JN516x 低功耗睡眠唤醒流程

(Shaozhong.Liang)

NXP JN516x 无线微控制器系列包括超低功耗、高性能 MCU，带有符合 IEEE802.15.4 标准的 2.4GHz 无线电收发器。集成了超低功耗睡眠模式，深度睡眠模式功耗仅 100nA，开启定时器唤醒功耗仅 0.64μA。

NXP 提供了运行在 JN516x 芯片上满足各种物联网应用的无线组网协议栈，例如 ZigBee Pro Home Automation、ZigBee Pro Light Link、ZigBee 3.0 等协议栈，极大地简化了使用 JN516x 开发 ZigBee 应用的难度，加快了开发进度。

Operation	Current (mA)	Notes
Active Processing		
CPU active (Radio off)	1.7 + 0.205/ MHz	CPU can run at 32, 16, 8, 4, 2 or 1 MHz. GPIOs enabled. When in CPU Doze mode, the current related to CPU speed is not consumed.
Radio transmitting	15.3	CPU in Doze mode
Radio receiving	17.0	CPU in Doze mode
The following current figures should be added to those above if the feature is being used		
ADC	0.555	Temperature sensor and battery measurements require ADC
Comparator	0.073/0.0008	Normal/Low power
UART	0.06	For each UART
Timer	0.021	For each timer
2-wire Serial Interface	0.046	
Sleep Modes		
Sleep with IO wake-up	0.00012	Waiting on IO event
Sleep with IO and 32-kHz RC oscillator timer wake-up	0.00064	
32-kHz Crystal Oscillator	0.0014	As an alternative sleep timer
The following current figures should be added to those above if the feature is being used		
RAM Retention	0.0009	
Comparator (low power mode)	0.0008	Reduced response time
Deep Sleep Mode		
Deep Sleep	0.00010	Waiting on chip RESET or IO event

Table 1: JN516x Current Consumption Summary

为了实现低功耗睡眠唤醒功能，JN516x 硬件集成两个专用于唤醒 CPU 的 41 位递减定时器，由 32kHz 时钟驱动，在唤醒定时器启动后从设定值开始递减，到计数值为零时触发中断。唤醒定时器默认使用片内 32kHz RC 振荡器，也可以选择外部 32kHz 晶振作为时钟源。片内 32kHz RC 时钟可能有高达 18%的计时误差，不过这个误差可以利用 32MHz 的外设时钟校正从而大大减小。

ZigBee 协议栈 SDK 的 Power Manager (PWRM)模块实现低功耗睡眠、唤醒功能，提供了下面的一系列 PWRM API 函数，并有完整的低功耗睡眠 End-Device 设备参考设计源代码。

Core Functions

The PWRM core functions are listed below, along with their page references:

PWRM_vInit
PWRM_eStartActivity
PWRM_eFinishActivity
PWRM_u16GetActivityCount
PWRM_eScheduleActivity
PWRM_vManagePower

Callback Set-up Functions

The PWRM callback set-up functions are used to introduce user-defined callback functions that must be defined when using the Power Manager.

vAppMain
PWRM_vRegisterPreSleepCallback
PWRM_vRegisterWakeupCallback
vAppRegisterPWRMCallbacks
PWRM_vWakeInterruptCallback

