TIM外设

STM32定时器

本节课重点

- 时基单元的计数原理以及其计数频率的计算方法

  --手搓微秒定时器

- TIM外设的更新中断

- 什么是PWM

  --点亮呼吸灯

认识STM32的时钟系统

RCC与时钟树

时钟在单片机中的作用

- **时钟是单片机运行的基础，时钟信号推动单片机内各个部分执行相应的指令。**时钟系统就是CPU的脉搏，决定CPU速率，像人的心跳一样。
- 与时序有关的数字电路都需要一个定时系统来支持。
- 每个外设都工作在某一频率的时钟之下

STM32的时钟系统

** stm32有四个时钟源：

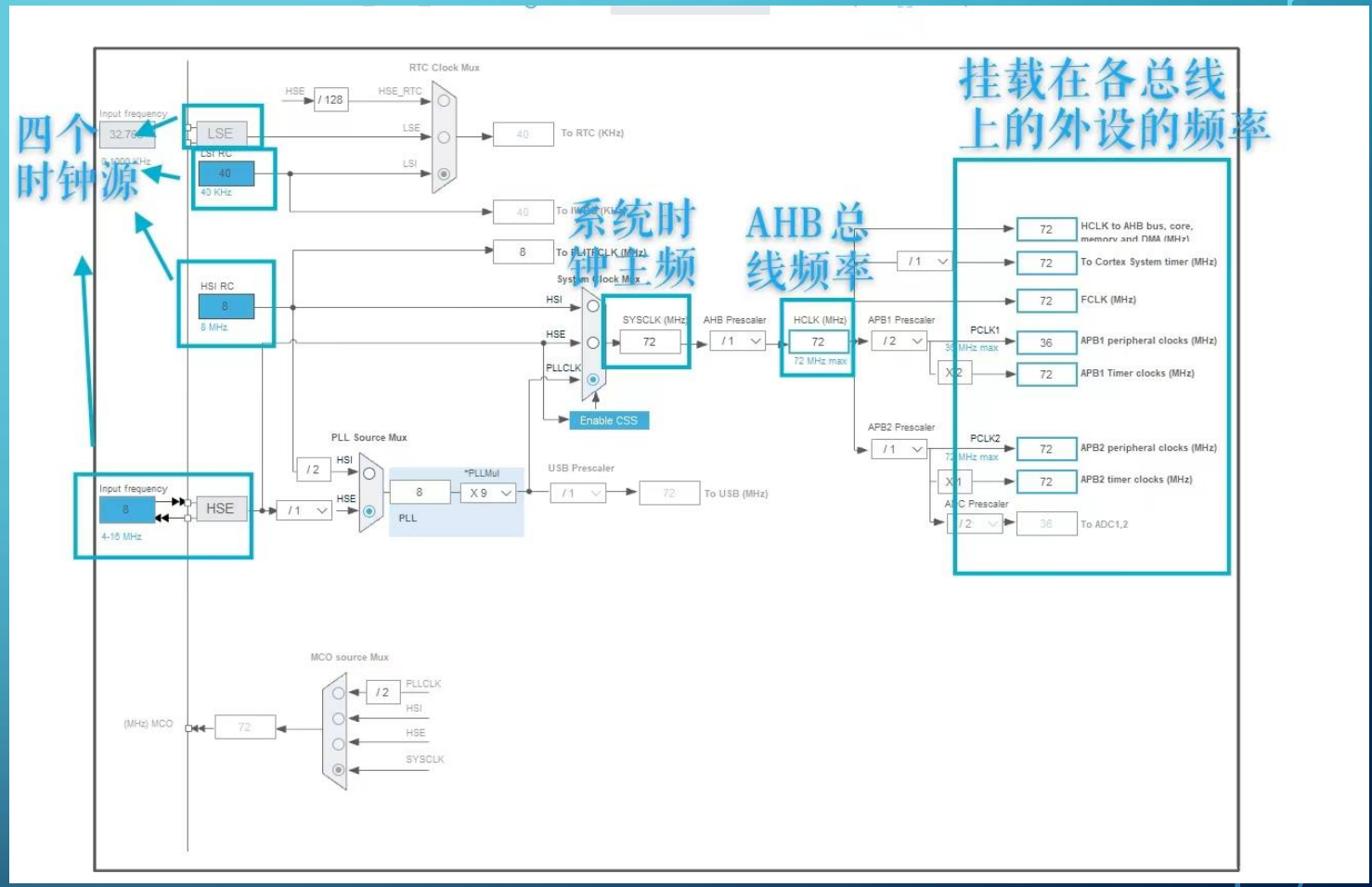
HSE： 高速外部时钟

LSE： 低速外部时钟

HSI： 高速内部时钟（RC）

LSI： 低速内部时钟（RC）

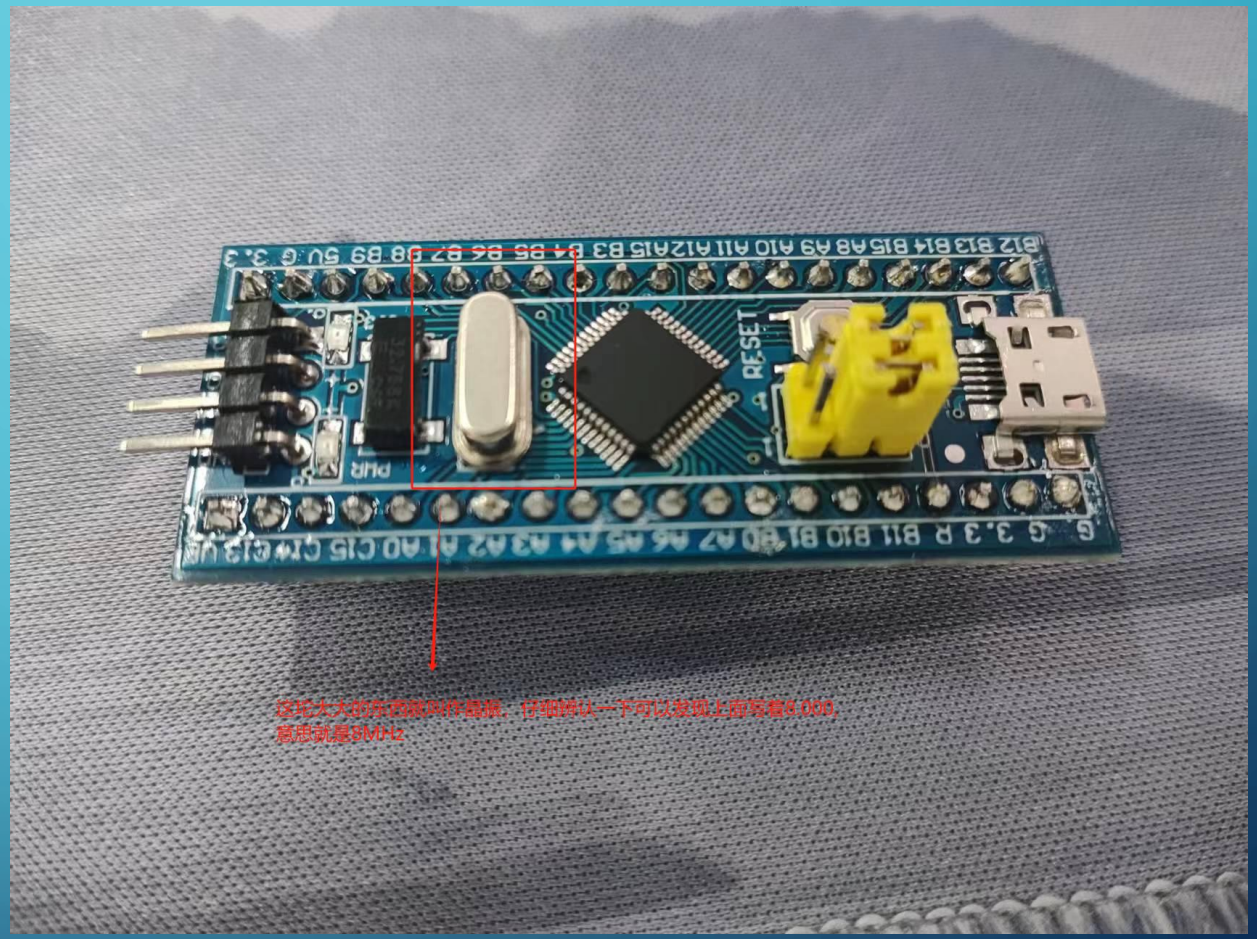
- 通过在时钟树中配置时钟源的选择和倍频分频系数即可配置系统时钟频率和外设频率
- （关键是要知道时钟树怎么配）



HSE(HIGH SPEED EXTERNAL CLOCK)

