



# AN1418: Running Zigbee and OpenThread Concurrently on a System-on-Chip

---

This document describes how to run a combination of Zigbee and OpenThread networking stacks and the Zigbee application layer on a System-on-Chip (SoC). One of the main functions of a Concurrent Multiprotocol (CMP) device is to act as a bridge between Zigbee and OpenThread networks.

Note that, depending on the chip, memory size restrictions may prevent running Matter on SoC devices.

## KEY POINTS

- Important features of the sample application
- Making a Zigbee-OpenThread CMP application from a Z3Light

## 1 Introduction

This document describes a Concurrent Multiprotocol (CMP) application that runs Zigbee and OpenThread stacks on a single EFR32 radio. The primary use for such an application is to allow Zigbee line-powered devices to also be part of an OpenThread network simultaneously and therefore serve as a bridge between the two networks.

## 2 Concurrent Multiprotocol (CMP) Sample Application (z3-light\_ot-ftd)

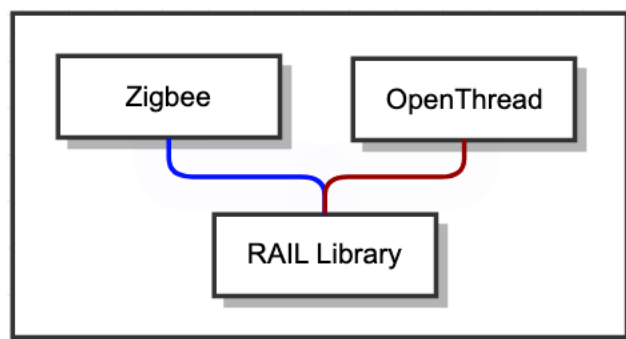


Figure 2-1. Zigbee + OpenThread Concurrent Multiprotocol Application

The CMP sample application consists of a Z3Light, which is a Zigbee router, and an OpenThread FTD (Full Thread Device). Both protocol stacks operate by multiplexing a single EFR32 radio. Both protocols need to use the same radio channel to ensure proper operation.

### 2.1 RTOS

Within the CMP application, scheduling is managed using a Real Time Operating System (RTOS). Each protocol runs in a dedicated RTOS task. The Zigbee and OpenThread tasks operate at the same priority while the Command Line Interface (CLI) is made available using a CLI RTOS task that operates at a lower priority.

**Caution:** It is critical to note that Zigbee and OpenThread APIs are not thread-safe. Calling them from different threads can result in unexpected behavior. In addition, any references to `EmberMessageBuffer` must be contained within the Zigbee task.

### 2.2 Command Line Interface

This application supports all CLI commands that can be found in the Z3Light sample application. A subset of the OpenThread CLI has been ported to demonstrate form, join and ping operations. This functionality can be extended further, if necessary, by following the example commands in the `ot_up_cli.c` file from the `ot_up_cli` component. Note that OpenThread APIs are only invoked from `sl_ot_rtos_application_tick` since they are not thread-safe.

#### 2.2.1 OpenThread Commissioning

This device can be commissioned on to an OpenThread network out-of-band using CLI commands. Setting the OpenThread network parameters, such as network key and channel, before starting the network allows the CMP device to join a Thread network as a child or router device.

