Freescale Semiconductor
Application Note

Document Number: AN4398
Rev. 0, 7/2013

Low-Power Mode with the MC1323x Family

1 Introduction

The MC1323x family is Freescale's low cost System-on-Chip (SoC) platform for the IEEE® 802.15.4 Standard that incorporates a complete, low power, 2.4 GHz radio frequency transceiver with Tx/Rx switch, an 8-bit HCS08 CPU, and a functional set of MCU peripherals into a 48-pin LGA package. This family of products is targeted for wireless RF remote control and other cost-sensitive applications ranging from home TV and entertainment systems such as ZigBee BeeStack Consumer (RF4CE) to low cost, low power, IEEE 802.15.4 and ZigBee end nodes.

This application note describes requirements and techniques for using the MC1323x with lowest power. Following the hardware discussion, the use of Freescale's software products and tools is discussed and illustrated to show how these tools can streamline writing of a low-power-oriented application. The software example targets the BeeStack Consumer codebase (RF4CE).

Contents

freescale™ semiconductor

NOTE

You should be familiar with the MC1323x device and the following two documents for reference:

- MC1323x Data Sheet (MC1323x.pdf)
- MC1323x Reference Manual (MC1323xRM.pdf)

2 Overview of Design for Low-Power Operation

The MC1323x is inherently a low-power device either when in a normal “run” mode or when in a low-power or “sleep” mode. Both general operational modes must be considered for absolutely lowest energy usage, i.e., longest battery life.

- Run mode — Dominated by the frequency of MCU operation and minimizing the use of active functional device peripherals.
- Sleep mode — Dominated by programming lowest possible power mode and maximizing sleep time.

NOTE

- This discussion is highly concerned about minimum current usage. Although power is actually the product of voltage times current, the MC1323x has extensive onboard voltage regulation that tends to minimize current variation due to power supply (VBATT or VDD) variation. Also, battery capacity is typically rated as available mAH (milliampere-hours) due to the variance of battery voltage due to lifetime, use model, and temperature.
- There are many options available for low-power operation. These are discussed in this application note, however, it is your responsibility to properly apply any of these to their unique application. You are directed to the MC1323x Reference Manual for details of implementing the options discussed here.

2.1 Run Mode Overview

The CPU running code from Flash and RAM forms the heart of the device operation. The MCU peripherals (timers, communication devices, etc.) are enabled and used as required by the application. The transceiver (radio) provides over-the-air RF communication and defines the primary reason to use this type of device. Key issues to minimize current usage include:

- Although the 32 MHz reference oscillator must always be running, the CPU/bus clocks may be programmed for lower operational frequency. CPU run current is linearly proportional to the CPU clock rate.
- The MC1323x provides bus clock gating control to all the peripherals — The peripheral clocks can be disabled when not required to help minimize run current.
- The transceiver when active consumes the bulk of the run-time current — The radio has relatively low idle current as the analog circuitry tends to be powered only when in actual use; the transmitter

to send a data packet and the receiver to acquire a packet. Receiver “on” time tends to be higher because it must wait in anticipation of an incoming message. In contrast, the transmitter turns on and sends its data in a very deterministic manner.

2.2 Low Power Overview

The MC1323x provides a number of low-power mode options and attention to GPIO usage affect low-power current:

- All GPIO must be configured for low-power operation. GPIO need to be re-configured when entering/exiting low-power mode.
- Wakeup is possible from:
  - External IRQ pin interrupt request
  - Pushbutton operation via KBI (Keyboard Interrupt Module) interrupt request
  - Timer operation using the Real Time Counter (RTC)
  - UART receive activity via an RXD transition interrupt request
- Low voltage detect (LVD) is available while in low-power mode but uses significant current.

3 Configuring MC1323x GPIO for Best Practice

Best practice design demands that you configure all device GPIO to a known condition.

- Unused GPIO should be configured either as an input with a pullup/pulldown or an output in the low state
- Putting GPIO in a known condition prevents a GPIO from “floating” as an input with an unknown state which can result in undesired excess “shoot-through” leakage current.
- GPIO used as peripheral functions may need re-configured when entering low power and then re-initiated for peripheral use after wakeup/recovery for run time.

NOTE

When first writing software for low-power operation, you can sometimes find that the actual device current usage is in the 10’s of μA versus expected 1-2 μA or less. When this occurs, the most probable cause is some unexpected condition on an input that is causing “shoot-through” current or driving the input to a state where current is being drawn through a pullup/pulldown.

3.1 GPIO Special Pin Usage

The MC1323x has two GPIO worthy of special mention (for a detailed description, see MC1323x Reference Manual, Section 3.3):

- Signal PTA2 must be driven low upon exiting a power-on reset (POR) — This prevents entering a factory test mode.
  - Suggested configuration is an external pulldown resistor.

  - No additional initialization of this pin is required.
  - It is good practice to not use this pin for target applications if possible.
- Signal PTA7/BKGD/MS is the serial debug bus port — It is good practice to not use this pin for target applications if possible. Refer to the MC1323x Reference Manual for a detailed description of the pin usage.

Although not a GPIO, the hardware RESET pin is also a special use pin and is detailed in the MC1323x Reference Manual.

4 Minimizing Run Time Current

Greater current draw is the result of the device run time mode, the current draw is highly dependent on the frequency of operation. There are two primary strategies then for minimizing run time current:

- Disable as much logic use as possible
- Lower system clock rates where possible

In the following paragraphs, specific actions suitable to the MC1323x are discussed.

