Hi3861 V100 / Hi3861L V100 Lower Power

Development Guide

Issue 01

Date 2020-04-30

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 system low-power mode and application development guide of Hi3861 V100/Hi3861L V100, and provides solutions to common problems.

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. ● **1.1 Prerequisites** is added. ● In **Why cannot the system enter the sleep mode?** of **5 FAQs**, the description that the low-power mode is not configured for the protocol stack (lwIP) is added.
00B01	2020-04-03	This issue is the first draft release.

Contents

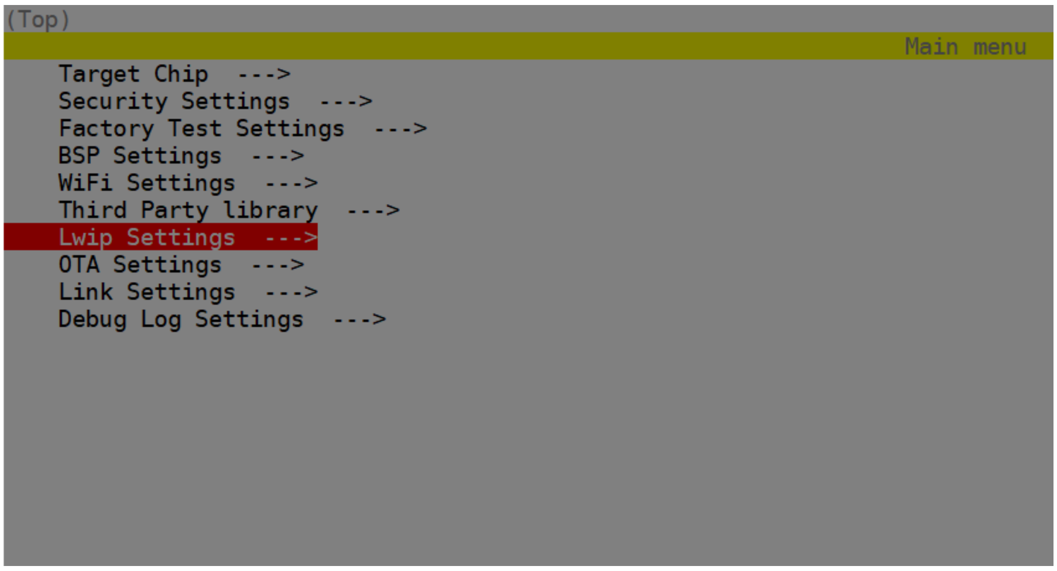
1 Introduction

1.1 Prerequisites

In the low-power scenario, you need to enable the low-power mode of the protocol stack (lwIP) in the menuconfig of the SDK. The configuration procedure is as follows:

Step 1 Run **./build.sh menuconfig** in the SDK code directory, select **Lwip Settings --->** (as shown in **Figure 1-1**), and press **Enter**.

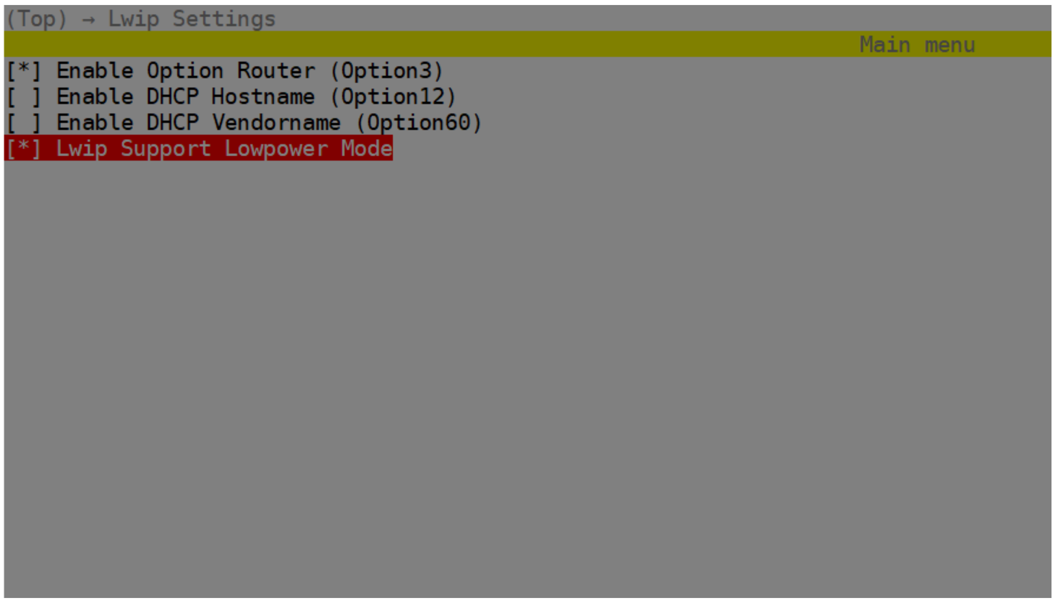
Figure 1-1 menuconfig GUI



```
(Top)
                                                              Main menu
    Target Chip  --->
    Security Settings  --->
    Factory Test Settings  --->
    BSP Settings  --->
    WiFi Settings  --->
    Third Party library  --->
    Lwip Settings  --->
    OTA Settings  --->
    Link Settings  --->
    Debug Log Settings  --->
```