Table 2.1. OpenThread CLI commands

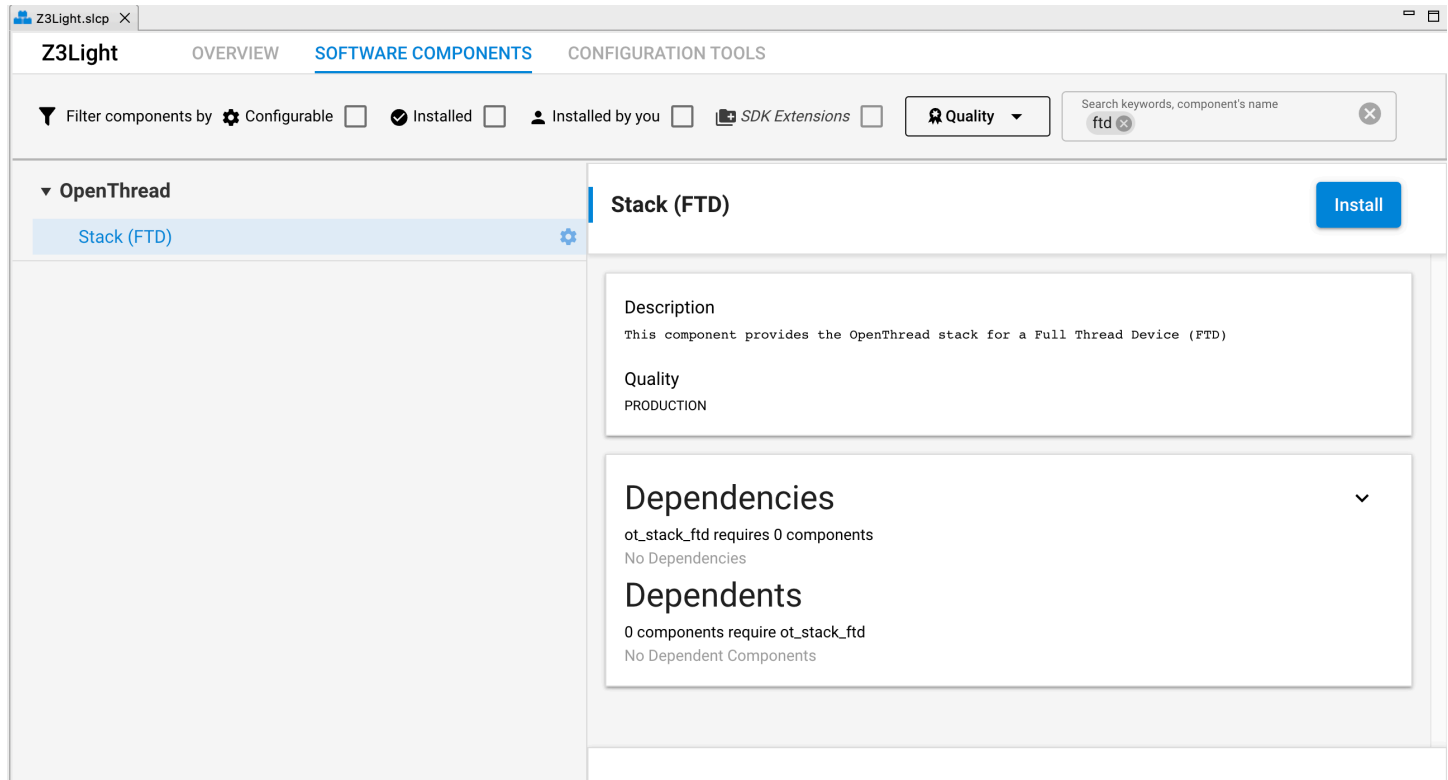
CLI Command	Description
<code>dataset</code>	View OpenThread network configuration.
<code>dataset_new</code>	Creates a new OpenThread dataset.
<code>dataset_commit_active</code>	Commits dataset to NVM.
<code>factory_reset</code>	Removes all NVM OpenThread settings.
<code>dataset_networkkey</code>	Presets the network key on the device to help with joining an existing OpenThread network out-of-band.
<code>dataset_channel</code>	Presets the radio channel used by the OpenThread network. This command can be used to force both Zigbee and OpenThread networks to use the same radio channel.

CLI Command	Description
dataset_pan_id	Presets the PAN ID on the device to help with joining the OpenThread network out-of-band.
dataset_extended_pan_id	Presets the extended PAN ID on the device to help with joining the OpenThread network out-of-band.
ifconfig_up	Enables OpenThread interface.
thread_start	Enables and attaches OpenThread protocol operation.
thread_state	Reads current status: offline, disabled, detached, child, router, or leader.

