RoboMaster 开发板 A 型嵌入式软件教程文档

v1.0 2022.08

ROBOMASTER 开发板套件

功能丰富　生态系统　多样例程　应用广泛

猎隼战队

电控组 制

特别声明:

此教程基于官方 C 板教程，缺少的相关知识参考：《RoboMaster 开发板 C 型嵌入式软件教程文档.pdf》

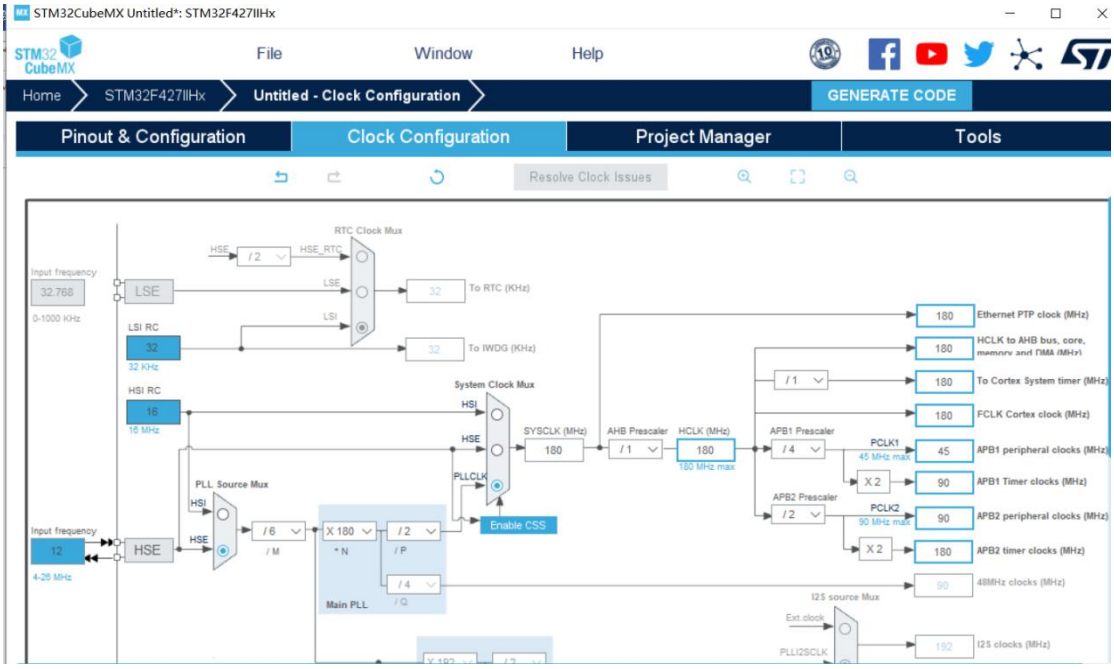
此页放目录

第一章cubeMX 配置

1、New Project->F427IIH6;

2、.在 System Core 下选择 RCC 选项，在 RCC mode and Configuration 中的 High Speed
Clock(HSE)下选择 Crystal/Ceramic Resonator;

3、顶部的 Clock Configuration，进行主频配置；将 Input frequecncy 设置为 12，点击旁边的 HSE 圆形按钮，配置/M 为/6，配置*N 为 X180，配置/P 为/2 ，选择 PLLCLK 圆形按钮，配置 APB1 Prescaler 为/4，配置 APB2 Prescaler 为/2;



4、点击顶部的 Pinout & Configuartion，选择 SYS，在 Debug 下拉框中选择 Serial Wire;

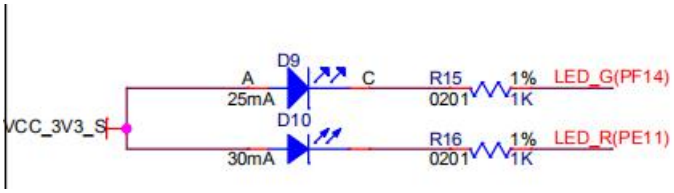
5、点击顶部的 Project Manager，给工程起名，选择存放目录，在 Toolchain/IDE 中选择 MDKARM V5.32;

6、点击旁边的 Code Generator，勾选 Copy only the necessary library files 以及 Generate peripheral initialization as a pair of '.c/.h' files per peripheral;

7、点击顶部的 GENERATE CODE，等待代码生成，打开工程。

第二章点亮 LED

1、通过原理图可以看出 LED_G、LED_R 为 PF14,PE11;



2、在 cubeMX 中配置 GPIO 为输出模式，在 cubeMX 找到对应引脚，配置 GPIO_Output 模式;

3、在 cubeMX 中修改对应引脚的名字。在左侧找到 System core->GPIO； 找到对应的 GPIO，例如 PF14； 在下方的配置单中 user label 填写命名;

4、生成代码,点击 GENERATE CODE 按键。

5、HAL_GPIO_WritePin 函数

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

函数名	**HAL_GPIO_WritePin**
函数作用	**使得对应的引脚输出高电平或者低电平**
返回值	**Void**
参数 1:GPIOx	**对应 GPIO 总线，其中 x 可以是 A...I。** **例如 PH10，则输入 GPIOH**
参数 2:GPIO_Pin	**对应引脚数。可以是 0-15。** **例如 PH10，则输入 GPIO_PIN_10**
参数 3:PinState	**GPIO_PIN_RESET：输出低电平** **GPIO_PIN_SET：输出高电平**

6、程序流程:

程序开始-》HAL_Init 初始化-》SystemClock_Config 时钟配置-》MX_GPIO_Init 引脚配置-》while(1)输出高电平

第三章闪烁 LED

1、采用 PF14,PE11 引脚的输出功能；

2、在 cubeMX 的左侧边栏中的 System Core 下有 GPIO 选项，在该选项下可以看到已经开启的引脚的配置信息；

3、选中需要配置的 GPIO，并查看其详细状态，其中 Maximum output speed 就是可以选择的翻转速度模式。可选的输出速度分为 Low，Medium，High，Very High 四档，一般使用 GPIO 输出驱动 LED 等功能时选择 Low 档翻转速度即可，而一般用于通信的 GPIO 需要设置为 High 或者 Very High，具体设置可以根据相关通信协议对 GPIO 的翻转速度的要求进行设置。

4、HAL_Delay 函数

__weak void HAL_Delay(uint32_t Delay)（使用_weak 修饰符说明该函数是可以用户重定义的）

函数名	**HAL_Delay**
函数作用	**使系统延迟对应的毫秒级时间**
返回值	**void**
参数	**Delay，对应的延迟毫秒数，比如延迟 1 秒就为 1000**

5、HAL_GPIO_TogglePin 函数

void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

函数名	**HAL_GPIO_TogglePin**
函数作用	**翻转对应引脚的电平**
返回值	**Void**
参数 1：GPIOx	**对应 GPIO 总线，其中 x 可以是 A…I。例如 PH10，则输入 GPIOH**
参数 2：GPIO_Pin	**对应引脚数。可以是 0-15。例如 PH10，则输入 GPIO_PIN_10**

6、计数延时：

```
void user_delay_us(uint16_t us)
{
    for(; us > 0; us--)
    {
        for(uint8_t i = 50; i > 0; i--)
        {
            ;
        }
    }
}
void user_delay_ms(uint16_t ms)
{
    for(; ms > 0; ms--)
    {
        user_delay_us(1000);
    }
```

```
}
```

