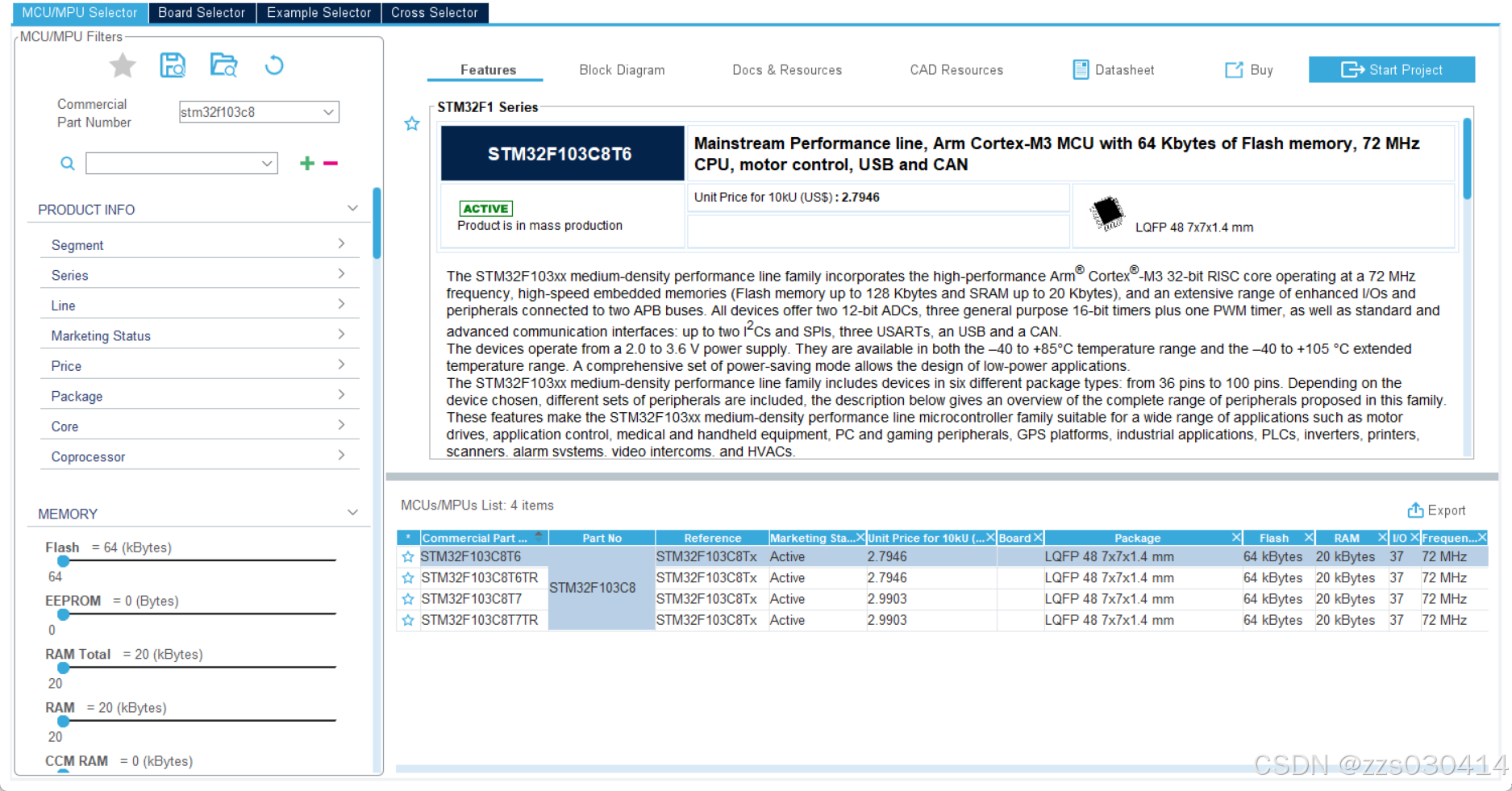
# **STM32使用定时器进行串口通信和点灯和stm32基于PWM的呼吸灯**

