AN706: EZSP-UART Host Interfacing Guide

This application note describes how to connect a Host processor to a Network Co-Processor (NCP) using the UART-based EmberZNet Serial Protocol (EZSP) or EZSP over Co-Processor Communication (CPC). It assumes that you already have a basic understanding of the EZSP-UART Gateway protocol, as well as the signals needed by the UART interface. If not, refer to *UG101: UART Gateway Protocol Reference Guide* before continuing.

Zigbee EmberZNet SDK 7.0 introduced a new component-based architecture, along with a Project Configurator and other tools to replace AppBuilder and plugin configuration. In general, the new software components are comparable to the plugins. When applicable, instructions for both version 7.0 and higher and 6.10.x and lower are provided. For more information, see *AN1301: Transitioning from Zigbee EmberZNet SDK 6.x to SDK 7.x.*

KEY POINTS

- EZSP-UART protocol overview
- Physical interfaces
- Command line options for Host applications
- Hardware design considerations
- Powering up, power cycling, and rebooting
- Bootloading

1. Protocol Overview

Silicon Labs designed EZSP as a protocol to allow communications between components running pieces of the EmberZNet PRO wireless mesh stack, namely a Host processor and an NCP. The Host processor runs the application layer and executes on the POSIX platform such as Mac OS, Linux, or a Windows PC running Cygwin or the new Windows 10 BASH shell. You can also run this on an embedded platform like the Raspberry Pi. This lets you develop and test your application on an easy-to-use platform before porting your solution to a different Host processor with few changes. The NCP runs the EmberZNet PRO stack and physical layer (PHY). It is deployed and executed on a Silicon Labs wireless mesh chip processor such as the Wireless Gecko EFR32™. As of EmberZNet 7.1.0, EZSP can be run over CPC. This can be done by using Z3GatewayCpc project with a Zigbee NCP, with the Zigbee EZSP CPC component. This new feature enables a more seamless Zigbee integration with Silicon Labs multiprotocol solutions, and it provides heightened security to Zigbee applications.

Co-Processor Communication (CPC) is a protocol developed by Silicon Labs that allows a host system to communicate with a Network co-processor device (NCP), also known as the secondary device. This communication is facilitated over a physical transport link such as UART or SPI. In CPC, data transfers between processors are segmented in sequential packets, and these transfers are guaranteed to be error-free and sent in order.

A key component of this system is the CPC daemon (CPCd), which allows applications on Linux to interact with a secondary device running CPC. The CPCd is distributed as three components: the daemon binary (cpcd), a library that enables C applications to interact with the daemon (libcpc.so), and a configuration file (cpcd.conf).You can learn more about how CPC works for co-processor communication from the following sources:

- *AN1259: Using the Silicon Labs Bluetooth® Stack v3.x and Higher in Network Co-Processor Mode*
- *Gecko Platform: Co-Processor Communication*
- *CPCd Overview*
- *Co-Processor Communication Daemon (CPCd)*
- *AN1351: Using the Co-Processor Communication Daemon (CPCd)*
- *Using the Co-Processor Communication Daemon (CPCd)*
- *Developing with Silicon Labs Wi-SUN: Using Co-Processor Communication Daemon with wsbrd*
- *Libcpc Python Bindings*

2. EZSP-UART Physical Interfaces

The following figure represents the EZSP-UART physical interfaces.