7、 nop 延时：

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

8、 HAL_Delay 延时：

HAL_Delay 函数的实现是基于滴答计时器（Systick）。

滴答定时器也称为 SysTick，是 stm32 内置的倒计时定时器，每当计数到 0 时，触发一次 SysTick 中断，并重载寄存器值。滴答计时器的初始化在 HAL_Init 函数中完成，配置成 1ms 的中断。

9、 程序流程：

程序开始-》HAL_Init 初始化-》SystemClock_Config 时钟配置-》MX_GPIO_Init 引脚配置-》while(1)延时 500ms，翻转 LED 引脚的电平

第四章定时器闪烁 LED

1、驱动 LED 的 GPIO 配置为 LED_G 和 LED_R;

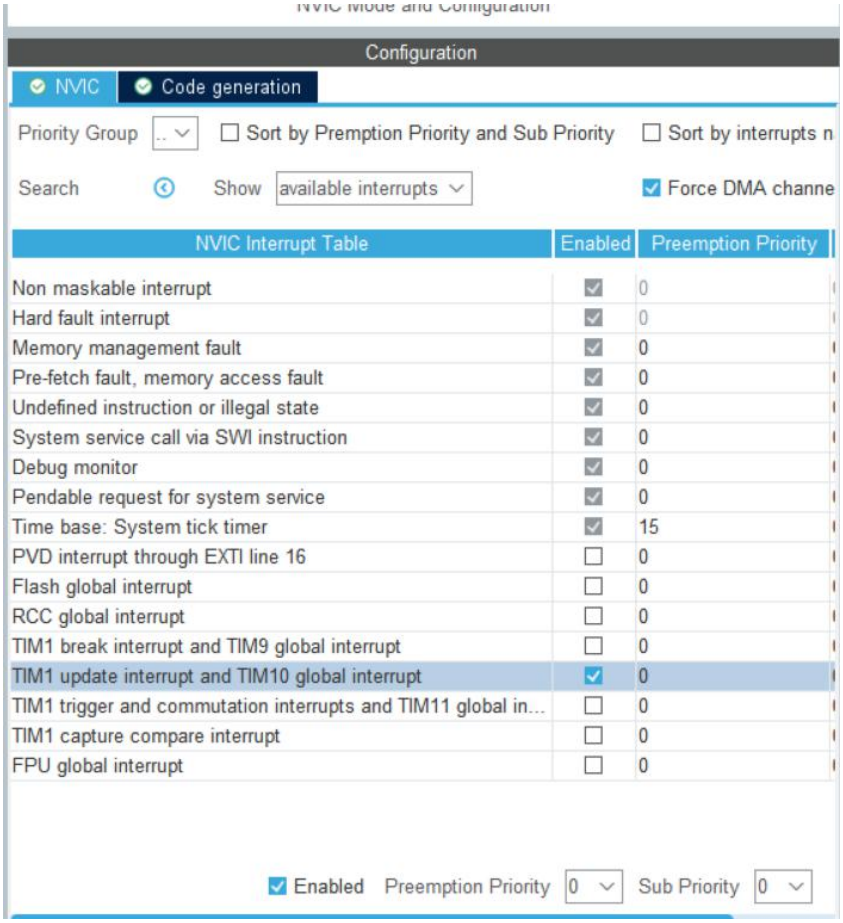
2、在左侧的标签页中选择 Timer，点击标签页下的 TIM1;

3、在弹出的 TIM1 Mode and Configuration 中，在 ClockSouce 的右侧下拉菜单中选中 Internal Clock;

4、接下来需要配置 TIM1 的运转周期。需要打开 Clock Configuration;
通过查阅数据手册资料，可以知道 TIM1 的时钟源来自 APB2 总线;
APB2 Timer clocks（MHz）为 180MHz，这意味着提供给 TIM1 预分频寄存器的频率就是 180MHz;
500ms 对应的频率为 2Hz，为了得到 2Hz 的频率，可以将分频值设为 18000-1，重载值设为 5000-1，则可以计算出定时器触发频率为

$$\frac{180000000HZ}{(18000-1+1)*(5000-1+1)}=2HZ$$

5、在 cubeMX 的 NVIC 标签页下可以看到当前系统中的中断配置
使能中断则在 Enable 一栏打勾，这里选中 TIM1 update interrupt，打勾，开启该中断。
这里为定时器 1 的中断保持默认的 0，0 优先级。



6、定时器回调函数介绍
HAL_TIM_PeriodElapsedCallback 函数

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

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

通过配置 TIM1 的分频值和重载值，使得 TIM1 的中断以 500ms 的周期被触发。因此中断回调函数也是以 500ms 为周期被调用。

bsp_led_toggle 函数翻转 LED 引脚电平

7、 HAL_TIM_Base_Start 函数 仅让定时器以定时功能工作

HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)

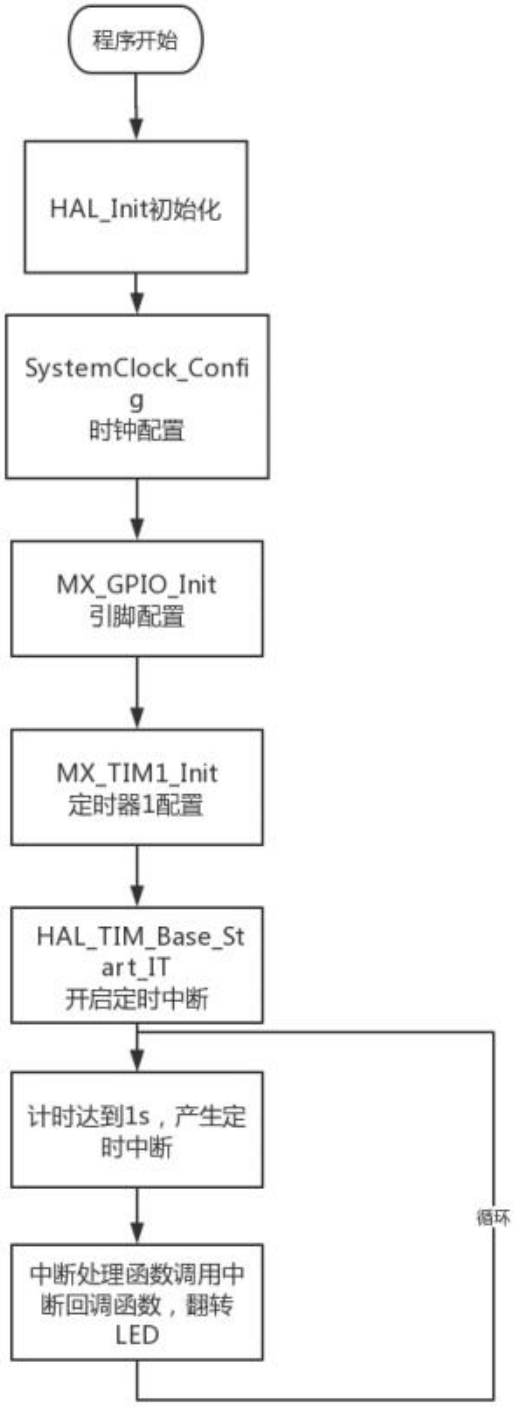
函数名	HAL_TIM_Base_Start
函数作用	使对应的定时器开始工作
返回值	HAL_StatusTypeDef，HAL 库定义的几种状态，如果成功使定时器开始工作，则返回 HAL_OK
参数	*htim 定时器的句柄指针，如定时器 1 就输入&htim1，定时器 2 就输入&htim2

