AN1333: Running Zigbee, OpenThread, and Bluetooth Concurrently on a Linux Host with a Multiprotocol Co-Processor

This document describes how to run any combination of Zigbee EmberZNet, OpenThread, and Bluetooth networking stacks on a Linux host processor, interfacing with a single EFR32 Radio Co-processor (RCP) with multiprotocol and multi-PAN support. The intended use case is for gateway products that wish to run any combination of the three protocols on the Linux host processor with a single, shared EFR32 RCP. It also describes how to run the Zigbee stack on the EFR32 as a network co-processor (NCP) alongside the OpenThread RCP or the Bluetooth NCP. Each stack can use the co-processor to communicate simultaneously and independently. The Zigbee and OpenThread stacks operate on separate 802.15.4 PANs, but they must be on the same 802.15.4 channel. The Zigbee NCP with OpenThread RCP or Bluetooth NCP configurations are experimental quality. Silicon Labs will continue to test and welcomes your comments.

KEY POINTS

- Reviews the components of the Multi-protocol and Multi-PAN RCP solution
- Explains how to set up both host and co-processor
- Describes how to run the Zigbee EmberZNet, OpenThread, and BlueZ stacks on a Linux host
- Covers how to run Zigbee as an NCP alongside the OpenThread RCP.

The RCP solution was designed so that existing Zigbee host applications built to work with a Zigbee Network Co-processor (NCP) can continue to run with little or no modifications when the Zigbee stack runs on the host.

1 System Architecture

1.1 Overview

The system architecture includes various software components:

- A co-processor image that runs on the EFR32 co-processor.
- A Linux host process called Co-processor Communication Daemon (CPCd) that communicates with the co-processor over a Universal Asynchronous Receiver/Transmitter (UART) or Serial Physical Interface (SPI) physical link, and multiplexes protocol streams.
- A Linux daemon called Zigbeed that runs the Zigbee stack and sends and receives Spinel messages to CPCd over a socket when running in RCP mode.
- A Zigbee host application that communicates with Zigbeed using EmberZNet Serial Protocol / Asynchronous Serial Host (EZSP / ASH) over a virtual serial port, or directly to CPCd in the case of the Zigbee NCP/OpenThread RCP.
- An OpenThread host application such as the OpenThread Border Router (otbr-agent), which includes the OpenThread protocol stack and which connects to CPCd over a socket and operates on a separate PAN from the Zigbee network.
- The BlueZ Bluetooth stack, which communicates with the Bluetooth Controller on the RCP via the Host Controller Interface (HCI) protocol. A small cpc-hci-bridge program allows the HCI commands to be transported to the RCP via CPCd.

The following figure illustrates the system architecture for the RCP:

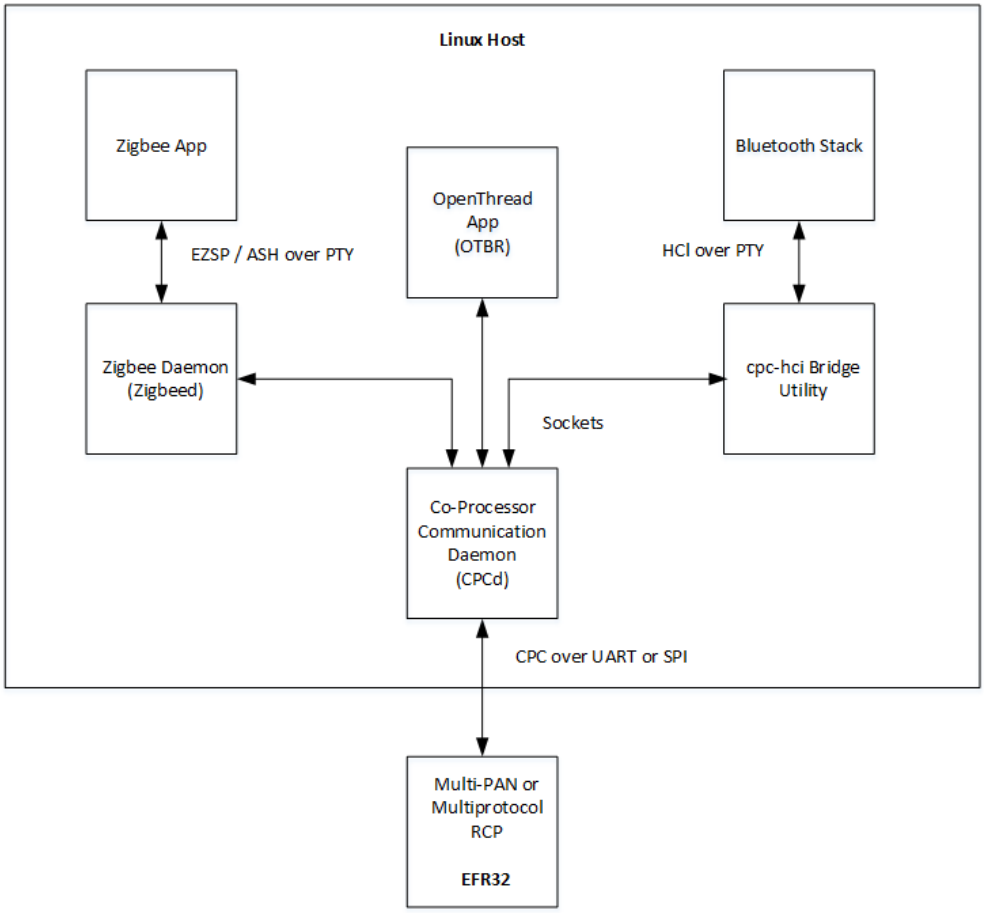


Figure 1-1. System Architecture for the Multiprotocol RCP

The following figure illustrates the system architecture for the Zigbee NCP/OpenThread RCP.

