Chipcon Products from Texas Instruments

Application Note: Power Management For The CC2430DB

Document Number: F8W-2006-0019

Texas Instruments, Inc.
San Diego, California USA
(619) 497-3800

Version	Description	Date
1.0	Initial release.	10/30/2006
1.1	Updated for CC2430 Rev D Workarounds	12/01/2006
1.2	Updated for Release 1.4.0 to 1.4.1 migration, Rev D to Rev E migration, and port pin termination.	08/28/2007

Table of Contents

1. Purpose

This document describes power management concepts for the Chipcon CC2430DB 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 CC2430 SOC on the CC2430DB 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 CC2430 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?

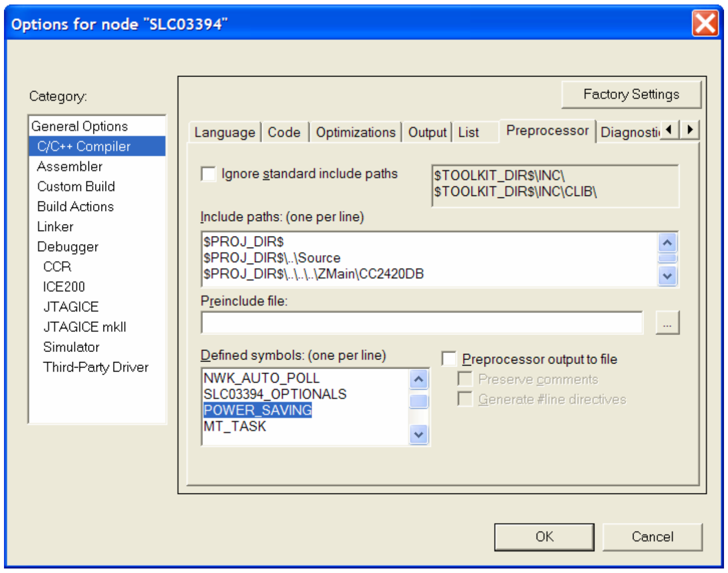
Power management is used by battery powered End-Devices to minimize power consumption between brief periods of radio communications. Normally, an End-Device disables power consuming peripherals and enters a sleep mode during idle periods. Z-Stack provides two sleep modes, designated *LITE* sleep and *DEEP* sleep. *LITE* 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. *LITE* 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 (LITE 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:



Ignore standard include paths

```
$TOOLKIT_DIR$\INC\
$TOOLKIT_DIR$\INC\CLIB\
```

Include paths: (one per line)

```
$PROJ_DIR$
$PROJ_DIR$\..\Source
$PROJ_DIR$\..\..\..\ZMain\CC2420DB
```

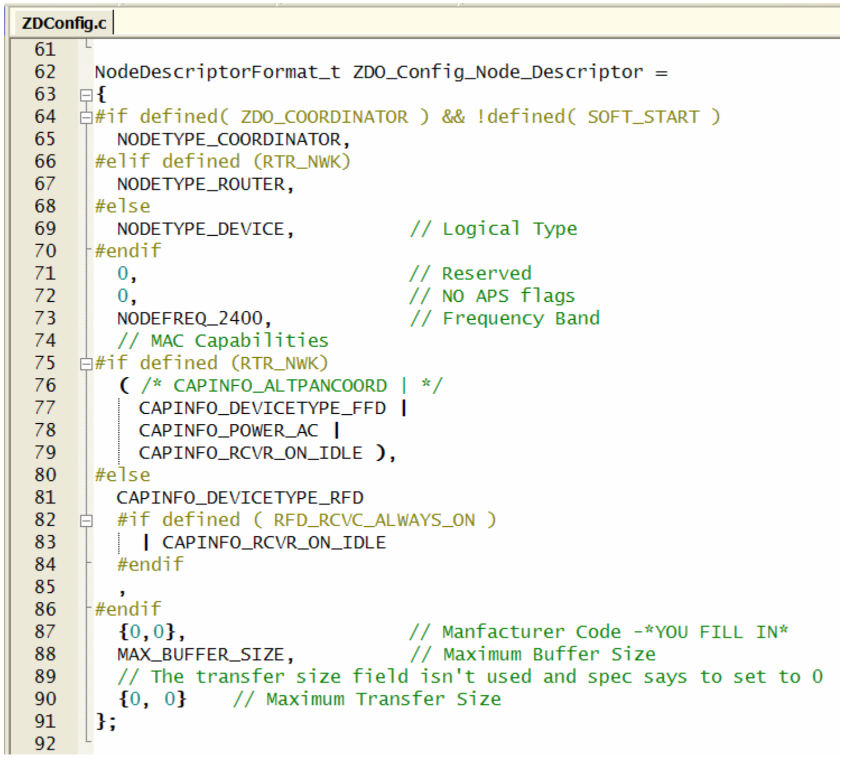
Preinclude file:

Defined symbols: (one per line)

```
NWK_AUTO_POLL
SLC03394_OPTIONALS
POWER_SAVING
MT_TASK
```

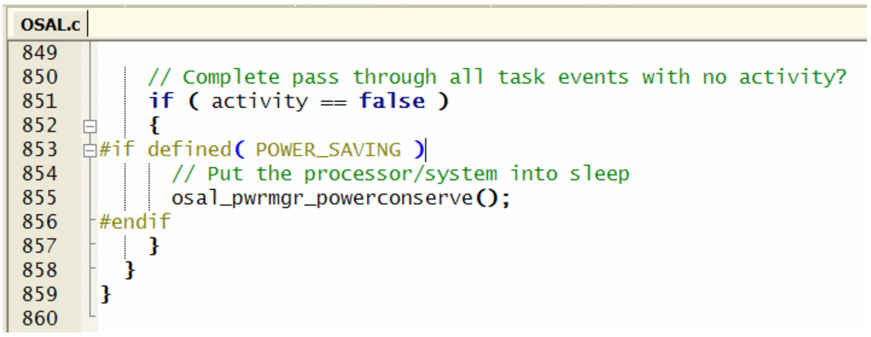
Preprocessor output to file
Preserve comments
Generate #line directives

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:



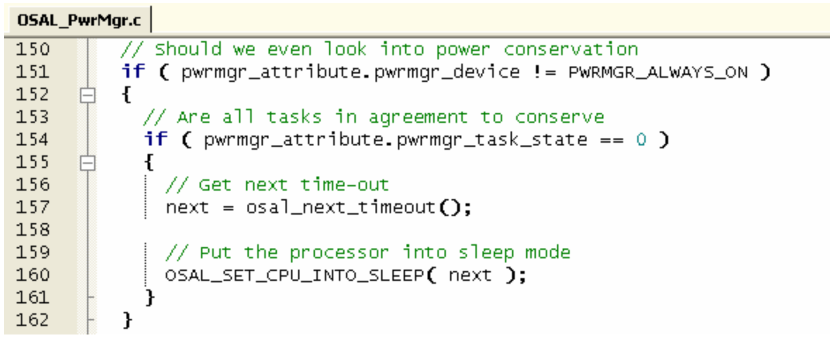
```

NodeDescriptorFormat_t ZDO_Config_Node_Descriptor =
{
#if defined( ZDO_COORDINATOR ) && !defined( SOFT_START )
  NODETYPE_COORDINATOR,
#elif defined (RTR_NWK)
  NODETYPE_ROUTER,
#else
  NODETYPE_DEVICE,          // Logical Type
#endif
  0,                        // Reserved
  0,                        // NO APS flags
  NODEFREQ_2400,            // Frequency Band
  // MAC Capabilities
#if defined (RTR_NWK)
  ( /* CAPINFO_ALTPANCOORD | */
    CAPINFO_DEVICETYPE_FFD |
    CAPINFO_POWER_AC |
    CAPINFO_RCVR_ON_IDLE ),
#else
  CAPINFO_DEVICETYPE_RFD
  #if defined ( RFD_RCVC_ALWAYS_ON )
    | CAPINFO_RCVR_ON_IDLE
  #endif
  ,
#endif
  {0,0},                    // Manfacturer Code -*YOU FILL IN*
  MAX_BUFFER_SIZE,          // Maximum Buffer Size
  // The transfer size field isn't used and spec says to set to 0
  {0, 0}     // Maximum Transfer Size
};