8、 HAL_TIM_Base_Start_IT 函数 使用定时中断

HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)

函数名	HAL_TIM_Base_Start_IT
函数作用	使对应的定时器开始工作，并使能其定时中断
返回值	HAL_StatusTypeDef，HAL 库定义的几种状态，如果成功使定时器开始工作，则返回 HAL_OK
参数	*htim 定时器的句柄指针，如定时器 1 就输入&htim1，定时器 2 就输入&htim2

9、 程序流程:

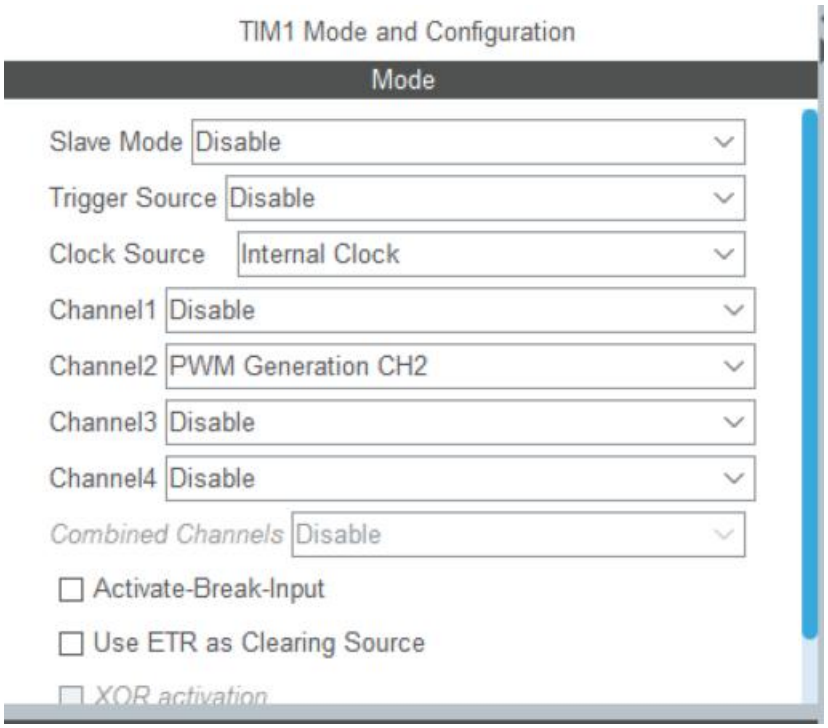

程序开始
HAL_Init初始化
SystemClock_Config
时钟配置
MX_GPIO_Init
引脚配置
MX_TIM1_Init
定时器1配置
HAL_TIM_Base_Start_IT
开启定时中断
计时达到1s，产生定时中断
中断处理函数调用中断回调函数，翻转LED
循环

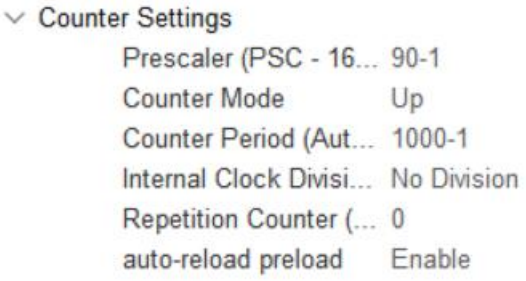
第五章PWM 控制 LED 亮度

1、 PWM 在 cubeMX 中配置

在 cubeMX 中设置定时器 1 的通道 2 为 PWM 输出。可以注意到三个通道对应的引脚正是 LED_R 引脚;

定时器 1 如下配置，设置重载值为 1000-1。



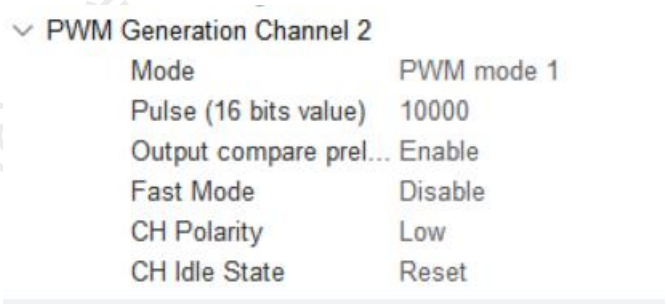


Prescaler (PSC - 16 bits value)　　分频系数 90-1

Counter Period (AutoReload Register - 16 bits value) 重装载值 1000-1

Internal Clock Division　No Division　　系统不分频

Auto-reload preload Enable　　自动重装载 使能



Mode：PWM 模式设置，我们选择 PWM1 模式
Pulse：占空比设置
Output compare preload：通道输出，使能
Fast Mode：快速模式， 不使能
CH Polarity：输出极性 ，低电平有效

2、 HAL_TIM_PWM_Start 函数

HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)

函数名	**HAL_TIM_PWM_Start**
函数作用	**使对应定时器的对应通道开始 PWM 输出**
返回值	**HAL_StatusTypeDef，HAL 库定义的几种状态，如果成功使定时器开始工作，则返回 HAL_OK**
参数 1	**htim 定时器的句柄指针，如定时器 1 就输入&htim1，定时器 2 就输入&htim2**
参数 2	**Channel 定时器 PWM 输出的通道，比如通道 1 为 TIM_CHANNEL1**

3、 程序流程：

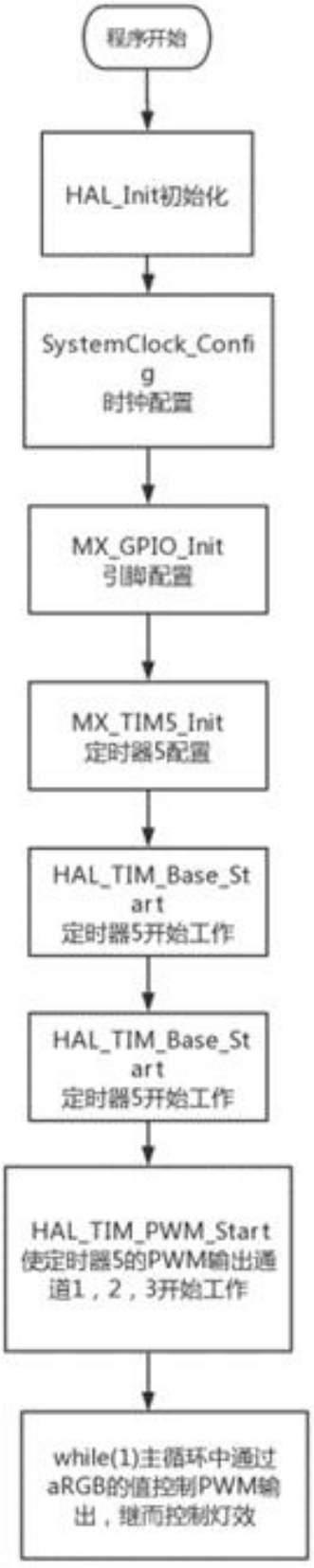
呼吸灯程序关键：

```
int i;
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);  //打开 PWM 通道



for(i=0;i<1000;i+=2)
    {
        TIM3->CCR2 = i;     //改变占空比，逐渐增加低电平的时间
        HAL_Delay(1);
    }
    for(i=1000;i>0;i-=2)
    {
        TIM3->CCR2 = i;     //改变占空比，逐渐降低低电平的时间
        HAL_Delay(1);
}
```