下图是 JN516x 低功耗睡眠设备的冷启动和睡眠唤醒启动流程图。

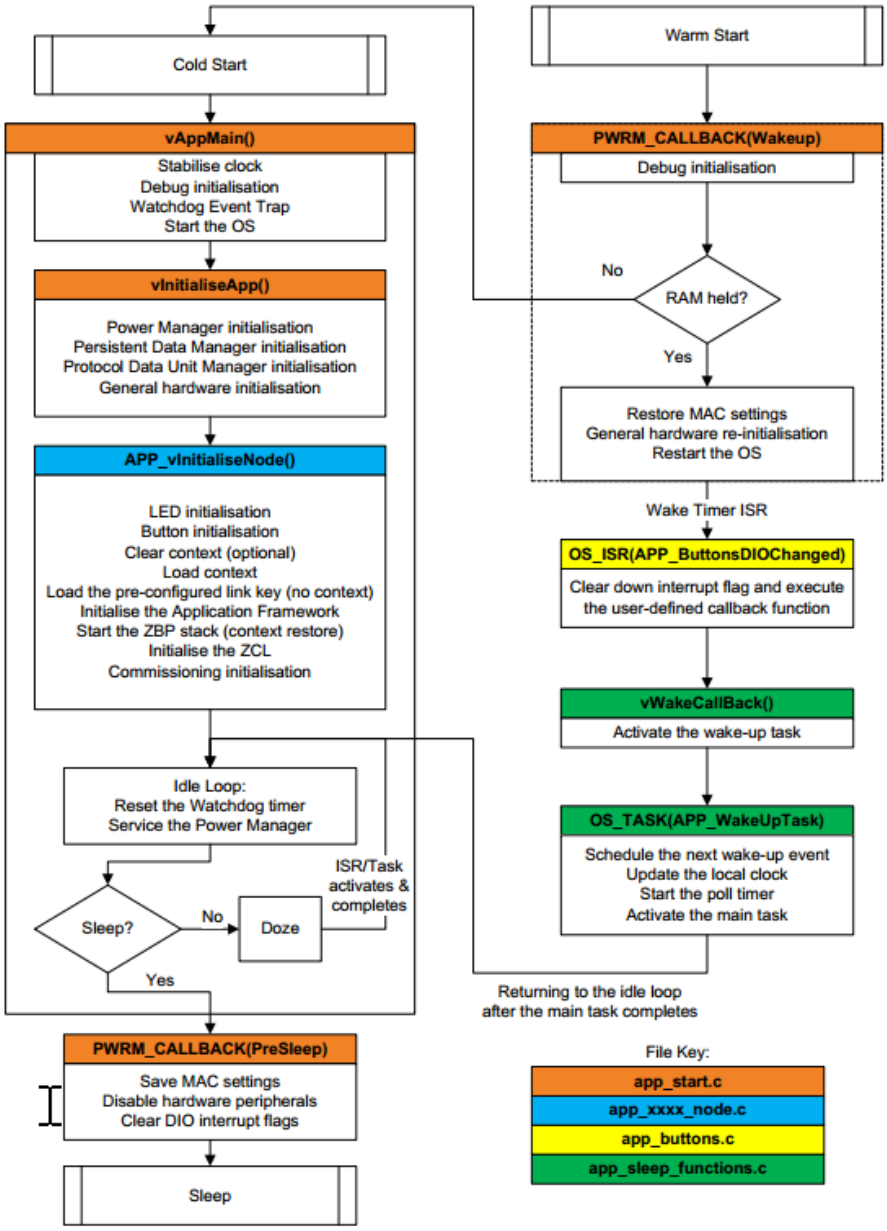

Figure 3: Typical Start-up Flow

本文以 NXP JN5169 ZigBee 参考设计 JN-AN-1189-ZigBee-HA-Demo 中的 LightSensor 应用代码为例，分析 JN516x 低功耗睡眠唤醒设备的代码结构。为了便于理解代码的主要脉络， 相关函数只保留关键的代码片段。 请参照完整的源代码了解更多细节。

JN516x 复位启动后将运行 vAppMain()，这是整个程序的初始化入口。系统初始化函数调用顺序：

vAppMain

→ vInitialiseApp

→ APP_vInitialiseNode

```
PRIVATE void vInitialiseApp(void)
{
    /*
     * Initialise JenOS modules. Initialise Power Manager even on non-sleeping nodes
     * as it allows the device to doze when in the idle task.
     * Parameter options: E_AHI_SLEEP_OSCON_RAMON or E_AHI_SLEEP_DEEP or ...
     */
    PWRM_vInit(E_AHI_SLEEP_OSCON_RAMON);
```

调用 PWRM_vInit 初始化 Power Manager (PWRM)模块。一般情况下，使能 OSC Wake Timer 并保持 RAM 数据。这个函数将会调用 **vAppRegisterPWRMCallbacks** 注册回调函数，注册二个应用层睡眠前、唤醒后的处理函数。

```
void vAppRegisterPWRMCallbacks(void)
{
    PWRM_vRegisterPreSleepCallback(PreSleep);    //进入休眠前的处理函数，例如关闭串口，使能唤醒源
    PWRM_vRegisterWakeupCallback(Wakeup);        //CPU唤醒后的处理函数，例如重新打开串口，重新使能OS调度
}
```

PreSleep 和 Wakeup 是二个非常重要的回调函数。开发人员往往需要根据产品应用的需求，修改这二个函数的具体实现。例如在 CPU 进入睡眠之前调用 PreSleep 函数，设置 GPIO 状态，设定唤醒源(唤醒 GPIO 管脚，触发边沿)，关闭芯片的外设(节省功耗)。

