XR806 PM 应用

开发指南

版本号：1.0

发布时间：2021-02-23

版本历史

版本	日期	责任人	版本描述
1.0	2021-02-23	AWA 0498	创建文档。

目录

版本历史 i
目录 ii
表格目录 iv
1 前言 1
1.1 文档简介 1
1.2 目标读者 1
1.3 适用范围 1
1.4 文档约定 1
1.4.1 标志说明 1
1.4.2 地址与数据描述方法约定 1
1.4.3 数值单位约定 2
2 概述 3
2.1 背景说明 3
2.2 规格特性 3
2.3 文件位置 4
3 技术说明 6
3.1 休眠唤醒流程 6
3.2 休眠前处理 7
3.3 配置唤醒源 7
3.3.1 Wakeup IO Debounce 配置 7
3.4 新加外设或模块的休眠唤醒处理 8
3.5 Wakelock 介绍 9
3.6 系统启动状态 9
4 应用说明 11
4.1 应用简述 11
4.2 配置说明 11
4.3 接口说明 12
4.3.1 PM 接口 12
4.3.1.1 pm_register_ops 12
4.3.1.2 struct soc_device_driver 12
4.3.1.3 struct soc_device 13
4.3.1.4 pm_unregister_ops 13

4.3.1.5 pm_enter_mode 13
4.3.1.6 pm_enter_mode_timeout 14
4.3.1.7 pm_suspend_abort 14
4.3.2 Wakeup 接口 14
4.3.2.1 HAL_Wakeup_GetEvent 14
4.3.2.2 HAL_Wakeup_SetIO 15
4.3.2.3 HAL_Wakeup_ClrIO 15
4.3.2.4 HAL_PRCM_SetWakeupDebClk0 15
4.3.2.5 HAL_PRCM_SetWakeupDebClk1 16
4.3.2.6 HAL_PRCM_SetWakeupIOxDebSrc 16
4.3.2.7 HAL_PRCM_SetWakeupIOxDebounce 16
4.3.2.8 HAL_Wakeup_SetTimer_mS 17
4.3.2.9 HAL_Wakeup_SetTimer_Sec 17
4.3.3 Wakelock 接口 17
4.3.3.1 pm_wake_lock 17
4.3.3.2 pm_wake_unlock 18
4.3.3.3 pm_wake_lock_timeout 18
4.3.3.4 pm_wakelocks_show 18
5 示例说明 19
5.1 示例简介 19
5.1.1 获取方式 19
5.1.2 准备工作 19
5.1.3 操作步骤 19
5.2 关键实现 19
5.3 效果展示 20
5.4 新加设备的休眠唤醒处理 21
6 常见问题 22
6.1 进入低功耗模式后功耗偏高 22
6.2 系统进入不了低功耗模式 22
6.3 系统进入休眠无反应也无法唤醒 22
6.4 系统不能唤醒 22
附录 A： 唤醒源定义 23
附录 B： Wakeup IO 定义 24

表格目录

表 2-1 PM 模式 3
表 2-2 唤醒源列表 4
表 3-1 系统启动状态定义 9
表 4-1 配置项说明 11
表 4-2 pm_register_ops 接口函数说明 12
表 4-3 struct soc_device_driver 结构体的成员说明 12
表 4-4 struct soc_device 结构体的成员说明 13
表 4-5 pm_unregister_ops 接口函数说明 13
表 4-6 pm_enter_mode 接口函数说明 13
表 4-7 pm_enter_mode_timeout 接口函数说明 14
表 4-8 pm_suspend_abort 接口函数说明 14
表 4-9 HAL_Wakeup_GetEvent 接口函数说明 14
表 4-10 HAL_Wakeup_SetIO 接口函数说明 15
表 4-11 HAL_Wakeup_ClrIO 接口函数说明 15
表 4-12 HAL_PRCM_SetWakeupDebClk0 接口函数说明 15
表 4-13 HAL_PRCM_SetWakeupDebClk1 接口函数说明 16
表 4-14 HAL_PRCM_SetWakeupIOxDebSrc 接口函数说明 16
表 4-15 HAL_PRCM_SetWakeupIOxDebounce 接口函数说明 16
表 4-16 HAL_Wakeup_SetTimer_mS 接口函数说明 17
表 4-17 HAL_Wakeup_SetTimer_Sec 接口函数说明 17
表 4-18 pm_wake_lock 接口函数说明 17
表 4-19 pm_wake_unlock 接口函数说明 18
表 4-20 pm_wake_lock_timeout 接口函数说明 18
表 4-21 pm_wakelocks_show 接口函数说明 18

1 前言

1.1 文档简介

XR806 PM (Power Manager)即功耗管理子系统，主要用于降低系统功耗，管理系统运行和休眠状态下的设备功耗和系统功耗。

1.2 目标读者

本文档适合于 XR806 PM 模块开发及维护的人员。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示(寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

2 概述

2.1 背景说明

很多 IoT 设备采用电池供电，不会经常更换电池或充电，因此节省功耗对系统而言非常重要。XR806 SDK 采用了高效、灵活的功耗管理技术，可以在功耗控制、唤醒延迟之间实现最佳平衡。

2.2 规格特性

XR806 SDK 支持 Sleep，Standby，Hibernation 三种休眠模式，可以满足用户不同的场景需求。以上三种休眠模式的功耗依次递减，唤醒所需时间依次递增。另外，系统运行期间采用了 WFI、Tickless 等机制以降低非休眠状态下的功耗。

表 2-1 PM 模式

阶段	支持模式	状态
系统运行	WFI、Tickless	默认都开启，不需要做任何配置。
系统休眠	Sleep、Standby	可以配置是否使用，默认全模式支持。
	Hibernation	默认支持，功耗最低。

