



**Hi3861V100 / Hi3861LV100 OSA&FreeRTOS**

## **API 适配指南**

文档版本 01

发布日期 2020-04-30

版权所有 © 上海海思技术有限公司2020。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。

# 上海海思技术有限公司

地址：            深圳市龙岗区坂田华为总部办公楼    邮编：518129

网址：            <https://www.hisilicon.com/cn/>

客户服务邮箱：  [support@hisilicon.com](mailto:support@hisilicon.com)



# 前言

## 概述

本文档主要介绍Hi3861V100、Hi3861LV100的OSA层API接口与FreeRTOS API接口适配使用。

## 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 |
|---------|------|
| Hi3861  | V100 |
| Hi3861L | V100 |



## 读者对象

本文档主要适用于以下工程师：




- 技术支持工程师
- 软件开发工程师

## 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                              |
|---|---------------------------------|
|  <b>危险</b> | 表示如不可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。 |
|  <b>警告</b> | 表示如不可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。 |



| 符号   | 说明   |
|--|--|
|  注意 | 表示如不可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。  |
|  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。<br>“须知”不涉及人身伤害。 |
|  说明 | 对正文中重点信息的补充说明。<br>“说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。                        |

## 修改记录

| 文档版本  | 发布日期       | 修改说明       |
|-------|------------|------------|
| 01    | 2020-04-30 | 第一次正式版本发布。 |
| 00B01 | 2020-04-03 | 第一次临时版本发布。 |



# 目录

|                       |    |
|-----------------------|----|
| 前言.....               | i  |
| 1 任务管理.....           | 1  |
| 1.1 API 对比.....       | 1  |
| 1.2 API 说明.....       | 2  |
| 1.2.1 创建任务.....       | 2  |
| 1.2.2 删除任务.....       | 3  |
| 1.2.3 获取任务名称.....     | 3  |
| 1.2.4 查询某个任务的优先级..... | 3  |
| 1.2.5 设置某个任务的优先级..... | 3  |
| 1.2.6 挂起任务.....       | 4  |
| 1.2.7 恢复任务.....       | 4  |
| 1.2.8 任务睡眠.....       | 4  |
| 2 内存管理.....           | 5  |
| 2.1 API 对比.....       | 5  |
| 2.2 API 说明.....       | 5  |
| 2.2.1 申请动态内存.....     | 5  |
| 2.2.2 释放申请的内存.....    | 6  |
| 3 消息队列管理.....         | 7  |
| 3.1 API 对比.....       | 7  |
| 3.2 API 说明.....       | 7  |
| 3.2.1 动态创建队列.....     | 8  |
| 3.2.2 删除消息队列.....     | 8  |
| 3.2.3 发送消息到消息队列.....  | 8  |
| 3.2.4 接收消息队列中的数据..... | 9  |
| 3.2.5 检查消息队列是否已满..... | 9  |
| 4 事件管理.....           | 10 |
| 4.1 API 对比.....       | 10 |
| 4.2 API 说明.....       | 10 |
| 4.2.1 创建事件.....       | 11 |
| 4.2.2 发送事件.....       | 11 |
| 4.2.3 等待事件.....       | 11 |
| 4.2.4 清除指定的事件.....    | 12 |



|                      |           |
|----------------------|-----------|
| 4.2.5 删除事件.....      | 12        |
| <b>5 信号量管理.....</b>  | <b>13</b> |
| 5.1 API 对比.....      | 13        |
| 5.2 API 说明.....      | 13        |
| 5.2.1 创建信号量.....     | 14        |
| 5.2.2 创建二值信号量.....   | 14        |
| 5.2.3 创建互斥量.....     | 14        |
| 5.2.4 删除互斥量/信号量..... | 15        |
| 5.2.5 获取互斥量/信号量..... | 15        |
| 5.2.6 释放互斥量/信号量..... | 16        |
| <b>6 定时器.....</b>    | <b>17</b> |
| 6.1 API 对比.....      | 17        |
| 6.2 API 说明.....      | 17        |
| 6.2.1 创建定时器.....     | 17        |
| 6.2.2 启动定时器.....     | 18        |
| 6.2.3 停止定时器.....     | 18        |
| 6.2.4 删除定时器.....     | 19        |



