SCM1612
Wi-Fi 6 和 BLE 5 低功耗 SoC

低功耗开发指南

文档版本 1.0
发布日期 2023-08-15

联系方式
速通半导体科技有限公司 (www.senscomm.com)
江苏省苏州市工业园区苏州大道西 2 号国际大厦 303 室
销售或技术支持，请发送电子邮件至
support@senscomm.com

免责声明和注意事项

本文档仅按"现状"提供。速通半导体有限公司保留在无需另行通知的情况下对其或本文档中包含的任何规格进行更正、改进和其他变更的权利。

与使用本文档中的信息有关的一切责任，包括侵犯任何专有权利的责任，均不予承认。此处不授予任何明示或暗示、通过禁止或其他方式对任何知识产权的许可。

本文档中的所有第三方信息均按"现状"提供，不对其真实性和准确性提供任何保证。

本文档中提及的所有商标、商号和注册商标均为其各自所有者的财产，特此确认。

版本历史

版本	日期	描述
1.0	2023-08-15	1.0 发布

目录

1 简介

本文档描述了如何使用电源管理 API 来实现低功耗应用。

1.1 低功耗状态

设备的内部电源管理单元（PMU）提供了多种节电模式。但并不是所有的 PMU 模式都适用于实际使用场景。相反，建议用户使用 SDK 中提供的 PM API。

SDK 支持以下 PM 状态：

- Active 活跃状态
- Light Sleep 浅睡模式
- Deep Sleep 深睡模式
- Hibernation 超深睡模式

在活跃状态下，SoC 的每个组件都保持供电和运行。相反，超深睡模式是最节能的，其中大部分组件都关闭电源，除了始终开启（AON）电源域中的那些组件。

1.2 状态转换

在 SDK 中启用电源管理后，设备会自动在不同功耗模式之间转换以优化功耗。

每个功耗模式都需要特定的保存和/或恢复操作，以确保在进入低功耗状态之前和之后的上下文保持一致。更节能的状态涉及更广泛的操作，因此需要更多的时间。

设备的下一个功耗模式由其空闲周期决定，即直到下一个预定任务的时间间隔。每个 PM 状态的时间条件如下所示，但可以根据应用需求进行调整。

功耗模式	预期空闲周期
活跃模式	-
浅睡模式	> 550 微妙
深睡模式	> 10.2 毫秒

超深睡模式	> 1.7 秒

当设备切换到功耗模式（除活跃模式外）时，它在唤醒事件发生后将始终返回到活跃模式。不支持两个不同功耗模式之间的直接转换。

转换图示:

[浅睡模式] --- [活跃模式] --- [深睡模式]
|
[超深睡模式]

1.3 电源管理条件

通常，转换到低功耗状态取决于以下因素：

- 操作系统有足够的空闲时间，这意味着所有的线程都处于空闲状态并等待事件。
- 外设驱动程序没有禁用电源管理。
- 用户应用程序启用了电源管理和特定的电源管理状态。

1.4 网络保活

在所有低功耗状态下都保持网络连接。

在深睡模式和超深睡模式中，专门的软件操作用于监视 WiFi 活动，并在必要时引起唤醒事件。此软件从 RAM 运行，无需使用闪存和其他高功耗组件。这种方法确保了 WiFi 连接，同时最大限度地减少了设备的总体功耗。

2 唤醒源

当设备处于低功耗状态时，以下外设可以用作唤醒源：

- RTC
- GPIO
- UART
- SDIO
- USB

唤醒事件并不总是可用的，它们取决于特定的电源管理状态。RTC 和 GPIO 可以在所有低功耗状态中用来唤醒设备。

有关使用 UART、SDIO、USB 唤醒的详细信息，请咨询 Senscomm。

2.1 RTC

RTC（实时时钟）可以设置为在指定的未来时间提示设备唤醒。这种配置通常由操作系统管理，确保操作系统线程在正确的间隔内操作。

用户不需要深入了解电源管理操作来利用这一点。他们可以使用标准的操作系统 API，如延迟函数或进程间通信函数。在后台，操作系统收集线程相关数据，将 RTC 调整到最近的所需唤醒时间，并在所有线程都处于空闲状态时转换到低功耗状态。

