

Realtek

Bluetooth

Mass Production Flow

(windows API of Realtek)

Draft1.0

Date: 2020/07/20

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.

COPYRIGHT

©2019 Realtek Semiconductor Corp. All rights reserved. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form or by any means without the written permission of Realtek Semiconductor Corp.

DISCLAIMER

Realtek provides this document 'as is', without warranty of any kind. Realtek may make improvements and/or changes in this document or in the product described in this document at any time. This document could include technical inaccuracies or typographical errors.

TRADEMARKS

Realtek is a trademark of Realtek Semiconductor Corporation. Other names mentioned in this document are trademarks/registered trademarks of their respective owners.

USING THIS DOCUMENT

This document is intended for the software engineer's reference and provides detailed programming information.

Though every effort has been made to ensure that this document is current and accurate, more information may have become available subsequent to the production of this guide.

REVISION HISTORY

Revision	Release Date	Summary
1.0 beta		

Table of Contents

1.	OVERVIEW.....	6
2.	PREREQUISITES.....	7
3.	MASS PRODUCTION API'S FUNCTION USAGE	8
3.1	BTMPAPI_ExternFunctionRTK.....	9
3.2	Interface Setting.....	11
3.3	Realtek BT MP Function Build	14
3.4	Device Open	15
3.5	Control DUT Action	16
3.6	Device Close	18
4	DIRECT TEST MODE(DTM).....	19
4.1	BR/EDR-Direct Test Mode (Certification)	19
4.2	BLE Direct Test Mode (BQB Test)	20
4.3	LE DUT Test Mode-TX/RX (MP)	21
5	RF TEST MODE (CERTIFICATION).....	23
5.1	TX Test -Continuous Tx(Certification).....	23
5.2	DUT Packet TX (MP).....	24
5.3	DUT Packet RX (MP)	25
5.4	LE Certification TX Mode(Certification)	26
5.5	Hopping Test (Certification- Dwell time).....	27
6	MASS PRODUCTION TEST FLOW.....	28
6.1	Bluetooth Device Initial.....	29
6.2	Bluetooth calibration Frequency	35
6.3	Bluetooth calibration Tx Power	40
6.4	Bluetooth verify flow	45
7.	WRITE EFUSE DATA TO BLUETOOTH DEVICE(BT CONTROLLER)	55
7.1	Write BT MAC Address to EFUSE.....	55
7.2	Write Power On Function to EFUSE	57
7.3	Write Crystal Cap Calibration Value to EFUSE	59
7.4	Write VID (Vendor ID) and PID (Product ID) to EFUSE	61
8.	APPENDIX: DEFINE	62
8.1	INTERFACE_TYPE	62
8.4	BT_ACTIONCONTROL_TAG.....	63
8.5	PKT_TYPE	66
8.6	BLE_MODULATION_TYPE.....	67
8.7	PAYLOAD_TYPE.....	67
8.8	PACKET_HEADER.....	68
8.9	Report ActiceItem	69
8.11	TX Gain K Define Table.....	70
8.12	Flatness Define Table	70
9.	APPENDIX: STRUCTURE	71
9.1	BASE_INTERFACE_MODULE :Interface Base structure	71

9.2 BT_PARAMETER Structure	72
9.3 BASE_CAL_PARAMETER.....	74

Table

TABLE 1 BLUETOOTH CHIP SUPPORT SERIES.....	7
TABLE 2 BTMPAPI_EXTERNFUNCTIONRTK API	10
TABLE 3 BTMPAPI_BUILDINTERFACEVENDOR API	12
TABLE 4 BTMPAPI_BUILDINTERFACERTK API	13
TABLE 5 BTMPAPI_BUILDINTERFACERTK API	14
TABLE 6 DOWNLOADPATCHCODE API	15
TABLE 7 UPDATAPARAMETER API	16
TABLE 8 ACTIONCONTROLEXCUTE API	17
TABLE 9 ACTIONREPORT API	18
TABLE 10 HOPPING MODE:PACKET TYPE	27
TABLE 11 SET FLATNESS VALUE COMMAND TABLE	33
TABLE 12 SET FLATNESS VALUE FLOW	33
TABLE 13 SET TX PATH LOSS MODULE VALUE TABLE	34
TABLE 14 API_CALCULATE_XTALBOUNDARY API	37
TABLE 15 API_CALCULATE_XTALNORMAL API	38
TABLE 16 THE TX POWER FLATNESS VALUE DEFINE TABLE(2BYTE)	42
TABLE 17 THE TX POWER FLATNESS VALUE DEFINE SAMPLE TABLE (2BYTE)	42
TABLE 18 THE TX POWER FLATNESS VALUE DEFINE TABLE(4BYTE)	43
TABLE 19 THE TX POWER FLATNESS VALUE DEFINE SAMPLE TABLE (4BYTE)	43
TABLE 20 BLUETOOTH LEGACY TX CRITERION	45
TABLE 21 BLUETOOTH BLE TX CRITERION(BT4.0)	48
TABLE 22 BLUETOOTH BLE 5.0 TX CRITERION	49
TABLE 23 VERIFY BLUETOOTH LEGACY RX PERFORMANCE SPEC	51
TABLE 24 THE RECOMMENDED TEST ITEMS OF BLUETOOTH BLE RX	53
TABLE 25 THE RECOMMENDED TEST ITEMS OF BLUETOOTH BLE 5.0 RX	54
TABLE 26 WRITE EFUSE SUPPORT CHIPS	55
TABLE 27 INTERFACE TYPE INDEX	62
TABLE 28 BT_ACTIONCONTROL DEFINE INEX	64
TABLE 29 PKT_TYPE FOR BDE/EDR,HOPPING INDEX	66
TABLE 30 BLE PKT_TYPE INDEX	66
TABLE 31 BLE_MODULATION_TYPE	67
TABLE 32 PAYLOAD_TYPE INDEX FOR BDR/EDR	67
TABLE 33 PAYLOAD_TYPE INDEX FOR BLE	67
TABLE 34 REPORT ACTICE ITEM INDEX	69
TABLE 35 TX GAIN K DEFINE TABLE	70
TABLE 36 FLATNESS DEFINE TABLE	70
TABLE 37 UPDATAPARAMETER DEFINE: BT_PARAM	73