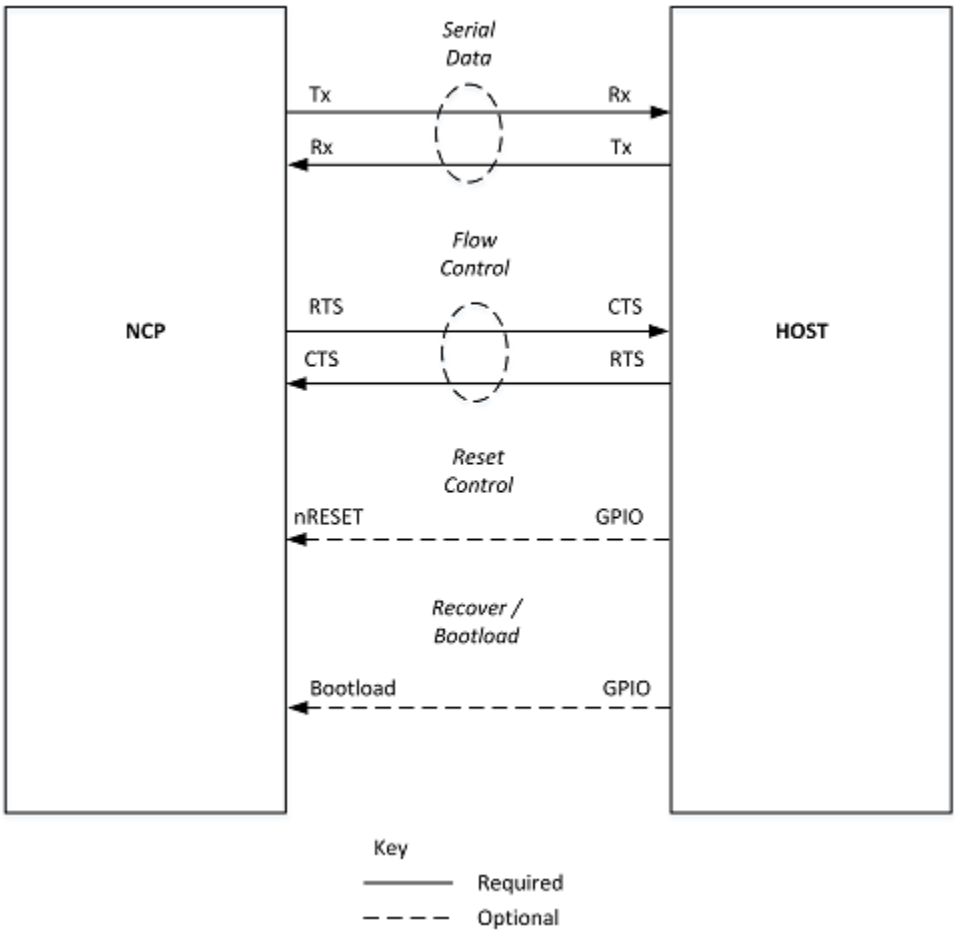

Figure 2.1. EZSP-UART Physical Interfaces

2.1 Serial Data: Tx and Rx

The EZSP protocol sends data in both directions, utilizing a standard Universal Asynchronous Receiver-Transmitter (UART). Both the Tx and Rx pins are utilized on the NCP and Host. When connecting the NCP and Host, the connections should be from NCP Tx to Host Rx and NCP Rx to Host Tx. This interface is required for NCP-to-Host communications.

2.2 Flow Control: RTS and CTS

EZSP requires flow control for operation. This can be in the form of either XON/XOFF software flow control or RTS/CTS hardware flow control. RTS/CTS hardware flow control is part of the standard UART interface. When connecting the NCP and Host, the connections should be from NCP CTS to Host RTS and NCP RTS to Host CTS. With this interface, RTS enables transmission from the Host to the NCP and CTS enables NCP transmissions to the Host.

If needed, the Host can restrain the NCP from sending to it by deasserting the NCP's CTS input. The NCP will stop sending immediately, except to finish sending a byte already in progress. The NCP will do the same to the Host when it requires the Host to stop sending data.

While hardware or software flow control can be utilized, EZSP-UART will have better performance and speed using hardware flow control. Hardware flow control allows operation up to 115,200 bps (other baud rates may be selected by a configuration value programmed into flash during manufacturing).

2.3 Reset Control: nRESET

nRESET is a hardware input that can reset the NCP. For normal operation, the nRESET line should be held high. While pulling the line low, the NCP will hold in a reset state. Upon returning the line high, on the rising edge of the signal, the part will restart operation. Silicon Labs NCP chips include an internal pull-up resistor on this input so adding one is not necessary.

The Host must be able to reset the NCP to run the EZSP protocol. This can normally be accomplished utilizing software reset frames sent via EZSP from the Host to the NCP. However, in cases where the NCP fails to respond to EZSP frames, this hardware backup can be utilized.

While this interface is optional for NCP-to-Host communications, Silicon Labs recommends including nRESET in all Host-NCP hardware designs as a backup for situations where continued NCP operation is impossible or where unforeseen NCP operation blocks the Host.

For secure designs (utilizing secure EZSP) or security products, these connections should be considered mandatory.

2.4 Bootloader Recovery Pin

The Bootload line allows the Host to force an NCP into bootloader mode during a reset or power cycle. For normal operation, the Bootload line should be held high. During a reset (while the nRESET line is held low) or a power cycle, the Bootload line should be pulled low. Upon release of the nRESET line or powering up the NCP, the low input on the Bootload pin will signal the bootload software (Gecko or legacy) to override normal NCP operation, allowing the Host to update the firmware of the NCP. Because the Bootload pin can be configured on any input, a pull-up resistor should be placed on this input or set internally.

In Simplicity Studio, the GPIO associated with this Bootloader Pin can be found in the bootloader project: GPIO Activation component.

Bootloading is normally accomplished utilizing software bootload frames sent via EZSP from the Host to the NCP. However, in cases where the NCP fails to respond to EZSP frames, this hardware backup, in conjunction with nRESET, can be utilized.

