

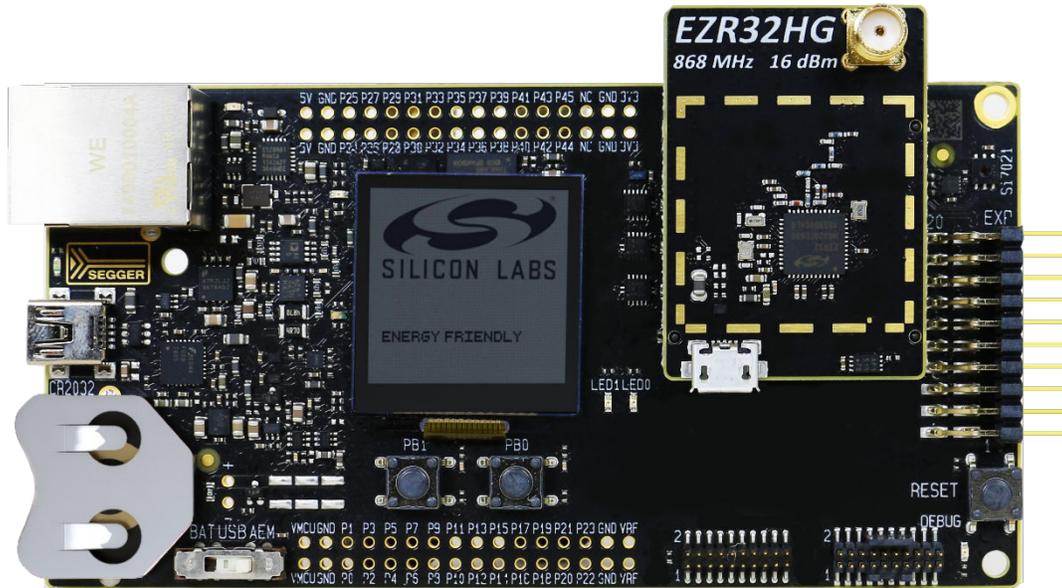
AN1004: RF Evaluation with EZR32



This document describes using the EZR32 sample codes for RF evaluation tests on EZR32 kits and on custom hardware equipped with EZR32.

KIT FEATURES

- Transmit tests with unmodulated carrier
- Transmit test with PN9 pseudorandom modulated signal
- BER measurement
- PER measurement



1. Using an Example

1.1 Opening an Example

All of the RF evaluation measurements start with opening an application.

Start Simplicity Studio™ and connect your board. You should see it under “Detected hardware”. Click on the Software Examples tile:

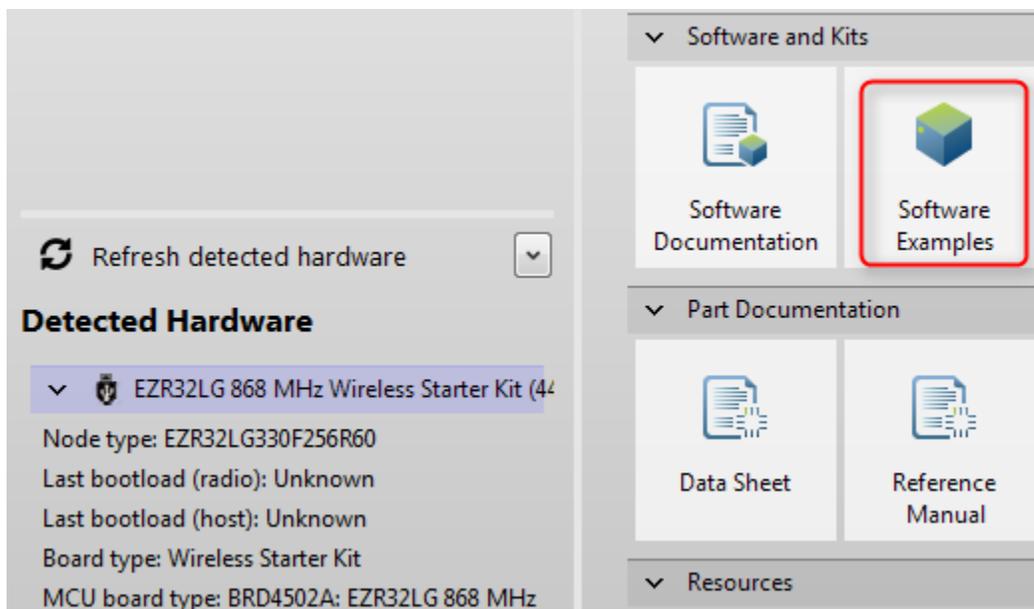


Figure 1.1. Software Examples

On the next screen, you can modify your kit, your MCU type, and the SDK, which should always be EFM32 SDK. If you detected your kit, it should automatically fill in the "Kit" and the "Part" parameters. If you want to test a custom hardware, see chapter 5. [Testing with Custom Board](#).

