AN1333: Running Zigbee, OpenThread, and Bluetooth Concurrently on a Linux Host with a Multiprotocol RCP

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

The Bluetooth features described in this document are released as alpha quality. Silicon Labs will continue to test and welcomes your comments.

KEY POINTS

- Describes the components of the Multiprotocol and Multi-PAN RCP solution.
- Provides details for building and running the components of the Silicon Labs Zigbee and OpenThread stacks running on a Linux host.

1 System Architecture

The system architecture includes various software components:

- An RCP image that runs on the EFR32 coprocessor.
- A Linux host process called Coprocessor Communication Daemon (CPCd) that communicates with the coprocessor 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.
- A Zigbee host application that communicates with Zigbeed using EmberZNet Serial Protocol / Asynchronous Serial Host (EZSP / ASH) over a virtual serial port.
- An OpenThread host application such as the OpenThread Border Router (otbr-agent) that 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.

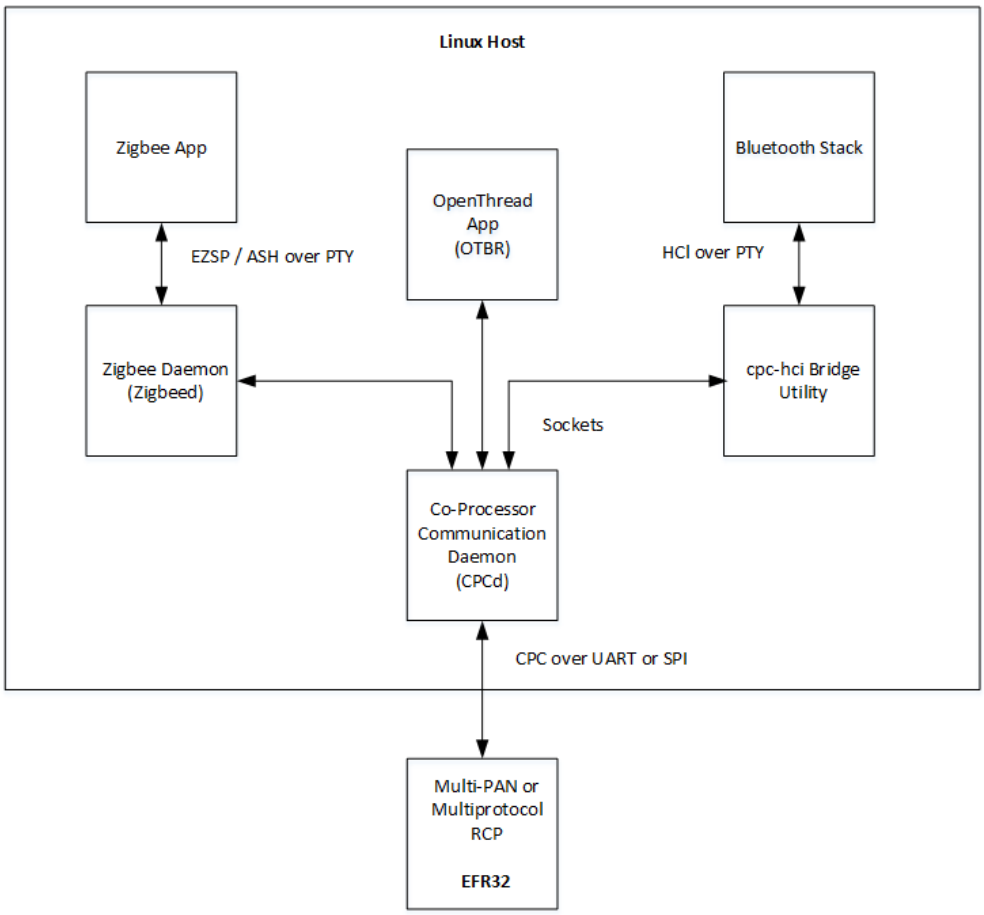


Figure 1.1. System Architecture

The RCP image comes in two flavors: multi-PAN (802.15.4 only), and multiprotocol (with Bluetooth added).

The multi-PAN 802-15.4-only 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 802.15.4 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. The project files are **rcp-uart-802154.slcp** and **rcp-spi-802154.slcp**.

The multiprotocol RCP adds the Bluetooth Controller and FreeRTOS to the 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. The project files are **rcp-uart-802154-blehci.slcp** and **rcp-spi-802154-blehci.slcp**.

