

Guide for Manufacture Testing of RF4CE Products

Low-Power Wireless Products

ABSTRACT

This application report describes a sample implementation of a manufacturing line test program for RF4CE products. It is based on RemoTI-Linux, which is the Linux host driver for CC253x RemoTI Network Processor (RNP). It includes the ability to setup a fake pairing between the remote control and target side, how to enter test mode, how to change RF parameters, and so forth. It helps customers to understand RemoTI-Linux structure and utilize the API of TI RF4CE stack. Based on this application report, customers can easily implement their own test flow and meet varying factory environments.

The last section discusses the way to perform basic RF functions with test equipment. Besides implementing the test program, customers can perform RF tests, too.

Contents

1	RemoTI-Linux Introduction	2
2	Implement Test Program	5
3	Setup for RF Test	12
4	RemoTI Mass Production Test	21
5	Summary	22
6	References	23

List of Figures

1	RemoTI Network Processor Component Architecture	3
2	RemoTI-Linux Software Architecture	3
3	RemoTI-Linux Application Architecture	4
4	Network Enabling for NPI Client	4
5	The setup for the Test Program Implementation	5
6	Controller Could Send Packet to Target After Doing Fake Pairing.....	9
7	Setup for RF Test	12
8	Tx Test Mode0 Launch on Linux Laptop	13
9	Tx Test Mode0 Perform on Spectrum Analyzer	13
10	Tx test Mode1 Launch on Linux Laptop	14
11	Tx Test Mode1 Perform on Spectrum Analyzer	15
12	Rx Test Launch on Linux Laptop	16
13	Rx Test Perform on Spectrum Analyzer.....	16
14	PER Test Launch on Linux Laptop	17
15	Sniff Log While Performing PER Test	18
16	NPI Server and Client Run on Different Devices.....	18
17	Setup for PER Test With SmartRF Studio	19
18	Configuration on SmartRF Studio	21
19	Execute massProduction_Inx_x86_client	22
20	Result of RemoTI Mass Production Test.....	22

List of Tables

All trademarks are the property of their respective owners.

1	Acronyms	2
2	Configuration for Fake Pairing on Target and Controller	8
3	Pairing Table After Fake Pairing on Target and Controller.....	8
4	Configuration for Device Own Information on Target and Controller	9
5	Definition of nodeCapabilities	9
6	Test Modes Supported by RTI_TestModeReq	10
7	Server Configuration Difference on Target and Controller.....	17

1 RemoTI-Linux Introduction

This section introduces the architecture of RemoTI-Linux, which utilizes the RNP architecture for a Linux host. To simplify the development effort, this document provides some code snips that are described in the following sections. Then, you can implement your own manufacturing test program based on the RemoTI-Linux source code.

1.1 Acronyms Used in This Document

Table 1. Acronyms

Acronym	Description
Device	A physical object consisting of an IEEE 802.15.4 radio
Node	A device containing RF4CE functionality
Target	A Node that implements the Target functionality as defined in the RF4CE specification (for example, a TV)
Controller	A Node that implements the Controller functionality as defined in the RF4CE specification (for example, a remote control)
ACK	Acknowledge
API	Application Programming Interface
ARC	Advance Remote Control
DUT	Device under test
EM	Evaluation module
IEEE	Institute of Electrical & Electronics Engineers, Inc.
LAN	Local Area Network
NPI	Network Processor Interface
RTIS	RemoTI application framework surrogate
RemoTI-Linux	A RF4CE application for a Linux host using a CC253x RNP
RF	Radio Frequency
RF4CE	Radio Frequency for Consumer Electronics
RNP	RemoTI Network Processor. An application configuration of the RemoTI stack that configures the CC253x device as a network processor
RTI	RemoTI Application Framework
STB	Set-Top Box
TE	Target Emulator
TI	Texas Instruments Incorporated
ZID	Zigbee Input Device, a name for a ZigBee RF4CE profile
ZRC	Zigbee Remote Control, a name for a Zigbee RF4CE profile

1.2 RemoTI Network Processor Architecture

Figure 1 shows the architecture of RemoTI Network Processor [1]. The right side is the network processor. An RTI surrogate module (RTIS) decodes the command frames sent over universal asynchronous receiver/transmitter (UART), serial peripheral interface, (SPI), inter-integrated circuit (I2C), or universal serial bus (USB) and calls appropriate C function of RemoTI application framework [2]. The left side is the application processor. The RTIS module abstracts the same C function interface on host processor, enabling application to call C functions remotely instead of building network processor interface directly.

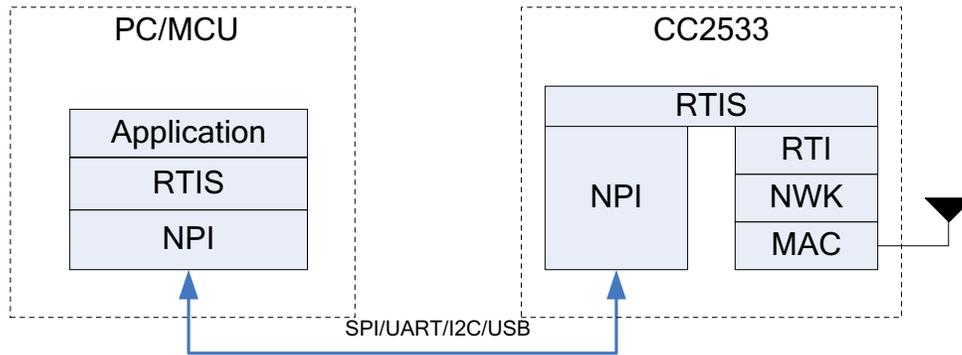


Figure 1. RemoTI Network Processor Component Architecture

1.3 Structure of RemoTI-Linux

RemoTI-Linux is a simple RF4CE target application on a linux host [3]. Figure 2 is an overview of the architecture. The linux software is split in two executables:

- An NPI server that creates an IP socket and connects it to the requested serial interface
- An NPI client that runs the RemoTI application

The NPI client sends all commands to the socket. The NPI server takes care of translating them to the RNP. The NPI client uses the RemoTI interface to drive the RNP.

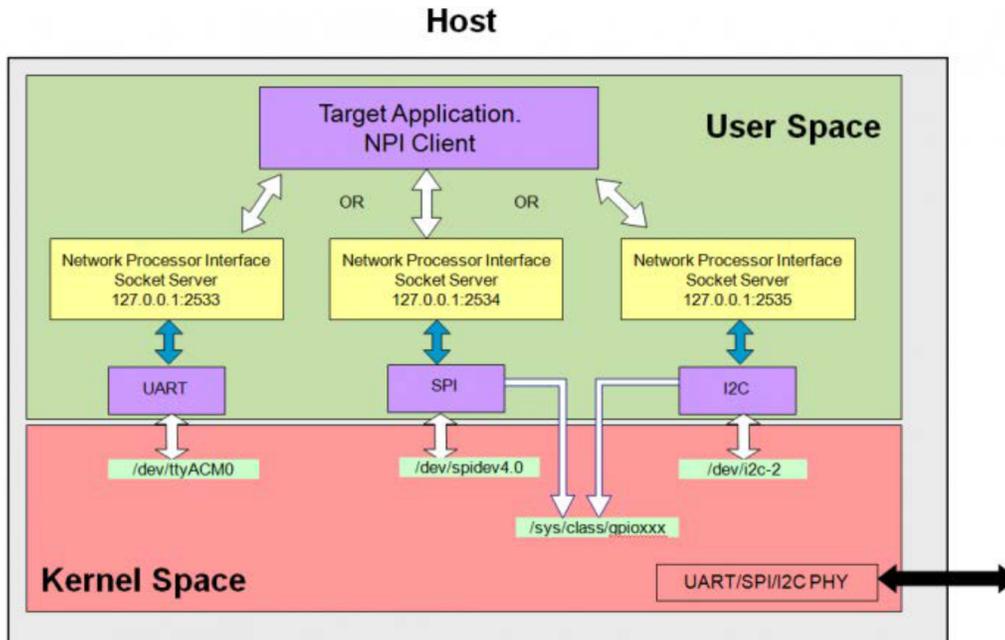


Figure 2. RemoTI-Linux Software Architecture

Figure 3 illustrates the application architecture. The simple application creates an RF4CE network. By default, it allows pairing with any RF4CE controller node using ZRC or ZID profile. Once paired, all data sent by the controller node will be displayed on the terminal.