Figure

FIGURE 1 MASS PRODUCTION API USE FLOW	8
FIGURE 2 DEVICE OPEN FLOW	15
FIGURE 3 BR/EDR–DIRECT TEST MODE	19
FIGURE 4 BLE DIRECT TEST MODE	20
FIGURE 5 BLE DIRECT TEST MODE CONNECTER	20
FIGURE 6 START LE TX DUT TEST FLOW	21
FIGURE 7 START LE RX DUT TEST FLOW	21
FIGURE 8 TX TEST -CONTINUOUS Tx FLOW	23
FIGURE 9 DUT PACKET TX FLOW	24
FIGURE 10 DUT PACKET RX FLOW	25
FIGURE 11 MASS PRODUCTION TEST FLOW	28
FIGURE 12 DISABLE TX POWER TRACKING FLOW	30
FIGURE 13 SET TX POWER LEVEL TO RAM FLOW	31
FIGURE 14 SET TX GAIN CAL (K) VALUE	32
FIGURE 15 SEARCH CRYSTAL CAPACITANCE INDEX FLOW	35
FIGURE 16 SET CALIBRATES CRYSTAL (XTAL) CAP VALUE FLOW	36
FIGURE 17 BLUETOOTH CRYSTAL BOUNDARY CAP CALIBRATION FLOW	39
FIGURE 18 CRYSTAL ACCURACY CHECK FLOW (CRYSTAL SOURCE COMES FROM MODULE) FLOW	39
FIGURE 19 BLUETOOTH CALIBRATION Tx POWER SCHEMATIC	40
FIGURE 20 BLUETOOTH TX POWER CALIBRATION DIAGRAM	40
FIGURE 21 SET POWER Tx LEVEL FLOW	41
FIGURE 22 CALIBRATION TO TARGET POWER	41
FIGURE 23 BLUETOOTH TX POWER FLATNESS CALIBRATION FLOW	42
FIGURE 24 VERIFY BLUETOOTH LEGACY Tx PERFORMANCE FLOW	45
FIGURE 25 VERIFY BLUETOOTH LE TX PERFORMANCE FLOW(BT4.0)	48
FIGURE 26 VERIFY BLUETOOTH LE TX PERFORMANCE FLOW(BT5.0)	49
FIGURE 27 VERIFY BLUETOOTH LEGACY RX API'S FLOW	51
FIGURE 28 THE RECOMMENDED TEST ITEMS OF BLUETOOTH RX	51
FIGURE 29 THE RECOMMENDED TEST ITEMS OF BLUETOOTH BLE Rx FLOW	53
FIGURE 30 VERIFY BLUETOOTH BLE5.0 RX	54

1. Overview

This document is used to introduce MP (Mass Production) test DLL 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.

Realtek

2. Prerequisites

Before MP tool startup, below items should be checked:

➤ **Support:**

a) The Bluetooth Chip embedded is provided by Realtek;

Bluetooth Combo Chip	Bluetooth controller Chip	BT SOC
RTL8723B series	RTL8761A series	TBD
RTL8821A series	RTL8761B series	
RTL8703B series		
RTL8723D series		
RTL8822B series		
RTL8821C series		
RTL8822C series		
RTL8852A series		
RTL8723F series		

Table 1 Bluetooth Chip support series

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

MP tool package is provided to customers in binary format:

RtlBluetoothMP.dll	MP DLL library
RtlBluetoothMP.h	MP DLL header
bt_mp_calibration_base.h	MP DLL 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
BTPatchCode\ Patch_rtl8703b.bin	rtl8703b fw patch
BTPatchCode\ Patch_rtl8723d.bin	rtl8723d fw patch
BTPatchCode\ Patch_rtl8822b.bin	rtl8822b fw patch
BTPatchCode\ Patch_rtl8821c.bin	rtl8821c fw patch
BTPatchCode\ Patch_rtl8822c.bin	rtl8822c fw patch
BTPatchCode\ Patch_rtl8761b.bin	rtl8761b fw patch
BTPatchCode\ Patch_rtl8852a.bin	rtl8852a fw patch
....later chip of Realtek	

BT Soc chip isn't need patch binary file.