Step 2 Select **Lwip Support Lowpower Mode**, and then press **Enter** (the symbol * is displayed in the square brackets ([]). The low-power mode of the protocol stack (lwIP) is selected, as shown in **Figure 1-2**.

Figure 1-2 Example for configuring the low-power mode of the protocol stack (lwIP)



```
(Top) → Lwip Settings
                                                                  Main menu
[*] Enable Option Router (Option3)
[ ] Enable DHCP Hostname (Option12)
[ ] Enable DHCP Vendorname (Option60)
[*] Lwip Support Lowpower Mode
```

Step 3 Save the low-power configuration of the protocol stack (lwIP) and recompile the code. For details, see the *Hi3861 V100/Hi3861L V100 SDK Development Environment Setup User Guide*.

----End

1.2 Low Power Mode

Hi3861 V100/Hi3861L V100 supports three system low-power modes, as shown in **Table 1-1**. You can select a low-power policy based on the actual application scenario. By default, no low-power mode is enabled.

Table 1-1 Low-power mode

Mode	Chip Status	Features
Ultra-deep sleep mode	• The CPU, RAM, and all peripherals are powered off. • Only GPIO3, GPIO5, GPIO7, and GPIO14 can be woken up by high level.	• The power saving mode specified in the Wi-Fi protocol is not supported. • The system restarts after wakeup (similar to power-on restart). • AP connection cannot be retained. • The GPIO output level cannot be retained. • This mode can be entered after configuration in deep sleep, light sleep, or non-low power mode.

Mode	Chip Status	Features
Deep sleep mode	● The CPU and peripherals except the GPIO and RTC are powered off. The RAM is not powered off. ● Only GPIO, SDIO, and Wi-Fi can be woken up.	● The power saving mode specified in the Wi-Fi protocol is supported. ● AP connection can be retained (after an STA is associated with an AP). ● The GPIO and SDIO services are normal, but other peripherals cannot work properly in sleep mode. ● The GPIO output level can be retained. ● This mode is mutually exclusive to the light sleep mode. The two modes cannot be set at the same time.
Light sleep mode	The CPU and RAM are not powered off, and all peripherals are powered off under control.	● The power saving mode specified in the Wi-Fi protocol is supported. ● AP connection can be retained (after an STA is associated with an AP). ● The Wi-Fi module circuit (including the RF) is disabled based on the low power consumption of the Wi-Fi subsystem. ● All peripherals can work properly. ● This mode is mutually exclusive to the deep sleep mode. The two modes cannot be set at the same time.

NOTE

For details about the power consumption specifications, see the *Hi3861 V100/Hi3861L V100 Demo Board Power Consumption Test Guide*.

2 Ultra-Deep Sleep Mode

2.1 Overview

In ultra-deep sleep mode, only the modules related to I/O wakeup are powered on. The I/O pin configuration is restored to the default value of the chip. Other modules are powered off. The memory information is not retained, and the Wi-Fi connection is torn down. In ultra-deep sleep mode, the user directly enters the ultra-deep sleep mode after the API is called. The API parameter can be used to specify the wakeup I/O. When the I/O level is high, the system is woken up from the ultra-deep sleep mode and enters the non-low-power mode after restart. If the I/O output level needs to be retained in ultra-deep sleep mode, a special hardware circuit design is required.