The Coprocessor Communication Daemon (CPCd) enables users to have multiple stack protocols interact with the radio coprocessor over a shared physical link. CPCd is distributed as three components: the daemon binary cpcd; a library that enables C applications to interact with the daemon; and a configuration file. 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 *AN1351: Using the Coprocessor Communication Daemon (CPCd)*.

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 coprocessor to interface with Zigbeed unchanged, simply by supplying the proper device name to it.

Both Zigbee and OpenThread stacks 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.

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

2 System Setup

2.1 RCP Setup

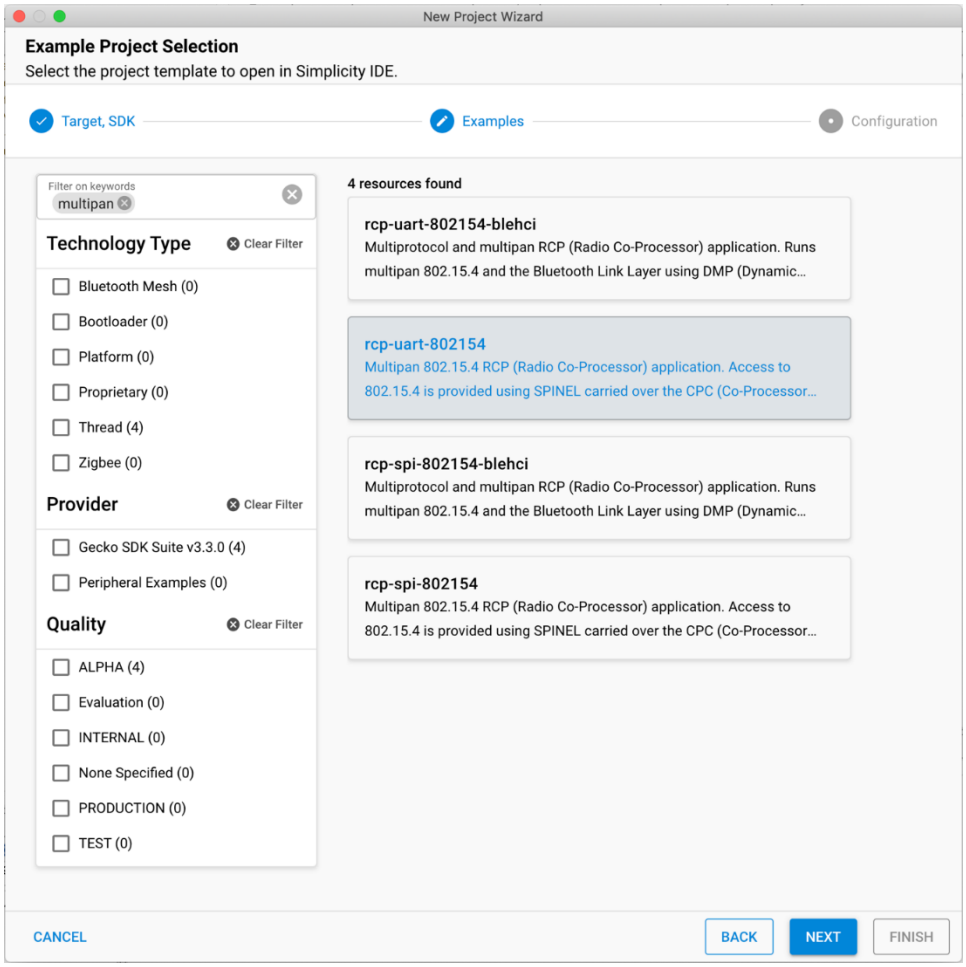
Simplicity Studio has precompiled demo RCP projects for certain boards.

1. On the toolbar, click **Install**. In the resulting Installation Manager dialog, click **Manage installed packages**.
2. 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.
3. Select the .s37 file compatible with your board, and click **Install**.
4. 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 an RCP image for any board, use the Silicon Labs New Project Wizard in Simplicity Studio 5 and complete these steps:

1. On the Example Project Selection page, type "**multipan**" as the keyword filter.
2. From the list of projects, for the multi-PAN (802.15.4 only) RCP select either **rcp-uart-802154** or **rcp-spi-802154**, depending on whether your physical link is UART or SPI. For the multiprotocol RCP with Bluetooth, select either **rcp-uart-802154-blehci** or **rcp-spi-802154-blehci**. The following figure shows an example project that uses **rcp-uart-802154**.