3. Mass Production API's function Usage

This chapter briefly introduces how to link the dynamic link library to control DUT. The flow show as:

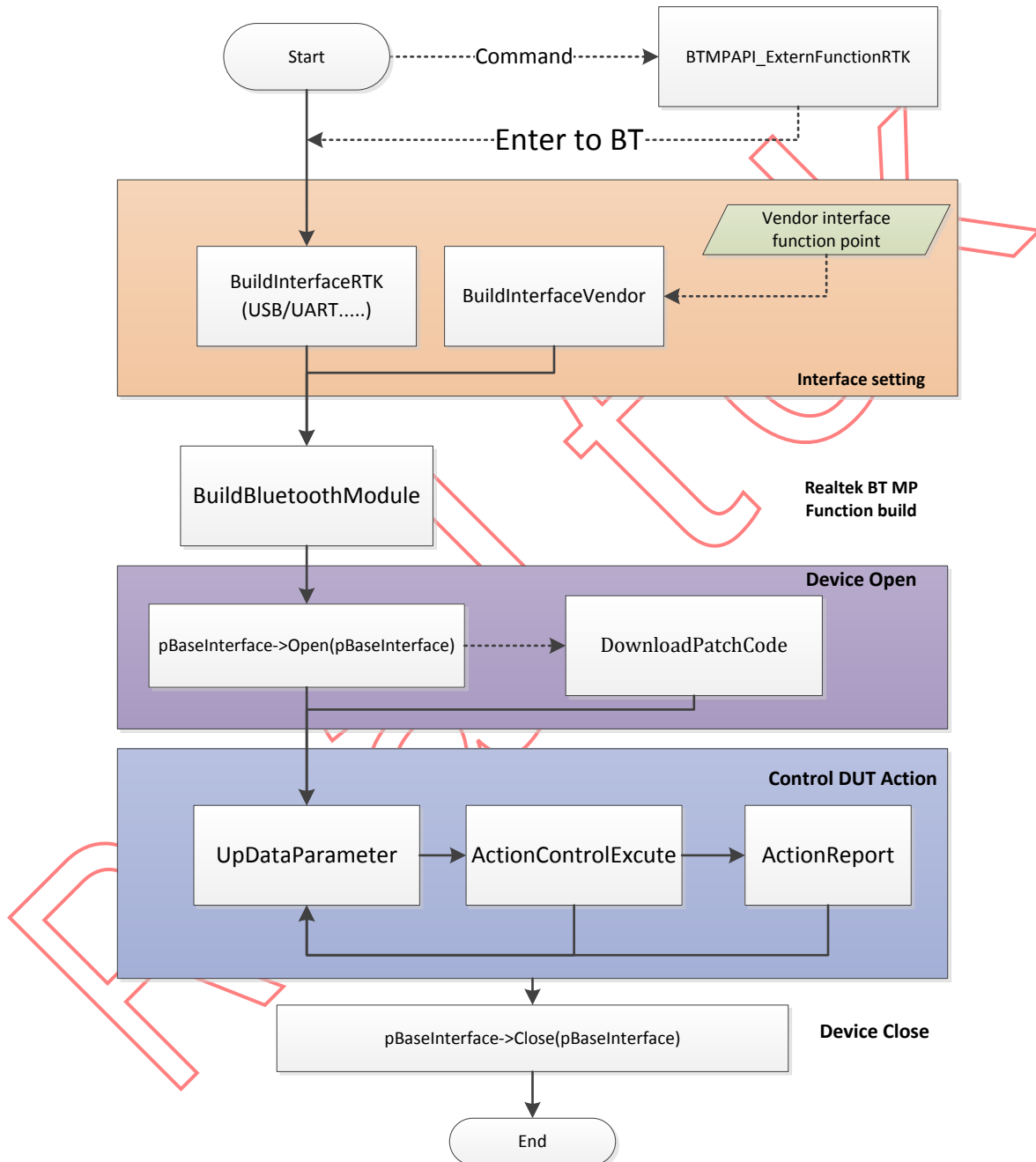


Figure 1 Mass Production API use flow

3.1BTMPAPI_ExternFunctionRTK

If you need to send a special string to enter the BT interface then can use the API below to ENTER.

