Freescale Platform

Reference Manual for ZigBee 2007

Document Number: FSPRMZB2007
Rev. 1.3
09/2011

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2006, 2007, 2008, 2009, 2010, 2011. All rights reserved.

Contents

About This Book

The *Freescale Platform Reference Manual for ZigBee 2007* describes in detail the API to the Freescale platform components shared among Freescale networking solutions (for example, BeeStack, the Freescale IEEE 802.15.4 MAC, and the Freescale Simple MAC). Many components interact with reference hardware such as switches, the LCD and LEDs. Other components include timers and the task scheduler.

Audience

This document is for engineers developing BeeStack or other 802.15.4 networking applications.

Organization

This document is organized into the following sections.

Chapter 1	Introduction – provides an overview of all the platform components and where they can be found in the directory structure in a BeeStack project.
Chapter 2	Task Scheduler – describes the task scheduler API and compile-time options
Chapter 3	Timer – describes the timer API and compile-time options.
Chapter 4	LED – describes the LED API and compile-time options.
Chapter 5	Display – describes the LCD API and compile-time options.
Chapter 6	Keyboard – describes the keyboard API and compile-time options.
Chapter 7	UART – describes the UART (SCI port) API and compile-time options.
Chapter 8	Non-Volatile Memory – describes the non-volatile memory API and compile-time options.
Chapter 9	Low Power Library – describes the lower power API and compile-time options.
Chapter 10	Touch Pad – describes the touch pad API and compile-time options.
Chapter 11	Accelerometer r– describes the accelerometer API and compile-time options.
Chapter 12	I^{2}C – describes the I^{2}C API and compile-time options.
Chapter 13	SPI – describes the SPI API and compile-time options.
Chapter 14	CMT – describes the CMT API and compile-time options.
Chapter 15	OTAP – describes the Over-the-air programmer API and compile-time options.
Chapter 16	EEPROM – describes the EEPROM API and compile-time options.
Chapter 17	Bootloader – describes the Bootloader API and compile-time options.

Conventions

This document uses the following conventions:

Courier — Is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.

Italic — Is used for emphasis, to identify new terms, and for replaceable command parameters.

All source code examples are in C.

Definitions, Acronyms, and Abbreviations

The following list defines the abbreviations used in this document.

APS	Application Support sub-layer, a ZigBee stack component
APL	Application Layer, a ZigBee stack component
BDM	Background Debug Mode: The HCS08 MCUs used here have a BDM port that allows a computer to program its flash memory and control the MCU's operation. The computer connects to the MCU through a hardware device called a BDM pod. In this application, the pod is the P&E USB HCS08/HCS12 Multilink
BeeKit	Freescale Wireless Connectivity Toolkit networking software
Binding	Associating two nodes in a network for specific functions (e.g., a light and switch)
Cluster	A collection of attributes accompanying a specific cluster identifier (sub-type messages.)
EVB	Evaluation Board, a Freescale development board
GUI	Graphical User Interface: BeeKit and CodeWarrior, the two Windows tools discussed here, each uses a GUI
HCS08	A member of one of Freescale's families of MCUs
IDE	Integrated Development Environment: A computer program that contains most or all of the tools to develop code, including an editor, compiler, linker, and debugger
MAC	IEEE 802.15.4 Medium Access Control sub-layer
MC13193	One of Freescale's IEEE 802.15.4 transceivers
MCU	Micro Controller Unit: A microprocessor combined with peripherals, typically including memory, in one package or on one die
NCB	Network Coordinator Board, a Freescale development board
Node	A device or group of devices with a single radio
NWK	Network Layer, a ZigBee stack component
OUI	Organizational Unique Identifier (The IEEE-assigned 24 most significant bits of the 64-bit MAC address)
PAN	Personal Area Network
Profile	Set of options in a stack or an application
SARD	Sensor Application Reference Design, a Freescale development board

SMAC	Freescale Simple MAC, a very simple, very small proprietary wireless protocol that uses the Freescale IEEE 802.15.4 radios
SRB	Sensor Reference Board, a Freescale development board
SCI	Serial Communication Interface. This is a hardware serial port on the HCS08. With the addition of an external level shifter, it can be used as an RS232 port
SPI	Serial Peripheral Interface. This is a serial port intended to connect integrated circuits that are together on one circuit board
SSP	Security Service Provider, a ZigBee stack component
Stack	ZigBee protocol stack
Toggle	A toggle switch moves from one state to its other state each time it is toggled. For instance, if the first toggle takes the switch to Off, the next toggle will be to On, and the one after that will be to Off again. In the applications this document describes, the switches are momentary push buttons with no memory of their states. The HCS08 maintains each switch's state
UART	Universal Asynchronous Receiver Transmitter, an MCU peripheral for access to devices not on the same circuit board. With level shifting, the UART implements RS-232
UI	User Interface
ZC	ZigBee Coordinator: one of the three roles a node can have in a ZigBee network
ZED	ZigBee End Device: one of the three roles a node can have in a ZigBee network
ZR	ZigBee Router: one of the three roles a node can have in a ZigBee network
802.15.4	An IEEE standard radio specification that underlies the ZigBee specification

References

The following documents were referenced to build this document.

- Freescale BeeStack Software Reference Manual for ZigBee 2007, Document BSSRMZB2007.
- The data sheets for the Freescale family of ZigBee transceivers
- Freescale MC9S08GB/GT Data Sheet, Document MC9S08GB60, December 2004
- Freescale MC9S08QE128 Reference Manual*,* Document MC9S08QE128RM

Revision History

The following table summarizes revisions to this manual since the previous release (Rev. 1.2).

Revision History

Location	Description
Chapters 15 and 16	Multiple updates for OTAP.

Chapter 1
Introduction

This section provides an overview of all the platform components and where they can be found in the directory structure in a BeeStack project. This document is a reference manual, not a tutorial. For advice on using the platform components, see the Freescale *BeeStack Application Development Guide for ZigBee 2007*.

Platform components generally interact with hardware and are designed in such a way as allow applications to modify these components for custom boards or situations. For example, the Non-Volatile Memory (NVM) API is designed so that if the storage that is readily available is not native flash, but static RAM or I^2C, then the NVM code can be replaced and will still function the same from the application perspective. Likewise, the LED, LCD and keyboard other interfaces can be easily adapted to suit any hardware.

Platform components come with full source, as they are intended to be modified for any particular board design.

The platform components include

- Task scheduler – Allows non-preemptive, prioritized scheduling
- Timer – Allows events to be occur on a time basis
- LED – provides the ability to set and blink Light Emitting Diodes
- LCD – provides multi-line Liquid Crystal Display
- Keyboard – provides basic button presses
- UART – provides interaction with a host processor or desktop PC
- NVM – Non-volatile memory provides permanent storage
- LPM – Low Power Module

Most platform components can be found in the Platform Module (PLM) directory. The task scheduler can be found in the System Support Module (SSM) directory as shown in Figure 1-1.

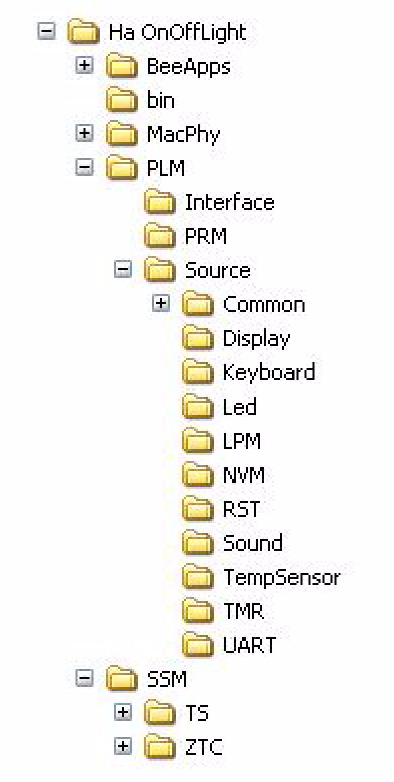

Figure 1-1. Platform Component Directory Structure

Chapter 2
Task Scheduler

The task scheduler is a non-preemptive priority based scheduler, used to conceptually separate various portions of BeeStack, or any Freescale network that uses the task scheduler.

2.1 Task Scheduler Overview

In BeeStack, the application is contained in one task by default but can be split up into multiple tasks for a particularly complex application.

The MCU interrupts operate independently of tasks, and may often pass control to a task through the use of the TS_SendEvent() function.

The following table shows task priorities in the system.

Table 2-1. Task Priorities

Priority	Description
0	Idle task
1 – 63	Platform component task priorities
64 – 191	gTsAppTaskPriority_c – Application Task priority default
192 - 198	BeeStack task priorities
254	Timer task

Freescale recommends that applications use a task priority between 64 and 191. This allows BeeStack to use priorities both lower and higher than the application as appropriate for the networking task.

Each task must have the following event handler for the task. Initialization code for a task is optional. Each event is a single bit in an event bit mask, and is defined by the task. Multiple events may be set at the same time.

```
void TaskEventHandler
  (
   event_t events
);
```

Some functions use a combination of an event bit and a message queue to communicate data, for example on the BeeAppDataIndication() function, multiple indications may be waiting, each in a separate message buffer. The messages are removed from a queue of messages.

Tasks are non - preemptive. That is, once a task gains control, it has full control until the task completes. (Returns from the task event handler function.) Tasks should complete in less than 2ms to avoid starving other tasks of processing time. If the task takes too long to process, it prevents the stack's lower layers from processing incoming packets. Under no circumstance should a task take longer than 10 ms.

2.2 Task Scheduler Properties

2.2.1 gTsMaxTasks_c

C Module Property	gTsMaxTasks_C
BeeKit Property Name	TS: Number of Tasks
BeeKit Component Containing BeeKit Property	SSM

The property gTsMaxTasks_c can be set in BeeKit (property TS: Number of tasks from the SSM component) to allow for a maximum number of tasks in the system. At initialization time, all of the tasks necessary for BeeStack are created using TS_CreateTask(). The default is 15. Currently BeeStack uses 11 tasks, plus 1 task for the application (a total of 12). The other 3 are available for task creation at run-time.

2.3 Task Scheduler API

2.3.1 TS_ClearEvent

Prototype

```
void TS_ClearEvent
  (
  tsTaskID_t taskID,/* IN: Which task. */
  event_t events/* IN: Which event(s) to clear. */
);
```

Description

Use TS_ClearEvent() to clear event bits prior to the event firing. Event bits are cleared by the task scheduler automatically prior to a task receiving control. This function might be used when resetting a task, for example, so that any pending events won't fire.

2.3.2 TS_CreateTask

Prototype

```
tsTaskID_t TS_CreateTask
  (
  tsTaskPriority_t taskPriority,/* IN: priority of new task. */
  pfTsTaskEventHandler_t pfTaskEventHandler/* IN: event handler */
  );
```

Description

Add a task to the task scheduler. Returns a task ID that can be passed to TS_SendEvent() to uniquely identify the task. If the task table is full, returns gTsInvalidTaskID_c. The taskPriority of 0 is reserved for the idle task, and must never be specified for any other task.

