AN972: EFR32 RF Evaluation Guide

This version of AN972 has been deprecated.

For the latest version, see docs.silabs.com.

****

This evaluation guide provides an easy way to evaluate the performance of the Wireless Gecko EFR32 devices using the Silicon Labs Radio Abstraction Interface Layer (RAIL). RAILtest is a standalone test application, which is used for testing radio functionality, as well as peripherals, deep sleep states, and so on. Basic transmit and receive commands allow customers to fully evaluate the receiving and transmitting performance and to test RF functionality of development kit hardware or customer hardware. Note that some of the lab tests require RF test equipment, such as a spectrum analyzer and RF signal generator. It is a useful test application to be used for basic testing of your custom EFR32 design and characterization. The test can be used in the laboratory to measure the basic RF parameters of the radio (output power, sensitivity, and so on).

This evaluation guide describes how to install the RAILtest application to EFR32 devices and demonstrates how to make frequently-required, conducted transmission and reception tests using the RAILtest application.

Note: Measuring over-the-air range performance of EFR32 devices, please refer to the Range Test application, available in Simplicity Studio. Refer to *UG147: Range Test Demo User's Guide* for its usage.

The following measurements will be covered on the transmission side:

- Tx power
- Phase Noise
- Power Spectral Density Mask
- Spurious Emission
- Transient power

The following measurements will be covered on the reception side:

- Rx sensitivity
- Selectivity
- Blocking
- Intermodulation
- Maximum input power

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22.

KEY POINTS

- How to evaluate the performances of the Wireless Gecko EFR32 devices
- RAILtest is a standalone test application used for testing radio functionality
- RAILtest can be used in the laboratory to measure the basic RF parameters of the radio

1. Running Simplicity Studio

Simplicity Studio 5 and Flex SDK v 3.x introduced a number of new features and enhancements. This chapter describes how to use Simplicity Studio 5 and Flex SDK v3.x as well as the older Simplicity Studio 4 and Flex SDK 2.x for testing.

Before beginning, you should have installed Simplicity Studio and the Flex SDK according to the instructions in the relevant quick-start guide: *QSG138: Proprietary Flex SDK v2.x Quick Start Guide* or *QSG168: Proprietary Flex SDK v3.x Quick Start Guide*.

Before running RAILtest in either version, the test target must be configured according to the following instructions:

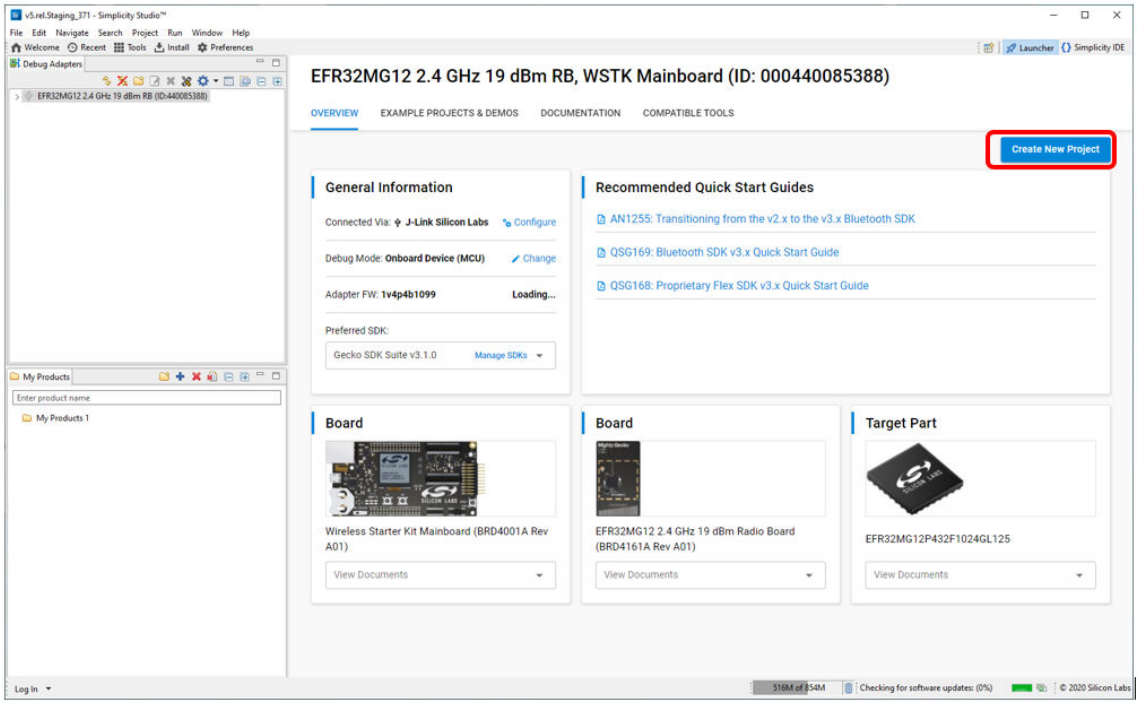
To start, set up an EFR32 development kit radio board with a mainboard for Wireless Starter Kits (WSTK). Once you have installed all the required software you can connect your EFR32 development kit hardware to your PC using a mini-USB cable. Make sure the 3-way power switch in the bottom left is set to AEM.

If you want to connect to your WSTK over Ethernet, plug in an Ethernet cable as well at this time. The IP address will be printed to the LCD screen during startup of the WSTK but may be lost when the app starts. To see this again, reboot the WSTK and press the reset button for several seconds to prevent the EFR32 from loading its application.

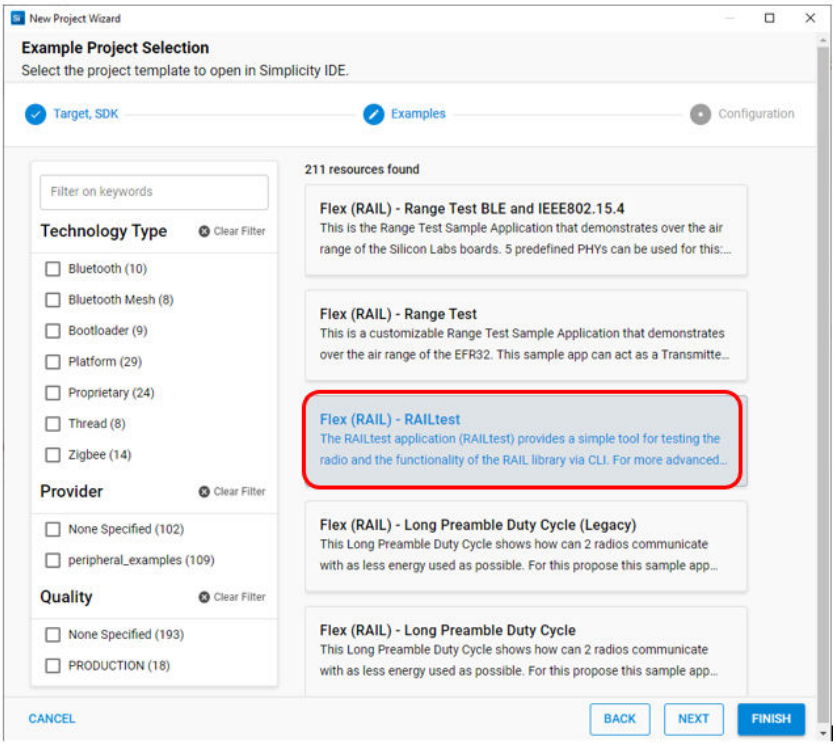
1.1 Select RAILtest application

1.1.1 Start Project In Simplicity Studio 5

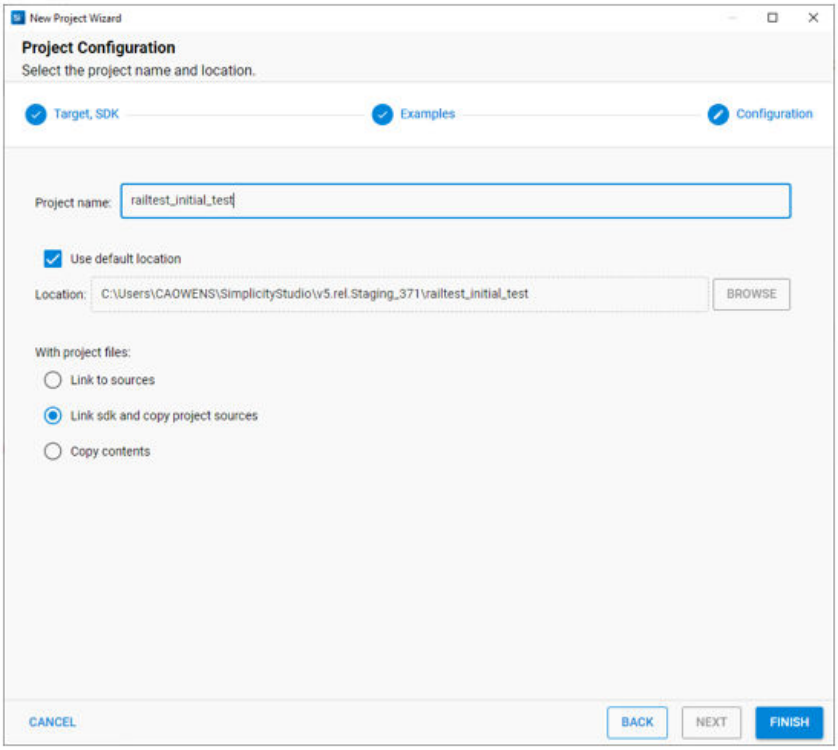
1. When Simplicity Studio 5 opens, select the target in the Debug Adapters view. If you have more than one device connected, select based on the serial number. After connecting the WSTK to the PC, the Start Screen on the LCD shows the device's serial number.
2. Click the OVERVIEW tab if it is not already selected.
3. Click [**Create New Project**].



4. Select the **Flex (RAIL)** -**RAILtest** application, and click [**Next**]. You can use keywords or filters to find the application.

