XRADIO PM User Guide

Revision 1.1

Mar 20, 2020

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-11-27	Initial Version
1.1	2020-03-20	Modify wakeup timer maximum time

Table 1- 1 Revision History

Contents

1 电源管理介绍

电源管理（Power Manager）的作用就是降低设备的功耗，提高设备的使用寿命。在不通的场景下可以进入不同的休眠模式达到降低功耗的效果。

1.1 休眠模式

SDK 支持三种休眠模式，分别是 sleep、standby 和 hibernation。三者对应的功耗都是依次递减，而唤醒的速度是依次递增。

- **Sleep**：是 CPU 直接进入 WFI 状态，等待中断等信号唤醒，因此可以支持的普通的中断唤醒，例如 uart 中断等，支持定时器 Timer、按键唤醒。
- **Standby**：是 CPU 进入 WFE 的状态，SRAM 进入 retention 状态，因此支持支持定时器 Timer、按键唤醒，另外还支持网络 Wlan 唤醒，可以通过 ping 本机 IP 即达到唤醒效果。
- **Hibernation**：是 CPU 进入 WFE 的状态，SRAM 是处于掉电状态，起来后只能是重新硬件上电复位，支持支持定时器 Timer、按键唤醒。

1.2 唤醒方式

SDK 支持多种模式，分别有定时器 timer 唤醒、按键唤醒、网络 WLAN 唤醒和普通设备中断唤醒，但是不同的休眠模式会只支持部分的唤醒方式，同一种休眠模式可以同时支持多种唤醒方式。

- **定时器 timer 唤醒**：最短支持 1s，最大范围到 67108S，约 18.64h，由于使用的是内部 32K 的时钟产生的，因此不会百分百准确。
- **按键唤醒**：最多同时支持 10 个唤醒按键。唤醒的按键和普通的 gpio 按键是不一样的，但是会跟普通 gpio 复用 pin 脚。
- **网络 Wlan 唤醒**：支持只要芯片收到本机的网络包就会进行唤醒，例如常见的 ping 包就可以。
- **普通设备中断唤醒**：该唤醒方式只在 sleep 模式下支持，支持一些普通设备中断可以唤醒，例如 adc 按键、uart 中断等。

2 系统层

XRadio 的 SDK 中 PM 是属于系统模块，是一个系统性的功能。目前 SDK 已经封装了多个接口提供给上层使用。

2.1 休眠进入

进入休眠状态的都是统一接口 pm_enter_mode， 其定义如下：

```
enum suspend_state_t {
    PM_MODE_ON              = 0,
    PM_MODE_SLEEP           = 1,
    PM_MODE_STANDBY         = 2,
    PM_MODE_HIBERNATION     = 3,
    PM_MODE_MAX             = 4,
};

int pm_enter_mode(enum suspend_state_t state);
```

功能：当调用 pm_enter_mode 后就会根据参数进入不同的休眠流程，当接收到唤醒源有效唤醒后就会从该函数返回退出。

参数：state 就是休眠模式，只能是 sleep、standby 和 hibernation 三种模式。

返回值：正常返回值为 0， 其他都是不正常返回，代表休眠失败。

2.2 休眠唤醒

- **定时器 timer 唤醒**：是调用 HAL_Wakeup_SetTimer_Sec，其定义如下：

```
#define HAL_Wakeup_SetTimer_Sec(sec) \
    HAL_Wakeup_SetTimer((sec) * HAL_GetLFClock())

int32_t HAL_Wakeup_SetTimer(uint32_t count_32k);
```

功能：配置当进入休眠状态后 sec 秒会产生一次 wakeup 唤醒中断。

参数：sec 就是以秒为单位的定时时间

返回值：正常返回值为 0， 其他都是不正常返回

- **按键唤醒**：wakeup io 和 gpio 是复用 pin 脚的，需要先配置好对应的 gpio 后，然后设置为中断模式或者输入模式（不许允配置为输出模式）和对应的 hold 功能配置为 wakeup io 功能。调用的函数分别是 HAL_GPIO_Init 和 HAL_Wakeup_SetIO。

```
GPIO_InitParam param;
    param.driving = GPIO_DRIVING_LEVEL_1;
    param.pull = GPIO_PULL_UP;
    param.mode = GPIOx_Pn_F0_INPUT;
    HAL_GPIO_Init(BUTTON_WAKEUP_PORT_DEF, BUTTON_WAKEUP_PIN_DEF, &param);
    /*Wakeup IO 5 enable, negative edge,  */
```

```
    HAL_Wakeup_SetIO(WAKEUP_IO_PIN_DEF,WAKEUP_IO_MODE_DEF,
WAKEUP_IO_PULL_DEF);
```