```

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 *activity* variable will be **false**. As shown below, the compile option **POWER_SAVING** determines whether the *osal_pwrmgr_powerconserve()* function gets called:


OSAL.c

```
    // Complete pass through all task events with no activity?
    if ( activity == false )
    {
#if defined( POWER_SAVING )
      // Put the processor/system into sleep
      osal_pwrmgr_powerconserve();
#endif
    }
  }
}
```

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*, **SLEEP_LITE** 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* is to prevent very short sleep threshing defined in *hal_sleep.c*. The **SLEEP_DEEP** 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

```
  // Should we even look into power conservation
  if ( pwrmgr_attribute.pwrmgr_device != PWRMGR_ALWAYS_ON )
  {
    // Are all tasks in agreement to conserve
    if ( pwrmgr_attribute.pwrmgr_task_state == 0 )
    {
      // Get next time-out
      next = osal_next_timeout();

      // Put the processor into sleep mode
      OSAL_SET_CPU_INTO_SLEEP( next );
    }
  }
```

The *OSAL_SET_CPU_INTO_SLEEP* macro is called to begin the sleep process. For the CC2430DB, 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 CC2430DB, DEEP sleep mode terminates only by external interrupt from the joystick and S1 or from an MCU reset. This mode is used by remote control type devices which sleep until externally stimulated, such as by button press. LITE 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 LITE sleep mode (timer not expired), the Z-Stack timing system adjusts for the elapsed fraction of the scheduled wake-up time delay.

Note that the interrupt can only be generated when S1 is pressed down while joystick is toggled. S1 is located next to the green LED. Refer to the board image towards the end of this application note.

5. Sleep Timer Considerations

LITE sleep mode on the CC2430DB is implemented in a 24-bit hardware timer (SLEEP_TIMER) driven by 32.768 kHz crystal clock source. Power manager uses sleep timer to keep track of elapsed time and to wake up MCU after timer expires. The sleep timer has a 24-bit counter and a 24-bit comparator. CC2430 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). The OSAL uses a 16-bit timer structure. Therefore, the OSAL timer has a limitation of 65 seconds based on 1 ms timer tick.

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*

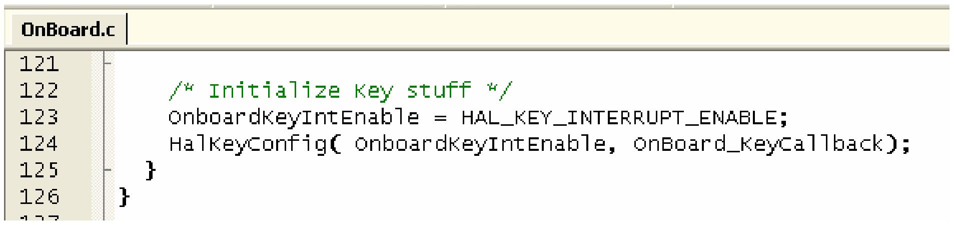
6. Application Considerations

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 *gNWK_POLL_RATE* 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 *gQUEUED_POLL_RATE* 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_SetResponsePollRate()* or storing it directly in *gRESPONSE_POLL_RATE*. This feature permits rapid "unloading" of response messages, such as APS Acknowledgements, irregardless of the Data Request Poll rate.

The default settings for **POLL_RATE**, **QUEUED_POLL_RATE**, **RESPONSE_POLL_RATE** compile options in *f8wConfig.cfg* that then setup poll rates in *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 LITE sleep mode. To minimize power consumption by creating a DEEP sleeping device, repetitive polling should be disabled by setting the *gNWK_POLL_RATE* 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.

One other polling activity is the key polling. The key polling is enabled at 100 millisecond rate by default. To disable the key polling, change *OnboardKeyIntEnable* to **HAL_KEY_INTERRUPT_ENABLE**.



