AN1010: Building a Customized NCP Application

The ability to build a customized NCP application image was introduced in EmberZNet PRO 5.4.1 and Silicon Labs Thread 1.0.6. This application note provides instructions for configuring various aspects of the application using the Application Builder tool within Simplicity Studio.

KEY POINTS

- Instructions cover starting from an example or from a new file.
- Customizations include target hardware, initialization, main loop processing, event definition and handling, and host/NCP command extensions.

1. Introduction

In previous stack releases, developers were restricted to using the preconfigured NCP application images ncp-spi and ncp-uart, delivered with the Silicon Labs Thread stack, and similar precompiled binaries delivered in pre-5.4.1 versions of the EmberZNet PRO stack. Both stacks now support the ability to build NCP applications in the Simplicity Studio IDE (also known as AppBuilder), with customizations for target hardware, initialization, main loop processing, event definition and handling, and host/NCP command extensions. This application note describes how to configure a customized NCP application using the Silicon Labs Thread or EmberZNet PRO stack for either the EM35x or members of the Wireless Gecko (EFR32) platforms.

If you are not familiar with using Simplicity Studio to configure an example application or start from a blank application, and then build the application image and load it and a bootloader, refer to *QSG113: Getting Started with Silicon Labs Thread* or *QSG106: Getting Started with EmberZNet PRO*.

For EFR32 platforms, a Hardware Configurator is now available that allows you to modify peripheral configurations, including pin settings. Some configuration options can be changed in the relevant plugin's Hardware Configurator Options interface. All options, including pin configurations, can be changed in the Hardware Configurator tool. You can access the tool either by double-clicking the .hwconf file or through the [**Open Hardware Configurator**] control on the HAL tab. Refer to *AN1115: Configuring Peripherals for 32-Bit Devices in Simplicity Studio* for more information than is provided in this document.

2. Silicon Labs Thread

The Silicon Labs Thread stack includes example applications that work over either SPI or UART. These can be used as starting point for building a customized NCP application, or you can start from a blank application. The first two sections describe these two approaches. The third section details the customizations you can make after starting your application in one of the two ways described.

Customizing the pinout used for the serial interface signals is described in **Hardware Configuration** under section 2.3 Customizations.

2.1 Starting from an Example Application

Silicon Labs recommends choosing one of the Silicon Labs Thread example applications as a starting point for building a customized NCP application. Three examples are provided: NCP-SPI, NCP-UART (with software flow control enabled) and NCP-UART-HW (with hardware flow control enabled). As configured, these NCP example applications are used to build the corresponding binary images delivered with the Silicon Labs Thread stack. Therefore, you not only have a configuration starting point, but can also easily evaluate changes in functionality from your customizations against the binary images provided. See section 2.3 Customizations for information on various common customizations.

When you start a new project in Simplicity Studio, you can select the target part for which the application is being built. The Silicon Labs Thread NCP example applications are preconfigured with the EM3xx Device Type, shown in the Device Type box on the General tab. For these examples, there is no material functional difference between EM3xx and EFR32 device types. The device type selections merely choose a set of included default plugins for the build. As these currently are equivalent between the two device types, you can safely leave this setting unchanged. In Thread 2.9 and above you may not see an EM3xx or EFR32 device type. This is not a problem.

2.2 Starting from a Blank Application

If, instead of beginning with one of the example applications, you prefer to begin with a blank application, make the following changes under the indicated AppBuilder project tabs to configure basic NCP functionality. Following these changes, other customizations can be made as described in section 2.3 Customizations.

General Tab

1. Click the Device type box to open it.
2. If using Thread 2.8 or below, select and add either EM3xx or EFR32, according to selected part.
3. Select and add NCP.
4. Select and add either NCP SPI or NCP UART, according to host/NCP link type.
5. Click [**OK**].

HAL Tab

Set **Bootloader** as Standalone.

For EFR32 platforms, in the Hardware Configurator Interface group ensure that the **Enable integration with Hardware Configurator** checkbox is checked.