任务一：STM32使用定时器进行串口通信和点灯

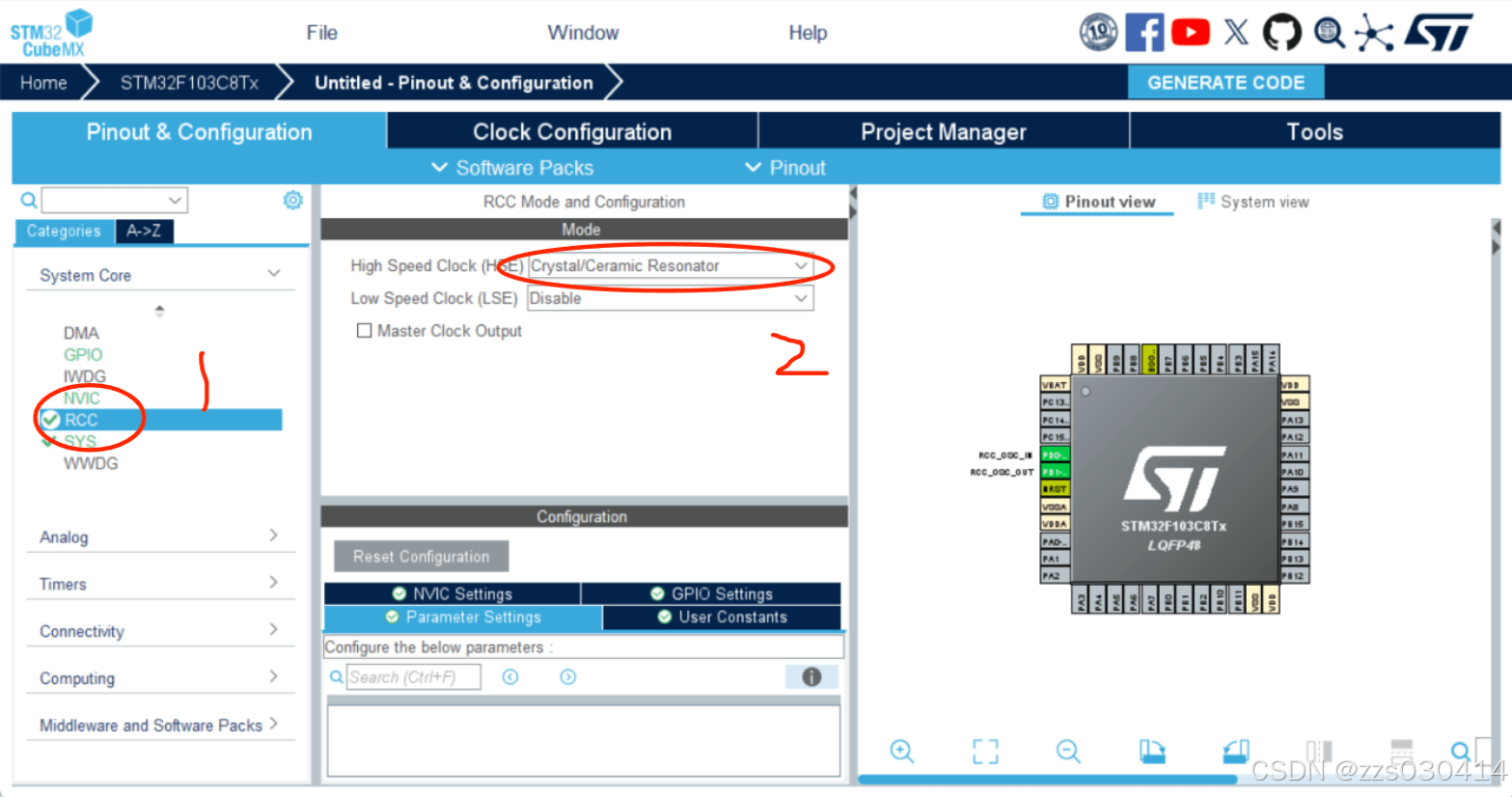
一 工程建立

1 创建新项目

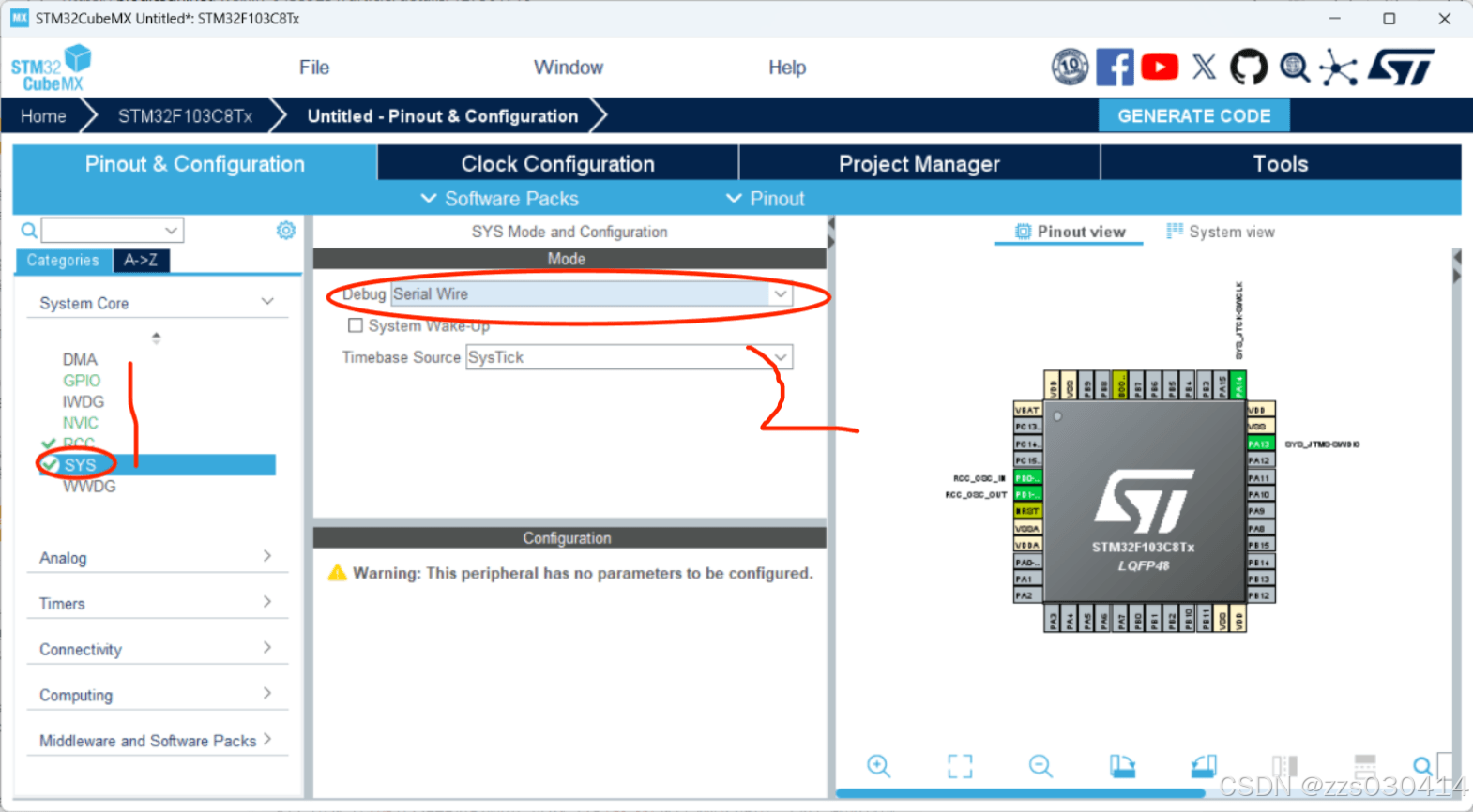
点击ACCEE TO MCU SELECTOR，选择stm32f103c8t6



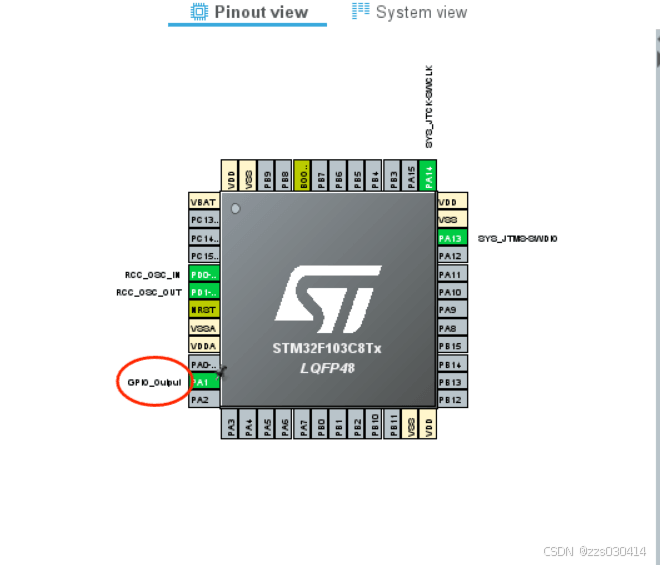
2 配置rcc



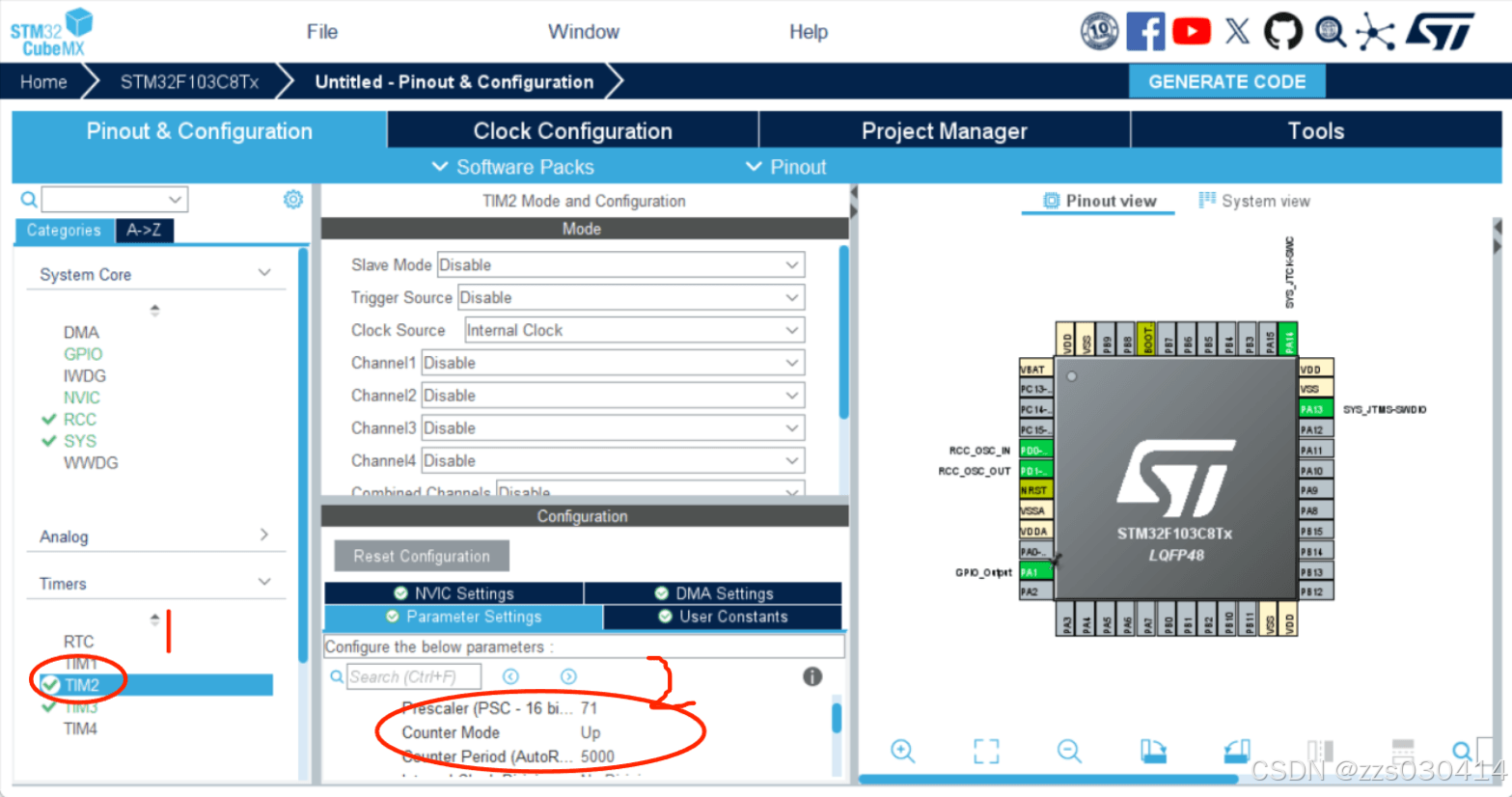
