OSEK OS

软件需求规格说明书

**普华基础软件股份有限公司**

**文档控制**

| 日期 | 作者 | 版本 | 状态 | 说明 |
| --- | --- | --- | --- | --- |
| 2012-8-15 | 华明 | V0.1 | 【初稿】 | 提供需求规格说明书初稿。 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目 录**

[1 简介 1](#_Toc342307087)

[1.1 项目背景和目的 1](#_Toc342307088)

[1.2 适用对象 1](#_Toc342307089)

[1.3 缩略语/术语 1](#_Toc342307090)

[1.4 参考文档 2](#_Toc342307091)

[2 需求标识规则 2](#_Toc342307092)

[3 OSEK OS概述 4](#_Toc342307093)

[3.1 OSEK OS简介 4](#_Toc342307094)

[3.2 对其它模块的依赖 4](#_Toc342307095)

[3.3 OS文件结构 4](#_Toc342307096)

[4 功能需求 5](#_Toc342307097)

[4.1 一般行为 5](#_Toc342307098)

[4.1.1 OSEK OS体系结构 5](#_Toc342307099)

[4.1.2 OSEK OS处理级别 6](#_Toc342307100)

[4.1.3 OSEK OS符合类 6](#_Toc342307101)

[4.1.3.1 OSEK OS符合类 6](#_Toc342307102)

[4.1.4 OSEK OS的调度策略 8](#_Toc342307103)

[4.2 处理和交互 8](#_Toc342307104)

[4.2.1 OSEK OS的任务管理 8](#_Toc342307105)

[4.2.1.1 任务概念 9](#_Toc342307106)

[4.2.1.2 任务状态模型 9](#_Toc342307107)

[4.2.1.3 激活任务 12](#_Toc342307108)

[4.2.1.4 任务切换机制 12](#_Toc342307109)

[4.2.1.5 任务优先级 12](#_Toc342307110)

[4.2.1.6 调度策略 13](#_Toc342307111)

[4.2.1.7 任务终止 14](#_Toc342307112)

[4.2.2 OSEK OS的中断管理 14](#_Toc342307113)

[4.2.2.1 中断概念 14](#_Toc342307114)

[4.2.2.2 中断类型 14](#_Toc342307115)

[4.2.2.3 中断级别控制 15](#_Toc342307116)

[4.2.2.4 中断的响应过程 15](#_Toc342307117)

[4.2.3 OSEK OS的资源管理 16](#_Toc342307118)

[4.2.3.1 资源的概念和目的 16](#_Toc342307119)

[4.2.3.2 OSEK 优先级天花板协议 17](#_Toc342307120)

[4.2.3.3 OSEK 优先级天花板扩展到中断级 17](#_Toc342307121)

[4.2.3.4 资源的分类 19](#_Toc342307122)

[4.2.4 OSEK OS的事件管理 19](#_Toc342307123)

[4.2.5 OSEK OS的定时器与告警 21](#_Toc342307124)

[4.2.5.1 计数器的概念 21](#_Toc342307125)

[4.2.5.2 报警管理 22](#_Toc342307126)

[4.2.5.3 报警回调函数 23](#_Toc342307127)

[4.2.6 OSEK OS的系统控制 23](#_Toc342307128)

[4.3 时序状态图 24](#_Toc342307129)

[4.4 版本检测 24](#_Toc342307130)

[4.5 错误分类 24](#_Toc342307131)

[4.5.1 应用错误 24](#_Toc342307132)

[4.5.2 致命错误 25](#_Toc342307133)

[4.6 错误检测 26](#_Toc342307134)

[4.7 错误通知 26](#_Toc342307135)

[5 API接口需求 26](#_Toc342307136)

[5.1 输入类型 26](#_Toc342307137)

[5.2 类型定义 27](#_Toc342307138)

[5.2.1 通用数据类型定义 27](#_Toc342307139)

[5.2.1.1 优先级 27](#_Toc342307140)

[5.2.1.2 堆栈 27](#_Toc342307141)

[5.2.1.3 内核调度策略 27](#_Toc342307142)

[5.2.1.4 内核符合类 27](#_Toc342307143)

[5.2.1.5 内核可裁剪类 27](#_Toc342307144)

[5.2.1.6 模块自启动 27](#_Toc342307145)

[5.2.1.7 内核状态 27](#_Toc342307146)

[5.2.1.8 内核模式 27](#_Toc342307147)

[5.2.1.9 平台相关类型 27](#_Toc342307148)

[5.2.2 任务模块类型定义 27](#_Toc342307149)

[5.2.2.1 任务ID 27](#_Toc342307150)

[5.2.2.2 任务控制块 28](#_Toc342307151)

[5.2.3 中断模块类型定义 28](#_Toc342307152)

[5.2.4 资源模块类型定义 28](#_Toc342307153)

[5.2.4.1 资源ID 28](#_Toc342307154)

[5.2.4.2 资源类型 28](#_Toc342307155)

[5.2.4.3 资源控制块 29](#_Toc342307156)

[5.2.5 事件模块类型定义 29](#_Toc342307157)

[5.2.5.1 事件ID（事件的屏蔽位） 29](#_Toc342307158)

[5.2.6 定时器与告警模块类型定义 29](#_Toc342307159)

[5.2.6.1 定时器ID 29](#_Toc342307160)

[5.2.6.2 定时器控制块 30](#_Toc342307161)

[5.2.6.3 告警ID 30](#_Toc342307162)

[5.2.6.4 告警器控制块 30](#_Toc342307163)

[5.2.7 堆栈检测模块类型定义 31](#_Toc342307164)

[5.3 API接口描述 31](#_Toc342307165)

[5.3.1 任务API 31](#_Toc342307166)

[5.3.1.1 ActivateTask 31](#_Toc342307167)

[5.3.1.2 TerminateTask 31](#_Toc342307168)

[5.3.1.3 ChainTask 32](#_Toc342307169)

[5.3.1.4 Schedule 32](#_Toc342307170)

[5.3.1.5 GetTaskID 32](#_Toc342307171)

[5.3.1.6 GetTaskState 33](#_Toc342307172)

[5.3.2 中断API 34](#_Toc342307173)

[5.3.2.1 EnableAllInterrupts 34](#_Toc342307174)

[5.3.2.2 DisableAllInterrupts 34](#_Toc342307175)

[5.3.2.3 ResumeAllInterrupts 34](#_Toc342307176)

[5.3.2.4 SuspendAllInterrupts 35](#_Toc342307177)

[5.3.2.5 ResumeOSInterrupts 35](#_Toc342307178)

[5.3.2.6 SuspendOSInterrupts 35](#_Toc342307179)

[5.3.3 资源API 36](#_Toc342307180)

[5.3.3.1 GetResource 36](#_Toc342307181)

[5.3.3.2 ReleaseResource 36](#_Toc342307182)

[5.3.4 计数器与告警器API 37](#_Toc342307183)

[5.3.4.1 IncrementCounter 37](#_Toc342307184)

[5.3.4.2 GetCounterValue 37](#_Toc342307185)

[5.3.4.3 GetElapsedCounterValue 37](#_Toc342307186)

[5.3.4.4 GetAlarmBase 38](#_Toc342307187)

[5.3.4.5 GetAlarm 38](#_Toc342307188)

[5.3.4.6 SetRelAlarm 39](#_Toc342307189)

[5.3.4.7 SetAbsAlarm 39](#_Toc342307190)

[5.3.4.8 CancelAlarm 40](#_Toc342307191)

[5.3.5 事件API 41](#_Toc342307192)

[5.3.5.1 SetEvent 41](#_Toc342307193)

[5.3.5.2 ClearEvent 41](#_Toc342307194)