建议在电源管理处于活动状态时避免直接访问 RTC API。

2.2 GPIO

GPIO（通用输入/输出）可以在所有电源管理状态下触发唤醒事件。

要指定 GPIO 作为唤醒源，其引脚应设置为输入，并应激活相关的中断。配置上拉或下拉设置也是有益的。随后，应用程序必须通知电源管理模块将其识别为唤醒源。

应该强调的是，只有 AON（始终开启）域中的 GPIO 具有唤醒功能的能力。特别指出的是，这些 GPIO 从 GPIO0 到 GPIO7。

3 电源管理状态

3.1 活跃状态

在此状态下，SoC（系统芯片）完全运行。所有外设都已上电并准备使用。

3.2 潜睡模式

在浅睡模式状态下：

- 处理器和外设都是时钟门控的。
- 所有寄存器内容和内存都被保留。
- 定时器计数保持不变。

软件的职责包括：

- 进入此状态时保存系统时间。
- 退出时恢复系统时间，考虑到在此状态中所花费的时间。

3.3 深睡模式

在深睡模式状态下：

- 处理器和外设都是电源门控的。
- XTAL（晶体振荡器）和 PLL（锁相环）都被关闭。
- 处理器和外设的寄存器不被保留。

软件的职责包括：

- 进入此状态时保存系统时间并挂起外设设备。
- 退出时恢复系统时间并恢复外设设备。

3.4 超深睡模式

在超深睡模式期间：

- 处理器、外设、XTAL 和 PLL 都是电源门控的。
- 额外的处理器 LDO（低压差稳压器）和 DCDC（直流-直流转换器）被停用。
- 所有内存部分，除了 AON（始终开启）SRAM 中的部分，都被断电。

软件任务包括：

- 进入超深睡模式时保存系统时间、挂起外设设备并将内存内容存储到闪存中。
- 离开此低功耗状态时，唤醒时恢复系统时间、恢复外设设备并从闪存中检索内存内容。

深睡模式存在一个限制。考虑到某些上下文存储在闪存中，闪存的擦除和写入周期是有限的。考虑到 SoC 集成闪存的特性，建议将超深睡模式的发生次数限制为每天最多 27 次。

4 API 接口

4.1 前提条件

为了启用电源管理，用户应从 menuconfig 中配置 PM 选项。

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

要使用 PM API 接口，用户还应从 menuconfig 中启用 PM API 选项。

```
--- 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)
```

以下 API 可以用于客制化电源管理模块的行为：

- 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 关机(带持续时间)

函数：

- int scm_pm_power_down(uint32_t duration)

The `scm_pm_power_down` 可以用于使设备无条件地进入超深睡模式。所有网络连接都会丢失。

如果 duration 设置为非零值，设备将在指定时间后唤醒。零持续时间意味着只有 GPIO 事件可以唤醒设备。唤醒后，设备初始化，就像经历电源上电复位。

4.3 GPIO 唤醒控制

函数：

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

在 AON 域中总共有 8 个 GPIO（从 GPIO0 到 GPIO7）。每个 GPIO 都可以被指定为唤醒引脚。但是，如果不调用 scm_pm_enable_wakeup，即使将 GPIO 设置为输入，也不会启动唤醒事件。

4.4 低功耗状态控制

函数：

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

这些函数允许应用程序控制 PM 操作。如果应用程序需要为关键任务临时禁用 PM，它应该调用 scm_pm_disable_lowpower。任务结束并需要重新激活 PM 时，应调用 scm_pm_enable_lowpower。

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)

由于支持不同的 PM 状态，应用程序可以启用或禁用每个特定的 PM 状态。当禁用特定状态时，设备在空闲期间将转换到下一个可用的 PM 状态。

如果所有状态都被关闭，那么设备将不会进入任何低功耗状态。

4.6 电源管理状态变化通知

原型：

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

函数：

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

应用程序可以得到 PM 状态变化的通知。它应该调用 scm_pm_register_handler 来注册要调用的特定于应用程序的函数。

可以通过多次使用不同的回调调用 scm_pm_register_handler 来注册多个回调。

当进入低功耗状态时，回调将被调用，状态参数是：SCM_PM_STATE_LIGHT_SLEEP 、SCM_PM_STATE_DEEP_SLEEP 或 SCM_PM_STATE_HIBERNATION 之一。

从低功耗状态唤醒时，回调将被调用，状态参数始终为 SCM_PM_STATE_ACTIVE。

例如，这些回调函数可以用于在进入低功耗状态之前执行特定于应用程序的清理操作，如关闭 LED、禁用传感器等。

注意：

这些回调函数是从操作系统的空闲任务中调用的，因此应用程序代码的大小受限于空闲任务的堆栈大小。此外，所有的中断都被禁用了。回调函数不应调用任何操作系统 API 或与中断相关的函数。建议该函数尽快完成。