Hi3861 V100 / Hi3861L V100 OSA&FreeRTOS API

Adaptation Guide

Issue	01
Date	2020-04-30

Trademarks and Permissions

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: https://www.hisilicon.com/en/

Email: support@hisilicon.com

About This Document

Purpose

This document describes the adaptation between the OSA APIs and FreeRTOS APIs of Hi3861 V100 and Hi3861L V100.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3861	V100
Hi3861L	V100

Intended Audience

The document is intended for:

- Technical support engineers
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
⚠ DANGER	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.

Symbol	Description
⚠ CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
🕮 NOTE	Supplements the important information in the main text. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Issue	Date	Change Description
01	2020-04-30	This issue is the first official release.
00B01	2020-04-03	This issue is the first draft release.

Contents

1 Task Management

1.1 API Comparison

Function	OSA API	FreeRTOS API
Create a task in a dynamic way.	hi_task_create	xTaskCreate
Create a task in a static way.	Not supported	xTaskCreateStatic
Delete a task.	hi_task_delete	vTaskDelete
Obtain the task name.	Obtain the task ID from hi_task_get_current_id or hi_task_create. Software can search for the task name based on the task ID.	pcTaskGetName
Obtain the task priority.	hi_task_get_priority	uxTaskPriorityGet
Set the task priority.	hi_task_set_priority	vTaskPrioritySet
Suspend a task.	hi_task_suspend	vTaskSuspend
Suspend all tasks.		vTaskSuspendAl
Resume a task.	hi_task_resume	vTaskResume

Function	OSA API	FreeRTOS API
Lock a task.	hi_task_lock	**taskENTER_CRITICAL**: Enters the critical section. **taskENTER_CRITICAL_FROM_ISR**: Enters the critical section (interrupt version). **taskDISABLE_INTERRUPTS**: Disables the interrupt.
Unlock a task.	hi_task_unlock	**taskEXIT_CRITICAL**: Exits the critical section. **taskEXIT_CRITICAL_FROM_ISR**: Exits the critical section (interrupt version). **taskENABLE_INTERRUPTS**: Enables the interrupt.
Implement task sleep.	hi_sleep	vTaskDelay

1.2 API Description

1.2.1 Creating a Task

Function Prototype	hi_u32 hi_task_create(hi_u32* taskid, const hi_task_attr* attr, hi_void * (*task_route)(hi_void*), hi_void* arg)	BaseType_t xTaskCreate(TaskFunction_t pxTaskCode, const char * const pcName, const uint16_t usStackDepth, void * const pvParameters, UBaseType_t uxPriority, TaskHandle_t * const pxCreatedTask)
Return Value	**HI_ERR_SUCCESS**: A task is created successfully. Other values: A task fails to be created.	**pdPASS**: A task is created successfully. **errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY**: A task fails to be created due to insufficient heap memory.
Input Argument	**task_route**: task function	**pxTaskCode**: task function
	attr: task attributes, including the task name, task priority, and task stack size	**pcName**: task name. The length is less than or equal to **configMAX_TASK_NAME_LEN**.
		usStackDepth: task stack size
		uxPriotiry: task priority

	arg: arguments passed to a task function	**pvParameters**: arguments passed to a task function
	taskid: task ID. After a task is created, the task ID is returned. This ID is used by the task management API.	**pxCreatedTask**: task handle. After a task is created, the task handle is returned. This task handle is also the task stack. This argument is used to save the task handle, which may be used by other API functions.

1.2.2 Deleting a Task

Function Prototype	hi_u32 hi_task_delete(hi_u32 taskid)	Void vTaskDelete(TaskHandle_t xTaskToDelete)
Return Value	**HI_ERR_SUCCESS**: A task is deleted successfully. Other values: failure	None
Input Argument	**taskid**: task ID	**xTaskToDelete**: task handle

1.2.3 Obtaining the Task Name

Function Prototype	None	char*pcTaskGetName(TaskHandle_t xTaskToQuery)
Return Value	None	**SUCCESS**: The task name is returned. **FAIL**: NULL is returned.
Input Argument	None	**xTaskToQuery**: task handle

1.2.4 Querying the Task Priority

Function Prototype	hi_u32 hi_task_get_priority(hi_u32 taskid, hi_u32 *priority)	UBaseType_t uxTaskPriorityGet(const TaskHandle_t xTask)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	The task priority is returned.