Printing Tab

In Command Line Configuration, uncheck the **Enable command line interface** checkbox. Interaction with the NCP firmware will be through the TMSP binary protocol with the host processor.

Plugins Tab

For EFR32 platforms only:

In the HAL section, select the **NCP SPI Link** or **NCP UART Link** plugin, as appropriate for your application. In the Hardware Configurator Dependent Module options for the plugin, check the **Enabled** box for the SPINCP or UARTNCP peripheral, respectively. Optionally, change the USART port.

In the Stack section, select the NCP Library plugin. In the Hardware Configurator Dependent Module options, check the **Enabled** box for the SERIAL peripheral. No other properties for this peripheral need to be set for the NCP to operate correctly.

2.3 Customizations

Hardware Configuration (EFR32 only)

If using custom hardware or if you wish to customize your NCP's peripheral usage, launch the Hardware Configurator tool or use applicable plugin options displayed for related plugins in the **Plugins** tab to configure custom peripheral interfaces.

The following instructions describe how to change the default pins used for SPI or UART communication with the Silicon Labs Thread NCP:

For SPI NCP designs:

- In the Plugins tab, in the HAL section, select the **NCP SPI Link** plugin. In the Hardware Configurator Dependent Module options, check the **Enabled** box for the SPINCP peripheral. This should already be enabled if you are starting from an example application. Check that the **SPI NCP USART Port** setting in the Property listing matches your desired USART for SPI NCP communication and that the **nWAKE** and **nHOST_INT** pin settings match your desired signal pinout for SPI communication.
- Scroll down to the Stack group and select the **NCP Library** plugin, which should already be enabled. In the Hardware Configurator Dependent Module Options check the **Enabled** checkbox for the SERIAL peripheral, if it is not checked already.
- To override the default pin assignments for this SPI port, open the Hardware Configurator tool for your NCP, select the DefaultMode Peripherals view, and click the USARTn peripheral selected as your "SPI NCP USART Port" in the **NCP SPI Link** plugin settings. From there you can check that the **Mode** setting in the Property listing for that USART is set to "Synchronous" and that the pin selection properties in that section match your desired signal pinout for SPI NCP communication, making any pinout changes as necessary.

For UART NCP designs:

- In the Plugins tab, in the HAL section, select the **NCP UART Link** plugin. In the Hardware Configurator Dependent Module options, check the **Enabled** box for the UARTNCP peripheral. This should already be enabled if you are starting from an example application. Check that the **UART NCP USART Port** setting in the Property listing matches your desired USART for UART NCP communication.
- Scroll down to the Stack group and select the **NCP Library** plugin, which should already be enabled. In the Hardware Configurator Dependent Module Options check the **Enabled** checkbox for the SERIAL peripheral, if it is not checked already.
- To override the default pin assignments for this UART port, open the Hardware Configurator tool for your NCP, select the DefaultMode Peripherals view, and click the USARTn peripheral selected as your "UART NCP USART Port" in the **NCP UART Link** plugin settings. From there you can check that the **Mode** setting in the Property listing for that USART is set to "Asynchronous" and that the pin selection properties in that section match your desired signal pinout for UART NCP communication, making any pinout changes as necessary.

Custom Code Implementation Files

Source files, include paths, and libraries that implement custom capabilities should be registered in the corresponding configuration table found on the Other tab. These files will contain implementation for callbacks, event handlers, and messaging customizations.

Callbacks

You may wish to enable and implement certain optional callback functions in your NCP firmware. This can be done by checking the "Is used?" checkbox beside each callback and leaving the "Stub?" checkbox unchecked when you generate your project. Some of the more common callbacks are as follows:

`emberAfMainCallback` - NCP Initialization immediately following NCP reset.

`emberAfInitCallback` - NCP Initialization following Silicon Labs Thread stack initialization.

`emberAfTickCallback` - Custom processing per iteration of NCP main processing loop.