BTMPAPI_ExternFunctionRTK	
To send a special string to enter or leave the BT interface	
Syntax	
<pre>RTKBTMP_API int BTMPAPI_ExternFunctionRTK(int Chip_type, int Enter1_Leave2, BASE_INTERFACE_MODULE **ppBaseInterface, BASE_INTERFACE_MODULE *pBaseInterfaceModuleMemory, unsigned int InterfaceType, unsigned char PortNo, unsigned long Baudrate, unsigned char *pWriting, unsigned char *pReading, unsigned int return_check, unsigned char *pData, unsigned int Ending_close);</pre>	
Parameters	
Chip_type,	Integer Value
Enter1_Leave2	Integer Value 1:Enter to BT interface Path 2.Leave BT interface Path
**ppBaseInterface	Structure BASE_INTERFACE_MODULE double point
*pBaseInterfaceModuleMemory	Structure BASE_INTERFACE_MODULE point
InterfaceType,	Unsigned integer Value ,interface type
PortNo,	Unsigned integer Value,UART /USB Port number
Baudrate,	Unsigned integer Value, UART baudrate
*pWriting,	Unsigned char array, Written array
*pReading,	Unsigned char array, Array after reading
return_check,	Whether to return the data to check: 1:Yes 0:No
*pData,	Unsigned char point , Extern Array.
Ending_close	Integer Value , send after close port. 1:Yes 0:No
Return Value	
Status	0: The operation completed successfully.

		Other:Error
Remakes		
Enter string		
Chip_type	Item	String
RTK_BT_CHIP_ID_AMEBAD = 15	Enter	“ATM0=bridge,baudrate0,baudrate1,parity\n\r”
RTK_BT_CHIP_ID_AMEBAZII =16		
Leave string:		
Chip_type		String
RTK_BT_CHIP_ID_AMEBAD = 15	Leave	“ATM0=bridge,baudrate0,baudrate1,parity\n\r”
RTK_BT_CHIP_ID_AMEBAZII =16		

Table 2 BTMPAPI_ExternFunctionRTK API

Example :

For Ameba D/ZII Sample, you can refer below:

// Enter BT HCI Mode

```
Int rtn=ExternFunctionRTK( RTK_BT_CHIP_ID_AMEBAD
    1, //ener = 1,leave =2
    NULL, NULL,
    InterfaceType,
    PortNo,
    Baudrate,
    NULL, NULL,
    1, //need check event
    NULL, // extern data
    1 ); // send after close port
```

3.2 Interface Setting

There are two options for the interface:

1. Customer-defined interface: The format of the defined interface API needs to be completed
2. Realtek has defined the interface

Therefore, you could select one between BuildInterfaceVendor and BuildInterfaceRTK.

3.2.1 BTMPAPI_BuildInterfaceVendor: Use the customer's interface

BTMPAPI_BuildInterfaceVendor

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.

Syntax

```
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
);
```

Parameters

**ppBaseInterface	Structure BASE_INTERFACE_MODULE double point
*pBaseInterfaceModuleMemory	Structure BASE_INTERFACE_MODULE point
InterfaceType,	Unsigned integer Value ,interface type
PortNo,	Unsigned integer Value,UART /USB Port number
Baudrate,	Unsigned integer Value, UART baudrate
Open	BASE_FP_OPEN function point
Send	BASE_FP_SEND function point
Recv	BASE_FP_RECV function point
Close	BASE_FP_CLOSE function point
WaitMs	BASE_FP_WAIT_MS function point

Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
Base Funvtion point define: <pre>typedef int(*BASE_FP_OPEN)(BASE_INTERFACE_MODULE *pBaseInterface); typedef int(*BASE_FP_SEND)(BASE_INTERFACE_MODULE *pBaseInterface, unsigned char *pWritingBuf, unsigned long Len); typedef int(*BASE_FP_RECV)(BASE_INTERFACE_MODULE *pBaseInterface, unsigned char *pReadingBuf, unsigned long Len, unsigned long *pRetLen); typedef int(*BASE_FP_CLOSE)(BASE_INTERFACE_MODULE *pBaseInterface);</pre>	

Table 3 BTMPAPI_BuildInterfaceVendor API

3.2.2 BTMPAPI_BuildInterfaceRTK: Use the RTK's interface

BTMPAPI_BuildInterfaceRTK	
This function uses Realtek default functions to open, close, send, receive and wait functions.	
Syntax	
<pre>int BTMPAPI_BuildInterfaceRTK(BASE_INTERFACE_MODULE **ppBaseInterface, BASE_INTERFACE_MODULE *pBaseInterfaceModuleMemory, unsigned int InterfaceType, unsigned char PortNo, unsigned long Baudrate);</pre>	
Parameters	
**ppBaseInterface	Structure BASE_INTERFACE_MODULE double point
*pBaseInterfaceModuleMemory	Structure BASE_INTERFACE_MODULE point
InterfaceType,	Unsigned integer Value ,interface type
PortNo,	Unsigned integer Value,UART /USB Port number
Baudrate,	Unsigned integer Value, UART baudrate
Return Value	
Status	0: The operation completed successfully. Other>Error
Remakes	
None	

