AN1019: Using the NodeTest Application

NodeTest is a pre-built application supplied by Silicon Labs for the purpose of performing RF evaluation, functional testing, and hardware validation on development boards or custom-designed hardware. It contains RF test functions pertinent to IEEE 802.15.4-based radio configurations such as those used by the Silicon Labs EmberZNet PRO and Silicon Labs Thread networking stacks on platforms such as the EM35x, Mighty Gecko, or Flex Gecko families.

Bluetooth Low Energy (BLE) and other non-802.15.4-based modulation schemes are not supported by NodeTest. RF testing of other modulation schemes requires a different application or mechanism, such as BLE's Direct Test Mode (DTM) or a test application based on Silicon Labs Radio Abstraction Interface Layer (RAIL) libraries.

KEY FEATURES

- Uploading and running the NodeTest application
- Performing a simple functional test using NodeTest.

1 Introduction

NodeTest is a pre-built application supplied by Silicon Labs for the purpose of performing RF evaluation, functional testing, and hardware validation on development boards or custom-designed hardware. The NodeTest application provides low-level control of the radio and can be used to perform these tasks:

- Characterize radio performance.
- Set manufacturing and stack parameters (tokens) (EM35x).
- Verify proper functionality after manufacturing.
- Control the radio properly for the certification process required by many countries.

The NodeTest application supports a command parser and provides results in a consistent, easy-to-parse format.

While no source code is available for NodeTest, most of NodeTest’s RF test functionality is also available through the Manufacturing Library API (“mfglib”) exposed in EmberZNet PRO and Silicon Labs Thread through the mfglib library (for SOC platforms) or mfglib serial commands (for NCP platforms). As a result, custom-built applications can incorporate this functionality natively in an application designed specifically for their board configuration without relying on the pre-built NodeTest firmware.

The NodeTest application is included in the EmberZNet PRO and Silicon Labs Thread stack installation directories, both of which contain a set of /build/nodetest-*xxx* subdirectories for a given processor/architecture variant, *xxx*. These directories contain.s37 and ebl file images that can be installed onto a chip through the normal SerialWire- or JTAG-based firmware loading procedure.

Note: The NodeTest file images are built to expect a bootloader in the main flash.

The NodeTest application is also pre-programmed onto the Radio Communication Modules (RCMs) included with the EM35x Development Kit.

2 Uploading and Running the NodeTest Application

The methods for loading and running the NodeTest application differ depending on whether you are working with Mighty Gecko or Flex Gecko (EFR32), or an EM35x. In both cases, you must upload a bootloader first. If you don't see serial output indicating that NodeTest is starting up after having reset the device and pressed a carriage return on one of the serial ports, check that you've uploaded a bootloader to the device and haven't erased the main flash block since that upload was performed.

Note: Every time NodeTest resets, it will not automatically print any characters. Instead, NodeTest will listen on both the Physical UART and the VCOM port looking for a carriage return (Enter key). Whichever port NodeTest finds a carriage return on first will become the port that NodeTest uses for all serial interaction until NodeTest resets again.

2.1 Uploading and Running NodeTest on the Mighty Gecko or Flex Gecko

NodeTest can be installed using either Simplicity Studio to upload the application or using a Command Line Interface (CLI) with Simplicity Commander. The following is an example of loading NodeTest and an accompanying bootloader using the Simplicity Commander CLI:

```
$ commander flash nodetest.ebl ..\..\tool\bootloader-efr32mg1p132f256gm48\serial-uart-bootloader\se-
rial-uart-bootloader.s37
```

Replace the bootloader-efr32mg1p132f256gm48 string with the proper bootloader-xxx subdirectory for your variant and bootloader choice.

The command above flashes the NodeTest firmware and serial UART bootloader firmware to the USB-connected target device during a single process.

Note: Additional --serialno or --ip arguments can be added to the command to specify a particular target WSTK. For more information on how to program chips using Simplicity Commander from a command line, refer to UG162, *Simplicity Commander Reference Guide*.

Once the upload completes, you can interact with NodeTest via the Physical UART or the VCOM port. Using the VCOM port is done by accessing Simplicity Studio's **Launch Console** action from the Adapters view for your connected WSTK and attach to the **Serial 1** tab to interact with the VCOM port.

Press **Enter** in the Console window or serial terminal window to start the NodeTest application.

Note: If you are using the Physical UART directly rather than via the VCOM interface through the WSTK, you must configure your serial terminal program to 115,200 bps, no parity bits, 1 stop bit.

2.2 Uploading and Running NodeTest on the EM35x

NodeTest can be installed using either Simplicity Studio to upload the application or with em3xx_load.exe from a command line. The following is an example of loading NodeTest from a command line:

```
$ em3xx_load.exe ––ip myisahostname ./app/NodeTest/NodeTest-em357.s37
```

For a discussion on how to program chips using em3xx_load.exe from a command line, refer to document UG107, *EM35xx Utilities Guide.*

Once the upload completes, you can interact with NodeTest via the Physical UART or the Virtual UART.