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 guide the user through 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 unfortunate 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. An example version is included in the Appendix: example run.sh script. Copy it to your Raspberry Pi's home directory. Make sure the RCP device is flashed and attached to the Raspberry Pi prior to starting the Docker container.

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, cpcd logging is disabled. To enable it, see section 3 Configuration Files.

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, 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
```

Or 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, and *-L* starts Bluetooth and opens a terminal with bluetoothctl.

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 Raspberry Pi / ARM Host without Docker Setup

Using Docker or Systemd is not required. They are both meant as a convenience for rapid prototyping. To run the system manually, complete the following steps.

2.2.3.1 CPCd and General Setup

- Copy the *app/host/multiprotocol/zigbeed/multiprotocol-container/_artifacts* folder from the GSDK to the host. This folder contains etc, bin_arm32v7, bin_arm64v8, and systemd folders. Binaries in bin_arm32v7 or bin_arm64v8 go in */usr/local/bin*, depending on your architecture. Alternatively, you can place them in */opt* or some other directory, but then you will need to update the PATH environment variable so they can be called without an absolute path.
- Configuration files in etc go in */usr/local/etc*. For more information, see section 3 Configuration Files.
- (optional) Systemd files can be placed in */etc/systemd/system*. This will instruct Systemd to start and manage all the processes in the proper order.

The libcpc.so library must be built from source by following the instructions at https://github.com/SiliconLabs/cpc-daemon. Use the git tag corresponding to the GSDK version of the RCP, e.g., 4.0. After building it, copy libcpc.so to /lib. Alternatively, you can set the LD_LIBRARY_PATH environment variable. Once done, run `ldconfig` to update the library database.

Install the following cpcd dependency:

```
sudo apt-get install mbedtls-dev
```

Edit the *cpcd.conf* configuration file to match the system setup. For more information, see section 3 Configuration Files.

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

2.2.3.2 Zigbee Setup

Create a file */accept_silabs_msla*. This is required to start Zigbeed. By creating this file, you affirm that you have read and agree to the Silicon Labs MSLA.

Zigbeed / Z3Gateway depends on a few standard applications that can be installed as follows:

```
sudo apt-get install socat libreadline7
```

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.

2.2.3.3 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.

2.2.3.4 Bluetooth Setup

To run Bluetooth outside of the Docker container, 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 RCP successfully. This can be done by looking at the cpcd logging output.

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 is located in the GSDK at *app/bluetooth/example/example_host/bridge*. A precompiled cpc-hci-bridge binary is included in the *bin_arm32v7* and *bin_arm64v8* folders as described above. 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, and creates a numbered virtual serial device, 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.

2.2.4 Building Host Applications

Precompiled binaries are available in the *app/host/multiprotocol/zigbeed/multiprotocol-container/_artifacts* folder of the GSDK.

Any of the binaries contained within the docker container may be extracted for use outside the container as follows.

```
docker cp $(docker create --rm siliconlabsinc/multiprotocol):/usr/local/bin/zigbeed ~/zigbeed
```

CPCd and libcpc.so may be built from source by following the instructions at https://github.com/SiliconLabs/cpc-daemon.

2.2.4.1 Building OpenThread Host Applications

To build an OpenThread Linux host application with multi-PAN and CPC support, a number of configurations are required:

```
OPENTHREAD_CONFIG_MULTIPAN_RCP_ENABLE=1
OPENTHREAD_CONFIG_RCP_BUS=OT_POSIX_RCP_BUS_CPC
OPENTHREAD_SPINEL_CONFIG_RCP_RESTORATION_MAX_COUNT=10
OPENTHREAD_POSIX_CONFIG_MAX_POWER_TABLE_ENABLE=0
OPENTHREAD_CONFIG_CHANNEL_MANAGER_ENABLE=0
OPENTHREAD_CONFIG_CHANNEL_MONITOR_ENABLE=0
```

To build the otbr-agent from the util/third_party/ot-br-posix directory of the GSDK 4.0 using cmake:

```
OTBR_OPTIONS="-DOT_MULTIPAN_RCP=ON -DOT_POSIX_CONFIG_RCP_BUS=CPC" ./script/setup
```

To build the posix ot-cli app from the util/third_party/openthread directory of the GSDK 4.0 using cmake:

```
./script/cmake-build posix multipan_rcp -DOT_POSIX_CONFIG_RCP_BUS=CPC
```

To build the posix ot-cli app from the util/third_party/openthread directory of the GSDK 4.0 using make:

```
make -f src/posix/Makefile-posix RCP_BUS=cpc MULTIPAN_RCP=1 RCP_RESTORATION_MAX_COUNT=10
MAX_POWER_TABLE=0 CHANNEL_MANAGER=0 CHANNEL_MONITOR=0
```

2.2.4.2 Building Zigbee Host Applications

Zigbee host applications such as Z3Gateway 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.

2.2.4.3 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. libcpc.so built from CPC 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
```

From within the generation directory, invoke make as follows:

```
make -f zigbeed.Makefile -e LIBRARY_PATH=<path to libcpc.so>
```

2.2.4.4 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 coprocessor" 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.