Table 4 BTMPAPI_BuildInterfaceRTK API

3.3 Realtek BT MP Function Build

BTMPAPI_BuildInterfaceRTK	
This function to build mp function of Realtek in DLL.	
Syntax	
<pre>typedef int (*pfBTMPAPI_BuildBluetoothModule)(BASE_BTMPDLL_MODULE *pBaseBTMPDLLModule, BASE_INTERFACE_MODULE *pBaseInterface, void *pExtra, unsigned char *pTxGainTable, unsigned char *pTxDACTable);</pre>	
Parameters	
* pBaseBTMPDLLModule	Structure BASE_BTMPDLL_MODULE point
* pBaseInterface	Structure BASE_INTERFACE_MODULE point for interface
pExtra,	TBD, Realtek define is NULL
*pTxGainTable,	TBD, Realtek define is NULL
*pTxDACTable,	TBD, Realtek define is NULL
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
The MP function of Realtek Bluetooth device includes four main functions that are "DownloadPatchCode", "UpDataParameter", "ActionControlExcute", and "ActionReport".	

Table 5 BTMPAPI_BuildInterfaceRTK API

3.4 Device Open

There is an action to open the device:

1. Open the interface
2. download patch code to device: **BT SOC series Chip is not required.**



Figure 2 Device Open flow

3.4.1 Interface Open

Please use the open function in the data structure of the interface to open the interface.

As shown in the sample below:

```
pBaseInterface->Open(pBaseInterface);
```

3.4.2 DownloadPatchCode

DownloadPatchCode	
This function is used to download FW patch. Before downloading FW patch, vendor should read different patch files according to different BT chips.	
Syntax	
<pre>typedef int(*BT_DLL_MODULE_FP_ACTION_DLFW)(BASE_BTMPDLL_MODULE *pDLLBtBaseModule, char *pPatchcode, int patchLength, int Mode);</pre>	
Parameters	
* pDLLBtBaseModule	Structure BASE_BTMPDLL_MODULE point
* pPatchcode	Byte point, Array of Patchcode
patchLength,	Integer , Array length of Patchcode
Mode,	integer Value,0:Single 1:merge patch
Baudrate,	Unsigned integer Value, UART baudrate
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
None	

Table 6 DownloadPatchCode API

3.5 Control DUT Action

The Control function includes three main functions that are “UpDataParameter”, ActionControlExcute”, and ActionReport”.

3.5.1 UpDataParameter

UpDataParameter	
This function is to set parameters to DLL.	
Syntax	
<pre>typedef int (*BT_DLL_MODULE_FP_UPDATA_PARAMETER)(BASE_BTMPDLL_MODULE *pDLLBtBaseModule, BT_PARAMETER *pParam);</pre>	
Parameters	
* pDLLBtBaseModule	Structure BASE_BTMPDLL_MODULE point
* pParam	Structure BT_PARAMETER point
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
None	

Table 7 UpDataParameter API

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.

3.5.2 ActionControlExcute

ActionControlExcute	
Excute.	
Syntax	
<pre>typedef int (*BT_DLL_MODULE_FP_ACTION_CONTROLEXCUTE)(BASE_BTMPDLL_MODULE *pDLLBtBaseModule);</pre>	
Parameters	
* pDLLBtBaseModule	Structure BASE_BTMPDLL_MODULE point
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
None	

Table 8 ActionControlExcute API

According to “ParameterIndex” in the “BT_PARAMETER” structure, “ActionControlExcute” performs different functions, such as “PACKET_TX_START”, “PACKET_TX_STOP”, etc.

3.5.3 ActionReport

ActionReport	
Report to the execution result information.	
Syntax	
<pre>typedef int (*BT_DLL_MODULE_FP_ACTION_REPORT)(BASE_BTMPDLL_MODULE *pDLLBtBaseModule, int ActiceItem, BT_DEVICE_REPORT *pReport);</pre>	
Parameters	
* pDLLBtBaseModule	Structure BASE_BTMPDLL_MODULE point
ActiceItem	Integer ,Report item.
*pReport	Structure BT_DEVICE_REPORT point.
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
None	

Table 9 ActionReport API

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”.

3.6 Device Close

Please use the open function in the data structure of the interface to open the interface.
As shown in the sample below:

```
pBaseInterface->Close(pBaseInterface)
```

4 Direct Test Mode(DTM)

This **chapter briefly introduces Direct test mode**. If you want to know more details, please refer to the Bluetooth Spec

4.1 BR/EDR–Direct Test Mode (Certification)

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 :