2.2 API Description

NOTE

For details about the APIs, see the *Hi3861 V100/Hi3861L V100 API Development Reference*.

Table 2-1 API description for ultra-deep sleep mode

API Name	Description
hi_lpc_enable_udsleep	Enables the ultra-deep sleep mode and sets the wakeup I/O.
hi_lpc_get_udsleep_wakeup_src	Obtains the wakeup I/O interface after wakeup (this API must be used after **hi_lpc_init**).

2.3 Application Example

The ultra-deep sleep mode is generally used in scenarios with extremely low power requirements, such as button wakeup and long-time standby wakeup of the master and slave devices.

NOTE

- If there is no high-level wakeup source, the device remains in the ultra-deep sleep state.
- During this period, the device memory is powered off. After the device is woken up from the ultra-deep sleep state, the wakeup source can be obtained through the interface. Other operations are the same as those for powering on the device again.
- It is recommended that the high level for pin wakeup be maintained at least 100 μs.

Code sample:

```
  /* Set GPIO5 and GPIO7 as the wakeup sources.*/
   hi_lpc_enable_udsleep(HI_UDS_GPIO5 | HI_UDS_GPIO7);
 /* System hibernation... */
   /* After wakeup, re-execute the initialization process and print the wakeup source. For
details, see hi_udsleep_src. */
   (hi_void)hi_lpc_init();
   ret = hi_lpc_get_udsleep_wakeup_src(&src);
   if (ret == HI_ERR_SUCCESS) {
       printf("udsleep wakeup src: %x\r\n", src);
   } else {
     /* Exception handling (omitted)*/
   }
```

3 Deep Sleep Mode

3.1 Overview

In low-power mode, Hi3861/Hi3861L uses idle tasks for system management. An idle task has the lowest priority and is executed only when the system is idle. If the system determines that there is no service to be executed, the system enters the deep sleep mode, that is, the CPU is powered off. In this mode, the system can be woken up by using deep sleep wakeup sources, including the GPIO, SDIO, and system tick (RTC). The tickless mechanism is used to prevent each system tick from being woken up.

After the deep sleep mode is set for a module that functions as a station (STA), the module automatically enters the sleep mode when the system is idle. The default sleep time depends on the DTIM and beacon interval of the associated access point (AP).

NOTE

Delivery Traffic Indication Message (DTIM) indicates the frequency at which an AP sends broadcast or multicast packets.

3.2 API Description

NOTE

- You can set the mode by calling **hi_lpc_set_type**, and veto the vote by calling **hi_lpc_add_veto** and **hi_lpc_remove_veto**. The one-vote veto mechanism is used. As long as one module is disabled from hibernation, the system cannot enter the low-power mode.
- For details about the APIs, see the *Hi3861 V100/Hi3861L V100 API Development Reference*.

Table 3-1 API description for deep sleep mode

API Name	Description
hi_lpc_get_type	Obtains the current low-power mode of the system.
hi_lpc_set_type	Sets the low-power mode of the system (excluding the ultra-deep sleep mode, which is configured by calling **hi_lpc_enable_udsleep**).
hi_lpc_add_veto	Votes for a mode. Once the vote is vetoed, the system enters the sleep state. You can add the **hi_lpc_id** enumeration to veto a vote. Each ID corresponds to only one vote.
hi_lpc_remove_veto	Votes for a mode. Once the veto is removed, the system enters the sleep state.
hi_lpc_register_wakeup_entry	Registers the entry into deep sleep wakeup. Only one group can be registered. This API is used as the entry function for reinitializing peripherals.
hi_lpc_register_check_handler	Registers sleep check. Multiple groups can be registered. The user registration API is called before the system enters the sleep state. The return value indicates whether the system is allowed to enter the sleep state. This API is used to check the status in real time, for example, check whether the UART is receiving or transmitting data. In other scenarios, you are advised to use **hi_lpc_add_veto** or **hi_lpc_remove_veto** with higher execution efficiency.
hi_lpc_register_hw_handler	Registers hardware processing. Only one group can be registered. The last registration prevails. The corresponding processing functions are called before sleep and after wakeup. For example, set the I/O to the high impedance state during deep sleep to further reduce the leakage current, and restore the configuration after wakeup.
hi_lpc_register_sw_handler	Registers software processing. Only one group can be registered. The last registration prevails. This API is executed when an idle task is entered or exited. It is used in the interrupt disable phase of an idle task and is generally used for debugging.
hi_lpc_config_dsleep_wakeup_io	Configures the GPIO corresponding to the wakeup in deep sleep mode. If I/O interrupts need to be handled, this API needs to be used in conjunction with the GPIO external interrupt API.
hi_wifi_set_pm_switch	Enables or disables the low-power mode of the Wi-Fi subsystem.