While this interface is optional for NCP-to-Host communications Silicon Labs recommends including the Bootload pin in all Host-NCP hardware designs as a backup for situations where unforeseen operation or corruption block the NCP from responding to any software commands from the Host.

For secure designs (utilizing secure EZSP) or security products, these connections should be considered mandatory.

Note that when utilizing EZSP over CPC, the security can be implemented by configuring CPC to run with Security Enabled. This will encrypt all packets between the NCP and the Host.

2.5 Pin Connections

The following table lists the pin connections.

Table 2.1. Pin Connections

Pin number	Description
Tx	Output
Rx	Input
RTS	Input
CTS	Output
nRESET	Input
Bootload	Input, Pull-up

3. Command Line Options for Host Applications

To test your Host-NCP interface Silicon Labs recommends that you build a Host application using Simplicity Studio. (If necessary, refer to *QSG106: Zigbee EmberZNet PRO Quick-Start Guide for SDK v6.10.x and Lower* or *QSG180: Zigbee EmberZNet Quick-Start Guide for SDK 7.0 and Higher* for additional help.) <Z3GatewayHost> is the generic name for a Host application.

To build your application in EmberZNet Zigbee SDK 7.0 and higher, follow these steps:

1. Launch Simplicity Studio.
2. Change your perspective to the Simplicity IDE perspective.
3. Create a new Z3Gateway Host project:
    a. Select **File > New > Silicon Labs Project Wizard**.
    b. Configure the following:
        i. Target Boards: Custom Board.
        ii. Target Device: Linux (32 bit or 64 bit).
        iii. SDK: Gecko SDK Suite 4.0.2.
        iv. IDE/Toolchain: Makefile IDE.
    c. Click **Next**.
    d. Select **Z3Gateway** and click **Next**.
    e. Enter a name for your project and click **Finish**.
4. Simplicity Studio will now display your Z3 Gateway slcp project file. The default settings are enough to test your EZSP settings. This will create the files for your project. The project is a POSIX-compliant Make project.
5. You can transfer your project to your platform and build it using a standard `make` command.
6. Once your project is built, you can run it and use the command line options to test your interface. As long as your application properly interfaces the NCP and properly executes, you can be assured that your connections are working properly. You should see a message similar to this one:

```
./Z3GatewayHost -p /dev/tty.usbmodem0004402507811
Reset info: 11 (SOFTWARE)
ezsp ver 0x08 stack type 0x02 stack ver. [7.0.2 GA build 406]
Ezsp Config: set address table size to 0x0002:Success: set
```

To build your application in EmberZNet Zigbee SDK 6.10.x and lower, follow these steps:

1. Launch Simplicity Studio.
2. Change your perspective to the Simplicity IDE perspective.
3. Create a new AppBuilder Host project:
    a. Select **File > New > Project**.
    b. Select **Silicon Labs AppBuilder Project** and click **Next**.
    c. Select **Silicon Labs Zigbee** and click **Next**.
    d. Select **EmberZNet <version> Host** and click **Next**.
    e. Select the **Z3 Gateway** application and click **Next**.
    f. Enter a name for your project and click **Next**.
    g. On the Project setup page, leave the board section blank. For the part, make sure **None** is selected. The toolchain should also reflect no toolchain as well (or **None**).
    h. Click **Finish**.
4. Simplicity Studio will now be displaying your Z3 Gateway ISC project file. The default settings are enough to test your EZSP settings. Select **Generate**. This will create the files for your project. The project is a POSIX-compliant Make project. You can transfer your project to your platform and build it using a standard `make` command.
5. Once your project is built, you can run it and use the command line options to test your interface. As long as your application properly interfaces the NCP and properly executes, you can be assured that your connections are working properly. You should see a message similar to this one:

```
./Z3GatewayHost -p /dev/tty.usbmodem0004402507811
Reset info: 11 (SOFTWARE)
ezsp ver 0x08 stack type 0x02 stack ver. [7.0.2 GA build 406]
Ezsp Config: set address table size to 0x0002:Success: set
```

The following table summarizes the <Z3GatewayHost> command line options.

Table 3.1. <Z3GatewayHost> Command Line Options

Command Line Option	Optional values	Description
-n	0,1	0=115200 bps (RTS/CTS), 1= 57600 bps (XON/XOFF) Note that, if present, this must be the first option.
-b	<baud rate>	Baud Rate: 9600, 19200, 38400, 57600, 115200, etc.
-f	r, x	Flow control: r=RST/CTS, x=XON/XOFF
-h	-	Display usage information.
-i	0, 1	Enable/disable input buffering.
-o	0, 1	Enable/disable output buffering.
-p	<port name>	Serial port name or number (for example, COM1, ttyS0, or 1)
-r	d, r, c	NCP reset method: d=DTR, r=RST frame, c=custom
-s	1, 2	Stop bits
-t	<trace flags>	Trace B0=frames, B1=verbose frames, B2=events, B3=EZSP
-x	0,1	Enable/disable data randomization.
-d	<OTA directory path>	Sets the directory to search OTA files for Zigbee OTA cluster server role. Note that this option is only present when the Zigbee OTA POSIX Filesystem Storage Module is also present.