# 1 任务管理

- 1.1 API对比
- 1.2 API说明

## 1.1 API 对比

| 功能      | OSA层API   | FreeRTOS API   |
|---------|---|--|
| 动态创建任务  | hi_task_create  | xTaskCreate  |
| 静态创建任务  | 不支持   | xTaskCreateStatic  |
| 任务删除    | hi_task_delete  | vTaskDelete  |
| 获取任务名称  | hi_task_get_current_id或<br>hi_task_create中获取任<br>务ID，软件可以通过任务<br>ID查找任务名称 | pcTaskGetName  |
| 获取任务优先级 | hi_task_get_priority  | uxTaskPriorityGet  |
| 设置任务优先级 | hi_task_set_priority  | vTaskPrioritySet   |
| 挂起任务    | hi_task_suspend   | vTaskSuspend   |
| 挂起所有任务  |   | vTaskSuspendAll  |
| 任务恢复    | hi_task_resume  | vTaskResume  |
| 锁定任务    | hi_task_lock  | taskENTER_CRITICAL：进入临界段<br>taskENTER_CRITICAL_FROM_ISR：<br>进入临界（中断版本）<br>taskDISABLE_INTERRUPTS：关中断 |



| 功能   | OSA层API        | FreeRTOS API   |
|------|----------------|--|
| 解锁任务 | hi_task_unlock | taskEXIT_CRITICAL: 退出临界<br>taskEXIT_CRITICAL_FROM_ISR: 退出临界（中断版本）<br>taskENABLE_INTERRUPTS: 使能中断 |
| 任务睡眠 | hi_sleep       | vTaskDelay   |

## 1.2 API 说明

### 1.2.1 创建任务

|      |   |  |
|------|---|--|
| 函数原型 | hi_u32<br>hi_task_create(hi_u32*<br>taskid, const hi_task_attr*<br>attr, hi_void *<br>(*task_route)(hi_void*),<br>hi_void* arg) | BaseType_t xTaskCreate(<br>TaskFunction_t pxTaskCode,<br>const char * const pcName,<br>const uint16_t usStackDepth,<br>void * const pvParameters,<br>UBaseType_t uxPriority,<br>TaskHandle_t * const pxCreatedTask ) |
| 返回值  | HI_ERR_SUCCESS: 任务创建成功。<br>其他: 任务创建失败。  | pdPASS: 任务创建成功。<br>errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY: 任务创建失败, 堆内存不足。   |
| 入参   | task_route: 任务函数。   | pxTaskCode: 任务函数。  |
|      | attr: 任务属性, 包括: 任务名称、任务优先级、任务栈大小。   | pcName: 任务名称, 长度 ≤ configMAX_TASK_NAME_LEN。  |
|      |   | usStackDepth: 任务堆栈大小。  |
|      |   | uxPriority: 任务优先级。   |
|      | arg: 传递给任务函数的参数。  | pvParameters: 传递给任务函数的参数。  |
|      | taskid: 任务ID。任务创建成功后, 返回此任务的ID（任务管理的API会使用此ID）。   | pxCreatedTask: 任务句柄。任务创建成功后, 返回此任务的句柄（即此任务的堆栈）。此参数用于保存该任务句柄, 其他API函数可能会使用到该任务句柄。   |





## 1.2.2 删除任务

|      |                                      |  |
|------|--------------------------------------|--|
| 函数原型 | hi_u32 hi_task_delete(hi_u32 taskid) | Void vTaskDelete(TaskHandle_t xTaskToDelete) |
| 返回值  | HI_ERR_SUCCESS: 删除任务成功。<br>其他: 失败。   | 无  |
| 入参   | taskid: 任务ID。                        | xTaskToDelete: 任务句柄。                         |

