

# Realtek

# Bluetooth

# MP Flow

(windows)

Date: 2014/9/2

**This document is subject to change without notice. The document contains Realtek confidential information and must not be disclosed to any third party without appropriate NDA.**

# **1. Overview**

This document is used to introduce MP (Mass Production) test tool for Realtek Bluetooth chip series. Customers should comply with the steps and requirements under this document. Contact Realtek Bluetooth FAE if any problem arises in the use of MP flow.

## 2. Prerequisites

Before MP tool startup, below items should be checked:

- a) The Bluetooth Chip embedded is provided by Realtek;

Realtek Bluetooth Chip Number
RTL8723B series
RTL8761A series
RTL8821A series

- b) The Bluetooth on target production is available and works normally;

### 2.1 Files

MP tool package is provided to customers in binary format:

RtlBluetoothMP.dll	MP library
RtlBluetoothMP.h	MP header
BTPatchCode\ Patch_rtl8723a.bin	rtl8723a fw patch
BTPatchCode\ Patch_rtl8723b.bin	rtl8723b fw patch
BTPatchCode\ Patch_rtl8821a.bin	rtl8821a fw patch
BTPatchCode\ Patch_rtl8761a.bin	rtl8761a fw patch

## 3. MP function Usage

### 3.1 BTMPAPI\_BuildInterfaceVendor

```
int BTMPAPI_BuildInterfaceVendor(  
    BASE_INTERFACE_MODULE **ppBaseInterface,  
    BASE_INTERFACE_MODULE *pBaseInterfaceModuleMemory,  
    //Parmater  
    unsigned int InterfaceType,  
    unsigned char PortNo,  
    unsigned long Baudrate,  
    //basic fuction  
    BASE_FP_OPEN Open,  
    BASE_FP_SEND Send,  
    BASE_FP_RECV Recv,  
    BASE_FP_CLOSE Close,  
    BASE_FP_WAIT_MS WaitMs  
);
```

This function provides vender to implement how to open interface, close interface, send hci command, receive hci event, and wait mini-second functions by themselves.

### 3.2 BTMPAPI\_BuildInterfaceRTK

```
int BTMPAPI_BuildInterfaceRTK(  
    BASE_INTERFACE_MODULE **ppBaseInterface,  
    BASE_INTERFACE_MODULE *pBaseInterfaceModuleMemory,  
    unsigned int InterfaceType,  
    unsigned char PortNo,  
    unsigned long Baudrate  
);
```

This function uses Realtek default functions to open, close, send, receive and wait. Therefore, you could select one between BuildInterfaceVendor and BuildInterfaceRTK.

### 3.3 BTMPAPI\_BuildBluetoothModule

The Bluetooth Module includes four main functions that are “DownloadPatchCode”, “UpDataParameter”, “ActionControlExcute”, and “ActionReport”.

#### 3.3.1 DownloadPatchCode

```
typedef int
```

```

(*BT_DLL_MODULE_FP_ACTION_DLFW)(
    BASE_BTMPDLL_MODULE *pDLLBtBaseModule,
    char *pPatchcode,
    int patchLength,
    int Mode
);

```

This function is used to download fw patch. Before downloading fw patch, vendor should read different patch files according to different BT chips.

### 3.3.2 UpDataParameter

typedef int

```

(*BT_DLL_MODULE_FP_UPDATA_PARAMETER)(
    BASE_BTMPDLL_MODULE *pDLLBtBaseModule,
    BT_PARAMETER *pParam
);

```

This function is used to pass “BT\_PARAMETER” data structure to RtlBluetoothMP.dll. However, BT hardware registers don’t be changed until “ActionControlExcute” is called. The following tables show more detail information about each parameter.

Name	Value Range	Table Index
ParameterIndex	0~24	See <b>BT_ACTION_CONTROL</b>
mPGRawData	Row data	None
mChannelNumber	0~78	None
mPacketType	0~8	<b>See PKT_TYPE</b>
mTxGainIndex	1~7	None
mTxGainValue	Realtek define	Realtek define
mTxPacketCount (for packet tx)	0~0xFFF	0 : infinite Tx packet count
mPayloadType	0~7	<b>See PAYLOAD_TYPE</b>
mPacketHeader	0x0~0x3FFFF	None
mWhiteningCoeffValue	0x00~0x7F	0x00~0x7F : Enable Whitening 0x80~0xFF: Disable Whitening
mTxDAC	Realtek define	Realtek define
mHitTarget	6 bytes	None
bHoppingFixChannel (for Hopping mode)	0 : Disable 1 : Enable Fix Channel	None
Rtl8761Xtal	0~0x3F	None

**Table BT\_PARAM**

The packet types are defined in Table PKT\_TYPE:

NAME	INDEX	Payload Length in bits
BT_PKT_DH1	0	0~27*8
BT_PKT_DH3	1	0~183*8
BT_PKT_DH5	2	0~339*8
BT_PKT_2DH1	3	0~54*8
BT_PKT_2DH3	4	0~367*8
BT_PKT_2DH5	5	0~679*8
BT_PKT_3DH1	6	0~83*8
BT_PKT_3DH3	7	0~552*8
BT_PKT_3DH5	8	0~1021*8
BT_PKT_LE	9	0~39*8

**Table PKT\_TYPE**

The payload types are defined in Table PAYLOAD\_TYPE.

