SCM1612
Wi-Fi 6 and BLE 5 Low-Power SoC

Low Power Development Guide

Revision 1.0
Date 2023-08-15

Contact Information
Senscomm Semiconductor (www.senscomm.com)
Room 303, International Building, West 2 Suzhou Avenue,
SIP, Suzhou, China
For sales or technical support, please send email to
info@senscomm.com

Disclaimer and Notice

This document is provided on an “as-is” basis only. Senscomm reserves the right to make corrections, improvements and other changes to it or any specification contained herein without further notice.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

All third party’s information in this document is provided as is with NO warranties to its authenticity and accuracy.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners and are hereby acknowledged.

Version History

Version	Date	Description
1.0	2023-08-15	Format Modification

Table of Contents

1 Introduction

This document describes how to make use of the PM APIs to implement low power applications.

1.1 Low Power State

The device's internal Power Management Unit (PMU) offers various levels of power-saving modes. However, not all the PMU modes are applicable for the real use cases. Instead, users are advised to utilize the PM APIs available in the SDK.

The SDK supports the following PM states:

- Active
- Light Sleep
- Deep Sleep
- Hibernation

In the Active state, every component of the SoC remains powered and operational. Conversely, the Hibernation state is the most power-efficient, where most components are powered down, excluding those in Always On (AON) power domain.

1.2 State Transitions

With PM enabled in the SDK, the device automatically transitions between PM states to optimize power consumption.

Each PM state necessitates specific save and/or restore operations to ensure consistent contexts before and after entering a low power state. More power-efficient states involve more extensive operations and consequently, more time.

The device's next PM state is determined by its idle time, which is the period until the next scheduled activity. The timing conditions for each PM state are outlined below, but they can be adjusted based on application needs:

PM State Selected	Expected Idle Time
Active	-
Light Sleep	> 550 usec
Deep Sleep	> 10.2 msec
Hibernation	> 1.7 sec

When the device transitions to a low power state (other than Active), it will always revert to the Active state upon a wakeup event. Direct transitions between two different low power states are not supported.

Transition Diagram:

[Light Sleep] --- [Active] --- [Deep Sleep]
|
[Hibernation]

1.3 Conditions for PM

Typically, transitions to low power states are contingent upon:

- The OS having sufficient idle time, indicating all its threads are idle and awaiting events.
- Peripheral drivers not disabling PM.
- The user application enabling PM and specific PM states

1.4 Network Connectivity

Network connectivity is preserved across all low power states.

In both the Deep Sleep and Hibernation states, a specialized software operates to monitor WiFi activity and causes wakeup event if necessary. This software operates from the RAM, eliminating the need for flash and other power-intensive component. This approach ensures WiFi connectivity while minimizing the device's overall power consumption.

2 Wakeup Source

The following peripherals can be used as wakeup sources when the device is in a low power state.

- RTC
- GPIO
- UART
- SDIO
- USB

The wakeup events are not always available, and they depend on the specific PM states. RTC and GPIO can be used in all low power states to wake up the device.

For details on using UART, SDIO, USB wakeup, please consult Senscomm.

2.1 RTC

The RTC (Real-Time Clock) can be set to prompt the device wakeup at a designated future time. This configuration is typically managed by the operating system, ensuring that the OS thread operates at the correct intervals.

Users don't need in-depth knowledge of PM operations to leverage this. They can utilize standard OS APIs, such as delay functions or inter-process communication functions. Behind the scenes, the OS gathers thread-related data, adjusts the RTC to the nearest required wakeup time, and transitions to low power states when all threads are idle.

It's advisable to refrain from directly accessing RTC APIs when PM is active.

2.2 GPIO

GPIO (General-Purpose Input/Output) can trigger a wakeup event across all PM states.

To designate a GPIO as a wakeup source, its pin should be set as an input, and the associated interrupt should be activated. It's also beneficial to configure the

pull-up or pull-down settings. Subsequently, the application must inform the PM to recognize it as a wakeup source.

It should be highlighted that only the GPIOs in the AON (Always On) domain have the capability for wakeup functions. In particular, these GPIOs span from GPIO0 to GPIO7.

3 PM states

3.1 Active State

In this state, the SoC (System on Chip) is fully operational. All peripherals are powered up and ready for use.

3.2 Light Sleep

During the light sleep state:

- The core and peripherals are clock-gated.
- All register contents and memories are retained.
- Timer counts remain unchanged.

Software responsibilities include:

- Saving the system time upon entering this state.
- Restoring the system time upon exit, taking into account the time spent in this state.

3.3 Deep Sleep

In the deep sleep state:

- Both the core and peripherals are power-gated.
- The XTAL (Crystal Oscillator) and the PLL (Phase-Locked Loop) are turned off.
- The core and peripheral registers are not retained.