## 1.2.3 获取任务名称

|      |   |   |
|------|---|---|
| 函数原型 | 无 | char*pcTaskGetName(TaskHandle_t xTaskToQuery) |
| 返回值  | 无 | SUCCESS: 返回任务名称。<br>FAIL: 返回NULL。             |
| 入参   | 无 | xTaskToQuery: 任务句柄。                           |

## 1.2.4 查询某个任务的优先级

|      |  |   |
|------|--|---|
| 函数原型 | hi_u32 hi_task_get_priority(hi_u32 taskid, hi_u32 *priority) | UBaseType_t uxTaskPriorityGet(const TaskHandle_t xTask) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。                               | 返回任务优先级。  |
| 入参   | taskid: 任务ID。  | xTask: 任务句柄。  |
|      | priority: 返回任务优先级。   |   |

## 1.2.5 设置某个任务的优先级

|      |   |  |
|------|---|--|
| 函数原型 | hi_u32 hi_task_set_priority(hi_u32 taskid, hi_u32 priority) | void vTaskPrioritySet(TaskHandle_t xTask, UBaseType_t uxNewPriority) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。                              | 无  |
| 入参   | taskid: 任务ID。   | xTask: 任务句柄。   |



|  |                     |                          |
|--|---------------------|--------------------------|
|  | priority: 设置的任务优先级。 | uxNewPriority: 设置的任务优先级。 |
|--|---------------------|--------------------------|

## 1.2.6 挂起任务

|      |                                       |  |
|------|---------------------------------------|--|
| 函数原型 | hi_u32 hi_task_suspend(hi_u32 taskid) | Void<br>vTaskSuspend(TaskHandle_t<br>xTaskToSuspend) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。        | 无  |
| 入参   | taskid: 任务ID。                         | xTaskToSuspend: 任务句柄。                                |

## 1.2.7 恢复任务

|      |                                      |  |
|------|--------------------------------------|--|
| 函数原型 | hi_u32 hi_task_resume(hi_u32 taskid) | void vTaskResume(TaskHandle_t<br>xTaskToResume ) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。       | 无  |
| 入参   | taskid: 任务ID。                        | xTaskToResume: 任务句柄。                             |

## 1.2.8 任务睡眠

|      |                                |  |
|------|--------------------------------|--|
| 函数原型 | hi_u32 hi_sleep(hi_u32 ms)     | void vTaskDelay(const<br>TickType_t xTicksToDelay) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。 | 无  |
| 入参   | ms: 睡眠时间 (单位: ms)。             | xTicksToDelay: 睡眠时间<br>(单位: tick)。                 |



# 2 内存管理

2.1 API对比

2.2 API说明

## 2.1 API 对比

| 功能             | OSA层API             | FreeRTOS API                    |
|----------------|---------------------|---------------------------------|
| 申请动态内存         | hi_malloc           | pvPortMalloc                    |
| 释放动态内存         | hi_free             | vPortFree                       |
| 初始化内存堆函数       | 不需要，无需用户初始化内存管理     | vPortInitialiseBlocks           |
| 获取当前未分配的内存堆大小  | hi_mem_get_sys_info | xPortGetFreeHeapSize            |
| 获取未分配的内存堆历史最小值 | 不支持                 | xPortGetMinimumEverFreeHeapSize |

## 2.2 API 说明

### 2.2.1 申请动态内存

|      |  |                                    |
|------|--|------------------------------------|
| 函数原型 | hi_pvoid hi_malloc(hi_u32 mod_id, hi_u32 size) | void *pvPortMalloc( size_t xSize ) |
| 返回值  | SUCCESS：返回内存地址。<br>FAIL：返回NULL。                | SUCCESS：返回内存地址。<br>FAIL：返回NULL。    |
| 入参   | mod_id：内存标识（未使用）。                              | 无                                  |



|  |                         |                          |
|--|-------------------------|--------------------------|
|  | size: 申请内存的长度（单位：byte）。 | xSize: 申请内存的长度（单位：byte）。 |
|--|-------------------------|--------------------------|