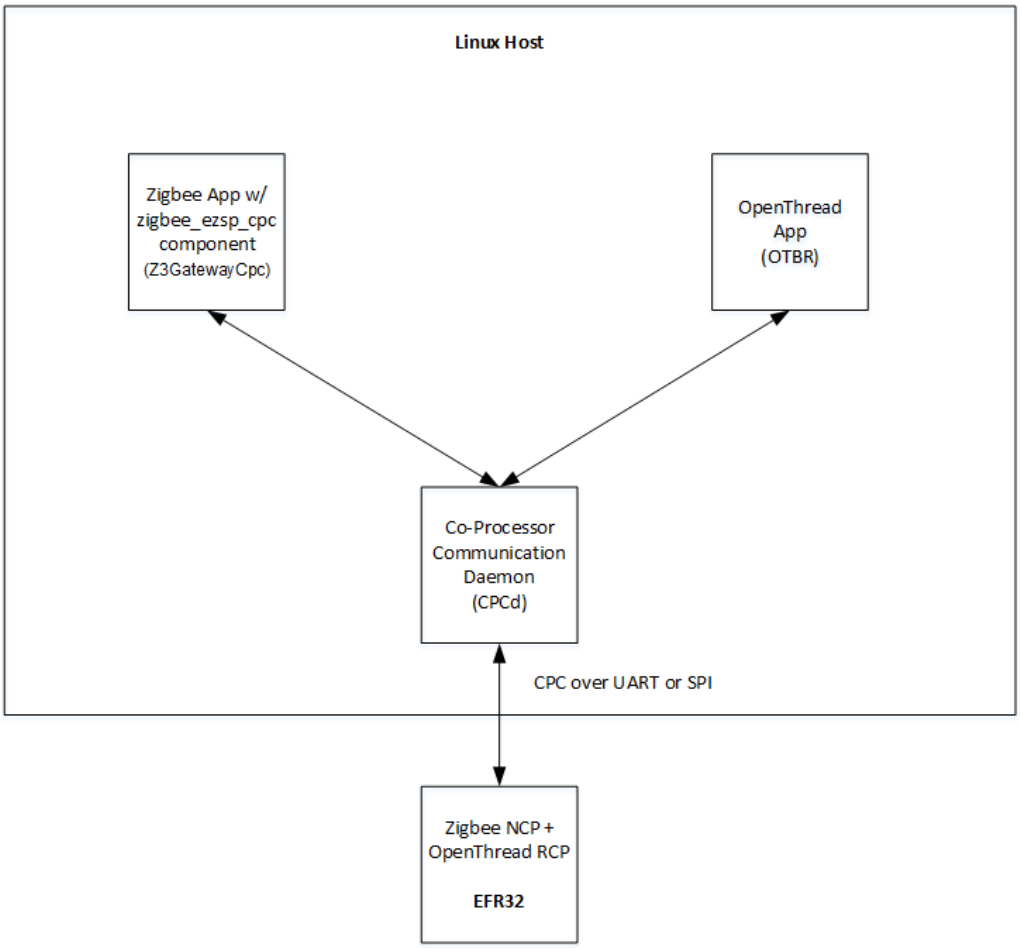


Figure 1-2. System Architecture for the Zigbee NCP + OpenThread RCP

The following figure illustrates the system architecture for the Dynamic Multiprotocol (DMP) Zigbee + Bluetooth NCP.

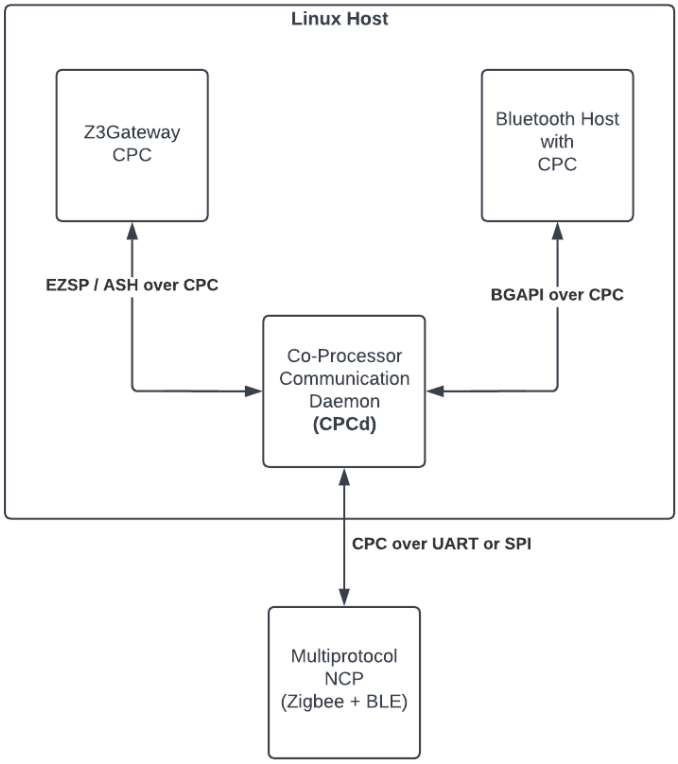


Figure 1-3. System Architecture for the Zigbee NCP + BLE NCP

1.2 EFR32 Software

The EFR32 co-processor image comes in four flavors:

1. The multi-PAN 802-15.4 RCP is based on the OpenThread 802.15.4 RCP with added multi-PAN and CPC support. It has a small flash footprint (~150K) and uses the Spinel protocol to serialize commands. The Spinel messages are further encapsulated by the CPC protocol before being sent over the physical link. Both UART and SPI links are supported. This is illustrated in Figure 1-1. System Architecture for the Multiprotocol RCP (ignoring the Bluetooth components). The project files are **rcp-uart-802154.slcp** and **rcp-spi-802154.slcp**.
2. The multiprotocol RCP adds the Bluetooth Controller and FreeRTOS to the above 802.15.4 RCP. It has a larger flash footprint (~250k). HCI is used to serialize Bluetooth commands over CPC. Both UART and SPI links are supported. See Figure 1-1. System Architecture for the Multiprotocol RCP. The project files are **rcp-uart-802154-blehci.slcp** and **rcp-spi-802154-blehci.slcp**.
3. The Zigbee NCP with OpenThread RCP runs the Zigbee stack on the EFR32 alongside the OpenThread RCP. The Zigbee application still runs on the host, and uses EZSP to send commands to the Zigbee NCP over CPC. Note that this solution does not make use of Zigbeed, because the Zigbee stack is running on the EFR32, not on the host. OpenThread runs on the host as in the other cases, and uses Spinel over CPC to communicate with the OpenThread RCP. Both UART and SPI links are supported. See Figure 1-2. System Architecture for the Zigbee NCP + OpenThread RCP. The project files are **zigbee_ncp-ot_rcp-uart.slcp** and **zigbee_ncp-ot_rcp-spi.slcp**.
4. The multiprotocol NCP runs Zigbee and BLE stacks on the EFR32. The Zigbee application (Z3GatewayCPC) runs on the Linux host and communicates with the NCP using EZSP over CPC (SPI and UART are both available options). The Bluetooth host app `bt_host_empty` is present in the container and is compiled with CPC=1 option to enable communication with the DMP NCP over CPC. Note that the -R option must be used when running the Bluetooth host to prevent it from rebooting the NCP as part of the startup sequence. For convenience, after running the container you can start the `bt_host_empty` app with the command `run.sh -B`. See Figure 1-3. System Architecture for the Zigbee NCP + BLE NCP.

1.3 Host Software

The Co-Processor Communication Daemon (CPCd) enables users to have multiple stack protocols interact with the radio co-processor over a shared physical link. CPCd is distributed as three components: the daemon binary **cpcd**; a library **libcpc.so** that enables C applications to interact with the daemon; and a configuration file **cpcd.conf**. In CPC, data transfers between processors are segmented in sequential packets. Transfers are guaranteed to be error-free and sent in order. Multiple applications can send or receive on the same endpoint without worrying about collisions. A library *libcpc.so* is provided to simplify the interaction between the user application and the daemon via Unix domain sockets. Code to interface the OpenThread Spinel driver to libcpc.so is provided as part of this solution. For more information on CPCd, see https://github.com/SiliconLabs/cpc-daemon/blob/main/readme.md.

The Zigbee Daemon (Zigbeed) runs the Zigbee networking stack on the host. It is built with a Spinel adaptation layer that translates between 802.15.4 MAC primitives and Spinel messages, and uses the Spinel driver-to-libcpc.so interface code referred to above. Spinel messages from Zigbeed are sent to CPCd where they are encapsulated and forwarded to the RCP. Zigbeed also opens a virtual serial port for communicating with the host application using the EZSP/ASH protocol.

The **socat** command line utility is used to create two virtual serial ports (PTY) and link them to each other. This allows a Zigbee host application that was built for a Zigbee NCP co-processor to interface with Zigbeed unchanged, simply by supplying the proper device name to it.

Both Zigbeed and the OpenThread stack can connect to CPCd and use the multi-PAN RCP at one time. Spinel messages for each application are labelled with a Spinel Interface ID (IID) which is supplied to the application at startup via the OpenThread Radio URL command line argument. The fact that the RCP is being shared between multiple PANs is transparent to the host applications. The only requirement is that all PANs must operate on the same 802.15.4 channel. Coordination must happen between applications and no mechanism is provided as part of this solution.