[5.3.5.3 GetEvent 41](#_Toc342307195)

[5.3.5.4 WaitEvent 42](#_Toc342307196)

[5.3.6 系统执行API 43](#_Toc342307197)

[5.3.6.1 StartOS 43](#_Toc342307198)

[5.3.6.2 ShutdownOS 43](#_Toc342307199)

[5.3.6.3 GetActiveApplicationMode 43](#_Toc342307200)

[5.3.7 错误处理API 44](#_Toc342307201)

[5.3.7.1 ErrorHook 44](#_Toc342307202)

[5.3.7.2 PreTaskHook 44](#_Toc342307203)

[5.3.7.3 PostTaskHook 45](#_Toc342307204)

[5.3.7.4 StartupHook 45](#_Toc342307205)

[5.3.7.5 ShutdownHook 45](#_Toc342307206)

[6 配置需求 46](#_Toc342307207)

[7 特定应用需求 46](#_Toc342307208)

[7.1 特定应用类型定义 46](#_Toc342307209)

[7.2 特定应用API接口 46](#_Toc342307210)

[7.3 特定应用配置需求 46](#_Toc342307211)

[8 非功能性需求 46](#_Toc342307212)

[8.1 软硬件环境需求 46](#_Toc342307213)

[8.1.1 硬件环境 46](#_Toc342307214)

[8.1.2 软件环境 47](#_Toc342307215)

[8.2 性能需求 47](#_Toc342307216)

[8.3 质量属性需求 47](#_Toc342307217)

[9 其它方面的需求 48](#_Toc342307218)

# 简介

## 项目背景和目的

本文档主要用于：对汽车电子部新OS内核的总体架构、OS各服务对象的功能、API接口、配置及其他非功能性需求进行详细定义；明确OS内核重构的工作目标，为架构设计师提供总体设计提供依据；为开发人员的详细设计、编码、编写用户手册提供指导和依据；为测试人员对本文档需求进行功能和非功能测试提供依据；为用户提供了解本模块的功能和性能提供参考。

## 适用对象

本文档适用于如下对象：

开发人员、测试人员。

## 缩略语/术语

|  |  |  |
| --- | --- | --- |
| **缩略语/术语** | **原语** | **解释** |
| OSEK/VDX | open systems and the corresponding interfaces for automotive electronics/vehicle distributed executive | OSEK，是指德国的汽车电子类开放系统和对应接口标准，而VDX则是汽车分布式执行标准。 |
| CC | Conformance Classes | 符合类 |
| OIL | OSEK Implementation Language | OSEK应用语言，用于描述OS各个服务对象的静态和动态信息，以便OS配置工具的开发。 |
| ORTI | OSEK Run Time Interface | OSEK实时接口，用于描述OS内核的动态特性。 |

## 参考文档

[1]《OSEK/VDX Specifications》V2.2.3,2005.2.17

[2]《基于OSEK/VDX规范的车控电子产品开发》李秀梅，杨国青，浙江大学，2005.4

[3]《Specification of Operating System》V3.1.1，R3.1 Rev 0002

[4]《OS内核需求跟踪表检查报告》V1.0 2012-08-16

# 需求标识规则

需求的标识（即编号）由以下几部分组成：

<组件名称>-<需求类型>-XXX [对应AUTOSAR SWS文档中的规格号]

其中：

<组件名称>采用与AUTOSAR SWS文档中规格号相同的大写字母组合，比如通信服务组件的规格号为“COMXXX”的形式，则通信服务组件的名称就为COM；

<需求类型>为4个大写字母的组合，包括：

功能需求：FUNR(Functional Requirement)

API接口需求：APIR（Application Programming Interface Requirement）

配置需求：CFGR（Configuration Requirement）

软硬件环境需求：ENVR（software/hardware Environment Requirement）

性能需求：PFMR(Performance Requirement)

质量属性需求：QLTR(Quality Requirement)

其他需求：MSLR(Miscellaneous Requirement)

<需求类型>后面的3位数字为需求点的编号，每一类需求独立地顺序编号。对于来自于AUTOSAR规范的需求点，从001开始顺序编号；对于不是来自于AUTOSAR规范即完全是我们自定义的需求点，则其编号从9打头，比如901，902等等。

[对应AUTOSAR SWS文档中的规格号]：为便于对所开发软件的AUTOSAR规范符合程度进行跟踪，对于来自于AUTOSAR规范的需求点，需要在后面跟上此字段，比如：

COM-FUNR-001 [COM432] : 本需求点是通信服务软组件COM的一个功能性需求点，其在COM的AUTOSAR SWS文件（即AUTOSAR\_SWS\_COM.pdf）中的规格号为COM432，而本需求点是我们定义的功能需求001号，通过这种方式实现了一个对应，便于后续跟踪。

由于OSEK OS规范【附件1】中对功能需求的描述方式与AUTOSAR OS规范的不一致（OSEK OS并未明确的定义需求点），所以无法确立需求点在规范中的准确位置。为了便于后续功能检查，设计人员整理出了一个OSEK OS需求跟踪表（详情见附件4）。

# OSEK OS概述

## OSEK OS简介

OSEK操作系统是符合OSEK/VDX标准并专门为汽车电子控制应用设计的实时嵌入式操作系统。OSEK操作系统具有可移植性，通过硬件抽象层，能支持8位、16位及32位的ECU。

汽车电子软件平台具有可靠性、实时性、网络化及价格敏感的要求。为了满足汽车电子环境的需求，汽车电子嵌入式实时操作系统具有如下特征：操作系统采用静态方法进行配置裁剪，包括任务、中断、事件、报警等操作系统的资源对象通过开发环境进行配置，以保证资源分配的确定性和有效性。汽车电子硬件平台内存比较小，操作系统支持在代码ROM上运行，减少对内存的需求。内存分配采用静态分配的方法保证内存分配的确定性和有效性，支持任务堆栈共享，进一步减少内存需求。操作系统能提供运行时监控手段，检测应用的运行序列，发现运行时应用的时序问题。操作系统提供硬件无关的定时报警服务，报警服务的抖动时间固定短小，为网络通讯模块提供确定性的时间调度。操作系统能配置成不同的“符合类”和调度策略，对内核进行深层次裁剪，满足不同类型的应用需求。

## 对其它模块的依赖

无

## OS文件结构

由于OS内核有不同的服务组成，每种服务会单独以一个源文件的形式存在，所以这里给出符合汽车电子事业部流程的头文件。

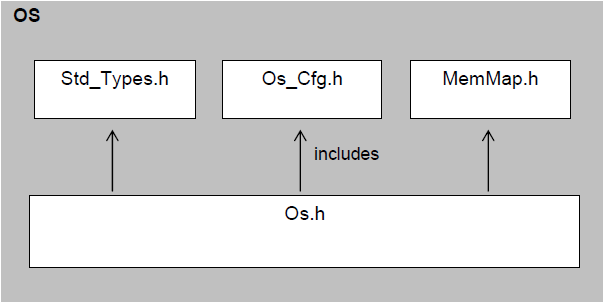


图3.1 OSEK OS头文件

# 功能需求

## 一般行为

### OSEK OS体系结构

OSEK OS的结构可通过下图来描述：

硬件环境

中断控制

InterruptHandling

资源管理

ResourceManagement

OSEK/VDX 应用

事件控制

EventControl

报警管理

Alarm

任务管理

TaskManagement

操作系统控制

OSControl

图4.1 OSEK OS体系结构

OS内核主要由以下几个模块组成：

* 任务管理模块：为应用提供多任务功能。
* 中断控制模块：为应用提供直接开关中断和中断处理的功能。
* 资源管理模块：为任务和ISR提供资源互斥访问的功能。
* 事件控制模块：为任务间提供同步功能。
* 报警管理模块：为应用提供周期性动作的接口。
* 操作系统控制模块：为应用提供操作系统符合类、调度策略、启动、关闭以及定义各HOOK函数的功能。