各模式说明如下：

- WFI(wait for interrupt)是 ARM CPU 没任务时进入的 low-power standby 模式的指令，由 ARM core 实现，处在 WFI 状态的 CPU 功耗比全速运行时有明显降低。
- CPU 在低负载时，大部分的周期性时钟中断并不会触发进程调度和定时事件，Tickless 可以让系统根据一下次需要执行调度的时刻或未来最早的定时器到期前的时刻重新调整 tick 中断，从而减少不必要的 tick 中断，让 CPU 有更多时间处在 WFI 状态。
- Sleep 模式下，全部外设都可以处在工作状态。例如需要 UART 唤醒系统，在进入 Sleep 模式时，UART 保持正常工作状态就可以在接收到字符时唤醒系统。该模式的功耗降低最小，除非所使用的唤醒源只能在 Sleep 模式下工作，否则尽量使用 Standby 模式。
- Standby 模式下，大部分外设停止工作，不能唤醒系统，只保留少部分可唤醒外设以及网络，可唤醒外设见表 2-2。该模式下功耗较低，且唤醒较快。
- Hibernation 模式下，系统只保留 RTC 域电源，只有特定的 Wakeup IO 和 Wakeup Timer 能唤醒系统，系统唤醒后执行系统复位流程，所有软硬件会被重新初始化。

XR806 SDK 支持多种唤醒源，如按键唤醒、网络 WLAN 唤醒、普通设备中断唤醒、Wakeup Timer 唤醒等，同一种休眠模式可以同时支持多种唤醒方式。

表 2-2 唤醒源列表

模式/场景	唤醒源					响应中断速度/唤醒所需时间
	所有可产生中断的外设	GPIO / GPADC / LPUART	Wakeup IO & Timer / RTC alarm	WLAN	BLE	
WFI	√	√	√	√	√	极快(us 级别)
Sleep	√	√	√	√	√	较快
Standby	×	√	√	√	√	较快
Hibernation	×	×	√	×	×	最慢

各唤醒源说明如下：

- 全部设备中断唤醒：该唤醒方式只在 Sleep 模式下支持，例如 UART 中断。
- 部分设备中断唤醒：GPIO/GPADC/LPUART 设备可在 standby 模式下正常工作，可唤醒系统，注意 GPADC 和 LPUART 会使用 32K 时钟，LPUART 波特率不能高于 9600，在 9600 波特率时不能连续发送多个字节。
- WLAN 唤醒：WLAN 系统接收到特定网络数据包（一般为发送给本机的网络数据包）后触发系统唤醒，例如 ping 包。
- BLE 唤醒：BLE 系统在需要接收或发送数据包时会提前唤醒系统，唤醒处理完相关事件后可再次进入休眠。
- Wakeup IO 唤醒：最多同时支持 14 个 GPIO 唤醒源。Wakeup IO 是具有 Wakeup 功能的 GPIO，编号与 GPIO 不相同（详见附录 B）。Wakeup IO 一般在 Hibernation 模式使用，只支持上升沿和下降沿触发唤醒。
- Wakeup Timer 唤醒：支持最短 10 毫秒，最长约 18 小时的定时时间。定时结束后即产生唤醒事件。Wakeup Timer 使用 32K 时钟进行计时，如果使用内部 32K，受温度等因素影响，会产生一定的计时误差。

2.3 文件位置

以 SDK 包为根目录，本技术方案涉及到的主要文件位置如下。

```
.
├── include
│   └── pm
│       ├── pm.h
│   └── driver
│       ├── chip
│       │   ├── hal_wakeup.h
└── project
    ├── example
    │   ├── pm
```

```
│   │   ├── command.c
│   │   ├── command.h
│   │   ├── gcc
│   │   │   ├── defconfig
│   │   │   └── Makefile
│   │   ├── main.c
│   │   ├── prj_config.h
│   │   └── readme.md
└── src
    ├── pm
    │   ├── Kconfig
    │   ├── pm.c
    │   └── wakelock.c
    ├── rom
    │   ├── rom_bin
    │   │   └── src
    │   │       ├── driver
    │   │       │   └── chip
    │   │       │       └── hal_wakeup.c
    │   │       ├── pm
    │   │       │   ├── cpu.s
    │   │       │   ├── pm.c
    │   │       │   └── wakelock.c
```

注意

rom 中的代码不可更改，若要更改需要去掉 rom 中的导出符号并在 SDK 中重新实现相关代码。

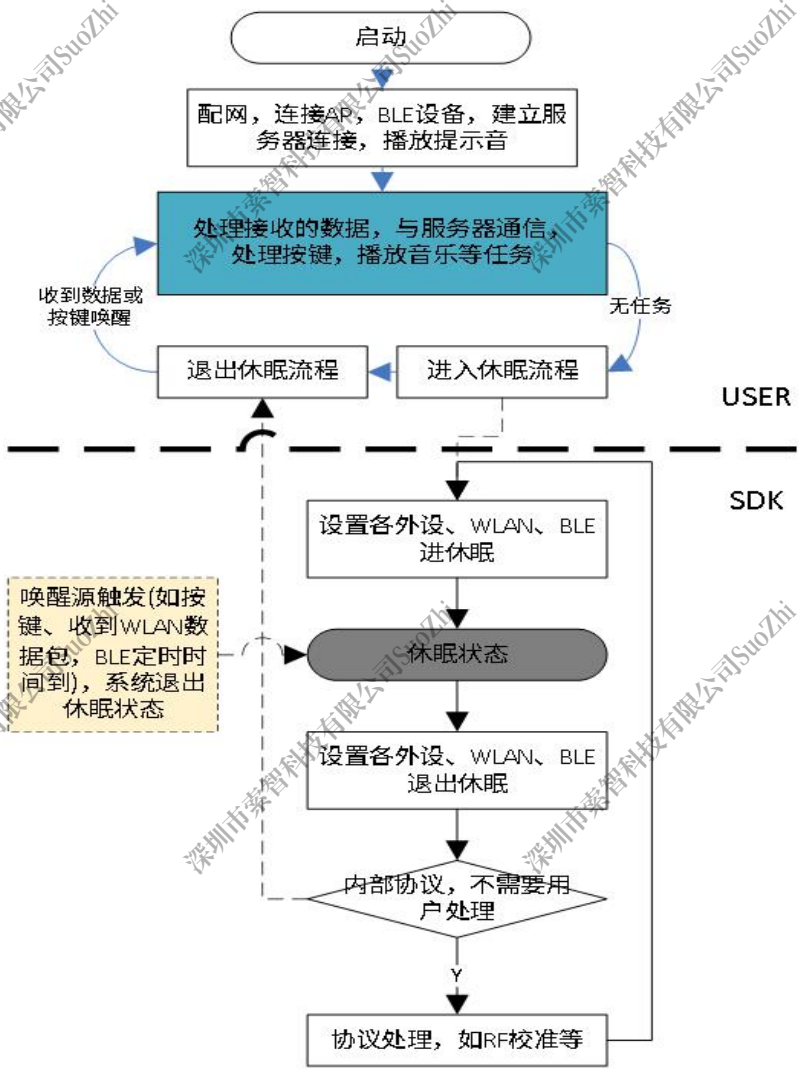
3 技术说明

XR806 SDK 中 PM 是一个系统性的功能，SDK 中的外设、模块都已支持休眠唤醒，但用户新加的设备或模块需要用户处理，可以选择休眠前关闭，唤醒后重新打开，也可以参考 SDK 中的实现机制，为新外设或模块注册休眠唤醒函数。

