AN844: Guide to Host/Network Co-Processor Communications Using Silicon Labs Thread and Connect

This document describes how to set up and test UART communication between a host and NCP (Network Co-Processor) using ASHv3-UART, and how to load NCP. It assumes that you have a Raspberry Pi, USB cable (for UART communication), and a development board. It applies to Silicon Labs Thread stack version 1.0 or later, and Silicon Labs Connect stack version 1.2 or later, provided as part of the Flex SDK.

KEY FEATURES

- Hardware configuration information.
- Building and using the ASHv3-UART test application
- Building and using the standard UART host application
- Building and using a host bootloader ap-plication

1 Introduction

A Network Co-Processor (NCP) runs the Silicon Labs Thread or Connect stack and is controlled by the host processor through ASHv3-UART commands. The NCP must be a chip in the Ember® EM35x family or Wireless Gecko (EFR32™) portfolio. The host is a Linux device such as a Raspberry Pi. ASHv3 is the third revision of the Asynchronous Serial Host (ASH). It is a reliable and efficient UART communication protocol that is used to facilitate NCP and Host communication. ASHv3 operates on the same level as the UART and interfaces directly with it.

This document assumes that the NCP platform is loaded with an application image. Both stacks come with NCP UART examples (one with hardware and one with software flow control) that can be compiled and loaded onto the NCP platform. The NCP platform should also be loaded with a correctly-configured serial bootloader (serial-uart-bootloader). For Silicon Labs Thread, *.hex images contain both a bootloader and application. *.ebl and *.s37 images contain just an application; a bootloader must be loaded separately when using these images. For Connect, building an example in Studio generates both types of images: Bootloader+application and application alone. See *AN961: Bringing up Custom Nodes for the Mighty Gecko and Flex Gecko Families*, and *AN710: Bringing Up Custom Devices for the EM35x*, to learn how to configure a serial bootloader on an EFR32 and EM35x, respectively.

The Linux host can be loaded with the ASHv3-UART test application (ash-v3-test-app) or, for Silicon Labs Thread, with ip-driver-app and a host application, referred to in these instructions as [host-app]. When ip-driver-app is loaded, you can also load the host bootloading application (bootload-ncp-uart-app).

This application note describes the following:

- Hardware configuration information.
- Building and using the ASHv3-UART test application
- Building and using the standard UART host application
- Building and using a host bootloader application

For additional information on using Simplicity Studio and building Thread or Connect applications, see *QSG113: Getting Started with Silicon Labs Thread*, or *QSG138: Getting Started with the Silicon Labs Flex SDK for the Wireless Gecko (EFR32™) Portfolio.* For additional information on ASHv3, see *UG115: ASHv3 Protocol Reference*.

2 Hardware Configuration

For UART NCP, Silicon Labs uses USART0 directed to the same pins that are used for the virtual COM port, as shown in the following table.

Table 1. EFR32 and EM35x Pin Configurations

Function	EFR32 Pin	EM35x Pin	Peripheral	Location	Description
TX	PA0	PB1	USART0_TX	0	Data output from NCP
RX	PA1	PB2	USART0_RX	0	Data input to NCP
CTS*	PA2	PB3	(N/A)*	(N/A)*	Clear to send hardware flow control input to NCP
RTS*	PA3	PB4	(N/A)*	(N/A)*	Request to send hardware flow control output from NCP

*CTS and RTS are software enabled

For UART NCP implementations on EFR32-based development kits, these pins are also on the WSTK expansion header. Only UART TX and UART RX are labeled (12 and 14), but CTS and RTS can be found at 3 and 5 respectively.

If you are using an EM35x UART NCP over a USB-to-serial connection (J5 on the EM35x Breakout Board), set the jumpers on the NCP breakout board to connect the four serial port signals (TX, RX, CTS, RTS) to USB and disconnect them from the ISA3. Access to these signals using direct TTL serial connections or the DB9 port is also possible. Refer to *TS6: EM35x Breakout Board Technical Specification*, specifically the section entitled "Serial Communication for EM35xx SC1 UART", for further information about configuring the EM35x Breakout Board to access the UART signals.

Connect the Linux host, and make sure that it recognizes the USB connection, if using USB-to-serial interfacing to the UART NCP.

The device entry for the newly connected USB or UART device can vary by operating system but will typically have the form of /dev/tty<extension>, for example: /dev/ttyUSB0 (EM35x) or /dev/ttyACM0 (EFR32). Instructions later in this document reference these device formats.

