第五章 Z-STACK 的低功耗电源管理

第一节 Z-STACK 协议栈下的电源管理机制

5.1.1 电源管理的概念

电源管理是指设备（一般是采用电池供电的终端节点）通过软件设置在设备空闲时自动进入多种睡眠模式，从而延长电池的使用寿命。

Zigbee 的低功耗特性即通过恰当的电源管理实现，即需要尽量减少短暂无线电通信之间的功耗。通常有以下两种途径：

◆ 禁用外设以降低设备功耗。

◆ 空闲期间进入睡眠模式。 Z-STACK 提供了两种睡眠模式：轻度定时器休眠和深度休眠。轻度定时器睡眠在系统需要周期性被唤醒去执行预定时间延迟的任务时被使用。深睡眠通常在没有明确时间延迟，需要外部中断刺激（如按一个按钮）唤醒设备才开始任务的情况下使用。 轻度定时器睡眠可以降低功耗到几毫安，而深度睡眠可以减少到几个微安。

睡眠的终端设备的例子产品包括：传感器——它被周期性的唤醒汇报它们采集的传感器数据；还有遥控设备——它被用户按键唤醒并发送数据。在这些类型的设备共同特点是它们可以在大部分时间里处于睡眠模式，最大限度地减少系统功耗。

5.1.2 Z-Stack 协议栈下的系统电源管理机制

Z-SATCK 协议栈下的系统电源管理通过 OSAL 实现。OSAL 主循环监控每个任务完成后的系统状态。如果系统当前没有任务有预定的事件发生，同时电源管理功能启用，系统将判断是否进入睡眠模式。系统进入睡眠模式必须同时满足如下条件：

◆ - 睡眠 POWER_SAVING 编译选项启用。

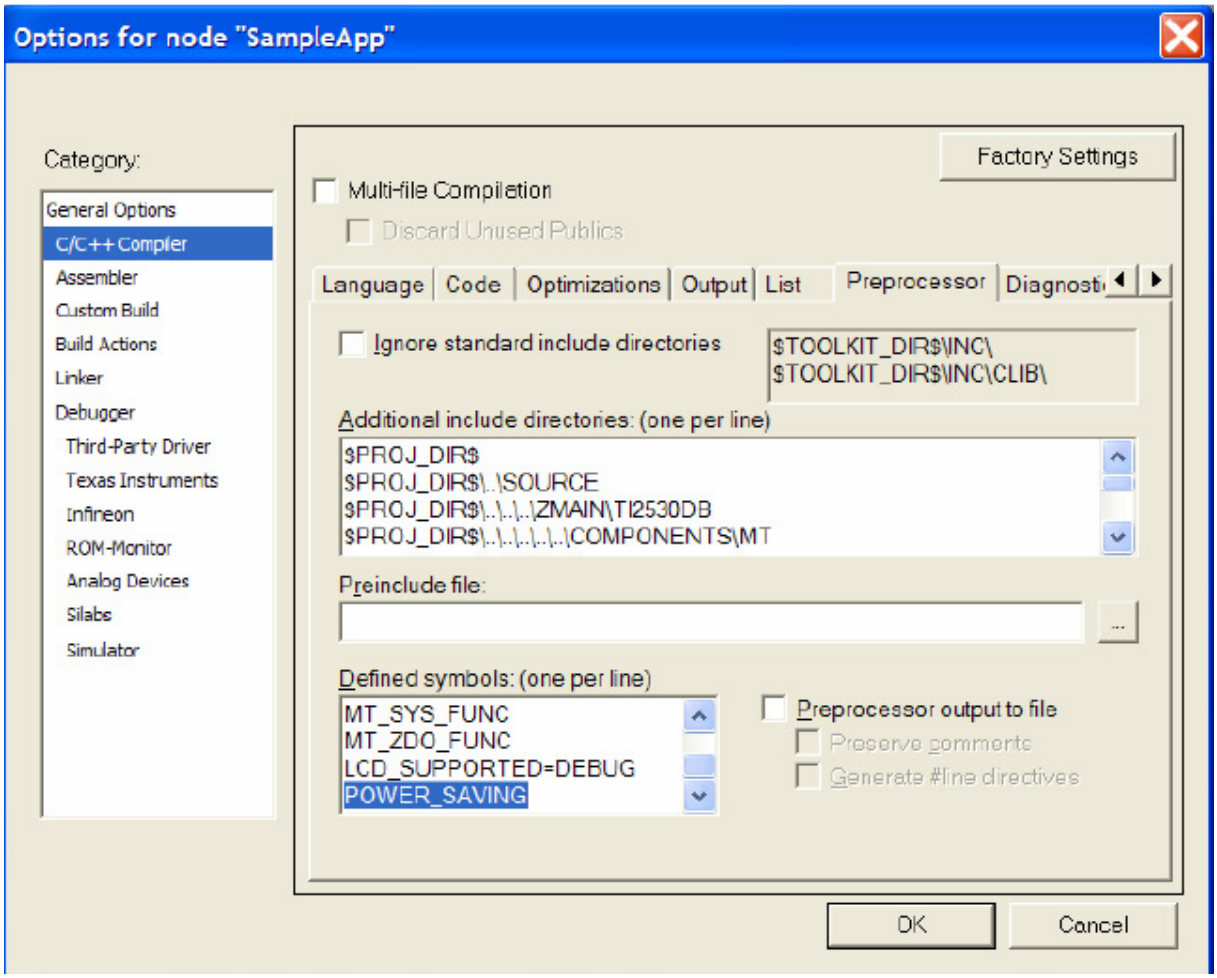
◆ - ZDO 节点状态描述显示“RX 在系统空闲时关闭”：设置 RFD_RCVC_ALWAYS_ON 在 f8wConfig.cfg 为 FALSE。