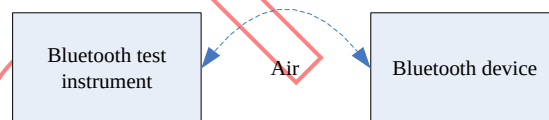


Figure 3 BR/EDR–Direct Test Mode

4.2 BLE Direct Test Mode (BQB Test)

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

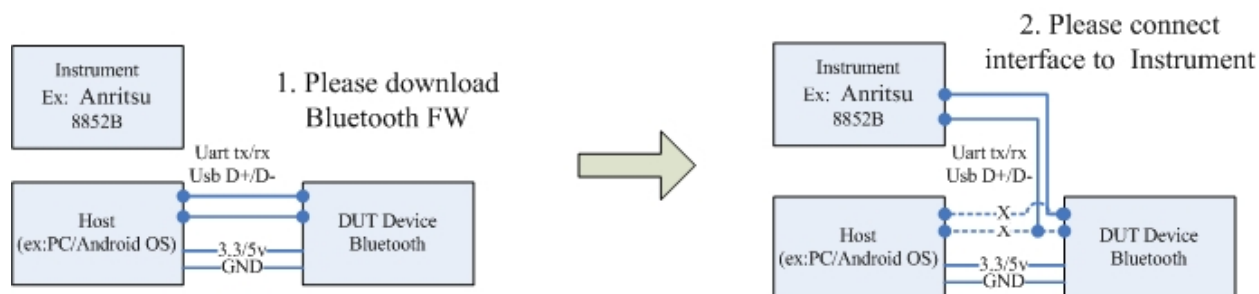


Figure 4 BLE Direct Test Mode

Step 1: Open Device

Step 2: download Bluetooth Patch Code

Step 3: **Jump hardware interface(UART) to the instrument.**

Step 4: Begin test.

BLE is a schematic diagram of the test for Certification:

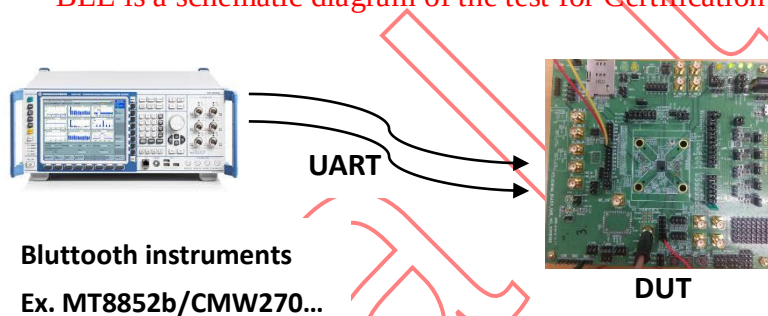


Figure 5 BLE Direct Test Mode connector

4.3 LE DUT Test Mode-TX/RX (MP)

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

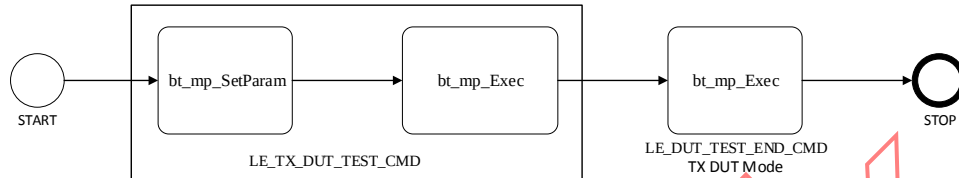


Figure 6 Start LE TX DUT test flow

Start LE RX DUT test mode, follow the steps below:

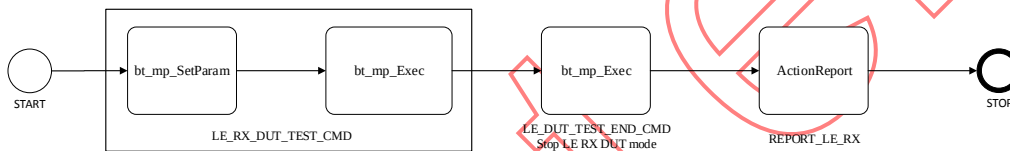


Figure 7 Start LE RX DUT test flow

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 RF Test Mode (Certification)

This chapter describes the APIs of some operations of DUT during authentication.

5.1 TX Test -Continuous Tx(Certification)

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

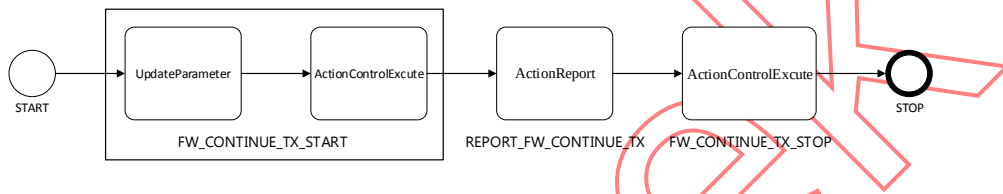


Figure 8 TX Test -Continuous Tx flow

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.

5.2 DUT Packet TX (MP)

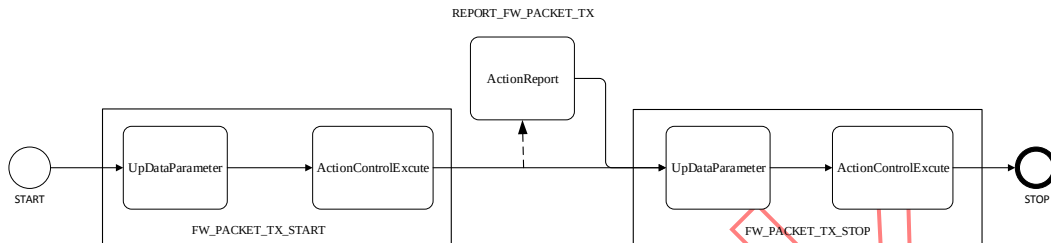


Figure 9 DUT Packet TX flow

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.

