VKware AUTOSAR4.3 BSW V1.2

AutosarOs 用户手册

2020.12.18

文档控制

日期	作者	版本	状态	说明
2020.11.03	赵建森	V0.1	【初稿】	
2020.12.18	赵建森	V1.0	【初版】	修订文档格式及内容

目录

1.0 关于本文档 6
1.1 概述 6
1.1.1 目的 6
1.1.2 受众 6
2.0 简介 7
3.0 配置与集成 9
3.1 配置步骤 9
3.1.1 OIL 导入 9
3.1.2 手动配置 10
3.1.3 校验 11
3.1.4 下载 11
3.2 集成 14
3.2.1 基于 Demo 工程 14
3.2.2 新建工程 15
3.2.2.1 IAR 编译器 15
3.2.2.2 HighTec 编译器 15
3.3 OS 占用硬件资源 15
4.0 实例 17
4.1 任务的激活、挂起、切换、终止 17
4.1.1 激活并抢占任务 17
4.1.2 主动调度任务 18
4.1.3 获取任务状态 19
4.1.4 任务的终止 19
4.2 事件同步 19
4.3 中断的抢占、屏蔽、挂起 20
4.3.1 中断的抢占 20
4.3.2 中断屏蔽和挂起 21
4.4 资源的使用 21
4.4.1 标准资源 21
4.4.2 内部资源 22
4.4.3 链接资源 22
4.4.4 调度资源 22
4.5 报警的使用 22

4.5.1 激活任务 22
4.5.2 设置事件 23
4.5.3 执行回调 24
4.5.4 累加计数器 25
4.6 计数器的使用 25
4.7 调度表的使用 26
4.7.1 激活任务 26
4.7.2 设置事件 27
5.0 配置参数介绍 29
5.1 通用配置（General） 29
5.2 钩子（Hooks） 31
5.3 任务（Task） 32
5.4 中断（Isr） 35
5.5 计数器（Counter） 37
5.6 事件（Event） 38
5.7 报警（Alarm） 39
5.8 应用模式（AppMode） 41
5.9 资源(Resource) 42
5.10 调度表（Schedule Table） 43
5.11 外设访问（Peripheral Area） 46
6.0 API 接口 48
6.1 Hook 接口 48
6.1.1 ErrorHook 48
6.1.2 PostTaskHook 48
6.1.3 PreTaskHook 48
6.1.4 StartupHook 49
6.1.5 ShutdownHook 49
6.2 Task 接口 49
6.2.1 ActivateTask 49
6.2.2 TerminateTask 49
6.2.3 ChainTask 50
6.2.4 Schedule 50
6.2.5 GetTaskID 50
6.2.6 GetTaskState 51
6.3 中断接口 51
6.3.1 EnableAllInterrupts 51
6.3.2 DisableAllInterrupts 52

6.3.3 ResumeAllInterrupts....52
6.3.4 SuspendAllInterrupts....52
6.3.5 ResumeOSInterrupts....53
6.3.6 SuspendOSInterrupts....53
6.3.7 GetISRID....53
6.3.8 EnableInterruptSource....53
6.3.9 DisableInterruptSource....54
6.3.10 ClearPendingInterrupt....54
6.4 Event 接口....54
6.4.1 SetEvent....54
6.4.2 ClearEvent....55
6.4.3 GetEvent....55
6.4.4 WaitEvent....56
6.5 Alarm 接口....56
6.5.1 GetAlarmBase....56
6.5.2 GetAlarm....57
6.5.3 SetRelAlarm....57
6.5.4 SetAbsAlarm....58
6.5.5 CancelAlarm....58
6.6 Resource 接口....59
6.6.1 GetResource....59
6.6.2 ReleaseResource....59
6.7 Counter 接口....59
6.7.1 IncrementCounter....59
6.7.2 GetCounterValue....60
6.7.3 GetElapsedValue....60
6.8 Schedule Table 接口....60
6.8.1 StartScheduleTableRel....60
6.8.2 StartScheduleTableAbs....61
6.8.3 StopScheduleTable....61
6.8.4 NextScheduleTable....61
6.8.5 GetScheduleTableStatus....62
6.9 Peripheral Area 接口....62
6.9.1 ReadPeripheral8....62
6.9.2 WritePeripheral8....62
6.9.3 ModifyPeripheral8....63
6.10 系统控制接口....63
6.10.1 StartOS....63
6.10.2 ShutdownOS....64

6.10.3 GetActiveApplicationMode.....64
6.11 扩展接口.....64
6.11.1 IdleHook.....64
6.11.2 GetTaskStackUsage.....64
6.11.3 GetISR2StackUsage.....65
6.11.4 GetSystemStackUsage.....65

1.0 关于本文档

1.1 概述

1.1.1 目的

本文档旨在引导用户从零开始入手使用 *VKware* 的 *AUTOSAR OS线上工具系统* 及代码使用。

AUTOSAR规范版本：4.3

VKware BSW代码版本：V1.2

其他参考文档：

《VKware线上系统用户手册.docx》

《VKware AUTOSAR OS培训课程.pptx》

1.1.2 受众

本文档仅限于基于VKware线上工具系统做开发的工程人员。

2.0 简介

VKware OS是专门为汽车电子领域设计的可抢占、多任务、高性能、资源消耗小、高度可裁剪的嵌入式实时操作系统。操作系统软件发布版V1.2，涵盖OSEK和SC1。有以下特性：

- 系统符合AUTOSAR4.3 标准且兼容OSEK/VDX OS 2.2.3 标准；
- 系统静态配置，在系统运行过程中不允许对任何配置进行更改；
- 最大支持 4095 个任务，可配置任务可抢占或不可抢占；
- 支持中断嵌套，提供一类和二类中断，满足不同实时性应用需求；
- 提供堆栈溢出和堆栈使用率检测；
- 提供CPU负载率检测；
- 提供核内资源互斥访问机制；
- 提供软硬件计数器功能；
- 支持周期警报、调度表；
- 提供外设地址访问接口。

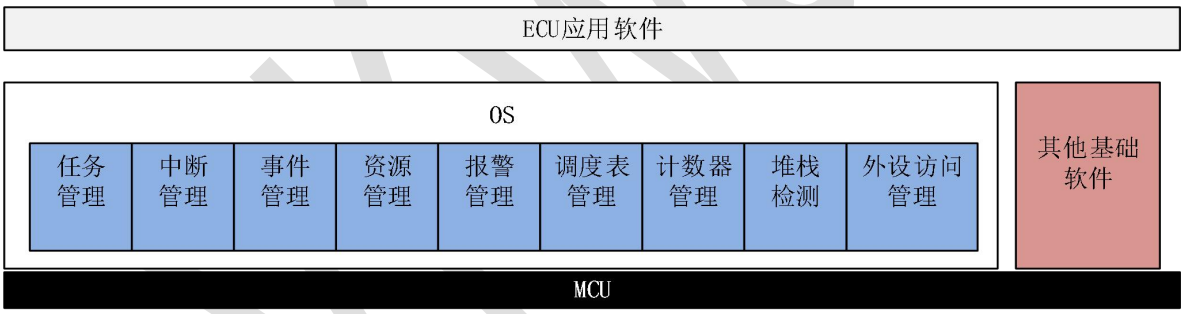


图2-1 VKware OS 软件层次结构

各个部分的功能如下：

1．任务管理：管理扩展任务、基本任务等的激活、结束、重调度，以及任务信息的获取功能。

2．中断管理：全局中断、操作系统中断的使能和禁止，中断源的控制功能。

3．事件管理：是一种任务间的同步机制，不是独立的对象，必须依赖任务。事件管理主要负责实现事件的发送、等待、查询、清除功能。

4．资源管理：此处的资源是一种用于资源的互斥访问的手段，资源管理主要实现资源的获取、释放功能。

5．计数器管理：计数器的计数，计数值获取，以及驱动警报和调度表工作。

6．警报管理：定时响应功能，在预定时间到达时触发相关的操作，如：设置事件、激活任务、进行回调等操作。

7．调度表管理：实现调度表的启动、停止、切换、同步、异步功能，以及在设置EP 点到达时触发指定操作。

8．堆栈检测：堆栈溢出检测，获取堆栈使用率功能。

9．CPU负载检测：实时测量CPU的使用率，该功能依赖于Tm模块。

10．外设访问管理：为外设地址访问提供 8 位、16 位、32 位的读、写、修改接口。

3.0 配置与集成

3.1 配置步骤

登录 *VKware 线上工具系统* 并新建配置项目，创建项目参考 *《VKware 线上系统用户手册.docx》*。项目配置可以选择导入 OIL 或全手动配置或导入 OIL 后手动配置。

3.1.1 OIL 导入

在 ECUC 配置页面点击“上传 oil 文件”，选中本地需要上传的 oil 文件，并点击打开。然后执行如下操作：

1．点击如图 3-1 所示“确认配置”，等待页面弹出“oil 配置文件提交成功，配置已更新！”，表示上传 oil 配置文件成功；

2．点击如图 3-1 所示“ ”按钮，校验上传的 oil 配置是否有配置错误，包括必配数据的丢失、关联目标错误、数据超出范围等。若有错误项，会在弹出的错误日志窗口中显示。

图 3-1 通过 oil 文件生成配置文件示例

用户可以根据需求进入 System Services/OS 中手动修改、增加、删除配置。

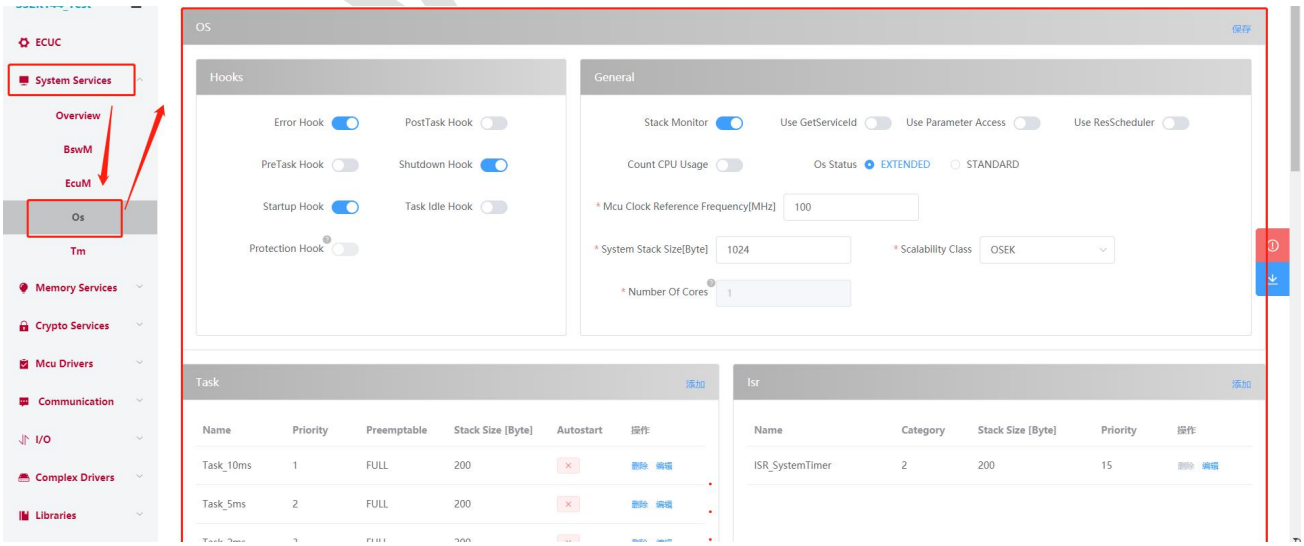


图 3-2 Os 配置界面

3.1.2 手动配置

在新建配置项目时，工具系统会自动为 OS 模块添加默认配置，内容如下。

基础配置项：

扩展类：OS_OSEK

系统状态：OS_STATUS_EXTENDED

堆栈检查：TRUE

应用模式：OSDEFAULTAPPMODE

系统堆栈：1024（Byte）

Mcu 时钟频率：100（MHz）

表 3-1 自动生成任务配置

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	65535	1	模式自启动 OSDEFAULTAPPMODE	200
Task_2ms	FULL	3	1	非自启动	200
Task_5ms	FULL	2	1	非自启动	200
Task_10ms	FULL	1	1	非自启动	200

表 3-2 自动生成警报配置

警报（ID）	自启动	驱动计数器	周期（Tick）	到期动作	自启动时间（Tick）
Alarm_2ms	TRUE	SystemTimer	2	激活 Task_2ms 任务	1
Alarm_5ms	TRUE	SystemTimer	5	激活 Task_5ms 任务	2
Alarm_10ms	TRUE	SystemTimer	10	激活 Task_10ms 任务	3

表 3-3 自动生成计数器配置

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	1ms

用户可根据自己的需求修改 Os 的配置，配置参数详细说明见第 5 章配置参数介绍。

当所选平台为多核时，OS 系统即可配置为单核操作系统，也可配置为多核操作系统。当 OS 为单核操作系统时，需要指定 OS 映射的物理核，即在 ECUC/Cores Definition 下选择 Master Core。如下图 3-3 所示。

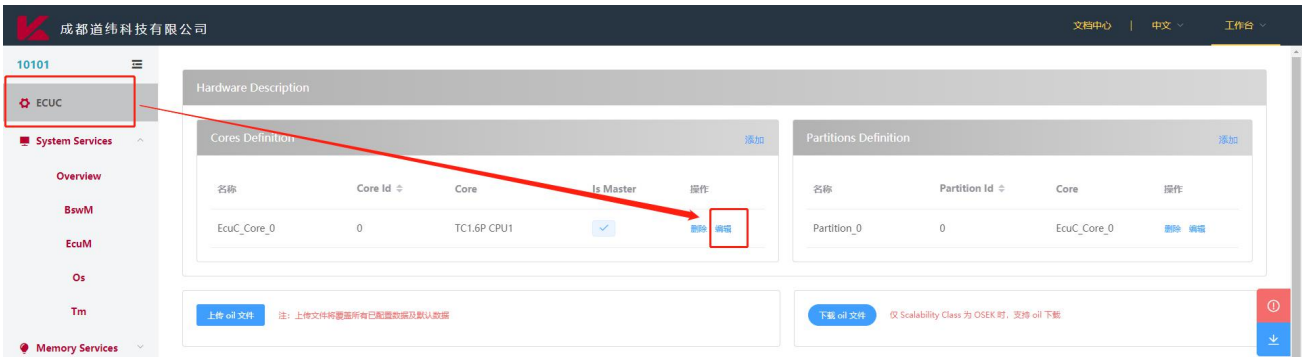


图 3-2 masterCore 配置界面

3.1.3 校验

用户配置完成后，可以点击“ ”校验配置内容是否正确，但不会生成配置文件与提取源码文件。

配置内容错误信息将在错误日志页面显示，每个配置错误会有一条配置错误信息描述，以及该错误配置在 OS 配置中的位置，以便查找更正。

图 3-3 校验错误日志

3.1.4 下载

用户配置完成后，配置内容没有错误，才能正常下载生成代码。在点击“generate”时，工具系统也会先校验配置内容是否正确，配置正确后，才会生成配置文件与提取源码文件。若有配置错误，错误信息也将在错误日志页面显示。

