AN912: SPI Host Interfacing Guide for Thread

This document describes how to set up and test communication between a host and Network Co-Processor (NCP) using the SPI (Serial Peripheral Interface). It assumes that you have a Raspberry Pi, jumper wires (for SPI communication), and a development board. It applies to Silicon Labs Thread stack version 1.0.4 or later. This document assumes familiarity with Silicon Labs Thread and its associated development environment. If you are new to either, refer to QSG113, *Getting Started with Silicon Labs Thread*.

KEY FEATURES

- Host/NCP signal configuration
- Physical interface configuration
- Low power operation and signal configu-ration
- Building and using host application
- SPI bootloader

1 Introduction

A Network Co-Processor (NCP) runs the Silicon Labs stack and is controlled by the host processor through serial 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.

This document assumes that the NCP platform is loaded with an application image (ncp-spi). The NCP platform should also be loaded with a correctly-configured serial bootloader (ezsp-spi-bootloader).

The Linux host is loaded with the spi-server application, the ip-driver-app, and the top-level host application, referred to in these instructions as [host-app]. You can also load the host bootloading application (bootload-ncp-spi-app).

This application note describes the following:

- Hardware configuration information.
- Building and using the SPI host applications
- Building and using the host bootloader application

2 Physical Configuration

The following illustrates the host-NCP signal configuration.

Table 1. Host-NCP Signal Configuration

	Signal	
Linux Host (spi-server ip-driver-app [host-app])	< nHOST_INT*	NCP (ncp-spi)
	nRESET >*	
	nWAKE >*	
	nSSEL >**	
	< SPI_MISO	
	SPI_MOSI >	
	SPI_CLK >	

*The pins used for these signals are specified when you start spi-server.

**The pin for this signal can optionally be specified when you start spi-server. If it is not, spi-server will assume it is automatically handled by the SPI driver.

The following shows the pins used for the signals in different hardware configurations.

Table 2. Host-NCP Pin Configurations for BRD4151A Rev B02

Function	EFR32 Name	WSTK header/pin	RasPi name	RasPi Header & Pin
Host interrupt (nHOST_INT)	PD10	EXP7	GPIO22	P1-15
Reset (nRESET)	RESETn	DEBUG10	GPIO23	P1-16
Wake (nWAKE)	PD11	EXP9	GPIO24	P1-18
SPI Chip Select (nSSEL)	PC9	EXP10	SPI0_CE0_N	P1-24
SPI MISO (SPI_MISO)	PC7	EXP6	SPI0_MISO	P1-21
SPI MOSI (SPI_MOSI)	PC6	EXP4	SPI0_MOSI	P1-19
SPI Clock (SPI_CLK)	PC8	EXP8	SPI0_SCLK	P1-23
Ground	VSS	EXP1	Ground	P1-6

Connect a pull-up resistor of roughly 10 kΩ from the host interrupt line to 3.3 V. A convenient place for this is on the bottom breakout header of the WSTK (located below the LCD and buttons). 3.3 V is labeled VMCU and PA4 is labeled "P4" on the bottom row.

The Linux host must have the GPIOs used for the above signals configured as In/Out with or without pull-up as appropriate for your board. The host must supply a pull-up on the nHOST-INT signal, as described in the following section.

2.1 Physical Interface Configuration

The NCP supports SPI Slave Mode 0 (clock is idle low, sample on rising edge) at a maximum SPI clock rate of 5MHz, as illustrated below.

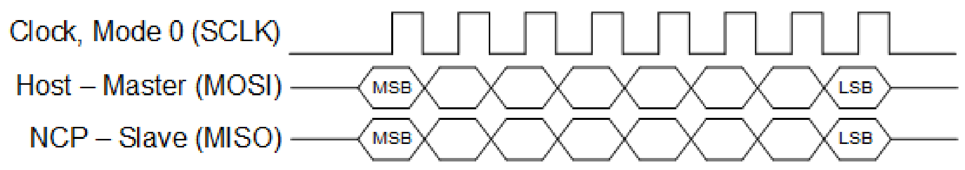

Figure 1. SPI Transfer Format

The nHOST_INT signal and the nWAKE signal are both active low. The Host must supply a pull-up resistor on the nHOST_INT signal to prevent errant interruptions during undefined events such as the NCP resetting. The NCP supplies an internal pull-up on the nWAKE signal to prevent errant interruptions during undefined events such as the Host resetting.