3 配置SYS

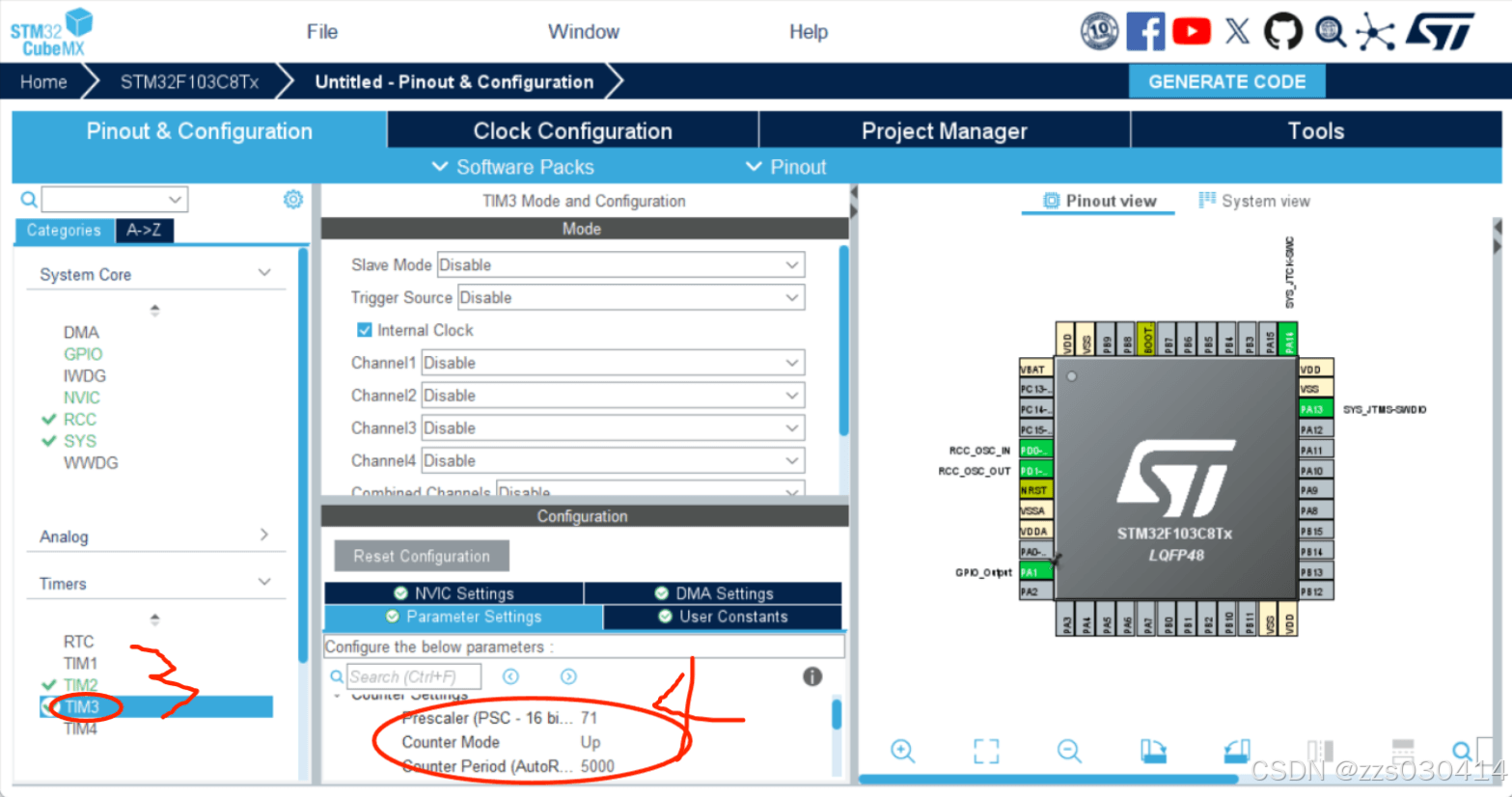


4 配置IO口输出

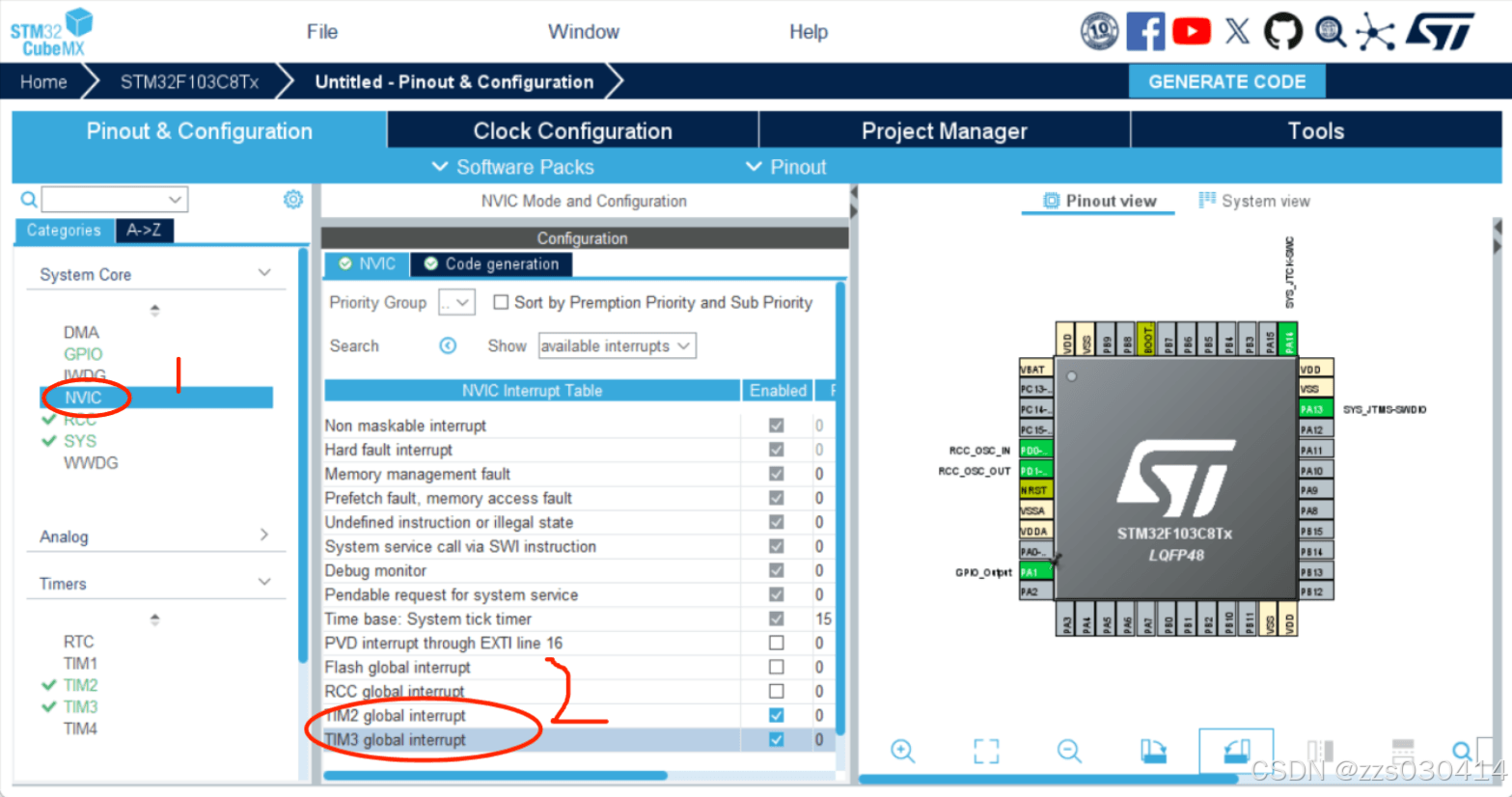


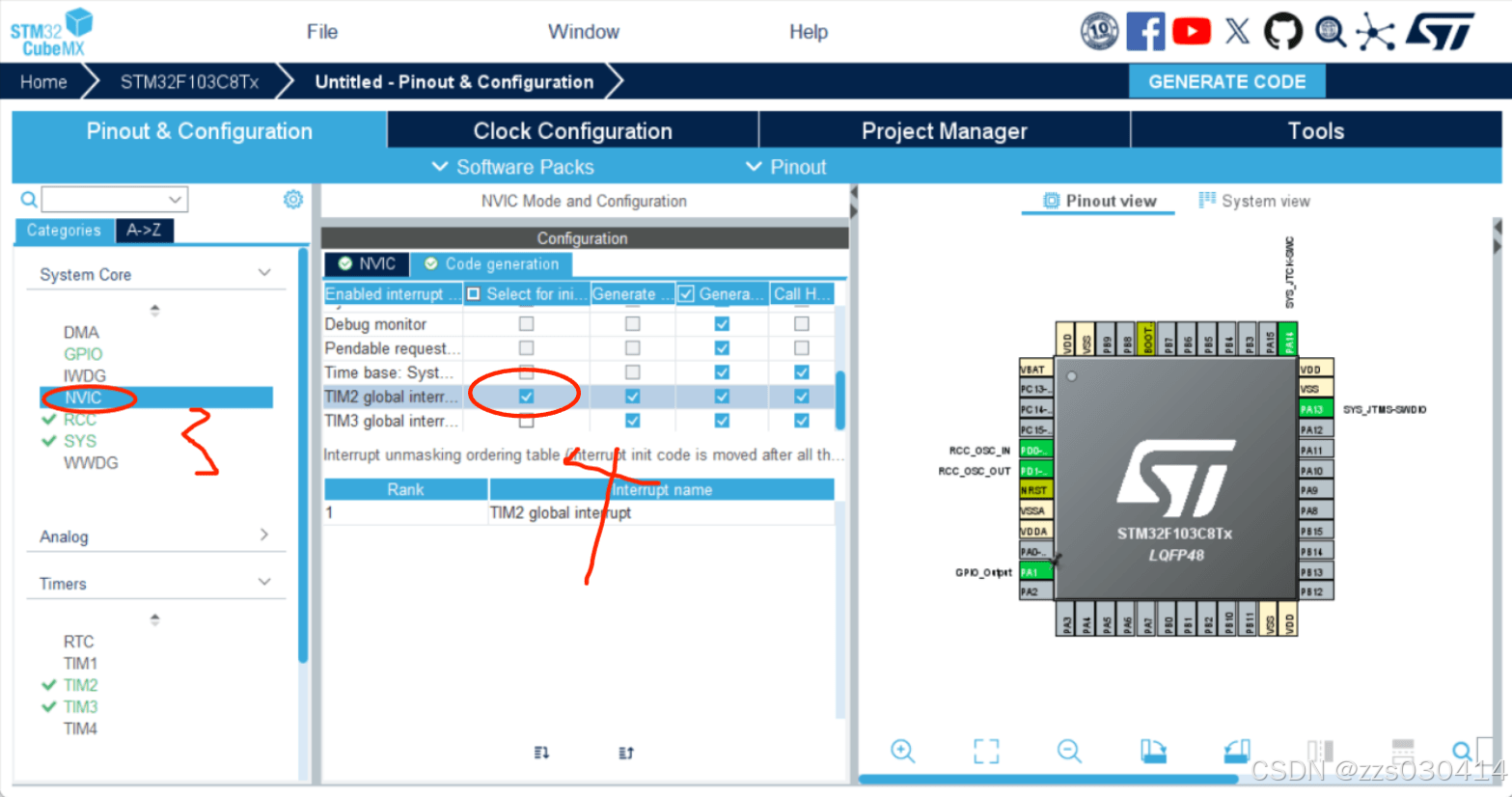
5 配置定时器2和定时器3



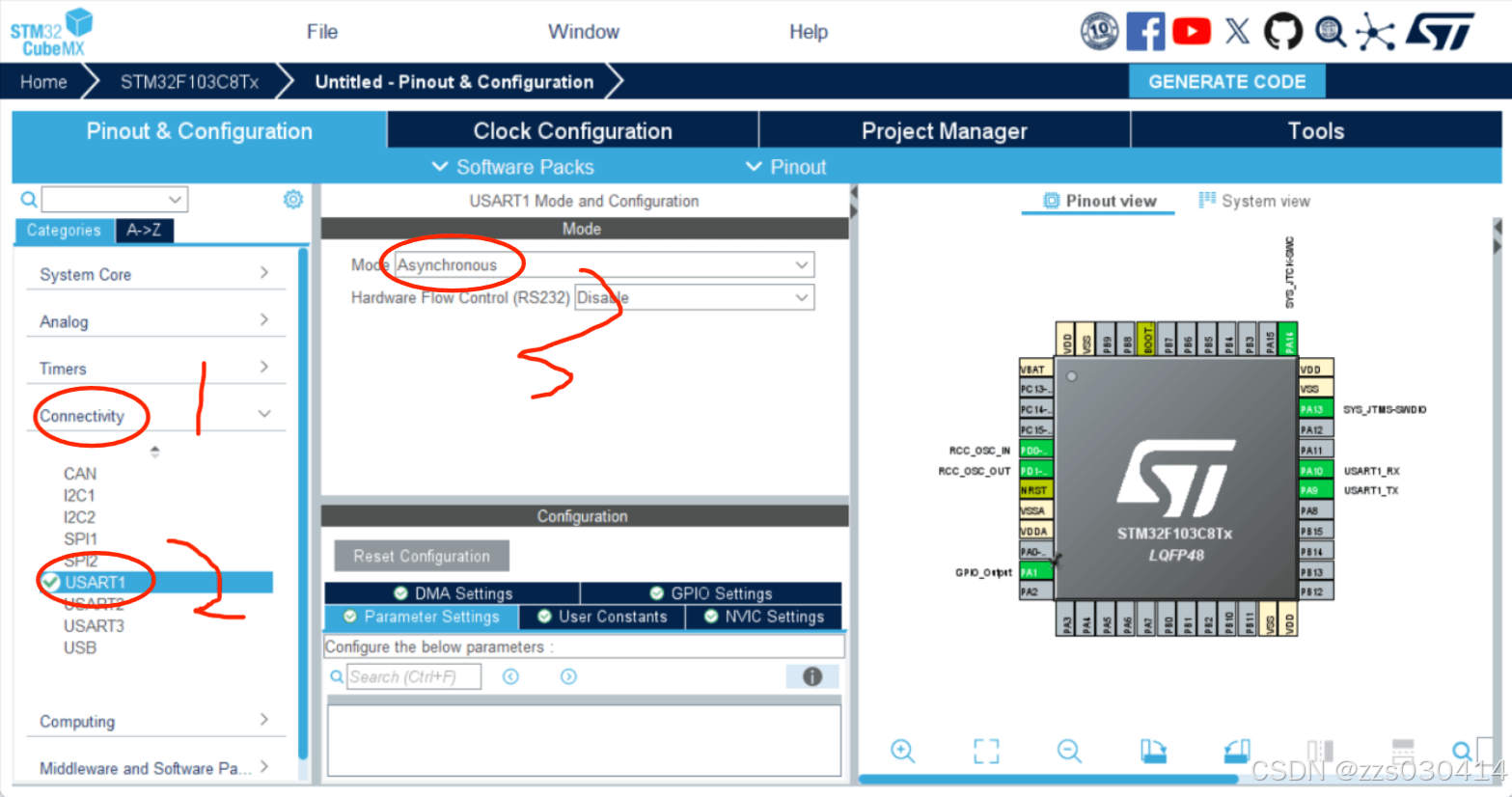


6 配置中断

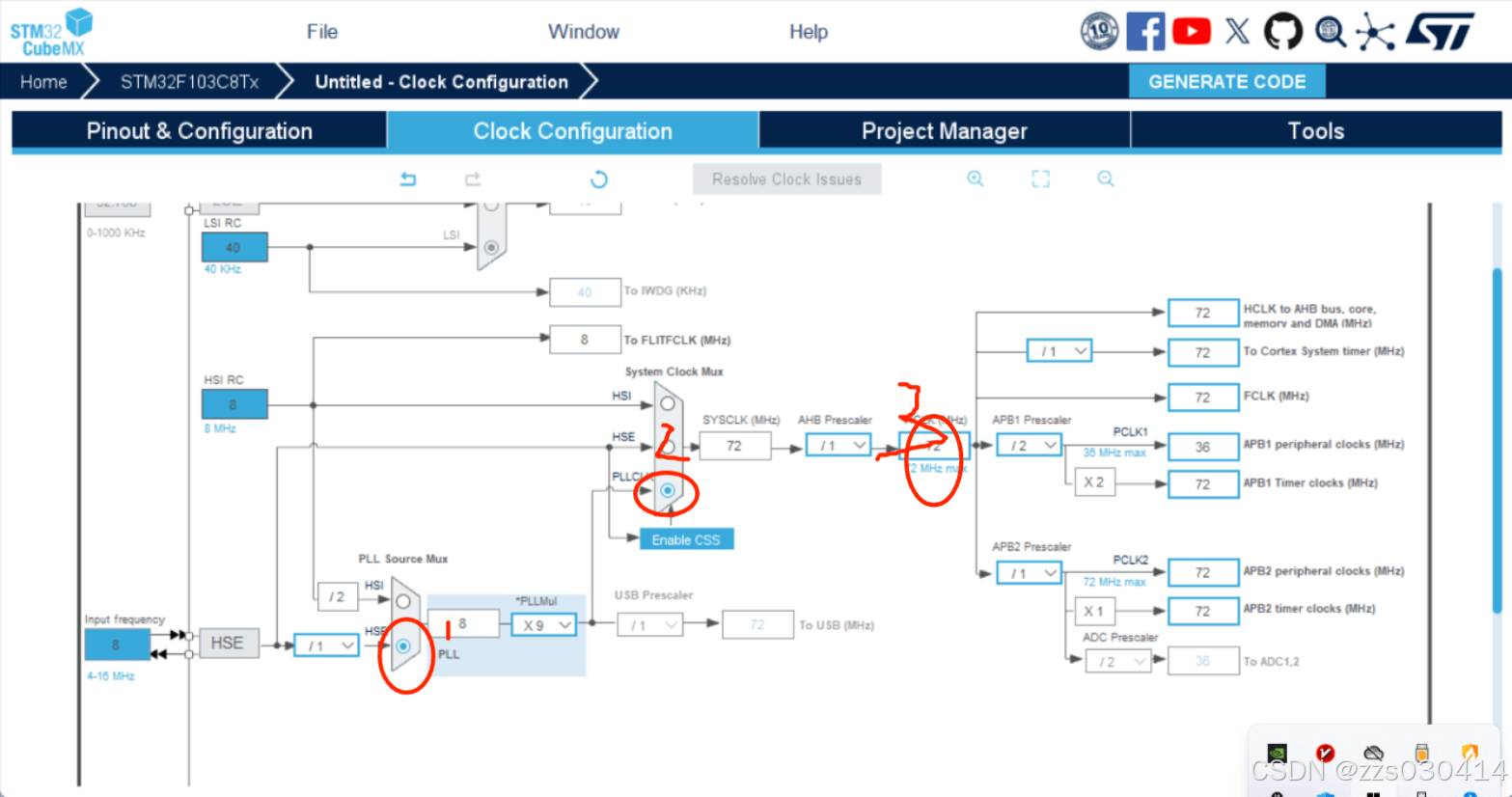