### OSEK OS处理级别

操作系统为用户定义了一系列接口，这些接口供OS内核实体调用以获得CPU的控制权。 OS提供两种类型的实体：

* 被操作系统管理的中断服务例程；
* 任务（基本任务和扩展任务）

操作系统提供的服务可以管理处理器的硬件资源。这些系统服务可以在应用程序中或者操作系统内部作为操作系统提供的接口来调用。

OS定义了三个处理级别：

* 中断级别
* 调度器的逻辑级别
* 任务级别

在任务级别中任务依据用户分配所分配的优先级大小来调度（非、全和混合抢占调度）；运行时的上下文在运行的一开始创建，在任务结束时被删除。

### OSEK OS符合类

#### OSEK OS符合类

符合类是一组操作系统的特性或属性。设立符合类的目的是：

* 便于理解和讨论 OSEK 操作系统，提供方便的操作系统特征组；
* 允许根据不同的需求提供OSEK的部分实现；
* 在不需要更改应用程序的情况下，系统能从较少功能的类升级到较多功能的类。

符合类在执行过程中不可改变。符合类由以下属性确定：

* 多重任务活动的请求；
* 任务类型；
* 每个优先级的任务数

定义以下符合类：

BCC1（仅为基本任务，限制为每个任务一个活动请求，每个优先级一个任务）

BCC2（BCC1+ 每个优先级可能多于一个任务，多个任务活动请求被允许）

ECC1（BCC1+ 扩展任务）

ECC2（ECC1+每个优先级可能多于一个任务，基本任务被允许多个任务活动请求 ）

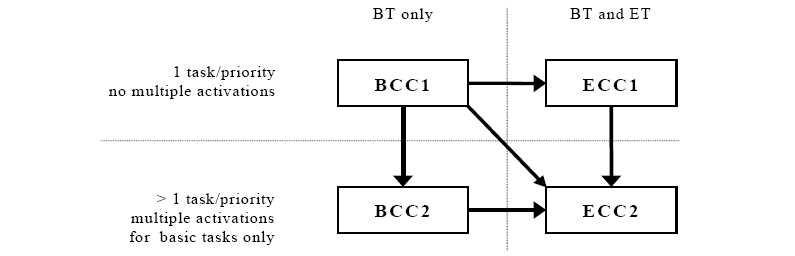
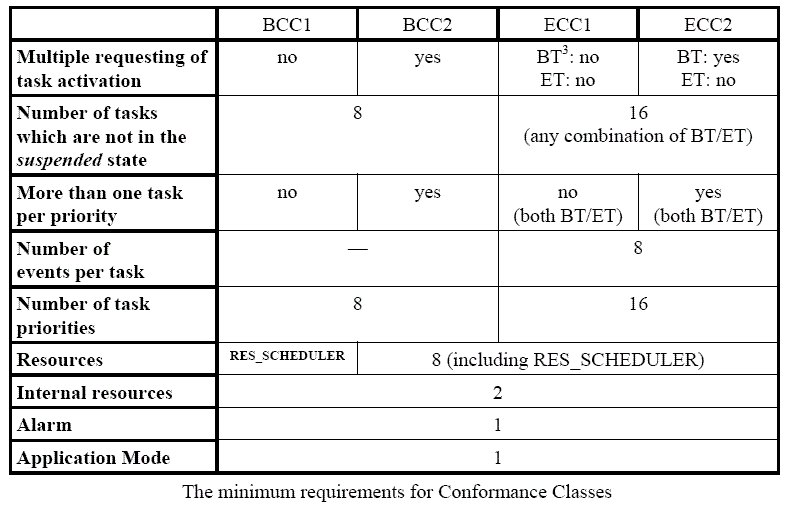


图4.2 OSEK操作系统的符合类兼容图

符合类的最低配置要求如下表所示：

表4.1符合类最低配置表



### OSEK OS的调度策略

针对应用需求，OSEK OS内核可提供以下三种内核调度策略：

* 完全可抢占式调度机制：是指当时处于运行状态的任务，可以被操作系统在任意指令处重新调度，完全可抢占式调度机制会把处于运行态的任务变成就绪态任务。是一个可抢占式的内核。
* 不可抢占式调度机制：是通过选择一个明确定义的系统服务来实现任务转换。
* 混合抢占式调度机制：如果在一个系统中既定义了可抢占式任务，又定义了不可抢占式任务，那么系统采用的调度机制就是混合式调度机制。在这种情况下，系统的调度机制取决于处于运行状态的任务的抢占属性。

## 处理和交互

### OSEK OS的任务管理

任务管理模块主要用于实现任务的激活、终止、以及各状态间的转换，并能进行基于优先级的调度。

#### 任务概念

复杂的控制软件能根据其运行时对实时性的要求方便的被分成几个部分。这些部分能用任务来实现。一个任务为功能的执行提供了一个框架。操作系统提供了并发和异步的方式来执行任务。调度器用来组织任务的执行顺序。

OSEK操作系统需要有任务的切换机制（scheduler, task switching mechanism)，包括一个当没有其它的系统或应用活跃时激活的空闲机制。OSEK操作系统提供了两种不同的任务概念：

* Basic Tasks

Basic tasks只有在以下情况才会释放处理器：

其终止

OSEK操作系统切换到一个更高优先级的任务

一个中断发生，并使处理器切换到一个中断处理服务过程(ISR)

* Extended Tasks

Extended tasks和Basic tasks的不同之处在于它允许使用系统调用WaitEvent，它将导致waitting状态。waitting状态允许在不终止正在运行的extended task的前提下释放处理器，并将其重新分配给一个优先级较低的任务。

从操作系统的层面看，理论上对extended tasks的管理比对basic tasks的管理要复杂，并需要更多的系统资源。

#### 任务状态模型

因为处理器一次只能执行一个任务的一条指令，所以任务需要在不同的状态之间转换，同时也可能有若干个任务在同一时间竞争处理器。OSEK操作系统负责在任何需要切换任务的时候保存和恢复任务的上下文。

* Extended tasks

Extended tasks 有4个任务状态：

运行（running）：在运行状态下，CPU被分配给一个任务，这个任务的指令就能被执行。在任何时刻只能有一个任务在这个状态下，而其它的状态都能同时拥有几个任务。

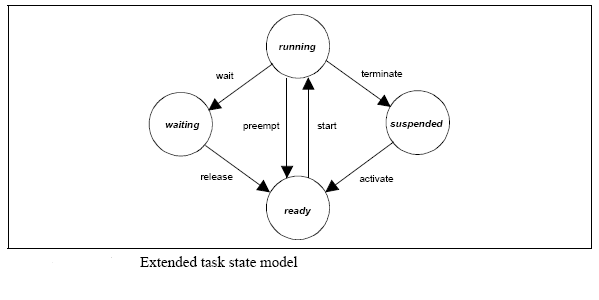


图4.3扩展任务状态模型图

就绪（ready）：所有进入运行状态的条件都具备，只差等待将处理器分配给它。调度器决定下个执行的是哪个就绪的任务。

等待（waiting）：一个任务因等待至少一个事件而不能继续执行。

挂起（suspended）：在挂起状态，任务是被动的，并能被激活。

表4.3扩展任务状态转换表

|  |  |  |  |
| --- | --- | --- | --- |
| 转换 | 前状态 | 新状态 | 描述 |
| 激活 | 挂起 | 就绪 | 用一个系统服务使一个任务进入就绪状态。OSEK保证任务从第一条指令开始执行 |
| 开始 | 就绪 | 运行 | 被调度器选中的就绪任务开始执行 |
| 等待 | 运行 | 等待 | 进入等待状态是由一个系统调用引起的，等待中的任务需要一个事件才能继续运行 |
| 释放 | 等待 | 就绪 | 至少有一个被任务等待的事件发生 |
| 抢占 | 运行 | 就绪 | 调度器决定启动另一个任务。正在运行的任务被转入就绪状态 |
| 终止 | 运行 | 挂起 | 运行状态通过调用一个系统调用进入挂起状态 |