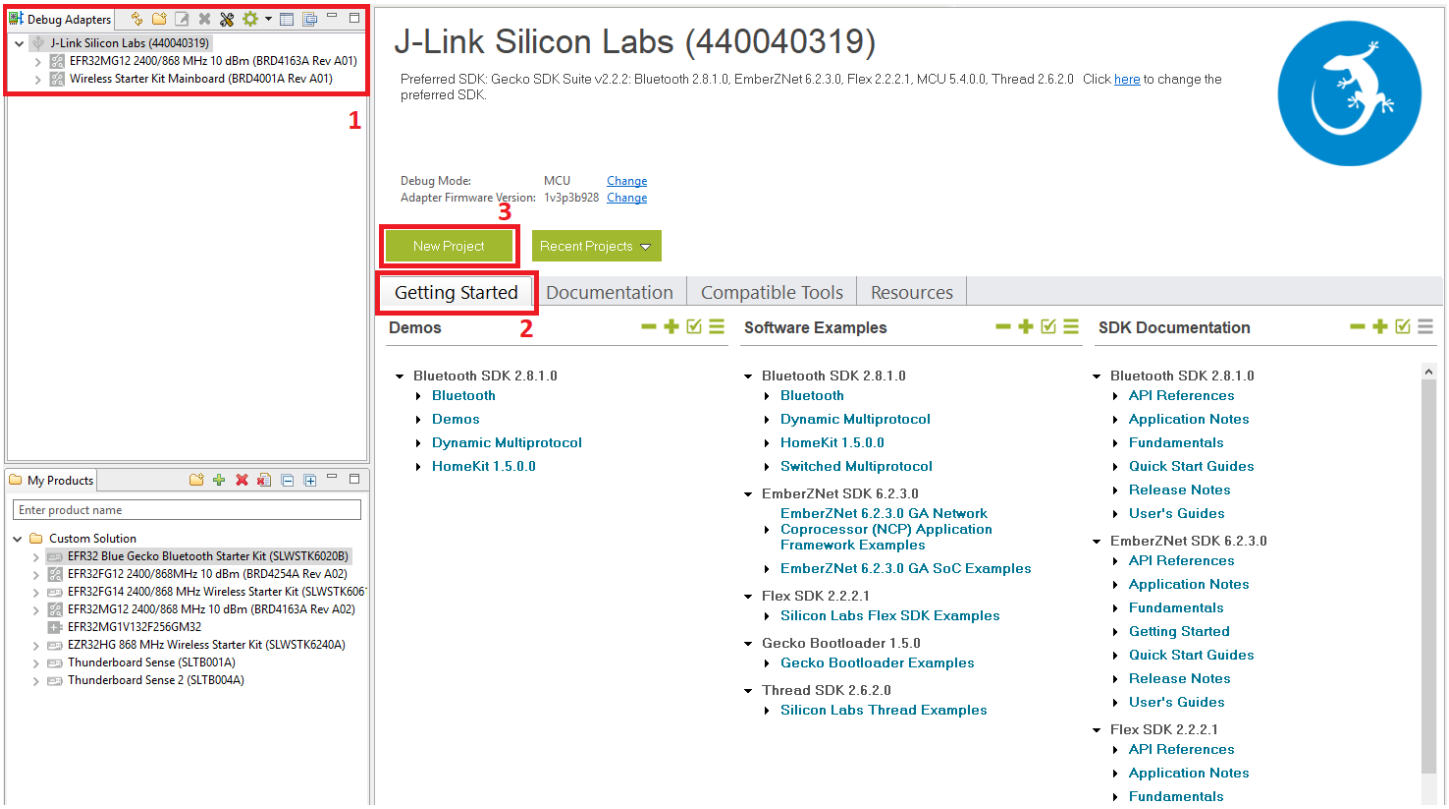


5. Rename the project and/or change the location, if desired. Click [**Finish**].

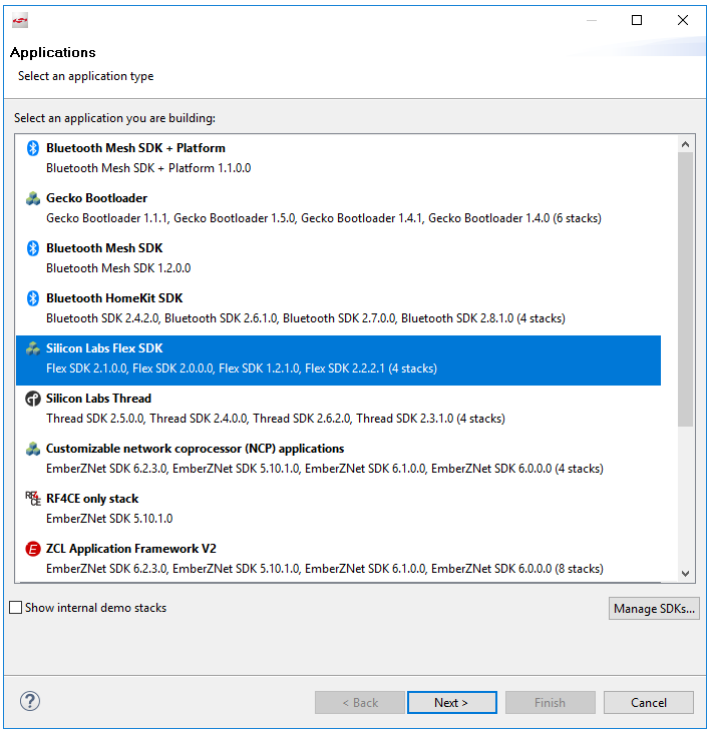


1.1.2 Start Project In Simplicity Studio 4

1. When Simplicity Studio 5 opens, select the target in the Debug Adapters view. If you have more than one device connected, select based on the serial number. After connecting the WSTK to the PC, the Start Screen on the LCD shows the device's serial number.
2. In the Launcher (default) perspective, if it is not already selected, click the **Getting Started** tab.
3. Click [**New Project**].

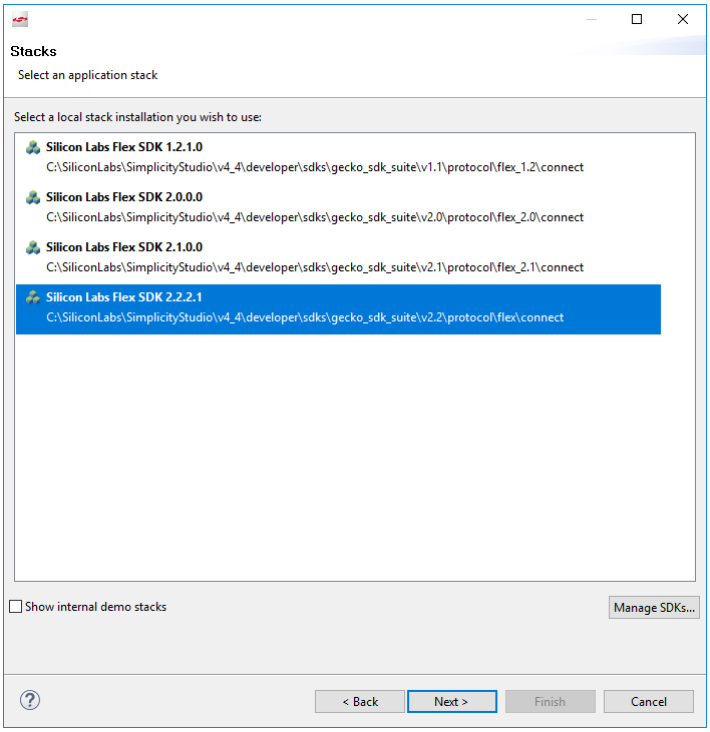


4. Select **Silicon Labs Flex SDK** as the application type.

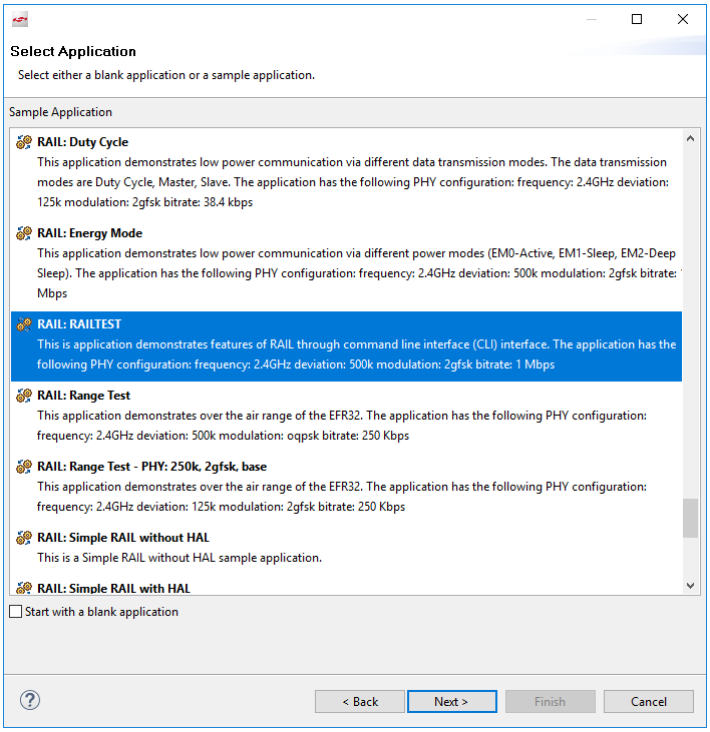


5. Click [**Next**] .

6. Select the Flex SDK version you would like to use (preferably the latest one).

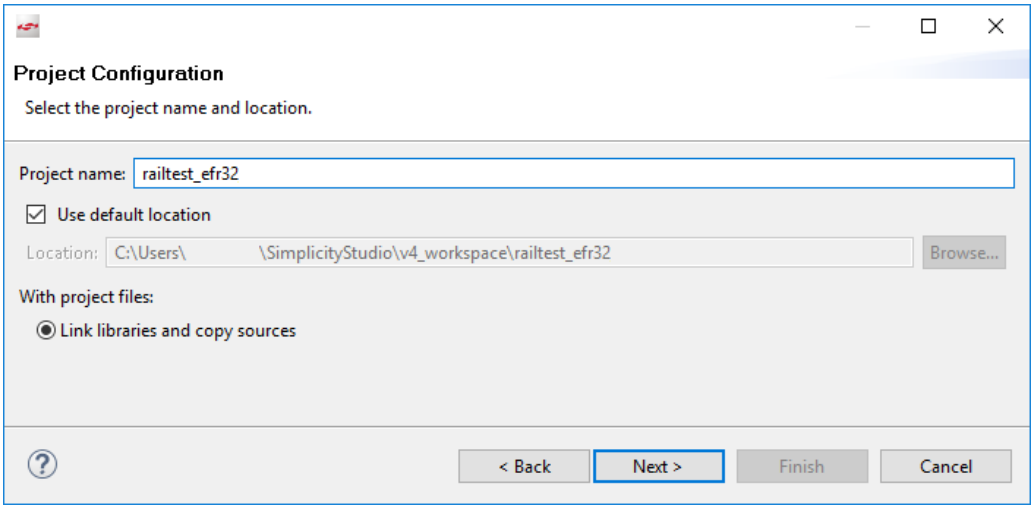


7. Click [**Next**].
8. Select the RAIL: RAILTEST application.



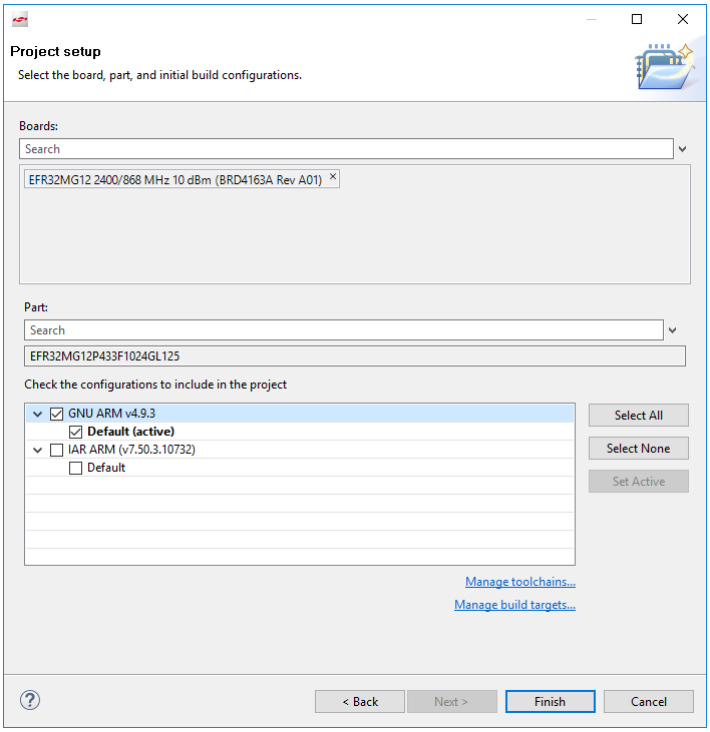
9. Click [**Next**].

10. In the Project Configuration dialog, specify a location for your application.



Note: the location must be on the same Windows partition as the stack.

11. Specify a name for your application.
12. Click [**Next**].
13. The Project Setup dialog shows the various build configurations available. Check one of the initial build configuration to include in the project. (You can edit these later through the “Manage Configurations” command.)



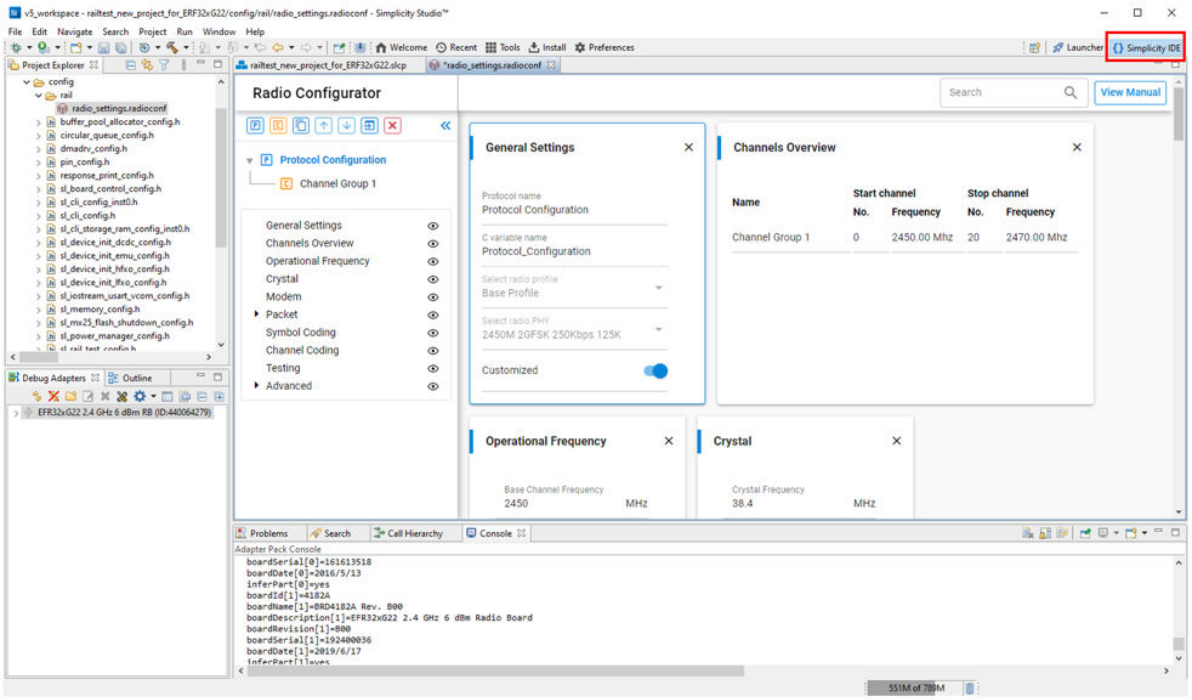
14. Click [**Finish**].