NOTE

- Freescale provides a substantial suite of codebases for the MC1323x, and in addition, a utility library to assist in configuring and using low-power modes. However, the codebases are designed to run at full CPU clock (32 MHz) and any reduction of the system CPU clock (and resulting bus clock) to facilitate lower current usage may impact functional operation or timing of the codebase/stack. It is your responsibility to prevent or correct any such changes if they detrimentally affect proper operation.
- MC1323x FLASH memory erase and write operations can only be done at full CPU/bus clock rate. If reduced CPU/bus clock operation is employed, then full clock frequency must be restored for any FLASH non-read operations.

4.1 Reducing CPU/Bus Clock Frequency

The MC1323x 8-bit HCS08 CPU defaults to a maximum available 32 MHz CPU clock, and the bus/peripheral clock is always ½ the CPU clock or 16 MHz default. One means to reduce run time current draw is to reduce the CPU/bus clocks if the system performance or peripheral timing does not require the higher clock rate.

LPRUN mode is one of the low-power modes is based on the frequency reduction:

- Enabling low-power run mode (LPRUN) — This mode can be directly entered through control of register SPMSC2 (Address 0x1809). The primary features of LPRUN is that the CPU clock is directly reduced to 500 kHz and the digital voltage regulator is held in standby mode while the radio remains active (see MC1323x Reference Manual, Section 5.7

for a detailed description of this mode). This mode is released back to normal RUN mode again through programming of register SPMSC2.

LPRUN is the lowest power run mode, however, changing the CPU clock through the RDIV prescaler retains the advantage of tuning performance with lower CPU/bus clock while retaining needed throughput.

NOTE

- The impact of running a communications stack or other application at a reduced clock rate is your responsibility. As stated previously, the Freescale provided stacks default to the maximum CPU clock for best performance and you must evaluate and handle any problems induced by going to a lower clock rate.
- Any active peripherals with lower bus clock may need timing or baud rate adjustments from when running at full bus rate.
- The communications stacks may require writing of parametric data to the FLASH on an occasional basis. Beware that the bus clock must be restored to full speed before writing data to FLASH.

4.2 Disabling MCU Peripherals and Peripheral Clocks When Unused

The MC1323x provides a host of peripherals and current can be saved by disabling any peripheral when unused or not needed. Two means are available for enabling peripherals:

1. Peripheral clock gating enable — Registers SCGC1 and SCGC2 (Addresses 0x180E and 0x180F) provide individual peripheral clock gate enable for the functional modules and peripherals (default is all enabled). When a specific peripheral is not in use, current can be saved by disabling the clock to that peripheral.
2. Individual peripheral control — Peripherals do not share a common control structure, but each has means to enable and disable function of the peripheral. Peripherals can be enabled on an “as needed” basis.

4.3 Managing Transceiver (Radio) Operation

The MC1323x transceiver consumes the greatest share of current when transmitting or receiving. Control of the radio is usually done through the codebase/stack services, but careful manipulation of the transceiver/modem control may save some current:

- The transceiver is capable of full operation with reduced CPU clock — Clock reduction techniques discussed previously can be used and still retain transceiver operation.
- Transmit time is very deterministic due to known packet length, and as a result, little or no control of the radio is possible to reduce TX “on” time
- Receive “on” time is typically more dominate due to not knowing the exact time an incoming packet will arrive. For a particular use model, review when the RX mode is enabled. Consider if the RX mode “on” time can be reduced; do not just leave RX continually on as standard practice.
- Disable the analog regulator for lowest power (same operation for low-power mode) — The analog regulator, as standard practice, is enabled during stack/transceiver initialization. If the radio is not

in use for an extended period of time, then the regulator can be disabled, and then re-enabled prior to radio use.

— Write 0x00 to IA Register 0x60 to disable the regulator.

— Restore value 0x01 to IA Register 0x60 to re-enable regulator for radio use.

NOTE

Caution must be observed if the analog regulator is disabled while the stack is active. You must prevent the stack from trying to use the radio while the analog regulator is disabled.

5 Means of Wakeup from Low Power

This section provides some overview of the different means available for wakeup from sleep mode:

- Real-Time Counter (RTC) — The RTC has a 16-bit counter and a 16-bit comparator that uses a powerful clock prescale function to implement a timer. The resulting RTC counter clock can range from microseconds to seconds. With the programmable comparator and its IRQ capability, the RTC can be used to wake the MC1323x from sleep mode. The selected oscillator source determines clock and timing period accuracy, available time period, and low power current usage.
- External interrupt request pin (IRQ) — The IRQ pin can be enabled and used as a wakeup source in Stop3. The sense and either edge vs. level detect mode is programmable.
- The KBI functions — The MC13234 has two KBI blocks that provide up to 12 individual inputs capable of generating an interrupt request from an input signal transition. When using the MC13237, you have only one KBI block providing up to 8 individual inputs. Typically, the KBI is used to provide wakeup from switch closures or keypad matrix activity. With a keypad and after wakeup, the MC1323x typically enters a scan routine of some sort to determine which key transition triggered the wakeup.
- UART RX signal activity — In Stop3 mode the UART IRQ can be used to provide wakeup such as from a host. Transitions on the RX pin can generate an interrupt request and wakeup the device. The UART will not be ready to immediately receive an incoming UART byte, but a wakeup protocol can be implemented where a host can “ping” the device until a reply is sent.
- LVD/LVW — The low voltage detection/warning circuit can be used to affect device wakeup in Stop3, but it is strongly advised NOT to be used. It uses significant current and under the very low current draw of low power (microamps), the battery voltage typically is going to change very little.
- Hardware reset — Although not necessarily preferred, a reset is a means of waking the MC1323x from sleep mode.