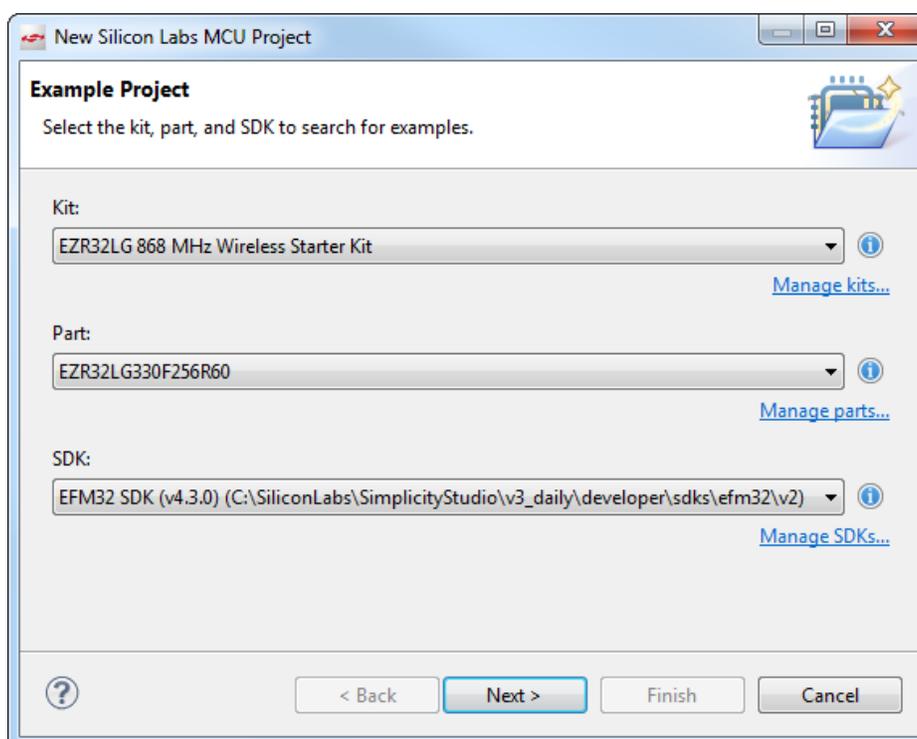


Figure 1.2. Example Project Hardware Configuration

On the next screen, you will see the available examples.

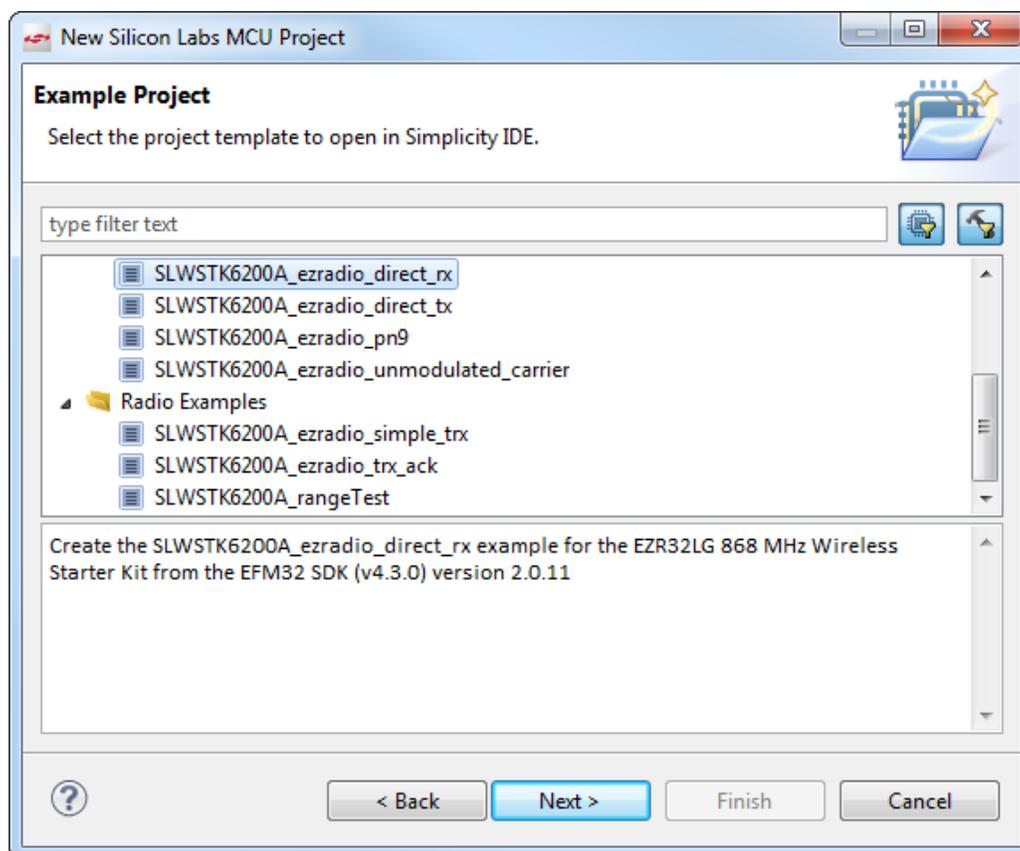


Figure 1.3. Example List

The important examples for RF performance measurement are:

- Direct RX, which is the base of BER measurement
- PN9, which implements a PN9 modulated transmission
- Unmodulated carrier, which transmits unmodulated carrier
- RangeTest, which is usable for PER measurements

Select any of these examples and click next. In the next windows, you can select the application name and the compiler toolchain. Generally, you don't have to modify anything there.

After Simplicity Studio generates the project, it switches to the Development perspective and opens the main.c automatically.

1.2 Modifying the Radio Configuration

All EZR32 projects provide a GUI-based radio configuration option, where you can set up the radio. The radio configurator is located in the RadioConfig folder and is called "radio-configurator_SLWSTK62xxA.isc".

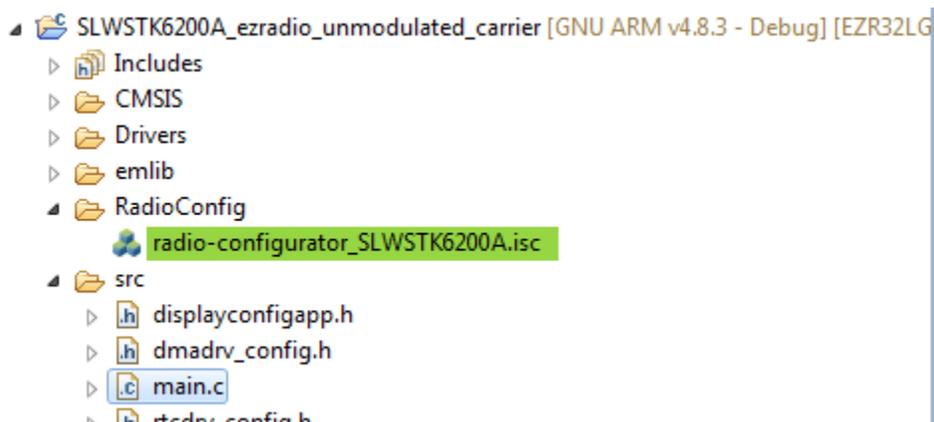


Figure 1.4. Radio Configurator ISC

The radio configurator is built up from sections. Some of the important sections for RF evaluation are:

- Frequency: The carrier frequency can be modified here
- Crystal: You can set the crystal parameters and the load capacitance here
- RF parameters: The modulation parameters can be modified here (PN9, DirectRX, and RangeTest only)
- Packet configuration: You can set the packet configuration and the per-field modulation settings here (RangeTest only)
- GPIOs: You can set the behavior of the radio's four GPIOs

After you set up the configuration, click the Generate button in the upper right corner to modify the project.

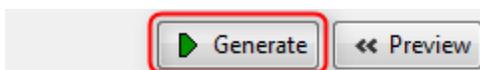


Figure 1.5. Generate Button

It will show a notification window with the generated files (if this is not the first time you generate headers, it will ask permission to overwrite files).

The most important generated file is **radio-config-wds-gen.h**, which holds the actual configuration for the radio.

1.3 Uploading the project

The easiest way to upload your application is to start debugging it. To do so, click on the Debug button. This will also build your application if it is not built yet.

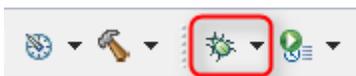


Figure 1.6. Debug Button

After successfully uploading your application, Studio will automatically switch to the Debug perspective, suspended before the first line of the main function. To start the program, click on the Resume button.



Figure 1.7. Resume Button

To exit from debug mode, click on the Disconnect button.



Figure 1.8. Disconnect Button

2. Transmit Tests

An unmodulated carrier can be used to measure output power, phase noise, and spurious emission (harmonics). PN9 is useful for measuring spectrum emission.

On the software side, these tests are pretty similar.

Open the Unmodulated carrier or PN9 sample application and open its radio configurator.

You probably want to configure the carrier frequency and the crystal parameters under the "Frequency" and "Crystal" sections, respectively:

The screenshot shows two sections of a radio configurator. The first section, titled "Frequency", contains a formula: (Base frequency: 868000000 Hz) + (Channel spacing: 250000 Hz) = Center frequency: 868000000 Hz. Below this is a link: << Previous: Radio profiles. The second section, titled "Crystal", contains: Crystal frequency: 26000000 Hz, Crystal tolerance TX: 20 ppm, Crystal cap. bank: 69, Crystal tolerance RX: 20 ppm, and a checkbox for "Use external TCXO/Ref source" which is currently unchecked.