Note: You must have a Toolchain and Build target selected and configured for the [**Finish**] control to enable. If you do not see [**Finish**] enabled, check your Toolchains and Build targets by clicking the links at the bottom of the dialog.

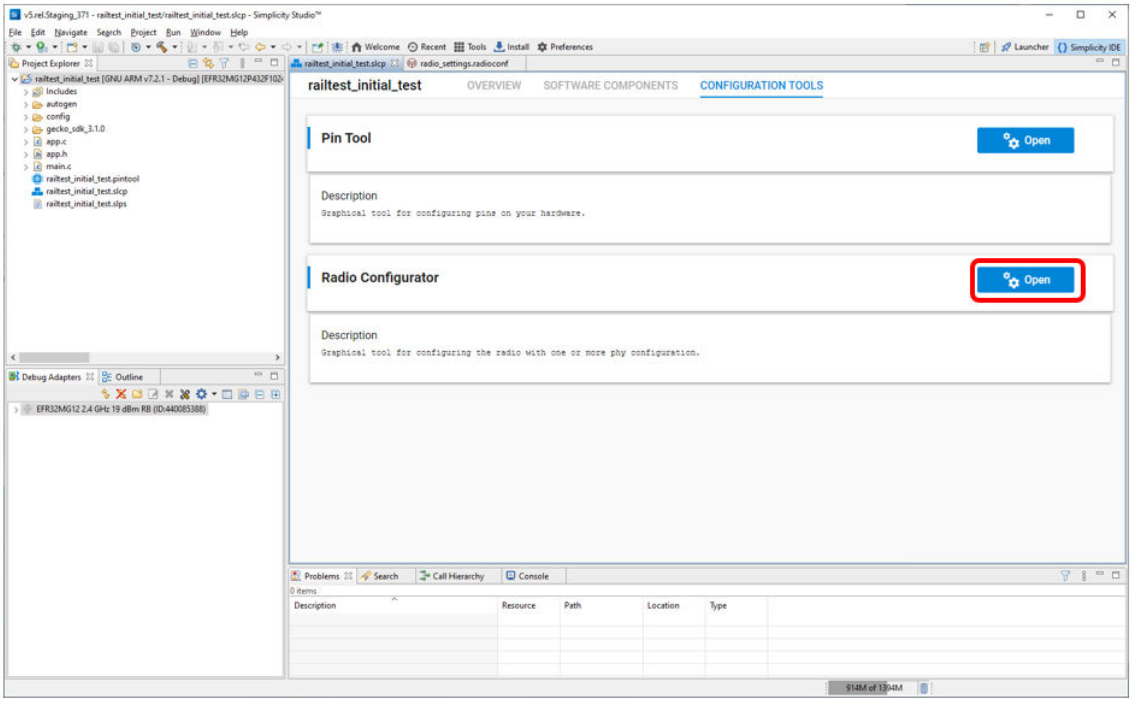
1.2 Configure and Generate the Application

1.2.1 Configure and Generate In Simplicity Studio 5

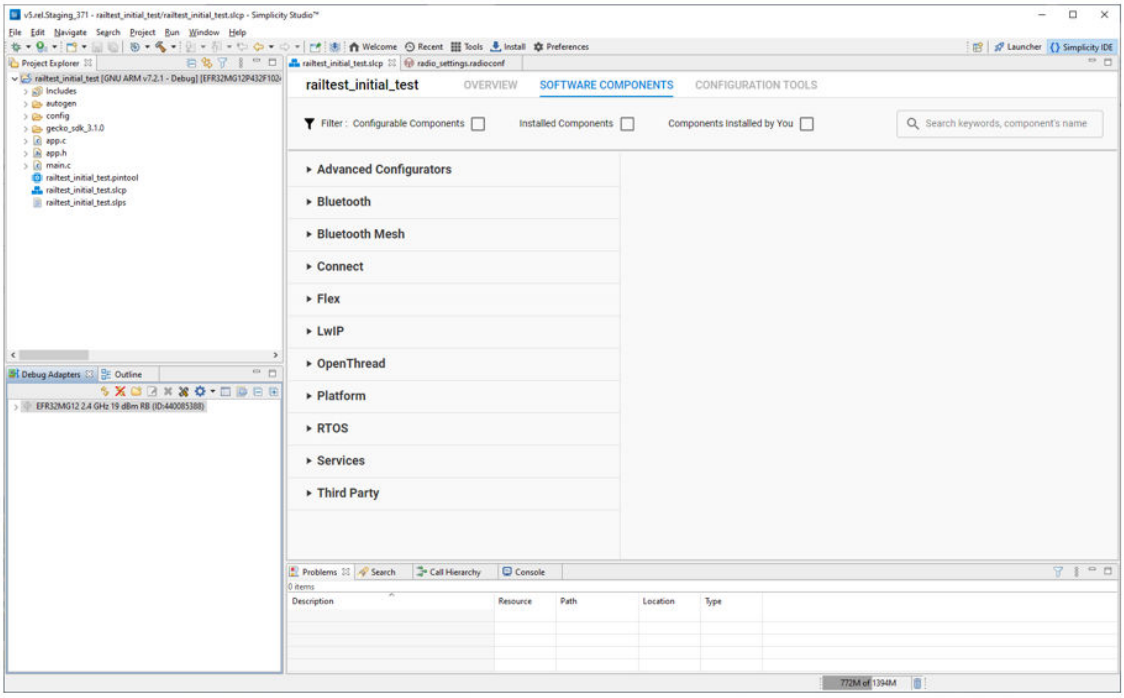
After creating the project, the Simplicity IDE perspective opens. You can switch between Simplicity IDE and Launcher perspectives using the controls in the top right corner. The project explorer view is on the top left. It contains all the files for the opened projects. By default the project's *<project_name>.slcp* and *radio_settings.radioconf* files are already opened. The project is defined in the .slcp file. These two files replace Simplicity Studio 4's .isc extension file.



If the Radio Configurator is not open, on the project .slcp tab click the CONFIGURATION TOOLS tab, and click [**Open**] on the Radio Configurator card. See *AN1253: EFR32 Radio Configurator Guide for Simplicity Studio 5* for more information.



Projects are configured by installing and uninstalling components, and configuring installed components. The SOFTWARE COMPONENTS tab displays categories of components on the left, and details about the selected component on the right. Several filters as well as a keyword search are available to help you explore the various component categories.



Simplicity Studio 5 disables buttons, LEDs and the LCD display in the RAILtest application by default. These features help the user to interact with RAILtest application and are enabled by default in older versions of Simplicity Studio. To enable these features, you may install the following software components: Button 0 (as btn0), Button 1 (as btn1), LED 0 (as led0), LED 1 (as led1) and RAILtest Graphics.

Note: Feature availability may differ on different boards.

These components are in the following component categories:

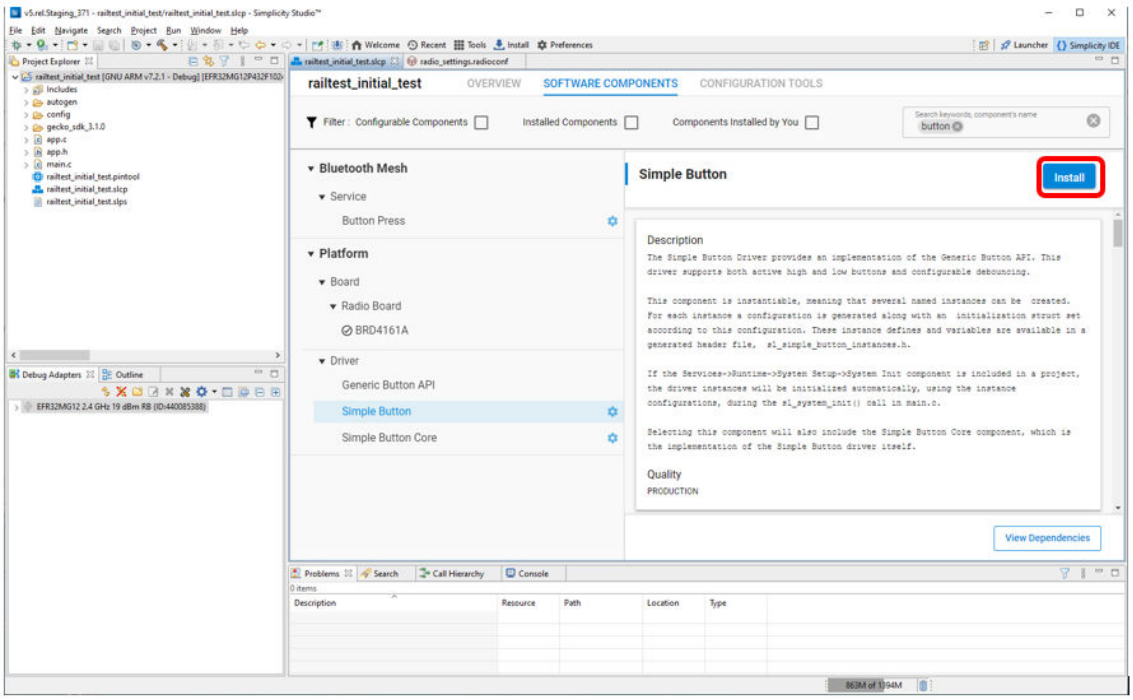
- Platform > Driver > Simple Button > btn0 and btn1
- Platform > Driver > Simple LED > led0 and led1
- Platform > Radio > RAILtest, Graphics

Note: These may install more than one dependency, depending on their function.

