Document Number: MKW41ZGENFSKLLSW102QSG
Rev. 1,
04/2017

Kinetis MKW41Z Generic FSK Link Layer Software

Quick Start Guide

This document is a brief presentation of the Kinetis Generic FSK (GENFSK) Software for the MKW41Z wireless microcontroller platforms, version 1.0.3. This software package is built using the Kinetis Software Development Kit (KSDK) version 2.2. This document covers installation of the software packages, hardware setup, build and usage of the provided demo applications.

Contents

1 Download

Navigate to https://mcuxpresso.nxp.com/en/welcome, and create a new configuration:

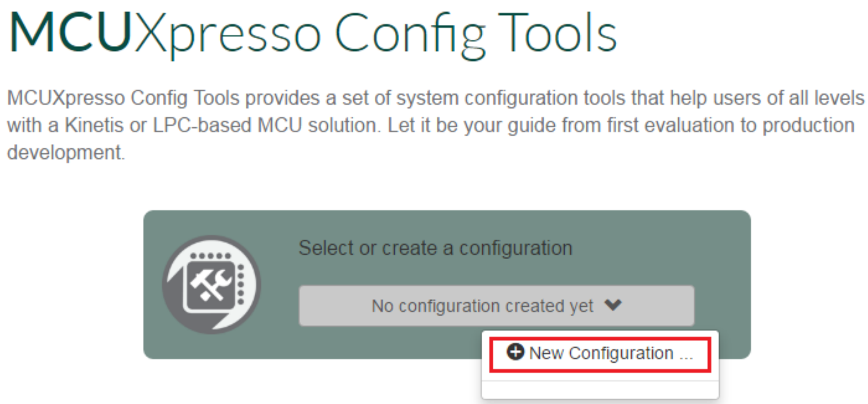

Figure 1: The MCUXpresso welcome screen

Type the desired platform or select from the list, then press “Specify Additional Configuration Settings”:

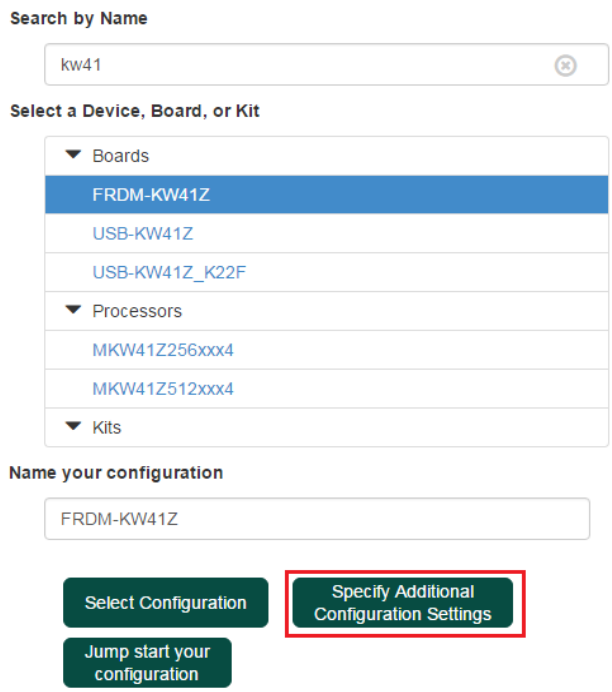

Figure 2: Available configurations

Select the desired toolchain(s) and middleware, then press the “Go to SDK Builder” button:

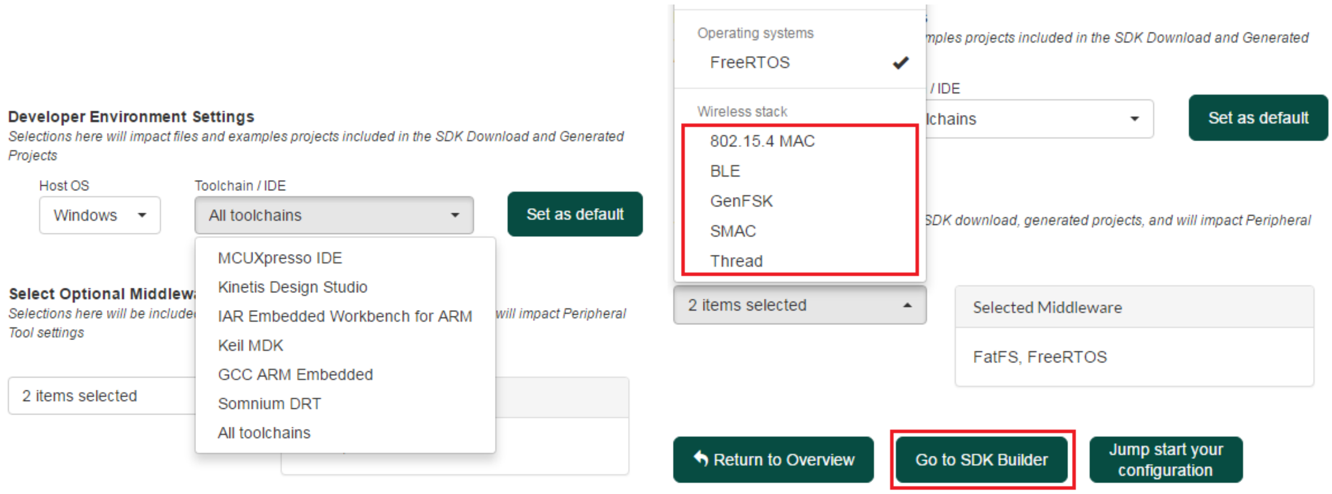

Figure 3: Select toolchain(s) and middleware

If the package is already built, the “Download” button will appear, else the “Request Build” button will appear:

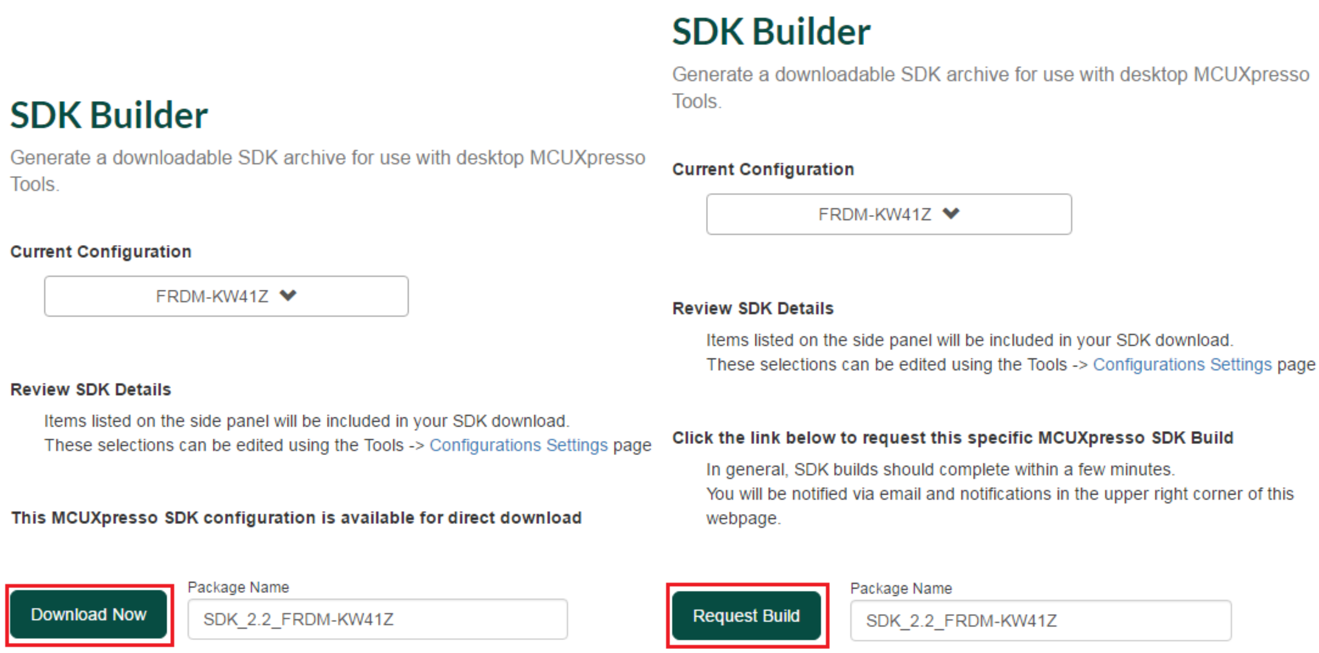

Figure 4: The SDK Builder

If one requests a new build, then a notification will be sent when process ends. To download the archive, access the SDK Archive manager:

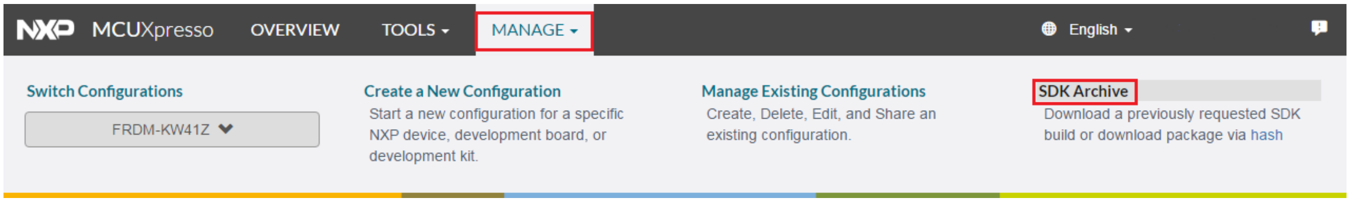

Figure 5: Access the Archive manager

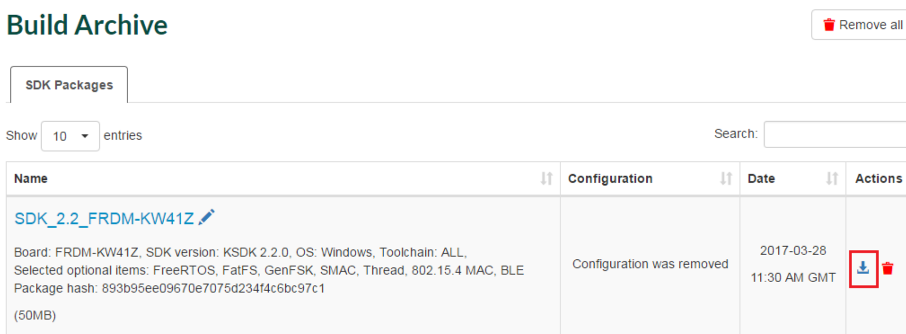

Figure 6: Download the archive

2 Building the Binaries

This section details the required steps for obtaining the binary files for usage with the boards.