我们的主频接的时钟源就是高速外部时钟，它是板子上集成的石英晶振，配置时钟树时应该提前查好其对应的频率并填入相应位置。

HSE与HIS频率是一样的，但是外部时钟接的是石英晶振，比内部的RC震荡电路频率更为稳定，所以我们一般选择HSE



TIM外设

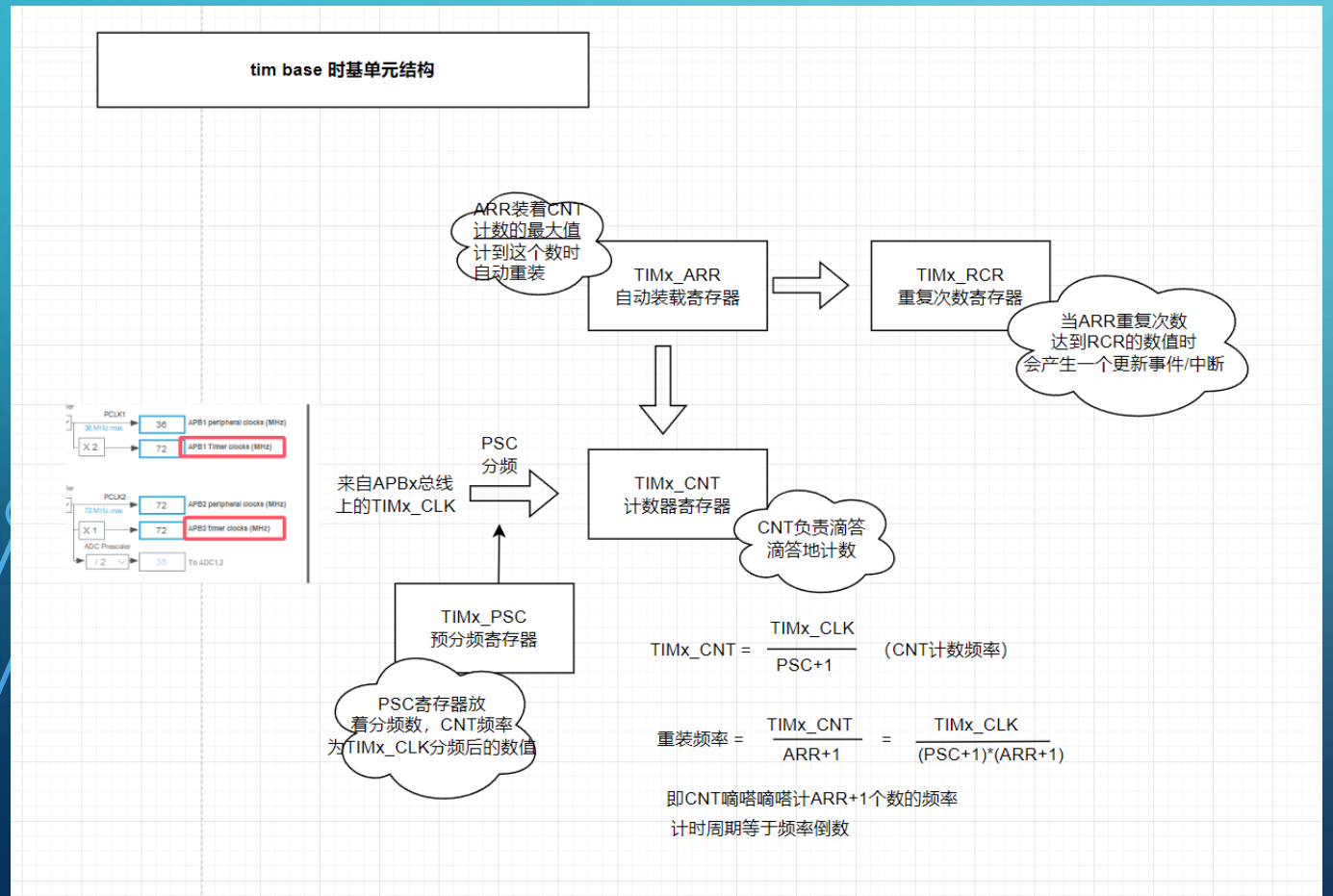
一个专门用来计数与定时的外设

什么是TIM

- TIM正如其名，是一个定时器。虽然把它和时钟树放在一起讲，但它们其实不是一个概念。时钟树控制着整个系统的时钟频率，而TIM是在时钟树下分出来的一个频率下计数的外设。定时器就是一个计数器。
- stm32的定时器分为基本定时器，通用定时器和高级定时器，c8t6有一个高级定时器和三个通用定时器