NAME	INDEX
BT_PAYLOAD_TYPE_ALL0	0
BT_PAYLOAD_TYPE_ALL1	1
BT_PAYLOAD_TYPE_0101	2
BT_PAYLOAD_TYPE_1010	3
BT_PAYLOAD_TYPE_0x0_0xF	4
BT_PAYLOAD_TYPE_0000_1111	5
BT_PAYLOAD_TYPE_1111_0000	6
BT_PAYLOAD_TYPE_PRBS9	7

**Table PAYLOAD\_TYPE**

Parameter Index Name	Index No.
HCI_RESET	0
TEST_MODE_ENABLE	1
WRITE_EFUSE_DATA	2
SET_TX_GAIN_TABLE	3
SET_TX_DAC_TABLE	4
SET_DEFAULT_TX_GAIN_TABLE	5
SET_DEFAULT_TX_DAC_TABLE	6
SET_POWER_GAIN_INDEX	7
SET_POWER_GAIN	8
SET_POWER_DAC	9
SET_XTAL	10
REPORT_CLEAR	11
PACKET_TX_START	12
PACKET_TX_UPDATE	13

PACKET_TX_STOP	14
CONTINUE_TX_START	15
CONTINUE_TX_UPDATE	16
CONTINUE_TX_STOP	17
PACKET_RX_START	18
PACKET_RX_UPDATE	19
PACKET_RX_STOP	20
HOPPING_DWELL_TIME	21
LE_TX_DUT_TEST_CMD	22
LE_RX_DUT_TEST_CMD	23
LE_DUT_TEST_END_CMD	24
READ_EFUSE_DATA	25

**Table BT\_ACTION\_CONTROL**

- *TxPacketCount parameter*

TxPacketCount is used to set how many TX packets will be transmitted. The range of TxPacketCount is from 0 to 0x3FFFF. However, if TxPacketCount equals to “0”, it means infinite TxPacketCounts. If users want to keep sending packet tx, “TxPacketCount” should be “0” and “bt\_mp\_Report 1” should be executed every 200~1000ms. To execute “bt\_mp\_Report 1” periodically can trigger bt hardware to send packets.

- *WhiteningCoeffValue parameter*

The range of WhiteningCoeffValue is from 0 to 0x7F. However, if WhiteningCoeffValue is larger than “0x7F”, it means to disable whitening.

- *TxGainIndex and TxGainValue parameter*

TxGainIndex and TxGainValue are both to set TX power gain. However, TxGainIndex have higher priority. The range of TxGainIndex is from 1 to 7. Only when TxGainIndex is out of range, TxGainValue is meaningful.

### 3.3.3 ActionControlExcute

```
typedef int
(*BT_DLL_MODULE_FP_ACTION_CONTROLEXCUTE)(
    BASE_BTMPDLL_MODULE *pDLLBtBaseModule
);
```

According to “ParameterIndex” in the “BT\_PARAMETER” structure, “ActionControlExcute” performs different functions, such as “PACKET\_TX\_START”, “PACKET\_TX\_STOP”, etc.

### 3.3.4 ActionReport

```
typedef int
```

```

(*BT_DLL_MODULE_FP_ACTION_REPORT)(
    BASE_BTMPDLL_MODULE *pDLLBtBaseModule,
    int ActiceItem,
    BT_DEVICE_REPORT *pReport
);

```

This function is used to report current TX/RX packet counts and chip status. The following table shows the relation between “ActionItem” and the parameters in “BT\_DEVICE\_REPORT”. For example, TotalTXBits and TotalTxCounts values are updated when “ActionItem” equals to “REPORT\_PKT\_TX”.