### 2.2.2 释放申请的内存

|      |   |                            |
|------|---|----------------------------|
| 函数原型 | hi_void hi_free(hi_u32 mod_id, hi_pvoid addr) | void vPortFree( void *pv ) |
| 返回值  | 无   | 无                          |
| 入参   | mod_id: 内存标识（未使用）。                            | 无                          |
|      | addr: 要释放内存的地址。                               | pv: 要释放内存的地址。              |



# 3 消息队列管理

- 3.1 API对比
- 3.2 API说明

## 3.1 API 对比

| 功能           | OSA层API              | FreeRTOS API             |
|--------------|----------------------|--------------------------|
| 动态创建队列       | hi_msg_queue_create  | xQueueCreate             |
| 删除队列         | hi_msg_queue_delete  | vQueueDelete             |
| 发送元素到队列      | hi_msg_queue_send    | xQueueSend               |
| 从中断发送元素到队列   | 不支持                  | xQueueSendFromISR        |
| 发送元素到队列尾部    | 不支持                  | xQueueSendToBack         |
| 从中断发送元素到队列尾部 | 不支持                  | xQueueSendToBackFromISR  |
| 从队列接收一个消息    | hi_msg_queue_wait    | xQueueReceive            |
| 从中断中队列接收一个消息 | 不支持                  | xQueueReceiveFromISR     |
| 检查消息队列是否已满   | hi_msg_queue_is_full | xQueueIsQueueFullFromISR |

## 3.2 API 说明



### 3.2.1 动态创建队列

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32<br>hi_msg_queue_create(HI_OUT<br>hi_u32* id, hi_u16 queue_len,<br>hi_u32 msg_size) | QueueHandle_t<br>xQueueCreate( UBaseType_t<br>uxQueueLength, UBaseType_t<br>uxItemSize ); |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。  | 返回消息队列句柄<br>QueueHandle_t。  |
| 入参   | id: 消息队列ID。   | 无   |
|      | queue_len: 消息队列长度 ( 即该消息队列支持存储多少条消息 )。  | uxQueueLength: 消息队列长度 ( 单位: byte )。   |
|      | msg_size: 每个消息的大小 ( 单位: byte )。   | uxItemSize: 每个消息的大小 ( 单位: byte )。   |

### 3.2.2 删除消息队列

|      |  |   |
|------|--|---|
| 函数原型 | hi_u32<br>hi_msg_queue_delete(hi_u32 id) | void<br>vQueueDelete( QueueHandle_t<br>xQueue ) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。           | 无   |
| 入参   | id: 消息队列ID。                              | xQueue: 消息队列句柄。                                 |

### 3.2.3 发送消息到消息队列

|      |  |   |
|------|--|---|
| 函数原型 | hi_u32<br>hi_msg_queue_send(hi_u32 id,<br>hi_pvoid msg, hi_u32<br>timeout_ms, hi_u32 msg_size) | BaseType_t<br>xQueueSend(QueueHandle_t<br>xQueue,const void *<br>pvltemToQueue, TickType_t<br>xTicksToWait ); |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。   | pdTRU: 消息发送成功。<br>errQUEUE_FULL: 消息发送失败。  |
| 入参   | id: 消息队列ID。  | xQueue: 消息队列句柄。   |
|      | msg: 消息数据地址。   | pvltemToQueue: 消息数据地址。  |



|  |                                |  |
|--|--------------------------------|--|
|  | timeout_ms: 消息发送最大超时时间（单位：ms）。 | xTicksToWait: 消息队列已经满时，等待消息队列有空间时的最大等待时间（单位：系统时钟节拍）。 |
|  | msg_size: 消息长度（单位：byte）。       | 无（在创建时已固定消息长度）。                                      |