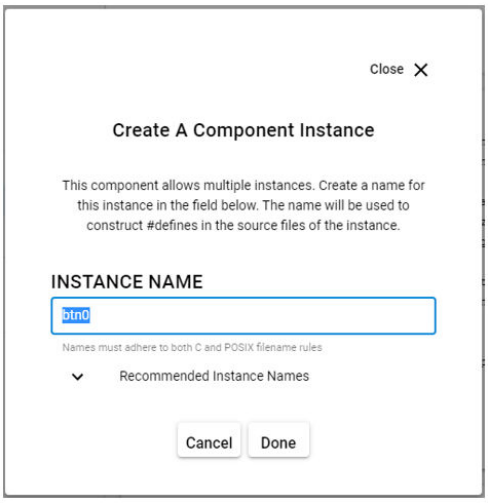
For example, to install a Simple Button:

1. On the Software Components tab, enter 'button' in the search field in the top right corner.

2. Select the Simple Button component and click[**Install**].



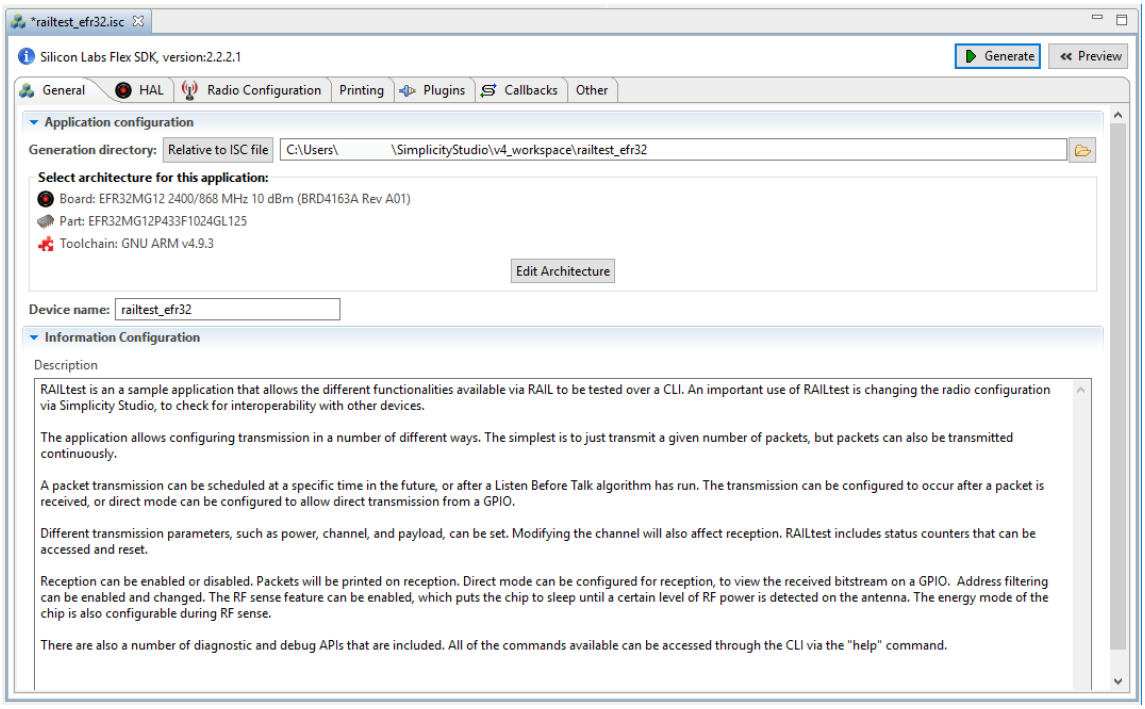
3. Name the component instance, then click [**Done**].



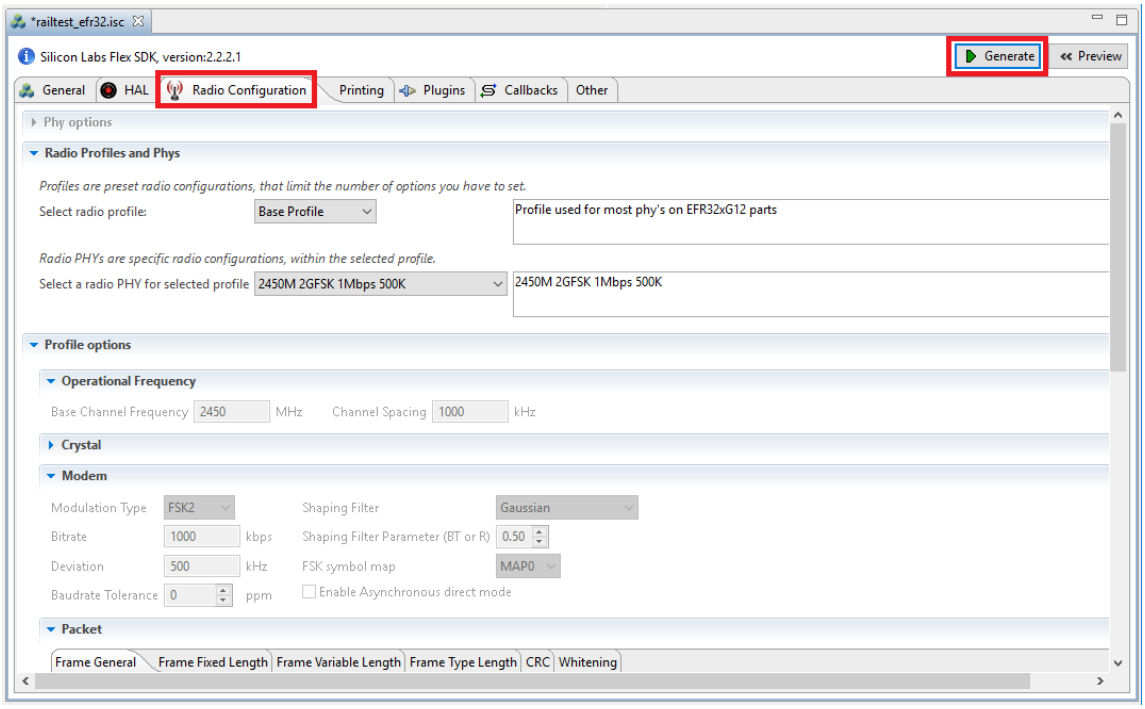
Project files autogenerate as you add, remove, or configure components.

1.2.2 Configure and Generate in Simplicity Studio 4

When you finish creating your sample application, an Application Builder General tab opens.

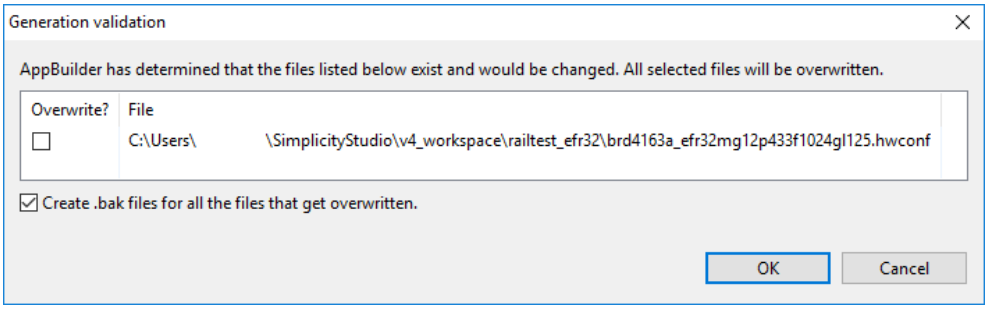


1. In the General Tab, if the architecture parameters shown for MCU and Radio and Board Type are not correct for your target device, click [**Edit Architecture**] to change the parameters. In general, the initial configuration settings for sample applications should be correct.
2. The RAIL application framework allows you to modify the PHY configuration for the application. Select the **Radio Configuration** tab to modify the PHY configuration for your application. For more information on how to set the modem parameters, please refer to *AN971: EFR32 Radio Configurator User's Guide*.

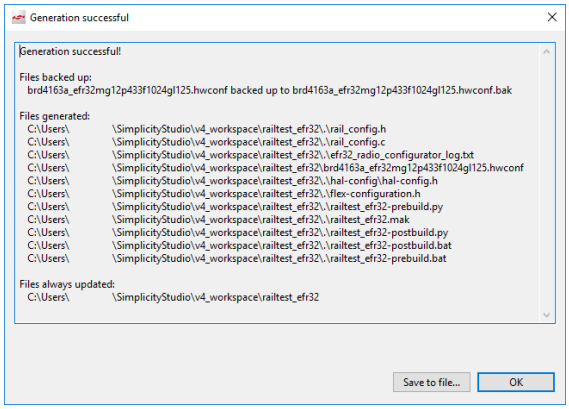


3. Click [**Generate**] in the upper right corner of the Application Builder window. Note that whenever the Build Configuration has been changed, you have to click [**Generate**] again.

4. Select if you want to overwrite any of the existing files. Click [**OK**].



5. The next dialog shows the generated files. Click [**OK**] to continue.

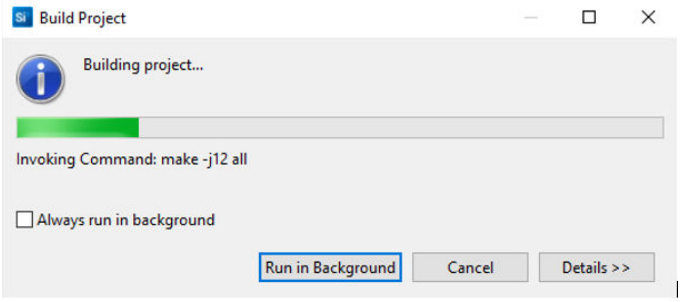


1.3 Build the Application

In Simplicity Studio 5, either right-click the .slcp file in the Project Explorer view and click **Build**, or, with the project open, click **Build** (hammer icon) in the toolbar.

In Simplicity Studio 4, double-click on app_main.c in the Project Explorer view to open it in the editor. Click **Build** (hammer icon) in the toolbar.

Wait until the build process is finished.



Verify that the project has built without any error in the Console view at the bottom of the Simplicity IDE.

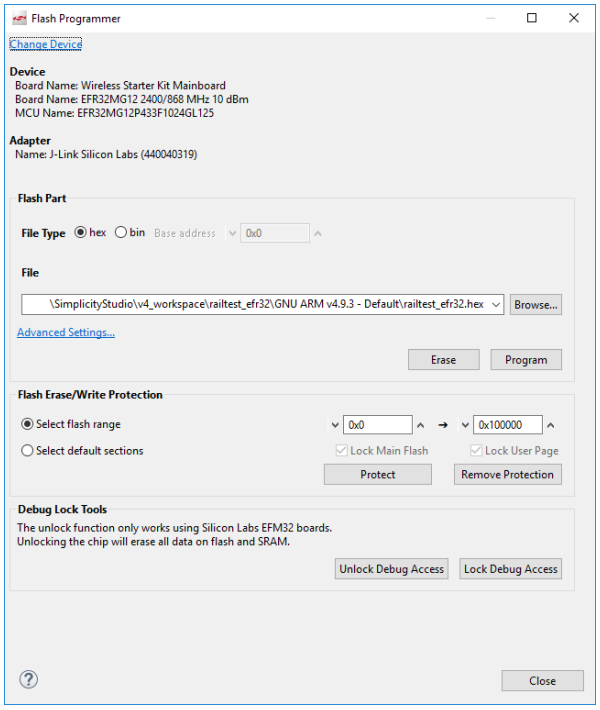
1.4 Load the Binary onto your Device/Flash Programming

If a full erase is not necessary, click **Debug** (bug icon) in the menu bar. This flashes the project into the board if it was successfully built. This will only update the program memory.

Once the project is flashed, the board can be started by clicking **Resume** in the menu bar.

Alternatively:

1. In the Debug Adapters perspective, select the target device.
2. With the device selected, open the Flash Programmer through the Launcher perspective's Compatible Tools tab or by clicking Flash Programmer (chip icon) in the Simplicity IDE menu bar.
3. The Flash Programmer opens. Select the file type.



4. Navigate to the .bin or .hex image you wish to upload.
5. Click [**Erase**], to make sure that any previous bootloader or other non-volatile data is erased before your new image is uploaded.
6. Click [**Program**] to program the flash.
7. You will be notified once the upload is complete.

After starting the demo, the screen shows three parameters: Rx Count, Tx Count, and Channel Number.

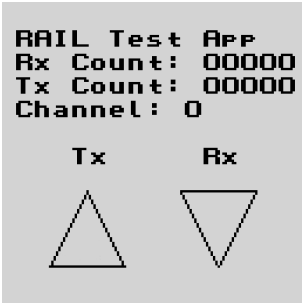


Figure 1.1. RAILtest Application Start Screen

1.5 Serial Port Communication

The RAILtest application should be accessed using a standard serial port terminal. The default version uses the WSTK's built-in serial port to provide access to this, so do not attempt to connect directly to the UART GPIOs.

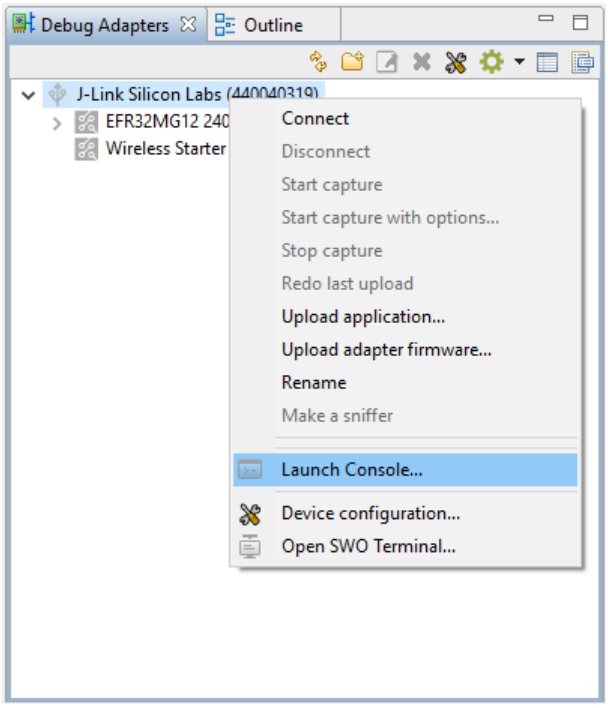
The serial port can be accessed using Ethernet or USB. Both provide the same level of functionality so it is completely up to you what to use.