### 3 Converting a Zigbee Application into a Zigbee-OpenThread CMP Application

This section describes the steps involved in converting a Z3Light into a Concurrent Multiprotocol application that includes the OpenThread stack.

1. Use the Simplicity Studio “Create New Project” wizard to create a Zigbee – SoC Light project for your board of choice.
2. Open the Software Components tab of the generated project to add the OpenThread > **Stack (FTD)** component. Note that the addition of this component automatically adds a Real Time Operating System (RTOS) to the project.



3. Select the IO Stream > Driver > IOS Stream: USART > **vcom** component and configure it by increasing “Receive buffer size” to 128.

The screenshot shows the Z3Light IDE's 'SOFTWARE COMPONENTS' page. On the left, a tree view shows the hierarchy: Platform > Board > Radio Board > BRD4161A > Services > Co-Processor Communication > Secondary Device > CPC: Auto Configure VCOM Speed > IO Stream > Driver > IO Stream: USART > **vcom**. The 'vcom' component is highlighted. On the right, the 'vcom' component details are shown, including a description: 'Instantiate the driver for using IO Stream over the Universal Synchronous Asynchronous Receiver Transceiver (USART) peripheral.' and a quality of 'PRODUCTION'. Below this, the 'Dependencies' section shows 'iostream\_usart-vcom requires 1 components' and the 'Platform' section. The 'Dependents' section shows '0 components require iostream\_usart-vcom'. At the bottom, there are buttons for 'Uninstall', 'Add New Instances', and 'Instances'.

The screenshot shows the 'IO Stream: USART (vcom)' configuration page. The page has a title bar with 'Pin Tool', 'View Source', and a close button. The main content area is titled 'USART settings' and contains several configuration options: 'Baud rate' (115200), 'Parity mode to use' (No Parity), 'Number of stop bits to use' (1 stop bits), 'Flow control' (CTS/RTS), 'Receive buffer size' (128), 'Convert \n to \r\n' (disabled), and 'Restrict the energy mode to allow the reception' (enabled). The 'Receive buffer size' is highlighted with a blue border.

4. Select the Toolchain > **Memory Configuration** component and configure it by increasing stack size to 4608 and heap size to 16384 to account for the addition of OpenThread networking stack.

The screenshot displays the 'Z3Light\_2' project interface. The 'SOFTWARE COMPONENTS' tab is active, showing a list of components under the 'Platform' > 'Toolchain' category. The 'Memory Configuration' component is selected and highlighted. To the right, the 'Memory Configuration' details panel is visible, showing a description, quality (PRODUCTION), and dependencies. Below this, a configuration window for 'Memory Configuration' is open, showing input fields for 'Stack size for the application' (set to 4608) and 'Minimum heap size for the application' (set to 16384).

**Z3Light\_2** OVERVIEW **SOFTWARE COMPONENTS** CONFIGURATION TOOLS

Filter components by ☒ Configurable ☐ Installed ☐ Installed by you ☐ SDK Extensions ☐ Quality

▼ Platform

▼ Toolchain

⚙️ Memory Configuration ⚙️

**Memory Configuration**

**Description**

This component provides configuration of the stack and heap for supported toolchains. For gcc it also adds support for `_sbrk()` for heap allocation. This is used in the newlib version of `malloc()`.

**Quality**

PRODUCTION

**Dependencies**

sLmemory requires 0 components

No Dependencies

Dependents

✖ Uninstall

readme.html Z3Light.slc Memory Configuration X

**Memory Configuration**   X

**Memory configuration**

Stack size for the application.

^ 4608 v

Minimum heap size for the application.

^ 16384 v

5. Select Micrium > Common > **Micrium OS Common Module Core** component and configure it to decrease “size of heap memory” to 0 to prevent Micrium RTOS from allocating its own heap memory.