3.1 休眠唤醒流程

通常使用流程是先配置唤醒源，进入休眠，等待唤醒事件触发唤醒，唤醒后处理唤醒事件，处理完后无任务时再次进入休眠。

图 3-1 PM 整体流程示例



用户调用接口进入休眠时，PM 子系统会先等待所有 Wakelock 锁释放，之后依次调用各个模块和设备的 suspend 函数，之后调用 suspend_noirq 函数，之后保存 CPU 上下文，硬件关闭非必要电源，此时整个芯片处于低功耗状态。收到唤醒源触发后，退出休眠状态，硬件打开之前关闭的电源，恢复 CPU 上下文，依次以相反的顺序调用各个模块和设备的 resume_noirq 函数，之后调用 resume 函数。

在 Sleep 或 Standby 模式下，系统唤醒后的所有软硬件上下文与休眠前一致。在 Hibernation 模式下，进入流程与上图进入流程一致，退出流程即系统复位流程，所有软硬件会被重新初始化。

3.2 休眠前处理

在休眠前通常需要用户对自己新加的模块做相应处理：用户应用做暂停处理，例如用户线程在休眠前需要设置为阻塞、暂停、空闲状态或退出线程，否则可能会在休眠过程中，由于线程切换而被执行到。以播放应用为例，合理的做法是先让播放控制线程暂停播放，再进入休眠，等退出休眠后，检测到有用户按键动作或网络播放指令才继续播放，而不是调度到播放控制线程就立即播放，不受用户行为的控制。

3.3 配置唤醒源

各唤醒源配置说明如下：

唤醒源	配置
普通设备中断唤醒	支持 UART 等普通外设的中断唤醒，以 UART 为例，可以调用 HAL_UART_SetBypassPmMode 把 UART 的 bypassPmMode 配置为 PM_SUPPORT_SLEEP，这样就可以支持 Sleep 模式下使用 UART 唤醒。
普通外设唤醒	GPIO 可以在 Sleep 和 Standby 模式下唤醒，休眠前不需要做任何处理，只要保证休眠期间 GPIO 的配置仍然是中断模式即可。GPADC/LPUART 在 Standby 模式下可以唤醒，时钟工作在 32K，详见《XR806_GPADC 驱动_开发指南》和《XR806_LPUART 驱动_开发指南》。
网络 WLAN 唤醒	只要休眠前不关闭网络，保持 WLAN 连接到路由器，即可使用网络 WLAN 唤醒功能。WLAN 支持不同的 DTIM 配置，对应不同的休眠功耗，可以在休眠功耗和实时性之间做权衡选择。
BLE 唤醒	只要休眠前不关闭 BLE，即可使用 BLE 唤醒功能。系统会在 BLE 收发包时提前一段时间唤醒，唤醒后处理完数据需可次进入休眠。
Wakeup IO 唤醒	Wakeup IO 和 GPIO 复用 pin 脚，需要先配置好对应的 GPIO 后，然后设置为中断模式，再将对应的 GPIO 配置为 Wakeup IO 功能。

注意

Wakeup IO 的唤醒配置与 GPIO 的配置相关。例如，GPIO 无触发时是高电平，下降沿触发中断，则对应 Wakeup IO 的唤醒配置应该是下降沿唤醒，上拉配置与 GPIO 的上拉配置相同。

3.3.1 Wakeup IO Debounce 配置

XR806 芯片支持 Wakeup IO 的 debounce 功能，例如用 Wakeup IO 做按键唤醒，当开启了 debounce 功能时，Wakeup IO 检查到按键按下后并不会立刻唤醒系统，而是在 debounce 指定的 cycle 时间内一直检查按键是否一直被按下，若一直按下则在 debounce 指定的 cycle 时间结束后唤醒，否则不唤醒。

目前有两个时钟源可以选择，分别是 debClk0 和 debClk1，时钟源都是来源于系统的 32K，接口说明见 4.3.2.4 和 4.3.2.5 节。

注意

所有的 Wakeup IO debounce clk 都是来源于这 2 个 clk，通常 clk0 设较小，clk1 设较大，较小可满足 debounce cycle 较短的需求，较大可满足 debounce cycle 较长的需求，要根据所有 Wakeup IO debounce Clk 时钟需求的最大公约数进行配置。

调用以下两个函数对 Wakeup IO 进行设置：

1. 调用 HAL_PRCM_SetWakeupIOxDebSrc 接口选择时钟源，接口说明见 4.3.2.6 节。
2. 调用 HAL_PRCM_SetWakeupIOxDebounce 接口设置 debounce 参数，接口说明见 4.3.2.7 节。

3.4 新加外设或模块的休眠唤醒处理

如果用户要增加新外设或模块，则要考虑新外设或模块的休眠唤醒处理。如果在休眠时不使用外设或模块唤醒系统，为了降低功耗，要把外设或模块关闭或者配置为低功耗模式。一种比较简单的方式是，即在休眠前关闭外设或模块，唤醒后重新打开外设或模块。另一种方法是，即添加外设或模块的休眠唤醒函数，由 PM 子系统管理外设或模块的休眠唤醒，具体的代码实现可参考 SDK 中的驱动模块代码（例如 UART 驱动），实现的关键点如下：