3 Testing the Interface

Two applications are needed for testing: NCP image and ash-v3-test-app.

3.1 ash-v3-test-app

ash-v3-test-app runs on your host and provides tests to verify that your ASHv3-UART implementation is working. This test program is much simpler than a full-fledged sample application. As a result, it can more efficiently pinpoint ASHv3-UART interface problems.

ash-v3-test-app has two modes of execution—interactive and non-interactive. In interactive mode, the application is operated by an interactive console. In non-interactive mode, the application is operated by command-line arguments. By default, ash-v3-test-app runs in interactive mode. Invoking ash-v3-test-app with the arguments `--test-echo`, `--test-bootstrap` or `--test-xon-xoff` forces it to run in non-interactive mode. (See the **Testing** section for a detailed explanation of these arguments.)

Note: ash-v3-test-app does not enable any wireless functionality because its purpose is to validate proper operation of the ASHv3 interface between the host and the NCP.

3.2 ash-v3-test-app Command Line Options

ash-v3-test-app takes the following arguments:

`--uart <uart_file>`: Specify the UART file descriptor, (see section **Hardware Configuration** for examples for your chip type). This option is required.

`--test-echo <echo_string>`: Run an echo test. For more information, see section **Echo Testing**.

`--test-bootstrap`: Run a bootstrap test. For more information, see section **Bootstrap Testing**.

`--test-xon-xoff`: Run an XON/XOFF test. For more information, see section **XON/XOFF Testing**.

`--verbose`: Display all TX'd and RX'd ASHv3 packets.

3.3 Testing

ash-v3-test-app provides tests to verify that your ASHv3-UART implementation is working properly. It verifies that an ASHv3 handshake can be performed and it also provides an 'echo' command. To build the application, execute the following command in the base release directory (building from a directory other than this is not guaranteed to work):

```
make –f <path_to_ash_v3_test_app>/ash-v3-test-app.mak
```

Before executing ash-v3-test-app, identify your TTY driver device for your USB or UART serial connection to the NCP. See section **Hardware Configuration** for TTY driver device examples for your chip type.

3.3.1 Bootstrap Testing

The bootstrap test verifies that ASHv3 can perform an initial handshake between the host and NCP**.** Invoke ash-v3-test-app with the `-u` and `--test-bootstrap` arguments as follows:

```
sudo <path_to_compiled_app>/ash-v3-test-app -u /dev/ttyUSB0 --test-bootstrap
```

The application displays 'ASHv3 is up' when ASHv3 successfully performs an initialization handshake. If a handshake cannot be performed, the application displays 'Failure'.

3.3.2 Echo Testing

The echo test sends a string from the host to the NCP, and the NCP then sends the string back to the host. The host verifies that the received string matches what was sent. Invoke ash-v3-test-app with the `–u` and`--test-echo` arguments as follows:

```
sudo <path_to_compiled_app>/ash-v3-test-app -u /dev/ttyUSB0 --test-echo “this is my string”
```

If the application receives a correctly formatted string back from the NCP, it displays 'Success'. Otherwise it displays 'Failure'.

3.3.3 XON/XOFF (Software Flow Control) Testing

The NCP implements XON/XOFF as follows:

- The NCP tells the host to stop sending it data by sending an XOFF.
- The NCP sends a series of XONs when it can accept serial data again.

Note: The NCP UART Software and Hardware Flow Control examples differ only in that the global defines described below are included in the Software Flow Control example.

EFR32 NCP

- The EFR32 NCP will respond similarly to XON/XOFF bytes from the host
- The thresholds for sending XON/XOFF can be independently configured in software. When the number of free bytes in the receive buffer is less than or equal to COM_USART0_RXSTOP (default=17), the NCP sends an XOFF byte. When the number of free bytes is equal to or greater than COM_USART0_RXSTART (default=21), the NCP sends an XON byte. XON/XOFF flow control is enabled in the application when it is compiled with the global #defines:

```
EMBER_APPLICATION_SUPPORTS_SOFTWARE_FLOW_CONTROL
EMBER_SERIAL1_XONXOFF
```