- For USB, plug the device into your computer and find the Virtual COM port that is created. It will show up as a "JLink CDC UART Port" in the Device Manager. The Virtual COM port should work with any UART settings, but the physical UART is configured by default for 115200 8-N-1 if you want to match that. For UART-based communication, you can use any terminal emulator program that supports serial port communication, like TeraTerm or PuTTY.
- Simplicity Studio contains a built-in terminal as well. To run it, right-click on the device on the Debug Adapters view and click **Launch Console**. The RAILtest API should be accessible from the Serial 1 port.
- For Ethernet, use a telnet client to connect to port 4901 on the WSTK. You can use a program like PuTTY for this or the standard command line telnet client.

1.6 Console View

Once the RAILtest application is loaded onto the EFR32 you can interact with the RAILtest application using the Command Line Interface. The Console View will give you a CLI interface in Simplicity Studio's Network Analyzer Perspective so that you can interact directly with the RAILtest application.

1. Right-click the device in the Debug Adapters view.
2. Select **Connect** (if you are not already connected) and then **Launch Console**. To get a prompt on the Console Serial 1 tab, press [**Enter**].



2. Basics of the RAILtest Application

The RAILtest application provides a simple tool for testing the radio and the functionality of the RAIL library. For any advanced usage you must write your own software against the RAIL library and create a custom radio configuration. The RAILtest application is further described in *UG409: RAILtest User's Guide*.

Note: If you are going to use the WSTK as a mobile device to run the RAILtest application, it is recommended that you connect an external AA battery pack or a USB power bank to the WSTK board. The coin cell battery will not have enough power to do long-term testing using sample applications like RAILtest.

2.1 CLI

The most powerful way to interact with the sample application is using the provided command line interface. The setup instructions in earlier sections describes how to connect to this interface.

2.2 Command Input

The syntax for this interface is the standard command [arg0, arg1, ...] syntax, where the number and type of arguments depend on the specific command. Numeric values can be prefixed with 0x to indicate hexadecimal values. Note that the maximum number of arguments to any command is 9 and the maximum length of a command line is 255 characters.

Use the `help` command to see a full listing of available command options.

2.3 Command Output

All responses to commands are formatted in a human readable yet parsable format. There are two variations of this format: single- and multi-line. Both of these follow these rules.

- Start and end with curly braces { }.
- List the command name, enclosed in parentheses ( ).
- Contain any number of tag/value pairs enclosed in curly braces { }.
- Carriage returns and line feeds are treated as whitespace by any parser.

2.3.1 Single Response

Used when there is only a single response to a command.

- There is a single start/end curly brace wrapper.
- Tag/value pairs are wrapped in a single set of curly braces, separated by a colon {tag:value}.

Example:

```
> getchannel
{{(getchannel)}{channel:4}}
```

2.3.2 Multi Response

Used when a command may have multiple responses, for example when reading a block of memory or receiving multiple packets.

- Response starts with a header, delimited by a hash # at the start of the line.
- Header includes the command name, followed by any tags individually wrapped with curly braces { }.
- Following the header, any number of responses can be provided.
- Data lines do not contain the command name or tags, only the values that correspond to the tags in the order described in the header.

Example:

```
> getmemw 0x20000000 4
#{{(getmemw)}{address}{value}}
{{0x20000000}{0x0000e530}}
{{0x20000004}{0x000051c6}}
{{0x20000008}{0x0000c939}}
{{0x2000000c}{0x0000e090}}
```

3. Transmission Test

3.1 Transmitted Signal Types

The device is capable to generate continuous unmodulated signal (CW mode, also referred as single tone or sinus wave signal), packets (packet mode) and a continuous pseudo random stream (PN9 mode). The following table helps to decide which transmission mode should be used for a corresponding measurement.

Table 3.1. Transmission Test

	CW mode	PN9 mode	Packet mode
Tx Power	x	—	—
Power Spectral Density Mask	—	x	—
Phase Noise	x	—	—
Spurious Emission	x	—	—
Transient Power	—	—	x

The following sections describe how to turn these modes on, and how to modify some corresponding parameters to customize the output signal.

3.2 Output a Continuous Unmodulated Tone

CW mode is used to test the output power of the test card and observe the unmodulated carrier spectrum. In this mode, only the frequency and the output power are configurable parameters. There are two modes for CW generation. One is designed to generate CW signal [tone] with the best possible quality. The other is specifically designed to measure phase noise of the device [phaseNoise]. For more information on phase noise measurement, see section 3.7 Phase Noise Measurement. The difference between these two modes of CW signal generation is the PLL bandwidth settings. In the [phaseNoise] CW mode, the PLL bandwidth is the same as the current radio settings. With this method the generated CW signal provides information on how the PLL behaves when transmitting packets during operation.

Note: Before setting power and some other parameters, the device must be in idle receiver mode, by calling `rx 0`.

Set Receiver to Idle:

```
rx 0
{{(rx)}{Rx:Disabled}{Idle:Enabled}{Time:539309681}}
```

Enable CW Mode:

```
setTxStream 1 0    [enable] [streamMode] Enable(1 0) or Disable(0 0) a CW tone from the radio.
```

The command above also sets the PLL bandwidth to narrower than it's set on the actual radio configuration to produce better quality CW tone.

Enable CW Signal for Phase Noise Measurement:

```
setTxStream 1 3    [enable] Enable(1 3) or Disable(0 3) a tone from the radio with the same PLL bandwidth as
the current PHY configuration.
```

Set Tx Power:

The `setPower` command can be used to change the output power; enter the desired power in deci dBm.

Note: You can check the Tx power limitation with the `getPower` command. The radio state must be IDLE for Rx and Tx side. Call `rx 0` and `SetXxTone 0` first.

```
setPower       s    [power] Set the current transmit power in deci dBm
```

Set Channel:

A way to change the frequency is to change the channel. The specific channel configuration depends on the PHY configuration you have chosen for your test app. To switch between channels, use the `setChannel [num]` command. In addition, you can use button PB1 to cycle between channels. (The channel spacing value is used for relative frequency configuration to configure a frequency so many channel spacing away from the base channel frequency.)

To get the current radio channel, use this command:

```
getchannel    Get the current radio channel
```

To set another radio channel, use this command:

```
setchannel u    Set the current radio channel
```

Set Frequency:

To modify the frequency to a value not defined in the channel list, set the application into the FREQUENCY_OVERRIDE debug mode via `debugMode`, which tells the application to ignore the current channel selection. Once in the FREQUENCY_OVERRIDE debug mode, you can use the `freqOverride` command to switch to another center frequency.

Note: The `freqOverride` command requires you to be in FREQUENCY_OVERRIDE debug mode.

Note: The radio state must be IDLE in order for the channel to be changed or the frequency to be modified. Call `rx 0` first.

Example:

```
rx                0               [enable] Enable(1) or Disable(0) receive mode
setDebugMode      1               [mode] 1 = Frequency Override. 0 = Disable debug mode
freqOverride      2404000000      [freq] Change to freq specified in Hz. Requires debug mode to be enabled.
```

Set Different PHY Configuration:

To switch between PHYs in a multi PHY configuration, use the `setConfigIndex` command.

Example:

```
> setConfigIndex 0
{{(setConfigIndex)}{configIndex:0}{firstAvailableChannel:0}}
```

3.3 Output a Continuous Modulated PN9 Stream

In this mode, the modulation content is generated internally using a pseudo-random (PN9 sequence) bit generator. The primary purpose of this mode is to observe the modulated spectrum without having to provide data for the radio. This mode is usually used to test the power spectral density mask.

In this mode, only the frequency and the output power are configurable parameters. (To change these parameters, refer to 3.2 Output a Continuous Unmodulated Tone.)

Enable PN9 mode:

```
setTxStream       Enable/Disable a PN9 stream from the radio
```

3.4 Output Packets

The application starts in packet mode with the receiver enabled. In this mode we receive and transmit packets using the radio's frame controller hardware.

To transmit a specified number of packets, use this command:

```
tx       w       [n] Transmit n packets. If n is 0 transmit infinitely
```

Or press button PB0. If you hold PB0 for a couple of seconds or run the `tx 0` command you can toggle the continuous transmit mode.

In this mode, the frequency, the output power, delay in between each transmitted packet, the length, and the value of the bytes to send are configurable parameters. (To change the frequency and the output power parameters, please refer to 3.2 Output a Continuous Unmodulated Tone.)

When transmitting multiple packets or infinite packets there is a configurable delay in between each transmit. By default this is 250 ms, but it can be set with the following command:

```
setTxDelay       w       [delay] Set the inter-packet delay in milliseconds for repeated Tx
```

Get the inter-packet delay in milliseconds for repeated Tx, use the `getTxDelay` command.

If the desired delay is lower than you can set with setTxDelay, all the peripherals must be disabled to improve performance.

```
setPeripheralEnable       u       [enable] Turn LEDs and LCD on/off
```

The application by default sends a fixed packet, but it is possible to override the values via `setTxPayload`. The command allows you to modify the values of the payload at specific offsets. For instance to modify the first 4 bytes sent in the packet to be 0x01 0x02 0x03 0x04, this is the command:

```
setTxPayload 0 0x01 0x02 0x03 0x04
```

Note: To view the currently configured TX Packet information, use the following command:

```
printTxPacket       Print the current Tx data and length
```

A byte in the payload holds the length of the packet for variable length schemes. The length byte in IEEE 802.15.4 is the first byte; this is what you can set using the `setTxPayload` command. The length byte also covers the CRC length of the packet, and the length parameter covers the length byte; this is what you can set using the `setTxLength` command.

All in all, here is an example to set 3-byte payload to 0x01 0x02 0x03, while using IEEE 802.15.4 configuration:

```
setTxLength 4
setTxPayload 0 0x05 0x01 0x02 0x03
```

A spectrum analyzer is used to view the over the air sent packets of the radio. It must be connected through a proper RF cable to the RF connector of the radio board..

Figure 3.1. Analog FM Demodulation of the Sent Data

3.5 Tx Power Measurement

A very common way to measure the transmitted power of a device is to transmit a CW signal, then measure the transmitted signal with a spectrum analyzer or an RF power meter device. The measurements should be conducted, and proper RF cabling is necessary.

A spectrum analyzer is used to measure the transmit performances of the radio and must be connected through a proper RF cable to the RF connector of the radio board. In this example the transmitted power is measured in the CW signals 5MHz frequency range and the CW signal is marked with a marker (Mkr 3). The transmitted power is displayed on the right upper corner, where the marker is placed.