OS 生成文件列表如下表所示：

表 3-4 源码提取需求

OS 编译器、芯片无关源文件生成	
源文件	**生成条件**
Os.c	无

Os.h	无
Os_Alarm.c	OsAlarm 对象个数不为 0
Os_Application.c	OsApplication 对象个数不为 0
Os_Counter.c	OsCounter 对象个数不为 0
Os_CpuUsage.c	CountCPUUsage 为 TRUE
Os_Event.c	OsEvent 对象个数不为 0
Os_ExtCheck.c	OsStatus 值为 EXTENDED
Os_Hook.c	OsHooks 容器下任意 Hook 为 TRUE
Os_Internal.h	无
Os_Interrupt.c	OsIsr 对象个数不为 0
Os_Ioc.c	OsIoc 对象个数不为 0
Os_MemProt.c	OsScalabilityClass 为 SC3 或 SC4
Os_MultiCore.c	OsNumberOfCores 值大于 1
Os_Peripheral.c	OsPeripheralArea 对象个数不为 0
Os_Resource.c	OsResource 对象个数不为 0
Os_SchedTable.c	OsScheduleTable 对象个数不为 0
Os_SerProt.c	OsScalabilityClass 为 SC3 或 SC4
Os_Spinlock.c	OsSpinlock 对象个数不为 0
Os_Stack.c	OsStackMonitoring 为 TRUE
Os_Task.c	OsTask 对象个数不为 0
Os_TimProt.c	任意 task 或 isr 配置了时间保护
Os_Types.h	无

OS 芯片相关源文件生成	
芯片	**生成文件**
CYT2B9	Os_Platform.c
	Os_Platform.h

	Os_VKX_CM0_Port.c
	Os_VKX_CM0_Port.h
	Os_VKX_CM4_Port.c
	Os_VKX_CM4_Port.h
S32K144	Os_Platform.h
	Os_VKX_CM4_Port.c
	Os_VKX_CM4_Port.h
TC233	Os_Platform.h
	Os_VKX_TC23x_Port.c
	Os_VKX_TC23x_Port.h
TC275	Os_Platform.h
	Os_VKX_TC27x_Port.c
	Os_VKX_TC27x_Port.h

依赖文件生成	
源文件	**生成条件**
Compiler.h	根据配置工程选择的编译器提取
Platform_Types.h	根据配置工程选择的芯片提取
Std_Types.h	无

配置文件生成	
配置文件	生成文件：Os_Cfg.c、Os_Cfg.h、Os_CPUx_Cfg.c
Rte 集成	生成文件：Rte_Os.c
其他	生成文件：MemMap.h、Compiler_Cfg.h、Os_MemMap.h、Rte_MemMap.h、启动代码、链接脚本

3.2 集成

3.2.1 基于Demo 工程

Demo 工程可以在成都道纬科技有限公司官网获取：https://www.vkware.com.cn/onlinevk。如下图 3-4 所示。

图 3-4 Demo 工程下载

集成步骤：

1．下载 Demo；

2．工具系统生成代码；

3．将工具系统生成代码中“ config/Os_Cfg.* ”和“ config/Os_XXX_Cfg.c ”文件替换到 Demo 工程”src/Generated”下；

4．将工具系统生成代码中”config/Rte_Os.c“或”config/Rte_Os_XXX.c“文件替换到 Demo 工程 ”src/Rte_Service_SWCs”下；

5．用户根据自己的需求在“Rte_Service_SWCs”文件中的“Rte_Os.c”或“Rte_Os_XXX.c”下进行代码集成，包含了用户配置的任务函数、中断处理函数、Hook 函数等，用户只需将应用代码集成到合适的函数里即可，OS 内核会自行调度并执行这些函数体。

注：在进行 Demo 集成时请注意使用的 Demo 版本号应该和工具新建工程时的版本号对应。

6．官网中的 Demo 版本为 VKware BSW V1.2；

7．工具系统中 AUTOSAR4.3 版本对应的是 VKware BSW V1.2 版本；

8．Demo 中的启动代码、链接脚本是针对 AUTOSAR OS 设计的，用户可以根据需求自行修改。

3.2.2 新建工程

3.2.2.1 IAR 编译器

1．新建工程；

2．工具系统生成代码；

3．将工具系统生成的代码放到新建的工程中；

4．将此工程 IAR 的链接脚本替换为工具系统生成的“config/linker_XXX.icf”；

5．在工程包含 main 函数的文件中包含头文件"Os.h"，并在 mian 函数中添加“StartOS(OSDEFAULTAPPMODE)”启动 OS；

6．用户根据自己的需求在工程“config”文件中的“Rte_Os.c”或“Rte_Os_XXX.c”下进行代码集成，包含了用户配置的任务函数、中断处理函数、Hook 函数等，用户只需将应用代码集成到合适的函数里即可，OS 内核会自行调度并执行这些函数体。

注：用户新建工程时请使用 VKware 工具系统生成的启动代码和链接脚本文件，启动代码和链接脚本是针对 AUTOSAR OS 设计的，用户可以根据需求自行修改。

在启动代码中主要做了中断向量表的初始化。在多核芯片平台，链接脚本主要实现了多核映射。

3.2.2.2 HighTec 编译器

1．新建工程；

2．工具系统生成代码；

3．将工具系统生成的代码放到新建的工程中；

4．将此工程的链接脚本替换为工具系统生成的“config/OsLink.ld”；

5．在工程包含 main 函数的文件中包含头文件"Os.h"，并在 mian 函数中添加“StartOS(OSDEFAULTAPPMODE)”启动 OS；

6．用户根据自己的需求在工程“config”文件中的“Rte_Os.c”或“Rte_Os_XXX.c”下进行代码集成，包含了用户配置的任务函数、中断处理函数、Hook 函数等，用户只需将应用代码集成到合适的函数里即可，OS 内核会自行调度并执行这些函数体。

注：用户新建工程时请使用 VKware 工具系统生成的链接脚本文件，该链接脚本是针对 AUTOSAR OS 设计的，用户可以根据需求自行修改。在多核芯片平台，链接脚本主要实现了多核映射。

3.3 OS 占用硬件资源

硬件平台	占用资源	描述
TC275	OS_SRC_STM0_SR0	Core0 的系统时钟，ISR_SystemTimer 或 ISR_SystemTimer_Core0
	OS_SRC_STM1_SR0	Core1 的系统时钟，ISR_SystemTimer_Core1
	OS_SRC_STM2_SR0	Core2 的系统时钟，ISR_SystemTimer_Core2

	OS_SRC_STM0_SR1	Core0 的时间保护，ISR_TimprotTimer 或 ISR_TimprotTimer_Core0
	OS_SRC_STM1_SR1	Core1 的时间保护，ISR_TimprotTimer_Core1
	OS_SRC_STM2_SR1	Core2 的时间保护，ISR_TimprotTimer_Core2
	OS_SRC_GPSR0_SR0	Core0 的跨核中断，ISR_RemoteCall_Core0
	OS_SRC_GPSR0_SR1	Core1 的跨核中断，ISR_RemoteCall_Core1
	OS_SRC_GPSR0_SR2	Core2 的跨核中断，ISR_RemoteCall_Core2
TC233	OS_SRC_STM0_SR0	系统时钟，ISR_SystemTimer
	OS_SRC_STM0_SR1	时间保护，ISR_TimprotTimer
CYT2B9	OS_SysTick_IRQn	系统时钟，ISR_SystemTimer
	tcpwm_0_interrupts_0_IRQn	CM4 的时间保护，ISR_TimprotTimer 或 ISR_TimprotTimer_Core0
	tcpwm_0_interrupts_1_IRQn	CM0 的时间保护，ISR_TimprotTimer_Core1
	cpuss_interrupts_ipc_5_IRQn	CM4 的跨核中断，ISR_RemoteCall_Core0：
	cpuss_interrupts_ipc_6_IRQn	CM0 的跨核中断，ISR_RemoteCall_Core1：
S32K144	OS_SysTick_IRQn	系统时钟，ISR_SystemTimer
	OS_LPIT0_Ch0_IRQn	时间保护，ISR_TimprotTimer

4.0 实例

4.1 任务的激活、挂起、切换、终止

本实例主要演示如何将一个任务从挂起状态转变为运行状态、从一个任务切换为另一个任务、查看当前任务的状态或 ID、被高优先级任务抢占、终止任务等。

基础配置项：

扩展类：OS_OSEK

系统状态：OS_STATUS_EXTENDED

堆栈检查：TRUE

使用调度资源：TRUE

应用模式：OSDEFAULTAPPMODE

系统堆栈：1024

4.1.1 激活并抢占任务

配置参数：

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	2	1	模式自启动 OSDEFAULTAPPMODE	200
Task_1	FULL	3	1	非自启动	200

使用服务 ActivateTask(TaskID)或 ChainTask(TaskID)可以激活任务（从挂起状态 SUSPENDED 切换到就绪状态 READY）。

- 使用系统服务 ActivateTask(TaskID)：

  1．OS 启动后激活并运行自启动任务 Task_Init；

  2．在 Task_Init 中调用 ActivateTask(Task_1)；

  ActivateTask(Task_1)将会激活 Task_1，由于 Task_1 的优先级高于 Task_Init，所以立即切换到 Task_1 运行。此时 Task_Init 将会被置于就绪队列等待执行，其上下文压栈保存于 Task_Init 的私有堆栈中。

  3．Task_1 运行结束，再次回到 Task_Init 中运行。

- 使用系统服务 ChainTask(TaskID)：

  1．OS 启动后激活并运行自启动任务 Task_Init；

  2．在 Task_Init 中调用 ChainTask(Task_1)；

ChainTask(Task_1)同样会激活 Task_1 并立即切换到 Task_1 中运行，但是这里与这两个任务的优先级没有关系，在 Task_Init 中调用 ChainTask(Task_1)将会直接结束 Task_Init 的运行，等 Task_1 结束后并不会回到 Task_Init 中。

注：这里的参数 TaskID 直接引用配置界面中 Task 名称，在 Os_Cfg.h 定义。

4.1.2 主动调度任务

切换任务有多种方式，可以通过 4.1.1 章节的方式，也可以通过其他方式。当一个任务占据资源，用 4.1.1 章节的方式去切换任务将会失败。这里主要演示当一个 Task 长时间占据内部资源，一个高优先级任务想要抢占他，使用 Schedule()主动调的情况。

配置参数:

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）	资源
Task_Init	FULL	2	1	模式自启动 OSDEFAULTAPPMODE	200	Resource_0
Task_1	FULL	3	1	非自启动	200	Resource_0

资源（ID）	资源属性	被引用对象
Resource_0	内部资源	Task_Init

1．OS 启动后激活并运行自启动任务 Task_Init；

2．Task_Init 启动时自动获取内部资源 Resource_0（内部资源在所属任务启动时自动获取）；

3．在 Task_Init 中调用 ActivateTask(Task_1)；

Task_1 被激活，并将 Task_1 置入就绪队列。尽管 Task_1 配置优先级高于 Task_Init，但由于 Task_Init 获取内部资源 Resource_0 后，Task_Init 的优先级被置为 Resource_0 天花板优先级，Resource_0 天花板优先级为 3，因此此时的 Task_1 没法抢占 Task_Init。

4．调用 Schedule()；

5．切换到 Task_1 中运行，Task_Init 置入就绪队列。

调用 Schedule()后，将释放 Task_Init 占用的内部资源 Resource_0，Task_Init 恢复到原来的优先级。然后 OS 会执行一次任务调度器，由于此时 Task_1 处于就绪队列且优先级为最高，因此系统将切换到 Task_1 运行，等待 Task_1 运行结束后，再回到 Task_Init 继续运行。

注：通过以上步骤可以看出，当任务没有内部资源时，执行 Schedule()没有任何意义。

4.1.3 获取任务状态

查看指定任务的当前状态，可以通过调用 GetTaskState(TaskID, &State)，将想要查询的 TaskID 传入此 API 参数中，返回时将会把此 Task 的当前状态存入 State 中。

4.1.4 任务的终止

使用系统服务函数 ChainTask(TaskID)或 TerminateTask(void)均可结束当前运行的任务。

ChainTask(TaskID)结束当前运行任务，并激活 TaskID 表示的任务。只能在任务中使用且使用前必需释放任务所占用的资源。

TerminateTask(void)结束当前运行任务，只能在任务中使用且使用前必需释放该任务所占用的资源。

4.2 事件同步

扩展任务可等待某个事件触发后再运行，从而实现同步的效果。事件是依赖于任务的一种同步机制，因此发送（即设置）事件时要指明给哪个任务设置该事件。

本示例演示如何等待事件、发送事件、获取事件、清除事件操作。

配置参数:

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）	事件
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200	Event_0
Task_1	FULL	4	1	非自启动	200	无

事件（ID）	有访问权限的任务
Event_0	Task_Init

1．OS 启动后激活并运行自启动任务 Task_Init；

2．在 Task_Init 中调用 ActivateTask(Task_1)；

Task_1 的优先级低于 Task_Init 的优先级，Task_Init 不会被抢占，继续执行；

3．GetEvent（Task_Init, event_mask）,event_mask == 0，说明 Task_Init 当前没有任何事件被设置；

4．WaitEvent（Event_0）；

当调用 WaitEvent（Event_0）时，Task_Init 将进入 waiting 状态，不会结束当前任务，直到等待的事件发生，才会再次进入 ready 状态，参与调度。

