UG495: Silicon Labs Wi-SUN Developer's Guide

This document is a reference for those developing applications using the Silicon Labs Wi-SUN (Wireless Smart Ubiquitous Network, Field Area Network) SDK (Software Development Kit). The guide covers the (Wi-SUN) stack architecture, application development flow, steps to configure the application Wi-SUN radio settings and advanced debug features. This version applies to the Silicon Labs Wi-SUN SDK version 1.x.x and higher.

The purpose of this document is to fill in the gaps between the Silicon Labs Wi-SUN Field Area Network (FAN) API reference, Gecko Platform references, and documentation for the target EFR32xG part. This document provides details that will help developers optimize their application for their target environment.

KEY POINTS

- Wi-SUN SDK stack, including firmware and application project structure, and software components
- Application development guidelines
- Wi-SUN radio configuration
- Advanced tools to test and debug a Wi-SUN application

Table of Contents

1 Introduction

This document contains information for anyone developing applications in the Silicon Labs Wi-SUN SDK. It assumes that the current version of the Silicon Labs Wi-SUN SDK has been installed and that the developer is familiar with creating and flashing applications, and with the functionality available as a starting point in the example files contained in the SDK. If you are not familiar with these items, and are just getting started, see the Simplicity Studio 5 User's Guide and QSG181: Silicon Labs Wi-SUN Quick-Start Guide. For more information about configuring a Wi-SUN network, see AN1332: Silicon Labs Wi-SUN Network Setup and Configuration.

The Silicon Labs Wi-SUN API reference that matches the installed SDK is available on the Simplicity Studio DOCUMENTS tab. All versions are available at https://docs.silabs.com/wisun/latest/wisun-stack-api/.

2 Wi-SUN FAN Stack

2.1 Firmware Structure

The following figure describes the high-level firmware structure. The developer creates an application on top of the stack, which Silicon Labs provides as a precompiled object-file, enabling the Wi-SUN connectivity for the end-device.

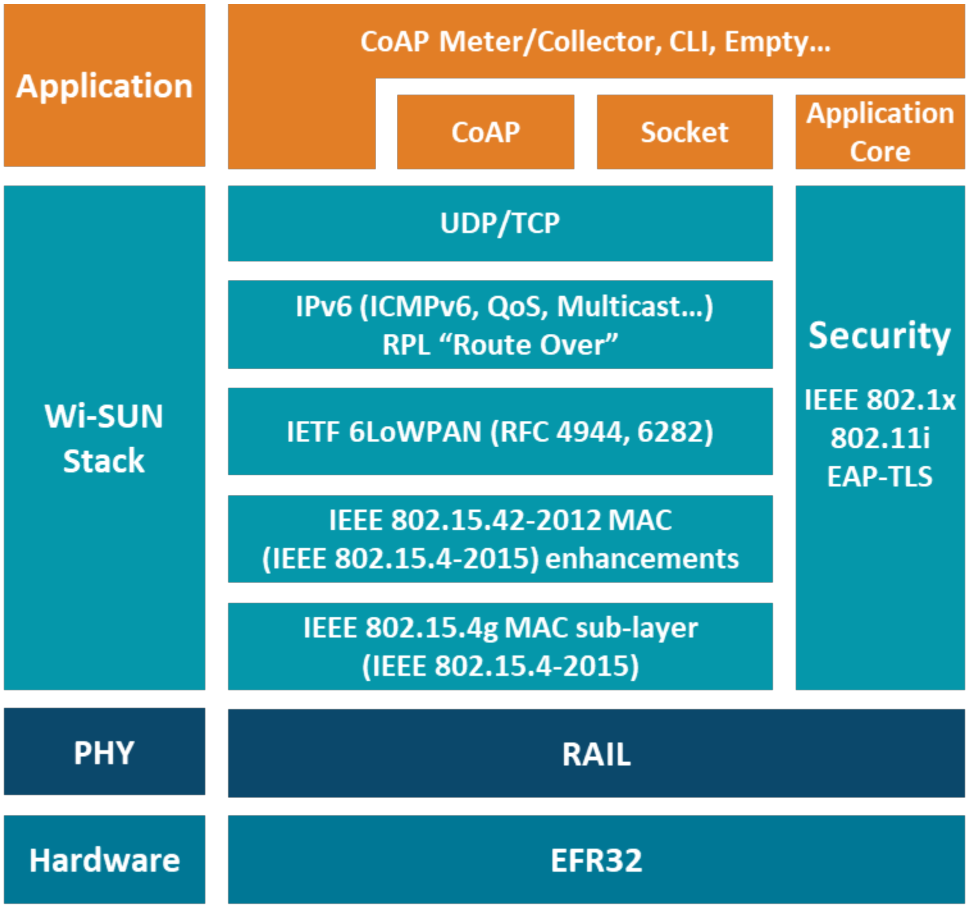


Figure 2.1. Wi-SUN Stack Architecture Block Diagram

The Wi-SUN stack contains following blocks.

- Wi-SUN stack – Wi-SUN functionality consisting of an IP stack, MAC layer, the routing protocol (RPL), and security manager.
- Wi-SUN RF test plugin – Optional software component to add an API to perform RF tests (for example, create an RF tone).
- Wi-SUN Util Functions – Optional software component to add helper functions to inform the application about the Wi-SUN PHY configured in the RAIL configuration file.

2.2 Application Project Structure

This section explains the application project structure and the mandatory and optional resources that must be included in the project.

2.2.1 Wi-SUN Files Library Files

The Wi-SUN stack libraries are summarized in the following table.

Table 2.1. Wi-SUN Stack Libraries

Wi-SUN stack library name	To use with
libwisun_router(_core)_efr32xg**X**x_micrium_gcc_debug.a	Wi-SUN router with Micrium OS, GCC, with debug traces
libwisun_router(_core)_efr32xg**X**x_micrium_gcc_release.a	Wi-SUN router with Micrium OS, GCC, no debug trace
libwisun_router(_core)_efr32xg**X**x_micrium_iar_debug.a	Wi-SUN router with Micrium OS, IAR, with debug traces
libwisun_router(_core)_efr32xg**X**x_micrium_iar_release.a	Wi-SUN router with Micrium OS, IAR, no debug trace
libwisun_router(_core)_efr32xg**X**x_freertos_gcc_debug.a	Wi-SUN router with FreeRTOS, GCC, with debug traces
libwisun_router(_core)_efr32xg**X**x_freertos_gcc_release.a	Wi-SUN router with FreeRTOS, GCC, no debug trace
libwisun_router(_core)_efr32xg**X**x_freertos_iar_debug.a	Wi-SUN router with FreeRTOS, IAR, with debug traces
libwisun_router(_core)_efr32xg**X**x_freertos_iar_release.a	Wi-SUN router with FreeRTOS, IAR, no debug trace
libwisun_rcp_efr32xg**X**x.a	Radio coprocessor (Linux border router)
libwisun_mac(_core)_efr32xg1**X**x.a	MAC layer for the Radio coprocessor (Linux border router)

The Wi-SUN stack library file names containing "_core_" do not support the LFN feature. The **X** stands for the supported MCU generation. The file names containing "_efr32xg1x_" are built to run on Series 1 MCUs and those containing "_efr32xg2x_" are built to run on Series 2 MCUs.

