PHY622X

PWR MGR Application Note

Version 1.0

Author: Bingliang.Lou

Security: Public

Date: 2020.12

PhyPlus

Revision History

Revision	Author	Date	Description
v1.0	Bingliang.Lou	2020.12	First Edition

目录

图表目录

1 概述

PHY622X 低功耗相关的功能在 PWR_MGR 模块实现，对应的 API 代码存放在 SDK 的 components\driver\ pwrmgr 目录下的 pwrmgr.c 和 pwrmgr.h 中。

PHY622X 有四类功耗模式：

- **普通模式：**CPU 和外设全速运行，不休眠。
- **CPU 休眠模式：**只有 CPU 会进入休眠，可由中断或事件唤醒。该模式由 OS 自行控制，应用程序无需干预。
- **深度休眠模式：**CPU 和大部分外设都会进入休眠。应用程序应根据需要设置休眠唤醒源（GPIO pin 和触发方式）和内存保持(memory retention,以保持运行时上下文) 。
- **standby 模式：**除了 AON 和一块 RAM 区域内存保持外，CPU 和其它外设都进入休眠。仅维持 RAM0 区域内容。应用程序应根据需要设置唤醒源（GPIO pin 和触发方式）。
- **关机模式：**除了 AON 外，CPU 和其它外设都进入休眠。唤醒后相当于系统重启，不能维持运行时上下文。应用程序应根据需要设置唤醒源（GPIO pin 和触发方式）。

