



Solarec 定时任务

开发指南

文档版本 02

发布日期 2023-09-27

版权所有 © 海思技术有限公司2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

海思技术有限公司

地址：上海市青浦区虹桥港路2号101室 邮编：201721

网址：<https://www.hisilicon.com/cn/>

客户服务邮箱：support@hisilicon.com



前言

概述

本文档介绍定时任务相关的概念以及使用指导。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
SolarA ²	1.0.1




读者对象

本文档（本指南）主要适用于以下工程师：



技术支持工程师
客户开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。



符号	说明
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2022-11-07	01	第1次正式版本发布。
2023-09-27	02	第2次正式版本发布。 样例图片更新。



目录

前言.....	i
1 基本概念.....	1
2 特性设计.....	3
2.1 应用框图.....	3
2.2 硬件资源.....	4
2.3 API 接口.....	4
3 接口说明.....	6
3.1 创建定时任务.....	6
3.2 启动定时任务.....	7
3.3 停止定时任务.....	7
3.4 删除定时任务.....	7
4 注意事项.....	8
5 样例.....	9
5.1 样例介绍.....	9
5.2 环境准备.....	9
5.3 新建工程.....	9
5.4 应用补丁.....	11
5.5 编译运行.....	12



插图目录

图 1-1 传统 MCU 系统.....	1
图 1-2 多任务系统.....	2
图 2-1 HAL_TIMER 硬件定时器应用框图.....	3
图 2-2 NOS_TimerTask 应用框图.....	4
图 5-1 新建工程示例.....	10
图 5-2 生成代码示例.....	10
图 5-3 编译代码.....	11
图 5-4 修改 SysTick 配置.....	11
图 5-5 新增 main 函数栈配置.....	11
图 5-6 替换 user 目录.....	12
图 5-7 点击重编译按钮.....	12
图 5-8 编译结果.....	12
图 5-9 输出结果开始.....	13
图 5-10 输出结果结束.....	14



表格目录

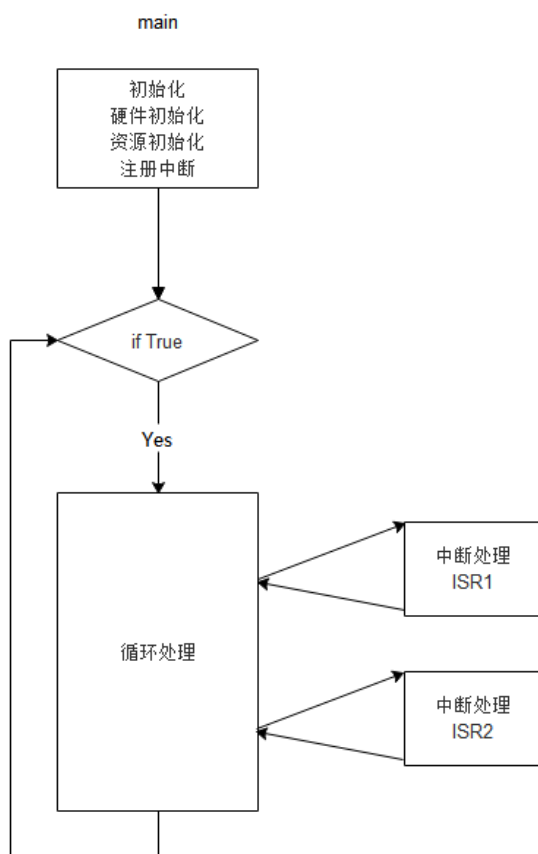
表 2-1 硬件资源对比.....	4
表 2-2 API 接口对比.....	4
表 3-1 timerParam 参数说明.....	6
表 3-2 timerTaskId 参数说明.....	6
表 3-3 taskId 参数说明.....	7