NCP Flow Control

The default configuration is hardware flow control. To turn on software flow control, on the Plugins tab under HAL, select the **NCP UART Link** plugin and check **Use software flow control**.

Note: Software flow control cannot be used through the WSTK's USB-to-serial interface; it requires use of the expansion header (EXP port) on the WSTK to access the UART signals. For assistance in using the expansion header instead of the USB port for serial communication with WSTK, please contact Silicon Labs support.

NCP Event Definition and Handling

Register custom event controls and handlers in the corresponding table on the AppBuilder project Other tab. Implement event triggers and handlers in the custom code implementation files.

Custom Messaging

To implement custom messages between NCP and host, the developer defines and implements the format, parsing, and serialization of the message set. The serialized messages are conveyed between NCP and host as opaque byte strings.

To send a custom message to the host, construct and serialize the message, then send the resulting byte string to the host using the Silicon Labs Thread API function `emberCustomNcpToHostMessage(const uint8_t* message, uint8_t messageLength)`.

To process a custom message received from the host, enable and implement the following handler (callback).

`emberCustomHostToNcpMessageHandler` - Process byte string received from host.

Similarly, to send/receive custom messages to/from the NCP, a compatible host application would use the complementary pair:

`emberCustomHostToNcpMessage`

`emberCustomNcpToHostMessageHandler`

3. EmberZNet PRO

The EmberZNet PRO stack includes an example application that can be configured to work over either SPI or UART. This can be used as starting point for building a customized NCP application, or you can start from a blank application. The first two sections describe these two approaches. The third section details the customizations you can make after starting your application in one of the two ways described.

3.1 Starting from an Example Application

Silicon Labs recommends that you use either the **NCP SPI, NCP UART (HW),** or **NCP UART SW** example applications as a starting point for building a customized NCP application. All of these examples are accessible by beginning a new Simplicity Studio project, choosing **Customizable network coprocessor (NCP) applications** from the list of application frameworks, choosing the relevant EmberZNet SDK version if you have more than one installed, and then choosing an example application from the list of provided options.

When you start a new project by clicking the example tile in the Launcher perspective, Simplicity Studio configures the project based on your connected part or scenario. When you start a new project by clicking [**New Project**]and moving through the dialogs, you can select the target part for which the application is being built. Regardless of the creation path, the information in the Architecture box on the General tab should reflect the target part. Regardless of part, the **Device Type** box on the General tab is preconfigured with "No device type selected" for all NCP example applications. The default selections provided by these examples allow them to work on either the EM3xx or EFR32 platform. Therefore, for NCP application purposes, there is no material functional difference between EM3xx and EFR32 device types, so you can leave the Device Type setting unchanged.

If you want to override the default stack settings for your NCP, on the Plugins tab In the Stack Libraries section, select the **ZigBee PRO Stack Library** plugin and alter the plugin properties using the fields to the right of the plugin list. For example, to change the maximum number of supported end device children from the default of 32, click the **Child Table Size** parameter and input the desired maximum number of end device children you can join directly to the NCP. Note that, while the on-screen text says the value range is 0-127, you cannot build the app if you enter a value greater than 64. The precompiled NCP binaries are limited to 32 children.

3.2 Starting from a Blank Application

If, instead of beginning with one of the example applications, you prefer to begin with a blank application, make the following changes under the indicated AppBuilder project tabs to configure basic NCP functionality. Following these changes, other customizations can be made as described in section 3.3 Customizations.

General Tab

(optional) Change the **Device Name** field to match your project name, so that the resulting binary will have a distinct name.

Verify that the information shown in the architecture box matches the part you are targeting. If necessary, click [**Edit Architecture**] to change it.

For Device Type:

1. Click [**No device type selected**].
2. Select **EM3xx NCP** or **EFR32** from the list on the left as appropriate for your target architectures. The device type selections merely choose a set of included default plugins for the build.
3. Click [**Add Device** →]. Make sure the selected NCP device type is added to the list on the right.
4. Click [**OK**].