If TS_CreateTask() is called with a taskPriority that is the same as the priority of another task, which one is called first by the scheduler is not specified.

The task can be removed from the scheduler with TS_DestroyTask().

2.3.3 TS_DestroyTask

Prototype

```
void TS_DestroyTask
  (
  tsTaskID_t tasked/* IN: Which task to destroy. */
  );
```

Description

Use to remove a task from the scheduler that was added with TS_CreateTask().

2.3.4 TS_PendingEvents

Prototype

```
bool_t TS_PendingEvents( void );
```

Description

Returns TRUE if any task in the scheduler has pending events. Used for low power management.

2.3.5 TS_SendEvent

Prototype

```
void TS_SendEvent
  (
  tsTaskID_t taskID,/* IN: Which task to send the event to. */
  event_t events/* IN: Event flag(s) to send. */
  );
```

Description

Sends an event to another task. This function is atomic and can be called from within an interrupt handler. Multiple event bits may be set at the same time. Event bits will have unique meaning per task.

Chapter 3
Timer

Timers can be used for many purposes, including:

- Waiting for time outs
- Regular communications or sensor readings
- Blinking LEDs
- Any time based operation

The timers are non-real-time software based timers that range from 4 milliseconds (time reference) to 65,535 minutes. The timer module supports three different types of timers:

- millisecond
- second
- minute

Millisecond timers can be either a single shot or an interval timer. Second and minute timers can only be a single shot timer, but support a longer interval than millisecond timers.

Each timer can be configured to have low power capability. Low power capability means that the timer will be synchronized with the low power module allowing for the timers to run during deep sleep. The Low Power Module uses the Real Time Clock (RTI) to continue counting down the timers. If a timer expires, the CPU wakes up. If the CPU is woken up by other means, the Low Power Module will re-synchronize the timers with the time spent in low power mode.

NOTE

The RTI clock is not very accurate and will make the timer inaccurate. See the appropriate CPU data sheet for more details on the RTI.

A single hardware timer is used (TPM1 in the case of the HCS08). Due to the non-preemptive task scheduler, the time event delivered is only approximate. A hardware timer should be used for real-time time requirements.

3.1 Timer Overview

Timers must be allocated before they can be started or stopped. Use the TMR_AllocateTimer() function for this purpose in the application initialization code. The timer functions (for code size reasons) do not have error checking on the parameters, so make sure that the results of the TMR_AllocateTimer() return is not gTmrInvalidTimerID_c.

Each timer, when it expires, calls a callback function. The callback is made in the context of the timer task. Applications may consider sending an event using TS_SendEvent() to the application task for further processing, or act on the expired timer immediately in the callback. The callback is in the form of

```
void AppTimerCallBack ( tmrTimerID_t timerId );
```

The callback may be any legal C name. The timerID may be ignored if this callback is not used for multiple timers.

3.2 Timer Properties

3.2.1 gTmrApplicationTimers_c

C Module Property	gTmrApplicationTimers_c
BeeKit Property Name	Number of available timers for the application
BeeKit Component Containing BeeKit Property	PLM

The gTmrApplicationTimers_c property is set in BeeKit (Number of available timers for the application from the PLM component property). This indicates how many timers the application will need. The default is four. Ensure that this value is set high enough for all the timers that the application requires.

3.3 Timer API

3.3.1 TMR_Init

Prototype

```
void TMR_Init(void);
```

Description

The function is called to initialize the timer module. It creates the timer task, initializes the hardware timer module to work in compare mode to count repeatedly and enable the timer interrupt.

NOTE

Before calling this function, be sure that TS module is initialized. Also this function must be called prior calling other timer functions.

3.3.2 TMR_AllocateTimer

Prototype

```
tmrTimerID_t TMR_AllocateTimer(void);
```

Description

Use to allocate a timer upon initialization of the application. TMR_FreeTimer() will free this timer if required. Returns a unique timer ID that should be stored in a static or global variable of type tmrTimerID_t.

3.3.3 TMR_AreAllTimersOff

Prototype

```
bool_t TMR_AreAllTimersOff(void);
```

Description

Returns TRUE if the regular timers are off. Used by the low power library.

3.3.4 TMR_FreeTimer

Prototype

```
void TMR_FreeTimer(tmrTimerID_t timerID);
```

Description

Free a timer allocated by TMR_AllocateTimer(). This function is not often used.

3.3.5 TMR_IsTimerActive

Prototype

```
bool_t TMR_IsTimerActive(tmrTimerID_t timerId);
```

Description

This function is called to check if a timer is active or not.

3.3.6 TMR_StartTimer

Prototype

```
void TMR_StartTimer
  (
  tmrTimerID_t timerId,
  tmrTimerType_t timerType,
  tmrTimeInMilliseconds_t timeInMilliseconds,
  void (*pfTmrCallBack)(tmrTimerID_t)
  );
```

Description

The function starts a timer. When the timer goes off the callback function is executed in non-interrupt context.

NOTE

If the timer is running when this function is called, it will be stopped and restarted.

3.3.7 TMR_StartLowPowerTimer

Prototype

```
void TMR_StartLowPowerTimer
  (
  tmrTimerID_t timerId,
  tmrTimerType_t timerType,
  uint32_t timeIn,
  void (*pfTmrCallBack)(tmrTimerID_t)
  );
```

Description

The function starts a low power timer that must have been allocated first. After the MCU wakes up, all the low power active timers are updated with the period spent in low power. When the timer goes off, the callback function is executed in non-interrupt context.

NOTE

If the timer is running when this function is called, it will be stopped and restarted. The MCU can enter into low power mode if there is no active timer, or if all active timers have low power capability.

3.3.8 TMR_StartIntervalTimer

Prototype

```
void TMR_StartIntervalTimer
  (
  tmrTimerID_t timerId,
  tmrTimeInMilliseconds_t timeInMilliseconds,
  void (*pfTmrCallBack)(tmrTimerID_t)
  );
```

Description

The function is a short version for TMR_StartTimer() function with one less parameter to pass. The timer will execute the callback function at the specified time interval and will not stop until TMR_StopTimer() is called.

NOTE

Only a millisecond timer can behave as an interval timer.

3.3.9 TMR_StartSingleShotTimer

Prototype

```
void TMR_StartSingleShotTimer
  (
  tmrTimerID_t timerId,
  tmrTimeInMilliseconds_t timeInMilliseconds,
  void (*pfTmrCallBack)(tmrTimerID_t)
  );
```

Description

This function starts a milliseconds single-shot timer. The callback is executed after the specified time interval. There is no need to call TMR_StopTimer() after the callback execution.

3.3.10 TMR_StartMinuteTimer

Prototype

```
void TMR_StartMinuteTimer
  (
  tmrTimerID_t timerId,
  tmrTimeInMinutes_t timeInMinutes,
  void (*pfTmrCallBack)(tmrTimerID_t)
  );
```

Description

This function starts a minutes single-shot timer. The callback is executed after the specified time interval. There is no need to call TMR_StopTimer() after the callback execution.

NOTE

There are no interval minute timers.

3.3.11 TMR_StartSecondTimer

Prototype

```
void TMR_StartSecondTimer
  (
  tmrTimerID_t timerId,
  tmrTimeInSeconds_t timeInSeconds,
  void (*pfTmrCallBack)(tmrTimerID_t)
  );
```

Description

This function starts a seconds single-shot timer. The callback is executed after the specified time interval. There is no need to call TMR_StopTimer() after the callback execution.

NOTE

There are no interval second timers.

3.3.12 TMR_StopTimer

Prototype

```
void TMR_StopTimer(tmrTimerID_t timerID);
```

Description

The function stops a running timer.

NOTE

If the timer has expired, but the callback function has not yet been called, this function will prevent the timer's callback from being executed. The function does not free the timer.

Chapter 4
LED

The LED component allows logical control of LEDs without the application understanding the physical interface to the LEDs.

4.1 LED Overview

By default, there are up to 8 LEDs controllable by the LED interface. If more LEDs are controlled, then the code in LED.c must be modified.

LEDs may be OR'ed together to perform an operation on more than one LED. For example, to turn on LEDs 1 and 3 and turn off all the rest, the following code sequence could be used:

```
LED_TurnOffAllLeds();
LED_TurnOnLed(LED1 | LED3);
```

`The LED.h` contains definitions for each LED, and it is easy to update for any particular hardware design. Each LED definition has the following 4 macros. Replace these with the appropriate equivalents for the hardware design.

```
#define Led1On()            (mLED_PORT1_c &= ~mLED1_PIN_c)
#define Led1Off()           (mLED_PORT1_c |= mLED1_PIN_c)
#define Led1Toggle()        (mLED_PORT1_c ^= mLED1_PIN_c)
#define GetLed1()           (~(mLED_PORT1_c & mLED1_PIN_c))
```

4.2 LED Properties

4.2.1 gLEDSupported_d

C Module Property	gLEDSupported_d
BeeKit Property Name	LED module enabled
BeeKit Component Containing BeeKit Property	PLM

LED support can be completely enabled or disabled by setting the gLEDSupported_d to TRUE or FALSE. The gLEDSupported_d property is set in BeeKit (LED module enabled from the PLM component property). If LED support is disabled, all of the interface functions and macros (e.g. LED_SetLed()) become empty macros, and the code is silently compiled out of an application.

4.2.2 gLEDBlipEnabled_d

C Module Property	gLEDBlipEnabled_d
BeeKit Property Name	LED blip enabled
BeeKit Component Containing BeeKit Property	PLM

Use the gLEDBlipEnabled_d property to enable the blip once code. The gLEDBlipEnabled_d property is set in BeeKit (LED blip enabled from the PLM component property). By default, flashing is turned on permanently on an LED until it is turned off, or set to solid on. The gLEDBlipEnabled_d property is disabled by default.

4.3 LED API

4.3.1 LED_TurnOffLed

Prototype

```
void LED_TurnOffLed(LED_t LEDNr);
```

Description

Turns off one or more LEDs. This is equivalent to LED_SetLed(LEDNr, gLedOff_c);

4.3.2 LED_TurnOnLed

Prototype

```
void LED_TurnOffLed(LED_t LEDNr);
```

Description

Turn on one or more LEDs. This is equivalent to LED_SetLed(LEDNr, gLedOn_c);

4.3.3 LED_ToggleLed

Prototype

```
void LED_ToggleLed(LED_t LEDNr);
```

Description

Toggle one or more LEDs. This is equivalent to LED_SetLed(LEDNr, gLedToggle_c);

4.3.4 LED_StartFlash

Prototype

```
void LED_StartFlash(LED_t LEDNr);
```

Description

Flash one or more LEDs. This is equivalent to LED_SetLed (LEDNr, gLedFlashing_c). When this function is called, the mLEDTimerID interval timer is started. Every time the interval timer expires, the LEDs toggle. To stop the flashing, call LED_StopFlash(LEDNr), LED_SetLed(LEDNr, gLedStopFlashing_c) or LED_StopFlashingAllLeds().