6 Summary Flow for Entering Low-Power Mode

The following flow summarizes actions for entering low-power mode:

1. Exit any stack/application — Save any required parameters, shutdown any tasks that are affected.

2. As required reconfigure any GPIO — It is assumed that during system initialization, any unused GPIO are configured properly so as to not cause excess leakage current. At this point, re-configure any GPIO in use that may special handling during low power so as to not cause excess leakage (see “Section 3 Configuring MC1323x GPIO for Best Practice”.
3. Disable the transceiver analog regulator if required — See Section 4.3, “Managing Transceiver (Radio) Operation.
4. Disable any active peripheral clocks.
5. Initiate any required wakeup sources / interrupts.
6. Clear LVW/LVD as required.
7. Prepare “Stop” mode conditions in System Control Registers, see MC1323x Reference Manual, Section 5.8. This includes setting the STOP Enable bit.
8. Execute a STOP instruction.

7 Managing Low Power from Freescale Software Stacks

Applications that use software stacks that are built on top of the Freescale 802.15.4 MAC manage low power by using the *Low Power Library* platform module. This platform module provides a common way of configuring low power across software stacks and hardware platforms, as well as a common API set for managing low-power entry and exit from an application viewpoint.

Managing low power from a software application perspective happens in two steps:

- The first step involves how an application is designed to enter low power, and subsequently exit low power (ready to re-enable the stack). This step differs across software stacks.
- The second step is to then call Low-Power Module APIs to place the hardware platform into a low-power mode.

The following sections detail how to configure the Low-Power Module for optimum low power, how to ready an application for low-power entry, and then how to use the Low Power APIs to enter and exit a low-power mode.

NOTE

The example used is a BeeStack Consumer (RF4CE) application running on a MC1323x.

7.1 Entering and Exiting Low Power from Applications

Applications must first be ready to enter low power and subsequently exit low power. After it is ready, the application must indicate to the stack. The methods of doing this are slightly different across software stacks; i.e., doing this in a BeeStack Consumer application is different than doing this in a BeeStack application. Note however, that how a stack does this is the same across platforms; i.e., a BeeStack Consumer application running on a MC1323x will use the same method as a BeeStack Consumer application running on a MC1321x or MC1322x.

The following sections detail the method of how to enter and exit low power for BeeStack Consumer.

7.1.1 Entering and Exiting Low Power from a BeeStack Consumer Application (RF4CE)

The BeeStack Consumer application manages entering and exiting low power as part of its IdleTask() operations, so as to not interfere intrusively with any stack or application operations. The application must ensure that it is ready to enter low power by checking the next:

1. Establish that no timers are active. If timers are required during low power, the application should use a low-power timer that is allowed to be active during low-power states.
2. Establish that no events are pending.
3. Establish that the receiver is off by calling: NLME_RxEnableRequest(0x00).
4. Indicate to the stack that the application is now ready to enter low power by calling: PWR_AllowDeviceToSleep().

Control of entering low-power mode is managed in the IdleTask(). It determines whether PWR_AllowDeviceToSleep() has been called. If so it then calls the low-power module API PWR_EnterLowPower(). This API determines that no timers other than low-power timers are active and that no events are pending. If both of these conditions are met, then the system will enter low-power mode as configured by PWR_Configuration.h. If these conditions are not met low-power mode will not be entered.

The PWR_EnterLowPower() API is a blocking function and will not return until the platform has woken up from sleep by a wakeup event and normal execution continues after PWR_EnterLowPower() exits.

7.2 Configuring MC1323x Specific Features for Optimum Low Power

The MC1323x has specific features that the application must configure for low power as summarized in Section 6, “Summary Flow for Entering Low-Power Mode. After these are configured, the application can enter low power by using the power module as described in Section 7.1, “Entering and Exiting Low Power from Applications. The following list provides specific details for device configuration:

1. Disabling/enabling the Analog Regulator — The analog voltage regulator, as standard practice, is enabled during transceiver initialization. If the radio is not in use for an extended period of time, then the regulator can be disabled, and then re-enabled prior to radio use. This is handled by the Low Power Library.
2. Disabling MCU peripheral clocks — Any peripheral clocks not in use should be disabled before entering low-power mode. This is accomplished by clearing bits in the SCGC1 and SCGC2 registers, see Section 4.2, “Disabling MCU Peripherals and Peripheral Clocks When Unused.
3. Disabling external interrupt request IRQ — If not in use, the IRQ pin should be disabled before entering low power.
   - — Clear the IRQF bit in the IRQSC register.
   - — Also, set the IRQEDG and IRQACK bits in that same register.
4. Clearing Low Voltage Detect Acknowledge — Before entering low power, an application should always clear the low voltage detect acknowledge. This is done by setting the LVDACK bit in the SPMSC1 register.

5. Stopping all oscillators — All oscillators should be stopped for absolute lowest power; wakeup then must be done via an IRQ, KBI, or SCI interrupt request. This is done by clearing all bits in the SOMC1 register. (If a timed wakeup is required, use either the 1 kHz RC oscillator or the optional 32 kHz crystal oscillator)
6. Disabling COP — If in use, the COP must be disabled before entering low-power mode. This is done by clearing the COPE bit in the SOPT1 register.

After configuration, Stop mode is enabled by setting the STOPE bit in the SOPT1 register.

NOTE

- The STOPE bit can be set in the same register write that disables the COP timer.
- The SOPT1 register appears in the code as “SIMOPT”.

8 Configuring Application and Low-Power Module (LPM) for Optimum Low Power

The low-power module (LPM) simplifies the process of configuring low-power or sleep modes across platforms. It provides a common way of configuring these modes as well as a common API to manage entering and exiting low power. The LPM is sometimes also referred to as the ‘low power library’.

NOTE

- The LPM assumes that Stop3 is the desired low-power mode — It is the recommended and most commonly used mode and has best recovery time.
- The LPM assumes that the RTC is NOT running on a continuous basis. If the RTC is used, the LPM sets-up and enables the RTC under the assumption that the RTC is only being used as a wakeup timer. It is possible, however, to use the RTC as a continuously running timebase (optionally using a time-tick to wakeup from low-power mode); implementing this mode is left as an exercise for you.

