Technical Article

基于 FreeRTOS 的 AWRL6432 低功耗底层实现详解

Chris Meng

低功耗是 AWRL6432 系列产品的一个重要功能。AWRL6432 的软件是基于 FreeRTOS 操作系统。FreeRTOS 操作系统的内核是支持低功耗的，但在不同硬件上的实现会有一些区别。本文结合 AWRL6432 MMWAVE_L_SDK 驱动以及例程，详细介绍在 AWRL6432 上的 FreeRTOS 低功耗底层内核驱动实现和低功耗示例的实现。本文的内容也适用于 IWRL6432、AWRL1432 和 IWRL1432。

FreeRTOS 里的低功耗支持

操作系统常用的低功耗实现方法是在空闲任务里让微控制器处于低功耗状态。如果使用系统节拍(tick)，低功耗省电的效果会受到定期退出并重新进入低功耗状态和处理节拍中断的限制。如果节拍中断的频率太高，则每次节拍进入和退出低功耗状态所消耗的能量和时间可能极大降低潜在的省电收益。FreeRTOS 使用无节拍空闲模式，操作系统会在空闲期间（在没有执行的应用任务的期间）停止周期性节拍中断、然后在节拍中断里重新启动时对实时系统节拍计数值进行更正调整。停止节拍中断可以使微控制器保持在深度节能状态、直到发生中断（微控制器内部时钟中断，或者外部信号触发中断）或者实时系统内核应该切换任务时，才将处于低功耗深度节能状态的微控制器转换到 ready 模式。

AWRL6432 FreeRTOS 内核驱动的低功耗实现

在 AWRL6432 的 FreeRTOS 里，通过在 FreeRTOSConfig.h 中将 configUSE_TICKLESS_IDLE 定义为 2，使能了用户自定义的无节拍空闲功能。使用 vPortSuppressTicksAndSleep()函数为相关功能的钩子函数。当软件进入空闲任务后，就会调用 vPortSuppressTicksAndSleep()函数。vPortSuppressTicksAndSleep 函数中会调用 Power_idleFunc 函数，而 Power_idleFunc()函数需要在 enablePolicy 标志位置位后，才会真正调用低功耗相关函数 Power_enterLPDS()。Power_enterLPDS()函数基于汇编，在 power_xwrLx4xx_asm.S(\source\drivers\power\v0)里实现。

FreeRTOSConfig.h(C:\ti\MMWAVE_L_SDK_05_05_02_00\source\kernel\freertos\config\xwrL64xx\m4f)

```
#define configUSE_TICKLESS_IDLE                 (2)  /* When 2, providing custom implementation for
vPortSuppressTicksAndSleep() as a hook function*/
```

power_xwrLx4xx_freertos.c (c:\ti\mmwave_l_sdk_05_05_02_00\source\kernel\freertos\dpl\common)

```
/* Tickless Hook */
void vPortSuppressTicksAndSleep(TickType_t xExpectedIdleTime)
{
#if (configUSE_TICKLESS_IDLE != 0)
    unsigned long long time;
    idleTicks = xExpectedIdleTime;
    // convert ticks to microseconds
    time = ((idleTicks / configTICK_RATE_HZ) * 1000000);
    Power_idleFunc(time);
#endif
}
```

power_xwrLx4xx.c (C:\ti\MMWAVE_L_SDK_05_05_02_00\source\drivers\power\v0)

```
/*
 *  ======== Power_enablePolicy ========
 *  Run the configured policy
 */
void Power_enablePolicy(void)
{
```

```
    Power_module.enablePolicy = TRUE;
    DebugP_log("Power: enable policy");
}
…
/*
 *  ======== Power_idleFunc ========
 *  Function needs to be plugged into the idle loop.
 *  It calls the configured policy function if the
 *  'enablePolicy' flag is set.
 */
void Power_idleFunc(unsigned long long sleepTimeus)
{
    if (Power_module.enablePolicy) {
        if (Power_module.policyFxn != NULL) {
            DebugP_log("Power: calling policy function (%p)",
                    (uintptr_t) Power_module.policyFxn);
            (*(Power_module.policyFxn))(sleepTimeus);
        }
    }
}
…
/*
 *  ======== Power_sleep ========
 */
int_fast16_t Power_sleep(uint_fast16_t sleepState)
{
…
                Power_enterLPDS(PRCMSleepEnter);
…
```