On xG24 and xG21 parts, the stacks may operate on separate channels when using the RCP. This feature is called Concurrent Listening and is experimental quality in GSDK 4.3.0.

Zigbeed stores non-volatile Zigbee stack tokens in a file called *host_token.nvm*. This allows Zigbeed to retain network information across resets.

For the Zigbee NCP with OpenThread RCP, the Zigbee host app and the OpenThread stack can also connect to CPCd and use the co-processor at the same time. They must coordinate at the application layer to ensure that both operate on the same 802.15.4 channel. Concurrent Listening is not yet available for the Zigbee NCP + OpenThread RCP.

2 System Setup

2.1 EFR32 Co-processor Setup

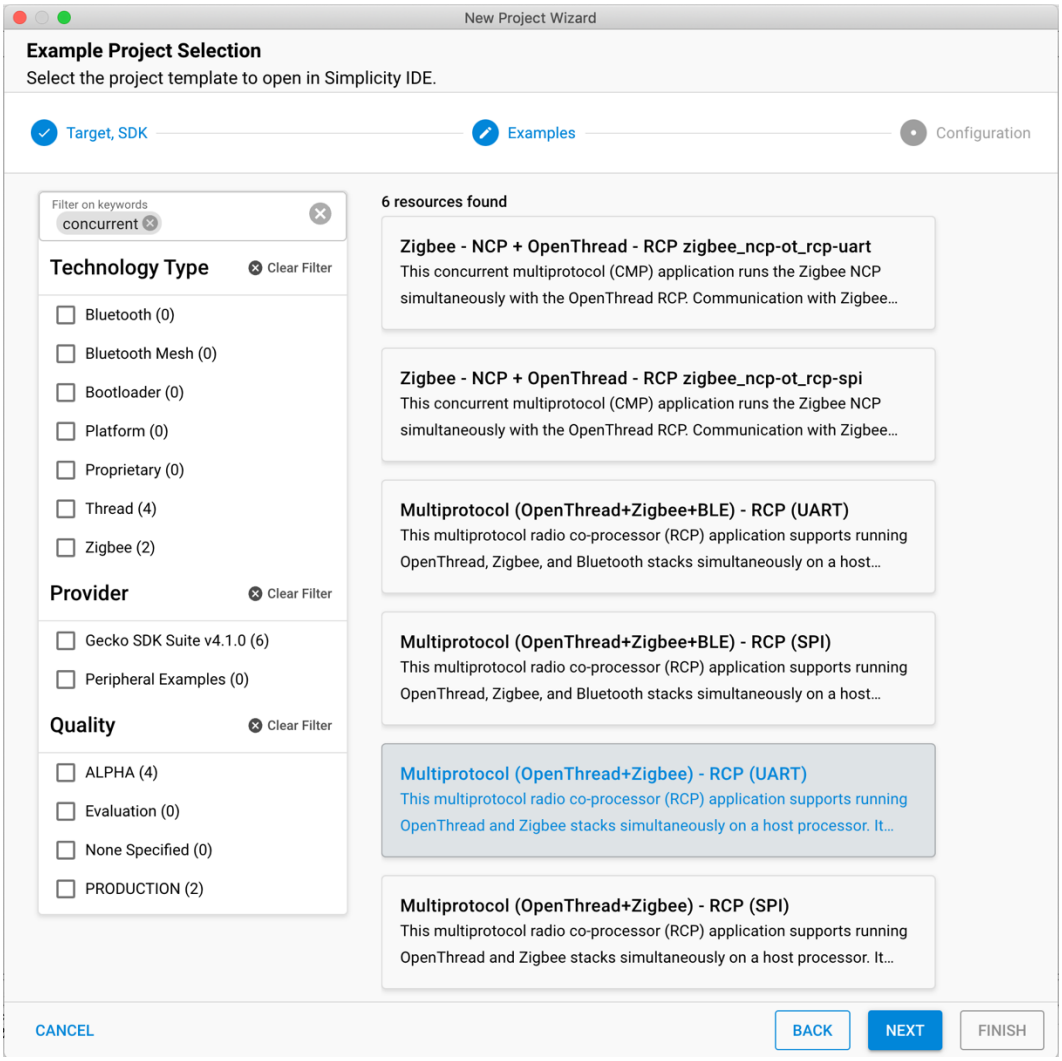
Simplicity Studio has precompiled demo RCP projects for certain boards.

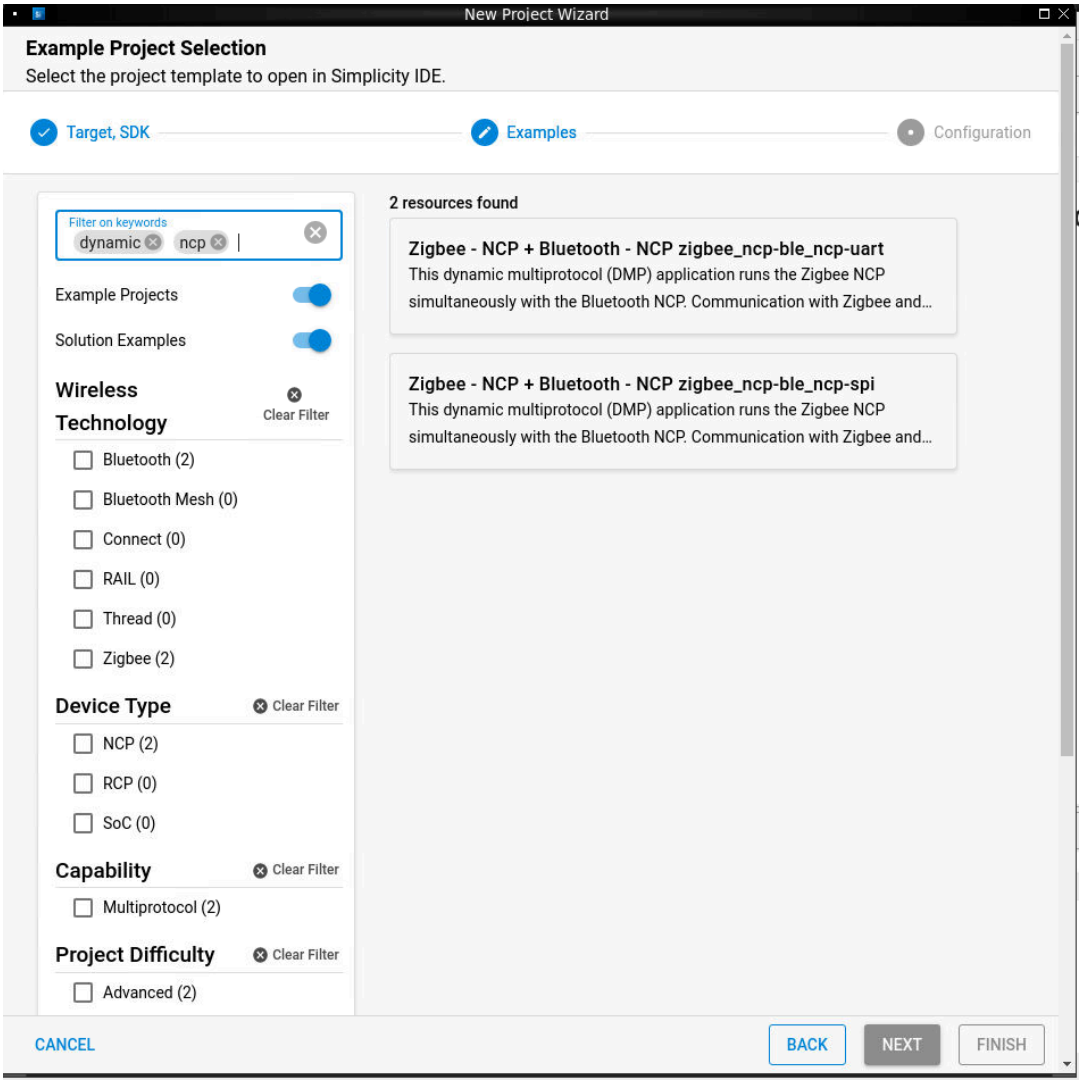
5. On the toolbar, click **Install**. In the resulting Installation Manager dialog, click **Manage installed packages**.
6. Go to the Assets tab, and turn off **Filter by connected product**. Expand the target release and browse for the file, for example protocol\openthread\demos\rcp-uart-802154.
7. Select the .s37 file compatible with your board, and click **Install**.
8. The file is installed under the Simplicity Studio installation location, for example:

```
C:\SiliconLabs\SimplicityStudio\v5_RC2\offline\com.silabs.sdk.stack.super_4.0.0\protocol\openthread\demos\rcp-uart-802154
```