ActiceItem	Index	BT_DEVICE_REPORT
REPORT_PKT_TX	1	TotalTXBits TotalTxCounts
REPORT_CON_TX	2	TotalTXBits TotalTxCounts
REPORT_RKT_RX	3	RXRecvPktCnts TotalRXBits TotalRxCounts TotalRxErrorBits RxRssi ber Cfo
REPORT_TX_GAIN_TABLE	4	CurrTXGainTable
REPORT_TX_DAC_TABLE	5	CurrTXDACTable
REPORT_XTAL	6	CurrRtl8761Xtal
REPORT_CHIP	9	pBTInfo BTInfoMemory
REPORT_LOGICAL_EFUSE	10	ReportData
REPORT_LE_RX	11	TotalRxCounts



## 4. MP Mode Test Control Steps

### 4.1 Enter Bluetooth 4.1BR/EDR Test Mode

Enter the Bluetooth device to Bluetooth DUT Test and stop DUT Test Mode command below:

--Enter DUT Test Mode:

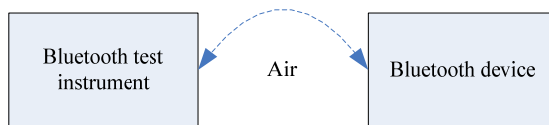
```
pParam->ParameterIndex = TEST_MODE_ENABLE;  
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);  
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

--Stop Test Mode

```
pParam->ParameterIndex = HCI_RESET;  
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);  
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Use bt\_mp\_Exec(HCI\_RESET) to stop Bluetooth test mode.

The Test connection diagram :



### 4.2 Continue TX Mode

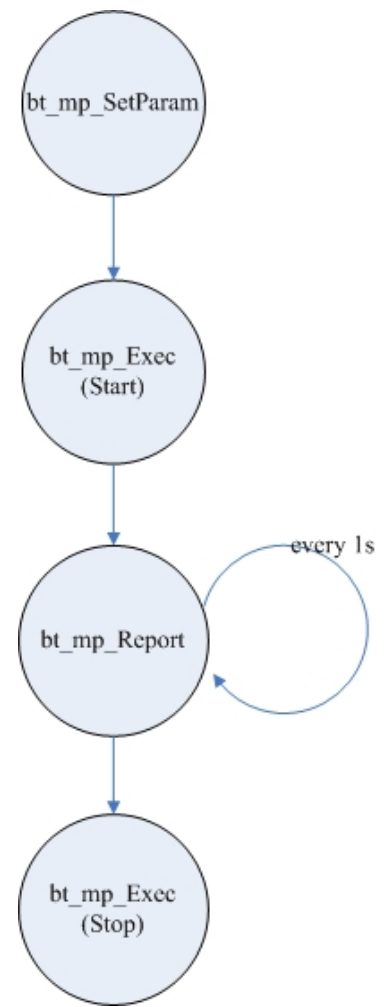
In order to transmit a continuous signal, follow the process below:

Step1: UpDataParameter

Name
ParameterIndex
mChannelNumber
mPacketType
mPayloadType
mWhiteningCoeffValue
mTxGainIndex
mPacketHeader
mHitTarget

Step2: ActionControlExcute (CONTINUE\_TX\_START)

```
pParam->ParameterIndex = CONTINUE_TX_START;
```



```
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
```

```
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step3: ActionReport (REPORT\_CON\_TX)

```
pBluetoothModule->ActionReport(pBluetoothModule, REPORT_CON_TX, pReport);
```

Step4: ActionControlExcute (CONTINUE\_TX\_STOP)

```
pParam->ParameterIndex = CONTINUE_TX_STOP;
```

```
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
```

```
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

PS. "ActionReport " should be called every 200ms.

## 4.3 Packet TX Mode

In order to transmit a packet signal, follow the process below:

Step1: UpDataParameter

Name
ParameterIndex
mChannelNumber
mPacketType
mPayloadType
mTxPacketCount
mWhiteningCoeffValue
mTxGainIndex
mPacketHeader
mHitTarget

Step2: ActionControlExcute (PACKET\_TX\_START)

```
pParam->ParameterIndex = PACKET_TX_START;
```

```
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
```

```
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step3: ActionReport (REPORT\_PKT\_TX)

```
pBluetoothModule->ActionReport(pBluetoothModule, REPORT_PKT_TX, pReport);
```

Step4: ActionControlExcute (PACKET\_TX\_STOP)

```
pParam->ParameterIndex = PACKET_TX_STOP;
```

```
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
```

```
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

PS. "ActionReport " should be called every 200ms.

## 4.4 Packet RX Mode

In order to receive a packet signal, follow the process below:

Step1: UpDataParameter

Name
ParameterIndex
mChannelNumber
mPacketType
mPayloadType
mWhiteningCoeffValue
mPacketHeader
mHitTarget

Step2: ActionControlExcute (PACKET\_RX\_START)

```
pParam->ParameterIndex = PACKET_RX_START;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step3: ActionReport (REPORT\_PKT\_RX)

```
pBluetoothModule->ActionReport(pBluetoothModule, REPORT_PKT_RX, pReport);
```

Step4: ActionControlExcute (PACKET\_RX\_STOP)

```
pParam->ParameterIndex = PACKET_RX_STOP;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

**PS. "ActionReport " should be called every 200ms.**

## 4.5 Hopping Mode(Hopping Dwell time Test)

In order to start hopping mode test, follow the steps below:

