

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



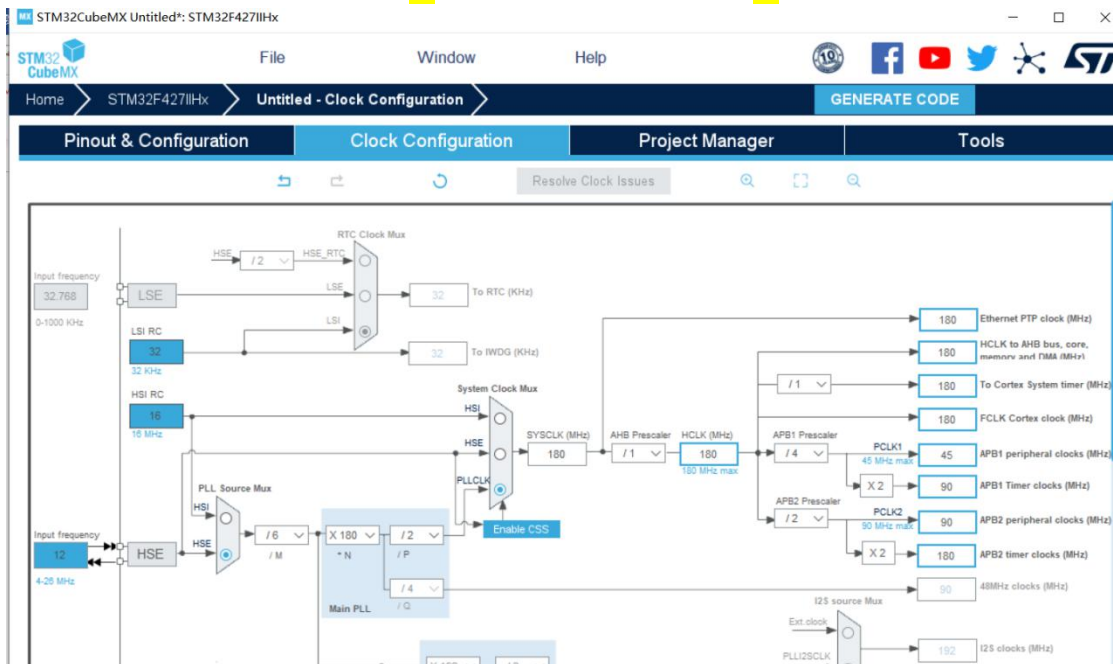
特别声明：

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

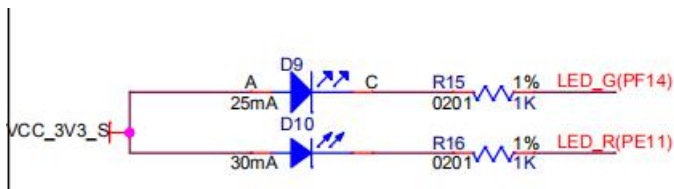
- 1、New Project->F427IIH6;
- 2、.在 System Core 下选择 RCC 选项, 在 RCC mode and Configuration 中的 High Speed Clock(HSE)下选择 Crystal/Ceramic Resonator;
- 3、顶部的 Clock Configuration, 进行主频配置; 将 Input frequency 设置为 12, 点击旁边的 HSE 圆形按钮, 配置/M 为/6, 配置*N 为 X180, 配置/P 为/2, 选择 PLLCLK 圆形按钮, 配置 APB1 Prescaler 为/4, 配置 APB2 Prescaler 为/2;



- 4、点击顶部的 Pinout & Configuration, 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();
            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:

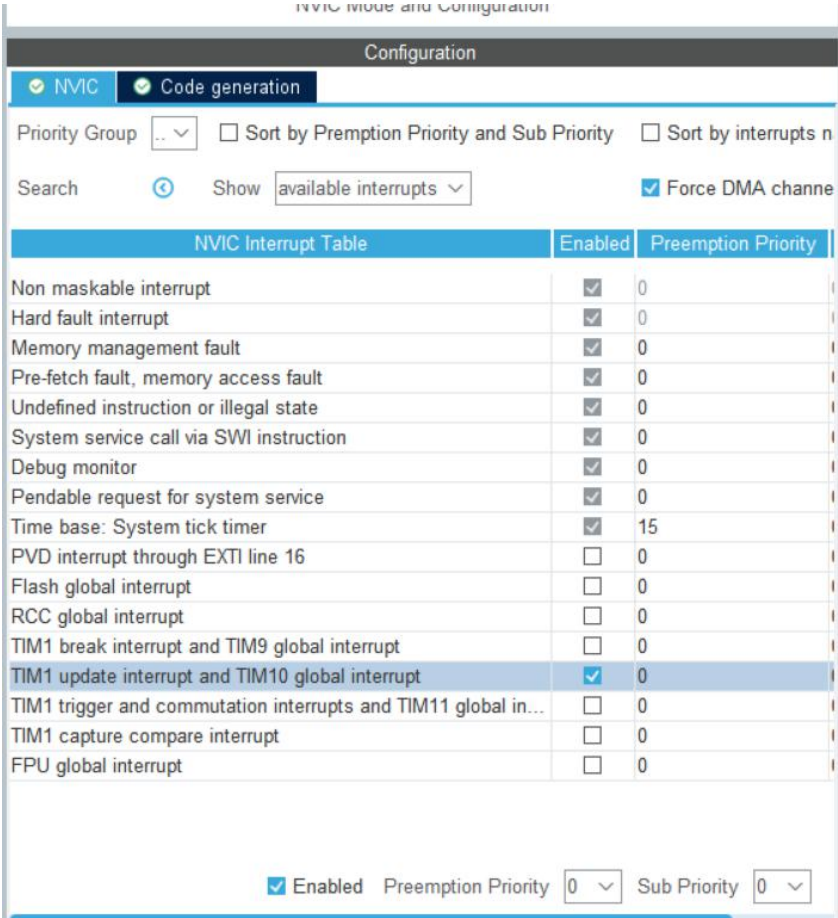
程序开始-》HAL_Init 初始化-》SystemClock_Config 时钟配置-》MX_GPIO_Init 引脚配置-》while(1)延时 500ms，翻转 LED 引脚的电平