- **网络 Wlan 唤醒**：需要将网络功能开启，和在 prjconfig.h 对应的 PRJCONF_NET_PM_EN 配置为 1，目前网络 Wlan 支持不同的 DTIM 模式。

```
/* network and wlan enable/disable */
#define PRJCONF_NET_EN               1

/* net pm mode enable/disable */
#define PRJCONF_NET_PM_EN             1
```

- **普通设备中断唤醒**：支持 uart、adc 普通中断唤醒，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 uart 唤醒。

```
HAL_Status HAL_UART_SetBypassPmMode(UART_ID uartID, uint8_t mode);
```

功能：开启 uartID 口的普通中断唤醒功能。

参数：uartID 是对应的 uart 号，mode 应选择为 PM_SUPPORT_SLEEP 即可。

返回值：正常返回值为 0， 其他都是不正常返回

2.3 状态查询

- **查询唤醒源**：当配置了多个唤醒源的时候，唤醒后可以调用 HAL_Wakeup_GetEvent 就可以获得休眠唤醒的源。

```
uint32_t HAL_Wakeup_GetEvent(void)
#define PM_WAKEUP_SRC_WKIO0     HAL_BIT(0 )                 /* 0x00000001 */
#define PM_WAKEUP_SRC_WKIO1     HAL_BIT(1 )                 /* 0x00000002 */
#define PM_WAKEUP_SRC_WKIO2     HAL_BIT(2 )                 /* 0x00000004 */
#define PM_WAKEUP_SRC_WKIO3     HAL_BIT(3 )                 /* 0x00000008 */
#define PM_WAKEUP_SRC_WKIO4     HAL_BIT(4 )                 /* 0x00000010 */
#define PM_WAKEUP_SRC_WKIO5     HAL_BIT(5 )                 /* 0x00000020 */
#define PM_WAKEUP_SRC_WKIO6     HAL_BIT(6 )                 /* 0x00000040 */
#define PM_WAKEUP_SRC_WKIO7     HAL_BIT(7 )                 /* 0x00000080 */
#define PM_WAKEUP_SRC_WKIO8     HAL_BIT(8 )                 /* 0x00000100 */
#define PM_WAKEUP_SRC_WKIO9     HAL_BIT(9 )                 /* 0x00000200 */
#endif
#define PM_WAKEUP_SRC_WKTIMER   HAL_BIT(10)                  /* 0x00000400 */
#define PM_WAKEUP_SRC_WKSEV     HAL_BIT(11)                 /* 0x00000800 */
#define PM_WAKEUP_SRC_WLAN      (PM_WAKEUP_SRC_WKSEV)        /* 0x00000800 */
#define PM_WAKEUP_SRC_DEVICES   HAL_BIT(12)                 /* 0x00001000 */
#define PM_WAKEUP_SRC_RTC_SEC   HAL_BIT(13)                 /* 0x00002000 */
#define PM_WAKEUP_SRC_RTC_WDAY  HAL_BIT(14)                 /* 0x00004000 */
```

功能：获取休眠唤醒源

参数：无

返回值：唤醒源 PM_WAKEUP_SRC_XXX

3 驱动层

每个设备驱动都应该支持电源管理的功能，避免设备在进入休眠唤醒后工作不正常。对于休眠唤醒的接口主要有注册电源管理口，电影管理的接口有两对 suspend 和 resume 接口，分别是跟中断相关的 irq 和 noirq 接口。在 suspend 接口中是会保存一些硬件相关的信息如寄存器到 SRAM 中，放到当 SoC 进入 standby 模式的时候会掉电丢失配置;在 resume 的时候再从 SRAM 中回复硬件相关的配置到寄存器中。而对于 Hibernation 模式就不需要保存配置，因为 Hibernation 唤醒后是直接冷启动的过程。

3.1 注册 PM 功能

在每个驱动初始化的时候，可以在最后的进行设备电源管理的注册，调用的接口是 pm_register_ops，在反初始化的时候调用 pm_unregister_ops。

```
int pm_register_ops(struct soc_device *dev);

struct soc_device_driver {
    const char *name;                    /* name of the device driver. */
    int (*suspend)(struct soc_device *dev, enum suspend_state_t state);
    int (*resume)(struct soc_device *dev, enum suspend_state_t state);
    int (*suspend_noirq)(struct soc_device *dev, enum suspend_state_t state);
    int (*resume_noirq)(struct soc_device *dev, enum suspend_state_t state);
};

struct soc_device {
    struct list_head node[PM_OP_NUM];
    unsigned int ref;
    const char *name;                    /* initial name of the device */
    const struct soc_device_driver *driver; /* which driver has allocated this
device */
    void *platform_data;                 /* Platform specific data, device core
doesn't touch it */
};
```