### 3.2.4 接收消息队列中的数据

|      |  |  |
|------|--|--|
| 函数原型 | hi_u32<br>hi_msg_queue_wait(hi_u32 id, hi_pvoid msg, hi_u32 timeout_ms, hi_u32 msg_size) | BaseType_t<br>xQueueReceive(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。   | pdTRUE: 消息发送成功。<br>errQUEUE_FULL: 消息发送失败。  |
| 入参   | id: 消息队列ID。  | xQueue: 消息队列句柄。  |
|      | msg: 消息队列中复制出数据后所储存的缓冲地址，缓冲区空间须≥消息队列创建函数 hi_msg_queue_create 所指定的单个消息大小。                 | pvBuffer: 消息队列中复制出数据后所储存的缓冲地址（缓冲区空间须≥消息队列创建函数 xQueueCreate 所指定的单个消息大小）。                    |
|      | timeout_ms: 消息队列为空时，等待消息队列有数据的最大等待时间（单位：ms）。   | xTicksToWait: 消息队列为空时，等待消息队列有数据的最大等待时间（单位：系统时钟节拍）。   |
|      | msg_size: 消息长度（单位：byte）。   | 无（在创建时已固定消息长度）。  |

### 3.2.5 检查消息队列是否已满

|      |  |  |
|------|--|--|
| 函数原型 | hi_bool<br>hi_msg_queue_is_full(hi_u32 id) | BaseType_t<br>xQueueIsQueueFullFromISR( const QueueHandle_t xQueue ) |
| 返回值  | HI_TRUE: 消息队列已满。<br>HI_FALSE: 消息队列未满。      | pdTRUE: 消息队列已满。<br>pdFALSE: 消息队列未满。                                  |
| 入参   | id: 消息队列ID。                                | xQueue: 消息队列句柄。  |



# 4 事件管理

- 4.1 API对比
- 4.2 API说明

## 4.1 API 对比

| 功能   | OSA层API                | FreeRTOS API                                 |
|------|------------------------|--|
| 创建事件 | hi_event_init: 初始化事件链表 | xEventGroupCreate: 创建事件组                     |
|      | hi_event_create : 创建事件 |  |
| 发送事件 | hi_event_send          | xEventGroupSetBits: 设置指定的事件标志位为1             |
|      |                        | xEventGroupSetBitsFromISR: 设置指定的事件标志位为1（中断版） |
| 等待事件 | hi_event_wait          | xEventGroupWaitBits: 设置指定的事件标志位为1            |
|      |                        | xEventGroupGetBitsFromISR: 设置指定的事件标志位为1（中断版） |
| 清除事件 | hi_event_clear         | xEventGroupClearBits: 指定的事件位清0               |
|      |                        | xEventGroupClearBitsFromISR: 指定的事件位清0（中断版）   |
| 删除事件 | hi_event_delete        | vEventGroupDelete: 删除事件标志组                   |

## 4.2 API 说明





## 4.2.1 创建事件

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32 hi_event_create(HI_OUT hi_u32 *id) | EventGroupHandle_t<br>xEventGroupCreate(void) |
| 描述   | 创建事件，获取事件使用ID。                            | 创建事件，获取事件句柄。                                  |
| 返回值  | HI_ERR_SUCCESS：成功。<br>其他：失败。              | EventGroupHandle_t：返回事件。<br>NULL：创建事件组失败。     |
| 入参   | id：事件ID。                                  | xQueue：消息队列句柄。                                |

## 4.2.2 发送事件

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32 hi_event_send(hi_u32 id, hi_u32 event_bits); | EventBits_t<br>xEventGroupSetBits(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet ) |
| 描述   | 发送事件。   | 发送事件，将对应事件的bit置1。   |
| 返回值  | HI_ERR_SUCCESS：成功。<br>其他：失败。                        | EventBits_t：返回当前的事件标志组数值。   |
| 入参   | id：事件ID。  | xEventGroup：事件组句柄。  |
|      | event_bits：事件bit位，支持24个可设置的事件标志位。                   | uxBitsToSet：表示24个可设置的事件标志位。EventBits_t为定义的32bit变量，低24bit用于设置事件标志。                                 |