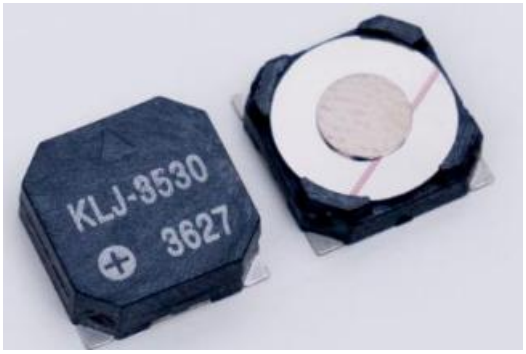
4、 呼吸灯程序流程:


程序开始
HAL_Init初始化
SystemClock_Config
时钟配置
MX_GPIO_Init
引脚配置
MX_TIM5_Init
定时器5配置
HAL_TIM_Base_Start
定时器5开始工作
HAL_TIM_Base_Start
定时器5开始工作
HAL_TIM_PWM_Start
使定时器5的PWM输出通道1，2，3开始工作
while(1)主循环中通过aRGB的值控制PWM输出，继而控制灯效

第六章PWM 蜂鸣器

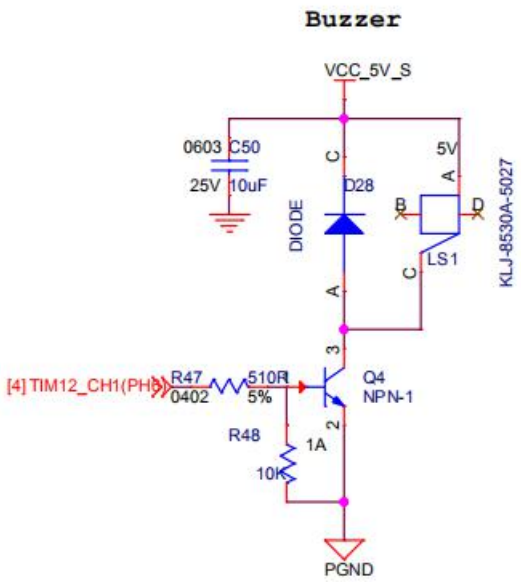
1、 蜂鸣器是一种能够通过电子信号控制的发声器件。

	有源蜂鸣器	无源蜂鸣器
内置震荡源	有	无
激励方式	直流电压	特定频率方波
音调	固定	可变



2、 蜂鸣器的 PWM 在 cubeMX 中配置。

蜂鸣器使用的引脚为 PH6，为定时器 12 的通道 1。



打开 cubeMX，使能定时器 12，预分配设置为 0，重载值设置为 20999，设置通道 1 为 PWM 输出，其余设置保持默认即可，此时开发板的 PH6 引脚变为绿色。

定时器 12 挂载在 APB1 总线上，对应的总线频率为 90MHz，分频值为 0，重载值为 22500-1，并通过公式计算得到 PWM 波的输出频率为 4000Hz。

3、 蜂鸣器程序说明。

改变 PWM 的频率就可以改变无源蜂鸣器的音调。故而改变定时器的分频系数和重载值，改变 PWM 的频率，就能够控制无源蜂鸣器发出的响声频率。

在主程序中，声明了 psc 和 pwm 两个变量，分别控制定时器 12 的分频系数和重载值，每一次循环中这两个变量进行一次自加。通过宏定义的方式，设置 pwm 的值在 MIN_BUZZER_PWM（10000）和 MAX_BUZZER_PWM（20000）之间变动，psc 的值在 0 和 MAX_PSC（1000）之间变动。

4、 主函数关键代码：

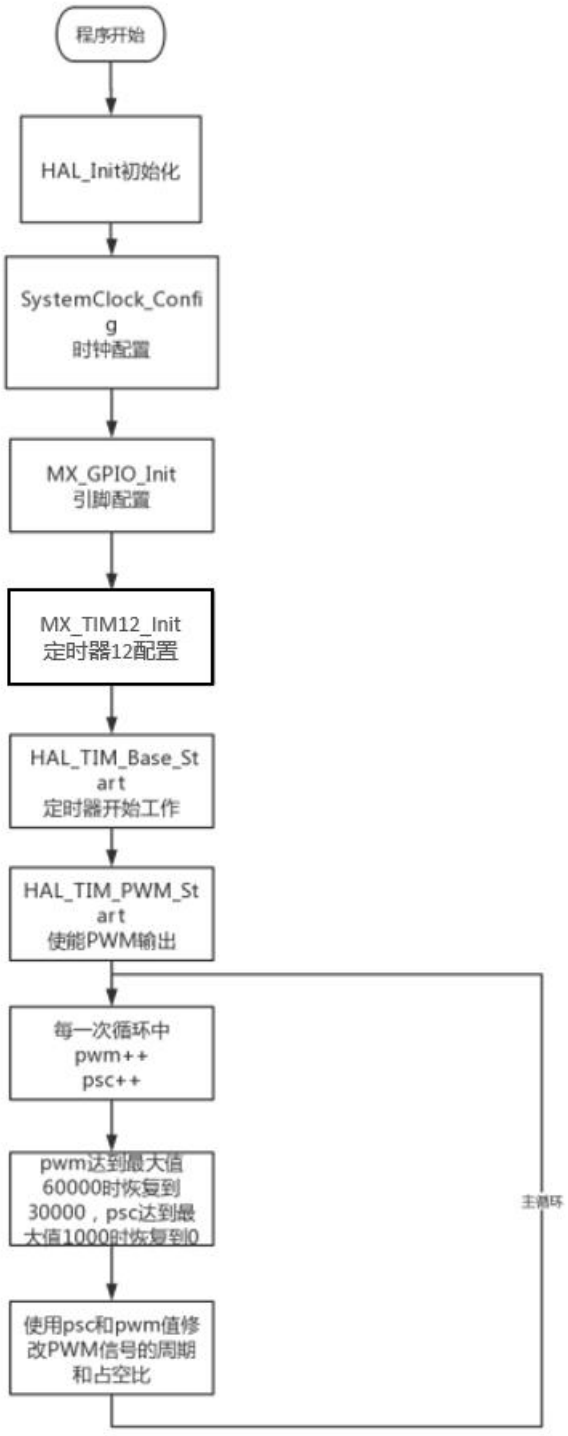
```
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
pwm++;
psc++;
if(pwm > MAX_BUZZER_PWM)
{
pwm = MIN_BUZZER_PWM;
}
if(psc > MAX_PSC)
{
psc = 0;
}
buzzer_on(psc, pwm);
HAL_Delay(1);
}
```

关键函数：

```
void buzzer_on(uint16_t psc, uint16_t pwm)
{
    __HAL_TIM_PRESCALER(&htim12, psc);
    __HAL_TIM_SetCompare(&htim12, TIM_CHANNEL_1, pwm);

}
void buzzer_off(void)
{
    __HAL_TIM_SetCompare(&htim12, TIM_CHANNEL_1, 0);
}
```