2.2.2 RAIL

The Wi-SUN stack uses RAIL to access the radio and RAIL libraries needs to be linked with Wi-SUN stack. RAIL has separate libraries for each device family and for single- and multi-protocol environments. RAIL libraries are provided in the Gecko SDK Suite. For more information refer to *UG103.13: RAIL Fundamentals* and other RAIL documentation.

2.2.3 EMLIB and EMDRV

The Wi-SUN stack uses EMLIB and EMDRV libraries to access EFR32 hardware. EMLIB and EMDRV peripheral libraries are provided in source code, and they must be included in the project. EMLIB and EMDRV are part of the Gecko SDK Suite. For more details on EMLIB and EMDRV, see platform EMDRV documentation and EMLIB documentation on https://docs.silabs.com.

2.2.4 Mbed TLS

The Wi-SUN stack uses the Mbed TLS library for cryptographic operations. The Mbed TLS library is provided in source code and must be included in the project. Mbed TLS is part of the Gecko SDK Suite. For more details, refer to the Mbed TLS documentation.

2.3 Optional Software Components

In addition to the Wi-SUN stack core functionality, the Wi-SUN SDK contains optional software components that you can leverage to customize the application. Add those components in the SOFTWARE COMPONENTS tab of a project, as shown in the following figure:

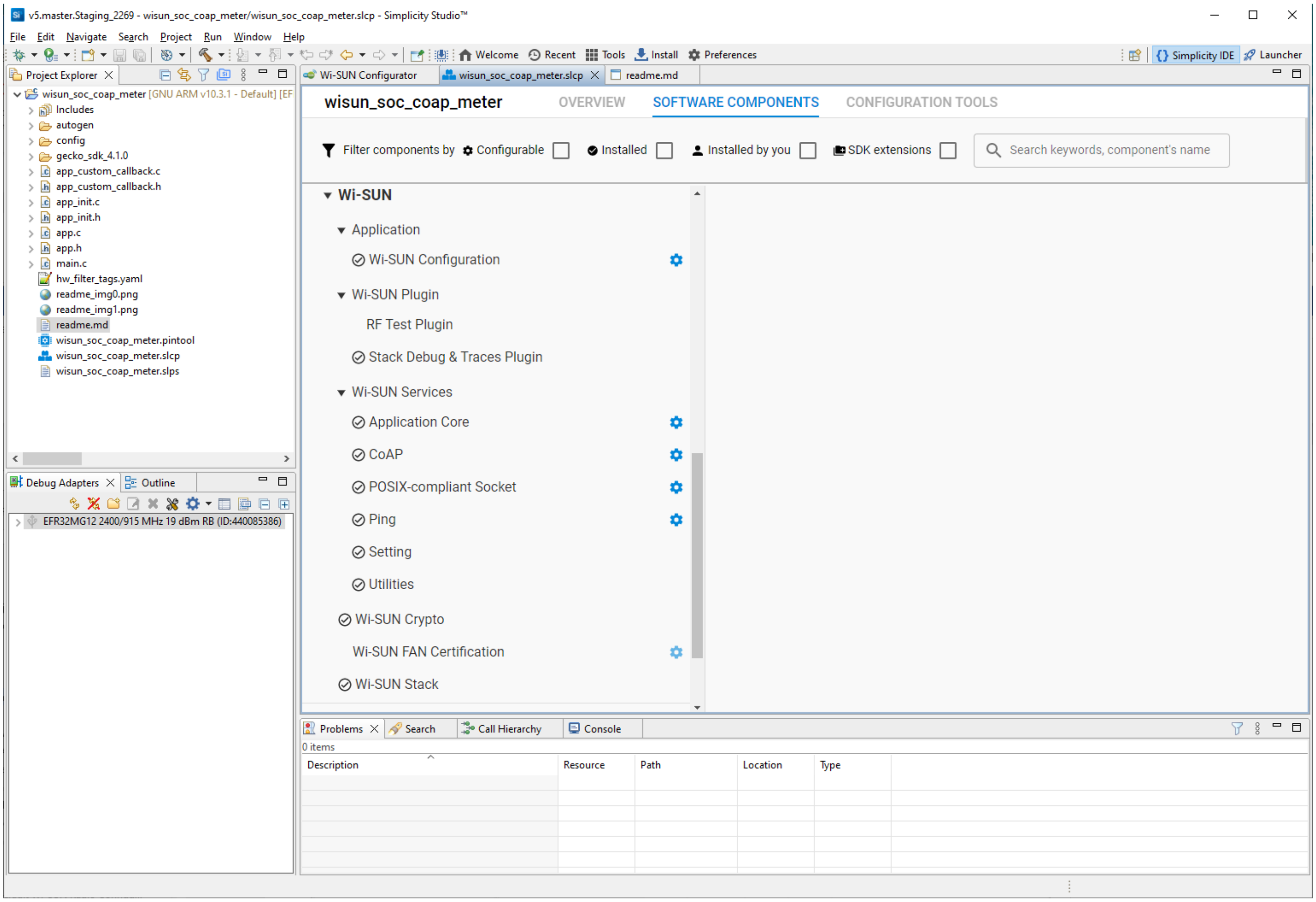


Figure 2.2. Wi-SUN Software Components

There are four important components:

- Application core: Provides application basic functionalities like event handling, callback management, and the Wi-SUN network connection.
- CoAP: Provides a Constrained Application Protocol (CoAP) implementation running on top of the Wi-SUN stack. The CoAP component should be used as an example of implementation of other software libraries on top of the Silicon Labs Wi-SUN stack.
- iPerf: Provides a widely supported tool to evaluate throughput on IP interfaces. The implementation can interoperate with other iPerf2 implementations. It currently only supports UDP server and client modes.
- POSIX-Compliant Socket: Provides a POSIX-like socket API on top of the standard Wi-SUN stack socket API. In addition to the API abstraction, this component makes the socket accesses thread-safe.

For the complete software documentation, visit https://docs.silabs.com/wisun/latest/wisun-stack-api/sl-wisun-services .