◆ - 所有的 Z-Stack 的任务“同意”睡眠。

◆ - 所有的 Z-Stack 的任务没有预定的事件安排。

◆ – MAC 没有预定的事件安排。

Z-Stack 中终端设备的工程项目在默认情况下没有电源管理功能。要启用此功能，在建立工程时需要将 POWER_SAVING 编译选项设置为使能。如下图所示，该编译选项位于 *C/C++ Compiler* 下 *Preprocessor* 里的 *Defined symbols:(one per line)*下：



为了将系统功耗降到最低，终端设备在进入睡眠模式之前需要尽可能多地把不必要的电路关闭。这包括外围设备、无线接收器和发射器以及 MCU 本身的一些重要部分。为了避免在睡觉模式下丢失信息，终端设备的父设备需要保存子节点的信息直至终端设备查询到这些信息为止（终端设备功能设置中需将 *CAPINFO_RCVR_ON_IDLE* 设置项设置关闭）。在 Z-Stack 的工程中，设备功能的默认设置在 *ZDO_Config_Node_Descriptor* 结构中描述，保存在 ZDConfig.c 文件中。终端设备的默认设置只设置了 *CAPINFO_DEVICETYPE_RFD 项*，表明它是电池供电并可在设备空闲时关闭其无线接收器。

ZDConfig.c

```
125
126   // MAC Capabilities
127   if ( ZSTACK_ROUTER_BUILD )
128   {
129     ZDO_Config_Node_Descriptor.CapabilityFlags
130       = (CAPINFO_DEVICETYPE_FFD | CAPINFO_POWER_AC | CAPINFO_RCVR_ON_IDLE);
131
132     if ( ZG_BUILD_COORDINATOR_TYPE && ZG_DEVICE_COORDINATOR_TYPE )
133       ZDO_Config_Node_Descriptor.CapabilityFlags |= CAPINFO_ALTPANCOORD;
134   }
135   else if ( ZSTACK_END_DEVICE_BUILD )
136   {
137     ZDO_Config_Node_Descriptor.CapabilityFlags = (CAPINFO_DEVICETYPE_RFD
138   #if ( RFD_RCVC_ALWAYS_ON == TRUE)
139       | CAPINFO_RCVR_ON_IDLE
140   #endif
141     );
142   }
```