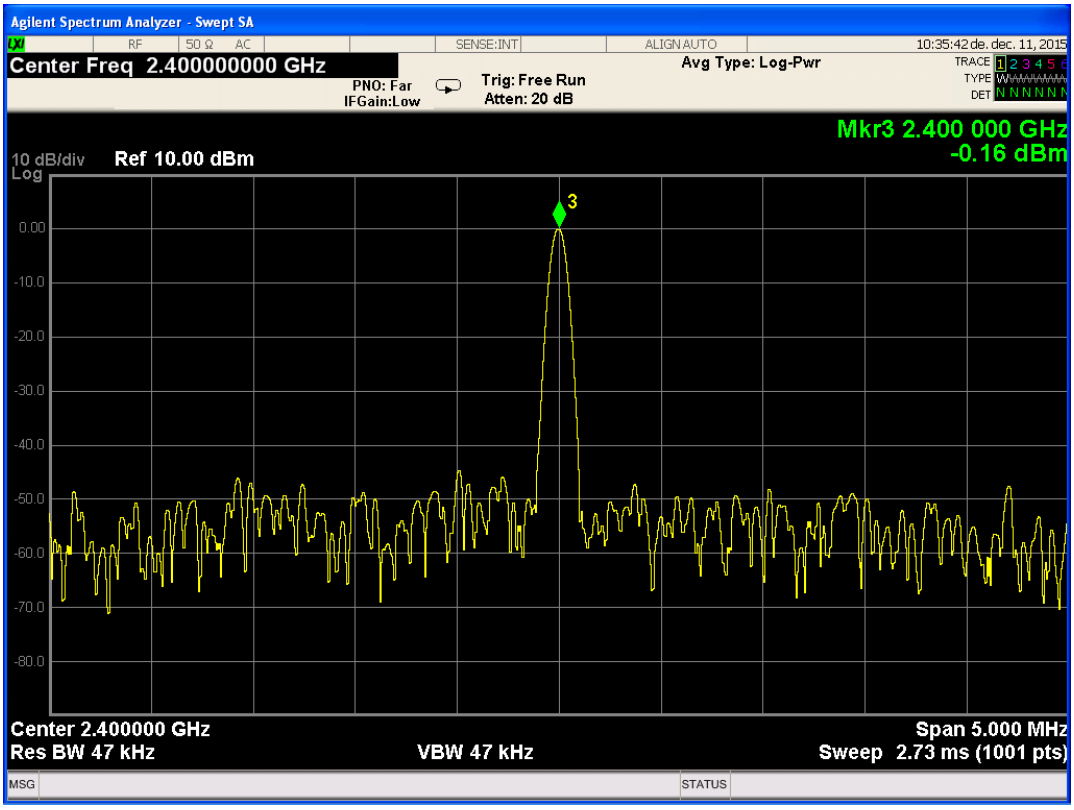


Figure 3.2. Unmodulated CW Signal Measured with Spectrum Analyzer

3.6 Power Spectral Density Mask Measurement

Power Spectral Density (PSD) gives feedback about the transmitted signal's spectrum usage. PSD is usually measured with a PN stream. However, Silicon Labs recommends measuring the device's PSD by enabling the PN9 stream. A spectrum analyzer is used to measure the power spectral density mask of the radio and must be connected through a proper RF cable to the RF connector of the radio board. RF cable loss should be compensated for accurate Tx power output level measurement. The proper reference level and span of the device is required as well.

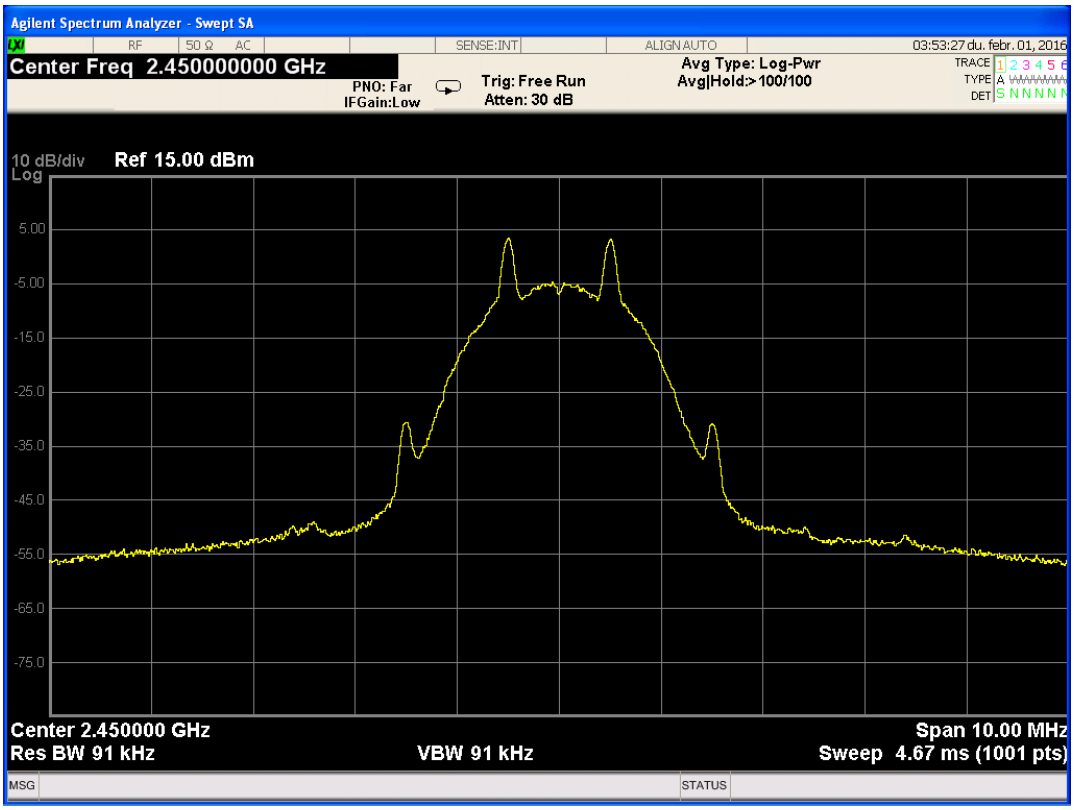


Figure 3.3. PN9 Modulated Signal (2GFSK, 1Mbps Data Rate, ± 500 kHz Deviation)

The figure above shows the demodulated signal when the radio board works in PN9 mode. The data rate is 1 Mbps; the deviation is 500 kHz, and the modulation type is 2GFSK.

3.7 Phase Noise Measurement

Phase noise (PN) is a frequency domain representation of a signal's jitter. Phase noise is usually measured when a reference signal's quality is specified. PN measurement gives a feedback on how the – ideally – stable reference signal or other CW signal's frequency changes in a short period of time. PN is usually represented as a frequency-amplitude graph or a notable point – from the graph – is presented in dBc/Hz.

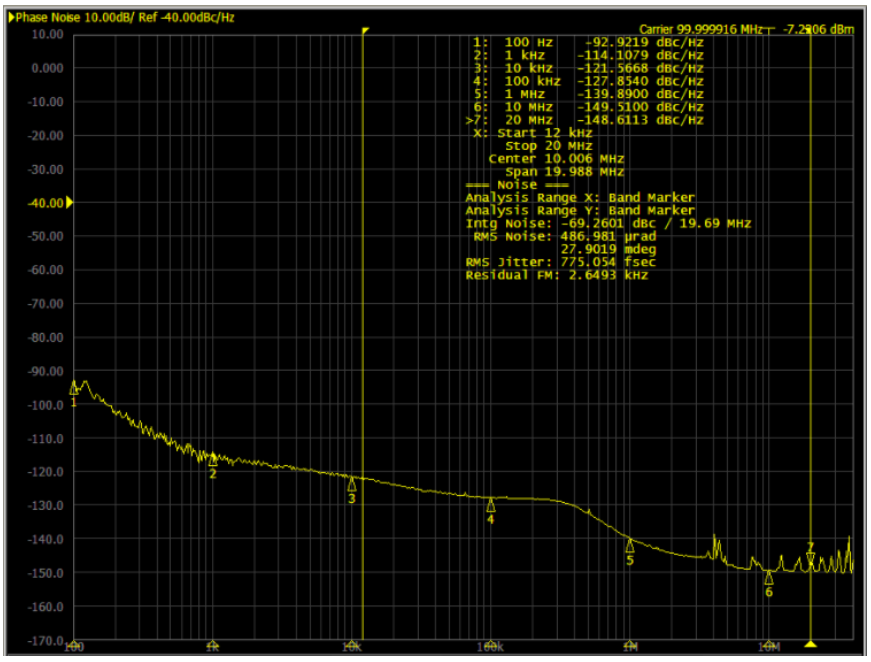


Figure 3.4. Phase Noise Measurement Example

Phase noise is measured with a spectrum analyzer, optimally one with a phase noise measurement feature. Proper connectors and cables are obligatory for this measurement. The PN graph usually contains only one side of the center frequency, as both sides are nearly identical. Frequency is usually represented logarithmically.

EFR32 devices' phase noise is usually measured, giving a feedback of the carrier's quality. If PN is measured on an EFR device, a CW signal is transmitted for measuring. PN not only effects the Tx performance but also the Rx side. The PN gives feedback on the oscillator quality, hence the transmit side and receiver side are affected as well. High PN on the transmitter side results in bad EVM and on the receiver side results in poor sensitivity, as PN is added to the noise level.

Note: Use RAILtest's special CW mode for phase noise measurement described in section 3.2 Output a Continuous Unmodulated Tone.

3.8 Spurious Emission Measurement

Spurious emissions are unwanted emissions at frequencies other than those of the transmitted channel. Spurious emission measurement is done by measuring the observed (transmitted) channel and its' neighbor while transmitting packets. A spectrum analyzer is used for measuring and at each frequency at which a spurious component is detected, the power level is measured and noted. The measured frequency bandwidth is typically2-10X as the occupied channel bandwidth. Some RF spectrum analyzers have a spurious emission measurements feature, to make performing these measurements easier. Spurious measurements are often taken while standard packets are transmitted, but some region's standard requires a different type of transmitted signal.

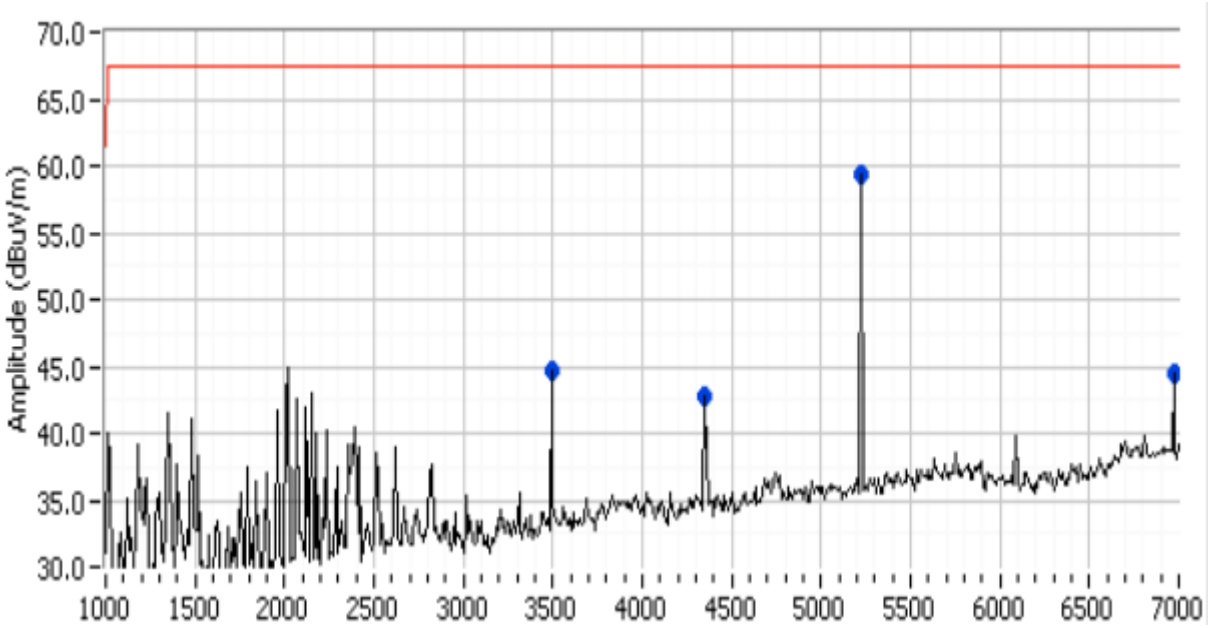


Figure 3.5. Spurious Emission Occurs as Spikes in the Measured Spectrum

3.9 Transient Power

Transient power is power appearing in frequencies other than the occupied channel when the transmitter is turned on or off. Transient power is expressed in dBm and measured at specific offset frequencies from the carrier. Standards usually describe offset frequencies with a maximum tolerated power of transient power.

Transient power can be measured with a spectrum analyzer, used in zero span mode. Zero span mode is used to measure a specific frequency. By changing the resolution bandwidth parameter of the analyzer, the measured frequency region can be set. A higher resolution bandwidth enables acquiring a wider frequency region, and allowing more power to the analyzer.

The EFR32 device is used in packet mode and sent at least 5 to 10 packets. The spectrum analyzer must be set to measure all of them with one measurement, by changing measurement / sweep time. After measurement the peak value must be recorded.

The following is an example of the measurement.