4.3.5 LED_StopFlash

Prototype

```
void LED_StopFlash(LED_t LEDNr);
```

Description

Stop flashing of one or more LEDs. When this function exits, the LED is turned off. If all LEDs are stopped, the interval timer (mLEDTimerID) is also stopped.

4.3.6 LED_StartSerialFlash

Prototype

```
void LED_StartSerialFlash(void);
```

Description

Serial flashing is the condition where the LEDs turn on in serial fashion, one after the other. All previous states of the LEDs are forgotten. When this function is called, the mLEDTimerID interval timer is started. To turn off serial flashing, use LED_SetLed (turn one or more LEDs on or off). When serial flashing is exited, all LEDs are set to off.

4.3.7 LED_TurnOffAllLeds

Prototype

```
void LED_TurnOffAllLeds(void);
```

Description

Turn off all LEDs.

4.3.8 LED_TurnOnAllLeds

Prototype

```
void LED_TurnOnAllLeds(void);
```

Description

Turn on all LEDs.

4.3.9 LED_StopFlashingAllLeds

Prototype

```
void LED_StopFlashingAllLeds(void);
```

Description

Turn off all LEDs and stop the mLEDTimerID interval timer.

4.3.10 LED_SetLed

Prototype

```
void LED_SetLed(LED_t LEDNr,LedState_t state);
```

Description

Set one or more LEDs to a particular state. The states include

- gLedFlashing_ c — flash at a fixed rate
- gLedBlip_c — (optional) just like flashing, but blinks only once
- gLedOn_c — on solid
- gLedOff_c — off solid
- gLedToggle_c — toggle to on or off

4.3.11 LED_SetHex

Prototype

```
void LED_SetHex(uint8_t hexValue);
```

Description

Display a hex nibble (0x0 – 0xf) on a 4 LED display. The number is expressed in binary with LED1 representing the highest bit and LED4 representing the lowest bit. Turns off any flashing or serial lights.

Chapter 5
Display

The display interface allows easy access to text oriented LCDs.

5.1 Display Overview

The Freescale NCB, QE128 EVB, and 1323x-RCM boards support an LCD display. The display has two lines with 16 characters per line. For all the display functions, if the length of the string is longer than the length of the LCD display, the string is truncated.

Support for the LCD display can be found in the `Display.h`, `Display.`c, and `LCD.c` files.

5.2 Display Properties

5.2.1 gLCDSupported_d

C Module Property	gLCDSupported_d
BeeKit Property Name	Display module enabled
BeeKit Component Containing BeeKit Property	PLM

The gLCDSupported_d enables or disables LCD support. The gLCDSupported_d property is set in BeeKit (Display module enabled from the PLM component property). If support is disabled, the LCD_ functions all become empty macros, so the same C source code can compile for boards with or without an LCD display.

5.3 Display API for MC1321x, QE128, and MC1322x Boards

5.3.1 LCD_ClearDisplay

Prototype

```
void LCD_ClearDisplay ( void );
```

Description

Clears all lines on the display.

5.3.2 LCD_Init

Prototype

```
void LCD_Init( void );
```

Description

Initializes the display. Called once by the application on startup.

5.3.3 LCD_WriteBytes

Prototype

```
void LCD_WriteBytes
  (
  uint8_t   *pstr,/* IN: pointer to the label to print with the bytes */
  uint8_t   *value,  /* IN: The bytes to print. */
  uint8_t   line, /* IN: The line in the LCD */
  uint8_t   length/* IN: number of bytes to print in the LCD */
  );
```

Description

Write a label followed by a set of bytes to the LCD display. The number of bytes to display are listed by the length parameter. The line parameter must be 1 or 2.

5.3.4 LCD_WriteString

Prototype

```
void LCD_WriteString
  (
  uint8_t line,/* IN: Line in display */
  uint8_t *pStr/* IN: Pointer to text string */
  );
```

Description

Displays a string on one LCD line. The line must be 1 or 2.

5.3.5 LCD_WriteStringValue

Prototype

```
void LCD_WriteStringValue
  (
  uint8_t *pstr,  /* IN: Pointer to text string */
  uint16_t value, /* IN: Value */
  uint8_t line,   /* IN: Line in display. */
  LCD_t numberFormat/* IN: Value to show in HEX or DEC */
  );
```

Description

Writes a value to the display, in either hex or decimal. The label is printed first, then the number. The format must be one of

- gLCD_HexFormat_c
- gLCD_DecFormat_c

The line must be 1 or 2.

5.4 Display API for 1323x-RCM

CAUTION

Contents in this section are preliminary and may change before the MC1323x platform public release.

The display interface for 1323x-RCM uses a callback function parameter which returns when the operation has completed. Users should wait for the callback to return before performing a new display operation.

5.4.1 LCD_Init

Prototype

```
void LCD_Init( void (*pfCallBack)(bool_t status) );
```

Description

Initializes the display. Called once by the application on startup.

5.4.2 LCD_ClearDisplay

Prototype

```
void LCD_ClearDisplay ( void (*pfCallBack)(bool_t status) );
```

Description

Clears all lines on the display.

5.4.3 LCD_WriteString

Prototype

```
void LCD_WriteString
  (
  uint8_t line,/* IN: Line in display */
  uint8_t *pStr/* IN: Pointer to text string */
void (*pfCallBack)(bool_t status) /* IN: Callback function called when operation completes */
  );
```

Description

Displays a string on one LCD line. The line must be 0 or 1.

5.4.4 LCD_WriteString

Prototype

```
void LCD_WriteString
  (
  uint8_t line,/* IN: Line in display */
  uint8_t *pStr/* IN: Pointer to text string */
void (*pfCallBack)(bool_t status) /* IN: Callback function called when operation completes */
  );
```

Description

Displays a string on one LCD line. The line must be 0 or 1.

Chapter 6 Keyboard

The keyboard interface allows each access to key (or switch) input.

6.1 Keyboard Overview

Keyboard input is given to the application in the form of a callback function, which defaults to BeeAppHandleKeys(). When a key is pressed, the application will receive a keyboard event in the callback. The event will be one of the following for MC1321x, MC1322x, or QE128 boards:

- gKBD_EventSW1_c
- gKBD_EventSW2_c
- gKBD_EventSW3_c
- gKBD_EventSW4_c
- gKBD_EventLongSW1_c
- gKBD_EventLongSW2_c
- gKBD_EventLongSW3_c
- gKBD_EventLongSW4_c

Do not confuse keyboard events with task event bits. Keyboard events are not a bit mask, but a value, and arrive one at a time. Task events are a 16-bit bit mask and may contain multiple events on each call to the task handler.

The keyboard handler gains control only if the application initialized the keyboard driver, which registers the callback. Typical initialization code follows

```
KBD_Init(BeeAppHandleKeys);
```

Keyboard support is found in the `Keyboard.c` and `Keyboard.h` files.

6.2 Keyboard Properties

6.2.1 gKeyBoardSupported_d

C Module Property	gKeyBoardSupported_d
BeeKit Property Name	Keyboard module enabled
BeeKit Component Containing BeeKit Property	PLM

The property gKeyBoardSupported_d enables or disables keyboard support. The gKeyBoardSupported_d property is set in BeeKit (Keyboard module enabled from the PLM component property). If disabled, the keyboard functions continue to exist in the form of empty macros so the same code can be used for boards that either support or do not support a keyboard.

6.2.2 gKeyScanInterval_c

C Module Property	gKeyScanInterval_c
BeeKit Property Name	Key Scan Interval

BeeKit Component Containing BeeKit Property PLM

The gKeyScanInterval_c property sets the time the key must be continuously pressed to register the keypress. This scan interval must be long enough to be past the debounce period typically switches require. The default is 50 milliseconds. The gKeyScanInterval_c property is set in BeeKit (Key Scan Interval from the PLM component property).

6.2.3 gLongKeyIterations_c

C Module Property	gLongKeyIterations_c
BeeKit Property Name	Key Press Duration

BeeKit Component Containing BeeKit Property PLM

The gLongKeyIterations_c property sets the number of iterations of the scan interval that means a long key was pressed. This allows a single switch to be used for 2 keys, short and long. The default is 20 iterations, or 1 second of time (20 * 50ms). The gLongKeyIterations_c property is set in BeeKit (Long key press duration from the PLM component property).

6.3 Keyboard API for MC1321x, QE128, and MC1322x Boards

6.3.1 KBD_Init

Prototype

```
void KBD_Init
  (
  KBDFunction_t pfCallBackAdr /* IN: Pointer to callback function */
  );
```

Description

The only interaction with the keyboard is through the callback, given to the KBD_Init() function. The callback must have the following prototype (although the name can be anything the appropriate for the application):

```
void BeeAppHandleKeys
  (
  key_event_t keyEvent  /*IN: Events from keyboard module */
  );
```

The callback is in the keyboard task context. Usually the keyboard action is carried out in the callback.

6.4 Keyboard API for MC1323x Boards

CAUTION

Contents in this section are preliminary and may change before the MC1323x platform release.

The matrix keyboard of the remote control style 1323x development boards requires a new callback function prototype for KBD_Init as described in the following section.

6.4.1 KBD_Init

Prototype

```
void KBD_Init
  (
  KBDFunction_t pfCallBackAdr /* IN: Pointer to callback function */
  );
```

Description

The only interaction with the keyboard is through the callback given to the KBD_Init() function. The callback must have the following prototype (although the name can be anything appropriate for the application):

```
void BeeAppHandleKeys
(
uint8_t events, /*IN: Events from keyboard module: gKBD_Event_c, gKBD_EventLong_c*/
uint8_t pressedKey /*IN: Pressed Key SW ID */
);
```

Chapter 7
UART

The UART driver allows easy access to the serial port found on the Freescale ZigBee development boards.

7.1 UART Overview

On some boards, such as the EVB, NCB, and QE128 EVB, two serial ports are supported. One is a 9-pin traditional serial port (UART1) and the other is a USB serial port (UART2). The Axiom board supports two traditional 9-pin serial ports. Both UARTs may be used simultaneously only if Uart1_ or Uart2_ routines are called. For example, Uart1_Transmit(). The SARD board supports a single 9-pin serial port (UART1). The SRB board supports a single USB serial port (UART2).

The ZigBee Test Client (ZTC), a debugging tool, may be used in some applications. The ZTC defaults to UART1 on the SARD and Axiom boards and to UART2 on the EVB, NCB, SRB, and QE128 EVB boards.