HAL Tab

For EFR32 platforms, in the Hardware Configurator Interface group ensure that the **Enable integration with Hardware Configurator** checkbox is checked.

Plugins Tab

- In the Core section, ensure that only one of the three NCP communication options (**NCP - SPI**, **NCP - UART**, or **NCP - USB**) is enabled and that the selection matches the communication and flow control style (if applicable) used by your host. Note that USB communication is only available for parts that have native USB serial controller functionality and is not recommended for new designs. If selecting **NCP - UART**, select the appropriate **Flow Control Type** option from the picklist in the plugin options area so that it matches the UART flow control style used by your corresponding host application. For EFR32 platforms, in the **NCP -SPI** or **NCP – UART** plugin Hardware Configurator Dependent Module options, check the **Enabled** box for the SPINCP or UARTNCP peripheral, respectively. Optionally, change the USART port.
- In the HAL section, enable either the **Debug JTAG** or **Debug JTAG Stub** plugin depending on whether you want to support use of the SerialWire/JTAG pins during program operation.
- In the I/O section, ensure the Serial plugin is enabled. For EFR32 NCP configurations, in the Hardware Configurator Dependent Module options check the **Enabled** checkboxes for both the SERIAL peripheral and the USARTn peripheral corresponding to the USART port used for SPI or UART NCP communication.

3.3 Customizations

Plugins

One way to customize your NCP design is through the Plugins tab. Examples of common plugin-related customizations follow.

- For EFR32 platforms, to change the default pins used for EZSP-SPI or EZSP-UART communication, use the following instructions:
  - *For SPI NCP designs*:
    - In the Core section, select the **NCP - SPI** plugin. In the Hardware Configurator Dependent Module options, check the **Enabled** box for the SPINCP peripheral. This should already be enabled if you are starting from an example application. Check that the **SPI NCP USART Port** setting in the Property listing matches your desired USART for SPI NCP communication and that the **nWAKE** and **nHOST_INT** pin settings match your desired signal pinout for SPI communication.
    - Scroll down to the I/O section and select the **Serial** plugin, which should already be enabled. In the Hardware Configurator Dependent Module Options select the SERIAL peripheral and check the **Enabled** checkbox, if it is not checked already. None of the Serial peripheral properties need to be changed for an NCP.
    - Select the USART peripheral corresponding to the USART port setting in the **NCP - SPI** plugin and check the **Enabled** checkbox. Check that the **Mode** setting in the Property listing for that USART is set to “Synchronous” and that the pin selection properties in that section match your desired signal pinout for EZSP-SPI communication.
  - *For UART NCP designs*:
    - In the Core section, select the **NCP - UART** plugin. In the Hardware Configurator Dependent Module options, check the **Enabled** box for the SPINCP peripheral. This should already be enabled if you are starting from an example application. Check that the **UART NCP USART Port** setting in the Property listing matches your desired USART for UART NCP communication.
    - Scroll down to the I/O section and select the **Serial** plugin, which should already be enabled. In the Hardware Configurator Dependent Module Options select the USART peripheral corresponding to the USART Port setting in the **NCP - USART** plugin and check the **Enabled** checkbox. Check that the **Mode** setting in the Property listing for that USART is set to “Asynchronous” and that the pin selection properties in that section match your desired signal pinout for EZSP-UART communication.
- In the EmberZNet Libraries section, **Binding Table Library** plugin, change the “Binding Table Size” parameter to the max desired binding table size used by the NCP.
- In the EmberZNet Libraries section, **Security Link Keys Library** plugin, change the **Link Key Table Size** parameter to the desired maximum number of unique APS link keys used by the NCP. Note that if you are configuring your NCP to act as a Trust Center with ZigBee 3.0 Security (as set in the Security Type area on the Znet Stack tab), it is not necessary to have a unique key table entry for every device. Instead, a single security key known as a Master Key is used to compute unique keys via an AES-HMAC hash function for each device. However, supporting install-code-based keys requires a link key table with as many entries as the number ofinstall-code-based keys you wish to support simultaneously for joining devices with install code support.
- In the EmberZNet Libraries section, **ZigBee PRO Stack Library** plugin, change the **Child Table Size** parameter to the desired maximum number of end device children joined directly to the NCP. Note that, while the on-screen text says the value range is 0-127, you cannot build the app if you enter a value greater than 64. The precompiled images are limited to 32 children.
- In the EmberZNet Libraries section, **ZigBee PRO Stack Library** plugin, increase/decrease other option parameters to meet your needs. You may need to reduce values like Packet Buffer Count, which has a high RAM overhead, if your build fails due to lack of available RAM in the memory map. However, note that most memory-related parameters here simply represent defaults when the NCP boots, and these settings can be overridden by the host during run-time configuration when the NCP is initialized.
- In the Command Handlers section, disable any EZSP command handler plugins for command sets not needed on your NCP, and enable additional EZSP command handler plugins for other features (like Manufacturing Library) that you may need.
- In the EmberZNet Libraries section, change plugins for any stack features not needed on your NCP to stub variants.