在 OSAL 主循环结束时，系统会作一个是否进入睡眠模式的判断。如果所有的 Z-Stack 的任务被检查且都没有任何事件需要处理，那么编译选项 POWER_SAVING 设置与否将决定是否调用函数 *osal_pwrmgr_powerconserve（）*：

OSAL.c

```
945       }
946   #if defined( POWER_SAVING )
947       else  // Complete pass through all task events with no activity?
948       {
949         osal_pwrmgr_powerconserve();  // Put the processor/system into sleep
950       }
951   #endif
952     }
953   }
```

同时，在系统决定进入睡眠模式之前，需要再做两个额外的检查：首先，检查 *pwrmgr_device* 变量是否被设置为电池设备。这项检查在设备加入网络时执行，见例程中的 ZDApp.c。第二，对 *pwrmgr_task_state* 变量进行检查，以确保没有任务需要“put a hold”即要求进行工作状态保持。这种机制使每个 Z-Stack 的任务在进行关键操作时禁止进入睡眠模式。

当这些条件都符合，系统睡眠时间将取决于 OSAL 系统定时器的下次溢出时间，如果这个下次溢出时间大于零且小于 *MIN_SLEEP_TIME*（变量值在 hal_sleep.c 定义，主要用于防止睡眠时间过短造成系统硬件冲击），系统将选择进入 TIMER_SLEEP 模式。在 TIMER_SLEEP 模式下， OSAL 系统定时计数器将在每次溢出时产生一个定时器中断来唤醒系统。如果没有任何 Z-Stack 的事件或睡眠定时器计划，系统将选择进入深度睡眠模式 SLEEP_DEEP，此时睡眠定时器的溢出时间将为零（即关闭），可以最大限度的降低设备功耗：

OSAL_PwrMgr.c

```
151   // Should we even look into power conservation
152   if ( pwrmgr_attribute.pwrmgr_device != PWRMGR_ALWAYS_ON )
153   {
154     // Are all tasks in agreement to conserve
155     if ( pwrmgr_attribute.pwrmgr_task_state == 0 )
156     {
157       // Hold off interrupts.
158       HAL_ENTER_CRITICAL_SECTION( intState );
159
160       // Get next time-out
161       next = osal_next_timeout();
162
163       // Re-enable interrupts.
164       HAL_EXIT_CRITICAL_SECTION( intState );
165
166       // Put the processor into sleep mode
167       OSAL_SET_CPU_INTO_SLEEP( next );
168     }
169   }
```

宏 *OSAL_SET_CPU_INTO_SLEEP（）*被认为睡眠进程的开始。对于 CC2530，此宏调用 *halSleep（）*函数执行关停 MAC，关闭外设，进入 MCU 睡眠模式。睡眠后唤醒 MCU，打开外设，最后重新启动 MAC 等一系列有序的操作。由于 Z-Stack 的 OSAL 系统循环独立运行于 MAC 层，Z-Stack 不知道的 MAC 层的运行状态。需要通过对函数 MAC_PwrOffReq（）的调用去请求一个关闭 MAC。应当指出，如果系统设置为无线接收器在空闲时不关闭，MAC 也就不会因为睡觉而关闭，因为这种设置相当于将设备配置成了无需进入睡眠模式的设备。

在 CC2530 上，深度睡眠模式只有通过五相开关 U1 或者轻触按键 S1 产生的外部中断进行唤醒或者直接通过 MCU 复位按键 S5（针对飞比科技仿真扩展板 FB2530EB）。这种模式主要用于遥控器之类的设备，可以通过外部中断例如按下按钮来进行设备唤醒。轻度睡眠 TIMER_SLEEP 模式可以被任何中断事件终止，包括外部事件以及定时器事件。在轻度睡眠 TIMER_SLEEP 模式下，如果外部中断唤醒了微控制器，而系统 OSAL 定时计数器没有溢出，Z-Stack 的 OSAL 定时系统将新的唤醒时间调整为预定唤醒时间的已计时部分。