3.3 Application Example

Code sample:

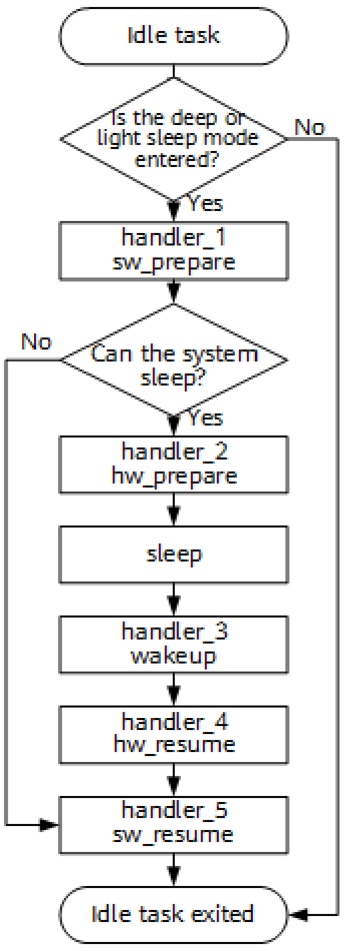
```
 /* Call this API before determining whether the system has entered the sleep state.*/
  hi_u32 sw_prepare(hi_void)
  {
      if (hi_lpc_get_type() == HI_DEEP_SLEEP) {
         /* Disable some timers that affect system hibernation as required. */
      }
      return HI_ERR_SUCCESS;
  }
 /* Call this API when an idle task exits.*/
  hi_u32 sw_resume(hi_void)
  {
      if (hi_lpc_get_type() == HI_DEEP_SLEEP) {
         /* Restore the timer or obtain maintenance and test information as required.*/
      }
      return HI_ERR_SUCCESS;
  }
/* Call this API before the system enters the sleep mode.*/
  hi_u32 hw_prepare(hi_void)
  {
      if (hi_lpc_get_type() == HI_DEEP_SLEEP) {
         /* Configure the I/O based on the actual I/O design to prevent leakage current in the
deep sleep phase. */
      }
      return HI_ERR_SUCCESS;
  }
 /* Call this API after system wakeup.*/
  hi_u32 hw_resume(hi_void)
  {
      if (hi_lpc_get_type() == HI_DEEP_SLEEP) {
      /* Restore the configuration based on the actual I/O design.*/
      }
      return HI_ERR_SUCCESS;
  }
  hi_u32 demo_init(hi_void)
  {
      hi_u32 ret;
      hi_pvoid handle;
     /* Register the entry function for deep sleep wakeup and initialize the powered-off
peripheral. For details, see the demo code delivered in the SDK. */
      ret = hi_lpc_register_wakeup_entry(wakeup);
      if (ret != HI_ERR_SUCCESS) {
         /* Exception handling (omitted)*/
      }
      /* Register the function for checking whether the system can enter the sleep mode. The
corresponding function is called before the system enters the sleep mode with idle tasks. */
      handle = hi_lpc_register_check_handler(check);
      if (handle == HI_NULL) {
         /* Exception handling (omitted)*/
      }
     /* Reduce the power consumption of leakage current in the deep sleep phase. */
      ret = hi_lpc_register_hw_handler(hw_prepare, hw_resume);
      if (ret != HI_ERR_SUCCESS) {
         /* Exception handling (omitted)*/
      }
```

```
    /* Implement special processing on the timer and collect maintenance and test information
in the deep sleep phase. Generally, registration is not required. */
     ret = hi_lpc_register_sw_handler(sw_prepare, sw_resume);
     if (ret != HI_ERR_SUCCESS) {
        /* Exception handling (omitted)*/
     }
   /* Enable the rising edge interrupt of GPIO7.*/
     ret = hi_gpio_init();
     if (ret != HI_ERR_SUCCESS) {
        /* Exception handling (omitted)*/
     }
     ret = hi_gpio_register_isr_func(HI_GPIO_IDX_7, HI_INT_TYPE_EDGE,
        HI_GPIO_EDGE_RISE_LEVEL_HIGH, demo_gpio7_wkup, HI_NULL);
     if (ret != HI_ERR_SUCCESS) {
        /* Exception handling (omitted)*/
     }
     /* Enable GPIO7 wakeup.*/
     ret = hi_lpc_config_dsleep_wakeup_io(HI_GPIO_IDX_7, HI_TRUE);
     if (ret != HI_ERR_SUCCESS) {
        /* Exception handling (omitted)*/
     }
    /*Associate an AP, obtain the IP address, and configure ARP_OFFLOAD. The code is
omitted.*/
     /* Set the system to deep sleep mode.*/
     ret = hi_lpc_set_type(HI_DEEP_SLEEP);
     if (ret != HI_ERR_SUCCESS) {
        /* Exception handling (omitted)*/
     }
     /* Enable the low-power mode of the Wi-Fi subsystem.*/
     ret = hi_wifi_set_pm_switch(HI_TRUE, 0);
     if (ret != HI_ERR_SUCCESS) {
        /* Exception handling (omitted)*/
     }
     return HI_ERR_SUCCESS;
  }
```

