority in the limited number of register bits, the NVIC uses an interrupt grouping mechanism.

STM32 will first group the interrupts, and then divide the priority into preemption priority (Prem priority) and response priority.

Subpriority, the number of preemption priority and response priority can all be passed through the AIRCR register in the NVIC.

The PRIGROUP[8:10] bits are configured to specify the division of NVIC_IPRx[7:4] by two priorities, according to the division

Determine the number of two priorities. A total of 5 cases can be divided into the following table

	Preemption priority number	Response priority number
000	0	0-15
001	0-1	0-7
010	0-3	0-3
011	0-7	0-1
100	0-15	0

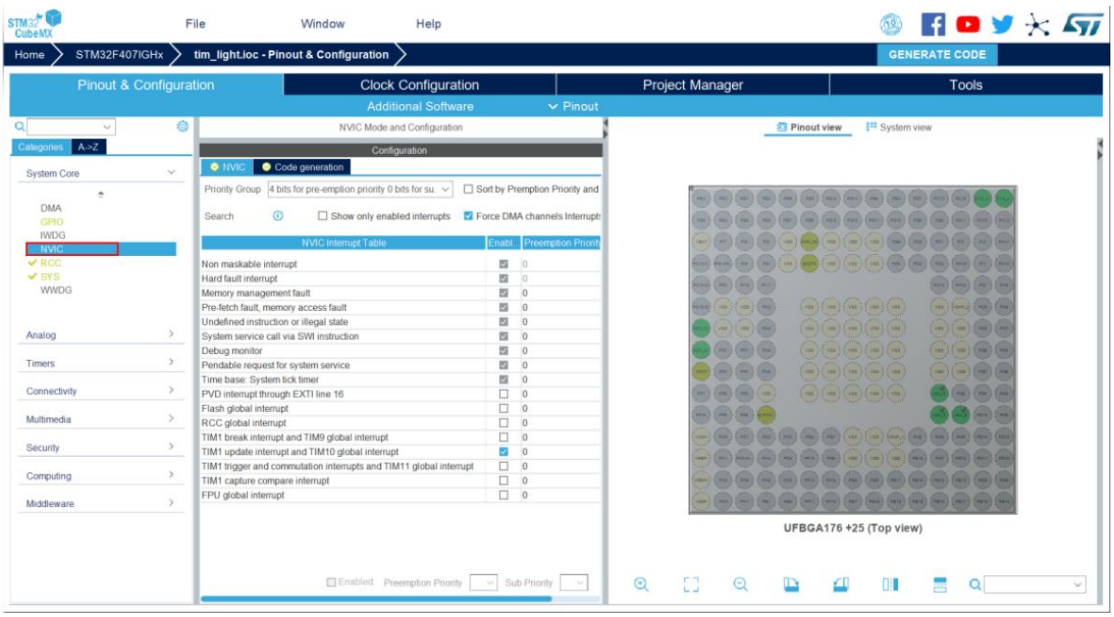
Interrupts with the same preemption priority are under the same interrupt group.

When multiple interrupts occur, first determine which interrupt group can respond first according to the preemption priority, and then go to this interrupt group.

According to the response priority of each interrupt, it is judged which interrupt has priority to respond.

3.4.3 Interrupt configuration and interrupt function management in cubeMX

You can see the interrupt configuration in the current system under the NVIC tab of cubeMX

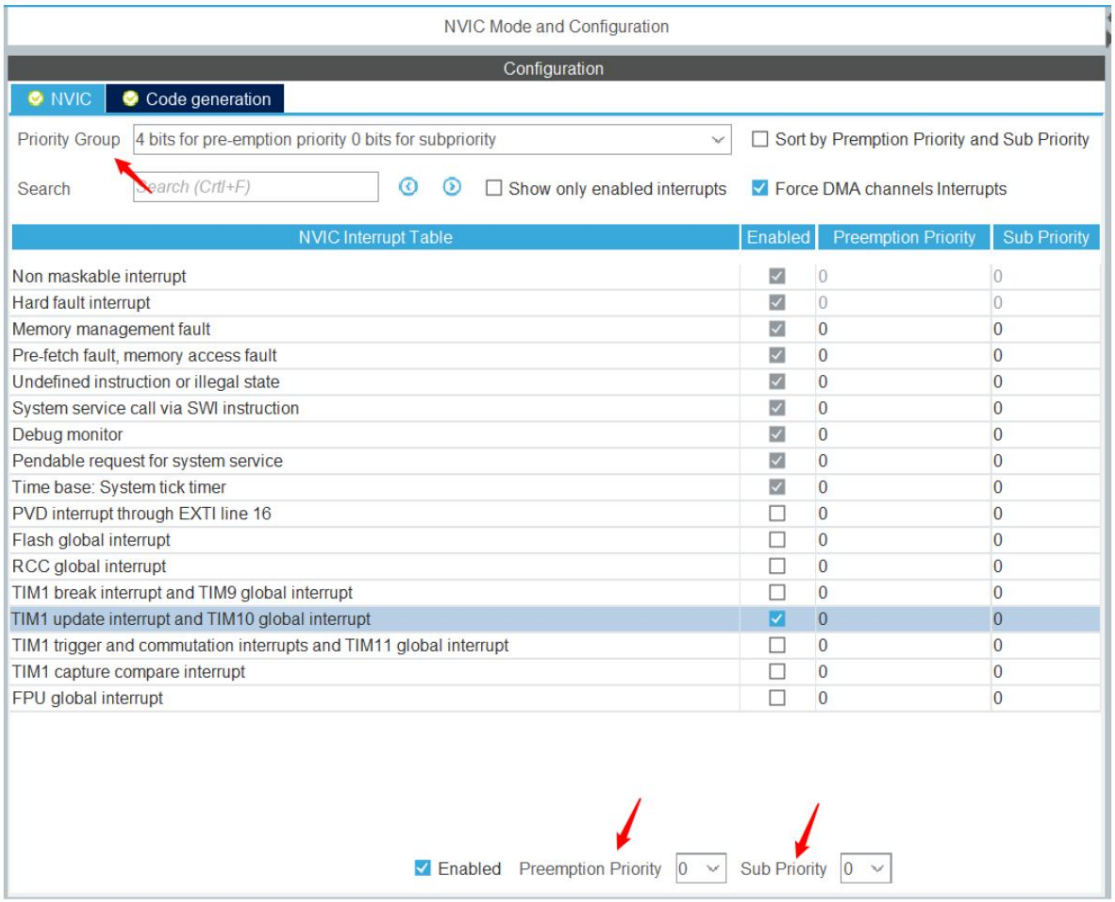


The list shows the enable conditions and priority settings of all interrupts in the current system. To enable the interrupt, in the Enable column

Tick, select TIM1 update interrupt here, tick to enable the interrupt. In addition, you can also grab the

Assignment of priority and response priority and configuration of two priority levels for interrupts. Here is the timer 1 interrupt keep silent

The default 0,0 priority.



Click Generate code to generate the code.

Let's take a look at how the HAL library handles interrupts.

In stm32f4xx_it.c, find the interrupt handler automatically generated by cubeMX

```
void TIM1_UP_TIM10_IRQHandler(void)
{
  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */

  /* USER CODE END TIM1_UP_TIM10_IRQn 0 */
  HAL_TIM_IRQHandler(&htim1);
  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 1 */

  /* USER CODE END TIM1_UP_TIM10_IRQn 1 */
```



```
}
```

This function calls the HAL_TIM_IRQHandler function provided by the HAL library

```
void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)
```

Function name	HAL_TIM_IRQHandler
function	HAL processes registers involved in interrupts
return value	void
parameter	*htim Timer handle pointer, if timer 1 is lost Input &htim1, Timer 2 input &htim2

After HAL_TIM_IRQHandler processes each register involved in the interrupt, the interrupt callback is automatically called

The function HAL_TIM_PeriodElapsedCallback, which is modified with the __weak modifier, that is, the user can

Redeclare the function at the place where the call will take precedence over the user-declared function.

```
__weak void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
```

Generally, we need to determine the source of the interrupt in the interrupt callback function and perform the corresponding user operation.

3.4.4 Timer callback function introduction

As mentioned above, the HAL library will automatically call the timer callback function after completing the timer's interrupt service function.

By configuring the frequency division value and reload value of TIM1, the interrupt of TIM1 is triggered with a period of 500ms. therefore interrupted

The callback function is also called every 500ms. Redeclare the timer callback function in main.c and write the content

as follows:

```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)
{
    if (htim == & htim1)
    {
        //500ms trigger
        bsp_led_toggle();
    }
}
```

It can be seen that the source of the interrupt is first judged in the callback function to determine whether the source is timer 1. if there is

His timer generates an interrupt, and the timer callback function will also be called, so it is necessary to judge the source.

After confirming that the interrupt source is timer 1, call the bsp_led_toggle function to flip the output of the RGB three-color LED pin.

output level.

3.4.5 HAL_TIM_Base_Start function

If the interrupt is not turned on, only the timer is allowed to work with the timing function. In order to make the timer work, the HAL library needs to be called.

provided function.

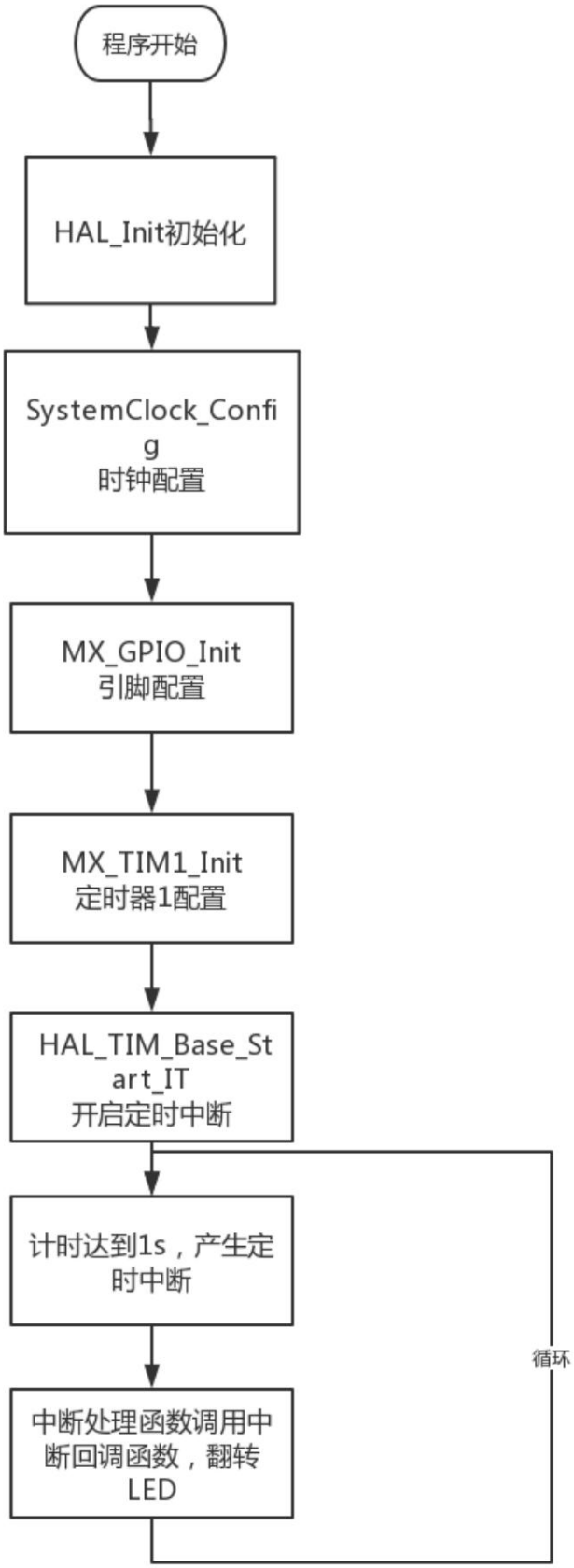
HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)	
Function name	HAL_TIM_Base_Start
function	Make the corresponding timer start working
return value	HAL_StatusTypeDef, several states defined by the HAL library, Returns HAL_OK if the timer starts working successfully
parameter	*htim Timer handle pointer, such as timer 1, enter &htim1, for timer 2, enter &htim2

If you need to use timed interrupts, you need to call the function

HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)	
Function name	HAL_TIM_Base_Start_IT
function	Start the corresponding timer and enable its timing interrupt
return value	HAL_StatusTypeDef, several states defined by the HAL library, Returns HAL_OK if the timer starts working successfully
parameter	*htim Timer handle pointer, such as timer 1, enter &htim1, for timer 2, enter &htim2

If the above two functions are to be used, they need to be called before the main loop while(1).

3.4.6 Program Flow



3.4.7 Effect display

The timer will trigger an interrupt at 500ms timing, so that the LED light on the development board C type will jump once every 500ms, showing

flickering effect. Use the timer to trigger the interrupt, the time accuracy depends on the accuracy of the crystal oscillator, which is greater than the software analog delay accuracy

Big boost. As shown in the figure, the lighting and extinguishing effects of the C-type LED of the development board are respectively.

LED lighting diagram

LED off graph

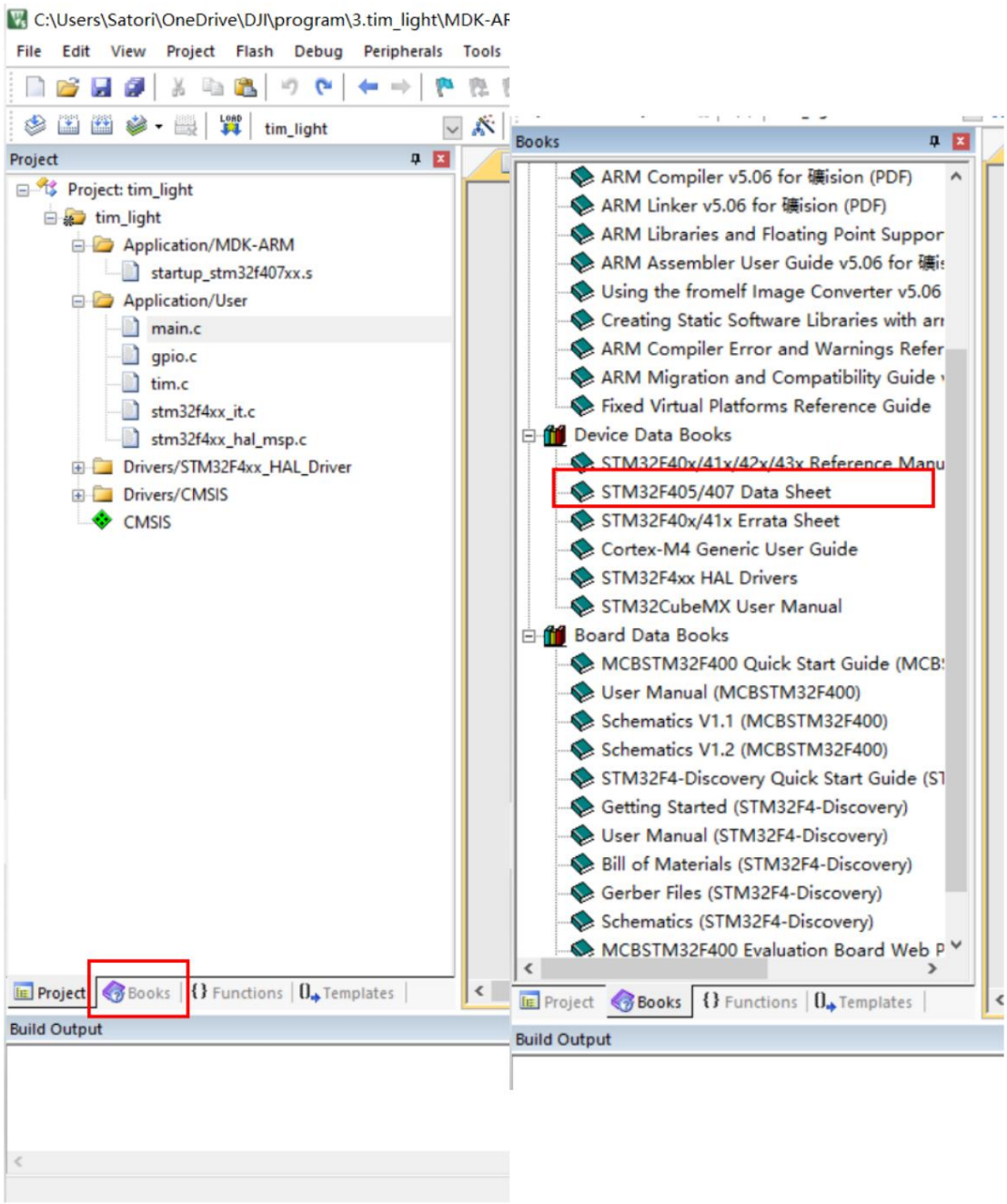
3.5 Advanced Learning

3.5.1 APB bus calculation timer timing time

As mentioned in the previous section, the timing period of the timer can be controlled by setting the divider ratio and the automatic reload value. In this section

Combined with the chip manual (Datasheet) to introduce the calculation process of the timing cycle in detail. The chip manual can be found in Keil's

Get it under the book tab. The STM32F407IGHx chip is used as an example here, and the same is true for other chips.

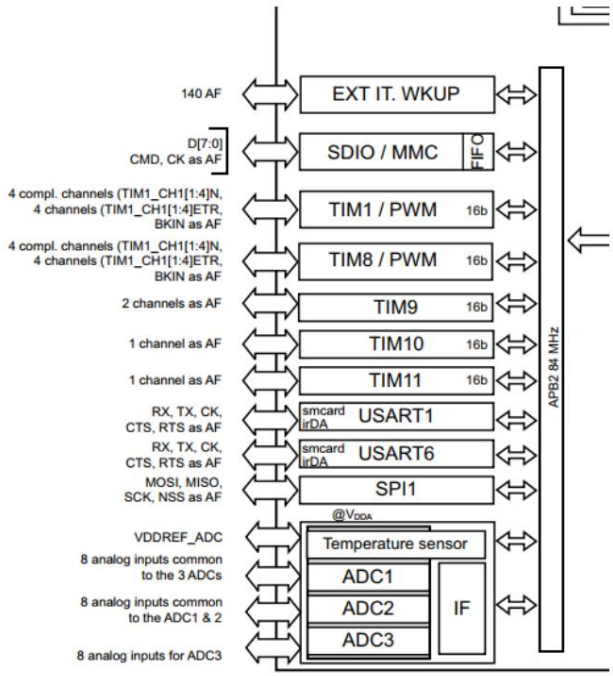


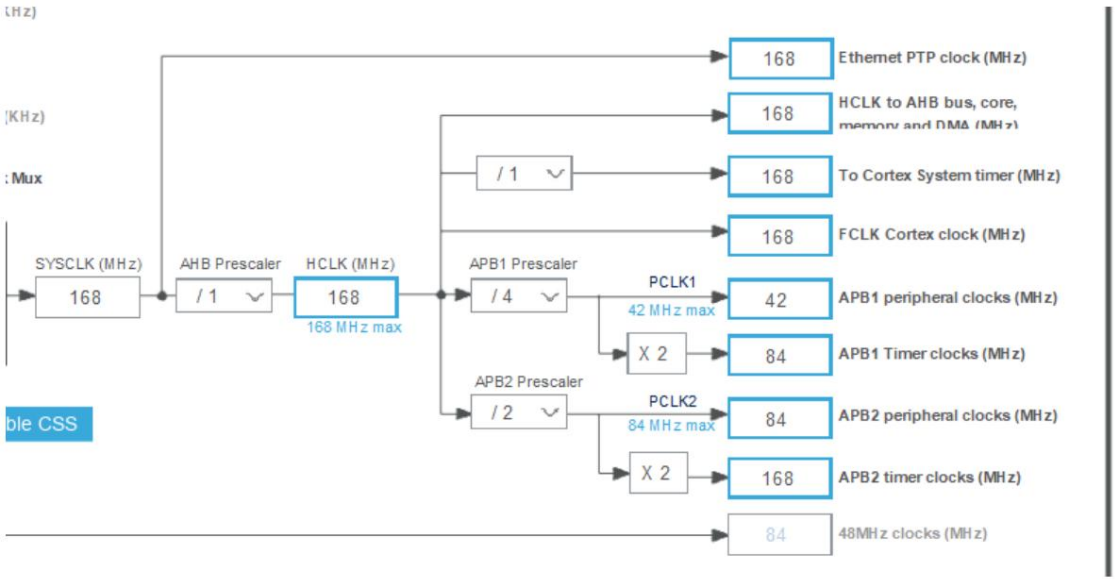
On page 19 of this document, which is the Device overview chapter, you can see that the clock inside the entire stm32 is as follows

How to connect to various peripherals, in the figure you can see a large number of peripherals mounted on the APB1 and APB2 buses. Certainly

Timers 1, 8, 9, 10, 11 are mounted on the APB2 bus. According to the total number of timers mounted in the datasheet

line to determine the clock source frequency of the timer before dividing.





Under the clock configuration tab of cubeMX, you can see the structure of the clock tree. at the far right of the entire clock tree

At the end, you can see the clock frequency settings of the two buses APB1 and APB2, where APBx peripheral clocks are hung.

Peripherals other than the timers mounted on the bus provide clock sources, APBx timer clocks are the timers mounted on the bus

Provide the clock source.

After the clock source frequency is determined, according to the content in page 640 of the data sheet as shown in the figure, the frequency division value is

Divider value +1. That is, when TIMx_SPC is 0, the frequency division ratio is exactly 1:1. If TIMx_SPC is 15, the frequency division ratio is

16:1, the incoming 16MHz frequency signal will be divided into 1MHz.

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).

The frequency after frequency division is the self-incrementing frequency of TIMx_CNT. According to the content introduced before, when the value of TIMx_CNT increases After reaching the value in TIMx_ARR, the reload occurs and the interrupt signal is triggered, which is equivalent to using the value in TIMx_ARR Another frequency division was performed. Therefore, the frequency of generating this interrupt signal should be (need to add 1 because CNT is opened from 0 counted).

= /([15: 0] + 1)

Combining the above two formulas, the final formula for calculating the trigger frequency of the timer can be obtained

= /(([15: 0] + 1) ÿ ([15: 0] + 1))

The value of f_CKPSK is confirmed by finding the corresponding APB bus through the method mentioned above.

3.6 Course Summary

In this lesson, I learned about the timer peripherals of stm32, and learned how to make the timer follow the period by configuring several important parameters. It runs at the desired cycle, and the timer can be used to realize an important background mechanism in the embedded system, that is, to execute according to a stable cycle. Scheduled tasks. In addition, this lesson also learns the important concept of interruption, by opening a variety of interruptions, and rationally formulating their prioritization, in order to be able to run rich tasks on the robot in real time.

4. PWM control LED brightness

4.1 Knowledge points

ÿ PWM basic knowledge

ÿ aRGB three primary color synthesis lighting effect explanation

ÿ Configure PWM timer configuration in cubeMX

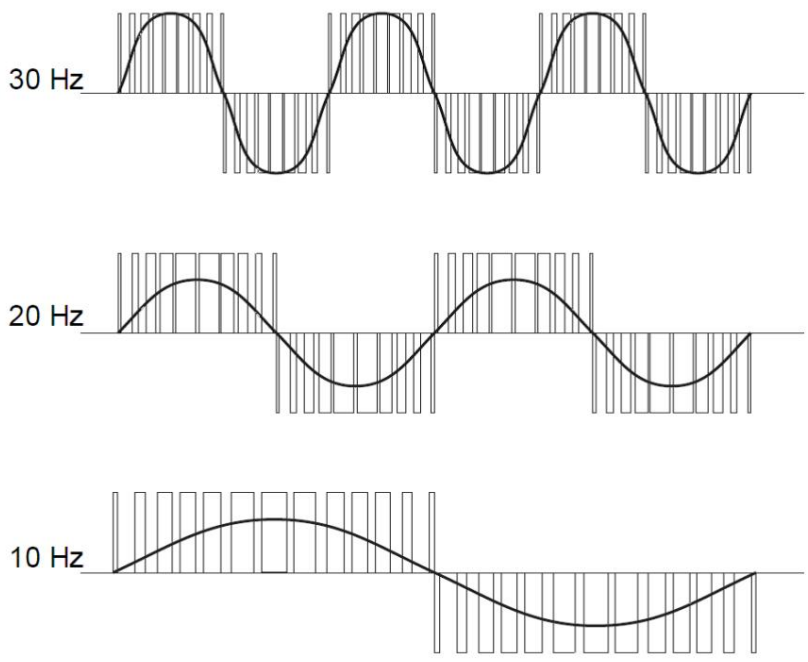
4.2 Course content

In this course, you will learn the basic principle of PWM, how to control the pin output PWM signal, aRGB three primary color synthesis The principle of lighting effect. This lesson will do the pin configuration in cubeMX, build the project, and use the PWM output to achieve aRGB composite equivalent.

4.3 Basic Learning

4.3.1 Basic knowledge of PWM

PWM is the abbreviation of "Pulse Width Modulation" in English, referred to as pulse width debugging. using micro A very efficient technique for controlling analog circuits using the digital output of a processor. Widely used in measurement, communication into many fields of power control and conversion.



For example, in the picture above, the rectangular pulse is the digital signal output by stm32. When this signal is connected to the peripheral, the effect can wait

The effect is this sine wave.

The ratio of the duration of the high level in one cycle to the total cycle becomes the duty cycle. By modifying the duty cycle, the output can be changed.

equivalent analog voltage. For example, if the output duty cycle is 50%, the pulse frequency is 10Hz, and the high level is 3.3V. Then its output

The analog effect is equivalent to outputting a 1.65V high level. In addition, the frequency of the PWM output will also affect the final PWM

Output effect, the higher the frequency of the PWM output, the better the "continuity" of the final output, the closer to the effect of the analog signal,

Low frequency will enhance the discreteness, and the final output effect will have a strong sense of "mutation".

Pulse modulation has two important parameters. The first one is the output frequency. The higher the frequency, the better the simulation effect. the second

is the duty cycle. The duty cycle is the magnitude of the voltage that changes the output analog effect. The larger the duty cycle, the larger the simulated voltage.

4.3.2 aRGB three primary colors

aRGB is a color mode, aRGB represents alpha (transparency) Red (red) Green (green) and

Blue (blue) has four elements. Generally, we set the value range of 0-255 in decimal for each element.

The standard representation is 0x00-0xFF, so an aRGB value can be described by an eight-digit hexadecimal number, from front to back each

The two bits correspond to a, R, G, B in turn.

In aRGB, the larger the alpha value, the more opaque the color is, and the larger the value in RGB, the stronger the corresponding color. for example

Pure red can be expressed as 0xFFFF0000 in 8-bit hexadecimal, pure green can be expressed as 0xFF00FF00, pure blue

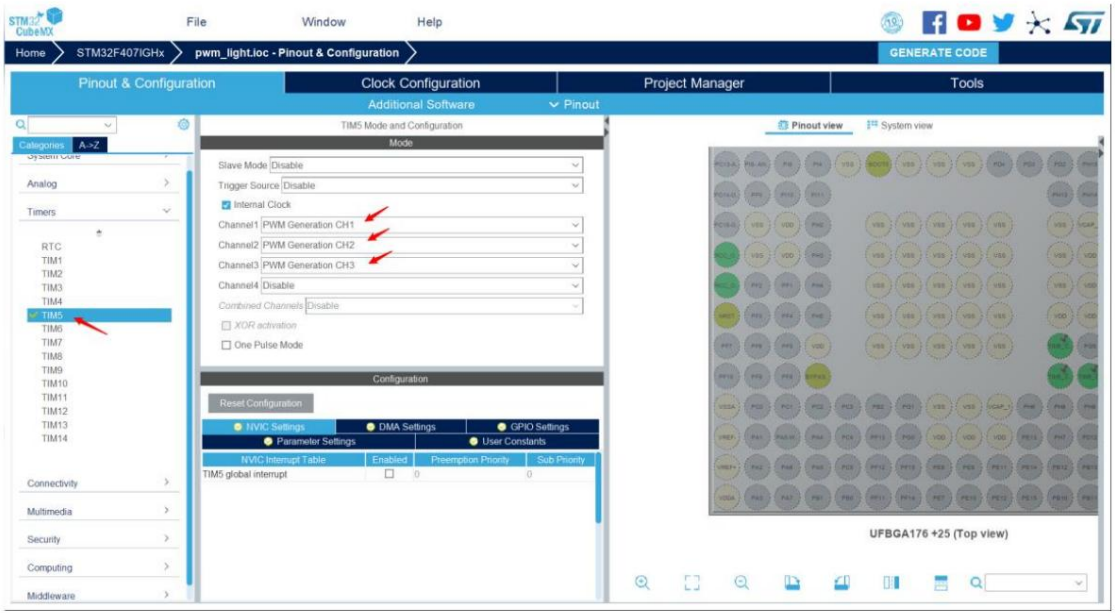
It can be represented as 0xFF0000FF, and yellow is composed of blue and green, so it can be represented as 0xFF00FFFF.

4.4 Program Learning

4.4.1 PWM configuration in cubeMX

Set the channels 1, 2, and 3 of timer 5 as PWM outputs in cubeMX. It can be noticed that the corresponding leads of the three channels

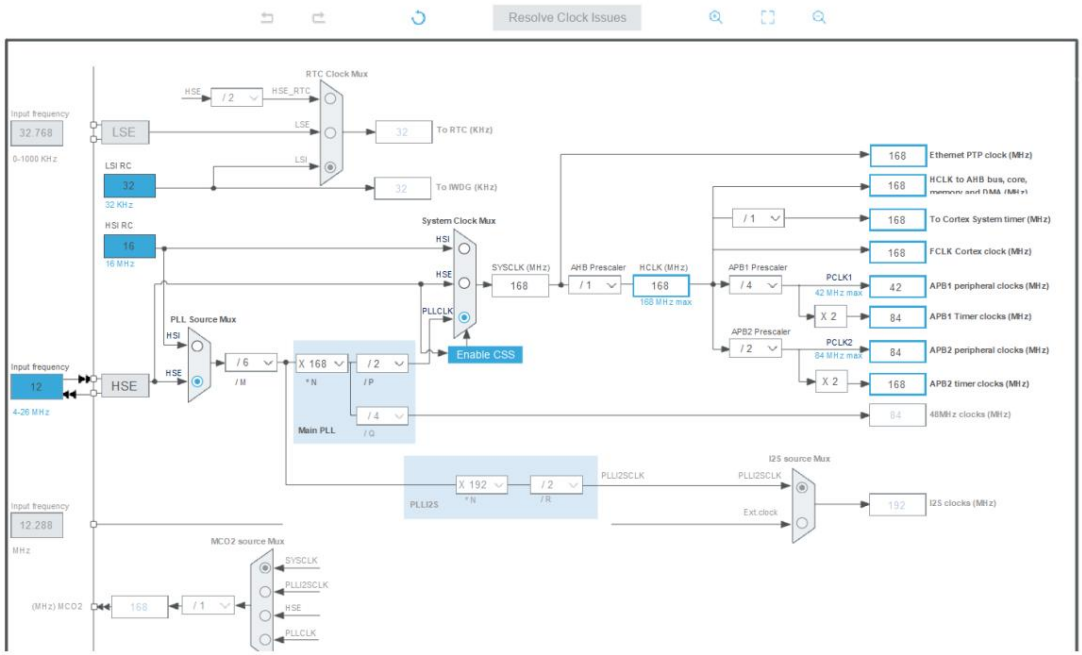
The pins are exactly the LED pins used in the previous experiments.



Timer 5 is configured as follows, and the reload value is set to 65535.

Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register..	65535
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (32 bits value)	10000
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 2	
Mode	PWM mode 1
Pulse (32 bits value)	10000
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 3	
Mode	PWM mode 1
Pulse (32 bits value)	10000
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

The clock tree configuration is as follows:



Click Generate Code to generate project code.

4.4.2 Introduction to PWM Configuration

In the previous class, the timer of STM32 was introduced, and it was mentioned that the PWM output is one of the functions of the timer of STM32,

To implement the PWM function, the compare register (TIMx_CCRx) in the timer needs to be used.

When the timer works in PWM mode, it will automatically compare the value of TIMx_CCRx with TIMx_CNT (count register)

Compared with the value of TIMx_CNT, when the value in TIMx_CNT is less than the value of TIMx_CCRx, the PWM output pin outputs a high level,

When it is greater than it will output a low level. Therefore, knowing the period and duty cycle of the PWM signal can be done by setting the compare register

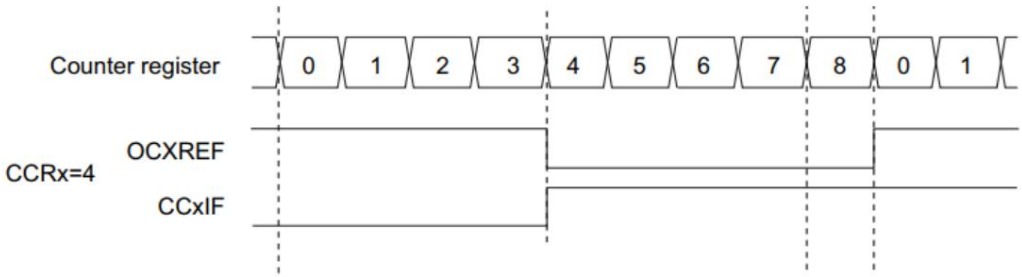
TIMx_CCRx and timer reload register TIMx_ARR to control. The duty cycle of PWM can be calculated by the following formula:

Calculate:

$$= \frac{\ddot{y}\ 1}{\ddot{y}}\ \ddot{y}\ 100\ddot{y}$$

The following figure is an example, the reload value of the timer is 8, the value of the compare register is 4, and the output signal is OCXREF, then it is occupied

ratio was 44.4%.



When a timer works in PWM output mode, there are 4 channels that can output PWM signals, each of which is set to

Each timer has a compare register corresponding to the label. For example, the compare register corresponding to the No. 1 channel of the No. 5 timer is:

TIM5_CCR1ÿ

Modify the value of the compare register TIMx_CCRx to control the duty cycle of the PWM output. In the function aRGB_led_show

, first extract the corresponding alpha, R, G and B channel values through the AND operation and shift operation, and then use the transparency

Alpha is multiplied by R, G, and B in turn, and finally assigns it to the corresponding value through the __HAL_TIM_SetCompare function.

The compare register TIM5->CCRx. By controlling different PWM duty cycles, control the brightness of a certain color LED,

In this way, the final output LED lighting effect can be controlled by setting the value of aRGB.

```
void aRGB_led_show(uint32_t aRGB)
{
    static uint8_t alpha;
    static uint16_t red,green,blue;

    alpha = (aRGB & 0xFF000000) >> 24;
    red = ((aRGB & 0x00FF0000) >> 16) * alpha;
    green = ((aRGB & 0x0000FF00) >> 8) * alpha;
    blue = ((aRGB & 0x000000FF) >> 0) * alpha;

    __HAL_TIM_SetCompare(&htim5, TIM_CHANNEL_1, blue);
    __HAL_TIM_SetCompare(&htim5, TIM_CHANNEL_2, green);
    __HAL_TIM_SetCompare(&htim5, TIM_CHANNEL_3, red);
}
#define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__) \
(((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1 = (__COMPARE__)) :\
 ((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2 = (__COMPARE__)) :\
 ((__CHANNEL__) == TIM_CHANNEL_3) ? ((__HANDLE__)->Instance->CCR3 = (__COMPARE__)) :\
 ((__HANDLE__)->Instance->CCR4 = (__COMPARE__)))
```

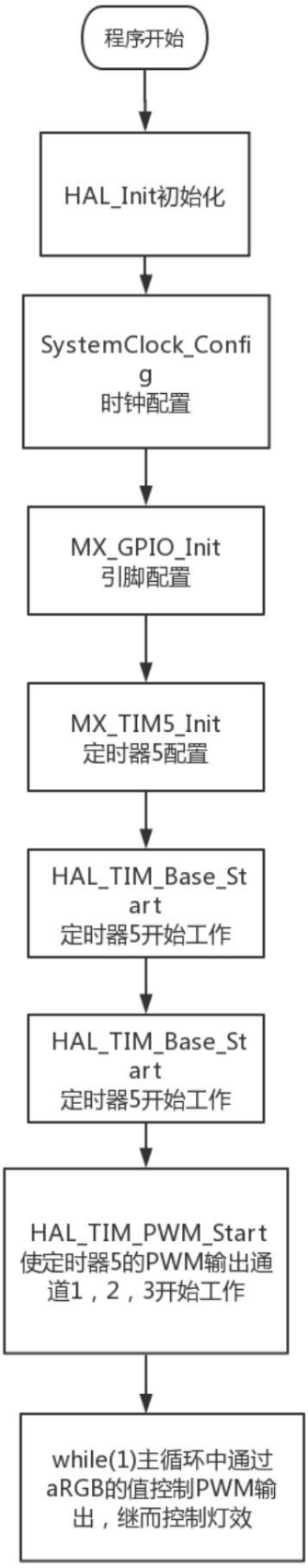
4.4.3 HAL_TIM_PWM_Start function introduction

In order to make the timer start PWM output, in addition to making the timer start working through HAL_TIM_Base_Start, it is also necessary to

The PWM initialization function HAL_TIM_PWM_Start provided by the HAL library needs to be called during initialization.

HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)	
Function name	HAL_TIM_PWM_Start
function	Make the corresponding channel of the corresponding timer start PWM output
return value	HAL_StatusTypeDef, several states defined by the HAL library, Returns HAL_OK if the timer starts working successfully
parameter 1	*htim Timer handle pointer, such as timer 1, enter &htim1, for timer 2, enter &htim2
parameter 2	Channel Timer PWM output channel, such as channel 1 for TIM_CHANNEL1

4.4.4 Program Flow



4.4.5 Effect display

Set the ARGB value to 0x7F123456, which is a half-brightness blue light effect, as shown in the figure.

PWM control LED lighting effect

4.5 Course Summary

The aRGB lighting effect is a common lighting effect, which can be used for signal indication, status display, error display and other purposes. this lesson

The conventional PWM output function can be used for steering gear and motor control. It is a very common control signal. It needs to be proficient in its original

reason.

Homework for this class: Use PWM to control the color cycle of LED lights on the development board, from red light to green light, and then

It turns into a blue light, and the switching process has a breathing effect.

5. Common **PWM** devices

5.1 Knowledge points

ÿ Knowledge of buzzer types

ÿ Buzzer tone

ÿ Servo control

5.2 Course content

This lesson will learn more about PWM controlled devices - buzzer, servo and snail motor, learn to use PWM function

Control the buzzer and servo. At the same time, we will learn the general structure and principle of active buzzer and passive buzzer, as well as the steering gear

basic knowledge.

5.3 Basic Learning

5.3.1 Buzzer

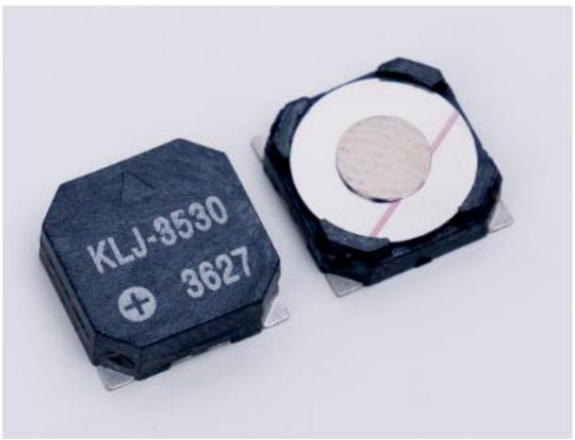
A buzzer is a sounding device that can be controlled by an electronic signal. In life, almost everything that can make a beeping sound

Buzzers are built into the electronics. The buzzer can provide users with intuitive sound information, which is a common human-computer interaction.

interactive mode. In RoboMaster competitions, it is often used to remind players to complete a certain inspection or for a problem with a module

time error. Common buzzers are divided into pin type and patch type. In the development board C type, the buzzer is the patch type, such as

as shown below.



According to whether the built-in oscillator circuit can be divided into active buzzer and passive buzzer. Active buzzer only needs to provide DC voltage

The oscillating current can be generated by the internal oscillating circuit to make a sound, while the passive buzzer needs to input a specific frequency.

Waves make sound. Compared with the two, the control of the active buzzer is simpler, but it can only emit a single frequency sound.

Although the passive buzzer is more troublesome to control, it can emit different tones by changing the frequency of the input square wave

The sound can even be used to play music.

The comparison between the two can be seen in the table below:

	Active buzzer	Passive buzzer
Built-in oscillator	have	without
Incentive	DC voltage	specific frequency square wave
tone	fixed	variable

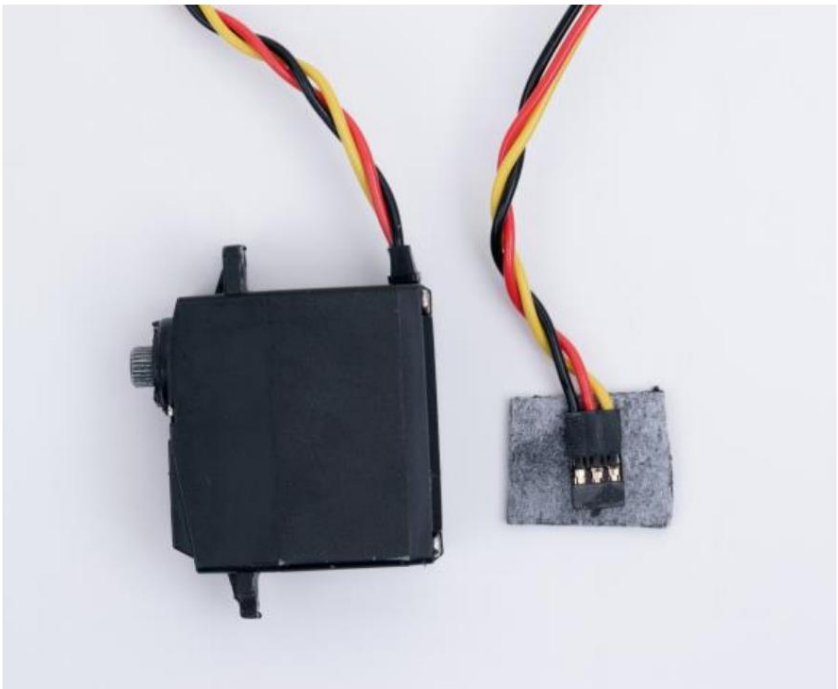
5.3.2 Control of the steering gear

The steering gear is a common executive component in the robot, which is usually controlled by PWM with a specific frequency.

The main components of the steering gear are composed of a small motor and transmission mechanism (gear set), which are mostly used to control the aircraft.

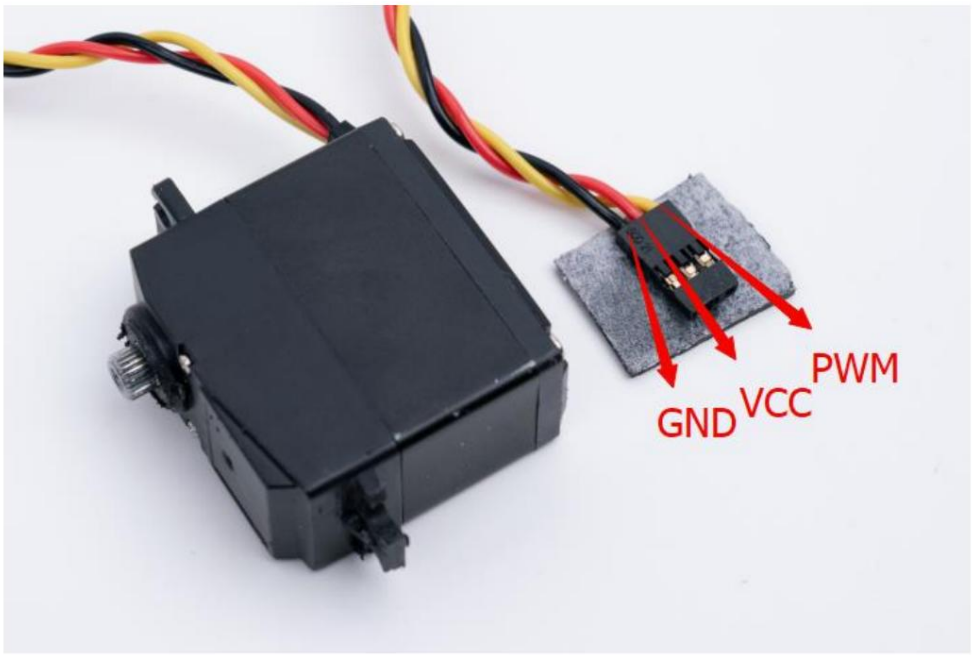
The rudder surface, hence the name steering gear. Due to the simple control and low price, in RoboMaster competition, it is used for simple motion

For example, use the servo to control the opening and closing of the magazine cover. The picture below shows the common servos on the market.



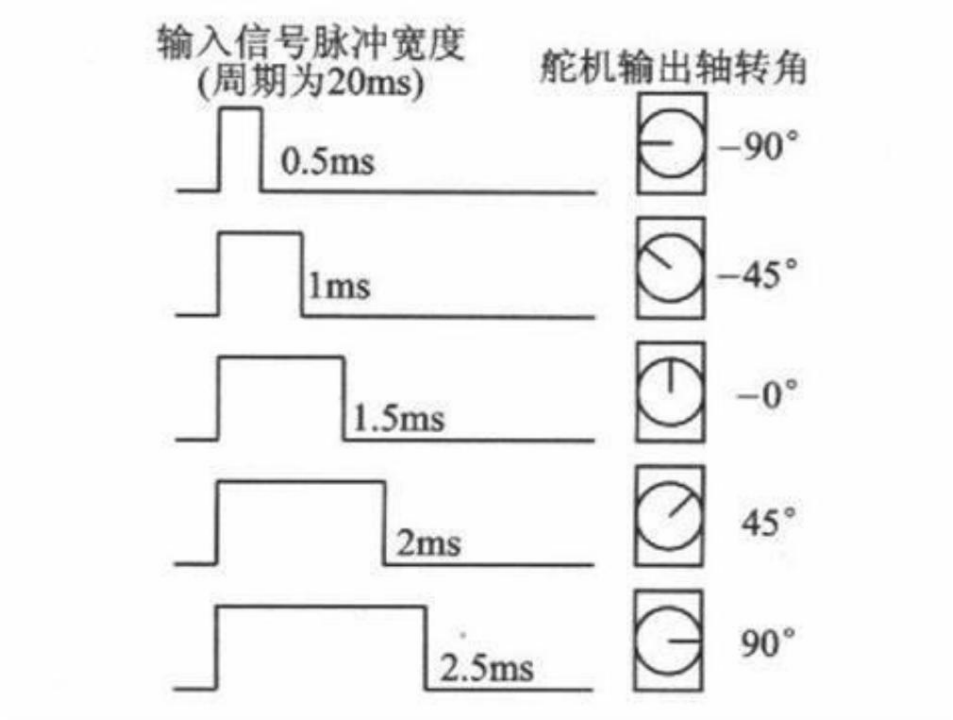
Usually, the three wires of the servo are respectively by color: black-GND, red-VCC, yellow-PWM signal. using the steering gear

, you only need to use a DuPont line or other connecting line to connect to the PWM interface corresponding to the C type of the development board.



The PWM signal used by the steering gear is generally a PWM signal with a frequency of 50Hz and a high level time of 0.5ms-2.5ms.

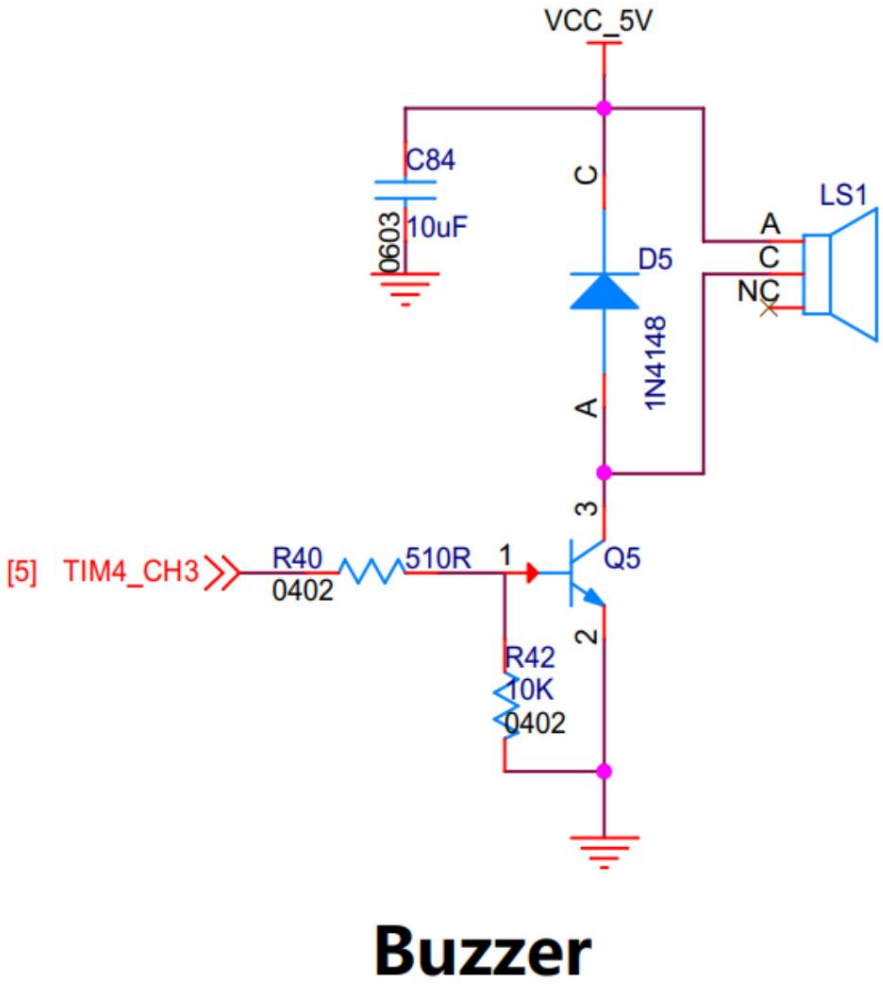
The PWM signal of the empty ratio corresponds to the rotation angle of the steering gear. Taking a 180-degree steering gear as an example, the corresponding angle diagram is shown in the figure below.



5.4 Program Learning

5.4.1 The PWM of the buzzer is configured in cubeMX

Check out the schematic of the development board, as shown.

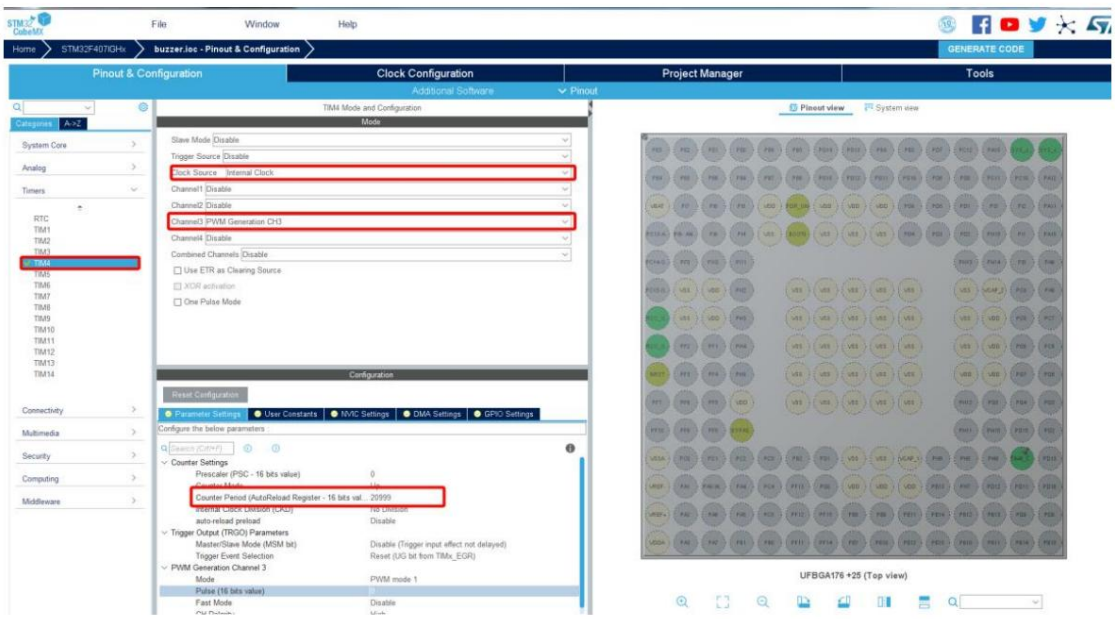


Buzzer

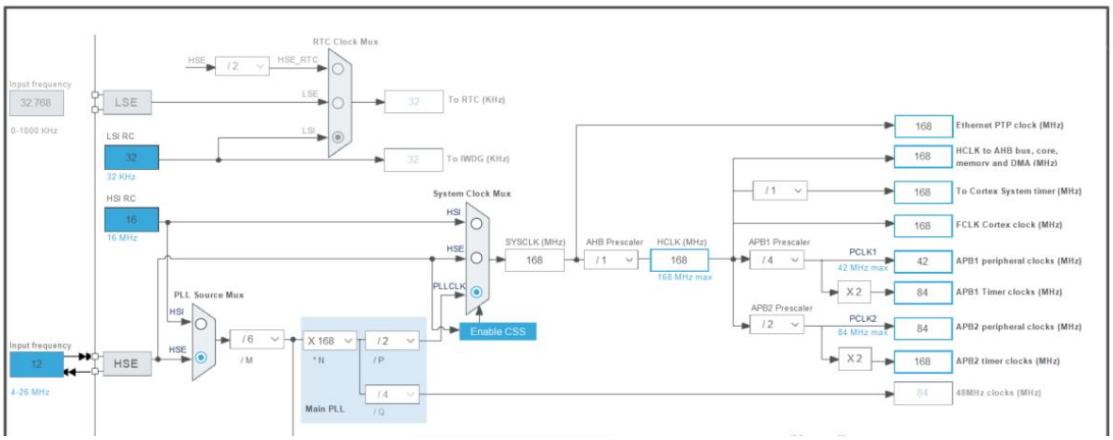
The pin used by the buzzer is PD14, which is channel 3 of timer 4.

Turn on cubeMX, enable timer 4, set preallocation to 0, set reload value to 20999, set channel 3 to PWM

Output, the rest of the settings can be kept as default, at this time the PD14 pin of the development board turns green.



The clock tree configuration is shown in the figure.



Click Generate Code to generate the project.

According to the knowledge in the advanced learning part of the timer chapter, you can know the timer by viewing the source code or data sheet.

Timer 4 is mounted on the APB1 bus, the corresponding bus frequency is 84MHz, the frequency division value is 0, and the reload value is 20999.

And through formula calculation, the output frequency of PWM wave is 4000Hz.

5.4.2 The program flow of the buzzer

In the previous principle introduction, changing the frequency of the PWM can change the tone of the passive buzzer. So change the timer's

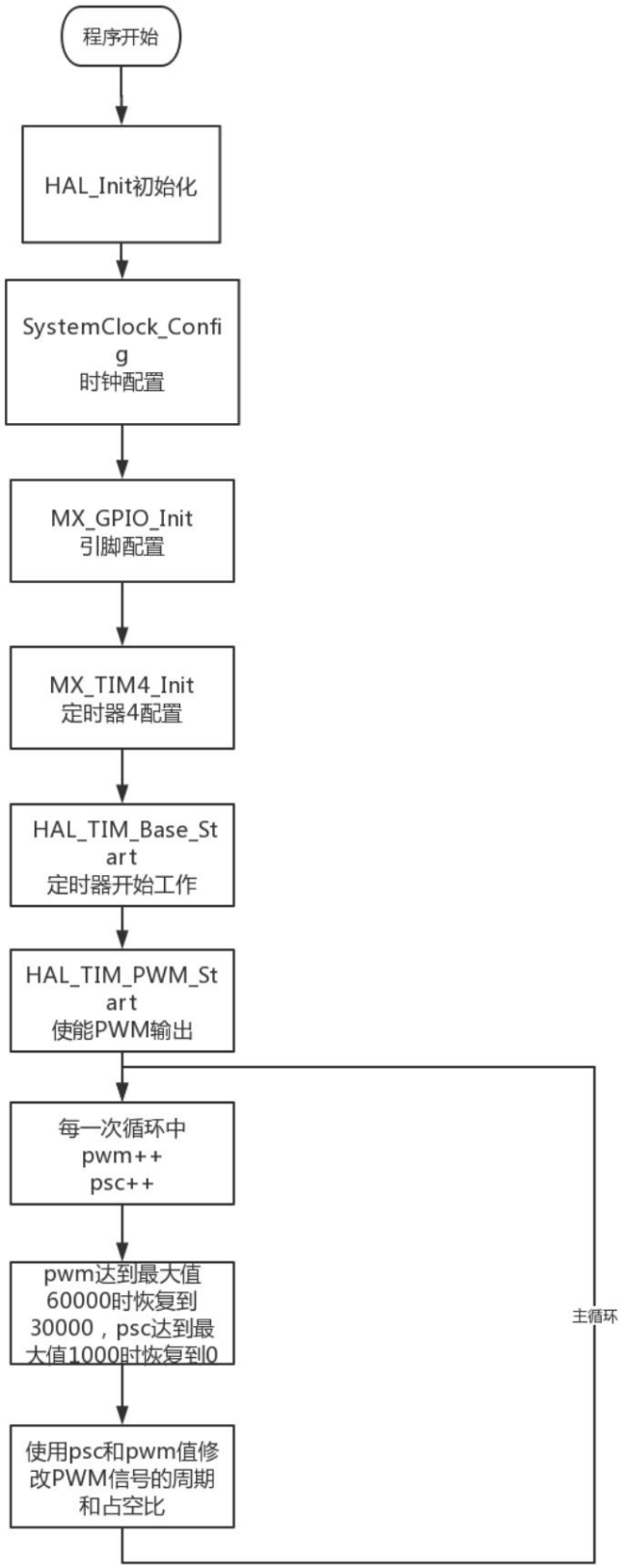
Frequency division factor and reload value, changing the frequency of PWM, can control the sound frequency of the passive buzzer.

In the main program, two variables, psc and pwm, are declared to control the frequency division coefficient and reload value of timer 4 respectively.

In the second loop, these two variables are self-added once. By way of macro definition, set the value of pwm in

Between MIN_BUZZER_PWM (10000) and MAX_BUZZER_PWM (20000), the value of psc is

Change between 0 and MAX_PSC (1000), the main flow of the program is shown in the figure.



The main function code is as follows:


```
  while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

            pwm++;
            psc ++;

            if(pwm > MAX_BUZZER_PWM)
            {
                pwm = MIN_BUZZER_PWM;
            }
            if(psc > MAX_PSC)
            {
                psc = 0;
            }
            buzzer_on(psc, pwm);
            HAL_Delay(1);

}
```

In the buzzer_on function, assign the two values of psc and pwm to the prescaler counter of timer 4 and channel 3

The comparison register of the PWM signal adjusts the period of the PWM signal, and then controls the buzzer to emit different tones and different loudness sounds.

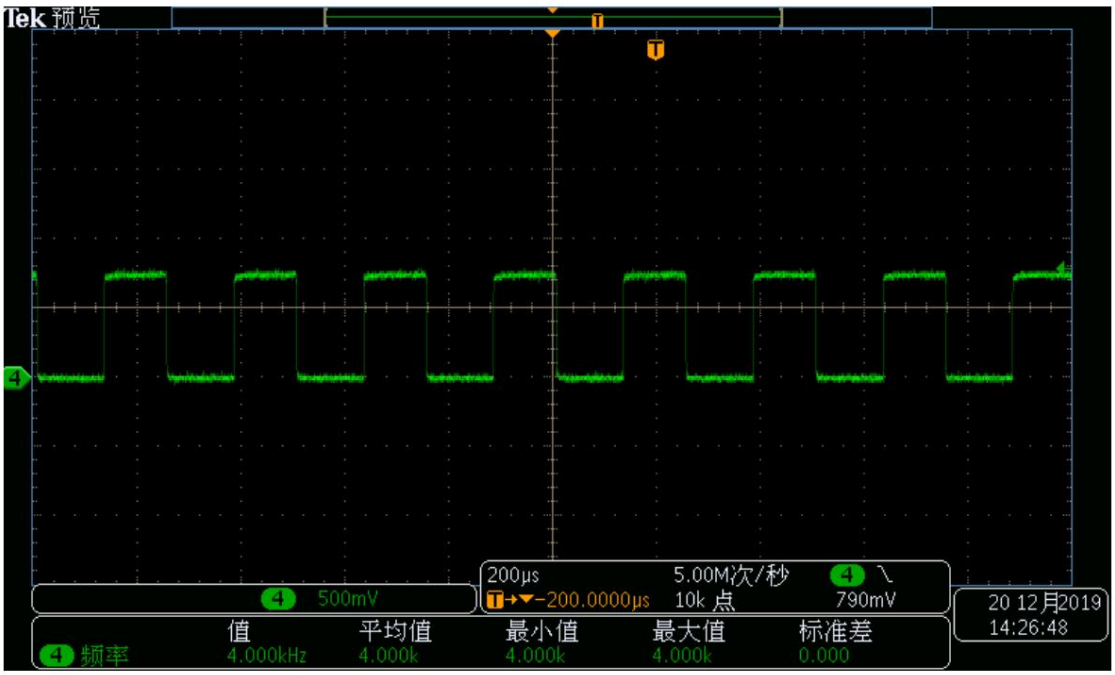
sound.

```
void buzzer_on(uint16_t psc, uint16_t pwm)
```

Function name	buzzer_on
function	Controls the divider and reload values of the buzzer timer
return value	Void
Parameter 1: psc	Set the frequency division factor of the timer
Parameter 2: pwm	Set the reload value of the timer

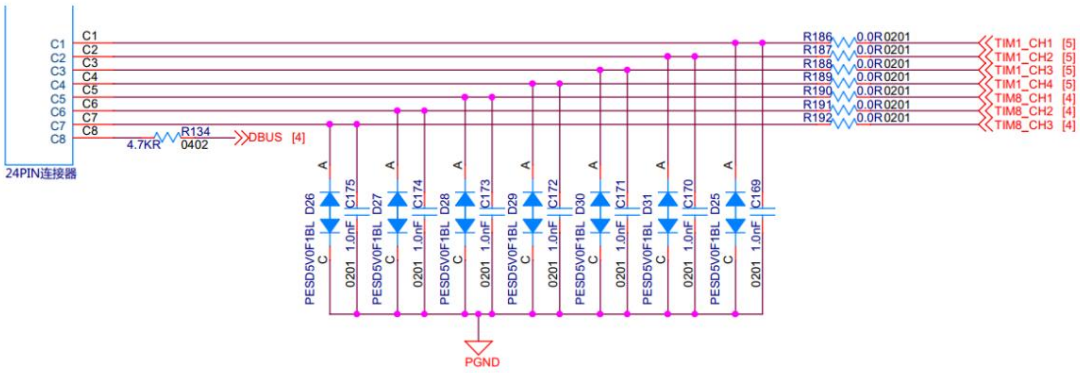
5.4.3 Effect display

Use the oscilloscope to observe the PWM output waveform, get the PWM waveform of 4kHz, and the buzzer will sound at the same time.



5.4.4 The PWM of the servo is configured in cubeMX

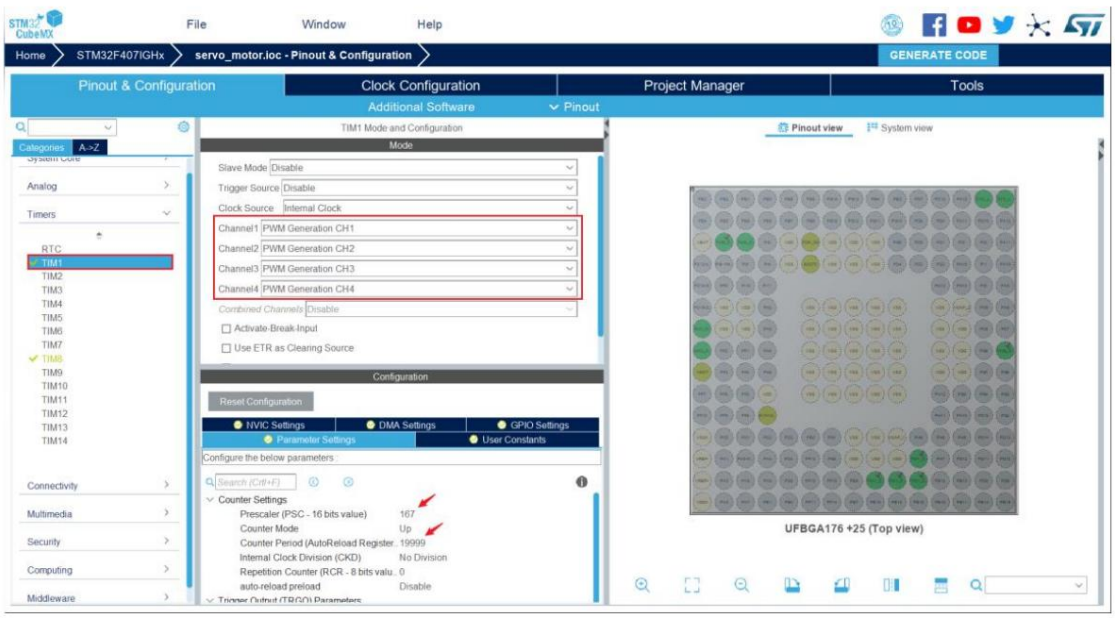
There are 7 PWM output interfaces on the development board, the schematic diagram is shown in the figure, and the physical diagram is shown in the figure.