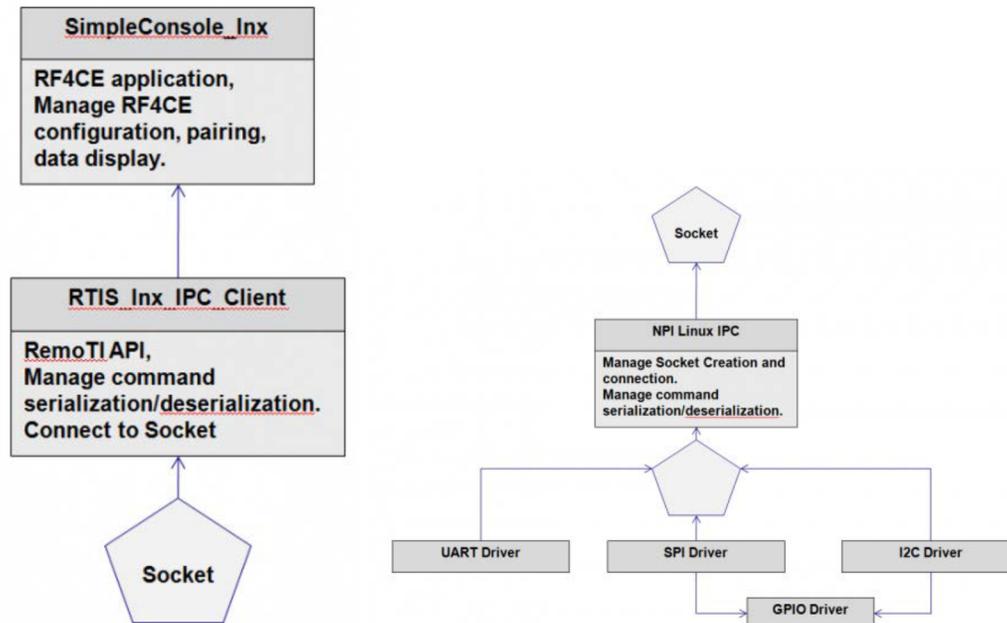


Figure 3. RemoTI-Linux Application Architecture

1.4 Network Enabling for NPI Client

Since the communication between NPI server and client adopts IP socket, NPI server and client can run on different devices. Figure 4 describes the scenario where NPI server is running inside the STB and NPI client is running inside a test PC. The STB and the test PC are under the same LAN via Ethernet on the production line. In this case, the NPI client including the test application can run on the test PC and test the CC253x on the actual STB. To enable this scenario, it just needs to set some configuration files. For more information, see Section 3.5.

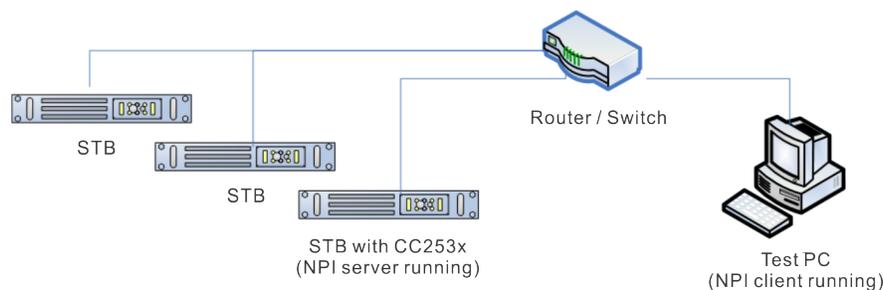


Figure 4. Network Enabling for NPI Client

1.5 Installation for RemoTI-Linux

The RemoTI-Linux is open source. It is available on Git HUB [4]. With the different configuration, RemoTI-Linux can perform as the RF4CE target or controller role. On the official release of RemoTI-Linux, there is a "SIMPLE_Inx_x86_client" application project. You can base your project on this, then modify the application for your own manufacturing usage.

2 Implement Test Program

The environment of manufacturing line is different from the RND working space. The equipments are limited, much noise exists and products need to be verified on time. Besides, some customers make target only or controller only. The RemoTI-Linux can simulate target or controller function. Customers could utilize devices of the same hardware to test transmitter and receiver function. [Figure 5](#) shows the setup for the implementation environment. Two CC2533 kits are connected to one laptop that runs Linux OS: EM and the RemoTI Target Board. One kit acts as controller, the other acts as target. Multiple instances of the client application can be run simultaneously, preferably in different sessions in separate terminals. For the setup with one controller and one target, two instances must run. The only difference is that the configuration file is used for controller and target.



Figure 5. The setup for the Test Program Implementation

[Example 1](#) lists the default functions supported by RemoTI-Linux. The latest RemoTI-Linux on the GitHub might be a little different. RemoTI provide many APIs [\[7\]](#). This section utilizes some APIs to implement useful features for testing RF4CE products.

Example 1. Default Init Menu list of RemoTI-Linux**Init MENU:**

1. Toggle Target/Controller
2. Set Node Capabilities
3. Supported Profiles
4. Supported Devices
5. Supported Target Types
6. Vendor ID
7. Vendor String
8. Toggle User String
9. User String
- i- Initialize without configuration. (Restore from NV)
- g- Get current configuration from RNP
- l- Show configuration. Note! Not Necessarily the One Written To RNP
- s- Apply Configuration and Move On To Application
- r- Back To Application, Do Not Apply Changes
- c- Start As Controller (default settings)
- t- Start As Target (default settings)

Example 2 provides an example of the modified Menu with code snips. Here, one fakePairing folder is created on "Projects\tools\LinuxHost\application\fakePairing". The NPI client is called FPA_Inx_x86_client.

Example 2. Updated Main Menu List

Init MENU:

- q- exit
- 0- Config
- 1- Pairing
- 2- Unpairing
- 7- Send Data
- 8- Clear Pairing Table
- 9- Display Pairing Table
- t- Toggle __DEBUG_TIME_ACTIVE on Server
- y- Toggle __BIG_DEBUG on Server
- c- Toggle simple data display
- a- Check States
- g- Get MAC Channel
- r- Reset RNP
- m- Show This Menu
- f- Enter RF Test Mode
- k- Configure Fake Pairing Table

Example 3. RF Test Mode Menu List

RF Test Mode MENU:

- r- Return to Main Menu
- 1- Perform Tx Mode 0: unmodulated carrier
- 2- Perform Tx Mode 1: pseudo-random data
- 3- Perform Rx Mode
- 4- Perform PER Test
- 5- Perform PER2 by SmartRFStudio
- 6- STOP PER2 by SmartRFStudio
- c- Set Channel
- p- Set TxPower
- n- Set PER Counter|- List Current Channel, TxPower
- m- Show This Menu

2.1 Fake Pairing

Pairing is the first step to connect the RF4CE target and controller. But, it may take some time in a noisy environment. It would be very helpful if the pairing tables between DUTs (target and remote) could be configured by programming. Since the pairing process in this case is not negotiated through RF packets, it is called “Fake Pairing”.

For RemoTI source code, the structure of “rcnNwkPairingEntry_t” stores the pairing table of the peer. [Table 2](#) lists an example to configure the pairing table on target and controller. Some elements (for example, panID, securityKey) need to be the same on both sides, while others must mirror each other (for example, NWK Address versus SRC NWK Address).

Table 2. Configuration for Fake Pairing on Target and Controller

Element of Pairing Table, rcnNwkPairingEntry_t	on Target	on Controller
	[PAIRING_INFO]	[PAIRING_INFO]
pairingRef	PAIRINGREF=0X00	PAIRINGREF=0X00
devTypeList[]	DEVTYPE0=0x01	DEVTYPE0=0x02
	DEVTYPE1=0x00	DEVTYPE1=0x00
	DEVTYPE2=0x00	DEVTYPE2=0x00
ieeeAddress	IEEE=00 12 4B 00 12 34 01 00	IEEE=00 12 4B 00 12 34 00 01
nwkAddress	NWKADDR=0xAAAA	NWKADDR=0xB BBB
srcNwkAddress	SRCNWKADDR=0xB BBB	SRCNWKADDR=0xAAAA
panId	PANID=0x1234	PANID=0x1234
recCapabilities	RECCAP=0x0C	RECCAP=0x0F
recFrameCounter	RECFRAMECOUNTER=0x00000000	RECFRAMECOUNTER=0x00000000
vendorIdentifier	VENDORID=0x0007	VENDORID=0x0007
securityKeyValid	KEY_VALID=1	KEY_VALID=1
securityKey[]	SECKEY=01 01 01 01 01 02 02 02 02 02 02 02 01 01 01 01	SECKEY=01 01 01 01 01 02 02 02 02 02 02 02 01 01 01 01
profileDiscs	PROFILE_DISC=0x02	PROFILE_DISC=0x02

[Table 3](#) lists the printout on target and controller.

