Application Note:
Power Management
For The CC2530

Document Number: SWRA248

Texas Instruments, Inc.
San Diego, California USA

Version	Description	Date
1.0	Initial release	01/09/2009
1.1	Updated for ZStack-2.2.0 release	04/02/2009
1.2	Updated Section 5 for 32-bit timers	06/14/2013

Table of Contents

1. Purpose

This document describes power management concepts for the Texas Instruments CC2530 for use by an application built upon the Texas Instruments Z-Stack™ ZigBee protocol stack. Power management is typically employed by battery powered devices to extend battery life through use of various sleep modes during periods of inactivity.

2. Definitions

The following terms are used in this document:

MAC – *M*edia *A*ccess *C*ontrol software that implements the IEEE 802.15.4 communication functions.

MCU – *M*icro *Controller U*nit – an 8051 processor embedded in CC2530 SoC on the CC2530EB board.

OSAL – *O*perating *S*ystem *A*bstraction *L*ayer – the platform independent task handler provided with Z-Stack.

Sleep – An MCU mode of operation in which certain functions are disabled in order to reduce power consumption. The CC2530 provides three different sleep modes, two of which are used by Z-Stack.

End-Device – A ZigBee device that joins a network without routing capabilities and normally turns off its receiver when idle. This requires its parent to hold messages until the End-Device polls for its messages.

3. What Is Power Management?

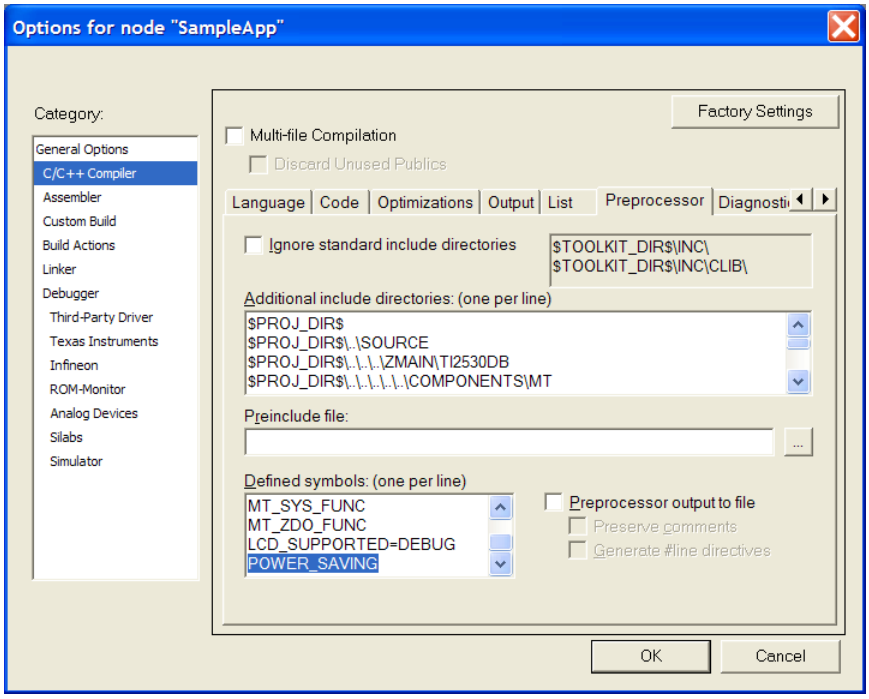
Power management is used by battery powered End-Devices to minimize power consumption between brief periods of radio communications. Normally, an End-Device disables power consuming peripherals and enters a sleep mode during idle periods. Z-Stack provides two sleep modes, designated *TIMER* sleep and *DEEP* sleep. *TIMER* sleep is used when the system needs to wake up to perform an activity related to a scheduled time delay. *DEEP* sleep is used when no future activity is scheduled, requiring external stimulus (such as a button press) to wake up the device. *TIMER* sleep generally reduces power consumption to a few milliamps, while *DEEP* sleep reduces it to a few microamps. Examples of sleeping End-Devices include sensors that wake up periodically to report their readings and remote control devices that wake up to send a message when a user presses a button. The common characteristic of these types of devices is that they spend most of their time in a sleep mode, minimizing consumption of power.