1.1 PWR_MGR 原理框图

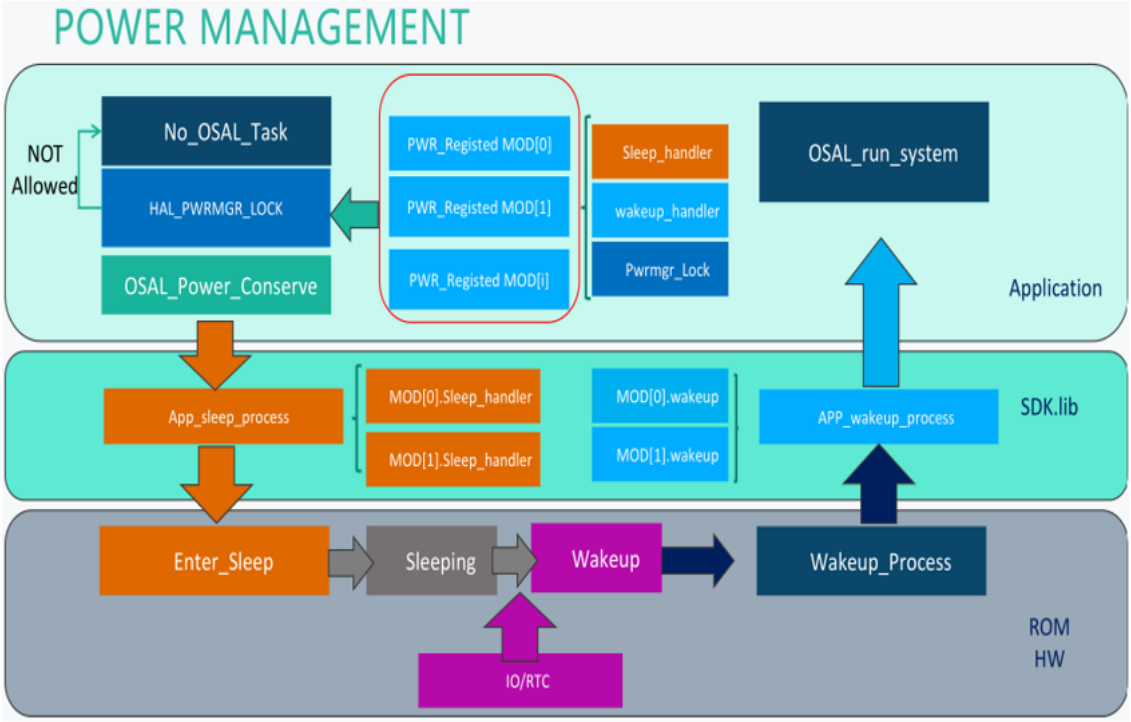

图 1：深度休眠模式原理框图

在 CPU/深度休眠模式下，当系统处于 IDLE 状态时，调用 sleep_process()尝试进入休眠模式。若符合 CPU 休眠条件，则调用__WFI()等待中断唤醒后返回；否则的话，系统将进入深度休眠，在此之前，应用程序通过 hal_pwrmgr_register() API 注册的 sleep_handler()函数都会被回调，然后系统进入休眠；而当有 IO/RTC 触发唤醒时，系统会被唤醒并做相应的初始化。在系统唤醒过程中，应用程序通过 hal_pwrmgr_register() API 注册的所有 wakeup_handler()函数也都会被回调，然后由 OSAL 调度 task。

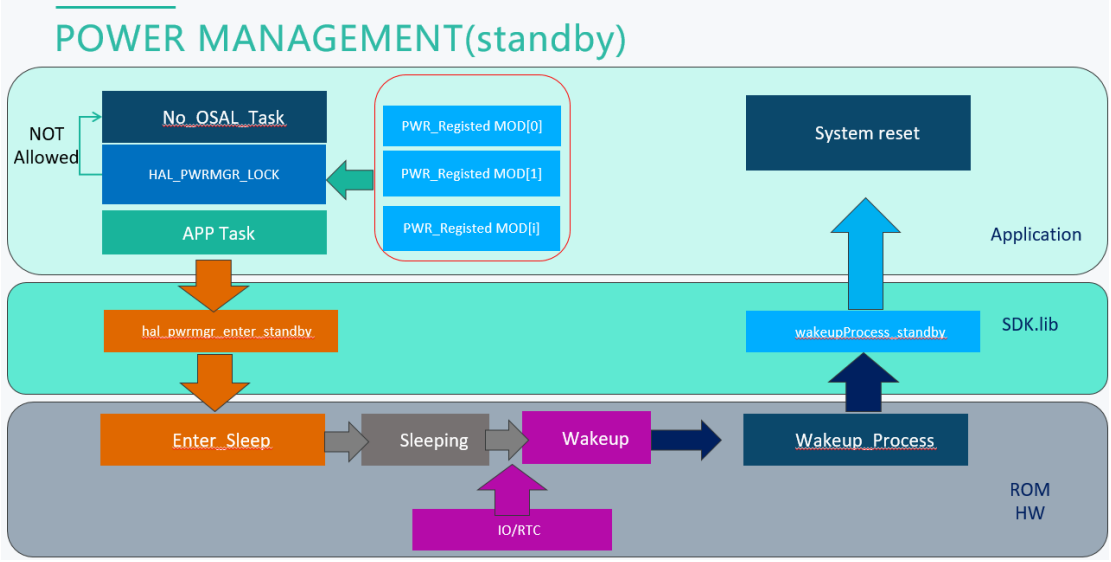

图 2：standby 模式原理框图

在 standby 模式下，APP Task 调用 hal_pwrmgr_enter_standby()进入 standby 模式；而当有 IO 触发唤醒时，系统会被唤醒并调用 wakeupProcess_standby()，若系统满足唤醒的条件，则触发系统 reset。