Input Argument	**taskid**: task ID	**xTask**: task handle
	priority: task priority	

1.2.5 Setting the Task Priority

Function Prototype	hi_u32 hi_task_set_priority(hi_u32 taskid, hi_u32 priority)	void vTaskPrioritySet(TaskHandle_t xTask, UBaseType_t uxNewPriority)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**taskid**: task ID	**xTask**: task handle
	priority: task priority	**uxNewPriority**: task priority

1.2.6 Suspending a Task

Function Prototype	hi_u32 hi_task_suspend(hi_u32 taskid)	Void vTaskSuspend(TaskHandle_t xTaskToSuspend)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**taskid**: task ID	**xTaskToSuspend**: task handle

1.2.7 Resuming a Task

Function Prototype	hi_u32 hi_task_resume(hi_u32 taskid)	void vTaskResume(TaskHandle_t xTaskToResume)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**taskid**: task ID	**xTaskToResume**: task handle

1.2.8 Implementing Task Sleep

Function Prototype	hi_u32 hi_sleep(hi_u32 ms)	void vTaskDelay(const TickType_t xTicksToDelay)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**ms**: sleep time (unit: ms)	**xTicksToDelay**: sleep time (unit: tick)

2 Memory Management

2.1 API Comparison

Function	OSA API	FreeRTOS API
Allocate dynamic memory.	hi_malloc	pvPortMalloc
Free up dynamic memory.	hi_free	vPortFree
Initialize the memory heap function.	This function is not required.	vPortInitialiseBlocks
Obtain the size of the free memory heap.	hi_mem_get_sys_info	xPortGetFreeHeapSize
Obtain the historical minimum value of the free memory heap.	Not supported	xPortGetMinimumEverFree-HeapSize

2.2 API Description

2.2.1 Allocate Dynamic Memory

Function Prototype	hi_pvoid hi_malloc(hi_u32 mod_id, hi_u32 size)	void *pvPortMalloc(size_t xSize)
Return Value	**SUCCESS**: The memory address is returned. **FAIL**: NULL is returned.	**SUCCESS**: The memory address is returned. **FAIL**: NULL is returned.
Input Argument	**mod_id**: memory ID (reserved)	None
	size: length of the memory to be allocated (unit: byte)	**xSize**: length of the memory to be allocated (unit: byte)

2.2.2 Freeing Up Allocated Memory

Function Prototype	hi_void hi_free(hi_u32 mod_id, hi_pvoid addr)	void vPortFree(void *pv)
Return Value	None	None
Input Argument	**mod_id**: memory ID (reserved)	None
	addr: address of the memory to be freed	**pv**: address of the memory to be freed

3 Message Queue Management

3.1 API Comparison

Function	OSA API	FreeRTOS API
Create a queue dynamically.	hi_msg_queue_create	xQueueCreate
Delete a queue.	hi_msg_queue_delete	vQueueDelete
Send elements to a queue.	hi_msg_queue_send	xQueueSend
Send elements from an interrupt to a queue.	Not supported	xQueueSendFromISR
Send elements to the tail of a queue.	Not supported	xQueueSendToBack
Send elements from an interrupt to the tail of a queue.	Not supported	xQueueSendToBackFromISR
Receive a message from a queue.	hi_msg_queue_wait	xQueueReceive
Receive a message from a queue in an interrupt.	Not supported	xQueueReceiveFromISR
Check whether a message queue is full.	hi_msg_queue_is_full	xQueueIsQueueFullFromISR

3.2 API Description

3.2.1 Creating a Queue Dynamically

Function Prototype	hi_u32 hi_msg_queue_create(HI_OUT hi_u32* id, hi_u16 queue_len, hi_u32 msg_size)	QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength, UBaseType_t uxItemSize);
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	The message queue handle **QueueHandle_t** is returned.
Input Argument	**id**: message queue ID	None
	queue_len: length of a message queue, that is, the number of messages that can be stored in a message queue	**uxQueueLength**: length of a message queue (unit: byte)
	msg_size: size of each message (unit: byte)	**uxItemSize**: size of each message (unit: byte)

3.2.2 Deleting a Message Queue

Function Prototype	hi_u32 hi_msg_queue_delete(hi_u32 id)	void vQueueDelete(QueueHandle_t xQueue)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**id**: message queue ID	**xQueue**: message queue handle

3.2.3 Sending a Message to a Message Queue

Function Prototype	hi_u32 hi_msg_queue_send(hi_u32 id, hi_pvoid msg, hi_u32 timeout_ms, hi_u32 msg_size)	BaseType_t xQueueSend(QueueHandle_t xQueue,const void * pvItemToQueue,TickType_t xTicksToWait);

Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdTRU**: A message is successfully sent. **errQUEUE_FULL**: A message fails to be sent.
Input Argument	**id**: message queue ID	**xQueue**: message queue handle
	msg: message data address	**pvItemToQueue**: message data address
	timeout_ms: maximum timeout for sending messages (unit: ms)	**xTicksToWait**: maximum waiting time (unit: system clock cycle) when the message queue is full but the waiting message queue has available space.
	msg_size: message length (unit: byte)	None (The message length is fixed when a message is created.)