## 4.2.3 等待事件

|      |  |   |
|------|--|---|
| 函数原型 | hi_u32 hi_event_wait(hi_u32 id, hi_u32 mask, HI_OUT hi_u32 *event_bits, hi_u32 timeout, hi_u32 flag) | EventBits_t<br>xEventGroupWaitBits(const EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToWaitFor, const BaseType_t xClearOnExit, const BaseType_t xWaitForAllBits, TickType_t xTicksToWait ); |
| 返回值  | HI_ERR_SUCCESS：成功。<br>其他：失败。   | EventBits_t：返回当前的事件标志组数值。   |



|    |  |   |
|----|--|---|
| 入参 | id: 事件ID。  | xEventGroup: 事件组句柄。   |
|    | mask: 预等待的事件集合。  | uxBitsToWaitFor: 等待24个事件标志位中的指定标志。                            |
|    | flag: 事件等待标志。 <ul style="list-style-type: none"> <li>HI_EVENT_WAITMODE_AND: 等待所有事件都置位;</li> <li>HI_EVENT_WAITMODE_OR: 等待任一事件置位;</li> <li>HI_EVENT_WAITMODE_CLR: 已等到事件标志位清零, 配合以上两个选项使用。</li> </ul> | xClearOnExit: 选择是否清除已经被置位的事件标志。                               |
|    |  | xWaitForAllBits: 是否等待所有的标志位都被设置。                              |
|    | timeout: 设置等待时间 (单位: ms)   | xTicksToWait: 设置等待时间 (单位: 时钟节拍周期)。如果设置为portMAX_DELAY, 表示永久等待。 |
|    | event_bits: 返回事件的bit位。   | 无   |

## 4.2.4 清除指定的事件

|      |   |  |
|------|---|--|
| 函数原型 | hi_u32<br>hi_event_clear(hi_u32 id,<br>hi_u32 event_bits) | EventBits_t<br>xEventGroupClearBits(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToClear) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。                            | EventBits_t: 返回当前的事件标志组数值。   |
| 入参   | id: 事件ID。   | xEventGroup: 事件组句柄。  |
|      | event_bits: 清除指定的事件位。                                     | uxBitsToClear: 清除指定的事件位。   |

## 4.2.5 删除事件

|      |                                   |   |
|------|-----------------------------------|---|
| 函数原型 | hi_u32 hi_event_delete(hi_u32 id) | void<br>vEventGroupDelete(EventGroupHandle_t xEventGroup) |
| 描述   | 删除事件。                             | 删除事件组。  |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。    | 无。  |
| 入参   | id: 事件ID。                         | xEventGroup: 事件组句柄。                                       |



# 5 信号量管理

## 5.1 API对比

## 5.2 API说明

## 5.1 API 对比

| 功能      | OSA层API              | FreeRTOS API  |
|---------|----------------------|---|
| 创建计数信号量 | hi_sem_create        | xSemaphoreCreateCounting  |
| 创建二值信号量 | hi_sem_bcreate       | xSemaphoreCreateBinary  |
| 创建互斥量   | hi_mux_create: 创建互斥锁 | vSemaphoreCreateMutex: 创建互斥型信号量   |
| 删除信号量   | hi_sem_delete        | vSemaphoreDelete  |
| 删除互斥锁   | hi_mux_delete        | 无   |
| 等待信号量   | hi_sem_wait          | xSemaphoreTake: 获取信号量<br>xSemaphoreTakeRecursive: 递归互斥量时获取<br>xSemaphoreTakeFromISR: 获取信号量 (中断版)  |
| 等待互斥量   | hi_mux_pend          |   |
| 释放信号量   | hi_sem_signal        | xSemaphoreGive: 释放信号量<br>xSemaphoreGiveRecursive: 释放一个递归互斥量<br>xSemaphoreGiveFromISR: 释放信号量 (中断版) |
| 释放互斥量   | hi_mux_post          |   |

## 5.2 API 说明



## 5.2.1 创建信号量

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32 hi_sem_create(hi_u32 *sem_id, hi_u16 init_value) | SemaphoreHandle_t<br>xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount) |
| 描述   | 创建信号量。  | 创建计数信号量。  |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。                          | 成功: 信号量句柄。<br>失败: 返回NULL。   |
| 入参   | id: 返回信号量ID。  | 无   |
|      | init_value: 有效信号的初始化个数, 范围: [0, 0xFFFF]。                | uxMaxCount: 最大计数量。  |
|      | 无   | uxInitialCount: 信号量的初始值。当此信号量用于事件计数时, 值应为0; 当用于资源管理时, 应设置为与uxMaxCount的值一致。                        |