Step 1: bt\_mp\_SetParam....(to setting packet type:DH1,DH2....3DH5)

Name
PacketType
HoppingFixChannel
ChannelNumber
WhiteningCoeffValue

Packet Type	INDEX
BT_PKT_DH1	0
BT_PKT_DH3	1
BT_PKT_DH5	2
BT_PKT_2DH1	3
BT_PKT_2DH3	4
BT_PKT_2DH5	5
BT_PKT_3DH1	6
BT_PKT_3DH3	7

BT_PKT_3DH5	8
BT_PKT_LE	9
BT_PKT_NULL	10

Step 2: bt\_mp\_Exec(HOPPING\_DWELL\_TIME)

```
pParam->ParameterIndex = HOPPING_DWELL_TIME;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule)
```

Step 3: bt\_mp\_Exec(HCI\_RESET) to disable hopping mode.

```
pParam->ParameterIndex = HCI_RESET;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

PS. If HoppingFixChannel = 1, it enable fix channel that is controlled by “ChannelNumber”.

If HoppingFixChannel = 0, “ChannelNumber” is useless.

## 4.6 LE Mode (BQB Test)

After testing BT4.0 only enable device and download patch code, you should jump interface to the instrument.

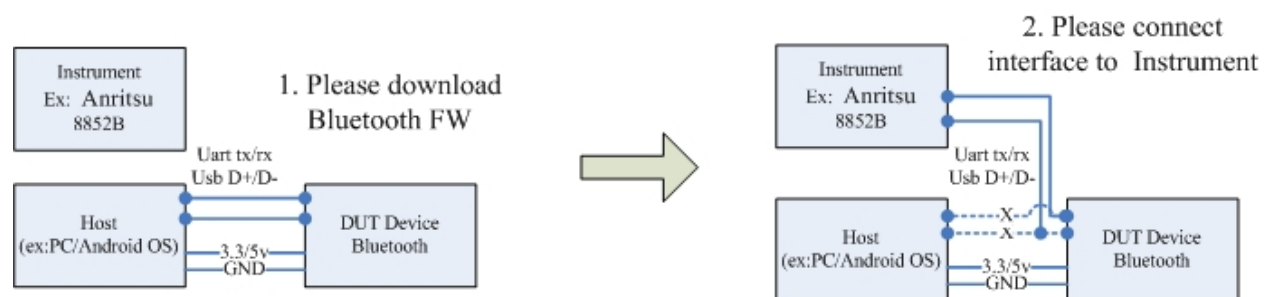
Step 1:rtlbtm

Step 2:enable (xxxxxxx)

Step 3: Jump hardware interface to the instrument.

Step 4:Begin test.

BLE is a schematic diagram of the test for Certification:



## 4.7 LE DUT Test Mode (TX/RX)

To start LE TX DUT test mode, follow the steps below:

Step 1: bt\_mp\_SetParam

INDEX	VALUE	Value Range
1	ChannelNumber	0~39
3	PayloadType	BT_LE_PAYLOAD_TYPE_PRBS9 = 0, BT_LE_PAYLOAD_TYPE_1111_0000 = 1, BT_LE_PAYLOAD_TYPE_1010 = 2, BT_LE_PAYLOAD_TYPE_PRBS15 = 3,

		BT_LE_PAYLOAD_TYPE_ALL1 = 4, BT_LE_PAYLOAD_TYPE_ALL0 = 5, BT_LE_PAYLOAD_TYPE_0000_1111 = 6, BT_LE_PAYLOAD_TYPE_0101 = 7,
15	LEDataLen	0x00~0x25

Step 2: bt\_mp\_Exec(LE\_TX\_DUT\_TEST\_CMD)

```
pParam->ParameterIndex = LE_TX_DUT_TEST_CMD;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step 3: bt\_mp\_Exec(LE\_DUT\_TEST\_END\_CMD) to stop LE TX DUT mode.

```
pParam->ParameterIndex = LE_DUT_TEST_END_CMD;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

To start LE RX DUT test mode, follow the steps below:

Step 1: bt\_mp\_SetParam

INDEX	VALUE	Value Range
1	ChannelNumber	0~39

Step 2: bt\_mp\_Exec(LE\_RX\_DUT\_TEST\_CMD)