功能：注册设备的电源管理接口

参数：对应的设备内容

返回值：正常返回值为 0， 其他都是不正常返回

3.2 IRQ 类型设备

该类型的设备是会在开启中断的状态下进行休眠和唤醒的操作的，而大部分设备都应该是在 IRQ 类型的设备，分别对应的回调函数是

```
int (*suspend)(struct soc_device *dev, enum suspend_state_t state);
int (*resume)(struct soc_device *dev, enum suspend_state_t state);
```

3.3 NOIRQ 类型设备

该类型的设备是会在关闭中断的状态下进行休眠和唤醒的操作的，保证后续的操作的原子性操作，这些都是底层的设备，如 PRCM、CCMU、PSRAM 等基础设备。

```
int (*suspend_noirq)(struct soc_device *dev, enum suspend_state_t state);
int (*resume_noirq)(struct soc_device *dev, enum suspend_state_t state);
```

4 应用层

三种休眠唤醒的方式都是各有优劣，在应用层需要根据具体的实际场景来选择不同的休眠模式，例如会考虑休眠的方式、唤醒的速度或者休眠的功耗等因素。

	Timer	WakeupIO	WLAN	DevIRQ	Consumption	Resume Speed
Sleep	Y	Y	N	Y	High	Quickly
Standby	Y	Y	Y	N	Mid	Mid
Hibernation	Y	Y	N	N	Low	Slow

4.1 Sleep 应用策略

Sleep 模式的特点是唤醒速度快，应用在那些需要响应速度高的的方案中或者就是需要 uart 唤醒或者 adc 按键唤醒的，因为只有 sleep 模块才能支持普通的中断唤醒。而对于需要频繁进行休眠唤醒的场景，也尽量使用 sleep 模式，因为 standby 和 hibernation 都是会对硬件进行上下电，如果会不断地进行硬件上下电也是会造成功耗损失。

4.2 Standby 应用策略

Standby 模式的特点是兼顾了 sleep 和 hibernation 两者的特点，会保持 SRAM 为 retention 的状态，可以保存数据的完整性但是又可以降低功耗，另外 CPU 核的掉电的，因此需要保存硬件相关的配置到 SRAM 中，但也因为 CPU 是掉电的，所以会大幅度地降低功耗。而且 standby 模式下可以依然保持 WLAN 的连接，当网络收到包的时候可以迅速响应。

4.3 Hibernation 应用策略

Hibernation 模式的特点就是极低的功耗休眠场景，可以降到几微安的功耗。因此该模式主要是用在长时间待机的场景，当需要工作的时候就重新启动进行响应操作。该场景很多都是对应的电源按钮配置，当按下遥控器按钮或者电源按键是就会进入 Hibernation 模式，同时又可以保持被控制起来的场景。

5 相关事项

5.1 SRAM 相关

对于 XR872 系列的芯片是支持 SRAM 分片 retention 的，可以在某些应用场景下，把不需要的 SRAM 给关掉，只让部分 SRAM 处于 retention 的状态。可以调用的函数是 pm_standby_sram_retention_only。

```
void pm_standby_sram_retention_only(uint32_t sramN)

enum pm_sram_N {
    PM_SRAM_0   = PRCM_SYS_SRAM_32K_SWM4_BIT,
    PM_SRAM_1   = PRCM_SYS_SRAM_32K_SWM3_BIT,
    PM_SRAM_2   = PRCM_SYS_SRAM_352K_SWM2_BIT,
    PM_SRAM_3   = PRCM_SYS_CACHE_SRAM_SWM1_BIT,
};
```

功能：配置在 standby 的时候只会 retention 指定的 SRAM

参数：sramN 是 pm_sram_N 类型的或组合

返回值：无

使用场景：当休眠的时候，需要尽量降低待机功耗，但是又希望起来的速度可以适当地提升，这个方案就可以只 retention 存放 Bootloader 阶段的 SRAM，在进入 standby 的时候把 Bootloader 先拷贝到指定的 SRAM 地址，当收到唤醒操作的时候直接跳转到 SRAM 中的 Bootloader 的其实地址中，这样可以减少从 flash 中加载 Bootloader 的过程，又可以把大部分的 SRAM 给关掉达到降低功耗的目的。

5.2 Debounce 相关

对于 XR872 和 XR808 系列的芯片是支持 wakep IO 的 debounce 功能，就是当使用 wakeup IO 作为唤醒源的时候，当是上升沿触发唤醒。当开启了 debounce 功能的时候，当 wakeup io 上升沿唤醒中断到的时候并不会立刻产生中断，而是需要延迟 debounce 指定的 cycle 数后再进行唤醒，调用的函数接口是需要配置时钟源和延迟的 cycle 数，三个函数组合来实现该功能。