3. 包含 pm/pm.h 头文件。
4. 编写休眠唤醒函数。

- 若休眠唤醒函数不需要在关闭中断情况下调用，则注册 suspend 和 resume 函数。
- 若需要在关闭中断情况下调用，则注册 suspend_noirq 和 resume_noirq 函数。通常设备注册 suspend 和 resume 函数即可。
- 为节省代码空间，休眠唤醒函数可以复用初始化和注销函数，但在初始化和注销函数中要注册和注销 pm 接口，要注意的是为了防止 pm 接口重复注册，需要在初始化和注销函数中区分调用上下文，通常通过全局变量进行状态记录。
- 休眠期间模块不能使用，若调用此模块的接口可以通过记录的状态判断返回错误，以 UART 为例：UART 休眠后不能再继续收发字符，此时要打印字符时，调用发送接口应返回错误，当前 SDK 的处理方式是在上层的 stdio 模块中缓存，待唤醒后再打印出来。

5. 注册及注销

如果要复用初始化/反初始化函数实现休眠/唤醒函数，在注册 pm 接口时，要注意备份初始化的配置参数，以备唤醒后重新初始化时使用。

设备休眠唤醒函数编写时需要注意：

- 设备 suspend 函数功能是将设备设为低功耗模式，如果在休眠期间不再使用，则应处于关闭状态，如果需要作为唤醒源，则应保留最低能工作的资源（power、clk、gating），关闭不需要的资源。
- 设备 resume 函数是将设备设置为正常工作模式，如果休眠时已经完全关闭，则此函数应该重新初始化设备。如果休眠时仍为工作状态，则此函数应该谨慎处理，防止丢失中断。
- 各模块注册休眠唤醒函数时，需要注册进入和退出休眠唤醒的时间，以便其他模块查询系统进出休眠的耗时。时间需要实测得到，可以通过调用系统提供的 time_perf_init，time_perf_tag 接口测量。

3.5 Wakelock 介绍

Wakelock 锁用于阻止系统进入休眠。PM 子系统会维护当前所有锁的持锁状态，若任何一个应用持有 wakelock 锁，则系统不会尝试进入休眠。只有当所有 Wakelock 锁被释放时，用户调用了进入休眠的函数，才会触发系统进入休眠，否则会一直等待，等到所有的锁释放才能触发进入休眠。

Wakelock 锁分为超时锁和长期锁两种类型，两种类型可在一个 Wakelock 锁中共存，即一个 Wakelock 锁先被长期锁锁住后，还可以继续作为超时锁使用，计时结束后会自动释放超时锁，但长期锁仍然有效，反之使用也可以。

超时锁：调用 pm_wakelock_timeout 可以在指定时间内锁住 Wakelock 锁以阻止系统进入休眠，超时后该锁自动释放。若前一次超时计时未到，后一次设置的超时时间会覆盖前一次的超时时间。如果调用 pm_wakelock_timeout 时的超时参数为 0，则超时锁立即失效，等同于释放超时锁。

长期锁：调用 pm_wakelock_lock，长期锁会导致系统一直不会尝试进入休眠。多次加锁时，会增加引用计数，取消时会对引用计数减一，申请和释放需要成对使用，即要与 pm_wakelock_unlock 成对使用。

Wakelock 采用关中断的方式保护临界资源，允许在中断中获取和释放锁。

注意

如果持有 Wakelock 锁不当会导致系统无法进入休眠，因此不推荐用户使用 Wakelock 锁。

3.6 系统启动状态

SDK 提供了 SysGetStartupState 接口获取系统启动状态，系统启动状态由枚举类型 SystemStartupState 定义，详见表 3-1。

表 3-1 系统启动状态定义

系统启动状态	说明
SYS_POWERON	上电重启
SYS_WATCHDOG_CHIP_RST	Watchdog 复位整个芯片重启，如系统挂死后 Watchdog 复位芯片重启
SYS_WATCHDOG_CPU_RST	Watchdog 复位 CPU 重启，如系统挂死后 Watchdog 复位 CPU 重启
SYS_SLEEP	从 Sleep 休眠模式唤醒
SYS_STANDBY	从 Standby 休眠模式唤醒
SYS_HIBERNATION	从 Hibernation 休眠模式唤醒重启
SYS_REBOOT	调用接口 HAL_WDG_Reboot，Watchdog 复位 CPU 重启
SYS_CPU_RST	调用接口 HAL_WDG_ResetCpu，Watchdog 复位 CPU 重启
SYS_NVIC_RST	XR806 不支持

当系统重启时，软件可通过调用 SysGetStartupState 接口获知系统重启的原因，从而进行区分处理。

XR806 提供了硬件 Watchdog 功能，可用于系统挂死时由硬件自动复位系统。Watchdog 支持两种复位方式，分别为 System reset 和 CPU reset，两种复位方式的区别如下：

- **System reset**：该方式会把整个 SoC 复位，包括整个 CPU 核、外围外设、电源配置等。所有外设都不需要在复位过程中保持工作状态时，选择此方式复位系统。
- **CPU reset**：该方式会只 reset CPU 核。通过 HAL_WDG_SetNoResetPeriph 设置不需要 reset 的模块。为了保持某些外设在复位过程中仍然正常工作，选择此方式复位系统。

注意

在某些场景下，系统复位后需要保留某些设备的配置，达到不影响使用体验的效果，例如产品上通过 GPIO 或 PWM 控制着灯，用 Watchdog 复位系统时选择只 reset CPU 的方式复位，这样不会导致因 GPIO 或 PWM 被 reset 而导致的灯闪烁的问题。