```
pParam->ParameterIndex = LE_RX_DUT_TEST_CMD;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step 3: bt\_mp\_Exec(LE\_DUT\_TEST\_END\_CMD) to stop LE RX DUT mode.

```
pParam->ParameterIndex = LE_DUT_TEST_END_CMD;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step 4: bt\_mp\_Report 11

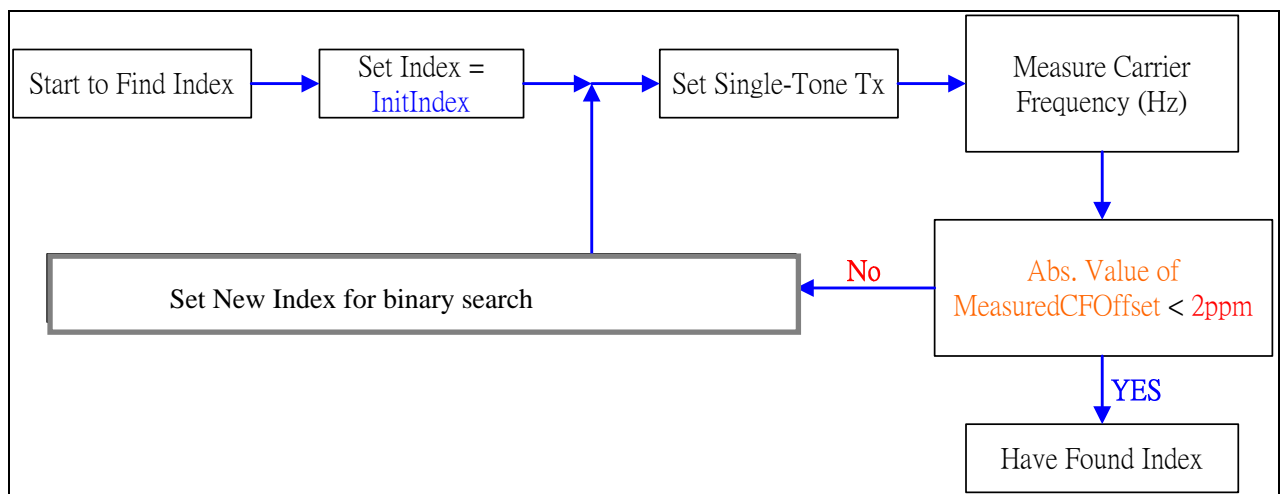
```
//Update mBT_DEVICE_REPORT.TotalRxCounts
pBluetoothModule->ActionReport(pBluetoothModule,REPORT_LE_RX,
&mBT_DEVICE_REPORT);
```

## 5.MP Test Flow

### 5.1 Bluetooth Calibrates Crystal (Xtal) Cap by Non-Signaling mode

First, take a look at eFuse content about setting of Crystal Cap. . Normal driver will load this value in initial step. So this value must be well-calibrated and filled on correct eFuse location. The steps to PG efuse is in section 6.3.

The flow of calibration Bluetooth step is as below:



#### Finding Crystal Cap. Index Flow

**InitIndex:** the default value is 0x20. Index range is 0x0 to 0x3F.

**MeasuredCFOffset:** Carrier frequency measured by instrument - Ideal Carrier Frequency Target range Abs. Value of 2ppm in 2.441GHz band is about 10KHz(±5KHz).

Step1 : Set Index = **InitIndex(0x20)**

```
pParam->ParameterIndex = SET_XTAL;  
pParam->Rtl8761Xtal = InitIndex(0x20);  
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);  
pBluetoothModule->ActionControlExcute(pBluetoothModule);
```

Step2: Set Single-Tone Tx

```
pBtParam ->mChannelNumber=39;  
pBtParam ->mPacketType=3DH5;  
pBtParam ->mPayloadType=ALL0;  
pBtParam ->mTxPacketCount=0(Infinite);  
pBtParam ->mWhiteningCoeffValue =0xFF(disable);  
pBtParam ->mTxGainIndex=7;  
pBtParam ->mPacketHeader=0x3FFFF;
```

```
pBtParam ->mHitTarget= 0x000000c6967e  
pBtParam ->ParameterIndex= CONTINUE_TX_START;  
UpDataParameter(pBtModule,pBtParam);  
ActionControlExcute(pBtModule);
```

Step 3: Stop Single-Tone Tx.