The UART must be initialized before it is used. The typical code sequence to initialize the UART is as follows:

```
UartX_SetRxCallBack(AppUartRxCallBack);
UartX_SetBaud(gUartDefaultBaud_c);
```

The baud rate must be set to one of the following values:

- gUARTBaudRate1200_c
- gUARTBaudRate2400_c
- gUARTBaudRate4800_c
- gUARTBaudRate9600_c
- gUARTBaudRate19200_c
- gUARTBaudRate38400_c

The UartX_SetBaud() function is not necessary if using the default of 38,400 baud, as that is the default baud rate. The rest of the serial settings are always 8 data bits, no stop bit, 1 parity bit (8N1), as defined in uart.c.

The UartX_ functions go to the default UART, as defined by gUart_PortDefault_d. To use an explicit UART, use Uart1_ or Uart2_ routines, for example, Uart1_SetRxCallBack().

When the callback is received, call the function UartX_GetByteFromRxBuffer() to retrieve bytes from the UART driver.

To send data, use UartX_Transmit(). The UartX_Transmit() function includes an optional callback to indicate when the data has been sent.

7.2 UART Properties

7.2.1 gUart1_Enabled_d

C Module Property	gUart1_Enabled_d
BeeKit Property Name	Enable the UART on SCI port 1
BeeKit Component Containing BeeKit Property	PLM

The gUart1_Enabled_d property enables or disables UART1. If this is disabled (FALSE), the Uart1_ functions are stubbed. The gUart1_Enabled_d property is set in BeeKit (Enable the UART on SCI port 1 from the PLM component property).

7.2.2 gUart2_Enabled_d

C Module Property	gUart2_Enabled_d
BeeKit Property Name	Enable the UART on SCI port 2
BeeKit Component Containing BeeKit Property	PLM

The gUart2_Enabled_d property enables or disables UART2. If this is disabled (FALSE), the Uart2_ functions are stubbed. The gUart2_Enabled_d property is set in BeeKit (Enable the UART on SCI port 2 from the PLM component property).

7.2.3 gUart_PortDefault_d

The gUart_PortDefault_d property sets a default port, so the UartX_ macros refer to the proper functions.

7.2.4 gUart_TransmitBuffers_c

C Module Property	gUart_TransmitBuffers_c
BeeKit Property Name	UART Transmit Buffers
BeeKit Component Containing BeeKit Property	PLM

Set gUart_TransmitBuffers_c to the number of simultaneous transmits expected in the application. This defaults to 3. The gUart_TransmitBuffers_c property is set in BeeKit (UART Transmit Buffers from the PLM component property).

7.2.5 gUart_ReceiveBufferSize_c

C Module Property	gUart_ReceiveBufferSize_c
BeeKit Property Name	UART Receive Buffer Size

BeeKit Component Containing BeeKit Property PLM

Set gUart_ReceiveBufferSize_c to the size of the largest expected packet plus 10% extra. This defaults to 32 bytes. The gUart_ReceiveBufferSize_c property is set in BeeKit (UART Receive Buffer Size from the PLM component property).

7.2.6 gUart_RxFlowControlSkew_d

C Module Property	gUart_RxFlowControlSkew_d
BeeKit Property Name	UART Rx Flow Control Skew

BeeKit Component Containing BeeKit Property PLM

The UART driver uses flow control to prevent a host PC or processor from sending data too quickly. gUart_RxFlowControlSkew_d defaults to 8. The gUart_RxFlowControlSkew_d property is set in BeeKit (UART Rx Flow Control Skew from the PLM component property).

7.2.7 gUart_RxFlowControlResume_d

C Module Property	gUart_RxFlowControlResume_d
BeeKit Property Name	UART Rx Flow Control Resume

BeeKit Component Containing BeeKit Property PLM

The UART driver will resume (remove hardware flow control) after the receive buffer empties. Set gUart_RxFlowControlResume_d to the number of bytes in the receive buffer before receiving resumes. The gUart_RxFlowControlResume_d property is set in BeeKit (UART Rx Flow Control Resume from the PLM component property).

7.3 UART API

7.3.1 Uart_ClearErrors

Prototype

```
void Uart_ClearErrors(void);
```

Description

Uart_ClearErrors() clears any errors on the serial line.

7.3.2 UartX_SetBaud

Prototype

```
void UartX_SetBaud(UartBaudRate_t baudRate);
void Uart1_SetBaud(UartBaudRate_t baudRate);
void Uart2_SetBaud(UartBaudRate_t baudRate);
```

Description

Sets the baud rate for the respective UART (default, UART1 or UART2). This can be called at any time.

For the rest of the functions, the prototype will be for UartX_ functions only, but they all apply to both Uart1_ and Uart2_ functions as well.

7.3.3 UartX_Transmit

Prototype

```
bool_t UartX_Transmit
(
unsigned char const *pBuf,
index_t bufLen,
void (*pfCallBack)(unsigned char const *pBuf)
);
```

Description

Transmit data output on the UART. The data is transmitted in place in the buffer, so the contents of pBuf must exist until the callback is issued. If the buffer was allocated as a message buffer, it can be freed when the callback is issued. The callback is made in the context of the UART task.

Returns FALSE if the UART transmit queue is full. Returns TRUE if the transmit buffer has been accepted and is now scheduled to be sent out the UART.

Transmitting data out the UART is interrupt driven, so this function always returns immediately.

7.3.4 UartX_IsTxActive

Prototype

```
bool_t UartX_IsTxActive(void);
```

Description

Returns TRUE if the UART is still actively transmitting data.

7.3.5 UartX_SetRxCallBack

Prototype

```
void UartX_SetRxCallBack(void (*pfCallBack)(void));
```

Description

This function sets up the application for receiving data on from the UART. If this function has not been set up and data is received on the UART, the data is discarded. Once the callback has been called, use UartX_GetByteFromRxBuffer() to retrieve the data.

7.3.6 UartX_GetByteFromRxBuffer

Prototype

```
bool_t UartX_GetByteFromRxBuffer(unsigned char *pDst);
```

Description

Retrieves one byte from the UART buffer.

7.3.7 UartX_UngetByte

Prototype

```
void UartX_UngetByte(unsigned char byte);
```

Description

This function will put a byte back on the UART's receive buffer. Similar to the ANSI C library function ungetch(). This can be useful when decoding a serial protocol.

Chapter 8
Non-Volatile Memory

Non-Volatile Memory (NVM) provides a permanent storage mechanism that can survive system resets and power outages.

8.1 Non-Volatile Memory Overview

The NVM system uses some of the HCS08 flash memory for storage. Three 512 up to 4096 byte pages are reserved for storage, 2 of which are in use at any given time with one spare for writing new data.

NVM must be managed carefully. The data sheet for the HCS08 lists:

- Up to 100,000 program/erase cycles at typical voltage and temperature

Given a 20 year life span for a product, the flash memory should not be written to more than once every 1.8 hours on average. To manage this, the NVM component provides three interfaces to list NVM data as "dirty", that is changed or needing to be written.

- NvSaveOnIdle()
- NvSaveOnInterval()
- NvSaveOnCount()

Use the NvSaveOnIdle() to save immediately (when the idle task next gains control). Use NvSaveOnInterval() to save after a period of time as defined by gNvMinimumTicksBetweenSaves_c. Use NvSaveOnCount() to save after a certain number of calls to NvSaveOnCount(). Whichever of the three method saves first will reset the others.

In BeeStack, NvSaveOnIdle() is used just after the node joins the network and has retrieved the network security key. This way, if the node is reset it will still be on the network. NvSaveOnInterval() is used when new routes or neighbors are discovered. NvSaveOnCount() is used in a secure network upon every message sent or received to update the security counters and store them only once every 256 messages. This prevents a node from saving too often and cause the flash to fail.

It is up to the application designer to decide which of these three methods (or combination thereof) is appropriate for application data.

Data items to be stored in NVM are grouped together to fit in a flash page. This group is termed a data set, and is a collection of pointers and sizes of items or structures to store into NVM. There are currently two data sets defined in BeeStack:

- gNvDataSet_Nwk_ID_c – the network data set
- gNvDataSet_App_ID_c – the application data set

These data sets are defined in `NV_Data.c`. The application data set may be modified to save application data. For BeeStack, the network data set should not be modified.

WARNING

The compiler cannot tell if the data set has exceeded the maximum length of 504 bytes. It is up to the application to ensure the data set fits in the flash page.

All the functions listed in the API section below may be called by the application at any time (except within an interrupt handler). Use NvSaveOnIdle() to save immediately to flash.

WARNING

Do NOT call any NV storage functions except those listed in the API section below. Calling on the NvSaveDataSet() function directly can cause the system to hang.

8.2 Non-Volatile Memory Properties

8.2.1 gNvStorageIncluded_d

C Module Property	gNvStorageIncluded_d
BeeKit Property Name	NVM Storage Enabled
BeeKit Component Containing BeeKit Property	PLM

Set gNvStorageIncluded_d to FALSE to disable NVM storage. All the NvXxx() functions become stubs if disabled. The gNvStorageIncluded_d property can be set in BeeKit (NVM storage enabled from the PLM component property).

8.2.2 gNvNumberOfDataSets_c

Set gNvNumberOfDataSets_c to define the number of data sets. Changing the number of data sets is a fairly tricky operation. In addition to changing this define, the data sets in `NV_Data.c` must also be changed, and the linker file, `BeeStack.prm` or `BeeStack_QE128_far_banked.prm` (PLM folder) must also be changed to reflect the new size of the flash storage area. Comments in the `NV_Data.c` file and the linker file are included to aid in changing the number of data sets. The default number of data sets is two (2).

8.2.3 gNvMinimumTicksBetweenSaves_c

This determines how many ticks must occur between when the data set is first marked dirty before it is saved. Each tick is 1 second. The total time may span up to 4 minutes (4 * 60 = 240 ticks).

8.2.4 gNvCountsBetweenSaves_c

The property gNvCountsBetweenSaves_c determines the number of counts between saving if calling only the NvSaveOnCount() function.

8.3 Non-Volatile Memory API

8.3.1 NvSaveOnIdle

Prototype

```
void NvSaveOnIdle(NvDataSetID_t dataSetID);
```

Description

Mark the data set as dirty and save it at the next possible moment. This will happen when the idle task is next called. The system must be idle for NVM to save data. The radio is disabled during the period the flash is getting updated.

8.3.2 NvSaveOnInterval

Prototype

```
void NvSaveOnInterval(NvDataSetID_t dataSetID);
```

Description

Mark the data set as dirty to be saved after the next interval as specified by the compile-time gNvMinimumTicksBetweenSaves_c. This defaults to 4 seconds.

8.3.3 NvSaveOnCount

Prototype

```
void NvSaveOnCount(NvDataSetID_t dataSetID);
```

Description

Mark the data set as dirty and save when the count reaches the value specified by the property gNvCountsBetweenSaves_c.

8.3.4 NvIsDataSetDirty

Prototype

```
bool_t NvIsDataSetDirty(NvDataSetID_t dataSetID);
```

Description

Returns TRUE if the data set is dirty and needs to be saved.

8.3.5 NvRestoreDataSet

Prototype

```
bool_t NvRestoreDataSet(NvDataSetID_t dataSetID);
```

Description

Restores a data set from NVM. This is used by ZDO when using a ZDO function to restore the state of the network from NVM. This can also be used by the application to restore the data any time that is appropriate.

