1.3.2 时钟树&定时器

引子

时钟，恰如生活中的时钟，授时就是它最大的作用。

在单片机中，所有发生的操作均是按一定的频率进行的，因此需要固定频率的脉冲存在，产生这样的脉冲的设备叫做时钟。

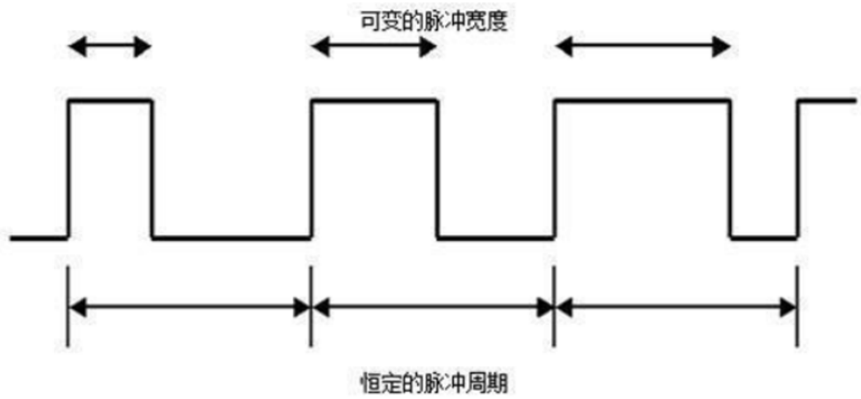
单片机中时钟共有四种，分别有两种分类方式：高速（high speed：HS）与低速（low speed：LS）、外部（external：E）与内部（internal：I），两者排列组合共有四种（HSE，HSI，LSE，LSI）。其中高速低速指的是产生脉冲的频率的高低；外部和内部指的是使用的时钟是内部时钟还是外部时钟，其中内部时钟一般为RC振荡电路组成，电阻和电容一般都容易受到温度的影响，因此这种时钟经常会产生温漂，在温度变化时会变得不准。而外部时钟一般为晶振，依靠晶体的振荡产生脉冲。

现在我们能够通过时钟获得一个固定频率的脉冲了。但是很多时候我们需要知道一些操作发生了多长时间，例如进行延时等操作时。

这时候就需要定时器。它就是一个用来计数的工具，计一定数量的脉冲，通过配置这些脉冲的数量，我们就能够获得一段记录的时间。

暂且不论定时器的底层原理。

我们先来看一种常用于控制的信号，称为PWM，即脉宽调制技术。



在数字电路中，我们的输出量只有高低电平两种，即1和0，这样两个分立的值并不能实现某些环境下需要的连续控制。

但是，如上图所示，如果我们能够固定脉冲周期不变，而改变高电平在一个脉冲周期中的占比，从0至100%变化，则可得到一个连续变化的量，这个比值称为占空比。

在一些环境下，例如我们想要控制LED灯的亮度或者蜂鸣器的响度时，改变占空比即可实现连续的调节。

很容易发现，在这个过程中，最重要的就是确定脉冲的周期以及一个周期中高电平存在的时长，即最重要的是确定频率及占空比，那么怎么确定？用定时器。

STM32F4中有三种定时器，分别为基本定时器、通用定时器以及高级定时器，除了基本定时器外均有PWM输出功能。

对于频率部分，为了满足板子上大部分外设的使用需要，一般会选取尽可能大的时钟频率，在C板上这一频率为168MHz，这一频率远远大于我们需要的频率，定时器上有两个寄存器来解决这个问题，分别为预分频寄存器和自动重装载寄存器，对应需要配置的预分频系数和自动重装载值。

生成的PWM的频率即为时钟频率除以 （二者分别加一 的乘积），即：

$$f_{out} = \frac{f_{clk}}{(K_{pre}+1)*(K_{Auto}+1)}$$

加一是因为两寄存器从零开始计数，总的值需要加一。

而占空比即为设置值与自动重装载值的比值。

预分频系数和自动重装载值如何理解呢?

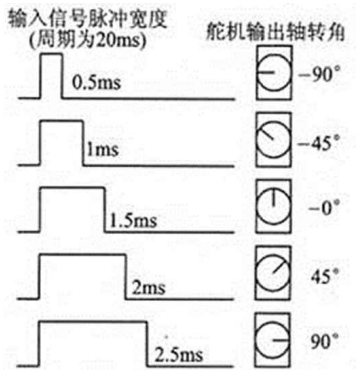
- 首先是预分频系数，可以认为经过预分频后的脉冲才是用于定时器的时钟信号。在目标频率确定的条件下，预分频系数确定了能够设置PWM的精度，预分频系数越小，则自动重装载值越大，最小单元所占比例越小。
- 其次是自动重装载值K_Auto，每一个周期所设置的时间是由该寄存器值确定的，它的作用在于当定时器的寄存器计数到自动重装载值后，定时器的寄存器会清零重新开始计数，即最终会产生一个周期为K_Auto个 上面预分频后脉冲的周期 的信号。

下面不妨先通过一个实例熟悉一下：

舵机是一个应用在很多车上的执行元件，例如步兵机器人上的弹仓盖、英雄、工程机器人的图传等等位置均是依靠舵机进行驱动的。舵机内部自带编码器，因此可以实现半闭环的控制，即我们只需要发送一定频率及占空比的PWM波，舵机本身可以闭环到目标位置或者目标速度。舵机一般分为可以连续朝一个方向转动的360度舵机以及有一定限位的非360度舵机，一般来说，连续转动的舵机通过PWM控制其旋转的速度，而非360度舵机一般通过PWM控制其所在位置。