Table 3. Pairing Table After Fake Pairing on Target and Controller

on Target	on Controller
*****	*****
* Max number of pairing entries: 10	* Max number of pairing entries: 10
*****	*****
* Pairing Index: 0x00	* Pairing Index: 0x00
* SRC NWK Address: 0xB BBB	* SRC NWK Address: 0xAAAA
* Logical Channel: 0x0F	* Logical Channel: 0x19
* IEEE Address: 00:12:4B:00:12:34:01:00	* IEEE Address: 00:12:4B:00:12:34:00:01
* PAN Id: 0x1234	* PAN Id: 0x1234
* NWK Address: 0xAAAA	* NWK Address: 0xB BBB
* Rec Capabilities: 0x0C	* Rec Capabilities: 0x0F
* Security Key Valid: 0x01	* Security Key Valid: 0x01
* Vendor Identifier: 0x0007	* Vendor Identifier: 0x0007
* Device Type List: [0x01, 0x00, 0x00]	* Device Type List: [0x02, 0x00, 0x00]
* Received Frame Counter: 0x00000000 (0)	* Received Frame Counter: 0x00000000 (0)
* Profiles Discovered: 0x0002	* Profiles Discovered: 0x0002

Afterwards, target and controller are in the same RF4CE network and can send data packets to each other. [Figure 6](#) shows the controller send data to the target. At the beginning, it sends data from channel 15. Because agility is enabled by default, it uses channel 20, 25. After that, it gets ACK on channel 25 and stays on that channel for the next packet.

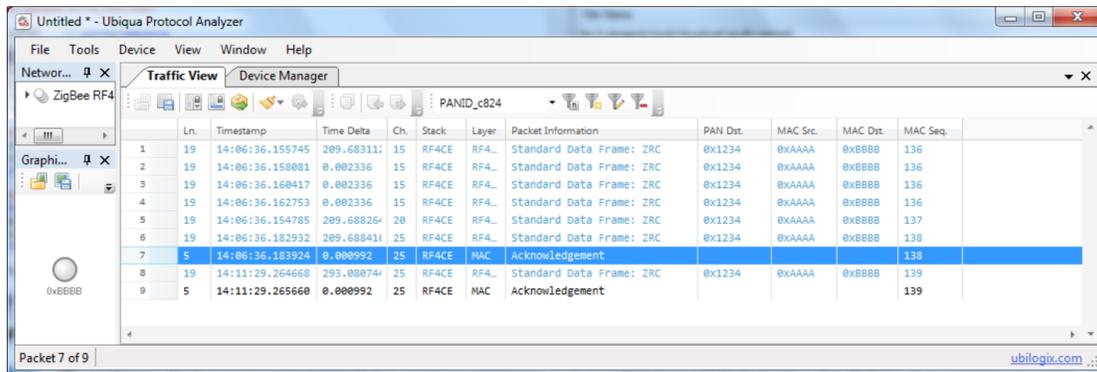


Figure 6. Controller Could Send Packet to Target After Doing Fake Pairing

2.2 Configured as Target or Controller

Customers might manufacture only target or controller devices. It would be helpful if their device could be configured to their peer types. For RemoTI source code, the structure of “appDevInfo_t” stores the device own information. [Table 4](#) provides an example to configure the device own information on target and controller.

Table 4. Configuration for Device Own Information on Target and Controller

Element of appDevInfo_t	on Target	on Controller
	[OWN_INFO]	[OWN_INFO]
	IEEE=00 12 4B 00 12 34 00 01	IEEE=00 12 4B 00 12 34 01 00
nodeCapabilities	NODECAP=0x0F	NODECAP=0x0C
tgtTypeList	TGTLIST=0x01 0x00 0x00 0x00 0x00 0x00	TGTLIST=0x02 0x00 0x00 0x00 0x00 0x00
appCapabilites	APPCAP=0x12	APPCAP=0x12
devTypeList	DEVTYPELIST=0x02 0x00 0x00	DEVTYPELIST=0x01 0x00 0x00
profileIdList	PROFILEIDLIST=0x01 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF	PROFILEIDLIST=0x01 0xFF 0xFF 0xFF 0xFF 0xFF
vendorId	VENDORID=0x0007	VENDORID=0x0007
vendorString	VENDORSTRING="TI_LPRF"	VENDORSTRING="TI_LPRF"

“nodeCapabilities” is the most important element. [Table 5](#) lists the bits definition [2]. For controller, bit0 and bit1 should be set as ‘0’.

Table 5. Definition of nodeCapabilities

Bit	Description
0	0 – Controller type
	1 – Target type
1	0 – Battery powered
	1 - AC mains powered
2	0 – Security incapable
	1 – Security capable
3	0 – Channel normalization incapable
	1 – Channel normalization capable
4 - 7	Reserved

2.3 Enable RF Test Modes

To verify the RF performance on manufactured products, devices need to enter specific test mode. RemoTI provides one API (RTI_TestModeReq) to place the radio in different test modes [2]. Table 6 lists the test modes supported by RTI_TestModeReq.

Table 6. Test Modes Supported by RTI_TestModeReq

Test Mode	Description
0x00	The device transmits unmodulated carrier at the specified frequency and with the specified transmit power.
0x01	The device will transmit pseudo-random data at the specified frequency and with the specified transmit power.
0x02	The device will have the radio placed in receive mode on the specified frequency.

Example code is shown in Example 4. The parameters of mode, txPower, and channel depend on test conditions. Disable agility when setting the channel on MainMenu to simplify the test.

Example 4. Example Code to Call RTI_TestModeReq

```
uint8 mode = 0x00; //tx unmodulated carrier
int8 txPower = 0; //0 dBm
uint8 channel = 15; // channel 15
RTI_TestModeReq(mode, txPower, channel );
}
```

2.4 Change RF Features

To meet some RF test cases, customers might need to change some RF parameters, (for example, RF channel, agility, discovery threshold). The example code is shown in Example 5.

About setting discovery threshold, the threshold value is unsigned (from 0 to 255). It is compared to the incoming discovery request link quality indicator (LQI), which is also an unsigned value (0 to 255, 255 meaning that it is a very strong or good signal). If the discovery request LQI is lower than the threshold, the request is ignored. With TI RF4CE kit, ARC and a dongle, if the threshold value is set to 205, they cannot pair when the distance between ARC and dongle is farther than 50 cm.

Example 5. Example Code to Change RF Parameters

```
// change device RF channel
{
  uint8 value=15; //depend on test case
  RTI_WriteItemEx(RTI_PROFILE_RTI, 0x61, 1, &value);
  //0x61: RTI_SA_ITEM_CURRENT_CHANNEL
}
// enable or disable agility
{
  uint8 value=1;
  RTI_WriteItemEx(RTI_PROFILE_RTI, 0x87, 1, &value);
  //0x87: RTI_SA_ITEM_AGILITY_ENABLE
}
// set discovery threshold
{
  uint8 value=205;
  RTI_WriteItemEx(RTI_PROFILE_RTI, 0x62, 1, &value);
  //0x62: RTI_SA_ITEM_DISCOVERY_LQI_THRESHOLD
}
```

2.5 Implement PER Test

PER test verifies the packet error rate between controller and target. Example code is provided in [Example 6](#). Controller call `RTI_SendDataReq()` is required to send each packet with ACK. If the target receives a valid packet, it would reply ACK. Then, DUT checks the status on callback function `RTI_SendDataCnf()`. After the number of successful transmission is received, the packet error rate could be calculated.