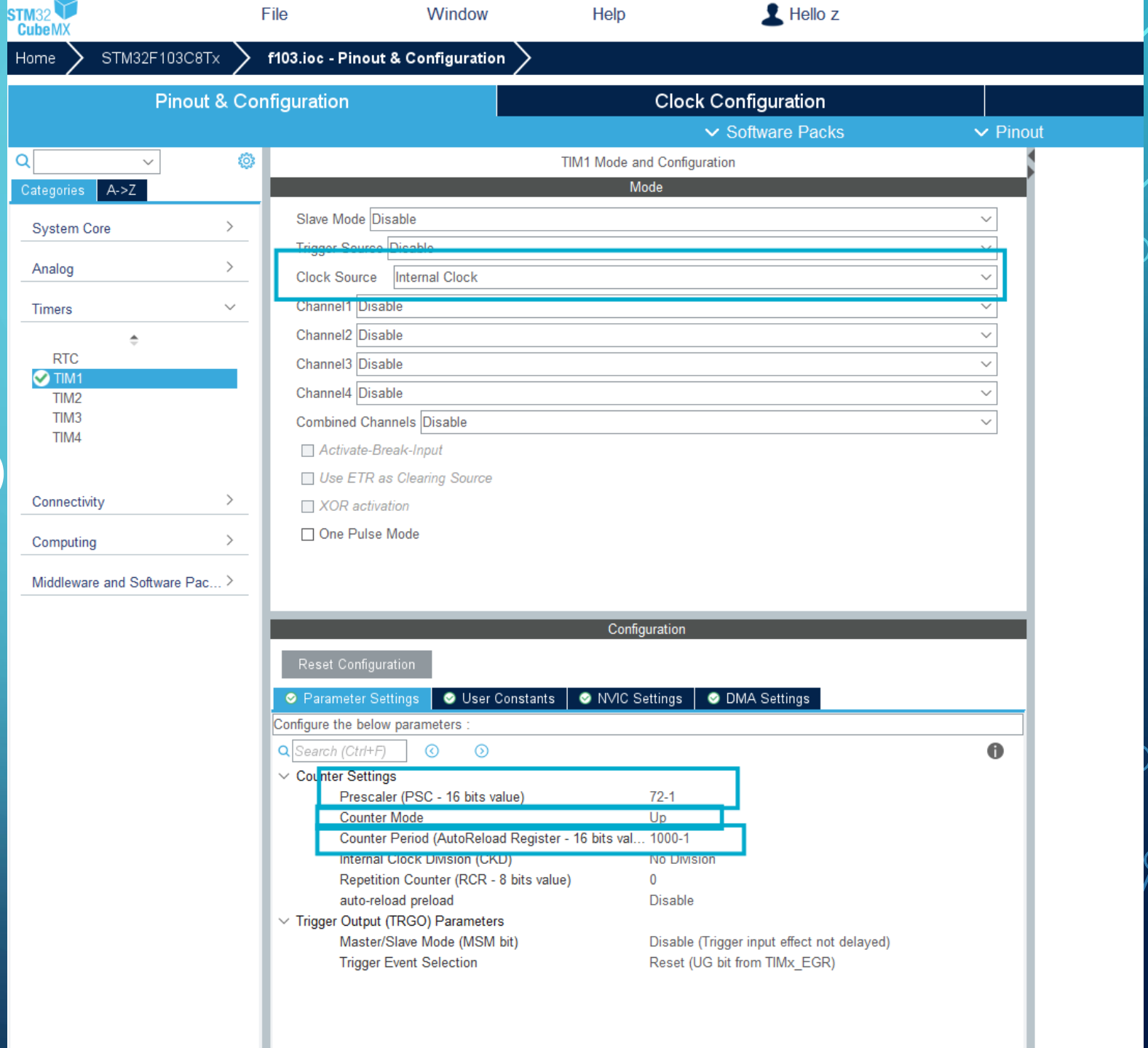
TIM的结构

- 定时器最核心最基本的结构是时基单元，基本定时器只有时基单元
- 时钟源输出的是方波脉冲，每一个经分频后的脉冲进来就使得CNT寄存器自增，CNT增满了（到达ARR中的数）就溢出，自动重装
- 注意这些寄存器都是16位的，范围0~65535



配置时基单元

- 设置脉冲来源
- 设置ARR（记得减一）
- 设置PSC（记得减一）
- 设置向上/向下计数



- 时基单元本身就可以实现所有计数功能，我们手搓定时器其实也只需要用到时基单元（不需要动用到外部引脚）
- 访问时基单元寄存器的函数/宏定义：
- 启动时基单元：

HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim);

设置ARR值：

__HAL_TIM_SetAutoreload(__HANDLE__, __AUTORELOAD__);

设置CNT：

__HAL_TIM_SetCounter(__HANDLE__, __COUNTER__)

读取CNT：

__HAL_TIM_GetCounter(__HANDLE__);

手搓一个微秒定时器

- 我们知道HAL_Delay()的单位是毫秒，也就是使用这个函数最短的延迟就是1ms，那么如果我想要延迟999us呢?
- 实现一个函数 void delay_us（uint16_t us）传入参数为延迟微秒数，传入几微秒则延迟几微秒。
- 如何使用上述函数/宏定义组合出定时器的逻辑呢?

--CNT每加一是过了多长时间?

--如何将定时器的计数时长转换为延迟时长?

（p.s.记得先打开时基单元）

动手试试叭

定时器的更新中断

- 定时器的中断和外部中断其实是同理的，只是触发中断的信号来源有所不同。
- 外部中断：指定的gpio引脚状态发生变化时触发中断
- 定时中断：时基单元计数溢出时发生中断（这种溢出中断也叫更新中断）