EM3x NCP

- The EM3X NCP ignores any XON or XOFF bytes sent by the host.
- The EM3X NCP sends an XOFF when its receive buffer fills up to a threshold. It sends a series of five XONs, with 100 milliseconds between each, when the buffer drains down to a second lower threshold. The NCP also sends the same sequence of XONs after it is reset. These additional XONs prevent the UART from hanging if an XON is lost or corrupted by noise, or if another byte sent by the NCP is corrupted into an XOFF. XON/XOFF flow control is enabled in the application when it is compiled with this global #define:

```
EMBER_APPLICATION_SUPPORTS_SOFTWARE_FLOW_CONTROL
```

The XON/XOFF test verifies that the NCP sends XONs when either of its RX buffers is full, and sends a series of XONs after it services its RX buffers. Invoke ash-v3-test-app with the `–u` and `--test-xon-xoff` arguments as follows:

```
sudo <path_to_compiled_app>/ash-v3-test-app -u /dev/ttyUSB0 --test-xon-xoff
```

The application displays 'Passed' upon success, and displays an error upon failure.

3.4 Debugging

If the steps in the **Testing** section fail, try these debugging tips:

- Some problems can be debugged by viewing the ASH frames that are sent between the host and NCP. To do this, invoke ash-v3-test-app with the --verbose argument:

```
sudo <path_to_compiled_app>/ash-v3-test-app -u /dev/ttyUSB0 --verbose
```

- Verify that a tty device exists in `/dev`, such as `/dev/ttyUSB0`. It might have a different name, such as `/dev/ttyUSB1`.
- If ash-v3-test-app is completely unable to communicate with the NCP, attach a logic analyzer to the UART TX and RX lines, and verify that traffic is being sent.

4 ip-driver-app (Thread Only)

ip-driver-app is an application that connects Silicon Labs Thread host applications with NCP devices. It connects the Linux networking stack with the Thread wireless mesh, and enables seamless communication between them.

Execute the following command in the base release directory to build the application:

```
make –f app/ip-ncp/ip-driver-app.mak
```

An example to start ip-driver-app and a host, when the NCP is an EM35x:

```
sudo ./build/ip-driver-app/ip-driver-app -u /dev/ttyUSB0 -t tun0 -m 4901 &
sudo [host-app] -m 4901
```

where

```
-u: Specifies the USB device (see section Hardware Configuration for examples)
-t: Specifies the tun0 virtual network interface.
-m: Specifies the port for the host app management plane (4901)
```

All options are required.

Troubleshooting:

- If ip-driver-app reports "connect failed: Connection refused", check whether an instance of ip-driver-app is already running. If so, kill the process.
- Check ip-driver-app.log for clues.

5 bootload-ncp-uart-app

This application provides the platform-specific means to bootload an NCP over UART. It transfers a file to the NCP via the xmodem protocol, then reboots the NCP. The bootload application assumes that the serial standalone bootloader is already running on the NCP. With Silicon Labs Thread, it can be co-resident with the ip-driver-app, but that process will terminate when the bootloader launches. The process is not required and must not be running when bootloading.

To launch the serial standalone bootloader, the host application must call

emberLaunchStandaloneBootloader(STANDALONE_BOOTLOADER_NORMAL_MODE)

To build the application, execute the following command in the base release directory:

For Thread:

```
make -f app/util/bootload/serial-bootloader/bootload-ncp-uart-app.mak
```

For Connect:

```
make –f connect/plugins/serial-bootloader/bootload-ncp-uart-app.mak
```

To start the application:

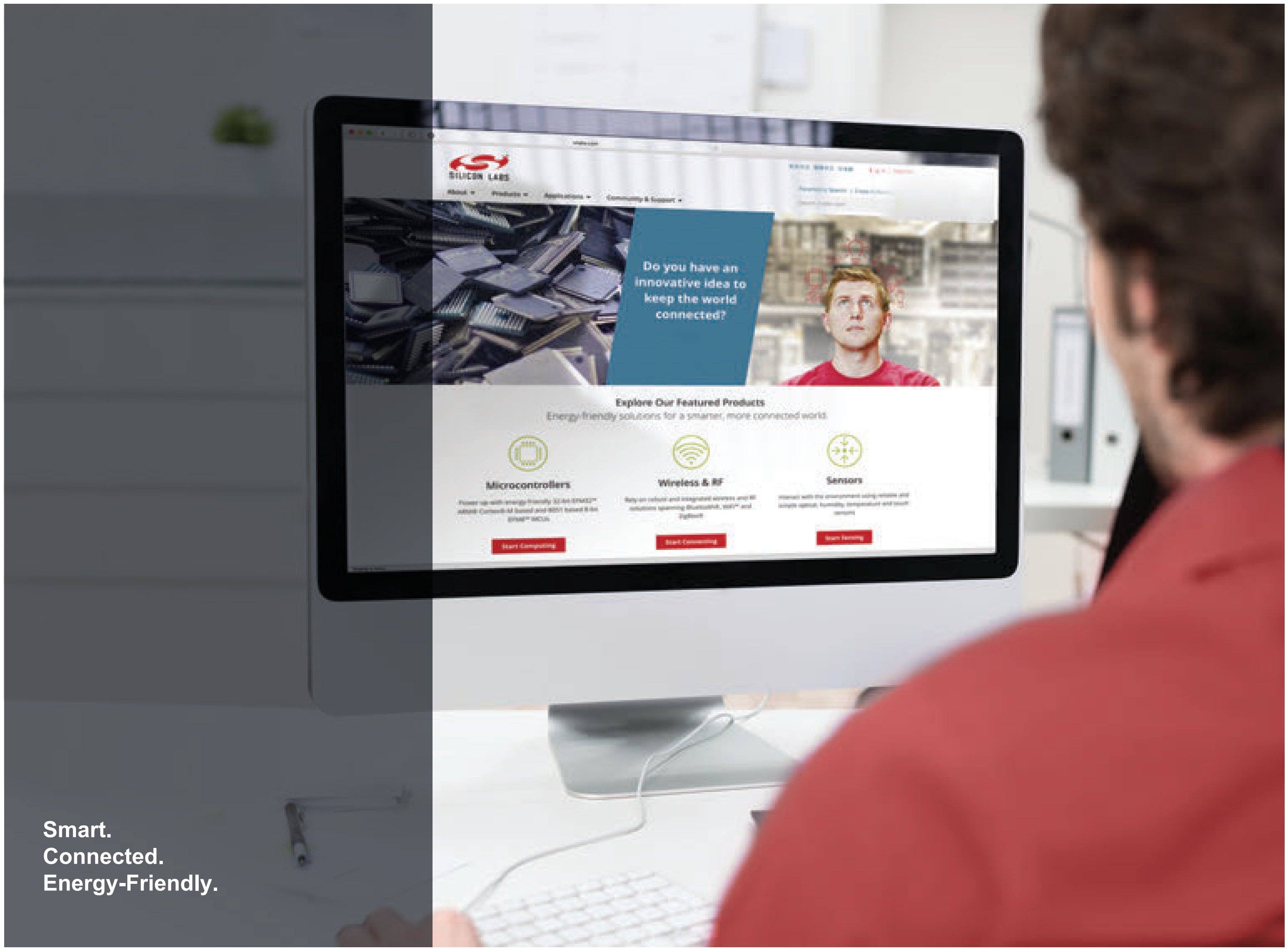
```
sudo ./build/bootload-ncp-uart-app/bootload-ncp-uart-app <image-path> <begin-offset> <length> [serial options]
```

for example:

```
sudo ./build/bootload-ncp-uart-app/bootload-ncp-uart-app ./em3588-ncp-uart.ebl 0 0xFFFFFFFF -p /dev/ttyUSB0
```

where:

```
<image-path>          Pathname to image (required)
<begin-offset>        Offset within image to begin (required)
<length>              Number of bytes to send, 0xFFFFFFFF = everything to end of file (required)
[serial options] {ncp type} {options}
   ncp type:
   -n 0,1             0=EM2xx/EM3xx @ 115200 bps, RTS/CTS
                      1=EM2xx/EM3xx @ 57600 bps, XON/XOFF (if present must be the first option)
   options:
   -b <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>          Serial port name or number (eg, 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
   -v[base-port]      Enables virtual ISA support (optional).
                      Both serial ports are available via telnet instead of local console. RAW serial
                      port is available on the first port (offset 0 from base port), and CLI is
                      available on the second port (offset 1 from base port). By default, 4900 is the
                      base-port, therefore RAW access is available from port 4900, and CLI access is
                      available on port 4901.
                      NOTE: No space is allowed between '-v' and [base-port].
   -x 0,1             Enable/disable data randomization
```


Smart.
Connected.
Energy-Friendly.
Do you have an innovative idea to keep the world connected?
Explore Our Featured Products
Energy-friendly solutions for a smarter, more connected world.
Microcontrollers
Wireless & RF
Sensors

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