NOTE

In order to be able to build any of these packages you need a copy of the IAR Embedded Workbench for ARM® version 7.80.4 or higher or MCUXpresso Integrated Development Environment version 10.0 or higher. This connectivity software package does not include support for any other toolchains.

The packages must be built with the debug configuration in order to enable debugging information.

This package includes various demo applications that can be used as a starting point.

The next section presents the steps required for building the *connectivity_test*. All applications can be found using the following placeholders for text:

- <connectivity_path> : represents the root path of the SDK package.
- <board> : represents the target board for the demo app, can be either “frdmkw41z” or “usbkw41z_kw41z”
- <RTOS>: represents the scheduler or RTOS used by the app, can be “bm” or “freertos”
- <demo_app> : represents the demo app name
- <IDE> : represents the integrated development environment used to build projects and can be “iar” or in the case of MCUXpresso IDE it can be ignored.

The demo applications general folder structure is the following:

<connectivity_path>\boards\<board>\wireless_examples\genfsk\<demo_app>\<RTOS>\<IDE>\

Kinetis Generic FSK Software Demo Application Build Example

Selected app: connectivity_test

Board: frdmkw41z

RTOS: FreeRTOS

Resulting location:

<connectivity_path>\boards\frdmkw41z\wireless_examples\genfsk\connectivity_test\FreeRTOS\<IDE>

NOTE

If your FRDM-KW41Z board is configured for the buck or boost modes of the DCDC converter inside the KW41Z microcontroller, please note that the following defines need to be set: *gDCDC_Enabled_d* to 1 and

APP_DCDC_MODE to *gDCDC_Mode_Buck_c* or *gDCDC_Mode_Boost_c* respectively, in the *app_preinclude.h* header file.

2.1 Building and Flashing the Kinetis Generic FSK Software Demo Applications using MCUXpresso IDE

Step 1:

Drag and drop the archive into the MCUXpresso Installed SDKs window.

Step 2:

Import the SDK examples into Workspace.

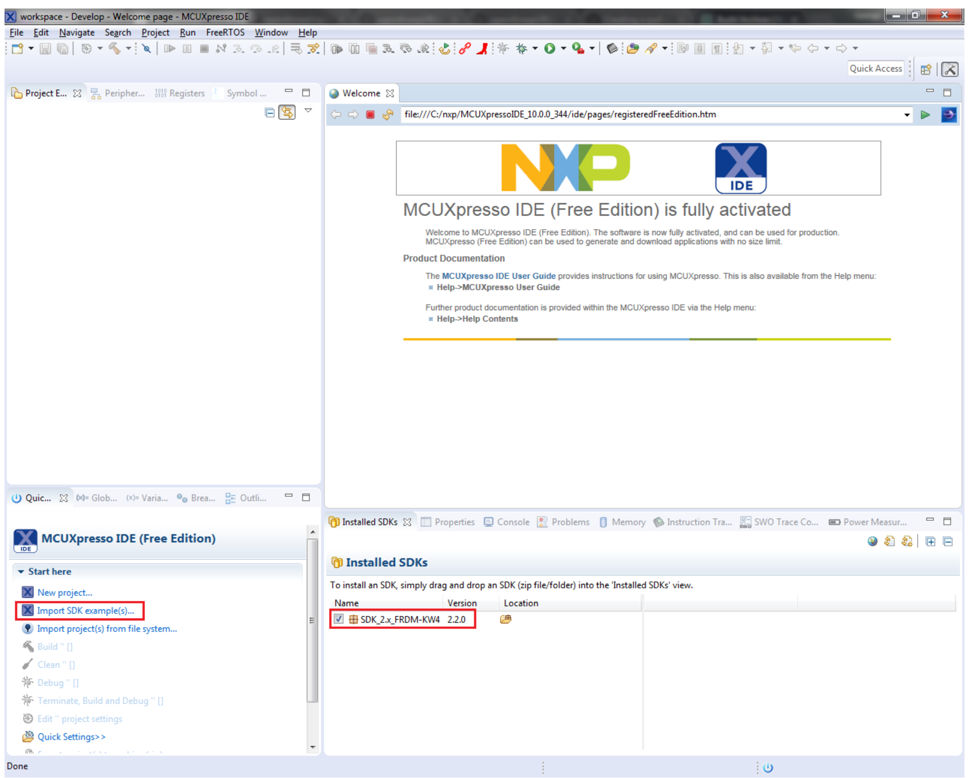

Figure 7: Installed SDKs

Step 3:

Select the board, then the desired example(s):

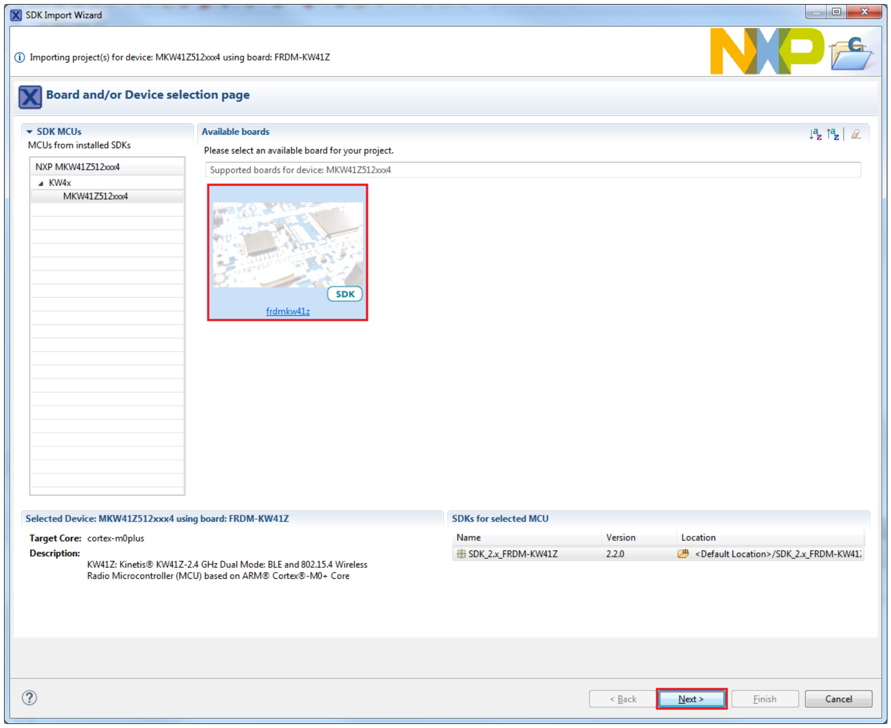

Figure 8: Select the board

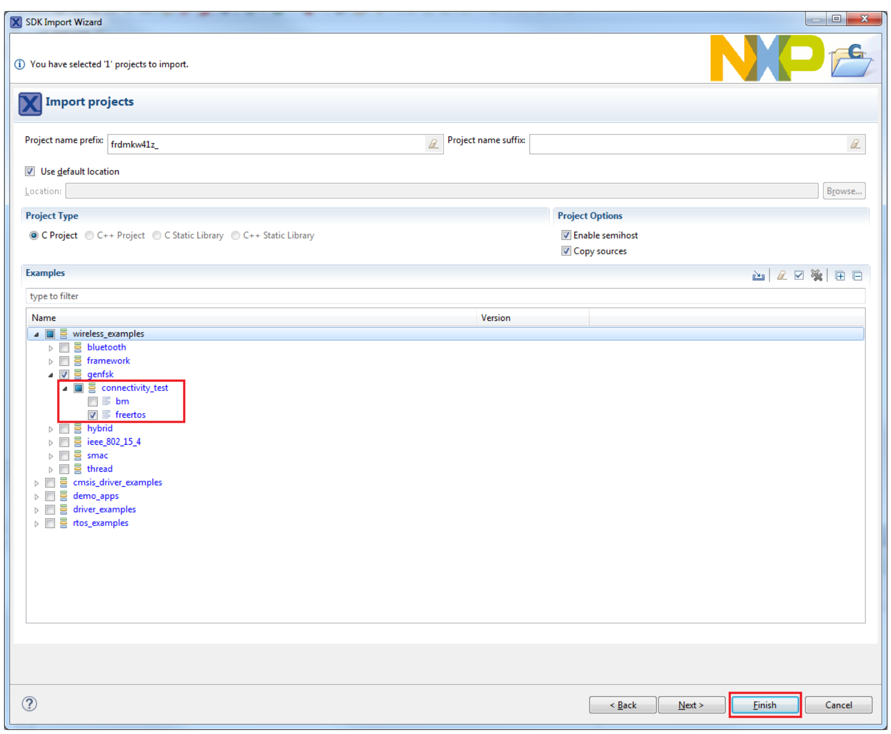

Figure 9: Select the example(s)

Step 4:

Build the connectivity_test project.

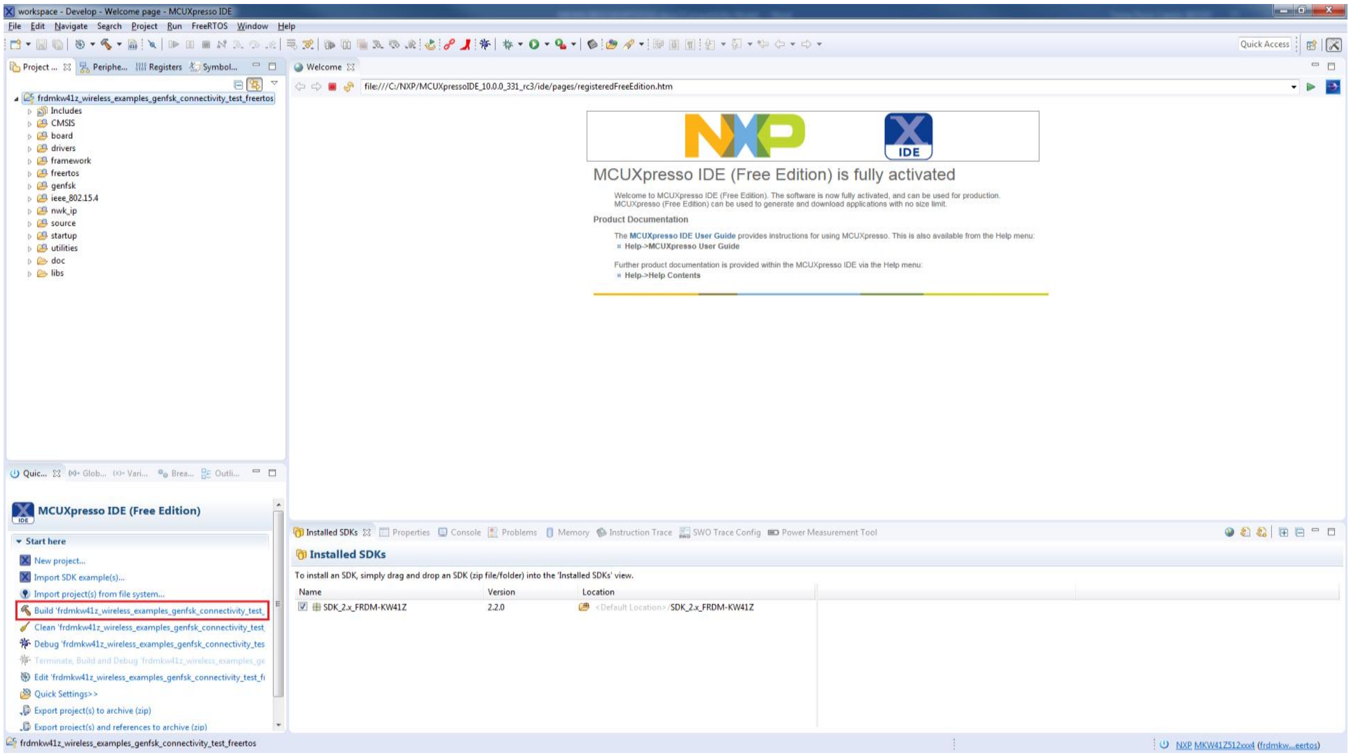

Figure 10: "connectivity_test" FreeRTOS build

Step 5:

Click the "Debug" button to flash the executable onto the board.