4 应用说明

4.1 应用简述

XR806 PM 子系统通过 make menuconfig 进行配置，先选中 Power management，打开 PM 功能（默认开启）。

4.2 配置说明

进入 Power management 可以有以下选项可设置：

图 4-1 PM 功能配置

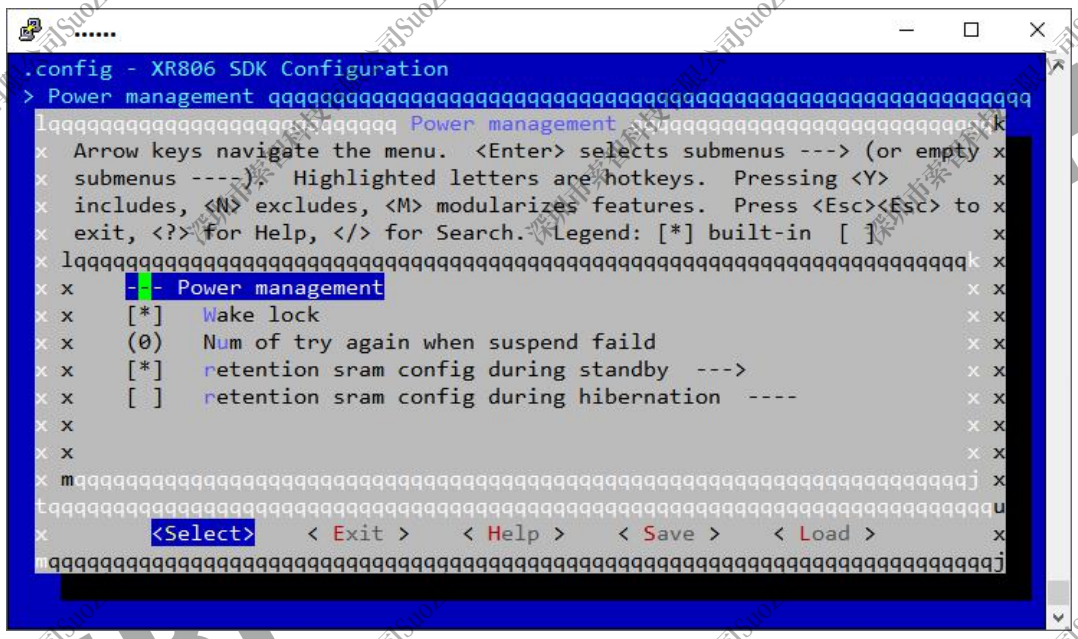


表 4-1 配置项说明

配置项	说明
Wake lock	是否使用 Wake lock 功能，若要使用 WLAN 或 BLE 功能，需要开启此选项。接口说明见 4.3.3 节。
Num of try again when suspend faild	默认为 0，代表如果此次休眠失败则退出休眠，其他值代表此次休眠失败后还会再次尝试进入休眠的次数，不能超过 5。
retention sram config during standby	代表 Standby 休眠过程中哪些 SRAM 会进入 retention 状态（数据不会丢失），没有选择的 SRAM 将会掉电，数据将会丢失。默认为 ALL(0x200000~0x253FFF)，代表全部 SRAM retention。
retention sram config during hibernation	代表 Hibernation 休眠过程中哪些 SRAM 会进入 retention 状态（数据不会丢失），没有选择的 SRAM 将会掉电，数据将会丢失。默认不选中，代表全部关闭。

4.3 接口说明

XR806 PM 接口包括 PM、Wakelock、Wakeup 三部分。

4.3.1 PM 接口

4.3.1.1 pm_register_ops

表 4-2 pm_register_ops 接口函数说明

信息项	说明
原型	int pm_register_ops(struct soc_device *dev)；
功能	注册休眠唤醒接口。
参数	struct soc_device *dev 含义解释：待注册的模块信息 使用说明：注册此设备的休眠唤醒函数
返回值	0：成功 -1：错误

4.3.1.2 struct soc_device_driver

表 4-3 struct soc_device_driver 结构体的成员说明

成员项	说明
const char *name	含义解释：设备驱动的名称 使用说明：需要填写，方便 debug 使用
unsigned int enter_latency	含义解释：设备进入 Standby 模式的耗时 使用说明：单位为 us，需要实测得到
unsigned int exit_latency	含义解释：设备退出 Standby 模式的耗时 使用说明：单位为 us，需要实测得到
int (*suspend)(struct soc_device *dev, enum suspend_state_t state)	含义解释：设备进入休眠模式的函数实现 使用说明：根据参数 state 的不同可进入 Sleep、Standby、Hibernation 模式，在开启中断的上下文中执行
int (*resume)(struct soc_device *dev, enum suspend_state_t state)	含义解释：设备退出休眠模式的函数实现 使用说明：根据参数 state 的不同需要从 Sleep、Standby、Hibernation 模式退出，在开启中断的上下文中执行
int (*suspend_noirq)(struct soc_device *dev, enum suspend_state_t state)	含义解释：设备进入休眠模式的函数实现 使用说明：根据参数 state 的不同可进入 Sleep、Standby、Hibernation 模式，在关闭中断的上下文中执行

成员项	说明
int (*resume_noirq)(struct soc_device *dev, enum suspend_state_t state)	含义解释：设备退出休眠模式的函数实现 使用说明：根据参数 state 的不同需要从 Sleep、Standby、Hibernation 模式退出，在关闭中断的上下文中执行

4.3.1.3 struct soc_device

表 4-4 struct soc_device 结构体的成员说明

成员项	说明
struct list_head node[PM_OP_NUM]	含义解释：用于将设备加入到链表中，PM 子系统内部使用
unsigned int ref	含义解释：用于设备引用计数，PM 子系统内部使用
const char *name	含义解释：设备名称 使用说明：需要填写，方便 debug 使用
const struct soc_device_driver *driver	含义解释：指向设备的驱动 使用说明：见表 4-3
void *platform_data	含义解释：设备私有数据 使用说明：设备自己使用

4.3.1.4 pm_unregister_ops

表 4-5 pm_unregister_ops 接口函数说明

信息项	说明
原型	int pm_unregister_ops(struct soc_device *dev)
功能	注销休眠唤醒接口。
参数	struct soc_device *dev 含义解释：待注销的模块信息 使用说明：注销此设备的休眠唤醒函数
返回值	0：成功 -1：错误

4.3.1.5 pm_enter_mode

表 4-6 pm_enter_mode 接口函数说明

信息项	说明
原型	int pm_enter_mode(enum suspend_state_t state)
功能	设置系统进入低功耗模式。
参数	enum suspend_state_t state

信息项	说明
	含义解释：进入休眠的模式 使用说明：指定系统进入的低功耗模式
返回值	0：成功 -1：错误

4.3.1.6 pm_enter_mode_timeout

表 4-7 pm_enter_mode_timeout 接口函数说明

信息项	说明
原型	int pm_enter_mode_timeout(enum suspend_state_t state, uint32_t tmo)
功能	设置系统进入低功耗模式，超时未进入则返回。
参数	enum suspend_state_t state 含义解释：进入休眠的模式 使用说明：进入系统指定的低功耗模式 uint32_t tmo 含义解释：超时时间，单位 ms 使用说明：进入休眠时，若有模块持有 Wakelock 锁，则系统会等待所有 Wakelock 锁释放，如果在超时时间结束时仍没有释放，则此接口返回错误。
返回值	0：成功 -1：错误