3.2.4 Receiving Data from a Message Queue

Function Prototype	hi_u32 hi_msg_queue_wait(hi_u32 id, hi_pvoid msg, hi_u32 timeout_ms, hi_u32 msg_size)	BaseType_t xQueueReceive(QueueHandle_t xQueue, void *pvBuffer,TickType_t xTicksToWait)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdTRU**: A message is successfully sent. **errQUEUE_FULL**: A message fails to be sent.
Input Argument	**id**: message queue ID	**xQueue**: message queue handle
	msg: buffer address for storing the copied data in the message queue. The size of the buffer space must be greater than or equal to that of a single message specified by **hi_msg_queue_create**.	**pvBuffer**: buffer address for storing the copied data in the message queue. The size of the buffer space must be greater than or equal to that of a single message specified by **xQueueCreate**.
	timeout_ms: maximum waiting time (unit: ms) when the message queue is empty but the waiting message queue has data	**xTicksToWait**: maximum waiting time (unit: system clock beat) when the message queue is empty but the waiting message queue has data

	msg_size: message length (unit: byte)	None (The message length is fixed when a message is created.)

3.2.5 Checking Whether a Message Queue Is Full

Function Prototype	hi_bool hi_msg_queue_is_full(hi_u32 id)	BaseType_t xQueueIsQueueFullFromISR(const QueueHandle_t xQueue)
Return Value	**HI_TRUE**: The message queue is full. **HI_FALSE**: The message queue is not full.	**pdTRUE**: The message queue is full. **pdFALSE**: The message queue is not full.
Input Argument	**id**: message queue ID	**xQueue**: message queue handle

4 Event Management

4.1 API Comparison

Function	OSA API	FreeRTOS API
Create an event.	**hi_event_init**: Initializes the event linked list. **hi_event_create**: Creates an event.	**xEventGroupCreate**: Creates an event group.
Send an event.	hi_event_send	**xEventGroupSetBits**: Sets the specified event flag bits to **1**. **xEventGroupSetBitsFromISR**: Sets the specified event flag bits to **1** (interrupt version).
Wait for an event.	hi_event_wait	**xEventGroupWaitBits**: Sets the specified event flag bits to **1**. **xEventGroupGetBitsFromISR**: Sets the specified event flag bits to **1** (interrupt version).
Clear an event.	hi_event_clear	**xEventGroupClearBits**: Clears the specified event bits. **xEventGroupClearBitsFromISR**: Clears the specified event bits (interrupt version).
Delete an event.	hi_event_delete	**vEventGroupDelete**: Deletes an event group.

4.2 API Description

4.2.1 Creating an Event

Function Prototype	hi_u32 hi_event_create(HI_OUT hi_u32 *id)	EventGroupHandle_t xEventGroupCreate(void)
Description	Creates an event and obtains the event ID.	Creates an event and obtains the event handle.
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**EventGroupHandle_t**: An event is returned. **NULL**: An event group fails to be created.
Input Argument	**id**: event ID	**xQueue**: message queue handle

4.2.2 Sending an Event

Function Prototype	hi_u32 hi_event_send(hi_u32 id, hi_u32 event_bits);	EventBits_t xEventGroupSetBits(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet)
Description	Sends an event.	Sends an event. You can set the corresponding bit to **1**.
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**EventBits_t**: The value of the current event flag group is returned.
Input Argument	**id**: event ID	**xEventGroup**: event group handle
	event_bits: event bits. A maximum of 24 event flag bits can be set.	**uxBitsToSet**: 24 event flag bits that can be set. **EventBits_t** is a 32-bit variable. The lower 24 bits are used to set the event flag.