OnBoard.c

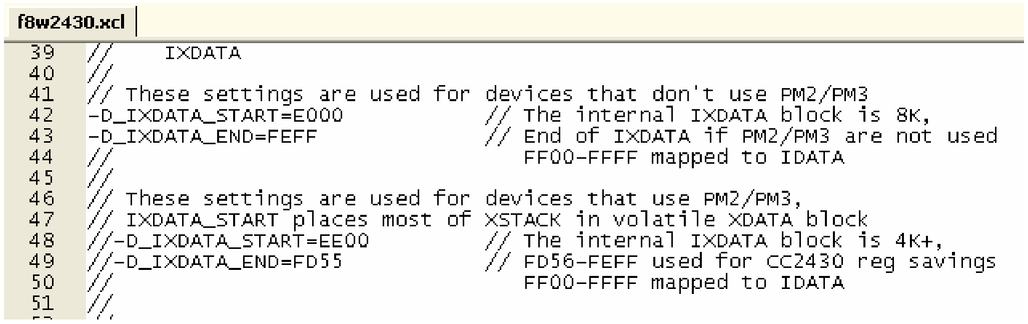
```

    /* Initialize Key stuff */
    OnboardKeyIntEnable = HAL_KEY_INTERRUPT_ENABLE;
    HalKeyConfig( OnboardKeyIntEnable, OnBoard_KeyCallback);
  }
}
```

7. Memory Considerations

DEEP sleep mode (PM2 or PM3) on the CC2430DB does not retain memory contents of the following XDATA locations:

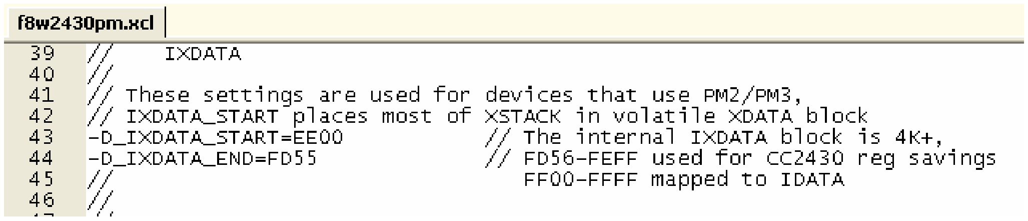
- 0xE000 – 0xEFFF (4096 bytes)
- 0xFD56 – 0xFEFF (426 bytes)

The linker configuration file, *f8w2430.xcl*, must be replaced with *f8w2430pm.xcl* for DEEP sleep devices.



f8w2430.xcl

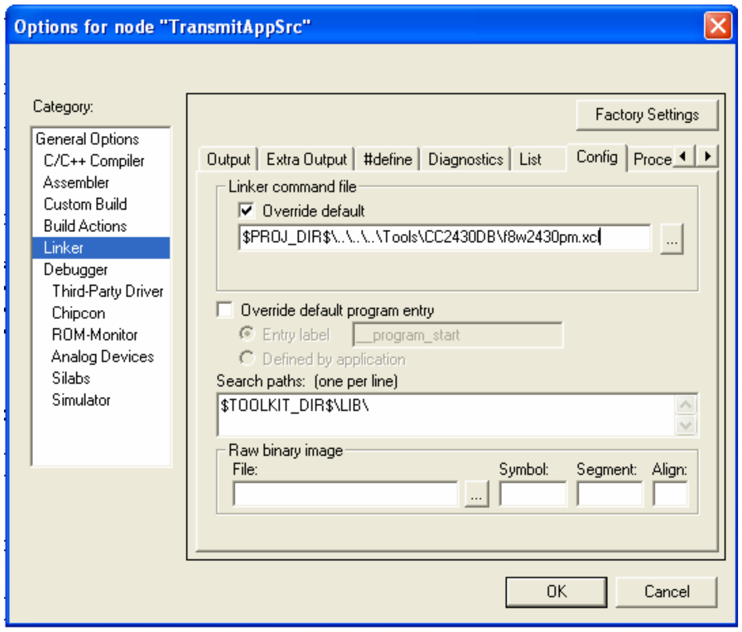
```
//    IXDATA
//
// These settings are used for devices that don't use PM2/PM3
-D_IXDATA_START=E000            // The internal IXDATA block is 8K,
-D_IXDATA_END=FEFF              // End of IXDATA if PM2/PM3 are not used
//                                 FF00-FFFF mapped to IDATA
//
// These settings are used for devices that use PM2/PM3,
// IXDATA_START places most of XSTACK in volatile XDATA block
//-D_IXDATA_START=EE00          // The internal IXDATA block is 4K+,
//-D_IXDATA_END=FD55            // FD56-FEFF used for CC2430 reg savings
//                                 FF00-FFFF mapped to IDATA
//
```