2 PWR_MGR 模块 API

2.1 数据结构和类型

2.1.1 MODULE_e 类型

在 mcu_phy_bumbee.h 文件中定义了下列 module ID。

```
typedef enum {
    MOD_NONE          = 0, MOD_CK802_CPU    = 0,
    MOD_DMA           = 3,
    MOD_AES           = 4,
    MOD_IOMUX         = 7,
    MOD_UART0         = 8,
    MOD_I2C0          = 9,
    MOD_I2C1          = 10,
    MOD_SPI0          = 11,
    MOD_SPI1          = 12,
    MOD_GPIO          = 13,
    MOD_QDEC          = 15,
    MOD_ADCC          = 17,
    MOD_PWM           = 18,
    MOD_SPIF          = 19,
    MOD_VOC           = 20,
    MOD_TIMER5        = 21,
    MOD_TIMER6        = 22,
    MOD_UART1         = 25,

    MOD_CP_CPU        = 0 + 32,
    MOD_BB            = MOD_CP_CPU + 3,
    MOD_TIMER         = MOD_CP_CPU + 4,
    MOD_WDT           = MOD_CP_CPU + 5,
    MOD_COM           = MOD_CP_CPU + 6,
    MOD_KSCAN         = MOD_CP_CPU + 7,
    MOD_BBREG         = MOD_CP_CPU + 9,
    BBLL_RST          = MOD_CP_CPU + 10, //can reset,but not gate in here
    BBTX_RST          = MOD_CP_CPU + 11, //can reset,but not gate in here
    BBRX_RST          = MOD_CP_CPU + 12, //can reset,but not gate in here
    BBMIX_RST         = MOD_CP_CPU + 13, //can reset,but not gate in here
    MOD_TIMER1        = MOD_CP_CPU + 21,
    MOD_TIMER2        = MOD_CP_CPU + 22,
    MOD_TIMER3        = MOD_CP_CPU + 23,
    MOD_TIMER4        = MOD_CP_CPU + 24,
```

```
    MOD_PCLK_CACHE    = 0 + 64,
    MOD_HCLK_CACHE    = MOD_PCLK_CACHE + 1,

    MOD_USR0          = 0 + 96,
    MOD_USR1          = MOD_USR0 + 1,
    MOD_USR2          = MOD_USR0 + 2,
    MOD_USR3          = MOD_USR0 + 3,
    MOD_USR4          = MOD_USR0 + 4,
    MOD_USR5          = MOD_USR0 + 5,
    MOD_USR6          = MOD_USR0 + 6,
    MOD_USR7          = MOD_USR0 + 7,
    MOD_USR8          = MOD_USR0 + 8,
} MODULE_e;
```

表 1：MODULE_e 类型定义 module ID

2.1.2 pwrmgr_Ctx_t

PWR_MGR 模块为每个注册的模块(与 MODULE_e 对应)维护一个该结构类型的变量。最多 10 个。

```
typedef struct _pwrmgr_Context_t {
    MODULE_e        moudle_id;
    bool            lock; //  为 TRUE 时表示禁止休眠；反之，允许休眠
    pwrmgr_Hdl_t sleep_handler; //  该模块对应的进入休眠之前会被调用的回调函数
    pwrmgr_Hdl_t wakeup_handler; //  该模块对应的唤醒之前会被调用的回调函数
} pwrmgr_Ctx_t;
```

2.1.3 pwroff_cfg_t

在系统调用 hal_pwrmgr_poweroff() API 进入关机模式之前，需要设置的唤醒源(GPIO pin)和触发方式保存在该类型的变量中。

```
typedef struct {
    gpio_pin_e pin;
    gpio_polarity_e type;    // POL_FALLING or POL_RISING
} pwroff_cfg_t;
```

2.2 APIs

PWR_MGR 模块的接口函数如下：

1. int hal_pwrmgr_init(void);

模块初始化函数

2. bool hal_pwrmgr_is_lock(MODULE_e mod);

查询模块 mod 的 lock 状态。TRUE:禁止休眠；FALSE:使能休眠

3. int hal_pwrmgr_lock(MODULE_e mod);

设置模块 mod 的 lock 为 TRUE,并禁止休眠

4. int hal_pwrmgr_unlock(MODULE_e mod);

设置模块 mod 的 lock 为 FALSE,并使能休眠

5. int hal_pwrmgr_register(MODULE_e mod, pwrmgr_Hdl_t sleepHandle, pwrmgr_Hdl_t wakeupHandle);

注册模块 mod,并提供相应的休眠/唤醒回调函数

6. int hal_pwrmgr_unregister(MODULE_e mod);

取消注册模块 mod

7. int hal_pwrmgr_wakeup_process(void) __attribute__((weak));
8. int hal_pwrmgr_sleep_process(void) __attribute__((weak));

休眠/唤醒过程中 PWR_MGR 模块定义的处理函数，应用程序不需要也不应该调用它们

9. int hal_pwrmgr_RAM_retention(uint32_t sram);

配置需要保持的 RAM 区域，可选的有 RET_SRAM0 | RET_SRAM1 | RET_SRAM2

10. int hal_pwrmgr_RAM_retention_clr(void);
11. int hal_pwrmgr_RAM_retention_set(void);