If the data set is dirty at the time of the restore, the changed data will be discarded and the data that was last stored in NVM will be placed into the RAM structures as defined by the data set.

8.3.6 NvClearCriticalSection

Prototype

```
void NvClearCriticalSection(void);
```

Description

Use to clear a NVM critical section as set by NvSetCriticalSection().

8.3.7 NvSetCriticalSection

Prototype

```
void NvSetCriticalSection(void);
```

Description

Use this to indicate to the NVM engine not to save while in a critical section. This is a counting semaphore, so for each NvSetCriticalSection(), exactly one NvClearCriticalSection() should be called.

Critical sections can be used to make sure changes to the data set are atomic across fields in the data set, or if the application or stack cannot be interrupted by an NVM save (which may take several milliseconds).

Chapter 9
Low Power Library

The low power module (LPM) simplifies the process of putting an HCS08/MC1322X node into low power or sleep modes to conserve battery life. Only ZigBee End-Devices (ZEDs) can sleep. ZigBee Coordinators and Routers must remain awake to route packets on behalf of other nodes.

9.1 Low Power Library Overview

The files `PWR.c`, `PWRLib.h`, `PWRLib.c`, `PWR_Configuration.h` and `PWR_Interface.h` comprise the low-power library. All configuration is done at compile-time through properties located in `PWR_Configuration.h`.

The low power library is initialized automatically by BeeStack and by the 802.15.4 MAC Codebase. If the gLpmIncluded_d property is selected, low power is enabled whenever the node is idle (no timers or events pending). The application can also elect to stay awake by using the PWR_DisallowDeviceToSleep() function.

NOTE

With the exception of PWR_DisallowDeviceToSleep() and PWR_AllowDeviceToSleep(), do not call the low power functions directly in the application. The idle task already contains the proper code for entering deep or light sleep as appropriate, in a way that coordinates with the entire system.

9.2 Low Power Library Properties

The following list is not an exhaustive list of properties, but includes the most important ones. For the entire list, see `PWR_Configuration.h`.

9.2.1 gRxOnWhenIdle_d/gLpmIncluded_d

Set gRxOnWhenIdle_d to TRUE in BeeStack to disable low power mode. Set gLpmIncluded_d to FALSE in 802.15.4 MAC Codebase to disable low power mode. All the low-power code will be compiled out if gRxOnWhenIdle is TRUE or if gLpmIncluded_d is FALSE. Low power can be disabled at run-time using PWR_DisallowDeviceToSleep() from the application, however this will not save code space.

Set gRxOnWhenIdle_d to FALSE or gLpmIncluded_d to TRUE to enable low lower in BeeStack ZigBee End-Devices (ZEDs) or in 802.15.4 MAC Codebase. The rest of the low-power properties only have meaning if low power is enabled.

gRxOnWhenIdle_d/gLpmIncluded_d can be found in `ApplicationConf.h`, and is a configurable property in BeeKit.

9.2.2 cPWR_DeepSleepMode

Set cPWR_DeepSleepMode to the appropriate deep sleeping mode for the application. The possible sleep modes for HCS08 are:

- Mode 1 - Ext. KBI int wake up. Wake on keyboard interrupt only.
- Mode 2 - RTI timer wake up +-30%. Note that the RTI clock can be calibrated to be accurate within 1% at a given temperature, but the calibration procedure is outside the scope of this manual.
- Mode 3 - Ext. KBI int and RTI timer wake up +-30%, radio in OFF/reset mode. This mode can wake from either a keyboard interrupt or the timer, whichever comes first. The time should be sufficiently short for application timing purposes, because if the MCU is woken by the keyboard interrupt, it cannot tell how much time has passed, and so will assume the same amount of time as specified by the cPWR_RTITickTime property. This will wake up slowly (approximately 1ms) . This mode uses less power than mode 4 because the radio is off.
- Mode 4 - Ext. KBI int and RTI timer wake up +-30%, radio in hibernate - not reset. This mode can wake faster than mode 3, but saves less power. It is useful for shorter sleep times.
- Mode 5 - Radio in acoma/doze, supplying 62.5kHz clock to MCU, MCU in STOP3, RTI wake up from ext clock 512mS (max avail with ext 62.5khz). This mode is useful because it is the only mode that allows the BDM to continue use after low is entered. All the other modes (1-4) cause the BDM to be disabled.

The deep sleep mode defaults to Mode 4.

The possible sleep modes for MC1322X are:

- Mode 1 - CPU sleep mode is Hibernate, wake-up on KBI(gKeyBoardSupported_d).
- Mode 2 - CPU sleep mode is Hibernate, wake-up on Wake up Timer.
- Mode 3 - CPU Sleep mode is Hibernate, wake-up on either KBI(gKeyBoardSupported_d) or Wake up Timer.
- Mode 4 - CPU sleep mode is Hibernate, wake-up on Reset.
- Mode 5 - CPU sleep mode is Doze, wake-up on KBI(gKeyBoardSupported_d).
- Mode 6 - CPU sleep mode is Doze, wake-up on Wake up Timer.
- Mode 7 - CPU sleep mode is Doze, wake-up on either KBI(gKeyBoardSupported_d) or Wake up Timer.

To configure the RAM retention mode used during sleep for MC1322X cPWR_RAMRetentionMode can be set to one of the following values:

- gRamRet8k_c to retain first 8k of RAM during sleep.
- gRamRet32k_c to retain first 32k of RAM during sleep.
- gRamRet64k_c to retain first 64k of RAM during sleep.
- gRamRet96k_c to retain first 96k (All) of RAM during sleep.

NOTE

gRamRet96k_c is the default retention mode.

9.2.3 cPWR_SleepMode

Leave the cPWR_SleepMode as TRUE (1). This option saves considerable power when the system is running. It works by entering an MCU halt when the system enters the idle task and will wake instantly when an interrupt occurs (UART, keyboard, timer expires, etc…) This can save 30% of power at run-time. The radio enters acoma/doze mode. The clock on the radio is still running but the receiver is disabled.

C Module Property	cPWR_DeepSleepMode
BeeKit Property Name	Deep Sleep Wake up
BeeKit Component Containing BeeKit Property	PLM

C Module Property	cPWR_SleepMode
BeeKit Property Name	Deep Sleep Handling
BeeKit Component Containing BeeKit Property	PLM

C Module Property	cPWR_RTITickTime
BeeKit Property Name	Distance Between RTI Interrupts
BeeKit Component Containing BeeKit Property	PLM

9.3 Low Power Library API

9.3.1 PWR_CheckIfDeviceCanGoToSleep

Prototype

```
bool_t PWR_CheckIfDeviceCanGoToSleep( void );
```

Description

Checks the flag and ensure it is allowed to go to low power at this time. Always ensure that this returns TRUE before calling PWR_EnterLowPower().

9.3.2 PWR_EnterLowPower

Prototype

```
void PWR_EnterLowPower(void);
```

Description

Enters low power mode (either deep or light sleep, depending on whether any timers are active).

9.3.3 PWR_DisallowDeviceToSleep

Prototype

```
void PWR_DisallowDeviceToSleep (void);
```

Description

Sets a critical section to prevent the system from entering low power. Use this if the device must stay awake, for example, during ZigBee commissioning or during a sensor reading.

9.3.4 PWR_AllowDeviceToSleep

Prototype

```
void PWR_AllowDeviceToSleep (void);
```

Description

Clears the critical section set by PWR_DisallowDeviceToSleep(). This is a counting semaphore. There should be one call to PWR_AllowDeviceToSleep() for every call to PWR_DisallowDeviceToSleep().

Chapter 10
Touch Pad

The touch pad interface allows easy access to touch pad devices.

10.1 Touch Pad Overview

The Freescale 1323x-RCM boards support a touch pad device. This device provides information about finger position coordinates relative to a default position, finger width values, finger contact as a measure of finger pressure, gesture recognition (pinch, press, flick, tap) or rotation.

Support for the touch pad device is provided in the `Touchpad.c` and `Touchpad.h` source files.

10.2 Touch Pad Properties

10.2.1 gTouchpadIncluded_d

C Module Property	gTouchpadIncluded_d
BeeKit Property Name	Touchpad module enabled
BeeKit Component Containing BeeKit Property	PLM

The gTouchpadIncluded_d enables or disables touch pad support. The gTouchpadIncluded_d property is set in BeeKit (Touchpad module enabled from the PLM component property). If support is disabled, the TP_ functions all become empty macros, so the same C source code can compile for boards with or without a touch pad device.

10.3 Touch Pad API for 1323x-RCM Boards

The touch pad API is declared in the `Touchpad_Interface.h` source file.

10.3.1 TP_Init

Prototype

```
statusCode_t TP_Init(void);
```

Description

Initialize the touchpad sensor, IIC communication, global variables

10.3.2 TP_Uninit

Prototype

```
void TP_Uninit(void);
```

Description

Un-initializes the touchpad sensor, IIC communication, de-allocates the memory for global variables.

10.3.3 TP_DevRestart

Prototype

```
statusCode_t TP_DevRestart(void);
```

Description

Resets the touchpad sensor.

10.3.4 TP_GetDevInfo

Prototype

```
statusCode_t TP_GetDevInfo(void);
```

Description

Collects the information about the device configuration.

10.3.5 TP_GetDevConfig

Prototype

```
statusCode_t TP_GetDevConfig(void);
```

Description

Collects information about gesture parameters.

10.3.6 TP_SetDevConfig

Prototype

```
statusCode_t TP_SetDevConfig(devConfig_t mDevConfig);
```

Description

Configures the list of gestures to be identified together with their associated parameters.

10.3.7 TP_SetCallback

Prototype

```
statusCode_t TP_SetCallback(tpCallback_t pfTpCallback);
```

Description

Provides a pointer to the callback function for any touchpad event.

Chapter 11
Accelerometer

The accelerometer interface allows easy access to accelerometer sensor devices.

11.1 Accelerometer Overview

The Freescale 1323x-RCM boards have an embedded accelerometer device. This device provides orientation detection. Tilt orientation is in three dimensions and is identified in its last known static position. This enables a product to set its display orientation appropriately to either portrait/landscape mode, or to turn off the display if the product is placed upside down. The sensor provides six different positions including: Left, Right, Up, Down, Back, and Front.

Support for the accelerometer device is provided in the `Accelerometer.c` and `Accelerometer.h` source files.

11.2 Accelerometer Properties

11.2.1 gAccelerometerSupported_d

C Module Property	gAccelerometerSupported_d
BeeKit Property Name	Accelerometer module enabled
BeeKit Component Containing BeeKit Property	PLM

The gAccelerometerSupported_d enables or disables accelerometer support. The gAccelerometerSupported_d property is set in BeeKit (Accelerometer module enabled from the PLM component property). If support is disabled, the ACC_ functions all become empty macros, so the same C source code can compile for boards with or without an accelerometer device.