Task_Init 进入 waiting 状态后，系统将会重新执行调度，Task_1 将会运行。

5．在 Task_1 中，执行 SetEvent(Task_Init，Event_0)；

执行 SetEvent(Task_Init，Event_0)，注意，是为任务 Task_Init 设置事件 Event_0，而不是其他任务。因此该操作仅影响 Task_Init，即使此时有其他任务也在等待 Event_0，也不会受该操作影响。

Task_Init 从 waiting 状态切换到 ready 状态，被置于就绪队列，系统重新执行调度，Task_1 的优先级低于 Task_Init 的优先级，所以系统切换回 Task_Init 中继续运行，Task_1 此时被置于就绪队列。此处即实现了事件 Event_0 和任务 Task_Init 的执行的同步。

6．在 Task_Init 中，调用 GetEvent（Task_Init, &event_mask），获取 Task_Init 当前已设置的事件，此时 event_mask = Event_0;

7．调用 ClearEvent（Event_0）清除 Task_Init 的事件 Event_0。

4.3 中断的抢占、屏蔽、挂起

AUTOSAR OS 将中断分为一类中断和二类中断。一类中断不能使用 OS 系统服务，且优先级高于二类中断优先级。

4.3.1 中断的抢占

配置参数:

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200

中断（ID）	中断类别	优先级	中断嵌套	堆栈大小（Bytes）
OS_SoftwareInterrupt_0	二类	1	TRUE	200
OS_SoftwareInterrupt_1	一类	10	FALSE	

1．OS 启动后激活并运行自启动任务 Task_Init;

2．将 OS_SoftwareInterrupt_0 使能，并触发 OS_SoftwareInterrupt_0;

此时 Task_Init 将被置入就绪队列，将立即跳转到 ISR_SoftwareInterrupt_0 中断处理函数中执行。

3．在 ISR_SoftwareInterrupt_0 中断处理函数中将 OS_SoftwareInterrupt_1 使能，并触发 OS_SoftwareInterrupt_1;

此时由于 OS_SoftwareInterrupt_0 允许嵌套，且 OS_SoftwareInterrupt_1 为一类中断，优先级高于 OS_SoftwareInterrupt_0，将抢占 ISR_SoftwareInterrupt_0，ISR_SoftwareInterrupt_0 将被挂起,立即跳转到 ISR_SoftwareInterrupt_1 中断处理函数执行；

4．ISR_SoftwareInterrupt_1 执行结束，返回 ISR_SoftwareInterrupt_0；

5．ISR_SoftwareInterrupt_0 执行结束，返回 Task_Init 继续执行。

4.3.2 中断屏蔽和挂起

某些代码的执行，为了保证数据的一致性、代码执行的连续性，是不希望在执行过程中被中断打断的，因此需要对这段代码的执行加上临界区保护。

通过调用 DisableAllInterrupts 来禁止所有中断服务响应，并关闭所有硬件中断。DisableAllInterrupts 和 EnableAllInterrupts 必须配对使用（禁止后必须使能），并且在受保护的区域不允许调用系统 API。

使用 DisableAllInterrupts 直接禁止所有中断，不会保存此时被挂起的中断。但是某些时候被挂起的中断可能是我们所需要的，也可能整个过程仅发生一次，而刚好被禁止掉，这不是我们所期望的结果。

使用 SuspendAllInterrupts 禁止所有中断。在禁止所有中断响应的同时，会将挂起的中断及状态保持下来。当执行 ResumeAllInterrupts 后，使能所有中断响应，同时恢复被挂起的中断。

使用 SuspendOSInterrupts 禁止所有二类中断。在禁止所有二类中断响应的同时，会将挂起的中断及状态保持下来，在受保护期间一类中断可以正常响应。当执行 ResumeOSInterrupts 后，使能所有二类中断响应，同时恢复被挂起的中断。

使用 DisableInterruptSource(IsrID)可以关闭单个指定的中断源。使用 EnableInterruptSource（IsrID，ClearPending）可再次使能该中断源，同时可选择是否需要清除该中断源之前未响应的中断触发。

4.4 资源的使用

资源管理是用于不同优先级的任务/中断对共享“资源”的互斥访问，例如：任务调度器、程序片段、内存或者硬件区域等。

4.4.1 标准资源

配置参数

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）	引用的资源
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200	Resource_0

资源（ID）	类型
Resource_0	标准资源

1、OS 启动后激活并运行自启动任务 Task_Init；

2、调用 GetResource（Resource_0）；

Task_Init 获得资源 Resource_0，同时 Task_Init 优先级上升到 Resource_0 的天花板优先级。在这期间如果有其他任务/中断想要获取 Resource_0 将会失败。

3、互斥内容的访问；

4、调用 ReleaseResource(Resource_0)；

Task_Init 释放资源 Resource_0，同时优先级变回获取 Resource_0 时的优先级。此时其他任务/中断才可以再次获取该资源。

4.4.2 内部资源

内部资源不需要调用系统服务（GetResource、ReleaseResource）获取和释放，调用系统服务访问内部资源将会报错 E_OS_ID。

内部资源在任务变为 running 状态时自动获取，变为非 running 状态时自动释放。

内部资源只能被任务访问，一个任务最多只有一个内部资源。

4.4.3 链接资源

链接资源相当于标准资源或内部资源的别名，链接到一个标准资源或内部资源，其属性由链接的资源赋予，但其仍然是一个独立的资源。

4.4.4 调度资源

在 AUTOSAR OS 中，任务调度器也被定义为一种资源（RES_SCHEDULER），归属于标准资源类。

调度资源可以被所有任务访问，当任务获取调度资源后，表示 OS 系统的任务调度器被该任务占用，此时整个 OS 系统都不能执行任务调度，直到该任务释放调度资源。

4.5 报警的使用

Alarm 主要实现报警功能，在预定时间到达时触发相关的操作：激活任务、设置事件、执行回调、累加指定计数器。

4.5.1 激活任务

配置参数

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200
Task_1	FULL	6	1	非自启动	200

警报（ID）	自启动	驱动计数器	到期动作
Alarm_5ms	FALSE	SystemTimer	激活 Task_1 任务

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	1ms

1、OS 启动后激活并运行自启动任务 Task_Init；

2、执行 SetRelAlarm(Alarm_5ms, MS2TICK(5), 0);

SetRelAlarm 使用见第 6 章 Alarm 接口描述。相对启动是从当前开始的时间往后进行偏移，此处调用，increment 为 MS2TICK(5)表示从当前开始往后偏移 5ms 的时间启动 Alarm_5ms，cycle 为 0 表示 Alarm_5ms 不周期执行。

3、等待 5ms；

4、Alarm_5ms 到期，执行 ActivateTask(Task_1)；

5、切换到 Task_1 运行；

Task_1 的优先级高于 Task_Init，的优先级，因此激活后的 Task_1 抢占 Task_Init，Task_Init 被置入就绪队列等待。

6、Task_1 运行结束，回到 Task_Init。

4.5.2 设置事件

配置参数

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）	事件
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200	Event_0

警报（ID）	自启动	驱动计数器	到期动作
Alarm_10ms	FALSE	SystemTimer	为 Task_Init 设置 Event_0

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	2ms

事件（ID）
Event_0

1、OS 启动后激活并运行自启动任务 Task_Init；

2、GetEvent（Task_Init, &event_mask）,event_mask == 0，说明 Task_Init 当前没有任何事件被设置;

3、SetAbsAlarm（Alarm_10ms, MS2TICK(10), 0）；

SetAbsAlarm 使用见第 6 章 Alarm 接口描述。绝对启动是当驱动 Counter 的计数值与 start 相等，此处调用，start 为 MS2TICK(10) 表示当 SystemTimer 计时为 10ms 时启动 Alarm_10ms，cycle 为 0 表示 Alarm_10ms 不周期执行。

注：本示例 1 Tick=2ms，10ms 即 5 Tick。当 SystemTimer 的值等于 5 时，启动 Alarm_10ms。

4、Alarm_10ms 到期，执行 SetEvent(Task_Init，Event_0)；

5、调用 GetEvent（Task_Init, &event_mask）获取当前设置的事件，event_mask = Event_0，设置 Event_0 成功。

4.5.3 执行回调

配置参数

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200

警报（ID）	自启动	驱动计数器	周期（Tick）	到期动作	自启动时间（Tick）
Alarm_10ms	TRUE	SystemTimer	10	调用回调函数 VK_CALLBACK	2

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	1ms

1、 OS 启动后激活并运行自启动任务 Task_Init；

2、 循环等待，当 SystemTimer 的值等于 2 时，启动 Alarm_10ms；

3、 启动 Alarm_10ms 时，会执行一次到期动作，调用回调函数 VK_CALLBACK；

4、 从 2 Tick 开始，往后每隔 10 个 Tick，Alarm_10ms 到期一次，调用一次回调函数 VK_CALLBACK。

4.5.4 累加计数器

配置参数:

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200

警报（ID）	自启动	驱动计数器	周期（Tick）	到期动作	自启动时间（Tick）
Alarm_10ms	TRUE	SystemTimer	10	INCREMENTCOUNTER （Counter_0）	10

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	1ms
Counter_0	软件计数器	100000	10ms

1、OS 启动后激活并运行自启动任务 Task_Init；

2、循环等待，当 SystemTimer 的值等于 10 时，启动 Alarm_10ms；

3、启动 Alarm_10ms 时，会执行一次到期动作，Counter_0 累加 1；

4、从 10 Tick 开始，往后每隔 10 个 Tick，Alarm_10ms 到期一次，执行 Counter_0 计数累加 1。

4.6 计数器的使用

本示例主要演示利用 SystemTimer 实现定时功能。

配置参数

Scalability Class = SC1

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	5	1	模式自启动 OSDEFAULTAPPMODE	200

计数器（ID）	类型	最大计数值	最小 tick 值	Ticks PerBase	1Tick 时间
SystemTimer	硬件计数器	100000	1	1	1ms

定义 ConsValue 变量保存期望的定时时间，假设为 100ms。

1、OS 启动后激活并运行自启动任务 Task_Init；

2、获取 SystemTimer 当前值：调用 GetCounterValue(SystemTimer, &Value)，将 SystemTimer 当前的值存放在变量 Value 中；

3、GetElapsedValue(SystemTimer, &Value, &ElapsedValue);

4、ConsValue = ConsValue - TICKS2MS(ElapsedValue);

5、当 ConsValue 等于 0 时，表示定时到期。

调用 GetElapsedValue(SystemTimer, &Value, &ElapsedValue)，获取传入参数 Value 里存放的时间到现在的时间间隔，将时间间隔存放在 ElapsedValue 中，同时将 Value 值更新为当前的时间。

每获取一次时间间隔就将 ConsValue 的值更新一次，当 ConsValue 等于 0 时，表示定时 ConsValue 到期。此处需注意时间单位的转换，SystemTimer 的计数值单位为 Tick，ConsValue 的时间单位为 ms。

注：此方式实现的定时会产生误差，其误差的大小取决于 Tick 的精度和轮询的周期，一般产生的误差都是毫秒级别，因此仅适用于精度要求不高的情况。若要实现高精度定时，建议使用 Tm 模块实现。

4.7 调度表的使用

调度表是 AUTOSAR OS 规范定义的一个系统对象，由一组终结点(Expiry Point,EP)组成,每个结点关联一系列的操作，操作可以是激活任务或者设置事件。每个调度表至少有一个结点，每个结点至少执行一个动作。

每个调度表都有一个持续时间(Duration),即调度表运行一次的时间，单位为时钟滴答(Tick)。下面主要演示如何配置生成调度表，以及调度表是如何工作的。

4.7.1 激活任务

配置参数

Scalability Class = SC1

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）
Task_Init	FULL	2	1	模式自启动 OSDEFAULTAPPMODE	200
Task_1	FULL	3	1	非自启动	200

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	1ms

调度表（ID）	自启动	持续时（Tick）	重复执行	驱动计数器
ScheduleTable_0	FALSE	1000	TRUE	SystemTimer

EP 点	偏移	到期动作

EP0	5	激活 Task_1 任务

1、OS 启动后激活并运行自启动任务 Task_Init;

2、调用系统服务函数 StartScheduleTableRel(ScheduleTable_0, Offset)，调度表将会在相对于调用时的 Offset 个 Tick 后启动，调度表 EP0 将在 ScheduleTable 启动后的 5 个 Tick 到期，到期后将执行 ActivateTask(Task_1);

3、当调度表启动后，会以持续时间(Duration)为周期执行整个调度表，如果需要停止此调度表，可调用系统服务 StopScheduleTable(ScheduleTable_0)。

4.7.2 设置事件

配置参数

Scalability Class = SC1

任务（ID）	抢占	优先级	最大激活次数	自启动	堆栈大小（Bytes）	事件
Task_Init	FULL	2	1	模式自启动 OSDEFAULTAPPMODE	200	无
Task_1	FULL	3	1	非自启动	200	Event_0

计数器（ID）	类型	最大计数值	1Tick 时间
SystemTimer	硬件计数器	100000	1ms

事件（ID）	有访问权限的任务
Event_0	Task_1

调度表（ID）	自启动	持续时（Tick）	重复执行	驱动计数器	自启动类型	启动时间
ScheduleTable_0	TRUE	1000	TRUE	SystemTimer	绝对启动	5

EP 点	偏移	到期动作
EP0	5	1、激活 Task_1 任务 2、为 Task_1 设置 Event_0

1、OS 启动后激活并运行自启动任务 Task_Init;