应用代码调用 `PWRM_eScheduleActivity(pwrm_tsWakeTimerEvent *psWake,uint32 u32Ticks,void (*prCallbackfn)(void))`函数设置 Wake Event。可以设置多个 Wake Event (建议一般情况设置一个唤醒 Wake Event 事件)，多个 Wake Event 按照唤醒时间由小到大组成一个链表，其中 u32TickDelta 保存各个 Wake Event 之间的差值，例如 Wake Event1 设置 5 秒后唤醒，Wake Event2 设置 10 秒后唤醒，则二个 Wake Event 的 u32TickDelta 都是 5。u32TickDelta 的数组值由 SDK 根据链表计算获得差值，用户不能修改这个数据结构的值。

函数 `PWRM_eScheduleActivity` 是实现睡眠的一个重要函数。PWRM_eScheduleActivit 将会启动 Wake Timer1 唤醒定时器，并设置 WakeTimer1 为激活状态。

用户程序尝试进入休眠的调用流程

vAttemptToSleep

→vScheduleSleep

→PWRM_eScheduleActivity

→Start Wake Timer1

→Set WakeTimer1 flag status

```
PUBLIC void vAttemptToSleep(void)
{
    DBG_vPrintf(TRACE_SLEEP_HANDLER, "\nAPP Sleep Handler: Activity Count = %d", PWRM_u16GetActivityCount());
    DBG_vPrintf(TRACE_SLEEP_HANDLER, "\nAPP Sleep Handler: Task Timers = %d", u8NumberOfTimersTaskTimers());

    /* Have we any software timers running that should prevent us from sleep*/
    if (0 == u8NumberOfTimersTaskTimers())
    {
        /* Stop any background timers that are non sleep preventing*/
        vStopNonSleepPreventingTimers();

        /* Check if Wake timer 0 is running.*/
        if (u8AHI_WakeTimerStatus() & E_AHI_WAKE_TIMER_MASK_0 || !bKeepalive)
        {
            vScheduleSleep(FALSE);
        }
        else
        {
            vScheduleSleep(TRUE);
        }
    }
}


PRIVATE void vScheduleSleep(bool_t bDeepSleep)
{
    ..................
    PWRM_teStatus eStatus = PWRM_eScheduleActivity(&sWake,
            (MAXIMUM_TIME_TO_SLEEP - u32GetNumberOfZCLTicksSinceReport())*APP_TICKS_PER_SECOND , vWakeCallBack);
    DBG_vPrintf(TRUE, "\nAPP Sleep Handler: Osc on, Sleep Status = %d \n", eStatus);
}
```

函数 PWRM_vManagePower 在 CPU 空闲时运行，不停地检查系统是否满足进入休眠条件。

```
PUBLIC void vAppMain(void)
{
    ..................................
    while (TRUE)
    {
        vAHI_WatchdogRestart();
        PWRM_vManagePower();
    }
}
```

下面是 PWRM_vManagePower 函数的 pseudocode，展示了 JN516x 如何进入休眠。如果满足没有更高优先级的任务，并且没有其他活跃的定时器的条件时，将会调用 PreSleep，然后 CPU 进入休眠状态。

```
void PWRM_vManagePower(void)
{
    if( 已经调用 PWRM_eScheduleActivity 启动 Wake Timer1 计数器   &&
        s_u16ActivityCounter 是否为 0   &&
        s_bTickTimerActive  是否为 0)
    {
        /* 没有激活的任务，没有激活的软件定时器，满足休眠条件,准备休眠 */
        PreSleep()    /* 调用之前注册的函数，完成休眠前准备工作 */

        vAHI_Sleep();   /* CPU 进入休眠 */

        Wakeup();      /* 唤醒后，调用之前注册的回调函数 */

        PWRM_vWakeInterruptCallback();   /* 调用 Wake Event 事件的回调函数 */
     }else {
        vAHI_CpuDoze();   /* CPU 进入 doze 状态，降低 CPU 功耗 */
     }
}
```