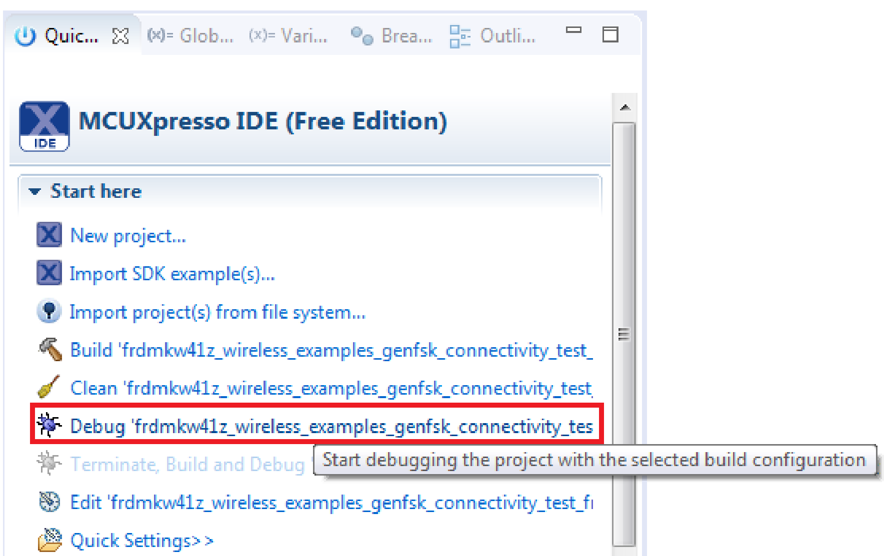

Figure 11: "connectivity_test" Debug

2.2 Building and Flashing the Generic FSK Software Demo Applications using IAR

Step 1:

Navigate to the resulting location in either the connectivity software installation directory or the cloned application root directory.

Step 2:

Open the highlighted IAR workspace file (*.eww file format):

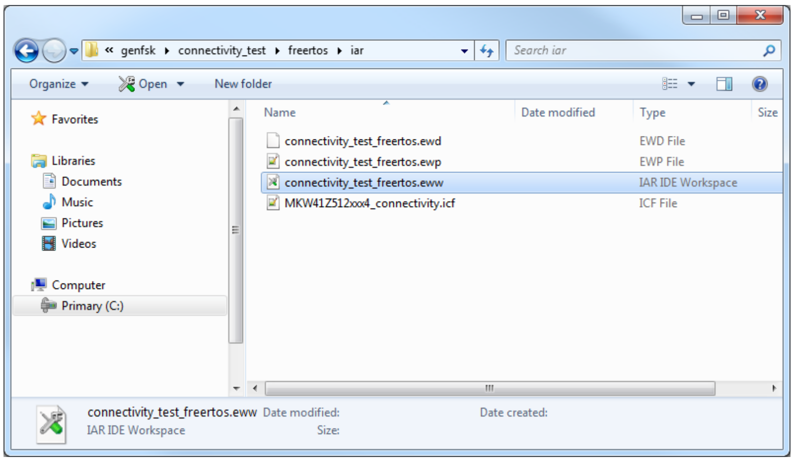

Figure 12: "connectivity_test" demo project location

Step 3:

Select the desired configuration for the connectivity_test project.

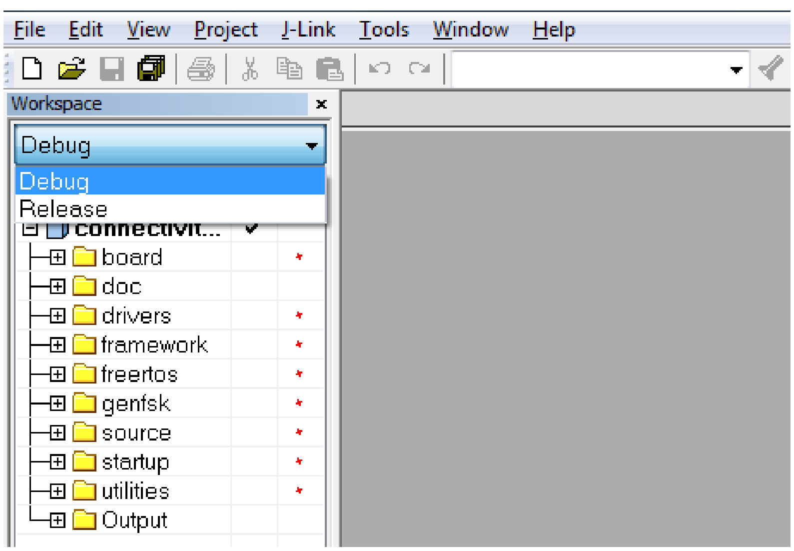

Figure 13: "connectivity_test" FreeRTOS IAR project

Step 4:

Build the project.

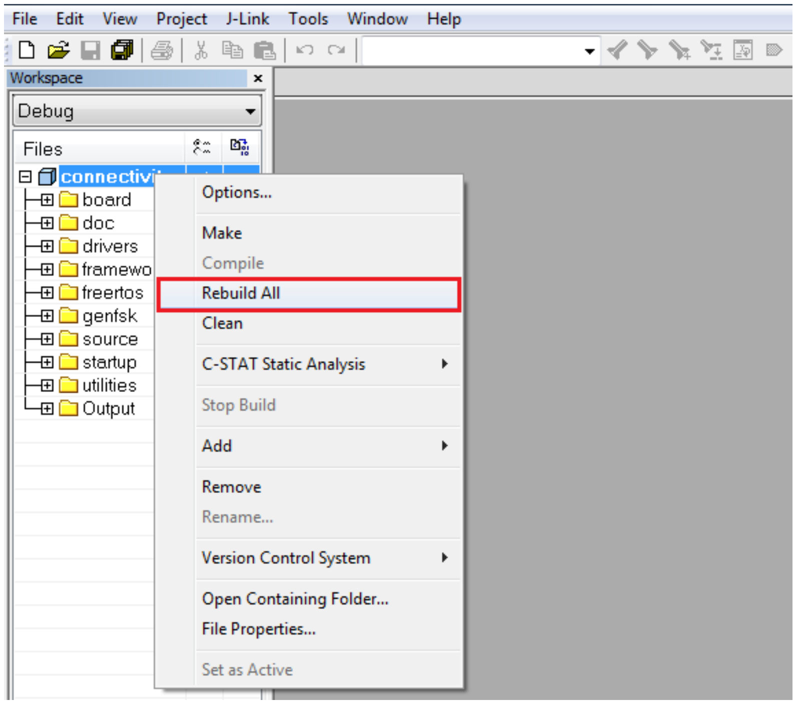

Figure 14: "connectivity_test" FreeRTOS build

Step 5

Make the appropriate debugger settings in the project options window:

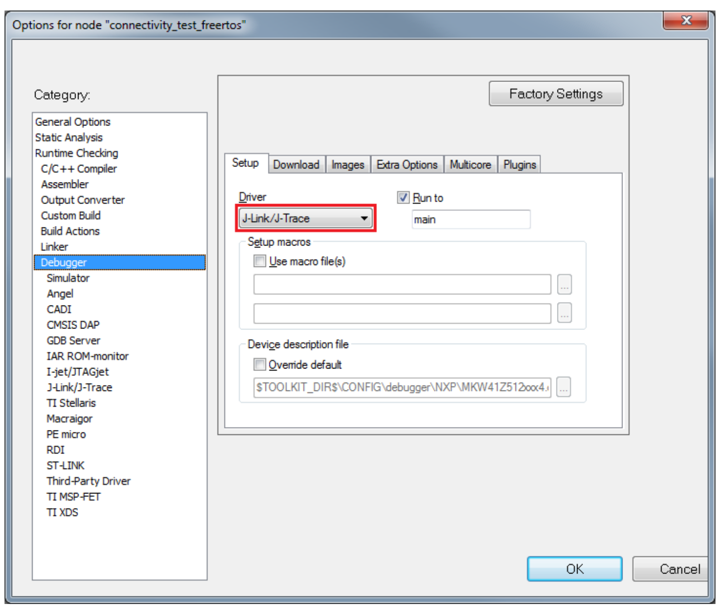

Figure 15: Debugger Settings

Step 6:

Click the "Download and Debug" button to flash the executable onto the board.

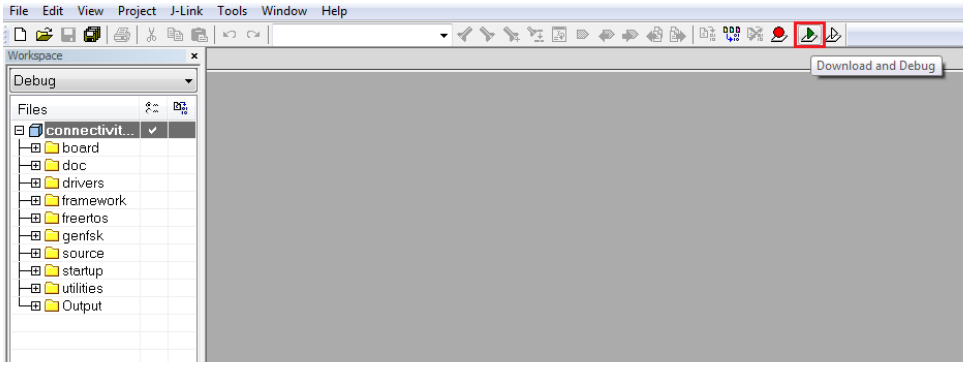

Figure 16: "connectivity_test" Download and Debug

NOTE

The projects are configured to use "J-LINK / J-Trace" as the default debugger. Please make sure that your board's OpenSDA chip contains a J-Link firmware or that the debugger selection corresponds to the physical interface used. See the section below for more information.