Figure 2.1. Frequency and Crystal Configuration

Set up the PA level and mode under the "Power amplifier (PA)" section:

The screenshot shows the "Power amplifier (PA)" section. It includes: PA mode: Switching-Amplifier Mode (for Class-E or Square Wave match), PA bias: 12, Ramp HV cascade: 29, PA power level: 127, Enable ramp control of External PA (checkbox), Regulator voltage: 15, and Ramping time: 14.

Figure 2.2. Power Amplifier Configuration

For a PN9 sequence, set up the modulation type, data rate, and deviation under the "RF parameters" section.

The screenshot shows the "RF parameters" section. It includes: Modulation mode: PN9 pseudo-random, Modulation type: 2FSK, Data rate: 10,000 kbps (=ksp), and Deviation: 20,000 kHz.

Figure 2.3. Modulation Configuration

Once you set up everything, generate the configuration and upload the project (see the previous chapter for details).

You should see the following on the WSTK:

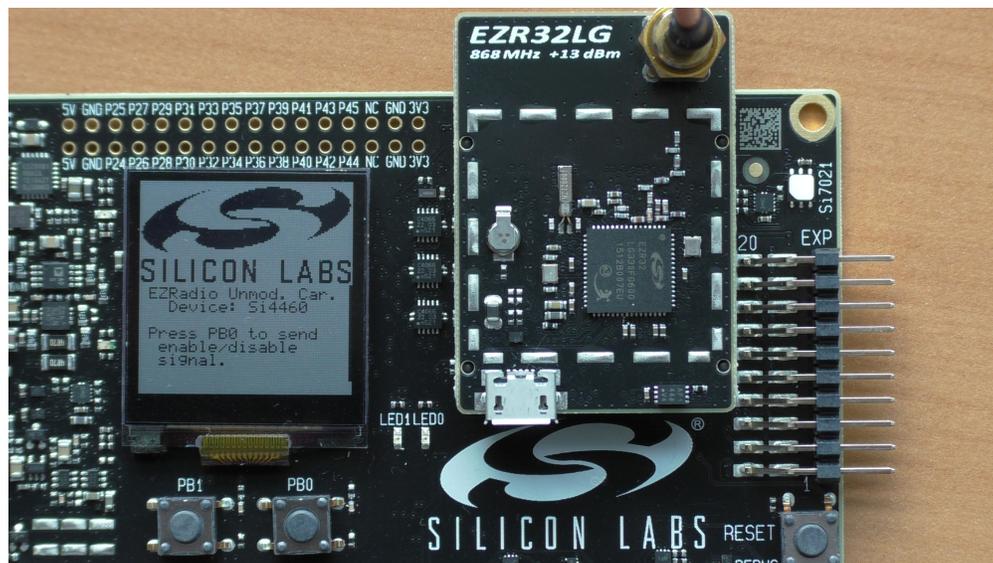


Figure 2.4. Unmodulated Carrier Example Running on a WSTK

As it says on the display, press PB0 to enable/disable the signal. When the signal is enabled, you should see “CW/PN9 signal: ON” on the bottom of the display.

3. BER Measurement with DirectRX

3.1 Set up the Configuration

The BER test can be used to establish baseline sensitivity, selectivity, and blocking performance of the receiver.

Note: Frequency offset and DR offset parameters cannot be measured in BER test mode; for those parameters, run PER tests.

Open the DirectRX example, and open its radio configurator. Set up the RF parameters for the frequency and crystal configurations as described in the previous chapter. Make sure to check "Enable BER mode". This mode creates a configuration for testing BER performance on a PN9 test sequence.

The screenshot shows the 'RF parameters' section of the DirectRX configurator. On the left, various parameters are set: Modulation mode (Direct), Modulation type (2FSK), Data rate (10,000 kbps), Deviation (20,000 kHz), RX bandwidth (150,000 kHz), and IF mode (Fixed). On the right, there are three sections: 'RX current' with 'High RX performance' selected, 'RX performance boost' with 'Disabled' selected, and a list of checkboxes including 'Enable BER mode' which is checked and highlighted with a red box. Other checkboxes include 'Enable PLL AFC', 'Enable adaptive Ch. Fil. BW', 'Enable antenna diversity', 'Enable IQ calibration', and 'Enable high performance Ch. Fil.'.