To build a co-processor image for any board in Simplicity Studio 5, select File > New > Silicon Labs New Project Wizard, and click **NEXT** to move to the Example Project Selection dialog.

1. On the Example Project Selection, type `concurrent` as the keyword filter to find RCP projects and `dynamic` and `NCP` as the keywords to filter the DMP NCP projects.
2. From the list of projects, select the appropriate image depending on the desired combination of protocols, and depending on whether your physical link is UART or SPI. The following figure shows an example project that uses **Multiprotocol (OpenThread+Zigbee) – RCP (UART).**





3. Click **NEXT** and build the project and flash it to your board using Simplicity Commander. For more information, see *UG162: Simplicity Commander Reference Guide*.

Note: Multiprotocol, Multi-PAN and CPC support for the RCP is currently only available in the GSDK and not in the OpenThread GitHub repo.

2.2 Host Setup

The following sections cover setup of all required host code for a demo on a Raspberry Pi or similar ARM host platform - either with or without using docker. Note that all pre-compiled binaries have been compiled for arm32v7 and arm64v8 architectures.

2.2.1 General Setup

Some versions of operating systems for the Raspberry Pi have enabled a swapfile. This file creates a lot of unnecessary wear on the SD card and causes performance to degrade drastically over time. It is recommended to disable this with the following commands:

```
sudo dphys-swapfile swapoff
sudo dphys-swapfile uninstall
sudo apt remove dphys-swapfile
```

Additionally, log files should be written to a tmpfs from the container. While this has the drawback that the logs are lost in the event of a total system crash, during development it preserves the life of the SD card. To do this, create a tmpfs at */tmp* with the following entry in */etc/fstab*:

```
tmpfs /tmp tmpfs nosuid,nodev,noatime 0 0
```

If you will be running OTBR, you must load the ip6table_filter module. You can use this command on the Raspberry Pi:

```
sudo modprobe ip6table_filter
```

A permanent solution is to add ip6table_filter into */etc/modules* so that it is loaded into the kernel at bootup.

2.2.2 Raspberry Pi / ARM Host with Docker Setup

The quickest method to set up the host is to use the pre-configured Multiprotocol Docker container on the Raspberry Pi. This container includes everything necessary to easily run the Z3Gateway Zigbee application, the OpenThread Border Router (OTBR) and the ot-cli application, and the BlueZ Bluetooth stack and bluetoothctl, an open source interactive CLI utility for sending commands to the BlueZ stack

The Multiprotocol Docker container is hosted on DockerHub (hub.docker.com) in the *siliconlabsinc/multiprotocol* repository. Pulling the `latest` tag is generally the best way to get the latest release by issuing this command:

```
docker pull siliconlabsinc/multiprotocol:latest
```

If necessary, you can replace the `latest` tag with a specific version.

To run the Docker container, a helpful *run.sh* script is provided in the GSDK in the *app/host/multiprotocol/zigbeed/multiprotocol-container* directory. The script is meant to simplify the docker setup process. Copy it to your Raspberry Pi's home directory. Make sure the RCP device is flashed and attached to the Raspberry Pi before starting the Docker container.

Before running the docker container, a CPCd security commissioning step is required. CPC is now configured by default to encrypt the data over the SPI or UART serial connection between the host and the EFR32. For detailed information, see https://github.com/SiliconLabs/cpc-daemon/blob/main/readme.md.

The run.sh script provides a convenient shortcut to perform the security commissioning. After flashing the EFR32 with the desired image, simply execute run.sh -K on the host. This will pull and run the docker container, use cpcd inside the container to commission security and generate a binding key file, then copy the binding key file outside of the container and place it at /etc/binding-key.key on the host filesystem. This file will then be used for all subsequent runs of the container to establish a secure connection with the EFR32.

After security is commissioned, to start the Docker container, execute the *run.sh* script without any arguments. The container will start `systemd`, which is a process management utility. This will make sure that all the components are started in the proper sequence and are kept running. At startup, only CPCd is started.

The */tmp/multiprotocol-container/log/* directory on the host will be mounted to */var/log/* inside the Docker container. It contains the file *syslog,* which will contain log output from `cpcd`, `zigbeed`, `zigbeed-socat`, `otbr-agent`, and `ot-cli`, among other programs. You can monitor the file live by issuing the following command:

```
tail -f /tmp/multiprotocol-container/log/syslog
```

You can also use the standard `journalctl` utility to monitor the logs for a given systemd service. For example, to monitor the cpcd log from outside of the container, use the following command:

```
docker exec -it multiprotocol journalctl -fexu cpcd
```

As a convenience, you can also execute this command by typing `run.sh -l`. Note that by default, verbose cpcd logging is disabled. To enable it, see section 3 Configuration Files. After starting the container, it is a good idea to verify that CPCd successfully connected to the EFR32 by running run.sh -l and looking for the log line "Daemon startup was successful".

A special file */accept_silabs_msla* will be created in the Docker container to indicate acceptance of the Silicon Labs Master Software License Agreement (MSLA) located at https://www.silabs.com/about-us/legal/master-software-license-agreement.

Once the Docker container is running, you can open a shell by issuing the `run.sh -o` command.

The Zigbeed configuration file is */usr/local/etc/zigbeed.conf* and the CPCd configuration file is */usr/local/etc/cpcd.conf*. For more information, see section 3 Configuration Files.

The cpcd, zigbeed, Z3Gateway, Z3GatewayCpc, XncpLedHost, otbr-agent, ot-ctl, ot-cli, and cpc-hci-bridge binaries are all installed in */usr/local/bin*. There are systemd configuration files to start up CPCd, zigbeed and socat for Zigbee, OTBR for OpenThread, and cpc-hci-bridge and hciattach for Bluetooth. The definition files for those are in */etc/systemd/system*. You can use systemd commands to start the services from within the Docker container shell as follows:

```
systemctl start zigbeed
systemctl start otbr
systemctl start hciattach
```

You can do this from outside of the container, as follows:

```
docker exec -it multiprotocol systemctl start zigbeed
docker exec -it multiprotocol systemctl start  otbr
docker exec -it multiprotocol systemctl start  hciattach
```

For Bluetooth, in order to avoid interference with Bluetooth running in the container, first disable Bluetooth on the native Raspi host using the following commands:

```
sudo systemctl stop bluetooth
sudo systemctl mask bluetooth.service
```

After starting the container and opening a shell, issue the command `service hciattach start`. This uses the systemd utility to start the necessary processes for the BlueZ stack to connect to the RCP via CPCd. To interact with the Bluetooth network, issue the command `bluetoothctl`.