5.3 DUT Packet RX (MP)

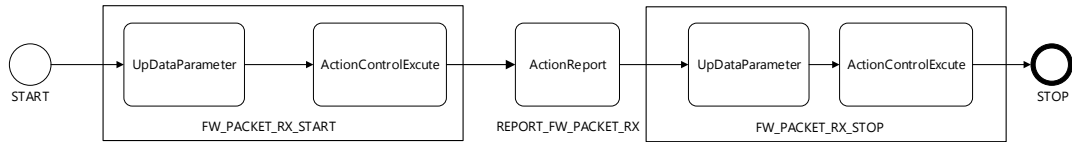


Figure 10 DUT Packet RX flow

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.

5.4 LE Certification TX Mode(Certification)

This function is only supported by some ICs .In order to facilitate RF safety testing, such as the band edge. We often use to facilitate continuous tx RF safety testing(eg.Bandedge).

Step1: UpDataParameter ; Channel Number = 0~39

INDEX	VALUE
1	ChannelNumber

Step2: ActionControlExcute (LE_TX_START_CMD)

Step4: ActionControlExcute (LE_TX_STOP_CMD)

5.5 Hopping Test (Certification- Dwell time)

In order to start hopping mode test, follow the steps below. **The function isn't to set Tx power level. The step by step in api's code show as:**

Step 1: Update Parameters(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

Table 10 Hopping mode:Packet type

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.

6 Mass Production Test Flow

The below diagram shows a global view of Bluetooth mass production flow:

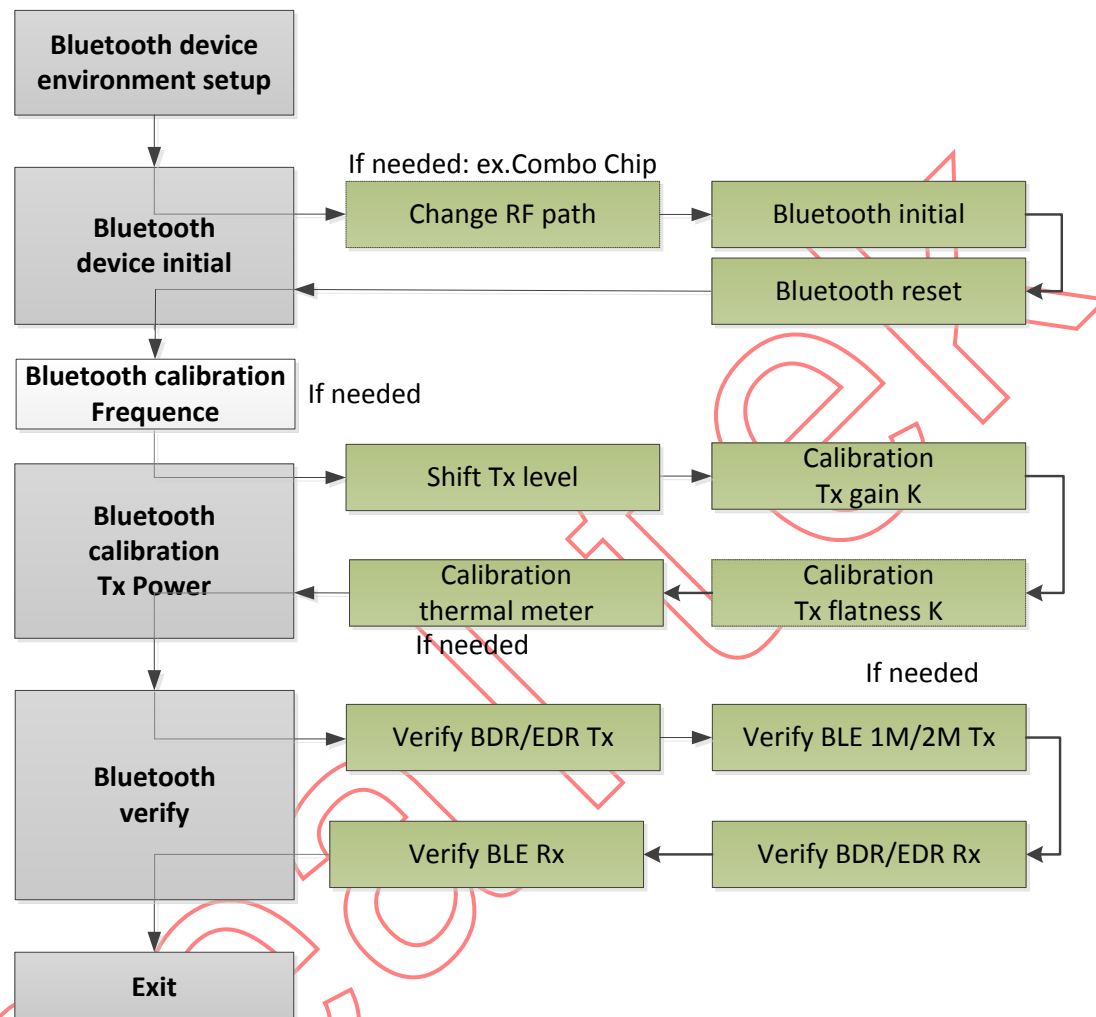


Figure 11 Mass Production Test Flow

6.1 Bluetooth Device Initial

In the initial stage, the Bluetooth device must set to factory default.