f8w2430pm.xcl

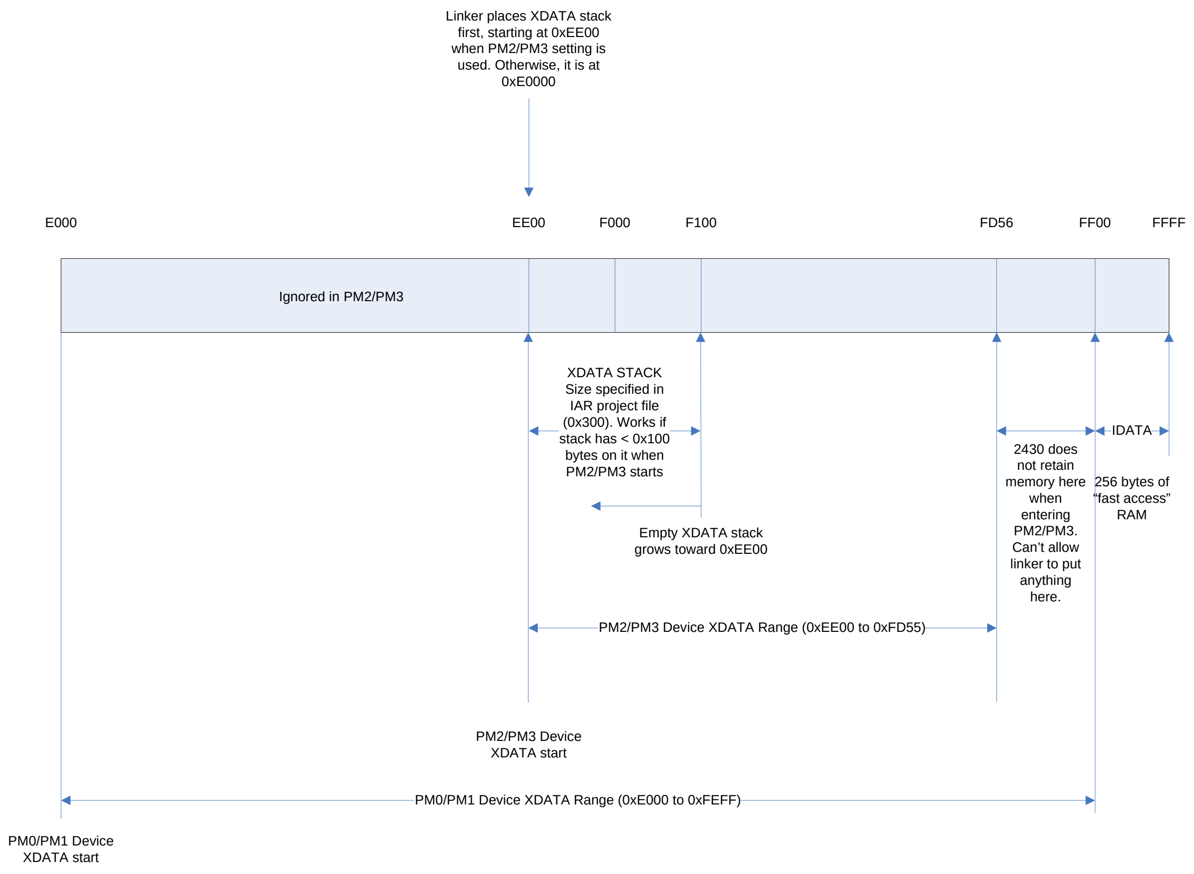
```
//    IXDATA
//
// These settings are used for devices that use PM2/PM3,
// IXDATA_START places most of XSTACK in volatile XDATA block
-D_IXDATA_START=EE00            // The internal IXDATA block is 4K+,
-D_IXDATA_END=FD55              // FD56-FEFF used for CC2430 reg savings
//                                 FF00-FFFF mapped to IDATA
//
```