我们一般使用的舵机为MG996R舵机，它是一种180度舵机，长这样：

如下图所示为它所需要的频率以及角度与占空比的对应关系：



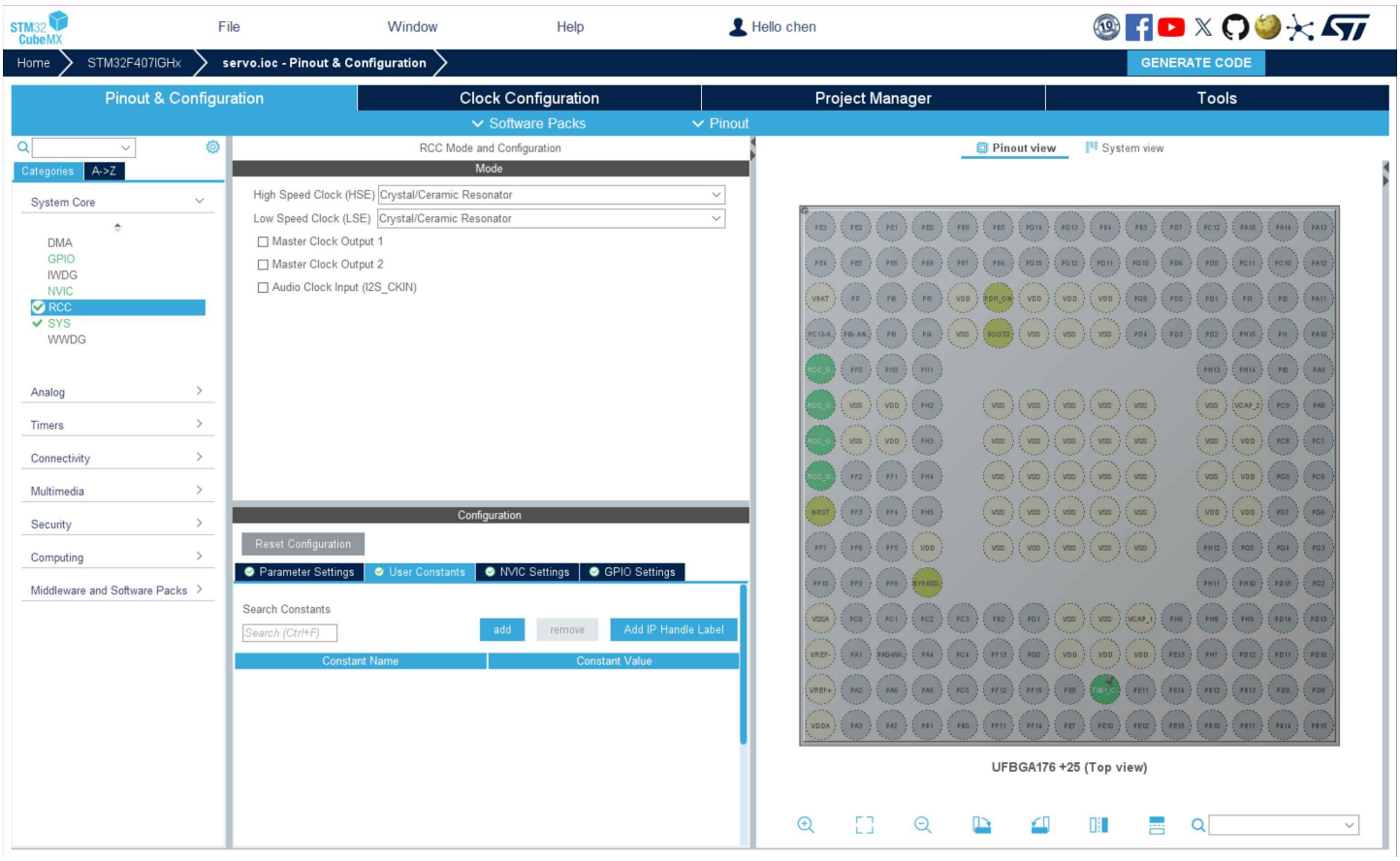
需要注意的是，此处只给定了几个特殊的角度，实际上由于PWM的连续控制性，中间角度也是可以取到的。