AWRL6432 例程中低功耗实现

下面用两个 MMWAVE_L_SDK 里的实例来说明低功耗代码具体实现调用过程。

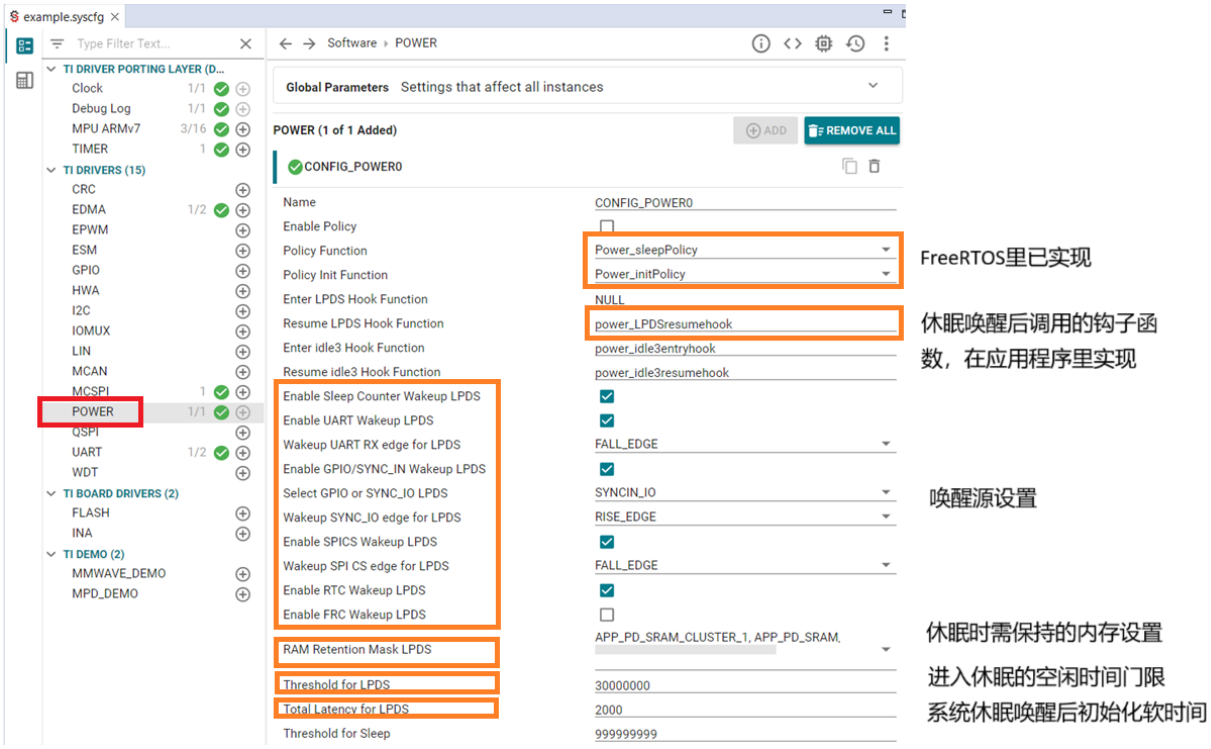
单任务例程

首先来看一下 MMWAVE_L_SDK 驱动里的 power mode 例程
（C:\ti\MMWAVE_L_SDK_05_05_02_00\examples\drivers\power\power_modes\xwrL64xx-evm）。

AWRL6432 的用户如果要在软件里使能深度睡眠的节电模式，先需要在 syscfg 里配置相应的 power 设置。设置后 syscfg 编译出来的 ti_drivers_config.c 文件里会有一个 power 配置的全局变量 Power_ConfigV1，这个全局变量会在功耗驱动中被使用。

power_xwrLx4xx.c (c:\ti\mmwave_l_sdk_05_05_02_00\source\drivers\power\v0):extern Power_ConfigV1 Power_config;

power_xwrLx4xx_freertos.c (c:\ti\mmwave_l_sdk_05_05_02_00\source\kernel\freertos\dpl\common):extern Power_ConfigV1 Power_config;