4.3.1.7 pm_suspend_abort

表 4-8 pm_suspend_abort 接口函数说明

信息项	说明
原型	void pm_suspend_abort(void)
功能	中止休眠，如休眠早期阶段发生了中断，处理中断后需要退出休眠，可调用此函数中止休眠流程。此函数可多次调用，休眠前调用无任何效果。
参数	无
返回值	无

4.3.2 Wakeup 接口

4.3.2.1 HAL_Wakeup_GetEvent

表 4-9 HAL_Wakeup_GetEvent 接口函数说明

信息项	说明
原型	uint32_t HAL_Wakeup_GetEvent(void)
功能	获取上次休眠的唤醒事件
参数	无

信息项	说明
返回值	PM_WAKEUP_SRC_XX，定义见附录 A。

4.3.2.2 HAL_Wakeup_SetIO

表 4-10 HAL_Wakeup_SetIO 接口函数说明

信息项	说明
原型	void HAL_Wakeup_SetIO(uint32_t pn, WKUPIO_WK_MODE mode, GPIO_PullType pull)
功能	配置 wakeup IO
参数	uint32_t pn 含义解释：Wakeup IO 编号 使用说明：取值 0~13，与 GPIO 的对应关系见附录 B WKUPIO_WK_MODE mode 含义解释：触发唤醒模式，仅支持上升沿或下降沿。 GPIO_PullType pull 含义解释：上下拉模式
返回值	无

4.3.2.3 HAL_Wakeup_ClrIO

表 4-11 HAL_Wakeup_ClrIO 接口函数说明

信息项	说明
原型	void HAL_Wakeup_ClrIO(uint32_t pn)
功能	取消 wakeup IO 功能
参数	uint32_t pn 含义解释：wakeup IO 编号 使用说明：取值 0~13，与 GPIO 的对应关系见附录 B
返回值	无

4.3.2.4 HAL_PRCM_SetWakeupDebClk0

表 4-12 HAL_PRCM_SetWakeupDebClk0 接口函数说明

信息项	说明
原型	void HAL_PRCM_SetWakeupDebClk0(uint8_t val);
功能	设置 wakeup IO debounce clk0 的时钟
参数	uint8_t val 含义解释：debounce clk0 的时钟分频 使用说明：clk0 frequency 等于 FCLK / (2^val), FCLK 是系统的低频时钟（32000Hz 或 32768Hz）
返回值	无

4.3.2.5 HAL_PRCM_SetWakeupDebClk1

表 4-13 HAL_PRCM_SetWakeupDebClk1 接口函数说明

信息项	说明
原型	void HAL_PRCM_SetWakeupDebClk1(uint8_t val);
功能	设置 wakeup IO debounce clk1 的时钟
参数	uint8_t val 含义解释：debounce clk1 的时钟分频 使用说明：clk1 frequency 等于 FCLK / (2^val), FCLK 是系统的低频时钟（32000Hz 或 32768Hz）
返回值	无

4.3.2.6 HAL_PRCM_SetWakeupIOxDebSrc

表 4-14 HAL_PRCM_SetWakeupIOxDebSrc 接口函数说明

信息项	说明
原型	void HAL_PRCM_SetWakeupIOxDebSrc(uint8_t ioIndex, uint8_t val)
功能	设置指定 Wakeup IO 的时钟源
参数	uint8_t ioIndex 含义解释：wakeup IO 编号，见附录 A 使用说明：取值 0~13，与 GPIO 的对应关系见附录 B uint8_t val 含义解释：时钟源 使用说明：0 是选择 debounce clk0；1 是选择 debounce clk1
返回值	无

4.3.2.7 HAL_PRCM_SetWakeupIOxDebounce

表 4-15 HAL_PRCM_SetWakeupIOxDebounce 接口函数说明

信息项	说明
原型	void HAL_PRCM_SetWakeupIOxDebounce(uint8_t ioIndex, uint8_t val)
功能	设置指定 Wakeup IO 的防抖参数
参数	uint8_t ioIndex 含义解释：wakeup IO 编号，见附录 A 使用说明：取值 0~13，与 GPIO 的对应关系见附录 B uint8_t val 含义解释：防抖参数 使用说明：Debounce clock cycles 等于 clkx * ((val + 1) + 16), clkx 是 debounce clk0 或 debounce clk1
返回值	无

4.3.2.8 HAL_Wakeup_SetTimer_mS

表 4-16 HAL_Wakeup_SetTimer_mS 接口函数说明

信息项	说明
原型	HAL_Wakeup_SetTimer_mS(uint32_t ms)
功能	设置 wakeup timer
参数	uint32_t ms 含义解释：Wakeup timer 定时时长，定时时间到将触发中断并唤醒系统，单位是毫秒，从进入休眠开始计时 使用说明：最小时间为 10 毫秒。使用 32768 Hz 晶振时，最大时间约为 65,535,999 毫秒；使用 32000 Hz 晶振时，最大时间约为 67,108,864 毫秒。
返回值	0：成功 -1：错误

4.3.2.9 HAL_Wakeup_SetTimer_Sec

表 4-17 HAL_Wakeup_SetTimer_Sec 接口函数说明

信息项	说明
原型	HAL_Wakeup_SetTimer_Sec(uint32_t sec)
功能	设置 wakeup timer
参数	uint32_t sec 含义解释：Wakeup timer 定时时长，定时时间到将触发中断并唤醒系统，单位秒从进入休眠开始计时 使用说明：最小时间为 1 秒，最大时间约为 65,535 秒
返回值	0：成功 -1：错误

4.3.3 Wakelock 接口

4.3.3.1 pm_wake_lock

表 4-18 pm_wake_lock 接口函数说明

信息项	说明
原型	int pm_wake_lock(struct wakelock *wl)
功能	锁住 wl，用于阻止系统进入休眠，可以多次加锁，每次加锁引用计数加 1，与 pm_wake_unlock 成对使用，在 unlock 之前一直有效，能起到阻止系统进入休眠的功能
参数	struct wakelock *wl 含义解释：待操作的锁 使用说明：第一次调用前，需要对此 wl 初始化，对成员 name 进行赋值，方便 debug，其他成员赋值为 0
返回值	0：成功 -1：错误