1 基本概念

传统的MCU系统，应用程序由一个无限循环和多个中断服务程序组成，无限循环中调用模块（即函数）来执行所需的操作，中断服务程序负责处理异步事件。这种系统下只有一个无限循环的任务（即main函数）。

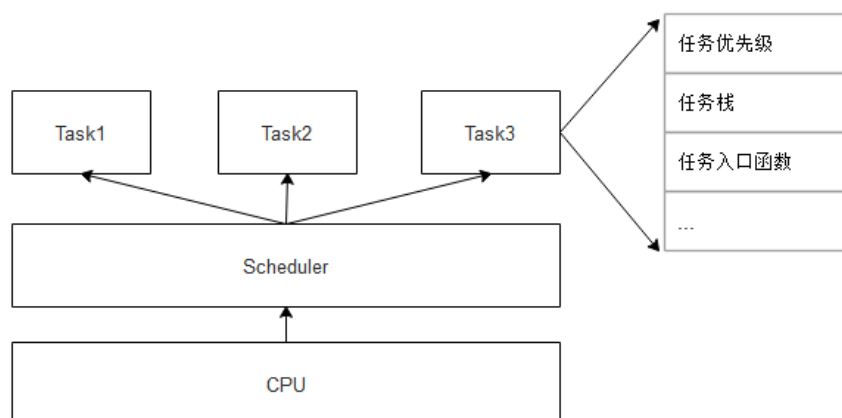
图 1-1 传统 MCU 系统



现代RTOS系统，引入了多任务的概念，任务是占用CPU的最小运行单元，每个任务都有自己的任务入口函数、任务栈和优先级。CPU根据调度器Scheduler的策略在多个任务之间切换执行，调度策略支持基于优先级的抢占式调度，即高优先级任务可以抢占低优先级任务。



图 1-2 多任务系统



本文的定时任务基于多任务系统实现，定时任务包含定时任务入口函数、任务栈、优先级和定时执行时间间隔等属性。启动定时任务后，定时任务入口函数按设定的定时执行时间间隔周期执行。定时任务的实现依赖系统定时器Systick，Systick按照设定的时间周期性产生中断，在Systick中断处理中判断定时任务是否需要调度执行。

系统在初始化时默认会创建一个最低优先级的main任务，main任务的入口函数就是main函数，用户可以在main函数里实现自己的业务代码，可以创建定时任务。系统中如果没有定时任务在运行，则会运行main任务。



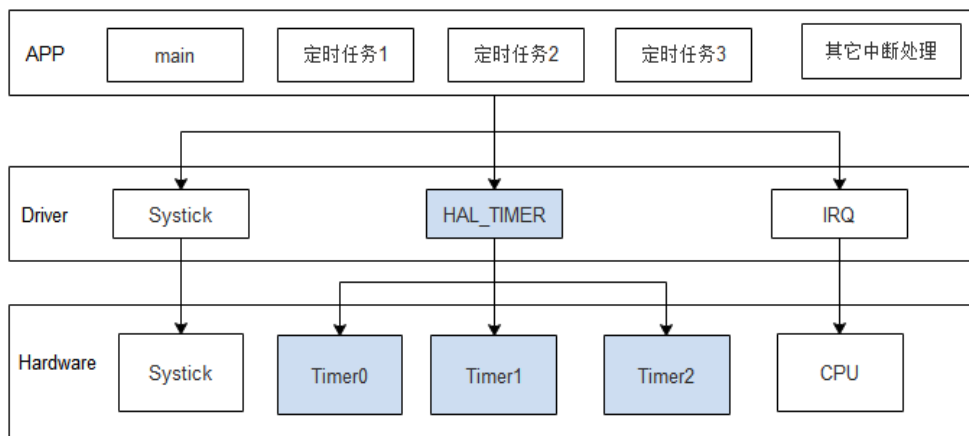
2 特性设计

本章节通过与硬件定时器驱动HAL_TIMER对比来说明NOS_TimerTask定时任务的相关设计。

2.1 应用框图

使用HAL_TIMER 实现定时任务，APP调用Driver层HAL_TIMER接口，分别创建/启动定时器。每创建一个定时器，HAL_TIMER会占用一个硬件Timer资源，在定时器周期到达时刻，触发定时任务执行。高优先级定时任务可以抢占低优先级定时任务。如图2-1所示，若创建3个定时器，则会用到硬件Timer0/1/2。