4.2.3 Waiting for an Event

Function Prototype	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 xEventGroupWaitBits(const EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToWaitFor, const BaseType_t xClearOnExit, const BaseType_t xWaitForAllBits, TickType_t xTicksToWait);
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**EventBits_t**: The value of the current event flag group is returned.
Input Argument	**id**: event ID	**xEventGroup**: event group handle
	mask: set of events to be waited for	**uxBitsToWaitFor**: specified bits among 24 event flag bits to be waited for
	flag: event waiting flag • **HI_EVENT_WAITMODE_AND**: Waits until all events are set to **1**. • **HI_EVENT_WAITMODE_OR**: Waits for any event to be set to **1**. • **HI_EVENT_WAITMODE_CLR**: Waits until the event flag bit is cleared. This parameter is used together with the preceding two options.	**xClearOnExit**: whether to clear the set event flag **xWaitForAllBits**: whether to wait for all flag bits to be set
	timeout: waiting time (unit: ms)	**xTicksToWait**: waiting time (unit: clock cycle). If this parameter is set to **portMAX_DELAY**, the system waits permanently.
	event_bits: event bits	None

4.2.4 Clearing a Specified Event

Function Prototype	hi_u32 hi_event_clear(hi_u32 id, hi_u32 event_bits)	EventBits_t xEventGroupClear-Bits(EventGroupHandle_t xEventGroup,const EventBits_t uxBitsToClear)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**EventBits_t**: The value of the current event flag group is returned.
Input Argument	**id**: event ID	**xEventGroup**: event group handle
	event_bits: specified event bits to be cleared	**uxBitsToClear**: specified event bits to be cleared

4.2.5 Deleting an Event

Function Prototype	hi_u32 hi_event_delete(hi_u32 id)	void vEventGroupDelete(EventGroupHandle_t xEventGroup)
Description	Deletes an event.	Deletes an event group.
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**id**: event ID	**xEventGroup**: event group handle

5 Semaphore Management

5.1 API Comparison

Function	OSA API	FreeRTOS API
Create a counting semaphore.	hi_sem_create	xSemaphoreCreateCounting
Create a binary semaphore.	hi_sem_bcreate	xSemaphoreCreateBinary
Create a mutex.	**hi_mux_create**: Creates a mutex.	**vSemaphoreCreateMutex**: Creates a mutex.
Delete a semaphore.	hi_sem_delete	vSemaphoreDelete
Delete a mutex.	hi_mux_delete	None
Wait for a semaphore.	hi_sem_wait	**xSemaphoreTake**: Obtains a semaphore. **xSemaphoreTakeRecursive**: Obtains a recursive mutex. **xSemaphoreTakeFromISR**: Obtains a semaphore (interrupt version).
Wait for a mutex.	hi_mux_pend	
Release the semaphore.	hi_sem_signal	**xSemaphoreGive**: Releases a semaphore. **xSemaphoreGiveRecursive**: Releases a recursive mutex. **xSemaphoreGiveFromISR**: Releases a semaphore (interrupt version).
Release a mutex.	hi_mux_post	

5.2 API Description

5.2.1 Creating a Semaphore

Function Prototype	hi_u32 hi_sem_create(hi_u32 *sem_id, hi_u16 init_value)	SemaphoreHandle_t xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount)
Description	Creates a semaphore.	Creates a counting semaphore.
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	Success: semaphore handle Failure: NULL
Input Argument	**id**: semaphore ID	None
	init_value: initial number of valid signals. The value range is [0, 0xFFFF].	**uxMaxCount**: maximum count
	None	**uxInitialCount**: initial value of a semaphore. When the semaphore is used for event counting, the value should be **0**. When the parameter is used for resource management, the value of this parameter must be the same as that of **uxMaxCount**.

5.2.2 Creating a Binary Semaphore

Function Prototype	hi_u32 hi_sem_bcreate(hi_u32 *sem_id, hi_u8 init_value);	SemaphoreHandle_t xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	Success: semaphore handle Failure: NULL
Input Argument	**id**: semaphore ID	None
	None	**uxMaxCount**: maximum count

	init_value: initial value of a semaphore. Generally, when the value is **HI_SEM_ONE**, the API is used for critical resource protection. When the value is **HI_SEM_ZERO**, the API is used for thread synchronization.	**uxInitialCount**: initial value of a semaphore. When the semaphore is used for event counting, the value should be **0**. When the parameter is used for resource management, the value of this parameter must be the same as that of **uxMaxCount**.