Example 6. Example Code for PER Test

```

void appProcessPER()
{
    //send 1st pkt
    RTI_SendDataReq(0, RTI_PROFILE_ZRC, 0, /*RTI_TX_OPTION_ACKNOWLEDGED*/4, 2, data);
    app_RF_test_data_s.Sent_cntr++;
    appRfTestMode = APP_RF_TEST_STATE_PER_RECEIVING;//update on RTI_SendDataCnf

    //add timeout mechanism
    app_RF_test_data_s.TMOUT = 0;    //clear timeout indication
    timeout = (app_RF_test_data_s.PER_CNTR / 10 ) * 5; //could be adjusted
    timer_start_timerEx(FPA_App_threadId, FAKEPAIRING_APP_EVT_RF_TMOUT, timeout);
    while((app_RF_test_data_s.Sent_cntr<app_RF_test_data_s.PER_CNTR) &&
        !(app_RF_test_data_s.TMOUT))

    //counter to be send
    {
        if(appRfTestMode == APP_RF_TEST_STATE_PER_SENDING)
        {
            RTI_SendDataReq(0, RTI_PROFILE_ZRC, 0, /*RTI_TX_OPTION_ACKNOWLEDGED*/4, 2, data)
                app_RF_test_data_s.Sent_cntr++;
                appRfTestMode = APP_RF_TEST_STATE_PER_RECEIVING;
                //update on RTI_SendDataCnf
        }
    }
    while((appRfTestMode == APP_RF_TEST_STATE_PER_RECEIVING) &&
        !(app_RF_test_data_s.TMOUT))
    {
        //wait the ack for the last packet
    }
    //calculate PER
    per =(1.0 - (app_RF_test_data_s.Rcv_cntr / app_RF_test_data_s.PER_CNTR))*100;
    printf("PER Test result = %2.1f %%\n", per);
    ...
}

void RTI_SendDataCnf(rStatus_t status)
{
    if (appState == AP_STATE_RF_TEST_MODE)    //under RF test mode
    {
        if(appRfTestMode == APP_RF_TEST_STATE_PER_RECEIVING)
        {
            if (status == RTI_SUCCESS)
                app_RF_test_data_s.Rcv_cntr++;
            printf("PER Test Rcv_cntr=%d\n",app_RF_test_data_s.Rcv_cntr);

            //appRfTestMode = APP_RF_TEST_STATE_PER_SENDING;
        }
        ...
        appRfTestMode = APP_RF_TEST_STATE_PER_SENDING; //toggle appRfTestMode
    }
}
    
```

3 Setup for RF Test

Using [Example 4](#) and [Example 6](#), this section introduces how to setup the test environment and perform RF tests. Continuous Tx, Rx and PER test is also introduced. Except PER test, one DUT (RemoTI Target Board with CC2533EM) is used (see [Figure 7](#)). DUT connects to the spectrum analyzer with RF cable. In addition, it connects to one Linux laptop with a USB cable. The laptop executes the test program, then DUT emits an RF signal to the spectrum analyzer.

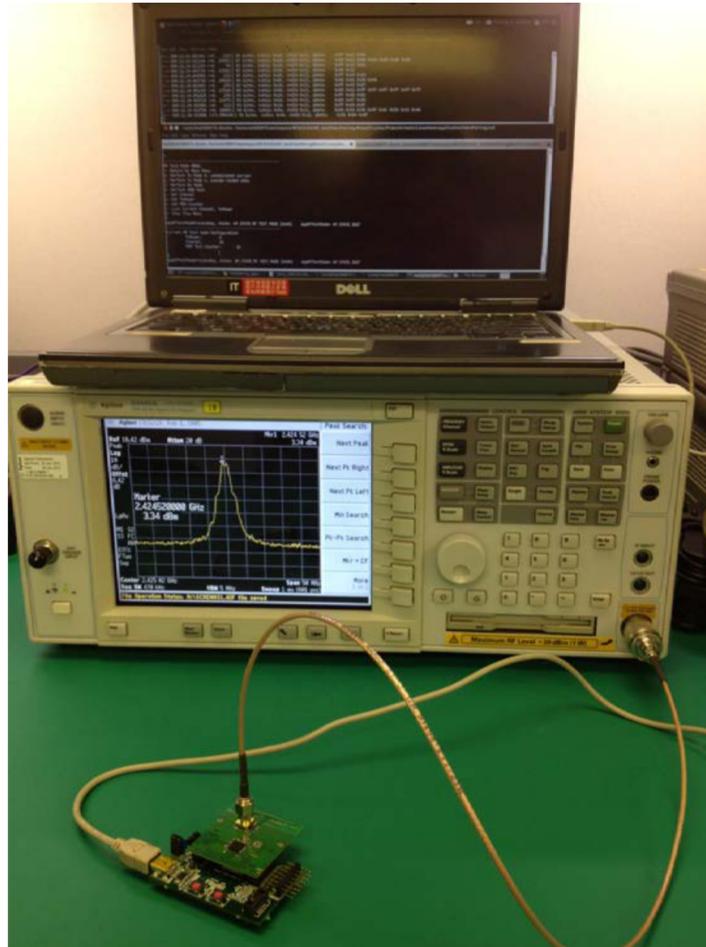


Figure 7. Setup for RF Test

3.1 Perform Tx Test Mode0

Tx test Mode0 verifies the RF quality while the DUT emits the unmodulated signal. To perform Tx test Mode0, open two terminals on a Linux laptop (see [Figure 8](#)) and run the following steps:

1. On one terminal (the upper one is shown in [Figure 8](#)), enter NPI server directory, "RemoTI-Linux/Projects/tools/LinuxHost/out".
2. Launch NPI server by running `./NPI_Inx_x86_server`.
3. On another terminal, enter NPI client directory, "RemoTI-Linux/Projects/tools/LinuxHost/application/fakePairing/out".
4. Launch NPI client by running `./FPA_Inx_x86_client -c ../defCtl.cfg`.
5. Type 'f' to select "Enter RF Test Mode".
6. Type '1' to select "Perform Tx Mode 0: unmodulated carrier".
7. Then, DUT emits an RF signal to the spectrum (see [Figure 9](#)).