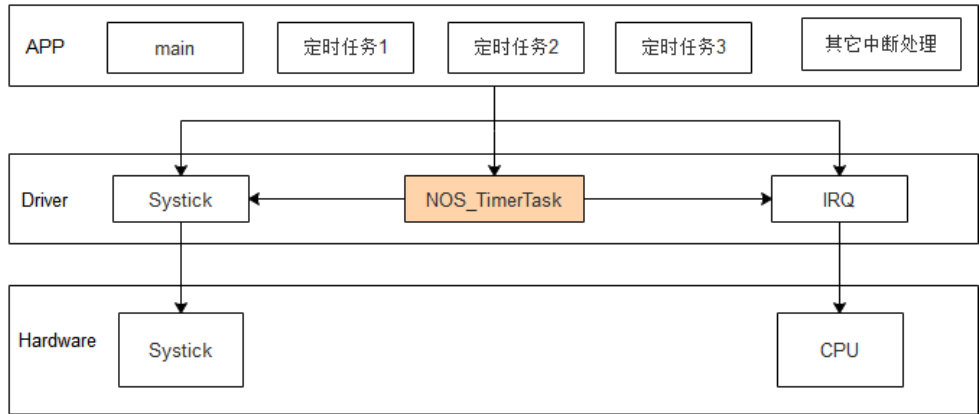
图 2-1 HAL_TIMER 硬件定时器应用框图



系统使用NOS_TimerTask实现定时任务，APP调用NOS_TimerTask接口创建/启动定时任务。在Systick中断处理中判断定时任务是否需要调度执行。高优先级定时任务可以抢占低优先级定时任务。



图 2-2 NOS_TimerTask 应用框图



2.2 硬件资源

如表2-1所示，使用HAL_TIMER硬件定时器，一个定时器任务绑定一个硬件Timer资源。使用NOS_TimerTask，仅使用硬件TIMER3作为Systick Timer，可以同时创建多个定时任务。

表 2-1 硬件资源对比

硬件定时器 编号	使用硬件定时器HAL_TIMER	使用NOS_TimerTask
Timer0	使用	不使用
Timer1	使用	不使用
Timer2	使用	不使用
Timer3	用作Systick计时	用作Systick中断

2.3 API 接口

NOS_TimerTask提供类似于硬件定时器HAL_TIMER的功能接口，接口对应关系如表2-2所示。具体用法可参考sample_timer_task样例。

表 2-2 API 接口对比

NOS_TimerTask	硬件定时器
NOS_CreateTimerTask	HAL_TIMER_Init IRQ_SetPriority IRQ_EnableN
NOS_StartTimerTask	HAL_TIMER_Start



NOS_TimerTask	硬件定时器
NOS_StopTimerTask	IRQ_DisableN HAL_TIMER_IrqClear HAL_TIMER_Stop
NOS_TaskDelete	HAL_TIMER_DeInit



3 接口说明

3.1 创建定时任务

接口: int NOS_CreateTimerTask(unsigned int *timerTaskId,
NOS_TimerTaskInitParam *timerParam);

表 3-1 timerParam 参数说明

参数	输入/输出	含义
const char *name	Input	表示任务名。
unsigned int timeout	Input	任务超时时长。
NOS_TimerCallBack callback	Input	超时回调函数。callback运行时长不能超过timeout。
void *callbackParam	Input	超时任务的参数，callback的入参。
unsigned int priority	Input	任务优先级，取值范围[0,4]，数值越大，优先级越低。
unsigned int stackAddr	Input	任务栈地址，该值不能为0，且需16字节对齐。详细见示例。
unsigned int stackSize	Input	任务栈stackAddr的大小。