函数的调用关系

vAppMain

→ PWRM_vManagePower

→ PreSleep

→ vAHI_Sleep

→ Wakeup

→ PWRM_vWakeInterruptCallback

```
PWRM_CALLBACK(PreSleep)
{
    /*Put off the LEDs Indicators to All off if the device is sleeping while the
     * stae is not running,In running the state LED will be used to retain the
     * occupied - unoccupied state.*/
    vGenericLEDSetOutput(2, 0);

    /* Set up wake up input */
    vSetUpWakeUpConditions();

    /* Disable UART */
    vAHI_UartDisable(E_AHI_UART_0);
}
```

与 PreSleep 相对应，当 CPU 唤醒后将调用 Wakeup 回调函数，恢复 CPU 正常运行环境。例如，重新初始化 UART 串口，使能 GPIO 状态，重新使能 JenOS 的任务调度。请特别注意，由于 PreSleep 已经关闭了 UART 串口输出，在 Wakeup 重新初始化串口之前，不能调用 DBG_vPrintf 串口调试输出，否则 JN-516x 会出现 crash dump 错误。

```
PWRM_CALLBACK(Wakeup)
{
    #if JENNIC_CHIP_FAMILY == JN516x
        /* Wait until FALSE i.e. on XTAL  - otherwise uart data will be at wrong speed */
        while (bAHI_GetClkSource() == TRUE);
        /* Now we are running on the XTAL, optimise the flash memory wait states */
        vAHI_OptimiseWaitStates();
        #ifndef PDM_EEPROM
            PDM_vWarmInitHW();
        #endif
    #endif

    /* Don't use RTS/CTS pins on UART0 as they are used for buttons */
    vAHI_UartSetRTSCTS(E_AHI_UART_0, FALSE);
    DBG_vUartInit(DBG_E_UART_0, DBG_E_UART_BAUD_RATE_115200);

    bRGB_LED_Enable();
    bRGB_LED_SetLevel(LED_MAX_LEVEL,LED_MIN_LEVEL,LED_MIN_LEVEL);
    bRGB_LED_Off();
    bRGB_LED_SetGroupLevel(LED_LEVEL);

    DBG_vPrintf(TRACE_START, "\nAPP: Woken up (CB)");
    DBG_vPrintf(TRACE_START, "\nAPP: Warm Waking powerStatus = 0x%x", u8AHI_PowerStatus());
    ..................

    /* Restart the OS */
    DBG_vPrintf(TRUE, "\nAPP Start: APP: Restarting OS, Tick Timer = %d", u32AHI_TickTimerRead());
    OS_vRestart();
}
```

当 CPU 的 Wake Timer0 或者 Wake Timer1 计数为 0 时，CPU 将会被唤醒，并触发 `vISR_SystemController` 硬件中断 ISR 服务函数。在 PWRM_vWakeInterruptCallback 中将会调用 `PWRM_eScheduleActivity` 注册的 Wake Event 回调函数。如果 Wake Event 链表中仍有其他未超时的事件，则继续启动 Wake Timer1 定时器，准备休眠。

Hardware Wake Timer0/1

→ OS_ISR(vISR_SystemController)

→ PWRM_vWakeInterruptCallback

→Wake Event callback function

→Start Wake Timer1

JN-516x 的 GPIO 和 Wake Timer0/1 硬件中断服务函数都是 vISR_SystemController。在中断 ISR 函数中，建议不要进行耗时的操作。由于在休眠前 UART 已经被关闭，在 vISR_SystemController 中不要进行 UART 操作。

```
OS_ISR(vISR_SystemController)
{
    /* clear pending DIO changed bits by reading register */
    uint8 u8WakeInt = u8AHI_WakeTimerFiredStatus();
    u32DioInterrupts|=u32AHI_DioInterruptStatus();

    if (u8WakeInt & E_AHI_WAKE_TIMER_MASK_1)
    {
        APP_tsEvent sButtonEvent;

        /* wake timer interrupt got us here */
        PWRM_vWakeInterruptCallback();

        /* Post a message to the stack so we aren't handling events
         * in interrupt context
         */

        sButtonEvent.eType = APP_E_EVENT_PERIODIC_REPORT;
        OS_ePostMessage(APP_msgEvents, &sButtonEvent);
    }
    ..............
}
```

系统中断重入问题:

当频繁产生 GPIO 中断时，vISR_SystemController 中断服务函数将会抢占休眠函数 PreSleep 的执行(中断 ISR 的优先级别高于函数代码)。在 OS_ISR(vISR_SystemController)中将会 Disable GPIO interrupt 中断，并影响 PWRM 休眠管理模块。另外，PreSleep 函数 Disable UART 串口，因此，不能在 vISR_SystemController 中断服务函数中调用串口打印输出，否则会 watchdog 复位。