因此我们只需要达到最终的频率，并且选取合适的占空比即可实现最终的控制效果。

实践

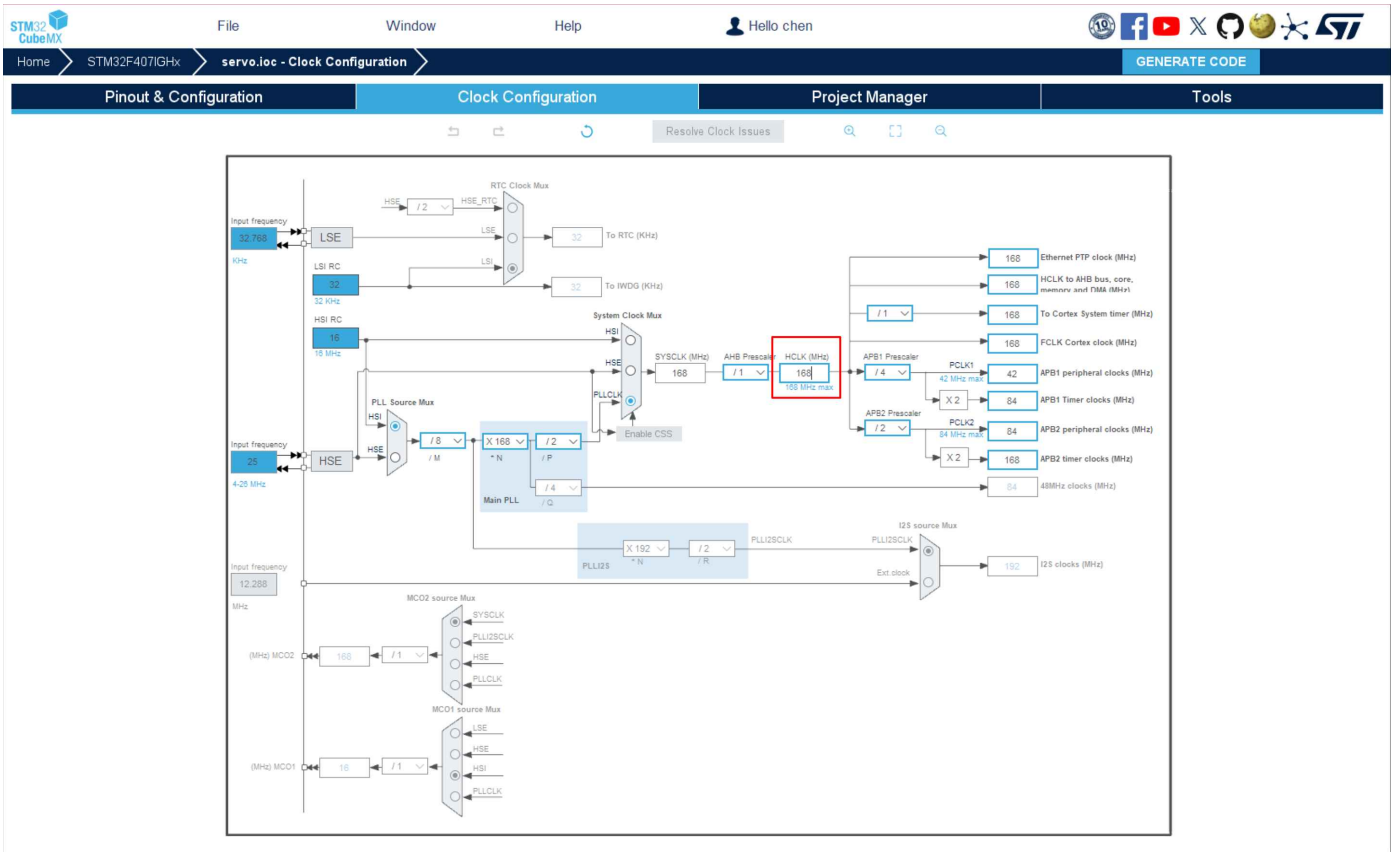
下面操作试一下：

1. 打开STM32CubeMX，新建一个工程，在左侧system core中的RCC下，选择高速时钟源和低速时钟源均为外部晶振。如下图所示：



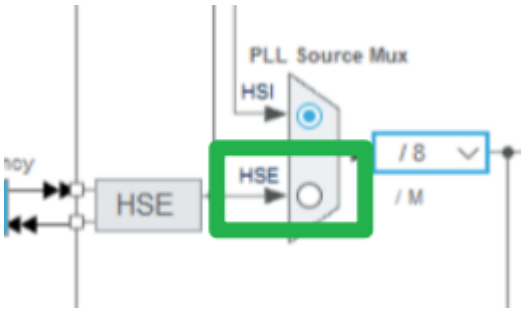
此时右侧芯片上会有四个引脚被点亮，它们就是对应的与晶振相连的引脚。

2. 点击上方选框至clock configuration，将HCLK的频率改为168MHz，即最大频率。回车，软件会自动更新后方的频率参数。

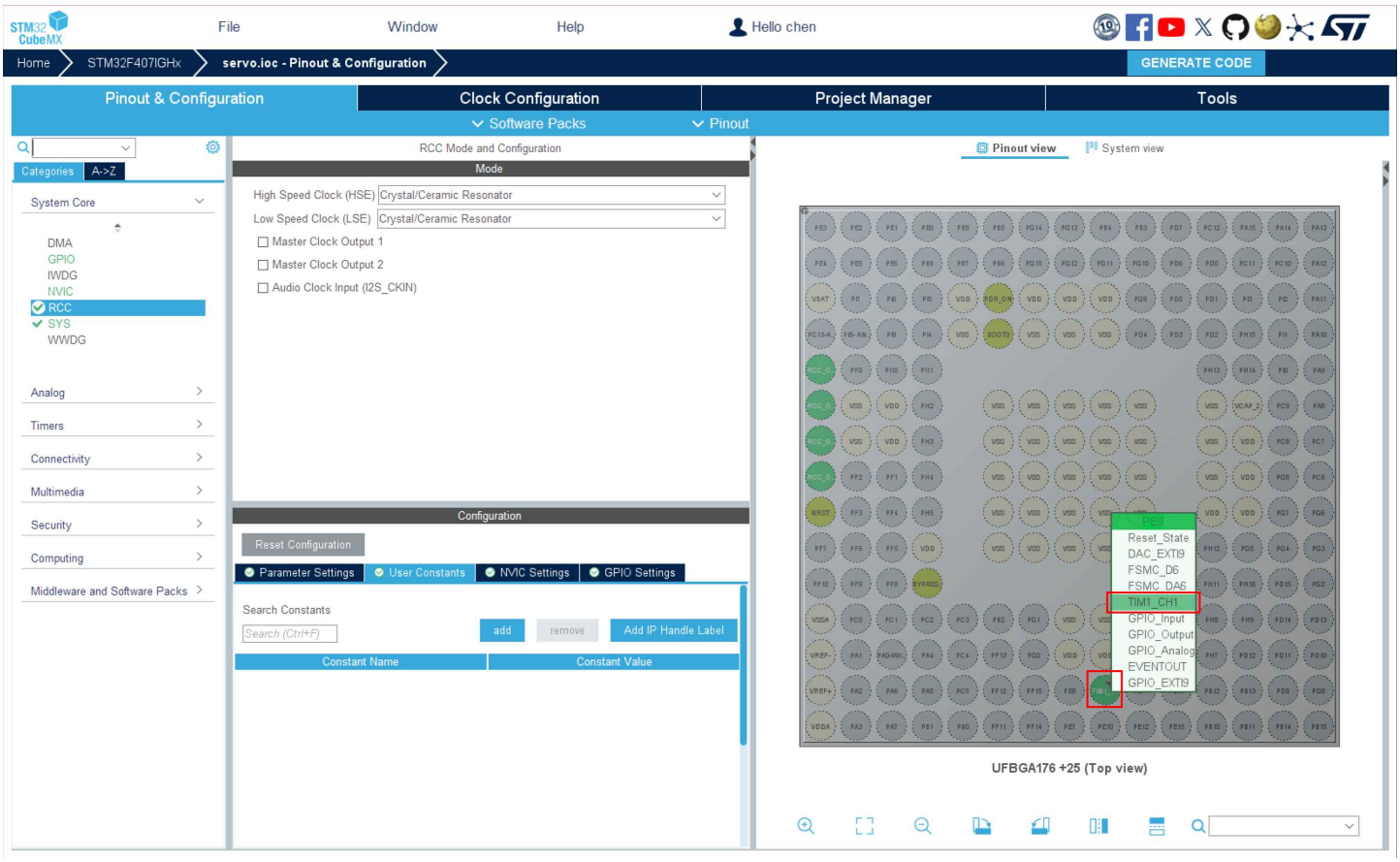


但此时默认使用内部高速时钟，为了更高的精度，可以选用高速外部时钟，此时需要更改外部高速时钟频率为12MHz（这是外部晶振的频率），并选择HSE，并保证后面的频率不变：