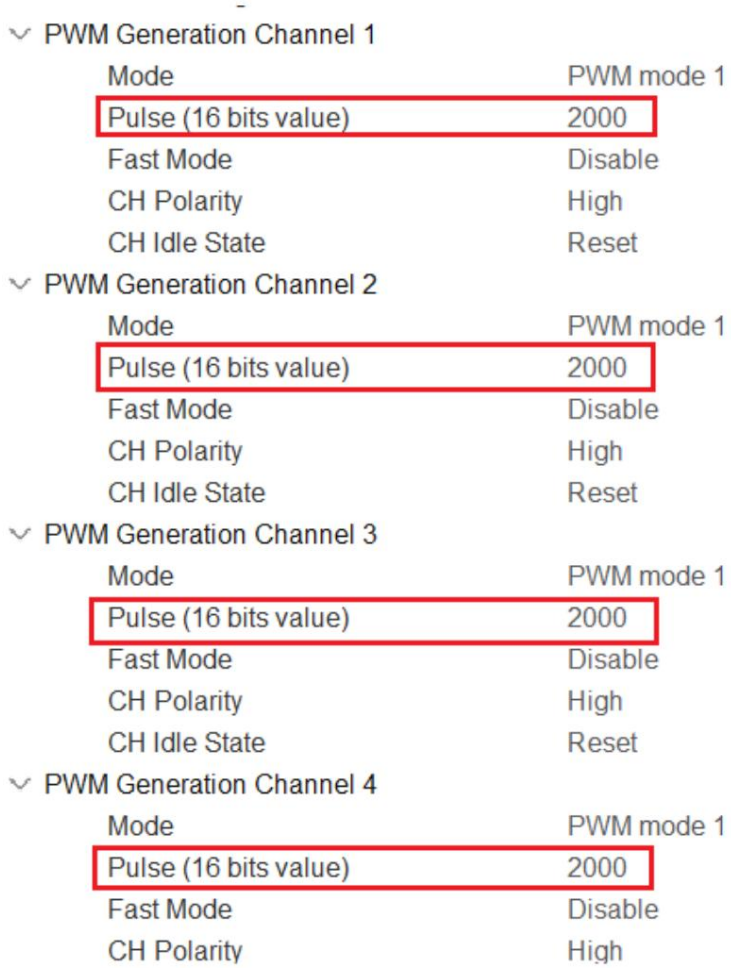


7路PWM输出

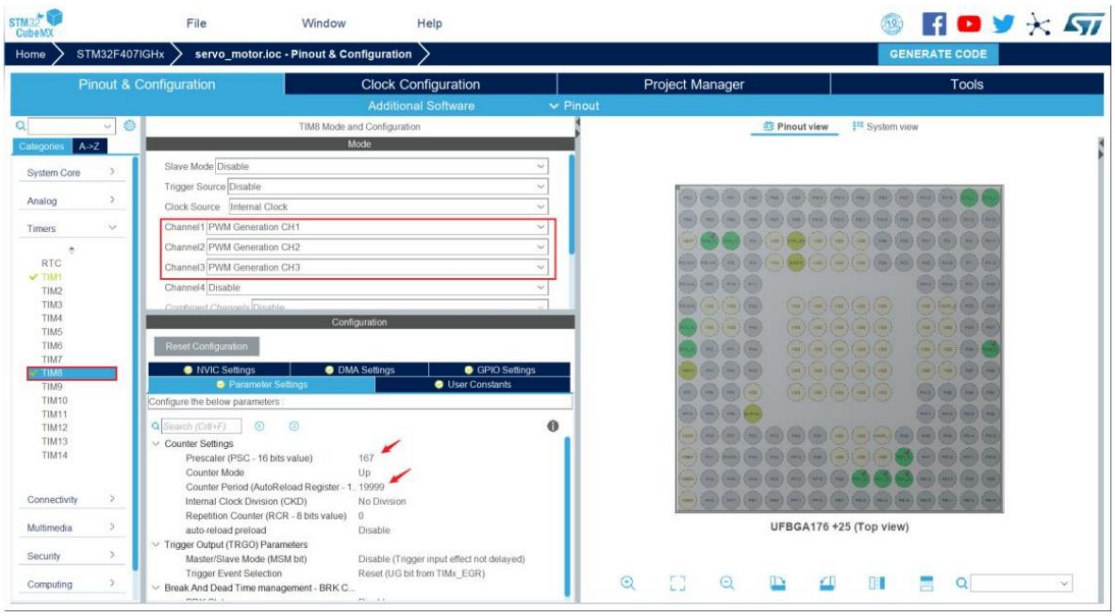
First, start timer 1, set the prescale value to 167, and set the reload value to 19999. Turn on the PWM output of 1-4

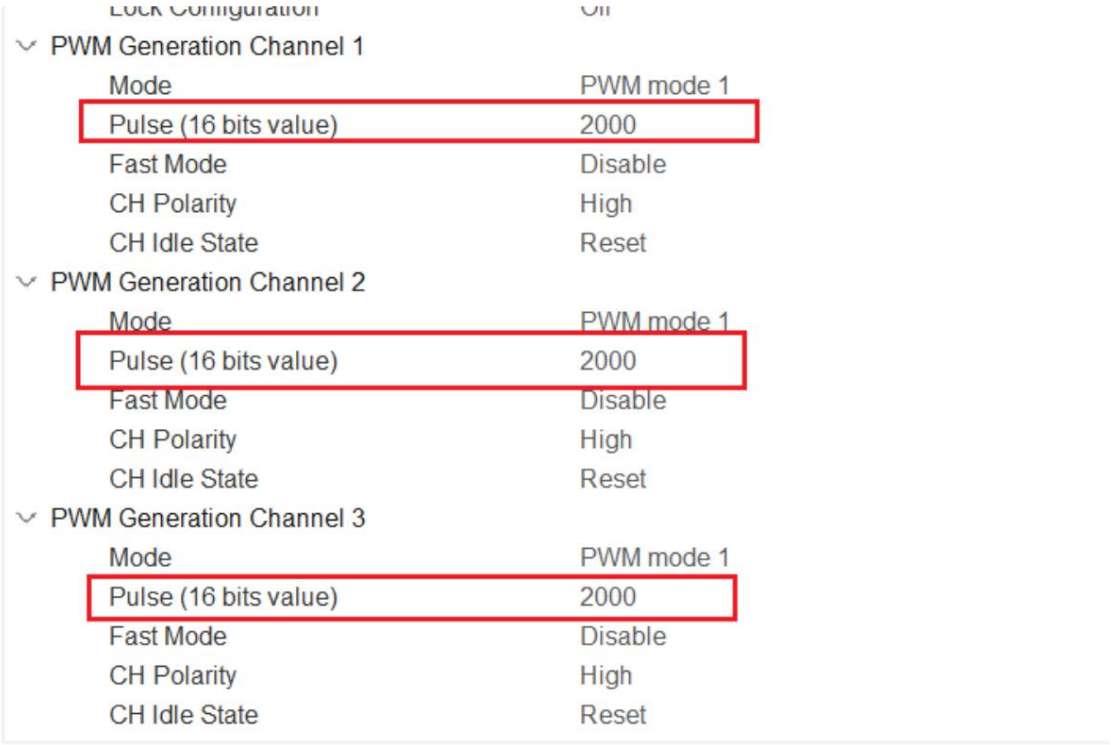
In the PWM channel setting, set the Pulse value to 2000, and the initial value of the compare register will be set to 2000ÿ



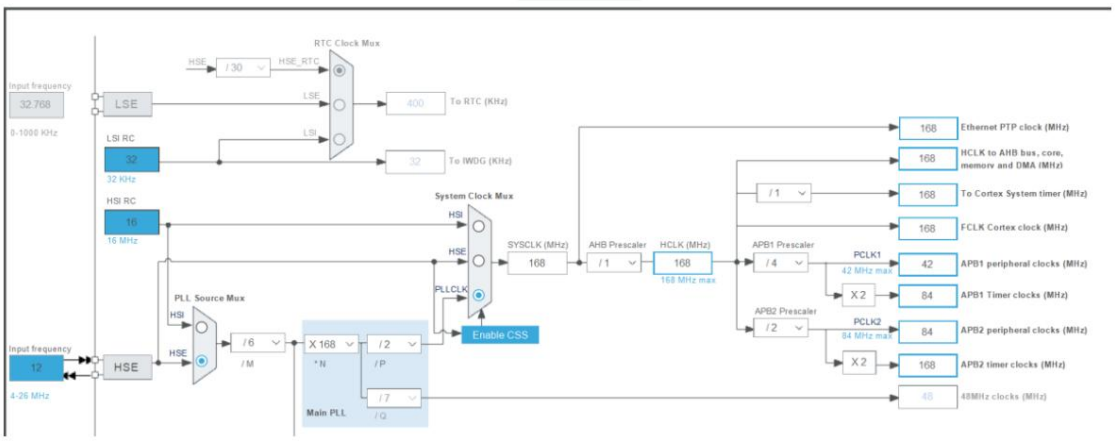


Then start timer 8, set the prescale value to 167, and set the reload value to 19999. Turn on PWM outputs 1-3 channel, in the PWM channel settings, set the Pulse value to 2000.





Finally, configure the clock tree as follows



By looking at the source code or data sheet, we know that timers 1 and 8 are mounted on the APB2 bus.

The corresponding bus frequency is 168MHz, the timer divider value is 167, the reload value is 19999, and the PWM is calculated by the formula

The output frequency of the wave is 50Hz, and the corresponding period is 20ms.

Through the knowledge learned in the PWM chapter, it is calculated that the minimum PWM duty cycle is 500/20000 or 2.5%, corresponding to high

The level time is 20ms multiplied by 2.5% is equal to 0.5ms, the maximum is 2000/20000 or 10%, and the corresponding high level time is

20ms times 10% equals 2ms.

5.4.5 Explanation of the main program of the servo

During initialization, start timer 1 and timer 8 through the HAL_TIM_Base_Start function, and then use the HAL_TIM_Base_Start function to start timer 1 and timer 8.

The HAL_TIM_PWM_Start function connects the PWM output of channels 1, 2, 3, and 4 of timer 1 to the PWM outputs of channels 1, 2, and 3 of timer 8.

The PWM output of the channel is turned on.

In the main loop, the duty cycle of the PWM is set by __HAL_TIM_SetCompare. It should be noted that

__HAL_TIM_SetCompare is not a function. After looking at the definition, you will find that this is actually a macro, and finally its function

It can still be achieved by assigning the value to a certain channel compare register of the timer.

```
#define __HAL_TIM_SetCompare                    __HAL_TIM_SET_COMPARE
#define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__) \
(((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1 = (__COMPARE__)) :\
 ((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2 = (__COMPARE__)) :\
 ((__CHANNEL__) == TIM_CHANNEL_3) ? ((__HANDLE__)->Instance->CCR3 = (__COMPARE__)) :\
 ((__HANDLE__)->Instance->CCR4 = (__COMPARE__)))
```

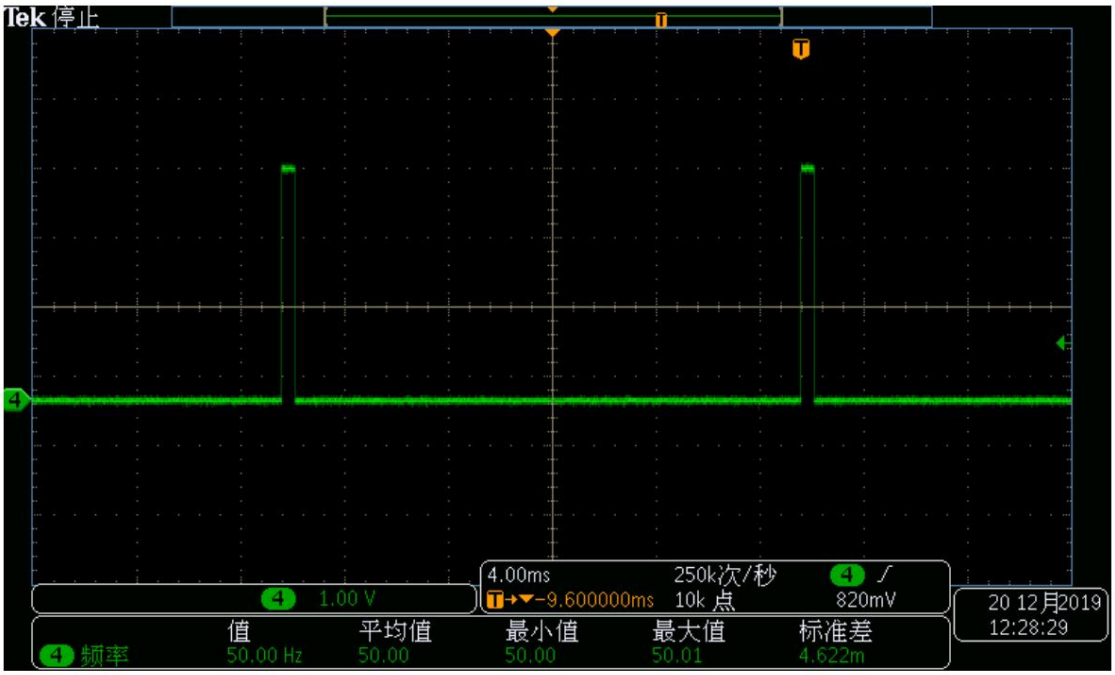
When using __HAL_TIM_SetCompare to set the duty cycle, the corresponding inputs are:

parameter 1	*htim timer handle pointer, such as timer 1 input &htim1, timer 2 output into&htim2
parameter 2	Channel Timer PWM output channel, for example, channel 1 is TIM_CHANNEL_1
parameter 3	The value that needs to be assigned to the compare register, the PWM duty cycle is equal to the compare value divided by the reload value

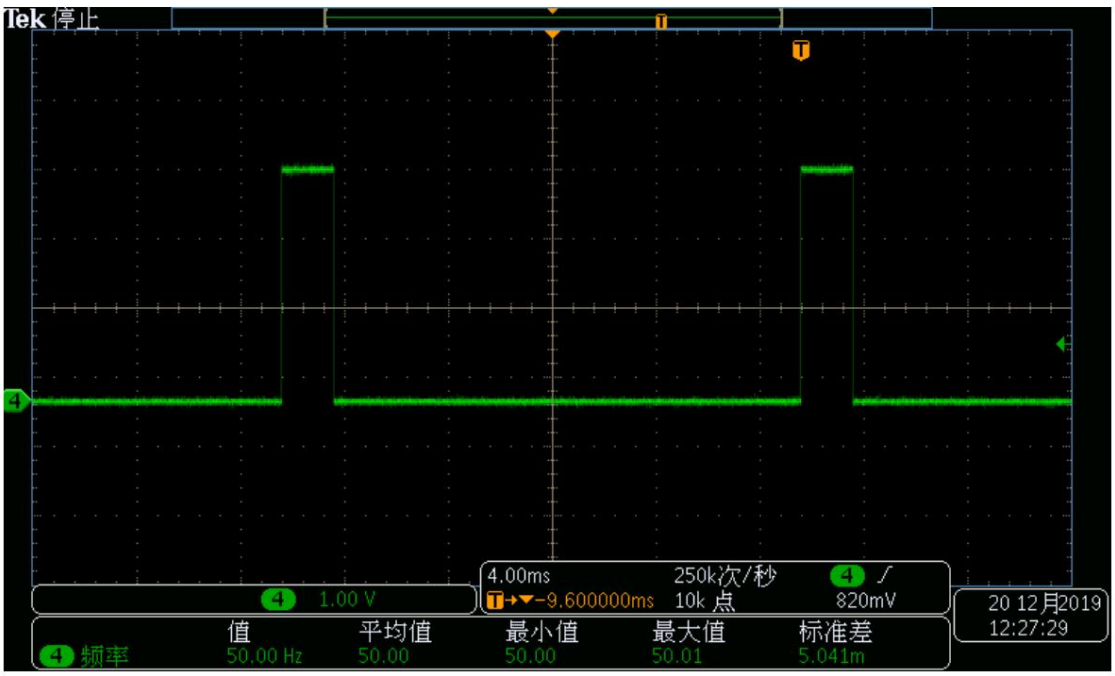
As the duty ratio of each PWM output in the main loop changes, the rotation angle of the servo also changes.

5.4.6 Demonstration of servo effect

Use an oscilloscope to observe the output waveform of PWM at 0.5ms high level and 2.5ms high level, as shown in the figure below.



0.5ms high level



2.5ms high level

5.5 Course Summary

Buzzer is a common human-computer interaction mode, which can be used to remind team members to complete a certain inspection, or to alarm and remind

A module is offline. The steering gear is a simple and easy-to-use power device that can be used to open and close the magazine cover of the robot.

The mechanical gripper control, etc.

5.6 Explanation of homework in the previous section

Procedure flow chart

The macro definition #define RGB_FLOW_COLOR_CHANGE_TIME 500 specifies that the time for RGB switching colors is

500ms, then create an array RGB_flow_color to load blue 0xFF0000FF, green 0xFF00FF00, red

0xFF0000 three colors, here the array length is set to 4 instead of 3 to facilitate the implementation of the subsequent loop switching function.

can.

First initialize timer 5 and PWM output of 1, 2, 3 channels and then enter the main loop. In order to achieve the breathing light effect, it is necessary to

Modify the duty cycle of the PWM little by little at a suitable speed, so that the light on and off will produce a smooth light and dark gradient effect.

The result is the so-called breathing lamp.

In the main loop, the changes delta_alpha, delta_red of each element in each cycle of the floating-point type are defined,

delta_green, delta_blue, and define the four element values of floating-point type alpha, red, green, blue, and finally set

Defines the aRGB value of the 32-bit representation provided to the PWM for output.

In the outer loop, first read the color of the lamp in this loop from the array, extract it through bit operation, and assign it to

alphaÿredÿgreenÿblueÿ

```
alpha = (RGB_flow_color[i] & 0xFF000000) >> 24;
red = ((RGB_flow_color[i] & 0x00FF0000) >> 16);
green = ((RGB_flow_color[i] & 0x0000FF00) >> 8);
blue = ((RGB_flow_color[i] & 0x000000FF) >> 0);
```

Next, use the difference between the next element value and the current element value, and then divide by the total switching time

(RGB_FLOW_COLOR_CHANGE_TIME) to get the change amount of each element in the inner loop

deltaÿ

```
delta_alpha = (fp32)((RGB_flow_color[i + 1] & 0xFF000000) >> 24) - (fp32)((RGB_flow_color[i] & 0xFF000000) >> 24);
delta_red = (fp32)((RGB_flow_color[i + 1] & 0x00FF0000) >> 16) - (fp32)((RGB_flow_color[i] & 0x00FF0000) >> 16);
delta_green = (fp32)((RGB_flow_color[i + 1] & 0x0000FF00) >> 8) - (fp32)((RGB_flow_color[i] & 0x0000FF00) >> 8);
delta_blue = (fp32)((RGB_flow_color[i + 1] & 0x000000FF) >> 0) - (fp32)((RGB_flow_color[i] & 0x000000FF) >> 0);

delta_alpha /= RGB_FLOW_COLOR_CHANGE_TIME;
delta_red /= RGB_FLOW_COLOR_CHANGE_TIME;
delta_green /= RGB_FLOW_COLOR_CHANGE_TIME;
```



```
        delta_blue /= RGB_FLOW_COLOR_CHANGE_TIME;
```

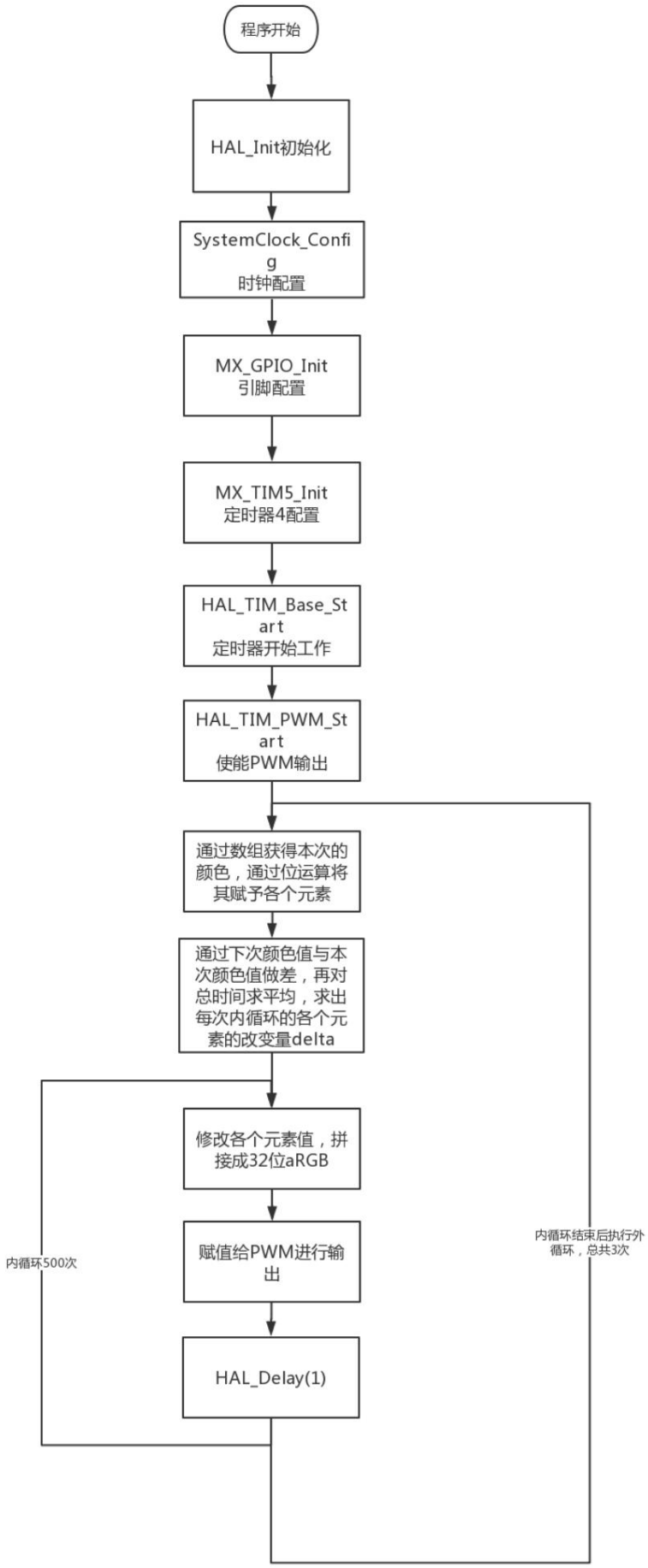
In the inner loop, the delta value is used to make small changes to the value of each element each time, and then the value of each element is changed.

The prime value is spliced into a 32-bit aRGB representation, which is used for output to the PWM channel, and finally a delay of 1ms is performed. so that

A breathing light effect that can produce light and dark gradients that switch colors.

```
for(j = 0; j < RGB_FLOW_COLOR_CHANGE_TIME; j++)
        {
            alpha += delta_alpha;
            red += delta_red;
            green += delta_green;
            blue += delta_blue;

            aRGB = ((uint32_t)(alpha)) << 24 | ((uint32_t)(red)) << 16 | ((uint32_t)(green)) << 8 | ((uint32_t)(blue)) << 0;
            aRGB_led_show(aRGB);
            HAL_Delay(1);
        }
```

The program flow chart is as follows:



6. External interrupt of key press

6.1 Key points of knowledge

ÿ Knowledge of the hardware principle of buttons

ÿ External interrupt configuration of GPIO

ÿ Button software debounce processing

ÿ Front and back in the program

6.2 Course content

In this lesson, we will introduce the hardware principle of keys, how to use the external interrupt function of stm32 to read key input, how to

Use software debounce to debounce the voltage generated by the key input. Also, will again help with the example of key interrupts

Everyone understands the idea of realizing the front and back of the program through interrupts.

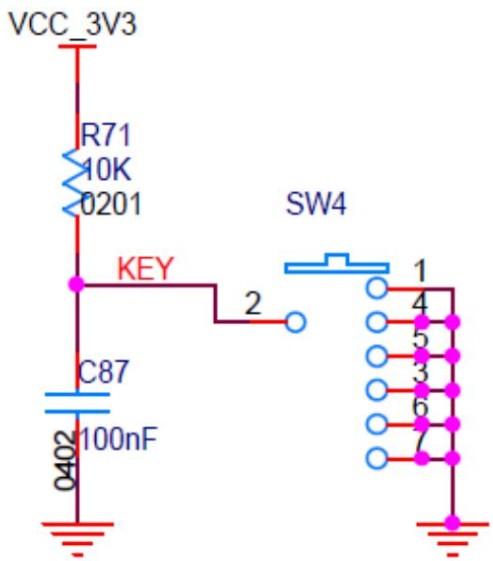
6.3 Basic Learning

6.3.1 Introduction of Button Schematic

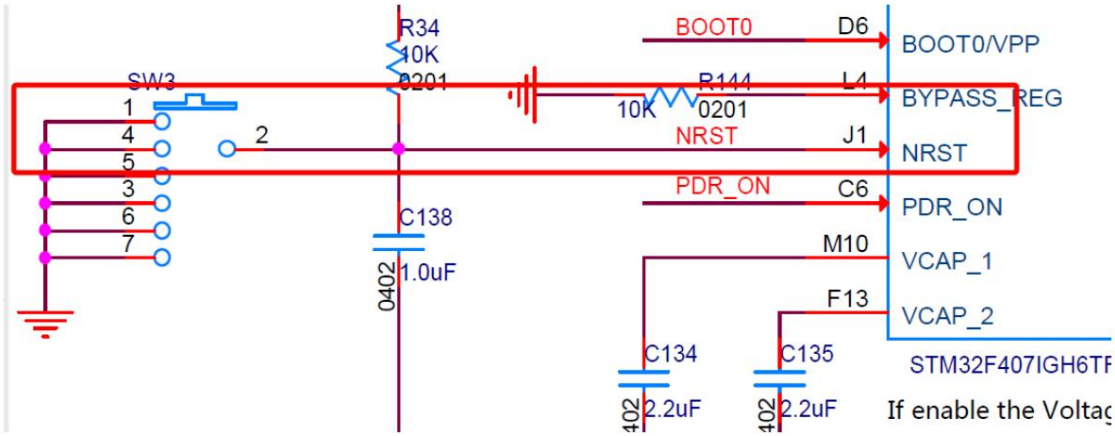
The development board type C has two buttons, one of which is a reset button and the other is a user-defined button, as shown in the figure. when pressing

When the key is not pressed, the pin will be clamped at a high level by the pull-up resistor; when the key is pressed, the pin will be directly connected to GND and at a low power level

flat.



User button



reset button

6.3.2 Button software debounce

First of all, let's introduce the jitter problem of key input. Due to the elasticity of the mechanical structure of the key, the switch will not immediately be pressed when pressed.

When it is turned on and off, it will not be disconnected immediately, which causes the input signal of the button to vibrate when it is pressed and disconnected, such as

If the jitter problem is not addressed first, the read key signal may be erroneous.

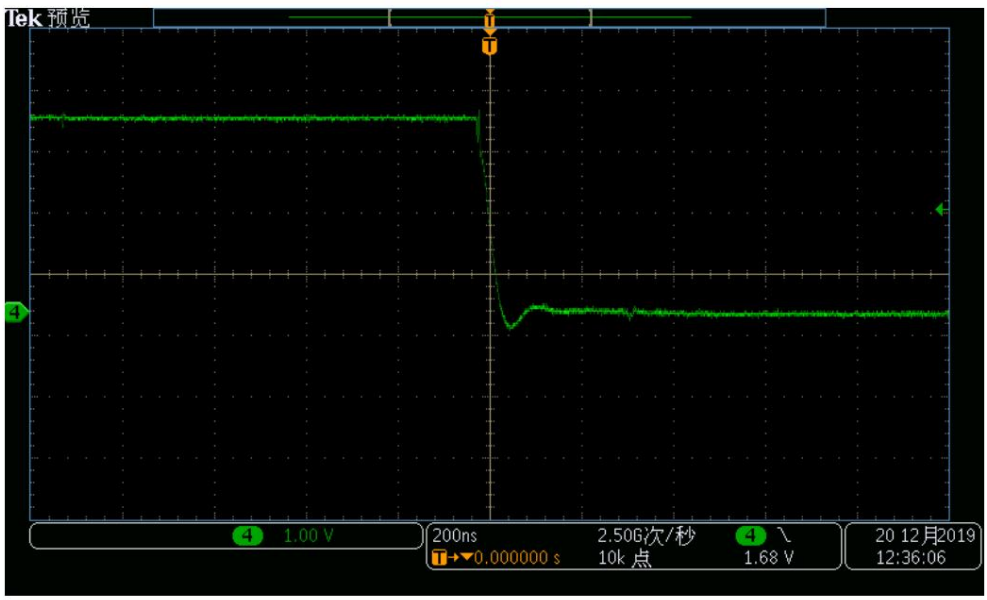
In order to eliminate this problem, it can be realized by software debounce and hardware debounce. This section introduces the practical application of software filtering.

present method. The idea of software filtering is very simple. The jitter is generated at the two edge moments of the button press and release, also called drop.

Edge (level from high to low) and rising edge (level from low to high), so only need to delay at the edge, wait until

After the key input has been stable, you can read the signal.

Generally, when software debounce is used, a delay of 20ms will be performed, and the oscilloscope captures the key waveform as shown in the figure.



button press waveform

6.3.3 External interruption

External interrupts are usually interrupts caused by GPIO transitions. In stm32, every GPIO can be used as external

External interrupt trigger source, there are 16 external interrupt lines, corresponding to GPIO pins 0-15, each external interrupt

Can be connected to any group of corresponding pins, but cannot be reused. For example, the external interrupt Line0 can be combined with PA0,

Any pin 0 such as PB0, PC0, etc. is connected, but if it has been connected to PA0, it cannot be connected to PB0, PC0 at the same time

other pins are connected.

External interrupts support three transition modes of GPIO, as follows:

ÿ Rising edge interrupt: When the level of GPIO jumps from low level to high level, an external interrupt is triggered.

ÿ Falling edge interrupt: When the level of GPIO jumps from high level to low level, an external interrupt is triggered.

ÿ Rising edge and falling edge interrupt: When the level of GPIO jumps from low level to high level and from high level to low level

Usually, an external interrupt can be triggered.

6.4 Program Learning

6.4.1 Configuration of external interrupt in **cubeMX**

The GPIO of STM32 provides an external interrupt function. When the GPIO detects a voltage transition, it will issue an interrupt trigger signal

To STM32, make the program enter the external interrupt service function.

Set the PA0 pin as the input pin of the key, and set it to the external interrupt mode.

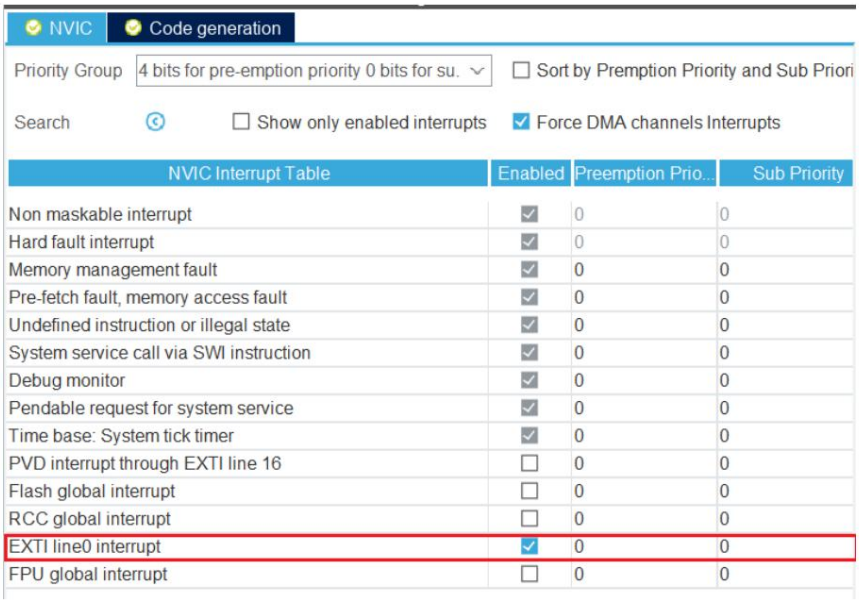
Then click on the GPIO tab, set the pins as follows, and set the GPIO mode to the external medium triggered by rising and falling edges.

off, the pull-up and pull-down resistors are set as pull-up resistors, and finally the user label is set as KEY.

There are three trigger modes for external interrupt: rising edge triggering, falling edge triggering and both upper and lower edge triggering. The similarities and differences can be seen in the following table:

Trigger method	name in cubeMX	Triggering conditions
Rising edge trigger	External Interrupt Mode with Rising edge trigger detection	Level on external interrupt trigger pin transition from low to high
Falling edge trigger	External Interrupt Mode with Falling edge trigger detection	Level on external interrupt trigger pin transition from high to low
Both upper and lower edges trigger	External Interrupt Mode with Rising/Falling edge trigger detection	There is a level on the external interrupt trigger pin jump

Under the NVIC tab, you can see that the external interrupt has been enabled.



6.4.2 HAL_GPIO_ReadPin function introduction

The HAL library provides the function HAL_GPIO_ReadPin to read the level on the pin. The function description is as follows:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Function name	HAL_GPIO_ReadPin
function	return pin level
return value	GPIO_PinState, if it is high, return GPIO_PIN_SET (corresponding to is 1), if it is low, it returns GPIO_PIN_RESET (corresponding to 0)
Parameter 1: GPIOx	Corresponds to a GPIO bus, where x can be A…I. For example, PH10, enter GPIOH
Parameter 2: GPIO_Pin	Corresponds to the number of pins. Can be 0-15. For example, PH10, enter GPIO_PIN_10

6.4.3 Introduction of interrupt callback function

Whenever an external interrupt is generated, the program will first enter the external interrupt service function. In stm32f4xx_it.c, you can find

The function EXTI0_IRQHandler, which performs the interrupt type by calling the function HAL_GPIO_EXTI_IRQHandler

Judge and process the registers involved in the interrupt, after the processing is completed, it will call the interrupt callback function

HAL_GPIO_EXTI_Callback, write the function that needs to be executed in this interrupt in the interrupt callback function.

6.4.4 The front and back of the program

In this experiment, it was found that there is code in both the main loop and the interrupt callback function. This is a very typical previous background mode

Organizational Engineering. What is the front and back mode? Imagine the operation mode of a restaurant, the restaurant is often divided into front desk callers

Diners and backstage chefs, the front desk will only work when there are guests, or when a dish is prepared in the backstage, while the backstage cooks only work.

I have been busy cooking, and only stop for a while when a new order comes from the front desk or when the dishes are already ready.

In the microcontroller, the interrupt is the foreground, and the loop is the background. The interrupt will only be processed when the interrupt source is generated.

The loop, on the other hand, keeps working and only pauses when interrupted by an interrupt. The similarities and differences between the front-end and back-end programs can be seen in the following table:

	foreground program	background program
Operation mode	interrupt	cycle
Type of task processed	burst task	repetitive tasks
Features of the task	The task is light, requiring a timely response; the task is heavy, and the execution is stable	

When writing front-end and back-end programs, you need to pay attention to avoid executing too long or time-consuming code in the front-end program.

The program can be executed as soon as possible to ensure that it can respond to sudden events in real time, and the more complicated and time-consuming tasks are generally placed.

Processed in the background program.

The front and back mode can help us improve the time utilization of the microcontroller, so as to organize more complex projects.

6.4.5 Program Flow

The tasks undertaken by the front-end and back-end tasks in the program of this lesson are:

foreground program	background program
Record the state rising_falling_flag of the key rollover	Execute processing work and press keys according to the recorded flip state Judgment of status

In the main loop, first use the edge detection flag rising_falling_flag to determine whether the button is pressed or released

Edge, if it is a falling edge (rising_falling_flag == GPIO_PIN_RESET), turn on the LED light,

If it is a rising edge (rising_falling_flag == GPIO_PIN_SET) then turn off the LED light. for

To prevent false triggering, after passing the judgment of edge detection, the program will read the level again, and confirm that the falling edge is followed by

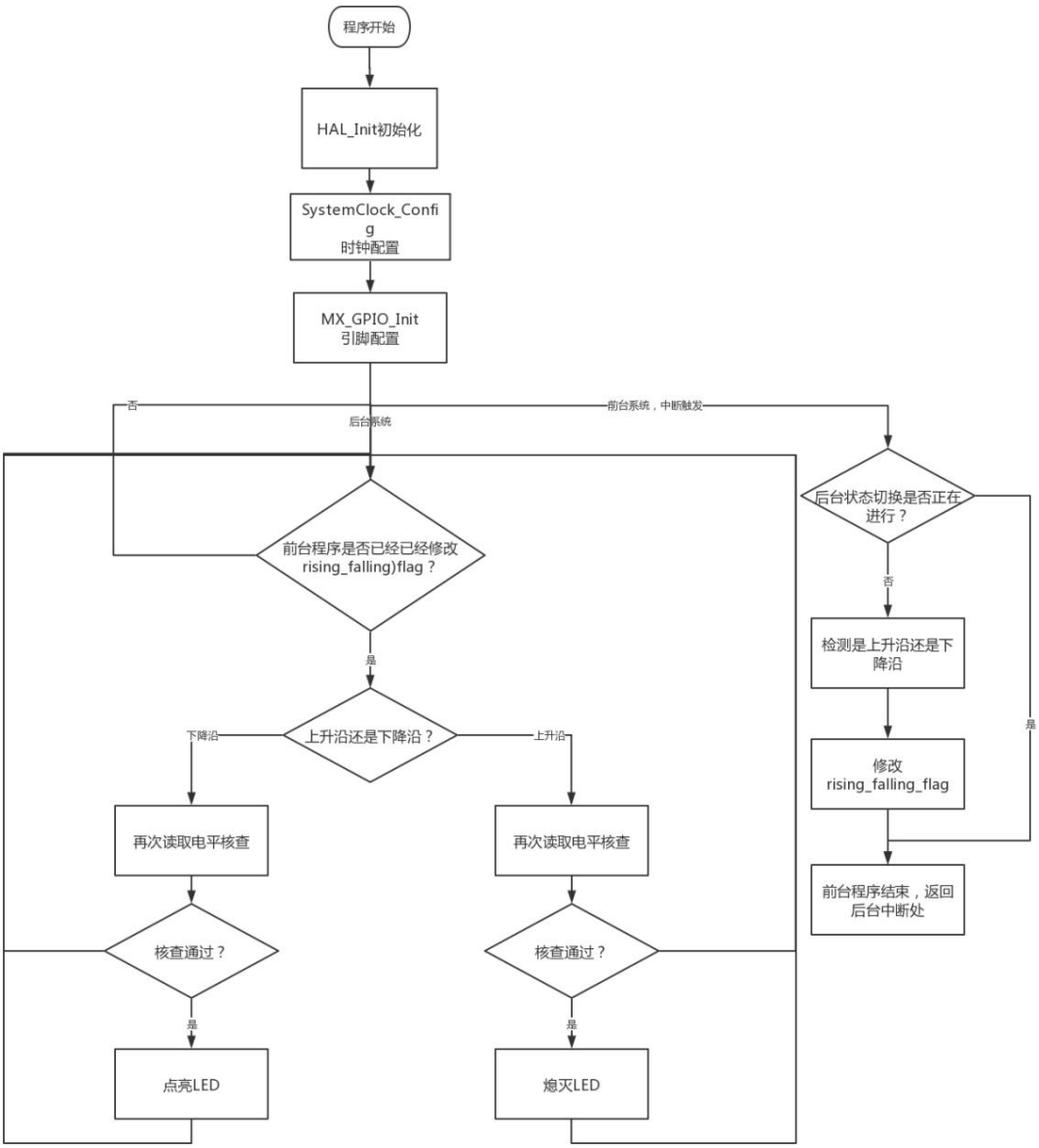
A low level is followed or a rising edge is followed by a high level, if not the LED state is not toggled.

In the interrupt callback function, use HAL_GPIO_ReadPin to assign a value to rising_falling_flag to judge

Whether the interrupt is triggered on a rising or falling edge.

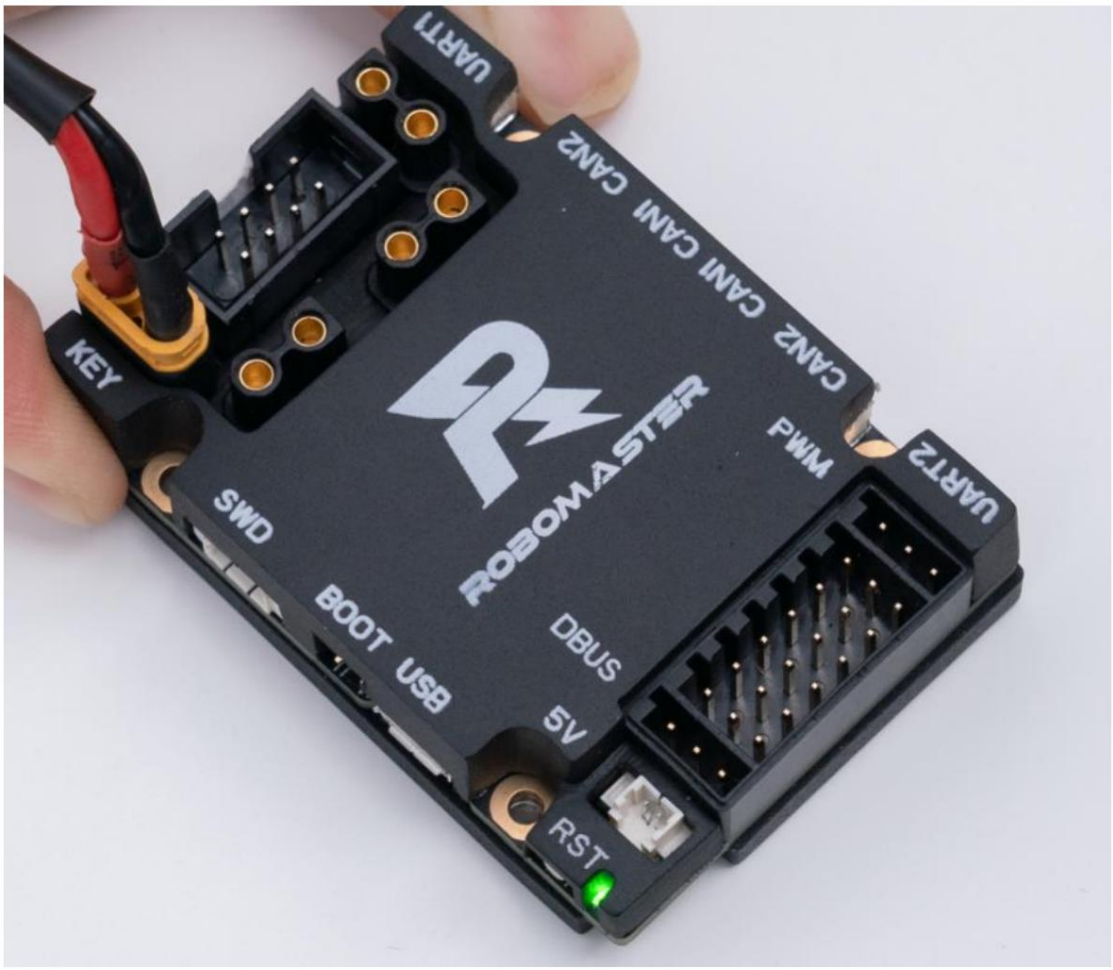
Use exit_flag to achieve mutual exclusion between the main loop and the interrupt callback function, to ensure the function in the interrupt handler function (judgment interrupt rising/falling edge) is only performed after the main loop completes the judgment, or the judgment of the main loop is only performed after the interrupt handler runs (that is, a rising or falling edge is detected) before proceeding.

The program flow chart is as follows:



6.4.6 Effect display

When the button is pressed, the green light is on; when the button is released, the green light is off, as shown in the figure.



6.5 Course Summary

The button is also a human-computer interaction operation, and can also be set to the input mode of stm32, which can be used to set a certain item function activation, etc. For example, pressing the button will start the RoboMaster robot gimbal calibration operation. External interrupts are also Important interrupt types in stm32, often used for sensor data processing, such as gyroscopes, accelerometers, magnetometers The data preparation is interrupted to notify stm32 that new sensor data has been generated. On both sides of the sentry robot are often installed red External sensors, when the sentinel robot approaches the uprights at both ends, will also generate an external interrupt to notify the robot to turn around in time.

7. ADC sampling battery voltage

7.1 Knowledge points

ÿ ADC principle

ÿ Internal VREFINT voltage introduction

ÿ Introduction to resistor divider circuit

ÿ CubeMX configuration of ADC

7.2 Course content

This lesson will introduce the ADC (Analog to Digital Converter) analog-to-digital conversion function of STM32. transmitted in the microcontroller

The input signals are all digital signals, which represent 1 and 0 of digital logic through discrete high and low levels, but in the real physical world

Only analog signals exist in the world, that is, signals that change continuously. convert these continuously changing signals - such as heat, light, sound,

The speed is converted into a continuous electrical signal through various sensors, and then the continuous analog signal is converted into a discrete signal through the ADC function

The digital signal is processed by the microcontroller.

7.3 Basic Learning

7.3.1 Introduction to ADC principle

The general ADC workflow is sampling, comparison, and conversion.

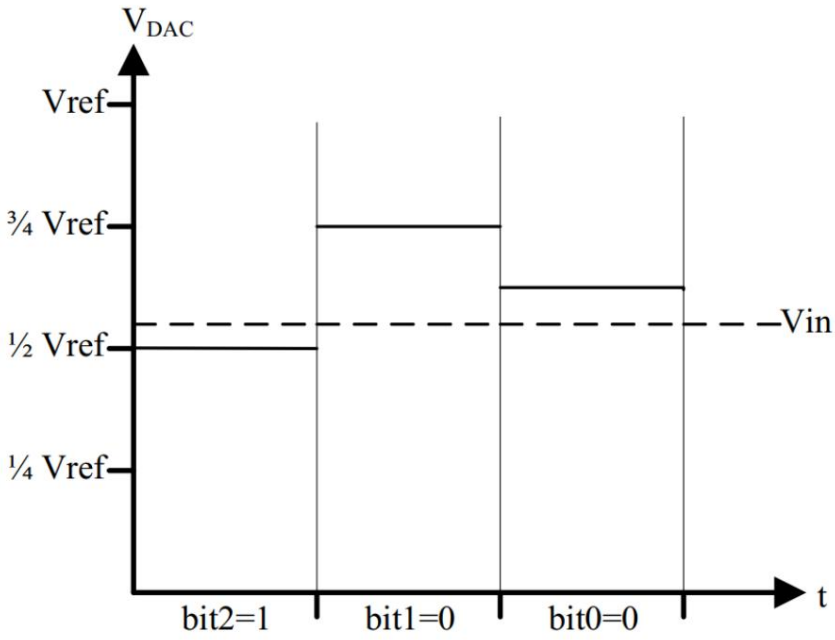
ÿ Sampling: refers to collecting the analog voltage at a certain moment,

ÿ Comparison: refers to comparing the sampled voltages in the comparison circuit,

ÿ Conversion: refers to converting the result in the comparison circuit into a digital quantity.

The stm32f4 uses a 12-bit successive approximation ADC (SAR-ADC). The following figure is an example to introduce the comparison of 3-bit ADC

Procedure



Different bits are given weights of 1/2, 1/4, and 1/8 respectively, and the sampling value of the analog signal is Vin,

1. Compare with 1/2Vref, if Vin is greater than 1/2Vref, mark the first digit as 1,

2. Compare with 3/4Vref, if Vin is less than 3/4Vref, mark the second bit as 0,

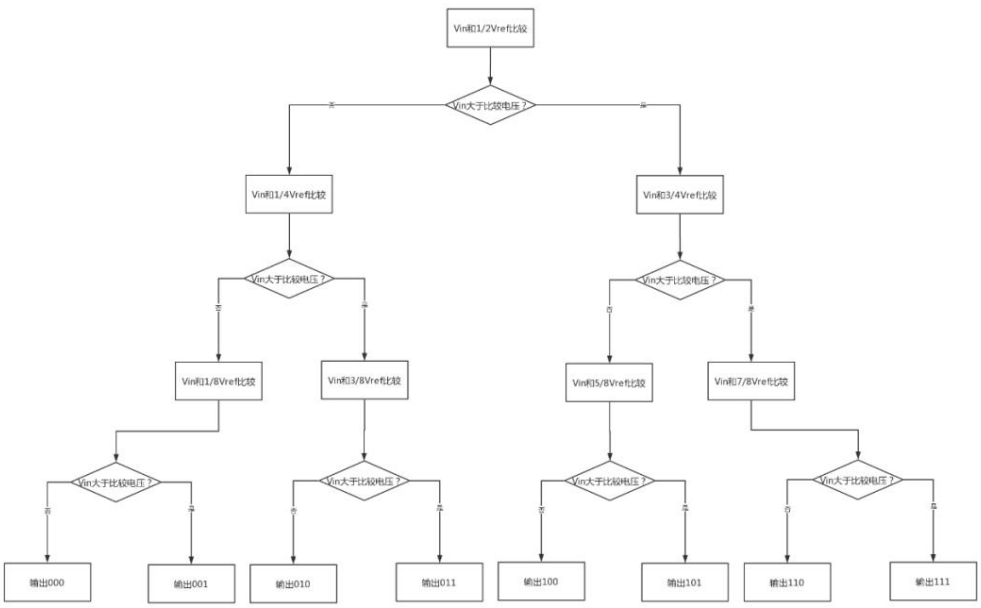
3. Compare with 5/8Vref, if Vin is less than 5/8Vref, mark the third bit as 0.

Vin in the figure outputs 100 after passing through this three-bit ADC. The result of the conversion is 1/2Vref, by doing this step by step

In this comparison process, the analog voltage obtained by sampling is compared with the weighted value of the internal reference voltage Vref.

assign different weights.

If the input is an unknown voltage, the complete comparison flow chart is as follows:



stm32 supports up to 12-bit ADC. Generally, the more bits of ADC, the higher the conversion accuracy, but at the same time the conversion speed is higher.

will also slow down.

In addition, there is a calibration voltage VREFINT inside stm32, the voltage is 1.2V, when the power supply voltage is not 3.3V, it can make

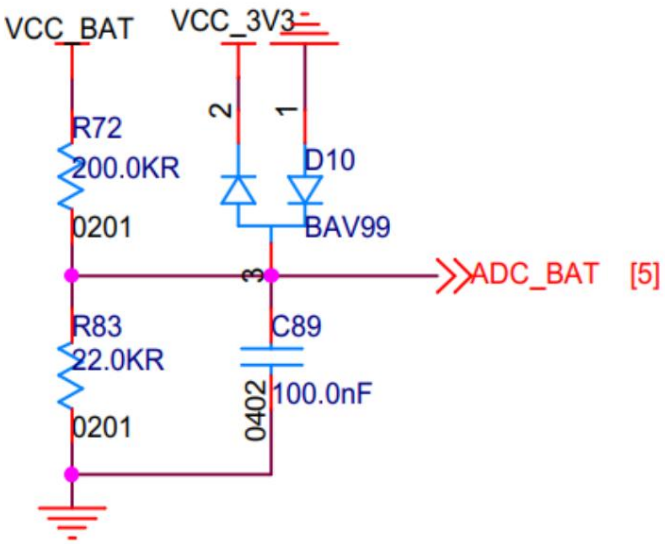
Use the internal vrefint channel to capture 1.2V as Vref to improve accuracy.

7.3.2 Introduction of resistor divider circuit

The figure below shows a resistor divider circuit used to read battery voltage. Since the power supply provided by the battery is a high voltage of 24V,

The withstand voltage of the single chip pin is only 0-3.3V, so it needs to be processed by a voltage divider circuit, and filters and diodes are used.

Limiting circuit for protection.



电压检测

Here, the 24V voltage is divided by a voltage divider circuit of 200Kÿ and 22Kÿ

$$= 24 \text{ ÿ } 22 \frac{22}{+ 200} = 2.38$$

The divided voltage can be obtained about 2.38V, and then the voltage is sent to the secondary circuit. In the secondary circuit, the first pass

A 100nF capacitor is used for filtering to make the output voltage more stable, and then a diode protection circuit is used to limit the voltage

Between 3.3V and 0V, when the voltage is greater than 3.3V, the diode conducts forward, and the voltage is limited to 3.3V, when generating

When the voltage is negative (the voltage is less than 0V), the diode conducts forward, and the ground voltage of the output point is limited to 0V.

7.4 Program Learning

7.4.1 Configuration of **ADC** in **cubeMX**

Among them, the power ADC pin is PF10, using channel 8 of ADC3, the schematic diagram is shown in the figure, and the internal 1.2V of stm32

The calibration voltage Vrefint is in ADC1.

The configuration in cubeMX is as follows:

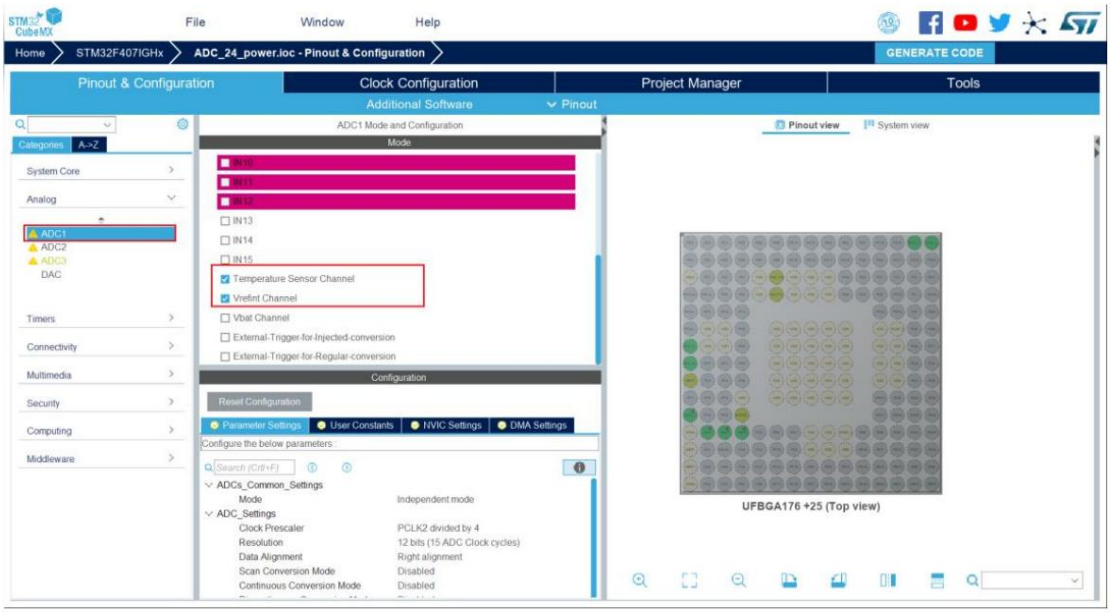
1. Turn on ADC1 and ADC3 for internal 1.2V Vrefint channel reading and battery voltage ADC3 channel respectively

   8 read.

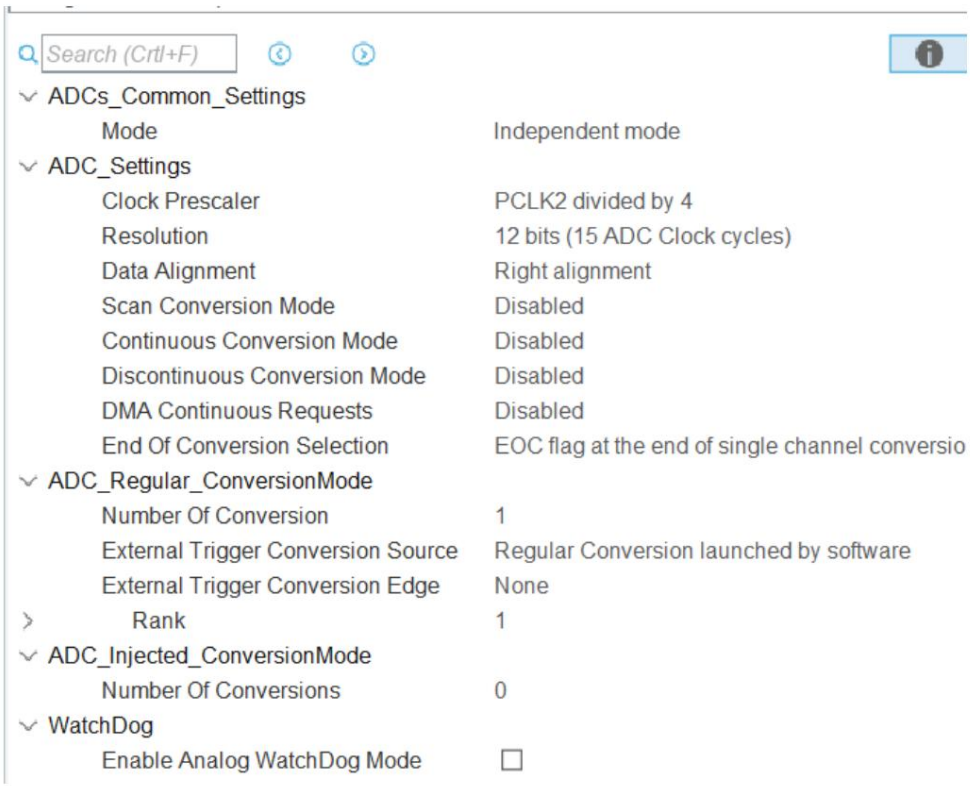
2. Turn on ADC1 in cubeMX, and check Vrefint Channel in the settings to read the internal reference voltage.

   The setting of ADC in cubeMX is as shown in the figure. The sampling frequency is set to PCLK2/4, the number of sampling bits is 12, and the data set

   Set to right alignment, the rest are left as default. Among them, Vrefine is completed inside stm32, and there is no corresponding pin.



The final ADC configuration is shown in the figure.

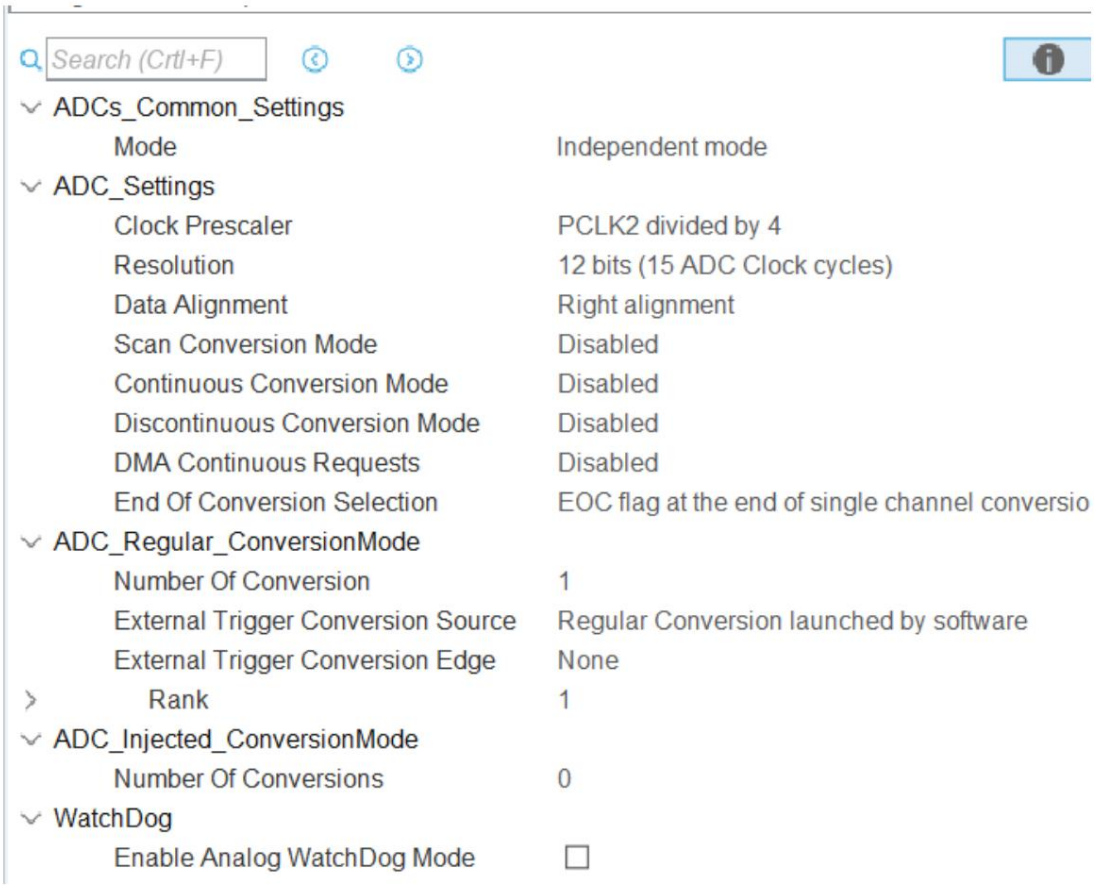


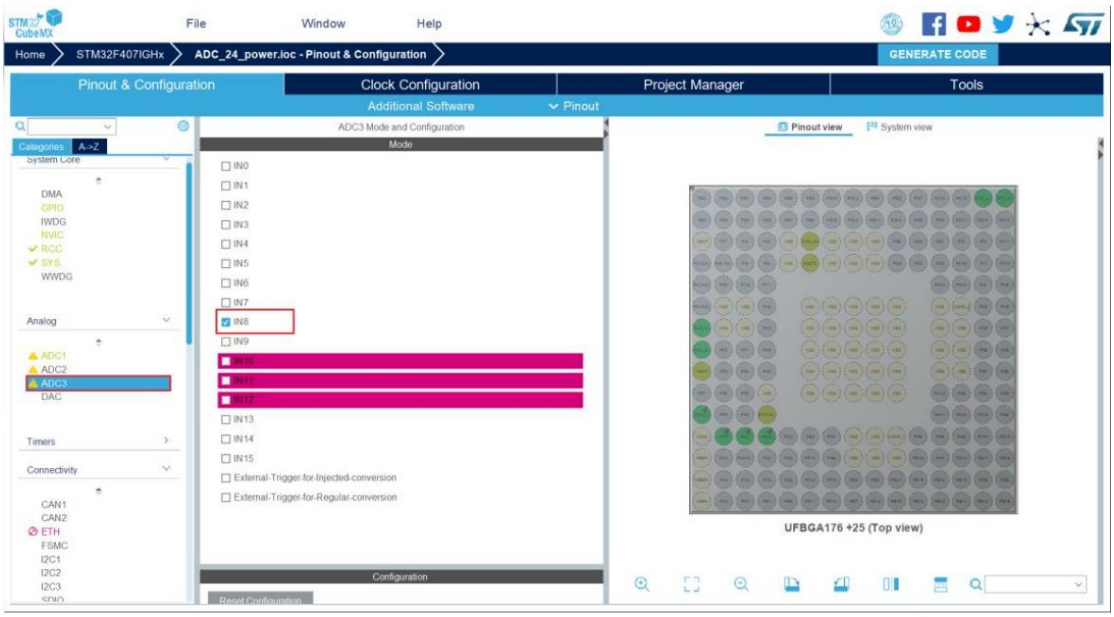
Here is a table showing the functions in the ADC settings in cubeMX.

name	Features
Clock Prescaler	Set the sampling clock frequency
Resolution	Set sampling precision
Data Alignment	Set data alignment
Scan Conversion Mode	Scan conversion mode on/off
Continuous Conversion Mode Continuous Conversion Mode On/Off	
Discontinuous Conversion Mode Discontinuous Conversion Mode On/Off	
DMA Continuous Requests	DMA continuous start on/off
End of Conversion Selection	EOC flag sent after each channel conversion / all channel conversions Send EOC flag when finished

3. Turn on ADC3 in cubeMX, and turn on its IN8 for battery voltage reading, and its setting is the same as ADC1.

You can see that the PF10 corresponding to ADC3 in the pin image turns green.





7.4.2 Use of the internal **VREFINT** voltage

VREFINT is the internal reference voltage of ADC 1.2V. By sampling the ADC value of the internal reference voltage of 1.2V and Vref

The weighted values are compared to obtain the output value of the ADC. Generally speaking, the ADC of STM32 uses Vcc as Vref,

However, in order to prevent Vcc from fluctuating greatly and causing Vref to be unstable, which in turn leads to inaccurate comparison results of sampling values, STM32

It can be calibrated by the internal reference voltage VREFINT, and then compare the ADC with VREFINT as a reference

The sampling value of , so as to obtain relatively high precision, the voltage of VREFINT is 1.2V.

The 1.2V voltage is sampled multiple times through a function, the average value is calculated, and then it is combined with the data from the ADC.

Compare the values to obtain the analog voltage value voltage_vrefint_proportion corresponding to the unit digital voltage. The calculation formula is as follows

Next, set the digital value obtained by sampling to average_adc:

$$\_ = \frac{\_}{200}$$

$$\text{voltage\_vrefint\_proportion} = \frac{1.2v}{\_} = 200 \ddot{y} 1.2/\_$$

```
void init_vrefint_reciprocal(void)
{
        uint8_t i = 0;
        uint32_t total_adc = 0;
        for(i = 0; i < 200; i++)
        {
                total_adc += adcx_get_chx_value(&hadc1, ADC_CHANNEL_VREFINT);
        }

        voltage_vrefint_proportion = 200 * 1.2f / total_adc;
}
```

7.4.3 Introduction to ADC sampling correlation function

The HAL library provides the following functions related to ADC sampling:

```
HAL_StatusTypeDef HAL_ADC_ConfigChannel(ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)
```

Function name	HAL_ADC_ConfigChannel
The function is used to set various attribute values of the ADC channel, including conversion channels, sequence sorting, sampling time, etc.	
return value	HAL_StatusTypeDef, several states defined by the HAL library, if the ADC starts successfully work, return HAL_OK
parameter 1	TIM_HandleTypeDef * hadc is the handle pointer of ADC, if it is adc1, enter it

	&hadc1, adc2 enter &adc2
parameter 2	ADC_ChannelConfTypeDef* sConfig is a pointer to the structure of ADC settings. We first assign a value to the sConfig structure, and then enter its pointer as a parameter to the function

HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc)

Function name	HAL_ADC_Start
function	Enable ADC sampling
return value	HAL_StatusTypeDef, several states defined by the HAL library, such as Return HAL_OK if the ADC starts to work successfully
parameter	TIM_HandleTypeDef * hadc is the handle pointer of ADC, If it is adc1, enter &hadc1, and for adc2, enter &adc2

HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout)

Function name	HAL_ADC_PollForConversion()
function	Wait for ADC conversion to end
return value	HAL_StatusTypeDef, several states defined by the HAL library, such as Return HAL_OK if the ADC starts to work successfully
parameter 1	TIM_HandleTypeDef * hadc is the handle pointer of ADC, If it is adc1, enter &hadc1, and for adc2, enter &adc2
parameter 2	uint32_t Timeout maximum time to wait

uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)

Function name	HAL_ADC_GetValue
function	Get ADC value
return value	HAL_StatusTypeDef, several states defined by the HAL library, such as Return HAL_OK if the ADC starts to work successfully

parameter	TIM_HandleTypeDef * hadc is the handle pointer of ADC, If it is adc1, enter &hadc1, and for adc2, enter &adc2

7.4.4 Program flow

In this program, first adc sample the internal reference voltage VREFINT and use it as the calibration value.

In init_vrefint_reciprocal, the VREFINT voltage is sampled 200 times, and then the average value is used.

The voltage value of VREFINT is 1.2V and the average value sampled by the ADC is removed to calculate voltage_vrefint_proportion,

The voltage value sampled in the subsequent ADC is multiplied by voltage_vrefint_proportion to calculate the internal reference voltage.

Press the calibrated ADC value.

```
void init_vrefint_reciprocal(void)
{
        uint8_t i = 0;
        uint32_t total_adc = 0;
        for(i = 0; i < 200; i++)
        {
                total_adc += adcx_get_chx_value(&hadc1, ADC_CHANNEL_VREFINT);
        }

        voltage_vrefint_proportion = 200 * 1.2f / total_adc;
}
```

Then, the battery voltage value that has passed through the voltage divider circuit is sampled by the ADC, and the sampling result is compared with the voltage value of the battery.

Voltage_vrefint_proportion is multiplied to get the ADC sampling value in the range of 0-3.3V.

The first sampling value is the result of the voltage division, and the value of the voltage needs to be calculated in reverse. The resistance values of the voltage divider are 200Kÿ and 22Kÿ,

Since (22K ÿ + 200K ÿ) / 22K ÿ = 10.09, after multiplying this value, the voltage value of the battery can be obtained.

```
fp32 get_battery_voltage(void)
{
        fp32 voltage;
        uint16_t adcx = 0;

        adcx = adcx_get_chx_value(&hadc3, ADC_CHANNEL_8);
```



```
    // (22K ÿ + 200K ÿ) / 22K ÿ = 10.090909090909090909090909090909
    voltage = (fp32)adcx * voltage_vrefint_proportion * 10.090909090909090909090909090909f;

    return voltage;
}
```

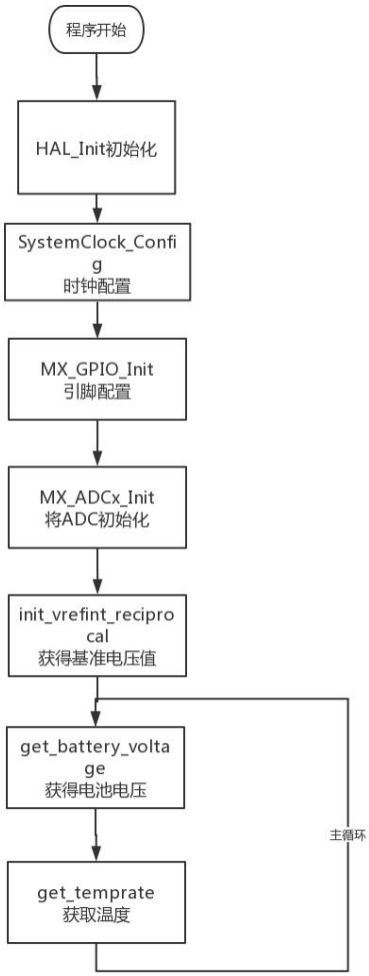
The temperature value of the onboard temperature sensor can also be obtained through the ADC, which is also sampled through the ADC value first, and then the sampling

After the end, bring the ADC sampling result adc into the formula temperate = (adc - 0.76f) * 400.0f + 25.0f, thus

Calculate the temperature value.

```
fp32 get_temprate(void)
{
    uint16_t adcx = 0;
    fp32 temperate;

    adcx = adcx_get_chx_value(&hadc1, ADC_CHANNEL_TEMPSENSOR);
    temperate = (fp32)adcx * voltage_vrefint_proportion;
    temperate = (temperate - 0.76f) * 400.0f + 25.0f;

    return temperate;
}
```

The program flow is as follows:



7.4.5 Effect display

Use a voltmeter to measure the supply voltage and compare it with the voltage measured by the program, as shown.