2、ScheduleTable_0 以绝对启动方自启动时，启动时间为 5Tick 时，即当 SystemTimer 为 5 Tick 时启动 ScheduleTable_0，并且在 ScheduleTable_0 启动后的 5 个 Tick 时执行动作：

a. ActivateTask(Task_1);

b. SetEvent(Task_1，Event_0)；

3、当调度表启动后，会周期执行整个调度表，如果需要停止此调度表，可调用系统服务 StopScheduleTable(ScheduleTable_0)。

5.0 配置参数介绍

5.1 通用配置（General）

General 配置是 OS 的通用配置项，如图 5-1 所示。General 各项配置参数描述见表 5-1。

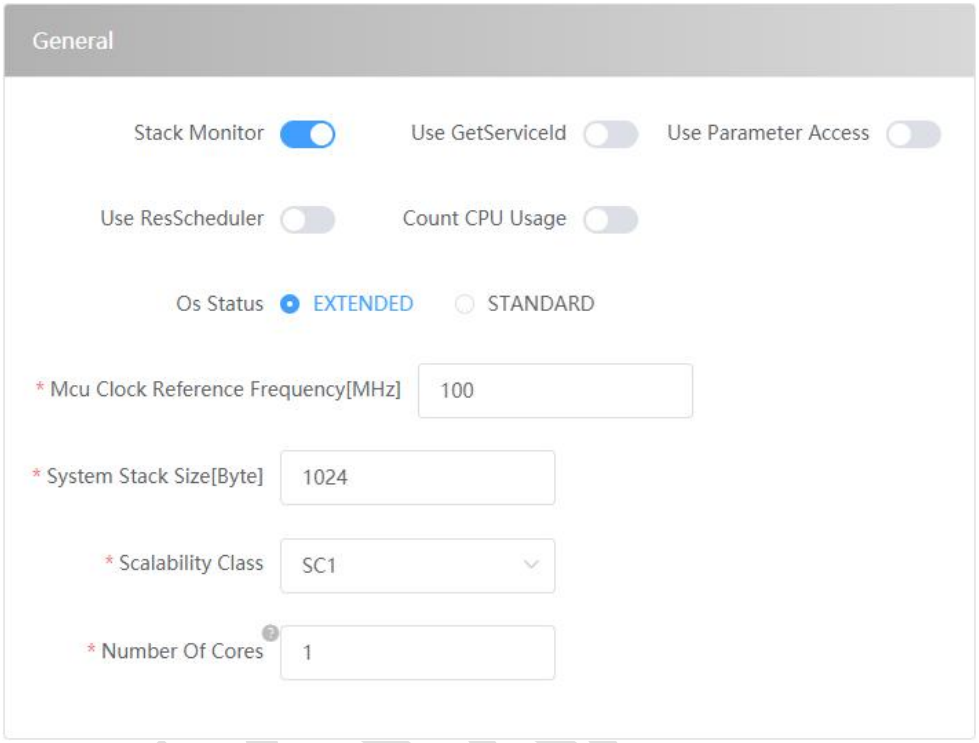


图 5-1 General 配置

表 5-1 General 配置说明

名称	描述			
Stack Monitor	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	TRUE：Os 将会做堆栈检测； FALSE，Os 不做堆栈检测。		
	依赖关系	无		
Use GetServiceId	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	TRUE：可以获取服务 ID； FALSE：无法获取服务 ID。		
	依赖关系	无		
Use Parameter Access	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	参数访问		

	依赖关系	无		
Use ResSchedule	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	TRUE：使用调度资源 FALSE：不使用调度资源		
	依赖关系	无		
Count CPU Usage	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	TRUE：计算 CPU 的使用率 FALSE：不计算 CPU 的使用率		
	依赖关系	无		
Os Status	取值范围	EXTENDED, STANDARD	默认值	EXTENDED
	参数描述	此宏定义用于指明操作系统各个功能接口运行时状态类型。 STANDARD：各个功能接口运行时不对调用参数做扩展检查，直接执行，返回值表示运行结果，非具体的错误信息。 EXTENDED：各个功能接口运行时需要对调用参数扩展状态检查，检查通过将继续执行各个功能接口，如果调用参数不符合规范要求，则会返回相应的错误信息。		
	依赖关系	无		
Mcu Clock Reference Frequency[MHz]	取值范围	1 ... 2048(MHZ)	默认值	100(MHZ)
	参数描述	Mcu 的时钟频率		
	依赖关系	是芯片平台 Mcu 的系统时钟频率，Os 系统时钟的时钟频率依赖于 Mcu Clock Reference Frequency，根据此频率计算 SystemTimer 的定时。		
System Stack Size[Byte]	取值范围	1024 ...10240(Byte)	默认值	1024(Byte)
	参数描述	系统堆栈大小		
	依赖关系	无		
Scalability Class	取值范围	OSEK、SC1、SC2、SC3、SC4	默认值	OSEK
	参数描述	AUTOSAR OS 的扩展类别		
	依赖关系	无		
Number Of	取值范围	1 ... 16	默认值	1

Cores	参数描述	核的个数,当 EcucCoreDefinition=1 时，OsNumberOfCores 为默认值 1，不可编辑。当 EcucCoreDefinition>1 时，OsNumberOfCores 可编辑，且取值为 1<= OsNumberOfCores <=EcucCoreDefinition。		
	依赖关系	当 Scalability Class 为 OSEK 时，OsNumberOfCores =1，且不可编辑；Scalability Class 非 OSEK，OsNumberOfCores 可配的值受 EcucCoreDefinition 值的影响。		
Context Save Area Size	取值范围	1 ... 512(CSA)	默认值	64
	参数描述	上下文保存区域大小，1CSA = uint32 * 16.		
	依赖关系	芯片平台相关		

5.2 钩子（Hooks）

OS 提供一系列特定的 Hook 函数，以实现在 Os 内部处理用户操作。配置界面如图 5-2 所示，各项配置参数描述见表 5-2，HOOK 相关函数描述见 Hook 接口描述。

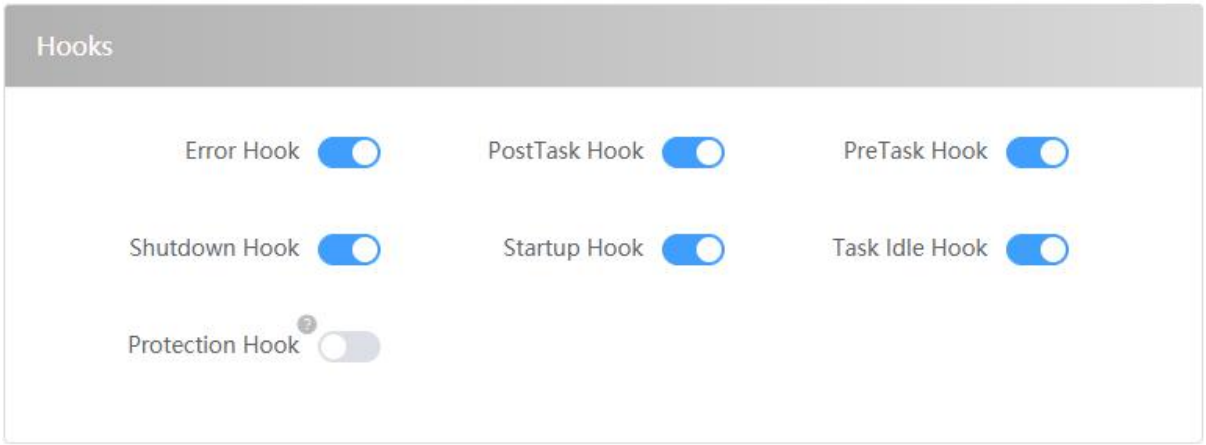


图 5-2 Hooks 配置界面

表 5-2Hooks 配置说明

名称	描述			
Error Hook	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	TRUE：使用 ErrorHook 函数 FALSE：不使用 ErrorHook 函数		
	依赖关系	无		
PostTask Hook	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	TRUE：使用 PostTaskHook 函数		

		FALSE：不使用 PostTaskHook 函数		
	依赖关系	无		
PreTask Hook	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	TRUE：使用 PreTaskHook 函数 FALSE：不使用 PreTaskHook 函数		
	依赖关系	无		
Shutdown Hook	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	TRUE：使用 ShutdownHook 函数 FALSE：不使用 PreTaskHook 函数		
	依赖关系	无		
Startup Hook	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	TRUE：使用 StartupHook 函数 FALSE：不使用 StartupHook 函数		
	依赖关系	无		
Task Idle Hook	取值范围	TRUE, FALSE	默认值	FALSE
	参数描述	TRUE：使用 TaskIdleHook 函数 FALSE：不使用 TaskIdleHook 函数		
	依赖关系	无		

5.3 任务（Task）

当创建一个新的配置项目时，工具系统会自动添加 4 个较常用的任务，他们的相关属性如图 5-3 所示。自动添加的任务仅为减少用户配置的工作量，用户可根据需要修改、删除。当新添加任务时，所需要配置的属性如图 5-4 所示，各项配置参数描述见表 5-3。

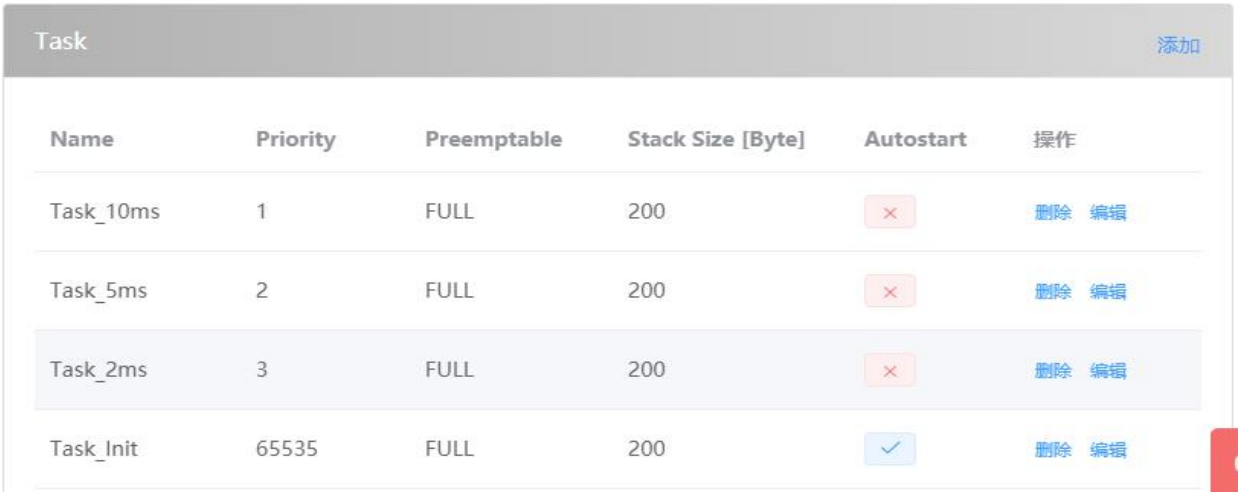


图 5-3 新建项目时自动添加的任务

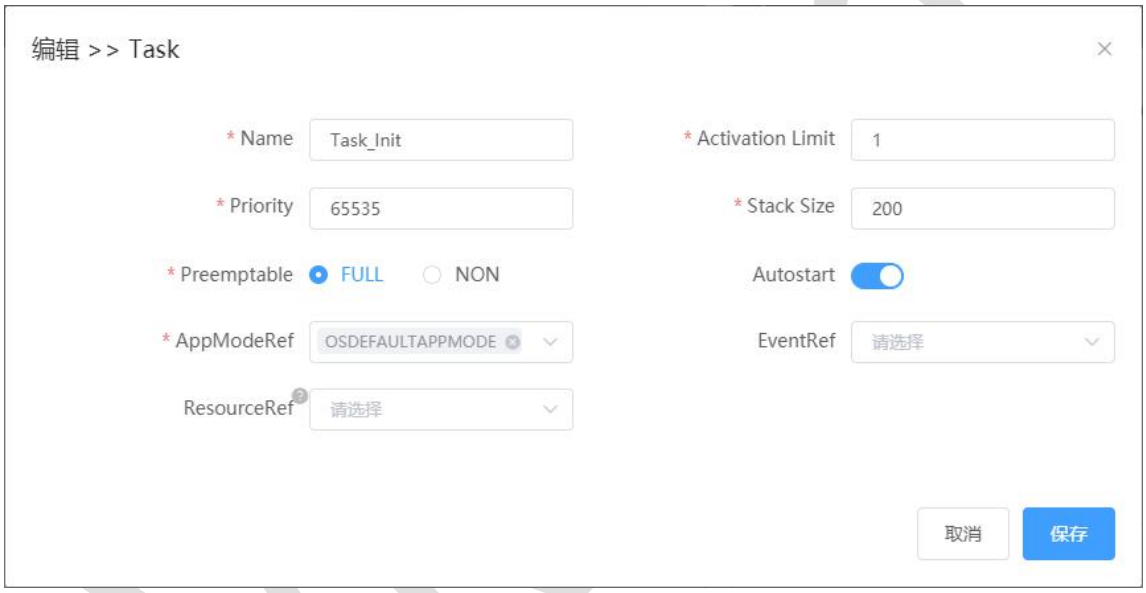


图 5-4 Task 配置界面

表 5-3 Task 配置说明

名称	描述			
Name	取值范围	无	默认值	Task_X
	参数描述	Task 的名称；此名称配置后将生成宏定义，代表任务 ID。		
	依赖关系	无		
Activation Limit	取值范围	1 <= value <= 4294967295	默认值	1
	参数描述	此任务允许的最大激活次数		
	依赖关系	无		

Priority	取值范围	0 <= value <= 4294967295	默认值	2
	参数描述	任务优先级，数值越大优先级越高		
	依赖关系	无		
Stack Size	取值范围	100 <= value <= 65535(Byte)	默认值	200
	参数描述	任务的堆栈空间大小		
	依赖关系	无		
Preemptable	取值范围	FULL, NON	默认值	FULL
	参数描述	描述此任务的调度是否可抢占；FULL：可抢占；NON：非抢占 可抢占：表示允许优先级高的任务抢占优先级低的任务运行 非抢占：表示不允许优先级高的任务抢占优先级低的任务运行，除非任务自动放弃 CPU		
	依赖关系	无		
Autostart	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	是否自启动； TRUE：在 OS 被启动起来之后自动被激活，若有多个任务自启动，则这多个任务同时被激活，但先进入优先级高的任务执行 FALSE：不能自动激活，需要 ActivateTask()API 才能激活该任务		
	依赖关系	无		
AppModeRef	取值范围	配置的 App Mode 列表	默认值	OSDEFAULTAPPMODE
	参数描述	Task 在哪种工作模式下自启动		
	依赖关系	当 Autostart 为 FALSE 时，该配置项不可配置； 当 Autostart 为 TRUE 时，该配置项的选取受 AppMode 配置的影响。		
EventRef	取值范围	配置的 Event 列表	默认值	无
	参数描述	该任务所引用的事件		
	依赖关系	仅能选择已有的 Event		
ResourceRef	取值范围	配置的 Resource 列表	默认值	无
	参数描述	该任务所引用的资源； 若选择 Property = INTERNAL 的 Resource， 或者 Property = LINKED 且链接的目标资源为 INTERNAL 的 Resource 时， 此类 Resource 有且只能选择一个		