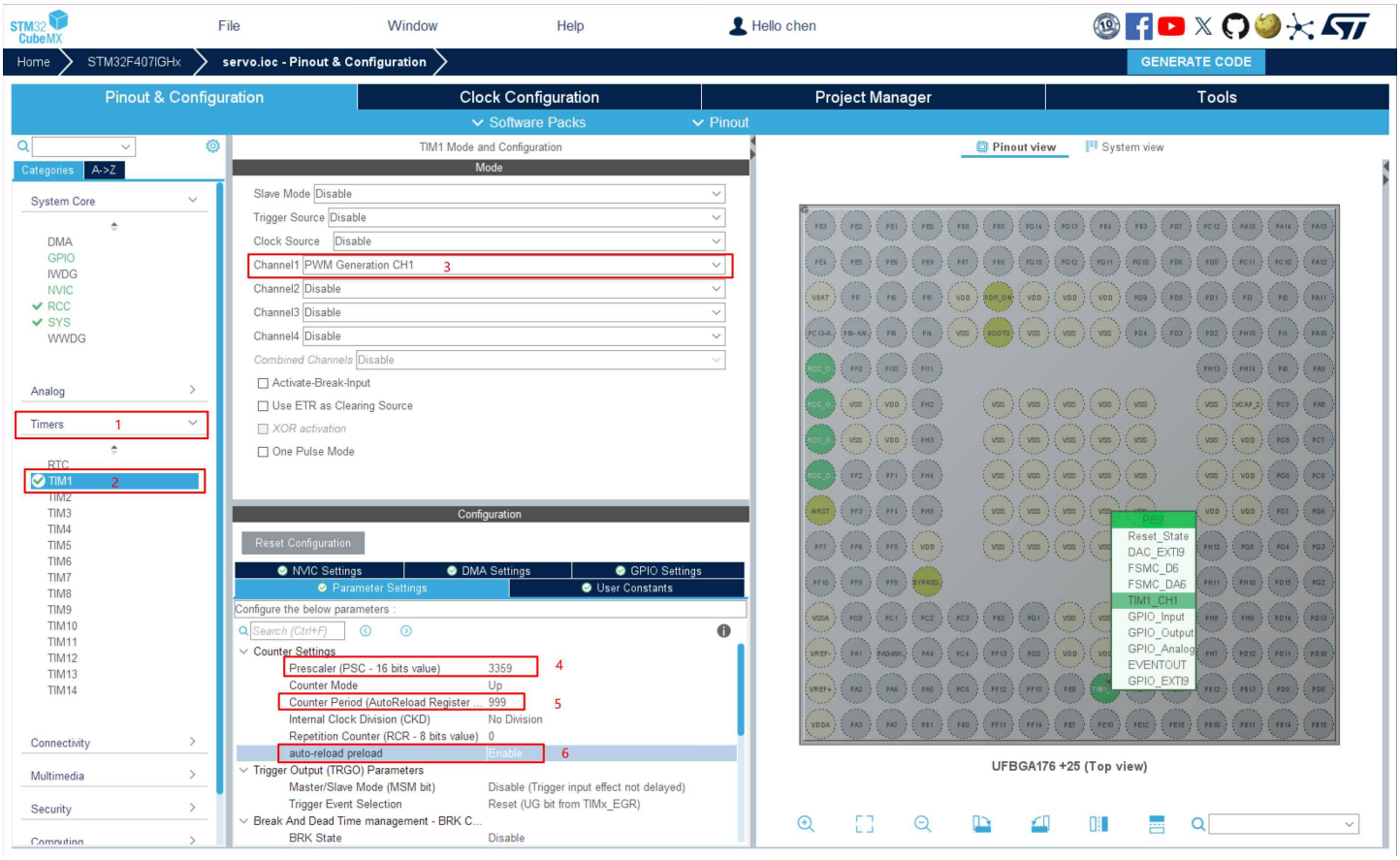
（建议以后调车时还是配置外部高速时钟以获得更稳定的时钟频率）



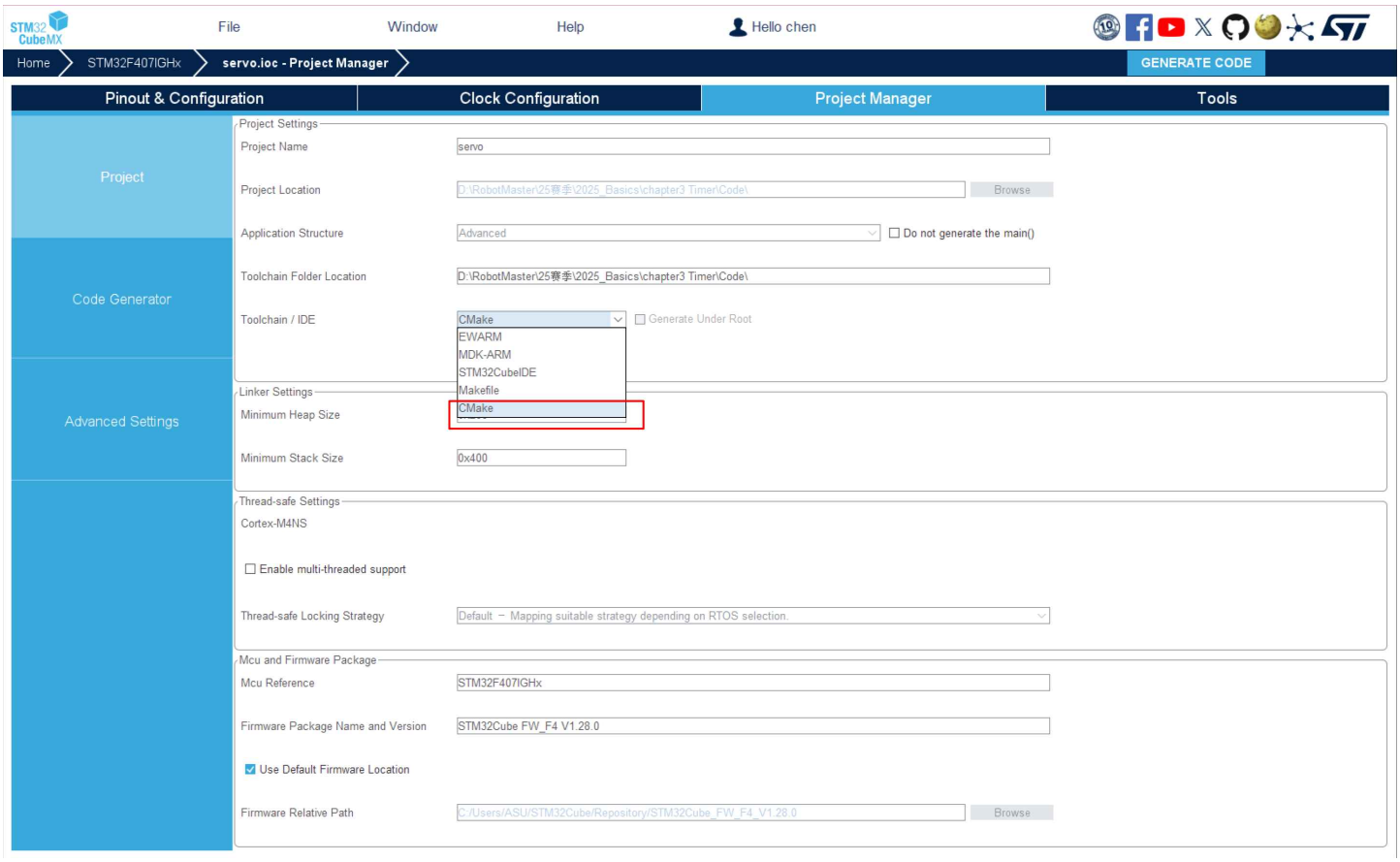
3. 根据C板用户手册第12页及第18页可知，C板上共有7个PWM接口，且均为高级定时器输出的PWM。我们不妨就使用第一个PWM进行实验，即TIM1_CH1，意思是第一个定时器的第一个通道。其对应的pin口为PE9。则在右边芯片引脚图上寻找该引脚，并配置为TIM1_CH1，如下图所示：