Once you are at the bluetoothctl prompt, you can use the bluetoothctl commands such as `list`, `advertise`, `connect`, and `scan` to exercise the host Bluetooth stack and RCP. You can find further documentation for bluetoothctl and BlueZ online.

Note that the zigbeed service automatically starts the zigbeed-socat service, and the hciattach service automatically starts the cpc-hci-bridge service.

For convenience, the run.sh script has arguments to start each of these services after the container is running: *-Z* starts zigbeed and launches a terminal with Z3Gateway; *-T* starts OTBR and opens a terminal with ot-ctl; *-L* starts Bluetooth and opens a terminal with bluetoothctl, and *-C* launches a terminal with Z3GatewayCpc (for use with the Zigbee NCP/OpenThread RCP).

You can see the processes by running `ps aux` inside the Docker container. Alternatively, you can use systemd commands to see the state of the various services as follows:

```
systemctl status zigbeed
```

You may need to edit the cpcd and zigbeed configuration files. You can do so within the container using the *nano* text editor. Or you can mount a configuration file from the host filesystem for either CPCd or Zigbeed, by appending `-v <hostConfFile>:<containerConfFile>` to the `docker run` command in the *run.sh* script. See also section 3 Configuration Files.

After modifying a configuration file, you should restart services by issuing these commands:

```
systemctl restart cpcd
systemctl restart zigbeed
```

The system is now ready to run host applications. For more information, see section 4 Running Host Applications.

To stop the docker container, issue `run.sh -s`.

2.2.3 Building Host Applications

Using Docker or Systemd is not required. They are both meant as a convenience for rapid prototyping. To run the system without Docker, you will need to build each component for your target platform.

2.2.3.1 Building CPCd

CPCd and the libcpc.so library must be built from source on your target platform by following the instructions at https://github.com/SiliconLabs/cpc-daemon/blob/main/readme.md. Use the git tag corresponding to the GSDK version of the RCP, for example 4.1.0.0. After running `make install`, make sure to run `ldconfig` to update the library database.

Note: You may need to install the following CPCd dependency:

```
sudo apt-get install libmbedtls-dev
```

By default, `make install` places libcpc.so in */usr/local/lib/arm-linux-gnueabihf* and sl_cpc.h in */usr/local/include*.

2.2.3.2 Building OpenThread Host Applications

OpenThread host applications can be built from Silicon Lab's GSDK as well as GitHub repositories. Silicon Labs provides a vendor extension to build an OpenThread Linux host application with multi-PAN and CPC support and needs the following configurations.

The Vendor extension needs the following configurations:

- Specify OT_POSIX_CONFIG_RCP_VENDOR_INTERFACE to the vendor interface source file. Silicon Labs provides 'cpc_interface.cpp' source file for CPC support located at '/protocol/openthread/platform-abstraction/posix/' under the GSDK directory.
- Specify CMAKE_MODULE_PATH to the CMake module directory to find the vendor-specific dependencies. This needs to be set to '/protocol/openthread/platform-abstraction/posix under theGSDK directory.
- Set OT_POSIX_CONFIG_RCP_VENDOR_DEPS_PACKAGE dependency name to "SilabsRcpDeps".
- Specify CPCD_SOURCE_DIR to the cpc daemon project path. This is needed to link/resolve CPC symbols. This can be '/platform/service/cpc/daemon' under the GSDK directory or a path to a cloned GitHub repository as per section 2.2.3.1 Building CPCd.

Other OpenThread configurations are as follows:

- Set OT_MULTIPAN_RCP to 'ON'
- Set OT_POSIX_CONFIG_RCP_BUS to 'VENDOR'
- Set OT_CONFIG to the Silicon Labs posix config header path to build apps with Silicon Labs configuration settings. It is located at '/protocol/openthread/platform-abstraction/posix/openthread-core-silabs-posix-config.h' under the GSDK directory. This file must be copied under openthread repo as motioned in the instructions below.

Note that host applications can be built using 'CMake' only; 'make' is unsupported.

To build using the Silicon Lab's GSDK, first, make sure you are using the GSDK OpenThread and OTBR artifacts:

- GSDK OpenThread repo is in **util/third_party/openthread** – make sure this folder is symlinked under **util/third_party/ot-br-posix/third_party/openthread/repo**
- GSDK OTBR repo is in **util/third_party/ot-br-posix**

To use Silicon Labs specific configuration settings for border-router and ot-cli, you can grab the special configuration header hosted in the gsdk under /protocol/openthread/platform-abstraction/posix/openthread-core-silabs-posix-config.h.

```
# Copy the config file to a known include path

sudo cp /protocol/openthread/platform-abstraction/posix/openthread-core-silabs-posixconfig.h
/util/third_party/openthread/src/posix/platform/
```

The command to build the otbr-agent from the util/third_party/ot-br-posix directory of the GSDK using CMake as follows. Make sure to provide the absolute path to $GSDK_DIR variable, for example:

```
export GSDK_DIR=<An absolute to GSDK directory>

OTBR_OPTIONS="-DOT_MULTIPAN_RCP=ON -DOT_POSIX_CONFIG_RCP_BUS=VENDOR -DOT_CONFIG=openthread-core-
silabs-posix-config.h -DCPCD_SOURCE_DIR=$GSDK_DIR/platform/service/cpc/daemon  -
DOT_POSIX_CONFIG_RCP_VENDOR_INTERFACE=$GSDK_DIR/protocol/openthread/platform-
abstraction/posix/cpc_interface.cpp -DCMAKE_MODULE_PATH=$GSDK_DIR/protocol/openthread/platform-
abstraction/posix -DOT_POSIX_CONFIG_RCP_VENDOR_DEPS_PACKAGE=SilabsRcpDeps" ./script/setup
```

Make sure to specify the path to CPCD_SOURCE_DIR; otherwise, you may encounter undefined references to cpc symbols as follows.

```
protocol/openthread/platform-abstraction/posix/cpc_interface.cpp:93: undefined reference to
`cpc_init'

/usr/bin/ld: protocol/openthread/platform-abstraction/posix/cpc_interface.cpp:99: undefined refer-
ence to `cpc_open_endpoint'

/usr/bin/ld: src/posix/platform/librcp-vendor-intf.a(cpc_interface.cpp.o): in function
`ot::Posix::CpcInterfaceImpl::Deinit()':
```

```
protocol/openthread/platform-abstraction/posix/cpc_interface.cpp:130: undefined reference to
`cpc_close_endpoint'

/usr/bin/ld: src/posix/platform/librcp-vendor-intf.a(cpc_interface.cpp.o): in function
`ot::Posix::CpcInterfaceImpl::Read(unsigned long long)':

protocol/openthread/platform-abstraction/posix/cpc_interface.cpp:155: undefined reference to
`cpc_set_endpoint_option'
```

To fix the error, specify the location of the cpc-daemon sources (see section 2.2.3.1 Building CPCd) by adding `-DCPCD_SOURCE_DIR=<path to cpc-daemon>` to `OTBR_OPTIONS` OR '/platform/service/cpc/daemon/' under GSDK directory.