4.3.3.2 pm_wake_unlock

表 4-19 pm_wake_unlock 接口函数说明

信息项	说明
原型	int pm_wake_unlock(struct wakelock *wl)
功能	释放锁 wl，与 pm_wake_lock 成对使用。
参数	struct wakelock *wl 含义解释：待操作的锁 使用说明：只能释放 pm_wake_lock 加的锁
返回值	0：成功 -1：错误

4.3.3.3 pm_wake_lock_timeout

表 4-20 pm_wake_lock_timeout 接口函数说明

信息项	说明
原型	int pm_wake_lock_timeout(struct wakelock *wl, uint32_t timeout)
功能	在指定时间内锁住 wl 以阻止系统进入休眠，超时后该锁自动释放。若前一次超时计时未到，后一次设置的超时时间会覆盖前一次的超时时间
参数	struct wakelock *wl 含义解释：待操作的锁 使用说明：第一次调用前，需要对此 wl 初始化，对成员 name 进行赋值，方便 debug，其他成员赋值为 0 uint32_t timeout 含义解释：超时时间，单位是毫秒 使用说明：超时后自动释放锁。该参数为 0 时表示超时锁立即失效，等同于释放超时锁
返回值	0：成功 -1：错误

4.3.3.4 pm_wakelocks_show

表 4-21 pm_wakelocks_show 接口函数说明

信息项	说明
原型	void pm_wakelocks_show(void)
功能	显示系统当前所有持锁信息。
参数	无
返回值	无

5 示例说明

5.1 示例简介

本示例通过演示配置 Wakeup IO 并进入低功耗模式的过程，简要介绍对应接口的使用方法。

5.1.1 获取方式

低功耗方案示例有示例工程代码，位于 XR806 SDK 的/project/example/pm 目录，以下此示例工程简称为 pm 示例工程。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

5.1.2 准备工作

pm 示例工程的硬件准备有如下。

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 Uart0 插针，用于 console 控制台的输入输出。
3. PC 机：用于镜像烧录和 console 控制的输入输出。
4. 高精度电源：用于给 XR806 芯片供电，并测试统计功耗情况。

pm 示例工程的软件准备，包括烧写工具、代码编译和烧写操作，请参见《XR806_SDK_快速入门指南》。

5.1.3 操作步骤

pm 示例工程无需控制命令，需要先将代码中的 GPIO 和 Wake up IO 修改为实际使用的 GPIO 编号和配置，然后进行编译，重新烧写，完成后复位即可，示例代码自动运行。

5.2 关键实现

系统平台的初始化在 XR806 SDK 的 platform_init()函数中完成，具体请阅读此函数。之后会配置 GPIO 和 Wakeup IO（需要和原理图中一致），然后进入休眠状态，按键后自动唤醒，示例结束，唤醒后处理唤醒事件。

具体操作见下面的示例代码。

```
#Include “pm/pm.h”
#Include “driver/chip/hal_wakeup.h”

int main(void)
{
    platform_init();
```

```
    OS_MSleep(3000);

    /*wakeup timer 10 seconds*/
    HAL_Wakeup_SetTimer_mS(10000);

    pm_enter_mode(PM_MODE_STANDBY); /*10 秒后系统会被唤醒 */
    printf("Wakeup Event:%x\n", HAL_Wakeup_GetEvent());

    GPIO_InitParam param;
    GPIO_IrqParam Irq_param;
    uint32_t io_num, mode, pull = GPIO_PULL_UP;

    param.mode = GPIOx_Pn_F6_EINT;
    param.driving = GPIO_DRIVING_LEVEL_1;
    param.pull = pull ;
    HAL_GPIO_Init(GPIO_PORT_A, WakeIo_To_Gpio(io_num), &param);

    Irq_param.event = GPIO_IRQ_EVT_BOTH_EDGE;
    Irq_param.callback = key_IrqCb;
    Irq_param.arg = (void *)0;
    HAL_GPIO_EnableIRQ(GPIO_PORT_A, WakeIo_To_Gpio(io_num), &Irq_param);
    HAL_Wakeup_SetIO(io_num, mode, pull);

    pm_enter_mode(PM_MODE_STANDBY); /*用户若按下 PA13 对应的按键，即可唤醒系统 */
    printf("Wakeup Event:%x\n", HAL_Wakeup_GetEvent());

    /*Wakeup IO enable */
    HAL_Wakeup_SetIO(WAKEUP_IO_PIN_DEF,WKUPIO_WK_MODE_FALLING_EDGE, GPIO_PULL_UP);

    pm_enter_mode(PM_MODE_HIBERNATION) /*触发 Wakeup IO ，系统会重新启动*/
}
```

5.3 效果展示

在评估板中运行 pm 示例程序后，控制台中的打印如下所示。

```
use default flash chip mJedec 0x0
[FD I]: mode: 0x10, freq: 96000000Hz, drv: 0
[FD I]: jedec: 0x0, suspend_support: 1
mode select:e
... /*省略一些启动信息 */
PMA: appos enter mode: standby
```

```
suspend devices took 10 ms
resume noirq devices took 0 ms
resume devices took 9 ms
Wakeup event:10000 /*查询到唤醒源为：PM_WAKEUP_SRC_WKTIMER */
PMA: appos enter mode: standby
suspend devices took 10 ms
resume noirq devices took 1 ms
key_IrqCb,254
resume devices took 9 ms
Wakeup event:2000 /*查询到唤醒源为：PM_WAKEUP_SRC_WKIO13 */


PMA: appos enter mode: hibernation
net_sys_onoff set net to poweroff
en1: CTRL-EVENT-TERMINATING
… /*省略一些打印 */
defuldefault flash chip mJedec 0x0 /*按键后系统重启 */
[FD I]: mode: 0x10, freq: 96000000Hz, drv: 0
[FD I]: jedec: 0x0, suspend_support: 1
mode select:e

… /*省略一些启动信息 */

PMA: appos enter mode: standby
suspend devices took 10 ms
```