2.3 Flashing a Binary Image File Without Using an IDE

The MKW41Z connectivity software package contains in the *<connectivity_path>\tools\wireless\binaries* folder a series of pre-compiled binary applications that can be flashed onto a development board.

In order to flash the corresponding binaries to the FRDMKW41Z board, the best approach is to use the OpenSDA on-board interface J-Link Mass Storage Device functionality, by simply dragging and dropping the binary image in the mass storage drive exposed by this OpenSDA firmware.

In order to flash the firmware on the USBKW41Z, a J-Link probe is needed along with the latest J-Link software from www.segger.com.

Run the *jlink.exe* executable provided in the J-Link software installation follow the steps below for flashing the image on the microcontroller. Make sure that the binary file is in the same folder with the *jlink.exe* executable, or specify the absolute path to the file.

Step 1: Select MKW41Z512xxx4 device.

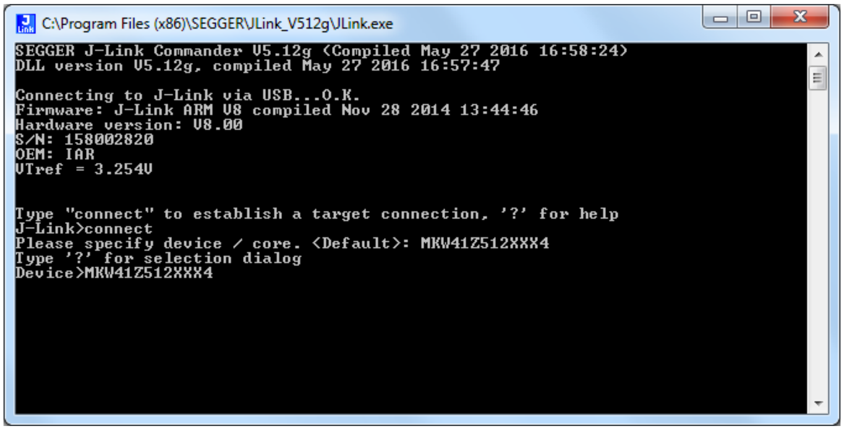

Figure 17: MKW41Z512xxx4 device selection

Step 2: Select SWD target interface.

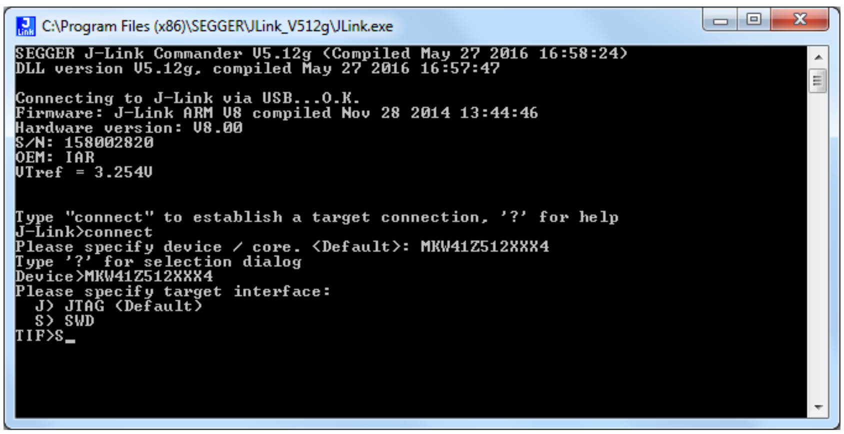

Figure 18: SWD interface selection

Step 3: Press “Enter” to select the default interface speed.

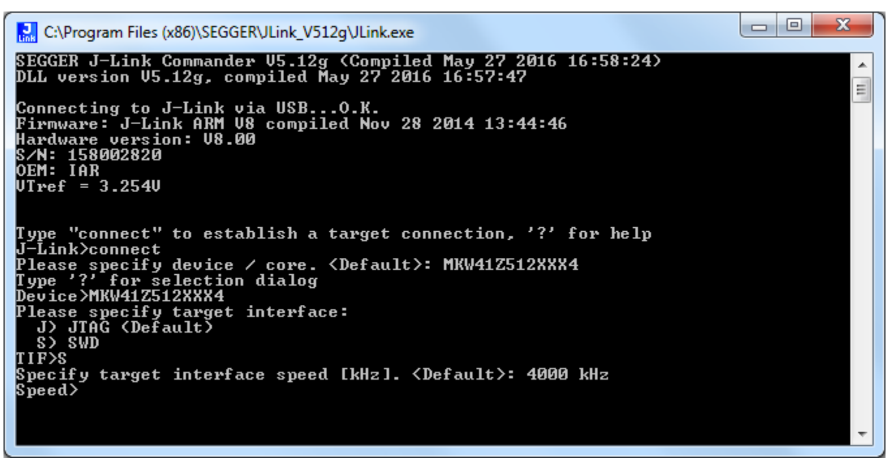

Figure 19: SWD interface speed selection

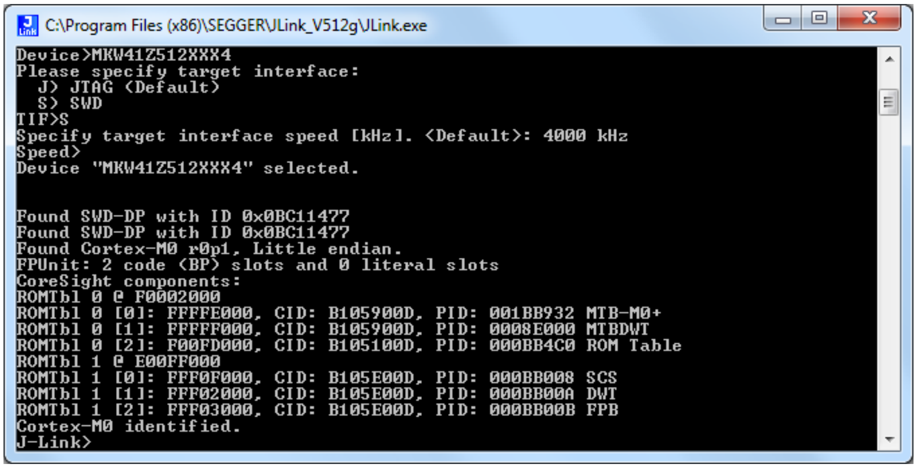

Figure 20: Cortex-M0 identified

Step 4: Type **loadbin app.bin 0** in order to flash the binary file (assuming application name is *app.bin*).

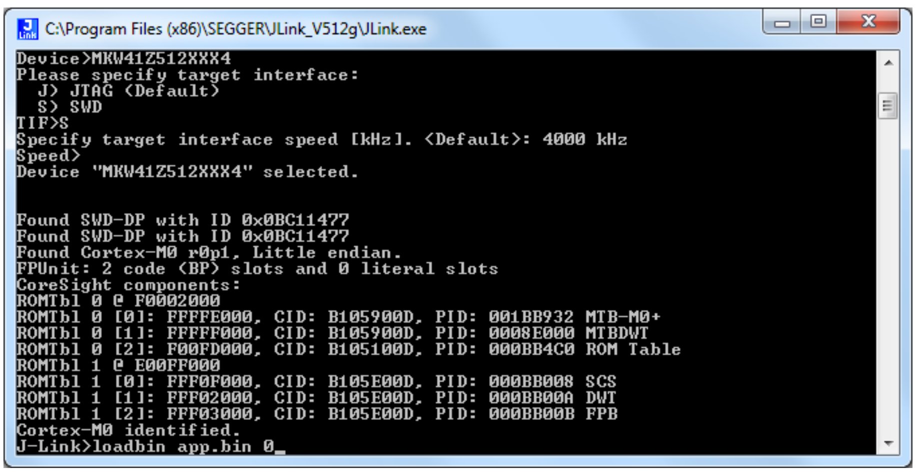

Figure 21: Load binary file

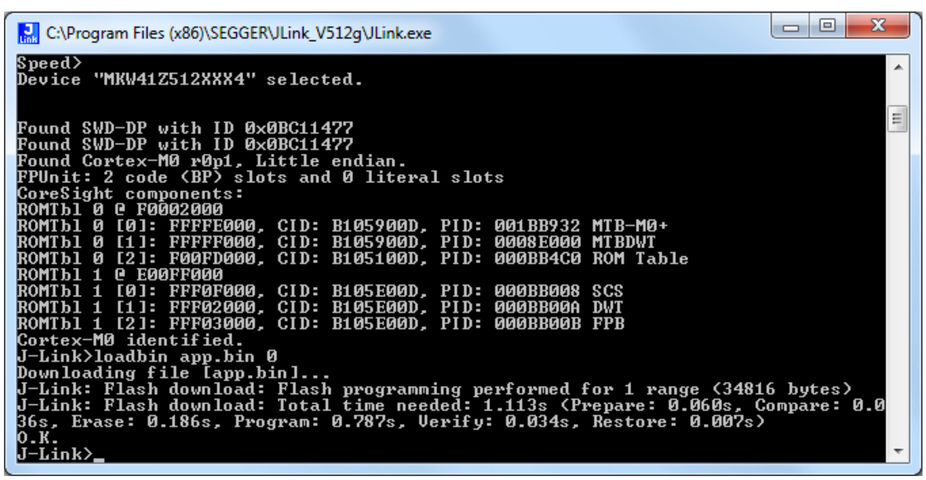

Figure 22: Download completed successfully

3 Hardware Setup

The hardware setup in this example uses either a FRDM-KW41Z or USB-KW41Z development platform, shown in the figure below:

Figure 23: FRDM-KW41Z and USB-KW41Z

The FRDMKW41Z and USBKW41Z boards should have their OpenSDA USB ports connected to a Windows PC. The OpenSDA chip on the boards should have appropriate firmware flashed, with debugging and virtual serial COM port capabilities. For more information on OpenSDA please refer to the following webpage: www.nxp.com/opensda.