The screenshot shows the Z3Light interface with the 'SOFTWARE COMPONENTS' tab selected. On the left, a tree view shows the component hierarchy: RTOS > Micrium OS > Common > Micrium OS CPU Module, Kernel > Micrium OS Kernel, and Services > Micrium > Common. The 'Micrium OS Common Module Core' component is selected and highlighted in blue. On the right, the details for this component are displayed, including a description, quality (PRODUCTION), and a section for dependencies.

**Common APIs for CMSIS-Compliant Kernels**

**Description**  
 This component provides "sl\_cmsis\_os2\_common.h" header file, which in turn provides typedefs like osSemaphore\_t and osThread\_t. Those types are defined by CMSIS RTOS2 standard, yet their implementation is specific to the operating system. Traditionally, the user would need to include OS-specific header file (like "os.h" for MicriumOS) to have access to those types. This implies that the application needs to be aware of the kernel being used (MicriumOS, FreeRTOS, etc.) For applications that need to be OS-agnostic, this component provides an abstract header file "sl\_cmsis\_os2\_common.h" that provides the same functionality without the requirement of knowing which OS is used per se.

**Quality**  
 PRODUCTION

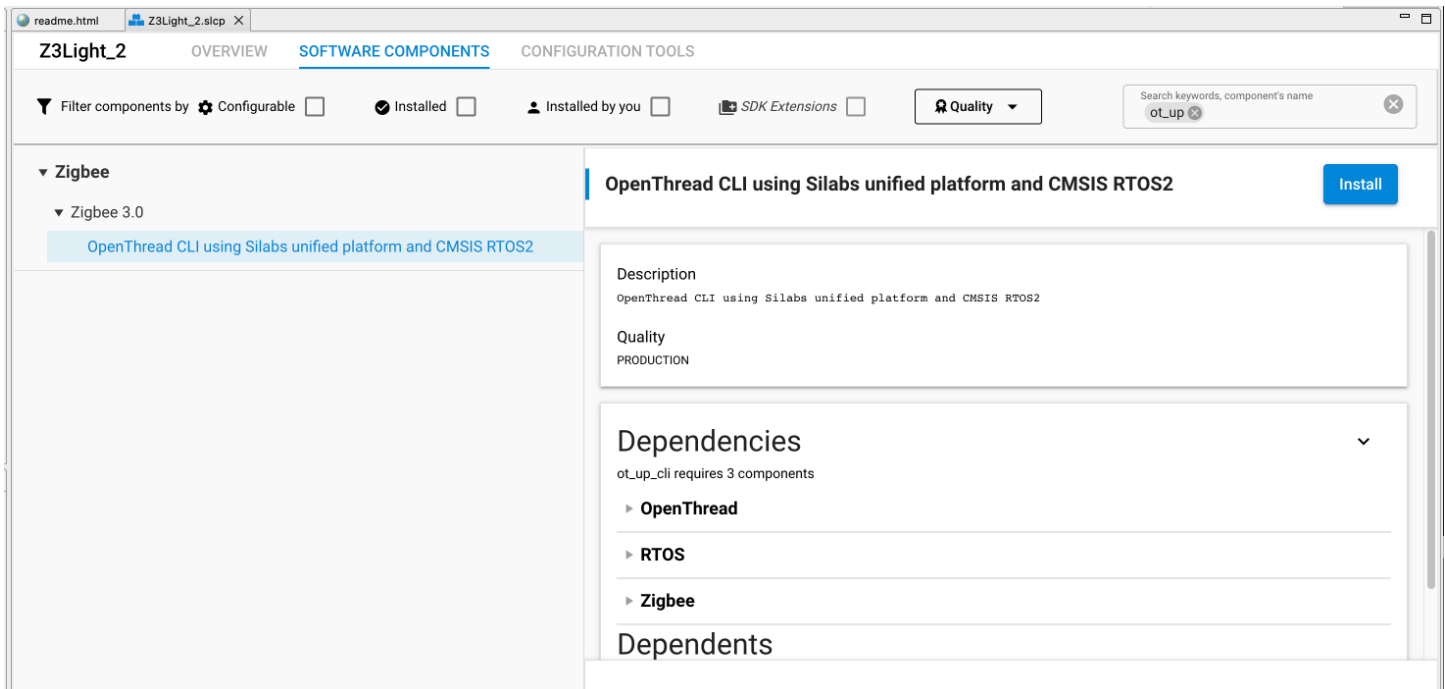
**Dependencies**

The screenshot shows the configuration page for the 'Micrium OS Common Module Core' component. The 'Memory Library Configuration' section is expanded, showing several settings:

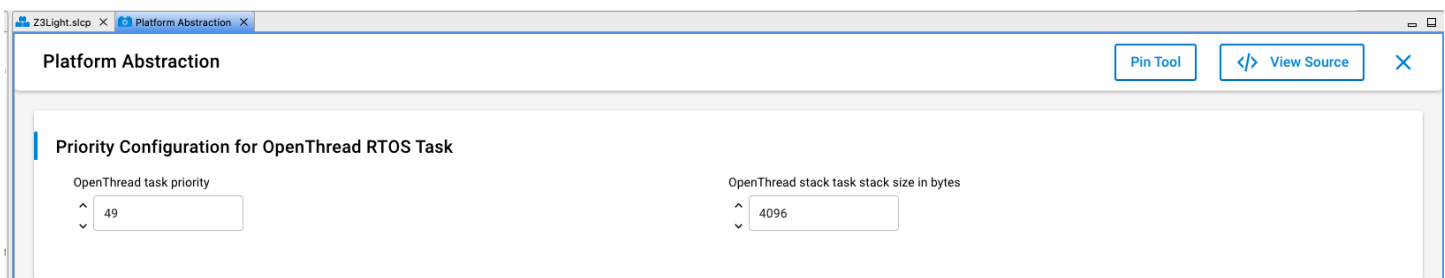
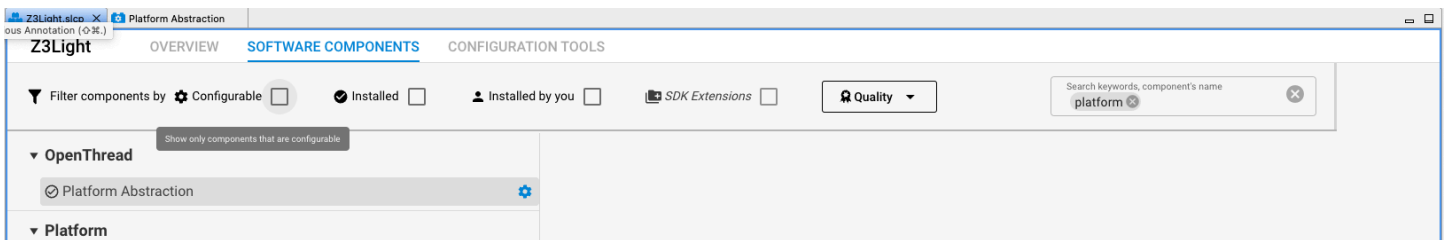
- Replace common lib memory functions with standard C lib functions: ☐
- Enable Memory allocation usage tracking: ☐
- Size of heap memory (in octets):
- Padding alignment for hardware allocations on heap (in octets):
- Enable Custom heap location: ☐



6. Add OpenThread CLI commands by installing the Zigbee > Zigbee 3.0 > **OpenThread CLI using Silabs unified platform (ot\_up\_cli)** component.