NOTE

The registration API can increase the flexibility for the user when the code source is closed. **Figure 3-1** shows the mapping between the registration API and an idle task.

Figure 3-1 Relationship between the registration API and an idle task



4 Light Sleep Mode

4.1 Overview

The implementation principle of the light sleep mode is similar to that of the deep sleep mode. The difference is that the CPU and peripherals are not powered off. In light sleep mode, the power consumption of the CPU itself is not reduced, but by reducing the power consumption of submodules. In the current solution, the low-power mode of the Wi-Fi subsystem is enabled to reduce the system power consumption (the light sleep mode does not need to be set). If in-depth customization is required (for example, enabling or disabling the peripheral clock), register the API after the light sleep mode is enabled, and enable or disable the module clock before or after the system enters the sleep mode.

4.2 API Description

NOTE

For details about the APIs, see the *Hi3861 V100/Hi3861L V100 API Development Reference*.

Table 4-1 API description for light sleep mode

API Name	Description
hi_wifi_set_pm_switch	Enables or disables the low-power mode of the Wi-Fi subsystem.

4.3 Application Example

Code sample:

```
  /* Enable the low-power mode of the Wi-Fi subsystem. The sleep time is configured on the AP side.*/
    hi_wifi_set_pm_switch(HI_TRUE, 0);
```

5 FAQs

Why cannot the system enter the sleep mode?

The common causes for the failure to enter the sleep mode:

- Timer usage error
  - Example: Start a 10 ms periodic timer.
  - Analysis: Before going to sleep, the system checks whether a timer is about to expire. If yes, the system is not allowed to enter the low-power mode.
  - Suggestion: Start the timer based on actual services and disable the timer (especially the periodic timer) when the services are idle.
- Task usage error
  - Example: Operate the task body cyclically and perform no proactive release action, for example, calling the blocking API or **hi_sleep**.
  - Analysis: If a task is not released, other tasks with lower priorities cannot be executed or even the watchdog may be reset. Before going to sleep, the system checks whether there are tasks to be scheduled. If there are too few tasks that are released, the system is not allowed to enter the low-power mode.
  - Suggestion: In service design, try to call the blocking API and set the timeout interval to infinite or a proper value.
- hi_sleep usage error
  - Example: Call **hi_sleep** and set the input argument to 10 ms.
  - Analysis: This behavior is similar to starting a 10 ms periodic timer. The task is scheduled at a 10 ms period. Before going to sleep, the system checks whether a task is about to expire and forbid the system to enter the low-power mode.
  - Suggestion: Use the blocking API and set the blocking time to infinity or a proper timeout period. Alternatively, use a timer for implementation, and disable the timer when services are idle.
- hi_lpc_add_veto/hi_lpc_remove_veto usage error
  - Example: **hi_lpc_add_veto** and **hi_lpc_remove_veto** are not used in pairs.