Variants of embedded firmware for the OpenSDA chip can be downloaded from:

https://github.com/mbedmicro/CMSIS-DAP
https://www.segger.com/opensda.html
http://www.pemicro.com/opensda/

J-LINK/J-TRACE is the default interface selected in the IAR Embedded Workbench for ARM® projects with FRDMKW41Z and USBKW41Z included in this release.

The FRDM-KW41Z board can be configured via jumpers to be in the two modes of the DCDC converter inside the KW41Z microcontroller or to bypass it entirely, as shown in the figure below:

Power Configuration

	PWR_CFG J18	PSW_CFG J16	DCDC_CFG J17	REG_CFG J22
Bypass Mode (auto start) VDCDC_IN (1.71 to 3.6V) Operation 1.8V - 3.6 V	1-2	1-2	3-4	1-3 2-4 5-6
Boost Mode (auto start) VDCDC_IN (0.9V to 1.8V) Single Battery Operation	2-4	3-4	1-2 5-6	5-6
Buck Mode (manual start) VDCDC_IN (1.8V to 4.2V) Coin Cell Battery Operation	2-4	5-6 press SW6 to start	3-4	5-6
Buck Mode (auto start) VDCDC_IN (1.8V to 4.2V) Coin Cell Battery Operation	2-4	3-4	3-4	5-6

Figure 24: FRDM-KW41Z Jumper Configuration for DCDC Modes

4 Example: Running the Connectivity Test Demo Application

The Generic FSK “connectivity_test” demo application requires a serial terminal program to connect to the boards. For this example, [Tera Term](#) was chosen.

Step 1:

Load the applications on the boards using IAR Embedded Workbench for ARM® by clicking “Download and Debug”.

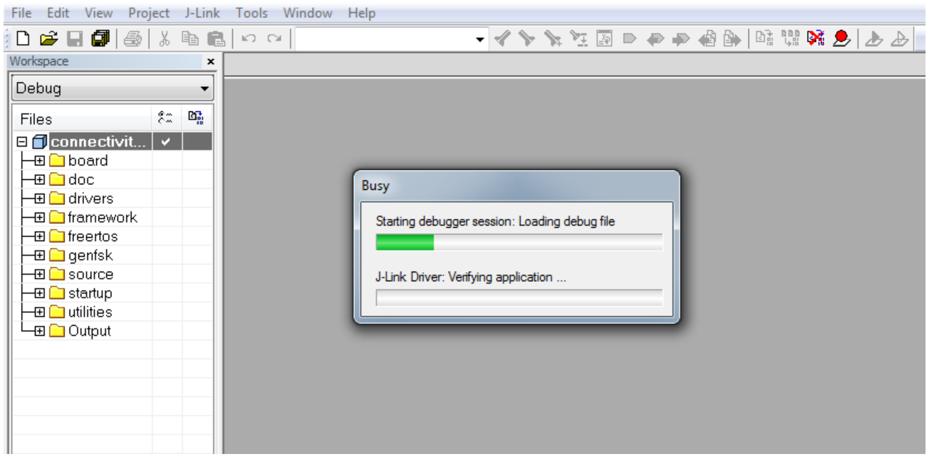

Figure 25: “connectivity_test” loading stage example

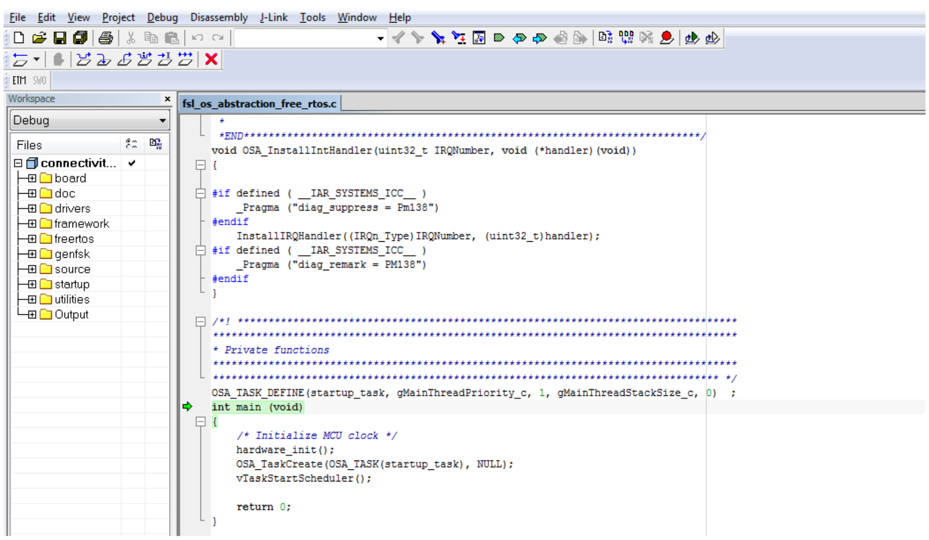

Figure 26: “connectivity_test” application loaded

Step 2:

After loading the application check “Device Manager” to get the serial ports numbers. These should appear with the prefix “JLink”.

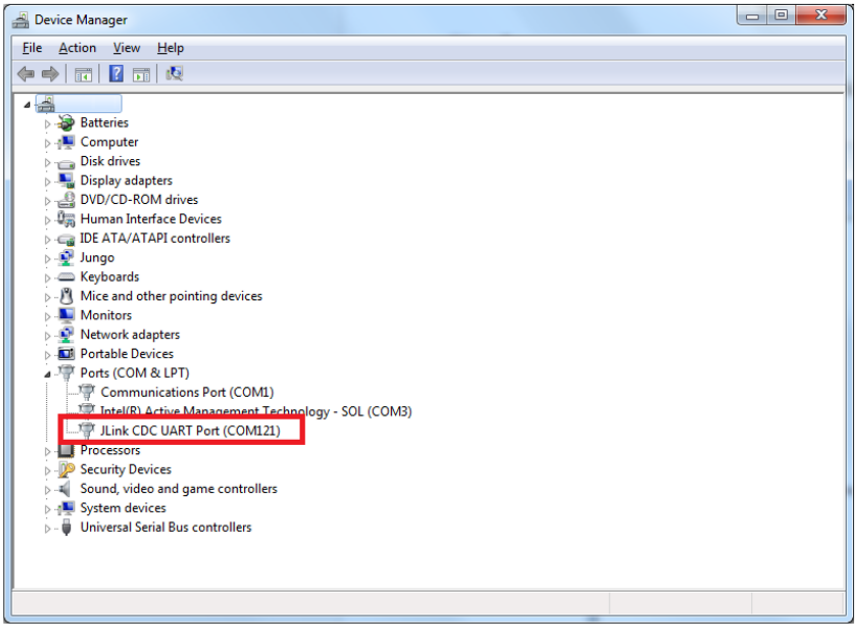

Figure 27: Device Manager serial port lookup

Step 3:

Using the port numbers specified in Device Manager, open a Tera Term instance and connect to the device using the 115200 baud rate. To change the baud rate of the terminal go to “Setup-> Serial Port” menu.

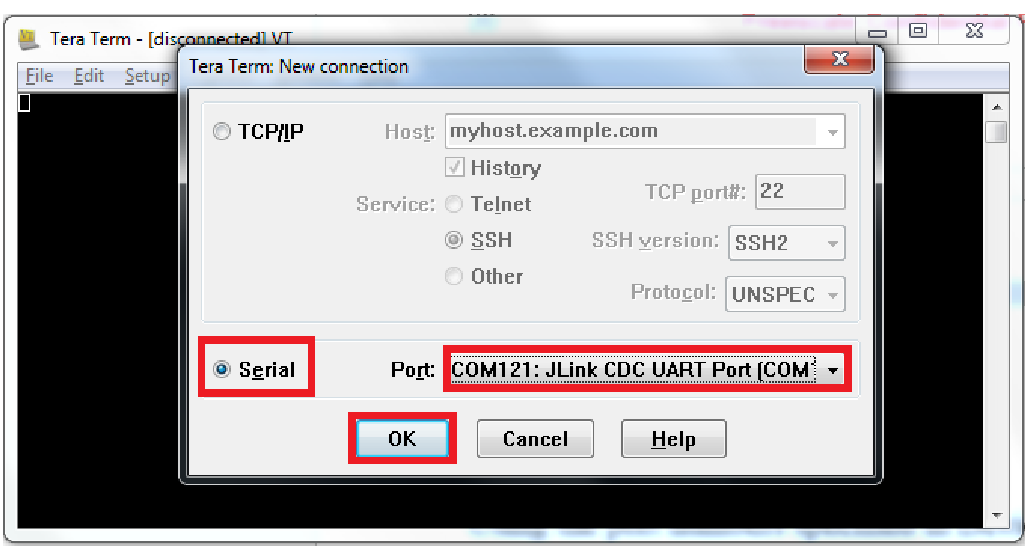

Figure 28: Select JLink serial connection COM port

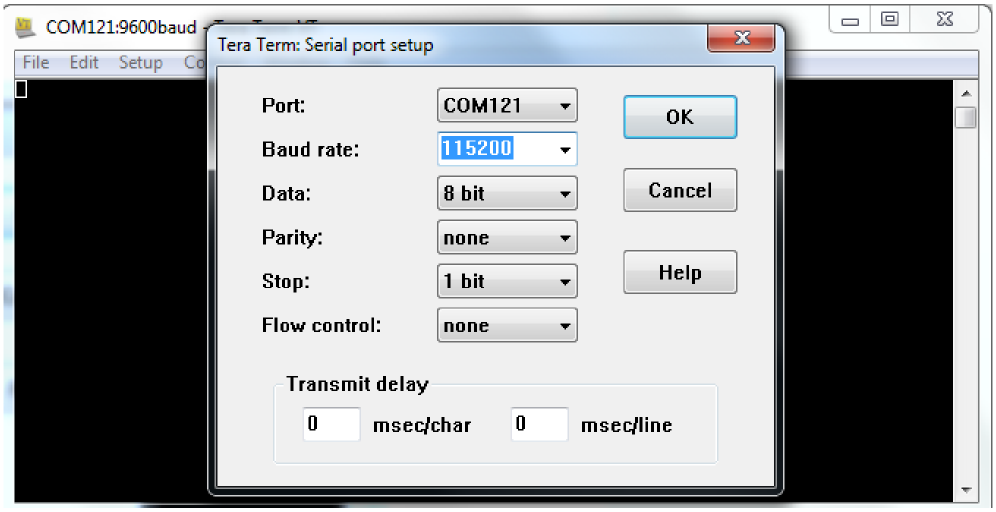

Figure 29: Setting correct baud rate

Step 4:

Start the applications by pressing the ENTER key. Any other key will display the logo screen again.