To use the Virtual UART you can

- Telnet to a Debug Adapter (ISA3), port 4900, with the Packet Trace Port cable connected to the chip.
- Use Simplicity Studio's **Launch Console** action from the Adapters view for your connected ISA3 and attach to the **Serial 0** tab to interact with the Virtual UART.

To use the Physical UART you can:

- Use a direct UART connection (as the Breakout Board provides an RS-232 connector).
- Use a USB-to-serial connection (through the Breakout Board's TTL-to-USB converter or through the chip's native USB port in the case of applicable EM358x ICs).

- Use the passthrough UART mode and telnet to the Debug Adapter (ISA3), port 4901, connected to a Breakout Board using the DEI Port.
- Use the passthrough UART mode: use Simplicity Studio's **Launch Console** action from the Adapters view for your connected ISA3 and attach to the **Serial 0** tab to interact with the Virtual UART.

Press **Enter** in the telnet window or com port window to start the NodeTest application.

Note: If using the Physical UART without the Debug Adapter (ISA3), you must configure your comm port program to 115,200 bps, no parity bits, 1 stop bit. If using the Physical UART via the passthrough mode of the Debug Adapter (ISA3), port 4901, configure the Debug Adapter (ISA3) to 115,200 bps by executing the command `ember> port 1 115200`. Execute the Debug Adapter (ISA3) configuration command by telnetting to the Debug Adapter (ISA3) administration interface at TCP port 4902.

3 NodeTest Commands

A Return key initiates the NodeTest application on reset. This displays the power-up prompt, ending in the > (greater than) symbol. Pressing the Return key at the prompt will always cause another prompt to be displayed. Executing the `help` command causes NodeTest to print a list of all available commands with a brief description of the commands. The commands are listed by functional modules. All input parameters to NodeTest are specified in hexadecimal without a “0x” prefix.

4 Performing Functional Testing

The following describes the procedure to use the NodeTest application to perform a simple send/receive test on a target device to determine its range and generally test its radio functionality. A typical usage scenario for the NodeTest application is measurement of Packet Error Rate (PER). PER is defined in 802.15.4 as the percentage of transmitted packets that are not detected correctly. The NodeTest application provides two commands, `rx` and `tx`, that are designed to interoperate with each other across two nodes running this application

Note: When programming a device for test or retest, use the `--erase` option to ensure that all previous calibration data is erased. Silicon Labs recommends erasing the flash contents of a device prior to testing to ensure the calibration is executed at the time the device is tested.

1. Connect a device known to be in good operating order either to your computer or to a different computer through a serial port.
2. Upload NodeTest to the known good device and then run it.
3. Make sure that NodeTest is installed and running on the target device (you should see the > prompt).
4. Set both devices to a channel by typing `setchannel X`, where X is the channel in hex.
5. Optional: Set a power level on the test device by typing `settxpower X`, where X is the power level in hex.
6. On the known good device, type `rx`, which sets the device to receive and display statistics for each packet received. Type `e` to exit the test.
7. On the test device, type `tx X` to transmit X packets where X is in hex. (Note that tx 0 sends infinite packets.) Type `e` to exit the test.

   This command will transmit 10 (hex a) packets of the form required for PER analysis. For actual PER analysis, send 1000 or more packets.
8. Reverse this procedure to test receiving on the test device.

Note: The fourth column in the display output, labeled “per”, shows the packet error rate. For this value to be accurate, the two devices being used must be configured as described in section **Uploading and Running the NodeTest Application**, and the receiver should not hear any other devices. Exiting the test and restarting clears the values and reset the values being displayed. Your packet lengths may be different than the ones shown in the example.

```
> rx
{{(rx)} test start ('e'nd)}
#{{(rx)}
  {num}   {oflo}  {seq}   {per}  {err} {lqi}  {rssi}{ed}   {gain}       {status} {time}       {fp}{length}}
{ {    1} {    0} {    1} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x0000659F} {0} {0x12}  }
{ {    2} {    0} {    2} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x0000CD76} {0} {0x12}  }
{ {    3} {    0} {    3} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x0001354E} {0} {0x12}  }
{ {    4} {    0} {    4} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x00019D26} {0} {0x12}  }
{ {    5} {    0} {    5} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x000204FE} {0} {0x12}  }
{ {    6} {    0} {    6} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x00026CD6} {0} {0x12}  }
{ {    7} {    0} {    7} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x0002D4AE} {0} {0x12}  }
{ {    8} {    0} {    8} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x00033C86} {0} {0x12}  }
{ {    9} {    0} {    9} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x0003A45D} {0} {0x12}  }
{ {   10} {    0} {   10} {  0} {   0} {0xFF} {-41} {0xEC} {0x00000090} {0x4000} {0x00040C35} {0} {0x12}  }
```

Note: NodeTest attempts to print packet data as fast as it can, but it is possible to receive packets faster than NodeTest can print. Therefore, there may be gaps in the printed packets.