一个任务的终止只可能由此任务终止自己。这个限制降低了操作系统的复杂性。不能直接从挂起状态转移到等待状态，这个转移使多余的，而且会为调度器增加复杂性。

* Basic tasks

basic tasks的状态模型和extended tasks的状态模型几乎是一样的。唯一的区别在于basic tasks没有等待状态。

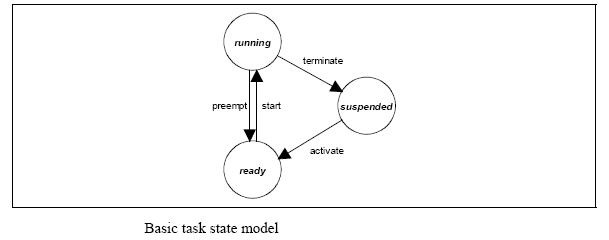


图4.4 基本任务状态模型图

运行（running）：在运行状态下，cpu被分配给一个任务，这个任务的指令就能被执行。在任何时刻只能有一个任务在这个状态下，而其它的状态都能同时拥有几个任务。

就绪（ready）：所有进入运行状态的条件都具备，只差等待将处理器分配给它。调度器决定下一个执行的是哪个就绪任务。

挂起（suspended）：在挂起状态，任务是被动的，可被激活。

表4.4基本任务状态转换表

|  |  |  |  |
| --- | --- | --- | --- |
| 转换 | 前状态 | 新状态 | 描述 |
| 激活 | 挂起 | 就绪 | 用一个系统服务使一个任务进入就绪状态。OSEK保证任务从第一条指令开始执行 |
| 开始 | 就绪 | 运行 | 被调度器选中的就绪任务开始执行 |
| 抢占 | 运行 | 就绪 | 调度器决定启动另一个任务。正在运行的任务被转入就绪状态 |
| 终止 | 运行 | 挂起 | 运行状态通过调用一个系统调用进入挂起状态 |

#### 激活任务

任务激活的操作是通过操作系统服务ActivateTask或ChainTask达到的。当任务激活之后，它准备从第一条语句开始执行。

OSEK操作系统不支持在启动任务时传递C语言形式的参数。那些参数的传递需要通过消息或者全局变量。

##### 任务激活的多次请求

不同的conformance class允许一个basic task被激活一次或多次。“任务激活的多次请求”的意思是OSEK操作系统接收到，并记录对一个已激活的任务的更多次激活请求。

多次激活请求的数量上限是在系统生成时的basic task描述属性里确定的。如果多次激活请求数量的上限没有达到，则请求被排队。basic task激活请求以每个优先级为一个队伍，按激活先后次序排队。

#### 任务切换机制

调度器是决定哪个任务该被启动，或者哪个操作系统内部活动该被触发的系统组件。根据不同的调度策略，每当一个任务切换发生时，调度器就会被启用。调度器可以被看作是一个资源，它能被任务占有或释放。因此，一个任务能占有调度器以防止任务切换，直到它被释放。

#### 任务优先级

调度器根据优先级决定哪个任务将被转换到运行状态。

值0被定义成最低的优先级。相应的值越大代表优先级越高。

为了提高效率，不支持动态优先级管理。这样一个任务的优先级是定义成静态的，也就是用户不能在运行时改变一个任务的优先级。但是，在特定的情况下操作系统可以将一个任务的优先级提高到一个预先定好的较高的优先级。

在BCC2和ECC2中支持同优先级的任务。

同优先级的任务的运行顺序是根据其激活顺序的，在等待状态的extended tasks并不阻碍与其相同优先级的后续的任务的开始。

被抢占的任务被看作是其优先级就绪列表中第一个（最老的）任务。

从等待状态中被释放的任务被看作是其优先级队列中最后一个（最新的）任务。

#### 调度策略

##### 完全抢占调度

可抢占任务调度的方式是当高优先级的任务进入就绪状态时，调度器将高优先级的任务切换到运行状态，任务切换延迟和当前低优先级的任务无关。

完全抢占调度将发生在以下情况：

* 一个任务调用TerminateTask成功结束
* 一个任务调用ChainTask成功结束并激活后续任务
* 在任务级调用ActivateTask成功激活一个任务
* 一个任务调用WaitEvent进入等待状态
* 在任务级调用SetEvent设置一个等待状态中任务的事件
* 在任务级调用ReleaseResource释放一个资源
* 从中断级返回任务级

##### 非抢占调度

非抢占调度策略是只有在运行任务主动放弃处理机情况下，调度器从就绪状态中选择最高优先级的任务进行调度。

非抢占调度发生在以下情况：

* 一个任务调用TerminateTask成功结束
* 一个任务调用ChainTask成功结束并激活后续任务
* 一个任务调用WaitEvent进入等待状态
* 显式调用Schedule服务

##### 任务组

所有获取同一个内部资源的任务为一组，对组外的任务来说，任务组的任务表现为组内最高任务优先级。当组外的任务优先级高于组内最高任务优先级，执行可抢占调度，当组外的任务优先级低于组内最高任务优先级，执行不可抢占调度。

#### 任务终止

在OSEK操作系统中，一个任务只能也必须由自己终止。OSEK提供了ChainTask系统服务来终止当前任务并在任务终止后马上激活另一个任务。每个任务都必须在任务结尾调用TerminateTask或ChainTask服务，OSEK/VDX operating system标准规定任务在未调用TerminateTask或ChainTask的情况下结束是严格禁止的。

### OSEK OS的中断管理

#### 中断概念

中断处理是嵌入式应用的一个重要组成部分，也是操作系统管理的一个重要内容。中断的优先级通常高于所有任务的优先级。

中断可以通过以下方式与系统中的任务进行联系：

* 中断可以激活任务。
* 中断可以通过拷贝方式发送和接收非队列消息。
* 中断可以触发计时器。
* 中断可以获得任务ID。
* 中断可以获得任务状态。
* 中断可以为任务设置事件。
* 中断可以获得任务的事件掩码。
* 中断可以操纵报警机制。

#### 中断类型

根据是否调用系统服务（API）这一标准，可将中断分为一类、二类两种中断，而两种类型的中断有以下区别：

* 1类中断优先级高于2类中断。最快的中断是1类中断，它不允许进行系统调用。因此，不会导致任务切换，所以执行速度快，占用资源少。该类中断执行完毕后，返回到被中断的下条指令继续执行，对任务的调度没有影响。在操作系统设计时优化中断框架，减少保存中断上下文内容，在操作系统不锁1类中断的操作，可达到快速的中断响应。这类中断的服务例程产生一个输出信号比如产生一个脉冲信号、频率信号等。
* 2类中断允许进行系统调用。在中断服务例程中可以进行如增加计数器的值、激活任务、设置事件、发送消息等操作，因此该类中断执行完毕后如果被中断的任务可抢占且没有嵌套中断，可以引发任务调度进行任务切换。在操作系统设计时，减少对2类中断锁的时间，减少中断延迟。这类中断的举例：接收一系列脉冲进行计数，接收串口信息或识别用户事件等。