As shown below, this option is placed in the ***Linker command file*** box under the ***Config*** tab of the ***Linker*** options:



Linker command file
[✓] Override default
$PROJ_DIR$\..\..\..\Tools\CC2430DB\f8w2430pm.xcl
[ ] Override default program entry
Entry label __program_start
Defined by application
Search paths: (one per line)
$TOOLKIT_DIR$\LIB\
Raw binary image
File: Symbol: Segment: Align:

Note that the RAM requirement for some applications may be too big for DEEP sleep devices. **INT_HEAP_LEN** may need to be reduced for those applications. Also note that when the device has entered DEEP sleep once, the JTAG debugger stops working. Refer to the following memory map for more details regarding deep sleep memory data retention with CC2430:


Linker places XDATA stack first, starting at 0xEE00 when PM2/PM3 setting is used. Otherwise, it is at 0xE0000
E000
EE00
F000
F100
FD56
FF00
FFFF
Ignored in PM2/PM3
XDATA STACK Size specified in IAR project file (0x300). Works if stack has < 0x100 bytes on it when PM2/PM3 starts
Empty XDATA stack grows toward 0xEE00
2430 does not retain memory here when entering PM2/PM3. Can't allow linker to put anything here.
IDATA
256 bytes of "fast access" RAM
PM2/PM3 Device XDATA Range (0xEE00 to 0xFD55)
PM2/PM3 Device XDATA start
PM0/PM1 Device XDATA Range (0xE000 to 0xFEFF)
PM0/PM1 Device XDATA start

8. CC2430 Rev D Workarounds

There are two separate issues related to CC2430 Rev D power modes:

- When entering power modes 1, 2 or 3 there is a time interval after PCON.IDLE has been set in which interrupts (pin interrupts or sleep timer interrupt) may lead to system failure.
- When waking up from power mode 2 or power mode 3, the voltage regulator is turned on again. However, the internal chip state may be corrupted if the chip is woken up in the interval when the voltage on the DCOUPL pin is between certain thresholds.

Refer to ***Application Note AN044CC2430 Revision D, Power Modes*** for problem statements, software workaround suggestions, and their limitations. Several macros and defines are implemented in *_hal_sleep.c* as software workarounds. The software workarounds can be tailored for different applications.

HAL_SLEEP_DEBUG_LED

AN044 – COMBINING POWER MODES. This is default off. When turned on, LED1 is on when entering PM1 and LED1 is off when entering PM2/PM3. This helps with debugging PM1-PM2 sequencing algorithm.

PM_MIN_SLEEP_TIME

AN044 – MINIMUM SLEEP PERIODS WITH PULL-DOWN RESISTOR. This is default to 14 milliseconds for CC2430 Rev B. The device won't enter any sleep unless the requested sleep time is greater than PM_MIN_SLEEP_TIME. This macro is to:

- avoid thrashing in-and-out of sleep with short OSAL timer (~2ms)
- define minimum safe sleep period for different CC2430 revisions

PM2_HOLDOFF_TIME

AN044 – COMBINING POWER MODES. This is default to 0. When a device is woken up be key interrupt, it should stay in PM1 for PM2_HOLDOFF_TIME expecting more key presses. When the timer is expired, the device is allowed to go back to PM2 sleep.