定时器的更新中断

tim base 时基单元结构

通用定时器的中断在这里触发

高级定时器的
中断在这里触发

ARR装着CNT
计数的最大值
计到这个数时
自动重装

TIMx_ARR
自动装载寄存器

TIMx_RCR
重复次数寄存器

当ARR重复次数
达到RCR的数值时
会产生一个更新事件/中断
即它标记了多少次重装
会发生一个更新事件/中断

来自APBx总线
上的TIMx_CLK

PSC
分频

TIMx_CNT
计数器寄存器

CNT负责滴答
滴答地计数

TIMx_PSC
预分频寄存器

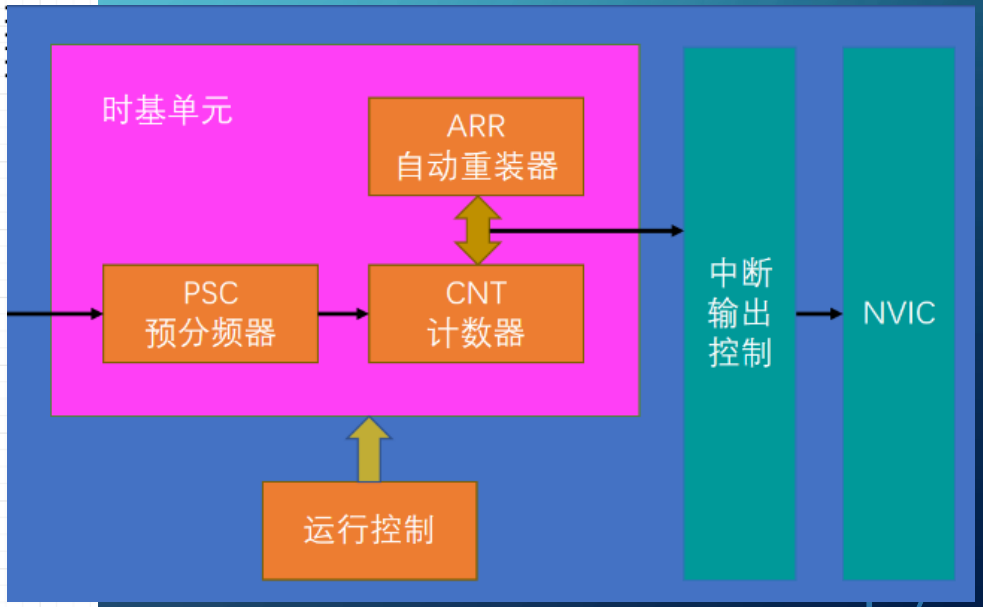
PSC寄存器放
着分频数，CNT频率
为TIMx_CLK分频后的数值

$$TIMx\_CNT = \frac{TIMx\_CLK}{PSC+1} \quad (CNT计数频率)$$

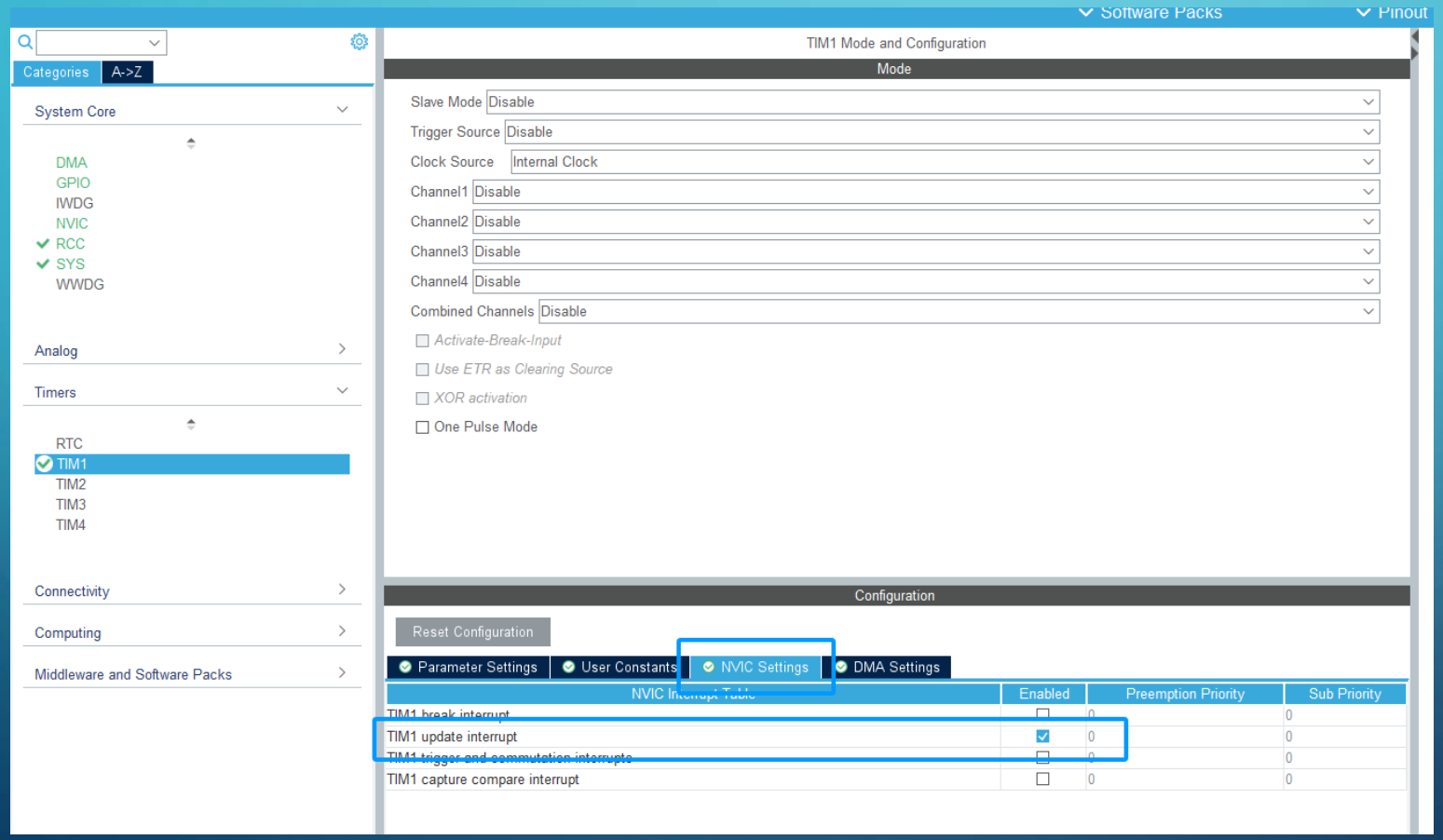
$$重装频率 = \frac{TIMx\_CNT}{ARR} = \frac{TIMx\_CLK}{(PSC+1)*(ARR+1)}$$