5.1.3 睡眠定时器

在 CC2530 上，轻度睡眠 TIMER_SLEEP 模式依靠一个 32.768 kHz 晶振驱动的 24 位硬件定时/计数器（SLEEP_TIMER，即睡眠定时器）工作。电源管理器通过睡眠定时器进行计时，溢出产生定时

器中断来唤醒 MCU。睡眠定时器有一个 24 位计数器和一个 24 位比较器。24 位的 CC2530 的睡眠定时器能够保持在睡眠期间长达 512s（2^{24}/32768=512），因此最长的睡眠时间是 510s（四舍五入）。

5.1.4 软件应用设计注意事项

在 Z-Stack 的示例应用程序中，终端设备被预设为禁止电源管理和使能自动查询消息。系统支持三种不同延时参数的查询选项设置。当启用电源管理（POWER_SAVING 编译选项使能）时，任何一种查询选项设置，睡眠模式都将会受到影响。需要特别指出的是，预定延时的查询会阻止系统进入深度睡眠，因此功耗降低有限。三种不同延时参数的查询选项设置包括：

数据请求查询(Data Request Polling) – 周期性发送数据请求到父设备查询排队消息。修改查询延时可以通过改变 *zgPOLLRATE* 变量值或通过立即调用函数 *NLME_SetPollRate（）*。如果之前被禁用，调用此函数就将开始查询，以 1 为时间间隔调用该函数将立即查询一次。

排队数据查询(Queued Data Polling) – 当收到一个数据提示后查询父设备中的排队消息。修改查询延时可以通过调用函数 *NLME_SetQueuedPollRate（）* 或存储在 *zgQueuedPollRate* 变量中，此功能还允许快速“卸载排队消息”，而不管 *Data Request Poll Rate*。

响应数据查询(Response Data Polling) – 收到一个数据确认后查询父设备中的响应消息，修改查询延时可以通过调用函数 *NLME_SetResponsePollRate（）* 或直接存储在 zgResponsePollRate 变量。此功能允许迅速“卸载响应消息”，如 *APS Acknowledgements*，而不管 *Data Request Poll Rate*。

上述三种查询速率的默认值在源文件 nwk_globals.c 中定义和初始化，定义终端设备将自动查询消息。默认状态下，POWER_SAVING 编译选项使能，系统将被限制在轻度睡眠模式（TIMER_SLEEP）。为了通过创建一个深度睡眠（DEEP SLEEP）设备来降低系统功耗，应通过设置 *zgPOLLRATE* 为 0 来禁止重复查询。适当的设置这三个查询速率可以得到多种查询策略。例如，对于从不需要接收消息的设备来说，一旦加入网络，应设置这三个查询速率为零。如果采用 APS Acknowledge，查询应该被使能直到它收到应答消息（ACK）。在某些系统应用中，设置不同的查询速率对降低系统功耗，优化网络性能是很有帮助的。

另外一个查询操作就是按键查询。默认情况下，按键查询在 100 毫秒被使能。要禁用的按键查询，可以将变量 *OnboardKeyIntEnable 设置为 HAL_KEY_INTERRUPT_ENABLE*。

OnBoard.c

```
121
122     /* Initialize Key stuff */
123     OnboardKeyIntEnable = HAL_KEY_INTERRUPT_ENABLE;
124     HalKeyConfig( OnboardKeyIntEnable, OnBoard_KeyCallback);
125   }
126 }
```

5.1.5 硬件应用设计注意事项

未使用的 I/O 口应该有一个确定的电平，一种简易的办法就是将这些引脚通过寄存器配置成带上拉电阻的通用输入口，当然也可以配置成为通用的输出口。但切记不要将这些引脚接到电源或者地上以降低系统功耗。