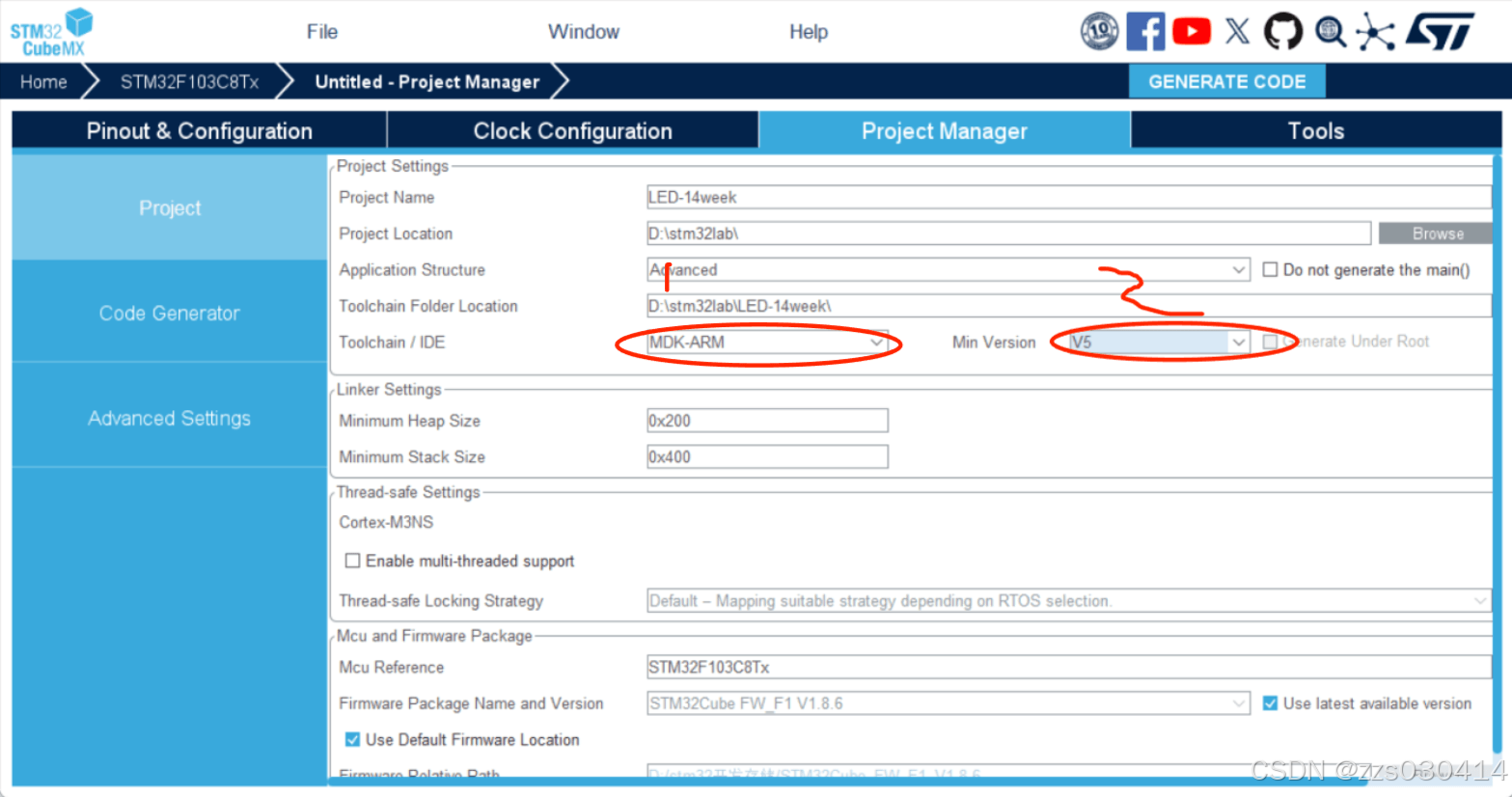
7 配置usart



8 时钟配置



9 配置项目设置



10 生成项目

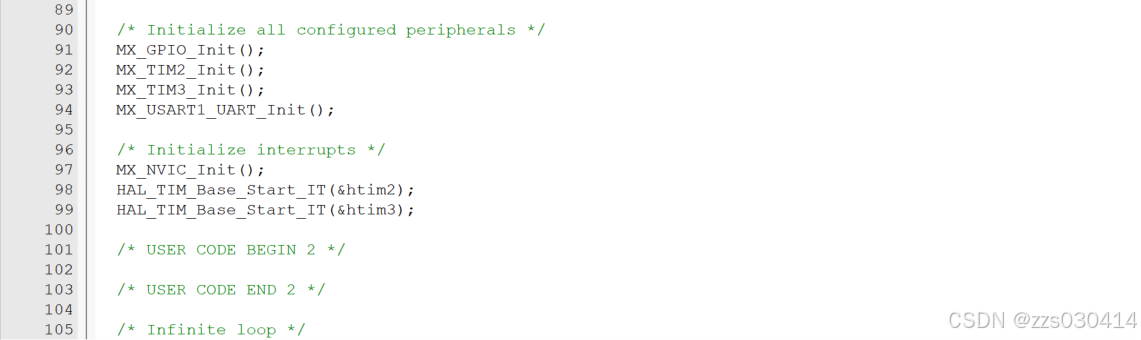


二 代码编写

1 启动定时器

HAL\_TIM\_Base\_Start\_IT(&htim2);

HAL\_TIM\_Base\_Start\_IT(&htim3);



2 串口通信

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

static uint32\_t time\_cnt =0;

static uint32\_t time\_cnt3 =0;

if(htim->Instance == TIM2)

{

if(++time\_cnt >= 400)

{

time\_cnt =0;

HAL\_GPIO\_TogglePin(GPIOA,GPIO\_PIN\_1);

}

}

if(htim->Instance == TIM3)

{

if(++time\_cnt3 >= 1000)