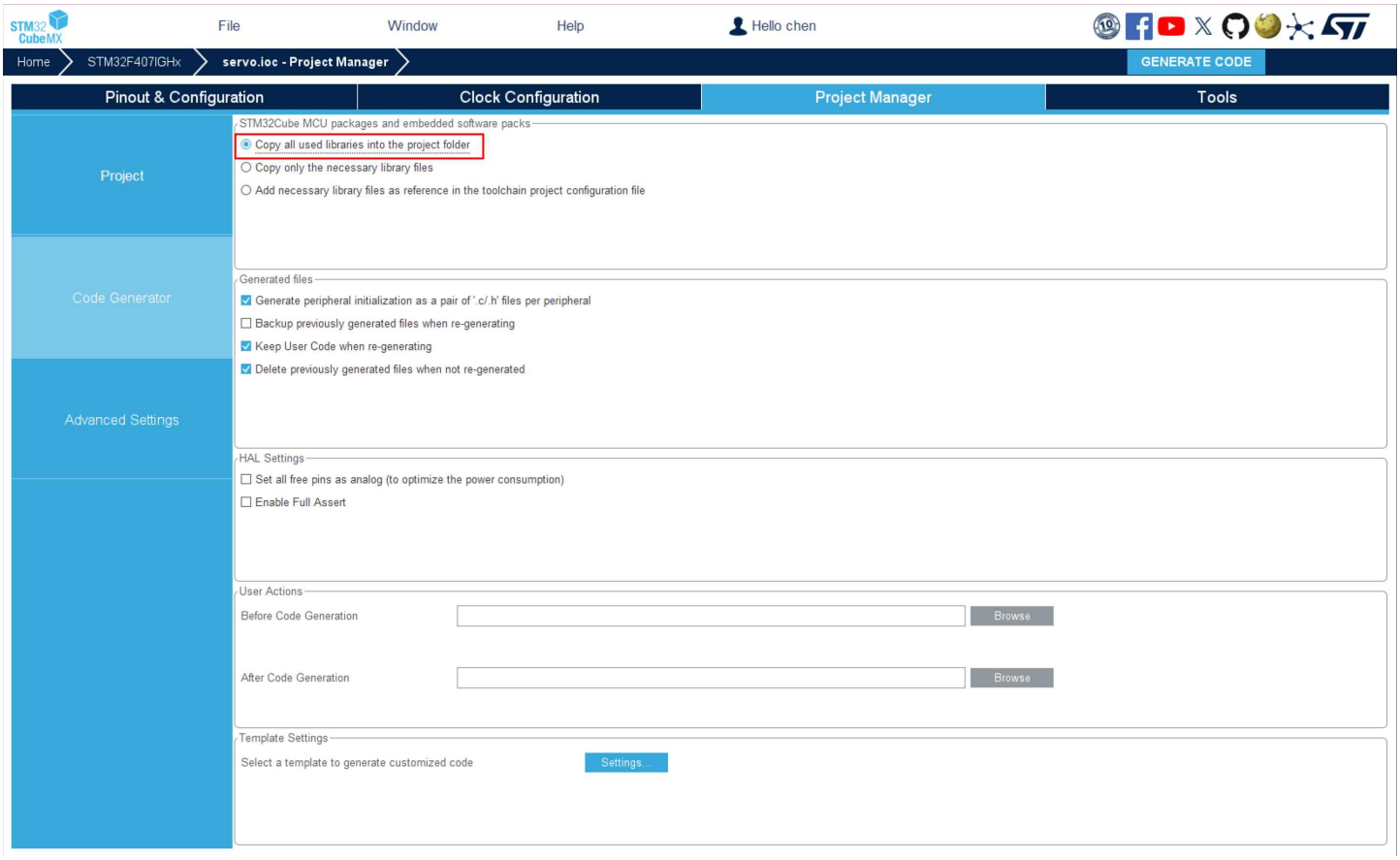


4. 在右边timer中选择TIM1进行配置，配置通道1为生成PWM，并配置下面的参数中预分频值为3359，自动重装载值为999，并配置auto-reload preload为开启。



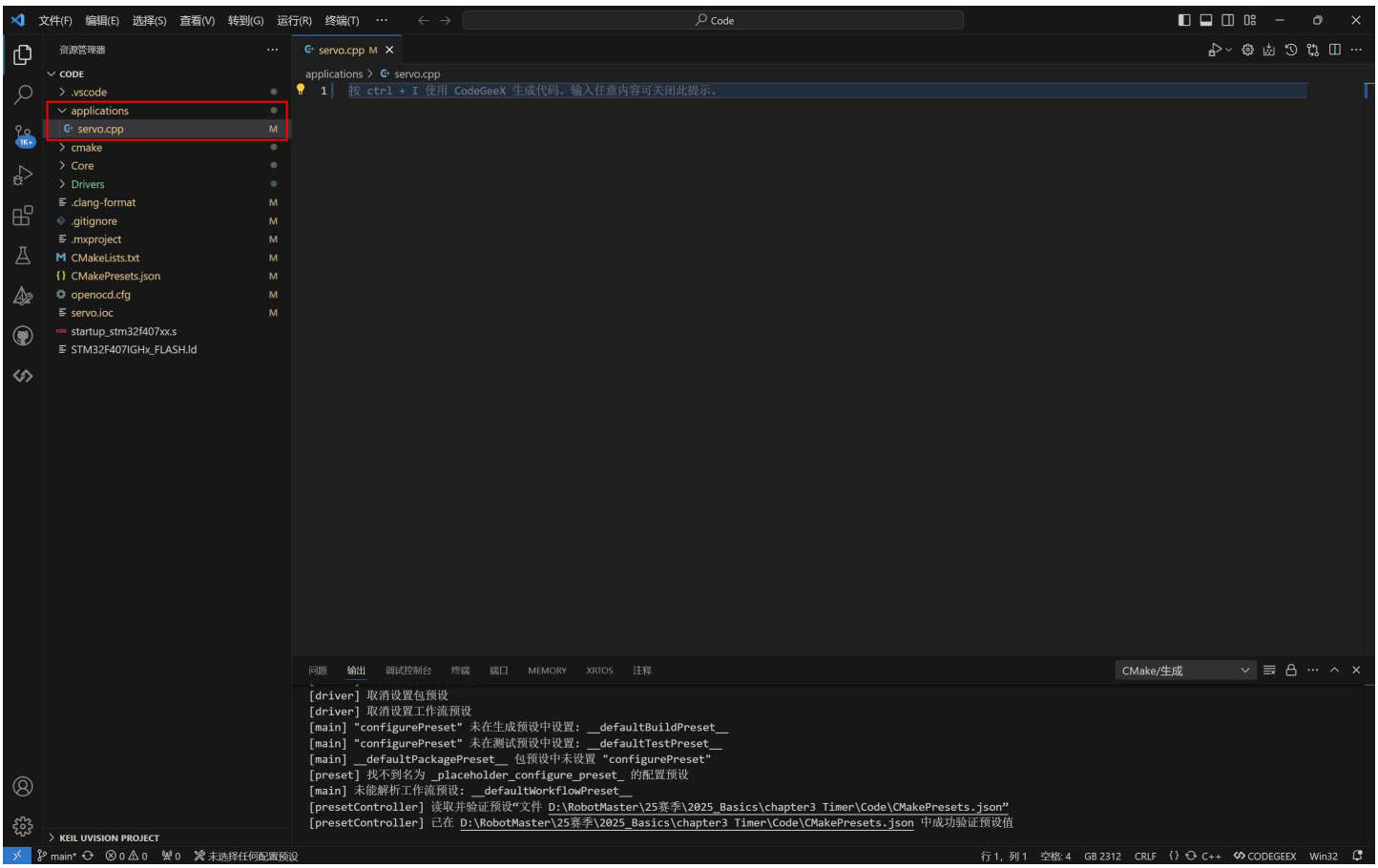
5. 在project manager下的project以及code generator页面进行类似于上一节的配置，并生成代码。