8.1 Low-Power Module Overview

The following files comprise the low-power module:

- PWR_Interface.h
  - — public APIs for managing entering and exiting low power used by the application.
- PWRLib.h
  - — defines, typedefs and private APIs for use by the power module. For use with MC1321x, MC9S08QE128 and MC1323x.
- PWR_Configuration.h
  - — defines for configuring low power for MC1221x, MC9S08QE128 and MC1323x.
- PWR_MC1323x.c

  - implementation for public APIs for managing entering and exiting low power used by the application. For use with MC1323x (the subject of this application note).
- PWRLib.c
  - implementation for private APIs used by the power module. For use with MC1321x, MC9S08QE128 and MC1323x.

All configuration is done at compile-time through properties located in PWR_Configuration.h. The low-power module is initialized automatically by the stack.

If the gLpmIncluded_d property is selected (set to TRUE) low power is enabled whenever the application is idle (no timers or events pending, and the receiver is disabled). The application can elect to stay awake by using the PWR_DisallowDeviceToSleep() function.

NOTE

With the Exception of PWR_DisallowDeviceToSleep( ) and PWR_AllowDeviceToSLeep(), an application should not call the low-power functions directly. The idle task already contains the proper code for entering low-power modes as appropriate in a way that coordinates with the entire system.

The following descriptions detail how to configure properties in PWR_Configuration.h for optimum low-power configuration of a MC1323x MRB. The associated example BeeStack Consumer application uses the low-power module to configure the GPIOs on the MC1323x MRB for optimum low power.

A sample RF4CE application is provided in conjunction with this document. It will demonstrate approximately 700 nanoamp current in low-power Stop3 mode with the capability of being waking up from SCI, KBI or RTC interrupts.

8.2 Configuring PWR_Configuration.h Low-Power Module Properties

As described, PWR_Configuration.h controls configuration of the MC1323x as the LPM places the device into low-power or sleep mode. The following sections detail configuration of properties in PWR_Configuration.h used to achieve optimum low power for a MC1323x MRB platform.

NOTE

Configuration of MC1321x or MC9S08QE128 is handled in similar fashion by configuring the file PWR_Configuration.h, however this application note focuses on configuring the low-power module for the MC1323x MRB.

This example configures the low-power module for:

- Stop3
- Wake on SCI, RTI or KBI interrupts.
- The low-power module also manages configuration of:
  - The KBI registers and interrupt vector for wakeup
  - The GPIO setup and restoration for optimum low power.

The following sections describe individual properties and how they are to be used.

NOTE

The PWR_Configuration.h file included in the project attached to this application note can be referenced while perusing the following paragraphs.

8.2.1 cPWR_UsePowerDownMode

This property enables or disables all of the low-power module code. It must be set to '1' (TRUE) to enable the Low Power Library.

```
//------------------------------------------------------------------------------
// To enable/disable all of the code in this PWR/PWRLib files.
//   TRUE =  1: Use PowerDown functions (Normal)
//   FALSE = 0: Don't use PowerDown. Will cut variables and code out. But
//     functions still exist. Useful for debugging and test purposes
#ifndef cPWR_UsePowerDownMode
#define cPWR_UsePowerDownMode                  1
#endif
```

8.2.2 cPWR_DeepSleepMode

This property selects the deep sleep mode. This example selects option '3,' which sets the MCU to Stop3 and RTI/SCI/KBI wakeup with KBI enabled.

```
//------------------------------------------------------------------------------
// The way that DeepSleep mode are handled. Following possibilities exist:
//   0: No DeepSleep done, but PTC application can set modes
//   1: Ext. KBI int wakeup
//   2: RTI timer wakeup +-30%

//   3: Ext. KBI int and RTI timer wakeup +-30%, radio in OFF/reset mode.
//   3: MC1323X: MCU in STOP3, RTI/SCI/KBI wakeup, KBI enabled

//   4: Ext. KBI int and RTI timer wakeup +-30%, radio in hibernate - not reset
//   4: MC1323X: MCU in STOP3, RTI/SCI/KBI wakeup, KBI enabled. RTI wakeup timeout selectable
at runtime. RTI resolution fixed to 32us

//   5: Radio in acoma/doze, supplying 62.5kHz clock to MCU, MCU in STOP3,
//      RTI wakeup from ext clock 512mS (max avail with ext 62.5khz)
//   5: MC1323X: Mode 5: MCU in STOP3, RTI/SCI/KBI wakeup, KBI enabled

//      Suggested use: debug mode for modes 3 or 4

//   6: Radio in doze mode, supplying 62.5KHz to MCU, MCU in STOP3
//      Radio wakeup after cPWR_DozeDurationMs. KBI Int Wakeup available also.
//   6: MC1323X: MCU in STOP3, Wakeup on 802.15.4 timer, Other wakeup sources: KBI/SCI

//-------- Test modes : ----------------------------------------------------
//  30: Test of some stack code (Not good or valid anymore)
//  31: Test as #3 but power down only done once
//  33: Test of MCU Wait
//  34: Test of MCU Stop3
```

```
//  36: Test of MCU Stop1
//  37: Test of RADIO Doze without clkout
//  38: Test of RADIO AcomaDoze without clkout
//  39: Test of RADIO Hibernate
//  40: Test of RADIO Off
#ifndef cPWR_DeepSleepMode
#define cPWR_DeepSleepMode                      3
#endif
```

8.2.3 cPWR_SleepMode

This property selects the way that sleep mode is handled. Set to ‘1’ (TRUE) to enable sleep mode.