#### 中断级别控制

中断处理模块能分别禁止或允许处理1类中断和2类中断；

OSEK操作系统不允许应用程序使用处理器的中断标志来控制单个中断，而必须使用EnableAllInterrupts，DisableAllInterrupts，ResumeAllInterrupts，SuspendAllInterrupts，ResumeOSInterrupts，SuspendOSInterrupts等系统服务来允许或禁用相应的中断服务。

#### 中断的响应过程

对中断的响应过程如下：

* 保存上下文
* 屏蔽优先级低于当前中断优先级的中断
* 执行中断处理例程
* 恢复被屏蔽的中断
* 恢复上下文
* 中断返回

### OSEK OS的资源管理

#### 资源的概念和目的

资源管理是用来协调多个任务或中断之间的资源共享。所有符合类级别都支持资源管理。资源管理确保：

* 两个任务或中断不能在同一时刻占有同一个资源；
* 使用资源时，不能发生优先级倒置；
* 资源使用不能发生死锁；
* 访问资源不会使任务进入等待状态。
* 资源管理的功能在以下情况下有用：
* 抢占任务；
* 非抢占任务，如果用户期望应用程序代码也可在其它策略下执行；
* 在任务和中断服务例程中资源共享；
* 中断服务例程中资源共享

可使用操作系统调用来使能或不使能中断。注意在中断服务例程中不能调用调度器资源。

OSEK 严格禁止对相同资源访问的嵌套。在很少的情况下需要嵌套，建议使用与第一资源具有相同行为的第二资源。OIL 语言支持相同行为的资源定义（所谓的相连资源 ‘linked resources’）；

使用资源的限制：

* 当一个资源被占有时，TerminateTask, ChainTask, Schedule, WaitEvent 不能被调用。
* 当占有一个资源时，中断服务例程将不能完成。
* 一个任务占有多个资源，用户必须依据 LIFO 原则（象堆栈）请求和释放资源。

#### OSEK 优先级天花板协议

为了避免死锁和优先级反转，OSEK 操作系统规定了以下行为：

* 在系统产生时，每个资源的自身天花板优先级被静态指定；天花板优先级是所有占有该资源任务或者其相关联资源的最高优先级
* 如果一个任务请求一个资源，且它当前的优先级低于资源的天花板优先级，任务的优先级就被提高到资源的天花板优先级。
* 如果任务释放资源，任务的优先级被重置为请求资源前的优先级。

优先级天花板可能引起一定时间的延迟。

#### OSEK 优先级天花板扩展到中断级

下面两个例图说明了在有中断发生时，资源的天花板优先级是如何确定的。

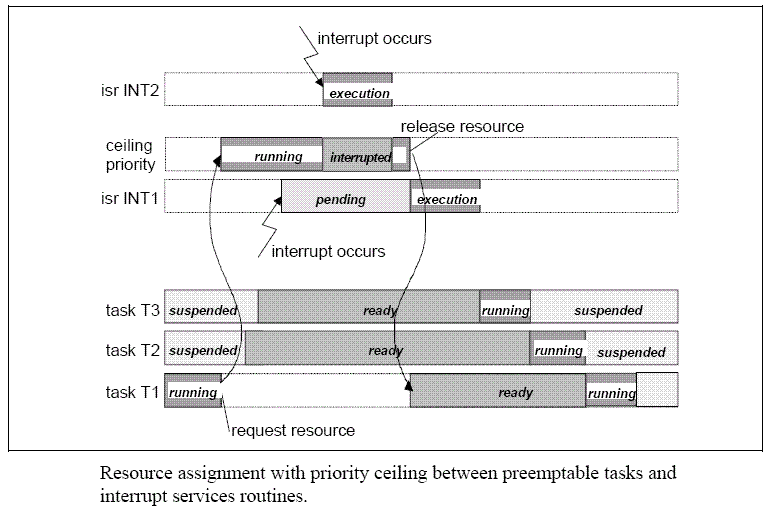


图4.5任务和中断共享资源的情况

在上图所示的例子中，INT1和task1共享同一资源。中断优先级INT2最高，INT1最低。任务task3优先级高于task2，任务task2优先级高于task1。task1先运行，并占用资源。将优先级提高到天花板优先级，在运行过程中Task1激活了

task2和task3。这时发生了中断INT1。由于这时Task1优先级最高，Task1继续运行。后来发生INT2，Task1被中断，INT2运行，此时Task1没有释放资源，仍能为天花板优先级。INT2运行完后Task1继续运行，然后释放资源，并降低优先级。此时INT1优先级最高，Task1被抢占。INT1运行完后，Task3运行，然后Task2运行，最后Task1运行。

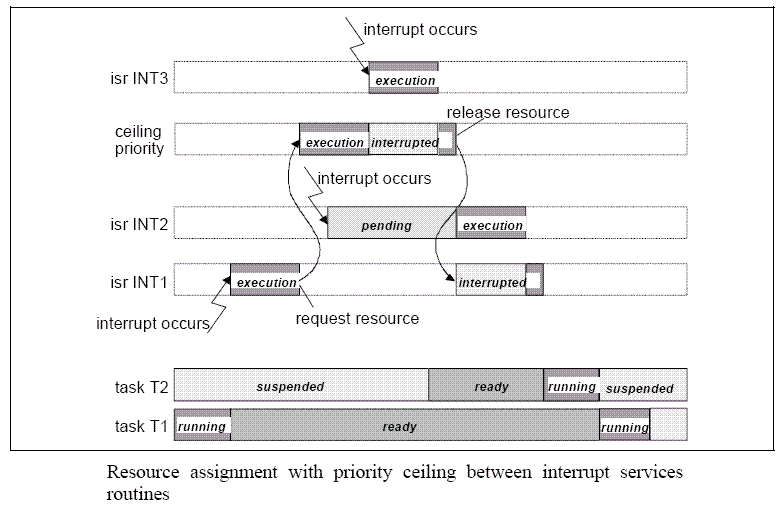


图4.6中断例程之间共享资源的情况

在上图所示的例子中，中断优先级INT3最高，INT1最低。任务task2优先级高于task1。INT1和INT2共享同一资源。task1先运行，当INT1发生时，请求占用资源。把INT1的优先级提高到天花板优先级，这时INT1的优先级高于INT2的优先级。当INT2发生时，由于INT1的优先级高于等于INT2的优先级，所以INT2不能立即执行。当INT3发生时，由于中断优先级INT3最高，所以先让INT3运行。INT3运行过程中task1就绪。INT3运行完后INT1继续运行。INT1运行中释放资源，INT1优先级恢复到原来的优先级。这时INT2的优先级高，INT2运行。INT2执行完后，INT1继续运行。然后task2运行，最后task1运行。

#### 资源的分类

##### 内部资源

内部资源是用户不可见的资源，因此，不能被系统函数 GetResource 和ReleaseResource 来获取和释放。内部资源被一组清晰定义的内部系统函数集严格的管理。除此之外，内部函数的行为与标准资源是一样的。

任务具有内部资源。在系统创建时一个任务最多有一个内部资源。内部资源的管理如下：

* 当任务进入到running 状态时，资源自动被取得，除非它已经得到的资源；作为结果，任务的优先级改变为资源的天花板优先级。
* 从重调度的角度资源被自动的释放。

##### 标准资源

标准资源可从属与一个或多个任务。

任务或ISR需要用GetResource系统服务来获得一个标准资源，并需要用ReleaseResource来释放。

在一个任务中标准资源必须在内部资源释放之前被释放。

##### LINKED资源

LINKED资源是被指向另一个LINKED资源或标准资源的对象，它的行为、属性是由其所指向的相应资源决定的。

##### 调度器作为资源

* 当一个任务要防止其它任务抢占时，它可以锁定调度器。因此一个资源，预定义的名字 RES\_SCHEDULER 自动产生。
* 中断被接收和处理独立于资源 RES\_SCHEDULER 的状态。