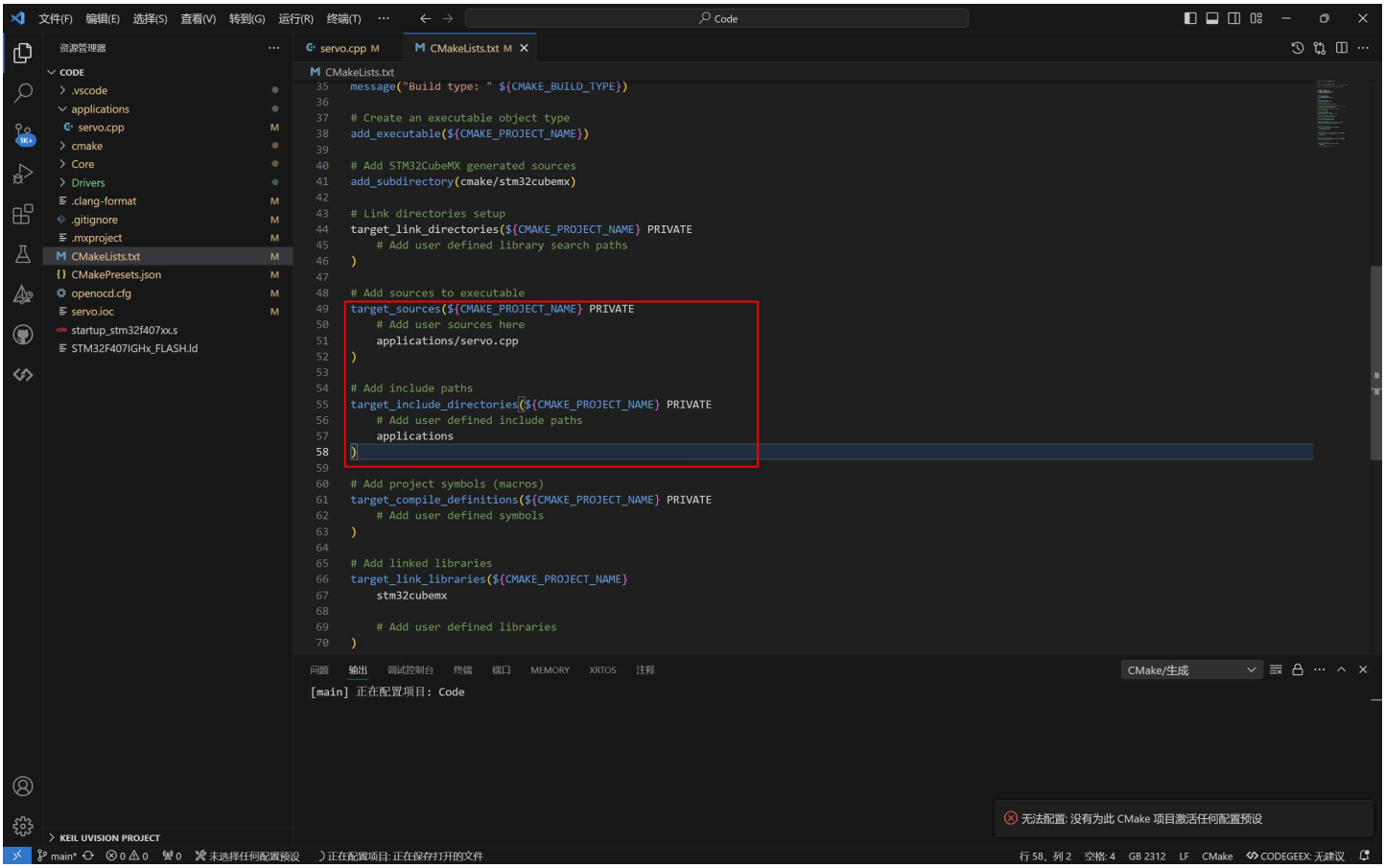


6. 将 `stm32_dev_env` 里file文件夹里的文件复制到CubeMX生成的文件夹中，并右键 `通过Code打开`

7. 同样地，新建 `applications` 文件夹，并且新建 `servo` 的源文件。



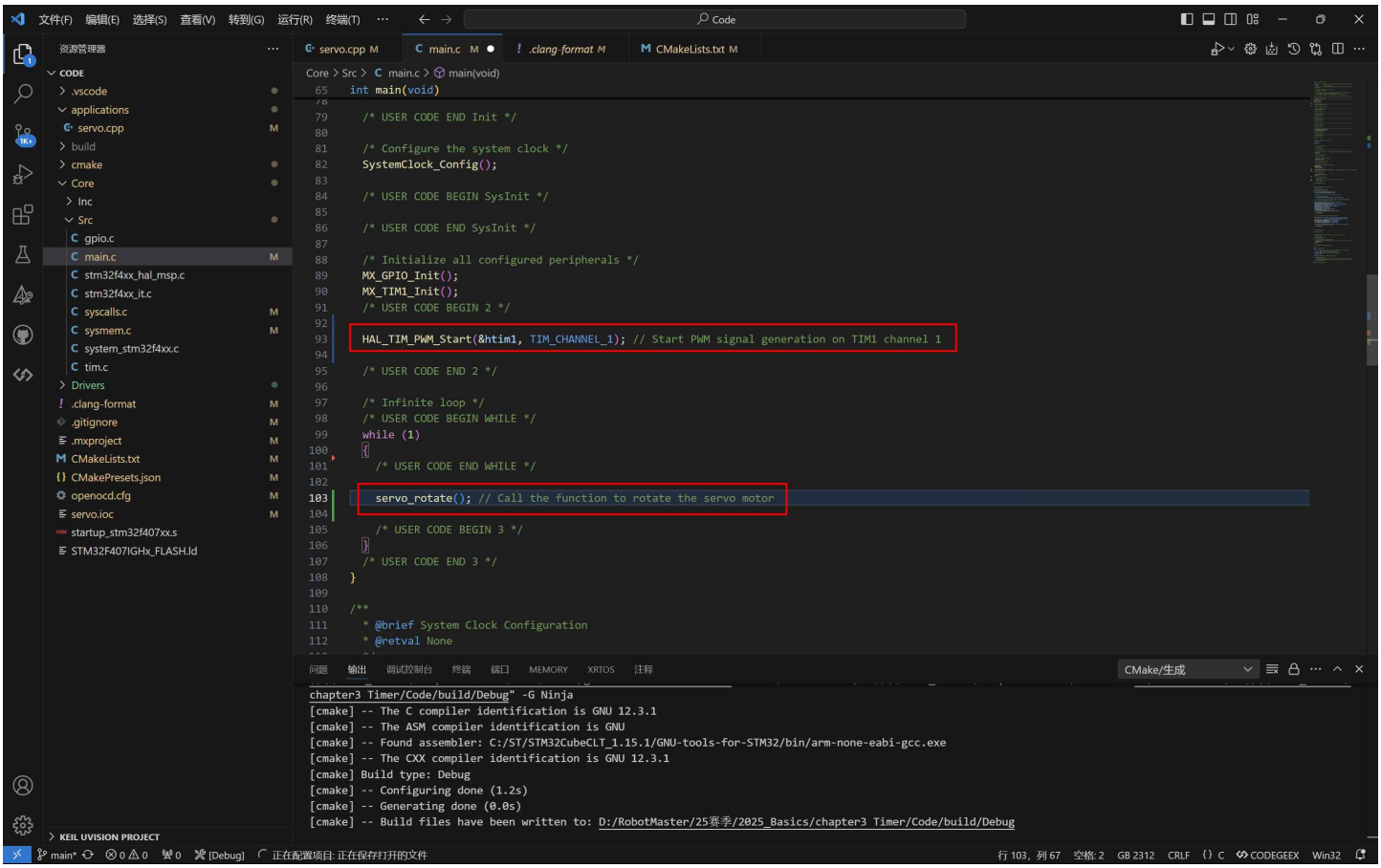
8. 在cmakelist.txt文件中添加刚才新建的头文件路径（`applicatios`）和源文件（`servo.cpp`）：



9. 将控制舵机运动的代码添加到`servo.cpp`

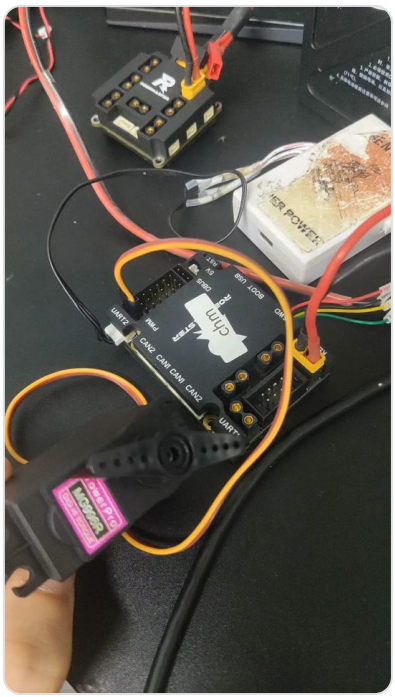
```cpp
#include "tim.h"

extern "C" void servo_rotate()
{
  __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1, 25);
  HAL_Delay(1000);