Doze mode NOT VALID CONDITION

```
//-------------------------------------------------------------------------
// The way that Sleep mode are handled. Following possibilities exist:
//   0: No Sleep done, but PTC application can set modes
//   1: Doze mode on RADIO with clock enabled (MAC 1.06 only) and WAIT on MCU
#ifndef cPWR_SleepMode
#define cPWR_SleepMode                          1
#endif
```

8.2.4 cPWR_RTITickTime

This property defines the distance between RTI interrupts when enabled in Stop3.

```
//-------------------------------------------------------------------------
// The distance between RTI interrupts when enabled and in Stop3
// Following possibilities exist:  cSRTISC_IntDisabled,
// cSRTISC_Int0008ms, cSRTISC_Int0032ms, cSRTISC_Int0064ms, cSRTISC_Int0128ms,
// cSRTISC_Int0256ms, cSRTISC_Int0512ms, cSRTISC_Int1024ms
// MC1323X: This define represents the RTC tick time and depends on the cPWR_RTIClockSource
#ifndef cPWR_RTITickTime
 #ifndef PROCESSOR_MC1323X
  #define cPWR_RTITickTime                        7
 #else
  #define cPWR_RTITickTime                        7
 #endif
#endif
```

8.2.5 cPWR_DozeDurationMs

This property defines the distance between Radio Timer interrupts. It is used when cPWR_DeepSleepMode==6. Default value is 2000. For this application note, we can ignore the value of this variable. Since cPWR_DeepSleepMode configuration used is 3.

```
//-------------------------------------------------------------------------
// The doze duration in ms when enabled in Stop3. Used in cPWR_DeepSleepMode == 6
// Default value of 3072ms set to match the default timeout set when using RTI modes
#ifndef cPWR_DozeDurationMs
 #ifndef PROCESSOR_MC1323X
```

```
  #define cPWR_DozeDurationMs                    (3072)
 #else
  #define cPWR_DozeDurationMs                    (2000)
 #endif
#endif
```

8.2.6 cPWR_RTITicks

This number multiplied with the above cPWR_RTITickTime time gives the time to DeepSleep.

```
//-------------------------------------------------------------------------
// This number multiplied with the above cPWR_RTITickTime time gives the time
// to DeepSleep
#ifndef cPWR_RTITicks
 #ifndef PROCESSOR_MC1323X
  #define cPWR_RTITicks                          5
 #else
  #define cPWR_RTITicks                          5
 #endif
#endif
```

8.2.7 cPWR_CallWakeupStackProcAfterDeepSleep and cPWR_DeepSleepWakeupStackProc

These properties determine whether or not an application defined routine is called every time the RTI clock expires.

```
//-------------------------------------------------------------------------
// Enabling of external call to a procedure each time that DeepSleep are exited
//   0: Don't call any functions after DeepSleep (MAC)
//   1: Call a function after DeepSleep (Stack)
#ifndef cPWR_CallWakeupStackProcAfterDeepSleep
#define cPWR_CallWakeupStackProcAfterDeepSleep  0
#endif

//-------------------------------------------------------------------------
// The extra function to call every time RTI clock run's out. Used by Stack.
#if (cPWR_CallWakeupStackProcAfterDeepSleep == 0)
  #define cPWR_DeepSleepWakeupStackProc         ;
#else
  extern void                                   DeepSleepWakeupStackProc(void);
  #define cPWR_DeepSleepWakeupStackProc         DeepSleepWakeupStackProc();
#endif
```

8.2.8 cPWR_KBIInitAndVectorEnable

This property selects or deselects whether the low power library manages the initialization/configuration of the KBI registers to enable KBI wakeup. In this example, select ‘1’ (TRUE) to have the Low Power Library handle this function for the application.

```
//-------------------------------------------------------------------------
// KBI interrupt vector and initialization enable
//   TRUE = 1: KBI interrupt vector used and KBI registers are initialized in PWRLIB(Normal)
//   FALSE= 0: KBI interrupt vector and init has to be done externally
#ifndef cPWR_KBIInitAndVectorEnable
#define cPWR_KBIInitAndVectorEnable            1
#endif
```

8.2.9 cPWR_KBIWakeupEnable

This property configures KBI wakeup enabled or disabled. Setting this will configure the low-power module to automatically enable the KBI interrupt. This example is set to ‘1’ to allow wake on KBI interrupt.

```
//-------------------------------------------------------------------------
// Select if the KBI interrupt has to be enabled or disabled if the define:
// cPWR_KBIInitAndVectorEnable are TRUE
//   TRUE = 1: Also use KBI interrupt to wakeup
//   FALSE= 0: Don't use KBI int. on PTAx pins
#ifndef cPWR_KBIWakeupEnable
#define cPWR_KBIWakeupEnable                   1
#endif
```

8.2.10 cPWR_SetupIOWhenInPD

This property configures whether or not and how the Low Power Library configures GPIOs entering and exiting low power. This example sets to option ‘1’ for safest operation of setting and restoring GPIOs. The associated example application uses this mode to set/restore the GPIOs for minimal power usage.

```
//-------------------------------------------------------------------------
// Setting up the port settings when entering and exiting PowerDown
//   0: No change done to IO when entering PD (Debug)
//   1: Setup all ports (Most as outputs) when entering PD. Restoring all
//      ports using saved values. Uses 21 bytes in RAM. (Safest)
//   2: Setup all ports (Most as outputs) when entering PD (Normal). Restoring
//      all ports. Only Data part from saved values restored from preset values.
//      Uses only 7 bytes in RAM. (Normal).
#ifndef cPWR_SetupIOWhenInPD
#define cPWR_SetupIOWhenInPD                    1
#endif
```

8.2.11 cPWR_UseSPIDisable

This property does not take effect in MC1323x family. We can keep this define as 0.

```
//-------------------------------------------------------------------------
// Whether to setup RADIO SPI port as I/O when entering PD mode
//   TRUE =  1: Use the SPI pins as outputs under PD (Normal)
//   FALSE = 0: Don't
#ifndef cPWR_UseSPIDisable
#define cPWR_UseSPIDisable                      0
```

```
#endif
```

8.2.12 cPWR_UseDebugOutputs, cPWR_UseRADIOStatus and cPWR_UseMCUStatus.