You can also build the posix ot-cli app using CMake. The command to build the posix ot-cli from the 'util/third_party/openthread' directory of the GSDK using CMake:

```
sudo ./script/cmake-build posix -DOT_MULTIPAN_RCP=ON -DOT_POSIX_CONFIG_RCP_BUS=VENDOR -
DCPCD_SOURCE_DIR=$GSDK_DIR/platform/service/cpc/daemon -
DCMAKE_MODULE_PATH=$GSDK_DIR/protocol/openthread/platform-abstraction/posix -
DOT_POSIX_CONFIG_RCP_VENDOR_INTERFACE=$GSDK_DIR/protocol/openthread/platform-
abstraction/posix/cpc_interface.cpp -DOT_CONFIG=openthread-core-silabs-posix-config.h -
DOT_POSIX_CONFIG_RCP_VENDOR_DEPS_PACKAGE=SilabsRcpDeps
```

You can also build 'otbr-agent' and 'ot-cli' host applications using GitHub repositories, but you would still need GSDK artifacts to provide OT_CONFIG, CPCD_SOURCE_DIR, CMAKE_MODULE_PATH and DOT_POSIX_CONFIG_RCP_VENDOR_INTERFACE configurations. Hence, the easiest way to build these host apps is to symlink GitHub openthread and ot-br-posix repositories into the GSDK 'util/third_party/openthread' and 'util/third_party/ot-br-posix' directories, respectively. With this setup, you can use the same build commands mentioned above.

2.2.3.3 Building Zigbee Host Applications

Zigbee host applications such as Z3Gateway for use with Zigbeed should be built just as they would be for running against a Zigbee NCP using EZSP/ASH over UART. No changes to the build process are required.

Zigbee host applications for use with the Zigbee NCP/OpenThread RCP image should be built using the **zigbee_ezsp_cpc** component in place of the **zigbee_ezsp_uart** or **zigbee_ezsp_spi** component. This component allows the host app to connect directly to CPCd. The Z3GatewayCpc sample app provided in the GSDK already includes the proper components and can be generated and built as is for this purpose.

2.2.3.4 Building Bluetooth Linux Host with CPC (bt_host_empty)

The `bt_host_empty` host sample application may be used with the Zigbee + BLE DMP NCP. It is precompiled inside the container in `/usr/local/bin` for convenience.

The makefile for `bt_host_empty` is located in the following GSDK directory: `app/bluetooth/example_host/bt_host_empty`. To build, CPCd must first be installed on the system. The application can be compiled using the following command:

```
make CPC=1
```

The application can then be run with the -C option to specify CPC instance name (ex: -C cpcd_0). Note that it is mandatory to use -R option to prevent the host from rebooting the NCP on startup as it is not desirable when working with the DMP NCP. (The -R option should not be used when running a single-protocol Bluetooth NCP.)

```
./bt_host_empty -C cpcd_0 -R
```

2.2.3.5 Building Zigbeed

Zigbeed may be built from application sources using the zigbeed.slcp project file and Zigbee stack libraries for arm32v7 or arm64v8 that are supplied with the GSDK. Requirements:

1. Gecko SDK, the suite of Silicon Labs SDKs, including Zigbee, OpenThread, and Bluetooth
2. cpcd must be made and installed from the sources at https://github.com/SiliconLabs/cpc-daemon

3. Silicon Labs Configurator CLI tool (SLC-CLI), which allows you to generate projects based on your customizations of Silicon Labs software components. See *UG520: Software Project Generation and Configuration with SLC-CLI* for more information.

The zigbeed.slcp project file is found in *<GSDK Installation>/protocol/zigbee/app/zigbeed/zigbeed.slcp*. It includes all the components necessary to build Zigbeed. It also includes the **zigbee_xncp** component by default, to support custom messaging between Zigbeed and the Zigbee host application. By default, the project uses the **linux_arch_64** component to build for arm64v8. Select the **linux_arch_32** component to build for arm32v7. The Makefile can then be generated using SLC-CLI as follows:

```
slc generate -s=../.. –with=linux_arch_32 -p=app/zigbeed/zigbeed.slcp -d=app/zigbeed/output
```

Note: If you generated the app on your PC but want to build it on a Raspberry Pi, add the -cp option to copy the necessary files, including libraries, to the generation directory.

```
Slc generate -cp -s=../.. –with=linux_arch_32 -p=app/zigbeed/zigbeed.slcp -d=app/zigbeed/output
```

Copy the generation directory to your target ARM-based system. From within the generation directory, invoke make as follows:

```
make -f zigbeed.Makefile
```

2.2.3.6 Custom EZSP Messaging in Zigbeed

The **XNCP** component (zigbee_xncp) allows custom EZSP messages to be added to Zigbeed in the same way that they can be added to the Zigbee NCP. See *AN1320: Building a Customized NCP Application with Zigbee EmberZNet 7.x* for more information on XNCP.

To implement custom messages between Zigbeed and the host app, the developer defines and implements the format, parsing, and serialization of the message set. The serialized messages are conveyed between Zigbeed and host as opaque byte strings. This “extensible network co-processor” functionality is provided by the **XNCP** component. To send a custom message to the host, construct and serialize the message, then send the resulting byte string to the host using the EmberZNet PRO API function `emberAfPluginXncpSendCustomEzspMessage()`. After enabling the XNCP Library component, the following callbacks are provided for custom 2-way messaging over EZSP:

`emberAfPluginXncpIncomingCustomFrameCallback` – Processing of custom incoming serial frames from the EZSP host.

`emberAfIncomingMessageCallback` – Custom processing of received Zigbee application layer messages before passing these (through Incoming Message Callback frames) to the EZSP host. Note that custom outgoing serial frames from Zigbeed to the EZSP host should be provided as response frames to the host in reply to a Callbacks EZSP command or some custom host-to-Zigbeed EZSP command, where they can be handled by the following host-side (such as in **XncpLedHost** host app) callback: `void ezspCustomFrameHandler(int8u payloadLength, int8u* payload)`.

The **zigbeed.slcp** project includes the source file *protocol/zigbee/app/zigbeed/zigbeed_custom_ezsp_commands.c* that contains an example implementation for custom messaging on the zigbeed xncp side. The example implements a small set of custom messages that can be sent from **XncpLedHost** app, which is supplied prebuilt in the docker container for convenience. This functionality can be tested as follows:

```
$ XncpLedHost -p ttyZigbeeNCP
XncpLedHost>custom set-led 1
XncpLedHost>custom get-led
custom get-led
Send custom frame: 0x00
 Response (state): 1 (OFF)
```

This example from **XncpLedHost.slcp** and *zigbeed_custom_ezsp_commands.c* can be followed to implement other custom EZSP messages between a host app and Zigbeed.

2.2.4 Raspberry Pi / ARM Host without Docker Setup

After building the host components for your target platform, additional setup is required before running the host applications.

2.2.4.1 CPCd Setup

Edit the *cpcd.conf* configuration file located at */usr/local/etc/cpcd.conf* to match the system setup. For more information, see section 3 Configuration Files.

CPC security is now enabled by default in the *cpcd.conf* file and in the SLCP project files. This means data sent over the serial line between the host and the EFR32 is encrypted. A security commissioning step is required to bind the host to the EFR32. See https://github.com/SiliconLabs/cpc-daemon/blob/main/readme.md for instructions.

Start CPCd with: `/usr/local/bin/cpcd`.

2.2.4.2 Zigbeed Setup

If you are using the mult-PAN or multiprotocol RCP, you will need to setup and run Zigbeed on the host. If you are using the Zigbee NCP/OpenThread RCP, skip this step.