解决方法：在 PreSleep 函数中禁止系统中断，从而避免中断抢占的问题。请参照下面代码，调用 MICRO_DISABLE_AND_SAVE_INTERRUPTS 在休眠前禁止系统中断。

```
#include "MicroSpecific.h"
PWRM_CALLBACK(PreSleep)
{
    uint32 u32Ints;
    /* Critical Section Cannot be interrupted as it changes parameters */
    /* that main ISR uses.  Ensures a clean stop without re-enter risk */
      MICRO_DISABLE_AND_SAVE_INTERRUPTS(u32Ints);
     ……
    /* Disable UART */
    vAHI_UartDisable(E_AHI_UART_0);
}

OS_ISR(vISR_SystemController)
{
    teInterruptType eInterruptType = E_INTERRUPT_UNKNOWN;

    /* clear pending DIO changed bits by reading register */
    uint8 u8WakeInt = u8AHI_WakeTimerFiredStatus();
    uint32 u32IOStatus=u32AHI_DioInterruptStatus();

      //DBG_vPrintf(TRACE_APP_BUTTON, "In vISR_SystemController\n");   //不能调用 UART 输出

     if( u32IOStatus & APP_BUTTONS_DIO_MASK )
    {
      /* disable edge detection until scan complete */
      vAHI_DioInterruptEnable(0, APP_BUTTONS_DIO_MASK);
      OS_eStartSWTimer(APP_ButtonsScanTimer, APP_TIME_MS(10), NULL);
      eInterruptType = E_INTERRUPT_BUTTON;
      }
      ……
}
```

在实际开发阶段，需要特别注意 PWRM_eScheduleActivity 函数调用时机，否则将无法使 JN516x 进入低功耗休眠状态：

A. 在调用 PWRM_eScheduleActivity 函数之前，应该保证没有激活的软件定时器。否则将不会调用 PreSleep 函数，无法进入睡眠状态。当调用 `PWRM_u16GetActivityCount` 函数返回当前活跃的个数为 0 时，表示系统允许进入低功耗睡眠。

B. 不能重复调度同一个 Wake Event，否则 PWRM_eScheduleActivity 将返回 PWRM_E_TIMER_RUNNING 错误。即使 PWRM_eScheduleActivity 返回成功后，JN516x 并不会马上休眠。只有当 PWRM_vManagePower 判断满足休眠条件后，JN516x 才会进入休眠。

C. 当 Wake Timer 启动后，将在 32kHz 时钟驱动下自动递减计数器。需要通过特殊步骤才能修改 Wake Timer 计数器。例如，应用设定 10s 的 Wake Timer，当在第 9 秒时由于 GPIO 唤醒了 CPU 后重新进入休眠，则 1s 后 Wake Timer 将会唤醒 CPU。如果需要在当前时刻点起重新设定 Wake Timer1 唤醒时间，可以调用 vAHI_WakeTimerStartLarge 函数重新设置计数器。

D. 即使 PWRM_eScheduleActivity 函数返回成功后，ZigBee 协议栈内部某些软件定时器(例如 MAC 重传定时器、APS 重传定时器等)将会阻止 PWRM_vManagePower 使 CPU 进入休眠状态。在 vISR_SystemController 中断服务函数不要遗漏执行 PWRM_vWakeInterruptCallback 函数，否则 Wake Event 事件不会被被触发，下一次

PWRM_eScheduleActivity 函数将返回 PWRM_E_TIMER_RUNNING 错误，导致系统无法进入休眠。

E. 如果在执行某些重要代码过程中不希望 CPU 进入休眠，可以使用 PWRM_eStartActivity 禁止休眠，执行完毕后需要使用 PWRM_eFinishActivity 允许休眠，这个二个函数改变 s_u16ActivityCounter 计数值。

F. ZigBee SDK 默认使用 Wake Timer1 作为唤醒定时器。用户可以使用 Wake Timer0 作为睡眠计时器。在设备进入睡眠前启动 Wake Timer0 并设置计数值，当唤醒时通过 u64AHI_WakeTimerReadLarge 函数读取 Wake Timer0 的当前计数值，即可获得睡眠时间。Wake Timer 的详细使用方法请参考<<JN-UG-3087 N516x Integrated Peripherals API User Guide>>文档。