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 as well as the GSDK. To run it, 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.

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 2.2.2.4 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.

Appendix: example run.sh script

```
#!/bin/bash

CONTAINER_NAME=${CONTAINER_NAME:-"siliconlabsinc/multiprotocol"}

DAEMON=1
while [[ $# -gt 0 ]]; do
    case $1 in
        -h|--help)
            echo -e "Usage: $0 [options]"
            echo -e "\t-s\tStop running multiprotocol container"
            echo -e "\t-c\tMount the supplied cpcd.conf file and start the multiprotocol container"
            echo -e "\t-o\tOpen a bash shell in multiprotocol container"
            echo -e "\t-O\tStart ot-cli application in the multiprotocol container"
            echo -e "\t-T\tStart OTBR and ot-ctl application in the multiprotocol container"
            echo -e "\t-Z\tStart Z3GatewayHost application in the multiprotocol container"
            echo -e "\t-L\tStart bluetoothctl application in the multiprotocol container"
            echo -e "\t-b\tStart multiprotocol container as a bash shell only"
            echo "By default, this script will start multiprotocol container and initialize all the
various programs and dependencies"
            exit
            ;;
        -s|--stop)
            docker stop -t 0 multiprotocol
            exit
            ;;
        -c|--cpcd-conf)
            shift
            CPCD_CONFIG_FILE=$(realpath $1)
            shift
            ;;
        -l|--log)
            docker exec -it multiprotocol journalctl -fexu cpcd
            exit
            ;;
        -o|--open)
            docker exec -it multiprotocol /bin/bash
            exit
            ;;
        -O|--ot-cli)
            docker exec -it multiprotocol /usr/local/bin/ot-cli 'spinel+cpc://cpcd_0?iid=2'
            exit
            ;;
        -T|--ot-ctl)
            docker exec -it multiprotocol systemctl start otbr
            docker exec -it multiprotocol ot-ctl
            exit
            ;;
        -Z|--zigbee-host)
            docker exec -it multiprotocol systemctl start zigbeed
            docker exec -it multiprotocol /usr/local/bin/Z3Gateway -p ttyZigbeeNCP
            exit
            ;;
        -L|--bluetoothctl)
            # stop bluetoothd on the host
            sudo service bluetooth stop
            # disable bluetoothd on the host
            sudo systemctl mask bluetooth.service
            docker exec -it multiprotocol systemctl start hciattach
            docker exec -it multiprotocol bluetoothctl
            exit
            ;;
        -b|--bash)
            DAEMON=0
```

```
            shift
            ;;
        *)
            echo "Unrecognized option '$1'"
            exit
            ;;
    esac
done

docker pull ${CONTAINER_NAME}
docker stop -t 0 multiprotocol # stop container if it is running

if [ -e "$CPCD_CONFIG_FILE" ]; then
    echo "Using host's cpcd config file: $CPCD_CONFIG_FILE"
    RUN_ARGS+=" -v $CPCD_CONFIG_FILE:/usr/local/etc/cpcd.conf:ro"
fi
RUN_ARGS="--rm --name multiprotocol " # Clean up after run
RUN_ARGS+=" -v /tmp/multiprotocol-container/log:/var/log " # Add in logging folder
RUN_ARGS+=" -v /accept_silabs_msla" # Accept the MSLA for Zigbeed
RUN_ARGS+=" --privileged --cap-add SYS_ADMIN" # Add in security permissions
RUN_ARGS+=" --cap-add=SYS_ADMIN --cap-add=NET_ADMIN --net=host" # Add bluetooth

if [ "$DAEMON" == "1" ]; then
    RUN_ARGS+=" -d" # Start as a daemon
else
    RUN_ARGS+=" -it --entrypoint /bin/bash" # Drop into a bash shell on start
fi

echo "Running 'docker run ${RUN_ARGS} ${CONTAINER_NAME}'"
docker run ${RUN_ARGS} ${CONTAINER_NAME}
```