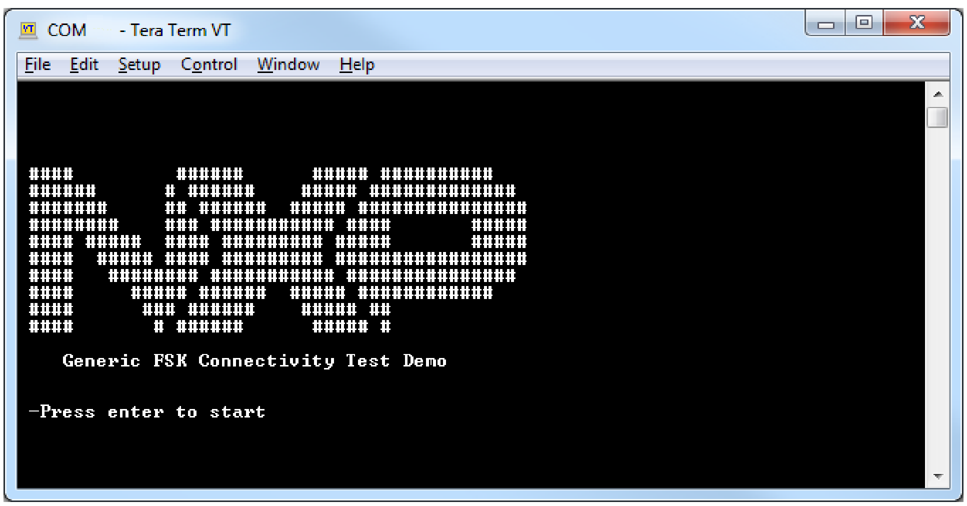

Figure 30: Application after a reset

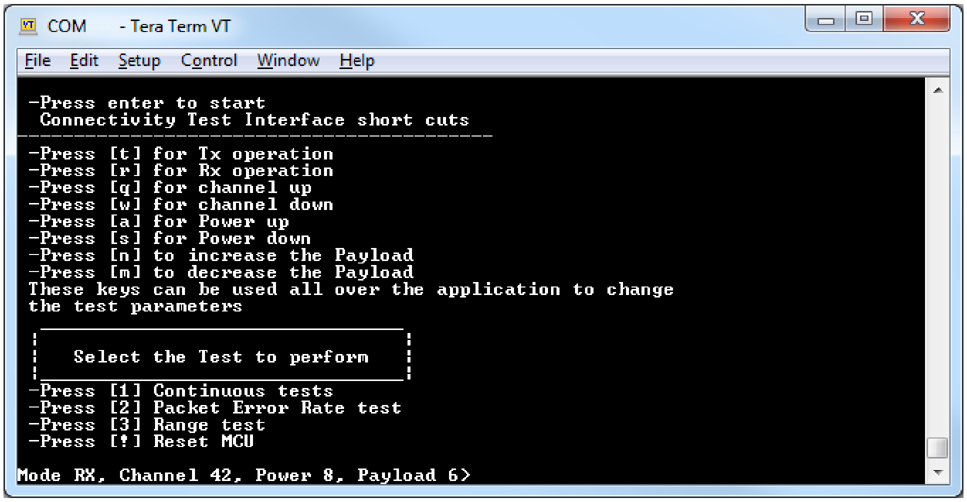

Figure 31: Connectivity Test started

Follow the on-screen instructions to run each test. If a test needs a second platform, follow the steps above to set it up.

The previous section demonstrates the basic steps to run a demo application. A brief description of the Generic FSK Connectivity Test application is presented in the next chapter.

5 Generic FSK Connectivity Test Application description

5.1 Default configuration

For the Generic FSK Connectivity Test application, the transceiver is configured to use BLE modulation and bitrate, the packet processor matches advertising packets and the default channel number (after reset) is set to 42 so the frequency matches the first BLE advertising channel frequency.

The whitener and CRC blocks are configured compatible with the first advertising channel of the BLE protocol. As a direct consequence, the continuous packet reception test is capable of capturing advertising packets.

These configurations (except channel number) are modifiable only at compile time, but the hardware block requirements must be met in order to obtain valid settings. For additional information, see the Generic FSK Link Layer API contained in this release package (*GENFSKLLAPIRM.pdf*)

5.2 Runtime configuration

Runtime configuration is available in most menus. In addition, several continuous tests (that are not using packet mode) can be configured while running. The parameters which can be updated at runtime are:

- Mode: RX or TX
- Channel: 0 to 127 (2360MHz + x*1Mhz)
- TX Power Level: 0 to 32 (power levels, not dBm)
- Payload: 0 to 63 (since default length field size is 6 bits)

The runtime configuration is updated using the shortcut keys and is applied before any test starts. Typically, the possibility of configuring one of the parameters described above is signaled by the presence of the parameters list.

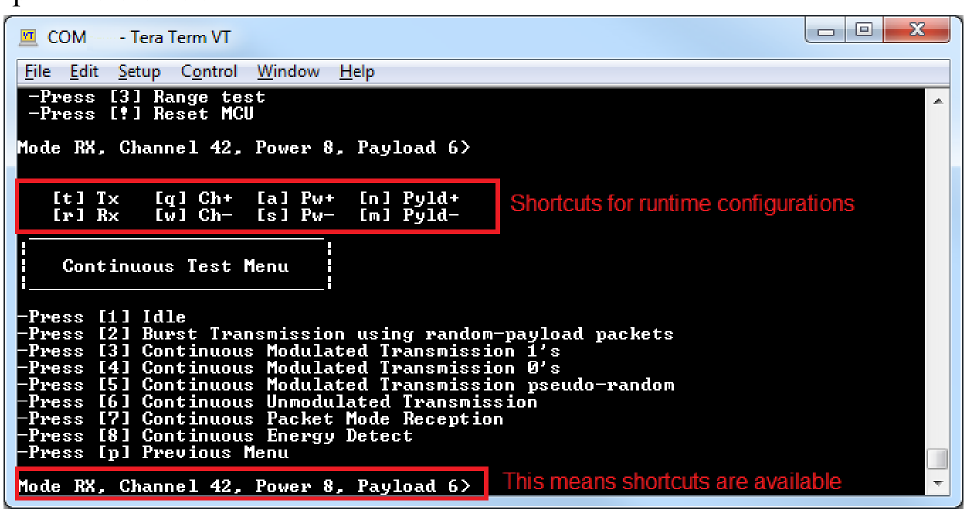

Figure 32: Runtime configuration

The “Mode” parameter is not used directly, but some of the tests display different menus and have different behaviors based on its value.

Similarly, the “Payload” parameter is considered only for some tests. The minimum size is set to 6 because these tests need to include relevant data in the payload.

5.3 Available tests

The Generic FSK Connectivity Test application contains several test suites. The available tests are printed in the main menu, which is displayed after pressing ENTER on the logo screen.

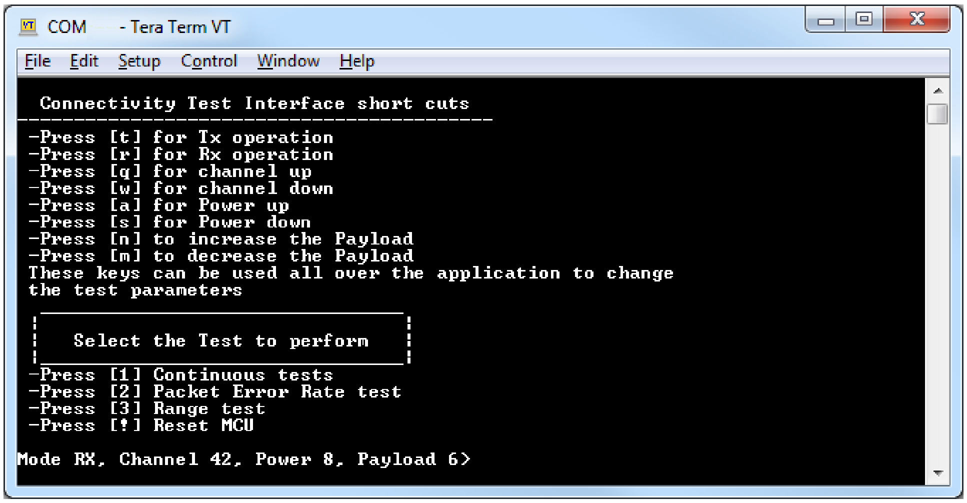

Figure 33: Connectivity Test main menu

5.3.1 Continuous tests

The Generic FSK Connectivity Test allows the user to: set the transceiver in several continuous TX modes, send packet bursts with random payloads, receive and display packets that match the compile time configurations of the packet processor and sample energy level on current channel.