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

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

- 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)

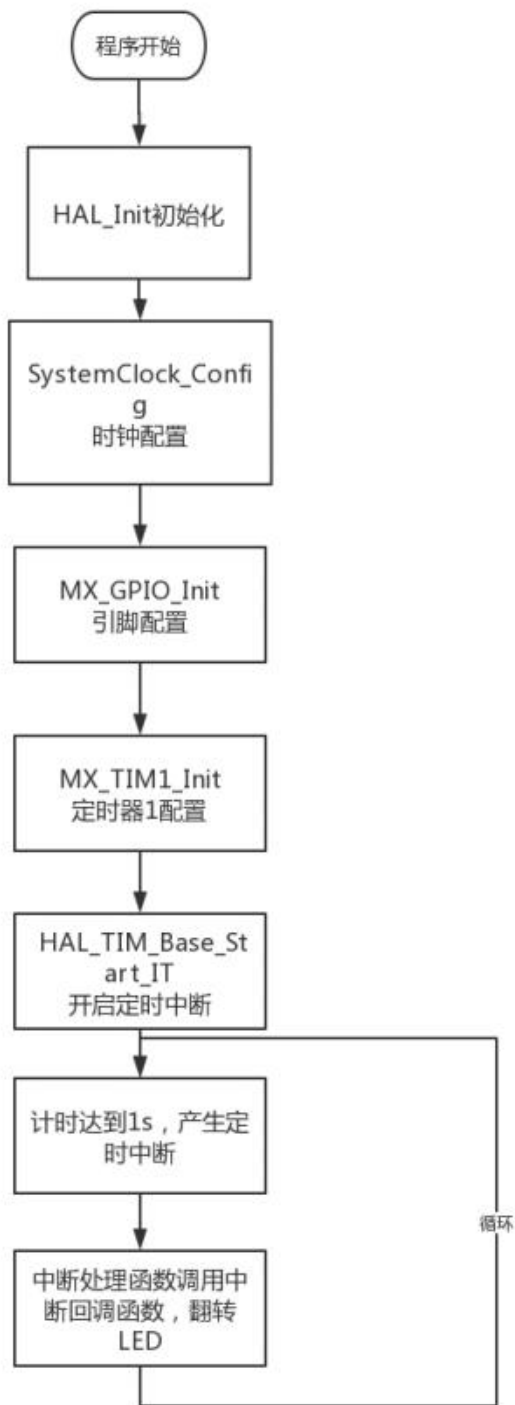
函数名	HAL_TIM_Base_Start
函数作用	使对应的定时器开始工作
返回值	HAL_StatusTypeDef, HAL 库定义的几种状态,如果成功使定时器开始工作,则返回 HAL_OK
参数	*htim 定时器的句柄指针,如定时器 1 就输入 &htim1,定时器 2 就输入&htim2

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:

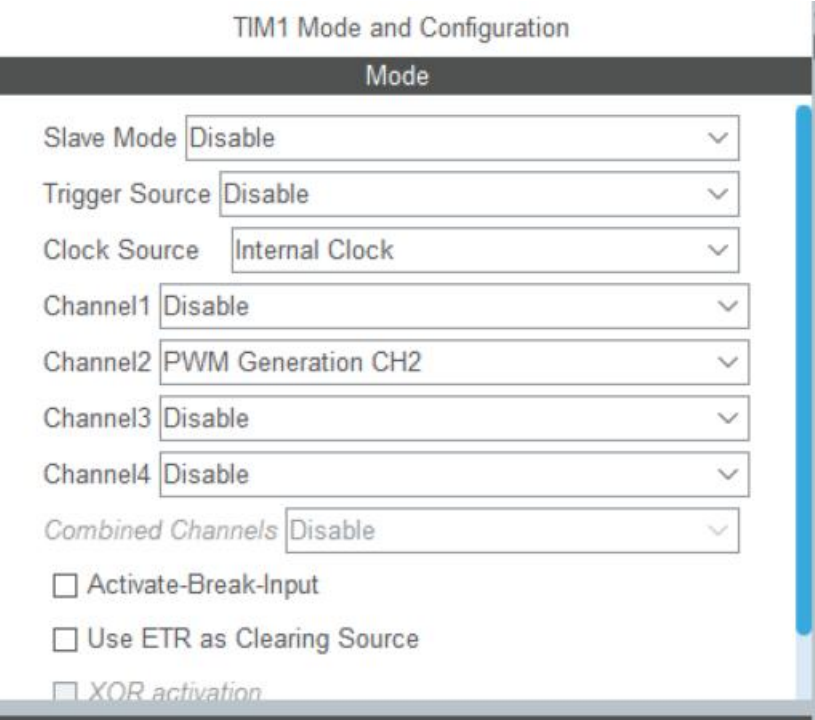


第五章PWM 控制 LED 亮度

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。



Counter Settings

Prescaler (PSC - 16... 90-1	
Counter Mode	Up
Counter Period (Aut... 1000-1	
Internal Clock Divisi...	No Division
Repetition Counter (... 0	
auto-reload preload	Enable

Prescaler	(PSC - 16 bits value)	分频系数 90-1
Counter Period	(AutoReload Register - 16 bits value)	重载值 1000-1
Internal Clock Division	No Division	系统不分频
Auto-reload preload	Enable	自动重载 使能

PWM Generation Channel 2

Mode	PWM mode 1
Pulse (16 bits value)	10000
Output compare prel...	Enable
Fast Mode	Disable
CH Polarity	Low
CH Idle State	Reset

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

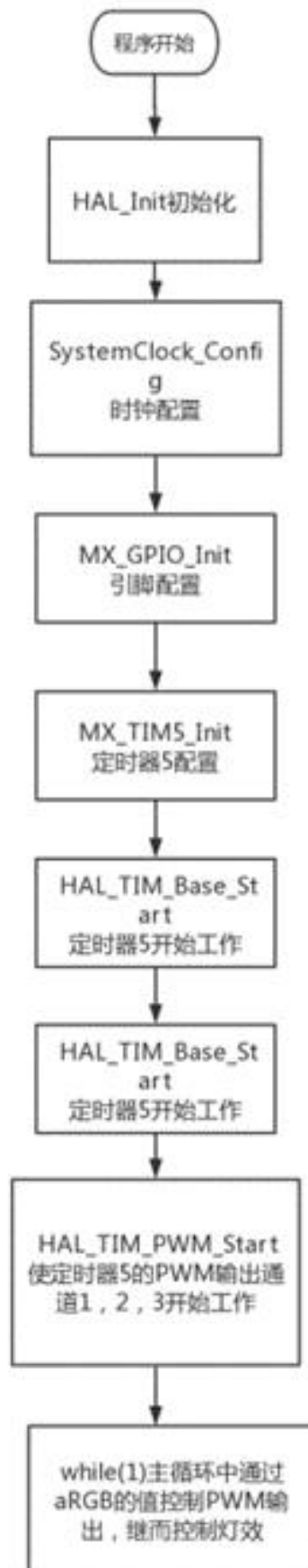
3、程序流程:

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);
}
```

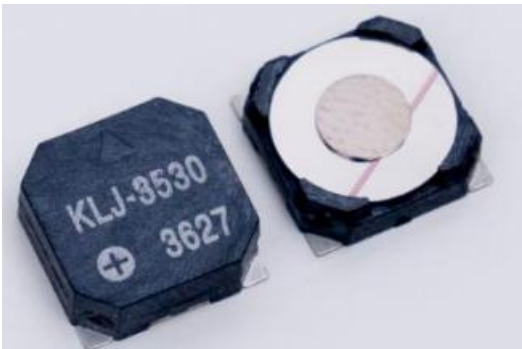
呼吸灯程序流程:



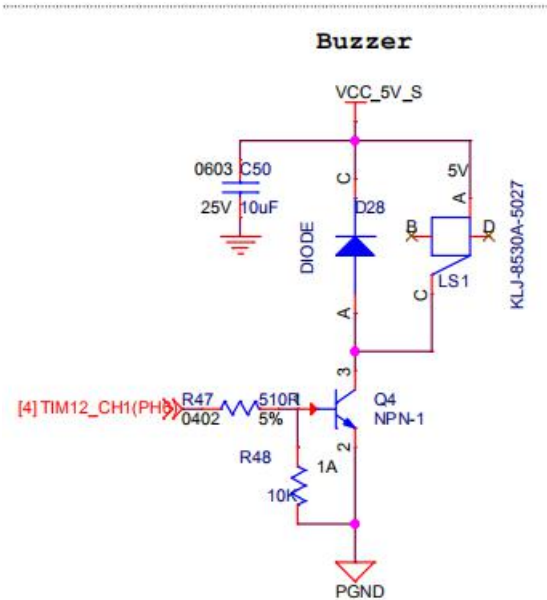
第六章PWM 蜂鸣器

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：