These defines are for debug purposes. Just keep them with a value of ‘0’.

8.2.13 cPWR_RTIClockSource

This property defines the clock source for the RTI in low-power modes. Possible values are defined in PWRLib.h:

```
#define cSRTISC_SourceInt1KHz  0x00
#define cSRTISC_SourceExt32MHz 0x20
#define cSRTISC_SourceExt32KHz 0x40

//-------------------------------------------------------------------------------
// Whether to run RTI from internal or external oscillator
//   TRUE =  1: Use external clock for RTI timer
//   FALSE = 0: Uses internal clock (Normal)

/* Use external clock in debugging mode */
/* Use internal clock in normal, lowest-power mode */
#ifndef cPWR_RTIClockSource
#define cPWR_RTIClockSource cSRTISC_SourceInt1KHz
#endif
```

8.2.14 cPWR_LVD_Enable, cPWR_LVD_Ticks and cPWR_LVD_LEVEL_50_Ticsk

This property enables or disables the low voltage detection (LVD). This example disables the LVD by setting the property to ‘0’. Because this value is kept as ‘0’, cPWR_LVD_Ticks and cPWR_LVD_LEVEL_50_Ticks are ignored.

```
//-------------------------------------------------------------------------------
// The use of Low Voltage detection has the following possibilities:
//   0: Don't use Low voltage detection at all
//   1: Use polled => Check made each time the function is called.
//   2: RTI timer used for handling when to poll LVD, according
//      to the cPWR_LVD_Ticks constant
//   3: LVDE and LVDRE  are set to hold MCU in reset while LVD_VH  condition is detected
//
#ifndef cPWR_LVD_Enable
#define cPWR_LVD_Enable                         0
#endif
```

8.2.15 cKBI1SC and cKBI1SC_Ack

This property determines the desired configuration for the KBI module. In this example is to enable interrupts and definition for the KBI ACK to clear interrupt flag.

```
//------------------------------------------------------------------------------
// Definitions for KBI interrupt setup (KBI1SC)
//                          .--------- KBEDG7    : 1:Rising edges/high. 0:Falling edge/low level
//                          |.-------- KBEDG6    : 1:Rising edges/high. 0:Falling edge/low level
//                          ||.------- KBEDG5    : 1:Rising edges/high. 0:Falling edge/low level
//                          |||.------ KBEDG4    : 1:Rising edges/high. 0:Falling edge/low level
//                            ||||.---- KBF       : Keyboard Int. status Flag (Readable only)
//                            |||||.--- KBACK     : Keyboard Int. Acknowledge (Writable only)
//                            ||||||.-- KBIE      : Keyboard Int. Enable
//                            |||||||.- KBIMOD    : 1:Edge and level, 0:Edge
#define cKBI1SC            0b00000010            // Enable Edge triggered interrupts from KBI
#define cKBI1SC_Ack        0b00000100            // Acknowledge of int
```

8.2.16 mPWRLib_SETUP_PORT_X

These macros (mPWRLib_SETUP_PORT_A, mPWRLib_SETUP_PORT_B, mPWRLib_SETUP_PORT_C, mPWRLib_SETUP_PORT_D) define the configuration of the GPIOs for low power consumption. As it was discussed on previous chapter, correct configuration of the GPIOs is critical to get the low power measure required for the application. Then, these macros should contain the correct configuration for the low power consumption, the definition of these macros should change according to the HW that is going to be tested. The associated project to this application note already handled the correct configuration for the 1323x-MRB.

NOTE

Before building the project, make sure that “Keyboard.h” file is included in the “PWRLib.c” file if you are using BeeStack Consumer 1.7.1 to avoid errors. It will be fixed in next releases. Look for the word “AN4398” in the attached project to identify the changes to accomplish low power measures.

9 Sample Application Demonstrating Low Power

All hardware references in this document are based on the Freescale 1323x-Modular Reference Board (1323x-MRB) evaluation board. This board is Freescale’s vehicle for MC1323x evaluation and application development. You should be familiar with this board, see *1323x Development Hardware Reference Manual* (1323xDHRM.pdf), Chapter 3.

This note is supplemented by one application that demonstrates low-power operation. The full stack-based project is based on BeeStack Consumer Application (RF4CE). This application is supplied as a full project bundle built with Freescale’s BeeKit IDE environment. It is an RF4CE application making use of the PWRLib tool. Use CodeWarrior 10.3 to test this application.

NOTE

Source code and application code is available on the Freescale website as a link related to this application note

The low power app target platform is the 1323x-MRB and it must be configured properly to observe the operating current. The schematic for the 1323x_MRB can be found in Chapter 3 of the *1323x Development Hardware Reference Manual* (1323xDHRM.pdf).

9.1 Configuring the 1323x-MRB Hardware

In the 1323x-MRB, the only thing that you need to do is to measure jumper J9 between pins 5-6. The 1323x-MRB needs to be connected in a 1323x-REM, because the configuration given in the application was done using these two boards. Take in consideration that GPIO configuration depends on the hardware you have connected to the MC1323x device.