2.2 Low-Power Operation and Signal Configurations

To minimize current consumption when the NCP is sleeping, the host needs to use a configuration that does not conflict with the NCP's configuration. The following table describes the NCP's signal configuration in sleep.

Table 3. NCP Signal Configuration in Sleep

Signal	EFR32 Configuration	EM35x Configuration
MOSI	Input, pulldown	Input, pullup
MISO	Input	Input, pullup
SCLK	Input, pulldown	Input, pullup
nSSEL	Input, pullup	Input, pullup
nHOST_INT	Output, high	Input, pullup
nWAKE	Input, pullup	Input, pullup

3 Building and Using Host Applications

The Linux host must have SPI configured (/dev/spidevX.Y exists and spidev module is loaded into kernel). If not, the easiest fix is to run “sudo raspi-config” and enable it under "Advanced Options > SPI".

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

```
make -f app/ip-ncp/ip-driver-app.mak
cd tool/spi-server
make spi-server
chmod 777 spi-server.sh
```

To start the SPI host:

```
cd tool/spi-server
sudo  ./spi-server.sh  <port>  <spidevice>  <nHOST_INT>  <nRESET>  <nWAKE>  <logMask>  <nSSEL>*
<no_logging>*
*optional
```

For example: `sudo ./spi-server.sh 4951 spidev0.0 22 23 24 0xFF 8 --nolog &`

`--noLog` overrides any `logMask` values by turning off logging.

`logMask = 0xff` is extremely discrete and logs out 10-100 s of IO per second. Silabs recommends using `0x0f` if `–noLog` is not used. Currently, there are 5 valid trace masks values, represented by bits 0-4.

The expected result is: `SPI Server is pid [xxx]`

Next:

```
cd ../..
sudo ./build/ip-driver-app/ip-driver-app -s -u 4951 -t tun0 -m 4901 &
sudo [host-app] -m 4901
```

where

```
-s: specifies to talk to the NCP through a socket rather than a uart device
-u: Specifies socket or uart device
-t: Specifies the tun0 virtual network interface.
-m: Specifies the port for the host app management plane (4901)
```

Troubleshooting:

- If ip-driver-app reports `Connect failed: Connection refused`, check whether an instance of ip-driver-app is already running, and if so, kill the process.
- Check whether the SPI Server failed to start or wrong port was specified to connect to it.
- Check ip-driver-app.log for clues.
- Check tool/spi-server/spi-server.log for clues.

4 SPI Bootloader

This application provides the platform-specific means to bootload an NCP over SPI. It transfers a file to the NCP via a protocol based on XMODEM-CRC, then reboots the NCP. It can be co-resident with the ip-driver-app and spi-server, but both processes will terminate when the bootloader launches. They are not required and must not be running when bootloading. The bootload application assumes that the ezsp-spi-bootloader is already running on the NCP.

To launch the bootloader, your host application must call:

```
emberLaunchStandaloneBootloader(STANDALONE_BOOTLOADER_NORMAL_MODE)
```

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

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

To start the application:

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

for example:

```
sudo ./build/bootload-ncp-spi-app/bootload-ncp-spi-app ./efr32-ncp-spi.ebl 0 0xFFFFFFFF
```

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] {options}
     options:
     -h                    Display usage information
     -t <trace flags>      trace B0=frames, B1=verbose frames, B2=events, B3=EZSP
                           e.g. to trace for frames and EZSP use bit(0) | bit(3) = 0b001 = -t 9
```