  - Analysis: Before going to sleep, the system checks whether a module has vetoed sleep. If yes, it will enter sleep mode with one vote. Otherwise, the system will never enter sleep mode.
  - Suggestion: Strictly check whether the two APIs are used in pairs and ensure that the duration for vetoing sleep is as short as possible to ensure that the system enters the sleep mode as soon as possible.
- The low-power mode of the Wi-Fi subsystem is disabled.
  - Example: Enable the low-power mode of the Wi-Fi subsystem without calling **hi_wifi_set_pm_switch**.
  - Analysis: Enabling the low-power mode of the Wi-Fi subsystem is the prerequisite for the system to enter the sleep mode. Otherwise, the system does not enter the sleep mode even if the low-power mode is set.
  - Suggestion: To use the low-power policy, the low-power mode of the Wi-Fi subsystem must be enabled.
- The low-power mode is not configured for the protocol stack (lwIP).
  - Example: To use the low-power policy, you must configure the low-power mode of the protocol stack (lwIP) subsystem in menuconfig of the SDK version. For details, see the *Hi3861 V100/Hi3861L V100 SDK Development Environment Setup User Guide*.
  - Impact: The low-power mode of the protocol stack may affect the performance of Wi-Fi services. By default, the low-power mode of the protocol stack is disabled.

Why does the system run abnormally after waking up from the deep sleep mode?

- Example: After the call to I^{2}C interface is added, the system runs properly after being powered on. After the system wakes up from the deep sleep mode, the I^{2}C interface works abnormally.
- Analysis: The peripheral modules are powered off in deep sleep mode and need to be reinitialized after wakeup.
- Suggestion: Add the I^{2}C initialization to **hi_lpc_register_wakeup_entry**. Note the initialization position. The corresponding initialization function must be called after the UART and flash are initialized.

Why does the peripheral service occasionally fail to receive or send packets?

- Example: During SPI device communication, data cannot be received occasionally or the transmitted data is not as expected.
- Analysis: If all peripherals are powered off after deep sleep, data cannot be received from the peer end. Alternatively, after the asynchronous TX interface is called, data is not sent out, so the system considers that no service is about to be executed or is being executed. As a result, the system enters the sleep mode and data transmission is abnormal.
- Suggestion: Add **hi_lpc_add_veto** or **hi_lpc_remove_veto** to the corresponding service to notify the system of whether the system can enter the sleep mode.

Why is the system woken up immediately after it enters the ultra-deep sleep mode?

Suggestion: Check whether the level of the pin corresponding to the wakeup source is stable low.

Why is the power consumption higher than expected?

Suggestion: Check the hardware and software.

- Hardware: Check whether the bottom current meets the expectation.
- Software: Perform the following operations to locate the fault:
  a. Check the value of **type** by calling **hi_lpc_get_info**. Ensure that the mode is correct. Check the value of **sleep_times** to determine whether the system has entered the sleep mode. Wait for a period of time and check whether the value increases continuously, that is, whether the system is continuously switched to the sleep or wakeup state.
  b. View the statistics **veto_info** by calling **hi_lpc_get_info** and check whether any module rejects sleep due to voting.
  c. Check the value of **sleep_threshold_refuse_times** by calling **hi_lpc_get_info**. If the value increases continuously, check the values of **task_xxx** and **timer_xxx** corresponding to the structure to determine whether the task or timer is about to expire.
  d. Obtain the system clock (based on the RTC clock) by calling **hi_systick_get_cur_tick** in the registration callback functions **hw_prepare** and **hw_resume** to calculate the sleep time. Check whether the sleep time is sufficient each time and calculate the wakeup ratio. However, obtaining the **hi_systick_get_cur_tick** time consumes hundreds of microseconds. Therefore, this step is applicable only to the debugging and fault locating phase.