11.3 Accelerometer API for 1323x-RCM Boards

The accelerometer API is declared in the `ACC_Interface.h` source file.

11.3.1 ACC_Init

Prototype

```
bool_t ACC_Init (ACCFunction_t pfCallBackAdr);
```

Description

This function initializes the accelerometer module.

11.3.2 ACC_SetPowerMode

Prototype

```
bool_t ACC_SetPowerMode(gAccPowerMode_t accPowerMode);
```

Description

This function sets the power mode of the accelerometer module.

11.3.3 ACC_SetEvents

Prototype

```
bool_t ACC_SetEvents(uint16_t maskAccEvents);
```

Description

This function sets what events will be generated from ACC module.

11.3.4 ACC_SetSamplesPerSecond

Prototype

```
bool_t ACC_SetSamplesPerSecond(gAccSamplesPerSecondSetup_t accSamplesPerSecondSetup);
```

Description

This function sets the number of samples per second.

11.3.5 ACC_SetTapPulseDetection

Prototype

```
bool_t ACC_SetTapPulseDetection(gAccTapSetup_t accTapSetup);
```

Description

This function sets the tap axes enable/disable and the number of threshold counts.

11.3.6 ACC_SetTapPulseDebounceCounter

Prototype

```
bool_t ACC_SetTapPulseDebounceCounter(uint8_t accTapDebounceCount);
```

Description

This function sets the tap de-bounce counter.

Chapter 12
I^{2}C

The I^{2}C (or IIC) interface allows easy access to the Inter-Integrated Circuit communication interface of the microcontroller units on the Freescale 802.15.4 compliant platforms.

12.1 IIC Overview

The inter-IC (IIC or I^{2}C) bus is a two-wire—serial data (SDA) and serial clock (SCL)— bidirectional serial bus that provides a simple efficient method of data exchange between the system and other devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The two-wire bus minimizes the interconnections between devices. The synchronous, multi-master bus of the I2C allows the connection of additional devices to the bus for expansion and system development. The bus includes collision detection and arbitration that prevent data corruption if two or more masters attempt to control the bus simultaneously.

Support for the IIC module is provided in the `IIC.c` and `IIC.h` source files.

12.2 IIC Properties for HCS08 MCU-based Platforms

12.2.1 gIIC_Enabled_d

C Module Property	gIIC_Enabled_d
BeeKit Property Name	IIC module enabled

BeeKit Component Containing BeeKit Property PLM

The gIIC_Enabled_d enables or disables IIC support. The gIIC_Enabled_d property is set in BeeKit (IIC module enabled from the PLM component property). If support is disabled, the IIC_ functions all become empty macros, so the same C source code can compile for MCUs with or without an IIC module.

12.2.2 gIIC_DefaultBaudRate_c

C Module Property	gIIC_DefaultBaudRate_c
BeeKit Property Name	IIC default Baud Rate

BeeKit Component Containing BeeKit Property PLM

This property defines the Baud Rate used for sending and receiving data over IIC.

12.2.3 gIIC_SlaveTransmitBuffersNo_c

C Module Property	gIIC_SlaveTransmitBuffersNo_c
BeeKit Property Name	IIC slave transmit buffers
BeeKit Component Containing BeeKit Property	PLM

The number of entries in the transmit-buffers-in-waiting list when the IIC is used in slave mode.

12.2.4 gIIC_SlaveReceiveBufferSize_c

C Module Property	gIIC_SlaveReceiveBufferSize_c
BeeKit Property Name	IIC slave receive buffer size
BeeKit Component Containing BeeKit Property	PLM

Size of the driver's Rx circular buffer when used in slave mode. This buffer is used to hold the received bytes until the application can retrieve them via the IIC_GetByteFromRxBuffer() function, and are not otherwise accessible from outside the driver.

12.2.5 gIIC_DefaultSlaveAddress_c

C Module Property	gIIC_DefaultSlaveAddress_c
BeeKit Property Name	IIC slave address
BeeKit Component Containing BeeKit Property	PLM

Address of the IIC module when used in slave mode.

12.2.6 gIIC_Slave_TxDataAvailableSignal_Enable_c

C Module Property	gIIC_Slave_TxDataAvailableSignal_Enable_c
BeeKit Property Name	IIC in slave mode signals when data is available to send
BeeKit Component Containing BeeKit Property	PLM

This property is applicable only when the IIC module is used in slave mode. Specifies whether or not the host processor will be signaled when the local IIC module has information to send. After being signaled, the host processor should address the local IIC module as a slave device and read the available information from. In the case when this property is enabled, the host processor is signaled through a GPIO that can be configured with the help of the properties below.

12.2.7 gIIC_TxDataAvailablePortDataReg_c

C Module Property	gIIC_TxDataAvailablePortDataReg_c
BeeKit Property Name	Data port to signal when data is available to send

BeeKit Component Containing BeeKit Property PLM

This property is applicable only when the IIC module is used in slave mode and the property that enables the host to be signaled when data is available to be sent by the local IIC module is activated. Specifies the local data port used for connecting the signal line between the local processor and the host processor.

12.2.8 gIIC_TxDataAvailablePortDDirReg_c

C Module Property	gIIC_TxDataAvailablePortDDirReg_c
BeeKit Property Name	Direction port to signal when data is available to send

BeeKit Component Containing BeeKit Property PLM

This property is applicable only when the IIC module is used in slave mode and the property that enables the host to be signaled when data is available to be sent by the local IIC module is activated. Specifies the local data direction port used for connecting the signal line between the local processor and the host processor.

12.2.9 gIIC_TxDataAvailablePinMask_c

C Module Property	gIIC_TxDataAvailablePinMask_c
BeeKit Property Name	Port pin mask to signal when data is available to send

BeeKit Component Containing BeeKit Property PLM

This property is applicable only when the IIC module is used in slave mode and the property that enables the host to be signaled when data is available to be sent by the local IIC module is activated. Specifies the mask of the pin on the local port used for connecting the signal line between the local processor and the host processor.

12.3 IIC API for HCS08 MCU-based Platforms

The IIC API is declared in the IIC_Interface.h source file.

12.3.1 IIC_Init

Prototype

```
void     IIC_ModuleInit(void);
```

Description

This function initializes the IIC module.

12.3.2 IIC_ModuleUninit

Prototype

```
void     IIC_ModuleUninit(void);
```

Description

This function un-initializes the IIC module.

12.3.3 IIC_SetBaudRate

Prototype

```
bool_t IIC_SetBaudRate(uint8_t baudRate);;
```

Description

This function sets IIC module Baud Rate.

12.3.4 IIC_SetSlaveAddress

Prototype

```
bool_t   IIC_SetSlaveAddress(uint8_t slaveAddress);
```

Description

This function sets the IIC module slave address.

12.3.5 IIC_BusRecovery

Prototype

```
void IIC_BusRecovery(void);
```

Description

This function tries to set free the SDA line if another IIC device keeps it low.

12.3.6 IIC_SetSlaveRxCallBack

Prototype

```
void     IIC_SetSlaveRxCallBack(void (*pfCallBack)(void));
```

Description

This function sets the slave receive side callback function.

12.3.7 IIC_GetByteFromRxBuffer

Prototype

```
bool_t   IIC_GetByteFromRxBuffer(unsigned char *pDst);
```

Description

This function retrieves one byte from the driver's Rx buffer and store it at *pDst.

12.3.8 IIC_Transmit_Slave

Prototype

```
bool_t IIC_Transmit_Slave(uint8_t const *pBuf, index_t bufLen, void (*pfCallBack)(uint8_t const
*pBuf));
```

Description

This function transmits in slave mode bufLen bytes of data from pBuffer over a port.

12.3.9 IIC_Transmit_Master

Prototype

```
bool_t IIC_Transmit_Master(uint8_t const *pBuf, index_t bufLen, uint8_t destAddress, void
(*pfCallBack)(bool_t status));
```

Description

This function begins transmitting in master mode bufLen bytes of data from *pBuffer.

12.3.10 IIC_Receive_Master

Prototype

```
bool_t IIC_Receive_Master(uint8_t *pBuf, index_t bufLen, uint8_t destAddress, void
(*pfCallBack)(bool_t status));
```

Description

This function begins receiving in master mode bufLen bytes of data from *pBuffer.

12.3.11 IIC_IsSlaveTxActive

Prototype

```
bool_t IIC_IsSlaveTxActive(void);
```

Description

This function checks if Slave Tx process is still running.

12.3.12 IIC_TxDataAvailable

Prototype

```
void IIC_TxDataAvailable(bool_t bIsAvailable);
```

Description

This function notifies the master to read data from the slave.

Chapter 13
SPI

The SPI interface allows easy access to the Serial Peripheral Interface of the microcontroller units on the Freescale 802.15.4 compliant platforms.

13.1 SPI Overview

The Serial Peripheral Interface module is a synchronous serial data input/output port that interfaces with serial memories, peripheral devices, or other processors. The SPI allows an 8-bit serial bit stream to be shifted simultaneously into and out of the device at a programmed bit-transfer rate (called 4-wire mode). The SPI module can be programmed for master or slave operation. It also supports a 3-wire mode where for master mode the MOSI becomes MOMI, a bidirectional data pin, and for slave mode the MISO becomes SISO, a bidirectional data pin. In 3-wire mode, data is only transferred in one direction at a time.

IEEE 802.15.4 compliant Freescale platforms are based on microcontroller units which have one or several SPI modules. Some of these platforms communicate with the radio transceiver via SPI, so the module is not available for other applications.

Support for the SPI module is provided in the `SPI.c` and `SPI.h` source files.

13.2 SPI Properties

13.2.1 gSPI_Enabled_d

C Module Property	gSPI_Enabled_d
BeeKit Property Name	SPI module enabled

BeeKit Component Containing BeeKit Property PLM

The gSPI_Enabled_d enables or disables SPI support. The gSPI_Enabled_d property is set in BeeKit (SPI module enabled from the PLM component property). If support is disabled, the SPI_ functions all become empty macros, so the same C source code can compile for MCUs with or without a SPI module.

13.2.2 gSPI_Slave_TxDataAvailableSignal_Enable_c

C Module Property	gSPI_Slave_TxDataAvailableSignal_Enable_c
BeeKit Property Name	SPI slave tx data available signal

BeeKit Component Containing BeeKit Property PLM

Configure if the slave transmitter can signal to the master if there's data available.

13.2.3 gSPI_AutomaticSsPinAssertion_c

C Module Property	gSPI_AutomaticSsPinAssertion_c
BeeKit Property Name	SPI automatic SS output assertion.
BeeKit Component Containing BeeKit Property	PLM

Configure SPI automatic SS output assertion.

13.2.4 gSPI_SlaveTransmitBuffersNo_c

C Module Property	gSPI_SlaveTransmitBuffersNo_c
BeeKit Property Name	SPI Slave - Number of Tx buffers
BeeKit Component Containing BeeKit Property	PLM

Configure the number of entries in the transmit-buffers-in-waiting list when the SPI is used in slave mode.