  Note: Certificate-based Key Establishment (CBKE) non-stub library plugins are enabled in the blank NCP template by default, but these are only required for NCP devices used in Zigbee Smart Energy networks, and they require an ECC Library component available on request from Silicon Labs technical support. If you do not need Zigbee Smart Energy features in your device, you should replace these CBKE-related libraries with their stub equivalents.

- In the HAL section, if Activity LED is not desired on your NCP, change the **HAL:LED** plugin to the **HAL:LED Stub** plugin.

Callbacks

Another method of customization involves enabling callbacks in the **Callbacks** tab. Select a callback to see detail about the callback. To enable custom code for that callback, activate the “Is used?” checkbox but deactivate the “Stub?” checkbox. When you (re)generate the callbacks C file, it will contain the prototype for that callback function for you to fill in the code. Some of the more common callbacks are as follows:

`emberAfMainCallback` - NCP Initialization immediately following NCP reset.

`emberAfMainTickCallback` - Custom processing per iteration of NCP main processing loop.

Custom Code Implementation Files

Source files, include paths, and libraries that implement custom capabilities should be registered in the corresponding configuration table found on the Other tab. These files will contain implementation for callbacks, event handlers, and messaging customizations.

Security

For devices implementing Trust Center functionality (either as a coordinator providing centralized trust center responsibilities for the network or a router in a decentralized trust center configuration), you may wish to override the EZSP Trust Center policy's decisions about when and how to provide the current network security key to a joining or rejoining device. In EmberZNet PRO releases beginning with version 5.7.1, use the following callback to provide this feature:

```
EmberJoinDecision emberAfPluginEzspSecurityTrustCenterJoinCallback(EmberNodeId newNodeId,
                                                                   const EmberEUI64 newNodeEui64,
                                                                   EmberDeviceUpdate status,
                                                                   EmberNodeId parentOfNewNode,
                                                                   EzspDecisionId decisionId,
                                                                   EmberJoinDecision joinDecision)
```

NCP Event Definition and Handling

Register custom event controls and handlers in the corresponding table on the project Other tab. Implement event triggers and handlers in the custom code implementation files.

Custom Messaging

To implement custom messages between NCP and host, the developer defines and implements the format, parsing, and serialization of the message set. The serialized messages are conveyed between NCP and host as opaque byte strings. This "extensible network coprocessor" functionality is provided by the **XNCP Library** plugin (as opposed to the **XNCP Stub Library** plugin) in the NCP Framework.