表 3-2 timerTaskId 参数说明

参数	输入/输出	含义
unsigned int *timerTaskId	Output	创建的task的ID，用于唯一标识一个任务。



示例:

```
unsigned char __attribute__((aligned(16))) g_task0StackSpace[0x300];
int main()
{
    NOS_TimerTaskInitParam param;
    param.name = "task0";
    param.timeout = 200; // us
    param.callback = Sample_timer0CallbackFunc;
    param.callbackParam = 200;
    param.priority = 1;
    param.stackSize = sizeof(g_task0StackSpace);
    param.stackAddr = g_task0StackSpace;
    int ret = NOS_CreateTimerTask(&pid, &param);
    ...
}
```

3.2 启动定时任务

```
int NOS_StartTimerTask(unsigned int taskId);
```

表 3-3 taskId 参数说明

参数	输入/输出	含义
unsigned int *taskId	Input	创建的task的ID，用于唯一标识一个任务。

启动定时任务后，会将任务添加到定时链表，等待超时调度。

3.3 停止定时任务

```
int NOS_StopTimerTask(unsigned int taskId);
```

停止定时任务后，会将任务从定时链表移除，任务将不会得到调度，但任务实例仍存在，资源未释放。

3.4 删除定时任务

```
int NOS_TaskDelete(unsigned int taskId);
```

删除定时任务后，会将任务实例删除，释放任务所占资源。



4 注意事项

- Systick中断间隔默认为100us，可以通过chip/target/userconfig.json文件，在define中增加CFG_SYSTICK_TICKINTERVAL_US配置TICK的大小；所有定时任务的周期必须是Systick中断间隔的整数倍；启动定时任务后，仅首次超时可能会有误差，误差范围小于1个Systick中断间隔。
- main中根据需要调用NOS_CreateTimerTask创建定时任务，定时任务优先级需要高于main本身（main优先级为NOS_TASK_PRIORITY_LOWEST）。定时任务创建后暂不支持动态修改优先级。
- 当前用户可创建的定时任务数最大支持4个。
- 创建任务时，任务栈空间必须由调用点提供，且栈地址必须16字节对齐。
- 系统启动时会创建任务入口函数为main的主任务MainTask，MainTask栈空间大小可通过CFG_NOS_MAINTASK_STACKSIZE配置。
- 使能NOS_TASK_SUPPORT后，系统中使用的栈分为中断栈和任务栈，ram划分为sram和stack。其中stack预留给中断栈，任务栈从sram中分配。用户可以根据实际需求，调整sram和stack大小。任务栈大小与函数调用深度以及局部变量大小有关，用户应合理分配。
- 定时任务仅支持创建/启动/停止/删除操作。



5 样例

5.1 样例介绍

- 样例在main函数中创建了3个定时器，分别定时100us，200us，1ms。
- 为了与硬件定时器对比，样例代码打开NOS_TASK_SUPPORT宏是NOS_TimerTask的实现方式，关掉NOS_TASK_SUPPORT宏是硬件定时器HAL_TIMER的实现方式。
- 用例在运行100ms后停止所有定时器，打印出三个Timer的触发执行时间，通过时间戳可以看出定时任务是否按照预定的定时间隔触发。

5.2 环境准备

准备SolarA² 1.0.0.2和SolarA² 1.0.0.2_TimerTask_Patch（后文简称补丁包），安装SolarA² 1.0.0.2中的deveco-device-tool。