3 Wi-SUN Application Development

To get started with Wi-SUN application development, Silicon Labs recommends that you become familiar with different Wi-SUN sample applications. Then, you can use the Wi-SUN SoC Empty sample application as a template and a starting point for a new application.

The development of a Wi-SUN application consists of two main steps:

1. Responding to the events raised by the Wi-SUN stack.
2. Implementing additional application logic.

Optionally, you can change several Wi-SUN application settings with a few clicks:

1. Operating system used by the application.
2. IDE (Integrated Development Environment) used during the development.

3.1 Responding to Wi-SUN Events

A Wi-SUN application is event-driven. The Wi-SUN stack generates events when a connection is successful, data has been sent, or an IP packet is received. The application has to handle these events in the `sl_wisun_on_event()` function. The prototype of this function is implemented in app.c. To handle more events, the switch-case statement of this function can be created and extended. For the list of Wi-SUN events, visit https://docs.silabs.com.

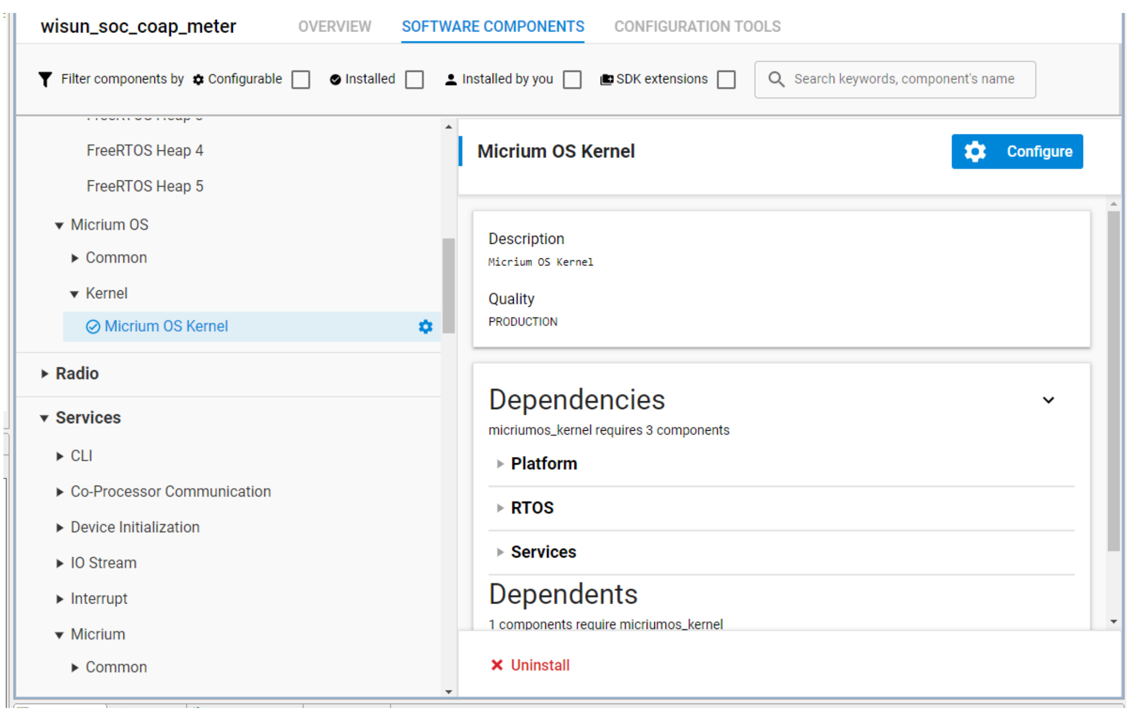
3.2 Implementing Application Logic

Additional application logic can be implemented in the `app_task()` function, defined in app.c. The `app_task()` function is called once after the device is booted and the Wi-SUN stack is initialized. Most Wi-SUN applications' first step is to call `sl_wisun_join()` to connect the Wi-SUN device to a Wi-SUN border router. The remaining implementation is up to the developer. Visit https://docs.silabs.com to check the list of Wi-SUN APIs available to the application.

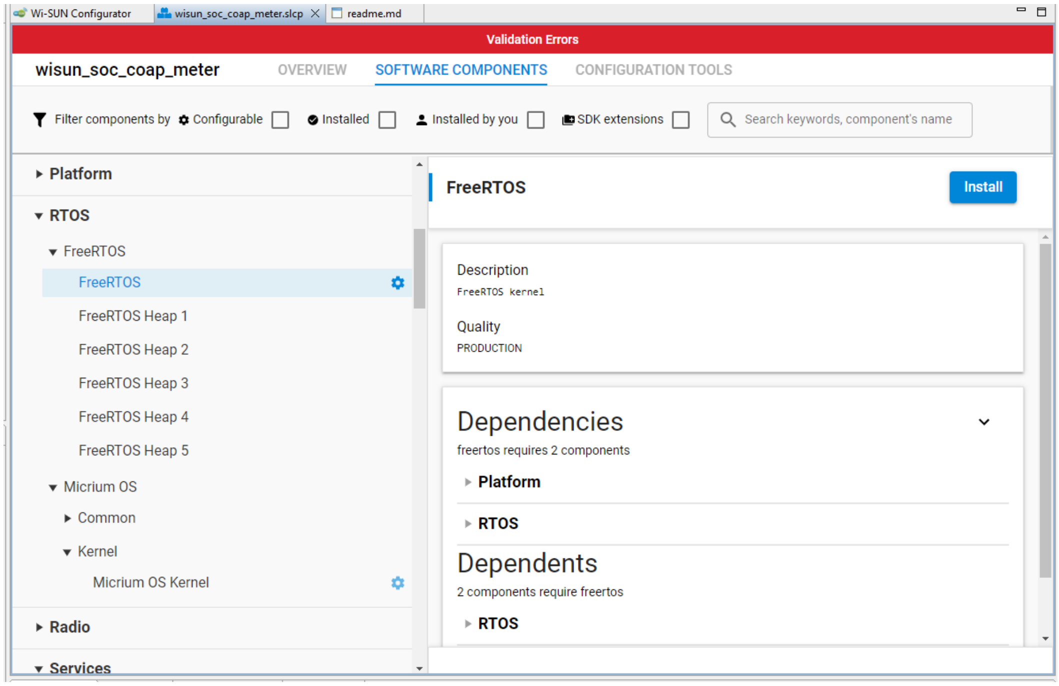
3.3 Changing Operating System

Simplicity Studio 5 provides the ability to easily replace software components. This feature is leveraged to change the Real-Time Operating System (RTOS) used by the application and the Wi-SUN stack. To change the RTOS, complete these steps:

1. Go to the project **SOFTWARE COMPONENTS** tab.
2. Uninstall the **Micrium OS Kernel** component (default RTOS).



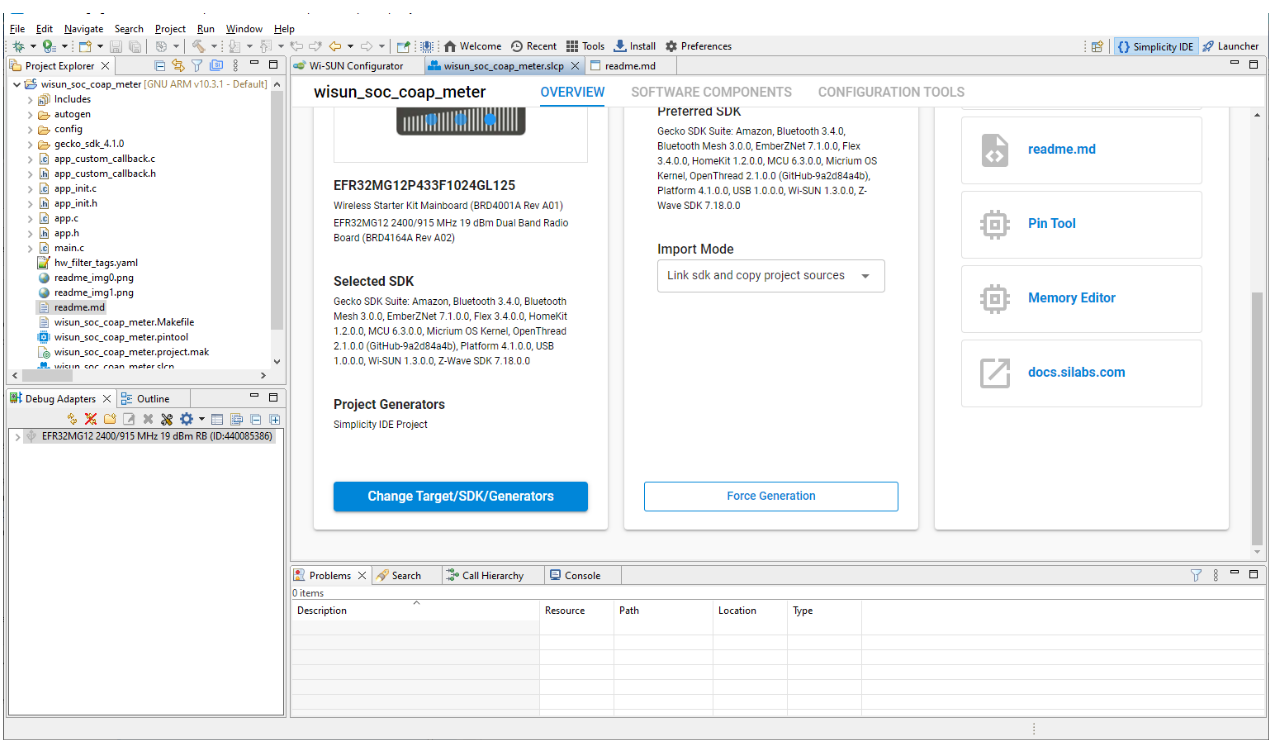
3. Install the **FreeRTOS** component.



3.4 Using a Different Development Environment

In addition to Simplicity Studio 5, you can use alternative Integrated Development Environment (IDEs). To generate a GCC makefile or an IAR Embedded Workbench project:

1. Go to the **OVERVIEW** tab.
2. Scroll down to the end of the Target and Tool Settings card and click **Change Target/SDK/Generators**.
3. In the CHANGE PROJECT GENERATORS list, select the type of projects to be generated.
4. Click **Save** and wait for Simplicity Studio to generate the project.



3.5 Wi-SUN Stack Heap Requirement

The Wi-SUN stack relies on dynamic memory allocation to function. It stores received and outgoing packets, security tokens, routing information, and more. The peak memory requirement is reached during the connection to a Wi-SUN network, especially during the authentication step. The heap size is defined under the *config/sl_memory_config.h* file in the Wi-SUN sample applications. By default, the heap size is configured through the SL_HEAP_SIZE define and is equal to 0x10000.

```
// <o SL_HEAP_SIZE> Minimum heap size for the application.
// <i> Default: 2048
// <i> Note that this value will configure the c heap which is normally used by
// <i> malloc() and free() from the c library. The value defines a minimum heap
// <i> size that is guaranteed to be available. The available heap may be larger
// <i> to make use of any memory that would otherwise remain unused.
#ifndef SL_HEAP_SIZE
  #define SL_HEAP_SIZE   0x10000

#endif
```

This heap size is largely inflated to accommodate potential application level requirements. The bare minimum heap size recommended to run the Wi-SUN stack is 0xC000.

In addition to the standard heap size requirement, the Wi-SUN stack relies on an RTOS: Micrium OS or FreeRTOS. The stack requires a number of tasks, queues, mutexes to be created. The size of this memory pool is defined either by:

- LIB_MEM_CFG_HEAP_SIZE when using Micrium OS
- configTOTAL_HEAP_SIZE when using FreeRTOS when using the heap_4 implementation

4 Mode Switch

The EFR32FG25 device supports the Wi-SUN FAN stack with the Mode Switch feature for OFDM and FSK PHYs specified in the Wi-SUN PHY Specification Revision 1VA8.

The purpose of Mode Switch is to allow data communication between nodes with a higher throughput PHY than the base PHY. Signaling still occurs using the base PHY. The Base PHY is always FSK and is set for the entire PAN. Mode switching can occur between the base (FSK) PHY and FSK or OFDM PHYs (with different MCS levels within the same OFDM option). POM-IEs (PHY Operating Mode Information Elements) are transmitted between neighboring nodes to mutually discover Mode Switching capabilities. Mode switching is only possible using PHYs listed in both nodes POM-IEs. Mode switching is signaled via the PHR (PHY Header) 'Mode Switch' field of a PPDU (PHY Protocol Data Unit) for the next PPDU, as per IEEE 802.15.4-2020 (section 'Mode Switch PHR').

To test Mode Switch for the OFDM or FSK modulations with the Wi-SUN stack, use the **Wi-SUN - SoC Border Router** project for the border router alongside the **Wi-SUN - SoC CLI** project for the node devices.