Figure 3.1. BER Mode Setting

For BER testing, you want to wire out the demodulated signal to a pin. This can be done under the "GPIOs" section:

The screenshot shows the 'GPIOs' section of the DirectRX configurator. It lists four GPIO pins with their configurations and pullup options:

- GPIO 0: Outputs the RX Data CLK signal. This signal is nominally a... Enable GPIO0 pullup
- GPIO 1: Outputs the demodulated RX Data stream, after... Enable GPIO1 pullup
- GPIO 2: Behavior of this pin is not modified. Enable GPIO2 pullup
- GPIO 3: Behavior of this pin is not modified. Enable GPIO3 pullup

Figure 3.2. GPIO Configuration for BER Measurement

Note that you cannot use GPIO on **all** of the boards. On boards with switched matching, GPIO2 and GPIO3 are used to control the RF switch.

GPIO0 and GPIO1 are not directly connected to a pin on the EZR32LG and EZR32WG parts. Using these pins is possible, but it will require a small code modification described in chapter 3.2 [Using GPIO0 and GPIO1 on EZR32LG or EZR32WG](#).

After you set up everything, generate the configuration.

3.2 Using GPIO0 and GPIO1 on EZR32LG or EZR32WG

On EZR32LG and EZR32WG, GPIO0 and GPIO1 are not wired directly to the pin. To use them with an instrument, you will have to set up PRS channels in the EZRadio driver. To enable it, add the following code to the end of the function `GpioSetup` in `main.c`:

```

/* Setup PRS for PTI pins */
CMU_ClockEnable(cmuClock_PRS, true);

/* Configure RF_GPIO0 and RF_GPIO1 to inputs. */
GPIO_PinModeSet((GPIO_Port_TypeDef)RF_GPIO0_PORT, RF_GPIO0_PIN, gpioModeInput, 0);
GPIO_PinModeSet((GPIO_Port_TypeDef)RF_GPIO1_PORT, RF_GPIO1_PIN, gpioModeInput, 0);

/* Pin PA0 and PA1 output the GPIO0 and GPIO1 via PRS to PTI */
GPIO_PinModeSet(gpioPortA, 0, gpioModePushPull, 0);
GPIO_PinModeSet(gpioPortA, 1, gpioModePushPull, 0);

/* Disable INT for PRS channels */
GPIO_IntConfig((GPIO_Port_TypeDef)RF_GPIO0_PORT, RF_GPIO0_PIN, false, false, false);
GPIO_IntConfig((GPIO_Port_TypeDef)RF_GPIO1_PORT, RF_GPIO1_PIN, false, false, false);

/* Setup PRS for RF GPIO pins */
PRS_SourceAsyncSignalSet(0, PRS_CH_CTRL_SOURCESEL_GPIOH, PRS_CH_CTRL_SIGSEL_GPIOPIN15);
PRS_SourceAsyncSignalSet(1, PRS_CH_CTRL_SOURCESEL_GPIOH, PRS_CH_CTRL_SIGSEL_GPIOPIN14);
PRS->ROUTE = (PRS_ROUTE_CH0PEN | PRS_ROUTE_CH1PEN);

/* Make sure PRS sensing is enabled (should be by default) */
GPIO_InputSenseSet(GPIO_INSENSE_PRS, GPIO_INSENSE_PRS);

```



Figure 3.3. Enabling PRS Channels

3.3 Using the Sample Application for Measurements

Once you set up everything, upload the configured sample application. Use the following table to locate the GPIOs where you can access the demodulated data stream:

Table 3.1. Radio GPIO Locations

Radio GPIO	MCU pin EZR32WG/EZR32LG	MCU pin EZR32HG	WSTK pin
GPIO0	PA0/pin10 (through PRS)	GPIO0/pin25	P30
GPIO1	PA1/pin11 (through PRS)	GPIO1/pin26	P31
GPIO2	GPIO2/pin63	GPIO2/pin15	P32
GPIO3	GPIO3/pin64	GPIO3/pin16	P33

After you have uploaded the application, you should see something like this:

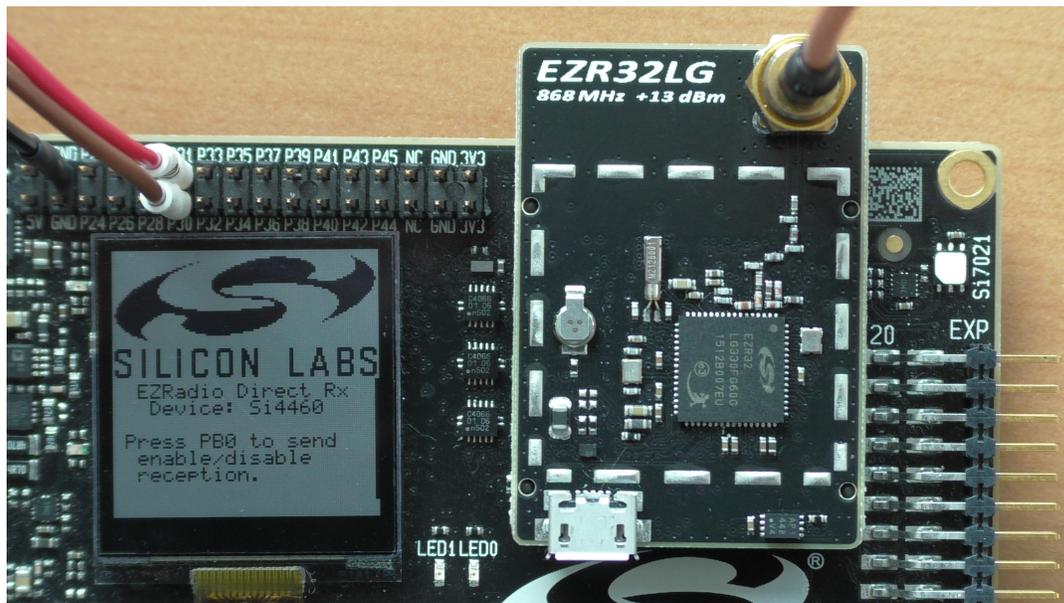


Figure 3.4. DirectRX Example Running on a WSTK

As it says on the display, press PB0 to enable/disable reception. When reception is enabled, you should see “Direct receive is ON” on the bottom of the display.

4. Using RangeTest for PER Measurement

The PER (packet error rate) measurement is useful for determining the RF performance in an application.

Note: RangeTest is not yet available for parts equipped with Si4467 or Si4468 radios.

Open the RangeTest example, and then open its radio configurator. Set up the frequency and crystal configurations similarly to the PN9 test. When setting up the RF parameters, note that you have many more options than you had in DirectRX mode, since this is a packet-based configuration.

The screenshot shows the 'RF parameters' configuration window. It includes the following settings:

- Modulation mode:** Packet
- Modulation type:** 2FSK
- Data rate:** 9,600 kbps (=kps)
- Deviation:** 4,800 kHz
- Enable manual RX bandwidth
- RX bandwidth:** 150,000 kHz
- RX data rate error:** 0% - 1%
- Deviation error + :** 0 %
- Deviation error - :** 0 %
- 4(G)FSK map(-3Δf, -1Δf, 1Δf, 3Δf):** 00 01 11 10
- TX Gaussian B*T:** 0,50 factor
- IF mode:** Scaled
- RSSI average:** The RSSI value is updated at 1*Tb bit period intervals but...
- RSSI latch:** Latch is disabled. The returned value of the Latched RSSI...
- RX current:**
 - High RX performance
 - Low current consumption
- RX performance boost:**
 - Disabled
 - Improved blocking
 - Improved selectivity
- Enable PLL AFC
- Enable BER mode
- Enable adaptive Ch. Fil. BW.
- Enable antenna diversity
- Enable IQ calibration
- Enable high performance Ch. Fil.
- OSR tune:** 0
- RSSI threshold:** 100
- Check threshold at latch

Figure 4.1. RF Configuration

Next, set up the packet configuration, according to your requirements:

The screenshot shows the 'Packet configuration' window with the following settings:

- General packet configuration:**
 - Enable preamble
 - Split TX and RX configuration
 - Packet TX threshold: 48 byte(s)
 - Packet RX threshold: 48 byte(s)
 - Number of fields: 1
 - Number of RX fields: 1
 - Skip sync word for TX
- Variable length configuration:** (tab selected)
- CRC configuration:** (tab selected)
- Data whitening configuration:** (tab selected)
- General field configuration:** (tab selected)
- Preamble configuration:**
 - Preamble TX length: 8 byte(s)
 - Preamble RX threshold: 20 bit(s)
 - Allowed bit error: 0 bit(s)
 - Use Manchester encoding
 - Manchester on Constant 1's or 0's
 - Preamble timeout: 15 nibble(s)
 - Preamble extended timeout: 0 * 15 nibbles
- Preamble pattern:**
 - Length: 0 bit(s)
 - Preamble value: 00 00 00 00
 - Preamble type: Std. 1010 pattern (>= 40 bits)
 - Standard 1010
 - Standard 0101
 - Non standard

Figure 4.2. Packet Configuration

Then generate the configuration and upload the sample application to your board.

On the usage of the RangeTest application, refer to chapters 2, 3 and 4 of the user guide, "UG147: Flex Gecko 2.4 GHz, 20 dBm Range Test Demo User's Guide". That application is basically the same as the one described here, but on a different platform.

5. Testing with Custom Board

These tests are usable on any board, but you will have to remove the WSTK user interface to use them.

5.1 Starting with the Right Sample

These examples do not use any board specific features, so the only important thing is to select the example for your MCU type (EZR32HG, EZR32LG or EZR32WG). Note that all MCUs have two main types: EZR32xx3yy with USB and EZR32xx2yy without USB. All kits have the USB enabled parts. Since these examples do not use the USB, you can use them on any EZR32xx2yy part. Studio will give you a warning before uploading, but you can safely ignore it.