{

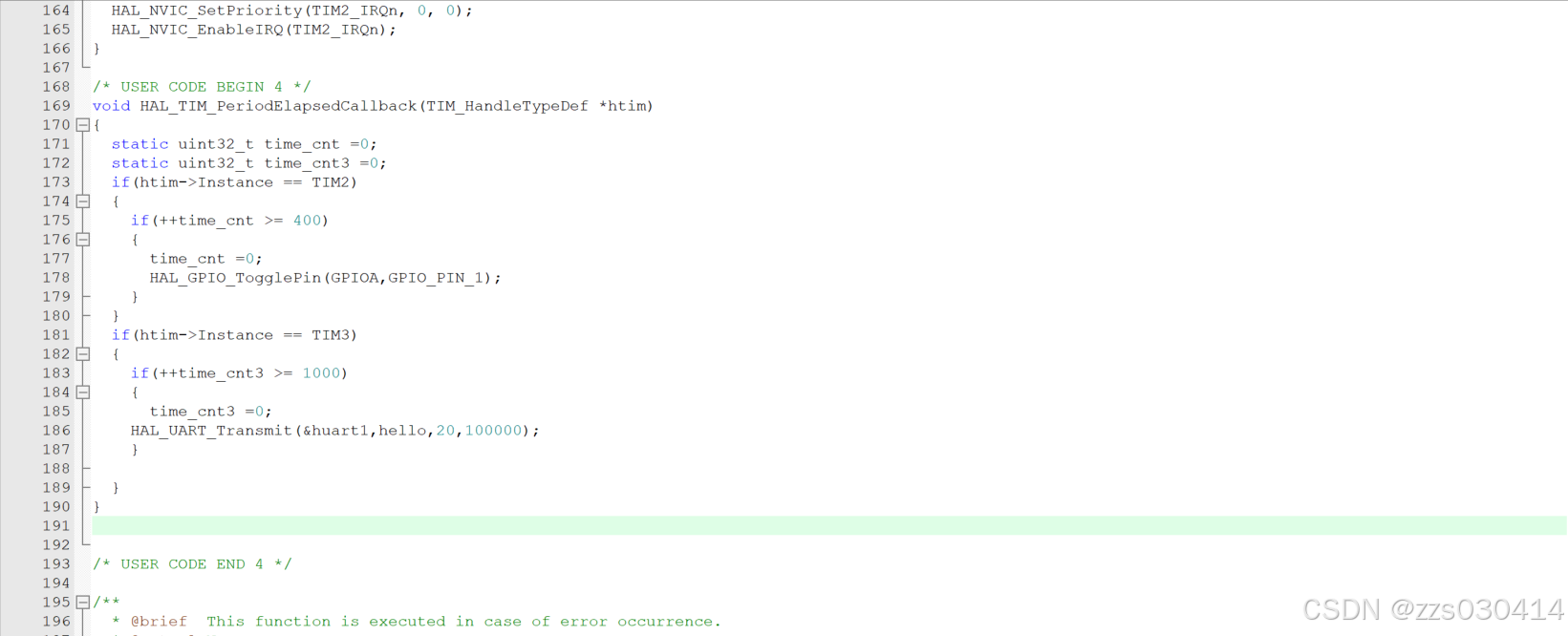
time\_cnt3 =0;

HAL\_UART\_Transmit(&huart1,hello,20,100000);

}

}

}



三 结果演示



四 总结

在这次实验中，我通过STM32微控制器使用定时器实现了串口通信和点灯功能，这个过程让我对STM32的硬件编程有了更深入的理解。

首先，我在STM32CubeMX软件中创建了新项目，并选择了STM32F103C8T6作为目标芯片。通过这个软件，我能够图形化地配置RCC、SYS、IO口、定时器、中断和USART等关键参数，这大大简化了配置过程，也减少了出错的可能性。

在代码编写阶段，我首先启动了两个定时器，TIM2和TIM3。TIM2用于控制LED灯的闪烁，通过在定时器的周期中断回调函数中切换GPIO状态来实现。我设置了一个计数器time\_cnt，每当计数器达到400时，就翻转GPIOA的第1个引脚，从而控制LED灯的亮灭。这种定时器中断的方式，让我能够精确控制LED灯的闪烁频率。

TIM3则用于串口通信。我在TIM3的周期中断回调函数中，通过HAL\_UART\_Transmit函数发送了一个字符串hello。这里我也设置了一个计数器time\_cnt3，每当计数器达到1000时，就发送一次数据。这种方式让我能够以固定的时间间隔发送串口数据，对于实现周期性数据传输非常有用。

通过这次实验，我不仅学会了如何配置STM32的硬件参数，还掌握了如何使用HAL库来操作STM32的外设。我了解到，通过合理地使用定时器中断，可以有效地控制硬件行为，实现精确的时间控制。同时，我也体会到了STM32CubeMX和HAL库在STM32开发中的便利性，它们极大地提高了开发效率，让我能够更专注于算法和逻辑的实现。