使能/清除对配置的 RAM 区域的保持(retention)功能

12. int hal_pwrmgr_LowCurrentLdo_enable(void);
13. int hal_pwrmgr_LowCurrentLdo_disable(void);

使能/禁用调节 LowCurrentLDO 的输出电压。

14. int hal_pwrmgr_poweroff(pwroff_cfg_t *pcfg, uint8_t wakeup_pin_num);

配置唤醒源后系统进入关机模式

15. void system_on_handler(GPIO_Pin_e pin, uint32_t timer);

关机模式下，系统唤醒处理函数

3 低功耗模式使用注意事项

为使用低功耗模式，编程时需要注意以下几个方面：

- 配置 CFG_SLEEP_MODE 宏：

在工程中需要设置 CFG_SLEEP_MODE=PWR_MODE_SLEEP 以使能休眠模式，而在其它模式下系统将不会进入休眠。

- 初始化 pwrmgr 模块：

若要使用低功耗模式，在系统初始化时须调用 hal_pwrmgr_init()以初始化 pwrmgr 模块。

- 配置不同 RAM 的 retention 属性：

若要使用低功耗模式，在系统初始化时需要调用 hal_pwrmgr_RAM_retention()以在系统休眠后保留对应 memory 区域的内容。可选的 RAM 区域有 RET_SRAM0，RET_SRAM1，和 RET_SRAM2，用户根据需要自行指定要保留的 RAM 区域。

- 选择模块 ID 并注册休眠或唤醒回调函数：

PHY62 系列 SDK 在 mcu_phy_bumbee.h 文件中定义了一些 MODULE_e 类型的模块名/ID，在 pwrmgr 模块中作为模块 ID 使用。APP task 可以在 MOD_USR1 和 MOD_USR8 之间任选一个作为模块 ID 使用。

若要使用低功耗模式，APP task 在初始化时需要调用以下函数进行注册：

```
hal_pwrmgr_register(MODULE_e mod, pwrmgr_Hdl_t sleepHandle,
  pwrmgr_Hdl_t wakeupHandle);
```

其中 mod 就是模块 ID，是必须项，在 MOD_USR1 和 MOD_USR8 之间任选一个即可；

sleepHandle()和 wakeupHandle ()分别对应的是可选的休眠和唤醒的回调函数。

一个常用的做法是：用户可以在 sleepHandle()函数中做一些休眠唤醒使用的 pin 和相应属性的设置；而在 wakeupHandle()函数中做唤醒源的判定和初始化，以便系统唤醒后能恢复回到休眠前的状态。

- 控制是否允许 APP task 进入休眠：

在使用低功耗模式时，APPtask 可以调用 pwrmgr 模块提供的下列接口来查询或控制是否允许进入休眠：

hal_pwrmgr_is_lock(MODULE_e mod)： 查询是否允许模块进入休眠；

hal_pwrmgr_lock(MODULE_e mod)： 禁止模块进入休眠；

hal_pwrmgr_unlock(MODULE_e mod)： 允许模块进入休眠

4 功耗相关实测数据

4.1 Scan 时测算的功耗

TxPwr：TX_POWER_N2DBM; PktFmt：PKT_FMT_BLE1M; Log：off

HCLK	32KCLK	扫描时间	current	底电流	峰值电流	平均电流	时间跨度
64M	RCOSC	500ms	4.886mA	0.723mA	5.099mA	1.699mA	29000rdgs
48M	RCOSC	500ms	4.509mA	0.720mA	4.665mA	1.624mA	43000rdgs
16M	RCOSC	500ms	3.425mA	0.720mA	3.825mA	1.429mA	23000rdgs
16M	XTAL	500ms	3.424mA	0.720mA	3.819mA	1.432mA	22000rdgs
48M	XTAL	500ms	4.496mA	0.721mA	4.686mA	1.622mA	60000rdgs
64M	XTAL	500ms	4.779mA	0.725mA	5.062mA	1.702mA	22000rdgs