To connect to Zigbeed, the host app (for example, Z3Gateway) depends on a standard application called `socat` that can be installed as follows:

```
sudo apt-get install socat
```

Use `socat` to create two connected PTY devices, one for Zigbeed and one for the host application:

```
socat -x -v pty,link=/dev/ttyZigbeeNCP pty,link=/tmp/ttyZigbeeNCP
```

Start Zigbeed with: `/usr/local/bin/zigbeed`

The system is now ready to run Zigbee host applications such as Z3Gateway. For more information, see section 4 Running Host Applications.

2.2.4.3 OpenThread Border Router Setup

OTBR setup is complex and beyond the scope of this document. For information on building otbr-agent with CPC and multi-PAN support, see section 2.2.3 Building Host Applications.

2.2.4.4 Bluetooth Host Setup for Use with RCP

To run BlueZ outside of the Docker container for use with the Zigbee+OpenThread+BLE RCP, install the following dependencies:

```
sudo apt-get install bluetooth bluez bluez-tools rfkill libbluetooth-dev
```

Start the Bluetooth service with the command `service bluetooth start`. In certain circumstances it can be useful to start Bluetooth with the `--experimental` flag, by editing the systemd file in */usr/lib/systemd/system/bluetooth.service*. If you have previously been using Bluetooth inside the Docker container, you must unmask the Bluetooth service.

Next, run CPCd as usual, and verify that it connects to the EFR successfully. This can be done by looking at the cpcd logging output for the message “Daemon connected successfully”.

Next, a simple cpc-hci-bridge program connects to CPCd, and exposes a virtual serial device on the Linux host in /dev/pts. The source code and makefile are located in the GSDK at gsdk/*app/bluetooth/example/example_host/bt_host_cpc_hci_bridge*. Building it requires cpcd to be installed, and the following directories to be located in the same gsdk directory structure: gsdk/*platform/common/inc* and gsdk/*protocol/bluetooth/bgstack/ll/utils/hci_packet/inc*. The virtual serial device will be used by the Bluetooth stack to communicate with the RCP via the HCI protocol.

Running cpc-hci-bridge connects to CPCd using the standard instance name cpcd_0, opens a CPC endpoint to the BLE RCP running on the EFR, and creates a numbered virtual serial device on the host, for example /dev/pts/2. The actual number may vary. For convenience, cpc-hci-bridge also creates a symlink to the device from pts_hci in the working directory.

Next, use the following command to attach the Bluetooth stack to the newly created virtual serial device, where <device> is the name of the virtual serial device:

```
sudo hciattach <device> any
```

Finally, run `sudo bluetoothctl` to start up the Bluetooth CLI utility. A utility that comes with the standard Bluetooth tools, called `btmon`, can be used to view the traffic going through the Bluetooth device.

3 Configuration Files

The configuration files *cpcd.conf* and *zigbeed.conf* are located in */usr/local/etc* within the docker container. The -c argument of the run.sh script provides a convenient way to mount a cpcd.conf file from your host system. This allows for persistent edits across container restarts.

You may need to modify the UART or SPI configurations in *cpcd.conf*. First select the BUS_TYPE. Make sure that the *UART_DEVICE_FILE* or *SPI_DEVICE_FILE* variable is correctly set for your system and points to the proper device. Logging is disabled by default to reduce flash wear. Set STDOUT_TRACE to true to send logs to syslog within the container. Other configurations are documented in the file.

For *zigbeed.conf*, the `radio-url` and `ezsp-interface` options are required. You should not have to change either of them from their default values. There is also an option for adjusting the Spinel driver debug level.

For the OpenThread Border Router otbr-agent, the radio-url is found in the */etc/systemd/system/otbr.service* file. In future releases this will be moved to the /etc/default/otbr-agent configuration file.

4 Running Host Applications

4.1 Zigbee Host Applications

For convenience, a precompiled Z3Gateway host sample application is included in the Docker container. To run it with Zigbeed, issue the following commands from inside the Docker container shell (or on the host, depending on your installation):

```
Systemctl start zigbeed
/usr/local/bin/Z3Gateway -p ttyZigbeeNCP
```

The `ttyZigbeeNCP` refers to the file */dev/ttyZigbeeNCP*. The Z3Gateway application automatically prepends `/dev` to a relative path in the `-p` argument. The `run.sh -Z` is a script that will run these commands on an already running container.

Any Zigbee host application built with EZSP/ASH for communicating with a Zigbee NCP over a UART can also be used with Zigbeed by passing it the PTY device name. This is because to the host application, the PTY device appears exactly like a normal serial device.

If your Zigbee host application was built for EZSP/SPI, you will have to rebuild it for EZSP/ASH to work with Zigbeed.

Some EZSP commands are not supported with zigbeed:

- Set/Get Manufacturing Tokens
  - EZSP_SET_MFG_TOKEN
  - EZSP_GET_MFG_TOKEN
- Secure EZSP Commands
  - EZSP_SET_SECURITY_KEY
  - EZSP_SET_SECURITY_PARAMETERS
  - EZSP_RESET_TO_FACTORY_DEFAULTS
  - EZSP_GET_SECURITY_KEY_STATUS
- Bootloader Commands
  - EZSP_LAUNCH_STANDALONE_BOOTLOADER
  - EZSP_SEND_BOOTLOADER_MESSAGE
  - EZSP_GET_STANDALONE_BOOTLOADER_VERSION_PLAT_MICRO_PHY
  - EZSP_INCOMING_BOOTLOAD_MESSAGE_HANDLER
  - EZSP_BOOTLOAD_TRANSMIT_COMPLETE_HANDLER
  - EZSP_AES_ENCRYPT

If you are using the Zigbee NCP/OpenThread RCP, then do not run Zigbeed on the host. Instead use a host app that has been built with the **zigbee_ezsp_cpc** component. For convenience, a precompiled Z3GatewayCpc host sample application has been included in the Docker container. No arguments are required to run it, as it connects directly to CPCd. An optional `-c` argument can be supplied to specify the CPC daemon instance name to connect to. The default instance connected to is cpcd_0.

4.2 OpenThread Host Applications

A precompiled OpenThread ot-cli sample application with multi-PAN and CPCd support is also included in the Docker container and GSDK. **ot-cli** is a stand-alone sample host application that includes the OpenThread stack and exposes the standard OpenThread CLI. You can run by issuing this command:

```
/usr/local/bin/ot-cli 'spinel+cpc://cpcd_0?iid=2'
```

The RCP uses the interface id parameter of the `radio-url` argument (iid=2) to distinguish between the OpenThread and Zigbeed applications. By default, the */usr/local/etc/zigbeed.conf* file uses a `radio-url` argument with iid=1. The string 'cpcd_0' in the radio-url is the default CPCd instance name, which is defined in the /usr/local*/etc/cpcd.conf* file. To enable ot-cli debugging output, use the command line argument `-d <level>` to enable logging at the desired level 1-5. By default log messages are printed to syslog. Add the argument `-v` as well to echo log messages to `stdout` as well.

The `run.sh -O` command can be used as a shortcut to start this application on a running container and open a shell to it.

4.3 OpenThread Border Router (OTBR) Application

The Multiprotocol Docker container is based on the OpenThread Border Router Docker container. It has all the necessary dependencies to run OTBR over CPC. To run it, issue the following commands from inside the Multiprotocol Docker container shell (or on the host, depending on your installation):

```
systemctl start otbr
ot-ctl
```

For convenience, this can be run with `run.sh -T` on a running container and open a shell to it. Note that OTBR uses `ot-ctl` instead of `ot-cli`. These two utilities offer similar CLI to the OpenThread network. But `ot-ctl` is a utility that connects to otbr-agent, which is the process that runs the OpenThread stack for OTBR. To change the radio-url argument for otbr-agent, see section 3 Configuration Files.