## 5.2.2 创建二值信号量

|      |  |   |
|------|--|---|
| 函数原型 | hi_u32 hi_sem_bcreate(hi_u32 *sem_id, hi_u8 init_value);                     | SemaphoreHandle_t<br>xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。   | 成功: 信号量句柄。<br>失败: 返回NULL。   |
| 入参   | id: 返回信号量ID。   | 无   |
|      | 无  | uxMaxCount: 最大计数量。  |
|      | init_value: 信号量初始值。一般情况下, 当值为HI_SEM_ONE时, 用于临界资源保护; 当值为HI_SEM_ZERO时, 用于线程同步。 | uxInitialCount: 信号量的初始值。当此信号量用于事件计数时, 值应为0; 当用于资源管理时, 应设置为与uxMaxCount的值一致。                        |

## 5.2.3 创建互斥量

|      |                                       |  |
|------|---------------------------------------|--|
| 函数原型 | hi_u32 hi_mux_create (hi_u32* mux_id) | SemaphoreHandle_t<br>xSemaphoreCreateMutex( void ) |
|------|---------------------------------------|--|



|     |                                |                           |
|-----|--------------------------------|---------------------------|
| 返回值 | HI_ERR_SUCCESS: 成功。<br>其他: 失败。 | 成功: 互斥量句柄。<br>失败: 返回NULL。 |
| 入参  | id: 返回互斥量ID。                   | 无                         |

## 5.2.4 删除互斥量/信号量

|      |  |  |
|------|--|--|
| 函数原型 | hi_u32 hi_mux_delete(hi_u32<br>mux_id)<br>hi_u32 hi_sem_delete(hi_u32<br>sem_id) | void<br>vSemaphoreDelete( SemaphoreHandle_t xSemaphore ) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。   | 无  |
| 入参   | id: 互斥量ID/信号量ID。   | xSemaphore: 互斥量句柄/<br>信号量句柄。                             |

## 5.2.5 获取互斥量/信号量

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32<br>hi_mux_pend(hi_u32<br>mux_id, hi_u32<br>timeout_ms)<br>hi_u32 hi_sem_wait(hi_u32<br>sem_id, hi_u32 timeout) | xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait )   |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。  | pdTRUE: 获取成功。<br>pdFALSE: 时间超时后, 还未得到信号量。   |
| 入参   | mux_id/ sem_id: 互斥量ID/<br>信号量ID。  | xSemaphore: 信号量的句柄。   |
|      | timeout_ms/timeout: 等待时间 (单位: ms)。当设置为HI_SYS_WAIT_FOREVER时, 永久阻塞等待。   | xTicksToWait: 阻塞等待时间。当值为portTICK_PERIOD_MSs时表示将ms单位转换成tick, 如果INCLUDE_vTaskSuspend = 1, 该值设为portMAX_DELAY, 则会限阻塞等待。 |



## 5.2.6 释放互斥量/信号量

|      |  |  |
|------|--|--|
| 函数原型 | hi_u32 hi_mux_post(hi_u32 mux_id)<br>hi_u32 hi_sem_signal(hi_u32 sem_id) | xSemaphoreGive( SemaphoreHandle_t xSemaphore ) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。   | pdTRUE: 获取成功。<br>pdFALSE: 释放出现错误。              |
| 入参   | mux_id/ sem_id: 互斥量ID/信号量ID。   | xSemaphore: 信号量的句柄。                            |