4. How Does Z-Stack Control Sleep Modes?

Power management is used by battery powered End-Devices to minimize power consumption between brief periods of scheduled activity (TIMER sleep) or during long periods of inactivity (DEEP sleep). System activity is monitored in the OSAL main control loop after each task finishes its processing. If no task has an event scheduled, and power management capability is enabled, the system will decide whether to sleep. All of the following conditions must be met in order for the device to enter a sleep mode:

- Sleep enabled by the **POWER_SAVING** compile option
- ZDO node descriptor indicates "RX is off when idle". This is done by setting **RFD_RCVC_ALWAYS_ON** to FALSE in *f8wConfig.cfg*.
- All Z-Stack tasks "agree" to permit power savings
- Z-Stack tasks have no scheduled activity
- The MAC has no scheduled activity

End-Device projects in the Z-Stack package are configured, by default, without power management. To enable this feature, the **POWER_SAVING** compile option must be specified when the project is built. As shown below, this option is placed in the ***Defined symbols*** box under the ***Preprocessor*** tab of the ***C/C++ Compiler*** options:



In order to reduce power consumption to minimum levels, an End-Device needs to turn off as much electronic circuitry as possible before entering a sleep mode. This includes peripheral devices, radio receiver and transmitter, and significant portions of the MCU itself. To avoid loss of messages while sleeping, the End-Device's parent needs to hold its messages until the End-Device polls for them. The parent device "knows" that the End-Device will poll for messages when the capabilities in the End-Device's association request has CAPINFO_RCVR_ON_IDLE turned off. In Z-Stack projects, default settings for device capabilities are specified in the *ZDO_Config_Node_Descriptor* structure, located in the ZDConfig.c file. The default End-Device only specifies CAPINFO_DEVICETYPE_RFD, indicating that it is battery-powered and will turn off its receiver when idle:

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   }
143
```

The decision whether to attempt power conservation is made at the end of the main OSAL loop. If all Z-Stack tasks were checked and none had any processing to do, the compile option **POWER_SAVING** determines whether the *osal_pwrmgr_powerconserve()* function gets called:

```
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   }
```

At this point, two more checks are performed before attempting to enter a sleep mode. First, the *pwrmgr_device* variable is checked to be set to be a battery device. This setting is established after the device joins the network – see ZDApp.c for examples. Second, the *pwrmgr_task_state* variable is checked to see that no task has "put a hold" on power conservation. This mechanism allows each Z-Stack task to disable sleep during critical operations. When both of these conditions are met, the desired sleep time is determined by the next expiration time of the OSAL timers. If the next expiration time is greater than zero and less than *MIN_SLEEP_TIME*, **TIMER_SLEEP** mode is selected. In this mode, the system timer is adjusted to provide a "wake up" interrupt for the timer event that is due to expire first. The *MIN_SLEEP_TIME*, defined in *hal_sleep.c*, is used to prevent very short sleep "thrashing". The **DEEP_SLEEP** mode is selected when there are no Z-Stack events or timers scheduled; therefore the next expiration is zero, allowing for maximum power savings:

```
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   }
```

The *OSAL_SET_CPU_INTO_SLEEP* macro is called to begin the sleep process. For the CC2530, this macro calls the *halSleep()* function that performs the sequence of shutting down the MAC, turning off peripherals, entering the MCU sleep mode, waking up the MCU after sleep, turning on peripherals, and finally restarting the MAC. Since the Z-Stack OSAL loop runs independently of the MAC scheduler, Z-Stack does not know the processing state of the MAC. The call to *MAC_PwrOffReq()* will request a MAC shutdown. It should be noted that the MAC will not shut down for sleep when the receiver is enabled when idle, therefore preventing the device from sleeping.

On the CC2530, DEEP sleep mode only terminates by an external interrupt or from an MCU reset. So on the SmartRF05EB, the two GPIO's that will trigger this external interrupt are the joystick press down or the S1. This mode would be used by remote control type devices which sleep until externally stimulated, such as by button press. TIMER sleep mode is terminated by any interrupt event, including the external events, as well as timer events. If an external interrupt wakes up the MCU while in the TIMER sleep mode (timer not expired), the Z-Stack timing system adjusts for the elapsed fraction of the scheduled wake-up time delay.

5. Sleep Timer Considerations

TIMER sleep mode on the CC2530 is implemented in a 24-bit hardware timer (SLEEP_TIMER) driven by 32.768 kHz crystal clock source. Power manager uses the sleep timer to keep track of elapsed time and to wake up the MCU after the timer expires. The sleep timer has a 24-bit counter and a 24-bit comparator. CC2530 sleep timer is capable of keeping track network time during sleep for up to 512 seconds (2^{24} / 32768). The longest sleep time is therefore 510 seconds (rounded). OSAL provides 32-bit event timers, based on a 1 ms HW timer tick, allowing timer events up to 4294967 seconds (2^{32} / 1000). The sleep module automatically handles OSAL sleep times which are longer than the maximum CC2530 sleep timer.

The SLEEP_TIMER compare value is set by the following equation where *timeout* is the next OSAL/MAC timer expiration in 320 usec unit and *ticks* is the current SLEEP_TIMER count:

*ticks += (timeout * 671) / 64*

The ratio of 32 kHz ticks to 320 usec ticks is 32768/3125 = 10.48576. This is nearly 671/64 = 10.484375. When the SLEEP_TIMER counts up to the compare value, an interrupt is generated and wakes up the MCU. After waking up from sleep, the elapsed time in milliseconds is calculated as ticks*1000/32768 or:

*ticks * 125 / 4096*

6. Application Considerations

End-Devices in the Z-Stack sample applications are setup initially with power management disabled and automatic polling for messages enabled. Three different polling options are supported, each controlled by a time delay parameter. When power management is enabled (**POWER_SAVING** compile option), along with any of the polling options, sleep modes will be affected. Specifically, time delays scheduled for polling preclude DEEP sleep, therefore limiting power conservation. The three time-delay polling options include:

- Data Request Polling – periodically sends a data request to the parent device to poll for queued messages. The time interval between messages can be altered by storing the desired time in *zgPollRate* or set immediately by calling the function *NLME_SetPollRate()*. Calling this function will start polling if it has been previously disabled. Calling with a time interval of 1 will poll immediately, one time.
- Queued Data Polling – polls the parent device for queued messages after receipt of a data indication. The time delay can be changed by calling the function *NLME_SetQueuedPollRate()* or by storing it in *zgQueuedPollRate* This feature permits rapid "unloading" of queued messages, irregardless of the Data Request Poll rate.
- Response Data Polling – polls the parent device for response messages after receipt of a data confirmation. The time delay can be changed by calling the function *NLME_SetResponseRate()* or storing it directly in *zgResponsePollRate*. This feature permits rapid "unloading" of response messages, such as APS Acknowledgements, irregardless of the Data Request Poll rate.

The default settings for these polling rates are defined and initialized in the **nwk_globals.c** source file, specifying that the End-Device will automatically poll for messages. If **POWER_SAVING** is enabled with these default polling rates, power conservation will be limited to TIMER sleep mode. To minimize power consumption by creating a DEEP sleeping device, repetitive polling should be disabled by setting the *zgPollRate* to zero. Various polling strategies can be achieved by setting the values of these three polling rates appropriately. For example, for a device that never needs to receive messages once it has joined the network should set all three polling rates to zero. If APS acknowledge is utilized, then polling needs to be enabled after each message transmission at least until the ACK is received. In some systems, it may be useful to vary the polling rate, depending on specific application activity.
Another polling activity is the key polling. The key polling is enabled at 100 millisecond rate by default. To disable key polling, define the ISR_KEYINTERRUPT compile option.

7. Hardware Considerations

Unused I/O pins should have a defined level and not be left floating. One way to do this is to leave the pin unconnected and configure the pin as a general purpose I/O input with pull-up resistor. Alternatively the pin can be configured as a general purpose I/O output. In both cases the pin should not be connected directly to VDD or GND in order to avoid excessive power consumption.

If the unused I/O pins are left floating in CC2530, the interrupt flag may not be cleared by software and constant interrupt from the unused pin may occur.

Power consumption on the CC2530EM+SmartRF05 combo board during sleep cycles is determined by the SoC sleep modes and various external components located on the board. To measure the power consumption, remove the "Power source" jumper (lower left corner in picture below). Connect an ampere meter between pins 2 and 3 if external power is used; connect an ampere meter between pins 1 and 2 if battery power is used. The measurement is the total current consumption of the CC2530 SoC and IO peripherals. Jumpers P13 "V_IO", P10 "P4 IO", P1 "P1 USB", and "RS232 Enable" switch, allow enable/disable of individual peripheral and IO pins. Refer to SmartRF05 schematics for details.