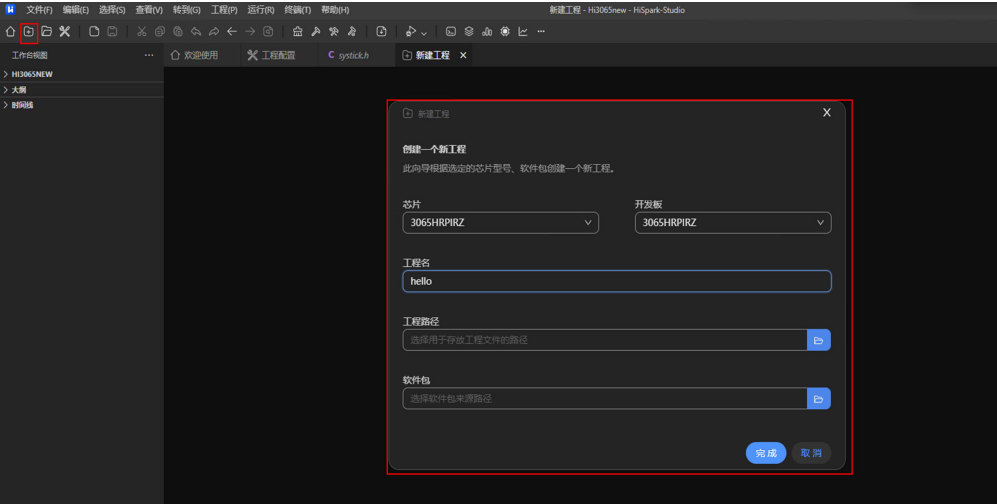
补丁包目录介绍可参考SolarA² 1.0.0.2_TimerTask_Patch/readme.txt。

5.3 新建工程

步骤1 点击新建工程。

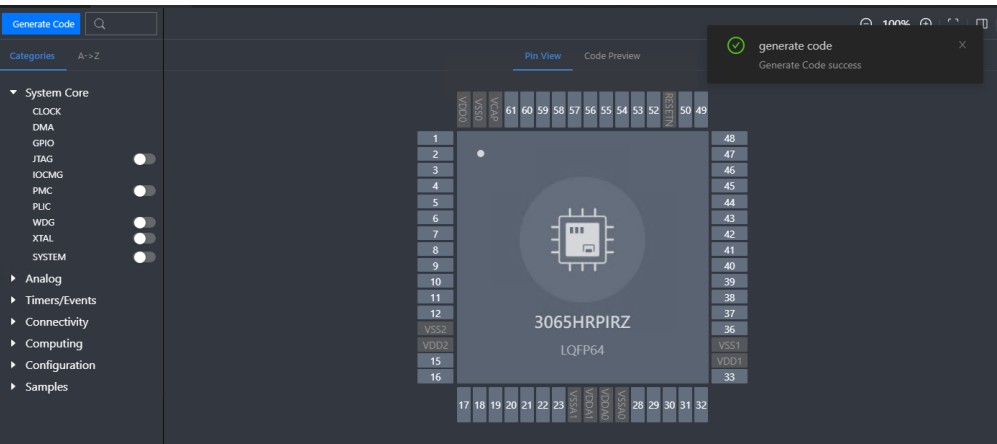


图 5-1 新建工程示例



步骤2 工程名自定义输入，工程路径自行选择，MCU选择3065HRPIRZ（以3065HRPIRZ为例），MCU驱动开发包选择指定的SDK开发包路径。如图 [生成代码示例](#) 所示。