5.2.3 Creating a Mutex

Function Prototype	hi_u32 hi_mux_create (hi_u32* mux_id)	SemaphoreHandle_t xSemaphoreCreateMutex(void)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	Success: mutex handle Failure: NULL
Input Argument	**id**: mutex ID	None

5.2.4 Deleting a Mutex/Semaphore

Function Prototype	hi_u32 hi_mux_delete(hi_u32 mux_id) hi_u32 hi_sem_delete(hi_u32 sem_id)	void vSemaphoreDelete(SemaphoreHandle_t xSemaphore)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	None
Input Argument	**id**: mutex/semaphore ID	**xSemaphore**: mutex/semaphore handle

5.2.5 Obtaining a Mutex/Semaphore

Function Prototype	hi_u32 hi_mux_pend(hi_u32 mux_id, hi_u32 timeout_ms) hi_u32 hi_sem_wait(hi_u32 sem_id, hi_u32 timeout)	xSemaphoreTake(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait)

Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdTRUE**: A semaphore is obtained successfully. **pdFALSE**: A semaphore is not obtained after the time expires.
Input Argument	**mux_id/sem_id**: mutex/ semaphore ID	**xSemaphore**: semaphore handle
	timeout_ms/timeout: waiting time (unit: ms). When this parameter is set to **HI_SYS_WAIT_FOREVER**, the system waits permanently.	**xTicksToWait**: waiting time. If the value is **portTICK_PERIOD_MSs**, the unit of ms is converted to tick. If **INCLUDE_vTaskSuspend** is set to **1** and the value is set to **portMAX_DELAY**, the waiting time is limited.

5.2.6 Freeing a Mutex/Semaphore

Function Prototype	hi_u32 hi_mux_post(hi_u32 mux_id) hi_u32 hi_sem_signal(hi_u32 sem_id)	xSemaphoreGive(SemaphoreHandle_t xSemaphore)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdTRUE**: A semaphore is obtained successfully. **pdFALSE**: An error occurs when the semaphore is freed.
Input Argument	**mux_id/sem_id**: mutex/semaphore ID	**xSemaphore**: semaphore handle

6 Timer

6.1 API Comparison

Function	OSA API	FreeRTOS API
Create a timer.	hi_timer_create	xTimerCreate
Start the timer.	hi_timer_start	**xTimerStart**: Starts the timer.
		xTimerStartFromISR: Starts the timer (interrupt version).
Stop the timer.	hi_timer_stop	**xTimerStop**: Stops the timer.
		xTimerStopFromISR: Stops the timer (interrupt version).
Delete the timer.	hi_timer_delete	xTimerDelete

6.2 API Description

6.2.1 Creating a Timer

Function Prototype	hi_u32 hi_timer_create(hi_u32 *timer_handle)	TimerHandle_t xTimerCreate (const char * const pcTimerName, const TickType_t xTimerPeriod, const UBaseType_t uxAutoReload, void * const pvTimerID, TimerCallbackFunction_t pxCallbackFunction)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	Success: A timer handle is returned. Failure: NULL
Input Argument	**timer_handle**: A timer handle is returned.	None
	None	**pcTimerName**: timer name
	None	**xTimerPeriod**: timer period (must be greater than 0)
	None	**uxAutoReload**: timer trigger flag ● **pdTRUE**: The timer is triggered at an interval specified by **xTimerPeriod**. ● **pdFALSE**: The timer is triggered only once.
	None	**pvTimerID**: ID allocated to the timer that is being created
	None	**pxCallbackFunction**: timer callback function

6.2.2 Starting the Timer

Function Prototype	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 xTimerStart(TimerHandle_t xTimer, TickType_t xBlockTime)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdFAIL**: The timer fails to be started. **pdPASS**: The timer is started successfully.
Input Argument	**timer_handle**: timer handle	**xTimer**: timer handle
	type: timer type	None

	expire: timeout period of the timer (unit: ms). The default value is 10 ms.	**xBlockTime**: timeout period of the timer (unit: tick)
	timer_func: timer callback function	Corresponds to the callback function of **xTimerCreate**.
	data: argument passed to a callback function	None

6.2.3 Stopping the Timer

Function Prototype	hi_u32 hi_timer_stop(hi_u32 timer_handle)	BaseType_t xTimerStop(TimerHandle_t xTimer, TickType_t xBlockTime)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdFAIL**: The timer fails to be stopped. **pdPASS**: The timer is stopped successfully.
Input Argument	**timer_handle**: timer handle	**xTimer**: timer handle
	None	**xBlockTime**: time for waiting the timer to be stopped (unit: tick)

6.2.4 Deleting the Timer

Function Prototype	hi_u32 hi_timer_delete(hi_u32 timer_handle)	BaseType_t xTimerDelete(TimerHandle_t xTimer, TickType_t xBlockTime)
Return Value	**HI_ERR_SUCCESS**: success Other values: failure	**pdFAIL**: The timer fails to be deleted. **pdPASS**: The timer is deleted successfully.
Input Argument	**timer_handle**: timer handle	**xTimer**: timer handle
	None	**xBlockTime**: time for waiting the timer to be deleted (unit: tick)