13.2.5 gSPI_SlaveReceiveBufferSize_c

C Module Property	gSPI_SlaveReceiveBufferSize_c
BeeKit Property Name	SPI Slave - Rx buffer size
BeeKit Component Containing BeeKit Property	PLM

Configure the size of the driver's Rx circular buffer when used in slave mode. This buffer is used to hold the received bytes until the application can retrieve them via the SPI_GetByteFromBuffer() function, and are not otherwise accessible from outside the driver.

13.2.6 gSPI_DefaultBaudRate_c

C Module Property	gSPI_DefaultBaudRate_c
BeeKit Property Name	SPI default Baud Rate
BeeKit Component Containing BeeKit Property	PLM

This property defines the Baud Rate used for sending and receiving data over SPI.

13.2.7 gSPI_DefaultMode_c

C Module Property	gSPI_DefaultMode_c
BeeKit Property Name	SPI Mode
BeeKit Component Containing BeeKit Property	PLM

SPI mode of operation: Master / Slave. The Master generates the clock signal.

13.2.8 gSPI_DefaultBitwiseShfting_c

C Module Property	gSPI_DefaultBitwiseShfting_c
BeeKit Property Name	SPI bitwise shifting
BeeKit Component Containing BeeKit Property	PLM

Normally, SPI data is transferred most significant bit (MSB) first. Selecting "gSPI_LsbFirst_c", the least significant bit first enable (LSBFE) bit is set, and SPI data is shifted LSB first.

13.2.9 gSPI_DefaultOperMode_c

C Module Property	gSPI_DefaultOperMode_c
BeeKit Property Name	SPI bus mode
BeeKit Component Containing BeeKit Property	PLM

Full duplex - SPI uses separate pins for data input and data output. Single wire - SPI configured for single-wire bidirectional operation.

13.2.10 gSPI_DefaultClockPol_c

C Module Property	gSPI_DefaultClockPol_c
BeeKit Property Name	SPI CLKpolarity
BeeKit Component Containing BeeKit Property	PLM

This setting effectively places an inverter in series with the clock signal from a master SPI or to a slave SPI device. SPI clock polarity: Active Low / Active High.

13.2.11 gSPI_DefaultClockPhase_c

C Module Property	gSPI_DefaultClockPhase_c
BeeKit Property Name	SPI CLK phase
BeeKit Component Containing BeeKit Property	PLM

This setting selects one of two clock formats for different kinds of synchronous serial peripheral devices. SPI clock phase: Odd Edge Shifting / Even Edge Shifting.

13.3 SPI API

The SPI API is declared in the `SPI_Interface.h` source file.

13.3.1 SPI_Init

Prototype

```
bool SPI_Init(void);
```

Description

This function initializes the SPI module.

13.3.2 SPI_Uninit

Prototype

```
void SPI_Uninit(void);
```

Description

Uninitializes the SPI module.

13.3.3 SPI_GetConfig

Prototype

```
bool_t SPI_GetConfig(spiConfig_t* pSpiConfig);
```

Description

This function gets the SPI module configuration.

13.3.4 SPI_SetConfig

Prototype

```
void SPI_SetConfig(spiConfig_t mSpiConfig);
```

Description

This function sets a specific configuration for the SPI module.

13.3.5 SPI_SetSlaveRxCallBack

Prototype

```
void SPI_SetSlaveRxCallBack(void (*pfCallBack)(void));
```

Description

Sets the callback function for the SPI slave mode receive operation.

13.3.6 SPI_MasterTransmit

Prototype

```
bool_t SPI_MasterTransmit(uint8_t *pBuf, index_t bufLen, void (*pfCallBack)(bool_t status));
```

Description

This function performs a data transmission in master mode.

13.3.7 SPI_MasterReceive

Prototype

```
bool_t SPI_MasterReceive(uint8_t *pBuf, index_t bufLen, void (*pfCallBack)(bool_t status));
```

Description

This function performs a data receive operation in master mode.

13.3.8 SPI_SlaveTransmit

Prototype

```
bool_t SPI_SlaveTransmit(uint8_t *pBuf, index_t bufLen, void (*pfCallBack)(uint8_t *pBuf));
```

Description

This function schedules a transmission over the SPI bus in slave mode.

13.3.9 SPI_GetByteFromBuffer

Prototype

```
bool_t SPI_GetByteFromBuffer(uint8_t *pDst);
```

Description

This function is called to retrieve a character from the circular receive buffer in slave mode.

Chapter 14
CMT

The CMT interface allows easy access to the Carrier Modulator Timer of the microcontroller units on the Freescale 802.15.4 compliant platforms.

14.1 CMT Overview

The Carrier Modulator Timer (CMT) module is a IR LED driver. The CMT consists of a carrier generator, modulator, and transmitter that drives the infrared out (IRO) pin. The module can transmit data to IRO output pin either in baseband or in FSK mode.

The Freescale 802.15.4 compliant platform MC1323x can be provided with a setup consisting of a MC1323x Modulator Reference Board (MRB) plugged into a MC1323x Remote Extender Motherboard (REM). Both of these boards are equipped with IR LEDs which can be used with the CMT module, depending of the boards interconnectivity setup.

Support for the CMT module is provided in the `CMT.c` and `CMT.h` source files.

14.2 CMT Properties for the 1323x-MRB

14.2.1 gCmtIncluded_d

C Module Property	gCmtIncluded_d
BeeKit Property Name	CMT module enabled

BeeKit Component Containing BeeKit Property PLM

The gCmtIncluded_d enables or disables SPI support. The gCmtIncluded_d property is set in BeeKit (SPI module enabled from the PLM component property). If support is disabled, the CMT_ functions all become empty macros, so the same C source code can compile for MCUs with or without a CMT module.

14.2.2 gCmtDefaultCarrierFrequency_c

C Module Property	gCmtDefaultCarrierFrequency_c
BeeKit Property Name	CMT default carrier frequency

BeeKit Component Containing BeeKit Property PLM

This parameter sets the CMT default Carrier frequency in Hz.

14.2.3 gCmtDefaultLog0MarkInMicros_c

C Module Property	gCmtDefaultLog0MarkInMicros_c
BeeKit Property Name	CMT default Mark 0 duration
BeeKit Component Containing BeeKit Property	PLM

This parameter sets the CMT default MARK duration in microseconds for the Logical 0 bit generation.

14.2.4 gCmtDefaultLog0SpaceInMicros_c

C Module Property	gCmtDefaultLog0SpaceInMicros_c
BeeKit Property Name	CMT default Space 0 duration
BeeKit Component Containing BeeKit Property	PLM

This parameter sets the CMT default SPACE duration in microseconds for the Logical 0 bit generation.

14.2.5 gCmtDefaultLog1MarkInMicros_c

C Module Property	gCmtDefaultLog1MarkInMicros_c
BeeKit Property Name	CMT default Mark 1duration
BeeKit Component Containing BeeKit Property	PLM

This parameter sets the CMT default MARK duration in microseconds for the Logical 1 bit generation.

14.2.6 gCmtDefaultLog1SpaceInMicros_c

C Module Property	gCmtDefaultLog1SpaceInMicros_c
BeeKit Property Name	CMT default Space 1duration
BeeKit Component Containing BeeKit Property	PLM

This parameter sets the CMT default SPACE duration in microseconds for the Logical 1 bit generation.

14.2.7 gCmtLsbFirstDefault_c

C Module Property	gCmtLsbFirstDefault_c
BeeKit Property Name	CMT default bit shifting method
BeeKit Component Containing BeeKit Property	PLM

This parameter sets the CMT default shifting method: LSB firts or MSB first. Setting this parameter to TRUE will configure the CMT to shift bits as LSB first, while setting this parameter to FALSE will configure the CMT to shift bits as MSB first.

14.2.8 gCmtTimeOperModeDefault_c

C Module Property	gCmtTimeOperModeDefault_c
BeeKit Property Name	CMT default bit operating mode

BeeKit Component Containing BeeKit Property PLM

This parameter sets the CMT default operating mode: Time Mode or Baseband Mode. Setting this parameter to TRUE will configure the CMT in Time Mode, while setting this parameter to FALSE will configure the CMT in Baseband Mode.

14.2.9 gCmtOutputPolarityDefault_c

C Module Property	gCmtOutputPolarityDefault_c
BeeKit Property Name	CMT default output polarity

BeeKit Component Containing BeeKit Property PLM

This parameter sets the CMT output polarity: IRO active low or IRO active high. Setting this parameter to TRUE will configure the IRO pin to be active high, while setting this parameter to FALSE will configure the IRO pin to be active low.

14.3 CMT API for the 1323x-MRB

The CMT API is declared in the `CMT_Interface.h` source file.

14.3.1 CMT_Initialize

Prototype

```
void CMT_Initialize(void);
```

Description

This function initializes the CMT module.

14.3.2 CMT_SetCarrierWaveform

Prototype

```
cmtErr_t CMT_SetCarrierWaveform(uint8_t highCount, uint8_t lowCount);
```

Description

Configures the CMT registers with high and low values from provided frequency and duty cycle.

14.3.3 CMT_SetTxCallback

Prototype

```
void CMT_SetTxCallback(cmtCallback_t callback);
```

Description

This function provides a pointer to the callback function after finishing any Tx activity.

14.3.4 CMT_SetMarkSpaceLog0

Prototype

```
cmtErr_t CMT_SetMarkSpaceLog0(uint16_t markPeriod, uint16_t spacePeriod);
```

Description

This function configures the mark and space width for logical 0 bit.

14.3.5 CMT_SetMarkSpaceLog1

Prototype

```
cmtErr_t CMT_SetMarkSpaceLog1(uint16_t markPeriod, uint16_t spacePeriod);
```

Description

This function configures the mark and space width for logical 1 bit.

14.3.6 CMT_TxBits

Prototype

```
cmtErr_t CMT_TxBits(uint8_t data, uint8_t bitsCount);
```

Description

Transfers a number of up to 8 bits via IRO pin.

14.3.7 CMT_TxModCycle

Prototype

```
cmtErr_t CMT_TxModCycle(uint16_t markPeriod, uint16_t spacePeriod);
```

Description

This function triggers a modulation cycle with mark/space parameters given as arguments.

14.3.8 CMT_Abort

Prototype

```
void CMT_Abort(void);
```

Description

This function aborts a CMT transmission operation if it ongoing.

14.3.9 CMT_IsTxActive

Prototype

```
bool_t CMT_IsTxActive(void);
```

Description

This function checks if whether a transmission onto the CMT's IRO pin is ongoing.

14.3.10 CMT_SetModOperation

Prototype

```
cmtErr_t CMT_SetModOperation(bool_t timeOperMode);
```

Description

This function sets the mode of operation: time or baseband.