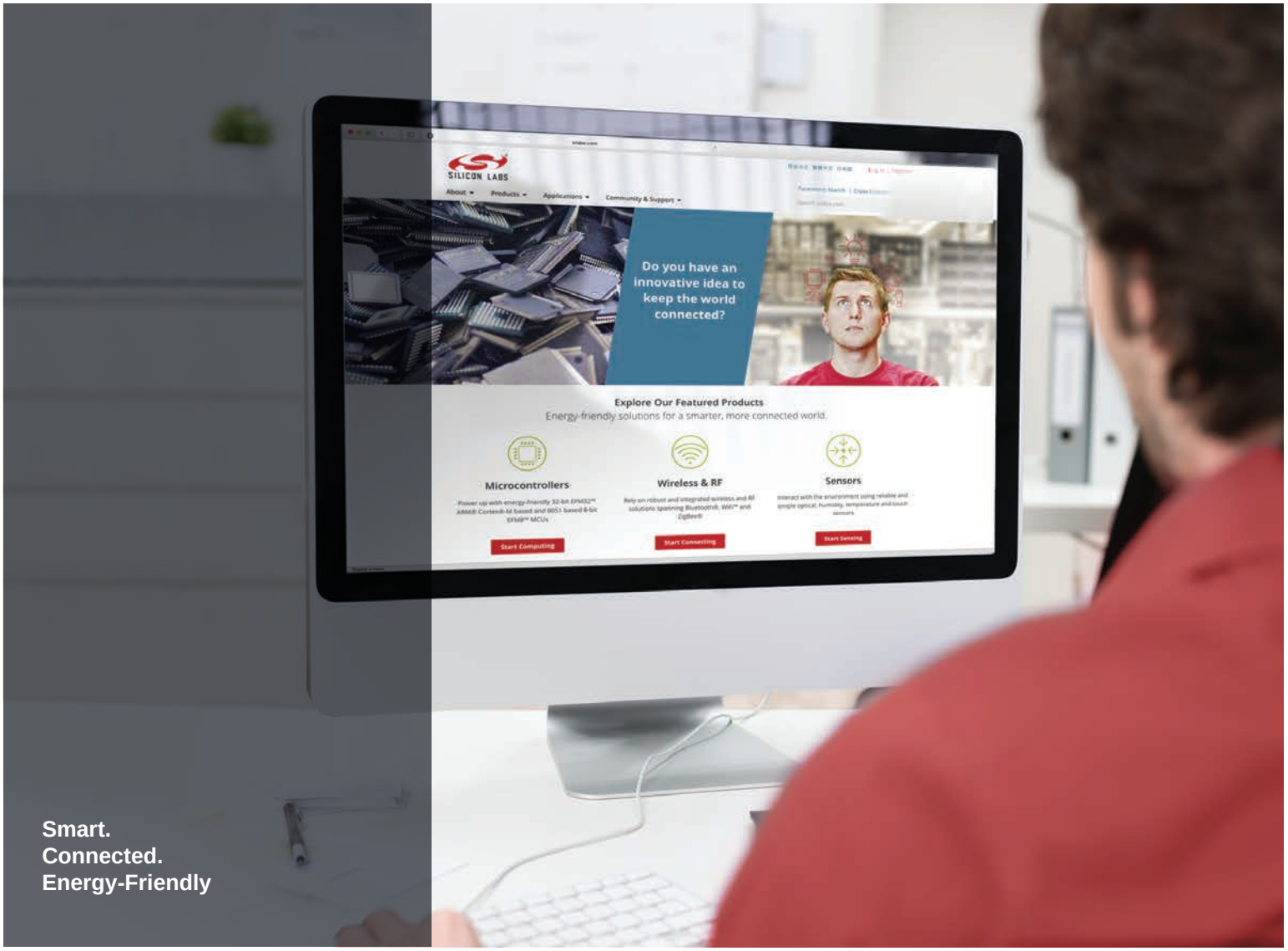
To send a custom message to the host, construct and serialize the message, then send the resulting byte string to the host using the EmberZNet PRO API function `emberAfPluginXncpSendCustomEzspMessage()`.

After enabling the **XNCP Library** plugin, the following callbacks are provided through the Callbacks tab 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 the NCP 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-NCP EZSP command, where they can be handled by the following host-side callback: `void ezspCustomFrameHandler(int8u payloadLength, int8u* payload)`



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

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

http://www.silabs.com