EXTERNAL_INTERRUPT_DELAY

AN044 – DELAYING EXTERNAL INTERRUPTS. Add code to set external interrupt line to output, drive the line to inactive level, delay for 180us (30us if P1.0 or P1.1), and set the line to input as close to PCON.IDLE = 1 as possible. This macro is using S1 key as an example. User should tailor this macro for different interrupt line(s). On CC2430EB or CC2430DB boards, the S1 key is mapped to P0.1. Thus 180us is used for delays.

PM1_ONLY

AN044 – RESTRICT USE TO PM1 ONLY. Disallow PM2/PM3 and use PM1 only. This is default to FALSE.

9. CC2430 Rev D to Rev E Migration

CC2430 Rev E changed the interrupt module designs which affect power management and interrupt clearing. Make sure that you use software release 1.4.1 or above. Three major components and their associated modules must be ported over to your code base in order to support power management in CC2430 Rev E:

- HAL Key
- HAL Sleep
- OSAL

Z-Stack software release 1.4.1 or higher includes all the required CC2430 Rev D to Rev E migration. The following sections highlight the key items.

9.1 PxIFG Clearing

When clearing PxIFG, write a zero to the corresponding bit. Read/Modify/Write can cause a problem. The peripheral can update the SFR between read and write. As a result, the update may be incomplete if Read/Modify/Write were used. For example:

- P1IFG &= 0x08; // Clear P1_3 interrupt flag

Should be changed to:

- P1IFG = ~0x08; // Clear P1_3 interrupt flag

9.2 Entering Power Mode

The following power mode macro must be used with CC2430 Rev E:

```
#define HAL_SLEEP_SET_POWER_MODE_REV_E(mode) st( SLEEP &= ~0x03;  /* clear mode bits */ \
                                        SLEEP |= mode;   /* set mode bits   */ \
                                        asm("NOP"); \
                                        asm("NOP"); \
                                        asm("NOP"); \
                                        if( SLEEP & 0x03 ) \
                                        { \
                                          PCON |= 0x01;  /* enable mode */ \
                                          asm("NOP");    /* first instruction after sleep*/ \
                                        }; )
```

9.3 Wakeup from Power Mode

For any interrupt being used to wakeup from sleep, the following macro must be is called in Rev E:

- CLEAR_SLEEP_MODE();

where CLEAR_SLEEP_MODE() is defined as SLEEP &= ~0x03;

9.4 Port 1 and Port 2 Interrupts

The sequence of clearing external interrupt must be changed for CC2430 Rev E:

```
HAL_ISR_FUNCTION( halKeyPortxIsr, PxINT_VECTOR )
{
  if( CHVER <= REV_D )
  {
    PxIF = 0;
  }
  halProcessKeyInterrupt();
  if( CHVER >= REV_E )
  {
    PxIF = 0;
    CLEAR_SLEEP_MODE();
  }
}
```

9.5 Port 0 Interrupt

If Port 0 interrupt is used, additional changes are required for CC2430 Rev E:

```
HAL_ISR_FUNCTION( halKeyPort0Isr, P0INT_VECTOR )
{
  /* P0IF is cleared by HW for CHVER < REV_E */
  halProcessKeyInterrupt();
  if( CHVER >= REV_E )
  {
    /* Make sure that we clear all enabled, but unused P0IFG bits.
     * For P0 we can only enable or disable high or low nibble, not bit by
     * bit. For P1 and P2 enabling of single bits are possible, therefore
     * will not any unused pins generate interrupts on P1 or P2.
     * We could have checked for low and high nibble in P0, but this
     * isn't necessary as long as we only clear unused pin interrupts.
     */
    P0IFG = (HAL_KEY_P0INT_LOW_USED | HAL_KEY_POINT_HIGH_USED);
    P0IF = 0;
    CLEAR_SLEEP_MODE();
  }
}
```

Where HAL_KEY_P0INT_LOW_USED and HAL_KEY_POINT_HIGH_USED must be correctly defined for the pins used to wakeup from sleep.

9.6 Critical Section in "OSAL_PwrMgr.c"

The "osal_next_timeout()" in "OSAL_PwrMgr.c" must be protected in critical section:

```
    // Hold off interrupts.
    HAL_ENTER_CRITICAL_SECTION( intState );
    // Get next time-out
    next = osal_next_timeout();
    // Re-enable interrupts.
    HAL_EXIT_CRITICAL_SECTION( intState );
```

9.7 Unused I/O Pins

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.