Voltmeter measurement diagram

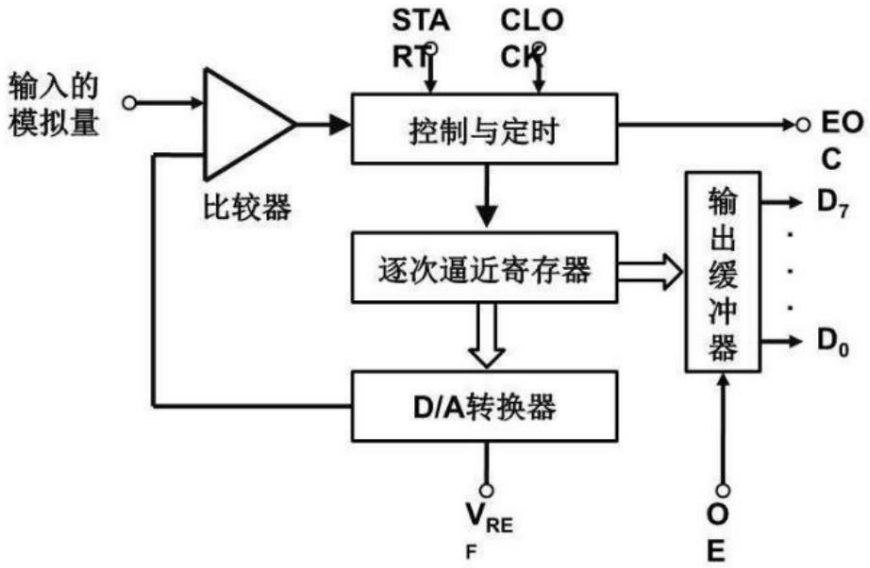
Watch 1

Name	Value	Type
voltage	25.5901451	float
temperature	27.6057968	float
handware_version	2	unsigned char
<Enter expression>		

Program measurement chart

7.5 Advanced Learning

The successive ADC obtains the conversion value through a bit-by-bit comparison, and its schematic diagram is shown below.



As shown in the figure above, the entire circuit consists of comparators, D/A converters, buffer registers and several control logic circuits. effect

As follows:

ÿ Comparator: used to compare the input voltage value with the output voltage of the D/A converter, when the input voltage is greater than the D/A converter

When the voltage is high, the output is 1, otherwise the output is 0;

ÿ D/A converter: the reverse process of ADC, which converts the digital quantity recorded in the buffer register into an analog quantity.

ÿ Buffer register: record the current converted digital quantity.

The whole process is as follows:

1. Clear the buffer register;

2. Set the highest bit of the successive approximation register to 1;

3. The digital value is sent to the D/A converter, and the analog value converted by D/A is sent to the comparator, which is called Vo;

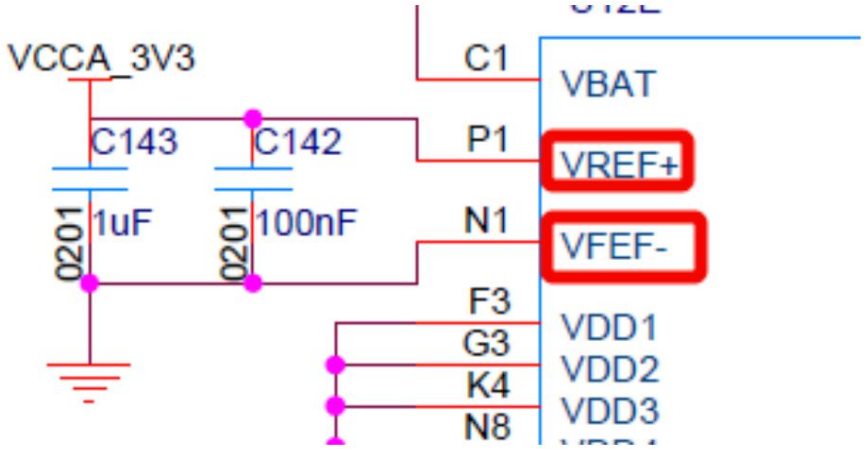
4. Compare Vo with the analog quantity Vi to be converted by the comparator. If Vo<Vi, this bit is reserved, otherwise it is cleared to 0.

5. Set the second highest bit of the register to 1, and send the new digital quantity in the register to the D/A converter,

6. The output Vo is compared with Vi, if Vo<Vi, the bit is reserved, otherwise it is cleared to 0.

7. This process is repeated until the lowest bit of the register is reached, and the digital output is obtained.

In this process, the voltages used by the D/A converter are the power supply voltages Vref+ and Vref- of the stm32, as shown in the figure:



The standard voltage of stm32 is 3.3V. For example, in the first conversion, the output voltage is 1/2Vre, that is, 1.65V. But due to external

The external power supply voltage is not necessarily 3.3V. Therefore, in order to improve the ADC accuracy in this case, stm32 has internal

VERFINT1.2V stable voltage, you can use ADC to sample this voltage to improve ADC accuracy.

7.6 Course Summary

ADC is the process of converting analog quantity into digital quantity, and the method of single-chip microcomputer to obtain analog quantity such as voltage and current sensor. via ADC

function, we can obtain the analog values of various sensors and convert them into digital quantities that can be processed by the microcontroller, or

Get the voltage value of the battery and display it in the form of power.

8. Serial port transceiver

8.1 Knowledge points

ÿ Introduction to serial port

ÿ Serial port configuration in cubeMX

ÿ Serial port receive interrupt and idle interrupt

ÿ Serial port sending function and interrupt function

ÿ APB clock calculates serial port baud rate

8.2 Course content

This lesson will introduce the serial port functions commonly used in microcontrollers. Serial port is a kind of serial port in single chip microcomputer, sensor, execution module, etc.

The commonly used communication interface on the device, in the game, the data sent by the remote control can be read through the serial port, or the serial

The data from sensors such as ultrasonic waves can be read through the port, and the serial port can also be used between the microcontroller and the computer running computer vision.

communication.

Through the study of this lesson, you will master how to calculate the baud rate of the serial port through the APB clock, and the configuration of the serial port in cubeMX.

Set method, serial port receive interrupt and idle interrupt function, serial port send function and send interrupt.

8.3 Basic Learning

8.3.1 Serial port receive interrupt and idle interrupt

The full name of the serial port is the universal serial communication interface, which is a very commonly used communication interface. Serial means that 1 and

0, the data is sent to the receiver one by one in sequence. Universal serial communication interface has the advantages of simple protocol and easy control.

The communication protocol of serial port consists of start bit, data bit, parity bit and end bit. Generally, a low level is used as a frame of data

The beginning of , followed by 8 or 9 data bits, followed by the parity bit, which is divided into odd parity, even parity and no parity, the last

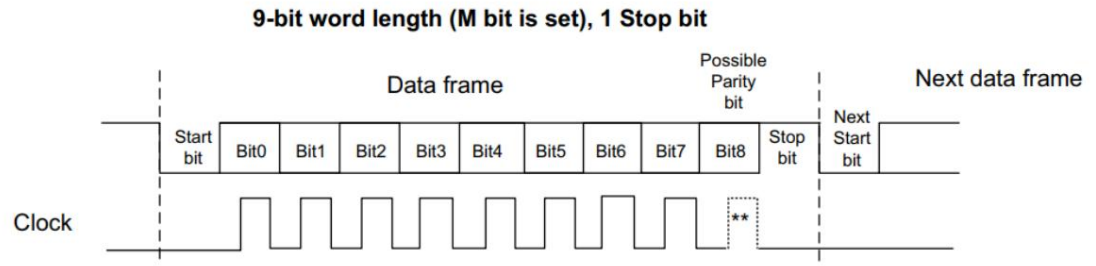
Then a high first and then low pulse is used to indicate the end bit, and the length can be set to 0.5, 1, 1.5 or 2 bits.

The principle of the parity check bit is to count the parity of high level '1' in the sent data, record the result in the parity check bit and send it to

The receiver, after the receiver receives the parity bit, compares it with the data it receives, and accepts the frame if the parity is consistent

data, otherwise it is considered that this frame of data is wrong.

The figure below is a serial port data frame with 8 data bits, 1 parity bit, and 1 stop bit.



data frame content	length	Features
start bit	First place	start of frame
data bits	8 or 9 bits	transfer data
parity bit	1 bit odd parity/even parity, or no parity bit to check whether the data in this frame is correct	
stop bit	0.5, 1, 1.5 or 2 digits	end of frame

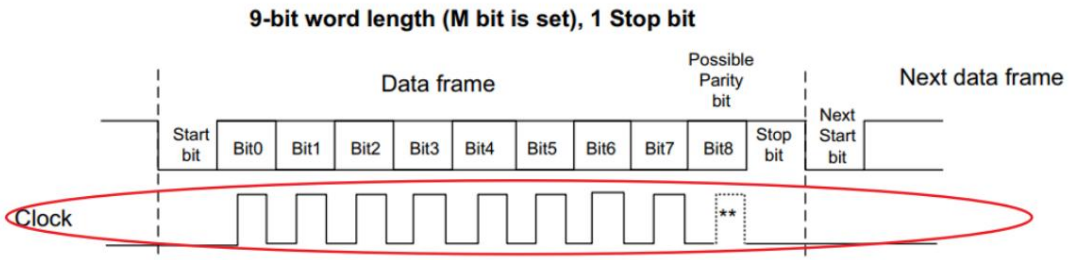
In general, when serial communication is performed, both the sender and the receiver must ensure that they comply with the same protocol in order to correctly complete the sending and receiving.

In addition to the agreement, there is another very important factor to be consistent, and that is the communication rate, that is, the baud rate. baud rate means

The rate of sending data, the unit is baud per second, the commonly used baud rates of serial ports are 115200, 38400, 9600 and so on. string

The baud rate of the port and the bus clock cycle (clock) have an inverse relationship, that is, the shorter the bus clock cycle, the more

The greater the number of symbols, the higher the serial port baud rate.



8.4 Program Learning

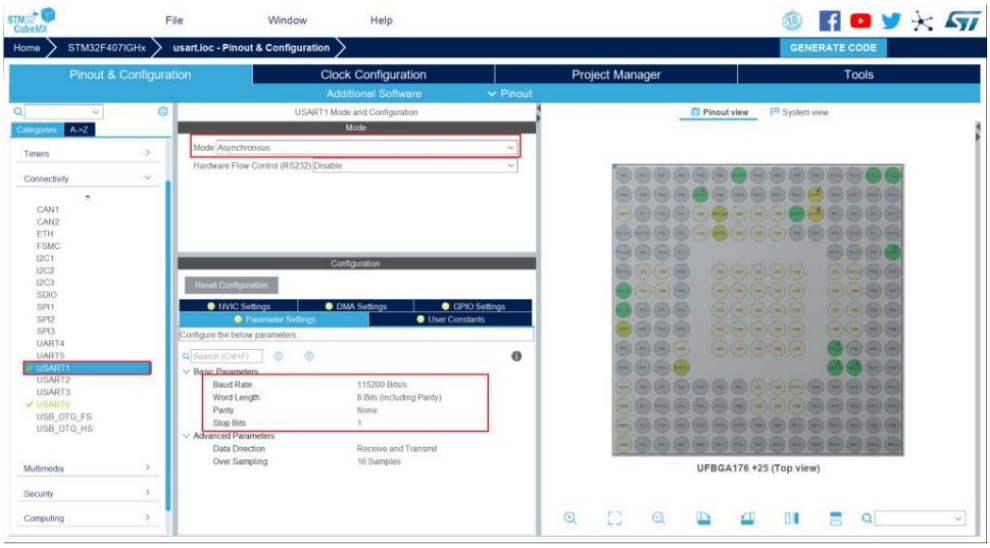
8.4.1 Serial port configuration in cubeMX

The configuration process of serial port in cubeMX is as follows:

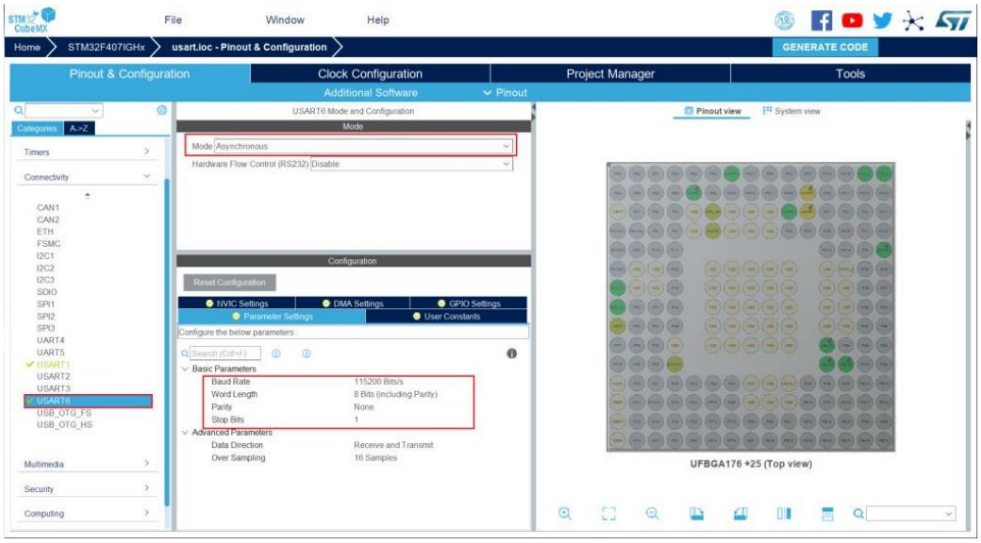
1. First open USART1 under the Connectivity tab and set its Mode to Asynchronous

communication method. Asynchronous communication is a communication method that does not rely on a synchronous clock signal between the sender and the receiver.

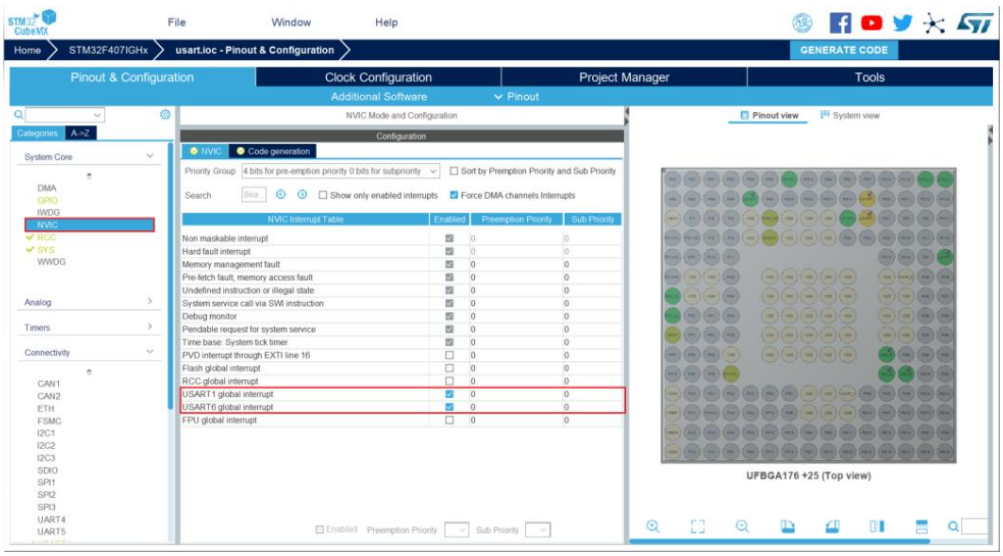
2. Then set the baud rate to 115200, the data frame to 8 data bits, no parity bit, and 1 stop bit.



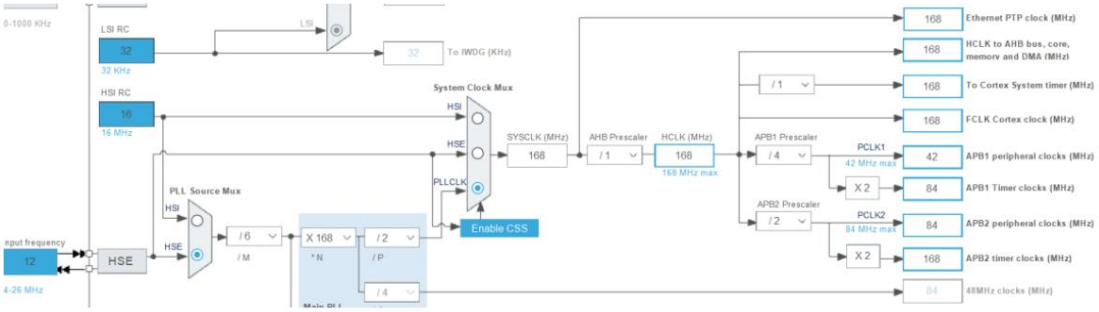
3. Similarly, turn on USART6 and set it up in the same way as USART1.



4. Then go to the NVIC tab and enable the USART1 and USART6 interrupts.



The engineering clock configuration is as follows:



5. Click Generate Code to generate the project.

8.4.2 Serial port receive interrupt and idle interrupt

This section introduces the receive interrupt and idle interrupt of the serial port, both of which may occur when the serial port is receiving.

break.

	receive interrupt	idle interrupt
handler function	USARTx_IRQHandler	USARTx_IRQHandler
Callback	HAL_UART_RxCpltCallback HAL library not provided	
in the USART status register Rank	UART_FLAG_RXNE	UART_FLAG_IDLE
Triggering conditions	Triggered after receiving a frame of data one interruption	After the serial port receives a frame of data, it passes a byte was not received at a time to any data

The serial port receive interrupt is to trigger an interrupt every time the serial port completes a receive. Corresponding interrupt handler in STM32

The number is USARTx_IRQHandler, and the interrupt callback function is HAL_UART_RxCpltCallback. able to pass

The UART_FLAG_RXNE bit in the USART status register determines whether a receive interrupt has occurred on the USART.

The serial port is idle interrupt, that is, whenever the serial port receives a frame of data and no data is received after one byte of time

Trigger an interrupt, the interrupt handling function is also USARTx_IRQHandler, which can be accessed through the USART status register

The UART_FLAG_IDLE in the UART_FLAG_IDLE determines whether an idle interrupt has occurred.

8.4.3 Serial port sending function and interrupt function

This section will introduce the send function and interrupt function of the serial port.

The HAL library provides the serial port sending function HAL_UART_Transmit, through which the serial port can be sent from the specified serial port

data.

HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)	
Function name	HAL_UART_Transmit
function	Send a piece of data from the specified serial port
return value	HAL_StatusTypeDef, several states defined by the HAL library, if this is sent successfully secondary data, return HAL_OK
parameter 1	UART_HandleTypeDef *huart The handle pointer of the serial port to be sent, such as For serial port 1, input &huart1, for serial port 2, input &huart2
parameter 2	uint8_t *pData The first address of the data to be sent, such as to send buf[]="Helloword" then enter buf, or you can directly enter the string to be sent
parameter 3	uint16_t Size The size of the data to be sent, that is, the length of the input string, also The size of the data can be obtained through the sizeof keyword
parameter 4	uint32_t Timeout send timeout time, if sending time exceeds this time, take cancel this send

After the transmission completion interrupt is enabled, whenever a serial port transmission is completed, the serial port will trigger a transmission completion interrupt, and the corresponding interrupt

The callback function is HAL_UART_TxCpltCallback.

8.4.4 Program flow

In this sample code, instead of writing code in the interrupt callback function, the interrupt handler function is directly modified

USARTx_IRQHandler. Therefore, after an interrupt occurs, it will directly enter the interrupt handler to execute user code.

First, initialize the receive interrupt and idle interrupt of serial port 1 and serial port 6 during initialization, and pass it in the main loop.

The HAL_UART_Transmit function completes the data transmission of serial port 1 and serial port 6, and the transmission content is "RoboMaster\r\n"

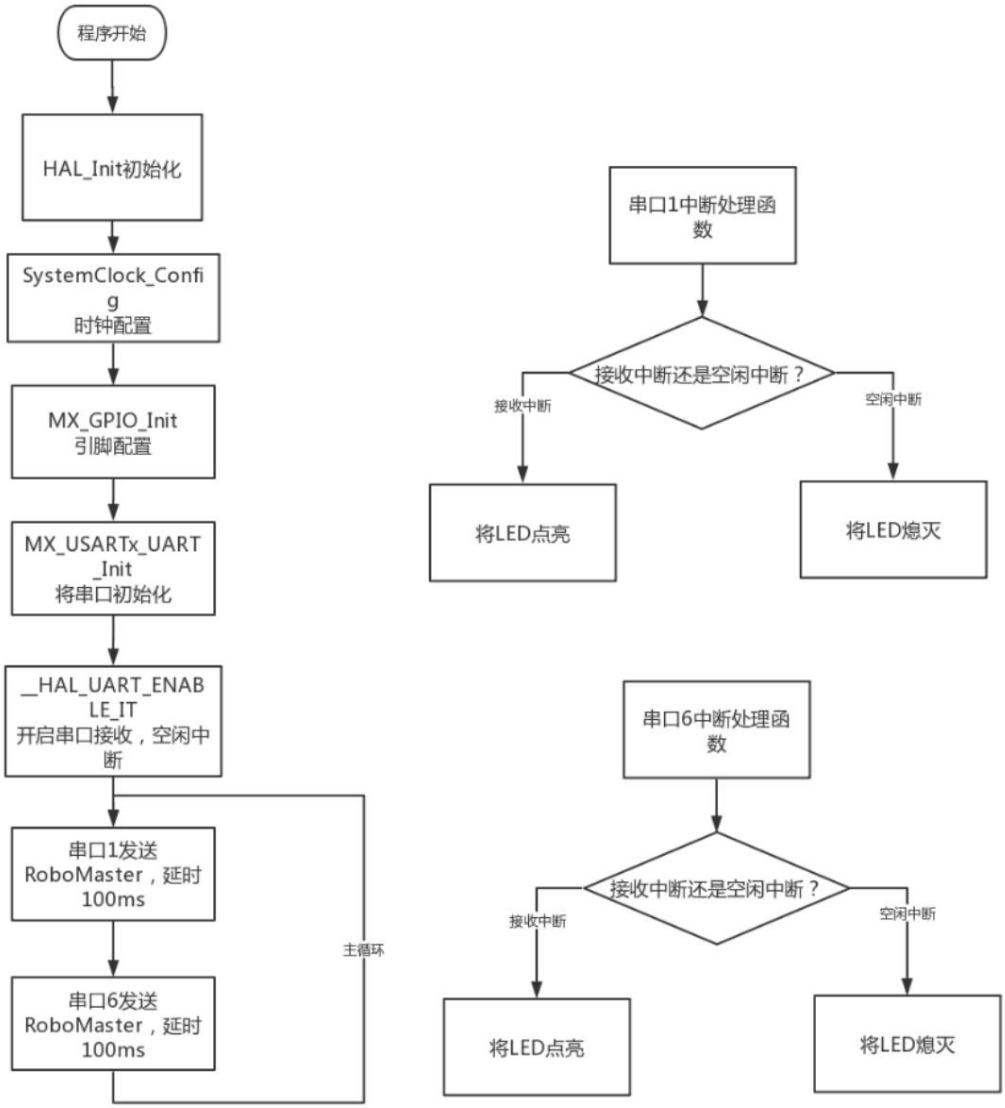
this string.

When the serial port has a receive interrupt or an idle interrupt, it will enter the USARTx_IRQHandler interrupt processing function. in

In the interrupt processing function, the status register of the serial port is used to determine whether the interrupt is a receive interrupt or an idle interrupt.

If the interrupt is received, flip the level of the red LED to turn it on, and if it is an idle interrupt, turn it off.

The program flow chart is as follows:

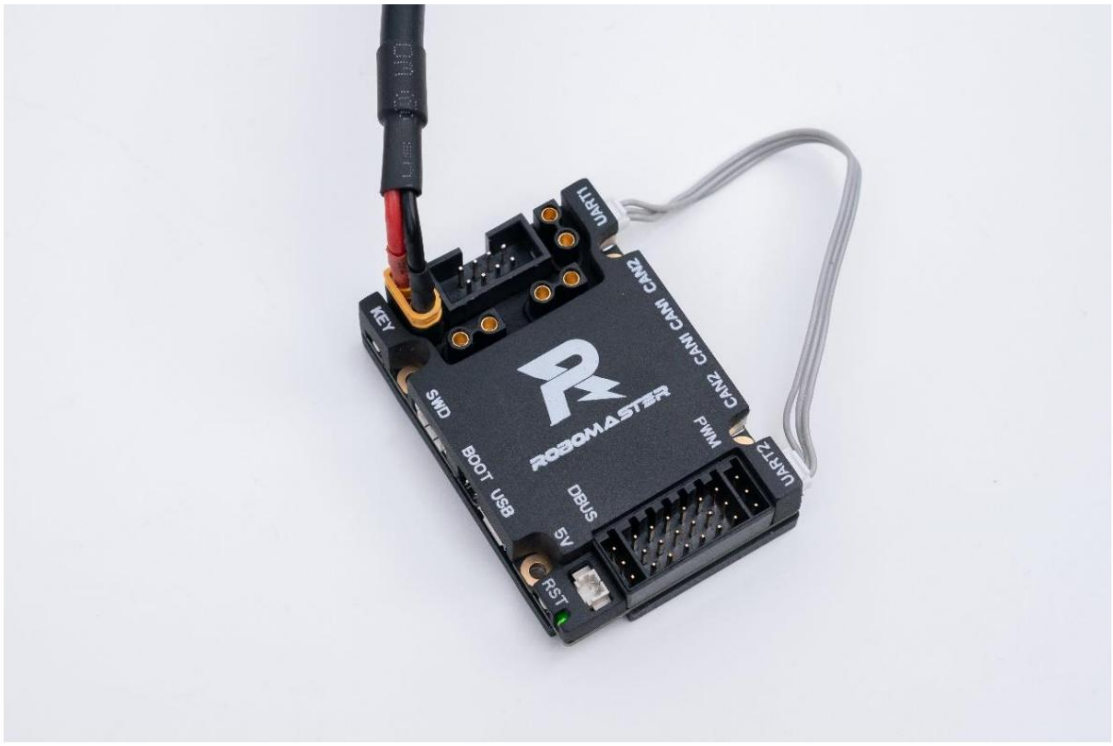


8.4.5 Effect display

When the serial port 1 receives the data reception interrupt, the LED red light will light up, and when the serial port 1 enters the idle interrupt, the LED red light will be turned on.

The light is off; in the same way, when serial port 6 receives data reception interruption, the green LED light will be on, and when serial port 1 enters idle state

If it is disconnected, the green LED light will be off, and the receiving effect is as shown in the figure.



green light on

red light on

8.5 Advanced Learning

8.5.1 APB clock calculates serial port baud rate

This summary will learn how to calculate the baud rate of the serial port from the APB clock. When calculating the baud rate of the serial port, you must first pass the

The method described in the timer chapter, through the datasheet to find out whether the bus corresponding to the serial port used is APB1 or APB2,

Then find the speed of the corresponding bus in the cube.

After obtaining the bus rate, the communication of the serial port can be calculated by the USARTDIV value in the USART_BRR register

rate. In this register, the first 12 bits are used to represent the integer part of USARTDIV, and the last 4 bits are used to represent the fractional part

cents, the fractional part is multiplied by 16, rounded up, and stored in the last 4 digits.

The formula for calculating the baud rate using USARTDIV is:

Tx / Rx baud rate = fPCLKx/(16*USARTDIV)

It is found in the data sheet that USART1 is connected to the APB1 bus, and the APB1 rate PCLK1 is 42MHz. And expect

The serial communication rate of USART1 is 115200.

1. Through the above baud rate calculation formula, the USARTDIV value is calculated as 42MHz/115200/16=22.786;

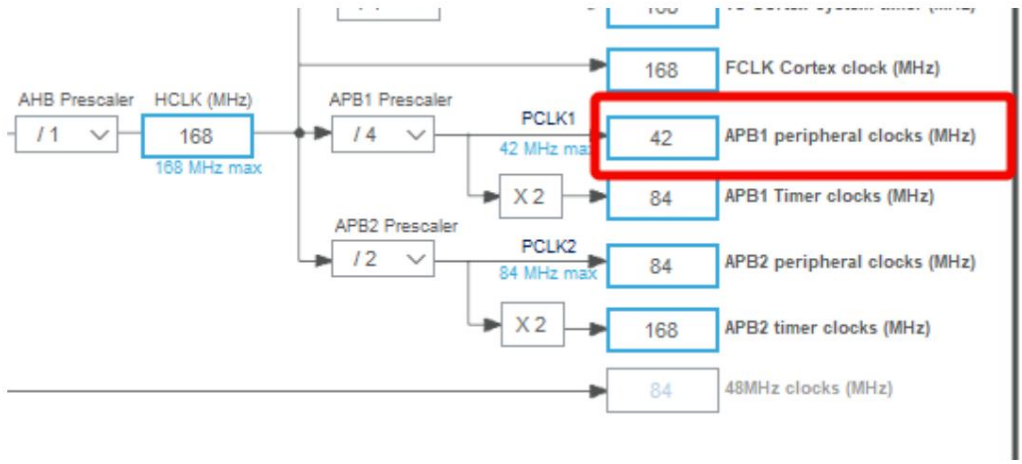
2. Convert the integer 22 to hexadecimal 0x16, and the decimal 0.786*16=12.576 (rounded to 12) is expressed in hexadecimal as

0x0C, so the value stored in USART_BRR ends up being 0x16C.

Since some approximate means of rounding and discarding the mantissa are taken here, the baud rate generated by the register and the final expectation

There is an error between the baud rates, generally a small error will not affect the final serial communication, but if the

If there is a problem, remember to check whether it is related to the large difference between the actual value of the communication speed and the ideal value.



8.6 Course Summary

This lesson learned the serial port function of STM32. The serial port is a communication interface used on many devices.

information so that we can pass information on these devices. During the game, the data sent by the remote control can be read through the serial port,

You can also read the data of ultrasonic sensors and other sensors through the serial port, or you can use the serial port in the microcontroller and the computer of the self-aiming program.

communication between the brains.

9. Serial print remote control data

9.1 Knowledge points

ÿ DMA function introduction

ÿ Serial port DMA receive and transmit configuration

ÿ DBUS protocol introduction

ÿ printf function implementation

ÿ Serial port DMA receive and transmit

9.2 Course content

This course will introduce the DMA function of STM32 serial port. DMA is a commonly used function when using serial port for communication.

Yes, using this function can complete the direct data transfer between the serial port and the memory without the need for CPU control, thus

Save CPU processing time.

Learn how to read the data of the remote control through the DMA function through the experiment, and then learn how to use the STM32

Implement the printf function that uses DMA for serial output, and use it to transmit the data from the remote control to the serial tool.

9.3 Basic Learning

9.3.1 DMA function introduction

The full name of DMA is Direct Memory Access (Direct Memory Access).

When stored in the memory, if the DMA method is not used, the external device data needs to be read into the CPU first, and then

The CPU stores data in memory, and if the amount of data is large, it will take up a lot of CPU time, and

By using the DMA controller to directly transfer the external device data into the memory, it does not need to occupy the CPU.

Many communications in STM32, such as USART, SPI, and IIC, support DMA mode for data transmission and reception.

9.3.2 Introduction of DBUS Protocol

The communication between the remote control and stm32 adopts DBUS protocol. DBUS communication protocol is similar to serial port, DBUS transmission

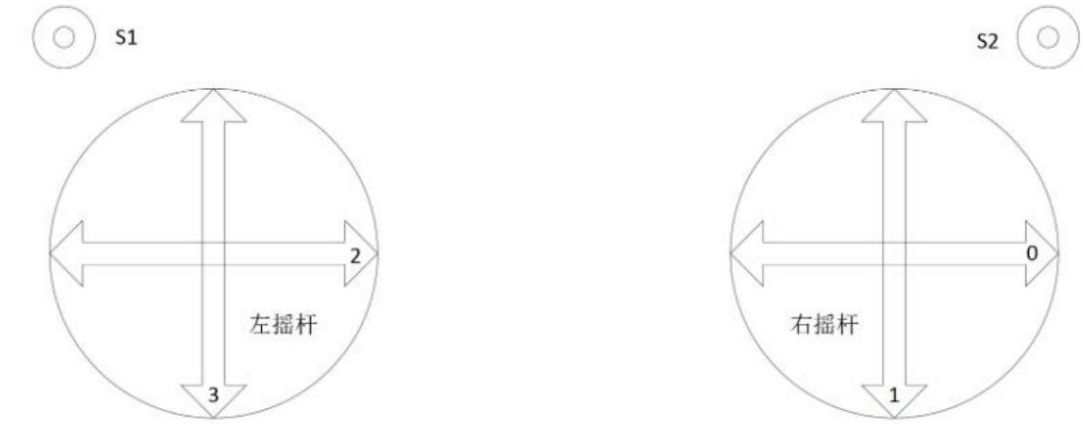
The transmission rate is 100k bit/s, the data length is 8 bits, the parity bit is even, and the end bit is 1 bit. have to be aware of is

The level standard used by DBUS is opposite to that of the serial port. In the DBUS protocol, high level means 0, and low level means 1, such as

If the serial port is used for receiving, an inverter needs to be added to the receiving circuit.

Use DBUS to receive data from the remote control, the length of one frame of data is 18 bytes, a total of 144 bits, according to the remote control

The manual can find out the meaning of each segment of data, so as to perform data splicing and complete the decoding of the remote control, as shown in the figure.



域	通道 0	通道 1	通道 2	通道 3	S1	S2
偏移	0	11	22	33	44	46
长度(bit)	11	11	11	11	2	2
符号位	无	无	无	无	无	无
范围	最大值 1684 中间值 1024 最小值 364	最大值 1684 中间值 1024 最小值 364	最大值 1684 中间值 1024 最小值 364	最大值 1684 中间值 1024 最小值 364	最大值 3 最小值 1	最大值 3 最小值 1
功能	无符号类型 遥控器通道 0	无符号类型 遥控器通道 1	无符号类型 遥控器通道 2	无符号类型 遥控器通道 3	遥控器发射机 S1 开关位	遥控器发射机 S2 开关位置

域	鼠标 X 轴	鼠标 Y 轴	鼠标 Z 轴	鼠标左键	鼠标右键	按键
偏移	48	64	80	96	104	112
长度	16	16	16	8	8	16
符号位	有	有	有	无	无	无

9.4 Program Learning

9.4.1 DMA configuration for serial transmission

First open USART1 and USART3 and configure them, in which USART1 enables the DMA transmission of the serial port, which is used for

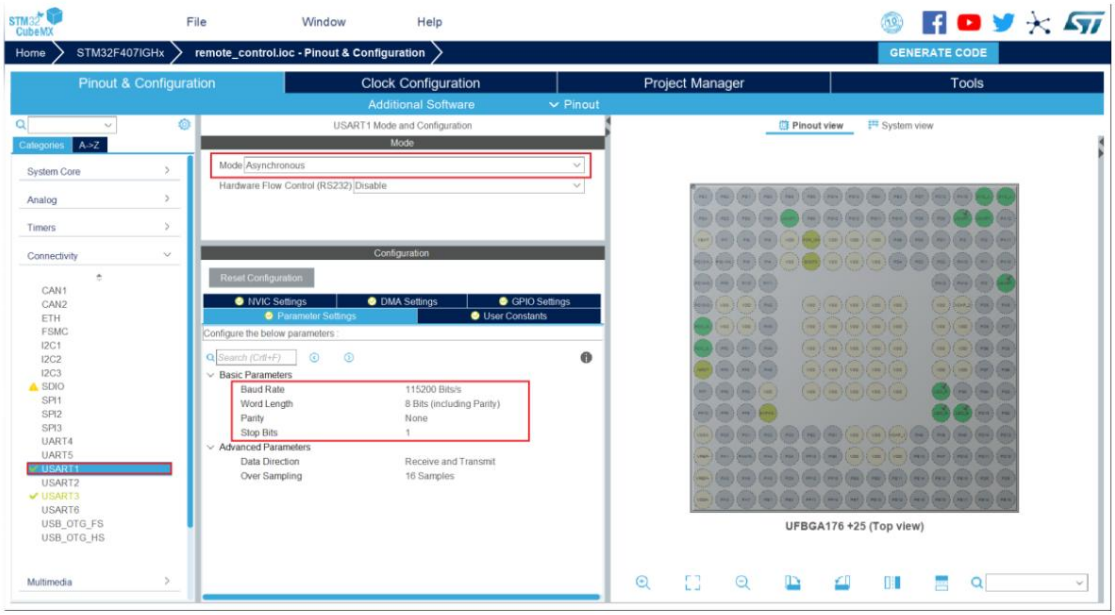
Serial port tool for data transmission PC, USART3 enables DMA reception of serial port, which is used for remote control data reception; configuration

as follows:

1. Open USART1 under the Connectivity tab and set its Mode to Asynchronous

   Way. Asynchronous communication is a communication method that does not rely on a synchronous clock signal between the sender and the receiver.

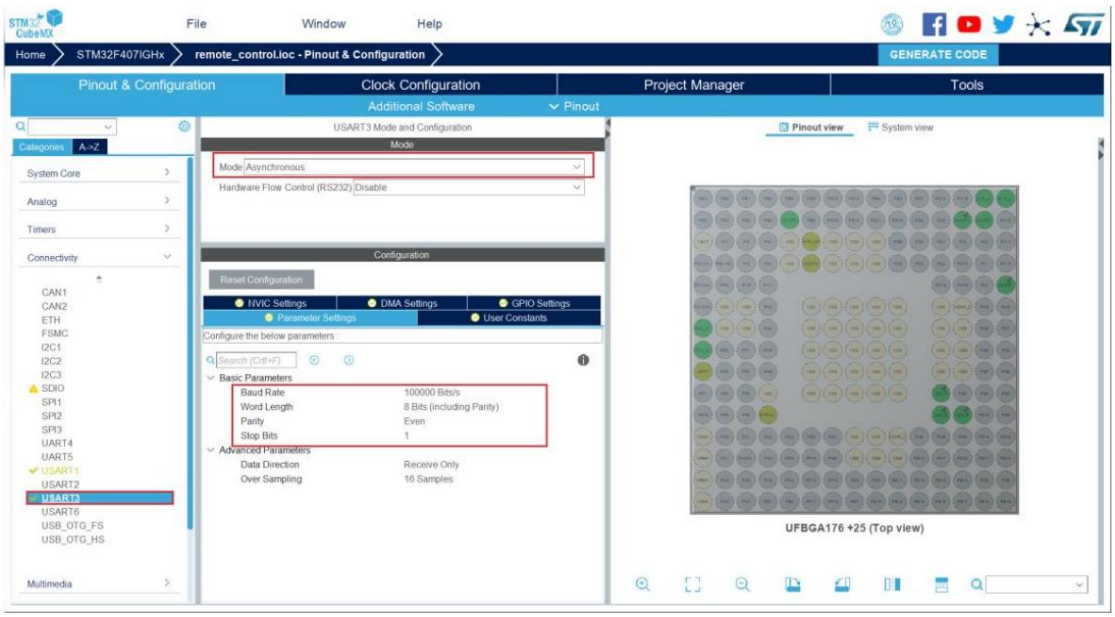
2. Set its baud rate to 115200, the data frame to 8 data bits, no parity bit, and 1 stop bit.



3. Open USART3 under the Connectivity tab and set its Mode to Asynchronous communication

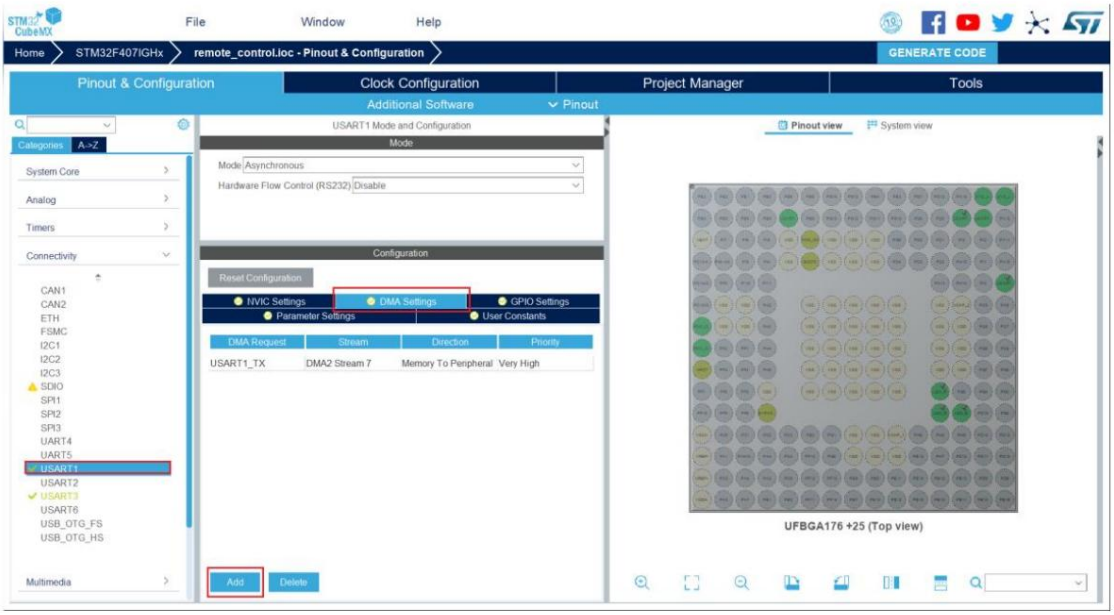
Way.

4. Set its baud rate to 100000, and set the data frame to 8 data bits, no parity bit, and 1 stop bit.



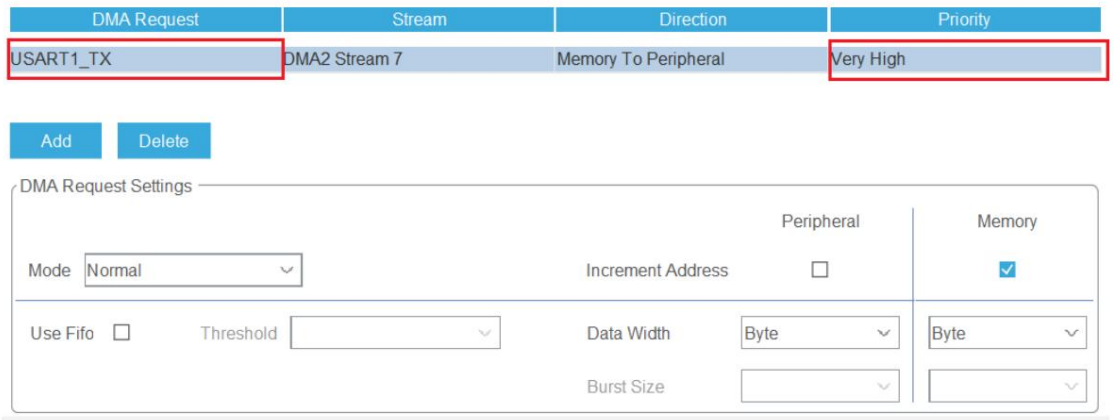
5. Then enable the DMA function of USART1 and USART3 respectively. Click on the setting page of USART1, open

On the DMA Settings tab, click Add.



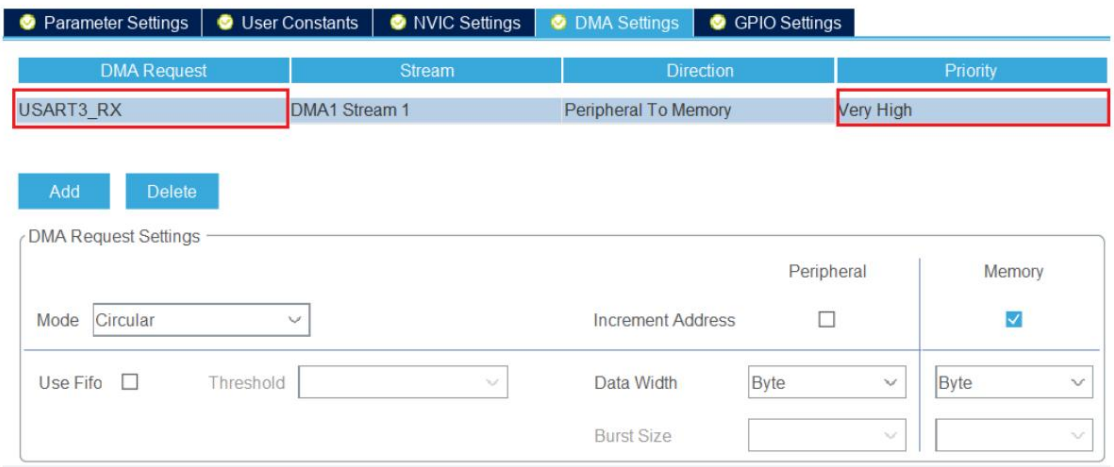
6. In the new entry that pops up, select DMA Request as USART1_TX, the data flows from the memory to the peripheral,

   Priority is selected as Very High.



7. Similarly, find the DMA Settings tab under USART3, and select DMA Request in USART3.

   For USART3_RX, data flows from peripheral to memory, and Priority is selected as Very High.



Through the above settings, the DMA settings of the two serial ports in cubeMX are completed.

9.4.2 Implementation process of **printf** function

Use the va_start function and the vsprintf function under stdarg.h to cooperate with the DMA transmission function of the serial port to realize the C language printf in . Through the operation of the above functions, the content of the data to be sent is stored in tx_buf, and the data to be sent is stored in tx_buf. The length is stored in the len variable, and then the first address of tx_buf and the data length len are passed to the DMA sending function.

Cost of DMA data transmission.

```
void usart_printf(const char *fmt,...)
{
        static uint8_t tx_buf[256] = {0};
        static va_list ap;
        static uint16_t len;
        va_start(ap, fmt);

        //return length of string
        //return the length of the string
        len = vsprintf((char *)tx_buf, fmt, ap);

        va_end(ap);

        usart1_tx_dma_enable(tx_buf, len);

}
```

9.4.3 Serial port **DMA** receiving and sending configuration

In this experiment, the DMA receiving function of USART3 is used to receive remote control data.

The DMA reception of USART3 is initialized through the function remote_control_init. During initialization, enable DMA Serial port reception and idle interrupt, configure the buffer stored when the peripheral data arrives, and enable the double buffer function here Yes, each frame of sbus data is 18 bytes, and the total size of the open double buffer is 36 bytes, which can avoid DMA

Transmission out of bounds.


```
void RC_init(uint8_t *rx1_buf, uint8_t *rx2_buf, uint16_t dma_buf_num)
{
    //enable the DMA transfer for the receiver request
    //Enable DMA serial port receive
    SET_BIT(huart3.Instance->CR3, USART_CR3_DMAR);

    //enalbe idle interrupt
    //enable idle interrupt
    __HAL_UART_ENABLE_IT(&huart3, UART_IT_IDLE);

    //disable DMA
    //fail DMA
    __HAL_DMA_DISABLE(&hdma_usart3_rx);
    while(hdma_usart3_rx.Instance->CR & DMA_SxCR_EN)
    {
        __HAL_DMA_DISABLE(&hdma_usart3_rx);
    }

    hdma_usart3_rx.Instance->PAR = (uint32_t) & (USART3->DR);
    //memory buffer 1
    //memory buffer 1
    hdma_usart3_rx.Instance->M0AR = (uint32_t)(rx1_buf);
    //memory buffer 2
    //memory buffer 2
    hdma_usart3_rx.Instance->M1AR = (uint32_t)(rx2_buf);
    //data length
    //Data length
    hdma_usart3_rx.Instance->NDTR = dma_buf_num;
    //enable double memory buffer
    //enable double buffer
```

```
    SET_BIT(hdma_usart3_rx.Instance->CR, DMA_SxCR_DBM);

    //enable DMA
    //enable DMA
    __HAL_DMA_ENABLE(&hdma_usart3_rx);

}
```

After completing the initialization, whenever USART3 generates an idle interrupt, it will enter USART3_IRQHandler for processing.

In the USART3_IRQHandler, the processing of the register interrupt flag bit is performed, and then the buffer for receiving is judged.

Whether the buffer is buffer No. 1 or buffer No. 2, use the set length to subtract the remaining length to obtain the number obtained by this DMA.

The length of the data, judge whether it is equal to the length of a frame of data (18 bytes), if it is equal, call the function sbus_to_rc to enter

Decoding of remote control data.

```
void USART3_IRQHandler(void)
{
    if(huart3.Instance->SR & UART_FLAG_RXNE)//Receive data
    {
        __HAL_UART_CLEAR_PEFLAG(&huart3);
    }
    else if(USART3->SR & UART_FLAG_IDLE)
    {
        static uint16_t this_time_rx_len = 0;

        __HAL_UART_CLEAR_PEFLAG(&huart3);

        if ((hdma_usart3_rx.Instance->CR & DMA_SxCR_CT) == RESET)
        {
            /* Current memory buffer used is Memory 0 */

            //disable DMA
            //fail DMA
            __HAL_DMA_DISABLE(&hdma_usart3_rx);
```



```
        //get receive data length, length = set_data_length - remain_length
        //Get the length of the received data, length = set length - remaining length
        this_time_rx_len = SBUS_RX_BUF_NUM - hdma_usart3_rx.Instance->NDTR;

        //reset set_data_lenght
        //reset the data length
        hdma_usart3_rx.Instance->NDTR = SBUS_RX_BUF_NUM;

        //set memory buffer 1
        //set buffer 1
        hdma_usart3_rx.Instance->CR |= DMA_SxCR_CT;

        //enable DMA
        //enable DMA
        __HAL_DMA_ENABLE(&hdma_usart3_rx);

        if(this_time_rx_len == RC_FRAME_LENGTH)
        {
            sbus_to_rc(sbus_rx_buf[0], &rc_ctrl);
        }
    }
    else
    {
        /* Current memory buffer used is Memory 1 */
        //disable DMA
        //fail DMA
        __HAL_DMA_DISABLE(&hdma_usart3_rx);

        //get receive data length, length = set_data_length - remain_length
```

```
                //Get the length of the received data, length = set length - remaining length
                this_time_rx_len = SBUS_RX_BUF_NUM - hdma_usart3_rx.Instance->NDTR;

                //reset set_data_lenght
                //reset the data length
                hdma_usart3_rx.Instance->NDTR = SBUS_RX_BUF_NUM;

                //set memory buffer 0
                //set buffer 0
                DMA1_Stream1->CR &= ~(DMA_SxCR_CT);

                //enable DMA
                //enable DMA
                __HAL_DMA_ENABLE(&hdma_usart3_rx);

                if(this_time_rx_len == RC_FRAME_LENGTH)
                {
                    //Process remote control data
                    sbus_to_rc(sbus_rx_buf[1], &rc_ctrl);
                }
            }
        }
}
```

The function of the remote control data processing function sbus_to_rc is to convert the raw data obtained through DMA according to the

The data protocol is spliced into the complete remote control data. Taking the data of channel 0 as an example, find the channel from the user manual of the remote control.

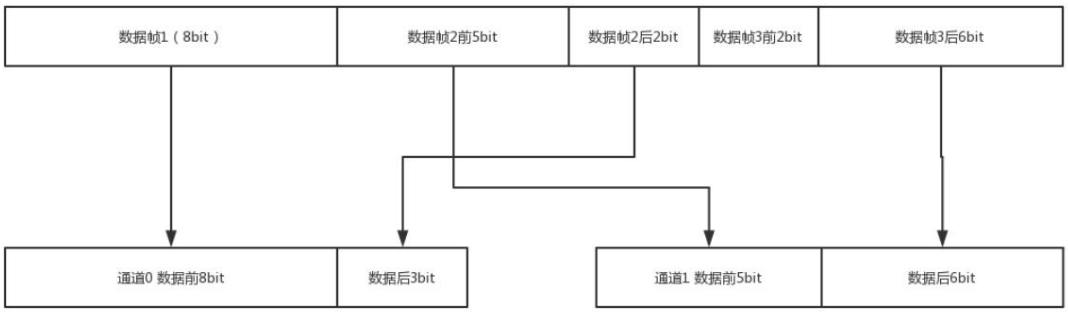
The length of 0 is 11 bits, and the offset is 0.

域	**通道 0**
偏移	0
长度(bit)	11
符号位	无
范围	最大值 1684 中间值 1024 最小值 364
功能	无符号类型 遥控器通道 0

This means that if you want to get the data of channel 0, you need to combine the 8bit data of the first frame and the last three bits of the second frame data

Splicing, if you want to get the data of channel 1, put the first 5 bits of the second frame data and the last 6 bits of the third frame data into

Line splicing, all data frames can be obtained through continuous splicing. The schematic diagram of the splicing process is as follows:



The decoding function sbus_to_rc completes the above data splicing work by means of bit operation, the second of the hexadecimal number 0x07ff

The base is 0b0000 0111 1111 1111, which is the 11-bit 1, and the AND operation with 0x07ff is equivalent to truncating 11

bits of data.

Data acquisition of channel 0: first perform OR operation on data frame 1 and data frame 2 shifted by 8 bits to the left, and splicing out a 16-bit number.

According to the data, the first 8 bits are the data frame 2, the last 8 bits are the data frame 1, and then add it with 0x07ff, intercept 11 bits, and get

Channel 0 data spliced by the last 3 bits of data frame 2 and data frame 1. The schematic diagram of the process is as follows:

Through the above method, you can obtain the data values of each channel and switch of the remote control, as well as the keyboard and mouse.

```c
static void sbus_to_rc(volatile const uint8_t *sbus_buf, RC_ctrl_t *rc_ctrl)
{
    if (sbus_buf == NULL || rc_ctrl == NULL)
    {
        return;
    }

    rc_ctrl->rc.ch[0] = (sbus_buf[0] | (sbus_buf[1] << 8)) & 0x07ff;          //!< Channel 0
    rc_ctrl->rc.ch[1] = ((sbus_buf[1] >> 3) | (sbus_buf[2] << 5)) & 0x07ff;   //!< Channel 1
    rc_ctrl->rc.ch[2] = ((sbus_buf[2] >> 6) | (sbus_buf[3] << 2) |            //!< Channel 2
                         (sbus_buf[4] << 10)) &0x07ff;
    rc_ctrl->rc.ch[3] = ((sbus_buf[4] >> 1) | (sbus_buf[5] << 7)) & 0x07ff;   //!< Channel 3
    rc_ctrl->rc.s[0] = ((sbus_buf[5] >> 4) & 0x0003);                         //!< Switch left
    rc_ctrl->rc.s[1] = ((sbus_buf[5] >> 4) & 0x000C) >> 2;                    //!< Switch right
    rc_ctrl->mouse.x = sbus_buf[6] | (sbus_buf[7] << 8);                      //!< Mouse X axis
    rc_ctrl->mouse.y = sbus_buf[8] | (sbus_buf[9] << 8);                      //!< Mouse Y axis
    rc_ctrl->mouse.z = sbus_buf[10] | (sbus_buf[11] << 8);                    //!< Mouse Z axis
    rc_ctrl->mouse.press_l = sbus_buf[12];                                    //!< Mouse Left Is Press ?
    rc_ctrl->mouse.press_r = sbus_buf[13];                                    //!< Mouse Right Is Press ?
    rc_ctrl->key.v = sbus_buf[14] | (sbus_buf[15] << 8);                      //!< KeyBoard value
    rc_ctrl->rc.ch[4] = sbus_buf[16] | (sbus_buf[17] << 8);                   //NULL

    rc_ctrl->rc.ch[0] -= RC_CH_VALUE_OFFSET;
    rc_ctrl->rc.ch[1] -= RC_CH_VALUE_OFFSET;
    rc_ctrl->rc.ch[2] -= RC_CH_VALUE_OFFSET;
    rc_ctrl->rc.ch[3] -= RC_CH_VALUE_OFFSET;
    rc_ctrl->rc.ch[4] -= RC_CH_VALUE_OFFSET;
}
```

Then use USART1 to send by DMA, and send the received remote control data. first through

The usart1_tx_dma_init function initializes the dma transmission. In the main loop, the usart_print function is called, and the

The decoded remote control data is sent from USART1 using DMA.

9.4.4 Program flow

The flow of this program is to initialize the DMA transmission of USART1 and the DMA reception of USART3 during initialization.

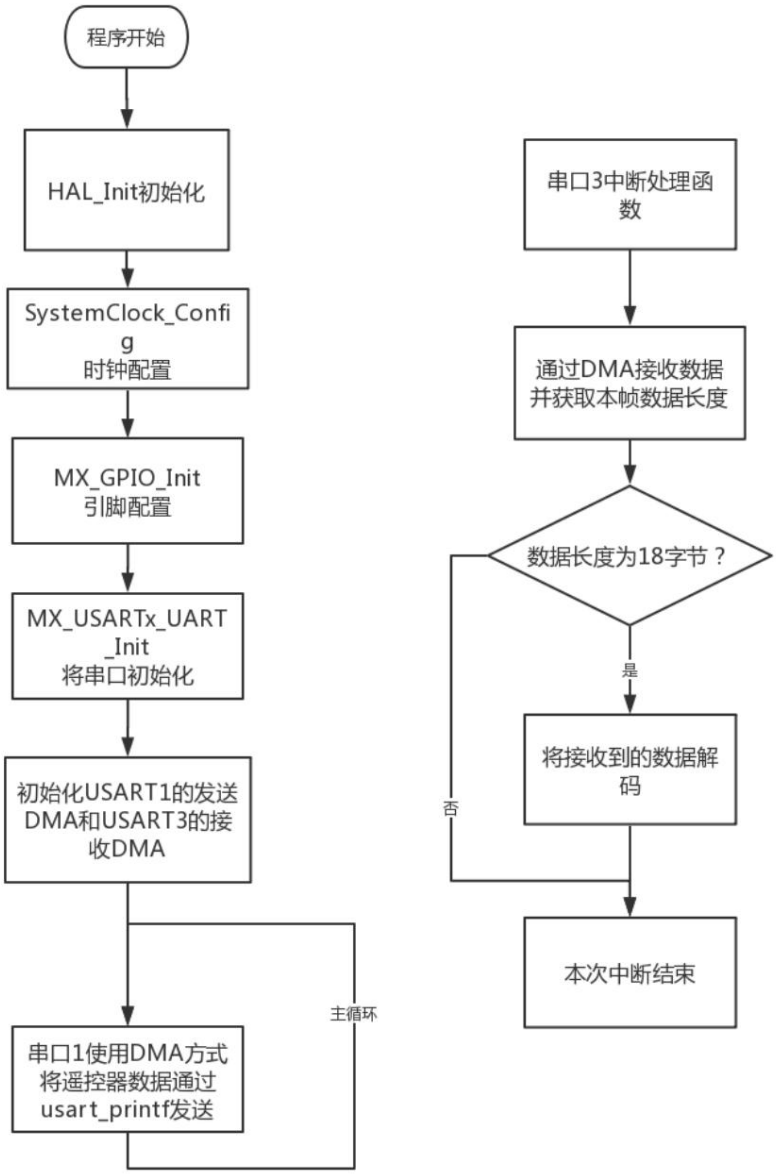
Initialize, then use DMA to receive data from the remote control in the serial port receive interrupt of USART3, and use the decoding function

Decode the data.

Then call the usart_printf function implemented by the serial port in the main loop, and pass the decoded remote control function through USART1

DMA send function to send out.

The program flow chart is as follows:



9.4.5 Effect display

The remote control receiver is connected to the DBUS port on the development board C type, and the remote control is viewed in the debug window and serial port tool.

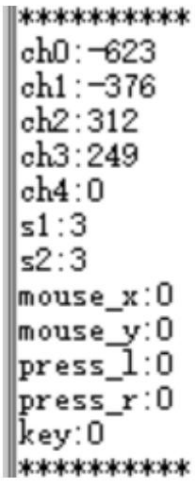
The channel value of the device is shown in the figure.

Remote control wiring diagram

Watch 1

Name	Value	Type
rc_ctrl	0x20000260 &rc_ctrl	struct <untagged>
rc	0x20000260 &rc_ctrl	struct <untagged>
ch	0x20000260 &rc_ctrl	short[5]
[0]	-306	short
[1]	126	short
[2]	307	short
[3]	381	short
[4]	0	short
s	0x2000026A "□□"	char[2]
[0]	3	char
[1]	3	char
mouse	0x2000026C	struct <untagged>
x	0	short
y	0	short
z	0	short
press_l	0	unsigned char
press_r	0	unsigned char
key	0x20000274	struct <untagged>
v	0	unsigned short

Program debugging remote control data diagram



Serial port tool receiving remote control data diagram

9.5 Course Summary

This lesson learns the DMA sending and receiving function of the serial port. The DMA send and receive function enables the CPU to efficiently complete the data

In addition, this lesson also learned the reception and decoding of the remote control, as well as the use of the serial port to realize the printf function. remote

The controller is the most important human-computer interaction device of the RM robot, which is used for the control input of the robot. The Printf function is a standardized output

The outgoing functions are mainly used for debugging functions.

10. Flash read and write

10.1 Key points of knowledge

ÿ Flash introduction of stm32

ÿ The boot function of stm32

ÿ Introduction to flash erase

ÿ flash write and read

10.2 Course Content

This lesson will learn about the stm32's flash (flash) peripherals. flash is used to store data, and the machine is often used in RM competitions

The robot characteristic data (such as the median value of the gimbal motor) is saved in the flash. In this course, learn stm32 to flash data

read and write, write RoboMaster characters into flash, and read written string data from flash.

10.3 Basic Learning

10.3.1 Flash introduction of stm32

All personal computers have memory and hard disk (external memory), and the development board chip stm32 also has SRAM with memory of 192Kbytes

And 1Mbytes of external memory flash. Flash and SRAM have the following characteristics.

FLASH	SHAME
large capacity	small capacity
Slower reading and writing	Faster reading and writing
No data loss in case of power loss	Lost data due to power failure
sector	no sector
Write needs to be erased first	Write without erasing
Stored programs, long-term saved data, etc.	variables etc.

For the use of flash, you need to understand the following two points:

1. In order to protect the security of data, flash has a special lock register. Every time you want to modify the flash page, first

To unlock the page through the lock register, lock it after the modification is completed.

2. The flash does not support modification while saving the original data, so when changing the flash page data, it is necessary to

To erase this page, write new data after erasing.

10.4 Program Learning

10.4.1 Introduction of flash erase function

This section will introduce the erase function of flash. The HAL library provides the flash erase function HAL_FLASHEx_Erase.

HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * SectorError)	
Function name	HAL_FLASHEx_Erase
Function to erase the specified flash page	
return value	HAL_StatusTypeDef, several states defined by the HAL library, if the flash page erase is successfully completed Otherwise, it will return HAL_OK, if it fails, it will return HAL_ERROR, if it times out, it will return HAL_TIMEOUT
parameter 1	FLASH_EraseInitTypeDef *pEraseInit The structure pointer used when erasing the flash. To create a structure flash_erase of type FLASH_EraseInitTypeDef, you need to assign Give this structure the following parameters: Sector (the first address of the page to be erased), TypeErase (erase division method), VoltageRange (voltage range), NbSectors (number of pages to be erased), the most Then we pass &flash_erase as a parameter to the function
parameter 2	uint32_t *SectorError If there is an error in this flash erasing, an erasing error will occur. The page number is stored in SectorError. Create a uint32_t error, enter &error as Yes, if there is an error in erasing, you can read the value in error to confirm the page number in error.

10.4.2 Introduction to flash write function

This section will introduce the write function of flash, HAL library provides functions to write data to flash

HAL_FLASH_Program

HAL_StatusTypeDef HAL_FLASH_Program(uint32_t TypeProgram, uint32_t Address, uint64_t Data)	
Function name	HAL_FLASH_Program

function	Write data to a page in flash in a specified way
return value	HAL_StatusTypeDef, several states defined by the HAL library, if completed successfully to the flash If the data is written, it will return HAL_OK, if it fails, it will return HAL_ERROR, and if it times out, it will return HAL_TIMEOUT
parameter 1	uint32_t TypeProgram Select the data format to write, you can select 8-bit byte FLASH_TYPEPROGRAM_BYTE ÿ 16-bit halfword FLASH_TYPEPROGRAM_HALFWORD ÿ 32-bit word FLASH_TYPEPROGRAM_WORD , or 64-bit double word FLASH_TYPEPROGRAM_DOUBLEWORD
parameter 2	uint32_t Address The address where data needs to be written
parameter 3	uint64_t Data data to be written

10.4.3 Introduction to flash reading

The reading of flash is relatively simple and can be read through the memcpy function

```
void flash_read(uint32_t address, uint32_t *buf, uint32_t len)

{

        memcpy(buf, (void*)address, len *4);

}
```

Function name	flash_read
function	read data from flash
return value	None
parameter 1	Flash address
parameter 2	The address of the stored variable after reading

parameter 3	length in bytes

10.4.4 Flash related operation functions

This section will continue to introduce the relevant operation functions of the flash, mainly the unlocking and locking functions of the flash.

As introduced in the principle part, it is necessary to unlock and lock the flash when writing or erasing data. First introduce the solution of flash

Lock function HAL_FLASH_Unlock. The function of this function is actually very simple, just write a specific

KEY value to complete the unlock.

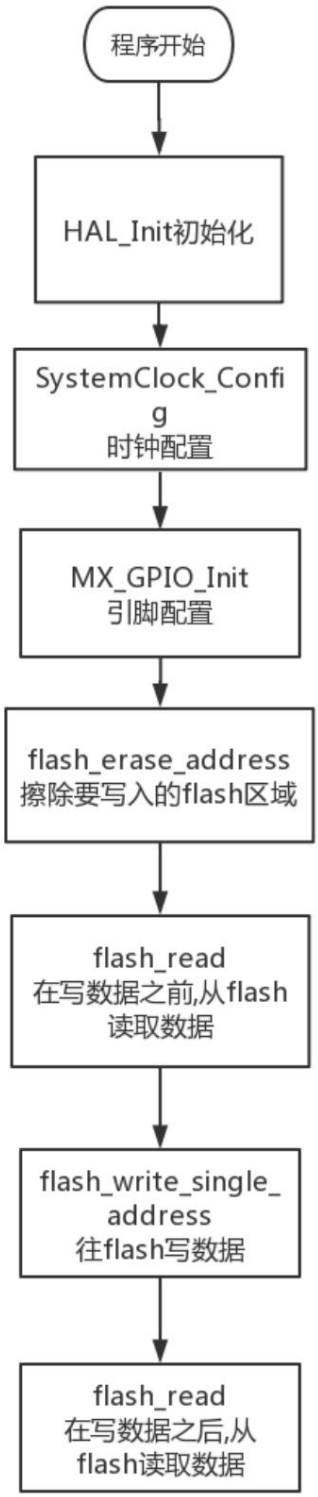
HAL_StatusTypeDef HAL_FLASH_Unlock(void)	
Function name	HAL_FLASH_Unlock
function	Unlock for flash, allowing users to modify flash content
return value	HAL_StatusTypeDef, several states defined by the HAL library, if the flash is successfully completed If unlocked, it will return HAL_OK, if it fails, it will return HAL_ERROR

Then introduce the flash locking operation function HAL_FLASH_Lock. The function of this function is also simple, that is, it is clear that it has been unlocked.

The KEY value of the lock register completes the locking function.