目前是有两个时钟源可以选择，分别是 debClk0 和 debClk1，一般都是一个使用为高频晶振源，一个为低频晶振源，这样可以防止高频源用于长时间计算而导致功耗提高。

```
void HAL_PRCM_SetWakeupDebClk0(uint8_t val);
void HAL_PRCM_SetWakeupDebClk1(uint8_t val);
```

功能：设置 debClk0 和 debClk1 的源时钟。

参数：0 是代表 32K 低频时钟源， 1 代表高频晶振源

返回值：无

```
void HAL_PRCM_SetWakeupIOxDebSrc(uint8_t ioIndex, uint8_t val);
```

功能：设置指定 wakeup io 的 debounce 源。

参数：ioIndex 是指定的 wakeup io 号， val 是指定的 debclk 源

返回值：无

```
void HAL_PRCM_SetWakeupIOxDebounce(uint8_t ioIndex, uint8_t val);
```

功能：设置制定 wakeup io 的 debounce cycle 数。

参数：ioIndex 是指定的 wakeup io 号， val 是指定的 debclk cycle 数为(16+val)。

返回值：无

5.3 供电相关

供电模式是支持两种，分别是 LDO 模式和 DCDC 模式。

- **LDO 模式:** 该模式是 SoC 的默认的设置，可以配置 LDO 的范围是 1.4V~3.6V 来满足不同的方案场景使用，在满足需求的前提下应该选择低电压的。
- **DCDC 模式:** 该模式相对与 LDO 模式而言会效率会更加高，在使用上需要配置一些来开启 DCDC 的模式。在硬件上，需要往 DCDC pin 输出需要的电压，然后将 SoC 内部的 LDO 降到低于外部的 DCDC 电压，这样就会形成电压差，外部 DCDC 灌电到 SoC 内部使用，同时需要把 PA23 配置为 DCXO 模式，在进入 standby 的时候 PA23 就会自动拉低，告知 DCDC 电源停止灌电，在休眠的状态的时候会使用内部 LDO 模式达到最低功耗的效果。

5.4 复位相关

复位模式是支持两种，分别是 System reset 模式和 CPU reset 模式。

- **System reset 模式**：该模式是会把整个 SoC 都会复位，包括整个 CPU 核、外围总线的控制器和 SRAM 等整个电路。
- **CPU reset 模式**：该模式是会只 reset CPU 核，而外围总线的控制器和 SRAM 是不会被 reset 的。

使用场景：在某些场景中在系统复位后需要保留某些设备的配置，达到快速响应的效果，例如想在 reset 后，uart 的通讯是可以保持的起来后可以快速地响应 uart 的中断，在这种场景就可以调用 noreset 的接口 HAL_WDG_SetNoResetPeriph。

```
void HAL_WDG_SetNoResetPeriph(uint32_t periphMask, int8_t enable)
```

功能：设置当 noreset cpu 的时候，可以指定只有某些模块不被 reset。

参数：periphMask 为 CCM_BusPeriphBit 的或组合， enable 就是开启还是关闭 noreset 模块。

返回值：无

5.5 软件相关

5.5.1 Tickless

SDK 中使用的是 FreeRTOS 的操作系统，FreeRTOS 里面也会自带有 PM 相关的一些功能和策略，例如 Tickless，

而且该功能是默认打开的。OS 进入空闲任务后，首先要计算可以执行低功耗的最大时间，也就是求出下一个要执行的高优先级任务还剩多少时间。 然后就是把低功耗的唤醒时间设置为这个求出的时间，到时间后系统会从低功耗模式被唤醒，继续执行多任务，这个就是所谓的 tickless 模式。

空闲钩子设置低功耗时，需要在 FreeRTOSConfig.h 中，将宏 configUSE_IDLE_HOOK 置为 1，然后自己实现固定接口：void vApplicationIdleHook(void);同时，这个钩子函数不可以调用会引起空闲任务阻塞的 API 函数（例如：vTaskDelay()、带有阻塞时间的队列和信号量函数），在钩子函数内部使用是被允许的。

在空闲任务钩子函数中设置微处理器进入低功耗模式来达到省电的目的。因为系统要响应系统节拍中断事件，因此使用这种方法会周期性的退出、再进入低功耗状态。如果系统节拍中断频率过快，则大部分电能和 CPU 时间会消耗在进入和退出低功耗状态上。

5.5.2 AudioPlay

SDK 中已经集成了播控流程 Cedarx，可以在工作状态下正常播歌，但是在需要进入休眠流程前，无论是 sleep、standby 或者 hibernation 都是需要先进入休眠流程钱关闭 Cedarx，然后再进入休眠；唤醒后再重新启动 Cedarx 进行播放。