```
ParameterIndex= CONTINUE_TX_STOP;  
UpDataParameter(pBtModule,pBtParam);  
ActionControlExcute(pBtModule);
```

Step 4: Go to Step1 and re-tune Index value until find the best Crystal index value.

Step 5: Write Crystal value to Efuse. (Please refer to Chapter 6.3)

## 5.2 Verify Bluetooth Tx Performance by Non-Signaling mode

To measure the DUT TX power/initial Carrier offset/modulation characteristics to check Tx performance is ok or not. Bluetooth TX criterion is shown as below:

	Test Item	Sub Test Item	Packet Type	Channel	Criterion
					Bluetooth Spec.
Verify Tx DH1	Maximum Output Power	Average Power	DH1	Low (CH6-2408MHZ)	> 0dBm
				Middle (CH42-2444MHZ)	> 0dBm
				High (CH70-2472MHZ)	> 0dBm
	Modulation Characteristics	Delta F1 Avg.	DH1	Low (CH6-2408MHZ)	140KHz ~ 175KHz
		Delta F2 Max.		Middle (CH42-2444MHZ)	> 115KHz
		Modulation Index		High (CH70-2472MHZ)	> 0.8
	Initial Carrier Frequency Error		DH1	Low (CH6-2408MHZ)	-20KHz ~ 20KHz
				Middle (CH42-2444MHZ)	
				High (CH70-2472MHZ)	
Verify Tx 3DH1	Maximum Output Power	Average Power	3DH1	Low (CH6-2408MHZ)	> 0dBm
				Middle (CH42-2444MHZ)	> 0dBm
				High (CH70-2472MHZ)	> 0dBm
	Modulation Characteristics	RMS DEVM	3DH1	Low (CH6-2408MHZ)	0.13
		Peak DEVM		Middle (CH42-2444MHZ)	0.25
		99% DEVM		High (CH70-2472MHZ)	0.20
	Initial Carrier Frequency Error		3DH1	All	-20KHz ~ 20KHz

**Table The recommended test items of Bluetooth Tx**

For example, use adb commands android platform, device is UART interface.

step by step command:



(1) Enter MP Mode and download patch code

(2) Set Parameter :

Test Item		adb command
	Test Item	Channel = 6
DH1	Maximum Power	mChannelNumber=0x06; mPacketType=0x00; mPayloadType=0x07; mTxPacketCount=0x00; mWhiteningCoeffValue=0x7F; mTxGainIndex=0x7; mPacketHeader=0x17F0E; mHitTarget=0x0000009e8b33
DH1	Delta F1	mChannelNumber=0x06; mPacketType=0x00; mPayloadType=0x05; mTxPacketCount=0x00; mWhiteningCoeffValue=0xFF; mTxGainIndex=0x7; mPacketHeader=0x17F0E; mHitTarget=0x0000009e8b33
DH1	Delta F2	mChannelNumber=0x06; mPacketType=0x00; mPayloadType=0x02; mTxPacketCount=0x00; mWhiteningCoeffValue=0xFF; mTxGainIndex=0x7; mPacketHeader=0x17F0E; mHitTarget=0x0000009e8b33
3DH1	ALL	mChannelNumber=0x06; mPacketType=0x06; mPayloadType=0x07; mTxPacketCount=0x00; mWhiteningCoeffValue=0x7F; mTxGainIndex=0x7; mPacketHeader=0x31B6E; mHitTarget=0x0000009e8b33

Test Item		adb command
	Test Item	Channel = 70
DH1	Maximum Power	mChannelNumber=0x46; mPacketType=0x00; mPayloadType=0x07; mTxPacketCount=0x00; mWhiteningCoeffValue=0x7F;

		mTxGainIndex=0x7; mPacketHeader=0x17F0E; mHitTarget=0x0000009e8b33
DH1	Delta F1	mChannelNumber=0x46; mPacketType=0x00; mPayloadType=0x05; mTxPacketCount=0x00; mWhiteningCoeffValue=0xFF; mTxGainIndex=0x7; mPacketHeader=0x17F0E; mHitTarget=0x0000009e8b33
DH1	Delta F2	mChannelNumber=0x46; mPacketType=0x00; mPayloadType=0x02; mTxPacketCount=0x00; mWhiteningCoeffValue=0xFF; mTxGainIndex=0x7; mPacketHeader=0x17F0E; mHitTarget=0x0000009e8b33
3DH1	ALL	mChannelNumber=0x46; mPacketType=0x06; mPayloadType=0x07; mTxPacketCount=0x00; mWhiteningCoeffValue=0x7F; mTxGainIndex=0x7; mPacketHeader=0x31B6E; mHitTarget=0x0000009e8b33

(3)Run Packe Tx

(4) measured by Bluetooth test instrument(ex. Letepoint IQNxN)

PS. “ActionReport ” should be called every 200ms.

(5)Stop Packet Tx

PS. If you need to test other parameters, please stop packet tx and go back to step 2