### OSEK OS的事件管理

事件机制是一种同步机制，通过对事件的等待、清除和设置，事件管理模块可将一个任务（只针对extended tasks）从初始状态转换到等待状态或从等待状态转换到原来的状态。

事件作为操作系统管理的对象可被分配到扩展任务。每个任务有确定个数的事件，但是每一个事件只允许有一个属主任务。事件由拥有此事件的任务和事件名（或掩码mask）定义。当扩展任务被激活时，其所有事件都被操作系统清除。

所有的任务或第2类ISR都能通过设置非挂起状态的任务的事件来通知该扩展任务进行状态转换，但只有拥有该事件的任务才能对其进行等待操作（事件的接收者只能时扩展任务。因此，任何ISR或基本任务都不能等待一个事件。）该事件发生时，拥有该事件的扩展任务从等待状态转换成就绪状态。当它获得CPU的控制权，运行完成后，必须对该事件进行清除操作。扩展任务只能清除其拥有的事件。

一个正在执行的任务如果等待一个已经发生了的事件，则这个任务仍然保持运行状态。

下图解释了在完全抢占调度中扩展任务通过事件来进行同步的情况，其中T1拥有较高的优先级。

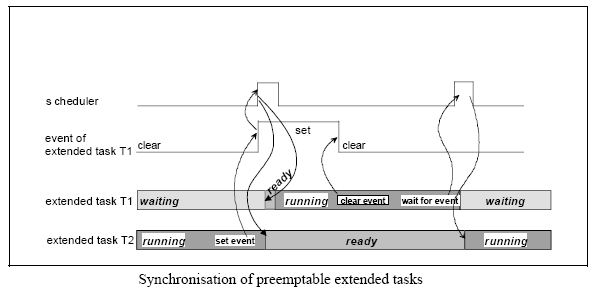


图4.7 可抢占型扩展任务的同步

而在非抢占调度中，事件发生后并不马上进行重调度操作。而需要任务调用Schedule，主动放弃执行。

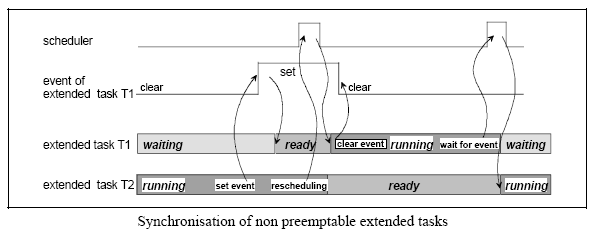


图4.8 不可抢占型扩展任务的同步

### OSEK OS的定时器与告警

定时器与告警模块用于实现一系列的周期性动作。

#### 计数器的概念

计数器是操作系统的对象，它记录已经出现的滴答数。当一个事件出现时它就增量计数。当计数器达到预定值，报警就触发。OSEK标准中没有定义计数器的接口，只定义了报警的API，而AUTOSAR则在这方面进行了扩展。

在操作系统实现中，一个计数器至少是基于一个硬件定时器或软件定时器。计数器被报警用作一个系统定时器。计数器的数量依赖于具体的硬件计数器数量。计数器通过开发环境建立。而一个系统中最少有一个Counter，以便提供必要的Tick服务。

计数器具有以下三个属性：

* Maxallowedvalue：计数器的最大值，计数器达到最大值时，下个增量为0。
* Ticksperbase：它是到达一个计数器单位所需的滴答数。计时器计数的到这个值产生一个Tick。
* Mincycle： 定义允许一个周期报警的最小的计数器滴答数。

下图描述了计数器couters和alarm的关系：

图4.9 告警与定时器

#### 报警管理

报警是与计数器关联的对象。报警定义后，即被静态赋予了一个计数器、一个任务或一个报警回调函数。多个报警可与一个给定的计时器相关联。

报警可以基于系统时钟或其他硬件/软件计数器，当计数器到达报警设定值时报警动作被触发，它可以用于激活任务，设置事件、调用报警回调函数或触发调度表的指定操作，具体行为由用户在系统配置时静态定义。报警值是可动态设置的，可以是相对值或绝对值，也可设为循环报警来激活周期性任务或调度表。此外操作系统提供了取消报警和取得报警器当前状态的接口。

下图描述报警管理的模型层次结构：



图4.10 告警管理的层次模型

OSEK 标准规定的对报警操作的系统调用为：设置相对报警SetRelAlarm(alarmId, increment, cycle)和设置绝对报警（SetAbsRelAlarm(alarmId, start, cycle)）。设置相对报警（SetRelAlarm）将报警设置为当前计数器值之后increment 个周期触发，若cycle 不为0，则为周期报警，否则，为只触发一次的报警，其实现为：将alarmId 的状态置为使用，然后设置报警的触发值和报警的周期数，然后按触发值从小到大的顺序挂接到对应计数器的报警链表上；设置绝对报警的不同是当计数器达到start 时触发报警。编写的报警处理函数如下：在每次计数器递增时，改变计数器的值，同时查看挂接在该计数器上的报警，从链表头开始比较，若报警的值等于计数器的值则触发相应动作。触发后，从链表头取下，若为周期cycle的报警，重新计算触发值后按顺序插入链表。

#### 报警回调函数

报警回调函数没有参数也没有返回值。报警回调函数定义的格式如下：

ALARMCALLBACK(AlarmCallbackRoutineName);

可以由ISR或调度器来调用报警回调函数。

### OSEK OS的系统控制

操作系统控制主要为应用提供操作系统符合类、可裁剪类、调度策略、启动、关闭以及定义各HOOK函数的功能。操作系统的启动涉及到对各个子模块的初始化，初始化的思路为各个模块提供相应的初始化函数，然后在启动函数中进行一一的调用。对于系统的关闭主要就是调用用户的系统关闭扩展，操作系统本身不完成任何实际的工作，实际的工作由用户扩展实现的功能来决定。

汽车电子应用分不同应用场景如（正常工作、在线测试、系统升级）等，需要不同的任务、资源、时钟频率，或者不同的检查粒度。 ECU上最少配置一个应用模式，通过应用模式将功能范围进行区分，减少对ECU关键计算机资源需求。当操作系统启动后，不支持应用模式切换。

## 时序状态图

OS内核基本模块的状态图已在功能描述中进行说明。这里只对trusted functions、ErrorHook、Protectionhook、StartupHook 、ShutdownHook 的流程图进行描述，详情请参考附件3的第9章即93页。