5.2 Radio Configuration

When you have opened the project and the radio configurator, select the part you have under the section "Application configuration":

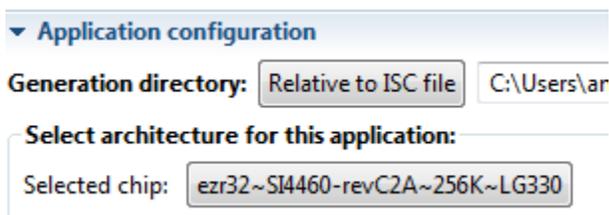


Figure 5.1. Part Selection in Radio Configurator

Otherwise, the radio configuration is exactly the same as the radio configuration with the kits.

5.3 Unmodulated Carrier

Replace the contents of the provided main.c with the following:

```
#include "em_chip.h"
#include "ezradio_cmd.h"
#include "ezradio_plugin_manager.h"
#include "em_cmu.h"

int main(void)
{
    EZRADIODRV_HandleData_t appRadioInitData = EZRADIODRV_INIT_DEFAULT;
    EZRADIODRV_Handle_t appRadioHandle = &appRadioInitData;
    CHIP_Init();
    CMU_ClockEnable(cmuClock_GPIO, true);
    ezradioInit( appRadioHandle );
    ezradioStartUnmodulatedCarrier( appRadioHandle );
}
```

5.4 PN9

Replace the contents of the provided main.c with the following:

```
#include "em_chip.h"
#include "ezradio_cmd.h"
#include "ezradio_plugin_manager.h"

int main(void)
{
    EZRADIODRV_HandleData_t appRadioInitData = EZRADIODRV_INIT_DEFAULT;
    EZRADIODRV_Handle_t appRadioHandle = &appRadioInitData;
    CHIP_Init();
    ezradioInit( appRadioHandle );
    ezradioStartPN9( appRadioHandle );
}
```

5.5 DirectRX (BER)

Replace the contents of the provided main.c with the following:

```
#include "em_chip.h"
#include "em_cmu.h"
#include "ezradio_cmd.h"
#include "ezradio_plugin_manager.h"

int main(void)
{
    EZRADIODRV_HandleData_t appRadioInitData = EZRADIODRV_INIT_DEFAULT;
    EZRADIODRV_Handle_t appRadioHandle = &appRadioInitData;
    CHIP_Init();
    CMU_ClockEnable(cmuClock_GPIO, true);
    ezradioInit( appRadioHandle );
    ezradioStartDirectReceive( appRadioHandle );
}
```