$$高级定时器\ 更新事件/中断频率 = \frac{重装频率}{RCR+1} = \frac{TIMx\_CLK}{(PSC+1)*(ARR+1)*(RCR+1)}$$

通用/基本定时器在ARR溢出时就会触发中断，因此没有RCR再分频
(只要一直把RCR配成0就不用想这么多啦！中断频率即等于重装频率)



使能更新中断


Software Packs
Pinout
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
RTC
TIM1
TIM2
TIM3
TIM4
Connectivity
Computing
Middleware and Software Packs
TIM1 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Disable
Activate-Break-Input
Use ETR as Clearing Source
XOR activation
One Pulse Mode
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
NVIC Interrupt Table
Enabled
Preemption Priority
Sub Priority
TIM1 break interrupt
TIM1 update interrupt
TIM1 trigger and commutation interrupts
TIM1 capture compare interrupt

关于更新中断的函数/宏定义

- 在需要中断的情况下用这句开启时基：

```
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim);
```

更新中断回调函数：

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
        if(htim->Instance == TIM1)  //判断是哪个定时器触发的中断
        {

        }
}
```

TIM的外围结构

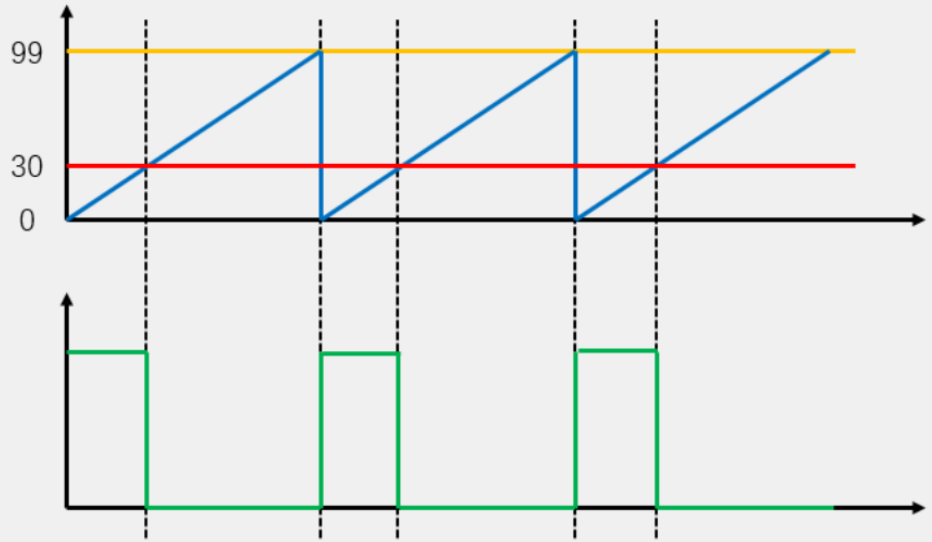
- 刚刚我们所用到的功能其实只有时基单元，但是通用&高级定时器还有非常复杂的外围结构，包括选择不同的输入源，以及四个独立的CHANNEL（可用于输入/输出）以及....(pdf里面有定时器的框图，在这里就不放了）....有非常多复杂的功能，今天我们主要要认识的是：

使用四个独立的通道输出PWM波

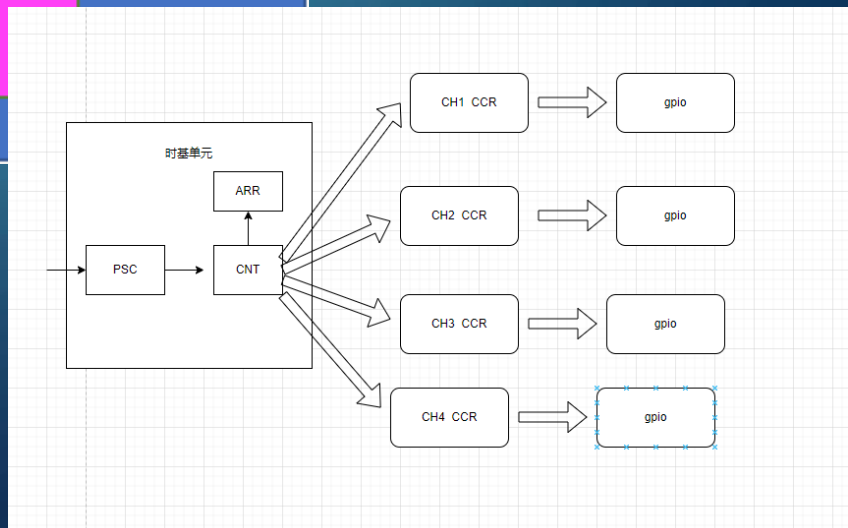
什么是PWM

- Pulse Width Modulation，即脉冲宽度调试，即可以调制脉冲的周期/占空比从而改变高&低电平的宽度/等效模拟电压的大小（因应用需求而异）
- 它可以通过调节占空比（高电平占整个脉冲长度的比例）改变数字信号控制中只有0/1的二极化状态，实现等效输出连续变化的模拟量（某种程度上可以当作一个DAC）

PWM基本结构

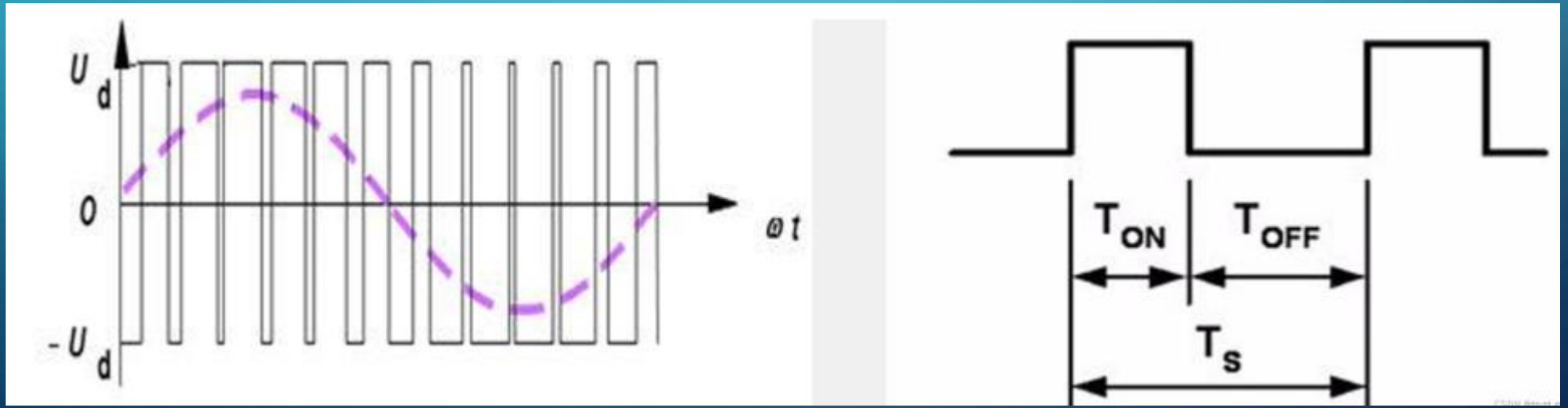

99
30
0

时基单元
ARR
自动重装器
PSC
预分频器
CNT
计数器
运行控制
CCR
捕获/比较器
CNT<CCR时，REF置有效电平
CNT≥CCR时，REF置无效电平
REF
极性选择
输出使能
GPIO
输出比较单元×4


时基单元
ARR
PSC
CNT
CH1 CCR
CH2 CCR
CH3 CCR
CH4 CCR
gpio
gpio
gpio
gpio

PWM的三个参数

- 频率/周期：pwm波电平一高一低为一周期，频率为周期倒数
- 占空比（duty): 在一个周期内高电平时长的比例