	依赖关系	仅能选择已有的 Resource

5.4 中断（Isr）

当创建一个新的配置项目时，工具系统会自动添加 OS 依赖的 Isr 配置，他们的相关属性如图 5-5 所示，该默认添加的 Isr 除优先级外均不允许修改、删除。除 OS 依赖的 Isr，其余 Isr 用户根据自己需求添加，Isr 所需要配置的属性如图 5-6 所示，各项配置参数描述见表 5-4。

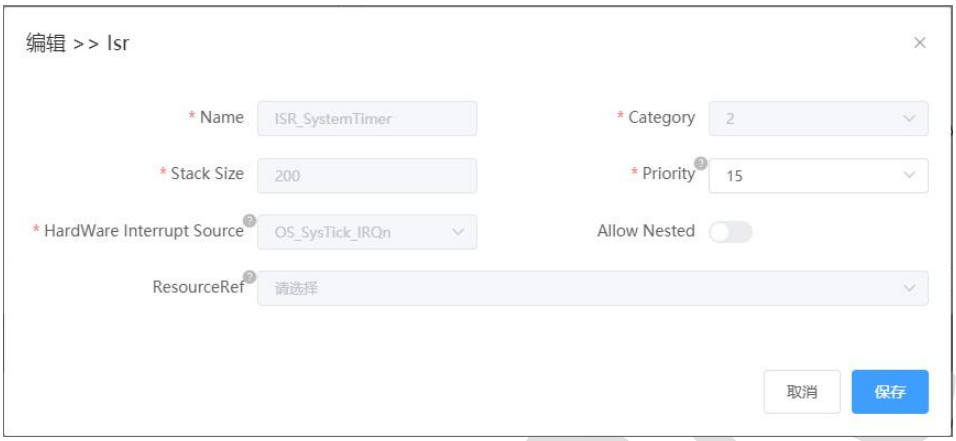


图 5-5 默认生成 Isr 配置界面

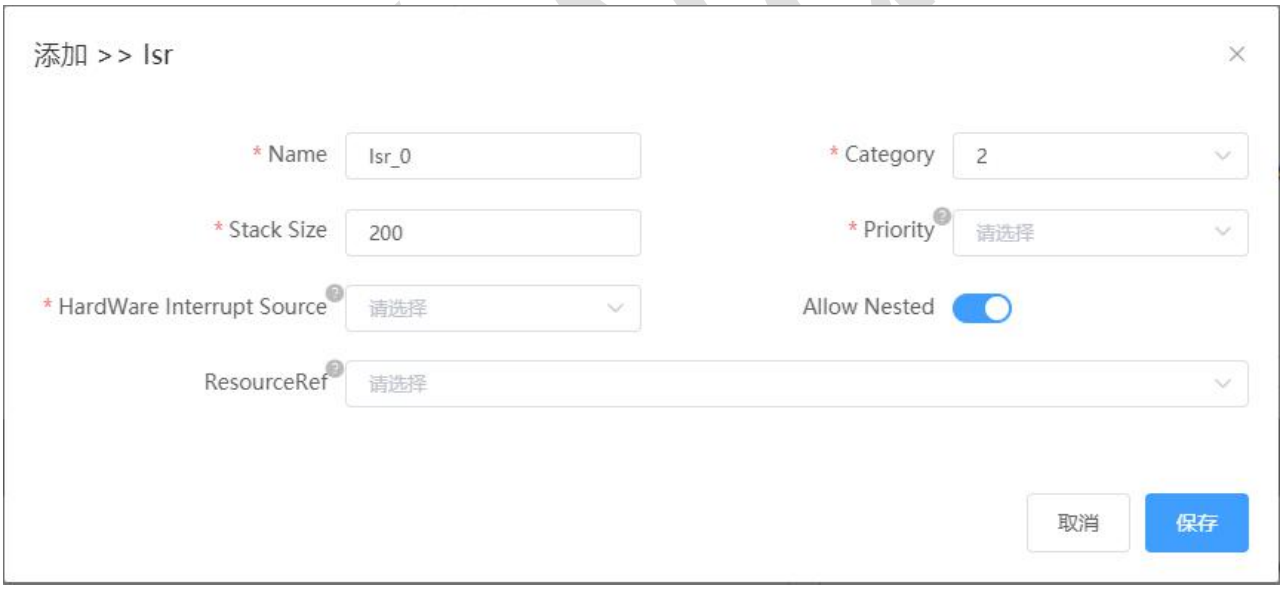


图 5-6 Isr 配置界面

表 5-4 Isr 配置说明

名称	描述			
Name	取值范围	无	默认值	Isr_X
	参数描述	中断的名称；此名称配置后将生成宏定义，代表中断 ID		

	依赖关系	无		
Category	取值范围	1， 2	默认值	2
	参数描述	中断类别； 1：代表配置的此中断为一类中断 2：代表配置的此中断为二类中断		
	依赖关系	无		
Stack Size	取值范围	100 ... 65535	默认值	200
	参数描述	中断堆栈空间大小		
	依赖关系	仅当 Category 为 2 时可配置		
Priority	取值范围	由芯片平台决定	默认值	无
	参数描述	中断优先级，数值越大优先级越高，但始终大于任务最高优先级 CYT2B9 芯片：取值 1 .. 8，可重复 TC233 芯片：取值 1 .. 255，不可重复 TC275 芯片：取值 1 .. 255，每个 CPU 内不可重复 S32K144 芯片：取值 1 .. 16，可重复		
	依赖关系	芯片平台		
HardWare Interrupt Source	取值范围	由芯片平台决定	默认值	无
	参数描述	此中断的中断源，中断源不可重复选择		
	依赖关系	芯片平台		
Allow Nested	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	TRUE：此中断允许被内嵌套 FALSE：此中断不允许被内嵌套		
	依赖关系	无		
ResourceRef	取值范围	配置的 Resource 列表	默认值	OSDEFAULTAPPMODE
	参数描述	该中断所引用的资源； 下拉选项中不包括 Property = INTERNAL 的 Resource， 也不包括 Property = LINKED 且链接的目标资源为 INTERNAL 的 Resource，或者链接的目标资源为 RES_SCHEDULER 或 RES_SCHEDULER_COREX 的 Resource。同时，也不包括 RES_SCHEDULER 和 RES_SCHEDULER_COREX		

	依赖关系	仅当 Category 为 2 时可配置

5.5 计数器（Counter）

Counter 管理主要实现对硬件计数器和软计数器的管理，为 Alarm 和调度表提供时间支持。

在 OSEK 扩展类中，不支持自定义添加 Counter，整个系统只有一个 Counter（系统时钟 SystemTimer），由工具自动生成，不可删除，但允许编辑 Max Allowed Value 和 Tick Time [us]。

在 SC1、CS2、CS3、CS4 扩展类中，支持自定义添加 Counter，且在有 Application 时 Counter 必须被一个 Application 引用，添加的 Counter 自定义添加的 Counter 配置界面如图 5-7 所示，各项配置参数描述见表 5-5。

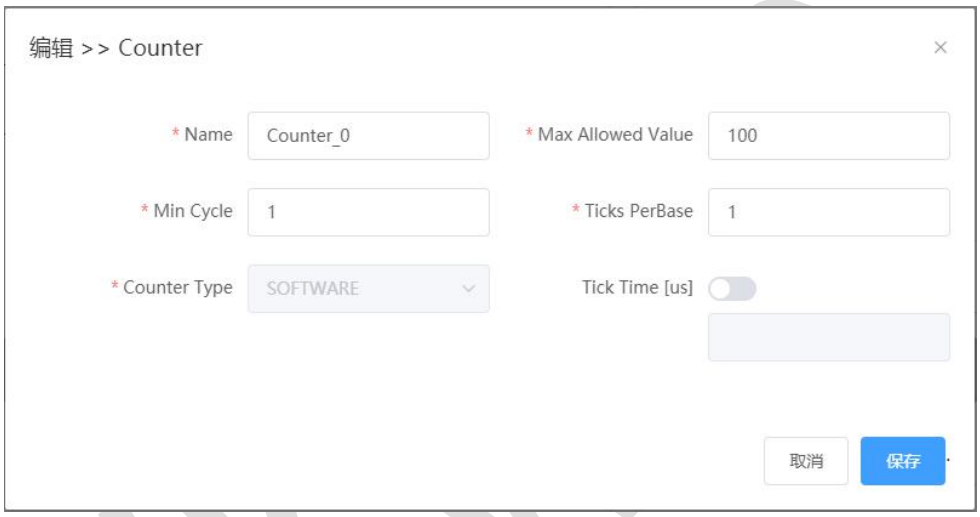


图 5-7 计数器配置界面

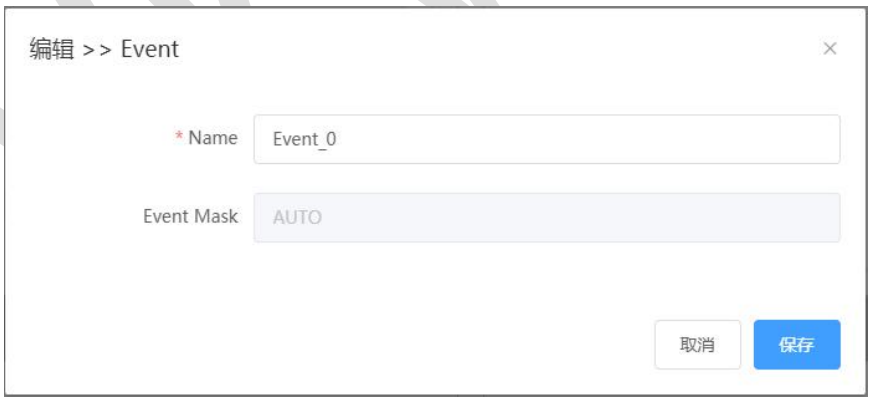
表 5-5 计数器配置参数描述

名称	描述			
Name	取值范围	无	默认值	Counter_X
	参数描述	Counter 的名称；此名称配置后将生成宏定义，代表计数器 ID		
	依赖关系	无		
Max Allowed Value	取值范围	1 ... 18446744073709551615(Tick)	默认值	100000(Tick)
	参数描述	最大计数值		
	依赖关系	无		
Min Cycle	取值范围	1 ... Max Allowed Value(Tick)	默认值	1(Tick)
	参数描述	与此计数器相连的周期 Alarm 允许的最小 tick 值		

	依赖关系	小于 Max Allowed Value		
Ticks PerBase	取值范围	1 ... Max Allowed Value(Tick)	默认值	1(Tick)
	参数描述	描述 Counter 和 Tick 对应关系，默认 1 个 Tick 为 1 个 Counter 单元		
	依赖关系	小于 Max Allowed Value		
Counter Type	取值范围	SOFTWARE, HARDWARE	默认值	SOFTWARE
	参数描述	OS 计数器的类型。SOFTWARE 类型的 Counter，由用户通过 IncrementCounter()接口，自行驱动计数；HARDWARE 类型的 Counter，由 OS 或者用户提供的硬件周期定时器驱动。默认生成的 Counter 为 HARDWARE 驱动；用户添加的 Counter 为 SOFTWARE 驱动。		
	依赖关系	无（不可配置）		
Tick Time [us]	取值范围	1 ... 4294967295(us)	默认值	1000
	参数描述	一个 Tick 所代表的时间，单位 us		
	依赖关系	Tick 是 OS 的时间单位，OS 与时间相关的 API 都以 Tick 为单位计算值		

5.6 事件（Event）

配置界面如图 5-2 所示，且最多可设置 32 个事件，Event 配置界面如图 5-8 所示，各项配置参数描述见表 5-6。



5-8 事件配置界面

5-6 事件配置参数

名称	描述			
Name	取值范围	无	默认值	Event_X
	参数描述	Event 的名称；此名称配置后将生成宏定义，代表事件 ID		

	依赖关系	无		
Event Mask	取值范围	AUTO	默认值	AUTO
	参数描述	Event 的事件掩码（工具生成），不可配置		
	依赖关系	添加 Event 时自动生成		

5.7 报警（Alarm）

Alarm 在预定时间到达时触发相关的操作有：设置事件、激活任务、执行回调、累加指定计数器；必须由一个 Counter 驱动，可以是 SystemTimer，也可以是自己定义的 Counter。报警配置界面如图 5-9 所示，各项配置参数描述见表 5-7。

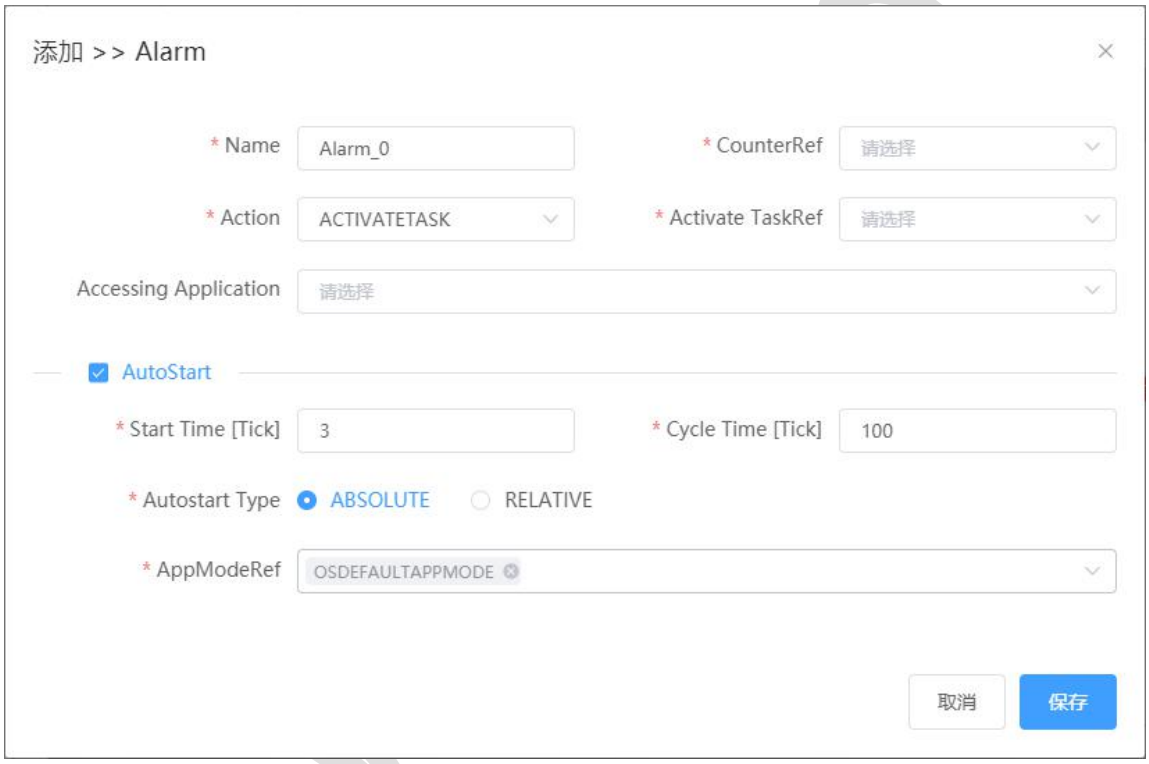


图 5-9 警报配置界面

表 5-7 警报配置参数

名称	描述			
Name	取值范围	无	默认值	Alarm_X
	参数描述	Alarm 的名称；此名称配置后将生成宏定义，代表报警 ID		
	依赖关系	无		
CounterRef	取值范围	配置的 Counter 列表	默认值	无