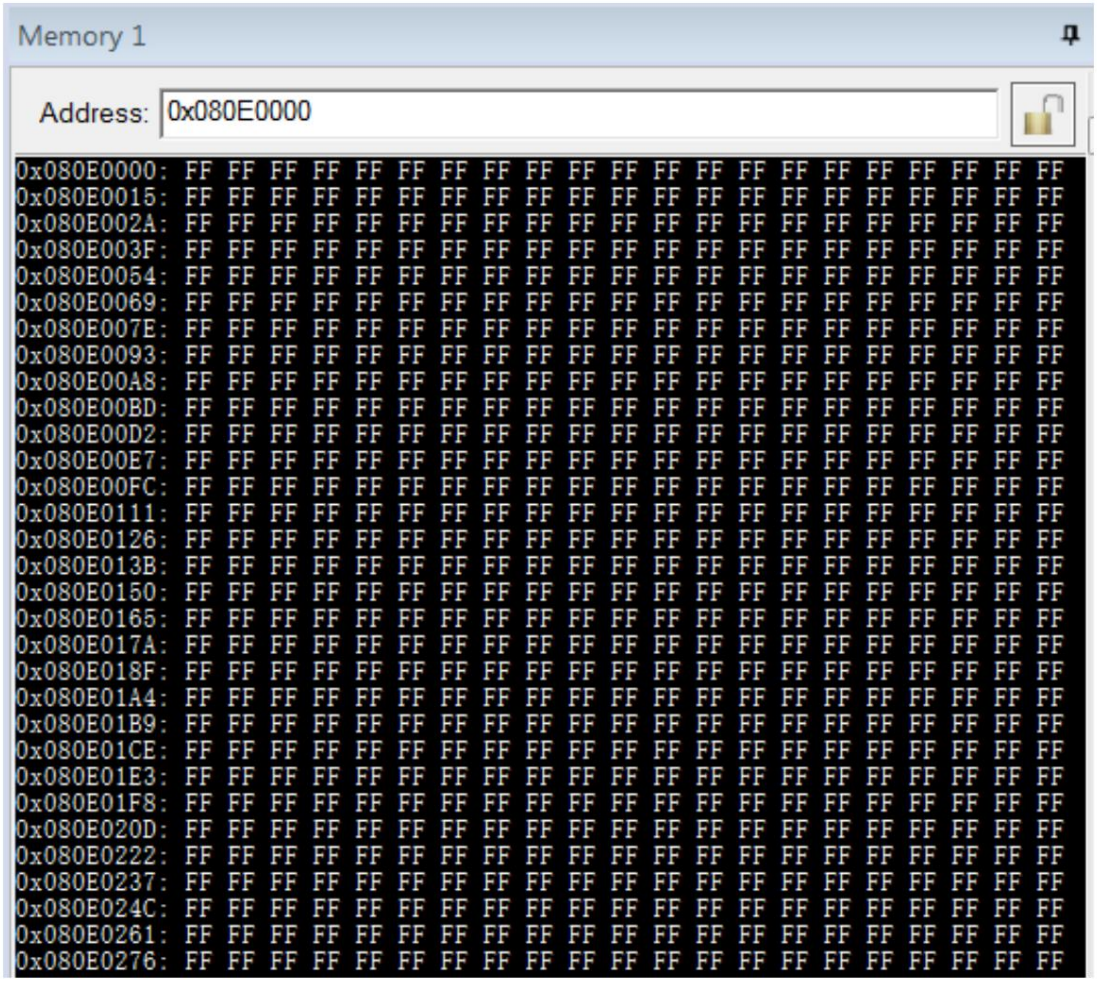
HAL_StatusTypeDef HAL_FLASH_Lock(void)	
Function name	HAL_FLASH_Lock
function	Lock the flash, and the flash content can only be modified after unlocking it in subsequent operations
return value	HAL_StatusTypeDef, several states defined by the HAL library, if the flash is successfully completed If the lock is locked, it will return HAL_OK, and if it fails, it will return HAL_ERROR

10.4.5 Program flow

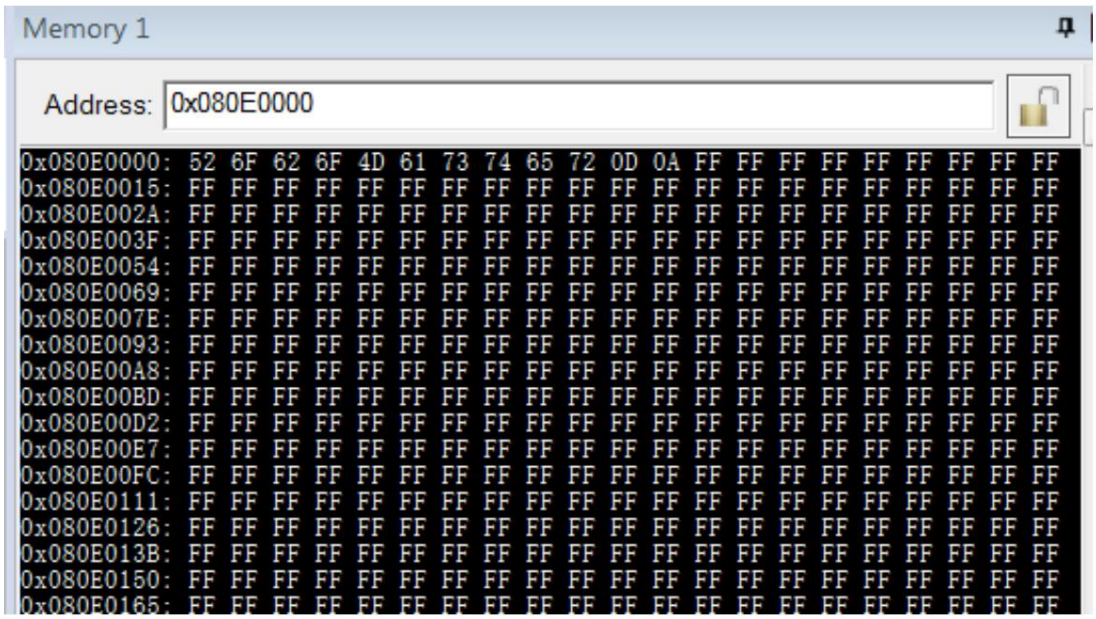


10.4.6 Effect display

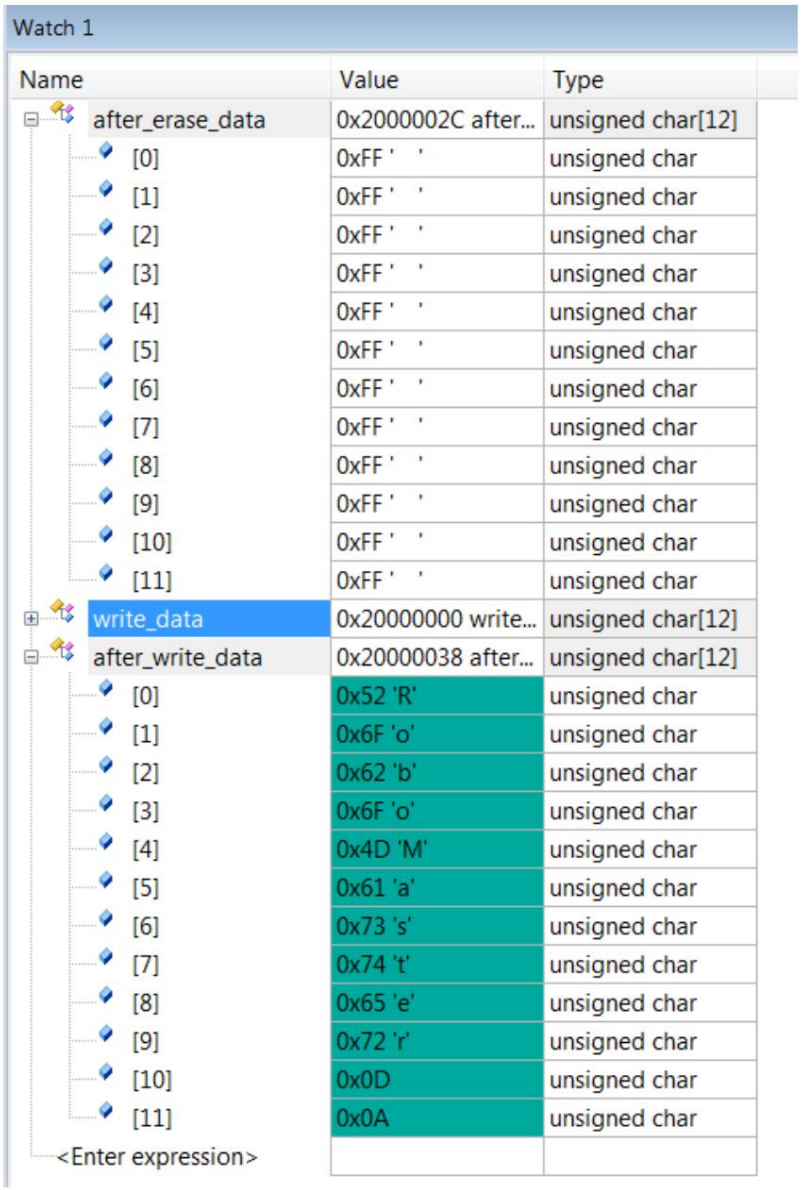
First erase a page of flash starting at 0x80E0000, as shown in the figure:



Then write RoboMaster characters, as shown in the figure:



Read data from flash into variables, as shown in the figure:



10.5 Advanced Learning

10.5.1 Flash Page Partitioning

The stm32f407IG has a total of 12 flash page partitions, as shown in the table below.

	name address		size
main memory	Sector 0	0x08000000 - 0x08003FFF	16 Kbytes
	sector 1	0x08004000 - 0x08007FFF	16 Kbytes
	Sector 2	0x08008000 - 0x0800BFFF	16 Kbytes
	Sector 3	0x0800C000 - 0x0800FFFF	16 Kbytes
	Sector 4	0x08010000 - 0x0801FFFF	64 Kbytes
	Sector 5	0x08020000 - 0x0803FFFF	128 Kbytes
	Sector 6	0x08040000 - 0x0805FFFF	128 Kbytes
	Sector 7	0x08060000 - 0x0807FFFF	128 Kbytes
	Sector 8	0x08080000 - 0x0809FFFF	128 Kbytes
	Sector 9	0x080A0000 - 0x080BFFFF	128 Kbytes
	Sector 10	0x080C0000 - 0x080DFFFF	128 Kbytes
	Sector 11	0x080E0000 - 0x080FFFFF	128 Kbytes

Among them, when stm32 is powered on, it will start to read the program from 0x08000000, so it is often stored at the beginning of sector 0.

Program, you need to use the sector after the program as the storage sector.

10.5.2 The role of boot

Boot is used to control how the boot program is loaded into the chip to start execution after the chip is powered on.

The startup mode of STM32 is divided into three ways, which way to start can be booted through the boot0 and boot1 of the chip.

feet to set.

The first way is the most common way, that is, to read the programmed program from flash, and guide it to STM32 for execution.

In the second way, the chip starts from a special system memory area, in which the chip manufacturer has written

A good bootloader program, the function of this program is to read the program into the flash through the serial port, and then put the program into the flash.

The program is guided to the microcontroller for execution. There are some special download software that allows us to carry out STM32 programs through the serial port

The principle is to use this boot method. The advantage of using this boot is that no additional downloader is required,

The disadvantage is that the download speed is relatively slow.

The third way is to start from the SRAM embedded in the STM32. A small program can be written into the SRAM for debugging.

However, because SRAM has the characteristics of power loss and loss, it is generally not used for booting in this way.

When the chip is newly powered on, the program must be rewritten to the SRAM.

The comparison of the three methods can be seen in the following table:

	way 1	way 2	way 3
BOOT pin state	BOOT0:GND BOOT1:GND or VCC	BOOT0:VCC BOOT1:GND	BOOT0:VCC BOOT1:VCC
start area	Flash user program storage Area	Bootloader program storage Area	Embedded SRAM
Features	Burn the user through the downloader The program is guided to the microcontroller for execution OK, is the most common way	Read the program through the serial port to flash, and then The program in the flash boots to a single executed in the computer.	Program the SRAM sequence to the microcontroller for execution line, generally used for debugging

10.6 Course Summary

This lesson introduces the flash principle of STM32 and its read and write functions. Flash is a commonly used non-volatile storage area.

Programs to be run can be stored as well as user data. Flash can be used to save parameters, read different robot's

Configure parameters, such as saving the median value of the gimbal motor.

11. I2C read IST8310

11.1 Key points of knowledge

ÿ Introduction to I2C protocol

ÿ Introduction to magnetometer

ÿ Register configuration of IST8310

ÿ I2C configuration and hal library function in cubeMX

11.2 Course Content

In this course, you will learn a commonly used serial synchronous communication bus protocol--I2C, and learn to use I2C, configure and read

IST8310 magnetometer settings and data. I2C is not only used for reading magnetometers, but can also read many other

sensors, such as temperature sensors, barometric pressure sensors, multiplex ADC modules, etc. A magnetometer is a measure of the strength of the Earth's magnetic field

The sensor, aka an electronic compass, can be used to calculate the robot's heading.

11.3 Basic Learning

11.3.1 Introduction to I2C

I2C is a half-duplex, bidirectional two-wire synchronous serial bus developed by PHILIPS. Two-wire system represents I2C only

Two signal lines are required, one is the data line SDA, and the other is the clock line SCL.

The I2C bus allows multiple master devices to be mounted, but the bus clock can only be generated by one master device at the same time, and each

Devices connected to the bus have unique I2C addresses, and slave devices can be addressed by master devices.

I2C communication has several types of signals:

ÿ Start signal S: When SCL is at high level, SDA is pulled down from high level to low level, representing the start of data transmission.

beginning.

ÿ End signal P: When SCL is at high level, SDA is pulled from low level to high level, which means the end of data transmission.

ÿ Data signal: The data signal transmits 8 bits of data each time, and each bit of data is transmitted in one clock cycle.

When SCL is at a high level, the level on the SDA data line needs to be stable. When SCL is at a low level,

The level on the SDA data line is allowed to change.

ÿ Response signal ACK/NACK: The response signal is that the master sends 8bit data, and the slave sends a low level to the master, indicating that

Data has been accepted.

The common I2C transmission process for reading sensor data is shown in the following table:

S start letter No	from the device site 7bit	R/W read and write Rank	ALAS	register address + ACK	N bytes of data + ACK	ACK / NACK	P stop letter No

The entire I2C communication process is understood as the process of sending and receiving express delivery, and the I2C address of the device is understood as the address of the school express cabinet.

The bit represents sending and signing for express delivery, the register address is the box number on the express cabinet, and the data needs to be sent or signed for.

Express delivery. The whole process is like going to the school's express cabinet (slave I2C address), to the number of the cabinet box (register address),

The process of sending out or signing for courier (data). For the detailed process, please refer to the reading of IST8310 in Advanced Knowledge

write process.

11.3.2 Introduction to Magnetometer

IST8310 is a 3-axis magnetic field sensor launched by ISentek, the size is 3.0*3.0*1.0mm, supports fast

I2C communication, up to 400kHz, 14-bit magnetic field data, measurement range up to 1600uT(x,y-axis) and 2500uT(z-axis),

Up to 200Hz output frequency.

Magnetic field sensors are often used in electronic compasses to calculate the angle of the geomagnetic field, which originates from the interior of the earth and extends to the

Empty magnetic field. The strength of the magnetic field at the surface is between 25-65 microTesla. At the same time, the earth's magnetic field is not the same as the earth's axis of rotation.

Together, there is an included angle of 11°, so there is a certain magnetic declination angle on the earth's surface, and it changes more with increasing latitude.

In most parts of mainland China, the magnetic declination is about -10°~+2°. Can be detected using the IST8310 magnetometer

Geomagnetic field strength, used to calculate the magnetic field angle.

The various functions of GPIO management of IST8310 are shown in the following table:

pin	Features
SCL	I2C clock line
SDA	I2C data line
RSTN	RESET of IST8310, low level restarts IST8310
DRDY	Data ready for IST8310

11.4 Software Learning

11.4.1 Hardware wiring

Since the IST8310 magnetometer is integrated on the development board, the wiring has been completed on the PCB, so no external wiring is required. correspond

The pins are shown in the table below.

IST8310 pins	MCU pins
SCL	PA8
SDA	PC9
RSTN	PG6
DRDY	PG3

11.4.2 I2C configuration in **cubeMX**

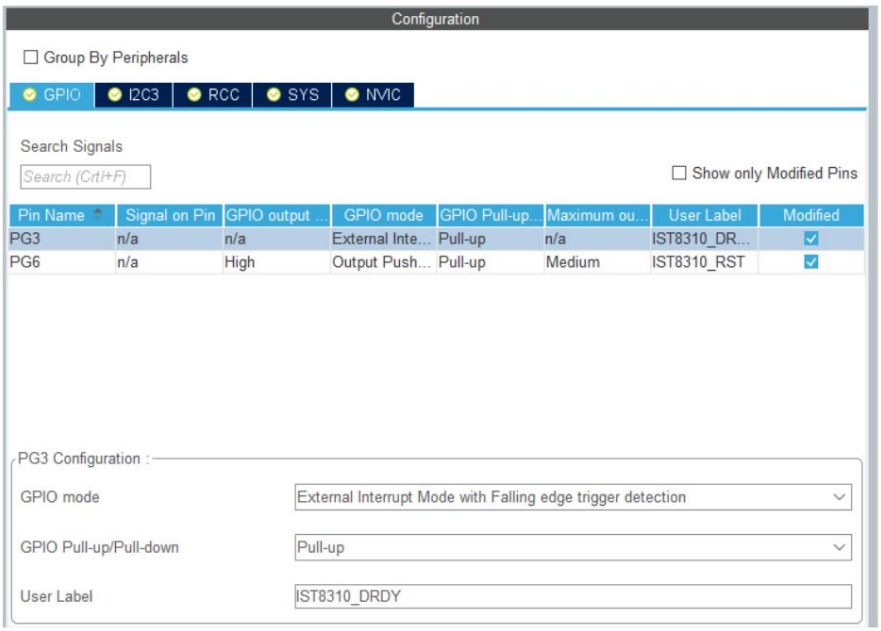
In this example, we have created ist8310driver.c ist8310driver.h, ist8310driver_middle.c and ist8310driver_middle.h Four files, of which ist8310driver.h and ist8310driver_middle.h are used to declare

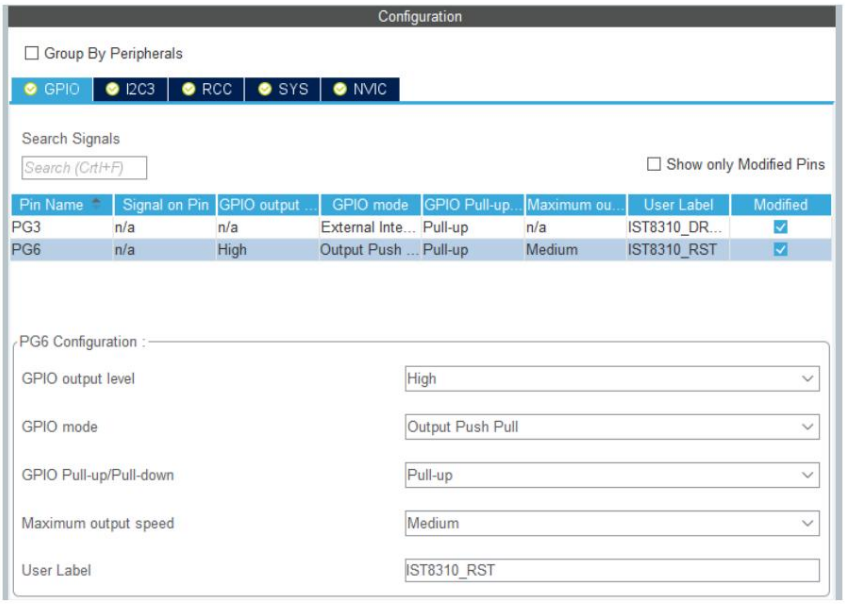
Driver function, ist8310driver.c implements the driver function, and ist8310driver_middle.c implements I2C communication as the middle layer

Encapsulation and delay function for easy porting.

First look at the configuration of cubeMX, PG3 configures external interrupt, triggered by falling edge, the configuration process of external interrupt can refer to

Refer to the external interrupt configuration of buttons in Chapter 6, PG6 is configured as GPIO output mode, pull-up mode, as shown in the figure.





The configuration of I2C3 is as follows, the final configuration is I2C3 into fast mode, the communication frequency is set to 400k, and the I2C address is configured to 7

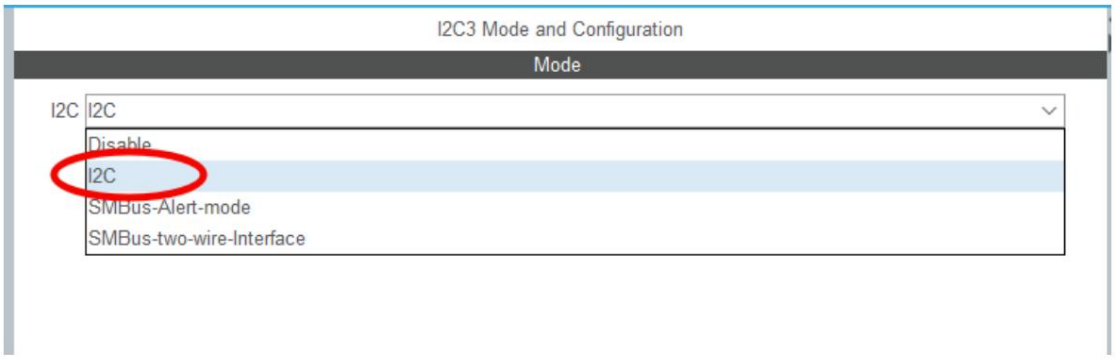
Bits and so on, the pins are configured as PA8, PC9, as shown in the figure:

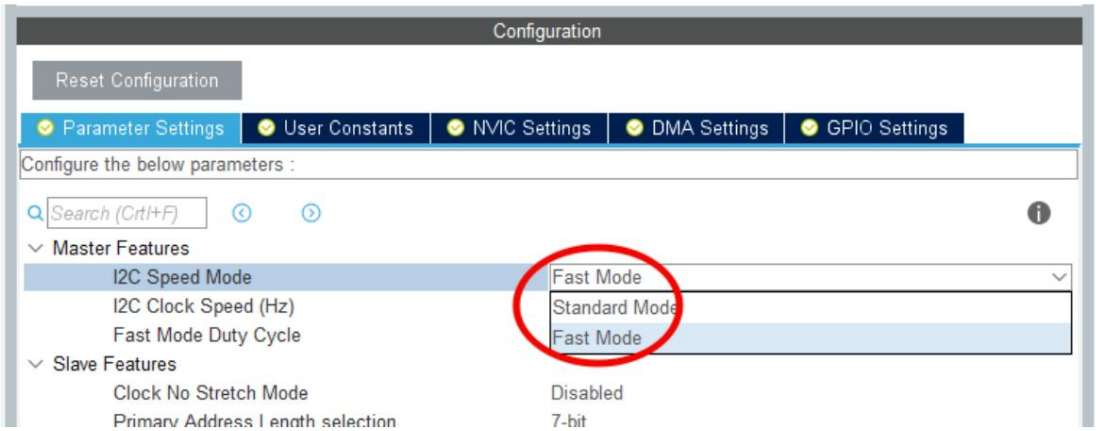
1. Find I2C3 under connectivity;

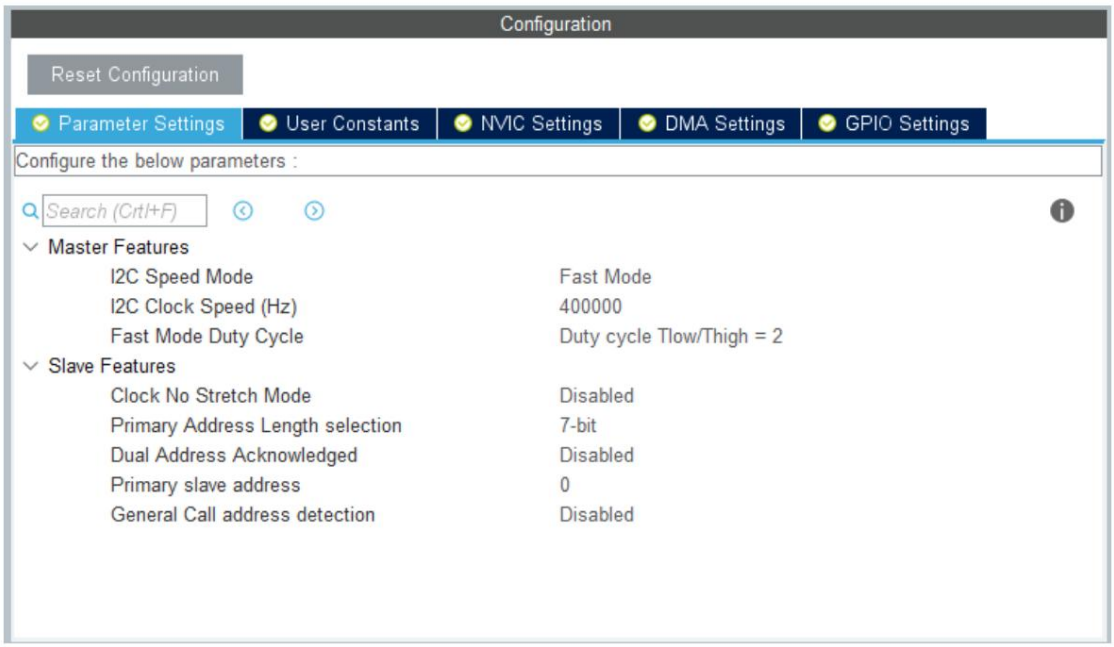
2. Configure as I2C;

3. Configure as Fast Mode;

4. Others are kept as default.







11.4.3 Introduction of main functions

First introduce the communication functions of I2C, ist8310_IIC_read_single_reg, ist8310_IIC_write_single_reg, ist8310_IIC_read_muli_reg, ist8310_IIC_write_muli_reg four functions, they are all for hal library

The encapsulation of I2C functions, because not only can communication through hardware I2C on stm32, but also through software simulation

I2C to communicate, for the convenience of transplantation, re-encapsulate the I2C function.

The library functions HAL_I2C_Mem_Read and HAL_I2C_Mem_Write are the I2C read and write in the HAL library.

It implements the register reading and writing process of the IST8310 in the introduction to I2C.

11.4.3.1 HAL_I2C_Mem_Read function

```
HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

Function name	HAL_I2C_Mem_Read
function	Read data from the registers of the I2C device
function return	HAL status, HAL_OK means read successfully
Parameter 1: hi2c	I2C handle
Parameter 2: DevAddress	I2C slave address
Parameter 3: MemAddress	register address
Parameter 4: MemAddSize	Register address increment size I2C_MEMADD_SIZE_8BIT: increase eight bits I2C_MEMADD_SIZE_16BIT: increase sixteen bits
Parameter 5: pData	data pointer
Parameter 6: Size	Data length
Parameter 7: Timeout	overtime time

11.4.3.2 HAL_I2C_Mem_Write function

```
HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

Function name	HAL_I2C_Mem_Write

function	Write data to the register of the I2C device
function return	HAL status, HAL_OK means read successfully
Parameter 1: hi2c	I2C handle
Parameter 2: DevAddress	I2C slave address
Parameter 3: MemAddress	register address
Parameter 4: MemAddSize	Register address increment size I2C_MEMADD_SIZE_8BIT: increase eight bits I2C_MEMADD_SIZE_16BIT: increase sixteen bits
Parameter 5: pData	data pointer
Parameter 6: Size	Data length
Parameter 7: Timeout	overtime time

Then introduce the initialization process of IST8310.

step	statement	Features	Remark
1. Initialize GPIO Japanese communication	ist8310_GPIO_init (); ist8310_com_init();	initialization pins and I2C communication interface, Ensure normal communication.	Hal library initialized configured, if the With analog I2C you can start here change.

2. Restart the device	ist8310_RST_L();
	ist8310_delay_ms(sleepTime
	);
	ist8310_RST_H();
	ist8310_delay_ms(sleepTime
	);
		Re-pass IST8310
		Turn on the pin for resetting
		start.	

3. Verify Device ID	By reading the WHO_AM_I register judge	Judge IST8310 pass Is the letter normal?	Detailed register introduction Refer to Advanced Learning The IST8310 registers device introduction.
4. Configuration IST8310	Configure the four registers of the IST8310 0x0B: Interrupt register, configured to open Interrupt, low level when interrupted; 0x41: Sampling times register, configuration Into x, y, z are 2 samples 0x42: needs to be configured as 0xC0; 0x0A: configured as 200Hz output frequency		

Secondly, introduce the read processing function ist8310_read_over, which is used to read from the STAT1 register to

The DATAZH register has a total of 7 data, which are processed into the magnetic field strength data whose unit is uT. The function prototype is shown in the figure,

Mainly by judging the value of the stat1 register to determine whether new data is generated, and combining the two eight-bit data into one 16

Bit integer data, multiplied by 0.3 to get the magnetic field strength data in ut.

```
void ist8310_read_over(uint8_t *status_buf, ist8310_real_data_t *ist8310_real_data)
{

    if (status_buf[0] & 0x01)
    {
        int16_t temp_ist8310_data = 0;
        ist8310_real_data->status |= 1 << IST8310_DATA_READY_BIT;

        temp_ist8310_data = (int16_t)((status_buf[2] << 8) | status_buf[1]);
        ist8310_real_data->mag[0] = MAG_SEN * temp_ist8310_data;
        temp_ist8310_data = (int16_t)((status_buf[4] << 8) | status_buf[3]);
        ist8310_real_data->mag[1] = MAG_SEN * temp_ist8310_data;
        temp_ist8310_data = (int16_t)((status_buf[6] << 8) | status_buf[5]);
        ist8310_real_data->mag[2] = MAG_SEN * temp_ist8310_data;
    }
    else
    {
        ist8310_real_data->status &= ~(1 << IST8310_DATA_READY_BIT);
    }
}
```

Finally, a general read magnetic field data function ist8310_read_mag is introduced. The prototype of the function is shown in the figure, mainly through

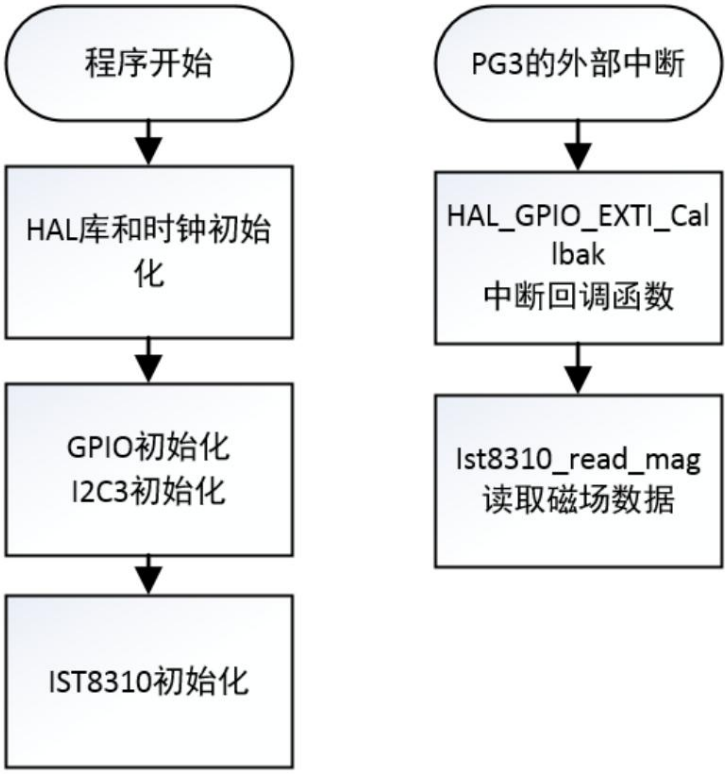
I2C read multiple bytes function, read the DATAXL register, and combine the two eight-bit data into a 16-bit integer

According to the data, multiplied by 0.3, the sensitivity becomes the magnetic field strength data in ut, which is output to the incoming array.


```c
void ist8310_read_mag(fp32 mag[3])
{
    uint8_t buf[6];
    int16_t temp_ist8310_data = 0;
    //read the "DATAXL" register (0x03)
    ist8310_IIC_read_muli_reg(0x03, buf, 6);

    temp_ist8310_data = (int16_t)((buf[1] << 8) | buf[0]);
    mag[0] = MAG_SEN * temp_ist8310_data;
    temp_ist8310_data = (int16_t)((buf[3] << 8) | buf[2]);
    mag[1] = MAG_SEN * temp_ist8310_data;
    temp_ist8310_data = (int16_t)((buf[5] << 8) | buf[4]);
    mag[2] = MAG_SEN * temp_ist8310_data;
}
```

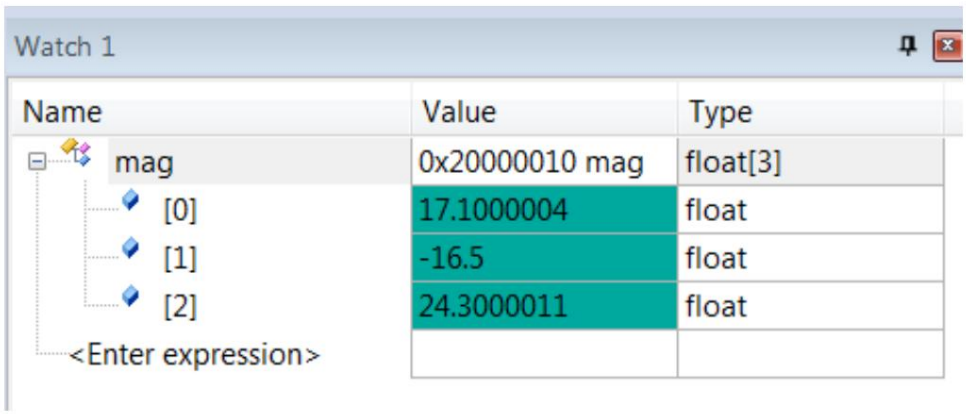
11.4.4 Program Flow

The program starts with the initialization of the HAL library, including the initialization of the clock, GPIO, and I2C3; then the configuration is completed IST8310, the DRDY pin of IST8310 will generate a 200Hz periodic signal; when DRDY falls, it will cause a single The falling edge of the microcontroller is externally interrupted; in the external interrupt callback function, the read function of ist8310 can be called to read the magnetic field data.



11.4.5 Effect display

After reading the data, enter the Debug mode and observe the data size from the watch window.



11.5 Advanced Learning

11.5.1 Read and write process of IST8310

11.5.1.1 IST8310 read single byte process

The IST8310 single-byte read process is shown.

1. Send a start signal;

2. Send the I2C address and read/write bit of IST8310; the read/write bit is write (0);

3. Wait for the ACK bit of the IST8310 bundle;

4. Send the register address that IST8310 needs to read;

5. Wait for the ACK bit of the IST8310 slave;

6. Generate a start signal again;

7. When sending the I2C address and read/write bit of IST8310, the read/write bit is read (1);

8. After waiting for the ACK bit of the IST8310 bundle;

9. The IST8310 plex will send the data of the corresponding register;

10. Since the host only accepts one byte of data, the host does not send an ACK bit;

11. The host stops this communication after sending the stop signal.

Single Byte Read:

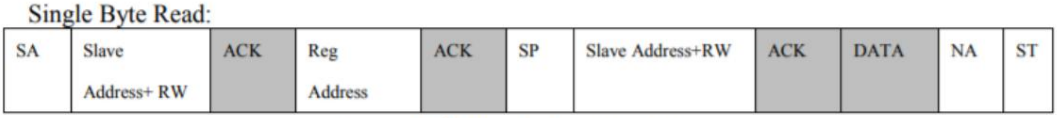


Figure 4. I²C Single Byte Read Operation

For the IST8310 of the C-type development board, the address of the I2C is 0x0E, read the value of the 0x00 register, the whole sending process as shown in the table below.

serial number 1		2	3	4
signal start signal		I2C address (write operation) ACK		register address
send value	SCL high SDA is pulled low from high	0x0E << 1	SDA pulls 0x00 from high	
sender	host MCU	host MCU	Slave IST8310	host MCU
No. 5		6	7	8
Signal ACK		start signal	I2C address (read operation) ACK	

Send value SDA pulled low from high level		SCL high SDA is pulled low from high	0x0E<<1 | 0x01	SDA is pulled low from high
sender	Slave IST8310	host MCU	host MCU	Slave IST8310
serial number 9		10	11	
0x00 register value of signal IST8310 NACK			stop signal	
send value 0x10		SDA remains high	SCL high SDA is pulled high from low	
sender	Slave IST8310	host MCU	host MCU	

11.5.1.2 IST8310 Read Multibyte Process

The multi-byte reading process of IST8310 is shown in the figure. The difference from the single-byte read process is in step 10.

After the master receives a byte, the master sends an ACK signal, then the slave IST8310 will then send the next register value, until the master sends a NACK signal, the slave stops sending data.

Multiple Byte Read:

SA	Slave Address+ RW	ACK	Reg Address	ACK	SP	Slave Address+RW	ACK	DATA	ACK	DATA	NA	ST

Figure 5. I^2C Multiple Byte Read Operation

ACK: Acknowledge, NA: Not Acknowledge, SA: START Condition, SP: Repeat Start Condition, ST: STOP Condition

■: Slave to Master □: Master to Slave

Continuously read data to register 0x00 of IST8310, the whole sending process is shown in the table below.

serial number 1		2	3	4
signal start signal		I2C address (write operation)	ACK	register address

send value	SCL high SDA is pulled low from high	0x0E << 1	SDA is pulled from high Low	0x00
sender	host MCU	host MCU	Slave IST8310	host MCU
No. 5		6	7	8
Signal ACK		start signal	I2C address (read operation do)	ALAS
Send value SDA pulled low from high level		SCL high SDA is pulled low from high	0x0E<<1 | 0x01 SDA is pulled low from high level	
sender	Slave IST8310	host MCU	host MCU	Slave IST8310
serial number 9		10	11	12
0x00 register value ACK of signal IST8310			0x01 of IST8310 register value	ALAS
send value 0x10		SCL high SDA is pulled low from high	0xNN	SCL high SDA is pulled low from high
sender	Slave IST8310	host MCU	Slave IST8310	host MCU
Serial number 13		...15 after N data		16
0x02 register value of signal IST8310...			NACK	stop signal
Send the lower eight bits of the value X-axis			SDA remains high	SCL high SDA is pulled high from low

sender	Slave IST8310		host MCU	host MCU

11.5.1.3 IST8310 write single byte process

The data writing process of IST8310 is simpler than the reading process, and there are fewer intermediate regenerating start signals and second I2C transmission.

The address process, the IST8310 single-byte writing process is shown in Figure 3.12.5.

1. First send a start signal;

2. Send the I2C address of IST8310 and write operation (1);

3. Wait for the ACK bit of the IST8310 bundle;

4. Send the register address value to be written;

5. Wait for the ACK bit of IST8310;

6. The host sends the value to be written;

7. After waiting for the ACK bit of IST8310;

8. The host generates a stop signal to complete the communication.

Single Byte Write:

SA	Slave Address +RW	ACK	Reg Address	ACK	DATA	ACK	ST

Figure 6. I^2C Single Byte Write Operation

ACK: Acknowledge, NA: Not Acknowledge, SA: START Condition, SP: Repeat Start Condition, ST: STOP Condition

■: Slave to Master □: Master to Slave

Assuming that the 0x0B value is written to the 0x0A address of the IST8310 register, the whole process is shown in the following table.

serial number 1		2	3	4
signal start signal		I2C address (write operation) ACK		register address
send value	SCL high SDA is pulled low from high	0x0E << 1	SDA pulls 0x0A low from high	
sender	host MCU	host MCU	Slave IST8310	host MCU

No. 5		6	7	8
Signal ACK		write value	ALAS	stop signal
Send value SDA	pulled low 0x0B from high level		SDA is pulled low from high	SCL high SDA is pulled high from low
sender	Slave IST8310	host MCU	Slave IST8310	host MCU

11.5.1.4 IST8310 Write Multibyte Process

The process of writing multi-byte data of IST8310 is similar to the process of writing one byte of data, the difference is that when writing a word

The stop signal is not sent after the data of the section, but the data is sent next, as shown in the figure.

Multiple Byte Write:

SA	Slave Address +RW	ACK	Reg Adderss	ACK	DATA	ACK	DATA	NA	ST

Figure 7. I²C Multiple Byte Write Operation

ACK: Acknowledge, NA: Not Acknowledge, SA: START Condition, SP: Repeat Start Condition, ST: STOP Condition

■: Slave to Master □: Master to Slave

Assuming that the 0x0B value is written to the 0x0A address of the IST8310 register, and 0x08 is written to 0x0B, the whole process is as follows

shown in the table.

serial number 1		2	3	4
signal start signal		I2C address (write operation) ACK		register address
send value	SCL high SDA is pulled low from high	0x0E << 1	SDA pulls 0x0A low from high	
sender	host MCU	host MCU	Slave IST8310	host MCU
No. 5		6	7	8

Signal ACK		0x0A write value	ALAS	0x0B write value
Send value SDA pulled low 0x0B from high level			SDA pulls down 0x08 from high	
sender	Slave IST8310	host MCU	Slave IST8310	host MCU
serial number 9		10		
Signal ACK		stop signal		
Send value SDA pulled low from high level		SCL high SDA is pulled high from low		
sender	Slave IST8310	host MCU		

11.5.2 Register information of IST8310

IST8310 has a total of 13 registers, as shown below:

1. Who Am I register 0x00

The device ID is stored in this register. For the IST8310 chip, the value of this register is fixed at 0x10, read-only.

Who Am I (0x00)			
Rank	illustrate	read and write	Defaults
7:0	Device ID	read only	0x10

2. State 1 Register 0x02

This register provides status information for the IST8310.

STAT1(0x02)			
bit description		read and write	Defaults

7:2 reserved			
1	Data overflow, when this bit is 1, it means there is old data has not been read, and new data has been generated.	read only	0
0	Data preparation status: When this bit is 1, it means there is already New measurement data is generated.	read only	0

3. Output data registers 0x03-0x08

DATAXL(0x02)			
Rank	illustrate	read and write	Defaults
7:0	X-axis magnetic field data low eight read-only		0
DATAXH(0x04)			
Rank	illustrate	read and write	Defaults
7:0	X-axis magnetic field data high eight read-only		0
DATAYL (0x05)			
Rank	illustrate	read and write	Defaults
7:0	The lower eight bits of the Y-axis magnetic field data are read-only		0
DATAYH(0x06)			
Rank	illustrate	read and write	Defaults
7:0	Y-axis magnetic field data high eight read-only		0
DATAZL(0x07)			
Rank	illustrate	read and write	Defaults

7:0	Z-axis magnetic field data low eight	read-only	0
DATAZH (0x08)			
Rank	illustrate	read and write	Defaults
7:0	Z-axis magnetic field data high eight	read-only	0

4. State 2 Register 0x09

This register provides status information for the IST8310.

STAT2(0x09)			
bit description		read and write	Defaults
7:4 Reserved			
3	Interrupt flag, when an interrupt event occurs, this bit will be set to 1 read only		0
2:0 reserved			

5. Control Setting 1 Register 0x0A

This register sets the measurement mode of the IST8310.

CNTL1(0x0A)			
Rank	illustrate	read and write	Defaults
7:4	Reserve		
3:0	Measurement mode: 0: sleep mode 1: Single measurement mode 11: Continuous measurement mode output frequency 200Hz	read/write 0	

6. Control Setting 2 Register

This register mainly sets interrupt IO.

CNTL2(0x0B)			
Rank	illustrate	read and write	Defaults
7:4	Reserve		
3	Interrupt function enable: 0: not enabled 1: Enable interrupt	read/write	1
2	DRDY pin interrupt level 0: low level 1: High level	read/write	1
1	Reserve		
0	software restart 0: do not restart 1: reboot	read/write	0

7. Self-test register 0x0C

CNTL2(0x0C)			
Rank	illustrate	read and write	Defaults
7	Reserve		
6	Self-check: When this bit is set to 1, the chip enters self-test mode	read/write 0	
5:0	Reserve		

8. Temperature register

TEMPL(0x1C)			
Rank	illustrate	read and write	Defaults
7:0	The lower eight bits of temperature data are read-only		0
TEMPH(0x1D)			
Rank	illustrate	read and write	Defaults
7:0	The upper eight bits of temperature data are read-only		0

9. Sample average register

This register sets the number of samples in a loop to reduce data noise, and a high number of samples can reduce data noise.

AVGCNTL(0x41)			
bit description		read and write	Defaults
7:6 reserved			0
5:3	Number of samples for the Y axis 3b00: one sample 3b001: Two-sample average 3b010: Four-sample averaging (recommended) 3b011: Eight-Sample Average 3b100: Average of sixteen samples Other: one sample	read/write	0
2:0	Number of samples for the x-axis and z-axis 3b00: one sample 3b001: Two-sample average	read/write	0

	3b010: Four-sample averaging (recommended) 3b011: Eight-Sample Average 3b100: Average of sixteen samples Other: one sample		

11.6 Course Summary

I2C communication is a common communication method that can be found in many sensors, such as temperature sensors, etc.

The sensor is the perception system of the robot, just like the eyes of the robot, without sensing the changes of the external environment, the robot is like a robot.

Walking in the dark, unable to run on the right track. A magnetometer is a sensor that senses the strength of the Earth's magnetic field.

It senses the direction through the earth's magnetic field and finds the route of returning to the north. The robot can also calculate the direction through the magnetometer, which can be used later.

The face pose solution section is explained.

12. OLED display

12.1 Key points of knowledge

ÿ Introduction to OLED

ÿ OLED communication protocol

ÿ Introduction to OLED configuration commands

ÿ Introduction to simple drawing functions

12.2 Course Content

In the previous chapter, I learned the I2C communication method and used I2C to obtain the magnetic field data of the IST8310. This course continues

Use I2C communication to light up the OLED module and display the Robomaster LOGO. Since the OLED module supports multiple

The I2C communication process of the OLED module is different from that of the IST8310.

The second package is to achieve the purpose of classifying the data packets, so as to adapt to the various communication methods of the OLED.

12.3 12.3 Basic Learning

12.3.1 Introduction to OLED

OLED (Organic Light-Emitting Diode), also known as organic electric laser display, organic light-emitting semiconductor (Organic Light-Emitting Diode)

Electroluminesence Display, OLED). OLED belongs to a current-type organic light-emitting device, the light-emitting principle

It is caused by the injection and recombination of carriers, and the luminous intensity is proportional to the injected current. OLED action in electric field

When used, the holes generated by the anode and the electrons generated by the cathode will move to the hole transport layer and the electron transport layer, respectively.

Implanted, migrated to the light-emitting layer. When the two meet in the light-emitting layer, energy excitons are generated, which excite the light-emitting molecules and finally produce

produce visible light. [Baidu Encyclopedia]

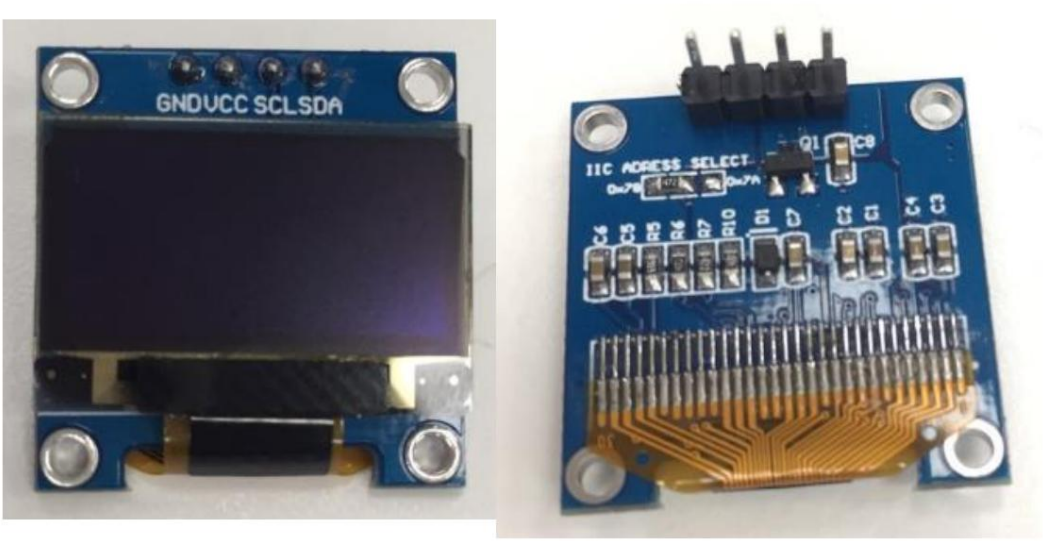
In the RM competition, the OLED screen module has the functions of displaying data and guiding operations, such as displaying sensor data, improving

Wake the team up to check the robot. OLED modules are cheap and available in many stores. The common screen size is 0.96 inches.

The resolution is 128*64, and the common communication method supports 6800, 8080 two parallel interfaces, or SPI interface and I2C interface

mouth. The following is an example of an OLED module that leads out the I2C interface.

The appearance of the module is shown in the figure:



OLED Module Appearance Diagram

The OLED module uses I2C communication, the default address of the I2C device is 0x78, you can manually modify the

resistor to configure the I2C address as 0x7A.

Since the OLED module itself does not have the process of reading data like the IST8310, the whole process is that the host keeps sending data to the slave OLED.

The module sends data. The first data after the I2C address in the sent data represents the control command (0x00) or the number

According to the instruction (0x40). Their similarities and differences are shown in the table below:

	IST8310	YOU ARE
difference	There are read and write operations	write operations only
	After sending the I2C address, send the memory address information	After sending the I2C address, send the data type and then send the data, where The data type is 0x00 for control instructions, 0x40 for data pointers make.
Same point	ý All have fixed I2C addresses, ý Both support I2C frequency up to 400K ý Both transmit 8 bytes of data	

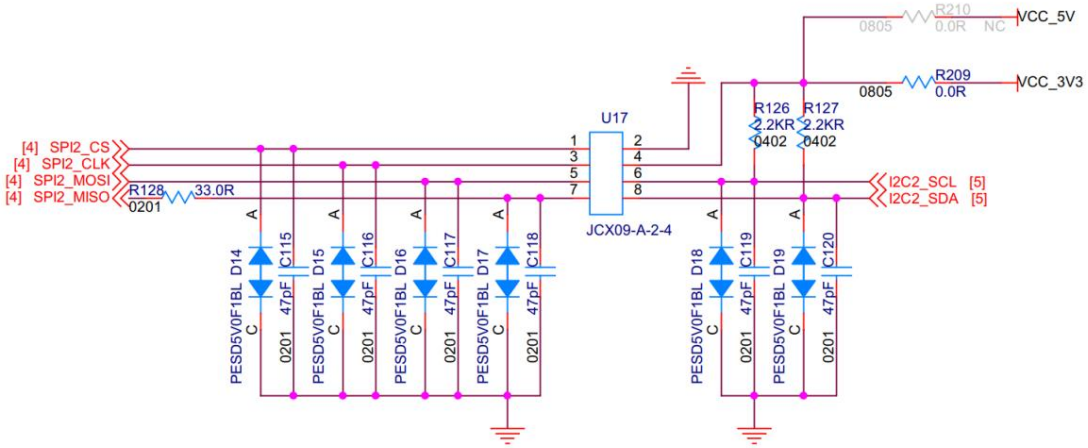
It can be seen that the biggest difference between the OLED communication process and the IST8310 communication process lies in the data after the I2C address is transmitted.

OLED distinguishes the following data types by one byte of data.

12.4 Software Learning

12.4.1 Hardware wiring

This example uses the user I2C interface on the C-type development board, and the schematic diagram is shown in the figure:



User I2C Schematic

The I2C used is I2C2, and the I2C line sequence on the C-type development board is consistent with the line sequence on the OLED module, so only

Just connect normally. The relationship between the IO ports of the development board is as follows:

ÿ PF0 corresponds to I2C2_SDA and is connected to SDA of OLED;

ÿ PF1 corresponds to I2C2_SCL and is connected to SCL of OLED.

Pay attention to use Dupont wire to connect the OLED module, note that the length of Dupont wire should not be too long, otherwise it may cause the I2C communication to be affected

Communication fails due to interference.

12.4.2 cubeMX configuration process

In this routine, three new files OLED.c, OLED.h and OLEDfont.h are created, of which OLED.h is used to declare the driver

OLED.c implements the driver function, OLEDfont.h stores the ASCII character code and Robomaster

Logo code.

The I2C configuration in cubeMX is as follows: the communication frequency is set to 400k, the pin configuration is PF0, PF1, and the I2C address is configured as

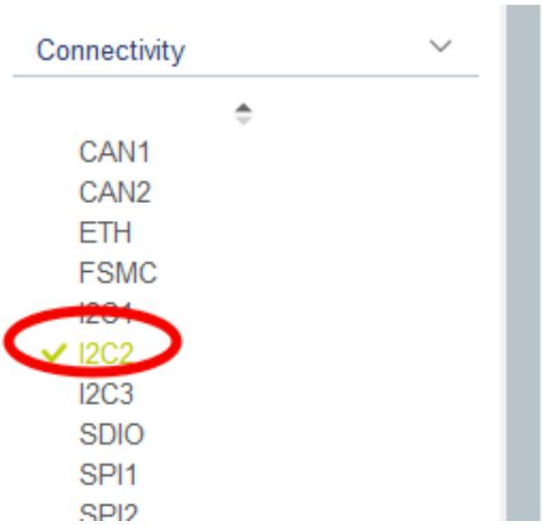
7 bits and so on, as shown:

1. Find I2C2 under connectivity;

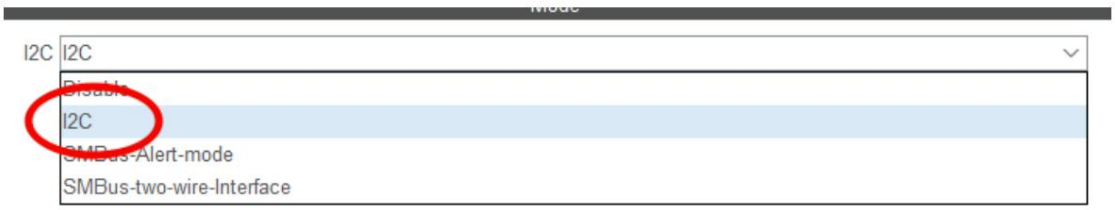
2. Configure as I2C;

3. Configure as Fast Mode;

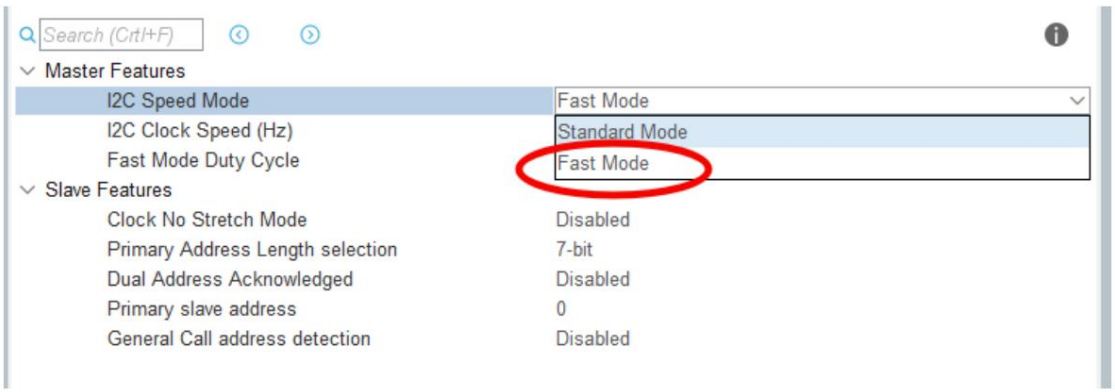
4. Others are kept as default.



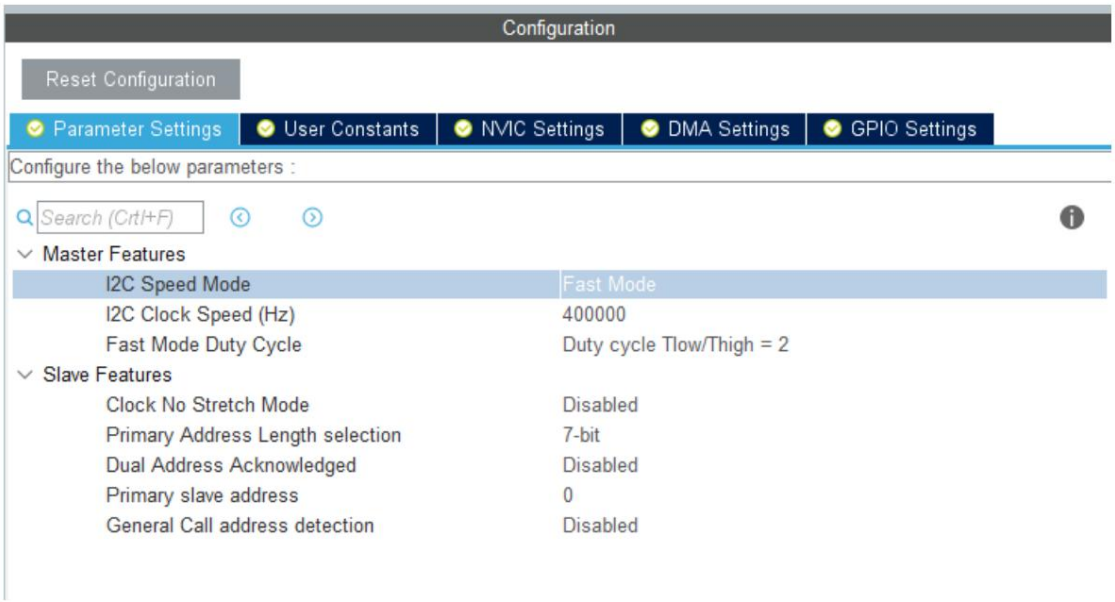
I2C2 under Connectivity



Configure I2C



Configure Fast Mode



I2C Configuration Final Diagram

12.4.3 Introduction of main functions

1. HAL_I2C_Master_Transmit function

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

Function name	HAL_I2C_Master_Transmit
function	Transfer data to an I2C device
function return	HAL status, HAL_OK means read successfully
Parameter 1: hi2c	I2C handle
Parameter 2: DevAddress	I2C slave address
Parameter 3: pData	data pointer
Parameter 4: Size	Data length
Parameter 5: Timeout	overtime time

2. oled_write_byte function

According to the communication method of OLED, the data type after that needs to be specified in the first byte, if it is a control command

You need to send 0x00, if it is a data command, you need to send 0x40. The function prototype is as follows:

```
/**
    * @brief                write data/command to OLED, if you use spi, please rewrite the function
    * @param[in]            dat: the data ready to write
    * @param[in]            cmd: OLED_CMD means command; OLED_DATA means data
    * @retval               none
    */
/**
    * @brief                Write data or command to OLED, if using SPI, please rewrite this function
    * @param[in]            dat: bytes to write
    * @param[in]            cmd: OLED_CMD means the written byte is a command; OLED_DATA means the written byte is data
    * @retval               none
    */
void oled_write_byte(uint8_t dat, uint8_t cmd)
{
        static uint8_t cmd_data[2];
        if(cmd == OLED_CMD)
        {
                cmd_data[0] = 0x00;
        }
        else
        {
                cmd_data[0] = 0x40;
        }
```

```
    cmd_data[1] = that;

    HAL_I2C_Master_Transmit (& hi2c2, OLED_I2C_ADDRESS, cmd_data, 2, 10);

}
```

Function name	Oled_write_byte
function	Send a command to the OLED
function return	None
Parameter 1: data	command data
Parameter 2: cmd	control command or data command OLED_CMD is the control command OLED_DATA is the data command

3. oled_init function

Introduce the OLED_init function, which mainly configures the OLED parameters, which are passed in by calling oled_write_byte

OLED_CMD, transmit the control command to complete the configuration, as shown in the figure. For detailed functions, refer to Advanced Learning.


```
void OLED_init(void)
{
    oled_write_byte(0xAE, OLED_CMD);    //display off
    oled_write_byte(0x20, OLED_CMD);    //Set Memory Addressing Mode
    oled_write_byte(0x10, OLED_CMD);    //00,Horizontal Addressing Mode;01,Vertical Addressing Mode;10,Pa
    oled_write_byte(0xb0, OLED_CMD);    //Set Page Start Address for Page Addressing Mode,0-7
    oled_write_byte(0xc8, OLED_CMD);    //Set COM Output Scan Direction
    oled_write_byte(0x00, OLED_CMD);    //---set low column address
    oled_write_byte(0x10, OLED_CMD);    //---set high column address
    oled_write_byte(0x40, OLED_CMD);    //--set start line address
    oled_write_byte(0x81, OLED_CMD);    //--set contrast control register
    oled_write_byte(0xff, OLED_CMD);    //brightness 0x00~0xff
    oled_write_byte(0xa1, OLED_CMD);    //--set segment re-map 0 to 127
    oled_write_byte(0xa6, OLED_CMD);    //--set normal display
    oled_write_byte(0xa8, OLED_CMD);    //--set multiplex ratio(1 to 64)
    oled_write_byte(0x3F, OLED_CMD);    //
    oled_write_byte(0xa4, OLED_CMD);    //0xa4,Output follows RAM content;0xa5,Output ignores RAM content
    oled_write_byte(0xd3, OLED_CMD);    //-set display offset
    oled_write_byte(0x00, OLED_CMD);    //-not offset
    oled_write_byte(0xd5, OLED_CMD);    //--set display clock divide ratio/oscillator frequency
    oled_write_byte(0xf0, OLED_CMD);    //--set divide ratio
    oled_write_byte(0xd9, OLED_CMD);    //--set pre-charge period
    oled_write_byte(0x22, OLED_CMD);    //
    oled_write_byte(0xda, OLED_CMD);    //--set com pins hardware configuration
    oled_write_byte(0x12, OLED_CMD);
    oled_write_byte(0xdb, OLED_CMD);    //--set vcomh
    oled_write_byte(0x20, OLED_CMD);    //0x20,0.77xVcc
    oled_write_byte(0x8d, OLED_CMD);    //--set DC-DC enable
    oled_write_byte(0x14, OLED_CMD);    //
    oled_write_byte(0xaf, OLED_CMD);    //--turn on oled panel
}
```

OLED_init function diagram

4. OLED_display_on function and OLED_display_off function

The OLED_display_on and OLED_display_off functions are used to turn off the OLED display and turn on the OLED respectively show.

function	Features
OLED_display_on	Turn on the OLED display
OLED_display_off	Turn off OLED display

5. OLED_operate_gram function

The OLED_operate_gram function operates on the OLED_GRAM[128][8] array, and we perform operations on the entire array.

After the operation is completed, the GRAM inside the OLED is refreshed as a whole through the OLED_refresh_gram function.

OLED_operate_gram is shown in the figure:

Function name	OLED_operate_gram
Features	Turn on the OLED display
function return	None

Parameter 1: pen	Operation Type: WRITE: means to light all pixels; CLEAR: means to turn off all pixels; INVERSION: Represents inverting all pixel states.

```c
/**
 * @brief   operate the graphic ram(size: 128*8 char)
 * @param   pen: the type of operate.
 *          PEN_CLEAR: set ram to 0x00
 *          PEN_WRITE: set ram to 0xff
 *          PEN_INVERSION: bit inversion
 * @retval  none
 */
void OLED_operate_gram(pen_typedef pen)
{
    uint8_t i, n;

    for (i = 0; i < 8; i++)
    {
        for (n = 0; n < 128; n++)
        {
            if (pen == PEN_WRITE)
            {
                OLED_GRAM[n][i] = 0xff;
            }
            else if (pen == PEN_CLEAR)
            {
                OLED_GRAM[n][i] = 0x00;
            }
            else
            {
                OLED_GRAM[n][i] = 0xff - OLED_GRAM[n][i];
            }
        }
    }
}
```

OLED_operate_gram provides three operation modes:

ÿ The first type: PEN_WRITE is to set the array to 0xff, which is full bright for OLED screen,

ÿ The second type: PEN_CLEAR is to set the array to 0x00, which means it is all off for the OLED screen.

ÿ The third type: PEN_INVERSION is to invert all the values of the array, which is realized by subtracting it from 0xff.

6. OLED_refresh_gram function

The function of OLED_refresh_gram function is to transfer the internal GRAM[8][128] to the GRAM of the OLED module,

This way the OLED will display the image. The implementation process is to first set the display start page address, and then continuously refresh this page

128 lines of data, call the oled_write_byte function, and pass in the OLED_DATA parameter, indicating the number of transmissions

The data type is "data command". The function implementation is shown in the figure:

```
/**
 * @brief   send the data of gram to oled sreen
 * @param   none
 * @retval  none
 */
void OLED_refresh_gram(void)
{
    uint8_t i, n;

    for (i = 0; i < 8; i++)
    {
        OLED_set_pos(0, i);
        for (n = 0; n < 128; n++)
        {
            oled_write_byte(OLED_GRAM[n][i], OLED_DATA);
        }
    }
}
```

7. OLED_set_pos function

The OLED_set_pos function is to set the OLED cursor position, and then if the display data is transmitted, it will be in the corresponding position

set to display. For the implementation principle of the function, please refer to chapter 10.1.3 and chapter 10.1.3 in the SSD1306 data manual page 34-35.

Chapter 10.1.13 explains.

```
/**
 * @brief   cursor set to (x,y) point
 * @param   x:X-axis, from 0 to 127
 * @param   y:Y-axis, from 0 to 7
 * @retval  none
 */
void OLED_set_pos(uint8_t x, uint8_t y)
{
    oled_write_byte((0xb0 + y), OLED_CMD);              //set page address y
    oled_write_byte(((x&0xf0)>>4)|0x10, OLED_CMD);      //set column high address
    oled_write_byte((x&0x0f), OLED_CMD);                //set column low address
}
```

OLED_set_pos function diagram

There are also some drawing functions, as shown in the following table.

Drawing function table

function	Features
OLED_draw_point	Point-in operation on a pixel of (x,y) coordinates
OLED_draw_line	Operate on the pixels passing through the line from (x1, y1) to (x2, y2)
OLED_show_char	display a character
OLED_show_string	display a string
OLED_printf	Printf function function
OLED_LOGO	Show Robomaster logo

The functions in the oled.c file are introduced here. The content of oled.h mainly contains some definitions, and the oledfrot.h file mainly defines the words character encoding and Robomaster LOGO encoding.

The display process is as follows:

1. OLED module initialization, call OLED_init
2. Operate the GRAM array in stm32 through the drawing function
3. Call the OLED_refresh_gram function to transfer the GRAM data to the GRAM of the OLED module for display.
4. The OLED_LOGO function integrates the data that refreshes the GRAM into LOGO, and finally calls the OLED_refresh_gram function.