总的来说，这次实验不仅加深了我对STM32编程的理解，也提升了我的嵌入式系统开发技能。我相信这些知识和技能将在我未来的学习和工作中发挥重要作用。

任务二：stm32基于PWM的呼吸灯

一 工程建立

1 创建新项目

和实验一一样

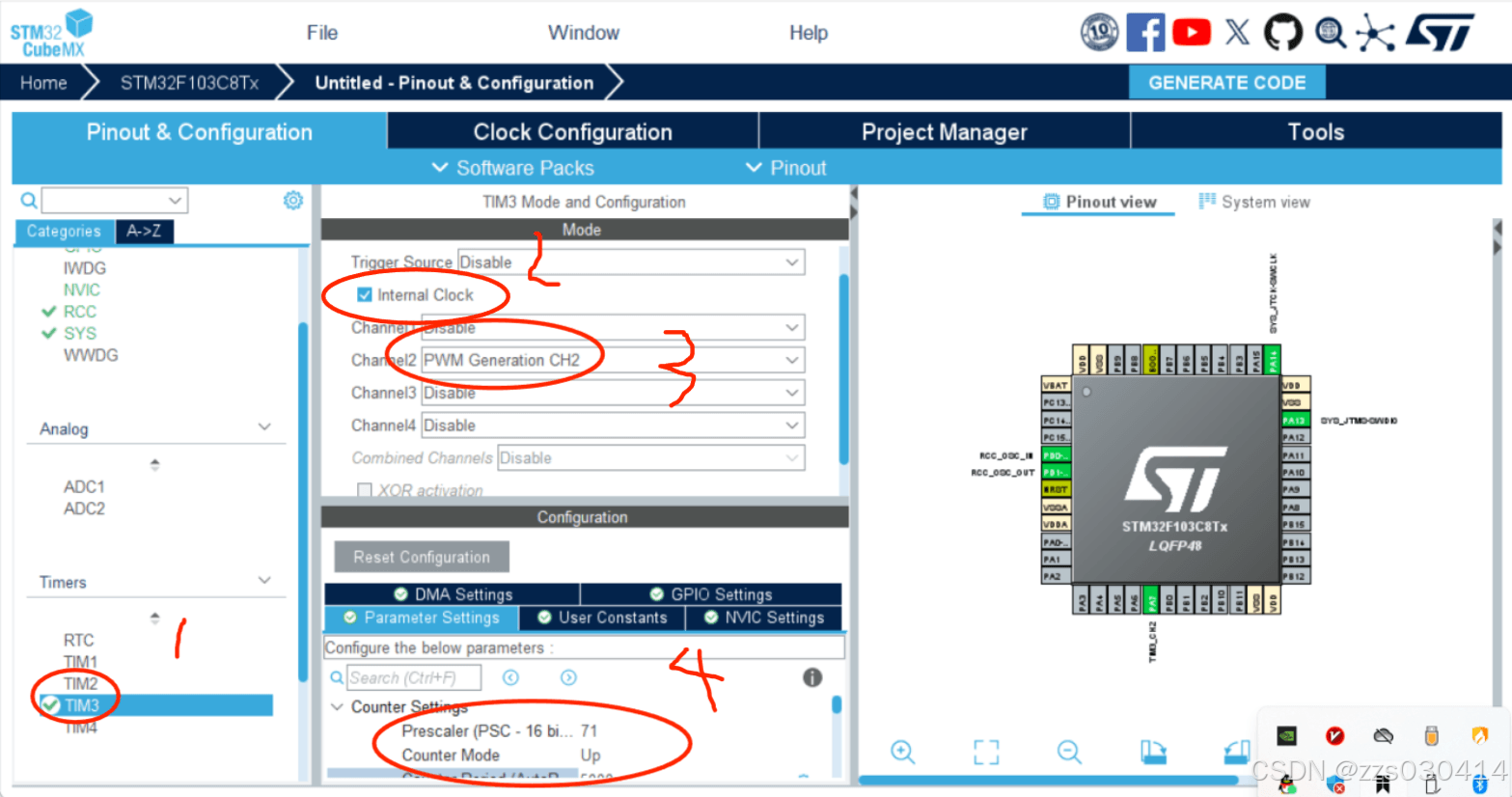
2 配置rcc

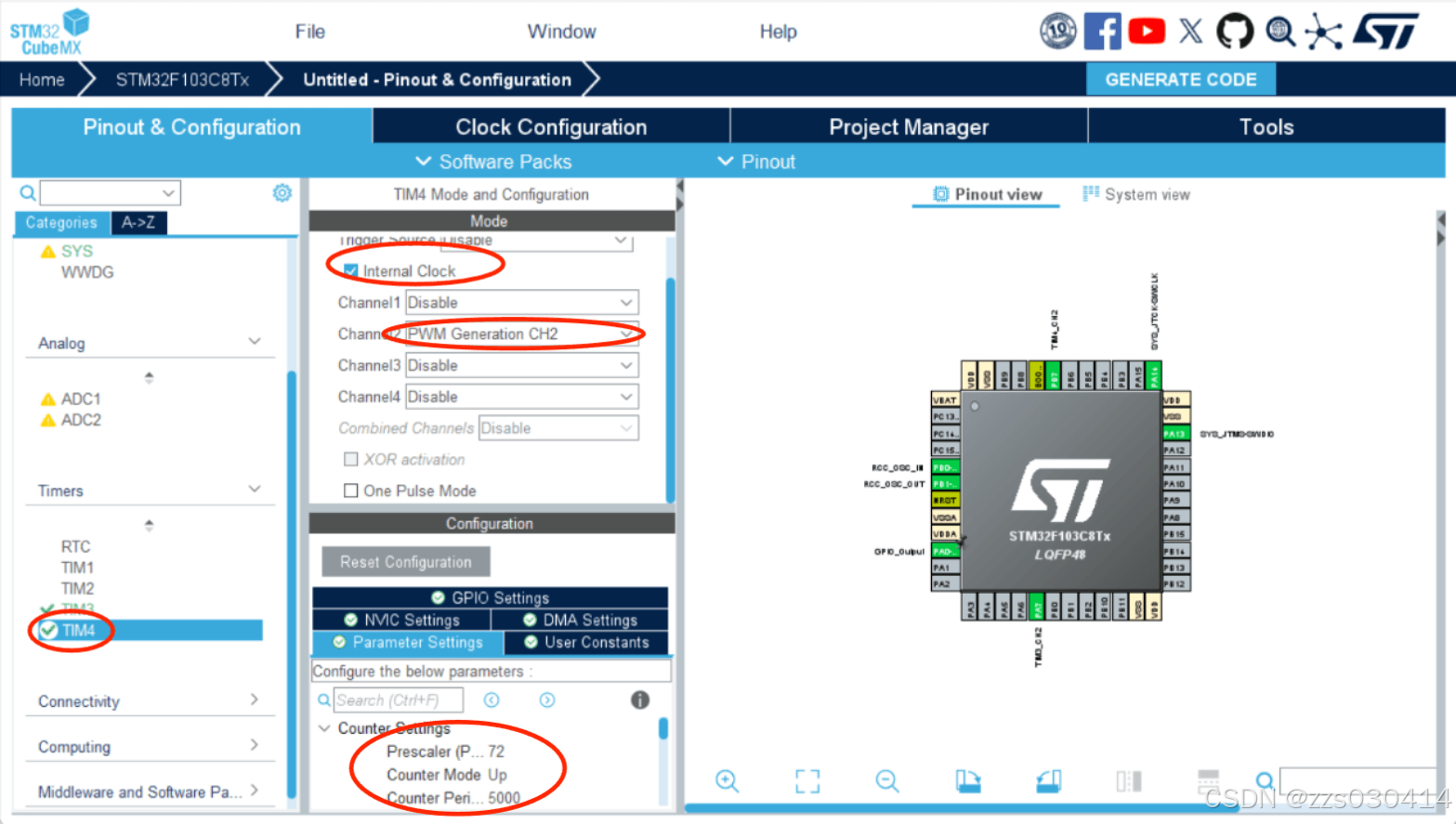
和实验一一样

3 配置sys

和实验一一样

4 配置定时器3和4





5 时钟配置

和任务一一样

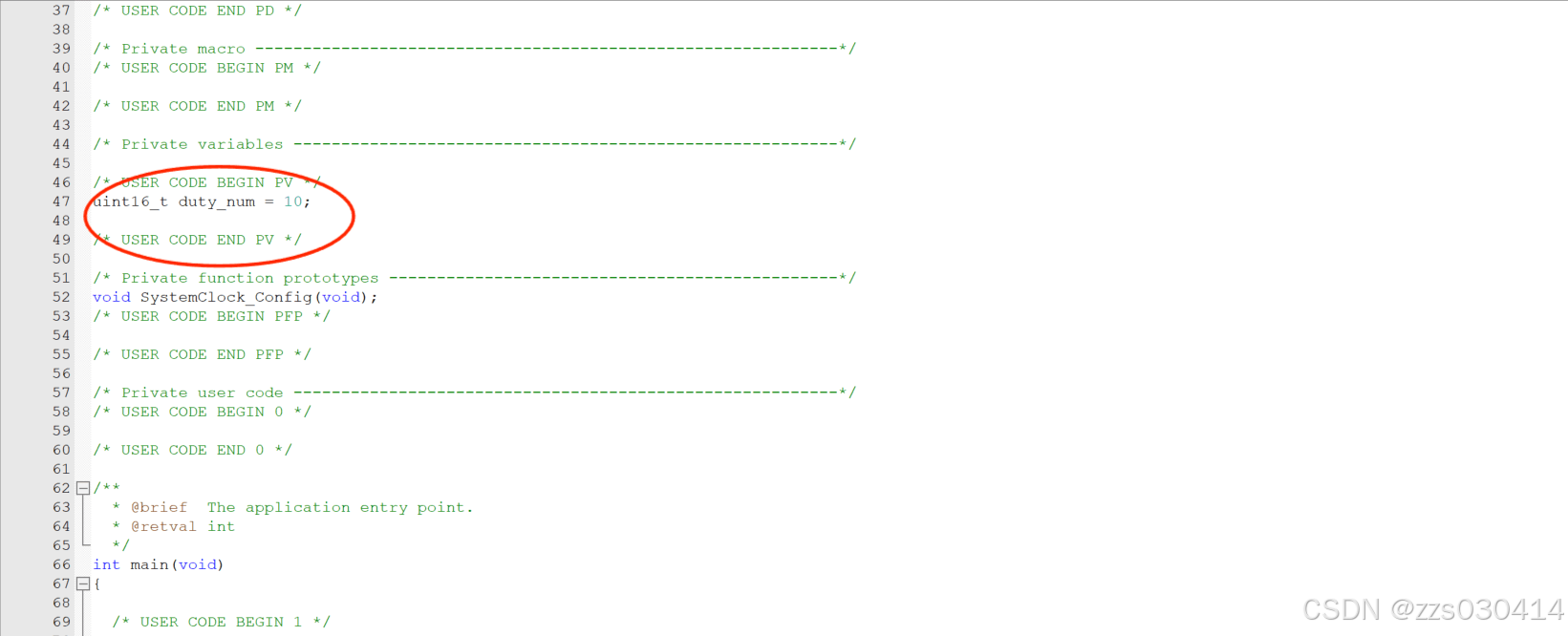
6 生成项目

和任务一一样

二 代码编写

1 代码位置1

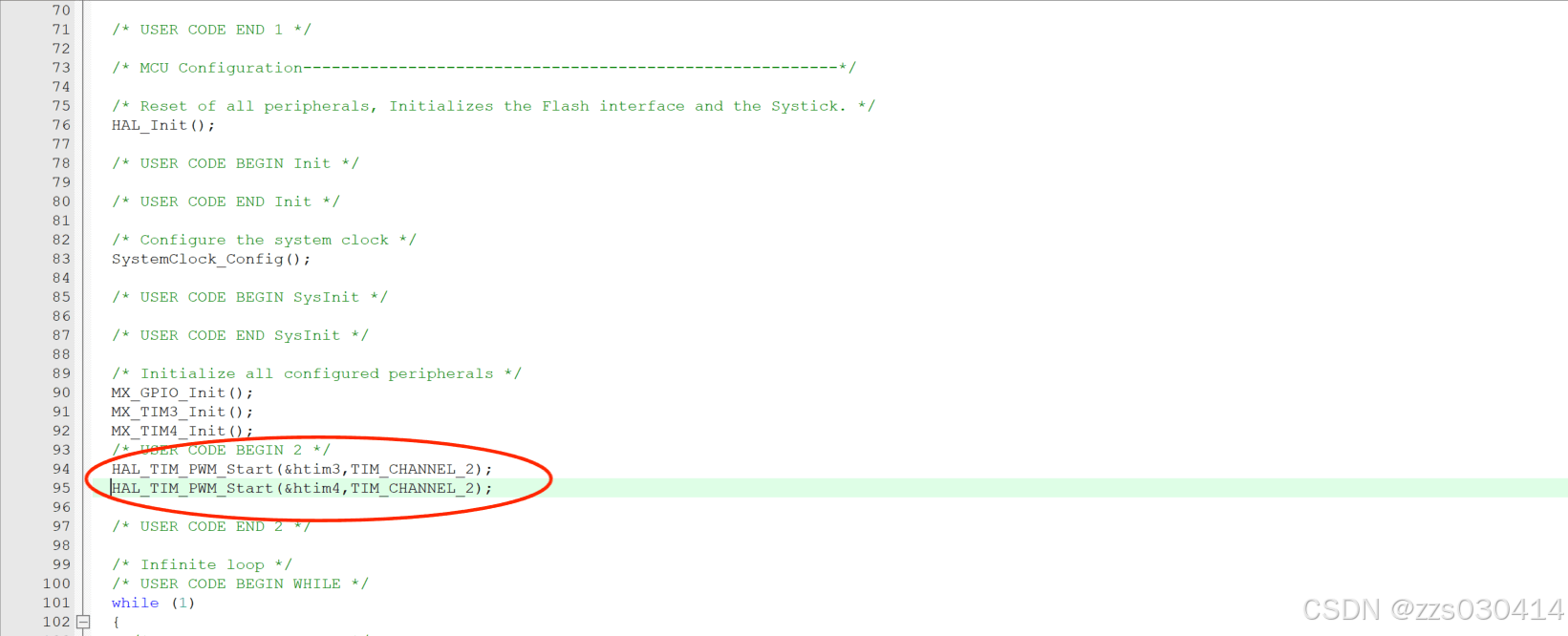
代码：uint16\_t duty\_num = 10;



2 代码位置2

开始TIM3和TIM4的通道2，输出PWM。代码：HAL\_TIM\_PWM\_Start(&htim3,TIM\_CHANNEL\_2);

HAL\_TIM\_PWM\_Start(&htim4,TIM\_CHANNEL\_2);



3 位置3

隔50毫秒，占空比加10，如果超过500（也就是PWM周期），自动变成0，代码：

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

HAL\_Delay(50);

duty\_num = duty\_num + 10;

if(duty\_num > 500)

{

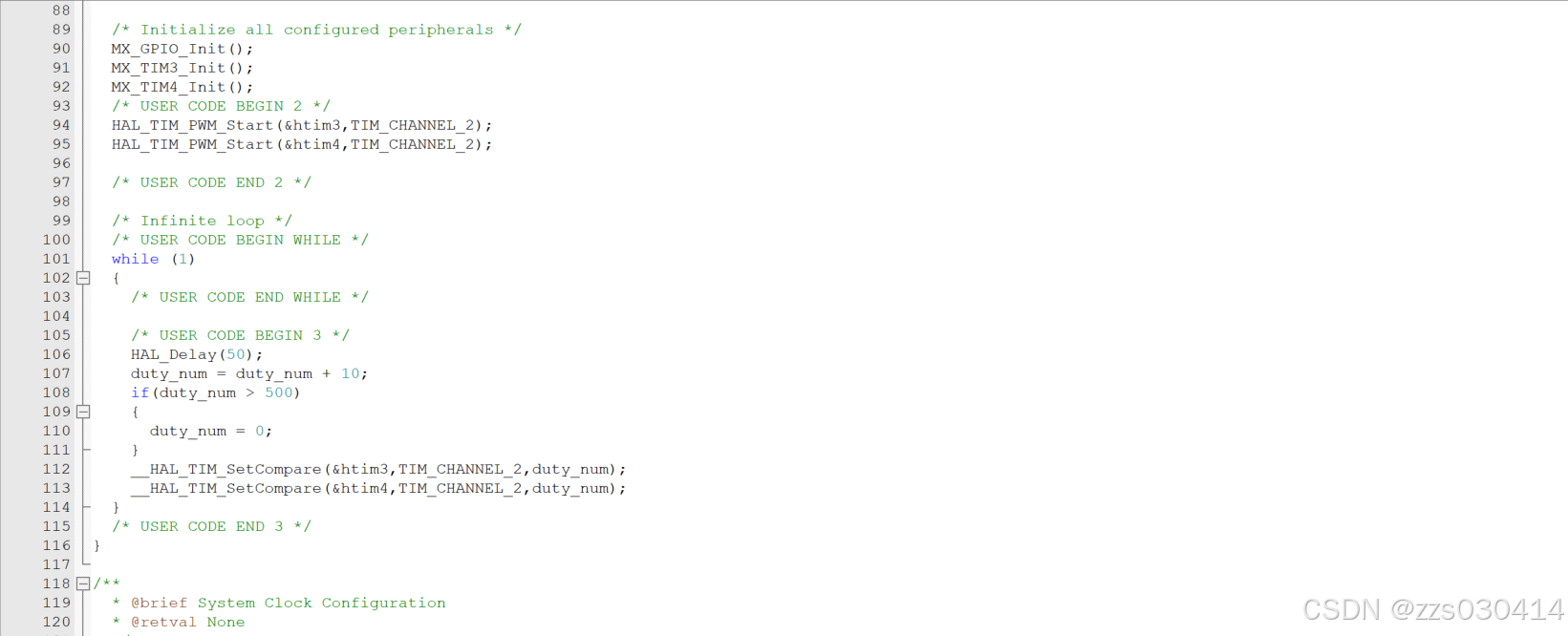
duty\_num = 0;

}

\_\_HAL\_TIM\_SetCompare(&htim3,TIM\_CHANNEL\_2,duty\_num);