表 2：SCAN 功耗测算值

4.2 Advertising 时测算的功耗

HCLK	32KCLK	wakeup time	work time	sleep time	total time	wakeup curt	RF curt	work curt	sleep curt	average power
64M	RCOSC	853.9	4211.1	500	505.065	3.071	6.199	2.549	5.8	0.0451213
48M	RCOSC	885.9	4266	500	505.152	3.03	5.86	2.258	5.6	0.04338933
16M	RCOSC	964.1	4256.7	500	505.221	3.028	5.244	1.648	5.6	0.03880097
16M	XTAL	885.3	4344.9	500	505.23	3.037	5.254	1.701	5.3	0.03837671
48M	XTAL	899.1	4188	500	505.087	3.052	5.817	2.188	5.4	0.04244181
64M	XTAL	863.9	4243.8	500	505.108	3.096	6.168	2.547	5.25	0.04456077

表 3：advertising 功耗测算值

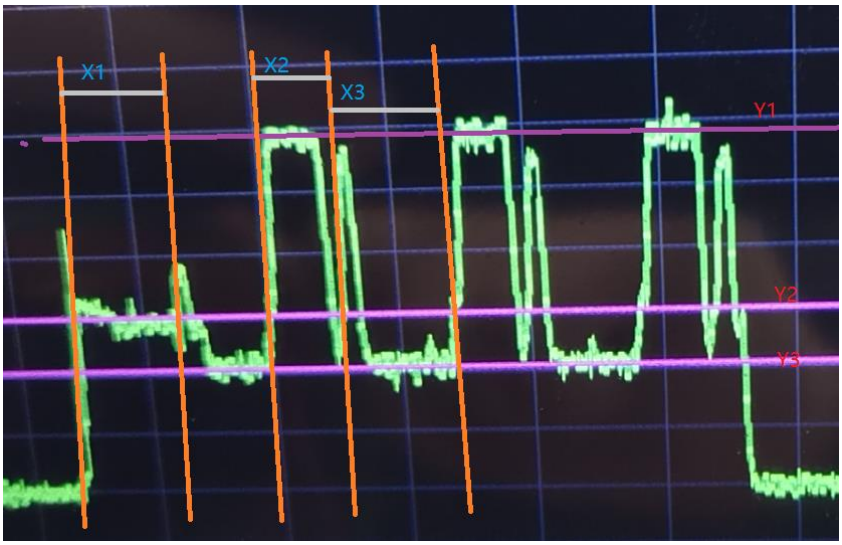

图 3：功耗数据图示

参数	说明	单位
X1	wakeup time	us
X2	1 cycle RF time	us
X3	1 cycle nonRF time	us
work time = 3*(X2+X3)		
total time = (X1+3*(X2+X3))/1000+sleep time		

表 4：功耗数据清单 – 需填入数据

参数	说明	单位
Y1	RF current	mA
Y2	wakeup current	mA
Y3	work current	mA
sleep current	直接从功率计上读取	uA
total power=(X1*Y2+X2*Y1*3+X3*Y3*3+slp_tim*slp_cur)/1000		
average power=total power / total time		

表 5：功耗数据清单 – 自动生成数据

其中，wakeup 对应的时间就是深度休眠情况下唤醒的时间；total 对应的时间是系统唤醒后再次进入休眠的总时间。

功耗估算：

平均功耗 = 总功耗 / 总时间

总功耗 = （唤醒时的电流 X 唤醒时间 + 3 X RF 时的电流 X RF 时间 + 3 X 工作电流 X 工作时间）/ 1000 + 睡眠时的电流 X 睡眠时间

总时间 = （唤醒时间 + 3 X RF 时间 + 3 X 工作时间）/ 1000 +睡眠时间

电池使用时间 = 电池容量 X 3600 / 平均功耗 (秒)

例如：电池容量为 520 mAh，平均功耗为 0.03880097，则电池使用时间为

520 X 3600 / 0.03880097 = 48246216 秒 = 13402 小时 = 558 天 = 1.5 年

4.3 系统上电启动时间

ble_scanner prj

g_system_clk	g_clk32K_config	reset ->main entry	main entry -> ble init done
SYS_CLK_DLL_48M	CLK_32K_XTAL	320ms	164ms
SYS_CLK_XTAL_16M	CLK_32K_XTAL	331ms	209ms

表 6：上电启动时间