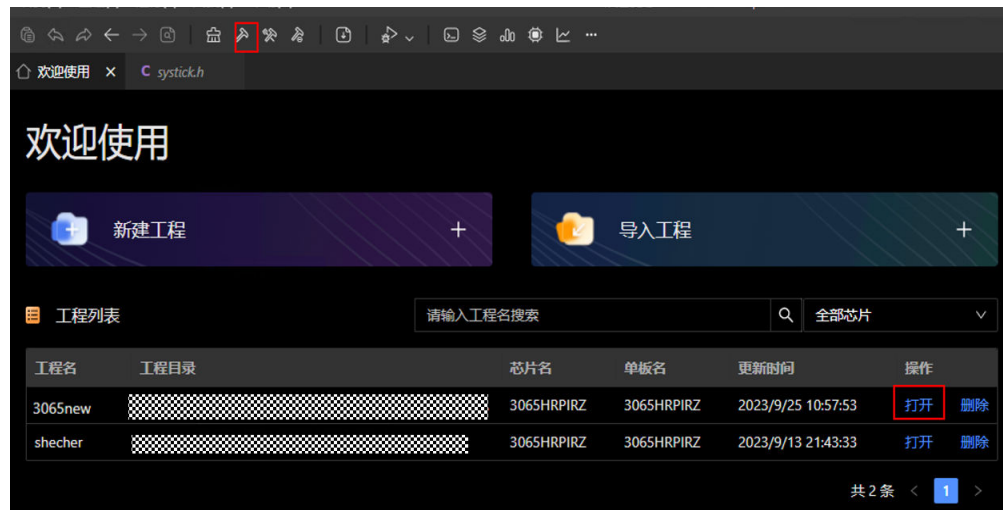
图 5-2 生成代码示例



步骤3 打开工程并编译代码，如图 [编译代码](#) 所示。



图 5-3 编译代码

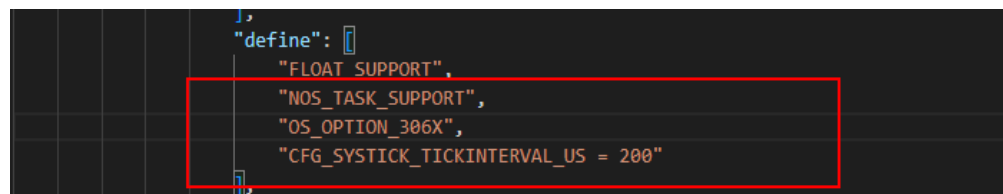


----结束

5.4 应用补丁

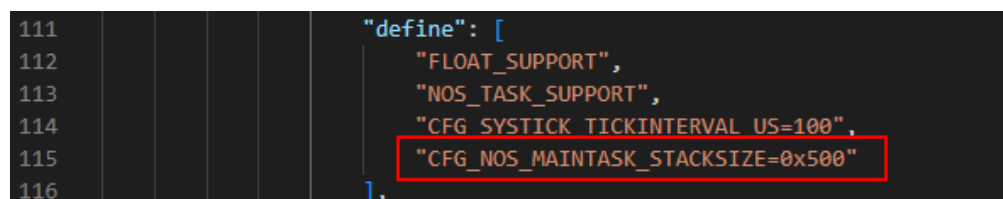
- 步骤1** 修改“chip/target/userconfig.json”文件，在define中增加“NOS_TASK_SUPPORT”，增加“OS_OPTION_306X”，增加“CFG_SYSTICK_TICKINTERVAL_US”支持配置TICK的大小（单位us，默认100us）。如图5-4所示。