5、 程序流程：

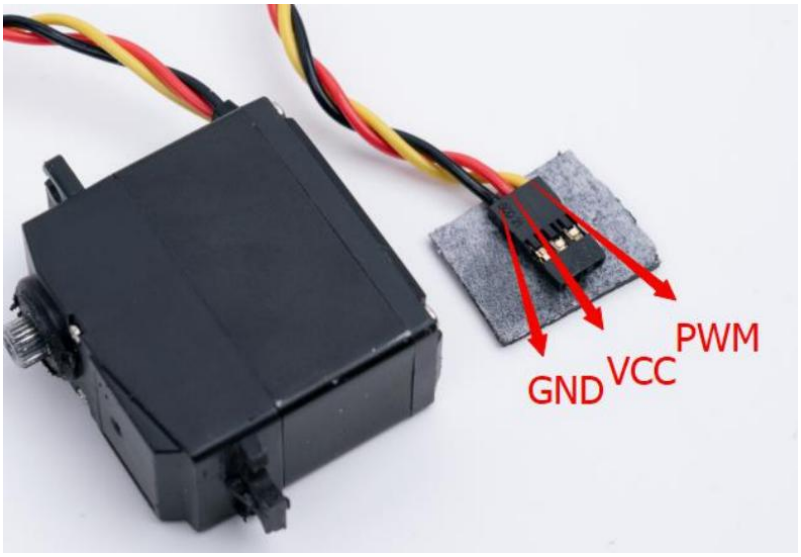

程序开始
HAL_Init初始化
SystemClock_Config
时钟配置
MX_GPIO_Init
引脚配置
MX_TIM12_Init
定时器12配置
HAL_TIM_Base_Start
定时器开始工作
HAL_TIM_PWM_Start
使能PWM输出
每一次循环中
pwm++
psc++
pwm达到最大值60000时恢复到30000，psc达到最大值1000时恢复到0
使用psc和pwm值修改PWM信号的周期和占空比
主循环

第七章PWM 舵机

1、 舵机介绍。

舵机是机器人中的常见的执行部件，通常使用特定频率的 PWM 进行控制。

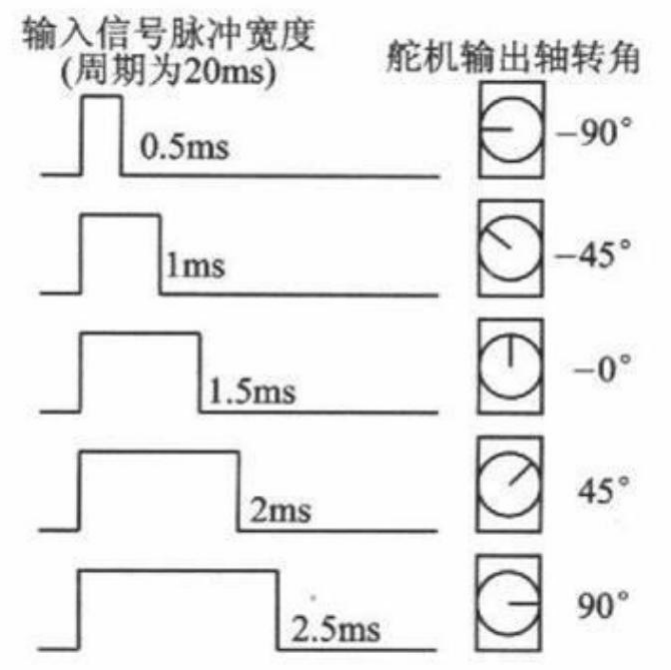
舵机的主要组成部位由一个小型的电机和传动机构（齿轮组）构成，多被用于操控飞行器上的舵面，故而得名舵机。由于控制简单，价格便宜，在 RoboMaster 比赛中，用于简单的动作控制，例如使用舵机控制弹仓盖的开合。



通常舵机的三根线按照颜色分别为：黑色-GND，红色-VCC，黄色-PWM 信号。在使用舵机 时，只需要使用杜邦线或者其他连接线接入对应的 PWM 接口。

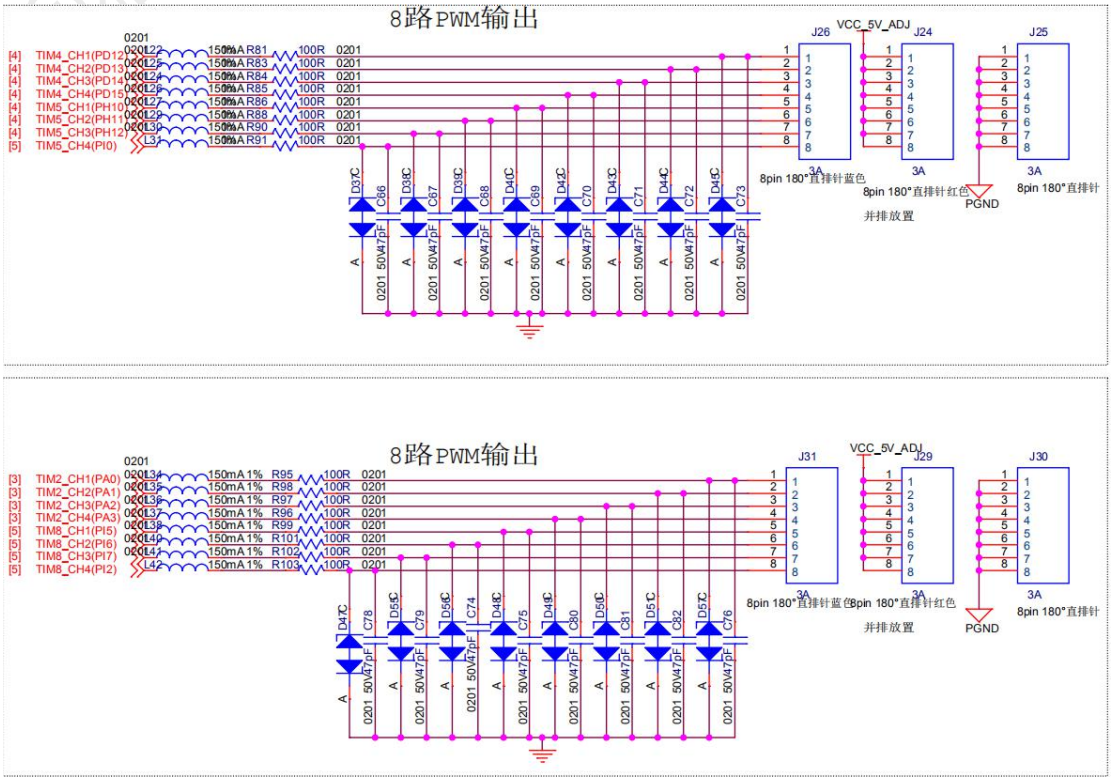
2、 舵机控制。

舵机使用的 PWM 信号一般为频率 50Hz，高电平时间 0.5ms-2.5ms 的 PWM 信号，不同占空比的 PWM 信号对应舵机转动的角度，以 180 度舵机为例，对应角度图如下图所示。