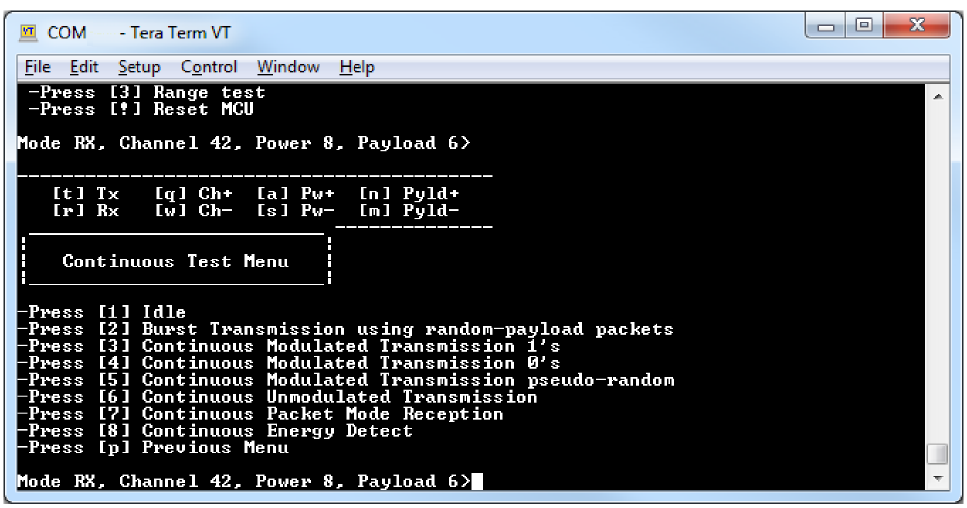

Figure 34: Continuous tests menu

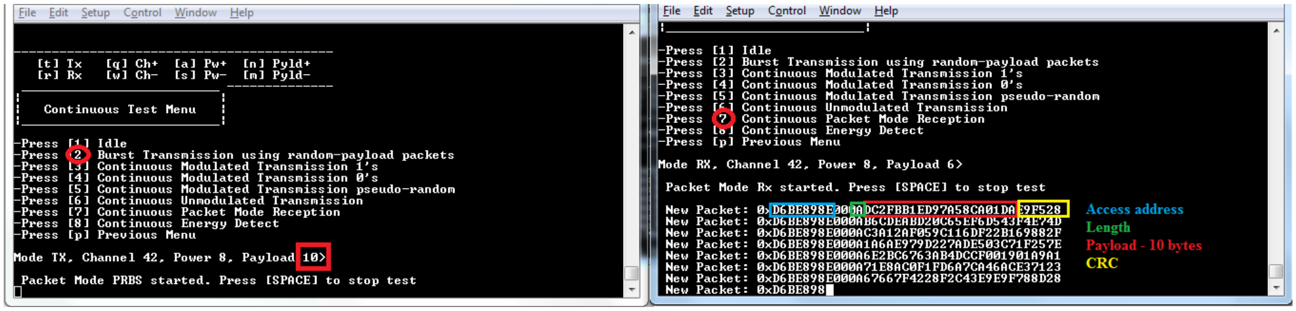

Figure 35: Testing PRBS and continuous packet mode reception

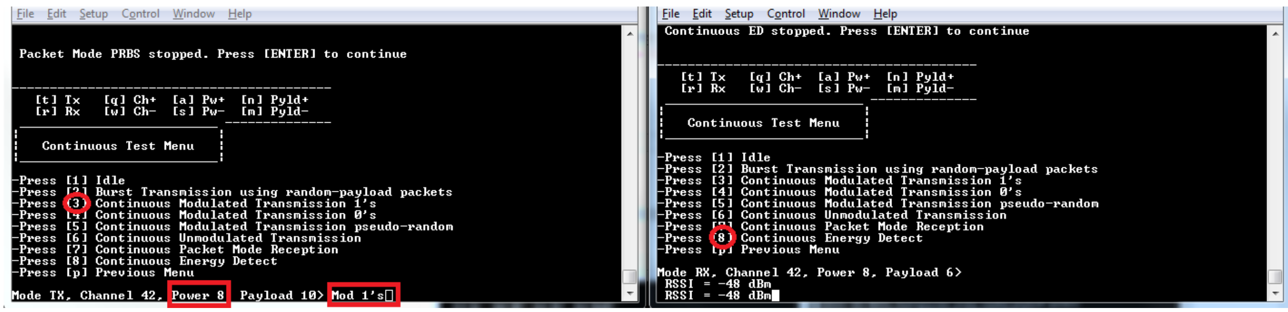

Figure 36: Testing energy detect and continuous modulated TX with 1's pattern

The figure above shows that during a continuous TX (modulated or unmodulated) the power level can be adjusted while running the test. By varying this parameter, the results of the energy detection test will differ.

5.3.2 Packet Error Rate test

To execute the PER test, a second platform is needed. One of the platforms must be set in RX mode (by pressing "r") and the other in TX (by pressing "t").

The TX mode displays several consecutive menus for configuring number of packets to be sent and minimum delay between packets. The payload includes 6 bytes of test-related data. If the payload size (configured using shortcut keys) is greater than 6, the payload is padded with additional data. The test is carried out on the configured channel, at the configured power level.

The RX mode displays a single menu which describes how to start and stop the test. For each PER packet, the test prints the packet index, the received packet index, RSSI and a 32 bit timestamp.

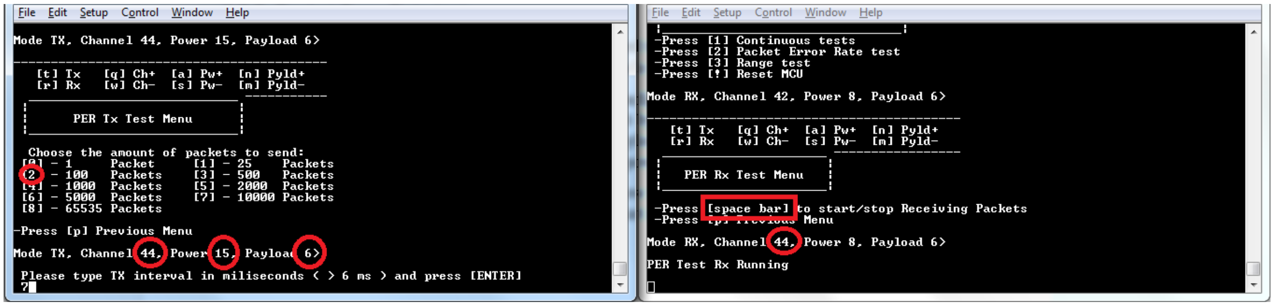

Figure 37: PER test configuration

The 6 milliseconds constraint in the PER TX test is not related to the Generic FSK functionality. It only allows the RX test to print packet related information immediately after the device receives a packet.

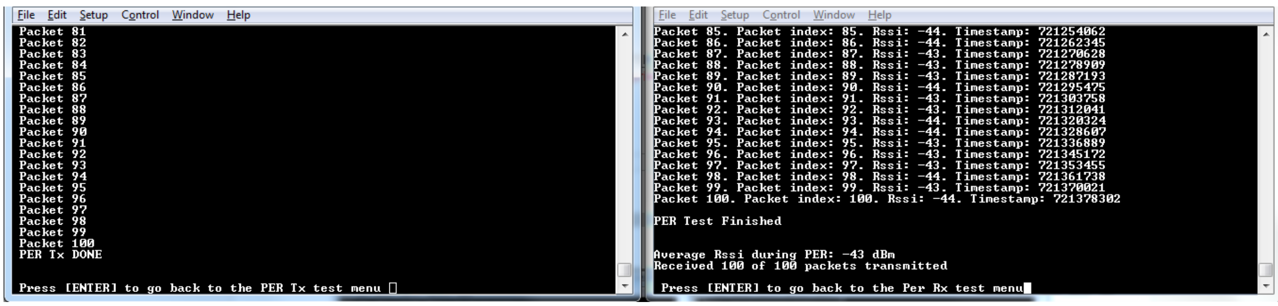

Figure 38: PER test finished

5.3.3 Range test

The Range Test is similar to the Packet Error Rate test. It also needs two platforms, one configured as RX and the other as TX, and it has different menus for each mode.

This test runs a “send-confirm” routine, and logs on both sides the RSSI for each packet received by the RX device.

The TX device starts the test by sending a packet with fixed payload length containing Range Test related payload. Then it waits for a response from the RX device which contains the RSSI associated to the aforementioned packet. If the RX device sends the response packet, the RSSI is displayed. If the TX device does not receive the confirm packet, a “Packet dropped” message is displayed. When the test is stopped, it displays the average RSSI and (on the TX device only) the number of dropped packets.

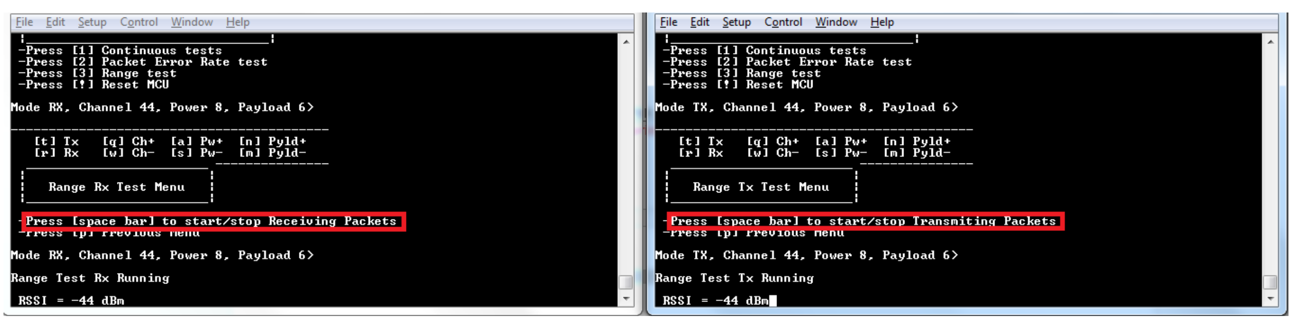

Figure 39: Range Test start

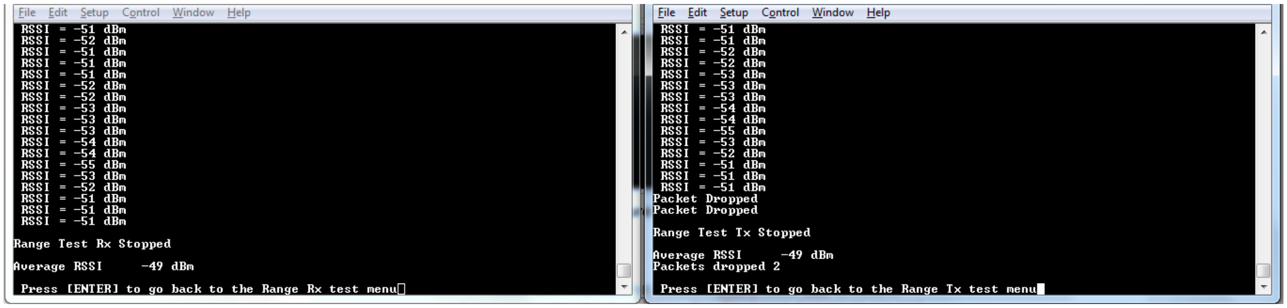

Figure 40: Range test stopped (RX is stopped first, generating dropped packets)

6 Generic FSK Link Layer

6.1.1 Overview

The Generic FSK Link Layer enables radio operation using custom GFSK (with configurable BT product, modulation index and modulation filter coefficients), FSK or MSK modulation formats.

Generic FSK Link Layer also offers a highly configurable packet structure, variable bit rate transmission and reception and packet processing.

The Generic FSK Link Layer provides the interface between the application and the Generic FSK Link Layer Controller. The Generic FSK Link Layer allows a highly configurable packet structure, defining the lengths, bit ordering and contents of individual packets fields, defining the start and end points for whitening, CRC and Manchester encoding/decoding and some primitive parsing of the packet header.

The Generic FSK Link Layer also features a RAW packet transfer mode, bypassing most of the hardware acceleration (with only preamble detect and network address match available), with the limitation that only a maximum of 35 bytes of data (including header and CRC if available) can be received or transmitted. The 35 bytes limitation does not include the preamble size and network address size.

The Generic FSK Link Layer is also instantiable, enabling use of several configurations in the same applications without the need to reconfigure when using one or the other. Important notice, only one instance can send/receive data at a moment of time due to the fact that all use the same Generic FSK Link Layer Controller.