Software responsibilities encompass:

- Saving the system time and suspending peripheral devices when entering this state.
- Restoring the system time and resuming peripheral devices upon exit.

3.4 Hibernation

During hibernation:

- The core, peripherals, XTAL, and PLL are all power-gated.
- Additional core LDOs (Low Drop-Out regulators) and DCDCs (DC-to-DC converters) are deactivated.
- All memory sections, except those in the AON (Always On) SRAM, are powered down.

Software tasks involve:

- Saving the system time, suspending peripheral devices, and storing memory contents to flash upon entering hibernation.
- When leaving this low power state, restoring the system time, resuming peripheral devices, and retrieving memory contents from flash upon waking up.

A limitation exists in the hibernation state. Given that certain contexts are stored in the flash memory, the flash memory's erase and write cycles are finite. Considering the characteristics of the SoC's integrated flash memory, it's recommended to restrict hibernation occurrences to a maximum of 27 times daily.

4 APIs

4.1 Prerequisite

To enable the power management, user should configure the PM options from the menuconfig.

```
[*] Enable SCM2010 PM
[*]   Enable Hibernation mode
(0x80018000) Hibernation partition address
(86400) Hibernation limit count clear duration (sec)
(27)    Hibernation limit count
```

To use the PM APIs, user should also enable the PM API options from the menuconfig.

```
--- SDK
[*]   Include Wi-FI APIs
        Wi-Fi API  --->
[ ]   Include Wi-FI CLIs for API testing
[*]   Include PM APIs
[ ]     Include PM CLIs for API testing (NEW)
```

The following API can be used to customize the behavior of the PM.

- scm_pm_power_down
- scm_pm_enable_wakeup_io
- scm_pm_disable_wakeup_io
- scm_pm_enable_lowpower
- scm_pm_disable_lowpower
- scm_pm_disable_lowpower_timeout
- scm_pm_enable_state
- scm_pm_disable_state
- scm_pm_register_handler
- scm_pm_unregister_handler

4.2 Power Down (with timeout)

Functions

- int scm_pm_power_down(uint32_t duration)

The `scm_pm_power_down` can be used to make the device enter hibernation state unconditionally. All network connections would be lost.

If the `duration` is set to a non-zero value, the device will awaken after the designated time. A zero duration means only a GPIO event can wake the device. Upon waking, the device initializes as if undergoing a power-up reset.

4.3 GPIO Wakeup Control

Functions:

- int scm_pm_enable_wakeup_io(uint8_t pin)
- int scm_pm_disable_wakeup_io(uint8_t pin)

There are total 8 GPIOs (ranging from GPIO0 to GPIO7) in the AON domain. Each GPIO can be designated as a wakeup pin. However, without invoking `scm_pm_enable_wakeup`, a GPIO, even if set as an input, won't initiate a wakeup event.

4.4 Low Power Control

Functions:

- int scm_pm_enable_lowpower(void)
- int scm_pm_disable_lowpower(void)

These functions grant applications control over PM operations. If an application needs to temporarily disable PM for critical tasks, it should call scm_pm_disable_lowpower. Once the task concludes and PM needs reactivation, scm_pm_enable_lowpower should be called.

4.5 Low Power State Control

Functions:

- int scm_pm_enable_state(enum scm_pm_state state)
- int scm_pm_disable_state(enum scm_pm_state state)

As there are different PM states supported, the applications can enable or disable each specific PM states. When a specific state is disabled, the device will transition to the next available PM state during idle periods.

If all the states are turned off, then the device will not enter any low power states.

4.6 Notifications on PM State Changes

Prototypes:

- typedef void (*scm_pm_notify)(enum scm_pm_state state);

Functions

- int scm_pm_register_handler(scm_pm_notify callback)
- int scm_pm_unregister_handler(scm_pm_notify callback)

Applications can get notified of the PM state changes. It should call `scm_pm_register_handler` to register application specific function to be invoked.

Multiple callbacks can be registered by calling `scm_pm_register_handler` multiple times with different callbacks.

When entering the low power state, the callback will be invoked with the state parameter being one of SCM_PM_STATE_LIGHT_SLEEP, SCM_PM_STATE_DEEP_SLEEP, or SCM_PM_STATE_HIBERNATION.

When waking up from the low power state, the callback will be invoked with the state parameter always being SCM_PM_STATE_ACTIVE.

These callbacks, for example, can be used to perform application specific cleanups before entering the low power state such as turning off LEDs, disable sensors, etc.

Note:
The callbacks are invoked from the OS's idle task, and therefore the application code is limited to the idle task stack size. Also, all the interrupts are disabled. The callback should not call any OS APIs or interrupt related functions. It is recommended that the function is finished as soon as possible.