3、 舵机的 PWM 输出接口选择。

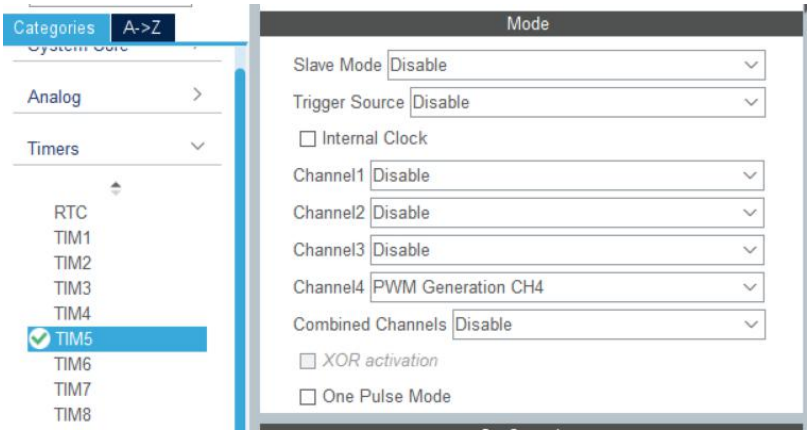
在开发板上具有 16 路 PWM 的输出接口，原理图如图所示：

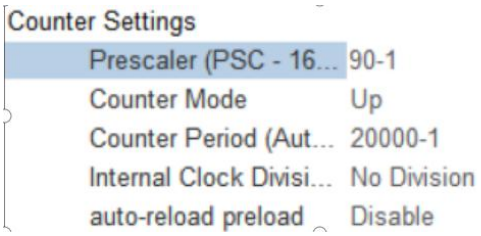


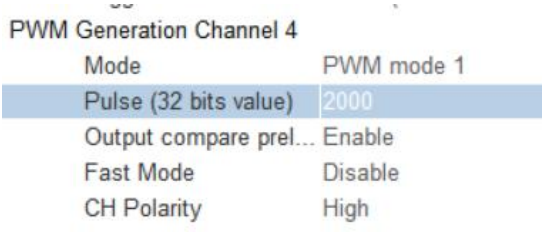
我们选 PI0（定时器 5 通道 4）。

4、 舵机的 PWM 在 cubeMX 中配置。

首先开启定时器 5，预分频值设置为 90-1，重载值设置为 20000-1。打开 通道 4 的 PWM 输出，在 PWM 通道的设置中，将 Pulse 值设置为 2000，比较寄存器的初始值就会被设成 2000。







可以通过查看源代码或者数据手册的方式我们知道定时器 5 挂载在 APB1 总线上，对 应的总线频率为 90MHz，定时器分频值为 90-1，重载值 20000-1，并通过公式计算得到 PWM 波的输出频率为 50Hz，对应的周期为 20ms。

通过 PWM 章节部分学习的知识，计算出 PWM 占空比最小为 500/20000 即 2.5%，对应高 电平时间为 20ms 乘以 2.5%等于 0.5ms，最大为 2000/20000 即 10%，对应高电平时间为 20ms 乘以 10%等于 2ms。

5、 舵机主程序讲解。

初始化：HAL_TIM_Base_Start 函数启动定时器 5

HAL_TIM_PWM_Start 函数将定时器 5 的 4 号通道的 PWM 输出开启。

主循环：通过__HAL_TIM_SetCompare 来设置 PWM 的占空比

```
int cnt=500;
for(int i=1;i<=5;i++)
{
  __HAL_TIM_SetCompare(&htim5,TIM_CHANNEL_4,cnt*i);
  HAL_Delay(2000);
}
```