14.3.11 CMT_SetBitsShifting

Prototype

```
cmtErr_t CMT_SetBitsShifting(bool_t bitsLsbShifting);
```

Description

This function sets the bits shifting mode: LSB or MSB first.

Chapter 15
Over-The-Air Programmer (OTAP)

The OTAP interface allows easy access to the Over-The-Air Programmer feature support.

15.1 OTAP Overview

The over the air programming feature allows a device to receive an upgrade image over the air. The device that is receiving the new image is called recipient. The device that has the new image is called server. The recipient must accesses memory support for upgrade image. The implementation is based on the "ZigBee Over-the-Air Upgrading Cluster" specification released by the ZigBee Alliance. The access to this feature is done with the prototypes provided in OtapSupport.h

15.2 OTAP API

The functions prototypes from `OtapSupport.h` are required for OTAP module, on client, to access memory support used for upgrade image. Some of these API functions are specific to certain devices such as the Freescale MC1322x.

15.2.1 Return Values Type

```
typedef enum
{
  gOtapSucess_c = 0,
  gOtapNoImage_c,
  gOtapUpdated_c,
  gOtapError_c,
  gOtapCrcError_c,
  gOtapInvalidParam_c,
  gOtapInvalidOperation_c,
  gOtapEepromError_c,
  gOtapInternalFlashError_c,
} otapResult_t;
```

15.2.2 OTAP_StartImage

Prototype

```
otapResult_t OTAP_StartImage(uint32_t length);
otapResult_t OTAP_StartImage(uint32_t length, uint32_t receivedCrc); //MC1322x specific
```

Description

The call to OTAP_StartImage, starts the process of writing a new image to the upgrade image support. The input parameter is the length of the upgrade image to be written in memory support. The call to OTAP_StartImage returns gOtapInvalidParam_c, in case the intended length is bigger than the memory support capacity, gOtapInvalidOperation_c, if the process is already started, gOtapEepromError_c, if there is a problem with memory support.

On MC1322x, the "*receivedCrc*" parameter holds the Over-The-Air received CRC value. Used by the OTAP implementation to compare with the calculated CRC on the receiver side.

15.2.3 OTAP_PushImageChunk

Prototype

```
otapResult_t OTAP_PushImageChunk(uint8_t* pData,
   uint8_t length,
   uint32_t* pImageLength);
```

Description

The call to OTAP_PushImageChunk writes the next image chunk in memory support. The input parameters are pData- pointer to the data chunk, length – the length of the data chunk, pImageLength – if it is not null, and the function call is successful, it will be filled with the current length of the transferred upgrade image. The call to OTAP_PushImageChunk returns gOtapInvalidParam_c if the pData is null or the resulting image would be bigger than the final image length specified with OTAP_StartImage(), gOtapEepromError_c if an error occurs while trying to write in memory support, and gOtapInvalidOperation_c if the process is not started.

15.2.4 OTAP_CommitImage

Prototype

```
otapResult_t OTAP_CommitImage(uint8_t* bitmap);
```

Description

The call to OTAP_CommitImage finishes the writing of a new image to memory support. The input parameter is a pointer to a byte array indicating the sector erase pattern for the internal flash. The call to OTAP_CommitImage returns gOtapInvalidOperation_c if the process is not started and gOtapEepromError_c if an error occurs while trying to write in memory support.

15.2.5 OTAP_CancelImage

Prototype

```
void OTAP_StartImage(void);
```

Description

The call to OTAP_CancellImage cancels the process of writing an upgrade image in memory suport.

15.2.6 OTAP_WriteNewImageFlashFlags

Prototype

```
void OTAP_WriteNewImageFlashFlags(uint32_t length); //not present on MC1322x
```

Description

The call to OTAP_WriteNewImageFlashFlags informs the bootloader that a new upgrade image is ready in memory support. This function is not used by Freescale private profile. This function is presented as an example and is used in the demonstration application.

15.2.7 OTAP_InitExternalMemory

Prototype

```
void OTAP_InitExternalMemory(void); //MC1322x specific
```

Description

The call to OTAP_InitExternalMemory will initialize and set up the communication with the external memory (flash or EEPROM) attached to the MC1322x device.

15.2.8 OTAP_ReadExternalMemory

Prototype

```
otapResult_t OTAP_ReadExternalMemory(uint8_t* pData, uint8_t length, uint32_t address);
//MC1322x specific
```

Description

The call to OTAP_ReadExternalMemory will read a data chunk of a specified length from an address in the external memory.

15.2.9 OTAP_WriteExternalMemory

Prototype

```
otapResult_t OTAP_WriteExternalMemory(uint8_t* pData, uint8_t length, uint32_t address);
//MC1322x specific
```

Description

The call to OTAP_WriteExternalMemory will write a data chunk of a specified length to a specified address in the external memory.

15.2.10 OTAP_CrcCompute

Prototype

```
uint16_t OTAP_CrcCompute(uint8_t *pData, uint16_t lenData, uint16_t crcValueOld); //MC1322x
specific
```

Description

The call to OTAP_CrcCompute will calculate and return the 16-bit CRC-CCIT (0x1021) of a data chunk of specified length.

15.2.11 OTAP_EraseExternalMemory

Prototype

```
otapResult_t OTAP_EraseExternalMemory(void); //MC1322x specific
```

Description

The call to OTAP_EraseExternalMemory will erase the external memory used for image storage, attached to the MC1322x device.

Chapter 16
EEPROM

The EEPROM interface allows easy access to various EEPROM modules that reside on the Freescale 802.15.4 compliant platforms development boards. and which communicate with the MCU using a serial interface such as I2C or SPI.

16.1 EEPROM Overview

The EEPROM modules communicate with the MCU using a serial interface such as I2C or SPI.

The support for the EEPROM modules is provided in the `Eeprom_AT45DB021D.c`, `Eeprom_AT25HP512C1.c` and `Eeprom_AT24C1024BW.c` source files.

16.2 EEPROM API

The EEPROM API is provided in the `EEPROM.h` header file.

16.2.1 Return Value Types

```
typedef enum
{
  ee_ok,
  ee_too_big,
  ee_not_aligned,
  ee_error
} ee_err_t;
```

16.2.2 EEPROM_Init

Prototype

```
ee_err_t EEPROM_Init(void);
```

Description

This function initializes the port pins and the EEPROM.

16.2.3 EEPROM_ReadData

Prototype

```
ee_err_t EEPROM_ReadData (uint16_t NoOfBytes, uint32_t Addr, uint8_t  *inbuf);
```

Description

This function Reads NoOfBytes bytes from the EEPROM.

16.2.4 EEPROM_WriteData

Prototype

```
ee_err_t EEPROM_WriteData(uint8_t  NoOfBytes, uint32_t Addr, uint8_t  *Outbuf );
```

Description

This function writes NoOfBytes bytes to the EEPROM.

Chapter 17 Bootloader

The Bootloader interface allows easy access to the platform component running on the Freescale 802.15.4 compliant platforms.

17.1 Bootloader Overview

The platform offers a reference implementation for an external bootloader component. The external bootloader support is included by setting the “Enable external storage bootloader” option to True in the PLM component of the BeeKit project. The bootloader offers support for booting from an external EEPROM and has a very small memory footprint that fits into a 1KB of flash. When enabled, bootloader code checks the ‘bootFlag’ variable stored in the IC internal flash for the existence of a new image in the external EEPROM. If the flag indicates that a new image is present, the bootloader starts the loading of the image from external EEPROM to internal Flash. Support for skipping various FLASH sectors is offered, so that sensitive information like pairing tables, NIBs etc. can be preserved across firmware updates. When the image loading from EEPROM to internal FLASH is done, bootloader marks the process as complete by updating the ‘bootImageEnd’ flag also stored in the internal FLASH. This flag assures that in case the IC is accidentally reset while loading the image from EEPROM to internal FLASH the loading process will be restarted from the beginning.

The bootloader reference implementation offers support for three distinct Atmel EEPROM devices: AT25HP512C1, AT24C1024BW and AT45DB021D.

Support for the Bootloader component is provided in the Bootloader.c source file.

17.2 Bootloader API

The Bootloader API is declared in the `Bootloader.h` header file.

17.2.1 Boot_LoadImage

Prototype

```
void Boot_LoadImage(void);
```

Description

This function is called when the startup code detects that a new image is available. Right now, the bootloader only supports loading the image from an external EEPROM. The image from EEPROM will be loaded into the internal MCU FLASH, after which the MCU will be reset.

17.2.2 Boot_Add8BitValTo32BitVal

Prototype

```
void Boot_Add8BitValTo32BitVal(uint8_t val8Bit, uint16_t* pVal32Bit);
```

Description

This is a utility function that adds a 8 bit value (val8Bit) to a 32 bit value (val32Bit). The code is written such that the compiler does not include any calls to 32 bit functions from the compiler library (ansiis or ansibim).

17.3 Bootloader as a Standalone Application

The MC1322x platform supports the bootloader as a standalone application. The bootloader must be downloaded in the first FLASH sector. The resident application must be downloaded starting with the second FLASH sector if it is downloaded in the internal FLASH, or with the first sector if it is downloaded in the external FLASH. After the bootloader and the resident application are downloaded, the bootstrap code in the MC1322x ROM will copy the bootloader application from the internal FLASH to RAM and it will then pass control to the RAM copy of the bootloader. In other words, it performs a jump in RAM.

17.3.1 Bootloader Functionality

Bootloader functionality is divided into three areas of operation:

- Load a new image from external FLASH to internal FLASH (if one exists)
- Load the internal FLASH image to RAM
- Give control to the RAM copy

To optimize RAM usage, the bootloader application is divided into the following two sections:

Critical	Holds the code responsible for copying the image from internal FLASH to RAM.
Non-critical	This section can be overwritten in RAM.

The bootloader's files are as follows:

`Bootloader.c`	Contains the implementation of the main function.
`BootloaderFlashUtils.c`	Contains the implementation of the functions that handle the operations for working with the FLASH modules.
`BootloaderInit.s`	Contains the initialization of the application.

17.3.2 Bootloader Functions

17.3.2.1 Bootloader_Check

Prototype

```
void Bootloader_Check(void);
```

Description

This function has three primary purposes:

- Inspect the external FLASH for a valid image
- Update the internal FLASH with the valid image found in the external FLASH
- Invalidate the external FLASH image. (This is a non-critical function.)

17.3.2.2 Bootloader_LoadFromInternalFlash

Prototype

```
void Bootloader_LoadFromInternalFlash(void);
```

Description

This function loads the image from the internal FLASH to RAM. This is a critical function.

17.3.2.3 BootloaderCrcCompute

Prototype

```
void BootloaderCrcCompute(uint8_t *pData, uint16_t lenData);
```

Description

This function computes the CRC over the pData array. This is a non-critical function.

17.3.2.4 BootloaderUpdateImage

Prototype

```
bootResult_t BootloaderUpdateImage(void);
```

Description

This function copies an image from external FLASH to internal FLASH and returns the status of the operation. This is a non-critical function.