# 6 定时器

- 6.1 API对比
- 6.2 API说明

## 6.1 API 对比

| 功能    | OSA层API         | FreeRTOS API                   |
|-------|-----------------|--------------------------------|
| 创建定时器 | hi_timer_create | xTimerCreate                   |
| 开启定时器 | hi_timer_start  | xTimerStart: 开启定时器             |
|       |                 | xTimerStartFromISR: 开启定时器（中断版） |
| 停止定时器 | hi_timer_stop   | xTimerStop: 停止定时器              |
|       |                 | xTimerStopFromISR: 停止定时器（中断版）  |
| 删除定时器 | hi_timer_delete | xTimerDelete                   |

## 6.2 API 说明

### 6.2.1 创建定时器

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32<br>hi_timer_create(hi_u32 *timer_handle) | TimerHandle_t xTimerCreate ( const char *<br>const pcTimerName, const TickType_t<br>xTimerPeriod, const UBaseType_t<br>uxAutoReload, void * const pvTimerID,<br>TimerCallbackFunction_t<br>pxCallbackFunction ) |
|------|---|---|



|     |                                |  |
|-----|--------------------------------|--|
| 返回值 | HI_ERR_SUCCESS: 成功。<br>其他: 失败。 | 成功: 返回定时器句柄。<br>失败: 返回NULL。  |
| 入参  | timer_handle: 返回定时器句柄。         | 无  |
|     | 无                              | pcTimerName: 定时器名称。  |
|     | 无                              | xTimerPeriod: 计时器周期 ( 必须 > 0 ) 。   |
|     | 无                              | uxAutoReload: 定时器触发标志。<br><ul style="list-style-type: none"> <li>pdTRUE: 以xTimerPeriod周期触发定时器。</li> <li>pdFALSE: 只触发一次。</li> </ul> |
|     | 无                              | pvTimerID: 分配给正在创建的计时器ID。  |
|     | 无                              | pxCallbackFunction: 定时器回调函数。   |

## 6.2.2 启动定时器

|      |  |  |
|------|--|--|
| 函数原型 | hi_u32 hi_timer_start(hi_u32 timer_handle, hi_timer_type type, hi_u32 expire, hi_timer_callback_f timer_func, hi_u32 data) | BaseType_t<br>xTimerStart( TimerHandle_t xTimer, TickType_t xBlockTime ) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。   | pdFAIL: 启动定时器失败。<br>pdPASS: 启动定时器成功。                                     |
| 入参   | timer_handle: 定时器句柄。   | xTimer: 定时器句柄。   |
|      | type: 定时器类型。   | 无  |
|      | expire: 定时器超时时间 ( 单位: ms ) 。配置为0时, 默认为10ms。  | xBlockTime: 定时器超时时间 ( 单位: ticks ) 。                                      |
|      | timer_func: 定时器回调函数。   | 对应xTimerCreate中的回调函数。  |
|      | data: 回调函数传参。  | 无  |

## 6.2.3 停止定时器

|      |   |   |
|------|---|---|
| 函数原型 | hi_u32 hi_timer_stop(hi_u32 timer_handle) | BaseType_t<br>xTimerStop( TimerHandle_t xTimer, TickType_t xBlockTime ) |
|------|---|---|





|     |                                |                                      |
|-----|--------------------------------|--------------------------------------|
| 返回值 | HI_ERR_SUCCESS: 成功。<br>其他: 失败。 | pdFAIL: 停止定时器失败。<br>pdPASS: 停止定时器成功。 |
| 入参  | timer_handle: 定时器句柄。           | xTimer: 定时器句柄。                       |
|     | 无                              | xBlockTime: 等待定时器停止时间 (单位: ticks)。   |

## 6.2.4 删除定时器

|      |   |  |
|------|---|--|
| 函数原型 | hi_u32<br>hi_timer_delete(hi_u32<br>timer_handle) | BaseType_t<br>xTimerDelete( TimerHandle_t xTimer,<br>TickType_t xBlockTime ) |
| 返回值  | HI_ERR_SUCCESS: 成功。<br>其他: 失败。                    | pdFAIL: 删除定时器失败。<br>pdPASS: 删除定时器成功。   |
| 入参   | timer_handle: 定时器句柄。                              | xTimer: 定时器句柄。   |
|      | 无   | xBlockTime: 等待定时器删除时间 (单位: ticks)。   |