  __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1, 50);
  HAL_Delay(1000);
}
```

10. 在main函数中进行定时器的初始化以及servo中函数的调用：



11. 进行编译并烧录。

有以下现象则成功：



详解

那么上面的配置过程中发生了什么？下面同样来详细说明一下：

同样地，下面的原理部分了解即可。实在不会可以只学会配置即可。

首先你可以看看这个视频，虽然使用的不是同样的芯片，但是其中讲解的内容大同小异，极其清晰：

https://www.bilibili.com/video/BV1ph4y1e7Ey/?spm_id_from=333.788&vd_source=f040fd83c8ffbd80e7534c96be8d990a

1. 下面两张图片中展示的分别是STM32F4系列芯片的时钟树结构以及整体芯片框架图

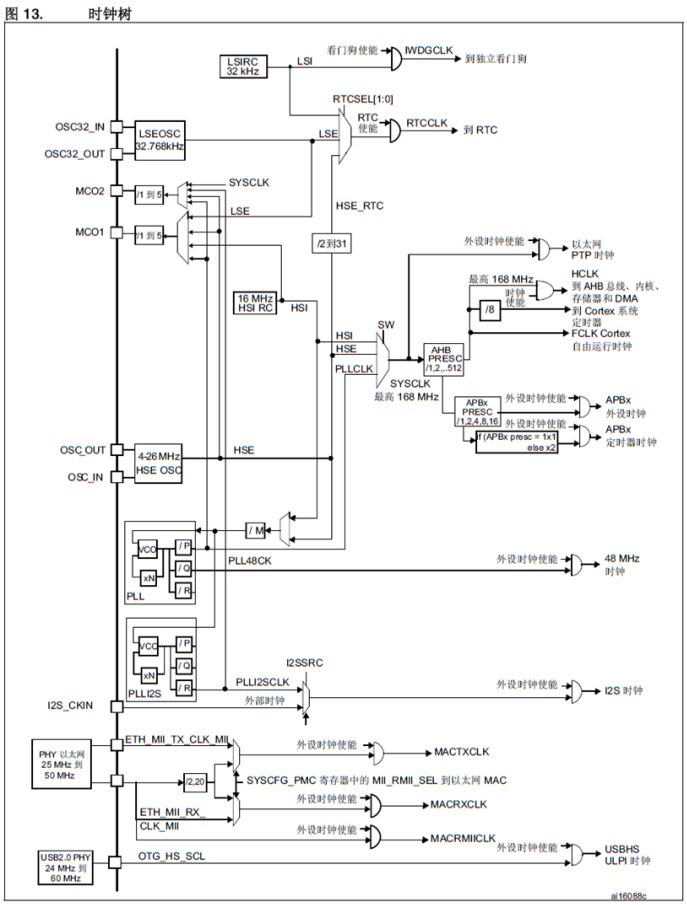
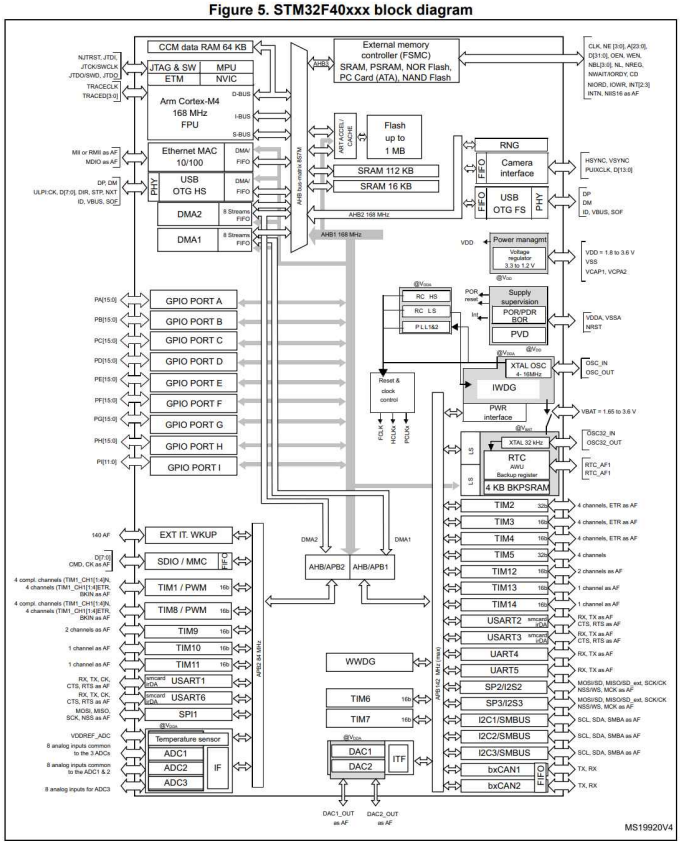