在 CC2530 上，如果未使用的 I/O 口悬空，那么这些引脚将不断产生软件不能消除的中断标志。

第二节 OSAL 下的低功耗电源管理实现（CC2530）

主要涉及以下几个文件：

OSAL_PwrMgr.h OSAL 电源管理的 API 头文件

OSAL_PwrMgr.C OSAL 电源管理的 API C 文件

hal_sleep.c 底层的电源管理文件

电源管理结构体

```
typedef struct
{
uint16 pwrmgr_task_state;       //任务状态
uint16 pwrmgr_next_timeout;     //下一次超时
uint16 accumulated_sleep_time; //睡眠时间
uint8 pwrmgr_device;            //电源管理设备属性，有 PWRMGR_ALWAYS_ON 和
                                //PWRMGR_BATTERY 两种
} pwrmgr_attribute_t;
```

```
#define  PWRMGR_ALWAYS_ON   0
#define  PWRMGR_BATTERY     1
```

选择 *PWRMGR_ALWAYS_ON* 的话将不会进入睡眠模式

选择 *PWRMGR_BATTERY* 将允许 HAL 管理 CPU 进入 SLEEP LITE 或者 SLEEP DEEP 状态。

```
#define  PWRMGR_CONSERVE  0
#define  PWRMGR_HOLD      1
```

低功耗标志，主要用于 osal_pwrmgr_task_state()这个函数中，用于标志每一任务是否需要低功耗。

```
extern  pwrmgr_attribute_t pwrmgr_attribute;
```

定义一个电源管理的全局变量。

函数

```
/*************************************************************************
* @fn osal_pwrmgr_init
*
* @brief 初始化电源管理函数，这个函数在 OSAL.C 里面的 osal_init_system()调用，也就 * 是在 OSAL 系统初始
```

```
化的时候将电源管理模式调成了不会进入睡眠模式的状态。
*
* @param none.
*
* @return none.
*/
void osal_pwrmgr_init( void )
{
pwrmgr_attribute.pwrmgr_device = PWRMGR_ALWAYS_ON; //默认没有睡眠模式
pwrmgr_attribute.pwrmgr_task_state = 0;             //清零
}
/*************************************************************************
* @fn osal_pwrmgr_device
*
* @brief 设置电源管理设备属性。
*
* @param pwrmgr_device -选择 PWRMGR_ALWAYS_ON 的话将不会进入睡眠模式，选择
* PWRMGR_BATTERY 将允许 HAL 管理 CPU 进入 SLEEP LITE 或者 SLEEP DEEP 状态。
*
* @return none
void osal_pwrmgr_device( uint8 pwrmgr_device )
{
pwrmgr_attribute.pwrmgr_device = pwrmgr_device;
}
/*************************************************************************
* @fn osal_pwrmgr_task_state
*
* @brief 这个函数可以被每一个任务调用，用于设置这个任务是否支持低功耗运行，如
* 果每一个任务不支持低功耗将无法进入低功耗模式运行。
*
* @param task_id - 任务 ID
* state - 任务是否需要低功耗
*
* @return SUCCESS if task complete
*/
uint8 osal_pwrmgr_task_state( uint8 task_id, uint8 state )
{
if ( task_id >= tasksCnt )
return ( INVALID_TASK );
if ( state == PWRMGR_CONSERVE )
{
// 清零
pwrmgr_attribute.pwrmgr_task_state &= ~(1 << task_id );
```

```
}
else
{
//置位
pwrmgr_attribute.pwrmgr_task_state |= (1 << task_id);
}
return ( SUCCESS );
}
#if defined( POWER_SAVING )
/*************************************************************************
* @fn osal_pwrmgr_powerconserve
*
*@brief 这个函数在 OSAL 循环中如果没有任何事件需要执行的话将被调用，将设备进入
* 睡眠模式，不可以在其他地方调用该函数。
* 需要打开 POWER_SAVING 的宏定义。
*
* @param none.
*
* @return none.
*/
void osal_pwrmgr_powerconserve( void )
{
uint16 next;
halIntState_t intState;
// 首先检查是否支持低功耗
if ( pwrmgr_attribute.pwrmgr_device != PWRMGR_ALWAYS_ON )
{
//是否所有任务支持低功耗
if ( pwrmgr_attribute.pwrmgr_task_state == 0 )
{
//关中断
HAL_ENTER_CRITICAL_SECTION( intState );
//查询软件定时器链表得到最近一次溢出时间
next = osal_next_timeout();
//开中断
HAL_EXIT_CRITICAL_SECTION( intState );
//将系统进入睡眠模式
OSAL_SET_CPU_INTO_SLEEP( next );
}
}
}
#endif /* POWER_SAVING */
```