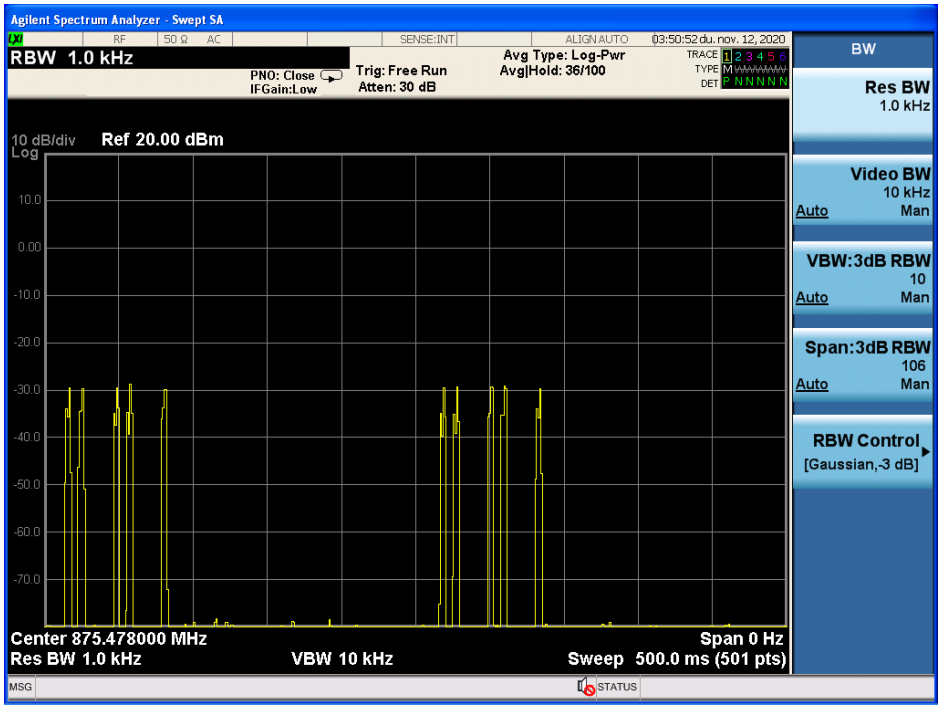


Figure 3.6. Transient Power Measurement 1

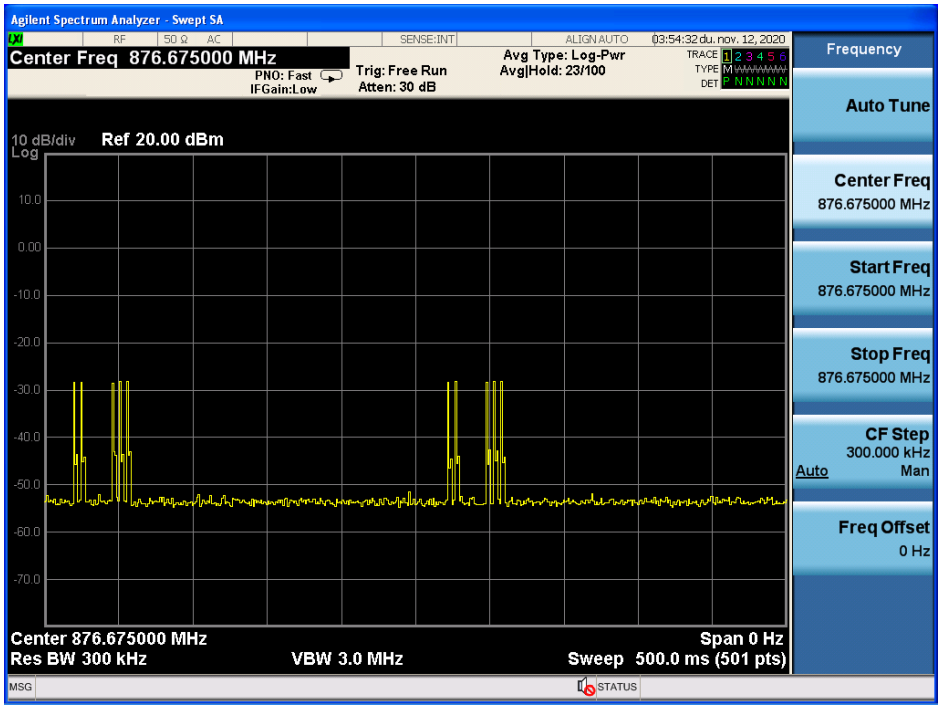


Figure 3.7. Transient Power Measurement 2

The first figure is a measurement at 78 kHz and the second is at 1275 kHz offset from the carrier frequency of the signal. The signal's power is 16 dBm. The measurement shows that at both settings the maximum power is around -28 to -30 dBm. It is important to take in account that the resolution bandwidth (RBW) is not the same at the two measurements and this should be compensated in some cases. In this case, the power density at the second measurement must be less than the first one, but a wider frequency range is measured. Larger RBW result in more power being received at the wider frequency range.

3.10 Enable Direct Mode

In direct mode, the radio will still attempt to decode received packets, but the data bit streams are input to and output from the chip in real-time on a physical I/O pin. This is often useful in legacy systems that do not use a conventional packet structure, or when the host MCU must perform customized packet handling or processing. In TX Direct mode, the TX modulation data is applied to an input pin of the chip and processed in "real time" (that is, not stored in a register for transmission at a later time).

The GPIOs for direct mode are fixed for now to the following pins.

```
DIN - EFR32_PC10 -> EXP_HEADER15/WSTK_P12
DOUT - EFR32_PC11 -> EXP_HEADER16/WSTK_P13
```

The data on these pins is asynchronous and can be connected directly to a UART. To enter direct mode issue the `directMode 1` command after starting the app. To leave direct mode use `directMode 0`. To transmit, enable the transmitter by issuing `directTx 1` and later stop it with `directTx 0`. Receive is controlled using the standard `rx 1/0` command, but is enabled by default when not transmitting.

In this mode, only the frequency and the output power are configurable parameters. To change these parameters, refer to 3.2 Output a Continuous Unmodulated Tone.

4. Reception Test

Packet Error Rate (PER) and Bit Error Rate (BER) are error ratios that could measure various reception tests

Table 4.1. Reception Test

	PER	BER
Rx sensitivity	triggered and non-triggered	x
Selectivity	x	x
Blocking	x	x
Intermodulation	x	x
Maximum input Power	x	x

EFR devices have a special mode to measure BER, with additional functions that can be used to calculate and measure BER easily. However, Silicon Labs does not recommend using this mode for RF evaluation, as regular frame detection does not occur in this mode. Instead, measuring PER with the normal operational mode turned on is recommended. Measuring PER gives more accurate feedback about how the EFR radio configuration perform, as it is testing in a mode that will be operational in the field after deployment.

4.1 PER Measurement

4.1.1 Basics of the Packet Error Rate

To measure PER, the accurate packet symbols must be known. PER [%] is calculated by the following:

$$PER = \frac{P_{Error}}{P_{Sent}} * 100$$

where

P_{error} = the number of packets not received correctly

P_{sent} = the number of packets sent

and the result is displayed as a percentage.

PER measurement requires an accurate device that is able to send the packets the receiver under test (RUT) expects to receive. An RF vector signal generator is a good fit for these measurements.

Typical PER measurement procedure using RAILtest:

1. Record the packet that should be transmitted to the RUT (receiver under test) and load it to the RF signal generator.
2. Connect the RF signal generator and the RUT with the proper RF cable.
3. Send 100 to 1000 packets to the RUT. (P_{sent})
4. View the received number of packets by using *status* (RAILtest) command for example. The “RxCount” value gives the successfully received number of packets. (P_{sent} - packets received = P_{error})
5. Calculate PER.

Using the `status` command, *RxCount* is the number of successfully received packets. *SyncDetect* is the number of sync word detection and *FrameErrors* is the number of CRC errors acquired.

4.1.2 Record a Transmitted Packet

To know how a packet is assembled on an EFR32 device, it is practical to record a transmitted packet. A packet can be captured and demodulated with an RF vector spectrum analyzer from the EFR32 device's RF port, or the transmitted symbols could be wired to a GPIO pin before being sent to the modulator. The second option offers an easy way to see the transmitted baseband packet symbols before the modulation and upconversion to RF frequencies.

To port the transmitted packet symbols to the GPIO port, the PRS channels should be configured by using the *setDebugSignal* command. Use the *setDebugSignal help* command to learn more about the function. For additional information about PRS channels, see the Silicon Labs knowledge base about *RAIL Tutorial: Debugging* or the EFR32 page of the RAIL API documentation.

For example (using EFRxG13):

setDebugSignal PC11 TXACTIVE -> PC11 GPIO pin active when TX is active

setdebugsignal PB2 CUSTOM_PRS 0x2a 0x5 -> Data Clock is routed to PB2 GPIO

setdebugsignal PB3 CUSTOM_PRS 0x2a 0x6 -> Data (Tx packet) is routed PB2 GPIO

During transmission, the data clock and data is recorded on the respective GPIO pins:

4.1.3 Non-Triggered PER

By default the application starts in packet mode with the receiver enabled. In this mode we receive and transmit packets using the radio's frame controller hardware. If a packet is received, the payload content and length, the RSSI, the CRC status, and the time can be read from the receiver response.

Example:

```
{{(rxPacket)}{len:16}{timeUs:290402914}{timePos:4}{crc:Pass}{rssi:-50}
{lqi:240}{phy:0}{isAck:False}{syncWordId:0}{antenna:0}{channelHopIdx:254}
{payload: 0x0f 0x0e 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xaa 0xbb
0xcc 0xdd 0xee }}
```

This test also involves an RF signal generator, which can transmit predefined packets. In this mode, the radio is set to receive and the generator must send packets. If the radio does not receive the packet, there is no RX response. To view the number of received packets, CRC errors, and sync detect errors, use the `status` command.

Example:

```
{{(status)}{UserTxCount:38}{AckTxCount:0}{UserTxAborted:0}{AckTxAborted:0}{UserTxBlocked:0}
{AckTxBlocked:0}{UserTxUnderflow:0}{AckTxUnderflow:0}{RxCount:0}{SyncDetect:8}{NoRxBuffer:0}{RfSensed:0}
{ackTimeout:0}{ackTxFpSet:0}{ackTxFpFail:0}{ackTxFpAddrFail:0}{RfState:Rx}{RAIL_state_active:0}
{RAIL_state_rx:1}{RAIL_state_tx:0}{Channel:0}{AppMode:None}{TimingLost:0}{TimingDetect:0}{FrameErrors:8}
{RxFifoFull:0}{RxOverflow:0}{AddrFilt:0}{Aborted:0}{RxBeams:0}{DataRequests:0}{Calibrations:1}
{TxChannelBusy:0}{TxClear:0}{TxCca:0}{TxRetry:0}{UserTxStarted:38}}
```

To reset all the counters use:

```
resetCounters       Resets the Tx and Rx counters
```

If a long and fast test needs to be run, it is better to disable the Rx responses, to make the communication and the test faster.

```
setNotifications setRxNotification                    [enable] Enable(1) or Disable(0) status prints
that happen asynchronously (rxPacket, txEnd, txError)
```

4.1.4 Triggered PER

RAILtest can be used to determine the packet error rate (PER) for a given setup. The Wireless Gecko is capable of triggering a connected RF signal generator to transmit a specific number of packets with a configured trigger interval, counting the correctly received packets, and calculating the actual PER value.

To set up this test, a piece of test equipment needs to be configured to send a packet on the rising edge of a GPIO. That should be connected to PC7 on the EFR32, or PC3 on the WSTK. `perRx 100 10000` will configure the Packet Error Rate test to send 100 packets, waiting 10000 µs between each packet. At the end of the test, the app will give an output indicating that PER mode has finished, and the statistics on the test can be recovered with `perStatus` and `status`. Note that calling `perRx 0 0` will cancel an ongoing test, and that calling `perRx` will have the same effect as calling `resetCounters`.

By adjusting the output power of the generator, the sensitivity of the radio can be determined. It is usually defined for a 1% or 20% packet error rate.