4.4 Bluetooth Host Applications

The multiprotocol solution uses the standard Linux BlueZ Bluetooth stack on the host. BlueZ and associated utilities are documented extensively online. Once Bluetooth is running as described in section 2.2.4.4 Bluetooth Setup, you can use standard Bluetooth tools such as `bluetoothctl` for a CLI utility and `btmon` to view the traffic going through the Bluetooth device.

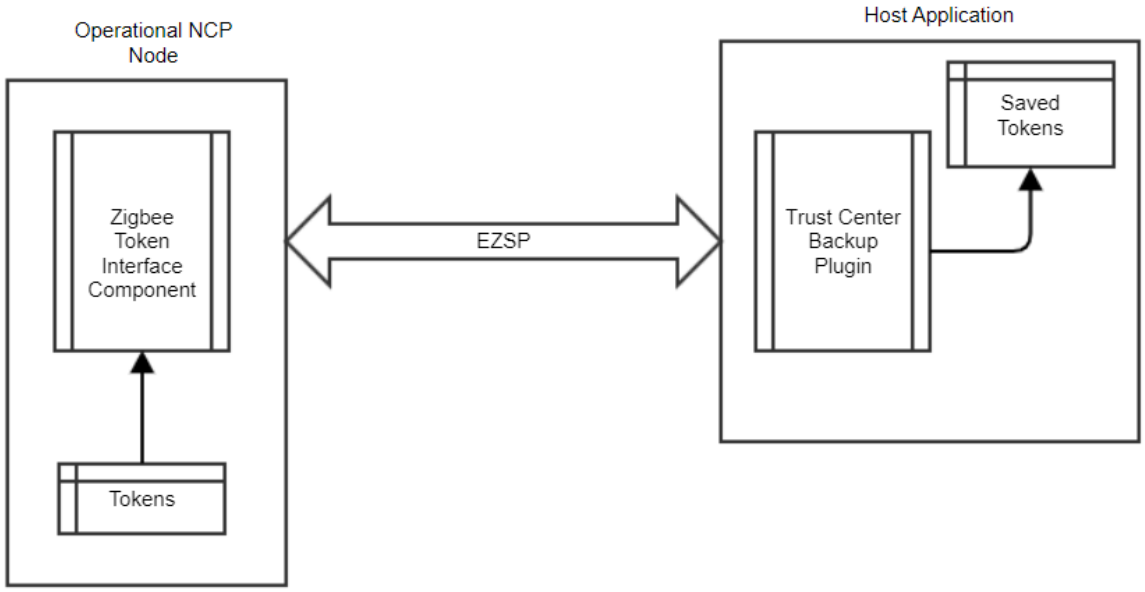
5 Zigbee Host+NCP to Host+Zigbeed+RCP Migration Note

This section describes one method of migrating a Zigbee Host+NCP system to a Host+Zigbeed+RCP system by backing up and restoring the token data. This method is very similar to backing up and restoring Z3Gateway explained in *AN1387: Backing Up and Restoring a Z3 Green Power Combo Gateway,* with certain differences as noted here. One of the major differences is that in this case the same NCP hardware is used as RCP hardware.

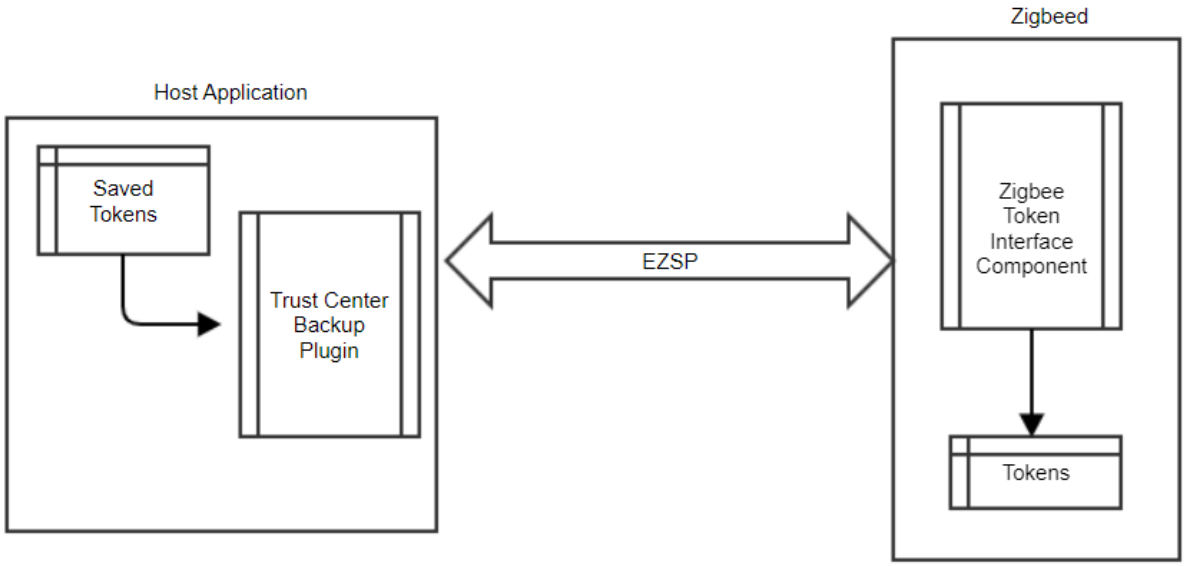
The non-volatile Zigbee network stack context on an NCP is stored using the on-chip token system. By moving that stack context from the NCP to Zigbeed on the host, we can migrate a Zigbee Host+NCP application to Host+Zigbeed+RCP application.

The migration procedure requires the NCP using the NVM3 token system and the **zigbee_token_interface** component. If it is not already using these, then the NCP must first be updated to do so. Similarly, the Zigbee host app needs to be updated to add the **zigbee_trust_center_backup** component with EMBER_AF_PLUGIN_TRUST_CENTER_BACKUP_POSIX_FILE_BACKUP_SUPPORT configuration set to 1.

With the above upgrades to NCP and host, the host can read the stack tokens from NCP and save them to a file.



The host then reads the saved tokens and updates the tokens on Zigbeed. Zigbeed's default configuration includes the **zigbee_token_interface** component, which allows the host to write the saved network stack tokens to it.



For simplicity, the **trust_center_backup** component provides command line interfaces to read and save ncp tokens and write to Zigbeed tokens. They are, respectively:

```
plugin trust-center-backup backup-tokens <file name to save the tokens>

plugin trust-center-backup update-zigbeed <the file that has saved the tokens above>
```

There are certain limitations:

- This method only works for migrating from NCPs with NVM3 tokens.
- It migrates the stack tokens from NCP to Zigbeed. It presently does not migrate the custom tokens.
- This method assumes the same NCP hardware is used as RCP, and therefore retains the IEEE address (EUI64).
- Since the network key frame counter (`NWK key FC`) value is stored truncated and only incremented after initialization, you may need to perform an extra reset cycle of the Z3Gateway/Zigbeed to obtain a working value for the FC. Otherwise, the starting value after migrating may be lower than the last sent value and therefore the gateway's packets might be ignored until the FC value exceeds the value before migration. Please note that, on initialization, the stack code will read the truncated FC and increment it by 0x1000. This incremented value will only be used at the next initialization (after a reset of the Z3Gateway).

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

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com