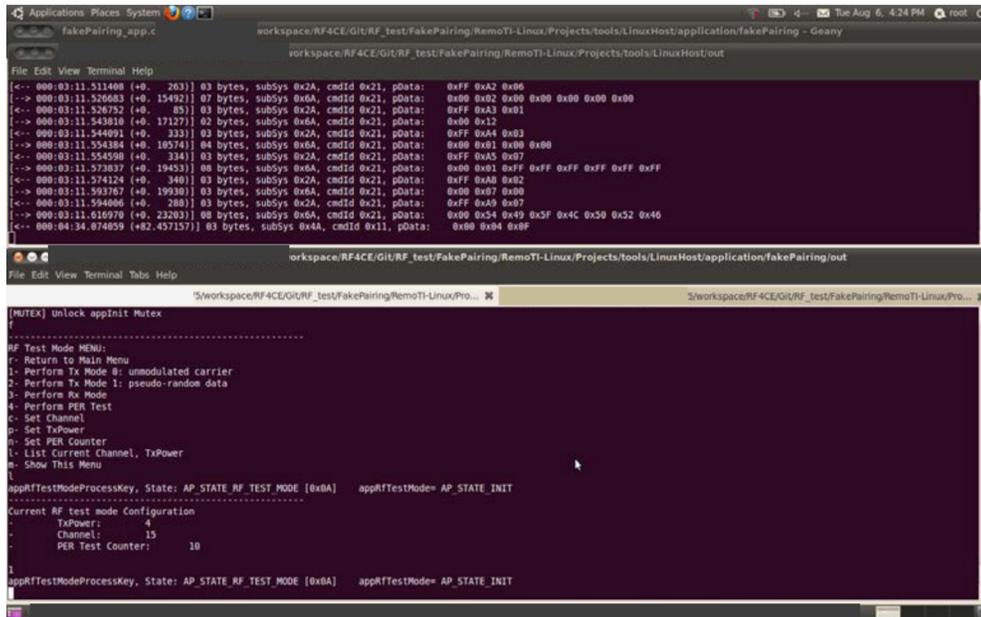


Figure 8. Tx Test Mode0 Launch on Linux Laptop

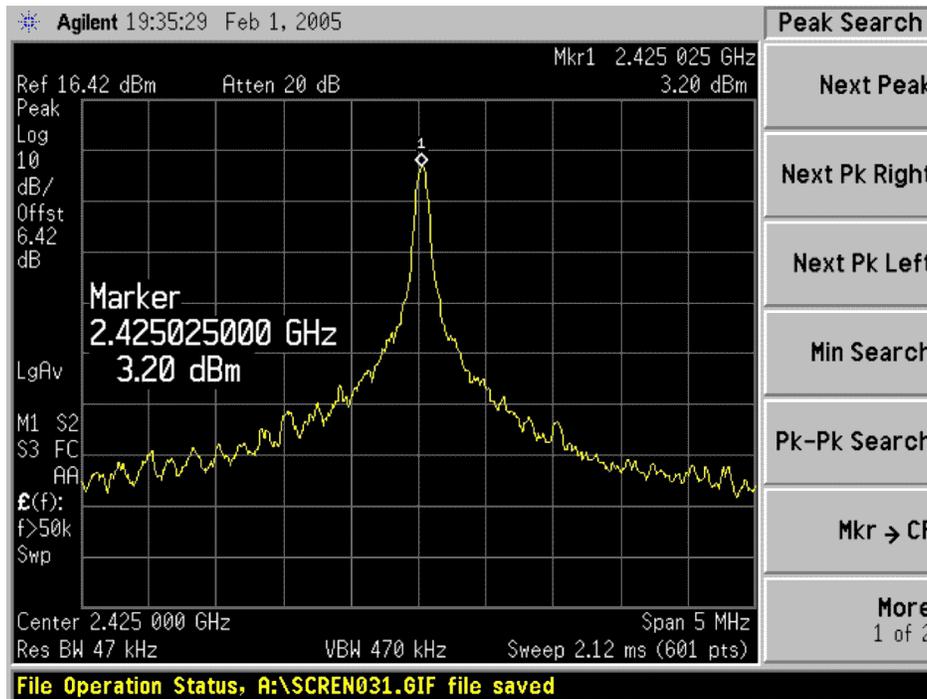


Figure 9. Tx Test Mode0 Perform on Spectrum Analyzer

3.2 Perform Tx Test Mode1

Tx test Mode1 verifies the RF quality while the DUT emits the modulated signal. To perform Tx test Mode1, open two terminals on a Linux laptop (see [Figure 10](#)) and run the following steps:

1. On one terminal, enter NPI server directory, “RemoTI-Linux/Projects/tools/LinuxHost/out”.
2. Launch NPI server by running “./NPI_Inx_x86_server”.
3. On another terminal, enter NPI client directory, “RemoTI-Linux/Projects/tools/LinuxHost/application/fakePairing/out”.
4. Launch NPI client by running “./FPA_Inx_x86_client -c ../defCtl.cfg”.
5. Type ‘f’ to select “Enter RF Test Mode”.
6. Type ‘2’ to select “Perform Tx Mode 0: unmodulated carrier”.
7. Then, DUT emits an RF signal (see [Figure 11](#)).

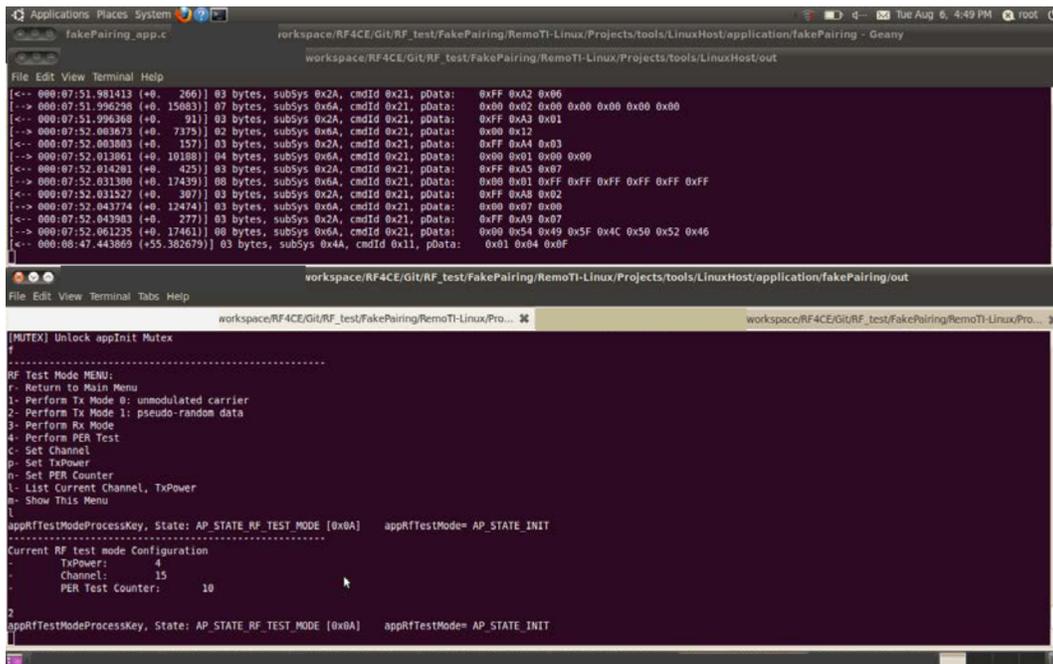


Figure 10. Tx test Mode1 Launch on Linux Laptop

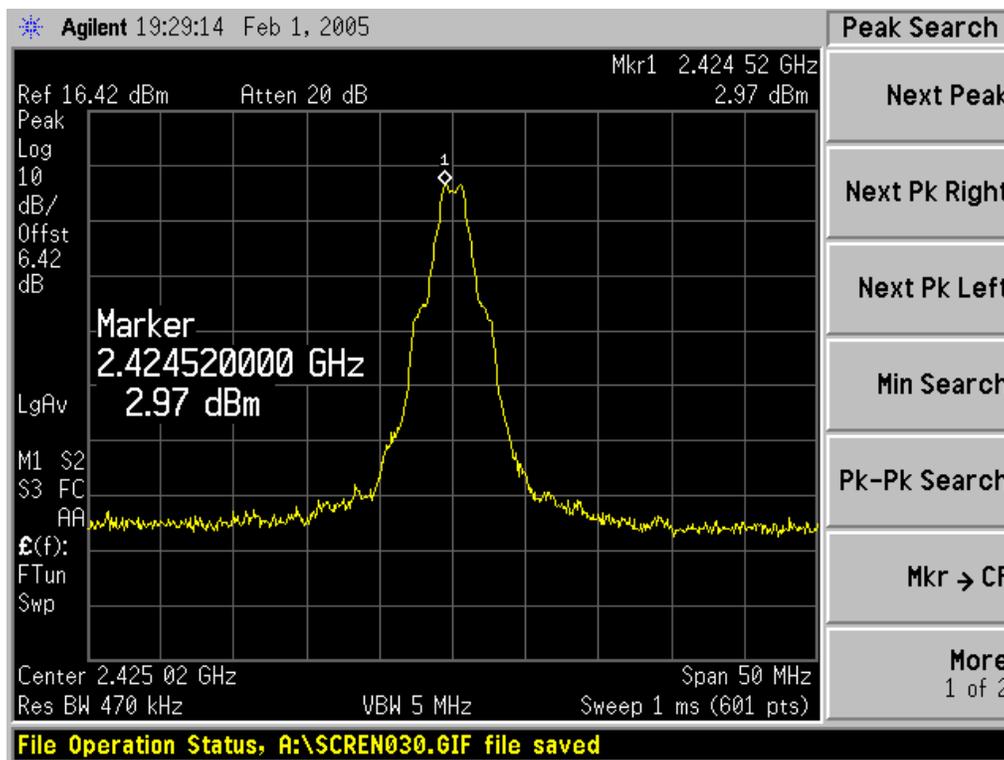


Figure 11. Tx Test Mode1 Perform on Spectrum Analyzer

3.3 Perform Rx Test

Rx test verifies whether DUT enters receiving mode and does not emit any RF signal on the specified frequency and its second harmonic. To perform Rx test, open two terminals on a Linux laptop (see [Figure 12](#)) and run the following steps:

1. On one terminal, enter NPI server directory, "RemoTI-Linux/ Projects/tools/LinuxHost/out.
2. Launch NPI server by running "./NPI_Inx_x86_server".
3. On another terminal, enter NPI client directory, "RemoTI-Linux/ Projects/tools/LinuxHost/application/fakePairing/out".
4. Launch NPI client by running "./FPA_Inx_x86_client -c ../defCtl.cfg".
5. Type 'f' to select "Enter RF Test Mode".
6. Type '3' to select "Perform Rx Mode".
7. Modify the spectrum with wider frequency range similar to [Figure 13](#). Verify whether or not DUT emits an RF signal on the specified frequency and its second harmonic.