ti_drivers_config.c

```
const Power_ConfigV1 Power_config = {
    .policyInitFxn              = Power_initPolicy,
    .policyFxn                  = Power_sleepPolicy,
    .enterLPDSHookFxn           = NULL,
    .resumeLPDSHookFxn          = power_LPDSresumehook,
    .enteridle3HookFxn           = power_idle3entryhook,
    .resumeidle3HookFxn          = power_idle3resumehook,
    .enablePolicy                  = false,
    .enableSleepCounterWakeupLPDS   = true,
    .enableUARTWakeupLPDS           = true,
    .enableSPICSWakeupLPDS          = true,
    .enableRTCWakeupLPDS            = true,
    .enableFRCWakeupLPDS            = false,
    .enableGPIOSyncIOWakeupLPDS           = true,
    .selectGpioSyncIOLpds          = POWER_SYNCIN_IO_WAKEUP_LPDS,
    .wakeupSyncIOEdgeLPDS           = PRCM_LPDS_RISE_EDGE,
    .wakeupSpiEdgeLPDS             = PRCM_LPDS_FALL_EDGE,
    .wakeupUartEdgeLPDS             = PRCM_LPDS_FALL_EDGE,
    .LPDSThreshold                   = 30000000,
    .totalLatencyForLPDS                = 2000,
    .sleepThreshold                   = 999999999,
    .totalLatencyForSleep                = 1500,
    .idleThreshold                   = 2000,
    .totalLatencyForIdle                = 1000,
    .ramRetentionMaskLPDS           = PRCM_APP_PD_SRAM_CLUSTER_1|PRCM_APP_PD_SRAM_CLUSTER_2|
PRCM_APP_PD_SRAM_CLUSTER_3|PRCM_FEC_PD_SRAM_CLUSTER_1|PRCM_FEC_PD_SRAM_CLUSTER_2,
    .pinParkDefs                    = parkInfo,
    .numPins                        = 24
};
```

Power mode 例程如果选择测试深度睡眠功能，会利用信号量获取 xSemaphoreTake()函数设置的深度睡眠的时间。在调用 xSemaphoreTake()函数前需要先调用 Power_enablePolicy()函数，设置 enablePolicy 标志位为 True。xSemaphoreTake()函数的第二个参数是在没有获取到信号量时等待的最大时间。在 Power mode 例程里 xSemaphoreTake()函数第二个参数设置为 SLEEP_TIME_TICKS，也就是睡眠时间。这个例程只有一个任务，所以在 xSemaphoreTake()函数调用后的等待时候会进入空闲任务，操作系统会调用 vPortSuppressTicksAndSleep()让 AWRL6432 进入深度睡眠。当 SLEEP_TIME_TICKS 到达之后，AWRL6432 会被时钟唤醒。

power_modes.c(C:\ti\MMWAVE_L_SDK_05_05_02_00\examples\drivers\power\power_modes)

```
        if (LPDS_USECASE == sel)
        {
            DebugP_log("Entering LPDS.... \n\r");
            Power_enablePolicy();
            xSemaphoreTake(gPowerSem, SLEEP_TIME_TICKS);
            Power_disablePolicy();
```

多任务例程

MMWAVE_L_SDK 里的 motion_and_presence_detection 例程 (\examples\mmw_demo) 使能了低功耗模式，也需要在 syscfg 里配置 power 相关选项。但是低功耗的具体实现上和 power mode 例程有所区别。因为 motion_and_presence_detection 例程有多个任务，就需要通过调用 Power_idleFunc()来实现。Power_enablePolic()仍需要在 Power_idleFunc()使用前预先调用。例程里仍然有 gPowerSem 信号量，但这个信号量主要是用于和串口输出任务做同步使用，设置的等待时间是最大值。

Power_managerment.c

(C:\ti\MMWAVE_L_SDK_05_05_02_00\examples\mmw_demo\motion_and_presence_detection\source\power_management)

```
        if(gMmwMssMCB.lowPowerMode == LOW_PWR_MODE_ENABLE)
        {
             /* Radar Power Management Framework driver call for getting to Low Power State */
            Power_idleFunc(slpTimeus);
            Power_disablePolicy();
```

低功耗模式调试小技巧

由于进入低功耗后仿真器会断开，这对于调试低功耗模式代码带来了不便。当低功耗代码出问题时，需要先判断是否时进入低功耗是否正常唤醒。可以在唤醒函数最开设置一个有条件循环，运行代码仿真器断开后，可以再连上仿真器，看是否停在这个循环里。如果是，那说明 AWRL6432 已经唤醒，排查唤醒后代码。如果不是，那就是没有正常唤醒，需要查看进入休眠的相关代码。下面是一个添加有条件循环的例子。

Power_managerment.c

```
void loop_forever()
{
    volatile uint32_t loop = 1;
    while(loop);
}
…
void power_LPDSresumehook(void)
{
    static uint8_t ledState = 0;
loop_forever();
…
```

参考资料:

1. Low Power Support - FreeRTOS™
2. (+) AWRL6432: Hard fault with power test example - Sensors forum - Sensors - TI E2E support forums
3. MMWAVE-L-SDK Software development kit (SDK) | TI.com

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月