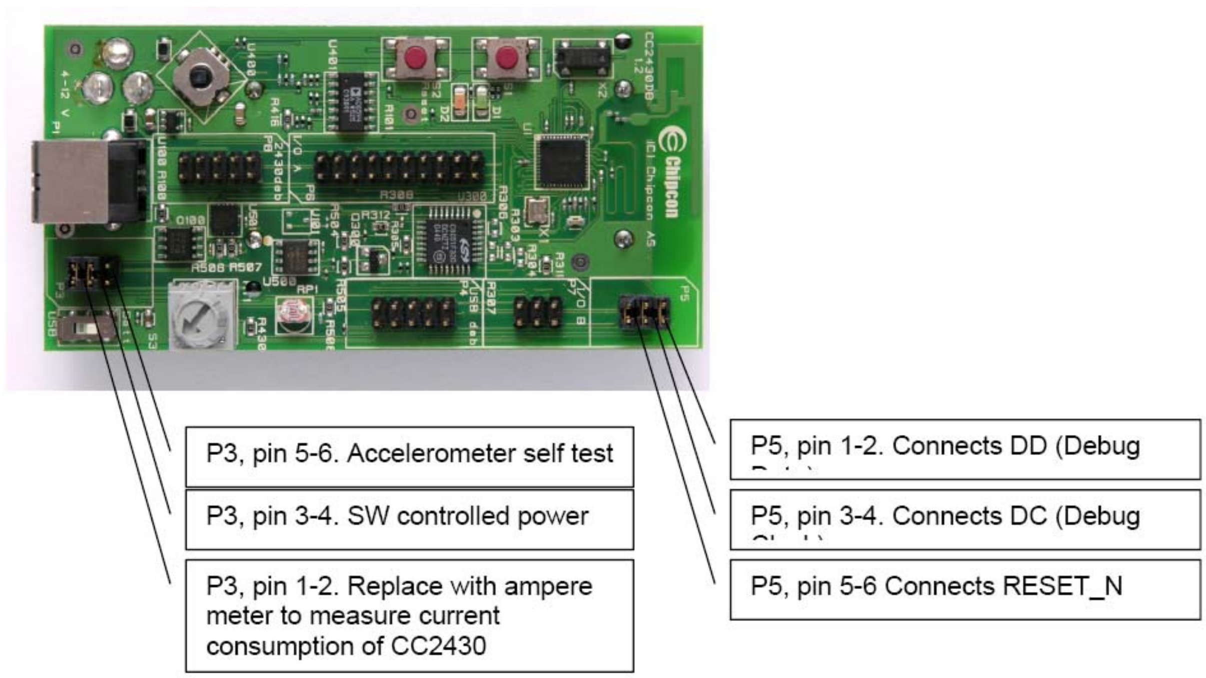
If the unused I/O pins are left floating in CC2430 Rev E, the interrupt flag may not be cleared by software and constant interrupt from the unused pin may occur.

10. Hardware Considerations

CC2430DB is designed for low power operation when running from batteries. Only the CC2430 is powered in this mode, the USB MCU is not powered.

To accurately measure the current consumption of the CC2430DB, the jumpers listed in the following table should be removed.

Header	Pin	Description	Low power setting
P3	1-2	Replace with ampere meter to measure current consumption of CC2430	Mounted, or connect ampere meter
P3	3-4	SW controlled power. Can be removed when performing current measurement to reduce leakage current	Not mounted
P5	5-6	Connects RESET_N signal between USB MCU and CC2430. Can be removed when performing current measurement to reduce leakage current	Not mounted


P3, pin 5-6. Accelerometer self test
P3, pin 3-4. SW controlled power
P3, pin 1-2. Replace with ampere meter to measure current consumption of CC2430
P5, pin 1-2. Connects DD (Debug
P5, pin 3-4. Connects DC (Debug
P5, pin 5-6 Connects RESET_N