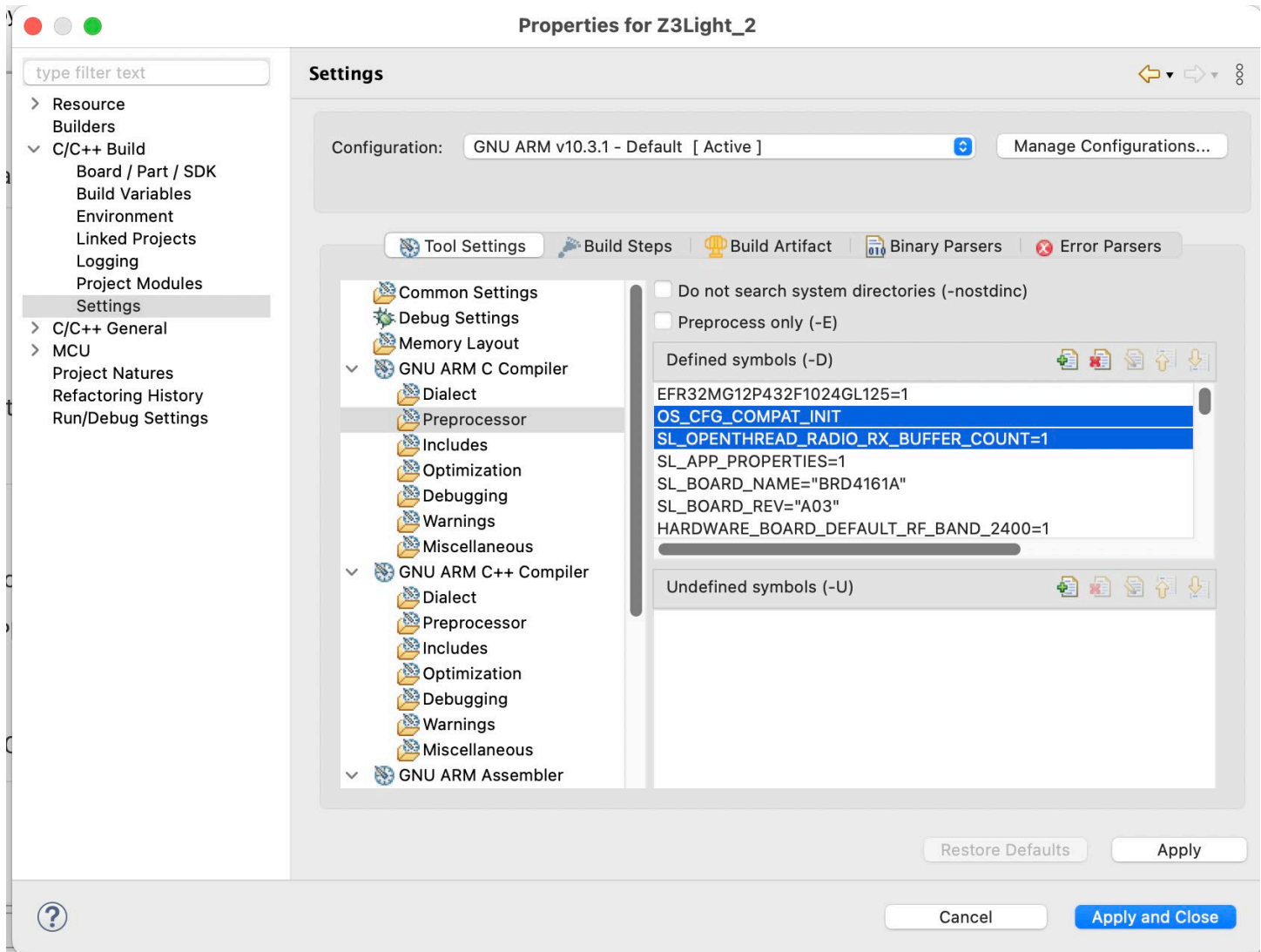


7. Select the OpenThread > **Platform Abstraction** component and configure it by setting priority to 49 to match the Zigbee RTOS task priority.