4. Design Considerations for Host and NCP Communications

Silicon Labs recommends these best practices for Host and NCP communications:

- Use a well-tuned crystal (38.4 MHz) for any wireless mesh application.
- Use of RTS/CTS hardware flow control is highly recommended. It will aid in electrically-noisy environments.
- UART-based NCP is not suitable for low-power designs due to the inability of the NCP to enter into any sleep modes. In designs where low power NCPs are required, Silicon Labs recommends using SPI-NCP instead. (For more information, refer to *AN711: SPI Host Interfacing Guide for Zigbee*.)
- Silicon Labs recommends including the nRESET and Bootload pins in all Host-NCP hardware designs as a backup for situations where serial communication are lost between the Host and NCP. For secure designs (utilizing secure EZSP) or security products, these connections should be considered mandatory.

5. Powering On, Power Cycling, and Rebooting

When the Host powers on or reboots, it cannot guarantee that the NCP is ready to receive commands. Therefore, the Host should always perform a reset of the NCP to place it into a known state. It is far simpler to reset the NCP and begin from a known state than to try to recover and resync with the previous (unknown) state of the NCP. This reset can be performed either via an ASH RST frame or toggling the nRESET line (if it is enabled). By guaranteeing that the NCP is freshly booted just like the Host, the Host can proceed with standard node and network initialization instead of consuming extra code space just trying to determine what state the NCP was left in.

When the NCP resets, it will issue an RSTACK frame to alert the Host it has reset, which includes the reset type as defined by platform/base/hal/micro/generic/em2xx-reset-defs.h. Once the RSTACK frame is received, the Host will know the NCP is in a known state to start operation.

6. Bootloading

The NCP can enter bootload mode through either a software command or a hardware toggle.

6.1 Software Bootloading

When operating in EZSP mode, issuing the bootloader frame launchStandaloneBootloader will cause the NCP to enter into the bootloader. This can be used at any time when the Host wants to restart the NCP in the bootloader.

6.2 Hardware Bootloading

Asserting the bootloader pin (low) while powering up or resetting the NCP will cause the chip to enter the bootloader instead of resuming normal NCP operation. In the Gecko Bootloader, a recovery pin can be defined in the board configuration to initiate bootloader mode. You define the recovery pin using the Gecko Bootloader's "GPIO Activation" component located in Platform > Utils, where the SL_BTL_BUTTON can be configured.

An example of performing this reset would be as follows:

1. Assert nRESET to hold the NCP in reset.
2. While nRESET is asserted, assert BOOTLOAD, and then deassert nRESET to boot the NCP.
3. The BOOTLOAD line can be deasserted once the reset line has been deasserted.

For more information on using the Gecko Bootloader, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.2 and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

6.3 Entering Bootloader Mode via CPC

If EZSP is being run over CPC, CPCD can set the NCP into bootloader mode. CPCD does this by either sending a software reset frame to the NCP or by directly toggling the NCP bootloader pins, similarly to section 6.2. The user can configure the CPCD bootloader pins in the CPCD.Conf configuration file. Another advantage of using CPCD is that CPCD includes a built in command to upgrade the NCP firmware. This can be done by running CPCD with the -f flag, which points to the firmware file to upgrade. More documentation on firmware upgrade with CPCD can be found in the Silicon Labs CPCD gihtub repository: https://github.com/SiliconLabs/cpc-daemon/blob/main/doc/firmware_upgrade.md.

6.4 Bootloader Operation

The target device's bootloader sends output over its serial port after it receives a carriage return from the source device at the expected baud rate. This prevents the bootloader from prematurely sending commands that might be misinterpreted by other devices that are connected to the serial port. Note that serial bootloaders typically do not enforce any timeout when awaiting the initial serial handshake via carriage return, so the bootloader will wait indefinitely in this mode until guided by the source device or until the chip is reset.

After the bootloader receives a carriage return from the target device, it displays a menu with the following ASCII-based output (where `X.Y.A` corresponds to the major, minor, and sub-minor fields of the bootloader version number, respectively).

```
Gecko Bootloader vX.Y.A
1. upload gbl
2. Run
3. ebl info
BL >
```

Once this is received, the Host can be assured that the NCP is operating in bootloader mode.

Smart. Connected. Energy-Friendly.

IoT Portfolio
www.silabs.com/products

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