5.4 新加设备的休眠唤醒处理

处理流程见 3.3 章，可以参考 UART 模块，文件位置：src/rom/rom_bin/src/driver/chip/hal_uart.c。

6 常见问题

6.1 进入低功耗模式后功耗偏高

检查所有设备是否处于合理状态：不使用的设备要设置处于低功耗模式或关闭状态；需要工作并能唤醒系统的设备，要设置成可工作的低功耗模式。

设备调用顺序可以通过打印查看，打开打印方式如下：

```
pm_set_debug_mask(ROM_TOTAL_MASKS);
HAL_Wakeup_SetDebugMask(ROM_TOTAL_MASKS);
uart_set_suspend_record(2048); /* 2048 为 UART 关闭期间的打印 buf 的大小 */
```

打印示例如：

```
PMA: appos enter mode: standby
(0x200008)-->xradio(0x2056f8)-->ce(0x205418)-->wdg(0x20018c)
(0x200000)-->xradio(0x2056f8)-->dcache(0x200060)-->Xip(0x21746c)-->Flash(0x217360)-->flashc(0x21722c)-->astdio(0x2052ac)-->uart0(0x2152f8)-->gpio(0x2000ac)-->rtc(0x200168)-->ccm(0x200040)-->nvic(0x200148)
```

第二行为 suspend 流程中依次调用的设备。第三行为 suspend_noirq 流程中依次调用的设备。

上述检查都进行后，功耗仍然不能降低至合理值的，需要测试各个硬件的功耗分量，找出功耗偏大的硬件模块。

6.2 系统进入不了低功耗模式

检查系统中断产生情况，休眠过程中，系统产生了中断，系统休眠流程会被打断而退出。例如某个设备在 suspend_noirq 阶段使用 DMA 接口，而 DMA 使用过程中会产生中断，则系统休眠流程会被打断。

检查系统是否有模块一直持有 wakelock 锁，可通过接口 pm_wakelocks_show 查询 wakelock 持有情况。

6.3 系统进入休眠无反应也无法唤醒

由于大部分代码在 XIP 或 PSRAM 中，而 Flash 和 PSRAM 芯片也会进入休眠，Flash 和 PSRAM 进入休眠后 XIP 和 PSRAM 不能使用，因此要注意新添加设备或模块的休眠顺序是否在 XIP 或 PSRAM 休眠处理流程之后。

6.4 系统不能唤醒

检查唤醒源配置是否正确，Wakeup IO 编号和模式（上升沿还是下降沿）是否正确，GPIO 的上下拉等配置是否正确。

附录 A： 唤醒源定义

唤醒源定义	对应 BIT 定义	说明
PM_WAKEUP_SRC_WKIO0	0x00000001	Wakeup IO 0 唤醒源
PM_WAKEUP_SRC_WKIO1	0x00000002	Wakeup IO 1 唤醒源
PM_WAKEUP_SRC_WKIO2	0x00000004	Wakeup IO 2 唤醒源
PM_WAKEUP_SRC_WKIO3	0x00000008	Wakeup IO 3 唤醒源
PM_WAKEUP_SRC_WKIO4	0x00000010	Wakeup IO 4 唤醒源
PM_WAKEUP_SRC_WKIO5	0x00000020	Wakeup IO 5 唤醒源
PM_WAKEUP_SRC_WKIO6	0x00000040	Wakeup IO 6 唤醒源
PM_WAKEUP_SRC_WKIO7	0x00000080	Wakeup IO 7 唤醒源
PM_WAKEUP_SRC_WKIO8	0x00000100	Wakeup IO 8 唤醒源
PM_WAKEUP_SRC_WKIO9	0x00000200	Wakeup IO 9 唤醒源
PM_WAKEUP_SRC_WKIO10	0x00000400	Wakeup IO 10 唤醒源
PM_WAKEUP_SRC_WKIO11	0x00000800	Wakeup IO 11 唤醒源
PM_WAKEUP_SRC_WKIO12	0x00001000	Wakeup IO 12 唤醒源
PM_WAKEUP_SRC_WKIO13	0x00002000	Wakeup IO 13 唤醒源
PM_WAKEUP_SRC_WKTIMER	0x00010000	Wakeup Timer 唤醒源
PM_WAKEUP_SRC_WLAN	0x00020000	WLAN 唤醒源
PM_WAKEUP_SRC_DEVICES	0x00040000	外设唤醒源(如 GPIO/UART 等)
PM_WAKEUP_SRC_RTC_SEC	0x00080000	RTC 秒闹钟唤醒源
PM_WAKEUP_SRC_RTC_WDAY	0x00100000	RTC 周闹钟唤醒源
PM_WAKEUP_SRC_LPUART	0x00200000	LPUART 唤醒源
PM_WAKEUP_SRC_CAPSENSOR	0x00400000	CAPSENSOR 唤醒源(XR806 不支持)
PM_WAKEUP_SRC_GPADC	0x00800000	GPADC 唤醒源
PM_WAKEUP_SRC_BLE	0x01000000	BLE 唤醒源
PM_WAKEUP_SRC_BT	0x02000000	BT 唤醒源(XR806 不支持)
PM_WAKEUP_SRC_SOFTWARE	0x08000000	软件唤醒源(如休眠过程中调用 pm_suspend_abort 或有模块持有 Wakelock 导致退出休眠流程)

附录 B：Wakeup IO 定义

Wakeup IO 编号	对应 GPIO 编号
WAKEUP_IO0	GPIO_PIN_10
WAKEUP_IO1	GPIO_PIN_11
WAKEUP_IO2	GPIO_PIN_12
WAKEUP_IO3	GPIO_PIN_13
WAKEUP_IO4	GPIO_PIN_14
WAKEUP_IO5	GPIO_PIN_15
WAKEUP_IO6	GPIO_PIN_16
WAKEUP_IO7	GPIO_PIN_17
WAKEUP_IO8	GPIO_PIN_18
WAKEUP_IO9	GPIO_PIN_19
WAKEUP_IO10	GPIO_PIN_20
WAKEUP_IO11	GPIO_PIN_21
WAKEUP_IO12	GPIO_PIN_22
WAKEUP_IO13	GPIO_PIN_23

注：所有 GPIO 均为 PORTA。

著作权声明

商标声明

免责声明