- 分辨率：占空比变化的快慢（细腻程度）



PWM频率与占空比的计算

- (CK_PSC就是TIMx_clk)

PWM频率： Freq = CK_PSC / (PSC + 1) / (ARR + 1)

PWM占空比： Duty = CCR / (ARR + 1)

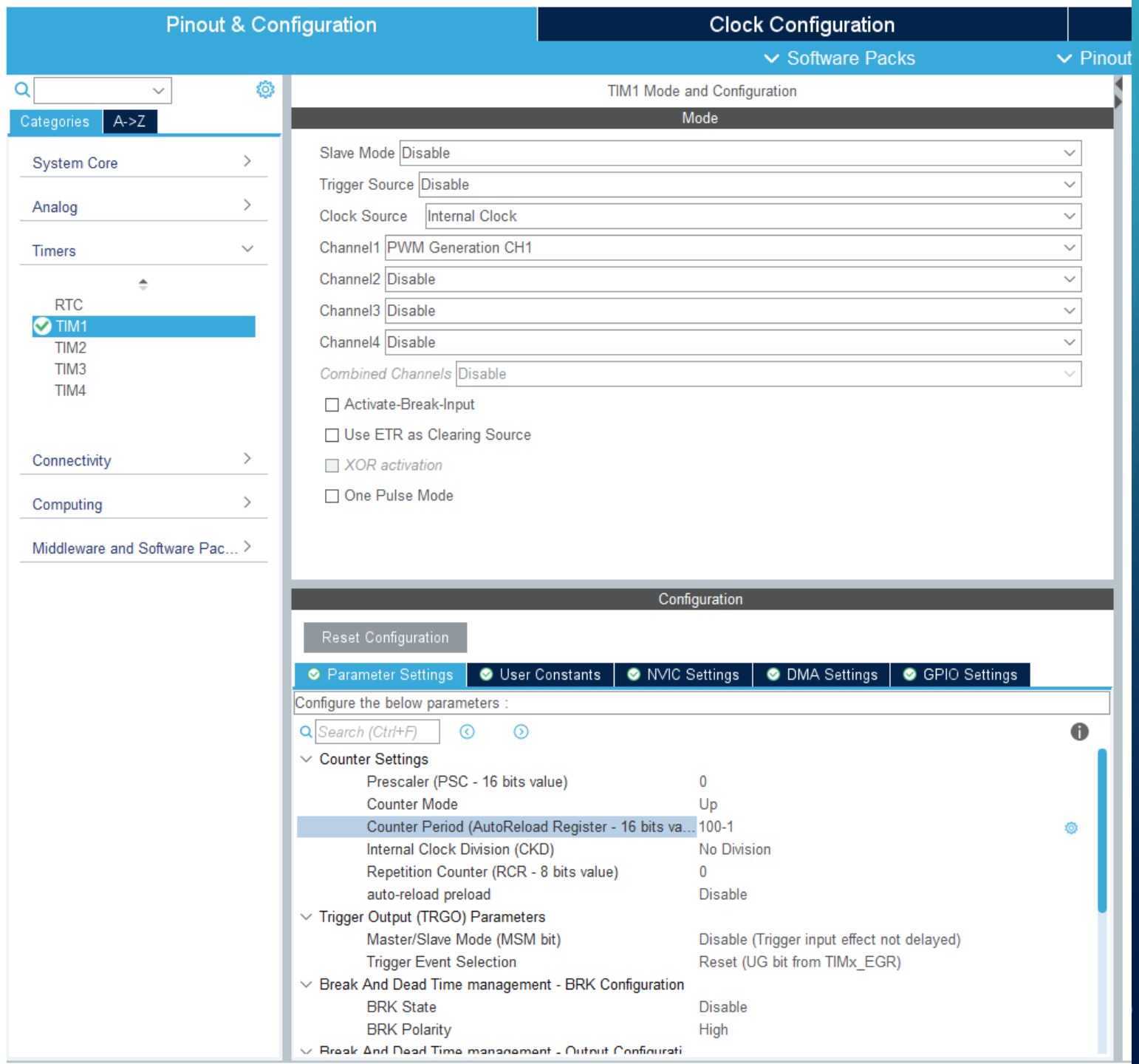
使用PWM点亮呼吸灯

- 什么是呼吸灯---呼吸灯是指灯的亮度柔和而连续地周期性亮灭，就像平缓地呼吸一样
- 当pwm的**频率足够高**时，其占空比的变化反应在人眼中就是模拟电压的高低（占空比高则亮度高，占空比低则亮度低）因此我们可以通过使占空比连续增长与连续降低实现让led”呼吸”的效果
- 注意我们点亮的led还是PC13，但是这个时候我们不需要去配置PC13本身，而是将TIM的CHx引脚生成的pwm直接引入PC13

配置PWM

- 挑一个通道，选择PWM Generation

*思考为什么要选择这个PSC和ARR?



函数/宏定义

- 启动PWM（启动PWM之前记得也要启动时基！）
  HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel) ;

设置CCR

__HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__);

修改ARR的宏定义同上

请大家思考一下如何实现用PWM点亮呼吸灯叭！

- 这次没有白给答案了噢 大家自己努力一下~

DK作业3

- 用PWM实现呼吸灯（4分）
- 手搓微秒定时器，并用它来使步进电机转得更快（具体速度不要求，只要体现出自己写的延迟函数并将它应用于步进的控制中即可，推荐使用多文件编程）（2分）
- 使用**定时更新中断**改变步进电机转向，要求每秒改变一次方向（4分）