\_\_HAL\_TIM\_SetCompare(&htim4,TIM\_CHANNEL\_2,duty\_num);

}



三 结果演示



四 总结

在完成基于PWM的呼吸灯实验后，我对STM32微控制器的PWM功能有了更深入的认识和理解。这个实验不仅让我掌握了PWM的基本原理，还让我学会了如何在实际项目中应用这一技术来实现特定的功能。

实验的第一步是建立工程，这与任务一中的步骤相似。我再次使用了STM32CubeMX工具来配置RCC、SYS、定时器3和4以及时钟设置。这个工具的便捷性让我能够快速地完成硬件的初始化配置，为后续的软件开发打下了良好的基础。

在代码编写阶段，我首先定义了一个duty\_num变量，用于存储PWM信号的占空比。接着，我启动了TIM3和TIM4的通道2，设置为PWM输出模式。这一步是通过调用HAL\_TIM\_PWM\_Start函数来实现的，它允许我通过软件控制硬件的PWM输出。

然后，我进入了一个无限循环，每隔50毫秒增加duty\_num的值，模拟呼吸灯的效果。当duty\_num超过500时，我将其重置为0，这样占空比就会从0开始逐渐增加到最大值，然后再回到0，形成一个周期性的呼吸效果。通过调用\_\_HAL\_TIM\_SetCompare函数，我能够实时更新TIM3和TIM4的占空比，从而控制连接到这两个定时器通道的LED灯的亮度。

这个实验让我深刻体会到了PWM在控制模拟设备中的重要性。通过调整占空比，我能够精确控制LED灯的亮度，实现平滑的亮度变化，这对于创建视觉效果非常有帮助。此外，我也认识到了STM32微控制器的强大功能和灵活性，它能够通过软件编程实现复杂的硬件控制。

通过这次实验，我不仅提升了自己的编程技能，还增强了对嵌入式系统的理解。我相信，这些知识和技能将在我的未来学习和职业发展中发挥重要作用。我期待将这些技术应用到更复杂的项目中，以实现更高级的功能和创新的设计。