The mode switching feature can be used from the wisun_cli and wisun_brcli applications with the command:

```
wisun mode_switch [Mode Switch mode] [PhyModeID] [MAC Address]
```

The command parameters are:

- Mode switch mode:
  - 0 = Mode Switch disabled for the selected neighbor(s)
    - `wisun mode_switch 0` ~~`PhyModeID`~~ `[MAC Address]`
    - The [PhyModeID] parameter is ignored in this case.
    - MAC Address: MAC address of the neighbor(s). Either a single neighbor, or `ff:ff:ff:ff:ff:ff:ff:ff` for all neighbors
  - 1 = Mode Switch enabled with a specific PhyModeID for the selected neighbor(s)
    - `wisun mode_switch 1 PhyModeID ff:ff:ff:ff:ff:ff:ff:ff`
      - Required before using [Mode Switch mode] 2
      - When [MAC Address] is `ff:ff:ff:ff:ff:ff:ff:ff`, the global PhyModeID is set to [PhyModID] for all neighbors.
    - `wisun mode_switch 1 PhyModeID [MAC Address]`
    - PhyModeID: PhyModeID to switch to
    - MAC Address: MAC address of the neighbor(s). Either a single neighbor, or `ff:ff:ff:ff:ff:ff:ff:ff` for all neighbors
  - 2 = Mode Switch enabled with the global PhyModeID for a unique neighbor
    - `wisun mode_switch 2` ~~`PhyModeID`~~ `[MAC Address]`
      - The [PhyModeID] parameter is ignored in this case.
      - `ff:ff:ff:ff:ff:ff:ff:ff` cannot be used.
        - The global PhyModeID must have been previously set with `wisun mode_switch 1` ~~`xx`~~ `ff:ff:ff:ff:ff:ff:ff:ff`.
        - MAC Address: MAC address of the neighbor

Note: Mode Switch only applies to unicast data frames. Multicast traffic always happens with the base PHY.

4.1 Fallback Mechanism

When Mode Switching fails too often with a neighbor (4 times more retries than successes), Mode Switching is disabled for this neighbor.

4.2 Mode Switching in action

Assume 3 neighbors:

1. `01:02:03:04:05:06:07:08`
2. `11:12:13:14:15:16:17:18`
3. `21:22:23:24:25:26:27:28`

1- To use PhyModeId 34 (OFDM option 1, MCS2) globally (for all existing neighbors), the command is:

```
wisun mode_switch 1 34 ff:ff:ff:ff:ff:ff:ff:ff
```

2- To use PhyModeID 52 (OFDM option 2, MCS4) for the first and second neighbors only, two commands are necessary:

```
wisun mode_switch 1 52 01:02:03:04:05:06:07:08
wisun mode_switch 1 52 11:12:13:14:15:16:17:18
```

3- To use the global PhyModeID (without the need to specify it) for the second neighbor, the command is:

```
wisun mode_switch 2 xx 11:12:13:14:15:16:17:18
```

4- To disable mode switch for the first neighbor, the command is:

```
wisun mode_switch 0 xx 01:02:03:04:05:06:07:08
```

5- To disable mode switch for all neighbors using the global PhyModeID, the command is:

```
wisun mode_switch 0 xx ff:ff:ff:ff:ff:ff:ff:ff
```

4.3 Mode switch PHYs

The Wi-SUN stack can switch from the base PhyModeIDs listed below to a set of higher throughput PhyModeIDs, for selected regulatory domains where Mode Switch is specified:

4.3.1 Regulatory Domain NA

Base PHY:

- PhyModeID 2 (50 kbps FSK, operating mode 1b)

Possible higher throughput PHYs from PhyModeID 2; Possible higher throughput PHYs:

- PhyModeID 5 (150 kbps FSK)
- PhyModeID 6 (200 kbps FSK)
- PhyModeID 8 (300 kbps FSK)
- PhyModeID 34 (400 kbps OFDM option 1, MCS2)
- PhyModeID 35 (800 kbps OFDM option 1, MCS3)
- PhyModeID 36 (1200 kbps OFDM option 1, MCS4)
- PhyModeID 37 (1600 kbps OFDM option 1, MCS5)
- PhyModeID 38 (2400 kbps OFDM option 1, MCS6)

4.3.2 Regulatory Domain EU

Base PHY:

- PhyModeID 1 (50 kbps FSK, operating mode 1a)

Possible higher throughput PHYs from PhyModeID 1:

- PhyModeID 3 (100 kbps FSK, operating mode 2a)
- PhyModeID 5 (150 kbps FSK, operating mode 3)
- PhyModeID 84 (150 kbps OFDM option 1, MCS4)
- PhyModeID 85 (200 kbps OFDM option 1, MCS5)
- PhyModeID 86 (300 kbps OFDM option 1, MCS6)

5 OFDM PHY Switching

The Wi-SUN stack is also able to switch dynamically between OFDM PHYs within a given OFDM option, that is selecting another MCS rate in the same OFDM Option. This is independent from the PPDU mode switch feature.

The purpose here is to reduce the transmission time to save power by selecting the most efficient MCS the link can support.

NB: A higher bandwidth MCS, even with a significant EVM level, can be more power efficient than a lower bandwidth MCS with no Tx errors, if the data can be corrected at the reception side.

5.1 Regulatory Domain NA, OFDM Option 1

Possible OFDM PHYs:

- PhyModeID 34 (400 kbps OFDM option 1, MCS2)
- PhyModeID 35 (800 kbps OFDM option 1, MCS3)
- PhyModeID 36 (1200 kbps OFDM option 1, MCS4)
- PhyModeID 37 (1600 kbps OFDM option 1, MCS5)
- PhyModeID 38 (2400 kbps OFDM option 1, MCS6)

5.2 Regulatory Domain NA, OFDM Option 2

Possible OFDM PHYs:

- PhyModeID 51 (400 kbps OFDM option 2, MCS3)
- PhyModeID 52 (600 kbps OFDM option 2, MCS4)
- PhyModeID 53 (800 kbps OFDM option 2, MCS5)
- PhyModeID 54 (1200 kbps OFDM option 2, MCS6)

5.3 Regulatory Domain NA, OFDM Option 3

Possible OFDM PHYs:

- PhyModeID 68 (300 kbps OFDM option 3, MCS4)
- PhyModeID 69 (400 kbps OFDM option 3, MCS5)
- PhyModeID 70 (600 kbps OFDM option 3, MCS6)

5.4 Regulatory Domain NA, OFDM Option 4

Possible OFDM PHYs:

- PhyModeID 84 (150 kbps OFDM option 4, MCS4)
- PhyModeID 85 (200 kbps OFDM option 4, MCS5)
- PhyModeID 86 (300 kbps OFDM option 4, MCS6)

5.5 Regulatory Domain EU, OFDM Option 4

Possible OFDM PHYs:

- PhyModeID 84 (150 kbps OFDM option 4, MCS4)
- PhyModeID 85 (200 kbps OFDM option 4, MCS5)
- PhyModeID 86 (300 kbps OFDM option 4, MCS6)

5.6 Fallback Mechanism

When using the higher throughput MCS fails, the receiver will not send an ack to the sender, which will then select a lesser MCS rate, making reception of the next packets easier.