图 13. 时钟树

1. 有关内部和外部时钟源特性的所有详细信息，请参见器件数据手册的电气特性部分。



1. The camera interface and ethernet are available only on STM32F407xx devices.

从第二张图片中可以看到，作为高级定时器，TIM1和TIM8位于APB2总线上，则我们上面配置的TIM1，获得到的时钟信号就是APB2的定时器时钟信号。

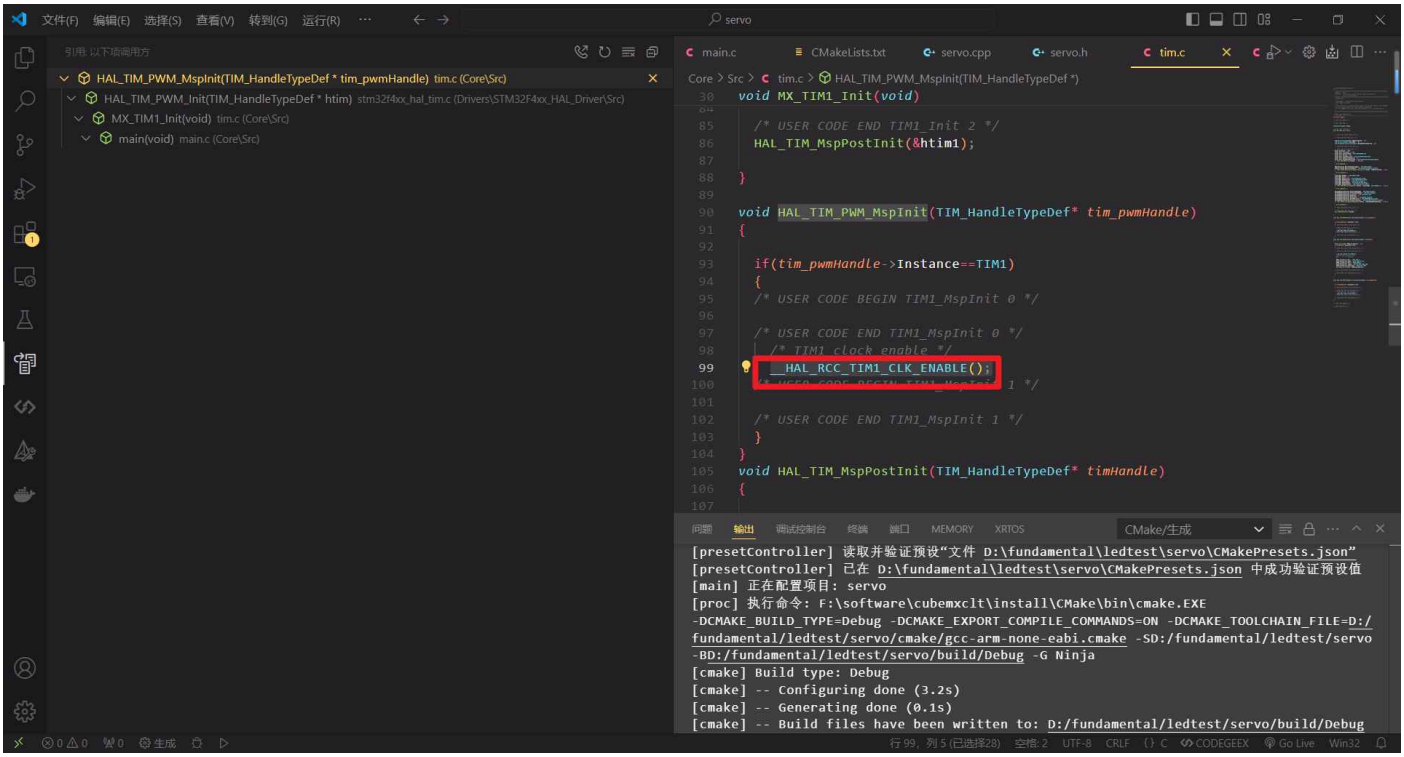
为了定时器能够达到足够高的频率，我们一般都需要设置cube中的RCC，以开启高速时钟，这样才能够将HCLK拉满到最高的168MHz，所以有上面的时钟树配置过程。

2. 下面的预分频值以及自动重装载值即按照最终得到的频率为50Hz进行配置，这里我们并不要求舵机有过高的精度，因此可以随机选取预分频值为3360-1，自动重装载值为1000-1以简化运算。
3. 在代码中需要注意，首先在初始化时需要对定时器进行初始化，进行使能后，后面才能进行进一步的操作。
4. 此处仅通过setcompare函数实现了对于占空比大小的改变，并没有涉及到在代码中改变预分频值以及自动重装载值。如果我们想要实现蜂鸣器在不同频率下的声音变化，则需要改变总的频率，此时需要直接对两个寄存器的值进行修改，同样有HAL库的函数进行这两种操作：

   在hal库函数文件中你可以找到这些函数，与setcompare函数在同一文件中。

需要注意，在上面的实例中，我们仅实现了定时器输出PWM的功能，事实上，定时器在计时上的用处也很大，利用定时器中断可以完成很多任务，在实现定时器中断等其他操作时，需要在配置时将clock source选为internal clock，以便使得定时器拥有时钟脉冲。

至于上面输出PWM的实例中没有设置，主要是因为：



在定时器初始化时，会调HAL_TIM_PWM_Init，这里面的HAL_TIM_PWM_MspInit会开rcc，使得定时器可以使用默认的时钟信号