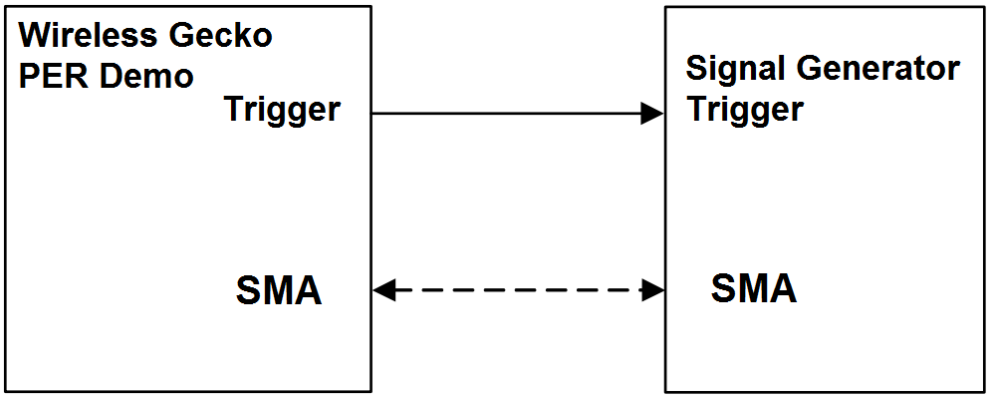


Figure 4.1. Packet Error Rate Measurement Setup

4.2 BER Measurement

4.2.1 Running Bit Error Test from the CLI

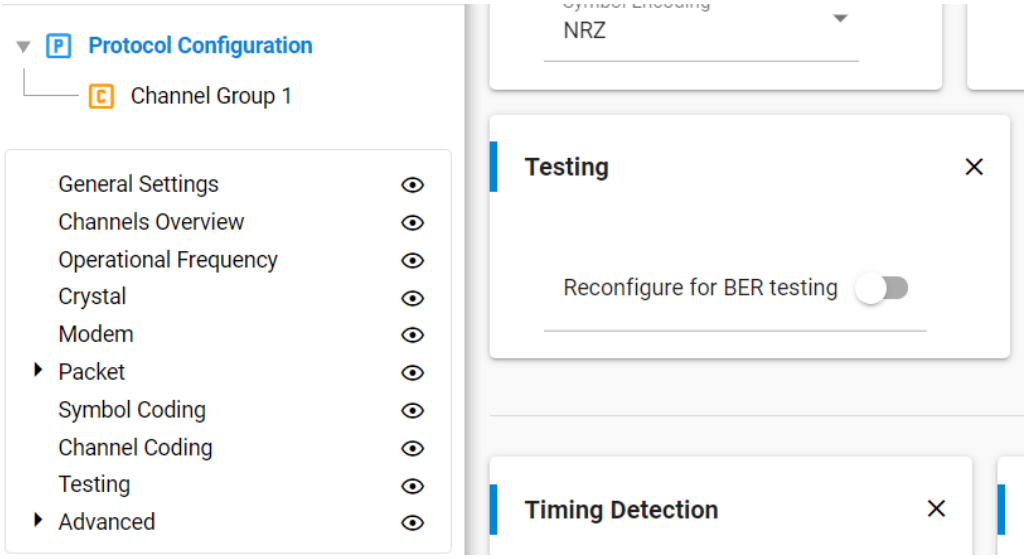
The sensitivity of the radio can be measured with a continuously-looped PN sequence (x^9 + x^5 + 1). This mode is called bit error rate (BER). The EFR32 hardware has the ability to enter BER receive mode for diagnostic purposes. The radio is set into continuous receive mode. The RF data needs to be fed to the RX SMA of the test card or radio board. By adjusting the output power of the signal generator or an EFR32, the sensitivity of the radio can be measured (it is typically measured for 10^-3 BER). See *UG409: RAILtest User's Guide* for further information about BER testing mode.

4.2.2 BER Measurement Procedure

To run the BER test successfully, a radio configuration specific to BER mode must be generated by the radio configurator. To enable BER testing mode by default on reset or startup:

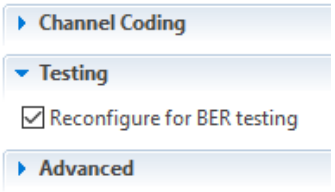
In Simplicity Studio 5:

In the Radio Configurator, on the General Settings card enable **Customized**, and then on the Testing card enable **Reconfigure for BER testing**.



In Simplicity Studio 4:

Check **Reconfigure for BER testing** on the Radio Configuration tab.



4.3 Sensitivity Measurement

Sensitivity of a device is the minimum level of received RF power that the device is capable of receiving. Sensitivity is often expressed in dBm. It is usually defined for a 1% or 20% packet error rate, which means that at the sensitivity level (in dBm) the receiver is capable to produce 1% or 20% PER. By adjusting the output power of the generator, the sensitivity of the radio can be determined.

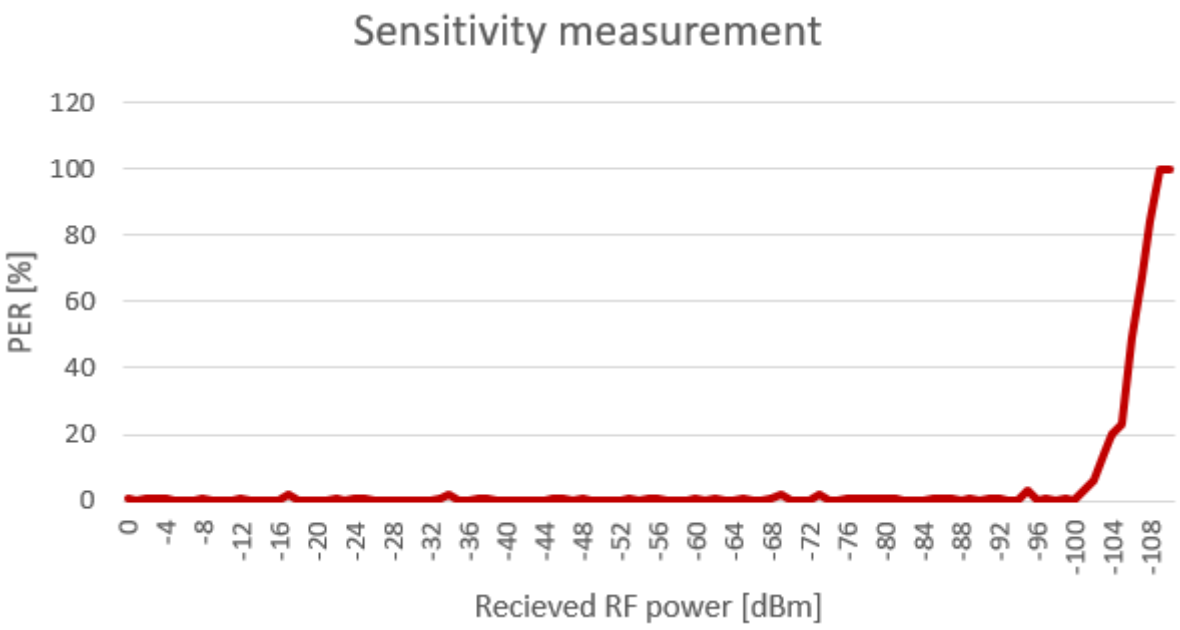


Figure 4.2. Sensitivity Measurement

From this graph it can be determined that the sensitivity of a device at 20% PER is -104 dBm. In practice the sensitivity of a device depends on the radio configuration, such as modulation, deviance, and frequency.

To measure the sensitivity of a receiver, PER must be measured at various receiver power levels, in practice between 0 to -120 dBm values with 1 to 5 dB steps.

The measurement results are more accurate if the RF signal generators power steps finer and more packets are sent at each power step. In the above example, the PF power step is 1 dB and 100 packets were sent at each frequency step.

4.4 Other Receiver Measurements

The measurement specifications may differ based on the standards for different regions, and are described in more detail in other documents.

Selectivity

Selectivity is a measure of the capability of the receiver to operate correctly while an unwanted signal is present at an adjacent channel.

Measurement procedure: Two generators are used for this measurement. One is generating packets on the operating channel at the sensitivity level of the receiver +3 dB. The second generator is generating an unmodulated signal on the nearest adjacent channel. The maximum power of the second generator at which the RUT (Receiver Under Test) is not able to receive the packets of the first generator is the selectivity, expressed in dBm.

Blocking

Blocking is a measure of the capability of the receiver to operate correctly while an unwanted signal is present at frequencies other than at an adjacent channel or spurious responses are.

Measurement procedure: Two generators are used for this measurement. One generates packets on the operating channel at the sensitivity level of the receiver +3dB. The other generates an unmodulated signal on a specified frequency offset, based on the local region's specification. The maximum power of the second generator at which the receiver under test is not able to receive the packets of the first generator is the blocking level, expressed in dBm.

Intermodulation

The intermodulation response is a measure of the ability of the receiver to receive a wanted modulated signal, without exceeding a given degradation due to the presence of two or more unwanted signals with a specific frequency relationship to the wanted signal frequency.

Maximum input power

Maximum input power is a measure of the maximum power at which the receiver is able to receive the packets without degradation of the signal quality.

4.5 RSSI Curve

Received signal strength indicator (RSSI) is an estimate of the signal strength in the channel to which the receiver is tuned. The RSSI value can be read with 0.25 dB resolution per bit. The RSSI may be read at any time while the radio is in Receive mode. The RSSI is not latched, but continuously updated while in Receive mode.

To read the RSSI value, use the following command:

```
getRssi       Get RSSI in dBm if the receiver is turned on.
```

Example:

```
> getRssi
getRssi
{{(getRssi)}{rssi:-94}}

> >
```

Note that a systematic offset (see figure below) will appear in the RSSI value returned by the RAILtest command due to matching network, radio configuration, and so on. You must profile the board and account for the offset when using the returned value.

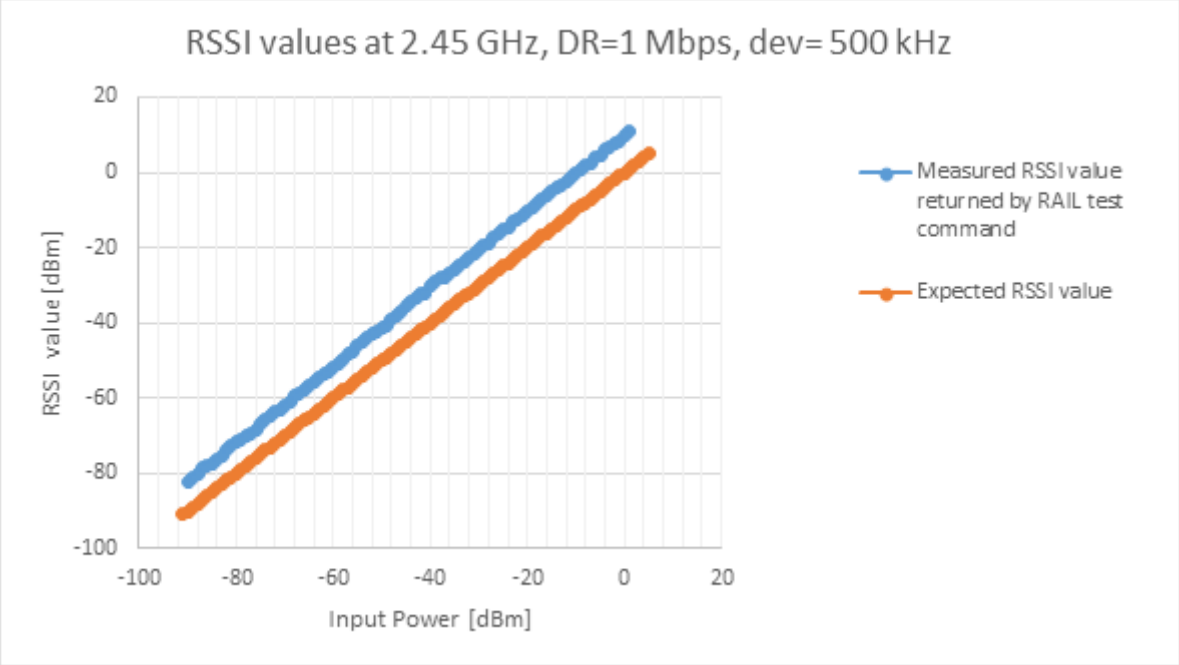


Figure 4.3. RSSI Curve

4.5.1 Customer Production Test: Transmission Power

The high resolution RSSI enables accurate channel power comparison measurement that can be useful for production tests transmission power without an instrument.

Measurement procedure:

1. Load the RAILtest application on two EFR32 nodes.
2. Set both nodes to the same channel with the `setchannel n` command, where "n" is the channel number.
3. Issue the `setTxTone` command on the first node to set CW mode.
4. Read the RSSI level on the second node with the `getRssi` command.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

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