6 Testing and Debugging

6.1 Access Debug Traces from the Wi-SUN Stack

The Wi-SUN stack provides a logging mechanism based on the Segger RTT feature to allow a finer tracing capability. To access the Wi-SUN stack RTT traces:

1. Install the J-Link RTT Viewer.
2. Open the J-Link RTT Viewer.
3. In the **Configuration** panel, **Connection to J-Link** section, select **USB**.
4. In the **Specify Target Device** list, select the connected part (for example EFR32MG12PXXXF1024). EFR32FG25 is not yet known to the RTT Viewer, so use any EFR32MG12 instead.
5. In the **Target Interface & Speed** panel, select **SWD** and **4000 kHz**.
6. In the **RTT Control Block** panel, select **Auto Detection**.
7. Click **OK**.
8. If you have several boards connected, a list appears. When you need to monitor several devices, open an instance of RTT viewer per device.
9. Select a WSTK board running the Wi-SUN stack (border router or node).
10. Click **OK**.

A terminal opens and the Wi-SUN stack traces are output as shown below.

```
[DBG ][wisun]: net_init_core: 0
[DBG ][wisun]: sli_wisun_task_event_handler_id: 2
[DBG ][SLRF]: sli_wisun_driver_register()
[DBG ][SLRF]: sli_wisun_driver_register() - driver_id: 0
[DBG ][SLRF]: rf_address_write: PHY_MAC_64BIT: 00:0d:6f:ff:fe:20:bd:95
[DBG ][mlme]: SW-MAC driver support rf extension 50000 symbol/seconds  20 us symbol time length
[DBG ][swm ]: Set MAC mode to IEEE 802.15.4-2011, MTU size: 127
[DBG ][SLRF]: rf_address_write: PHY_MAC_64BIT: 00:0d:6f:ff:fe:20:bd:95
[DBG ][wisun]: arm_nwk_interface_lowpan_init: 1
```

The application must install the 'Third Party/Segger/RTT/SEGGER RTT' component to generate RTT traces from the Wi-SUN stack. The component uses a configurable RTT trace 'channel' (0 by default).

The Trace and Debug component provides APIs to set the Wi-SUN stack traces level and filters. Visit Wi-SUN Stack traces and debug API for more information.

These logs can be used to report an issue to Silicon Labs support.

NB: RTT Viewer needs to be disconnected from a device to allow flashing again. The most convenient way to quickly disconnect is to press 'F3', flash, then press 'F2' once ready to reconnect. RTT viewer will use previous settings by default.

6.2 Export Wi-SUN Air Capture Traces to Wireshark

The Wi-SUN traces export feature requires Simplicity Studio 5.1.0 or higher. It relies on the 'Platform/Radio/RAIL Utility, PTI' component.

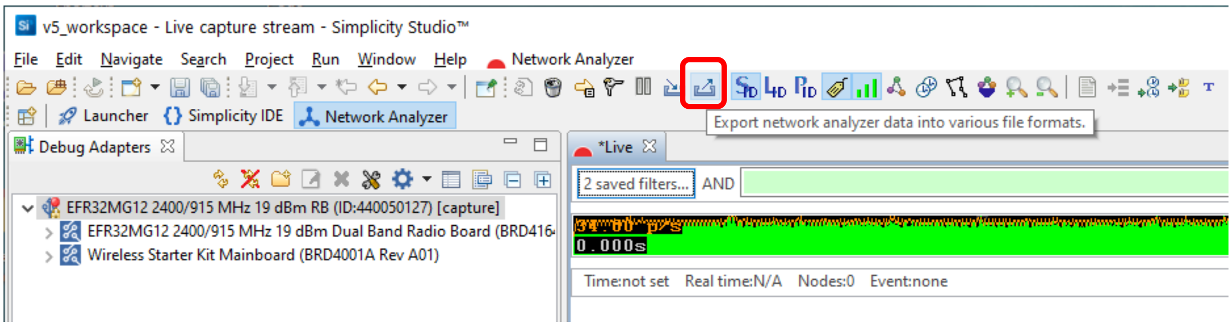
Simplicity Studio's Network Analyzer enables debugging of complex wireless systems on a number of Silicon Labs part families. Network Analyzer includes a partial Wi-SUN protocol analyzer (that is, the Wi-SUN payload cannot be decrypted). However, it can be used to export traces to another analyzer like Wireshark.

Beginning with GSDK 4.2.1, the PTI baud Rate has been changed from the default 1600000 to 3200000. Since all out-of-box WSTKs are set by default with 1600000, you need to change this once per WSTK in the 'Admin' tab, using '`pti config 0 efruart 3200000`'. Check the results using '`pti config`'. The hardware will select the closest possible baud rate and show it in the (current) section.