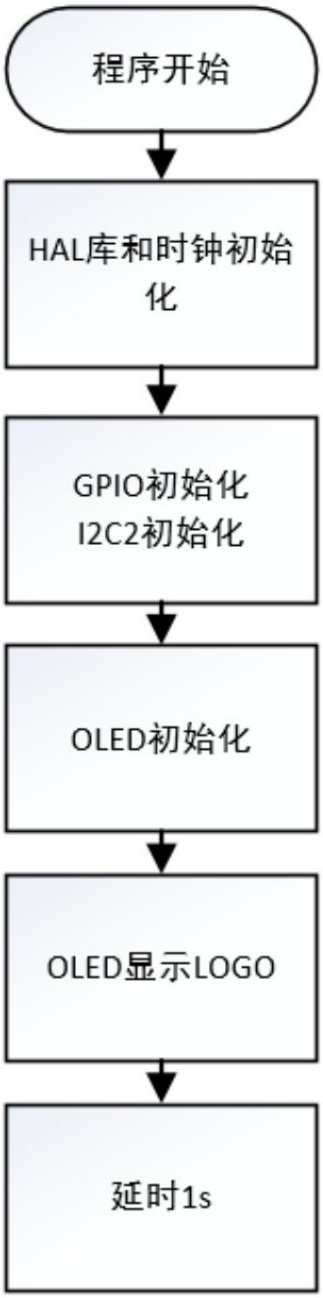
The main function main is shown below.


```
  /* USER CODE BEGIN 2 */
    OLED_init();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
        OLED_LOGO();
        HAL_Delay(1000);
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

main function diagram

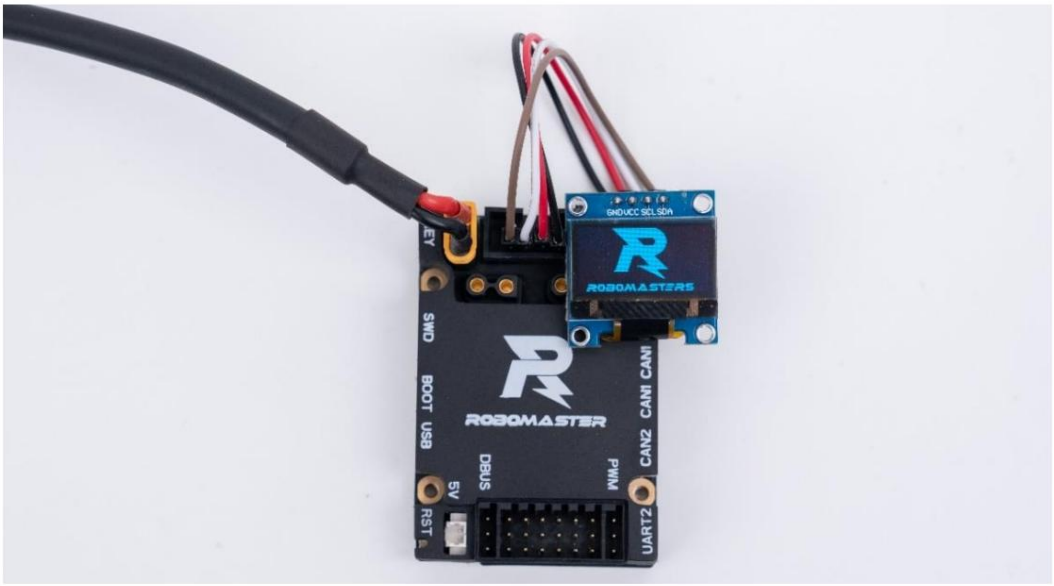
12.4.4 Program Flow

After the initialization that comes with the HAL library, initialize the OLED, enter the main loop, and refresh the OLED screen every second screen, showing the Robomaster LOGO.



12.4.5 Effect display

After connecting the OLED module, you can directly observe the RoboMaster LOGO on the OLED screen.



LOGO display renderings

12.5 Advanced Learning

12.5.1 OLED Communication Process

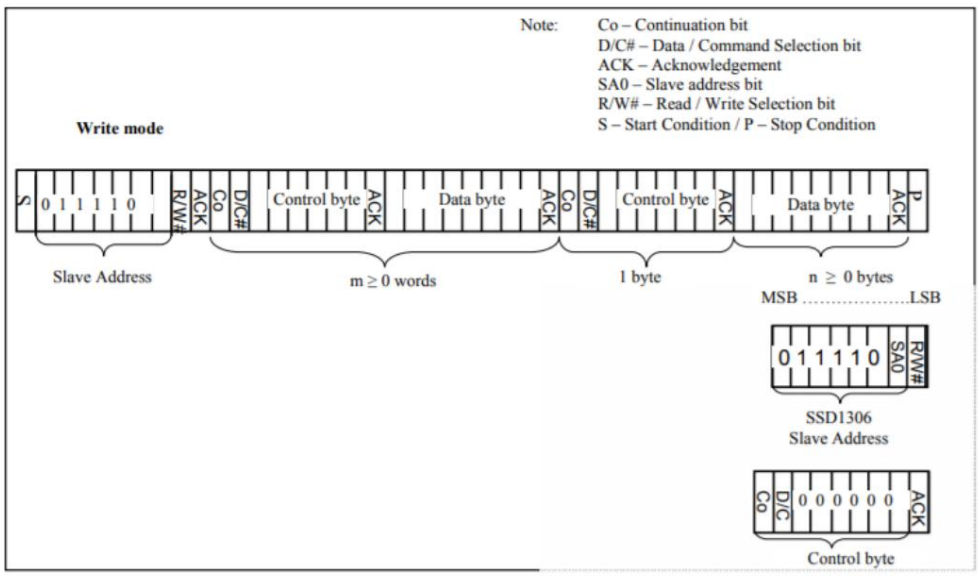
The controller used by the OLED module is SSD1306. For details, please refer to the data sheet of SSD1306 to understand SSD1306

How to control the OLED screen to display characters, SSD1306 controller supports 6800,8080 two parallel interface communication methods,

As well as 4-wire SPI communication and I2C communication, we will introduce the I2C communication below. Data hand from SSD1306

On page 20 of the booklet, we can see the following communication sequence diagram, as shown in the figure

Figure 8-7 : I²C-bus data format



I2C Communication Timing Diagram

The entire I2C communication process is as follows:

1. Generate the start bit of I2C;

2. Send the 7-bit I2C address of the OLED. For the I2C address of the SSD1306, according to the level of SA0, that is

Depending on the welding method of the resistor, it is divided into "b0111100" (SA0 low level) and "b0111101" (SA0 high level);

3. Send read and write bit, for SSD1306, this item is low level 0, add write and read bit 0 to the I2C address to be sent,

is 0x78("b0111100 0") or 0x7A("b0111101 0");

4. Wait for the ACK response from the slave OLED;

5. Transmit a control byte that will determine whether the data to be transmitted is a control command or a data command.

This control byte includes a Co bit and D/C bit, 6 0 bits;

1) The Co bit is set to 0, then the data transmitted after that only contains data bytes;

2) The D/C bit determines whether the next transmitted data is a control command or data. If the D/C bit is set to 0, it is a control command.

If the D/C bit is set to 1, it is data;

6. Wait for the ACK response from the slave OLED;

7. Send several data messages and wait for the ACK response from the slave OLED

8. When the host sends a stop bit, the entire transfer process ends. The stop bit is when the SCL pin is high

time, the SDA pin is pulled from low to high.

From the above process, it can be seen that if you need to send control commands to the OLED, you need to send the I2C address after sending the control command.

Send 0x00 data; if you need to send a data command to the OLED, you need to send it after sending the I2C address

0x40 data.

12.5.2 OLED Initial Configuration

During the initialization process, the configuration parameters can refer to the following table:

OLED_init configuration table

Command function		refer to
0xAE turns off OLED display		Chapter 10.1.12 on page 37 of the SSD1306 datasheet
0x20 0x10	Set the memory address mode of the OLED to page address mode.	Chapter 10.1.3 of the SSD1306 datasheet, pages 34-35
0xb0 Set the starting page address as the first page address		SSD1306 datasheet page 37 explained in chapter 10.1.13
0xC8 Set the COM scan mode of SSD1306. Explain in chapter 10.1.14 in page 37 of SSD1306 data sheet		

Command	function	refer to
0x00 0x10	Sets the row start address for the page address where 0x00 sets the lower 4 of the row start address Bit, 0x10 sets the upper 4 of the row start address Rank.	SSD1306 datasheet chapter 10.1.1 on page 34 and 10.1.2 Explanation
0x40	Set the starting line address	SSD1306 Datasheet 10.1.6 explained on page 36
0x81 0xFF	Set OLED brightness	SSD1306 Data Sheet 10.1.7 Explained on page 36
0xA1	Set segment redirection	SSD1306 Data Sheet 10.1.8 explained on page 36
0xA6	Set normal display, not reverse display	SSD1306 Data Manual page 37 in 10.1.10 explained
0xA8 0x3F	set polymorphism ratio set 63 (0x3F)	SSD1306 Datasheet 10.1.11 explained on page 37
0xA4	Set the output according to the GDDRAM content	SSD1306 datasheet page 37 explained in 10.1.9
0xD3	set display offset	SSD1306 Datasheet 10.1.15 explained on page 37
0xAF	Turn on OLED display	Chapter 10.1.12 on page 37 of the SSD1306 datasheet

12.6 Course Summary

OLED is a common display device in RoboMaster competitions. Through the OLED display, you can check the status and the status of the robot. Various information to facilitate the inspection of the robot. This chapter learned that OLED uses the data type byte in the communication process As a distinction between control instructions and data instructions. This way of redefining the protocol on the data layer is also often reflected in other protocols. Now, for example, in the serial port protocol of the RoboMaster referee system, frame header, CMD_ID, data length, etc. are defined, among which CMD_ID is similar to this byte of OLED, indicating the type of data.

13. BMI088 sensor

13.1 Key points of knowledge

ÿ Introduction to Gyroscope

ÿ Introduction to Accelerometer

ÿ Introduction to SPI protocol

ÿ BMI088 register introduction

ÿ BMI088 read function introduction

13.2 Course Content

This lesson will introduce the relevant knowledge of BMI088 sensor, BMI088 sensor is a six-axis inertial measurement unit (IMU),

It can be applied to the attitude calculation of the robot. In this chapter, we will learn the internal structure of the six-axis inertial measurement unit - the three-axis gyro

Gyroscopes and triaxial accelerometers, the function of the inertial measurement unit and its basic principles, and we will introduce an important communication

Protocol - SPI protocol, and how to use the BMI088 sensor, and write a project to drive the BMI088 sensor

move.

13.3 Basic Learning

13.3.1 Introduction to Gyroscope

The BMI088 high-performance inertial measurement unit (IMU) from Bosch is specifically designed for use in drone and robotics applications.

This 6-axis sensor integrates a 16-bit ADC precision three-axis gyro in a small 3 x 4.5 x 0.95mm³ LGA package

Gyroscope and triaxial accelerometer.

Gyroscope is a sensor that measures angular velocity and is an important part of IMU. The gyroscope can measure in three orthogonal directions

The angular velocity of rotation can also be used to estimate the angle of rotation in three directions.

There are many types of gyroscopes. Different gyroscopes are generally based on different working principles and can achieve different accuracy.

The basic principle of the most commonly used microelectromechanical (MEMS) gyroscopes in the field is to use the Coriolis force generated when rotating to induce capacitance

, thereby converting the angular velocity of rotation into an electrical signal.

13.3.2 Introduction to Accelerometers

Another important component in a six-axis IMU is the accelerometer; as the name implies, the accelerometer is capable of measuring three orthogonal directions

acceleration on. The principle of the MEMS accelerometer is to use the acceleration change to change the force generated by the internal mass, from

And change the capacitance size, converted into an electrical signal.

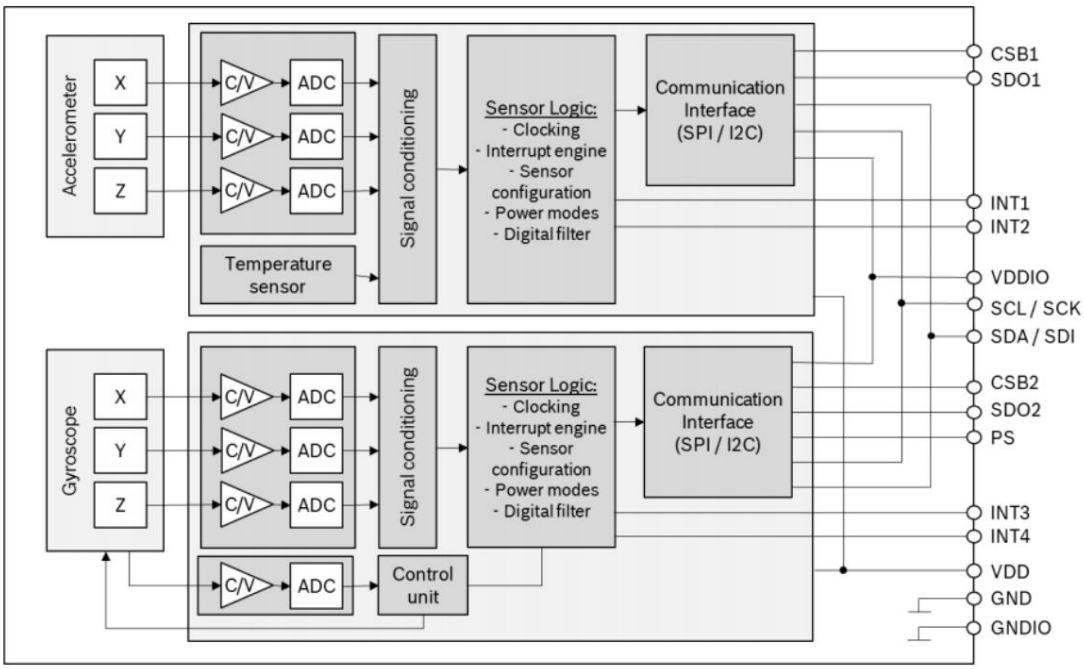
When the object is stationary, the accelerometer measures the components of the gravitational acceleration in three orthogonal directions, and cooperates with the angular velocity information of the gyroscope.

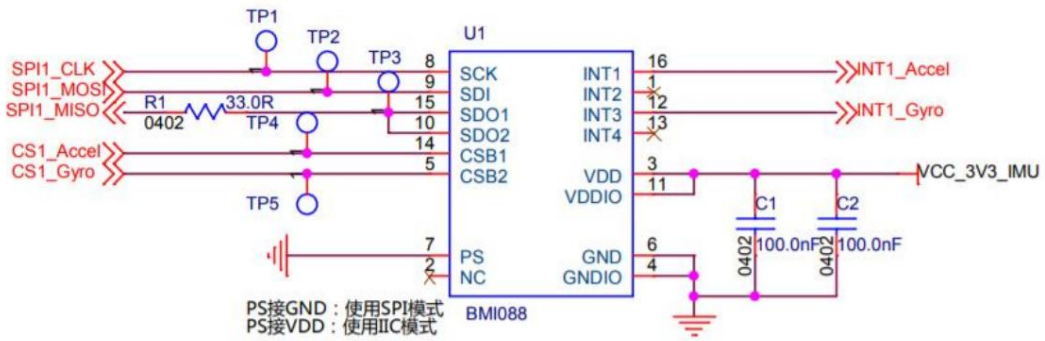
Information can be used to calculate the spatial pose of the object. For the detailed calculation process, please refer to Chapter 18, Pose Solving Tasks.

13.3.3 Introduction to SPI Protocol

BMI088 supports SPI protocol and I2C protocol. It is determined by the level state of PS pin whether it is SPI protocol or I2C protocol.

The internal frame diagram of the BMI088 chip is as follows:





When BMI088 uses SPI protocol, the function of each pin can be seen in the following table:

pin	Features
CSB1ÿCSB2	Connect the SPI chip select signal line, active low; CSB1 is used to select the accelerometer, CSB2 Used to select the gyroscope.
PS	Mode selection pin, BMI088 works in SPI mode when connected to low level
SCK	Connect the SPI clock line
SDI	Data Entry BMI088
SDO1	BMI088 output acceleration data
SDO2	BMI088 output angular velocity data
INT1, INT2 generate interrupt	signal when sending acceleration data
INT3, INT4 generate interrupt	signal when sending angular velocity data

The SPI protocol is a high-speed, full-duplex, synchronous communication bus developed by Motorola, using four wires for communication.

It is easy to use and has the characteristics of high communication speed. Multiple devices can be mounted on the SPI bus, and these devices are divided into

The master device (Master) and the slave device (Slave), the master device controls the slave device through the clock line and the chip select line.

The pins and their functions used by the SPI protocol are shown in the following table

name	Features
SCK (Serial Clock)	SPI is a synchronous communication bus protocol, the master device communicates Provide clock signal to each slave device through SCK
SDI (Serial Data Input) /MISO (Master In Slave Out)	One of the data lines of SPI, the transmission direction is from the device data, the master device receives
SDO (Serial Data Output) /MOSI (Master Out Slave In)	One of the data lines of SPI, the transmission direction is sent by the master device data, received from the device
SS (Slave Select) /CS (Chip Select)	The chip select line of SPI, the master device controls the slave device through the chip select line the working status of the device, select the target that needs to be communicated

SPI is a full-duplex communication protocol. When the master device communicates with the slave device, the sending and receiving at both ends are performed synchronously, that is, the master device.

When the standby and slave devices send data to each other, they are also receiving data from each other.

The communication process of SPI is as follows:

1. The master device pulls down the SS/CS chip select of the slave device to be communicated.

2. The master device provides the clock signal required for synchronous communication to the slave device through SCK,

3. The master device sends 8-bit data to the slave device through MOSI, and at the same time receives the 8-bit data from the slave device through MISO according to.

4. After the communication is over, the master device pulls up the SS/CS chip select.

The detailed SPI bus connection method and communication process introduction can be found in the advanced learning section.

13.4 Program Learning

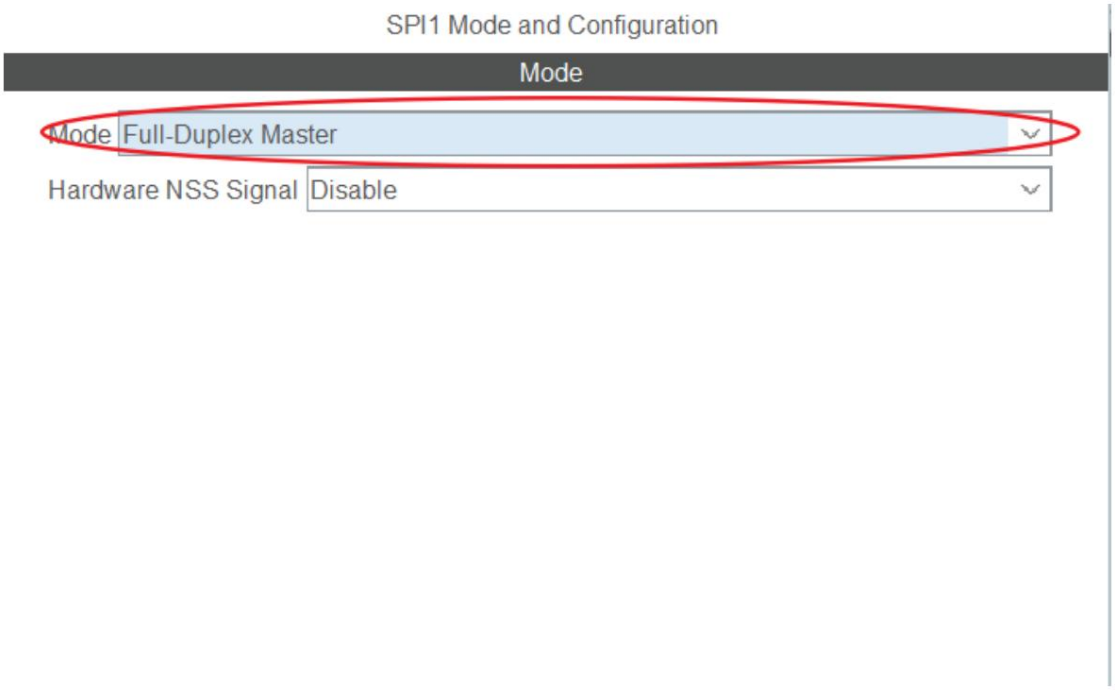
13.4.1 SPI configuration in cubeMX

This section will learn the configuration of SPI in cubeMX.

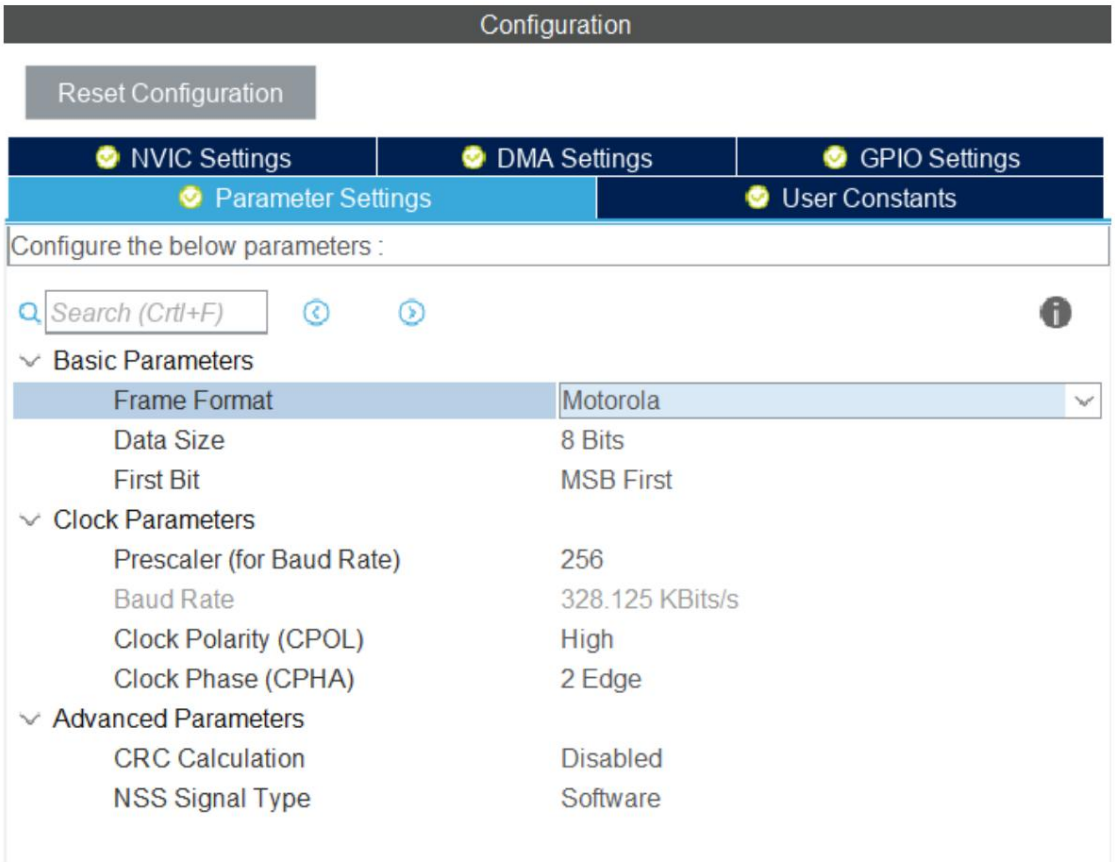
1. First select SPI1 under the Connectivity tab to enter the configuration page of SPI1;

2. In the Mode page, select the mode as Full-Duplex Master, that is, let stm32 work in full-duplex SPI

Next, as a host, set the hardware chip select signal Hardware NSS Signal to Disable;



3. On the configuration page, make the following settings:



The meaning of important parameters in the configuration page can be seen in the following table:

parameter	Features
Frame Format	Set the SPI frame format, optional Motorola format and TI (Texas Instruments) format
Data Size	The data length in a frame, generally selected as 8bit
First Bit	Whether to send the most significant MSB or the least significant LSB first
Prescaler	Bus frequency division value, set the communication clock frequency of SPI
Clock Polarity & Clock Phase is used to set the timing function of SPI	
CRC Calculation	CRC check calculation function
NSS Signal Type	Chip select signal type

13.4.2 Introduction to registers of BMI088

This section will introduce important registers in BMI088. Included in the BMI088 are controls for gyroscopes, accelerometers and

Control registers for the communication interface, data registers for storing sensor data, and registers for storing sensor IDs. when to

When communicating, first read the ID register, and after confirming that the ID is correct, first write data into the control register, and set the transmission parameters.

The working state of the sensor, and then read the data from the data register.

The accelerometer can use the software reset register to restore the value of all registers, for the address of 0x7E

Writing 0xB6 to the ACC_SOFTRESET register will restore all registers in the accelerometer to their default values (generally

is 0). During initialization, it is reset through the software reset register.

0x7E	ACC_SOFTRESET	0x00	softreset_cmd (0xb6)

Similarly, when the gyroscope is initialized, write to the software reset register GYRO_SOFTRESET at address 0x14

0xB6 to reset.

0x14	GYRO_SOFTRESET	N/A	softreset

The acceleration values of the three axes of the accelerometer (3 data, each data length is 16 bits) are divided into high-order eight bits and eighth bits to be stored separately.

Stored in 6 octet registers ACC_Z_MSB to ACC_X_LSB at addresses 0x12 to 0x17.

0x17	ACC_Z_MSB	0x00	acc_z[15:8]
0x16	ACC_Z_LSB	0x00	acc_z[7:0]
0x15	ACC_Y_MSB	0x00	acc_y[15:8]
0x14	ACC_Y_LSB	0x00	acc_y[7:0]
0x13	ACC_X_MSB	0x00	acc_x[15:8]
0x12	ACC_X_LSB	0x00	acc_x[7:0]

The speed values of the three axes of the gyroscope (3 data, each data length is 16 bits) are also divided into high-order eight bits and eighth bits to be stored separately.

Stored in 6-bit octet registers at addresses 0x02 to 0x07, RATE_Z_MSB to RATE_X_LSB.

0x07	RATE_Z_MSB	N/A	rate_z[15:8]
0x06	RATE_Z_LSB	N/A	rate_z[7:0]
0x05	RATE_Y_MSB	N/A	rate_y[15:8]
0x04	RATE_Y_LSB	N/A	rate_y[7:0]
0x03	RATE_X_MSB	N/A	rate_x[15:8]
0x02	RATE X LSB	N/A	rate_x[7:0]

In addition, the accelerometer also has a data register for storing the temperature value. The total length of the temperature data is 11 bits, and the first 3 bits are stored in the

The sixth to eighth bits in the eight-bit TEMP_LSB register, and the last eight bits are stored in the eight-bit register TEMP_MSB.

in the memory.

0x23	TEMP_LSB	0x00	temperature[2:0]	-
0x22	TEMP_MSB	0x00	temperature[10:3]	

In order to verify that the accelerometer and gyroscope are working properly, their respective ID registers need to be read. The ID register memory has a special

The user can compare the read register value with the standard ID value.

The ID register address of the accelerometer is 0x00, the standard ID value is 0x1E, the ID register address of the gyroscope is 0x00,

Standard ID value is 0x0F

0x00	ACC_CHIP_ID	0x1E	acc_chip_id

0x00	GYRO_CHIP_ID	0x0F	gyro_chip_id

13.4.3 BMI088 read function introduction

This program reads the angular velocity and acceleration data through the BMI088_read function. The functions of this function include:

There are two parts to register reading and data splicing.

When reading the acceleration data, first select the accelerometer through the BMI088_accel_read_muli_reg function, the chip select signal,

Then use SPI to read the data in the accelerometer data register into buf, and complete the data according to the upper eight bits and the eighth bit.

stitching.

When reading the angular velocity data, first select the gyroscope through the BMI088_gyro_read_muli_reg function, the chip select signal,

Then use SPI to read the ID of the angular velocity meter and the data in the data register into buf. After the detection ID is correct, the data is

The upper eight bits and the lower eight bits are spliced.

When reading temperature data, read the data of the temperature register in the accelerometer, and perform data splicing.

```
void BMI088_read(fp32 gyro[3], fp32 accel[3], fp32 *temperate)
{
    uint8_t buf[8] = {0, 0, 0, 0, 0, 0};
    int16_t bmi088_raw_temp;

    BMI088_accel_read_muli_reg(BMI088_ACCEL_XOUT_L, buf, 6);

    bmi088_raw_temp = (int16_t)((buf[1]) << 8) | buf[0];
    accel[0] = bmi088_raw_temp *   BMI088_ACCEL_SEN;
    bmi088_raw_temp = (int16_t)((buf[3]) << 8) | buf[2];
    accel[1] = bmi088_raw_temp *   BMI088_ACCEL_SEN;
    bmi088_raw_temp = (int16_t)((buf[5]) << 8) | buf[4];
    accel[2] = bmi088_raw_temp *   BMI088_ACCEL_SEN;

    BMI088_gyro_read_muli_reg(BMI088_GYRO_CHIP_ID, buf, 8);
    if(buf[0] == BMI088_GYRO_CHIP_ID_VALUE)
    {
        bmi088_raw_temp = (int16_t)((buf[3]) << 8) | buf[2];
        gyro[0] = bmi088_raw_temp *   BMI088_GYRO_SEN;
        bmi088_raw_temp = (int16_t)((buf[5]) << 8) | buf[4];
        gyro[1] = bmi088_raw_temp *   BMI088_GYRO_SEN;
        bmi088_raw_temp = (int16_t)((buf[7]) << 8) | buf[6];
        gyro[2] = bmi088_raw_temp *   BMI088_GYRO_SEN;
    }
```



```
    BMI088_accel_read_muli_reg(BMI088_TEMP_M, buf, 2);

    bmi088_raw_temp = (int16_t)((buf[0] << 3) | (buf[1] >> 5));

    if (bmi088_raw_temp > 1023)
    {
        bmi088_raw_temp -= 2048;
    }

    *temperate = bmi088_raw_temp *          BMI088_TEMP_FACTOR + BMI088_TEMP_OFFSET;
}
```

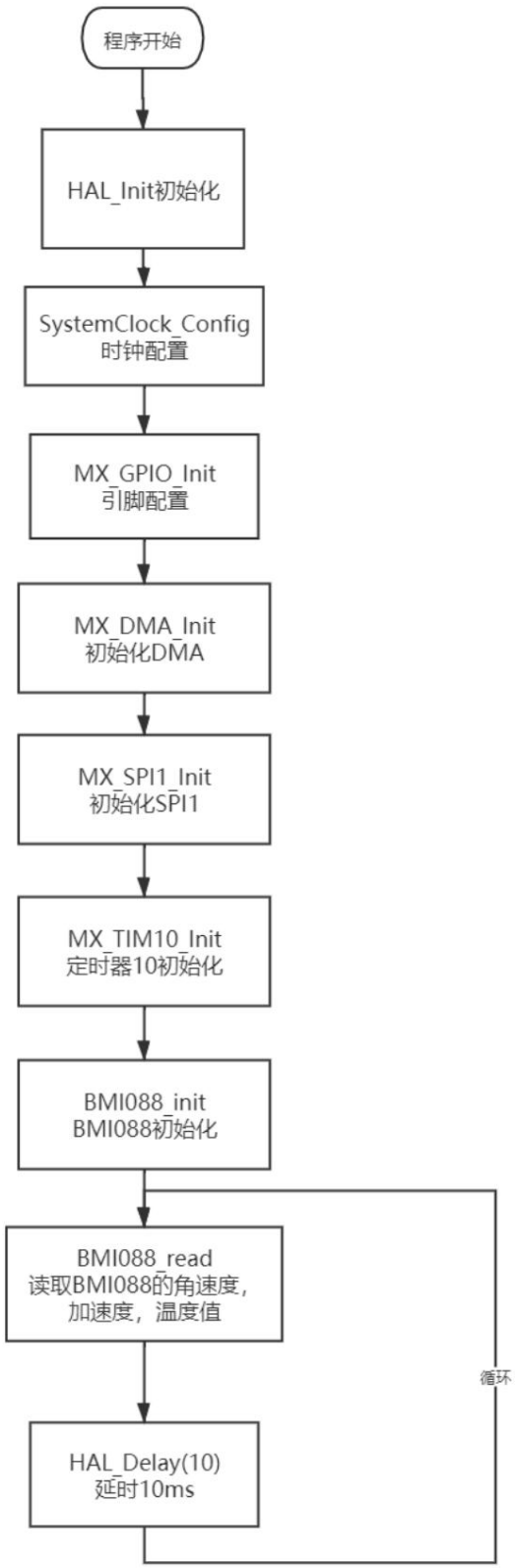
Among them, BMI088_accel_read_muli_reg and BMI088_gyro_read_muli_reg call the SPI of the HAL library

Communication function HAL_SPI_TransmitReceive

```
HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t *pTxData,
uint8_t *pRxData, uint16_t Size, uint32_t Timeout)
```

Function name	HAL_SPI_TransmitReceive
The function of the function	is to communicate between the master and the slave through SPI
return value	HAL_StatusTypeDef, several states defined by HAL library, if this SPI communication is successful, then return HAL_OK
parameter 1	SPI_HandleTypeDef *hspi is the handle pointer of SPI, if it is SPI1, enter it &hspi1, SPI2 input &hspi2
parameter 2	uint8_t *pTxData The first address pointer of the data to be sent
parameter 3	uint8_t *pRxData The first address of the area to receive data
parameter 4	uint16_t Size The length of the data to be sent
parameter 5	uint32_t Timeout maximum sending time

13.4.4 Program Flow



13.4.5 Effect demonstration

Download the program and enter the C-type development board, enter the Debug mode, and view the angular velocity, acceleration and temperature data in the watch window

data, as shown in Fig.

Watch 1

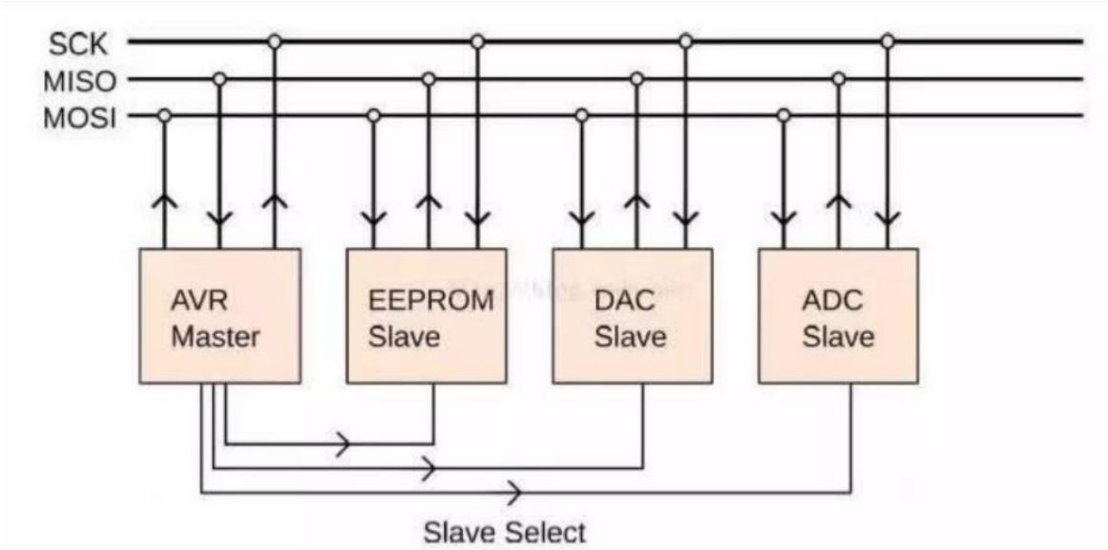
Name	Value	Type
gyro	0x20000040 gyro	float[3]
[0]	0.00639158674	float
[1]	0.00213052891	float
[2]	0.00319579337	float
accel	0x2000004C accel	float[3]
[0]	0.0888461545	float
[1]	-0.439743578	float
[2]	9.74525642	float
temp	28.875	float
<Enter expression>		

13.5 Advanced Learning

When using SPI for communication, the clock pins of each device are mounted on the SCK line together, the input of the master device and each slave

The output of the device is connected to MISO/SDI, the output of the master device and the input of each slave device are connected to MOSI/SDO, and the master device

The GPIO port of the device is connected to the CS of each slave device.

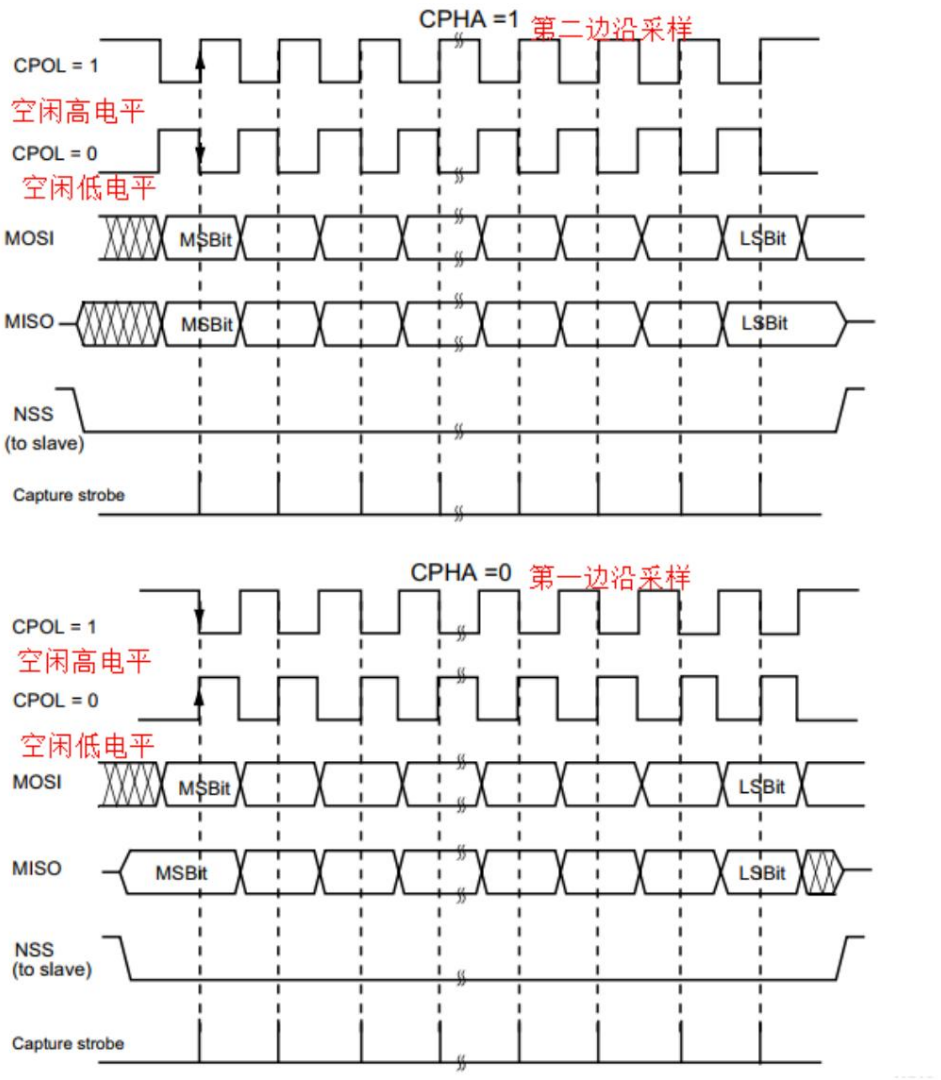


At the beginning of the SPI communication process, if a master device and a slave device want to communicate, the master device will communicate via CS.

The number line selects the slave device, and other unselected slave devices do not participate in this communication.

The timing of the SPI specifies different operating modes, as shown in the following table:

Operating mode	configuration value	model
CPOL idle clock level	0	Low when the clock is idle
	1	High when the clock is idle
CPHA edge sampling position	0	Data is sampled on the first edge
	1	The second edge samples the data



When the master device and the slave device communicate, both parties use a shift register to store and send data at the same time.

When the master device (slave device) completes one-bit transmission, the shift register performs a one-bit shift, and stores the received data in the

Stored in the new space generated by the shift, so that both parties can synchronize and complete data transmission and reception at the same time.

Here, take the master to send the data 10101010 to the slave, and the slave to send the data 01010101 to the master as an example.

The whole process of data transmission is shown, assuming that the shift direction of the master and slave shift registers is to the left.

sequential master shift register		Slave Shift Register
0	10101010	01010101
1	01010100	10101011
2	10101001	01010110
3	01010010	10101101
4	10100101	01011010
5	01001010	10110101
6	10010101	01101010
7	00101010	11010101
8	01010101	10101010

It can be seen that in step 8, the data to be sent by the master data all enters the shift register of the slave, and the data to be sent by the slave All enter the host register, the two parties complete the data exchange, after that the host cancels the chip selection of the slave, and continues communicate with the next object.

13.6 Course Summary

The gyroscope module and accelerometer module are commonly used sensors for attitude calculation and gimbal speed loop control. Introduction to this lesson Introduced the BMI088 six-axis IMU module, and learned how to drive it and read data through the SPI protocol.

14. CAN control RM motor

14.1 Key points of knowledge

ÿ Introduction to CAN protocol

ÿ Instructions for use of RM motor

ÿ CAN baud rate calculation in cubeMX

ÿ CAN transmit and receive interrupt introduction

14.2 Course Content

This lesson will learn how to control the RM motor through CAN communication, which is a common fieldbus communication

In this way, most of RoboMaster motors are controlled using CAN communication.

14.3 Basic Learning

14.3.1 Introduction to CAN Protocol

CAN is the abbreviation of Controller Area Network (CAN).

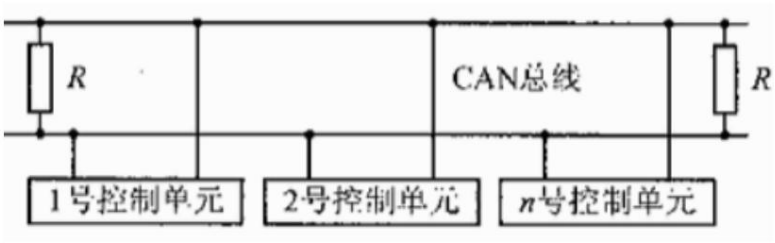
Developed by the well-known German BOACH company, and eventually became an international standard (ISO11898), CAN is an international application

One of the most extensive fieldbuses. In North America and Western Europe, the CAN bus protocol has become the standard for automotive computer control systems and embedded systems.

It is the standard bus of the embedded industrial control area network, and has CAN as the underlying protocol, which is specially designed for large trucks and heavy industrial machinery vehicles.

J1939 protocol for vehicle design.

The CAN bus is composed of CAN_H and CAN_L lines, and each device is mounted on the bus together.



RoboMaster series motors also use CAN protocol for communication. CAN protocol is more complicated, a complete data frame

It consists of the various parts in the following diagram:

Bus Idle Frame	Start	Arbitration Field	Control Field	Data Field	CRC Field	Response Field	Frame End	Frame Interval

Here we focus on the content of the CAN arbitration field and data field. Like the I2C bus, each is mounted on the CAN bus

The CAN on the bus has its own unique ID. Whenever a device sends a frame of data, other devices on the bus will check this.

Whether the ID is the object that needs to receive data, if so, receive the data of this frame, if not, ignore it.

The ID is stored in the arbitration field at the head of the data frame. The ID of CAN is divided into two types: standard ID and extended ID. The length of standard ID is

is 11 bits. If there are too many devices and the standard ID is not enough, you can use the extended ID. The length of the extended ID is 29

Rank.

	Court of Arbitration											
start of frame	Identifier (ID) 11 bits										RTR	control field
	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

After judging that the data of this frame can be received by ID, the DLC in the control field specifies the length of the data of this frame, and the data in the data field

The size of the data is 8 Byte, that is, 8 8-bit data. The effective number of data to be transmitted in one data frame of the CAN bus

It is actually this 8Byte.

control field				data field				CRC field
other	DLC (length)			data				
	Bit 3	Bit 2	Bit 1	Bit 0	Byte 7 …		Byte 1	Byte 0

14.3.2 RM motor usage

When using RM series motors, download the data sheets of the motors and ESCs on the RM official website, and find the

For the content related to CAN communication, here takes the RM3508 motor as an example.

In the booklet, you can find the following:

标识符：0x200　　帧格式：DATA
帧类型：标准帧　　DLC：8 字节

数据域	内容	电调 ID
DATA[0]	控制电流值高 8 位	1
DATA[1]	控制电流值低 8 位	
DATA[2]	控制电流值高 8 位	2
DATA[3]	控制电流值低 8 位	
DATA[4]	控制电流值高 8 位	3
DATA[5]	控制电流值低 8 位	
DATA[6]	控制电流值高 8 位	4
DATA[7]	控制电流值低 8 位	

This is the format of the ESC receiving message, that is, if you want to send data to ESCs No. 1 to No. 4 to control the output current of the motor, thus

When controlling the motor speed, it is necessary to set the ID of the sent CAN data frame to 0x200 according to the content in the table, and the data field

The 8Byte data in the ESC is filled in the order of the upper eight bits and the eighth bit of ESC 1 to 4, and the frame format and DLC are also in accordance with the table

The content is set, and finally the data is sent.

And when you want to receive the data sent by the ESC, follow the table below:

标识符：0x200 + 电调 ID
（如：ID 为 1，该标识符为 0x201）
帧类型：标准帧
帧格式：DATA
DLC：8 字节

数据域	内容
DATA[0]	转子机械角度高 8 位
DATA[1]	转子机械角度低 8 位
DATA[2]	转子转速高 8 位
DATA[3]	转子转速低 8 位
DATA[4]	实际转矩电流高 8 位
DATA[5]	实际转矩电流低 8 位
DATA[6]	电机温度
DATA[7]	Null

First, according to the received ID, determine which ESC sent the data. The manual specifies the ID of No. 1 ESC.

0x201, number 2 is 0x202, number 3 is 0x203, and number 4 is 0x204. After judging the data source, you can press

Decode according to the data format in the manual, and get the rotor mechanical angle of the motor by splicing the upper eight bits and the eighth bit.

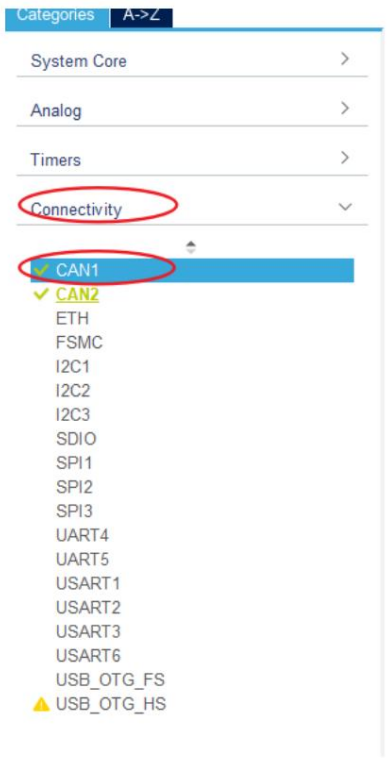
Rotor speed, torque current, motor temperature and other data.

14.4 Program Learning

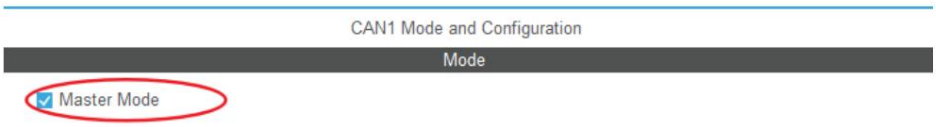
14.4.1 Configuration of **CAN** in **cubeMX**

This section will introduce the calculation method of CAN baud rate in cubeMX.

1. First enable CAN1 in cubeMX, open CAN1 under Connectivity, and configure CAN1.

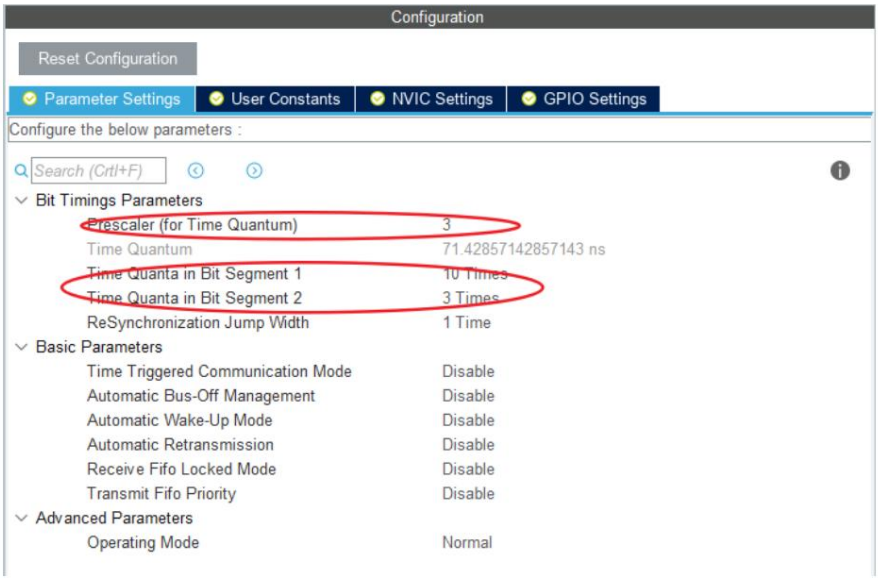


2. In Mode, tick Master Mode.



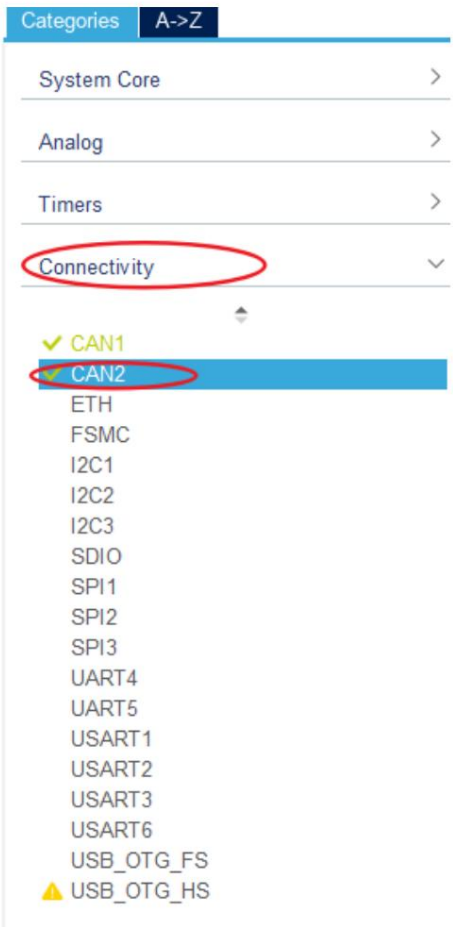
In the Configuration interface, you need to configure the baud rate of CAN. After setting the frequency division coefficient (Prescaler), cubeMX will automatically complete the calculation of Time Quantum (abbreviated as tq), multiply tq by tBS1 (Time Quanta in Bit Segment 1)ÿtBS2 (Time Quanta in Bit Segment 1)ÿRJW (ReSynchronization Jump Width) is exactly 1 microsecond, corresponding to a baud rate of 1M, which is the highest communication rate supported by the CAN bus.

$$71.42857142857143 \text{ ÿ } (10 + 3 + 1) = 1000 = 1$$

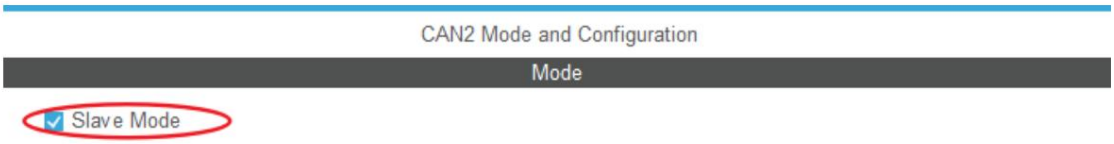


For the detailed calculation principle of CAN baud rate, see the advanced learning section.

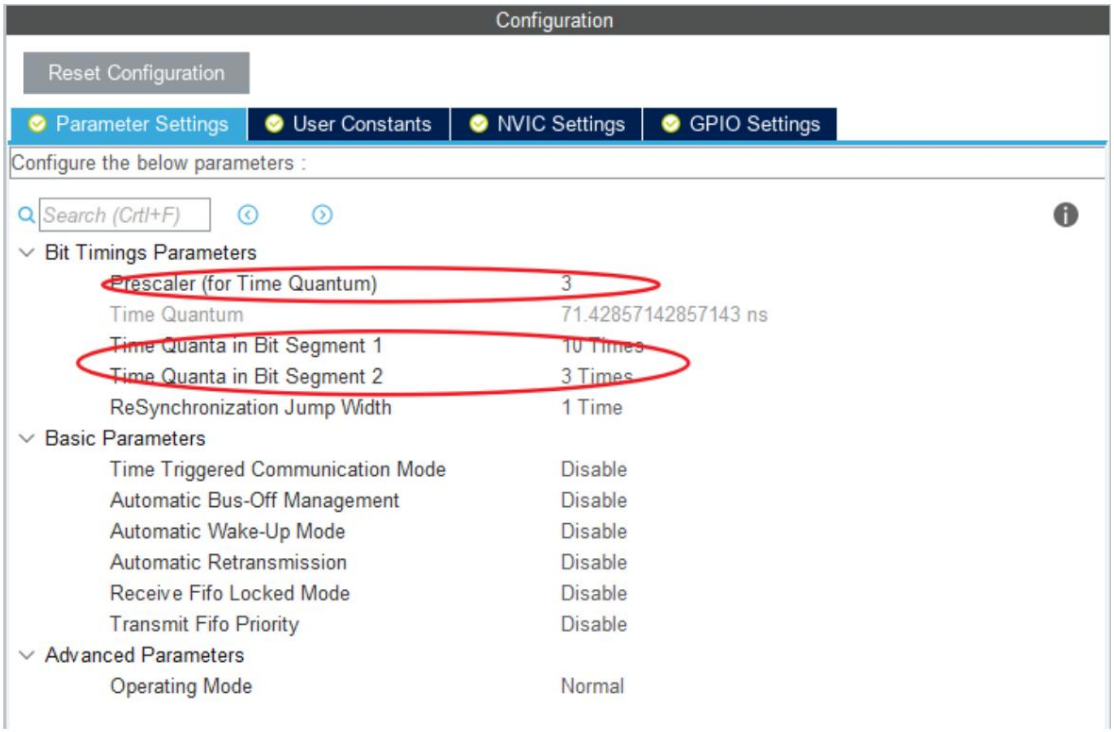
1. The configuration and calculation of CAN2 are similar to CAN1. First open CAN2 in cubeMX and open Connectivity Under CAN2, configure CAN2.



2. In Mode, check Slave Mode.



The baud rate configuration of CAN2 is the same as CAN1, as shown in the figure:



14.4.2 Introduction of CAN transmission function

This section will introduce the transmit function of CAN. The CAN_cmd_chassis function and

The CAN_cmd_gimbal function is used to send CAN signals to the chassis motor and gimbal motor to control the motor movement.

The input of the CAN_cmd_chassis function is the expected drive current value of motor1 to motor4 motor1 to motor4, the function

The number will split the expected value into the upper eight bits and the eighth bit, put it into the data field of the 8Byte CAN, and then add the ID

(CAN_CHASSIS_ALL_ID 0x200), frame format, data length and other information to form a complete CAN data

frame, sent to each ESC.

```
void CAN_cmd_chassis(int16_t motor1, int16_t motor2, int16_t motor3, int16_t motor4)
{
        uint32_t send_mail_box;
        chassis_tx_message.StdId = CAN_CHASSIS_ALL_ID;
        chassis_tx_message.IDE = CAN_ID_STD;
        chassis_tx_message.RTR = CAN_RTR_DATA;
```



```
        chassis_tx_message.DLC = 0x08;
        chassis_can_send_data[0] = motor1 >> 8;
        chassis_can_send_data[1] = motor1;
        chassis_can_send_data[2] = motor2 >> 8;
        chassis_can_send_data[3] = motor2;
        chassis_can_send_data[4] = motor3 >> 8;
        chassis_can_send_data[5] = motor3;
        chassis_can_send_data[6] = motor4 >> 8;
        chassis_can_send_data[7] = motor4;

        HAL_CAN_AddTxMessage(&CHASSIS_CAN, &chassis_tx_message,
chassis_can_send_data, &send_mail_box);
}
```

The HAL library provides the function HAL_CAN_AddTXMessage to implement CAN transmission

```
HAL_StatusTypeDef HAL_CAN_AddTxMessage(CAN_HandleTypeDef *hcan,
CAN_TxHeaderTypeDef *pHeader, uint8_t aData[], uint32_t *pTxMailbox)
```

Function name	HAL_CAN_AddTXMessage
Function	Function to send a piece of data through the CAN bus
return value	HAL_StatusTypeDef, several states defined by the HAL library, if the CAN transmission is successful this time, then return HAL_OK
parameter 1	CAN_HandleTypeDef *hcan, the handle pointer of can, if it is can1, enter it &hcan1, can2 enter &hcan2
parameter 2	CAN_TxHeaderTypeDef *pHeader, the structure of the CAN data frame information to be sent refers to Needle, including CAN ID, format and other important information
parameter 3	uint8_t aData[], the name of the array loaded with the data to be sent
parameter 4	uint32_t *pTxMailbox, used to store the mailbox number used for CAN transmission

The function of the CAN_cmd_gimbal function is to send control signals to the gimbal motor and the transmitter motor. The input parameters are yaw axis motor, pitch axis motor, and the expected value of the drive current of the launch mechanism motor yaw, pitch, shoot (rev is the guarantee left value), the function will split the expected value into the upper eight bits and the eighth bit, put it into the 8Byte CAN data field, and then add the Add ID (CAN_GIMBAL_ALL_ID 0x1FF), frame format, data length and other information to form a complete CAN Data frame, sent to each ESC.

```
void CAN_cmd_gimbal(int16_t yaw, int16_t pitch, int16_t shoot, int16_t rev)
{
        uint32_t send_mail_box;
        gimbal_tx_message.StdId = CAN_GIMBAL_ALL_ID;
        gimbal_tx_message.IDE = CAN_ID_STD;
        gimbal_tx_message.RTR = CAN_RTR_DATA;
        gimbal_tx_message.DLC = 0x08;
        gimbal_can_send_data[0] = (yaw >> 8);
        gimbal_can_send_data[1] = yaw;
        gimbal_can_send_data[2] = (pitch >> 8);
        gimbal_can_send_data[3] = pitch;
        gimbal_can_send_data[4] = (shoot >> 8);
        gimbal_can_send_data[5] = shoot;
        gimbal_can_send_data[6] = (rev >> 8);
        gimbal_can_send_data[7] = rev;
        HAL_CAN_AddTxMessage(&GIMBAL_CAN, &gimbal_tx_message,
gimbal_can_send_data, &send_mail_box);
}
```

14.4.3 Introduction of CAN Receive Interrupt Callback

This section will introduce the CAN receive interrupt callback. The HAL library provides the CAN receive interrupt callback function HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan), whenever CAN completes a When the frame data is received, it will trigger a CAN receive interrupt processing function, and the receive interrupt function completes some registers. After processing, the CAN receive interrupt callback function will be called.

In this program, in the interrupt callback function, first determine whether the ID of the receiving object is the required ESC to be received. data coming. After the judgment is completed, decode and load the corresponding motor data into the motor information array motor_chassis

in each corresponding bit.

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
        CAN_RxHeaderTypeDef rx_header;
        uint8_t rx_data[8];

        HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &rx_header, rx_data);

        switch (rx_header.StdId)
        {
                case CAN_3508_M1_ID:
                case CAN_3508_M2_ID:
                case CAN_3508_M3_ID:
                case CAN_3508_M4_ID:
                case CAN_YAW_MOTOR_ID:
                case CAN_PIT_MOTOR_ID:
                case CAN_TRIGGER_MOTOR_ID:
                {
                        static uint8_t i = 0;
                        //get motor id
                        i = rx_header.StdId - CAN_3508_M1_ID;
                        get_motor_measure(&motor_chassis[i], rx_data);
                        break;
                }
                default:
                {
                        break;
                }
        }
}
```

When receiving, the receiving function HAL_CAN_GetRxMessage provided by the HAL library is called

```
HAL_StatusTypeDef HAL_CAN_GetRxMessage(CAN_HandleTypeDef *hcan, uint32_t RxFifo,
CAN_RxHeaderTypeDef *pHeader, uint8_t aData[])
```

Function name	HAL_CAN_GetRxMessage
Function	Function to receive the data sent on the CAN bus
return value	HAL_StatusTypeDef, several states defined by the HAL library, if the CAN reception is successful this time, then return HAL_OK
parameter 1	CAN_HandleTypeDef *hcan, the handle pointer of can, if it is can1, enter it &hcan1, can2 enter &hcan2
parameter 2	uint32_t RxFifo, the CAN receiving FIFO number used when receiving, generally CAN_RX_FIFO0
parameter 3	CAN_RxHeaderTypeDef *pHeader, the structure that stores the received CAN data frame information Body pointer, including CAN ID, format and other important information
parameter 4	uint8_t aData[], the name of the array storing the received data

motor_chassis is an array of motor_measure_t type, which contains the motor rotor angle, motor rotor speed,

Control current, temperature and other information.

```
typedef struct
{
    uint16_t ecd;
    int16_t speed_rpm;
    int16_t given_current;
    uint8_t temperate;
    int16_t last_ecd;
} motor_measure_t;
```

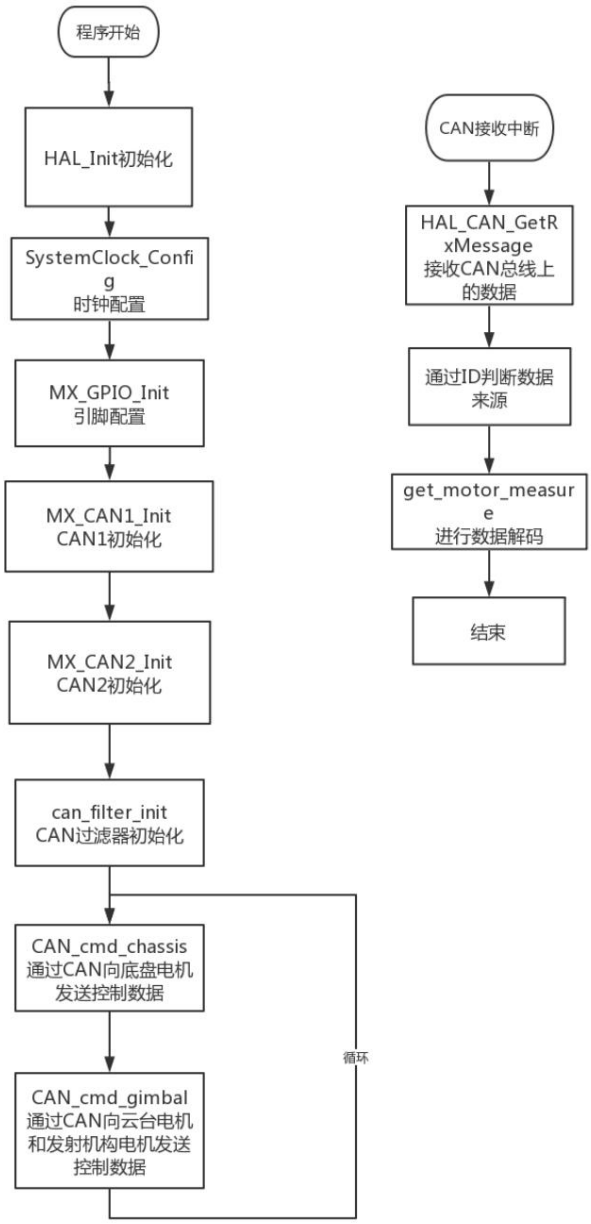
The actual work done by the decoding function is to splicing the received data according to the upper eight bits and the eighth bit, so as to obtain

to the various parameters of the motor.

```
#define get_motor_measure(ptr, data)                                    \
```

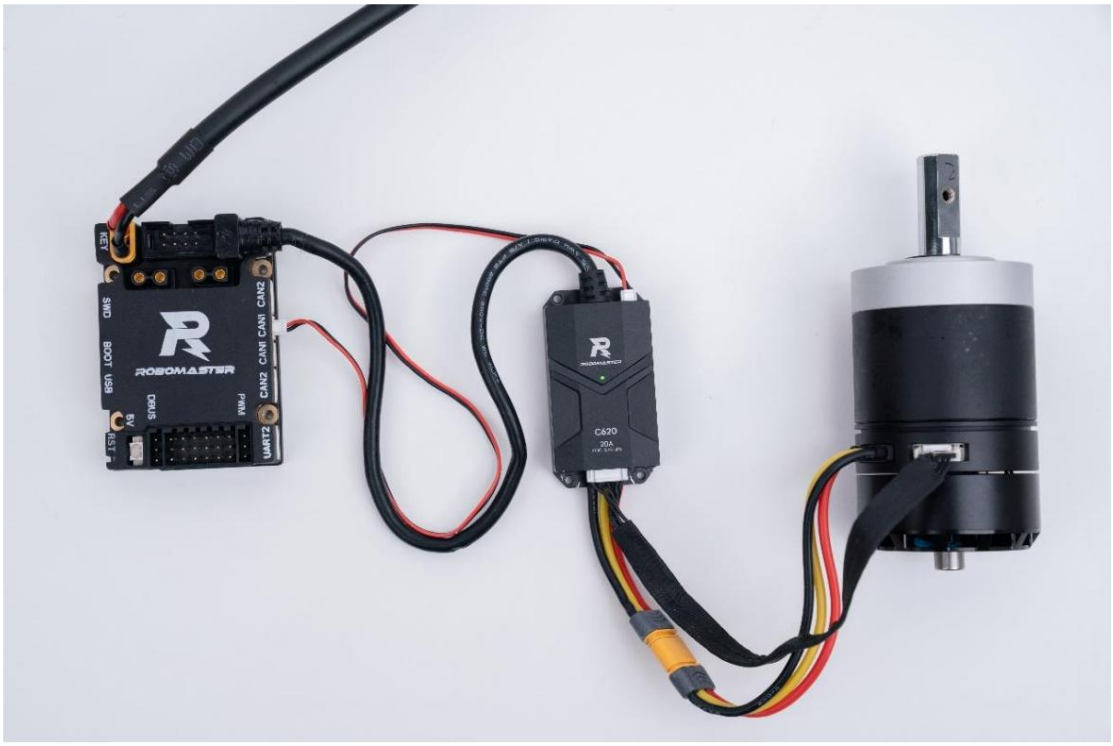
```
{                                                                                    \
    (ptr)->last_ecd = (ptr)->ecd;                                                    \
    (ptr) -> ecd = (uint16_t) ((data) [0] << 8 | (data) [1]);                        \
    (ptr) -> speed_rpm = (uint16_t) ((data) [2] << 8 | (data) [3]);                  \
    (ptr)->given_current = (uint16_t)((data)[4] << 8 | (data)[5]); \
    (ptr) -> temperate = (data) [6];                                                 \
}
```

14.4.4 Program Flow



14.4.5 Effect Demonstration

You can use the remote control to control the motor, and the overall wiring is shown in the figure.



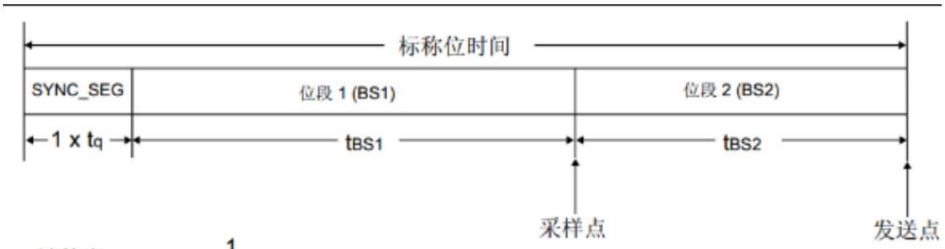
Development board C-type connection to M3508 motor

Development board C-type connection to GM6020 motor

14.5 Advanced Learning

14.5.1 Introduction to CAN Baud Rate

This section will learn the specific calculation method of CAN bus baud rate. The principle of CAN bus baud rate calculation can be seen in the following figure:



$$\text{波特率} = \frac{1}{\text{标称位时间}}$$

$$\text{标称位时间} = 1 \times t_q + t_{BS1} + t_{BS2}$$

其中

$t_{BS1} = t_q \times (TS1[3:0] + 1)$

$t_{BS2} = t_q \times (TS2[2:0] + 1)$

$t_q = (BRP[9:0] + 1) \times t_{PCLK}$

其中 tq 为时间片

t_{PCLK} = APB 时钟的时间周期

According to the above figure, CAN bus baud rate and tq (Time Quantum), tBS1 (Time Quanta in Bit Segment 1)

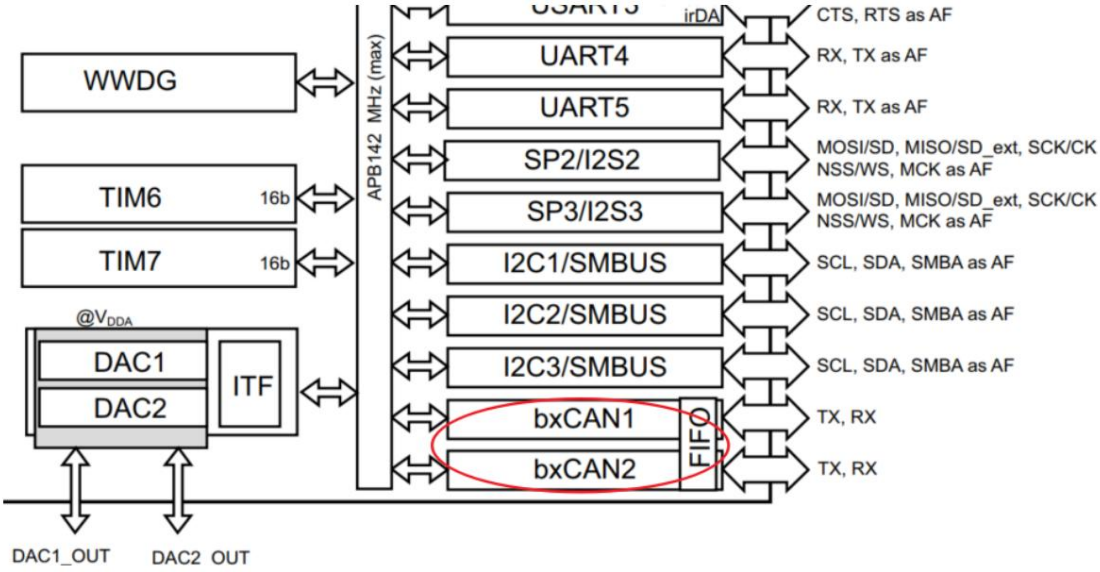
It is directly related to the value of tBS2 (Time Quanta in Bit Segment 1), tq is directly obtained by dividing the bus frequency, tBS1

and tBS2 are amplified to several times of tq by TS1 and TS2. It should be noted that the calculation of tBS1 and tBS2 in the following figure

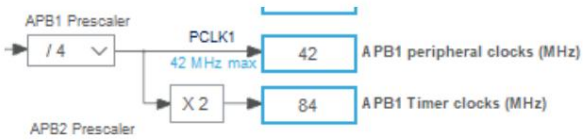
It is obtained by tq*(TS1+1) and tq*(TS2+1), and in cubeMX, we configure Time Quanta in

The value of Bit Segment corresponds to (TS1+1) and (TS2+1). This section takes CAN1 as an example to demonstrate the waveform of CAN

Bit rate calculation process. First of all, it can be known from the data sheet that both CAN1 and CAN2 are mounted on the APB1 bus.