	参数描述	当前 Alarm 所引用的 Counter(Alarm 由 Counter 驱动)；		
	依赖关系	在多核（General Number Of Cores >1）时，OsAlarmCounterRef 引用的 OsCounter 所属的 OsApplication 的 OsApplicationCoreRef，与该 OsAlarm 所属的 OsApplication 的 OsApplicationCoreRef 必须相等		
Action	取值范围	ACTIVATETASK、 INCREMENTCOUNTER、 ALARMCALLBACK、 SETEVENT	默认值	ACTIVATETASK
	参数描述	Alarm 的动作类型（警报被触发后的动作：激活任务，计数，设置事件）		
	依赖关系	INCREMENTCOUNTER 在 OSEK 中不可选		
Activate TaskRef	取值范围	配置的 Task 列表	默认值	无
	参数描述	Alarm 需要激活的任务		
	依赖关系	Action 选择为 ACTIVATETASK 时可配置		
Increment CounterRef	取值范围	配置的 Counter 列表	默认值	无
	参数描述	设置需要计数的 Counter		
	依赖关系	Action 选择为 INCREMENTCOUNTER 时可配置；且 AlarmIncrementCounterRef 引用的 OsCounter 所属的 OsApplication 的 OsApplicationCoreRef，与该 OsAlarm 所属的 OsApplication 的 OsApplicationCoreRef 必须相等		
ALARMCALLBACK	取值范围	无	默认值	无
	参数描述	到期要执行的回调函数名称		
	依赖关系	ActionAction 选择为 ALARMCALLBACK 时可配置；且在 OSEK 中可配		
Set Event TaskRef	取值范围	配置的 Task 列表	默认值	无
	参数描述	Alarm 需要设置事件的任务		
	依赖关系	Action 选择为 SETEVENT 时可配置		
Set EventRef	取值范围	配置的 Event 列表	默认值	无
	参数描述	Alarm 需要为该任务设置的事件		
	依赖关系	ActionAction 选择为 SETEVENT 时可配置；且 OsAlarmSetEventRef 选择的 Event,必须是 OsAlarmSetEventTaskRef 选择的 Task 的 OsTaskEventRef 引用的 Event。		
名称	描述			

AutoStart	取值范围	TRUE, FALSE	默认值	TRUE
	参数描述	该 Alarm 是否自启动；		
	依赖关系	无		
Start Time [Tick]	取值范围	OsAlarmAutostartType 选择 RELATIVE 时，该项取值范围限制在 1 到引用的 Counter 的 OsCounterMaxAllowedValue。 OsAlarmAutostartType 选择 ABSOLUTE 时，该项取值范围限制在 0 到引用的 Counter 的 OsCounterMaxAllowedValue。	默认值	3
	参数描述	Alarm 自启动时间		
	依赖关系	AutoStart 为 TRUE 且 Alarm 所引用的 Counter 的 Max Allowed Value		
Cycle Time [Tick]	取值范围	所引用的 CounterMinCycle ... 所引用的 CounterMaxAllowedValue	默认值	100
	参数描述	Alarm 在启动后的周期触发警报时间		
	依赖关系	AutoStart 为 TRUE 且 Alarm 所引用的 Counter 的 CounterMinCycle 和 Max Allowed Value		
Autostart Type	取值范围	ABSOLUTE、RELATIVE	默认值	ABSOLUTE
	参数描述	自启动的类型。ABSOLUTE 绝对自启动时间；RELATIVE 相对的自启动时间；其二者的差别主要在于起始触发的时间，前者是指定的时间点，后者是相对当前时间		
	依赖关系	AutoStart 为 TRUE		
AppModeRef	取值范围	AppMode 配置的列表	默认值	无
	参数描述	Alarm 在哪种工作模式下自启动		
	依赖关系	AutoStart 为 TRUE、配置的 AppMode		

5.8 应用模式（AppMode）

当工程创建完成时，工具自动生成 OSDEFAULTAPPMODE，且不可编辑和删除。用户可根据自己的需求添加应用模式，最多支持 8 个应用模式。应用模式配置界面如图 5-10 所示，各项配置参数描述见表 5-8。

图 5-10 应用模式配置界面

表 5-8 应用模式配置参数

名称	描述			
Name	取值范围	无	默认值	AppMode_X
	参数描述	引用模式的名称；此名称配置后将生成宏定义，代表应用模式 ID		
	依赖关系	StartOS 的传入参数		

5.9 资源(Resource)

资源管理是用于不同优先级的任务/中断对共享“资源”的互斥访问，调度器是一个特殊的资源，调度资源（ResSchedule）自动生成。是否使用调度资源由基础配置项中的 Use ResSchedule 配置。用户添加的资源必须被 Task/ISR 引用，未引用的无效资源会引起校验报错。资源配置界面如图 5-11 所示，各项配置参数描述见表 5-9。

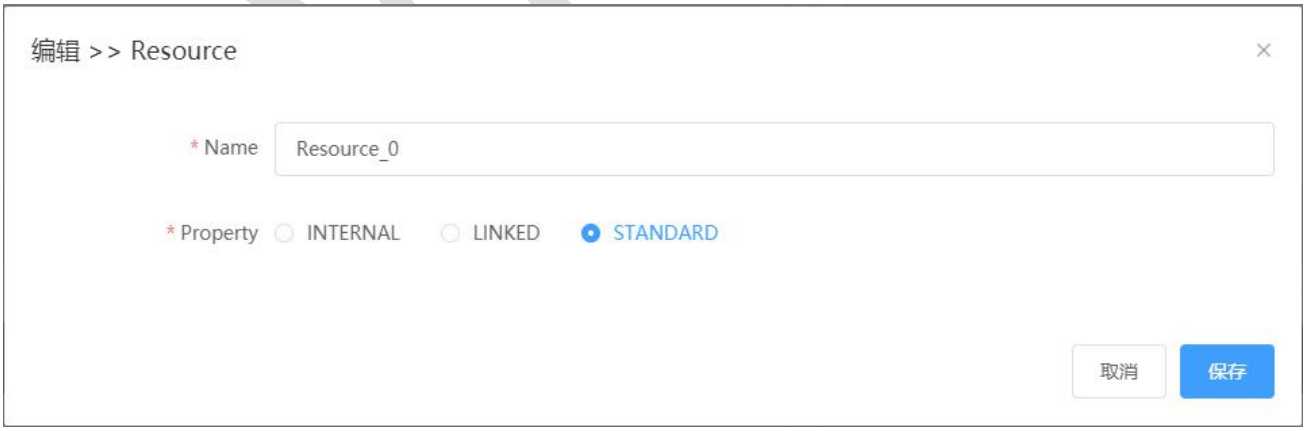


图 5-11 资源配置界面

表 5-9 资源配置参数描述

名称	描述

Name	取值范围	无	默认值	Resource_X
	参数描述	资源的名称；此名称配置后将生成宏定义，代表资源 ID		
	依赖关系	无		
Property	取值范围	INTERNAL/LINKED/STANDARD	默认值	STANDARD
	参数描述	选则资源所属类型：内部资源、链接资源、标准资源		
	依赖关系	无		

5.10 调度表（Schedule Table）

调度表是基于静态配置定义的，由 Counter 驱动执行调度处理机制，用户配置的每个调度表下至少应该有一个 EP 点，并且该 EP 下至少应该执行一个动作。调度表配置界面如图 5-12 所示，各项配置参数描述见表 5-10。

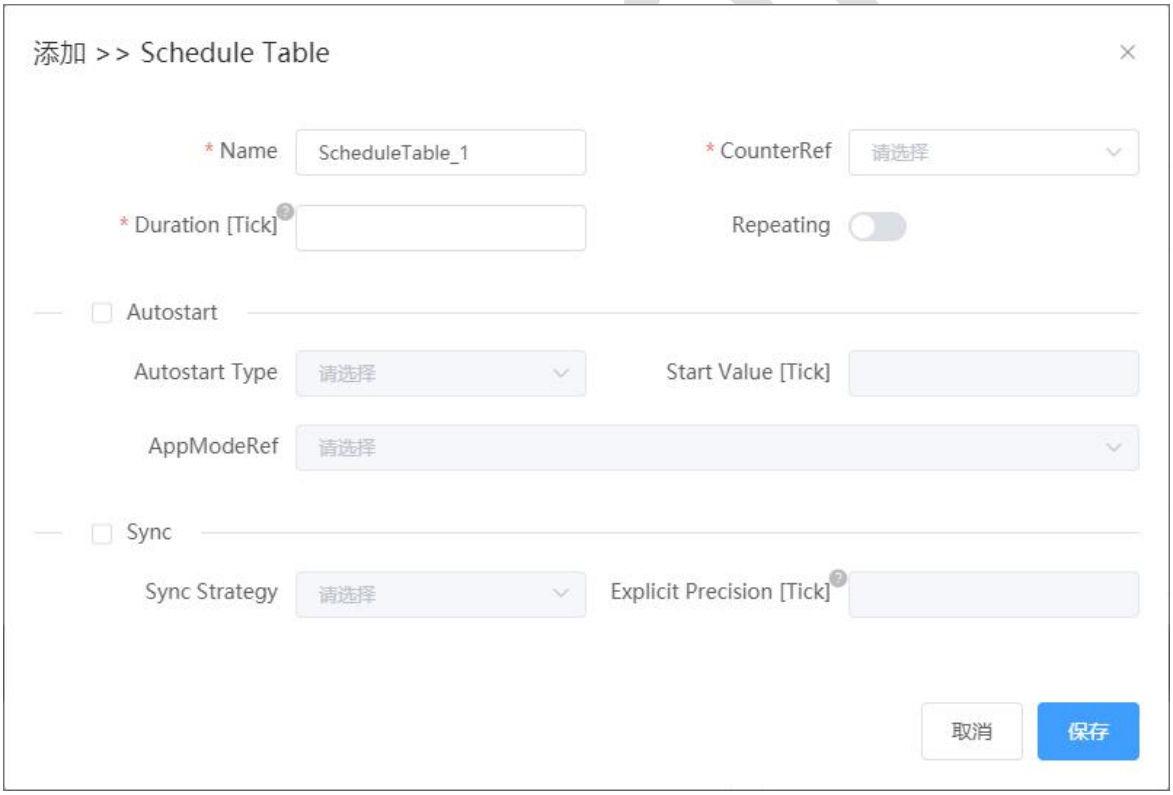


图 5-12 调度表配置界面

表 5-10 调度表配置参数

名称	描述			
Name	取值范围	无	默认值	ScheduleTable_X

	参数描述	调度表的名称；此名称配置后将生成宏定义，代表调度表 ID		
	依赖关系	SC1、SC2、SC3、SC4		
CounterRef	取值范围	配置的 Counter 列表	默认值	无
	参数描述	当前 ScheduleTable 所引用的 Counter(ScheduleTable 由 Counter 驱动)；		
	依赖关系	无		
Duration [Tick]	取值范围	1 … 所引用的 Counter 的 Max Allowed Value	默认值	无
	参数描述	调度表的持续时间；当 Sync （同步）勾选时，若同步策略为显式同步其值需和 CounterRef 关联的 Counter 的 Max Allowed Value 一致（工具自动实现），若同步策略为隐式同步时其值需和 CounterRef 关联的 Counter 的 Max Allowed Value + 1 一致（工具自动实现）		
	依赖关系	Sync 的状态以及所引用的 Counter 的 Max Allowed Value		
Repeating	取值范围	TRUE、FALSE	默认值	FALSE
	参数描述	调度表是否可重复执行		
	依赖关系	无		
名称	描述			
AutoStart	取值范围	TRUE、FALSE	默认值	TRUE
	参数描述	该 ScheduleTable 是否自启动		
	依赖关系	当 AutoStart 勾选时自启动，以下配置项可编辑		
Autostart Type	取值范围	ABSOLUTE, RELATIVE, SYNCHRON	默认值	无
	参数描述	ScheduleTable 自启动的类型； ABSOLUTE：绝对启动 RELATIVE：相对启动		
	依赖关系	OsScheduleTable.OsScheduleTableAutostart.AutostartType 下拉选项中		
Start Value [Tick]	取值范围	0 … 所引用的 Counter 的 Max Allowed Value	默认值	无
	参数描述	调度表自启动时间		
	依赖关系	当 OsScheduleTableAutostart.AutostartType 选择为 SYNCHRON 时，		

		OsScheduleTableStartValue 不能使用		
AppModeRef	取值范围	配置的 AppMode 列表	默认值	无
	参数描述	调度表自启动对应的 AppMode		
	依赖关系	配置的 AppMode		
Sync	是否选择同步策略；只有在 SC2 或 SC4 时才可选；			