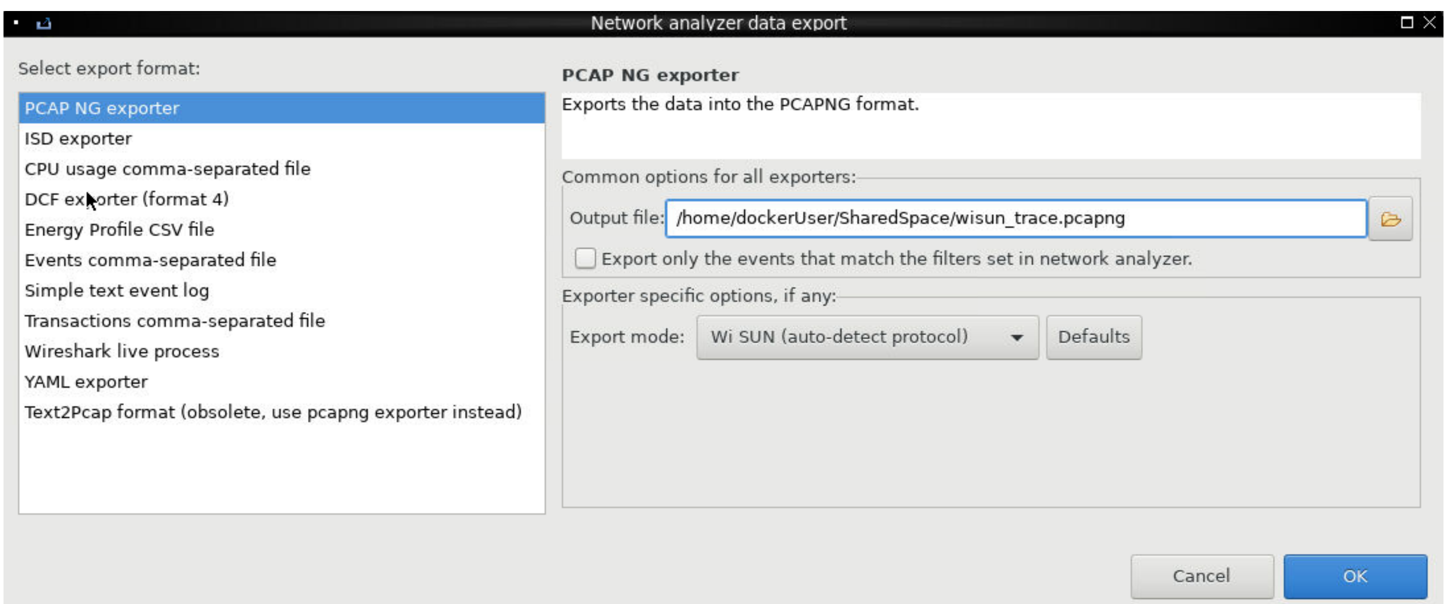
```
WSTK> WSTK> pti config 0 efruart 3200000
Configuration successful!
WSTK> WSTK> pti config
PTI enabled         : Yes
PTI configured      : Yes

----- PTI config (current)-----
Interface            : 0
Line protocol        : EFR UART
Bitrate              : 3230769

----- PTI config (stored)-----
Interface            : 0
Line protocol        : EFR UART
Bitrate              : 3200000
```

To export Wi-SUN traces with the Network Analyzer to Wireshark, install Wireshark and follow this procedure in Simplicity Studio 5:

1. In the Simplicity IDE perspective, **Debug Adapter** view, right-click an EFR32xG12 running the Wi-SUN stack.
2. Select **Start Capture**.
3. A Live tab opens in the Editor area. It traces packets sent and received by the Wi-SUN device.
4. When you have traced the communication, click **Export**.



5. Under **Select export format**, select **PCAP NG exporter**,
6. Enter a path and a file name in which to store the trace.
7. In **Export mode**, select **Wi-SUN (auto-detect protocol)**.
8. Click **OK**.



Open the new file in Wireshark. Wireshark should automatically analyze the file as a Wi-SUN exchange. The communication is initially encrypted thanks to the Wi-SUN encryption protocol. To decrypt the communications, the GAK key and key index set information are required. They can be retrieved on the border router CLI by issuing the following command:

```
> wisun get wisun

wisun.state = started (2)
wisun.network_name = "Wi-SUN Network"
wisun.phy_config_type = FAN 1.0 (0)
wisun.regulatory_domain = NA (1)
wisun.operating_class = 1
wisun.operating_mode = 0x1b
wisun.fec = 0
wisun.chan_plan_id = 1 (unused)
wisun.phy_mode_id = 2 (unused)
wisun.ch0_frequency = 863100 (unused)
wisun.number_of_channels = 69 (unused)
wisun.channel_spacing = 100kHz (0) (unused)
wisun.network_size = small (1)
wisun.tx_power = 20
wisun.unicast_dwell_interval = 255
wisun.broadcast_interval = 1020
wisun.broadcast_dwell_interval = 255
wisun.channel_mask =
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
wisun.allowed_channels = "0-255"
wisun.gak1 = 76:ab:9f:ce:8a:08:b8:88:6d:fc:56:fa:0e:6f:1e:ce
wisun.gak2 = 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
wisun.gak3 = 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
wisun.gak4 = 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
wisun.lfn_gak1 = 7b:92:f2:9e:68:a5:41:37:b3:5e:e7:14:04:be:37:04
wisun.lfn_gak2 = 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
wisun.lfn_gak3 = 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
wisun.mac_address = b4:e3:f9:ff:fe:c5:84:93
wisun.ip_addresses = [ll: fe80::b6e3:f9ff:fec5:8493 gua: :: dodagid: ::]
wisun.trace_filter.000-031 = 0xffffffff
wisun.trace_filter.032-063 = 0xffffffff
wisun.regulation = none (0)
wisun.rx_phy_mode_ids = 0x02,0x05,0x06,0x08,0x22,0x23,0x24,0x25,0x26
```

On Wi-SUN Border Router Linux, the GAK Key can be retrieved using wsbrd_cli or in /var/lib/wstk/network-keys:

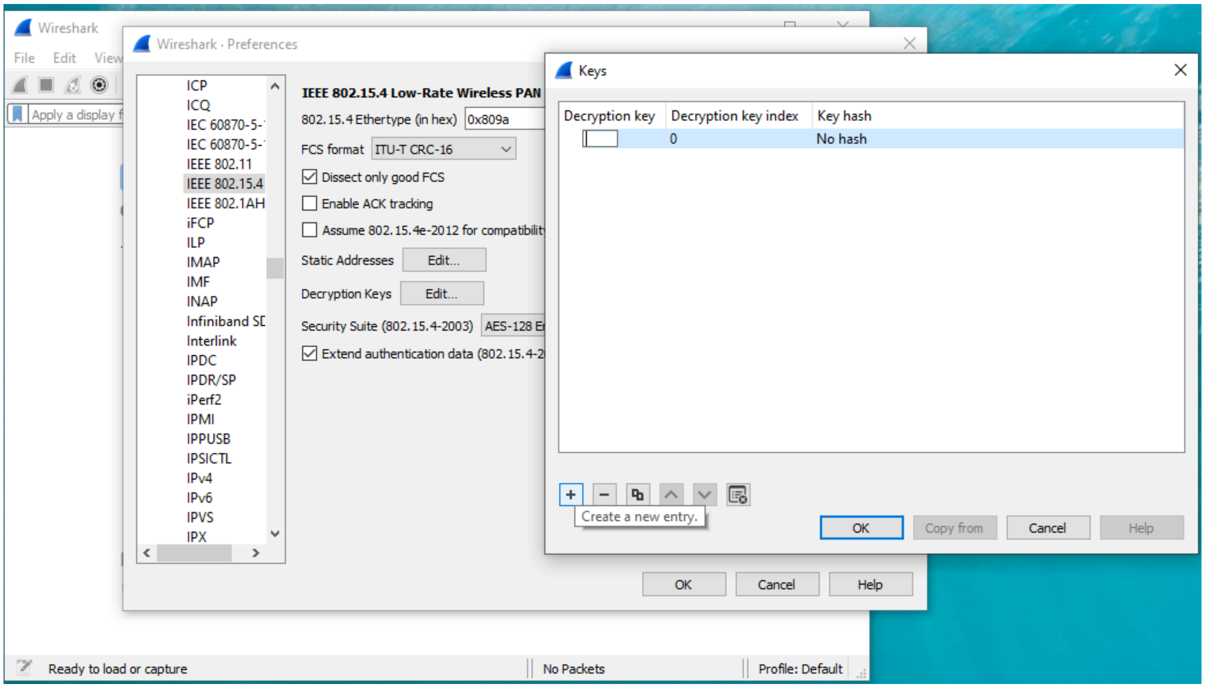
```
$ wsbrd_cli status
network_name: Wi-SUN Network
domain: EU
mode: 1a
class: 1
panid: 0x3287
size: SMALL
GAK[0]: eb:7b:11:dd:f0:23:c3:7e:36:64:c5:9b:5b:ed:c6:c7
GAK[1]: fc:d3:1f:a7:4c:c8:c4:0f:96:7e:d0:5d:0e:a1:01:fc
GAK[2]: fc:d3:1f:a7:4c:c8:c4:0f:96:7e:d0:5d:0e:a1:01:fc
GAK[3]: fc:d3:1f:a7:4c:c8:c4:0f:96:7e:d0:5d:0e:a1:01:fc
GTK[0]: a8:d6:dc:7c:66:14:2f:b8:db:dd:af:fb:c2:27:0c:18
GTK[1]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GTK[2]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GTK[3]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
84:fd:27:ff:fe:fe:55:bd
```