In the Clock Configuration of cubeMX, set the peripheral frequency of the APB1 bus to 42MHz



As shown in the figure below, in Configuration, the frequency division is set to 3, so the value of tq can be calculated first:

$$= \frac{1}{\frac{42}{3}} = 71.42857142857143$$

Set TS1+1 to 10, TS2+1 to 3, and calculate tBS1 and tBS2 from tq, TS1 and TS2

$_1$ = ÿ (1 + 1) = 71.42857142857143 ÿ 10 = 714.2857142857143

$_2$ = ÿ (2 + 1) = 71.42857142857143 ÿ 3 = 214.28571428571428

The full nominal bit time and baud rate can be calculated

$$= + \quad _1 + \quad _2 = 1000$$

$$= \frac{1}{} = \frac{1}{1000} = 1$$

Through the above configuration, the baud rate 1Mbps required for communication with the motor can be configured.

14.5.2 Introduction of DC Motor

DC motors can convert DC power into mechanical energy, and are divided into brushed DC motors and brushless DC motors according to whether there is a brush mechanism.

The basic formula for a DC motor is as follows:

$$U = E + IR \qquad (14\text{-}1)$$

$$E = C \ddot{y} n \quad (14\text{-}2)$$

$$T = C \ddot{y} I \quad (14\text{-}3)$$

$$C = 9.55C \quad (14\text{-}4)$$

in:

ÿ U is the voltage across the motor armature,

ÿ E is the induced voltage in the rotor,

ÿ I is the motor armature current,

ÿ R is the motor resistance,

ÿ C is the potential constant of the motor,

ÿ ÿ is the magnetic flux of the motor,

ÿ n is the speed of the motor, in RPM,

ÿ T is the torque of the motor,

ÿ C is the torque constant of the motor.

Arrange to get:

$$n = \frac{U}{C \ddot{y}} \cdot \frac{R}{C C \ddot{y}2} \quad (14\text{-}5)$$

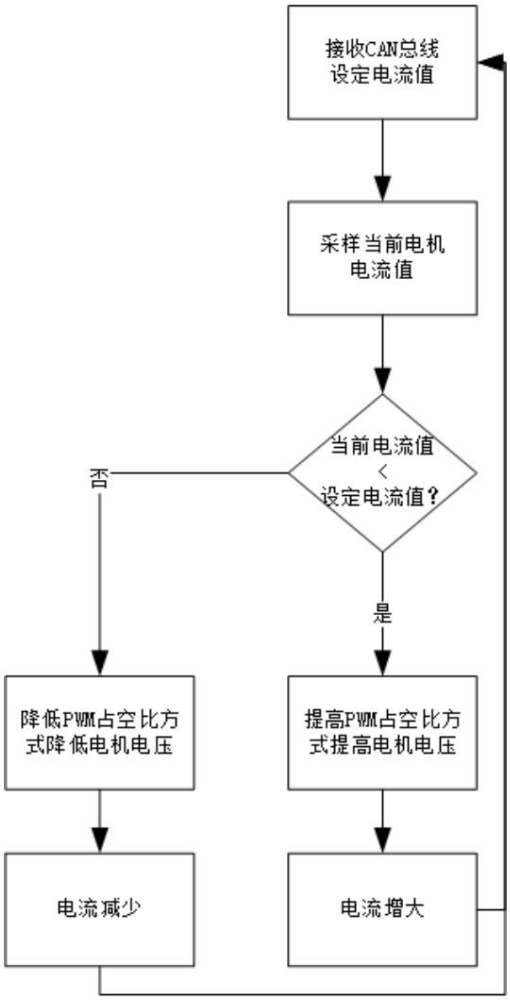
It can be concluded from 14-3 that the motor torque is proportional to the motor current, and from 14-5 it can be concluded that when the load torque is constant, the motor rotates

The speed is proportional to the motor voltage.

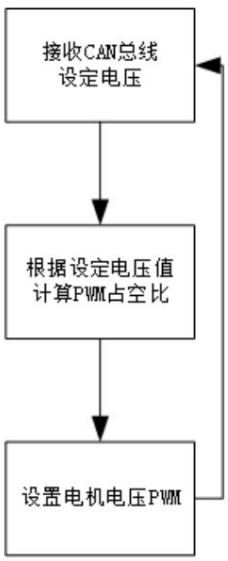
In the RoboMaster series motors, the GM6020 motor controls the motor voltage through the CAN bus, and the M3508 motor is

Motor current is controlled via CAN bus. Therefore, when the M3508 motor receives the current set value sent by the CAN bus,

A set of control algorithms will be used to ensure that the motor current is constant. The whole process is shown in the figure below.



After the GM6020 motor receives the voltage value set by the CAN bus, it will adjust the motor voltage according to the set voltage value.



Compared with the GM020 motor, the M3508 motor has more current comparison parts, and the motor voltage is adjusted through a set of control algorithms to ensure

It proves that the motor current is constant. Since the motor torque is proportional to the motor current, the motor torque is controlled to be constant. M3508 is empty

When the load starts, the motor load torque is small, even if a small current control value is set, the motor will accelerate quickly until the motor

The voltage reaches the maximum voltage, after which the current gradually decreases until the maximum speed.

14.6 Course Summary

CAN communication is a common field bus communication method, and RM motors are controlled by CAN communication. I am in this class

They learned the data frame format of the CAN protocol, the calculation and configuration method of the baud rate, and learned how to use the HAL library

function to send and receive CAN, and communicate with the ESC, so as to control the rotation of the RM motor and receive its

feedback data.

15. freeRTOS blinking LED

15.1 Key points of knowledge

ÿ Introduction to the operating system

ÿ Configuration of freeRTOS in cubeMX

ÿ How tasks are created

ÿ Task switching process

15.2 Course Content

This lesson will introduce the basic concepts of embedded operating systems and introduce a popular embedded operating system, freeRTOS, and will also

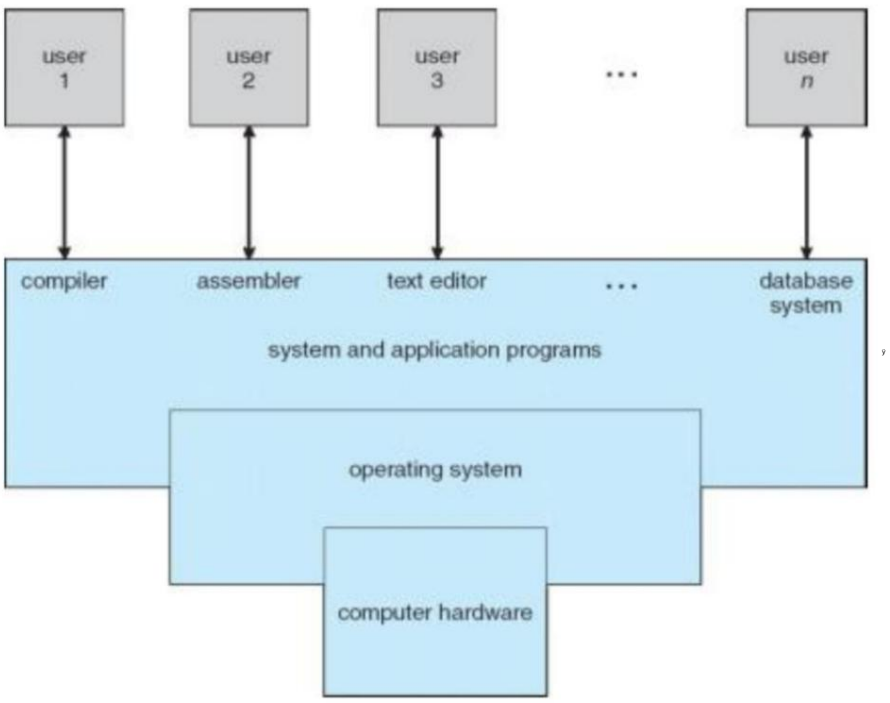
Learn how to configure freeRTOS through cubeMX, and create your own tasks to achieve LED blinking effects.

15.3 Basic Learning

15.3.1 Introduction to Operating Systems

The essence of an operating system is a software that helps users manage functions. operating system running

On top of the hardware, management tasks such as resource allocation are performed for software that does other work.



Generally speaking, the development method of single-chip microcomputer without operating system is called "bare metal development".

Design loop, interrupt, timing and other functions to control the execution sequence of each task.

When using the operating system for development, you only need to create tasks, and the operating system will automatically execute the tasks according to some specific mechanisms.

Running and switching of line tasks.

In addition to task management, the operating system can also provide many functions, such as communication between tasks, synchronization, tasks

stack management, control the mutually exclusive access of tasks to important resources, etc.

Because the resources of the single-chip microcomputer are relatively small, it is obviously impossible to run computer operating systems such as Windows and MacOS.

A specially designed embedded real-time operating system (RTOS) runs on the microcontroller.

The freeRTOS to be introduced in this lesson is one of them. Other common embedded real-time operating systems include

uCOSÿ, RTThread, etc. The freeRTOS operating system is a completely free operating system with open source code and portable

It has the characteristics of planting, cutting, and flexible scheduling strategy, which can be easily transplanted to various microcontrollers to run, and has a huge number of

community and ecology.

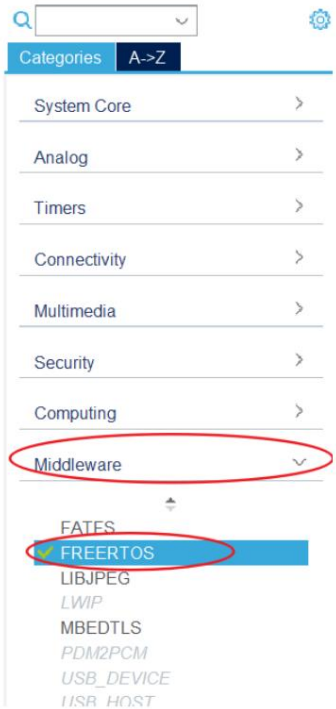
The switching scheduling of each task by freeRTOS needs to follow certain rules, which are described in detail in the Advanced Learning section.

15.4 Program Learning

15.4.1 Configuration of freeRTOS in **cubeMX**

This section will learn how to configure freeRTOS through cubeMX.

First open the Middleware tab, select the FREERTOS option, and enter the FREERTOS configuration page.

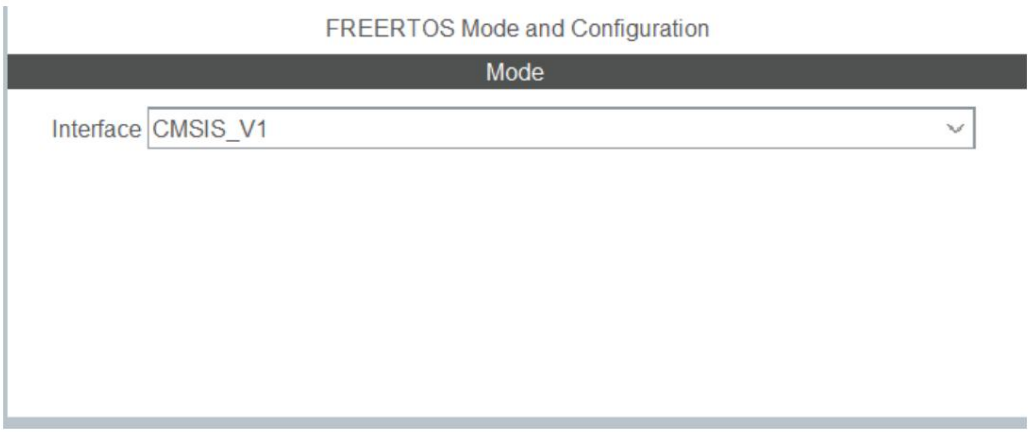


Under the Mode page, select the version of Interface, here select CMSIS_V1. CMSIS is provided by Keil

A special set of function interfaces, which encapsulates the functions of freeRTOS to make it easier to use.

When freeRTOS is used, there is no need to directly call the functions of freeRTOS, just call the functions provided by CMSIS.

The function is enough. Currently, CMSIS has two versions, V1 and V2.



The above method can complete the opening of freeRTOS. After the code is generated, freeRTOS will be automatically ported to

under construction. Then configure freeRTOS under the Configuration page

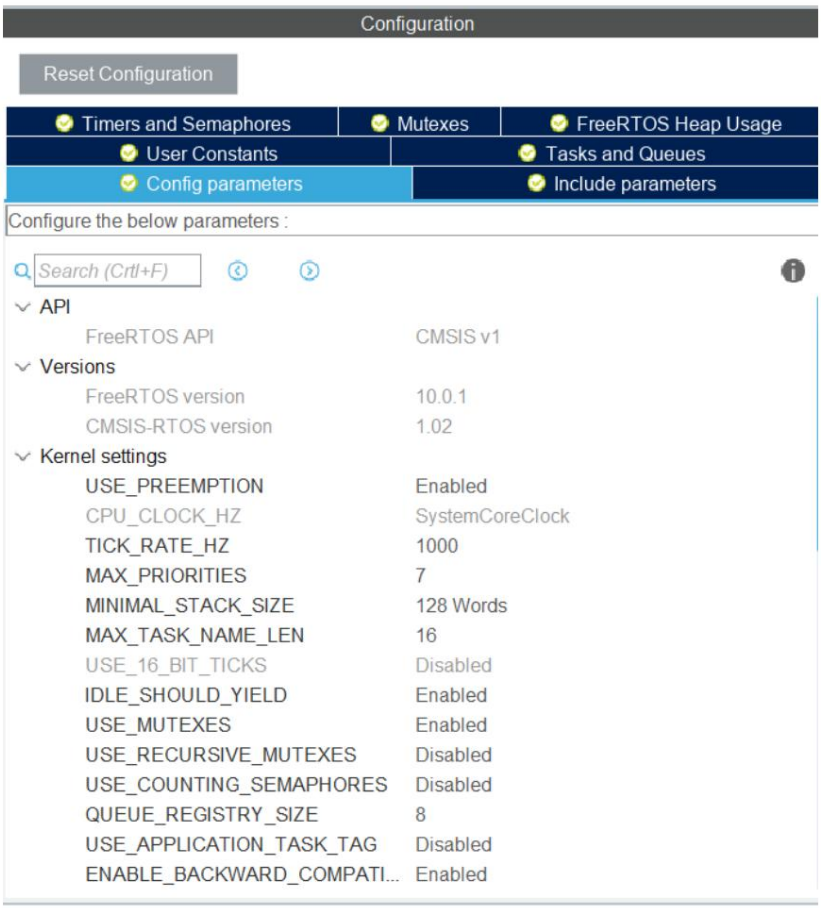
On the page, you can configure some important properties of freeRTOS, including whether to support the preemption mechanism, freeRTOS

System clock rate, maximum number of priorities, minimum task stack size, maximum task name length, etc. For some heavier

The list of required configurations is as follows:

name	Features
USE_PREEMPTION	Whether to support preemption mechanism, if yes, set it to Enabled

TICK_RATE_HZ	The system clock rate, according to which the clock is used for each task in freeRTOS Execution timing, set to 1000Hz, the minimum scheduling time of each task is 1ms
MAX_PRIORITIES	Maximum number of priorities, default is 7
MINIMAL_STACK_SIZE	The minimum task stack size. Whenever a task is created, it needs to be divided into Equipped with a certain size of stack space, the variables that the task needs to use are stored in the stack in space. The default minimum value is 128 words.
MAX_TASK_NAME_LEN	Maximum task name length. When creating tasks, you need to name each task a name For identification, this name can be represented by a string, this parameter specifies the word The upper limit of the string length, the default is 16

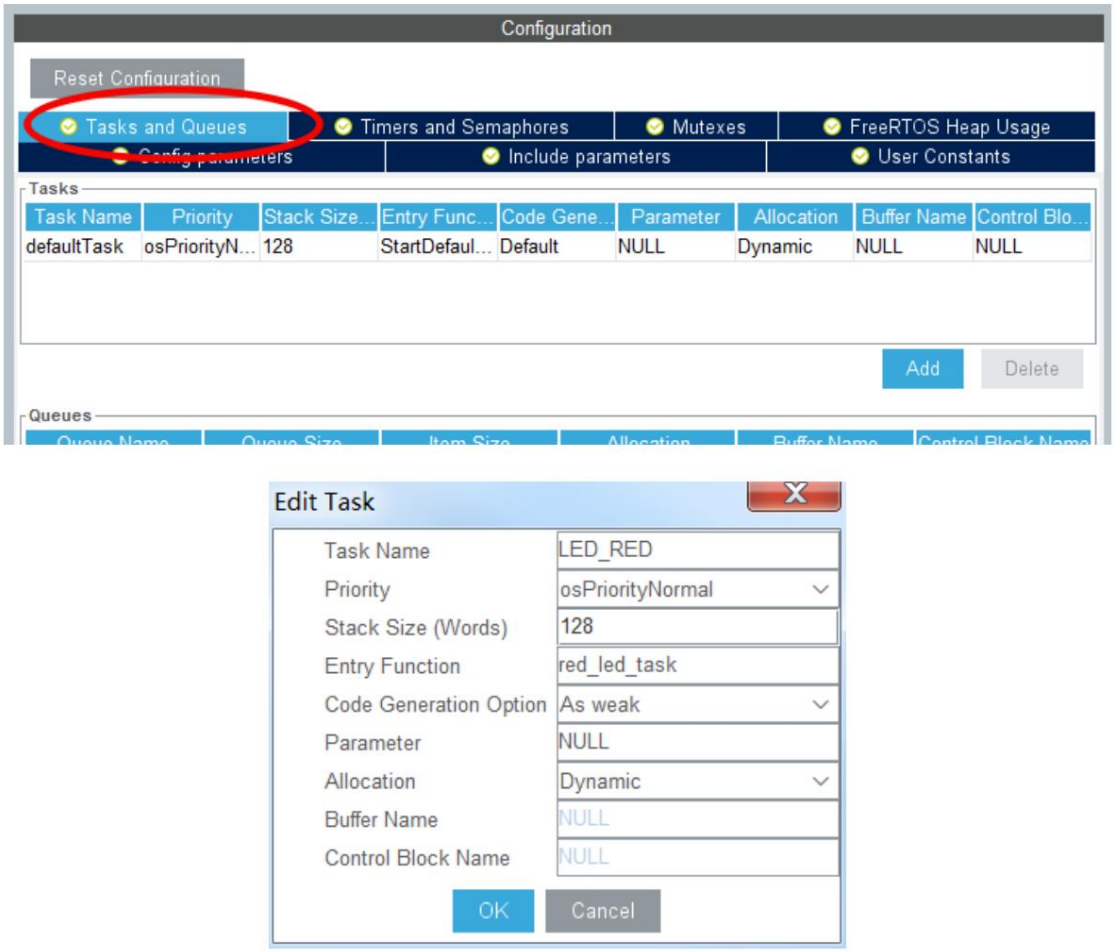


15.4.2 Creating tasks in cubeMX

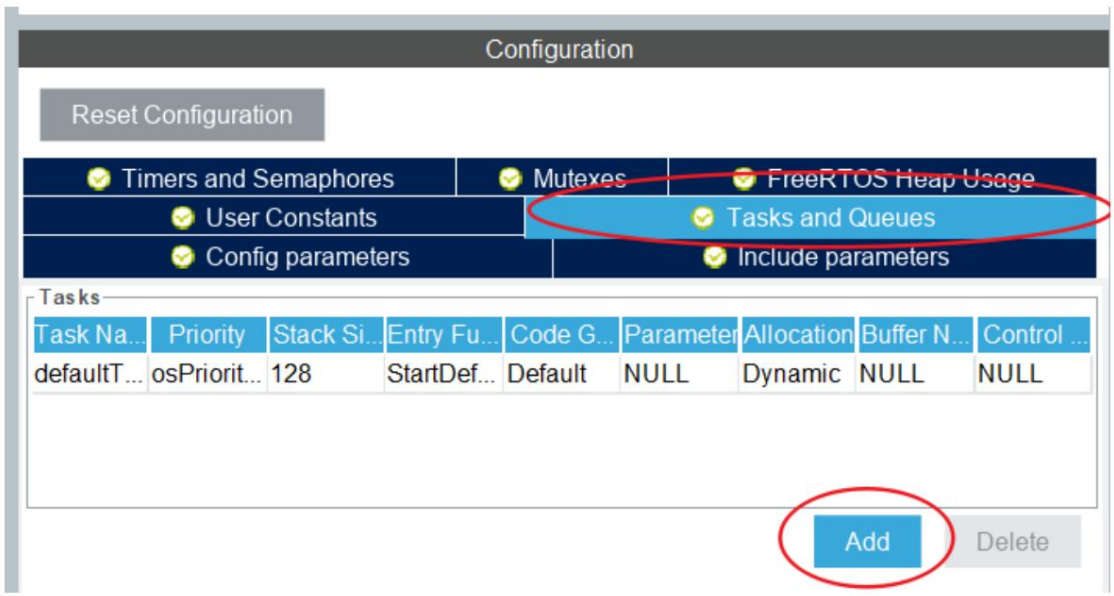
This section will learn how to create a task in cubeMX, first in the configuration page of freeRTOS

Under Configuration, select the Tasks and Queues tab, there is a default task that has been created as

"defaultTask", click to enter the configuration option and modify it as shown below.



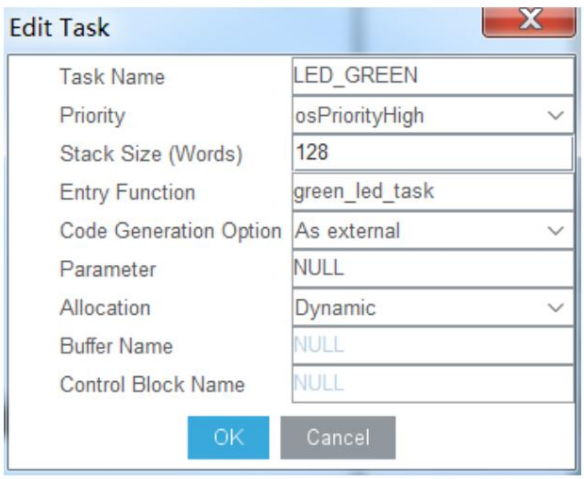
If you need to add a new task, click Add to create a new task.



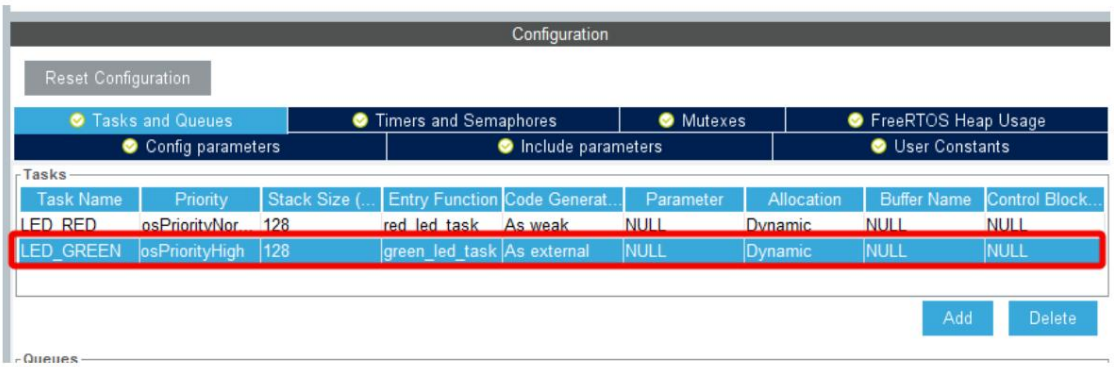
In the pop-up page, you can configure the task name, priority, stack size, entry function, etc. The specific function of each parameter

See the table below:

name	Features
Task Name	mission name
Priority	Priority of task creation
Stack Size (Words) The size of the task stack, the default unit is words	
Entry Function	Entry of the task function
Code Generation Option	The task function code generation method, set to Default will generate a common task function, As weak: Generate a task function modified with the __weak modifier; As external: a task function that generates an external reference, the user needs to has implemented the function; Default: Generates a task function in a default format, the user needs to number of functions.



After setting, click OK, you can see that there are more tasks created by yourself in the list.



The modified default task functions can also be found in freertos.c.


```
__weak void red_led_task(void const * argument)
{

    /* USER CODE BEGIN red_led_task */
    /* Infinite loop */
    for(;;)
    {
        osDelay (1);
    }
    /* USER CODE END red_led_task */
}
```

Because of the choice to create a weak function with the __weak modifier, the task function can be implemented elsewhere. program execution

This additionally implemented task function will be automatically found.

```
void red_led_task(void const * argument)
{

        while(1)
        {
                HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_SET);
                osDelay (500);
                HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
                osDelay (500);
                HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
                osDelay (500);

        }
}
```

Since a new task "LED_GREEN" is created and set to As external, the task function needs to be implemented elsewhere.

When the program is executed, it will automatically find the implemented task function.

```
void green_led_task(void const * argument)
```

```
{
    while(1)
    {
        HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
        osDelay (500);
        HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
        osDelay (500);
        HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
        osDelay (500);
    }
}
```

15.4.3 Creating tasks in the program

This section will learn how to create tasks programmatically.

Open the freertos.c file, in MX_FREERTOS_Init, find the code to create two processes, which are LED_RED and LED_GREEN. You can see that when you want to create a task, you only need to call osThreadDef and osThreadCreate can be used to create the "LED_BLUE" task.

```
/* Create the thread(s) */
/* definition and creation of LED_RED */
osThreadDef(LED_RED, red_led_task, osPriorityNormal, 0, 128);
LED_REDHandle = osThreadCreate(osThread(LED_RED), NULL);

/* definition and creation of LED_GREEN */
osThreadDef(LED_GREEN, green_led_task, osPriorityHigh, 0, 128);
LED_GREENHandle = osThreadCreate(osThread(LED_GREEN), NULL);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
osThreadDef(LED_BLUE, blue_led_task, osPriorityHigh, 0, 128);
led_blue_handle = osThreadCreate(osThread(LED_BLUE), NULL);
```



```
/* USER CODE END RTOS_THREADS */
```

First introduce osThreadDef, which is actually not a function, but a macro definition provided by CMSIS for

Set the task to be created

```
#define osThreadDef(name, thread, priority, instances, stacksz) \
extern const osThreadDef_t os_thread_def_##name
```

name	osThreadDef
Features	Set the task to be created
parameter 1	name, the name of the task to create
parameter 2	thread, the entry name of the task code to create
parameter 3	priority, the priority of the task to be created
parameter 4	instances, the number of threads that can be created under the task
parameter 5	stacksz, the size of the task stack

Then create a task through the osThreadCreate function provided by CMSIS

```
osThreadId osThreadCreate (const osThreadDef_t *thread_def, void *argument);
```

function name osThreadCreate	
Function function to create a task	
return value	osThreadId, task ID, ID is an important identifier of a task. When the task is created, it needs to be executed. When you modify the priority of this task, or destroy the task, you need to call the task ID, you need to mention Declare a variable of type osThreadId earlier to store the return value here.
parameter 1	const osThreadDef_t *thread_def, we pass the task parameter set by osThreadDef number, input by means of forced conversion + task name, such as setting the task in osThreadDef If the task name is LED_RED, enter osThread(LED_RED) here
parameter 2	void *argument, the initialization parameters required by the task, generally fill in NULL

Through the above two steps, a task is successfully created, and a name and thread parameter in osThreadDef are created.

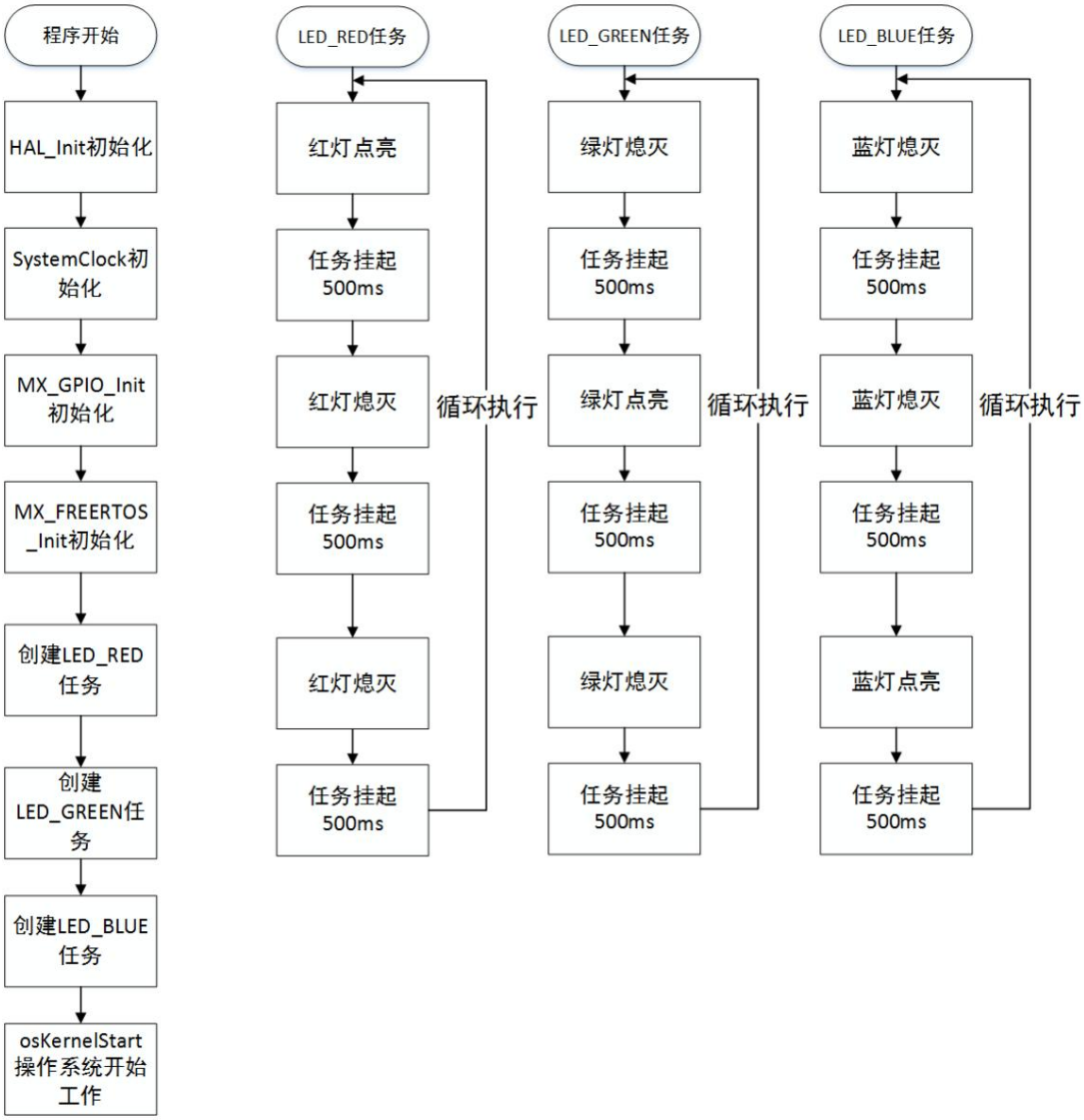
The corresponding function, the operating system will automatically find the function and execute it as a process. For example, declaring thread as

blue_led_task, you also need to execute the function void blue_led_task(void const * argument), while(1) loops

The content in the ring is the user's own code, here is to control the flashing of the blue led light.

```
void blue_led_task(void const * argument)
{
        while(1)
        {
                HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_RESET);
                osDelay (500);
                HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_RESET);
                osDelay (500);
                HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_SET);
                osDelay (500);
        }
}
```


15.4.4 Program Flow



15.4.5 Effect demonstration

Three tasks are created, and the three LEDs are flashed in sequence, the effect is as shown in the figure:

red light on

green light on

blue light on

15.5 Advanced Learning

The advanced learning section will introduce the task switching mechanism in the operating system.

In the operating system, each task to be performed, that is, the running process of a program is called a process. Process contains

With the concept of dynamic, it is the running process of a program, not a static program. Process is embodied in the program form

In fact, it is a piece of code that is executed in a loop. Take the following figure as an example. green_led_task is a function that makes the green led light flash.

After the process is created using the task creation function of the operating system, the operating system automatically finds this code and

implement.

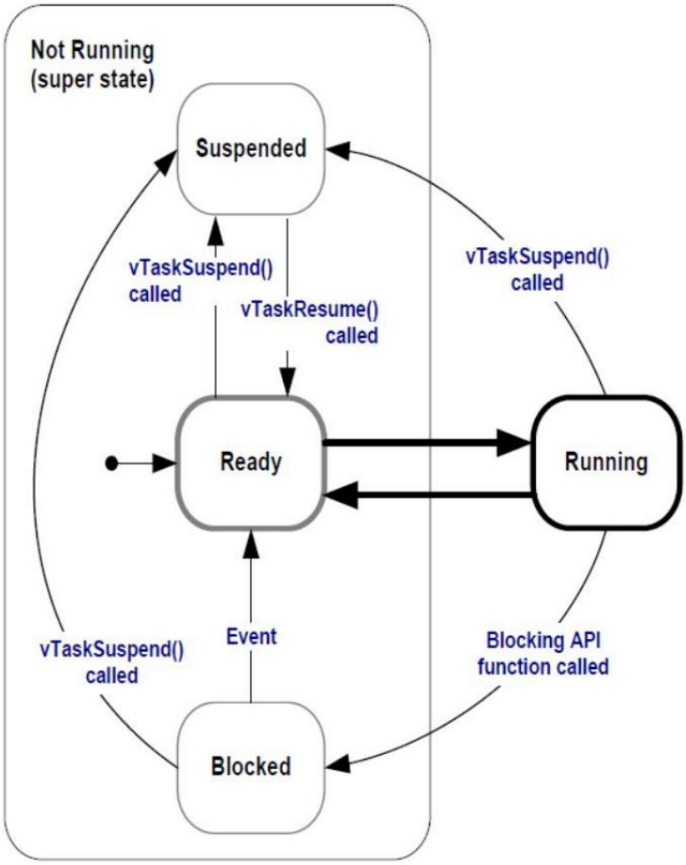
```
void green_led_task(void const * argument)
{
        while(1)
        {
                HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
                osDelay (500);
                HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
                osDelay (500);
                HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
                osDelay (500);
        }
}
```

When a program is executed, it is generally divided into three stages, starting execution--->executing--->execution complete. This also corresponds to

The working state of the process: ready state (Ready) ---> running state (Running) ---> terminated state (Blocked). if in

During the execution of a process, the function function that suspends the process is called, or the process has a higher priority during execution.

When the task is ready, the process will enter the suspended state (Suspended).



An important job of the operating system is to perform state switching of each process, because in fact the microcontroller can only run one at a time.

process, and the operating system, through proper management, allows each process to receive a timely response, allowing multiple processes to present

There is a sense of "concurrency" running at the same time.

When the operating system performs task switching, it must follow a certain scheduling algorithm.

Process interruption is divided into preemptive operating system and cooperative operating system.

Cooperative operating systems cannot interrupt running processes, and when multiple processes are ready, they must wait for the currently executing process.

The process ends, which is simpler to implement, but reduces the real-time performance of the process execution. A preemptive operating system can

The running process is interrupted, so it has a more complex scheduling mechanism, but also has better real-time performance.

In this section, the three important scheduling algorithms of the operating system are:

ÿ First come first serve (FCFS) scheduling algorithm

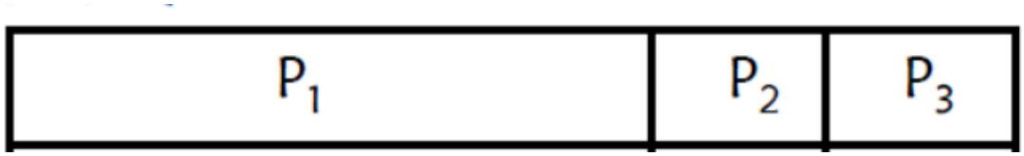
ÿ Priority scheduling algorithm

ÿ Time slice round-robin scheduling algorithm.

First, the FCFS scheduling algorithm is introduced. The principle of the algorithm is very simple. When there are multiple tasks, the task that is ready first occupies the

The CPU, until the task execution ends, enters the blocking state and releases the CPU, at which time the next task can occupy the CPU. Compare

As shown in the figure below, the ready sequence of the three tasks is P1, P2, P3:



The task ready sequence in the following figure is P2, P3, P1:

Then introduce the priority scheduling algorithm. We can give different priorities to each task in advance. When a process occupies

When the CPU is running, if a higher priority task is ready, the running task will be suspended and will be executed with high priority.

For the task of the first level, after knowing that the task of high priority is completed, it will resume the original task and continue to execute.

In the freeRTOS generated by cubeMX, each task has 7 optional priorities by default, ranging from -3 to 3.

The higher the priority, the higher the priority.

```
typedef enum {
    osPriorityIdle                = -3,          ///< priority: idle (lowest)
    osPriorityLow                 = -2,          ///< priority: low
    osPriorityBelowNormal = -1,                  ///< priority: below normal
    osPriorityNormal              = 0,           ///< priority: normal (default)
    osPriorityAboveNormal = +1,                  ///< priority: above normal
    osPriorityHigh                = +2,          ///< priority: high
    osPriorityRealtime            = +3,          ///< priority: realtime (highest)
    osPriorityError               = 0x84         ///< system cannot determine priority or thread has
illegal priority
} osPriority;
```

Finally, the time slice rotation algorithm is introduced, in order to avoid the situation where the execution time of a certain task is too long, causing other tasks to wait all the time

Appears, giving each task a specific length of time to run, called a time slice. If the task execution time exceeds

interval, the task will be suspended and other tasks will be executed instead.

Take the following figure as an example, the time required to execute the four tasks is P1 (53), P2 (17), P3 (68), P4 (24)

The time slice is 20.

Task P1		P2	P3	P4	P1	P3	P4	P1	P3	P4

Grand total time piece	20 unfinished into P1	37 Finish P2	57 unfinished into P3	77 unfinished into P4	97 unfinished into P1	117 unfinished into P3	121 Finish P4	134 Finish P1	154 unfinished into P3	174 Finish P3

When freeRTOS starts to run, each task will be scheduled according to the principles of the above three algorithms at the same time. for different

The priority task adopts the priority scheduling algorithm, and the task of the same priority adopts the FCFS algorithm and the time slice rotation calculation.

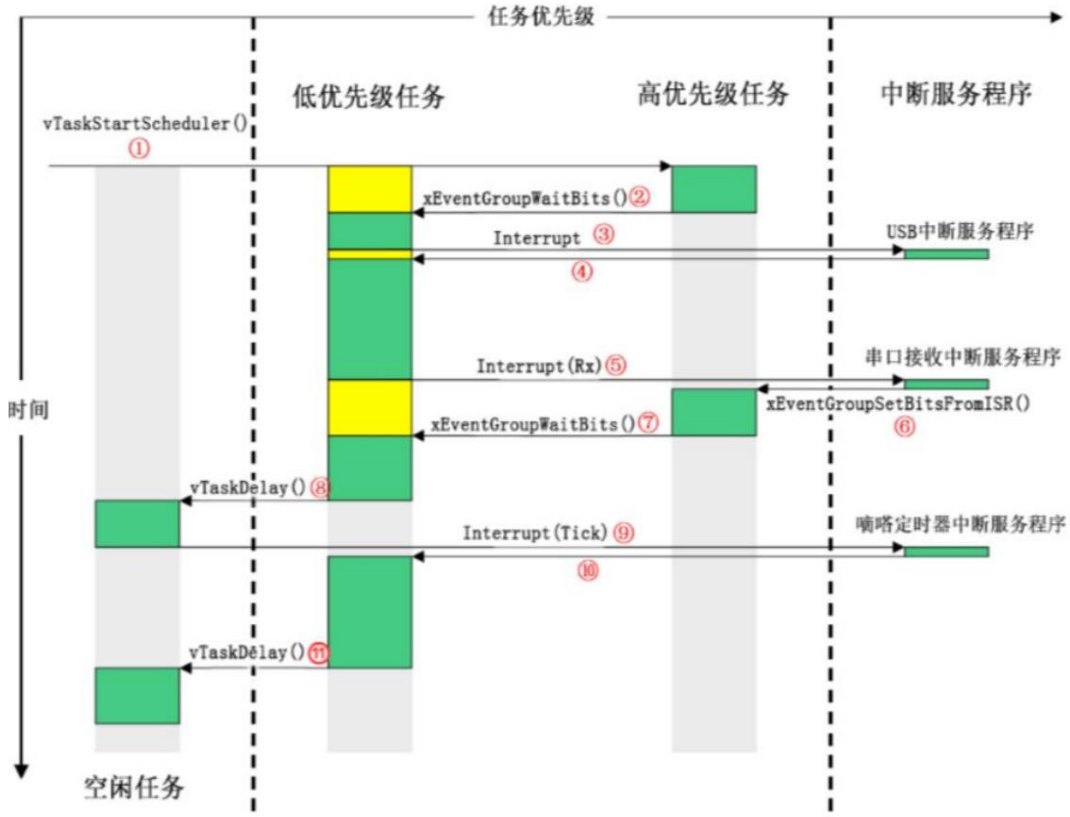
Law.

It should be noted that tasks in freeRTOS, regardless of priority, can be interrupted by interrupts. In addition, when it is necessary to

When the delay operation is performed in freeRTOS, it can be realized by suspending the task for a period of time, and the task is suspended when the task is suspended.

At the same time, other tasks can occupy the CPU, and after the delay expires, the previously suspended task resumes execution. Demonstration below

The switching process between suspended, low-priority tasks, high-priority tasks and interrupts in freeRTOS is implemented.



15.6 Course Summary

freeRTOS is an open source embedded operating system, which can be used to manage tasks and perform multitasking easily. Book

This lesson learned the basic principles of embedded operating systems and task management mechanisms, and learned and how to create

Build a project containing freeRTOS, and create tasks through cubeMX and programming.

16. IMU temperature control

16.1 Key points of knowledge

ÿ The significance of IMU controlling temperature

ÿ Introduction to PID Control

ÿ Special task switching - wake-up task

16.2 Course Content

This lesson will learn to use the PID control algorithm to control the temperature of the IMU, which can effectively reduce the temperature of the IMU.

Temperature zero drift. Among them, the PID control algorithm is used in the document control, and the PID control algorithm is a common control algorithm.

The method will also be briefly introduced in this chapter.

16.3 Basic Learning

16.3.1 Significance of IMU Controlling Temperature

In Chapter 13 BMI088, it was introduced that BMI088 is composed of two sensors, gyroscope and accelerometer. sensing

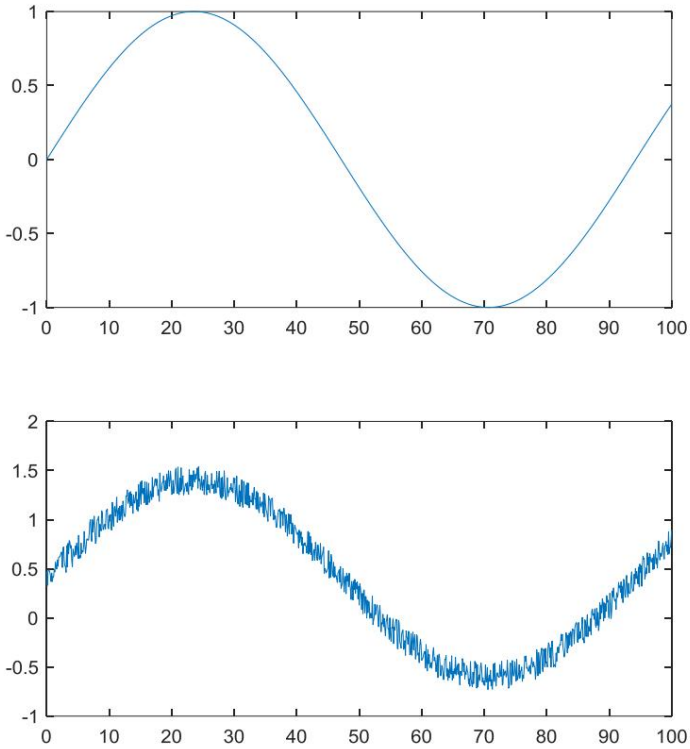
The basic function of the device is to convert different physical quantities, chemical quantities and biomass in the outside world into electrical signals or digital signals that are easy to process.

number and output. However, due to the fact that the amplification, filtering, sampling and other processes in the sensor inevitably introduce noise, one of them

An important noise is thermal noise. Thermal noise refers to random electrical signals due to the thermal motion of electrons, as shown in the figure above

It is an ideal sinusoidal signal. In the actual propagation process, the sinusoidal signal will become a clutter with clutter in the following figure after superimposing thermal noise.

A signal with an offset in the y-axis direction.



IMU data offset caused by thermal noise is one of the important reasons for the zero-drift phenomenon of the IMU.

Zero drift phenomenon refers to the phenomenon that when the physical quantity input is zero, the output measured by the sensor is not zero. That is, the IMU does not have any

If you move, the gyroscope and accelerometer will also read a certain size of data and regard it as generated by the IMU movement. because

This requires first measuring the size of the zero drift when the IMU is powered on, and subtracting the value measured by the IMU from the zero drift value, thereby reducing the

Zero drift effect. The premise of this method is that the zero-drift value of the IMU remains fixed, otherwise errors will still be introduced.

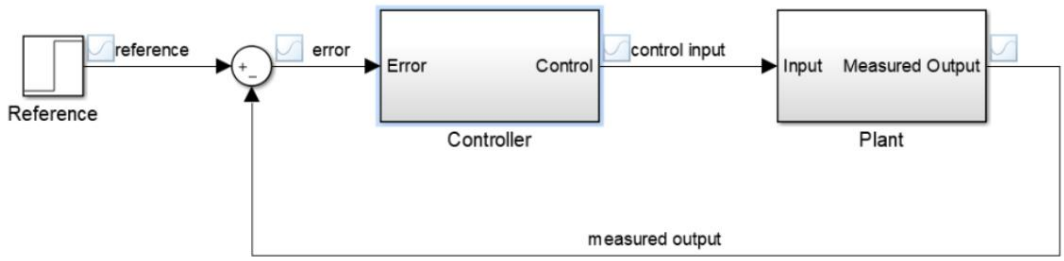
The magnitude of thermal noise is positively correlated with temperature, so the zero drift value is affected by temperature, so it is necessary to control the temperature protection of the IMU.

within a constant range to reduce the effect of zero drift.

16.3.2 Introduction to PID Control

PID is a commonly used control algorithm, and its basic idea is to use the error between the expected value and the actual value as the control variable to determine the final

Output.



As shown in the figure above, a control process is:

ÿ Control target input value (reference)

ÿ Make the difference to get the error value (error)

ÿ Controller

ÿ System object (plant)

ÿ Feedback value (measured output)

PID control belongs to a kind of controller, which consists of P (proportional), I (integral), and D (differential).

The expression of PID output value U(t) is as follows, where err(t) is the error value, Kp, Ki, Kd are proportional, integral and differential respectively

The coefficients of the three terms.

$$( ) = \ddot{y} \qquad ( ) + \ddot{y}\,\ddot{y}\,( ) + \ddot{y} \qquad \frac{( )}{}$$

Proportional item Kp: The output value of the proportional item of the controller maintains a linear relationship with the error value. If the error value is doubled, the output value will be the same.

Double the magnification and double the error value, and the output value will also double as well. A method that relies only on proportional terms for control is called proportional

For example control, proportional control can easily realize the basic functions of the controller, but there are often static errors and excessive

system oscillation problem.

Integral term Ki: The output value of the integral term of the controller has a linear relationship with the integral value of the error value, that is, the accumulated value of the error value is multiplied by one.

a constant. The research of the integral term accelerates the process of the system approaching the set value, but the integral gain is too large to easily cause the phenomenon of integral overshoot.

The differential term Kd: the magnitude of the differential term is positively correlated with the change in the output value, the differential term calculates the first derivative of the error, and sums

Multiply by a constant to get the output value of the differential term. The differential term can react to changes in the system, short-term changes to the system.

Change is helpful.

The following table summarizes the characteristics of the items in the PID controller.

	effect	shortcoming
Proportional term P	Amplifies or attenuates the error signal, proportional to The size of the number determines the strength of the control effect.	Excessive gain may cause the system to oscillate Qualitative deterioration, reducing the relative stability of the system, If the gain is too small, the control effect is not obvious and the response is slow. The effect of disturbance cannot be corrected; and the proportional term does not The steady-state error of the system can be completely eliminated.
Integral Item I	Influence the controller through the effect on error accumulation output, and is reduced by the negative feedback of the system Small deviation, as long as there is enough time, the integral control The control will be able to eliminate the steady-state error.	If the gain is too large, the system integral overshoot may occur, leading to If the adjustment time is too long, the gain may be too small. Convergence is too long to overcome the influence of interference in time ring.
Differential term D	It can reflect the speed of reaction error signal change degree, resulting in a large control when the error first appeared. production, has the effect of advanced control, helps In order to reduce the adjustment time and improve the dynamic performance of the system quality;	If the gain is too large, it is easy to cause the system to oscillate, resulting in stable Qualitative decrease; if the gain is too small, it may not improve the effect Obviously; at the same time, differentiation is easy to introduce high-frequency noise, leading to cause system instability.

Since the digital system is discrete, when the PID control algorithm is implemented in the microcontroller, the output of the PID controller needs to be expressed as The formula is rewritten into discrete form. The specific method is to change the output u(t) and error e(t) from functions to arrays u(k) and e(k), Integral is replaced by summation, and differentiation is replaced by difference. The discretized PID expression is as follows:

$$( ) = ÿ ( ) + ÿ ÿ ( ) + ÿ [( ) ÿ (ÿ 1)] =0$$

PID controllers can be divided into incremental PID controllers and positional PID controllers, all of which are described above are positional PIDs controller, that is, the error value directly determines the final output, while the incremental PID controller uses the error value to control each output The amount of change ÿu

$$( ) = ( ) ÿ (ÿ 1)$$

Its expression is:

$$( ) = ÿ [( ) ÿ (ÿ 1)] + ÿ ( ) + ÿ [( ) ÿ 2 ÿ (ÿ 1) + (ÿ 2)]$$

Compared with positional PID, incremental PID control has the following advantages:

ÿ There is no need to accumulate calculation and accumulation, the output increment is only related to the first three error sampling values, and the parameters are easier to adjust

ÿ Only the control increment is output each time, so the impact of failure is small

When using incremental PID, it is necessary to memorize the last output value, and add the last output value and the increment to get the current output value.

output value.

16.4 Program Learning

16.4.1 Task wake-up function

In some cases, certain tasks need to pass certain conditions (such as interruption or completion of another task execution)

trigger execution. It is usually done by using the task notification function. The main task will remain dormant when it does not receive a task notification.

When the trigger conditions are met, a task notification is issued, and the main task waiting for the trigger will enter the ready state and wait once after receiving the task notification.

Execute, and enter the sleep state after the execution is completed, until the next task notification arrives.

Use the external interrupt as the trigger condition in the example program, issue the task notification in the external interrupt handler, and the temperature of the IMU

The degree control task imu_temp_control_task waits for the task notification to wake up.

freeRTOS provides the function vTaskNotifyGiveFromISR to issue task notifications from interrupts

```
void vTaskNotifyGiveFromISR ( TaskHandle_t xTaskToNotify, BaseType_t * pxHigherPriorityTaskWoken)
```

Function name vTaskNotifyGiveFromISR	
The function sends a task notification from the interrupt handler to wake up the waiting task	
return value void	
parameter 1	TaskHandle_t xTaskToNotify, the task handle of the notified task
parameter 2	BaseType_t *pxHigherPriorityTaskWoken, used to save whether there are high-priority tasks ready. If the value of this parameter is pdTRUE after the function is executed, it means high priority task to be executed, otherwise no

You can obtain the task handle for the task waiting to wake up through the xTaskGetHandle function provided by freeRTOS. This

The task handle needs to be consistent with the parameters in the send task notification.

```
TaskHandle_t xTaskGetHandle( const char *pcNameToQuery )
```

Function name xTaskGetHandle	
Function to get the task handle	

Return value TaskHandle_t, the task handle	
parameter 1	const char *pcNameToQuery, the name of the task that needs to get the handle

The function of getting the task name here is implemented through the function pcTaskGetName provided by freeRTOS

```
char *pcTaskGetName( TaskHandle_t xTaskToQuery )
```

function name pcTaskGetName	
Function to get the task name	
Return value char *, task name	
parameter 1	TaskHandle_t xTaskToQuery, the handle of the task whose name needs to be obtained, if NULL then get the name of the currently running task

The ulTaskNotifyTake function provided by freeRTOS implements the mechanism of receiving task notifications and waking up dormant tasks.

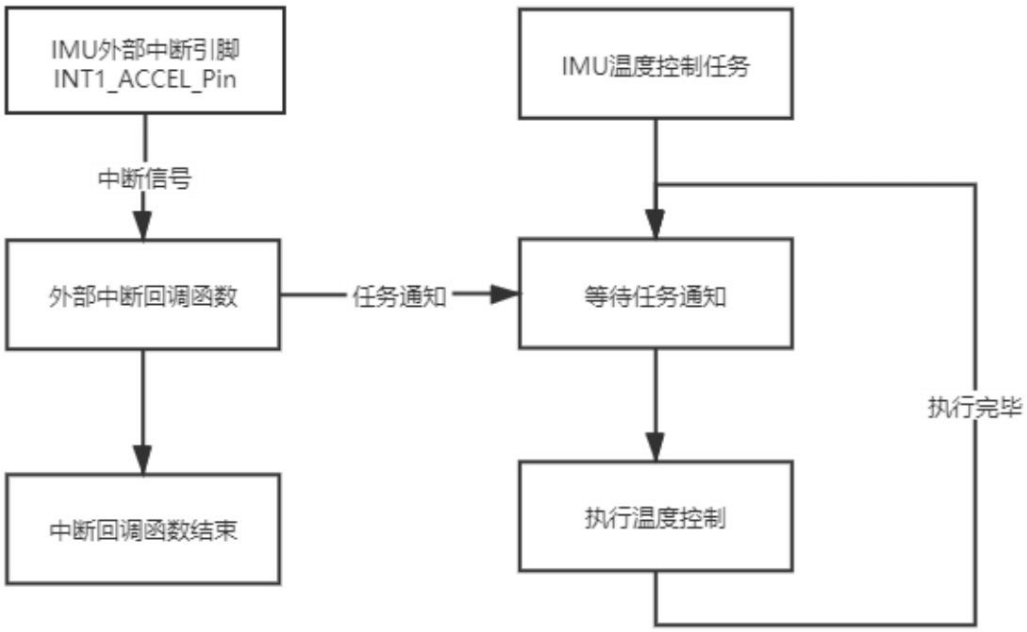
```
uint32_t ulTaskNotifyTake ( BaseType_t xClearCountOnExit, TickType_t xTicksToWait)
```

Function name ulTaskNotifyTake	
Function to receive task notifications and wake up dormant tasks	
return value task notification value	
parameter 1	BaseType_t xClearCountOnExit , choose whether to clear the ulNotifiedValue, configured as pdFALSE indicates the internal value used for the task semaphore before the function returns The value of the variable ulNotifiedValue is decremented by one, which is used for task counting semaphores. parameter configuration pdTRUE means the internal variable used for the task semaphore before the function returns
parameter 2	TickType_t xTicksToWait, the maximum wait time to wait for a semaphore to become available

16.4.2 Control Link

The control link of the entire running task is: the accelerometer external interrupt pin INT1_ACCEL_Pin of the IMU triggers the external interrupt, the vTaskNotifyGiveFromISR function is called in the external interrupt callback function HAL_GPIO_EXTI_Callback Send a task notification, the IMU temperature control task imu_temp_control_task starts to execute after receiving the task notification,

After the execution is completed, go to sleep and wait for the next task notification. The flow chart is as follows:



16.4.3 PID initialization and calculation function

This section will introduce PID initialization and its calculation function. In the sample program, the PID initialization is completed through the PID_Init function.

The function of the PID_Init function is to set the mode of the PID controller, various coefficients, the maximum output value and the upper limit of the integral, and finally set the

Both the input and output of the PID controller are initialized to 0.

```
void PID_init(pid_type_def *pid, uint8_t mode, const fp32 PID[3], fp32 max_out, fp32 max_iout)
{
    if (pid == NULL || PID == NULL)
    {
        return;
    }
    pid->mode = mode;
    pid->Kp = PID[0];
    pid->Ki = PID[1];
    pid->Kd = PID[2];
    pid->max_out = max_out;
    pid->max_iout = max_iout;
```



```
        pid->Dbuf[0] = pid->Dbuf[1] = pid->Dbuf[2] = 0.0f;
        pid->error[0] = pid->error[1] = pid->error[2] = pid->Pout = pid->Iout = pid->Dout =
pid->out = 0.0f;
}
```

The PID_calc function used to calculate the PID first distinguishes whether the PID mode is positional or incremental.

with formula

$$( ) = ÿ ( ) + ÿ ÿ ( ) \quad + ÿ [( ) ÿ (ÿ 1)] \quad =0$$

Incremental uses the formula:

$$( ) = ÿ [( ) ÿ (ÿ 1)] + ÿ ( ) + ÿ [( ) ÿ 2 ÿ (ÿ 1) + (ÿ 2)]$$

The current error value e(k) is obtained by subtracting the expected value set from the feedback value ref read through the temperature register inside the IMU

pid->error[0] to indicate. In the program, the output pid->output of the PID controller is used as the heating resistance applied to the IMU

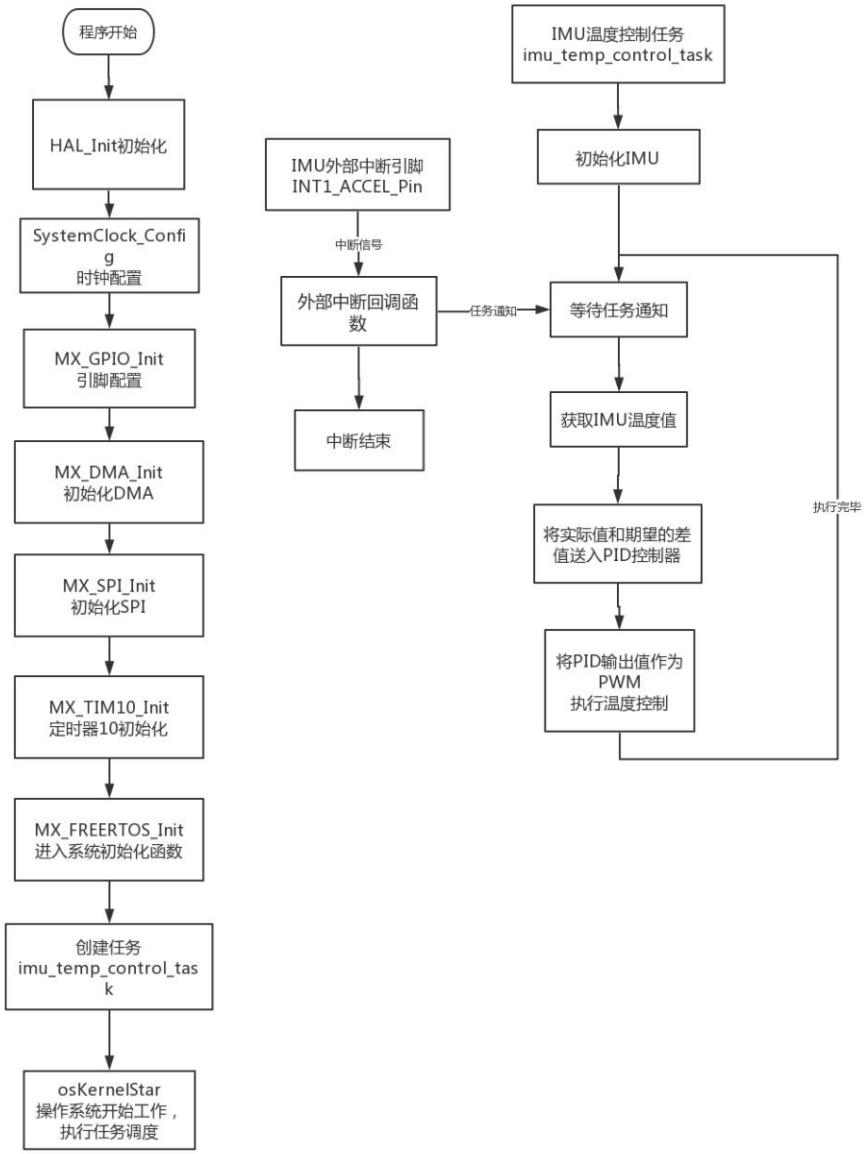
PWM value, the larger the output value, the higher the temperature of the heating resistor.

```
fp32 PID_calc(pid_type_def *pid, fp32 ref, fp32 set)
{
        if (pid == NULL)
        {
                return 0.0f;
        }

        pid->error[2] = pid->error[1];
        pid->error[1] = pid->error[0];
        pid->set = set;
        pid->fdb = ref;
        pid->error[0] = set - ref;
        if (pid->mode == PID_POSITION)
        {
                pid->Pout = pid->Kp * pid->error[0];
                pid->Iout += pid->Ki * pid->error[0];
                pid->Dbuf[2] = pid->Dbuf[1];
```



```
            pid->Dbuf[1] = pid->Dbuf[0];
            pid->Dbuf[0] = (pid->error[0] - pid->error[1]);
            pid->Dout = pid->Kd * pid->Dbuf[0];
            LimitMax(pid->Iout, pid->max_iout);
            pid->out = pid->Pout + pid->Iout + pid->Dout;
            LimitMax(pid->out, pid->max_out);
        }
        else if (pid->mode == PID_DELTA)
        {
            pid->Pout = pid->Kp * (pid->error[0] - pid->error[1]);
            pid->Iout = pid->Ki * pid->error[0];
            pid->Dbuf[2] = pid->Dbuf[1];
            pid->Dbuf[1] = pid->Dbuf[0];
            pid->Dbuf[0] = (pid->error[0] - 2.0f * pid->error[1] + pid->error[2]);
            pid->Dout = pid->Kd * pid->Dbuf[0];
            pid->out += pid->Pout + pid->Iout + pid->Dout;
            LimitMax(pid->out, pid->max_out);
        }
        return pid->out;
}
```


16.4.4 Program Flow



16.4.5 Effect Demonstration

Using a thermal imager, you can observe the temperature control effect of the IMU, as shown in the figure:

16.5 Advanced Learning

In the actual use of various electronic devices, it is often encountered that the actual output of the device is inconsistent with the expected value of the input, or even different.

far away. Taking the motor as an example, suppose we input the expected speed of the motor to be 100rpm. When the motor is no-load, its

The output speed is also exactly 100rpm, but when we add wheels or other loads to the motor, its speed has no

Instead of reaching the desired value of 100rpm, it was reduced to 80rpm. If the battery voltage starts after the motor has been running for a while

If it drops, the speed of the motor will be further reduced.

The motor in the example is a typical controlless system. In a controlless system, there is no difference between the input expected and the actual output

There may be huge errors between the two, resulting in poor system reliability. The control system will

The actual output adjusts the input to minimize the error.

A common way to implement a control system is to add a PID controller to a controlless system, which can

The error between the actual output value and the expected value adjusts the input value so that the final output value is close to the expected value.

Still taking the motor as an example, when a motor is hung with a load of 10 catties, if the input voltage to the motor is exactly 24V,

Then the actual output of the motor is only 80rpm, the error between the input and the actual output is 20rpm, and the difference 20rpm is input

PID controller, the PID controller will increase the input to 26V, then the actual output of the motor can reach 100rpm

. If the difference becomes larger to 40rpm, the PID controller will continue to increase the voltage input to the motor. In short, PID control

The controller will automatically adjust the input value of the motor voltage according to the size of the error, so as to keep the output value consistent with the expected value.

The following will present the effect diagram of the PID control shortcomings in basic learning.

If the proportional term Kp is too small, the response speed of the PID controller is slow and there is a static error. Static error refers to the final output guarantee of the controller

It is held as a value with a certain error from the expected value, and the cause of the static error is that the output of the proportional control and the error are linear

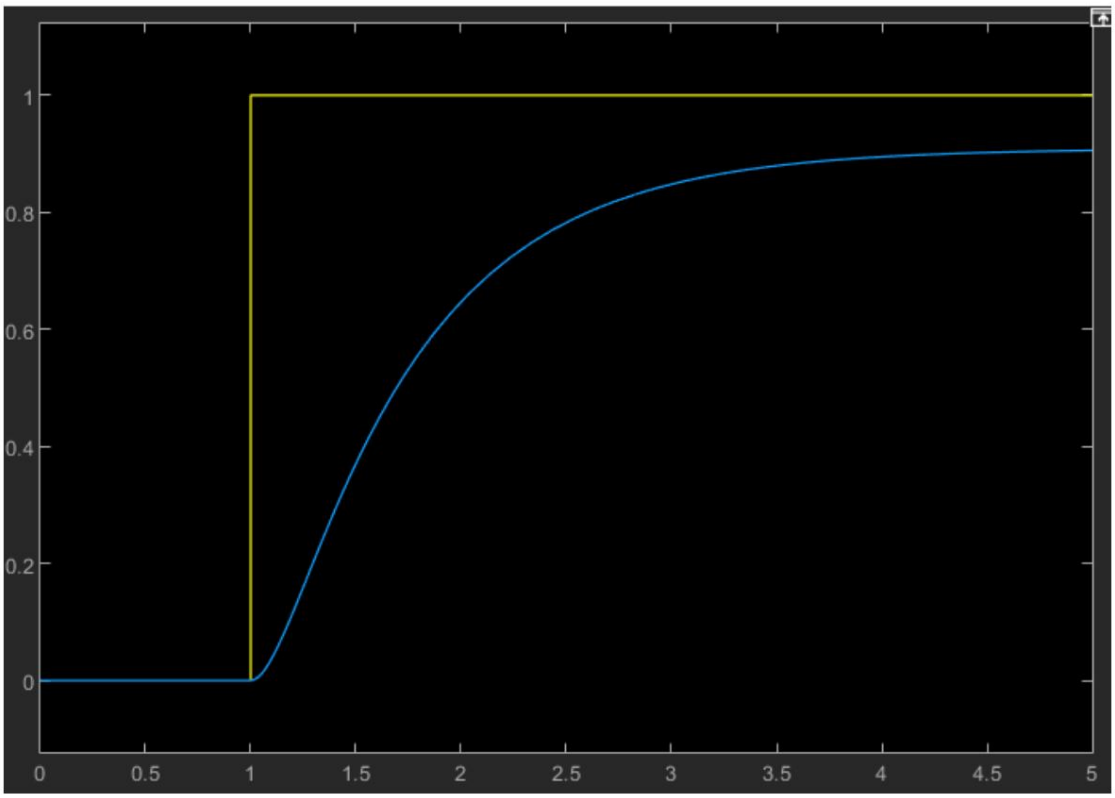
If the error value decreases, the output value of the proportional controller will also decrease, making it impossible for the proportional controller to achieve

is exactly the same as the expected value, i.e. the case where the error value is zero, because if the error value is zero then the output of the proportional controller will also be

zero.

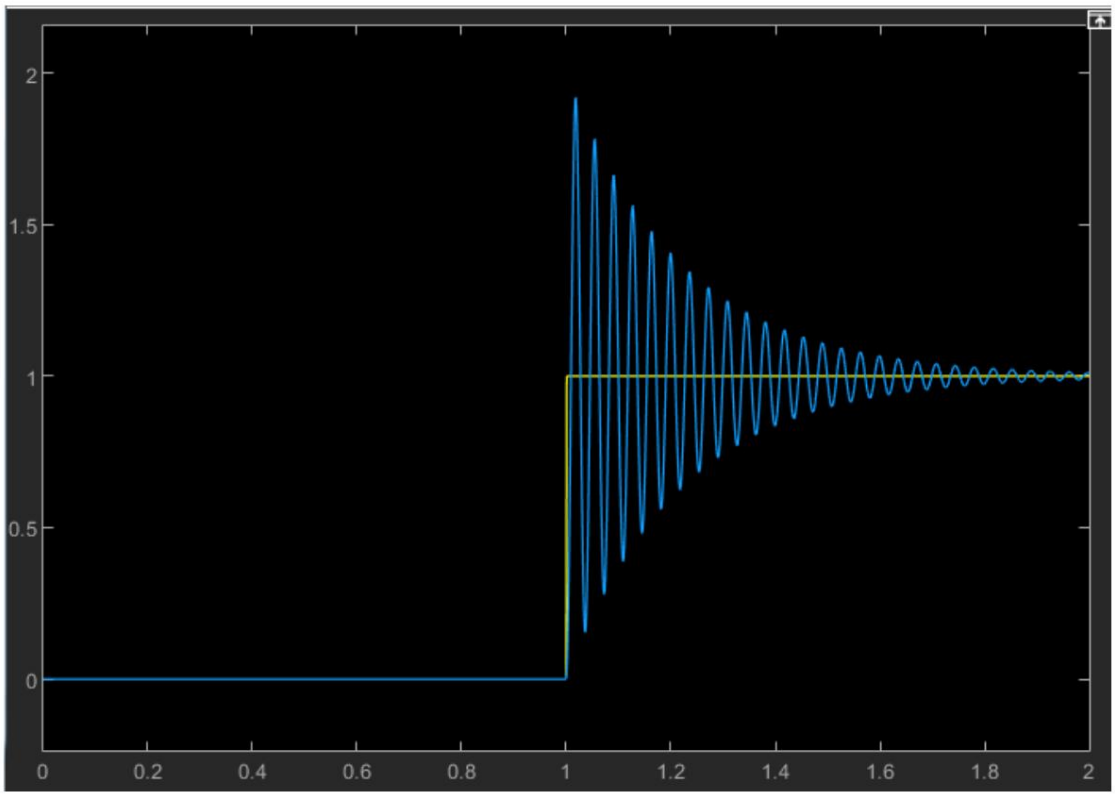
The following figure shows the simulation of the PID controller proportional term is too small, the blue curve is the output of the system, and the yellow curve is the set input.