由于实物舵机由 90 度变到-90 度消耗时间较长故延时 2 秒。

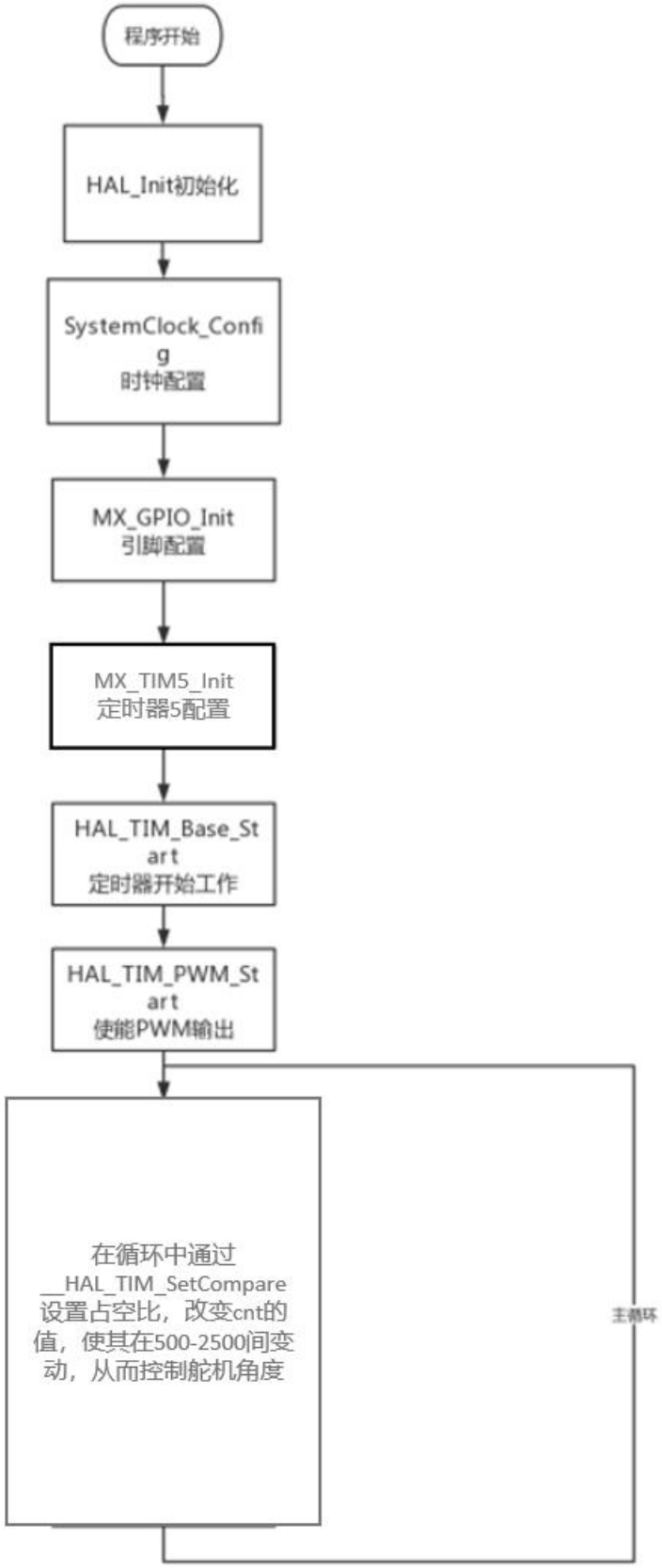
```
#define __HAL_TIM_SetCompare __HAL_TIM_SET_COMPARE
#define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__) \
(((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1 = (__COMPARE__)) :\
((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2 = (__COMPARE__)) :\
((__CHANNEL__)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

__HAL_TIM_SET_COMPARE 宏

__HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__)

参数 1	***htim 定时器的句柄指针，如定时器 1 就输入&htim1，定时器 2 就输入&htim2**
参数 2	**Channel 定时器 PWM 输出的通道，比如通道 1 为 TIM_CHANNEL_1**
参数 3	**需要赋值给比较寄存器的值，PWM 占空比等于比较值除以重载值**

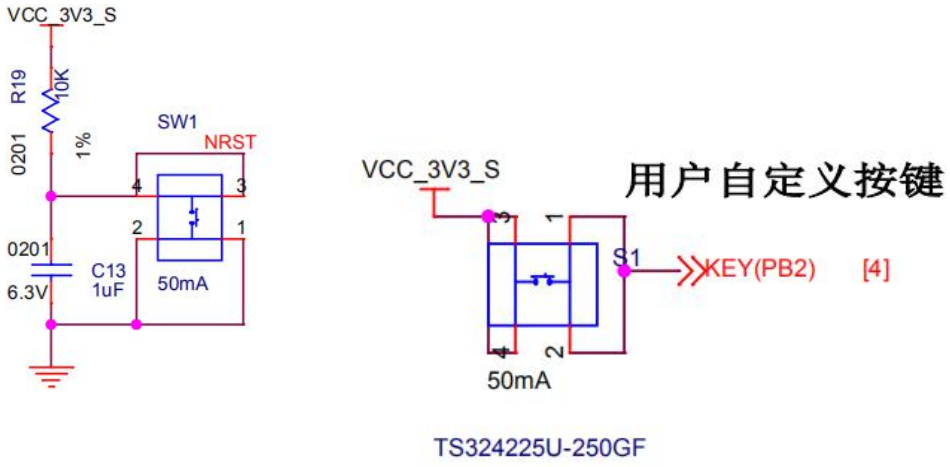
6、 程序流程：


程序开始
HAL_Init初始化
SystemClock_Config
时钟配置
MX_GPIO_Init
引脚配置
MX_TIM5_Init
定时器5配置
HAL_TIM_Base_Start
定时器开始工作
HAL_TIM_PWM_Start
使能PWM输出
在循环中通过__HAL_TIM_SetCompare设置占空比，改变cnt的值，使其在500-2500间变动，从而控制舵机角度
主循环

第八章按键的外部中断

1、 按键原理图介绍

开发板 A 型有两个按键，其中一个为复位按键，另一个为用户自定义按键(PB2)，如图所示。



2、 按键软件消抖

由于按键的机械结构具有弹性，按下时开关不会立刻接通，断开时也不会立刻断开，这就导致按键的输入信号在按下和断开时都会存在抖动，如果不先将抖动问题进行处理，则读取的按键信号可能会出现错误。一般采用软件消抖时，会进行 20ms 的延时，示波器采集按键波形如图所示。

3、 外部中断

外部中断通常是 GPIO 的电平跳变引起的中断。在 stm32 中，每一个 GPIO 都可以作为外部中断的触发源，外部中断一共有 16 条线，对应着 GPIO 的 0-15 引脚，每一条外部中断都可以与任意一组的对应引脚相连，但不能重复使用。例如外部中断 Line0 可以和

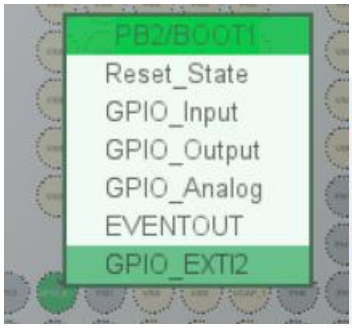
PA0，PB0，PC0 等任意一条 0 号引脚相连，但如果已经和 PA0 相连，就不能同时和 PB0，PC0 其他引脚相连。外部中断支持 GPIO 的三种电平跳变模式，如下所示：

上升沿中断：　当 GPIO 的电平从低电平跳变成高电平时，引发外部中断。
下降沿中断：　当 GPIO 的电平从高电平跳变成低电平时，引发外部中断。
上升沿和下降沿中断：当 GPIO 的电平从低电平跳变成高电平和从高电平跳变成低电平时，都能引发外部中断。

4、 外部中断在 cubeMX 中的配置

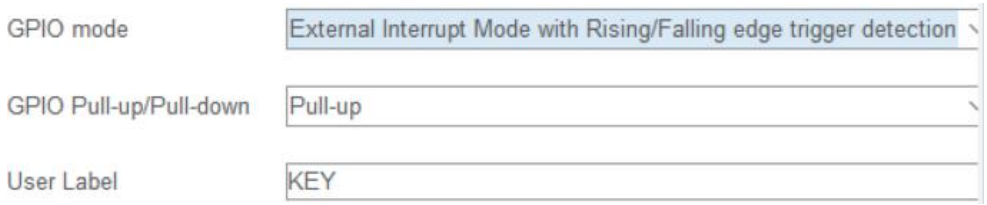
STM32 的 GPIO 提供外部中断功能，当 GPIO 检测到电压跳变时，就会发出中断触发信号给 STM32，使程序进入外部中断服务函数。

将 PB2 号引脚设置为按键的输入引脚，将其设置为外部中断模式。

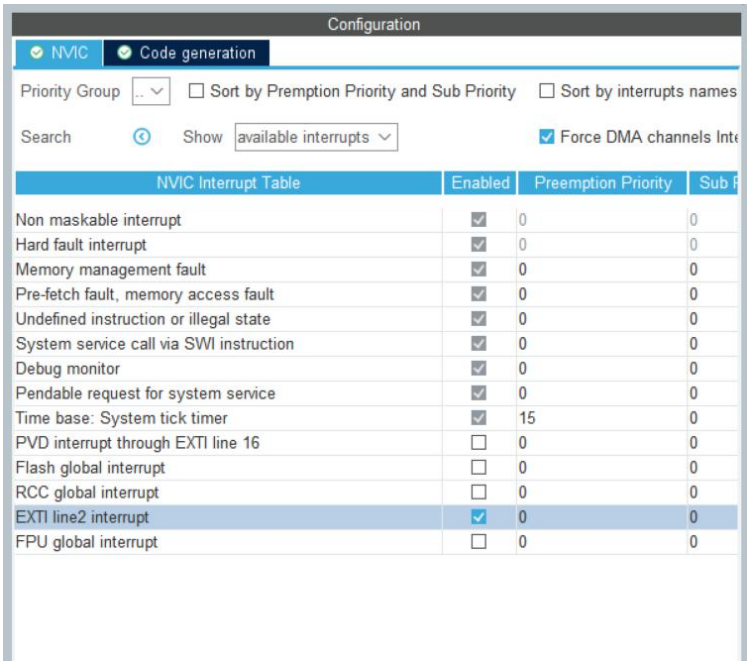


Pi...	Signal...	GPIO ...	GPIO ...	GPIO ...	Maxi...	User ...	Modified
PB2/...	n/a	n/a	Exter...	Pull-up	n/a	KEY	☑
PE11	n/a	Low	Outpu...	Pull-up	Low	LED_R	☑
PF14	n/a	Low	Outpu...	Pull-up	Low	LED_G	☑

接着点开 GPIO 标签页，对引脚进行如下设置，将 GPIO 模式设置为升降沿触发的外部中断，上下拉电阻设置为上拉电阻，最后设置用户标签为 KEY。



在 NVIC 标签页下，将外部中断开启。



5、 HAL_GPIO_ReadPin 函数介绍

GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

函数名	HAL_GPIO_ReadPin
函数作用	返回引脚电平
返回值	GPIO_PinState，如果是高电平则返回 GPIO_PIN_SET（对应为 1），如果是低电平则返回 GPIO_PIN_RESET（对应为 0）
参数 1：GPIOx	对应 GPIO 总线，其中 x 可以是 A…I。 例如 PH10，则输入 GPIOH
参数 2：GPIO_Pin	对应引脚数。可以是 0-15。 例如 PH10，则输入 GPIO_PIN_10

6、 中断回调函数介绍

每当产生外部中断时，程序首先会进入外部中断服务函数。在 stm32f4xx_it.c 中，可以找到函数 EXTI2_IRQHandler，它通过调用函数 HAL_GPIO_EXTI_IRQHandler 对中断类型进行判断，并对涉及中断的寄存器进行处理，在处理完成后，它将调用中断回调函数 HAL_GPIO_EXTI_Callback，在中断回调函数中编写在此次中断中需要执行的功能。