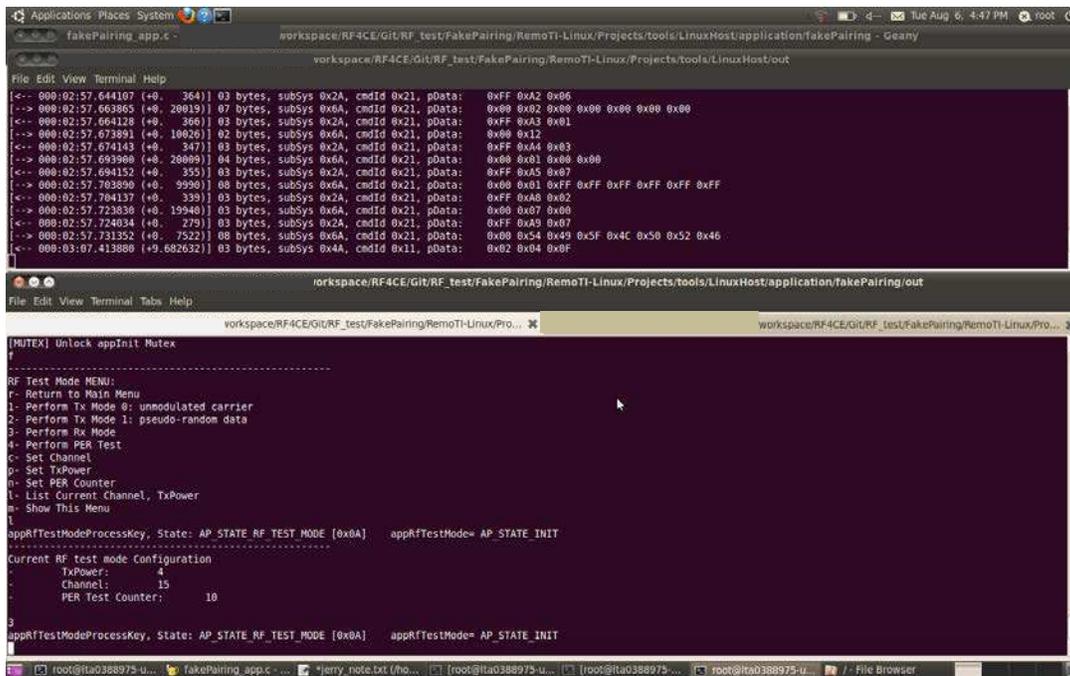


Figure 12. Rx Test Launch on Linux Laptop

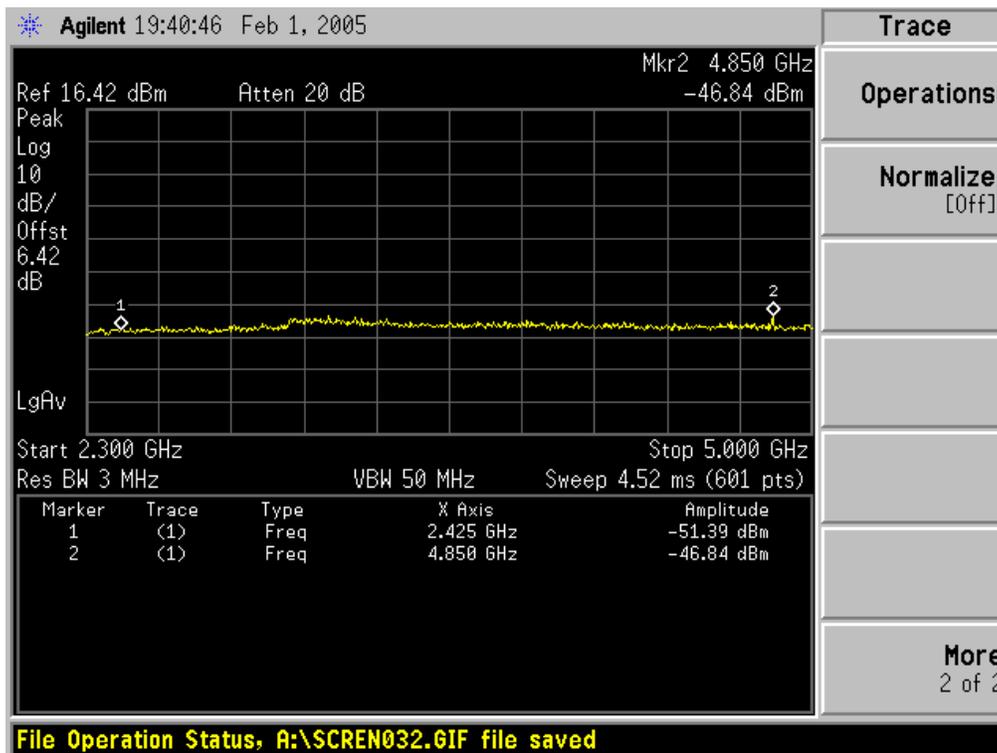


Figure 13. Rx Test Perform on Spectrum Analyzer

3.4 Perform PER Test

PER test verifies the packet error rate between the controller and target. The setup is similar to Figure 5. On a Linux laptop, two NPI server configuration files are needed for the controller and target; Table 7 lists the difference. Run two NPI server instances for controller and target. Open four terminals (see Figure 14) and run the following steps:

1. On Terminal1 (the upper left one is on Figure 14), enter NPI server directory, "RemoTI-Linux/Projects/tools/LinuxHost/out".
2. Launch NPI server for controller, "./NPI_Inx_x86_server ./RemoTI_RNP.cfg".
3. On Terminal2 (lower left), enter NPI client directory, "RemoTI-Linux/Projects/tools/LinuxHost/application/fakePairing/out".
4. Launch NPI client for controller, "./FPA_Inx_x86_client -c ../defCtl.cfg".
5. On Terminal3 (upper right), enter NPI server directory, "RemoTI-Linux/Projects/tools/LinuxHost/out".
6. Launch NPI server for target, "./NPI_Inx_x86_server ./RemoTI_RNP_Tgt.cfg".
7. On Terminal4, enter NPI client directory, "RemoTI-Linux/Projects/tools/LinuxHost/application/fakePairing/out".
8. Launch NPI client for target, "./FPA_Inx_x86_client -c ../defTgt.cfg".
9. On Terminal2, type 'f' to select "Enter RF Test Mode".
10. Type '4' to select "Perform PER Test". After finishing test, it would print out the PER value. Figure 15 shows the sniff log. For good condition, after controller sends each packet, target would reply ACK.

Table 7. Server Configuration Difference on Target and Controller

on Target	on Controller
[PORT]	[PORT]
port=2533	port=2530
devPath="/dev/ttyUSB1" ;	devPath="/dev/ttyUSB0" ;
[LOG]	[LOG]
log="/NpiLnxLog_2.txt"	log="/NpiLnxLog.txt"