加红部分是一个宏定义，在 OnBoard.h 里面定义的。

```
#define  OSAL_SET_CPU_INTO_SLEEP(timeout)  halSleep(timeout);
```

halSleep(timeout)是在 hal_sleep.c 中定义的。

这里面涉及的就是关于 CC2530 的电源管理寄存器的一些操作。具体可以看代码。

```
/* HAL 电源管理模式是设置电源管理状态的，默认状态是 HAL_SLEEP_OFF。其余设置均会关*闭系统时钟停止 CPU。
* HAL_SLEEP_TIMER 模式可以被睡眠定时器中断和 IO 中断以及复位唤起。
* HAL_SLEEP_DEEP 模式可以被 IO 中断以及复位唤起。
*/
#define HAL_SLEEP_OFF CC2530_PM0
#define HAL_SLEEP_TIMER CC2530_PM2
#define HAL_SLEEP_DEEP CC2530_PM3
#define CC2530_PM0 0
#define CC2530_PM1 1
#define CC2530_PM2 2
#define CC2530_PM3 3
#define MAX_SLEEP_TIME  510000
//  最大睡眠时间是 510000ms。
```

总结

其实可以在这个函数中可以看到在 OSAL 中是使用的睡眠定时器来控制睡眠时间的，在系统初始化的是将电源控制结构体中的 pwrmgr_device 设备属性设置为 PWRMGR_ALWAYS_ON，这样默认就不进入休眠状态。必须在应用层里面调用 void osal_pwrmgr_device(uint8 pwrmgr_device)这个 OSAL 的 API 来设置使得 OSAL 能够进入休眠状态。

在用户任务中需要用的这样一个 API——uint8 osal_pwrmgr_task_state(uint8 task_id, uint8 state)来设置这个任务是否支持休眠，如果有一个任务不支持休眠的话，整个系统就将不会进入休眠模式。这个在 void osal_pwrmgr_powerconserve(void)中有相关的查询。

在 OSAL 的主循环中 void osal_start_system(void)调用了 osal_pwrmgr_powerconserve 这个函数。

```
void osal_start_system( void )
{
#if !defined ( ZBIT ) && !defined ( UBIT )
for(;;) // Forever Loop
#endif
{
uint8 idx = 0;
```

```
osalTimeUpdate();
Hal_ProcessPoll(); // This replaces MT_SerialPoll() and osal_check_timer().
do {
if (tasksEvents[idx]) // Task is highest priority that is ready.
{
break;
}
} while (++idx < tasksCnt);
if (idx < tasksCnt)
{
uint16 events;
halIntState_t intState;
HAL_ENTER_CRITICAL_SECTION(intState);
events = tasksEvents[idx];
tasksEvents[idx] = 0; // Clear the Events for this task.
HAL_EXIT_CRITICAL_SECTION(intState);
events = (tasksArr[idx])( idx, events ); //最关键的一句话，如图一中，运行对应的任务
HAL_ENTER_CRITICAL_SECTION(intState);
tasksEvents[idx] |= events; // Add back unprocessed events to the current task.
HAL_EXIT_CRITICAL_SECTION(intState);
}
#if defined( POWER_SAVING )
else // Complete pass through all task events with no activity?
{
osal_pwrmgr_powerconserve(); // Put the processor/system into sleep
}
#endif
}
}
```

表示 OSAL 系统在检查完所有的任务事件之后发现没有事件需要处理，这样在 POWER_SAVING 宏定义打开的情况下将调用 osal_pwrmgr_powerconserve();函数，在这函数中将会根据选择系统进入睡眠模式。

退出睡眠模式

当出现 IO 中断或者复位时候会退出睡眠模式，或者在睡眠定时器中断时候也将会退出睡眠模式。如果是 IO 中断或者睡眠定时器中断退出之后将回到进入睡眠的地方继续向下执行，复位退出的话进入程序的初部分执行。