For the EZR32LG and EZR32WG, it is recommended that you enable the PRS channels for GPIO0 and 1:

```
#include "em_chip.h"
#include "ezradio_cmd.h"
#include "ezradio_plugin_manager.h"
#include "em_gpio.h"
#include "em_prs.h"
#include "em_cmu.h"

int main(void)
{
    EZRADIODRV_HandleData_t appRadioInitData = EZRADIODRV_INIT_DEFAULT;
    EZRADIODRV_Handle_t appRadioHandle = &appRadioInitData;
    CHIP_Init();
    CMU_ClockEnable(cmuClock_GPIO, true);

    /* Setup PRS for PTI pins */
    CMU_ClockEnable(cmuClock_PRS, true);

    /* Configure RF_GPIO0 and RF_GPIO1 to inputs. */
    GPIO_PinModeSet((GPIO_Port_TypeDef)RF_GPIO0_PORT, RF_GPIO0_PIN, gpioModeInput, 0);
    GPIO_PinModeSet((GPIO_Port_TypeDef)RF_GPIO1_PORT, RF_GPIO1_PIN, gpioModeInput, 0);

    /* Pin PA0 and PA1 output the GPIO0 and GPIO1 via PRS to PTI */
    GPIO_PinModeSet(gpioPortA, 0, gpioModePushPull, 0);
    GPIO_PinModeSet(gpioPortA, 1, gpioModePushPull, 0);

    /* Disable INT for PRS channels */
    GPIO_IntConfig((GPIO_Port_TypeDef)RF_GPIO0_PORT, RF_GPIO0_PIN, false, false, false);
    GPIO_IntConfig((GPIO_Port_TypeDef)RF_GPIO1_PORT, RF_GPIO1_PIN, false, false, false);

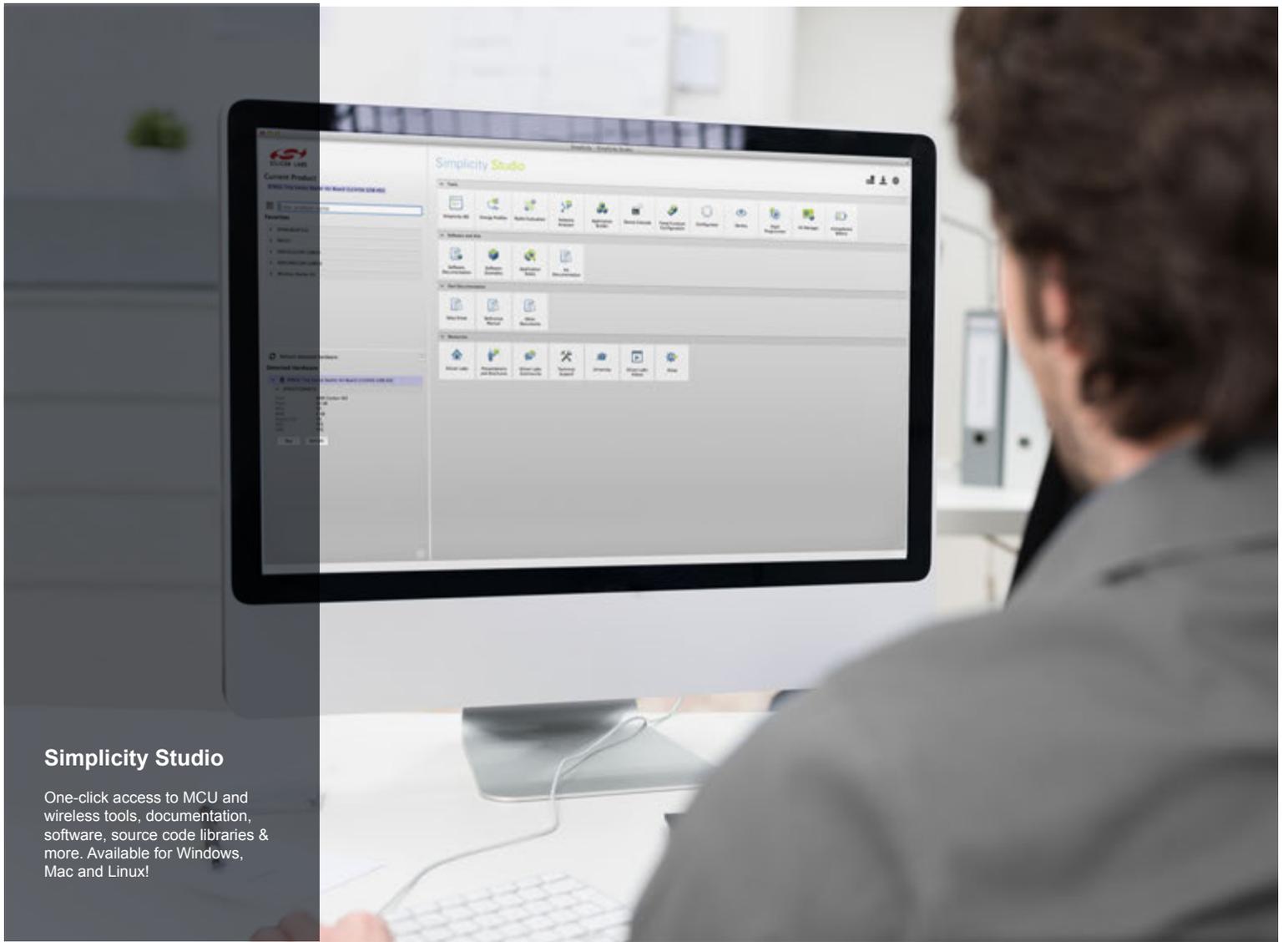
    /* Setup PRS for RF GPIO pins */
    PRS_SourceAsyncSignalSet(0, PRS_CH_CTRL_SOURCESEL_GPIOH, PRS_CH_CTRL_SIGSEL_GPIOPIN15);
    PRS_SourceAsyncSignalSet(1, PRS_CH_CTRL_SOURCESEL_GPIOH, PRS_CH_CTRL_SIGSEL_GPIOPIN14);
    PRS->ROUTE = (PRS_ROUTE_CH0PEN | PRS_ROUTE_CH1PEN);

    /* Make sure PRS sensing is enabled (should be by default) */
    GPIO_InputSenseSet(GPIO_INSENSE_PRS, GPIO_INSENSE_PRS);

    ezradioInit( appRadioHandle );
    ezradioStartDirectReceive( appRadioHandle );
}
```

5.6 RangeTest (PER)

Currently it is not possible to use RangeTest without the WSTK peripherals.



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 and Community
community.silabs.com

Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are not designed or authorized for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR®, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOModem®, Precision32®, ProSLIC®, Simplicity Studio®, SIPHY®, Telegesis, the Telegesis Logo®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.
 400 West Cesar Chavez
 Austin, TX 78701
 USA

<http://www.silabs.com>