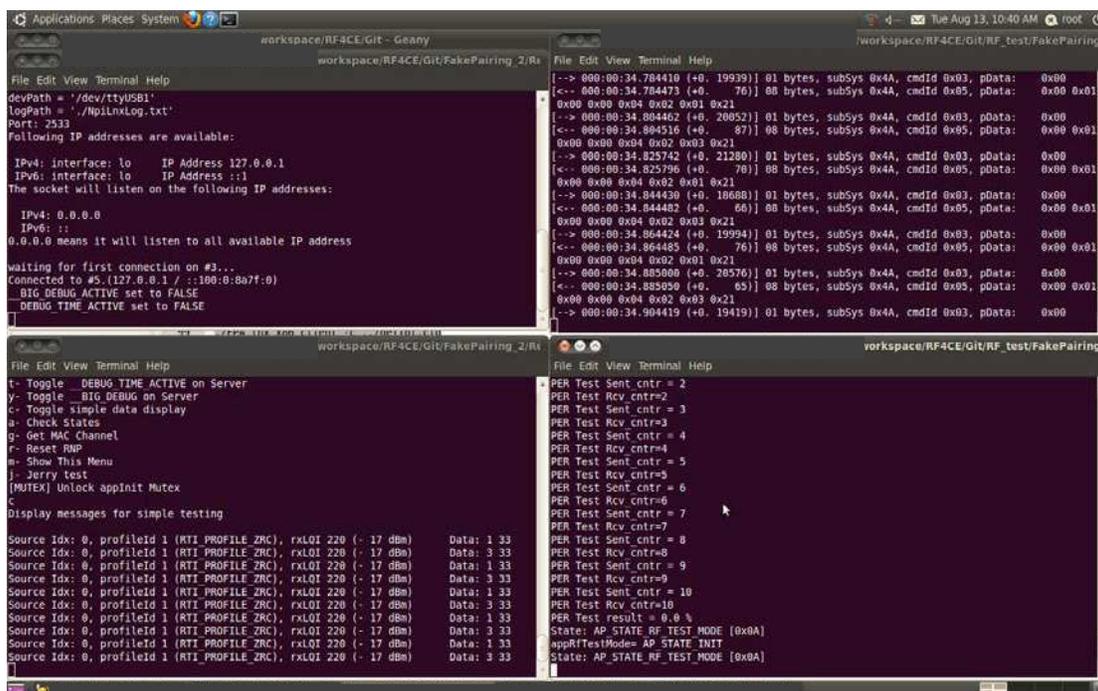


Figure 14. PER Test Launch on Linux Laptop

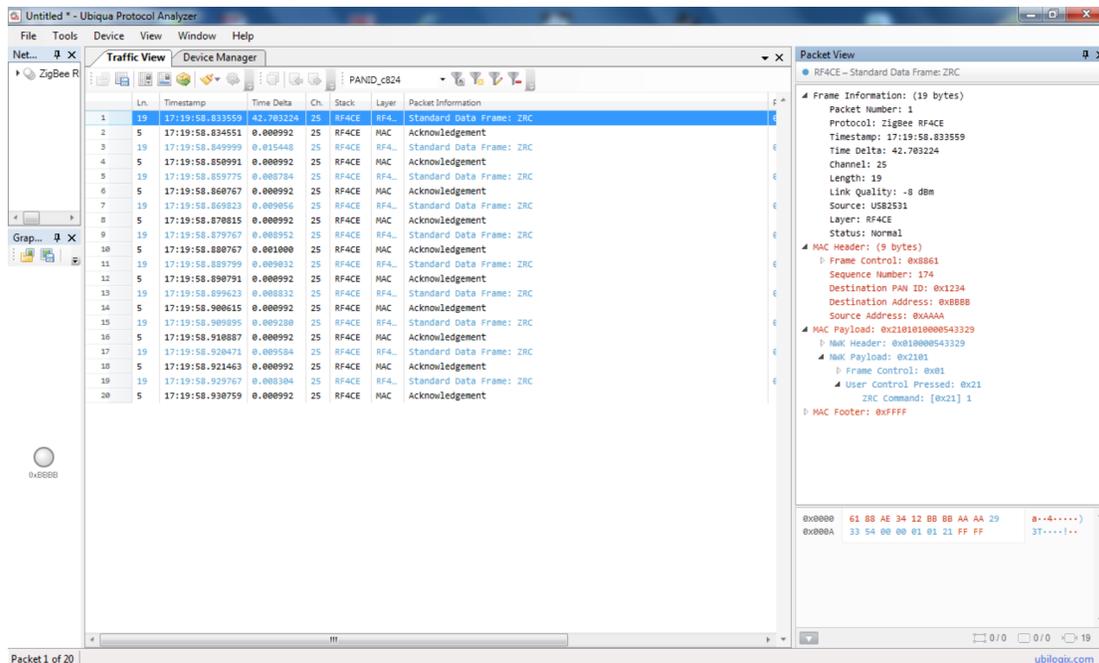


Figure 15. Sniff Log While Performing PER Test

3.5 Enable NPI Client on Network Test

As described in Section 1.4, the NPI server and client can run on different devices. An example is shown in Figure 16. Before performing the RF test, the NPI client needs to understand the physical IP address of the NPI server and fill to the IP_ADDR of the NPI client's configuration file. If the NPI client runs as control role, defCtl.cfg is used. Otherwise, defTgt.cfg is used for target role.

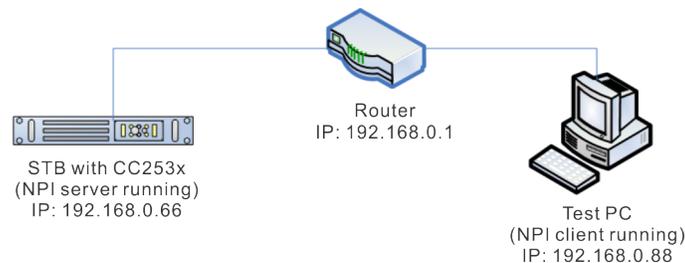


Figure 16. NPI Server and Client Run on Different Devices

Example 7. Modification on NPI Client

```

@defCtl.cfg
[DEBUG_MODE]
BIG=0 ; (1: ON, 0: OFF)
TIMER=1 ; (1: ON, 0: OFF)
APP=1 ; (1: ON, 0: OFF, 2: Verbose ON)

[BASE_SETTINGS]
IP_ADDR=192.168.0.66
@defTgt.cfg
[DEBUG_MODE]
BIG=0
TIMER=0
APP=1
CLIENT=0

[BASE_SETTINGS]
IP_ADDR=192.168.0.66

```

3.6 Perform PER Test With SmartRF Studio

Besides [Section 3.4](#), there is another option for PER test. The setup is shown in [Figure 17](#). The SmartRF Studio runs PER Tx side, and the RNP runs PER Rx side.

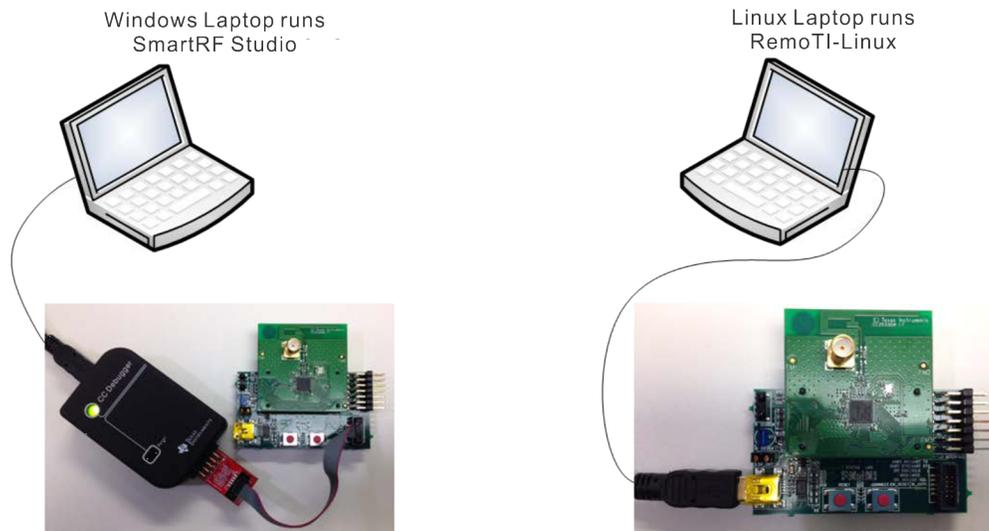


Figure 17. Setup for PER Test With SmartRF Studio

The RF4CE stack already embeds a test function, `RTI_TestModeReq(RTI_TEST_MODE_RX_AT_FREQ,..)`. After this API is called, the RF4CE stack starts to count the received MAC frames. If the sender finishes transmitting test packets, you can call `RTI_TestRxCounterGetReq(1)` to get the counter of the received MAC frames. If you know in advance the number of packets sent and the time needed to send them, then you can have an estimation of the PER.

Use the SmartRF Studio to send test packets; it takes around 8s to send 100 packets.

Note that the network layer is still linked on the MAC/RF PHY, which means it can change RF parameters. To avoid this, disable the following:

- Standby mode : `RTI_StandbyReq(RTI_STANDBY_OFF)`;
- Frequency agility

Below are a list of the test steps:

1. On the RNP side, set test channel and launch test mode (see [Example 8](#), call `appProcessPER2Start()`).
2. On SmartRF Studio, configure the broadcast packet (see [Figure 18](#)). Then start to send packets.
3. After all packets are sent, on RNP side, stop the test mode and calculate the PER (see [Example 8](#), call `appProcessPER2Stop()`). To simplify the test, disable agility when setting the channel on MainMenu.