## 5.3 Verify Bluetooth Rx Performance by Non-Signaling mode

Measure the DUT Rx sensitivity to check Rx performance is ok or not. The Rx performance test can be measured in Signaling mode (ex: Anritsu 8852B, Agilent N4010A) or Non-Signaling mode (ex:LitePoint IQNxN). Bluetooth Rx criterion is shown as below:

Verify Bluetooth Rx	Test Item	Packet Type	Criterion
			Bluetooth Spec

	Sensitivity	DH1 or 3DH5	< -70dBm
--	-------------	-------------	----------

For final MP, Rx can just test DH1, 3DH5 BER at sensitivity criterion power level at channel 0 and 78 to reduce time. All Bluetooth Rx criterion is shown as:

Test Item		Criterion(Bluetooth Spec)
Channel	Packet type	< -70dBm
6	DH1	< -70dBm
42	DH1	< -70dBm
70	DH1	< -70dBm
6	3DH1	< -70dBm
42	3DH1	< -70dBm
70	3DH1	< -70dBm

**Figure The recommended test items of Bluetooth Rx**

(1) Enter MP Mode and download patch code

(2) Set Parameter :

Test Item		adb command
Channel	Packet type	
6	DH1	mChannelNumber=0x06; mPacketType=0x00; mPayloadType=0x07; mWhiteningCoeffValue=0xFF; mPacketHeader=0x3FFFF; mHitTarget=0x000000c6967e
42	DH1	mChannelNumber=0x2a; mPacketType=0x00; mPayloadType=0x07; mWhiteningCoeffValue=0xFF; mPacketHeader=0x3FFFF; mHitTarget=0x000000c6967e
70	DH1	mChannelNumber=0x46; mPacketType=0x00; mPayloadType=0x07; mWhiteningCoeffValue=0xFF; mPacketHeader=0x3FFFF; mHitTarget=0x000000c6967e
6	3DH1	mChannelNumber=0x06; mPacketType=0x06; mPayloadType=0x07; mWhiteningCoeffValue=0xFF; mPacketHeader=0x3FFFF; mHitTarget=0x000000c6967e
42	3DH1	mChannelNumber=0x2a;

		mPacketType=0x06; mPayloadType=0x07; mWhiteningCoeffValue=0xFF; mPacketHeader=0x3FFFF; mHitTarget=0x000000c6967e
70	3DH1	mChannelNumber=0x46; mPacketType=0x06; mPayloadType=0x07; mWhiteningCoeffValue=0xFF; mPacketHeader=0x3FFFF; mHitTarget=0x000000c6967e

(3)To setting Parameter with the Bluetooth test instrument. Bluetooth test instrument begin transmit..

(4)Run Packet Rx

(5) Report Received Result.

PS. “ActionReport ” should be called every 200ms.

(6)Stop Packet Rx

PS. If you need to test other parameters, please stop packet Rx and go back to step 2.

## 6. Write EFUSE Data to Bluetooth Device

The EFUSE data is composited in entry unit and little endian format, which is 2-byte offset, 1-byte length and data in length size.

The EFUSE write/read operations are supported by the below series of Bluetooth chips.

Realtek Bluetooth Chip Number
RTL8761A series

### 6.1 Write MAC Address to EFUSE

The EFUSE data of Bluetooth MAC address is 6-byte length, and offset started at 0x3C.

For example, if the default value of Bluetooth MAC address is 0x00E04C829987 (00:E0:4C:82:99:87), and the user wants to update the value to 0x00E023345678 (00:E0:23:34:56:78) like below.

Offset	Default value	Example value	Description
0x3c-0x41	0x00E04C829987	0x00E023345678	BD_ADDR

The data array is organized as below: item index should be 0x00, and sub-index should be fixed as 0x01.

Byte Numb	0	1	2	3	4	5	6	7	8	9
Description	Sub-Index	Offset (Low)	Offset (High)	Length	BD_ADDR[0]	BD_ADDR[1]	BD_ADDR[2]	BD_ADDR[3]	BD_ADDR[4]	BD_ADDR[5]
Array	0x01	0x3C	0x00	0x06	0x78	0x56	0x34	0x23	0xE0	0x00

Use the command below to write MAC address to EFUSE.

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;  
pPGRawData[0] = 0x01;  
pPGRawData[1] = 0x3C;  
pPGRawData[2] = 0x00;  
pPGRawData[3] = 0x06;  
pPGRawData[4] = 0x78;  
pPGRawData[5] = 0x56;  
pPGRawData[6] = 0x34;  
pPGRawData[7] = 0x23;  
pPGRawData[8] = 0xE0;  
pPGRawData[9] = 0x00;
```

```
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

It is recommended that MAC address should be verified by the below commands.