图 5-4 修改 SysTick 配置



- 步骤2** 新增“CFG_NOS_MAINTASK_STACKSIZE”支持配置main函数栈大小，如图5-5所示。

图 5-5 新增 main 函数栈配置

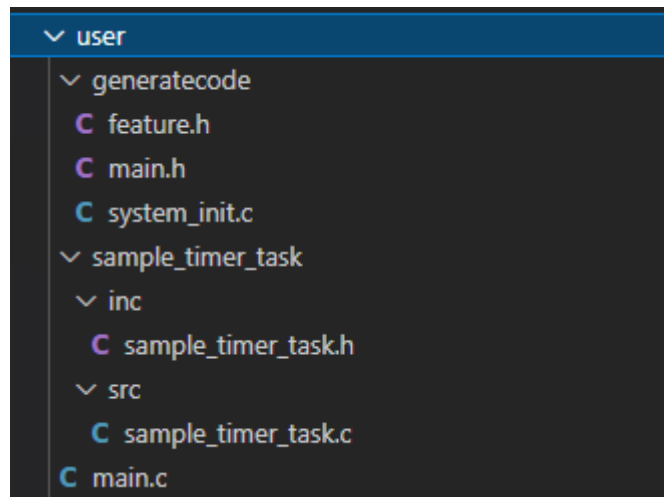


----结束

5.5 编译运行

步骤1 准备sample代码，将user目录下的文件替换为补丁包code/user中的文件。如[图5-6](#)中的文件结构。

图 5-6 替换 user 目录



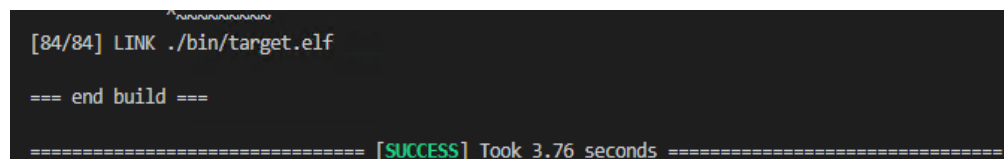
步骤2 点击重编译按钮编译代码。

图 5-7 点击重编译按钮



步骤3 可以在终端中看到编译结果。

图 5-8 编译结果



步骤4 通过调试或烧录的形式观察sample的输出，预期串口输出如[图5-9](#)和[图5-10](#)所示。



图 5-9 输出结果开始

```
*****
Test for: NOS_CreateTimerTask & NOS_StartTimerTask
*****
enter task timeout=100 timestamp=10421us
enter task timeout=100 timestamp=10521us
enter task timeout=200 timestamp=10524us
enter task timeout=100 timestamp=10620us
enter task timeout=100 timestamp=10721us
enter task timeout=200 timestamp=10724us
enter task timeout=100 timestamp=10820us
enter task timeout=100 timestamp=10921us
enter task timeout=200 timestamp=10924us
enter task timeout=100 timestamp=11020us
enter task timeout=100 timestamp=11121us
enter task timeout=200 timestamp=11124us
enter task timeout=100 timestamp=11220us
enter task timeout=100 timestamp=11322us
enter task timeout=200 timestamp=11324us
enter task timeout=1000 timestamp=11327us
enter task timeout=100 timestamp=11421us
enter task timeout=100 timestamp=11521us
enter task timeout=200 timestamp=11524us
enter task timeout=100 timestamp=11620us
```



图 5-10 输出结果结束

```
enter task timeout=100 timestamp=15020us
enter task timeout=100 timestamp=15121us
enter task timeout=200 timestamp=15124us
enter task timeout=100 timestamp=15220us
enter task timeout=100 timestamp=15322us
enter task timeout=200 timestamp=15324us
enter task timeout=1000 timestamp=15327us
enter task timeout=100 timestamp=15421us
enter task timeout=100 timestamp=15521us
enter task timeout=200 timestamp=15524us
enter task timeout=100 timestamp=15620us
enter task timeout=100 timestamp=15721us
enter task timeout=200 timestamp=15724us
enter task timeout=100 timestamp=15820us
enter task timeout=100 timestamp=15921us
enter task timeout=200 timestamp=15924us
enter task timeout=100 timestamp=16020us
enter task timeout=100 timestamp=16121us
enter task timeout=200 timestamp=16124us
enter task timeout=100 timestamp=16220us
enter task timeout=100 timestamp=16322us
enter task timeout=200 timestamp=16324us
enter task timeout=1000 timestamp=16327us
enter task timeout=100 timestamp=16421us
enter task timeout=100 timestamp=16521us
enter task timeout=200 timestamp=16524us
enter task timeout=100 timestamp=16620us
total cnt:100 cnt[100us]:63,cnt[200us]:31, cnt[1000us]:6
```

----结束