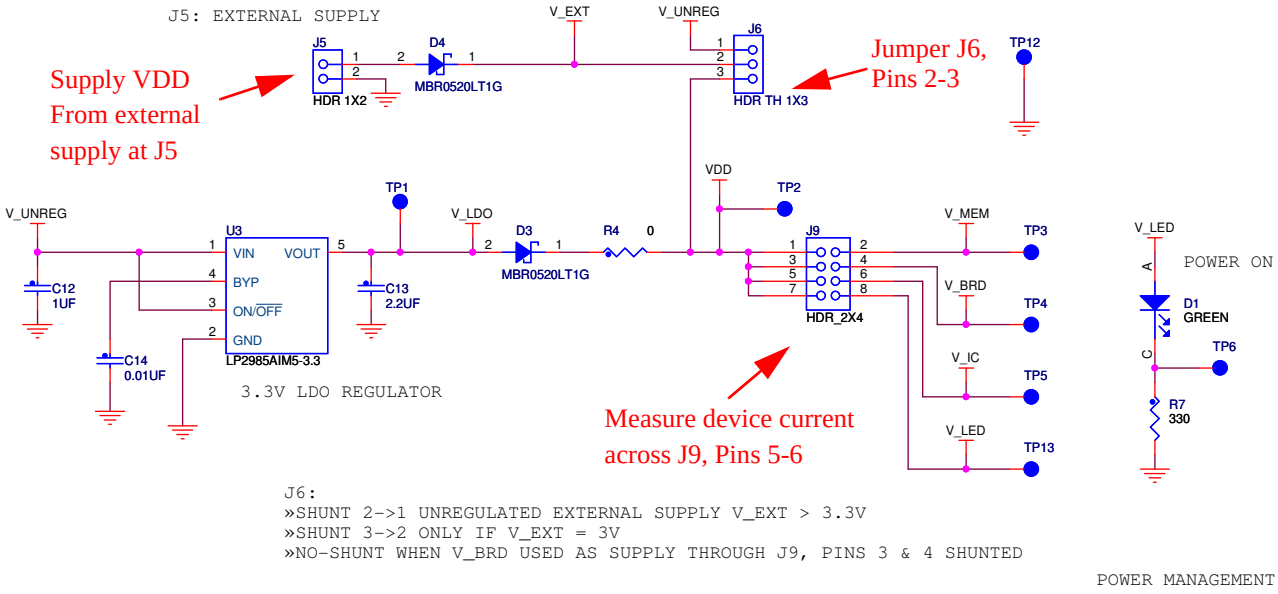

Figure 1. 1323x-MRB Power Management Circuit

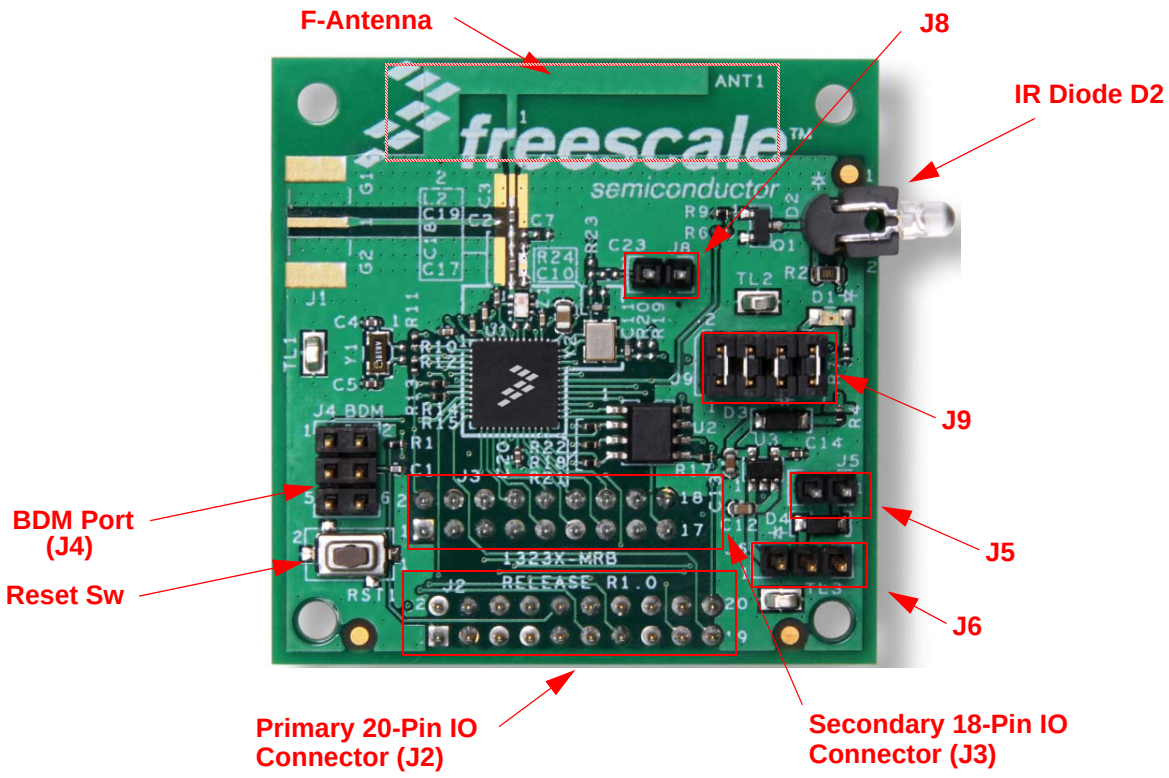

Figure 2. 1323x-MRB Layout

Figure 3 shows the 1323x-MRB configured for low power current measurements.

- Jumpers on Pins 5 and 6 (V_IC), 7 and 8 (V_LED) on J9 were removed.
- A current meter is connected across pins 5 and 6 (V_IC) on J9 with positive on Pin 5 (use a current range of 5-10 mA; having an ammeter that measures to 100 nA is suggested).