Example 8. Example Code for PER Test With SmartRF Studio

```

//To reset the counter and start the MAC layer in reception (here one channel 15):
RTI_TestModeReq( RTI_TEST_MODE_RX_AT_FREQ, 0, 15);
void appProcessPER2Start()

{
    uint8 value=0;

    printf("Note, the number of expected packet is 100. To modify this, need to modify the
    definitoin of PKT_exp inside the code\n");

    //the network layer is still linked to the MAC/PHY. Which means it can change RF parameters. To
    avoid that disable the following:
    // - Standby mode
    // - Be sure the RF never turn off by application
    //RTI_StandbyReq(RTI_STANDBY_OFF); //disable standby mode
    //RTI_StandbyReq(RTI_STANDBY_ON);
    //To reset the counter (rtiRxConter) and start MAC Layer in reception
    RTI_TestModeReq(RTI_TEST_MODE_RX_AT_FREQ,0,app_RF_test_data_s.Channel);
}

#define PKT_exp 100 //set expected packet to 100
void appProcessPER2Stop()
{
    double per=0.0;
    uint8 res=0;
    double pkt_exp = 1.0 * PKT_exp;
    appRfTestMode = APP_RF_TEST_STATE_INIT; //set state
    res = RTI_TestRxCounterGetReq(1); //get counter, then reset it.
    printf("---- receive counter = %d ----\n", res);
    //calculate PER
    per = (1.0 - (res / pkt_exp))*100;
    printf("---- PER2 Test result = %10.1f %% ----\n", per);
}

```

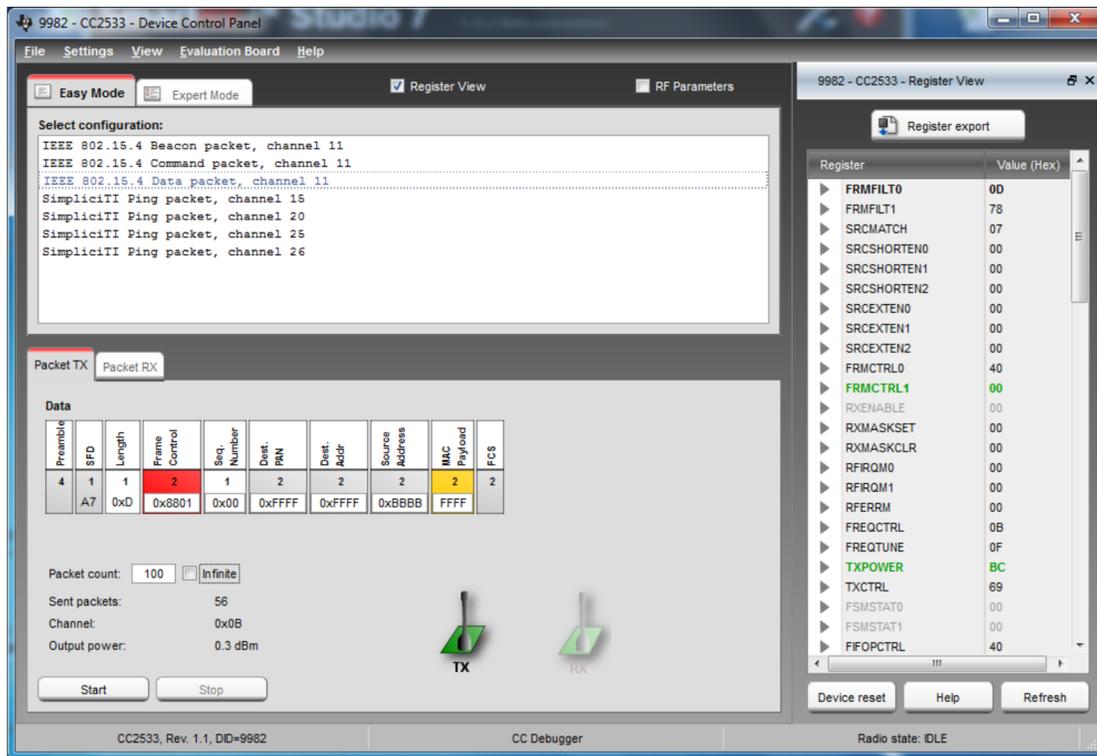


Figure 18. Configuration on SmartRF Studio

Since the process in [Example 8](#) uses broadcast packets, check that another RF4CE device is not transmitting, otherwise, the PER will be erroneous. The other possibility is to use unicast packet. On SmartRF Studio, “Source PAN” and “Source Address” need to match the values on the RNP target side (PANID and SHORT_ADDRESS).

4 RemoTI Mass Production Test

To simplify the RF4CE RF test, the RemoTI Mass Production Test [\[8\]](#) [\[9\]](#) was provided, which utilizes the PC application to analyze the test result. The test flow is depicted in [Figure 19](#). First, prepare the configuration on the PC side and launch the RemoTI-MassProduction-Test application. Then, execute the massProduction client on the STB side. The test packets automatically transmit. Finally, the PC application generates the test result (see [Figure 20](#)).

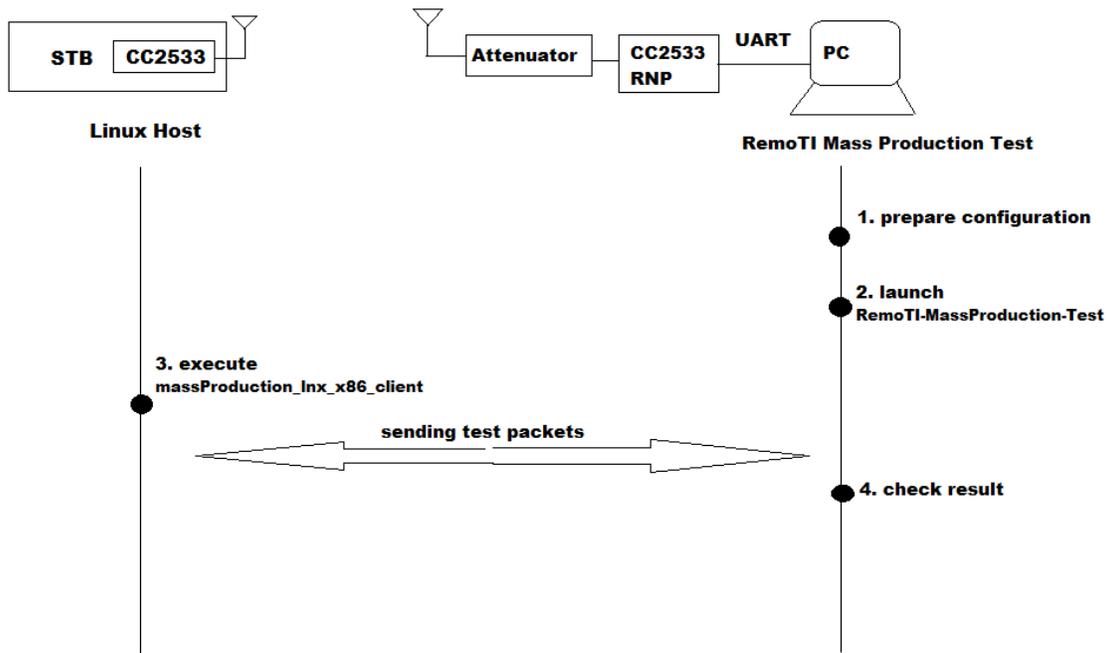


Figure 19. Execute massProduction_Inx_x86_client

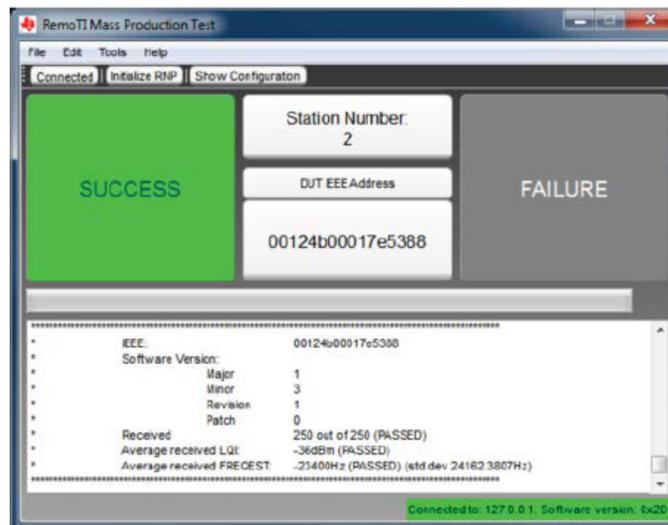


Figure 20. Result of RemoTI Mass Production Test

5 Summary

This application report illustrates the methods for “Fake Pairing” and provides some RF test cases, which can verify the manufactured devices. It also helps customers to manufacture RF4CE products easier.

6 References

1. *RemoTI Host Processor Sample Application and Porting Guide* ([SWRA259](#))
2. *RemoTI API* ([SWRA268](#))
3. [RF4CE, simple linux target application wiki](#)
4. [TI-LPRF-Software/RemoTI-Linux](#)
5. *RemoTI Network Processor Developer's Guide* ([SWRU223](#))
6. [RF4CE, simple linux target application wiki](#)
7. *RemoTI Network Processor Interface Specification API* ([SWRA271](#))
8. *NPI Server* (SWRA450)
9. *RemoTI Mass Production Test* (SWRA451)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com