## 版本检测

暂无

## 错误分类

OSEK OS为用户提供了系统运行时的错误检测和错误处理机制。要实现这些功能，首先要对错误进行分类。

通常来说，错误可分为以下几种：

### 应用错误

表4.5 错误类型

|  |  |  |
| --- | --- | --- |
| 编号 | 错误类型 | 错误描述 |
| 1 | E\_OK | 操作成功 |
| 2 | E\_OS\_ACCESS | 无权访问 |
| 3 | E\_OS\_CALLEVEL | 无效级别 |
| 4 | E\_OS\_ID | 无效ID |
| 5 | E\_OS\_LIMIT | 超出限制 |
| 6 | E\_OS\_NOFUNC | 对象未使用或服务被拒绝 |
| 7 | E\_OS\_RESOURCE | 资源未释放 |
| 8 | E\_OS\_STATE | 无效状态 |
| 9 | E\_OS\_VALUE | 超出取值范围 |
| 10 | E\_OS\_SYS\_ASSERTION |  |
| 11 | E\_OS\_SYS\_ABORT |  |
| 12 | E\_OS\_SYS\_DIS\_INT |  |
| 13 | E\_OS\_SYS\_API\_ERROR |  |
| 14 | E\_OS\_SYS\_ALARM\_MANAGEMENT |  |
| 15 | E\_OS\_SYS\_WARNING |  |
| 16 | E\_OS\_SERVICEID | 服务不能被调用 |
| 17 | E\_OS\_PROTECTION\_EXCEPTION | 异常或陷阱发生 |
| 18 | E\_OS\_ILLEGAL\_ADDRESS | 无效地址参数 |
| 19 | E\_OS\_MISSINGEND | 任务未调用TerminateTask() 或ChainTask() |
| 20 | E\_OS\_DISABLEDINT | 中断开关未配对使用 |
| 21 | E\_OS\_STACKFAULT | 检测到堆栈错误 |
| 22 | E\_OS\_PROTECTION\_MEMORY | 存储区域访问故障 |
| 23 | E\_OS\_PROTECTION\_TIME | 任务或二类中断执行时间超预算 |
| 24 | E\_OS\_PROTECTION\_LOCKED | 任务或二类中断锁定过长 |

### 致命错误

致命错误会导致OS的关闭，不过在本内核设计中中，尚未对致命错误进行明确定义。可取的方法是在应用错误类型中选取部分作为致命错误，当这些错误发生时，会调用相关的钩子函数ShutdownHook()，关闭OS。

## 错误检测

错误检测由钩子函数完成，根据不同的错误类型，操作系统提供了不同的钩子函数。

表4.6 钩子函数类型

|  |  |  |
| --- | --- | --- |
| 编号 | 钩子函数类型 | 钩子函数描述 |
| 1 | StartupHook | 启动OS钩子函数 |
| 2 | ShutdownHook() | 关闭OS钩子函数 |
| 3 | ErrorHook | 错误处理钩子函数 |
| 4 | PreTaskHook | 任务执行前钩子函数 |
| 5 | PostTaskHook | 任务执行后钩子函数 |

## 错误通知

无

# API接口需求

## 输入类型

|  |  |
| --- | --- |
| **模块** | **输入类型** |
| Std\_Types | Std\_ReturnType |
| Std\_VersionType |
| AppModeType |
| Modules | #define MODULE\_ID\_OS (1) |

## 类型定义

### 通用数据类型定义

#### 优先级

#### 堆栈

#### 内核调度策略

#### 内核符合类

#### 内核可裁剪类

#### 模块自启动

#### 内核状态

#### 内核模式

#### 平台相关类型

### 任务模块类型定义

#### 任务ID

|  |  |
| --- | --- |
| **类型名** | TaskType |
| **类型定义** |  |
| **类型描述** | 任务的数据类型 |
| **取值说明** |  |
| **备注** |  |

#### 任务控制块

|  |  |
| --- | --- |
| **类型名** | TCB |
| **类型定义** |  |
| **类型描述** | 任务运行时的数据信息 |
| **取值说明** |  |
| **备注** | 任务控制块应该包含任务ID、堆栈、任务优先级、自启动属性、最大激活次数、当前激活次数、状态、调度策略、函数入口、所拥有的资源、相关联的事件等信息。 |

### 中断模块类型定义

暂无

### 资源模块类型定义

#### 资源ID

|  |  |
| --- | --- |
| **类型名** | ResourceType |
| **类型定义** |  |
| **类型描述** | 资源的数据类型 |
| **取值说明** |  |
| **备注** | 用于描述资源ID |

#### 资源类型

|  |  |
| --- | --- |
| **类型名** | ResourceType |
| **类型定义** |  |
| **类型描述** | 资源类型 |
| **取值说明** | 枚举型 |
| **备注** | 用于描述资源类型如标准、内部或链接型 |

#### 资源控制块

|  |  |
| --- | --- |
| **类型名** | ResourceCB |
| **类型定义** |  |
| **类型描述** | 资源运行时的数据信息 |
| **取值说明** |  |
| **备注** | 资源控制块应该包含资源ID、资源使用状态、资源天花板优先级等信息。 |

### 事件模块类型定义

#### 事件ID（事件的屏蔽位）

|  |  |
| --- | --- |
| **类型名** | EventMaskType |
| **类型定义** | typedef uint32 EventMaskType; |
| **类型描述** | 事件的掩码类型 |
| **取值说明** |  |
| **备注** | 应该与所属的扩展任务的ID结合起来 |

### 定时器与告警模块类型定义

#### 定时器ID

|  |  |
| --- | --- |
| **类型名** | CounterType |
| **类型定义** |  |
| **类型描述** | 定时器类型 |
| **取值说明** | 枚举型 |
| **备注** | 用于描述定时器ID |

#### 定时器控制块

|  |  |
| --- | --- |
| **类型名** | CounterCB |
| **类型定义** |  |
| **类型描述** | 定时器运行时的数据信息 |
| **取值说明** |  |
| **备注** | 定时器控制块应包含定时器ID、定时器基值、定时器最大允许值、定时器当前值、定时器相关告警值等。 |

#### 告警ID

|  |  |
| --- | --- |
| **类型名** | AlarmType |
| **类型定义** |  |
| **类型描述** | 告警ID |
| **取值说明** |  |
| **备注** | 用于描述告警ID |

#### 告警器控制块

|  |  |
| --- | --- |
| **类型名** | AlarmCB |
| **类型定义** |  |
| **类型描述** | 告警器运行时的数据信息 |
| **取值说明** |  |
| **备注** | 告警器控制块应包含告警器ID、关联的定时器、告警器周期值，告警器当前值、回调函数等。 |

### 堆栈检测模块类型定义

暂无

## API接口描述

### 任务API

#### ActivateTask

|  |  |
| --- | --- |
| **函数原型** | StatusType ActivateTask( TaskType taskId ) |
| **功能概述** | * 将任务从挂起状态转变为就绪状态 * 可以在任务和ISR2中使用 * 适用于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### TerminateTask

|  |  |
| --- | --- |
| **函数原型** | StatusType TerminateTask( void ) |
| **功能概述** | * 将任务从运行状态转变为挂起状态 * 只能在任务中使用 * 使用前任务必需释放所占用的资源 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 使用前任务必需释放所占用的资源 * 允许需要激活的任务是当前任务 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### Schedule

|  |  |
| --- | --- |
| **函数原型** | StatusType Schedule( void ) |
| **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 * 使用前任务必需释放所占用的资源 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 没有任务运行时引用变量中的值为INVALID\_TASK * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 状态可能为running、ready、waiting、suspended之一 * 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

### 中断API

#### EnableAllInterrupts

|  |  |
| --- | --- |
| **函数原型** | void EnableAllInterrupts(void) |
| **功能概述** | * 恢复DisableAllInterrupts 保存的中断状态 * 可以在任务和ISR2 中使用，但不能在HOOK中使用 * 适用于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### DisableAllInterrupts

|  |  |
| --- | --- |
| **函数原型** | void DisableAllInterrupts(void) |
| **功能概述** | * 保存当前中断状态，关闭所有硬件中断 * 可以在任务和ISR2 中使用，但不能在HOOK中使用 * 主要用于一些临界区使用，但这些临界区中不能使用操作系统的API * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### ResumeAllInterrupts

|  |  |
| --- | --- |
| **函数原型** | void ResumeAllInterrupts (void) |
| **功能概述** |  |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### SuspendAllInterrupts

|  |  |
| --- | --- |
| **函数原型** | void SuspendAllInterrupts (void) |
| **功能概述** |  |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### ResumeOSInterrupts

|  |  |
| --- | --- |
| **函数原型** | void ResumeOSInterrupts (void) |
| **功能概述** |  |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### SuspendOSInterrupts

|  |  |
| --- | --- |
| **函数原型** | void SuspendOSInterrupts (void) |
| **功能概述** |  |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