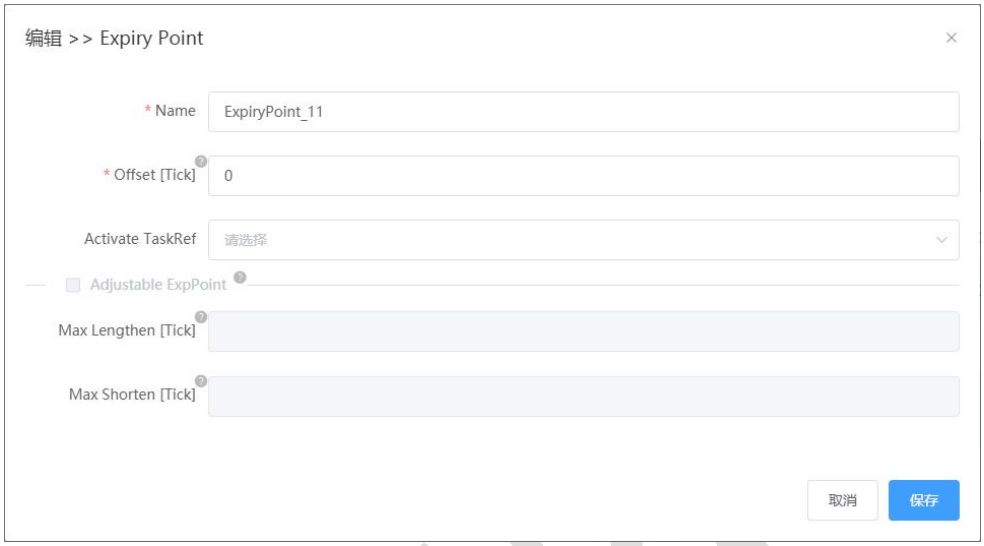


图 5-13 调度表 Expiry Point 配置界面

表 5-11 调度表 Expiry Point 配置参数

名称	描述			
Name	取值范围	无	默认值	ExpiryPoint_X
	参数描述	调度表 Expiry Point 的名称；此名称配置后将生成宏，关联 Expiry PointID。一个调度表至少有一个 Expiry Point		
	依赖关系	当调度表存在且选中某一个调度表		
Offset [Tick]	取值范围	0 或者 其父调度表所引用的 CounterMinCycle ... 其父调度表所引用的 CounterMaxAllowedValue	默认值	无
	参数描述	EP 点的偏移；至少存在一个且同一个调度表的多个 EP 点偏移不能相同		
	依赖关系	其父 Schedule Table		
Activate TaskRef	取值范围	配置的 Task 列表	默认值	无
	参数描述	EP 点到期需要激活的任务（非必填）；但同一个 EP 点下至少配置一个 Activate TaskRef 或者 Set Event TaskRef		

	依赖关系	配置的 Task

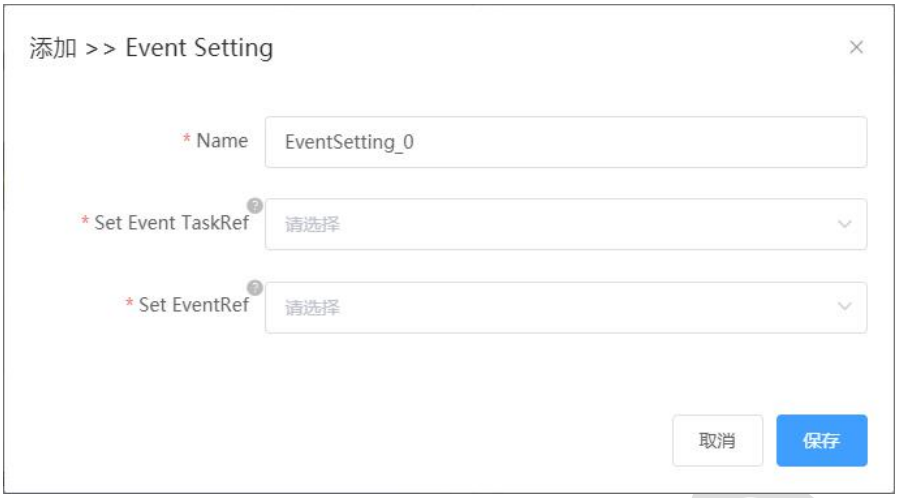


图 5-14 调度表 Expiry Point 设置事件配置界面

表 5-12 调度表 Expiry Point 设置事件配置参数

名称	描述			
Name	取值范围	无	默认值	EventSetting_X
	参数描述	名称		
	依赖关系	当 EP 点存在且选中某一个 EP 点时，此项可配置		
Set Event TaskRef	取值范围	引用了 Event 的 Task	默认值	无
	参数描述	该 EP 点需要设置事件的任务（非必填）；但同一个 EP 点下至少配置一个 Activate TaskRef 或者 Set Event TaskRef		
	依赖关系	无		
Set EventRef	取值范围	要设置的 Task 引用的 Event 列表	默认值	无
	参数描述	该 EP 点需要为该任务设置的事件；同一 EP 点下， Set EventRef 不可重复		
	依赖关系	无		

5.11 外设访问（Peripheral Area）

某些内存区域，只能在特定模式下（例如在特权模式下）访问。如果操作系统使用内存保护，则不受信任的 Os-Applications 的任务或中断无法直接访问此类区域，为了允许不受信任的 Os-Applications 访问此类区域，可以将这类地址区域配置为外设访问，使得 Os 可以通过特定的 API 访问。外设访问配置界面如图 5-15 所示，各项配置参数描述见表 5-13。

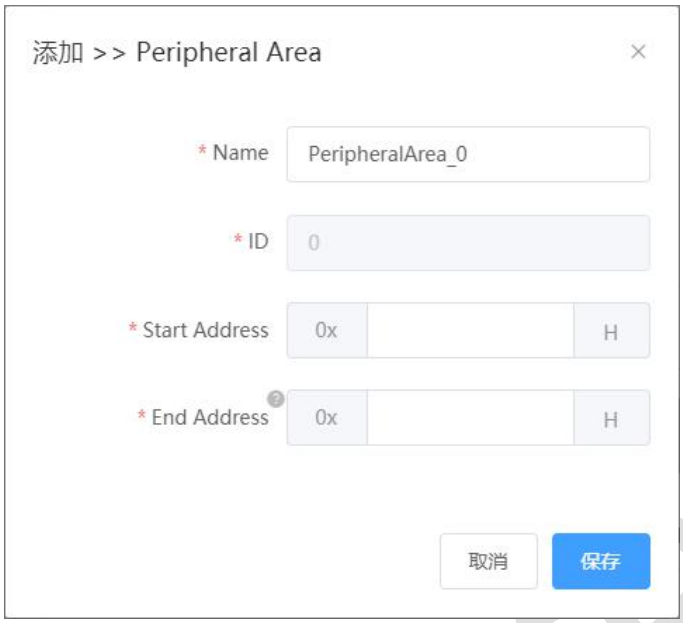


图 5-15 外设访问配置界面

表 5-13 外设访问配置参数

名称	描述			
Name	取值范围	无	默认值	PeripheralArea_X
	参数描述	外设访问的名称，此名称配置后将生成宏定义，代表任务 ID		
	依赖关系	当 EP 点存在且选中某一个 EP 点时，此项可配置		
ID	取值范围	0(不可编辑)	默认值	0
	参数描述	用户配置的 OsPeripheralArea 根据 OsPeripheralAreaId 从小到大依次显示，所有的 OsPeripheralArea 的 OsPeripheralAreaId 需满足从 0 开始，并且连续。		
	依赖关系	无		
Start Address	取值范围	无	默认值	无
	参数描述	允许对外围设备寄存器访问的起始地址		
	依赖关系	无		
End Address	取值范围	EndAddress 的值大于 StartAddress 的值	默认值	无
	参数描述	允许对外围设备寄存器访问的结束地址		
	依赖关系	EndAddress 的值大于 StartAddress 的值		

6.0 API 接口

6.1 Hook 接口

6.1.1 ErrorHook

函数原型	void ErrorHook(StatusType Error)
功能概述	➢ 应用执行错误处理接口 ➢ 当错误使用系统服务时，进入该函数 ➢ 错误使用系统服务是指：系统调用出错、Alarm 在激活任务或者设置事件时出错等 ➢ 该函数具体实现由用户定义，由操作系统调用 ➢ 依赖 OSEK, SC1, SC2, SC3, SC4 例：ActivateTask（TaskID），当 TaskID 不存在时，若 ErrorHook = TRUE，将进入 ErrorHook；
参数	Error：发生的错误，具体数值参见代码 Os_Types.h 类型

6.1.2 PostTaskHook

函数原型	void PostTaskHook(void)
功能概述	➢ 在切换任务且当前运行任务变为就绪态时调用该 HOOK ➢ 该函数具体实现由用户定义，由操作系统调用 ➢ 依赖 OSEK, SC1, SC2, SC3, SC4
参数	无

6.1.3 PreTaskHook

函数原型	void PreTaskHook(void)
功能概述	➢ 在开始执行新任务前且新任务变为运行态后调用该 HOOK ➢ 该函数具体实现由用户定义，由操作系统调用 ➢ 依赖 OSEK, SC1, SC2, SC3, SC4
参数	无

6.1.4 StartupHook

函数原型	void StartupHook(void)
功能概述	➢ 在操作系统完成初始化后，在进入调度前调用该 HOOK ➢ 该函数具体实现由用户定义，由操作系统调用 ➢ 依赖 OSEK, SC1, SC2, SC3, SC4
参数	无

6.1.5 ShutdownHook

函数原型	void ShutdownHook(StatusType Error)
功能概述	➢ 在 ShutdownOS 调用该 HOOK ➢ 该函数具体实现由用户定义，由操作系统调用 ➢ 依赖 OSEK, SC1, SC2, SC3, SC4
参数	Error：提供给用户观察的参数

6.2 Task 接口

6.2.1 ActivateTask

函数原型	StatusType ActivateTask(TaskType TaskID)
功能概述	➢ 激活任务 ➢ 将任务从挂起状态转变为就绪状态，并重调度执行就绪表最高优先级任务 ➢ 可以在任务和 ISR2 中调用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	TaskID： 任务编号；由工具自动生成，在 Os_Cfg.h 中定义

6.2.2 TerminateTask

函数原型	StatusType TerminateTask(void)

功能概述	➢ 结束当前任务 ➢ 将当前任务从运行状态转变为挂起状态 ➢ 只能在任务中使用且使用前必需释放任务所占用的资源 ➢ 依赖 OSEK, SC1, SC2, SC3, SC4
参数	无

6.2.3 ChainTask

函数原型	StatusType ChainTask(TaskType TaskID)
功能概述	➢ 把当前任务从运行态转变为挂起态，然后把指定的任务从挂起态转变为就绪态 ➢ 只能在任务中使用且使用前必需释放任务所占用的资源 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	TaskID： 任务编号；由工具自动生成，在 Os_Cfg.h 中定义

6.2.4 Schedule

函数原型	StatusType Schedule(void)
功能概述	➢ 产生调度 ➢ 若当前运行任务占用内部资源，则释放内部资源并执行一次调度，当返回时再次获取内部资源。此服务对没有内部资源的任务无效 ➢ 只能在任务中使用且使用前必需释放任务所占用的资源 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.2.5 GetTaskID

函数原型	StatusType GetTaskID(TaskRefType TaskID)
功能概述	➢ 获取当前运行任务的 ID ➢ 可以在任务、ISR2、ErrorHook、PreTaskHook、PostTaskHook 中

	使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	输入：无 输出：当前正在运行任务的 ID

6.2.6 GetTaskState

函数原型	StatusType GetTaskState(TaskType TaskID, TaskStateRefType State)
功能概述	➢ 获取指定任务的状态 ➢ 可以在任务、ISR2、ErrorHook、PreTaskHook、PostTaskHook 中使用 ➢ 任务的状态可能为 TASK_STATE_SUSPENDED, TASK_STATE_READY, TASK_STATE_RUNNING, TASK_STATE_WAITING ➢ 得到的状态只是在获取时的状态，在可抢占时，有可能获取的状态已经无效 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	TaskID：任务编号，由工具自动生成，在 Os_Cfg.h 中定义 State：存放任务的状态的指针

6.3 中断接口

6.3.1 EnableAllInterrupts

函数原型	void EnableAllInterrupts(void)
功能概述	➢ 使能所有中断服务响应 ➢ 可以在任务和 ISR2 中调用，但不能在 HOOK 中调用 ➢ 与 DisableAllInterrupts 配对使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.3.2 DisableAllInterrupts

函数原型	void DisableAllInterrupts(void)
功能概述	➢ 禁止所有中断服务响应 ➢ 保存当前中断状态，并关闭所有硬件中断；与 EnableAllInterrupts 配对使用 ➢ 可以在任务和 ISR2 中调用，但不能在 HOOK 中调用 ➢ 与 EnableAllInterrupts 配对使用，且在这两个函数之间不允许调用系统 API ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.3.3 ResumeAllInterrupts

函数原型	void ResumeAllInterrupts (void)
功能概述	➢ 恢复被函数 SuspendAllInterrupts 存储的中断状态 ➢ 可以在一类 ISR、二类 ISR 和任务级中被调用，但不能在 HOOK 中使用 ➢ 与 SuspendAllInterrupts 配对使用，且在这两个函数之间不允许调用系统 API ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.3.4 SuspendAllInterrupts

函数原型	void SuspendAllInterrupts (void)
功能概述	➢ 禁止所有中断并保存状态 ➢ 可以在一类 ISR、二类 ISR 和任务级中被调用，但不能在 HOOK 中使用 ➢ 与 ResumeAllInterrupts 配对使用，且在这两个函数之间不允许调用系统 API ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.3.5 ResumeOSInterrupts

函数原型	void ResumeOSInterrupts (void)
功能概述	➢ 恢复二类中断 ➢ 可以在一类 ISR、二类 ISR 和任务级中被调用，但不能在 HOOK 中使用 ➢ 与 SuspendOSInterrupts 配对使用，且在这两个函数之间不允许调用系统 API ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.3.6 SuspendOSInterrupts

函数原型	void SuspendOSInterrupts (void)
功能概述	➢ 禁止二类中断并保存状态 ➢ 可以在一类 ISR、二类 ISR 和任务级中被调用，但不能在 HOOK 中使用 ➢ 与 ResumeOSInterrupts 配对使用，且在这两个函数之间不允许调用系统 API ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.3.7 GetISRID