Remember that the hardware for this application note is one 1323x-MRB connected to a 1323x-REM. You only need to connect the current meter to do the measures on V_IC as explained above. See Figure 3 for your reference.

Finally, because development kit used in this application note was designed for general purpose and not for low power, there is an extra step you need to do to see the low power values. The UART_RXD pin must be disconnected on the 1323x-REM. It is pin 11 & 12 on the P1 header. See Figure 3 to locate the jumper. Otherwise, you will measure a leakage current instead of the desired low power value. This can be solved with level shifters for different voltage between MC1323x device and the USB to SER device.

NOTE

If UART_RXD is removed, then you will not be able to see the messages sent from the MC1323x device to the PC (Hyperterminal).

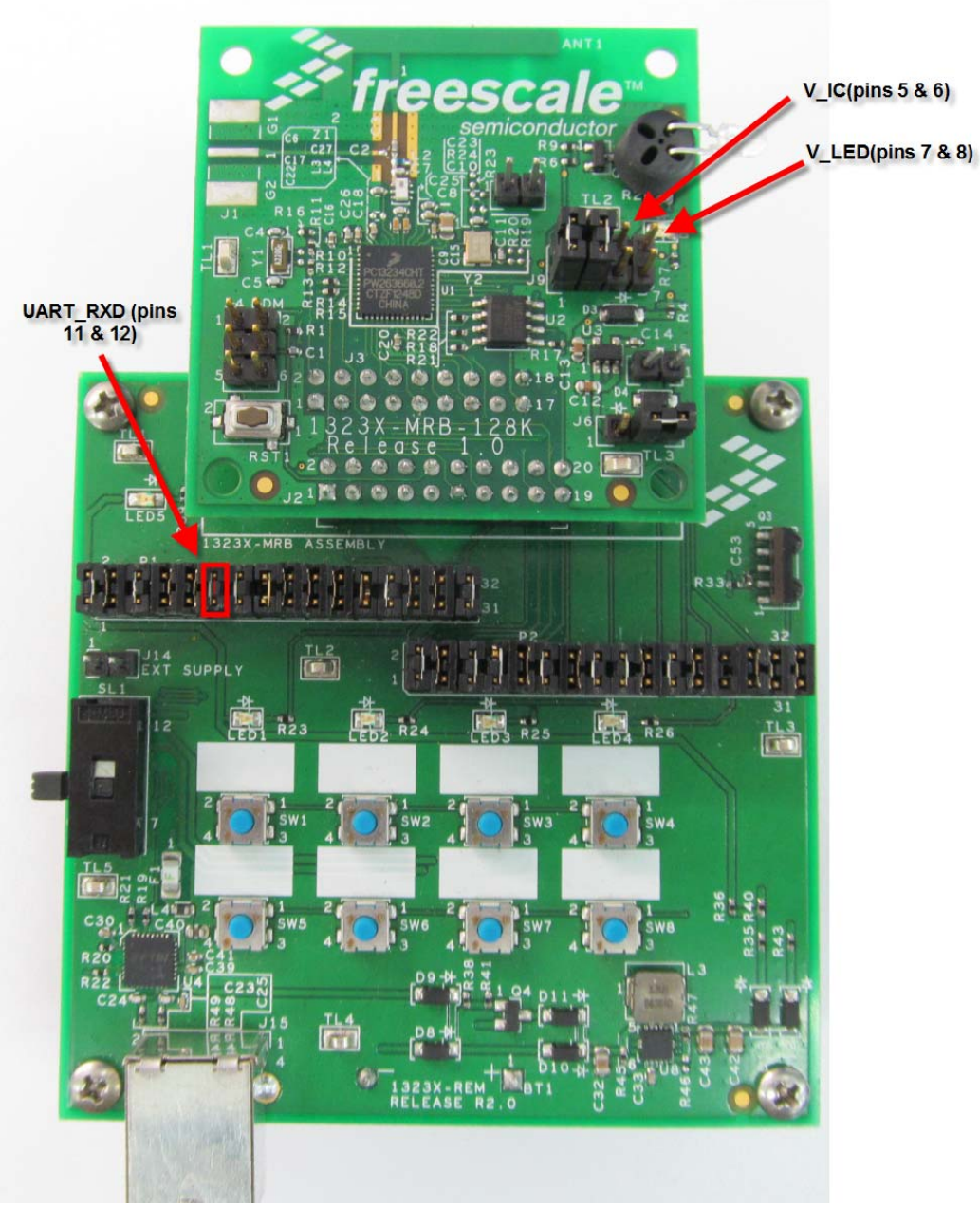

Figure 3. 1323x-MRB plus 1323x-REM

9.2 Running the RF4CE Stack-Based Sample Application to Measure Low Power on the 1323x-MRB

The project provided is based on the Freescale BeeStack Consumer Application (RF4CE) codebase version 1.7.1 and provides lowest power mode using Stop3 configuration with RTC, SCI, and KBI interrupts as a wakeup source. The project can serve as the basis of creating an app with low-power

implementation based on the PWRLib. To run the attached project to this application note, open the “Read Me” file included in the .zip file.

10 Summary

This application note gives a detailed description of areas that must be addressed to provide lowest power when using the MC1323x. No attempt is made to provide examples for all possible low-power combinations, but these guidelines can be applied to all combinations. To assist in programming low-power applications code, Freescale provides the Low-Power Module utility (Low Power Library) that is usable with all Freescale IEEE 802.15.4 MAC-based software stacks. A sample stack-based application with source is provided to demonstrate low power and give a starting example for user applications based on the RF4CE stack.

11 Reference

AN4573: *Low Power Considerations for ZigBee Applications Operated by Coin Cell Batteries.*

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4398
Rev. 0
7/2013