Silicon Labs Wi-SUN devices in the network use `wisun.gak1` as the GAK key. In the first capture the GAK key is `76:ab:9f:ce:8a:08:b8:88:6d:fc:56:fa:0e:6f:1e:ce` and in the second capture the GAK key is `eb:7b:11:dd:f0:23:c3:7e:36:64:c5:9b:5b:ed:c6:c7`. The key index to use in Wireshark is 1 in both cases, for the first GAK in the list.

In Wireshark:

1. Click **Edit**.
2. Click **Preferences...**
3. Expand the **Protocols** list and select **IEEE 802.15.4.**
4. Next to Decryption Keys, click **Edit**.

5. In the Keys window, click **+** (plus).
6. Under **Decryption key** enter the GAK key, and under **Decryption key index** enter the key index (starting at 1 for GAKs, starting at 5 for LFN-GAKs).
7. Click **OK**.



Wireshark is now able to decrypt the traces and the higher-level protocols (ICMP, TCP, UDP...). The following example of traces shows a router pinging its border router.

348	-140463039.786897	00:0d:6f:ff:fe:20:b6:f9		0.025546000	Wi-SUN	105
349	-140463039.761292	00:0d:6f:ff:fe:20:b6:f9		0.025605000	Wi-SUN	105
350	-140463039.735743	00:0d:6f:ff:fe:20:b6:f9		0.025549000	Wi-SUN	105
351	-140463065.782571	fd00:7283:7e00:0:20d:6fff:fe…	fd00:6172:6d00:0:20d:6fff:fe20:bd45	25.953172000	ICMPv6	151
352	-140463065.753483	00:0d:6f:ff:fe:20:bd:45	00:0d:6f:ff:fe:20:b6:f9	0.029088000	Wi-SUN	44
353	-140463065.332494	fd00:6172:6d00:0:20d:6fff:fe…	fd00:7283:7e00:0:20d:6fff:fe20:b6f9	0.420989000	ICMPv6	151
354	-140463065.306434	00:0d:6f:ff:fe:20:b6:f9	00:0d:6f:ff:fe:20:bd:45	0.026060000	Wi-SUN	44
355	-140463065.104863	fd00:6172:6d00:0:20d:6fff:fe…	fd00:7283:7e00:0:20d:6fff:fe20:b6f9	0.201571000	ICMPv6	151
356	-140463065.078796	00:0d:6f:ff:fe:20:b6:f9	00:0d:6f:ff:fe:20:bd:45	0.026067000	Wi-SUN	44
357	-140463067.250051	00:0d:6f:ff:fe:20:bd:45		1.828745000	Wi-SUN	105
358	-140463069.782009	00:0d:6f:ff:fe:20:bd:45		1.468042000	Wi-SUN	105
359	-140463073.927166	fd00:7283:7e00:0:20d:6fff:fe…	fd00:6172:6d00:0:20d:6fff:fe20:bd45	3.854843000	ICMPv6	151
360	-140463073.898023	00:0d:6f:ff:fe:20:bd:45	00:0d:6f:ff:fe:20:b6:f9	0.029143000	Wi-SUN	44
361	-140463073.880264	fd00:6172:6d00:0:20d:6fff:fe…	fd00:7283:7e00:0:20d:6fff:fe20:b6f9	0.017759000	ICMPv6	151
362	-140463073.854198	00:0d:6f:ff:fe:20:b6:f9	00:0d:6f:ff:fe:20:bd:45	0.026066000	Wi-SUN	44
363	-140463076.612640	fd00:7283:7e00:0:20d:6fff:fe…	fd00:6172:6d00:0:20d:6fff:fe20:bd45	3.241558000	ICMPv6	151
364	-140463076.557104	fd00:7283:7e00:0:20d:6fff:fe…	fd00:6172:6d00:0:20d:6fff:fe20:bd45	0.055536000	ICMPv6	151
365	-140463076.142182	fd00:6172:6d00:0:20d:6fff:fe…	fd00:7283:7e00:0:20d:6fff:fe20:b6f9	0.414922000	ICMPv6	151
366	-140463076.116118	00:0d:6f:ff:fe:20:b6:f9	00:0d:6f:ff:fe:20:bd:45	0.026064000	Wi-SUN	44
367	-140463077.809511	fd00:7283:7e00:0:20d:6fff:fe…	fd00:6172:6d00:0:20d:6fff:fe20:bd45	0.306607000	ICMPv6	151
368	-140463077.740374	00:0d:6f:ff:fe:20:bd:45	00:0d:6f:ff:fe:20:b6:f9	0.069137000	Wi-SUN	44

The PTI output is limited in number of bytes per messages. Packets above 1022 bytes are truncated using the WSTK Firmware 1v4p0 or later (200 bytes with earlier versions). This cannot be increased with WSTKs due to hardware limitations on BRD4001. It will be improved for Wireless Pro Kits (WPKs) once the proper firmware is available.

6.3 Connect the Wi-SUN Network to Another IP Network

Refer to the steps in *AN1332: Silicon Labs Wi-SUN Network Setup and Configuration* to open a backhaul connection from the Wi-SUN border router.