To see a constant error value between the final output value and the expected value, this error value is the static difference.



By increasing the proportional coefficient Kp, the static difference can be reduced and the reaction speed can be improved. But if Kp is too large, the output will be severely shaken

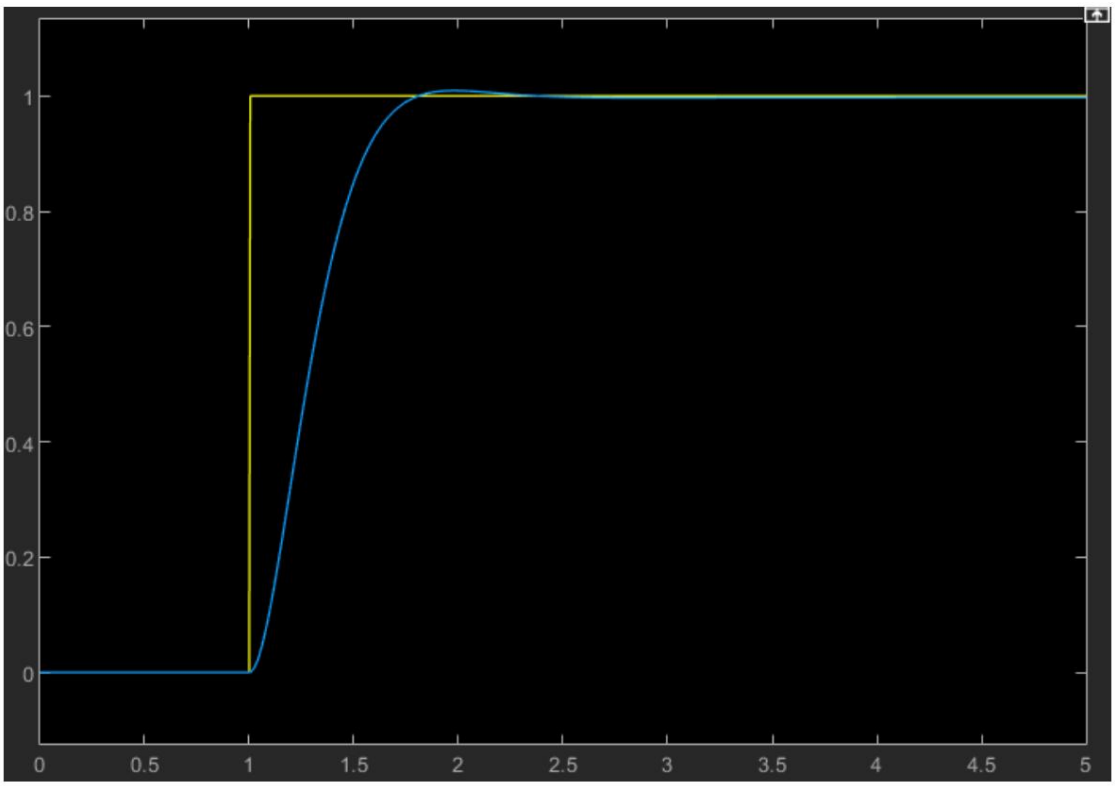
swing, as shown in the figure below. Oscillation will greatly reduce the stability of the system.



In order to overcome the static error, an integral term can be introduced. Due to the cumulative nature of the integral, the output value of the integral term is determined by the error of all previous moments.

Difference decision. Therefore, even when the error value is zero, the integral term will remain active, so that the output of the controller is not

Zero, the output value of the controller can be truly equal to the expected value only after the integral term is introduced.

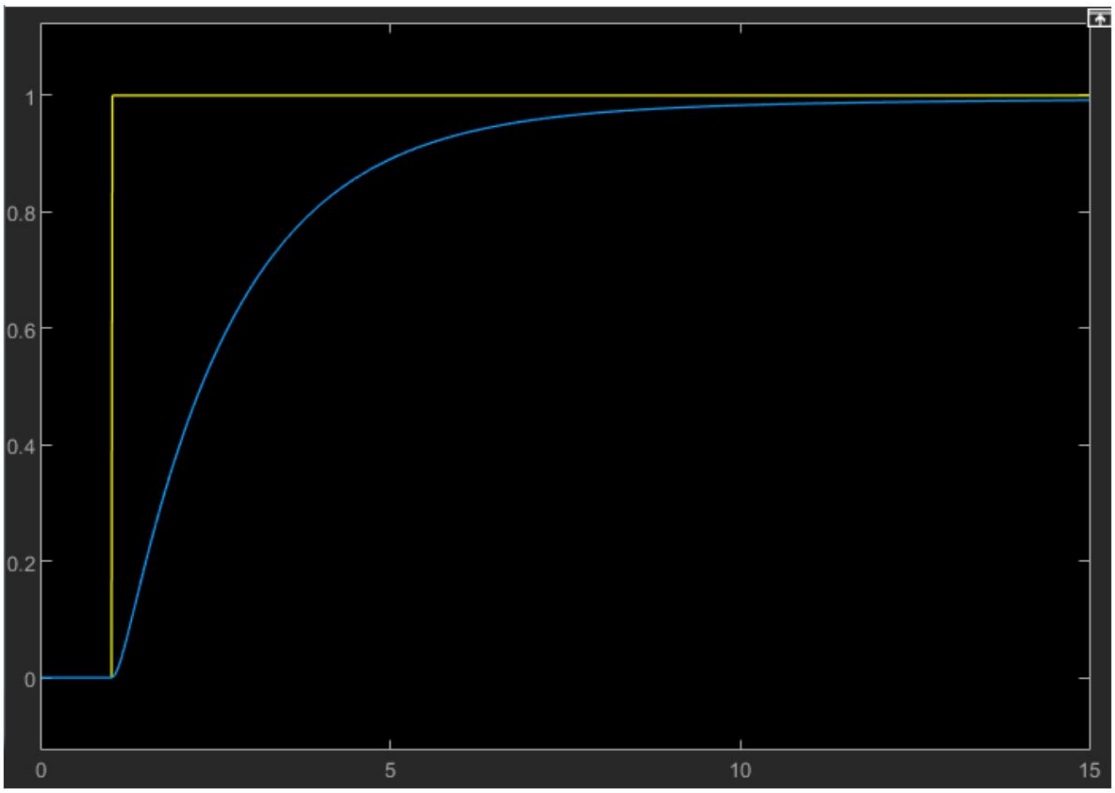


However, the introduction of the integral controller will prolong the adjustment time of the system, and the output value of the proportional-integral controller needs to be compared with that of the proportional controller.

The controller takes longer to stabilize, resulting in a slower system response.

If the integral term is too small, or the upper limit of the integral limit is too small, the output convergence time will be very long, which will lead to the system's noise.

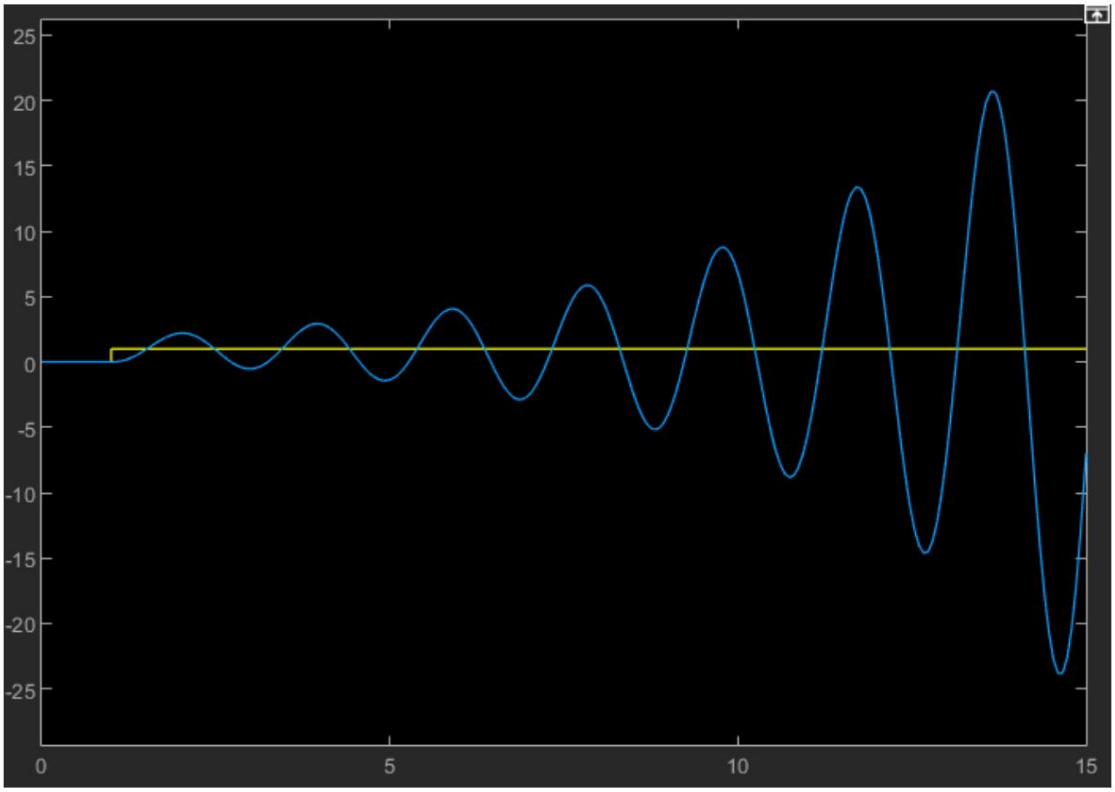
should be very slow, as shown in the image below.



If the integral term is too large, or the upper limit of the integral is too large, it will bring serious overshoot. Overshoot means that the peak value of the system output is far

Higher than the expected value, overshoot will cause the system to take a long time to stabilize, and may even fail to stabilize.

It will seriously affect the stability of the system and reduce the response speed of the system, as shown in the following figure.

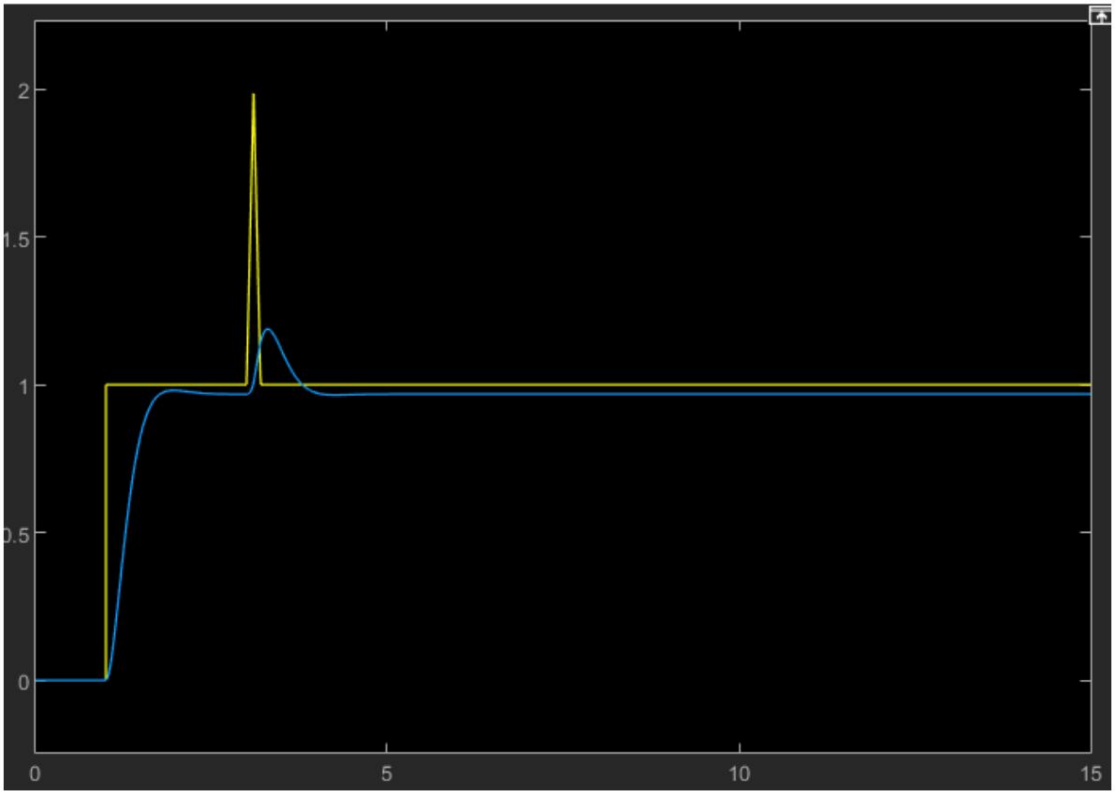


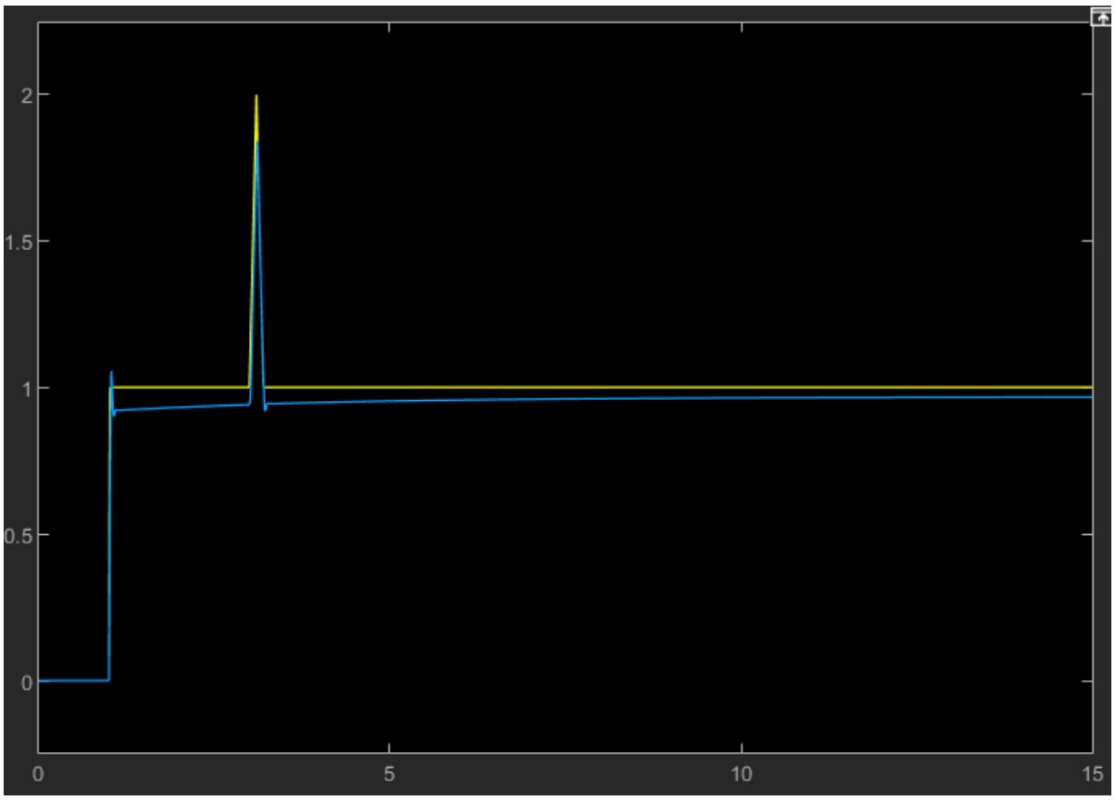
In order to improve the response speed of the system and reduce the amount of overshoot, we can introduce a differential term, the size of the differential term and the variation of the output value

is positively correlated, so whenever there is a sharp change in the input, a large differential term is generated to quickly keep up with this

changes, which can reduce the overshoot. In addition, due to the nature of the differential term to cancel out the change, the output fluctuation can be more

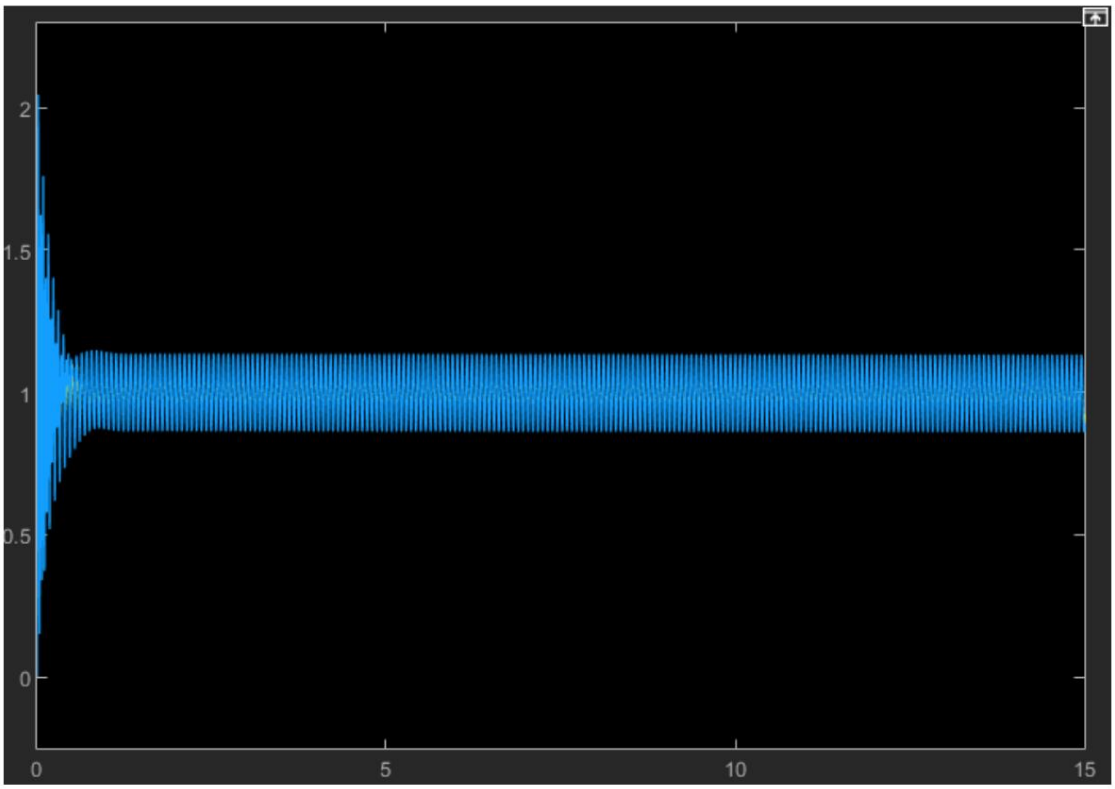
Fast decay reduces the time required for the system output to stabilize, thereby speeding up the response.





However, if the D term is too large, a small change in the output will produce a large output change, causing the system to oscillate, such as

As shown below:



Generally, when we design a PID controller, we need to adjust the coefficients of the three terms of the PID according to the actual output of the PID.

So that the output of the system is fast and stable.

The following table summarizes the functions of the items in the PID controller:

	Features
Proportional term P	The output is linearly regulated according to the error value, and there is static error and overshoot
Integral Item I	Can eliminate static difference, but will cause the system to respond slower
Differential term D	Reduce overshoot and speed up system response

16.6 Course Summary

PID control is a common control method, and temperature control is beneficial to suppress the effect of zero drift of IMU. In this lesson we have learned

Basic principles of PID control, and write a program to keep the working temperature of IMU constant through PID control, from

And reduce the effect of zero drift of the IMU.

17. Chassis control tasks

17.1 Knowledge points

ÿ The structure of the Mecanum wheel

ÿ Motor speed loop control

ÿ Positive motion process of chassis kinematics

17.2 Course Content

In this course, you will learn the chassis control of the RoboMaster robot. The chassis is to carry the RoboMaster robot movement

The mechanical structure, the chassis is generally composed of wheels, suspension, frame and so on. Among them, the RoboMaster competition chassis wheel set is used.

It is the Mecanum wheel, each Mecanum wheel is driven by a separate motor to complete the function of omnidirectional movement. RoboMaster

The robot chassis often needs to be controlled with the gimbal. For example, the angle control of the RoboMaster robot chassis requires

Follow the gimbal angle for control, and the chassis also needs the gimbal angle to rotate during the movement process to achieve smooth movement

Effect.

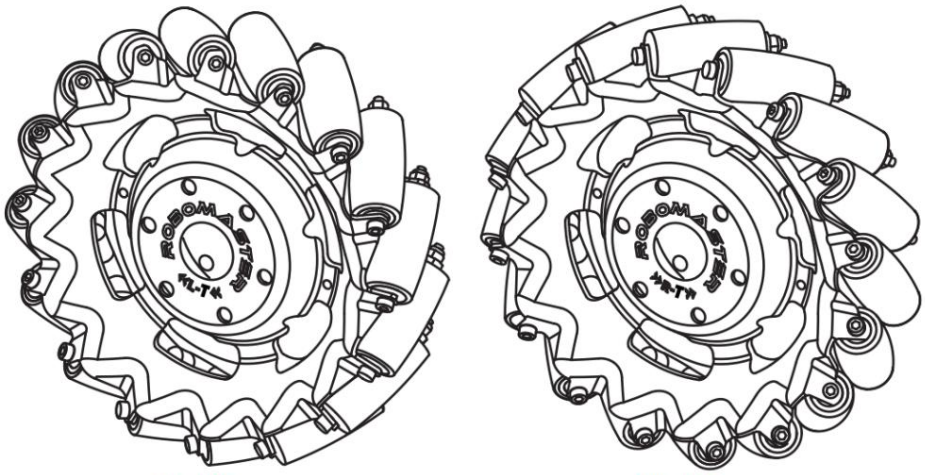
17.3 Basic Learning

17.3.1 Structure of Mecanum Wheel

The RoboMaster Mecanum wheel is a 45° omnidirectional wheel with a diameter of 152.5mm, with 16 wheels distributed around the hub

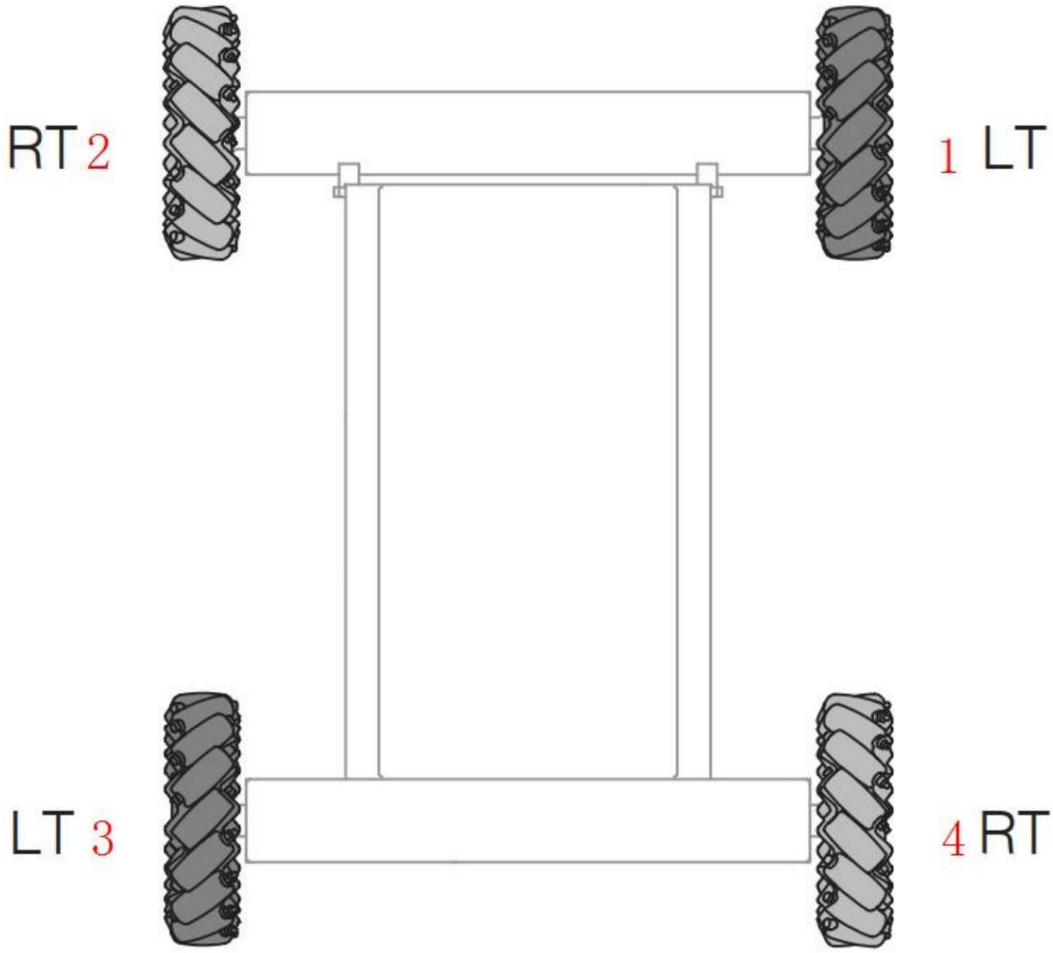
The small rubber roller is at an angle of 45° to the axis of the wheel, and the mechanical structure is shown in the figure. Mecanum wheel according to the angle of the small rubber wheel

It is divided into left-handed Mecanum wheel and right-handed Mecanum wheel, which are chiral symmetry and must be used in sets.



17.3.2 Installation of Mecanum Wheels

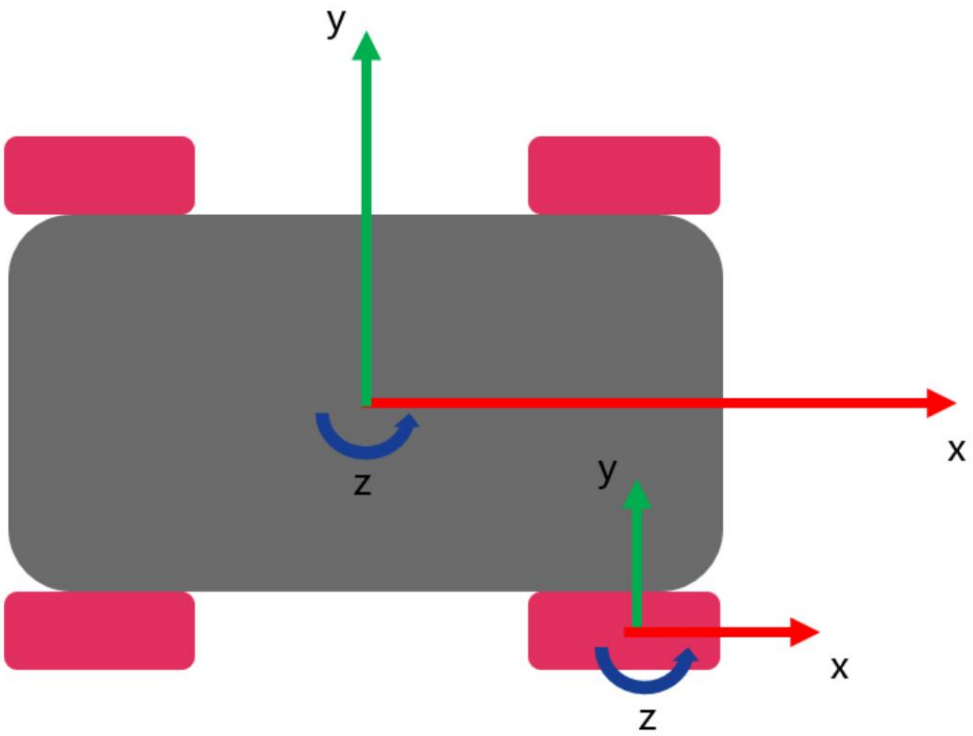
The top view of the chassis Mecanum wheel installation is shown in the figure. It can be seen that the 1st, 2nd, 3rd and 4th wheels are in different directions. 1 and 4 3 is a left-handed Mecanum wheel, and 2 and 4 are a right-handed Mecanum wheel.



17.3.3 Chassis Positive Kinematics

The movement direction of the chassis needs to be explained by establishing a coordinate system, as shown in the figure. The forward direction of the chassis is the x-axis, and the left direction

It is the y-axis, the z-axis is vertically upward, and the chassis rotates counterclockwise as the positive direction of rotation.



A rigid body has three degrees of freedom motion in the plane, the forward and backward motion Vx along the x-axis, and the left-right motion along the y-axis.

Motion Vy, rotational motion Wz around the z-axis. Calculate the motion of the four Mecanum wheels according to the three degrees of freedom motion of the chassis

The process is called the positive movement process of the chassis. Calculate the speed of the Mecanum wheel based on the chassis speed as shown in the table below.

Chassis speed	Mecanum wheel speed
Forward speed Vx	V1 = Vx V2 = Vx V3=Vx V4 = Vx
Left and right speed Vy	V1 = Vy V2=-Vy V3 = Vy V4 = -Vy
Rotation speed Wz	V1 = Wz V2 = - Wz V3 = - Wz V4 = Wz

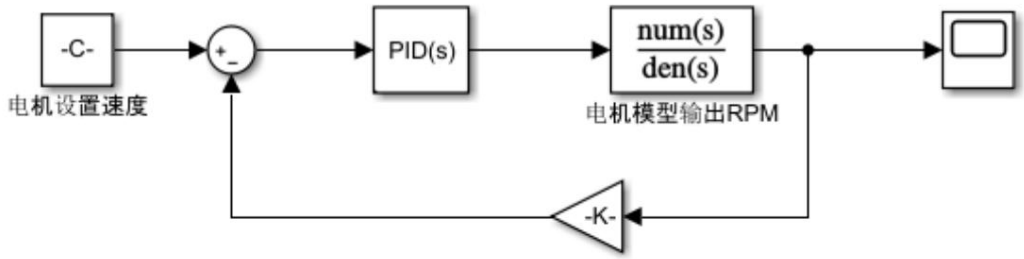
17.3.4 Speed Loop Control of Motors

Learn about the PID control method from what you learned in the previous lesson. In this course, the PID control method will be applied to the 3508 motor speed control.

The chassis is equipped with four motors, which control four Mecanum wheels respectively. The chassis completes the omnidirectional movement, and the 4 wheels need to be fine-tuned

Accurate speed control can ensure that the direction of the chassis movement will not be deviated due to different wheel speeds.

The control loop is shown in the figure:



17.4 Program Learning

17.4.1 Can Send Motor Control Function Review

In the CAN communication chapter, learn how to use CAN communication for motor control. The IDs assigned to the chassis motors are 0x201, 0x202, 0x203 and 0x204, they are all controlled by the CAN packet with ID 0x200, and in

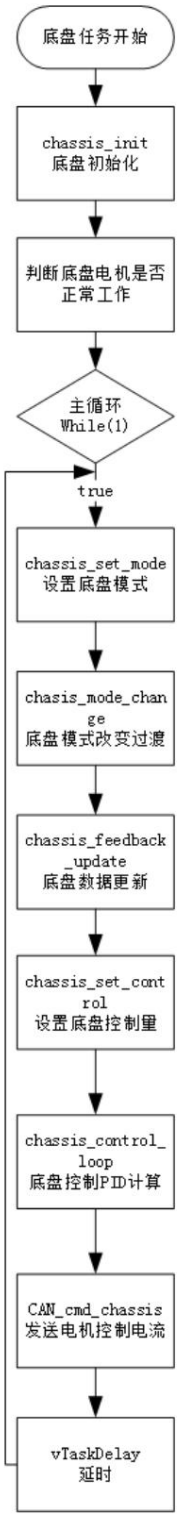
The chassis control function CAN_cmd_chassis is encapsulated in CAN_receive.c.

Function name	CAN_cmd_chassis
function	Motor current control via can bus
function return	None
Parameter 1: motor current control value with motor1 ID 0x201, range [-16384, 16384]	
Parameter 2: motor current control value with motor2 ID 0x202, range [-16384, 16384]	
Parameter 3: The motor current control value whose motor3 ID is 0x203, the range is [-16384,16384]	

Parameter 4: The motor current control value whose motor4 ID is 0x204, the range is [-16384,16384]	

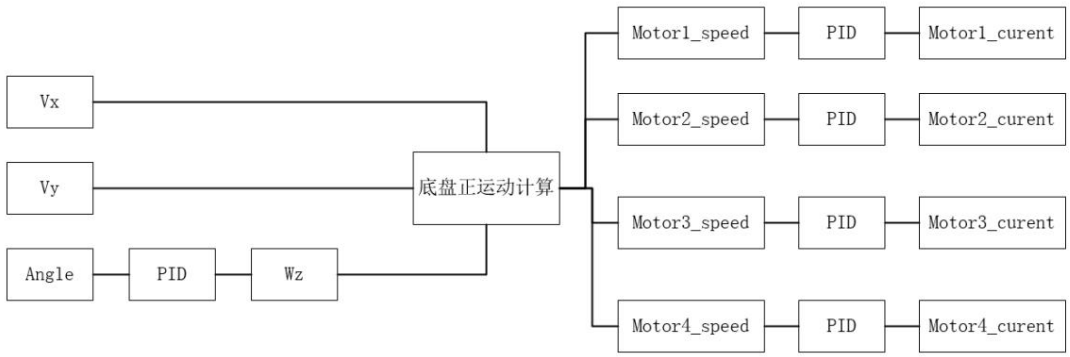
17.4.2 Explanation of program flow

Use freeRTOS to create a chassis task. The main loop process of the chassis task is as follows.



process	Features
chassis_init	Chassis initialization, mainly initialize motor speed PID, remote control Pointer and Attitude Angle Pointer.
chassis_set_mode	The chassis control is determined according to the switch value of the remote control and the state of the gimbal. The chassis control mode is determined according to the chassis control behavior.
chassis_mode_change_control_transit	Save data during chassis control mode switching to ensure control Switched control targets transition smoothly.
chassis_feedback_update	Chassis motors feedback speed, attitude angle, and update chassis motion speed.
chassis_set_control	According to the input of remote control and keyboard keys, calculate different motor control control speed
chassis_control_loop	The chassis calculates different PIDs according to different control modes.
CAN_cmd_chassis	Send motor current control value

The common control route of RoboMaster robot is shown in the figure, the movement speed is calculated by the channel value of the remote control and keyboard Vx, Vy, the rotation speed Wz is calculated by the gimbal angle PID, and the four motor control speeds are calculated through the forward kinematics of the chassis, The control current value is calculated by the respective speed PID.



17.4.3 Explanation of key functions

Here are a few main functions:

1. chassis_set_mode function and chassis_behaviour_mode_set function

The chassis_set_mode function calls the chassis_behaviour_mode_set function to set the control mode. and

In the chassis_behaviour_mode_set function, the behavior of the chassis is determined by the switch of the remote control, and then through the line

To determine the chassis control method, the behavior mode and control mode can be referred to the table below.

```
static void chassis_set_mode(chassis_move_t *chassis_move_mode)
{
    //in file "chassis_behaviour.c"
    chassis_behaviour_mode_set(chassis_move_mode);
}
void chassis_behaviour_mode_set(chassis_move_t *chassis_move_mode)
{
    //remote control set chassis behaviour mode
    // remote control setting mode
    if (switch_is_mid(chassis_move_mode->chassis_RC->rc.s[CHASSIS_MODE_CHANNEL]))
    {
        chassis_behaviour_mode = CHASSIS_NO_FOLLOW_YAW;
    }
    else if (switch_is_down(chassis_move_mode->chassis_RC->rc.s[CHASSIS_MODE_CHANNEL]))
    {
        chassis_behaviour_mode = CHASSIS_NO_MOVE;
    }
    else if (switch_is_up(chassis_move_mode->chassis_RC->rc.s[CHASSIS_MODE_CHANNEL]))
    {
        chassis_behaviour_mode = CHASSIS_INFANTRY_FOLLOW_GIMBAL_YAW;
    }

    //accord to beheviour mode, choose chassis control mode
    //Select a chassis control mode based on the behavior mode
```



```
    if (chassis_behaviour_mode == CHASSIS_ZERO_FORCE)
    {
        chassis_move_mode->chassis_mode = CHASSIS_VECTOR_RAW;
    }
    ……
    else if (chassis_behaviour_mode == CHASSIS_OPEN)
    {
        chassis_move_mode->chassis_mode = CHASSIS_VECTOR_RAW;
    }
}
```

behavior pattern	Chassis performance and application
CHASSIS_ZERO_FORCE	The current control value of the chassis motor is 0, and the chassis is in a powerless state. When the controller is disconnected or needs to be easily pushed when the chassis is powered on
CHASSIS_NO_MOVE	The speed control value of the chassis motor is 0. Due to the speed loop control of the motor, the chassis The performance is not moving, but there is resistance to pushing the chassis, which is applied to the remote control switch In the lower position, it is necessary to stop the movement of the chassis.
CHASSIS_INFANTRY_F OLLOW_GIMBAL_YAW	The movement speed of the chassis is determined by the remote control and the keyboard keys, and will control the bottom The disk follows the gimbal to calculate the rotational speed, and the chassis has a motor speed loop and cloud The stage angle ring control, applied to the remote control switch in the upper position, normally follows the gimbal In the case of angle, it is the common control logic of RoboMaster robot.
CHASSIS_ENGINEER_F OLLOW_CHASSIS_YAW	The movement speed of the chassis is determined by the remote control and the keyboard keys, and will control the bottom The disc follows the angle of the chassis to calculate the rotational speed, and the chassis has a motor speed loop And chassis angle loop control, common in engineering robot control occasions.
CHASSIS_NO_FOLLOW_ YAW	The movement speed and rotation speed of the chassis are determined by the remote control, and there is no angle closed loop control, so the chassis only has speed loop control, and it is used in applications that only require chassis control. occasion
CHASSIS_OPEN	Chassis open-loop control, there is no closed-loop control of the program, the channel value of the remote control is directly It is then converted into the motor current value and sent to the CAN bus.

There are four chassis control modes, as shown in the table below.

control mode	Features and Applications
CHASSIS_VECTOR_FOL LOW_GIMBAL_YAW	The movement speed of the chassis is determined by the remote control and the keyboard, and the rotation speed is determined by the angle difference of the gimbal count Calculate out ÿ Yes CHASSIS_INFANTRY_FOLLOW_GIMBAL_YAW selected control mode
CHASSIS_VECTOR_FOL LOW_CHASSIS_YAW	The moving speed of the chassis is determined by the remote control and the keyboard, and the rotation speed is determined by the angle difference of the chassis count Calculate out ÿ Yes CHASSIS_ENGINEER_FOLLOW_CHASSIS_YAW selection control mode
CHASSIS_VECTOR_NO_ FOLLOW_YAW	The chassis movement speed and rotation speed are determined by the remote control, no angle ring control, yes CHASSIS_NO_FOLLOW_YAW ÿ CHASSIS_NO_MOVE Selected control mode
CHASSIS_VECTOR_RA IN	The current control value of the chassis motor is directly calculated from the channel value of the remote control. sent directly to the CAN bus, is selected by CHASSIS_OPEN and CHASSIS_ZERO_FORCE control mode.

2. chassis_set_control function and chassis_behaviour_conotrol_set function

The chassis_set_control function calls the chassis_behaviour_conotrol_set function to get the three freedoms of the chassis
Control value of degree motion, when the chassis control mode is CHASSIS_VECTOR_FOLLOW_GIMBAL_YAW,
After one rotation operation, the control speed of the chassis is rotated into the angle direction of the gimbal to ensure the smooth movement of the chassis.
This rotational control can also be used for chassis motion control in small gyro rotations.

```
static void chassis_set_contorl(chassis_move_t *chassis_move_control)
{
    fp32 vx_set = 0.0f, vy_set = 0.0f, angle_set = 0.0f;
    chassis_behaviour_control_set(&vx_set, &vy_set, &angle_set, chassis_move_control);
    if (chassis_move_control->chassis_mode == CHASSIS_VECTOR_FOLLOW_GIMBAL_YAW)
    {
        fp32 sin_yaw = 0.0f, cos_yaw = 0.0f;
```

```
        //rotate chassis direction, make sure vertial direction follow gimbal

        //Rotate to control the speed direction of the chassis to ensure that the forward direction is the direction of the gimbal, which is conducive to smooth movement

        sin_yaw = arm_sin_f32(-chassis_move_control->chassis_yaw_motor->relative_angle);

        cos_yaw = arm_cos_f32(-chassis_move_control->chassis_yaw_motor->relative_angle);

        chassis_move_control->vx_set = cos_yaw * vx_set + sin_yaw * vy_set;

        chassis_move_control->vy_set = -sin_yaw * vx_set + cos_yaw * Hang;

        //set control relative angle set-point

        //Set the control relative gimbal angle

        chassis_move_control->chassis_relative_angle_set = rad_format(angle_set);

        // calculate ratation speed

        //Calculate the rotation PID angular velocity

        chassis_move_control->wz_set = -PID_calc(&chassis_move_control->chassis_angle_pid, chassis_move_control->chassis_yaw_motor->relative_angle,
chassis_move_control->chassis_relative_angle_set);

        //speed limit

        //speed limit

        chassis_move_control->vx_set = fp32_constrain(chassis_move_control->vx_set, chassis_move_control->vx_min_speed, chassis_move_control->vx_max_speed);

        chassis_move_control->vy_set = fp32_constrain(chassis_move_control->vy_set, chassis_move_control->vy_min_speed, chassis_move_control->vy_max_speed);

    }

    ……

}
```

3. chassis_vector_to_mecanum_wheel_speed function

The function of chassis_vector_to_mecanum_wheel_speed is to convert the three-degree-of-freedom motion speed of the chassis

The speed of the wheat wheel, according to the basic learning, the following formula can be obtained

v1 = + +

v2 = ÿ ÿ

v3 = + ÿ

v4 = - +

And because when the motor 1 and the motor 4 are at the forward speed, the motor 1 and the motor 4 rotate clockwise to a negative value,

Motor 2 and Motor 3 rotate counterclockwise as positive values, so the revised formula is:

v1 = ÿ ÿ ÿ

v2 = ÿ ÿ

v3 = + ÿ

v4 = - + -

Since the mounting position of the gimbal is not in the center of the chassis, it is often in the front of the chassis, so when rotating, motor 1 and motor 2

A slower speed is required, Motor 3 and Motor 4 require a faster speed, a correction factor is required

CHASSIS_WZ_SET_SCALE (less than 1) is assumed to be a parameter, so the revised formula is:

v1 = ÿ ÿ + (ÿ 1) ÿ

v2 = ÿ + (ÿ 1) ÿ

v3 = + + (ÿ ÿ 1) ÿ

v4 = - + + (- - 1) (

So the function is implemented as follows:

```
static void chassis_vector_to_mecanum_wheel_speed(const fp32 vx_set, const fp32 vy_set, const fp32 wz_set, fp32 wheel_speed[4])
{
    //because the gimbal is in front of chassis, when chassis rotates, wheel 0 and wheel 1 should be slower and wheel 2 and wheel 3 should be faster
    //When rotating, because the gimbal moves forward, the speed of the first two rounds 0 and 1 is slowed down, and the speed of the second two rounds 2 and 3 is faster.
    wheel_speed[0] = -vx_set - vy_set + (CHASSIS_WZ_SET_SCALE - 1.0f) * MOTOR_DISTANCE_TO_CENTER * wz_set;
    wheel_speed[1] = vx_set - vy_set + (CHASSIS_WZ_SET_SCALE - 1.0f) * MOTOR_DISTANCE_TO_CENTER * wz_set;
    wheel_speed[2] = vx_set + vy_set + (-CHASSIS_WZ_SET_SCALE - 1.0f) * MOTOR_DISTANCE_TO_CENTER * wz_set;
    wheel_speed[3] = -vx_set + vy_set + (-CHASSIS_WZ_SET_SCALE - 1.0f) * MOTOR_DISTANCE_TO_CENTER * wz_set;
}
```

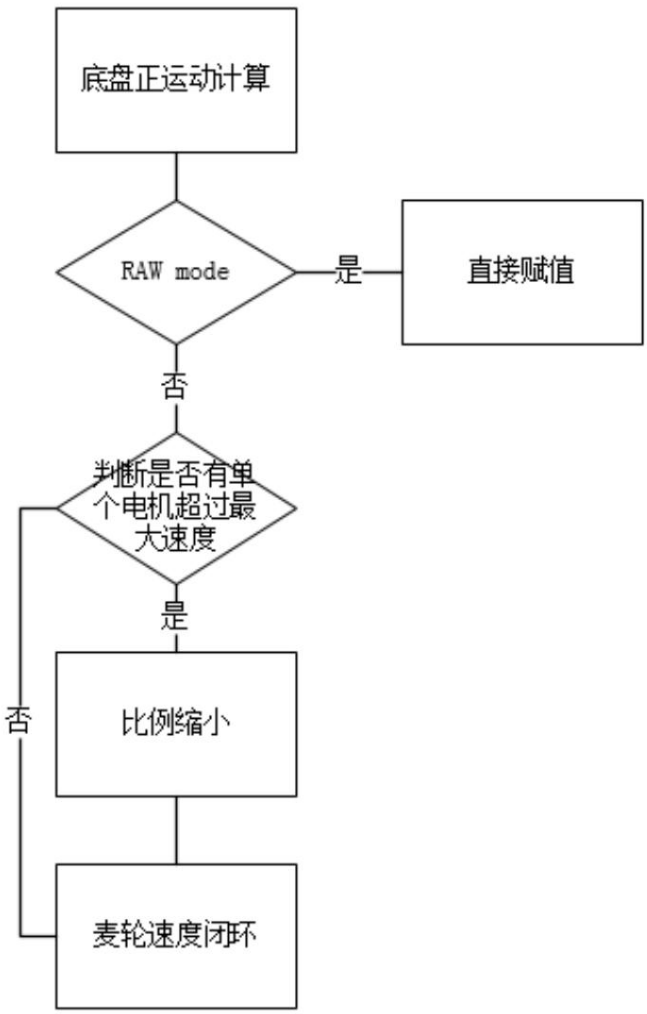
4. chassis_control_loop function

The chassis_control_loop function first enters the positive movement of the chassis to calculate the speed of the wheat wheel, and then judges if it is

CHASSIS_VECTOR_RAW, just assign the value directly to complete the calculation, otherwise judge the maximum speed of the wheel, enter

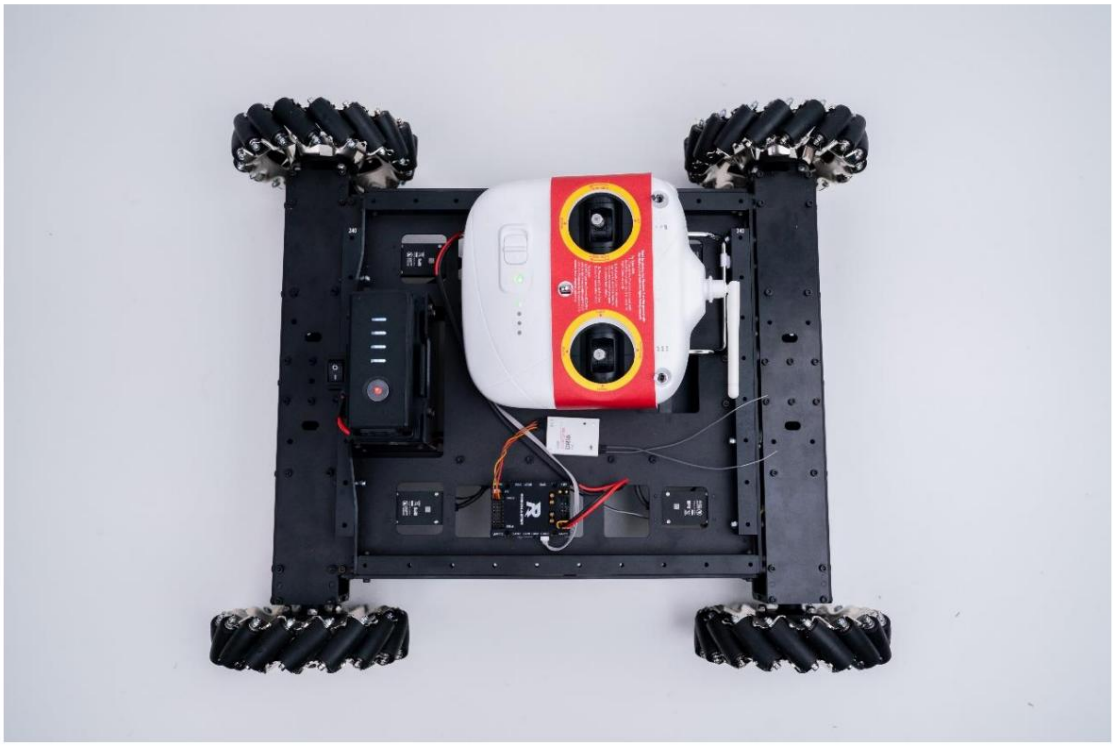
The line scale is reduced, and then the speed loop closed-loop control is performed. The code of the speed loop control is as follows:


```
        //calculate pid

        //calculate pid

        for (i = 0; i < 4; i++)

        {

                PID_calc(&chassis_move_control_loop->motor_speed_pid[i],

chassis_move_control_loop->motor_chassis[i].speed, chassis_move_control_loop->motor_chassis[i].speed_set);

        }
```



17.4.4 Effect display

Chassis motion control can be performed by using the remote control, the chassis is shown in the figure:

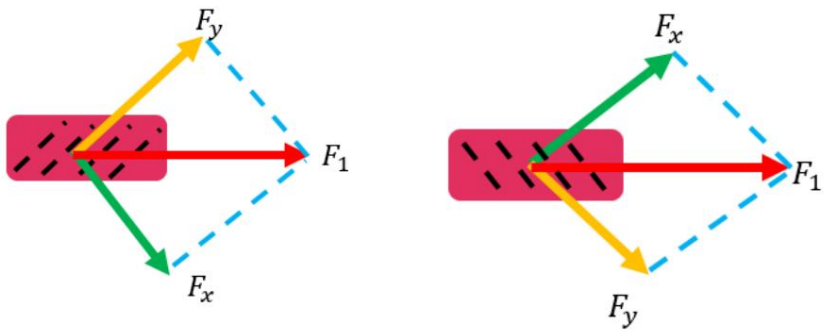


17.5 Advanced Learning

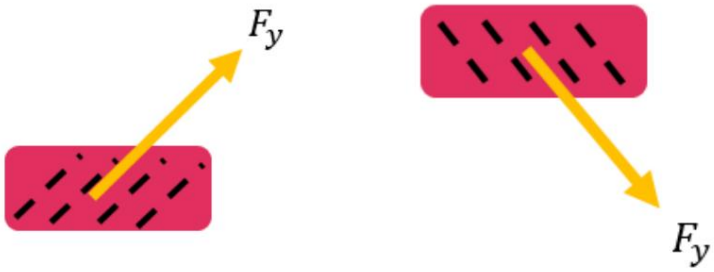
17.5.1 Positive Motion Process of Chassis Kinematics

The difference between the Mecanum wheel and the ordinary rubber wheel lies in the small rubber wheel on the outer ring. When the small rubber wheel touches the ground, it is 45° from the axis angle, the ground friction will be decomposed into Fy along the axis of the small rubber wheel and Fx perpendicular to the axis, where perpendicular to the

The component force Fx of the axis will make the small rubber wheel rotate, so that its component force will not affect the overall movement, and Fy will affect the overall movement of the chassis. Movement affects. The force analysis is shown in the figure:



Mecanum Wheel Force Analysis



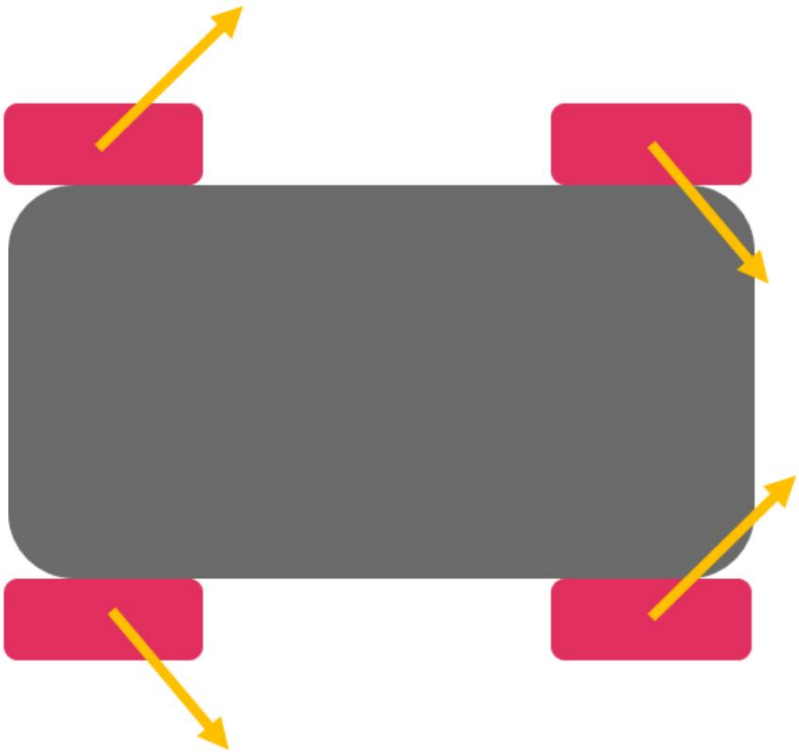
Mecanum Wheel Fy Component

Due to the different direction of the component force, Fy has different effects on the movement of the chassis. By controlling the rotation of the four motors, the bottom is controlled.

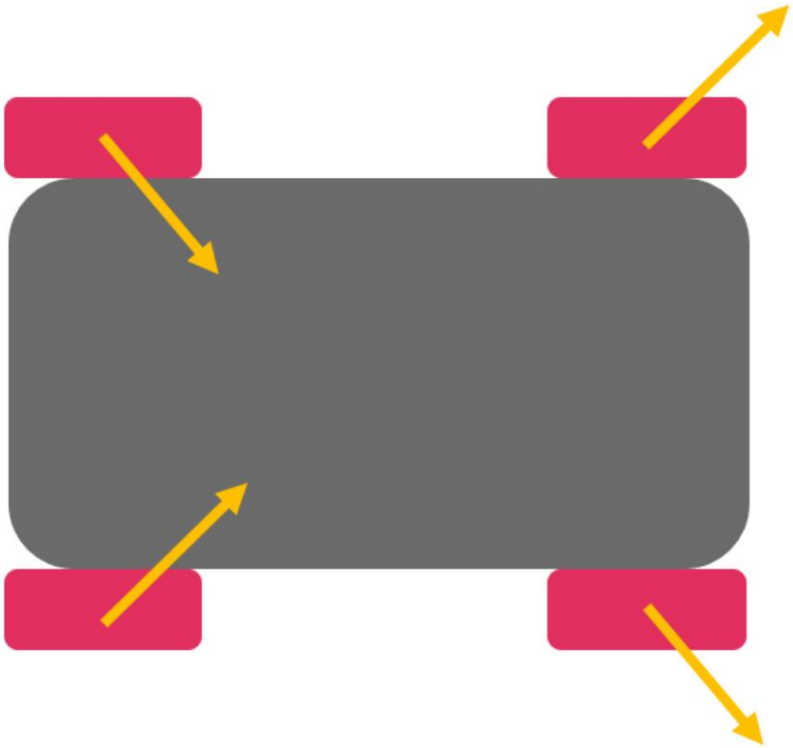
direction of disk movement.

Four Mecanum wheels are installed on the chassis, and two sets of left and right rotating Mecanum wheels are required to complete the forward and translation directions of the chassis.

There are two installation methods, namely O type and X type.



O type



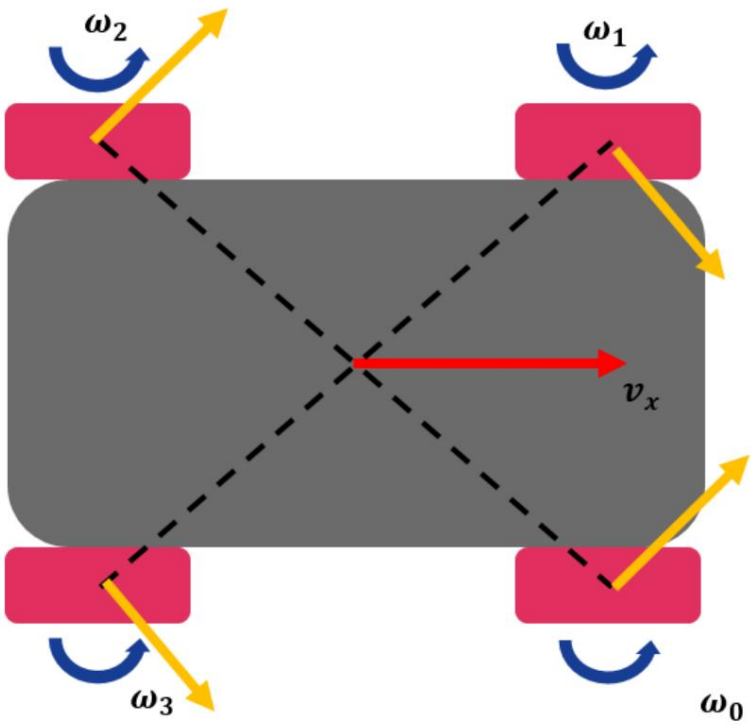
Type X

Among them, the X-type is difficult to rotate, so the Mecanum wheel installation form of the chassis is O-type.

When the chassis forward speed is xm/s, the speed of the four chassis motors is xm/s. As shown in the figure, the left and right direction of the first two wheels

The forces cancel each other, leaving only the force in the front and rear directions; the forces in the left and right directions of the rear two wheels cancel each other, leaving only the front and rear.

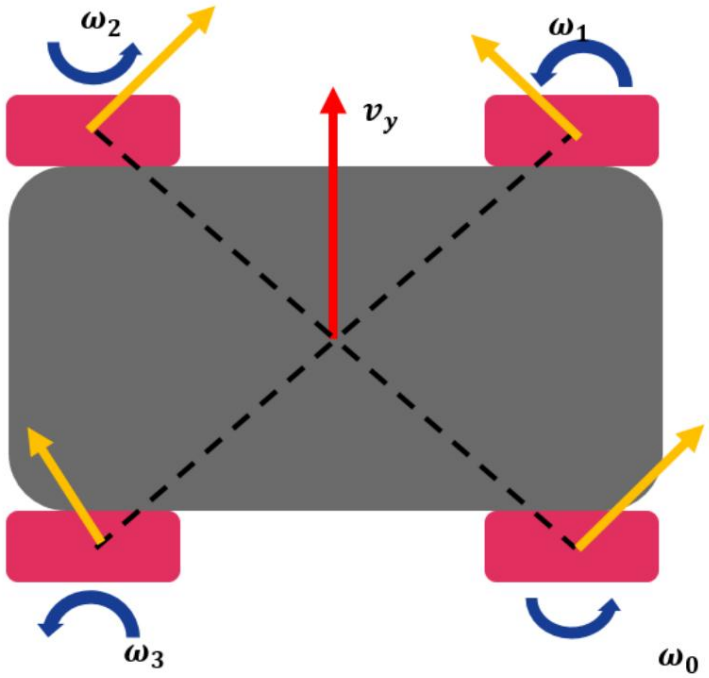
force in the direction.



When the chassis moving speed to the left is ym/s, the four chassis motor speeds are ym/s, -ym/s, ym/s, -ym/s respectively. as shown

It is shown that the forces in the front and rear directions of the first two wheels cancel each other, leaving only the forces in the front and left directions;

The forces cancel each other out, leaving only the left and right forces.



17.6 Course Summary

The chassis is the carrier of robot motion, and its performance determines the motion performance of the robot. Chassis can be completed using Mecanum wheels

The reason is the special structure of the Mecanum wheel and the independent motor speed loop control. Through Chassis Positive Kinematics

Solving, the motion vector of the three degrees of freedom of the chassis can be solved into the speed of the four chassis motors, and then the speed fed back by the motor

Perform speed loop control.

18. Attitude solving task

18.1 Knowledge points

ÿ Introduction to Attitude Angle

ÿ Quaternion and attitude angle conversion

ÿ Mahony algorithm porting

ÿ SPI DMA transfer

18.2 Course Content

In this course, we will learn the attitude calculation task. Attitude calculation is to use the angular velocity data of the gyroscope and the acceleration of the accelerometer.

The data of the degree meter and the magnetic field data of the magnetometer are fused to calculate the attitude angle of the current carrier. The quality of attitude solving algorithm

Will affect the accuracy of the attitude angle. We take the mahony algorithm as an example, transplant related algorithms, and create an attitude solution task.

At the same time, learn how to use the DMA mode of SPI to save CPU processing time.

18.3 Basic Learning

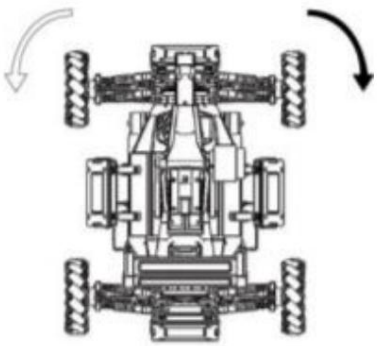
18.3.1 Introduction to Attitude Angle

Pose is the orientation of the reflected object relative to the reference frame. Euler angles are often used to represent the attitude of an object, and it is necessary to describe the attitude of an object.

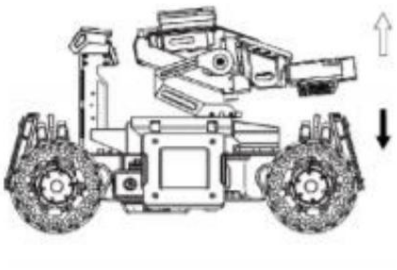
Three angles are required, namely yaw (yaw angle), pitch (pitch angle), and roll (roll angle). Yaw (yaw angle) is the angle around the z-axis

degrees, pitch is the angle around the y-axis, and roll is the angle around the x-axis. RoboMaster robots usually have a two-axis gimbal,

They are the yaw axis and the pitch axis respectively, as shown in the figure:



RoboMaster Robot Yaw Axis



RoboMaster Robot Pitch Axis

18.3.2 Transformation of Quaternion and Attitude Angle

In addition to using Euler angles to represent the attitude, you can also use the unit quaternion to represent the attitude, and the quaternion is represented by 4 parameters. Such as is shown in the following formula:

$$Q = q0 + q1 + q2 + q3 \quad (18\text{-}1)$$

Since it is a unit quaternion, the sum of squares needs to be equal to 1, as shown in the following formula:

$$2\ q0 + q1^{2} + q2^{2} + q3^{2} = 1 \quad (18\text{-}2)$$

When the Euler angle rotation order is yaw-pitch-roll, the formula for converting quaternions into Euler angles is:

$$yaw = \arctan(\ ) \quad (18\text{-}3)$$

$$pitch = \arcsin(2\ ÿ\ q0\ ÿ\ q2\ ÿ\ 2\ ÿ\ q1\ ÿ\ q3) \quad (18\text{-}4)$$

$$roll = \arctan(\ ) \quad (18\text{-}5)$$

The quaternion is integrated using the data of the gyroscope, and the integral iteration formula is:

$$q0 = q0 \quad (18\text{-}6)$$

$$q1 = q1 \quad (18\text{-}7)$$

$$q2 = q2 \quad (18\text{-}8)$$

$$q3 = q3 \quad (18\text{-}9)$$

in:

ÿ , is the x, y, z axis data of the gyroscope,

ÿ t is the timing time, and the program setting update frequency is 1000Hz, so t=0.001.

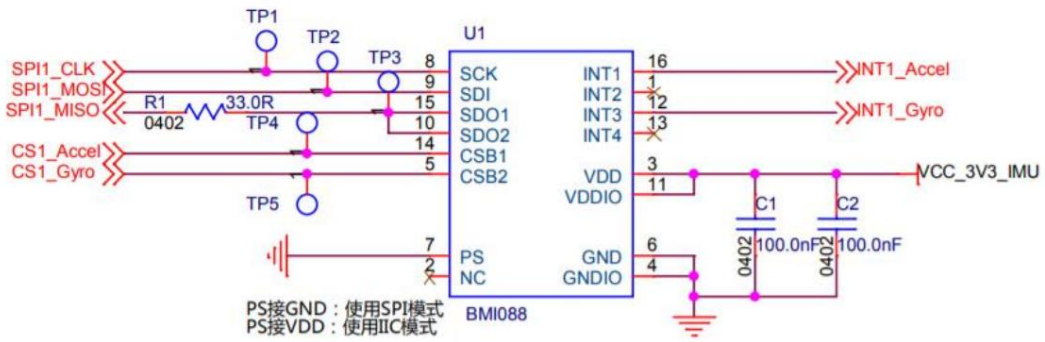
18.4 Program Learning

18.4.1 Configure GPIO, SPI and I2C in cubeMX

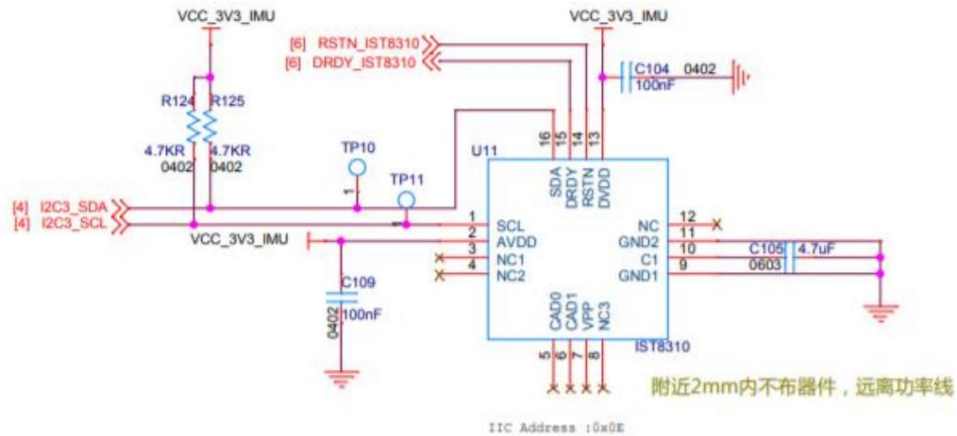
Learn how to configure I2C and SPI in Chapter 11 I2C Lesson and Chapter 13 SPI Lesson. For this course, I

We need to fuse magnetometer data, gyroscope data, accelerometer data, so we need to configure gyroscope and additional gyroscope

And the DRDY pin of the accelerometer, and configure it as an external interrupt mode. The following figure shows the hardware schematic diagram related to the gyroscope.