8. Right click the Z3Light project in Simplicity Studio's Project Explorer view and click Properties. Open C/C++ Build Settings and Under GNU ARM C Compiler, select Preprocessor. Add two preprocessor define symbols:
  - OS\_CFG\_COMPAT\_INIT (Used in conjunction with LIB\_MEM\_CFG\_HEAP\_SIZE to allow the application to handle heap allocation)
  - SL\_OPENTHREAD\_RADIO\_RX\_BUFFER\_COUNT=1 (This is a workaround for an issue where the Zigbee network cannot send beacons when the OpenThread network is up)

Click Apply and Close to save.



9. Open app.c file in the project folder and add the code below to the beginning of the file to initialize OpenThread. Save file and build the project.

```
#if defined(OPENTHREAD_FTD)
#include <assert.h>
#include <openthread-core-config.h>
#include <openthread/config.h>

#include <openthread/ncp.h>
#include <openthread/diag.h>
#include <openthread/tasklet.h>

#include "openthread-system.h"
```

```
static otInstance *    sInstance    = NULL;

void sl_ot_create_instance(void)
{
    #if OPENTHREAD_CONFIG_MULTIPLE_INSTANCE_ENABLE
        size_t    otInstanceBufferLength = 0;
        uint8_t *otInstanceBuffer        = NULL;

        // Call to query the buffer size
        (void)otInstanceInit(NULL, &otInstanceBufferLength);

        // Call to allocate the buffer
        otInstanceBuffer = (uint8_t *)malloc(otInstanceBufferLength);
        assert(otInstanceBuffer);

        // Initialize OpenThread with the buffer
        sInstance = otInstanceInit(otInstanceBuffer, &otInstanceBufferLength);
    #else
        sInstance = otInstanceInitSingle();
    #endif
    assert(sInstance);
}

otInstance *otGetInstance(void)
{
    return sInstance;
}
#endif // #if defined(OPENTHREAD_FTD)
```

This application can now form a distributed Zigbee network or join any Zigbee network (centralized or distributed). It can also function as a leader, child, or router on the OpenThread network.

**Caution:** It is imperative to ensure that both networks operate on the same radio channel.

Any channel changes will need to be done in a controlled fashion. A channel change on one protocol's network can cause the other protocol to stop working until its network is also switched to the same channel. It is important to note that only certain Zigbee device types (trust center) may initiate a channel change on the Zigbee side.

# Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!



**IoT Portfolio**  
[www.silabs.com/IoT](http://www.silabs.com/IoT)



**SW/HW**  
[www.silabs.com/simplicity](http://www.silabs.com/simplicity)



**Quality**  
[www.silabs.com/quality](http://www.silabs.com/quality)



**Support & Community**  
[www.silabs.com/community](http://www.silabs.com/community)

## Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

**Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit [www.silabs.com/about-us/inclusive-lexicon-project](http://www.silabs.com/about-us/inclusive-lexicon-project)**

## Trademark Information

Silicon Laboratories Inc.<sup>®</sup>, Silicon Laboratories<sup>®</sup>, Silicon Labs<sup>®</sup>, SiLabs<sup>®</sup> and the Silicon Labs logo<sup>®</sup>, Bluegiga<sup>®</sup>, Bluegiga Logo<sup>®</sup>, EFM<sup>®</sup>, EFM32<sup>®</sup>, EFR, Ember<sup>®</sup>, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals<sup>®</sup>, WiSeConnect, n-Link, ThreadArch<sup>®</sup>, EZLink<sup>®</sup>, EZRadio<sup>®</sup>, EZRadioPRO<sup>®</sup>, Gecko<sup>®</sup>, Gecko OS, Gecko OS Studio, Precision32<sup>®</sup>, Simplicity Studio<sup>®</sup>, Telegesis, the Telegesis Logo<sup>®</sup>, USBXpress<sup>®</sup>, Zentri, the Zentri logo and Zentri DMS, Z-Wave<sup>®</sup>, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.  
400 West Cesar Chavez  
Austin, TX 78701  
USA

[www.silabs.com](http://www.silabs.com)