6.1.2 Initializing the Generic FSK Link Layer

There are three most important APIs to initialize the Link Layer and to allocate and free Link Layer instances:

- **GENFSK_Init**()
- **GENFSK_AllocInstance**()
- **GENFSK_FreeInstance**()

The **GENFSK_Init()** is the first API that the application must call to initialize the required GENFSK task, instances and conditional variables. The Generic FSK Link Layer performs a low level initialization during this process. The Generic FSK instances initialization is not performed here.

The application is required to include "**genfsk_interface.h**" to be able to invoke **GENFSK_Init()**.

The **GENFSK_AllocInstance()** API allocates a new GENFSK instance (if the maximum number of available instances was not reached) and performs the GENFSK instance initialization and configuration. If no configuration structure is passed for one or several parameters, then the default values will be used. The application is required to include "**genfsk_interface.h**" to be able to invoke **GENFSK_AllocInstance()**. Refer to "GENFSKLLAPIRM.pdf" for more details.

The Generic FSK Link Layer does not provide an API to completely remove the Link Layer. There is no API to kill or exit tasks created during **GENFSK_Init()** API. If an instance is no longer needed it can be freed dynamically at any point of time using **GENFSK_FreeInstance()** API.

6.1.3 How to use Generic FSK Link Layer APIs

After the Generic FSK Link Layer initialization and instance allocation, interaction with Generic FSK Link Layer is the most important aspect for the application.

The Generic FSK APIs can be broadly categorized as shown in the table below:

Generic FSK API Category	Description/Example
Configuration APIs	These APIs allows the application to configure, read or set values for various parameters on a Generic FSK instance. Example: **GENFSK_RadioConfig()** **GENFSK_SetPacketConfig()** **GENFSK_GetPacketConfig()** **GENFSK_SetCrcConfig()** **GENFSK_GetCrcConfig()** **GENFSK_SetWhitenerConfig()** **GENFSK_GetWhitenerConfig()**
Callback Registration APIs	These API enable application to register various callbacks with Generic FSK Link Layer instances. Example: **GENFSK_RegisterCallbacks()**

Send and Receive APIs	These API enable application to send or receive data on a specific instance, or to cancel a pending transaction. Example: **GENFSK_StartTx()** **GENFSK_StartRx()** **GENFSK_AbortAll()**
Utility APIs	These APIs help application to perform various jobs such as formatting a packet before transmission, extracting parameters from byte stream after reception, get current timestamp, etc. Example: **GENFSK_PacketToByteArray()** **GENFSK_ByteArrayToPacket()** **GENFSK_GetTimestamp()**

For all Generic FSK APIs, application must include the "**genfsk_interface.h**" header file.

6.1.4 Generic FSK Link Layer Configuration APIs

The APIs in this category enables application to configure the radio, packet format, channel, TX power and network address. Each configuration can be changed at runtime as long as the affected instance is not active sending or receiving data. If the instance is active, in order to make configuration changes, the current active sequence has to be aborted first.

- **GENFSK_RadioConfig()** sets the radio configuration for given instance. The radio configuration includes radio modes (GFSK, FSK and MSK) and data rate (1Mbps, 500Kbps and 250Kbps).

  Important notice, in MSK mode only RAW packets are transmitted and received. The packet size is limited to 35bytes of data and most of the hardware acceleration is bypassed. Also if needed, the DSP library available in Framework offers APIs to perform software CRC calculation, Scrambling and also Forward Error Correction.
- **GENFSK_SetPacketConfig()** sets the packet configuration for the given instance. The packet configuration includes preamble size, packet type (formatted or RAW), length field size, h0 and h1 header fields sizes and also h0 and h1 mask and match fields.
- **GENFSK_GetPacketConfig()** returns the configuration structure for the given instance, if allocated.

- **GENFSK_SetCrcConfig()** sets the CRC configuration for the given instance, if allocated. The CRC configuration includes CRC enable (hardware CRC enable or disable), receive invalid CRC (if set, a packet received with invalid CRC is sent to the application, else an event for invalid CRC is sent to the application), CRC size, CRC start byte (the CRC start byte position, position #0 is the first byte of Sync Address), CRC reflect input, CRC reflect output, CRC byte order, CRC seed, CRC poly and CRC XOR out (masks the CRC result with this value).
- **GENFSK_GetCrcConfig()** returns the configuration structure for the given instance, if allocated.
- **GENFSK_SetWhitenerConfig()** sets the Whitener configuration for the given instance. The Whitener configuration includes Whiten enable (hardware Whitener enable or disable), Whiten start (whitening start byte), Whiten size, Whiten initialization, Whiten size threshold, and Manchester encoding/decoding. Refer to "GENFSKLLAPIRM.pdf" for more details.
- **GENFSK_GetWhitenerConfig()** returns the configuration structure for the given instance, if allocated.
- **GENFSK_SetNetworkAddress()** sets one of the network address used for network address match for the given instance, if allocated.
- **GENFSK_GetNetworkAddress()** returns the configured network address for one of the four available location for the given instance, if allocated.
- **GENFSK_EnableNetworkAddress()** enables one of the network address matching for the given instance, if allocated.
- **GENFSK_DisableNetworkAddress()** disables one of the network address matching for the given instance, if allocated.
- **GENFSK_SetChannelNumber()** sets the channel number for the given instance, if allocated.
- **GENFSK_GetChannelNumber()** returns the configured channel number for a given instance, if allocated.
- **GENFSK_SetTxPowerLevel()** sets the TX power level for the given instance, if allocated.
- **GENFSK_GetTxPowerLevel()** returns the configured TX power level for the given instance, if allocated.

6.1.5 Generic FSK Link Layer Send and Receive APIs

The APIs in this category enable the application to send or receive data on a given instance, or to cancel a pending sequence. **Only one sequence can be active at a time.**

In order to send data on one instance, the application have to use **GENFSK_StartTx()**. Based on the instance configuration, the **GENFSK_StartTx()** has a different behavior.

If **gGenfskFormattedPacket** is selected for Packet Type, then **GENFSK_StartTx()** expects that the input buffer to be compliant to the configured settings for packet format.

If **gGenfskRawPacket** is selected for Packet Type, then **GENFSK_StartTx()** bypasses all the hardware acceleration and a maximum of 35bytes packet length is expected.

Also if the selected radio mode is **MSK**, then **GENFSK_StartTx()** bypasses all the hardware acceleration and a maximum of 35bytes packet length is expected.

The transmission can also be delayed by setting **txStartTime** to a value other than 0. Time base roll over at 24bits (~16.7 seconds) must be considered in setting the **txStartTime**.

In order to receive data on one instance, the application have to use **GENFSK_StartRx()**. Based on the instance configuration, the **GENFSK_StartRx()** has a different behavior.

If **gGenfskFormattedPacket** is selected for Packet Type, then **GENFSK_StartRx()** expects that the data received over the air to be compliant to the configured settings for packet format. In this case **maxBufLengthBytes** field can be set to any value, and all the packets with length smaller than or equal to **maxBufLengthBytes** compliant with the packet format will be received.

If **gGenfskRawPacket** is selected for Packet Type, then **GENFSK_StartRx()** bypasses all the hardware acceleration and a maximum of 35bytes packet length can be received. Also if the selected radio mode is **MSK**, then **GENFSK_StartRx()** bypasses all the hardware acceleration and a maximum of 35bytes packet length can be received.

For both **gGenfskRawPacket** Packet Type or **MSK** radio mode, **maxBufLengthBytes** has to be set to the exact value for the received packet size. All packets with different length will be rejected.

The reception can also be delayed by setting **rxStartTime** to a value other than 0. Time base roll over at 24bits (~16.7 seconds) must be considered in setting the **rxStartTime**.

The reception duration can be set by **rxDuration**. If no valid packet is received during this time, the reception will timeout and an event is sent to the application.

Active transmissions or receptions can be aborted by using one of the sequence specific APIs, **GENFSK_CancelPendingTx()** and **GENFSK_CancelPendingRx()** or by using the generic API which will abort any active sequence, **GENFSK_AbortAll()**.

6.1.6 Generic FSK Link Layer Application Event Indication Callbacks

This callback mechanism is provided to notify the application of incoming packets or events as they are received from the Link Layer hardware or software. Each Generic FSK instance has its own callbacks and for proper usage the callbacks must be registered with **GENFSK_RegisterCallbacks()** before Link Layer usage. Refer to “GENFSKLLAPIRM.pdf” for more details.

6.1.7 Generic FSK Link Layer Utility APIs

The APIs in this category help application to perform various jobs such as formatting a packet before transmission, extracting parameters from byte stream after reception and return current timestamp.

GENFSK_PacketToByteArray() API is used to convert a packet buffer to a byte array format to be sent by GENFSK LL. The format used will be the one configured for the given instance.

GENFSK_ByteArrayToPacket() API is used to convert the byte array received over the air in **GENFSK_packet_t** format. The format used will be the one configured for the given instance.

GENFSK_GetTimestamp() returns the current value of the time base for the Generic FSK Link Layer. The time base is the same for all instances.

7 Connectivity Test Demo Application with Low Power enabled

The Connectivity Test demo application low power support can be enabled by setting the following defines in **app_preinclude.h** file:

```
/* Enable/Disable PowerDown functionality in PwrLib */
#define cPWR_UsePowerDownMode          1

/* Enable/Disable GENFSK Link Layer DSM */
#define cPWR_GENFSK_LL_Enable          1

/* Default Deep Sleep Mode */
#define cPWR_DeepSleepMode             7
```

The API **PWR_GENFSK_EnterDSM**() is used to put the Generic FSK Link Layer and radio along with the MCU (if possible) to sleep. **PWR_GENFSK_EnterDSM**() takes as input parameter the **dsmDuration** in milliseconds.

Low power support is included in "**Packet Error Rate test**" and "**Range test**".

Packet Error Rate Test

The sleep duration is sent in the packet payload by the device with TX role and interpreted by the device with RX role. After the packet was sent, the device with TX role will enter sleep mode for 1000ms (in the provided demo application).

If the device with RX role receives the packet, it enters sleep mode also but with 10ms less than the device with TX role in order to wake up earlier and start to listen for a new packet. If several packets are lost on the RX side, the device will not go back to sleep mode.

For this demo application, the transmitting device sends a packet every 1000ms and goes back to sleep.

Range test

For this demo application, the sleep duration is fixed at 100msec. After a packet was sent by the device with TX role, the device enters RX state and waits for the confirmation packet for 10msec. If the confirmation is received it enters back to sleep mode for 100msec. For the device with RX role, after a packet is received, the device enters TX state and sends the confirmation then enters sleep mode for 90msec in order to wake up earlier and listen for a new packet.

How to Reach Us:

Home Page:
www.nxp.com

Web Support:
www.nxp.com/support

.

Document number: MKW41ZGENFSKLLSW103QSG
Rev. 1
04/2017