BMI088 Hardware Schematic



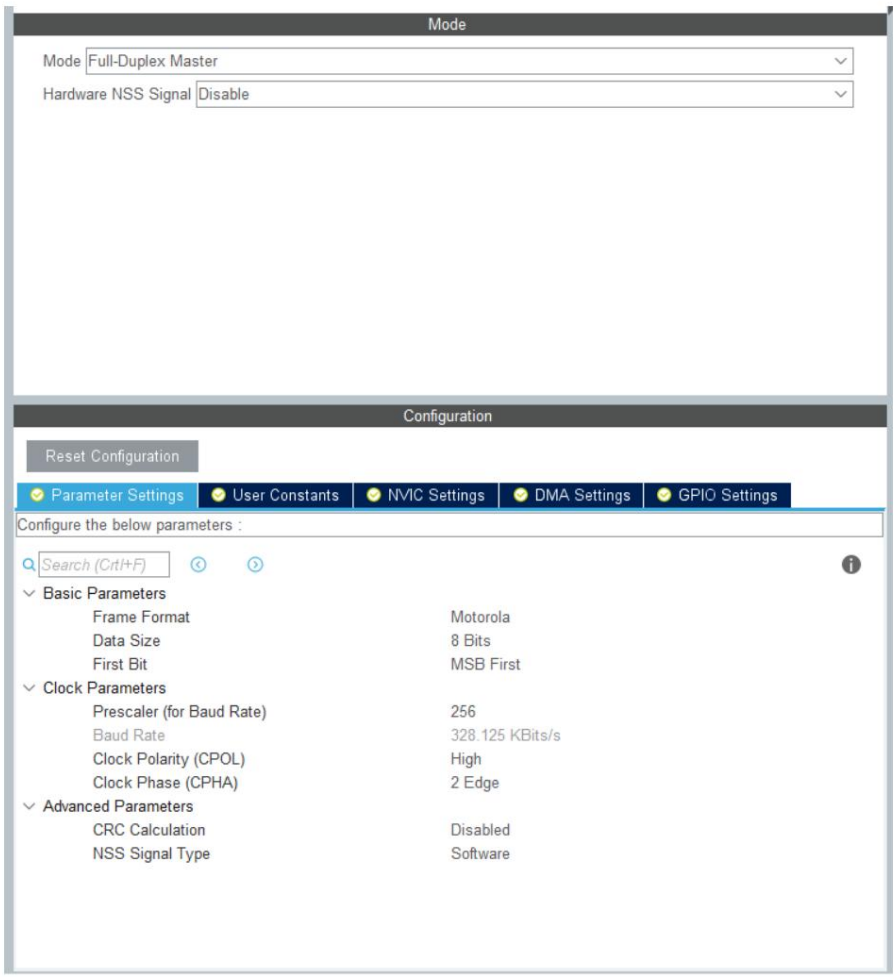
IST8310 hardware schematic

MCU hardware pin correspondence table

MCU pins	Hardware schematic name function	
PA7	SPI1_MOSI	MOSI of SPI1
PB3	SPI1_CLK	SPI1 CLK

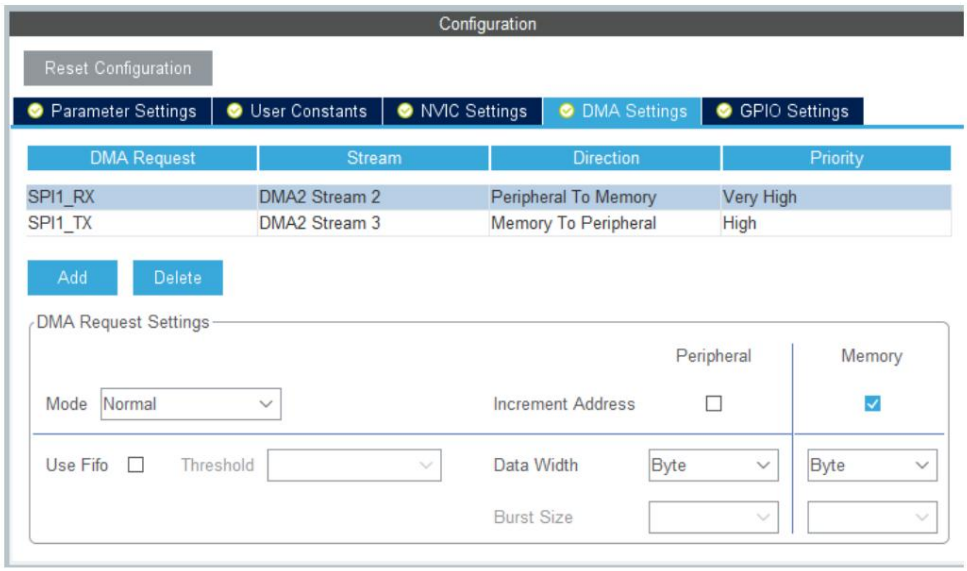
MCU pins	Hardware schematic name function	
PB4	SPI_MISO	MISO for SPI1
PA4	CS1_Accel	Chip select for accelerometer, active low
PB0	CS1_Gyro	Chip select for gyroscope, active low
PC4	INT1_Accel	DRDY of accelerometer, external falling edge interrupt
PC5	INT1_Gyro	DRDY of gyroscope, external falling edge interrupt
PA8	I2C3_SDA	SCL for I2C
PC9	I2C3_SCL	SDA for I2C
PG3	DRDY_IST8310 DRDY of magnetometer, external falling edge interrupt	
PG6	RSTN_IST8310 RSTN of the magnetometer, active low	

In cubeMX, enable SPI1 and configure SPI, as shown in the following figure:



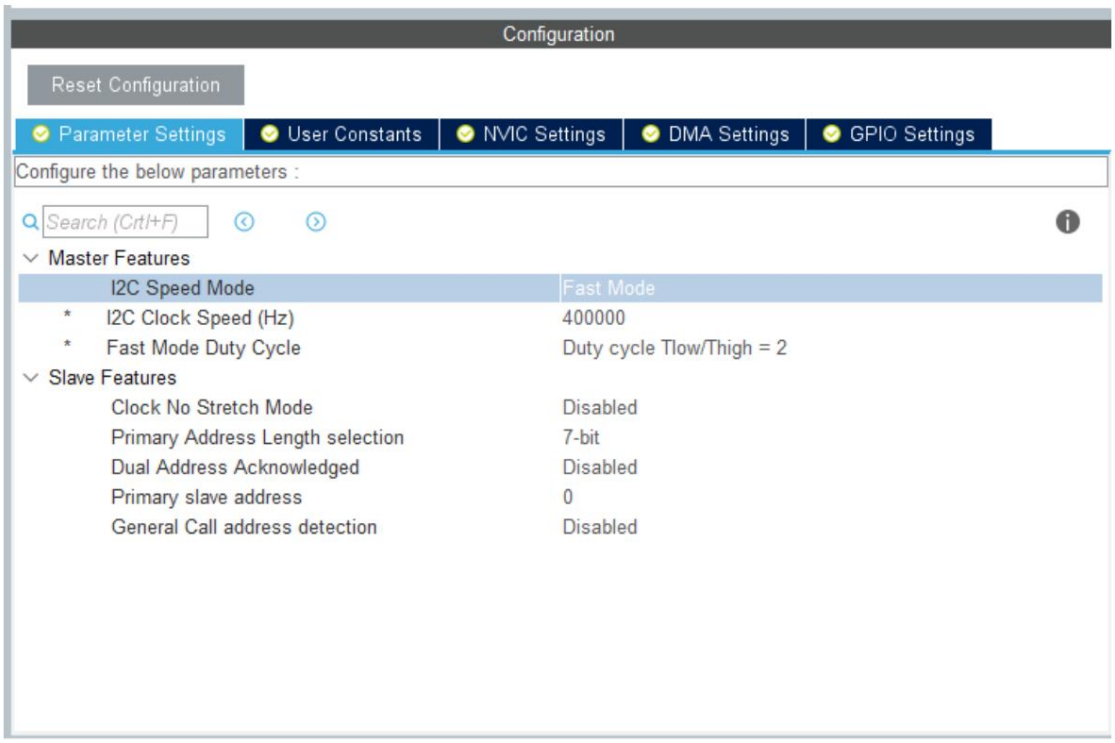
SPI configuration diagram in cubeMX

In the configuration of the SPI in the above figure, click the DMA Setting option, add the DMA of the SPI, and configure the DMA as shown in the figure.



SPI DMA configuration diagram in cubeMX

In cubeMX, enable I2C3 and configure I2C as shown below:



I2C configuration diagram in cubeMX

18.4.2 **Mahony** Algorithm Porting

The mahony algorithm is a common attitude fusion algorithm, which combines the accelerometer, magnetometer, and gyroscope with a total of nine-axis data to solve the problem.

To export the quaternion, the Mahony algorithm can go to the following website to download the source code.

https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/

Download the compressed package madgwick_algorithm_c.zip, decompress and transplant the files under the MahonyAHRS file to the program tool within the program. The original file has the following parameters.

parameter	significance	Settings
sampleFreq sampling frequency		1000.0f (raw value 512.0fÿ
twoKpDef	Accelerometer and magnetometer fusion weight Kp, the larger the value, the more The faster the convergence speed.	2.0f*0.5f
twoKpDef	Accelerometer and magnetometer fusion weight Ki, the larger the value, the convergence faster.	2.0f*0.0f
twoKp	twoKpDef macro assignment variable	twoKpDef
YOUR	twoKiDef macro assignment variable	twoKiDef
q0,q1,q2,q3 Quaternion		[1.0,0,0,0]
integralFBx integralFBy integralFBz	Error integral value	0.0f

For the convenience of use, the original function is modified on the parameter input, and the quaternion array variable is introduced.

1. MahonyAHRSupdate function

```
void MahonyAHRSupdate(float q[4], float gx, float gy, float gz, float ax, float ay, float az, float mx, float my, float mz)
```

Function name	MahonyAHRSupdate
Features	Fuse gyroscope, accelerometer, magnetometer data, update quaternion
function return	None
Parameter 1: q[4]	quaternion array
Parameters 2, 3, 4: gx, gy, gz	Gyroscope angular velocity data
Parameters 5, 6, 7: ax, ay, az	Accelerometer acceleration data
Parameters 8, 9, 10: mx, my, mz	Magnetometer Magnetic Field Data

A brief introduction to the function implementation process:

1. Normalize the accelerometer data and magnetometer data so that the sum of the squares of the acceleration and magnetic field data is equal to 1.

which is:

$$a^2 \quad 2 + a \quad 2 + a \quad = 1$$

$$2\ 2 + m \quad + \quad {}^2 = 1$$

2. Rotate the magnetic field data to calculate the current quaternion attitude direction.

3. The current quaternion attitude direction is different from the acceleration measurement direction and the magnetometer measurement direction

4. Perform PI compensation for gyroscope data.

5. According to formulas 18-6, 18-7, 18-8, 18-9, use the gyroscope data to update the quaternion data. The specific code is as follows:

```
gx   *= (0.5f * (1.0f / sampleFreq)); // pre-multiply common factors

 gy   *= (0.5f * (1.0f / sampleFreq));

 gz   *= (0.5f * (1.0f / sampleFreq));

 qa = q [0];

 qb = q[1];

 qc = q[2];

 q[0] += (-qb * gx - qc                *  gy - q[3]       * gz);

 q[1] += (qa * gx + qc                 *  gz - q[3] * gy);

 q[2] += (qa * gy - qb * gz + q[3] * gx);

 q[3] += (qa * gz + qb * * qx); qy - qc
```

The process of this procedure is as follows:

1. Multiply the gyroscope data by 0.5*1/sampleFreq, where sampleFreq is the sampling frequency, 1/sampleFreq

is the update interval.

2. Assign q0, q1, q2 to qa, qb, qc,

3. According to the quaternion update formula 18-6, 18-7, 18-8, 18-9, update the quaternion.

For the specific process and implementation principle of the fusion algorithm, please refer to the documents on the website

ÿmadgwick_internal_report.pdfÿÿ

2. MahonyAHRSupdateIMU function


```
void MahonyAHRSupdateIMU(float q[4], float gx, float gy, float gz, float ax, float ay, float az)
```

Function name	MahonyAHRSupdateIMU
Features	Fusion gyroscope, acceleration, update quaternion
function return	None
Parameter 1: q[4]	quaternion array
Parameters 2, 3, 4: gx, gy, gz	Gyroscope angular velocity data
Parameters 5, 6, 7: ax, ay, az	Accelerometer acceleration data

3. invSqrt function

```
float invSqrt(float x)
```

Function name	invSqrt
Features	Calculate the reciprocal of the square root, i.e. $\frac{1}{\sqrt{y}}$
function return	reciprocal of square root
Parameter 1:x	floating point number to be calculated

4. AHRS_init function

```
void AHRS_init(fp32 quat[4], fp32 accel[3], fp32 mag[3])
```

Function name	AHRS_init
Features	Initialize quaternion from accelerometer, magnetometer data
function return	None
Parameter 1:quat[4]	quaternion array
Parameter 2:accel[3]	Accelerometer

Parameter 3: mag[3]	Magnetic Field Strength Data

This function assigns the quaternion to [1.0, 0.0f, 0.0f, 0.0f], which is the initial value of the quaternion.

5. AHRS_update function

```
void AHRS_update(fp32 quat[4], fp32 time, fp32 gyro[3], fp32 accel[3], fp32 mag[3])
```

Function name	AHRS_update
Features	According to the angular velocity data of the gyroscope, the acceleration data of the accelerometer, and the magnetic field data of the magnetometer, the row quaternion iterative calculation
function return	None
Parameter 1: quat[4] Quaternion to be updated	
Parameter 2: time iteration time, unit s, since the attitude calculation task is 1ms, enter 0.001f	
Parameter 3: gyro[3] angular velocity data of gyroscope	
Parameter 4: accel[3] Accelerometer acceleration data	
Parameter 5: mag[3] Magnetic field strength data of magnetometer	

This function calls the MahonyAHRSupdate function for iterative calculation.

6. get_angle function

```
void get_angle(fp32 q[4], fp32 *yaw, fp32 *pitch, fp32 *roll)
{
    *yaw = atan2f(2.0f*(q[0]*q[3]+q[1]*q[2]), 2.0f*(q[0]*q[0]+q[1]*q[1])-1.0f);
    *pitch = asinf(-2.0f*(q[1]*q[3]-q[0]*q[2]));
    *roll = atan2f(2.0f*(q[0]*q[1]+q[2]*q[3]),2.0f*(q[0]*q[0]+q[3]*q[3])-1.0f);
}
```

The calculation formula of this program refers to the formulas 18-3, 18-4 and 18-5 for converting quaternions into Euler angles in Basic Learning.

Function name	get_angle
function	Get Euler angles based on quaternion
function return	None
Parameter 1: q[4]	quaternion array
Parameter 2: yaw	Yaw Corner Pointer
Parameter 3: pitch	Pitch corner pointer
Parameter 3: roll	Roll angle pointer

18.4.3 Program Flow

After introducing the fusion algorithm and related functions of quaternion, we will introduce the DMA transmission process of SPI, using SPI's DMA transfer process.

DMA transfer can save CPU processing time. Since SPI is MISO, MOSI transmits data at the same time, so

The RX and TX of the SPI need to be turned on at the same time.

1. Enable the DMA transfer function of SPI

```
void SPI1_DMA_enable(uint32_t tx_buf, uint32_t rx_buf, uint16_t ndtr)
```

Function name	SPI1_DMA_enable
function	Enable DMA transfer of SPI1
function return	None
Parameter 1: tx_buf	address to send data to
Parameter 2: rx_buf	address to receive data
Parameter 3: ndtr	Data length

2. DMA scheduling function of SPI

When using SPI communication, it is necessary to ensure that only one device is communicating at the same time, that is, only the data of the gyroscope can be obtained at the same time.

According to the data of the accelerometer or the data of the accelerometer, the SPI communication needs to be scheduled. Position

The flag bits of the three variables gyro_update_flag, accel_update_flag and accel_temp_update_flag,

Represents three different stages of transmitting data, gyroscope, accelerometer and temperature.

variable	stage
gyro_update_flag	BIT [0]: Set to 1 after the falling edge of the gyroscope's data ready external interrupt; BIT [1]: Successfully enable SPI DMA transfer of gyroscope; BIT [2]: The DMA transfer of the SPI of the gyroscope has been completed; BIT [3:7]: Reserved.
accel_update_flag	BIT [0]: After entering the falling edge of the accelerometer's data ready external interrupt 1; BIT [1]: Successfully enable SPI DMA transfer of acceleration data; BIT [2]: Completed SPI DMA transfer of acceleration data; BIT [3:7]: Reserved.
accel_temp_update_flag	BIT [0]: After entering the falling edge of the accelerometer's data ready external interrupt 1ÿ (since the temperature register is inside the accelerometer, it goes into the same interrupt); BIT [1]: Successfully enable SPI DMA transfer of temperature data; BIT [2]: Completed SPI DMA transfer of temperature data; BIT [3-7]: Reserved.

static void imu_cmd_spi_dma(void)

Function name imu_cmd_spi_dma	
The function determines to enable the corresponding chip select and SPI DMA according to the 0th bit of xxx_update_flag	
function returns None	
parameterNone	

3. Main task wake-up function

At the same time, in order to enable the wake-up main task, the external interrupt Line_0 is enabled, and the wake-up function is added to the interrupt function.


```
 else if(GPIO_Pin == GPIO_PIN_0)
{
      //wake up the task
      // wake up the task
      if (xTaskGetSchedulerState() != taskSCHEDULER_NOT_STARTED)
      {
            static BaseType_t xHigherPriorityTaskWoken;
            vTaskNotifyGiveFromISR(INS_task_local_handler, &xHigherPriorityTaskWoken);
            portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
      }
}
```

After the SPI DMA transfer of the gyroscope data is completed, the software enables the external interrupt Line_0.

```
          if(gyro_update_flag & (1 << IMU_UPDATE_SHFITS))
          {
                __HAL_GPIO_EXTI_GENERATE_SWIT (GPIO_PIN_0);
          }
```

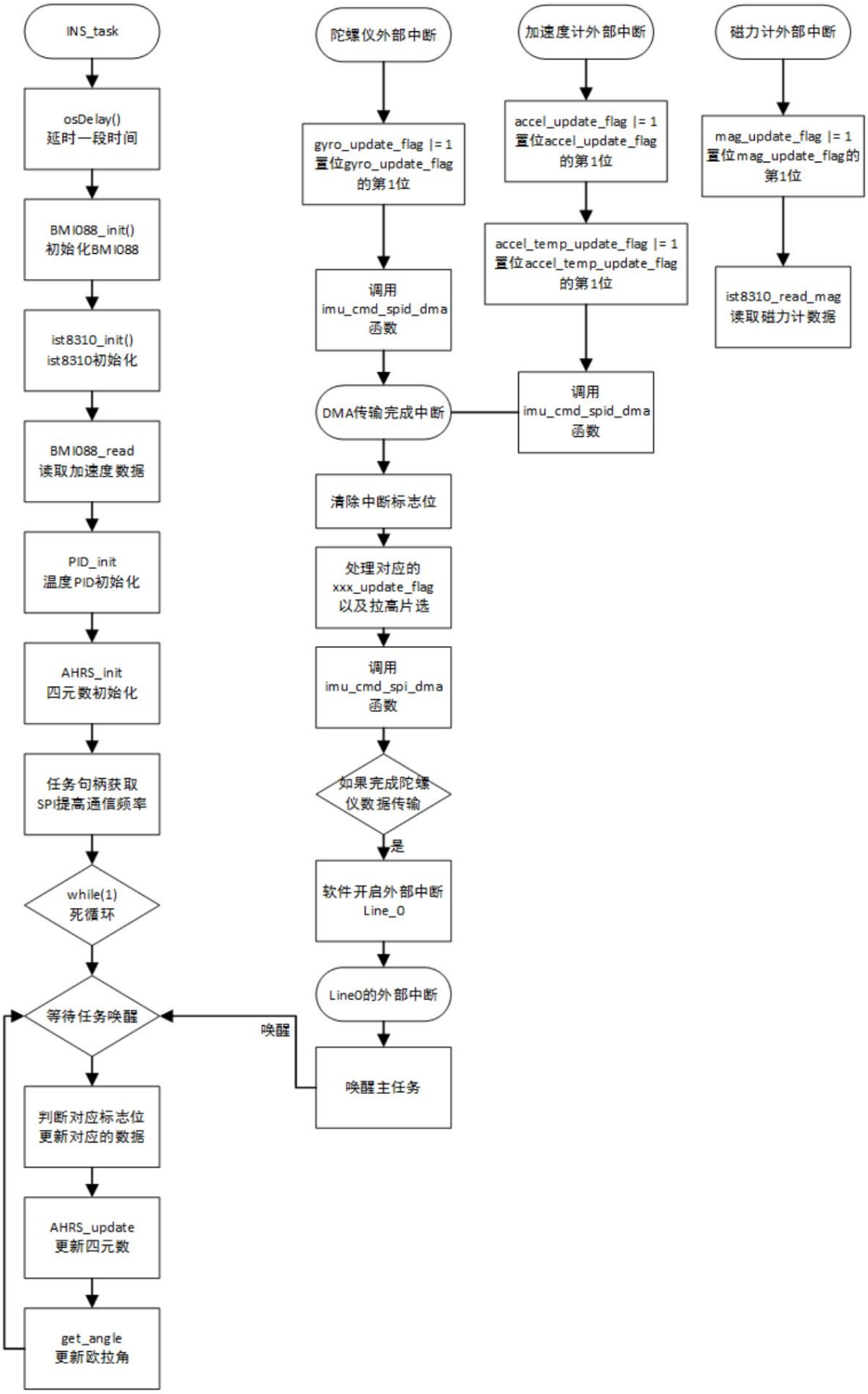
The overall process is as follows. After initializing the gyroscope, accelerometer and magnetometer, the main task enters the waiting state.

The program waits for the external interrupt of the gyroscope and accelerometer, and enables the corresponding SPI DMA transfer in the external interrupt.

After the DMA transfer of the SPI is interrupted, the DMA is interrupted. After the gyroscope data is transferred, the

Turn on the external interrupt Line_0 to wake up the main task. The main task, after waking up, handles gyroscope, acceleration and temperature data

After the data, call the mahony algorithm to integrate the nine-axis data to update the quaternion, calculate the Euler angle, and enter again to wait for wake-up.



Program flow chart

18.4.4 Effect display

Enter Debug mode, you can refer to INS_quat, INS_angle, ist8310_real_data, bmi088_real_data under

data, note that the unit of INS_angle is rad.

Name	Value	Type
INS_quat	0x20006C9C INS_...	float[4]
[0]	0.975852072	float
[1]	0.0300978851	float
[2]	0.0253171232	float
[3]	-0.206903562	float
INS_angle	0x20006CAC INS_...	float[3]
[0]	-0.417758465	float
[1]	0.0618976578	float
[2]	0.0486940593	float
ist8310_real_data	0x20006C44 &ist...	struct ist8310_real...
status	0x00	unsigned char
mag	0x20006C48	float[3]
[0]	20.1000004	float
[1]	34.2000008	float
[2]	106.200005	float
bmi088_real_data	0x20006C20 &bm...	struct BMI088_RE...
status	0x00	unsigned char
accel	0x20006C24	float[3]
[0]	0.000897435879	float
[1]	-0.741282046	float
[2]	9.70128155	float
temp	47.25	float
gyro	0x20006C34	float[3]
[0]	0.00319579337	float
[1]	-0.00319579337	float
[2]	-0.00213052891	float
time	3750	float
<Enter expression>		

Program renderings

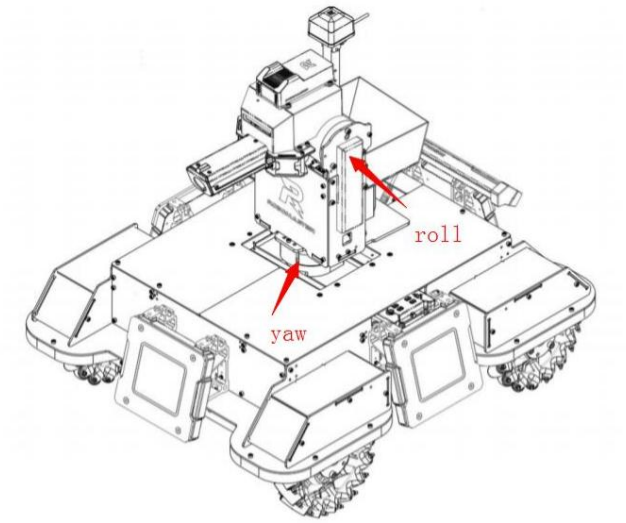
18.5 Advanced Learning

18.5.1 Euler Angle Rotation Order

The Euler angle rotation sequence is divided into two types: yaw-pitch-roll and yaw-roll-pitch, the RoboMaster robot chassis to the gimbal

After passing through the yaw axis motor, and then through the pitch axis motor; therefore, the RoboMaster robot Euler angle rotation sequence is

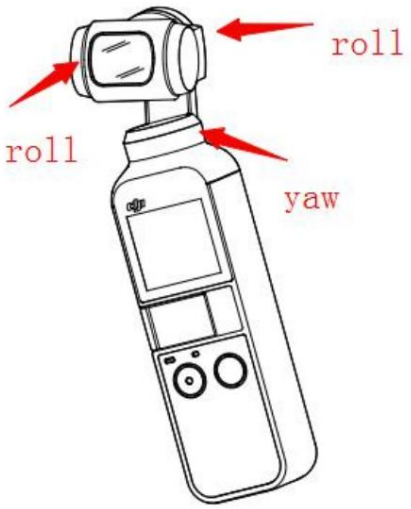
yaw-pitch-rollÿ



RoboMaster Robot Rotation Sequence

Taking the Pocket Osmo as an example, from the Pocket Osmo handle to the camera, through the yaw axis motor, then through the roll axis motor, and finally

After the pitch axis motor, the Euler angle rotation sequence of Osmo Pocket Osmo is yaw-roll-pitch.



Osmo Pocket Rotation Order

When the Euler angle rotation order is yaw-roll-pitch, the formula for converting quaternions into Euler angles is:

$$yaw = \arctan(\ ) \frac{q0 \ddot{y} q3 \ddot{y} q1 \ddot{y} q2}{q0 \ddot{y} q0 + q2 \ddot{y} q2 \ddot{y} 0.5}$$

$$pitch = \arctan(\ ) \frac{q0 \ddot{y} q0 + q0 \ddot{y} q2 \ddot{y} q1 \ddot{y} q3}{q3 \ddot{y} q3 \ddot{y} 0.5}$$

roll = arcsin(2 ÿ q0 ÿ q1 + 2 ÿ q2 ÿ q3)

Which formula is used in the program needs to be related to the structure of the gimbal. If it is the same as the RoboMaster robot, it is yaw

pitch-roll rotation order, use the first set of formulas, if it is yaw-roll-pitch like Pocket Osmo, use the first set of formulas

Two sets of formulas.

18.6 Course Summary

Through the study of this course, master the conversion formula of Euler angle and quaternion, as well as the quaternion update formula, learn mahony

Algorithms, in addition to learning SPI DMA transfers. Attitude angle is an important feedback data of the robot, in the RoboMaster machine

It has important applications in the field of robot PTZ control and automatic aiming.

19. PTZ control tasks

19.1 Key points of knowledge

ÿ The structure of the robot head

ÿ Cascade PID control

ÿ Cascade PID control of robot gimbal angle loop and speed loop

19.2 Course Content

The gimbal is a stabilization mechanism on the robot, which is used to reduce the vibration of the top structure of the robot and stabilize the image. Compared with RoboMaster

In the competition, the robots are equipped with a gimbal structure, which is used to carry devices such as image transmission and laser emission. In this lesson, learn

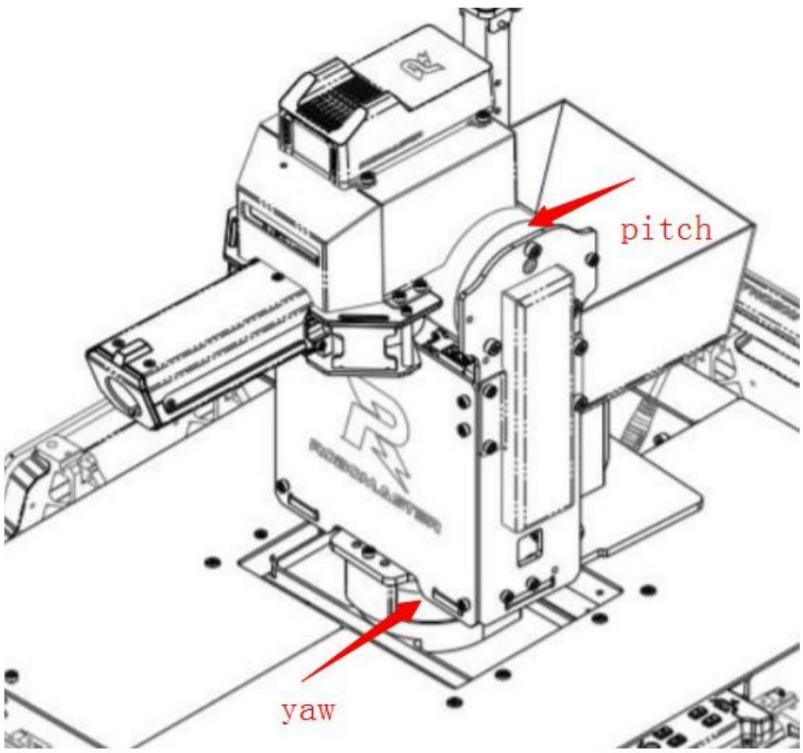
For the structure and control of the PTZ, learn the dual-loop cascade PID control of the PTZ angle loop and angular velocity loop.

19.3 Basic Learning

19.3.1 The structure of the robot head

RoboMaster robot gimbal is a two-axis gimbal structure, which can control the rotation of the yaw axis and the pitch axis. Appearance as

As shown in the figure:

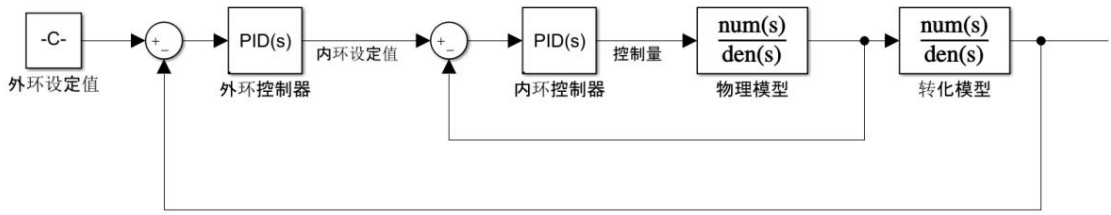


PTZ structure diagram

19.3.2 Cascade PID

Cascade PID is controlled by two controllers in cascade, and the cascade control is often to control different physical quantities of the same degree of freedom. For example, the PTZ control of the robot uses the angular velocity loop and the angle loop cascade control, and the angular velocity and angle belong to the rotation. degrees of freedom, angular velocity is the derivative of angle. The two control links are divided into an outer loop control link and an inner loop control. The general angle belongs to The outer loop control link, the angular velocity belongs to the inner loop control link.

The series connection is to connect the output of the outer loop with the input of the inner loop. The input of the outer loop angle loop is the control target, and the output of the outer loop angle loop is the control target. is the set angular velocity, which is the input of the inner angular velocity loop.



19.4 Program Learning

19.4.1 Can Send Motor Control Function Review

In the CAN communication chapter, learn how to use CAN communication to control the motor. The IDs assigned by the gimbal motors are: 0x205, 0x206, 0x207 and 0x208, they are all controlled by the CAN packet with ID 0x1FF, where the default 0x205 is the yaw axis motor, 0x206 is the pitch axis motor, 0x207 is the plucking motor, and in CAN_receive.c Encapsulates the PTZ control function CAN_cmd_gimbal.

Function name	CAN_cmd_gimbal
function	Send the motor current control value with ID 0x205-0x208
function return	None
Parameter 1: yaw	Motor current control value with ID 0x205, range [-30000, 30000]
Parameter 2: pitch	Motor current control value with ID 0x206, range [-30000, 30000]

Parameter 3: shoot	Motor current control value with ID 0x207, range [-16384,16384]
Parameter 4: res	Motor current control value with ID 0x208, reserved byte

19.4.2 Explanation of program flow

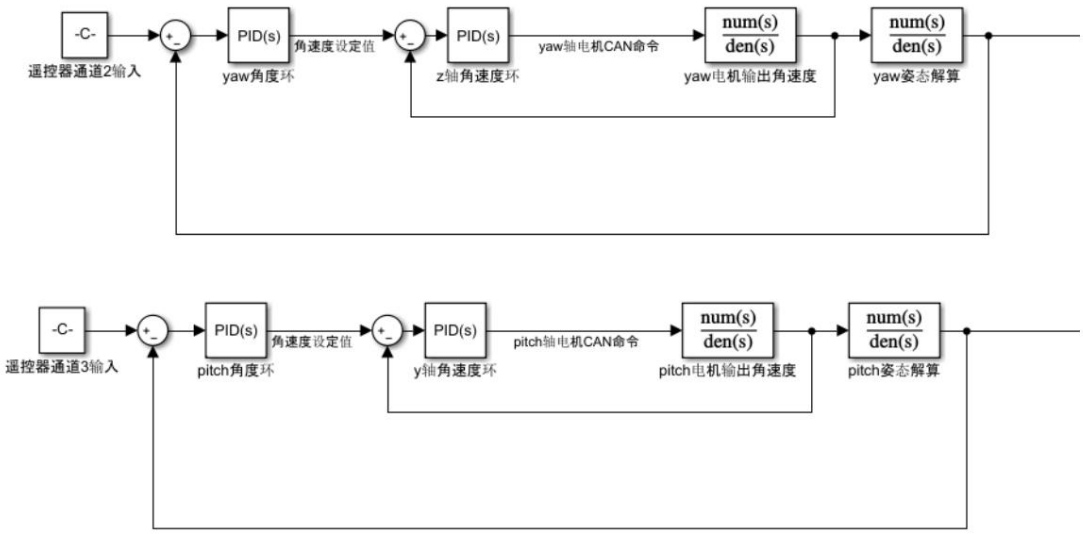
Use freeRTOS to create a gimbal task. The main loop process of the gimbal task is as follows:

process	Features
gimbal_init	PTZ initialization, mainly initialize the motor angle PID, angular velocity PID, initialize the remote controller pointer, initialize the attitude angle pointer, Initialize the motor data pointer.
gimbal_set_mode	The control behavior of the gimbal is determined according to the switch value of the remote control. The gimbal control behavior determines the gimbal control mode.
gimbal_mode_change_control_transit	The data is saved during the switching process of the PTZ control mode to ensure the control Switched control targets transition smoothly.
gimbal_feedback_update	Gimbal motor feedback speed, attitude angle, angular velocity update.
gimbal_set_control	According to the input of the remote control and mouse, the control target of the angle is calculated.
gimbal_control_loop	The PTZ calculates different PIDs according to different control modes.
CAN_cmd_gimbal	Send motor current control value

The common control route of the robot gimbal is shown in the figure. The angle yaw and pitch are calculated from the channel values of the remote control and the mouse.

The control target value of the corresponding axis is calculated by the pan-tilt angle PID, and the angular speed of the motor is calculated by the angular speed PID.

The CAN command control quantity is then sent to the motor through the CAN bus.



19.4.3 Explanation of key functions

Here are a few main functions:

1. gimbal_set_mode function and gimbal_behaviour_mode_set function

The gimbal_set_mode function calls the gimbal_behaviour_mode_set function to set the control mode. and

In the gimbal_behaviour_mode_set function, the different behaviors of the gimbal are determined by the switch of the remote control, and then the

Different PTZ control modes are determined by different behaviors. Please refer to the following table for behavior mode and control mode.

```
static void gimbal_set_mode(gimbal_control_t *set_mode)
{
    gimbal_behaviour_mode_set(set_mode);
}
static void gimbal_behavour_set(gimbal_control_t *gimbal_mode_set)
{
    ...
    //Switch to control the state of the gimbal
    if (switch_is_down(gimbal_mode_set->gimbal_rc_ctrl->rc.s[GIMBAL_MODE_CHANNEL]))
    {
        gimbal_behaviour = GIMBAL_ZERO_FORCE;
    }
    else if (switch_is_mid(gimbal_mode_set->gimbal_rc_ctrl->rc.s[GIMBAL_MODE_CHANNEL]))
    {
        gimbal_behaviour = GIMBAL_RELATIVE_ANGLE;
    }
    else if (switch_is_up(gimbal_mode_set->gimbal_rc_ctrl->rc.s[GIMBAL_MODE_CHANNEL]))
    {
        gimbal_behaviour = GIMBAL_ABSOLUTE_ANGLE;
    }
    ...
}
```


behavior pattern	PTZ performance and application
GIMBAL_ZERO_FORCE	The control value sent by the gimbal motor CAN is 0, and the gimbal is in a state of weakness. state, it is used when the remote control switch is in the lower position and the gimbal needs to stop moving. occasion.
GIMBAL_INIT	In gimbal initialization mode, the gimbal first slowly lifts the pitch axis, and then rotates yaw axis to the center point of the gimbal, applied to the gimbal from stop motion to start motion Controlled process stage to prevent excessive movement of the gimbal during power-on Mechanism is damaged.
GIMBAL_CALC	The gimbal calibration mode is used when the gimbal calculates the median value of the gimbal. Put down the pitch axis, then lift the pitch axis, and then rotate the yaw axis counterclockwise, Finally rotate the yaw axis clockwise. In this process, the feedback of the motor is collected. The feed angle is used to calculate the median value of the gimbal.
GIMBAL_ABSOLUTE_ANGLE	The gimbal uses the attitude angle calculated from the attitude to control the angle. The angle is relative to the ground coordinate system and does not change with the attitude angle of the chassis. It is suitable for normal motion control of robots.
GIMBAL_RELATIVE_ANGLE	The gimbal uses the angle of the motor feedback for angle control, due to the motor angle It is relative to the chassis coordinate system, following the attitude angle of the chassis, suitable for machine Human control motion control in special occasions.
GIMBAL_MOTIONLESS	The gimbal is still, to ensure the original motor angle control, suitable for machine Motion control for people who are stationary for long periods of time to reduce gyroscope drift impact.

There are three types of PTZ motor control modes, as shown in the table below.

control mode	Features and Applications
GIMBAL_MOTOR_RAW	The PTZ motor control value directly sends the CAN packet, which is GIMBAL_CALI and GIMBAL_ZERO_FORCE selected motor control mode.
GIMBAL_MOTOR_GYRO	The control target of the gimbal motor is the angle calculated by the gyroscope, which is GIMBAL_ABSOLUTE_ANGLE Selected motor control mode.

control mode	Features and Applications
GIMBAL_MOTOR_ENCONDE	The control target of the gimbal motor is the angle of the motor feedback, which is GIMBAL_RELATIVE_ANGLE GIMBAL_MOTIONLESS, motor control selected by GIMBAL_INIT control mode.

2. gimbal_set_control function and gimbal_behaviour_conotrol_set function

The gimbal_set_control function calls the gimbal_behaviour_conotrol_set function to get the two degrees of freedom of the gimbal

The control target value of the motion. The gimbal_behaviour_control_set function is called according to different gimbal behavior modes

For different functions, set different control quantities in the corresponding functions.

```
static void gimbal_set_control(gimbal_control_t *set_control)
{
    fp32 add_yaw_angle = 0.0f;
    fp32 add_pitch_angle = 0.0f;

    gimbal_behaviour_control_set(&add_yaw_angle, &add_pitch_angle, set_control);
    //yaw motor mode control
    if (set_control->gimbal_yaw_motor.gimbal_motor_mode == GIMBAL_MOTOR_RAW)
    {
        //In raw mode, send the control value directly
        set_control->gimbal_yaw_motor.raw_cmd_current = add_yaw_angle;
    }
    else if (set_control->gimbal_yaw_motor.gimbal_motor_mode == GIMBAL_MOTOR_GYRO)
    {
        //In gyro mode, gyro angle control
        gimbal_absolute_angle_limit(&set_control->gimbal_yaw_motor, add_yaw_angle);
    }
    else if (set_control->gimbal_yaw_motor.gimbal_motor_mode == GIMBAL_MOTOR_ENCONDE)
```



```
        {
            //Enconde mode, motor code angle control
            gimbal_relative_angle_limit(&set_control->gimbal_yaw_motor, add_yaw_angle);
        }
        ...
}
```

3. gimbal_absolute_angle_limit function

The gimbal_absolute_angle_limit function is to limit the maximum angle of the gimbal in the gyro angle control mode

In order to prevent the gimbal from rotating beyond the maximum angle and damage the limit mechanism.

1) Calculate the error angle between the currently set target angle and the current angle;

2) Determine whether the angle of the motor feedback plus the error angle and the increase angle will exceed the maximum relative angle;

3) If it exceeds the maximum angle, modify and increase the angle;

4) Add the increase angle to the set target angle.

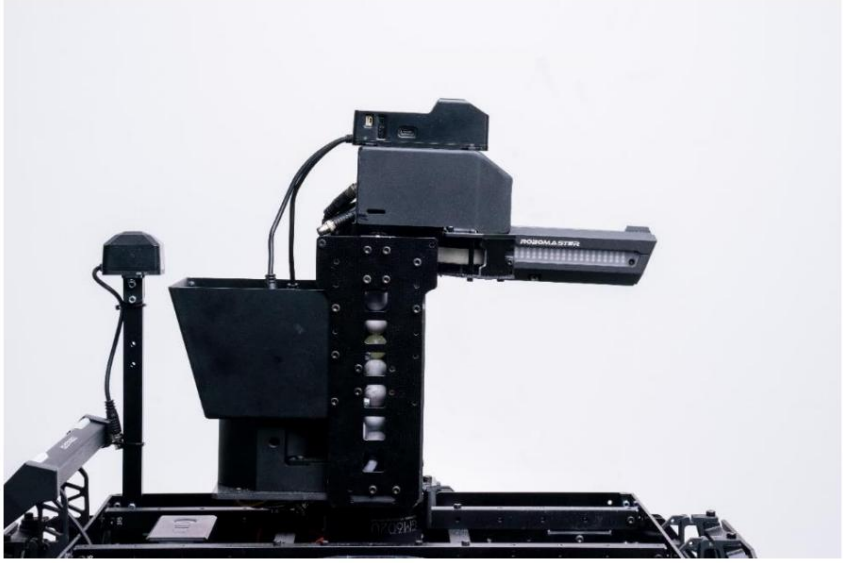
```
static void gimbal_absolute_angle_limit(gimbal_motor_t *gimbal_motor, fp32 add)
{
        static fp32 bias_angle;
        static fp32 angle_set;
        if (gimbal_motor == NULL)
        {
            return;
        }
        //now angle error
        //Current control error angle
        bias_angle = rad_format(gimbal_motor->absolute_angle_set - gimbal_motor->absolute_angle);
        //relative angle + angle error + add_angle > max_relative angle
        //Gimbal relative angle + error angle + new angle if it is greater than the maximum mechanical angle
```



```
    if (gimbal_motor->relative_angle + bias_angle + add > gimbal_motor->max_relative_angle)
    {
        //If it is to control the direction to the maximum mechanical angle
        if (add > 0.0f)
        {
            //calculate max add_angle
            // Calculate a maximum addition angle,
            add = gimbal_motor->max_relative_angle - gimbal_motor->relative_angle - bias_angle;
        }
    }
    else if (gimbal_motor->relative_angle + bias_angle + add < gimbal_motor->min_relative_angle)
    {
        if (add < 0.0f)
        {
            add = gimbal_motor->min_relative_angle - gimbal_motor->relative_angle - bias_angle;
        }
    }
    angle_set = gimbal_motor->absolute_angle_set;
    gimbal_motor->absolute_angle_set = rad_format(angle_set + add);
}
```

19.5 Effect display

The remote control can be used to control the PTZ, and the PTZ remains horizontal as shown in the figure.



19.6 Course Summary

The gimbal is a key component of the RoboMaster robot. The gimbal structure is generally a yaw-pitch two-axis gimbal structure.

Line yaw and pitch axis control, used to stabilize the image and carry the launch mechanism, etc. The performance of the gimbal will directly affect the operating experience

And automatic aiming, in order to improve the response and improve the control performance of the PTZ, cascade PID control is often used.

20. Introduction to robot functions

20.1 Key points of knowledge

ÿ Introduction to the overall functions of the robot

ÿ Introduction to other tasks of the robot

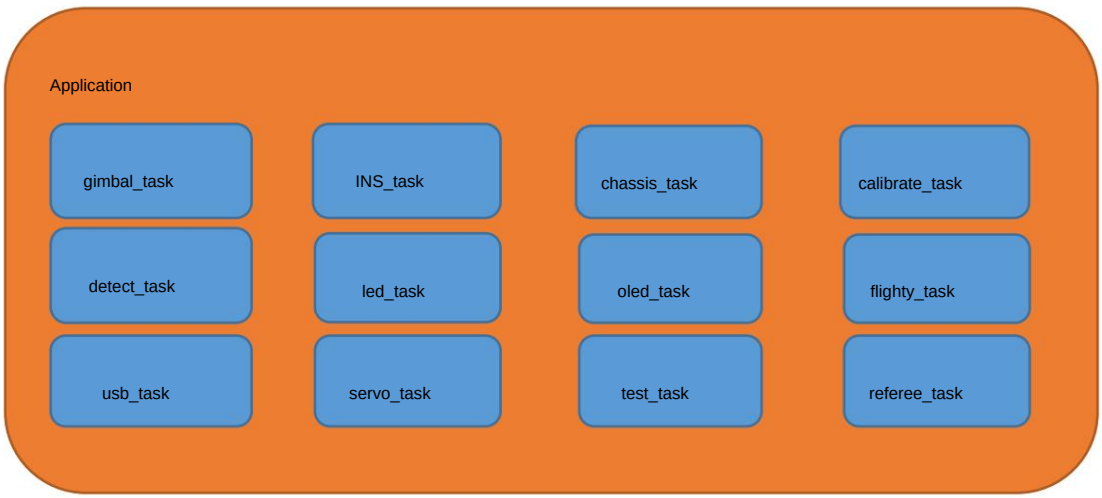
20.2 Course Content

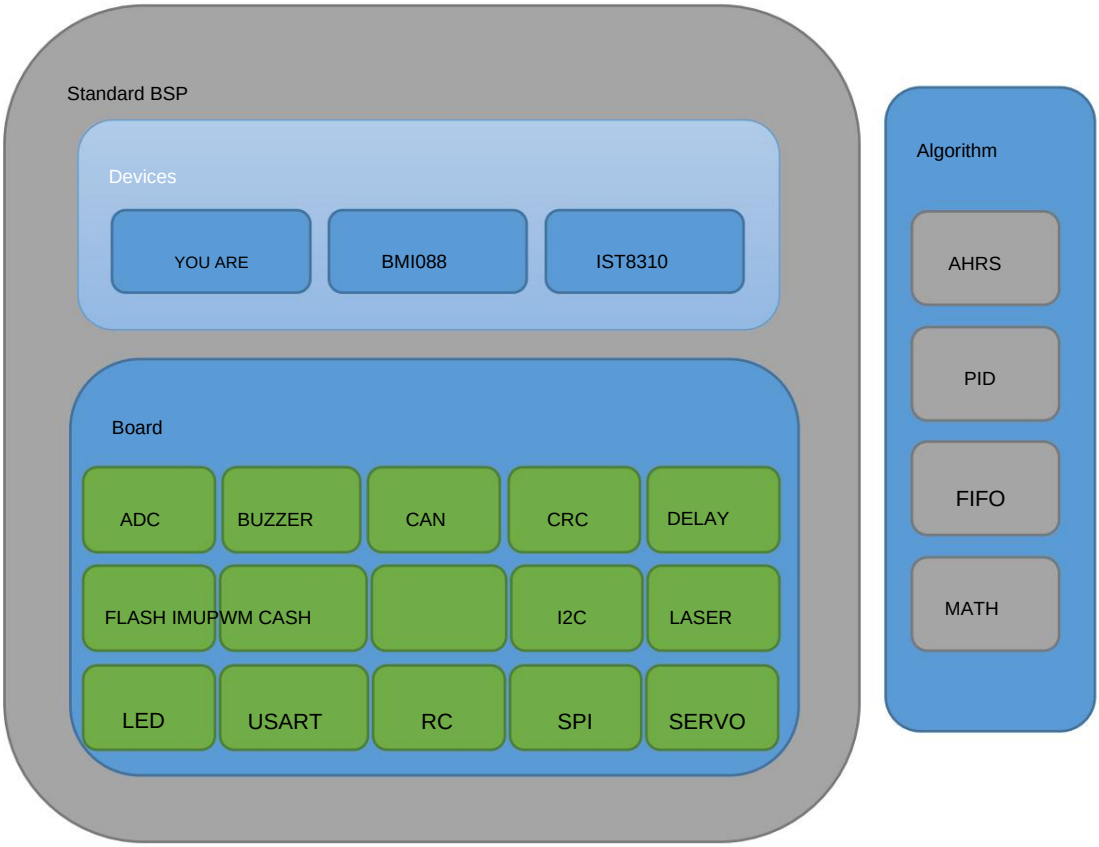
In addition to the basic chassis motion control, gimbal motion control and other functions, the RoboMaster robot also needs other auxiliary functions To better play a role in the competition, such as offline inspection, calibration and saving data, serial port analysis of the referee system and other functions. this chapter The content will introduce the overall function of the robot and explain some task functions.

20.3 Basic Learning

20.3.1 Robot Software Framework

The software framework of the robot is shown below, mainly including the application layer and the BSP layer. The BSP layer provides access to the underlying hardware Function package and driver implementation of bmi088, ist8310 and OLED, the algorithm layer implements attitude calculation algorithm, PID control control algorithms, FIFO data structures, and common mathematical processing functions. The Application layer is mainly implemented for task functions, including Including gimbal tasks, attitude calculation tasks, chassis tasks, calibration tasks, offline detection tasks, etc.





20.3.2 Function introduction

The following is a brief introduction to the application layer. For the peripheral calling relationship, please refer to the last calling relationship section.

1. Calibration function (calibrate_task): Provides gimbal calibration, gyro zero drift calibration, and chassis reset ID functions

2. Chassis control function (chassis_task): completes the McLun motion control of the chassis, the chassis power control, and provides 4 kinds of control functions.
   Control mode: follow the gimbal angle closed-loop control, follow the chassis angle closed-loop control, chassis rotation without angle closed-loop control,
   Native CAN control.

3. Offline judgment function (detect_task): judge whether the device is offline according to the time stamp of the data feedback.

4. Gimbal control function (gimbal_task): complete the angle control of the gimbal. Provides 3 control modes, gyro angle
   control, motor encoder angle control, native CAN control.

5. Attitude calculation function (ins_task): complete the angle fusion of the gyroscope accelerometer and solve the Euler angle.

6. LED RGB switching (led_trigger_task): use three-color LED to complete RGB display, breathing light effect. use
   It is used to display whether the program has crashed.

7. OLED display function (oled_task): display the battery power and device error information, which is convenient for users to locate
   question.

8. Referee system data parsing (referee_usart_task): Use single-byte parsing of referee system data, applicable to 2019

Annual referee system, the referee system needs to be upgraded to the final version.

9. Remote control data analysis (remote_control): Use the serial port idle interrupt function to analyze the data sent by the receiver.

10. Servo control (servo_task): Output the 4 idle PWM servo signals and control them through buttons, which is convenient

Then add magazine controls or simple mechanics.

11. Shooting control (shoot): control the bomb feeder to complete the firing logic.

12. Power sampling (volage_task): Sampling the power supply voltage and estimating the current battery power, as a simple power judgment,

It is used when the battery is inside the robot and it is inconvenient to observe the power.

20.4 Program Learning

We introduced the gimbal control task, attitude solution task, and chassis control task in the previous chapters. The following only introduces offline

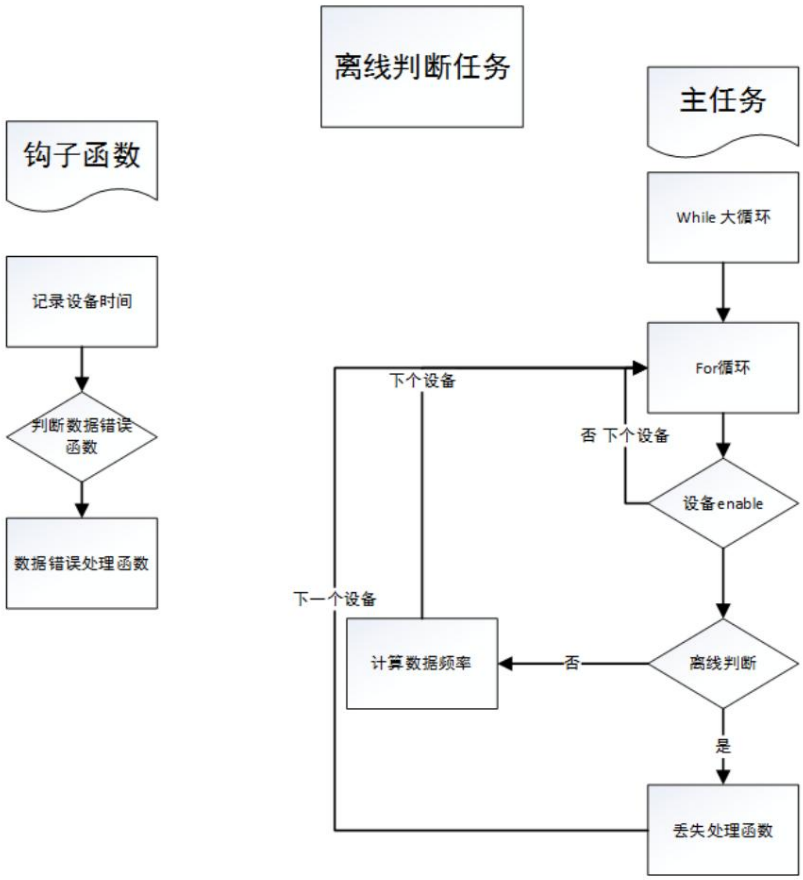
Detection tasks, calibration tasks and referee system unpacking tasks, as well as OLED tasks.

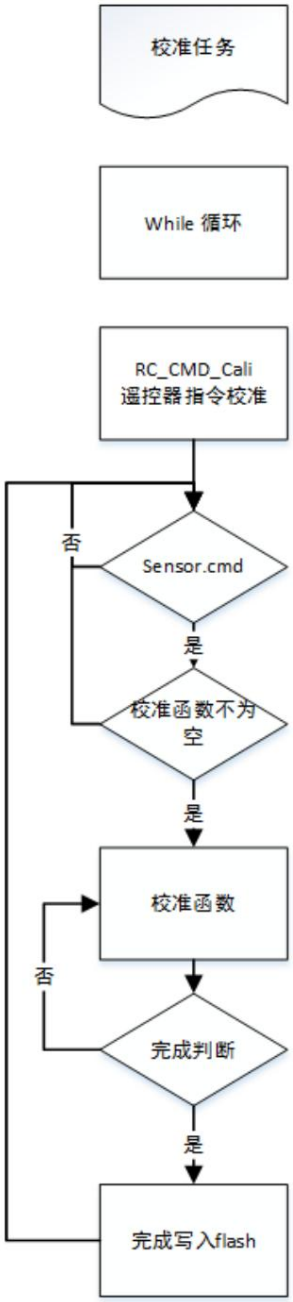
20.4.1 Introduction to Calibration Tasks and Offline Tasks

The calibration task mainly completes the zero drift calibration of the gyroscope, the median calibration of the gimbal, and the chassis enters the quick setting ID mode. module offline

The judgment task mainly judges whether the connection is dropped by judging the difference between the data sending time of the module and the current system time. these two

These tasks are mainly done through pointer functions. The flowchart is shown below.





The data structure used by the offline detection task is as follows:

```
typedef __packed struct
{
    uint32_t new_time;
    uint32_t last_time;
    uint32_t lost_time;
    uint32_t work_time;
```

```
    uint16_t set_offline_time : 12;
    uint16_t set_online_time : 12;
    uint8_t enable : 1;
    uint8_t priority : 4;
    uint8_t error_exist : 1;
    uint8_t is_lost : 1;
    uint8_t data_is_error : 1;
    fp32 frequency;
    bool_t (*data_is_error_fun)(void);
    void (*solve_lost_fun)(void);
    void (*solve_data_error_fun)(void);
} error_t;
```

variable	Features
New_time	Device data update time
Last_time	Device data last update time
Lost_time	Device offline lost time
Work_time	Device online time
Set_offline_time	Determine the data interruption time when the device is offline, for the device data update time exceeds the device After setting the value, it is judged that the device is offline
Set_online_time	The stabilization time after the device goes online, which is used to determine that the set value is positive after the device is online. often
enable	Device enable
Priority	Device priority

variable	Features
Error_exist	Whether the device has errors, including offline and abnormal data
Is_lost	Is the device offline
Data_is_error	Whether the device data is wrong
frequecy	Device update frequency
Data_is_error_fun Function pointer for device data judgment error	
Solve_lost_fun	Function pointer for device offline repair
Solve_data_error_fun Function pointer for device data error fix	

The data structures used in the calibration task are shown below.

```
typedef __packed struct
{
    uint8_t name[3];                                    //device name
    uint8_t cali_done;                                  //0x55 means has
been calibrated
    uint8_t flash_len: 7;                               //buf lenght
    uint8_t cali_cmd : 1;                               //1 means to run cali
hook function,
    uint32_t *flash_buf;                                //link to device
calibration data
    bool_t (*cali_hook)(uint32_t *point, bool_t cmd); //cali function
} cali_sensor_t;
```

variable	Features
Name	Equipment name
Cali_done	Device calibration flag bit, 0x55 means calibrated
Flash_len	Calibration data length
Cali_cmd	Calibration enable
Flash_buf	Data pointer for calibration data
Cali_hook	function pointer for calibration function

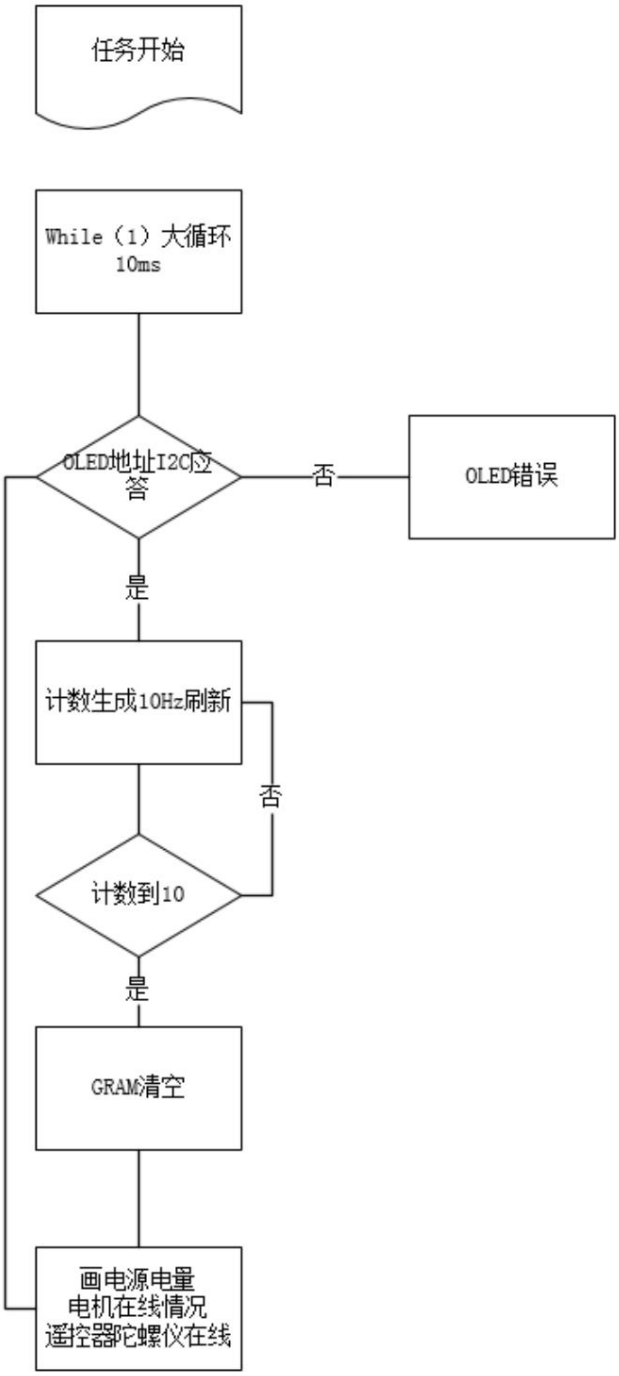
20.4.2 OLED Task Framework

The OLED task confirms the OLED connection by querying the I2C address of the OLED at 100Hz, and refreshes at 10Hz

OLED screen. OLED supports single-color and dual-color modules driven by SSD1306, for single-color and dual-color modules, please refer to the

Just modify the corresponding macro definition in the OLED.C file, as shown below:

```
#define OLED_ONE_COLOR

//#define OLED_TWO_COLOR
```

20.4.3 Unpacking the serial port protocol of the referee system

The referee system sends data through the user serial port of the power module, including information such as robot ID, chassis power, and muzzle heat. number

According to the data error during the transmission process, due to the cable length process, unstable external voltage and other factors. in order to determine

Error information needs to go through the CRC check algorithm to judge the correctness of the data. The serial port protocol of the referee system is as follows:

frame_header (5-byte)	cmd_id(2-byte)	data(n byte)	frame_tail (2-byte, CRC16, whole package check)

It is divided into four parts: frame header, cmd_id, data segment, and frame tail. For details, please refer to the "Referee System Serial Port Protocol".

CRC is an algorithm for checking, which uses the remainder obtained by binary division. And about the CRC check provides the following function.

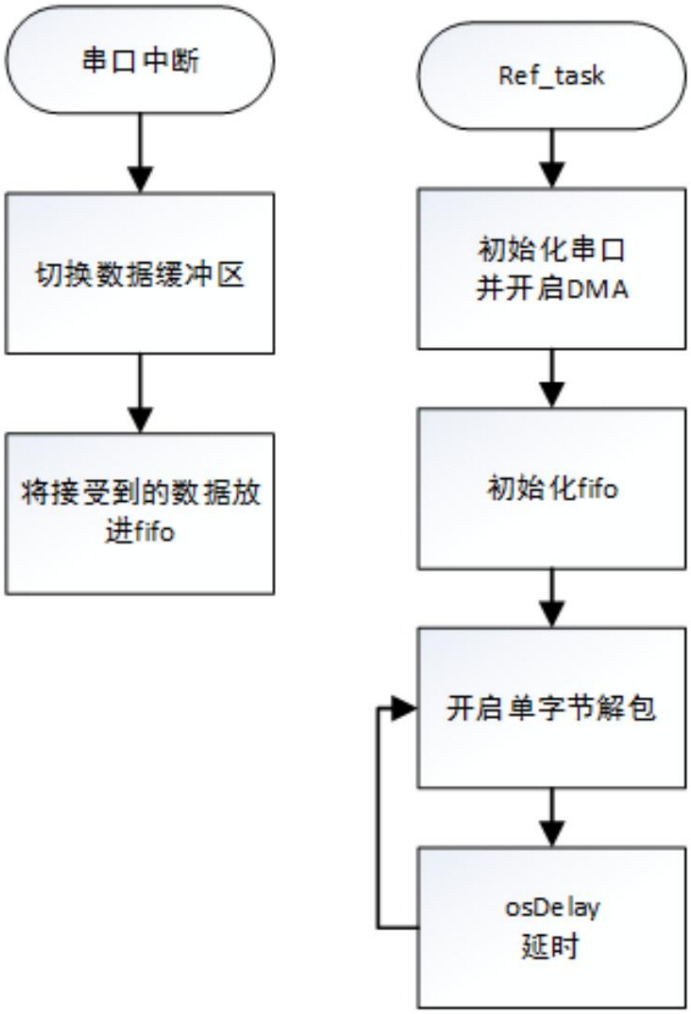
```
uint32_t verify_CRC8_check_sum(unsigned char *pch_message, unsigned int dw_length);
uint32_t verify_CRC16_check_sum(uint8_t *pchMessage, uint32_t dwLength);
```

Function name	Verify_CRC8_check_sum Verify_CRC16_check_sum
function	Check whether a piece of data passes the CRC check
return value	by returning true not by returning flase
Parameter 1: pch_message	data pointer
Parameter 2: dw_length	Data length

```
void append_CRC8_check_sum(unsigned char *pch_message, unsigned int dw_length)
void append_CRC16_check_sum(uint8_t * pchMessage,uint32_t dwLength)
```

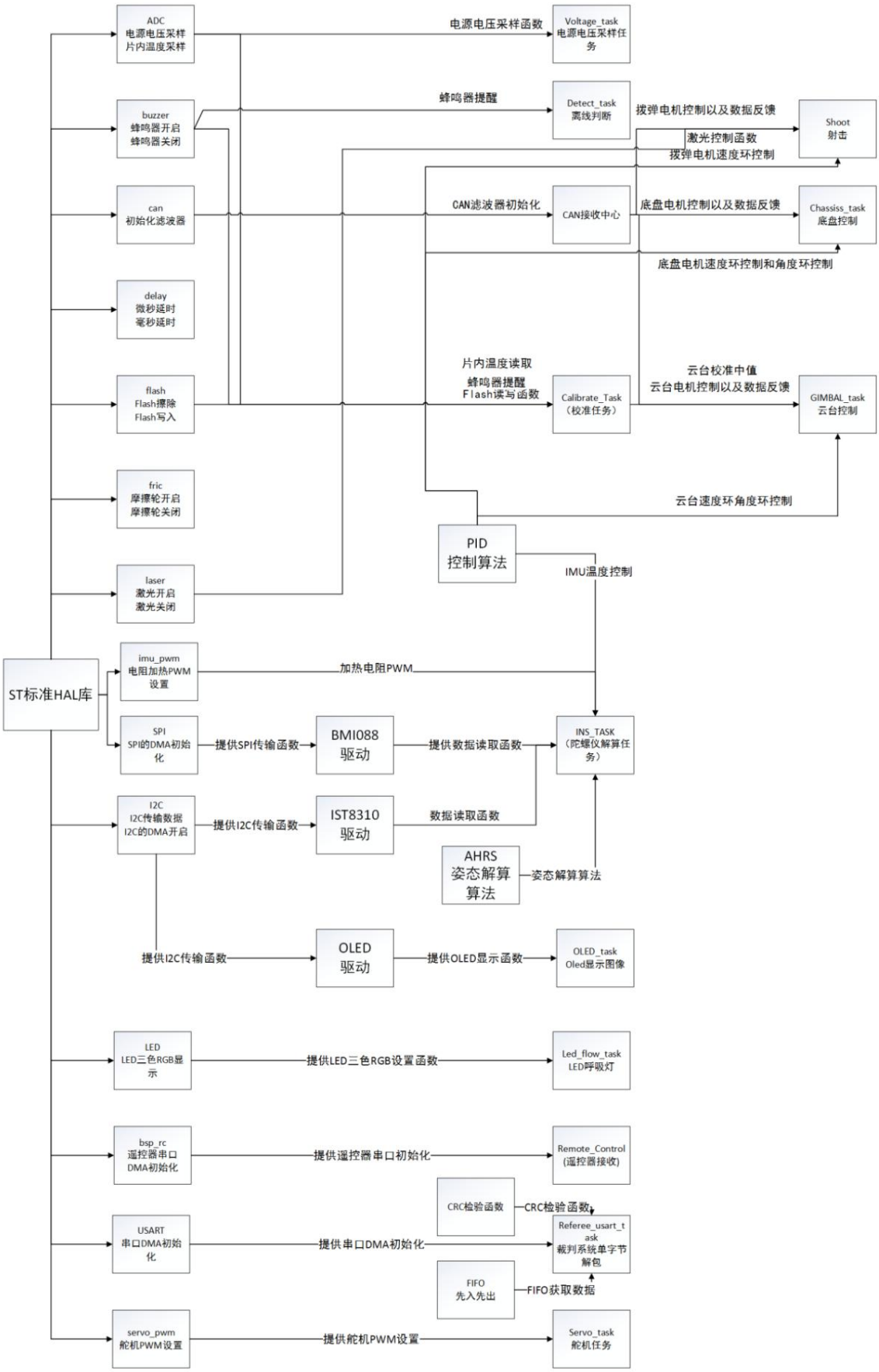
Function name	append_CRC8_check_sum append_CRC16_check_sum
function	Add CRC check value at the end of a piece of data
return value	None
Parameter 1: pch_message	data pointer
Parameter 2: dw_length	Data length

The referee system system unpacking process adopts a single-byte unpacking process, and the unpacking process is as follows.



20.4.4 Peripheral calling relationship

The calling relationship of peripherals is used in all tasks, as shown in the figure:



20.4.5 Effect display

The motion control of the chassis and the gimbal can be performed using the remote controller, as shown in the figure:

20.5 Course Summary

This chapter introduces the overall architecture of the robot, its functional characteristics, and the auxiliary function tasks of the robot, such as offline detection tasks, calibration

Quasi-tasks, referee system unpacking tasks, etc. These auxiliary functions can improve the stability of the robot during the competition and facilitate the operation.

邮箱：robomaster@dji.com
论坛：http://bbs.robomaster.com
官网：http://www.robomaster.com
电话：0755-36383255（周一至周五10:30-19:30）
地址：广东省深圳市南山区西丽镇茶光路1089号集成电路设计应用产业园2楼202