```
pParam->ParameterIndex = READ_EFUSE_DATA;
pPGRawData[0] = 0x01; // BT Efuse
pPGRawData[1] = 0x3c;
pPGRawData[2] = 0x00;
pPGRawData[3] = 0x06;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);

pBluetoothModule->ActionReport(pBluetoothModule,
REPORT_LOGICAL_EFUSE, &mBT_DEVICE_REPORT);
//Get BT addr = pReport[9], pReport[8],..., pReport[5], pReport[4]);
```

## 6.2 Write Power On Function to EFUSE

The EFUSE data of Power On function has two entries to configure: offset 0x0177 is used to set Power On function enabled or disabled, while offset 0x30-0x31 is used to set Power On level.

Offset	Default value	Example value	Description
0x30-0x31	0x5240	0x5052	If 0x30[2:1] == 0x01 && 0x30[4] == 0x01, then BT_WAKE_HOST pin is set to high by default; otherwise low.
0x0177	0x3C	0x3D	0x177[0]: enable power on

The data array is organized as below: item index should be 0x00, and sub-index should be fixed as 0x01.

Byte Number	0	1	2	3	4	5
Description	Sub-Index	Offset (Low)	Offset (High)	Length	Power On Enable Power On Level (Low)	Power On Level (High)
Power on Enable Array	0x01	0x77	0x01	0x01	0x3C[0x3D]	

<b>Power on Level Array</b>	<b>0x01</b>	<b>0x30</b>	<b>0x00</b>	<b>0x02</b>	<b>0x40[0x52]</b>	<b>0x52[0x50]</b>

Use the command below to disable power on function.

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;
pPGRawData[0] = 0x01;
pPGRawData[1] = 0x77;
pPGRawData[2] = 0x01;
pPGRawData[3] = 0x01;
pPGRawData[4] = 0x3C;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

Use the command below to enable power on function.

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;
pPGRawData[0] = 0x01;
pPGRawData[1] = 0x77;
pPGRawData[2] = 0x01;
pPGRawData[3] = 0x01;
pPGRawData[4] = 0x3D;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

Use the command below to set power on active level high (keep low level when idle).

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;
pPGRawData[0] = 0x01;
pPGRawData[1] = 0x30;
pPGRawData[2] = 0x00;
pPGRawData[3] = 0x02;
pPGRawData[4] = 0x40;
pPGRawData[5] = 0x52;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

Use the command below to set power on active level low (keep high level when idle).

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;
pPGRawData[0] = 0x01;
pPGRawData[1] = 0x30;
pPGRawData[2] = 0x00;
pPGRawData[3] = 0x02;
pPGRawData[4] = 0x52;
pPGRawData[5] = 0x50;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

## 6.3 Write Crystal Cap Calibration Value to EFUSE

Normal driver will load the Crystal Cap value in the initial stage, so it must be well-calibrated and written to EFUSE.

The EFUSE data of Crystal Cap Calibration is 1-byte length, and offset started at 0x01E6.

Offset	Default value	Example value	Description
0x01E6	0x20	0x00~0x3F	XTAL_K Value Bit[5:0], Xi, Xo Range

The data array is organized as below: item index should be 0x00, and sub-index should be fixed as 0x01.

Byte Number	0	1	2	3	4
Description	Sub-Index	Offset (Low)	Offset (High)	Length	Xtal value
Crystal Cap Calibration Array	0x01	0xE6	0x01	0x01	0~63

Use the command to write Crystal (Xtal) value 0x20 to EFUSE.

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;
pPGRawData[0] = 0x01;
pPGRawData[1] = 0xE6;
pPGRawData[2] = 0x01;
pPGRawData[3] = 0x01;
pPGRawData[4] = 0x20;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

It is recommended that Crystal Cap value should be verified by the below commands.

```
pParam->ParameterIndex = READ_EFUSE_DATA;
pPGRawData[0] = 0x01; // BT Efuse
pPGRawData[1] = 0xE6;
pPGRawData[2] = 0x01;
pPGRawData[3] = 0x01;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule);

pBluetoothModule->ActionReport(pBluetoothModule,
```



```
REPORT_LOGICAL_EFUSE, &mBT_DEVICE_REPORT);
//Get Crystal (Xtal) value = pReport[4];
```

## 6.4 Write VID (Vendor ID) and PID (Product ID) to EFUSE

The VID and PID configuration is used only for USB interface chips, and defined in system EFUSE.

Offset	Default value	Example value	Description
0x24	0xDA	0xDA	USB Vendor ID[7:0]
0x25	0x0B	0x0B	USB Vendor ID[15:8]
0x26	0x50	0x61	USB Product ID[7:0]
0x27	0x28	0x87	USB Product ID[15:8]

The data array is organized as below: item index should be 0x00, and sub-index should be fixed as 0x02.

Byte Number	0	1	2	3	4	5	6	7
Description	Sub-Index	Offset (Low)	Offset (High)	Length	VID LSB	VID MSB	PID LSB	PID MSB
VID & PID Array	0x02	0x24	0x00	0x04	0xDA	0x0B	0x61	0x81

Use the command below to write VID & PID to EFUSE.

```
pParam->ParameterIndex = WRITE_EFUSE_DATA;
pPGRawData[0] = 0x02;
pPGRawData[1] = 0x24;
pPGRawData[2] = 0x00;
pPGRawData[3] = 0x04;
pPGRawData[4] = 0xDA;
pPGRawData[5] = 0x0B;
pPGRawData[6] = 0x61;
pPGRawData[7] = 0x81;
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule) ;
```