```
void EXTI2_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI2_IRQn 0 */

  /* USER CODE END EXTI2_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(KEY_Pin);
  /* USER CODE BEGIN EXTI2_IRQn 1 */

  /* USER CODE END EXTI2_IRQn 1 */
}
```

7、 程序中的前后台

在本次实验中，发现主循环和中断回调函数中都有代码。这是一个非常典型的以前后台模式组织的工程。什么是前后台模式呢？想象一下一个餐厅的运作模式，餐厅往往分为前台的叫餐员和后台的大厨，前台只有在来了客人，或者后台做好了一道菜时才会工作，而后厨则一直在忙着做菜，只有前台来了新的单子或者已经有菜做好了才会停下一会手中的活。

在单片机中，中断就是前台，而循环就是后台，中断只在中断源产生时才会进行相应的处理，而循环则一直保持工作，只有被中断打断时才会暂停。前后台程序的异同可以参见下表：

	前台程序	后台程序
运行方式	中断	循环
处理的任务类型	突发型任务	重复型任务
任务的特点	任务轻，要求响应及时	任务重，稳定执行

编写前后台程序时，需要注意尽量避免在前台程序中执行过长或者过于耗时的代码，让前台程序能够尽快执行完毕，以保证其能够实时响应突发的事件，比较繁杂和耗时的任务一般放在后台程序中处理。

前后台模式可以帮助我们提高单片机的时间利用率，从而组织起比较复杂的工程。

8、 主要程序：

前台程序：记录按键翻转的状态 rising_falling_flag

后台程序：执行处理工作，根据记录的翻转状态进行按键状态的判断

在主循环中，首先通过边沿检测标志 rising_falling_flag 来判断按键是处于按下还是松开的边沿。

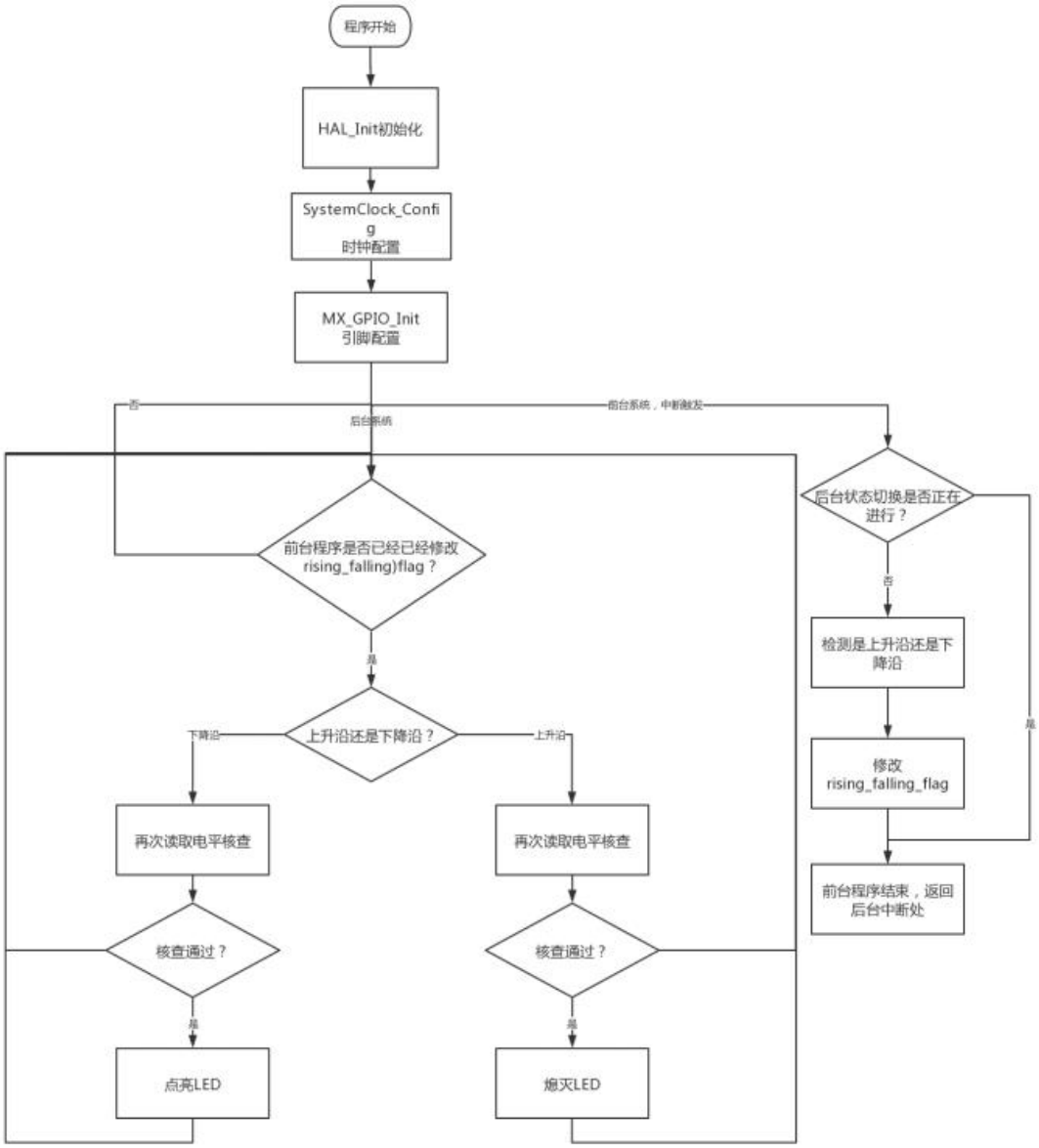
如果是下降的边沿（rising_falling_flag == GPIO_PIN_RESET）则将 LED 灯点亮，如果是如果是上升的边沿（rising_falling_flag == GPIO_PIN_SET）则将 LED 灯熄灭。

为了防止误触发，通过边沿检测的判断之后，程序还会再对电平进行一次读取，确认下降沿后跟随的是低电平或者上升沿后跟随的是高电平，如果不是则不切换 LED 状态。

在中断回调函数中，利用 HAL_GPIO_ReadPin 对 rising_falling_flag 进行赋值，从而判断触发中断的是上升沿还是下降沿。

使用 exit_flag 来实现主循环和中断回调函数之间的互斥，保证中断处理函数中的功能（判断上升/下降沿）只在主循环完成判断之后进行，或者主循环的判断只在中断处理函数运行（即检测到了一次上升沿或者下降沿）之后再进行。

9、 程序流程：



串口收发