### 资源API

#### GetResource

|  |  |
| --- | --- |
| **函数原型** | StatusType GetResource ( ResourceType <ResID> ) |
| **功能概述** | * 获得资源号为ResID的资源 * 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### ReleaseResource

|  |  |
| --- | --- |
| **函数原型** | StatusType ReleaseResource ( ResourceType <ResID> ) |
| **功能概述** | * 释放资源ResID * 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

### 计数器与告警器API

#### IncrementCounter

|  |  |
| --- | --- |
| **函数原型** | StatusType IncrementCounter(CounterType CounterID) |
| **功能概述** | * OS381: 如果输入参数<CounterID>无效或计数器是个硬件计数器，该服务返回E\_OS\_ID. * OS391: 如果输入参数有效，该服务让< CounterID>加1并返回E\_OK * OS321: 如果错误发生在警告操作时，会调用error HOOK，但该服务仍返回E\_OK |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### GetCounterValue

|  |  |
| --- | --- |
| **函数原型** | StatusType GetCounterValue( CtrType ctrId, TickRefType ticks ) |
| **功能概述** | * 获取指定Counter的计数值 * 可以在任务和ISR2 中使用，但不能在hook中使用 * 该接口是对OSEK OS的扩展，不属于标准接口，适用于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### GetElapsedCounterValue

|  |  |
| --- | --- |
| **函数原型** | StatusType GetElapsedCounterValue(CounterType CounterID,  TickRefType Value,TickRefType ElapsedValue) |
| **功能概述** | * OS381: 如果输入参数<CounterID>无效，该服务返回E\_OS\_ID. * OS391: 如果<Value>大于<CounterID>可允许的最大值，该服务返回E\_OS\_VALUE * OS382: 如果输入参数有效，该服务通过<Value>和<ElapsedValue>输出滴答值并返回E\_OK |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### GetAlarmBase

|  |  |
| --- | --- |
| **函数原型** | StatusType GetAlarmBase( AlarmType almId, AlarmBaseRefType info ) |
| **功能概述** | * 获取指定Alarm的信息 * 可以在任务、ISR2、ErrorHook、PreTaskHook、PostTaskHook中使用 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### GetAlarm

|  |  |
| --- | --- |
| **函数原型** | StatusType GetAlarm( AlarmType almId, TickRefType tick ) |
| **功能概述** | * 获取指定Alarm的信息 * 可以在任务、ISR2、ErrorHook、PreTaskHook、PostTaskHook中使用 * 适用于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### SetRelAlarm

|  |  |
| --- | --- |
| **函数原型** | StatusType SetRelAlarm( AlarmType almId, TickType increment, TickType cycle ) |
| **功能概述** | * 设置Alarm的相对触发时间，在设置后的指定时间数到达后触发第一次，如果周期时间不为零，则在第一次触发后按指定周期反复触发 * 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 * 属于BCC1, BCC2, ECC1, ECC2，事件方式只适用于ECC1、ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### SetAbsAlarm

|  |  |
| --- | --- |
| **函数原型** | StatusType SetAbsAlarm( AlarmType almId, 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到达后触发第一次，如果周期时间不为零，则在第一次触发后按指定周期反复触发 * 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 * 属于BCC1, BCC2, ECC1, ECC2，事件方式只适用于ECC1、ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### CancelAlarm

|  |  |
| --- | --- |
| **函数原型** | StatusType CancelAlarm( AlarmType almId ) |
| **功能概述** | * 取消指定的Alarm * 可以在任务、ISR2、但不能够在hook中使用 * 适用于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

### 事件API

#### 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 * 属于ECC1、ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 属于ECC1、ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### GetEvent

|  |  |
| --- | --- |
| **函数原型** | StatusType GetEvent(TaskType taskId, EventMaskRefType mask ) |
| **功能概述** | * 获取指定任务的当前事件，但不是在等待的事件 * 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 * 属于ECC1、ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### WaitEvent

|  |  |
| --- | --- |
| **函数原型** | StatusType WaitEvent( EventMaskType mask ) |
| **功能概述** | * 等待指定的事件之一 * 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 * 属于ECC1、ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

### 系统执行API

#### 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

### 错误处理API

#### ErrorHook

|  |  |
| --- | --- |
| **函数原型** | void ErrorHook(StatusType Error) |
| **功能概述** | * 执行应用的错误处理，该接口由用户实现，操作系统来调用 * 系统调用的时机包括：系统调用出错、Alarm在激活任务或者设置事件时出错 * ErrorHook中调用系统调用出错时不再调用ErrorHook * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### PreTaskHook

|  |  |
| --- | --- |
| **函数原型** | void PreTaskHook(void) |
| **功能概述** | * 在执行新任务前而且是在新任务变为运行态后调用该HOOK * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### PostTaskHook

|  |  |
| --- | --- |
| **函数原型** | void PostTaskHook(void) |
| **功能概述** | * 在离开当前任务，但当前任务仍然是运行态时调用 * 适用于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### 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 * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

#### ShutdownHook

|  |  |
| --- | --- |
| **函数原型** | void ShutdownHook(StatusType Error) |
| **功能概述** | * 在ShutdownOS调用时调用该HOOK * 属于BCC1, BCC2, ECC1, ECC2 |
| **同步/异步** |  |
| **服务器ID** |  |
| **参数说明** |  |
| **返回值** |  |

# 配置需求

# 特定应用需求

## 特定应用类型定义

无

## 特定应用API接口

无

## 特定应用配置需求

无

# 非功能性需求

## 软硬件环境需求

### 硬件环境

|  |  |  |
| --- | --- | --- |
| **需求名称** | **需求编号** | **详细要求** |
| 运行的目标硬件平台 |  | 支持的运行平台：XEP100 |

### 软件环境

|  |  |  |
| --- | --- | --- |
| **需求名称** | **需求编号** | **详细要求** |
| 支持的开发环境 |  | XEP100平台：   1. CodeWarrior V5.0集成开发环境和BDM调试器； |

## 性能需求

|  |  |  |  |
| --- | --- | --- | --- |
| **质量属性** | **需求名称** | **需求编号** | **详细要求** |
| 性能 |  |  |  |

## 质量属性需求

|  |  |  |  |
| --- | --- | --- | --- |
| **质量属性** | **需求名称** | **需求编号** | **详细要求** |
| 健壮性 | 错误处理 |  | 对错误情况提供处理，并且为提高效率，根据标准要求分为基本错误检查和扩展错误检查。 |
| 可靠性 | 长时间运行稳定 |  | 编码遵循MISRA规则；  采用全静态配置方式，避免运行期错误；  采用源码级的完全可裁剪方式，尽可能保障没有多余代码；  采用测试集对系统进行测试。 |
| 易用性 | － |  | － |
| 安全性 | － |  | － |
| 可扩展性 | 用户可扩展 |  | 通过hook机制实现在出错等时机的用户扩展能力。 |
| 兼容性 | 符合AUTOSAR标准 |  | 符合AUTOSAR 有关OS软件的规范 |
| 可移植性 | 支持多种CPU和目标板 |  | 能够方便的移植到不同的CPU和目标板。对于该软件各功能执行所需要的硬件平台资源需求有清楚的要求和相关的设计考虑 |
| 可维护性 | 源码提供文档注释 |  | 能够从源码提取技术参考文档。 |
| 可测试性 | 辅助调试接口 |  | 提供调试相关接口 |

# 其它方面的需求

|  |  |  |
| --- | --- | --- |
| **需求名称** | **需求编号** | **详细要求** |
| 无 | 无 | 无 |