函数原型	ISRType GetISRID(void)
功能概述	➢ 获取当前执行的二类中断的 ID ➢ 依赖 SC1, SC2, SC3, SC4
参数	无

6.3.8 EnableInterruptSource

函数原型	StatusType EnableInterruptSource(ISRType ISRID, boolean ClearPending)

功能概述	➢ 使能指定二类中断源，并选择是否清除原有挂起的标志位 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ISRID：中断源 ID，由工具自动生成，在 Os_Cfg.h 中定义 ClearPending：是否清除原有挂起的中断标志（TRUE/FALSE）

6.3.9 DisableInterruptSource

函数原型	StatusType DisableInterruptSource(ISRType ISRID)
功能概述	➢ 关闭指定二类中断源 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ISRID：中断源 ID，由工具自动生成，在 Os_Cfg.h 中定义

6.3.10 ClearPendingInterrupt

函数原型	StatusType ClearPendingInterrupt(ISRType ISRID)
功能概述	➢ 清除指定二类中断源挂起标志位 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ISRID：中断源 ID，由工具自动生成，在 Os_Cfg.h 中定义

6.4 Event 接口

6.4.1 SetEvent

函数原型	StatusType SetEvent(TaskType TaskID, EventMaskType Mask)

功能概述	➢ 设置指定任务的事件 ➢ 如果任务在等待事件则将指定任务变为就绪 ➢ 可以在任务、ISR2、但不能够在 hook 中使用 ➢ 事件中没有设置的位保持不变 ➢ 任务可以给自己设置事件 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	TaskID： 任务编号，由工具自动生成，在 Os_Cfg.h 中定义 Mask： 事件掩码， 当设置的事件为一个时：Mask=EventID；当设置的事件为多个时：Mask=(Event1ID | Event2ID | ... | EventnID)

6.4.2 ClearEvent

函数原型	StatusType ClearEvent(EventMaskType mask)
功能概述	➢ 清除指定的事件 ➢ 只能在扩展任务中使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	Mask：事件掩码，当清除的事件为一个时：Mask=EventID；当清除的事件为多个时：Mask=(Event1ID | Event2ID | ... | EventnID)

6.4.3 GetEvent

函数原型	StatusType GetEvent(TaskType TaskID, EventMaskRefType Event)
功能概述	➢ 获取指定任务当前设置了的事件 ➢ 可以在任务、ISR2、ErrorHook、PreTaskHook、PostTaskHook 中使用 ➢ 可以获取当前运行任务的事件 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4

参数	输入：TaskID： 任务编号，由工具自动生成，在Os_Cfg.h中定义 输出：Event：设置的事件编号，当Event相应的掩码位置1，表示该事件被设置

6.4.4 WaitEvent

函数原型	StatusType WaitEvent(EventMaskType Mask)
功能概述	➢ 等待指定的事件 ➢ 只能在扩展任务中使用 ➢ 调用时没有指定的任何事件发生则将当前任务置为等待状态并发生切换 ➢ 调用时如果有指定的任何事件发生则不会发生调度 ➢ 调用前必需释放所有占用的资源 ➢ 事件不会主动被清除，必需调用 ClearEvent 才会清除事件 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	Mask：事件掩码，当等待的事件为一个时：Mask=EventID；当等待的事件为多个时：Mask=(Event1ID | Event2ID | ... | EventnID)

6.5 Alarm 接口

6.5.1 GetAlarmBase

函数原型	StatusType GetAlarmBase(AlarmType AlarmID, AlarmBaseRefType Info)
功能概述	➢ 获取指定 Alarm 的信息 ➢ 信息内容参见代码 Types.h 中 AlarmBaseType 类型定义 ➢ 可以在任务、ISR2、Hook 中使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4

参数	AlarmID：警报器编号，由工具自动生成，在 Os_Cfg.h 中定义 Info：存放 AlarmID 警报信息的指针

6.5.2 GetAlarm

函数原型	StatusType GetAlarm(AlarmType AlarmID, TickRefType Tick)
功能概述	➢ 获取指定 Alarm 此时刻距离其触发报警时的相对 Tick 数 ➢ 可以在任务、ISR2、Hook 中使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	AlarmID：警报器编号，即警报名字，由工具自动生成，在 Os_Cfg.h 中定义 Tick：存放获取的 Tick 值

6.5.3 SetRelAlarm

函数原型	StatusType SetRelAlarm(AlarmType AlarmID, TickType increment, TickType cycle)
功能概述	➢ 设置 Alarm 的相对触发时间 ➢ 在设置后的 increment 时间数到达后触发第一次，如果周期时间 cycle 不为零，则在第一次触发后按指定周期 cycle 反复触发 ➢ 可以在任务、ISR2、但不能够在 hook 中使用 ➢ 时间到达时会根据配置要求触发相关的服务，如：激活任务、设置事件、回调等 ➢ 如果相对时间数为零则会立即触发相关服务 ➢ 如果周期时间不为零，则第一次触发后会按周期反复触发 Alarm ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	AlarmID：警报器编号，由工具自动生成，在 Os_Cfg.h 中定义 increment：相对当前时钟数，第一次触发的时钟数 cycle：alarm 的周期（Tick），cycle 为 0 表示只执行一此

6.5.4 SetAbsAlarm

函数原型	StatusType SetAbsAlarm(AlarmType AlarmID, TickType start, TickType cycle)
功能概述	➢ 设置 Alarm 的绝对触发时间 ➢ 在设置后的指定时间点 start 到达后触发第一次，如果周期时间不为零 cycle，则在第一次触发后按指定周期 cycle 反复触发 ➢ 可以在任务、ISR2 中使用，但不能够在 hook 中使用 ➢ 时间到达时会根据配置要求触发相关的服务，如：激活任务、设置事件、回调等 ➢ 如果绝对时间与当前时间相等或者接近则立即触发相关服务 ➢ 如果绝对时间已经过了，则会在下一次到达该时间时触发相关服务 ➢ 如果周期时间不为零，则第一次触发后会按周期反复触发 Alarm ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	AlarmID：警报器编号，即警报名字，由工具自动生成，在 Os_Cfg.h 中定义 start：第一次触发的绝对时钟数 cycle：alarm 的周期（Tick），cycle 为 0 表示只执行一次

6.5.5 CancelAlarm

函数原型	StatusType CancelAlarm(AlarmType AlarmID)
功能概述	➢ 取消警报 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	AlarmID：警报器编号，由工具自动生成，在 Os_Cfg.h 中定义

6.6 Resource 接口

6.6.1 GetResource

函数原型	StatusType GetResource (ResourceType ResID)
功能概述	➢ 获得资源号为 ResID 的资源 ➢ 该资源属性不能是内部资源 ➢ 可以在任务和 ISR2 中使用，但不能在 HOOK 中使用 ➢ 同一资源不能够嵌套，但可以多次获取不同的资源，但要 LIFO 方式匹配 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	ResID：资源编号，由工具自动生成，在 Os_Cfg.h 中定义

6.6.2 ReleaseResource

函数原型	StatusType ReleaseResource (ResourceType ResID)
功能概述	➢ 释放资源 ResID ➢ 该资源属性不能是内部资源 ➢ 可以在任务和 ISR2 中使用，但不能在 HOOK 中使用 ➢ 资源释放者可以与资源获取者不相同 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	ResID：资源编号，由工具自动生成，在 Os_Cfg.h 中定义

6.7 Counter 接口

6.7.1 IncrementCounter

函数原型	StatusType IncrementCounter(CounterType CounterID)
功能概述	➢ 递增软件计数器（调用一次，计数器值加 1） ➢ 依赖 SC1, SC2, SC3, SC4

参数	CounterID ： 计数器编号，由工具自动生成，在 Os_Cfg.h 中定义

6.7.2 GetCounterValue

函数原型	StatusType GetCounterValue(CounterType CounterID, TickRefType Value)
功能概述	➢ 获取指定计数器的当前计数值(Tick) ➢ 依赖 SC1, SC2, SC3, SC4
参数	CounterID ： 计数器编号，由工具自动生成，在 Os_Cfg.h 中定义 Value： 存放计数值的指针

6.7.3 GetElapsedValue

函数原型	StatusType GetElapsedValue(CounterType CounterID, TickRefType Value, TickRefType ElapsedValue)
功能概述	➢ 获取计数器从输入的参考时间到现在的时间间隔 ➢ 依赖 SC1, SC2, SC3, SC4
参数	CounterID：计数器编号，由工具自动生成，在 Os_Cfg.h 中定义 Value[in]：传入用于计算的起始值(上一次保存的计数器值) Value[out]：计数器现在的数值(可用作下次传入计算起始值) ElapsedValue：存放当前 Tick 与参考 Tick 的间隔时间指针

6.8 Schedule Table 接口

6.8.1 StartScheduleTableRel

函数原型	StatusType StartScheduleTableRel(ScheduleTableType ScheduleTableID, TickType Offset)

功能概述	➢ 在相对当前时间的 offset 偏移时，启动调度表 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ScheduleTableID：调度表编号，由工具自动生成，在 Os_Cfg.h 中定义 Offset： 从当前时间开始，到调度表开始处理的时钟数值（调度表对应 Counter 的 tick 值）

6.8.2 StartScheduleTableAbs

函数原型	StatusType StartScheduleTableAbs(ScheduleTableType ScheduleTableID, TickType Start)
功能概述	➢ 在绝对时间 Start 时，启动调度表 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ScheduleTableID：调度表编号，由工具自动生成，在 Os_Cfg.h 中定义 Start： 调度表开始处理的时钟数值（调度表对应 Counter 的绝对 tick 值）

6.8.3 StopScheduleTable

函数原型	StatusType StopScheduleTable(ScheduleTableType ScheduleTableID,)
功能概述	➢ 停止调度表 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ScheduleTableID：调度表编号，由工具自动生成，在Os_Cfg.h中定义

6.8.4 NextScheduleTable

函数原型	StatusType NextScheduleTable(ScheduleTableType ScheduleTableID_From, ScheduleTableType ScheduleTableID_To)
功能概述	➢ 停止调度表 ➢ 依赖 SC1, SC2, SC3, SC4

参数	ScheduleTableID_From：当前正在处理的调度表 ScheduleTableID_To：提供一系列触发点的下一个处理调度表

6.8.5 GetScheduleTableStatus

函数原型	StatusType GetScheduleTableStatus(ScheduleTableType ScheduleTableID, ScheduleTableStatusRefType ScheduleStatus)
功能概述	➢ 获取指定调度表的状态 ➢ 依赖 SC1, SC2, SC3, SC4
参数	ScheduleTableID：调度表编号，由工具自动生成，在 Os_Cfg.h 中定义 ScheduleStatus ：用于存放该调度表的状态指针

6.9 Peripheral Area 接口

6.9.1 ReadPeripheral8

函数原型	StatusType ReadPeripheral8(AreaIdType Area, const uint8* Address, uint8* ReadValue)
功能概述	➢ 按 uint8 类型读外设地址内容。（uint16、uint32 同理） ➢ 依赖 SC1, SC2, SC3, SC4
参数	Area：外设区域编号 Address：读取的目标地址 ReadValue：存放读取的值 ReadValue：用于存放读取内容的指针

6.9.2 WritePeripheral8

函数原型	StatusType WritePeripheral8(AreaIdType Area, uint8* Address, uint8 WriteValue)

功能概述	➢ 按 uint8 类型写外设地址内容。（uint16、uint32 同理） ➢ 依赖 SC1, SC2, SC3, SC4
参数	Area：外设区域编号 Address：要写的目标地址 WriteValue：要写入的值

6.9.3 ModifyPeripheral8

函数原型	StatusType ModifyPeripheral8(AreaIdType Area, uint8* Address, uint8 Clearmask, uint8 Setmask)
功能概述	按 uint8 类型修改外设地址内容，公式： *<Address> = ((*<Address> & <clearmask>) | <setmask>)。 （uint16、uint32 同理）
参数	Area：外设区域编号 Address：修改的目标地址 Clearmask：清除掩码 Setmask：置位掩码

6.10 系统控制接口

6.10.1 StartOS

函数原型	void StartOS(AppModeType Mode)
功能概述	➢ 启动操作系统 ➢ 该接口必需在系统初始化时使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	Mode：应用模式（启动模式），用户在工具上定义

6.10.2 ShutdownOS

函数原型	void ShutdownOS(StatusType Error)
功能概述	➢ 关闭操作系统 ➢ 可以在任务、中断、ErrorHook、StartupHOOK 以及操作系统内部使用 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	Error：发生的错误，具体数值参见 StatusType 类型

6.10.3 GetActiveApplicationMode

函数原型	AppModeType GetActiveApplicationMode(void)
功能概述	➢ 获取当前的应用模式 ➢ 允许在任务、中断、所有 HOOK 中使用可以获取当前运行任务的事件 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	无

6.11 扩展接口

6.11.1 IdleHook

函数原型	void IdleHook(void)
功能概述	当操作系统空闲时，进入该 HOOK，每个核一个空闲 HOOK.
参数	无

6.11.2 GetTaskStackUsage

函数原型	GetTaskStackUsage（ TaskType taskId, uint16* pTotal, uint16* pUsed

	)
功能概述	➢ 获取指定任务的堆栈使用情况 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	taskId：需要查看堆栈的任务 pTotal：用于存放堆栈总大小的指针，单位为“字节” pUsed：用于存放已使用空间大小的指针，单位为“字节”

6.11.3 GetISR2StackUsage

函数原型	GetISR2StackUsage（ ISRType isr2Id, uint16* pTotal, uint16* pUsed ）
功能概述	➢ 获取指定 ISR2 的堆栈使用情况 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	isr2Id：需要查看堆栈的二类中断 pTotal：用于存放堆栈总大小的指针，单位为“字节” pUsed：用于存放已使用空间大小的指针，单位为“字节”

6.11.4 GetSystemStackUsage

函数原型	GetSystemStackUsage（ uint16* pTotal, uint16* pUsed ）
功能概述	➢ 获取系统堆栈使用情况 ➢ 依赖 OSEK，SC1, SC2, SC3, SC4
参数	pTotal：用于存放堆栈总大小的指针，单位为“字节” pUsed：用于存放已使用空间大小的指针，单位为“字节”