6.1.1 Change RF Path (antenna)

The API's use in combo chip, please use BT DLL of Realtek to set antenna path in Windows platform. The support of this function is dependent on the needs of the chip. Generally, it is used in combo chips. The simple code show as:

```
pParam->ParameterIndex = SET_ANT_INFO;
```

If antenna is S0:

```
pParam->mParamData[0] = 0;
```

If antenna is S1:

```
pParam->mParamData[0] = 1;
```

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

```
if (pBluetoothModule->ActionControlExcute(pBluetoothModule) !=0){
```

```
goto error;
```

```
}
```

6.1.2 Bluetooth Initial :Disable TX power tracking

If platform is Windows, please use BT DLL of Realtek to set it. Use the function to enable or disable thermal power tracking for chip. A simple example to use “API_ThermalPowerTracking” of RF dynamic link library is as follows:

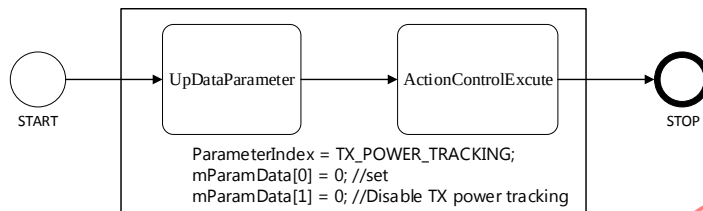


Figure 12 Disable TX power tracking flow

```

pParam->ParameterIndex = TX_POWER_TRACKING;
pParam->mParamData[0] = 0; //set
pParam->mParamData[1] = OnOff; //on(1) or off(0)
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
if (pBluetoothModule->ActionControlExcute(pBluetoothModule) !=0){
    goto error;
}
    
```

You can refer to “BTDevice_SetThermalPowerTracking(unsigned char OnOff) “ function of sample program.

6.1.3 Bluetooth Initial :Set TX Power Level To RAM

For verify TX performance need to setting temporarily set the current TX power level. You can use the APIs below to set the device to the ram.



Figure 13 Set TX Power Level To RAM flow

```

/*-----*/
#define HCI_VENDOR_SETTING_TX_LEVEL_TO_RAM 0xFCE7
unsigned char powerlevel[512]={0};
unsigned char pEvtBuf[512] = {0};

String text = ComboBox_MaxTXPowerofLegacy1M->Text.SubString(1,4);
powerlevel[0] = TX_level_1M;
powerlevel[1] = TX_level_2M;
powerlevel[2] = TX_level_3M;
powerlevel[3] = TX_level_BLE1M;
powerlevel[4] = TX_level_BLE2M;

if(pBluetoothModule->SendHciCommandWithEvent(pBluetoothModule,HCI_VENDOR_SETTING_TX_LEVEL_TO_RAM , LEN_5_BYTE, powerlevel, 0x0E, pEvtBuf))
{
    printf(">>>Set TX Power Level fail");
}
else
{
    printf(">>>Set TX Power Set TX Power Level successfully");
}
/*-----*/
  
```

The tx level value , you can refer Tx power Level table. The table by different Chip.

6.1.4 Bluetooth Reset: Set TX gain cal (K) value

If you need to calibrate to targeted power, then must reset the TX gain Cal Value. If platform is Windows, please set it through BT DLL of Realtek.

The TXGainK Value is a 2's complement.

Parameter Index	Param Data length	mParamData	
		0	
TX_POWER_GAIN_K	2	0x01 (Set)	TXgain cal Value

A simple example to use “API_ThermalPowerTracking” of RF dynamic link library is as follows:

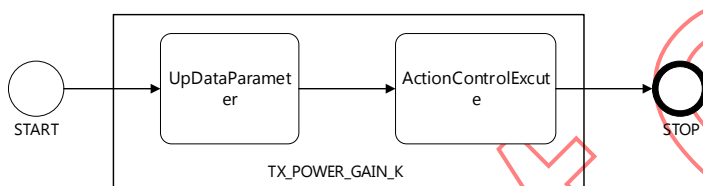


Figure 14 Set TX gain cal (K) value

```

pParam->ParameterIndex = TX_POWER_GAIN_K;
pParam->mParamData[0] = 0x01; //set
pParam->mParamData[1] = TXgain cal Value; //set value
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
if (pBluetoothModule->ActionControlExcute(pBluetoothModule) !=0){
    goto error;
}
  
```

If need to reset, then **TXgain cal Value is 0;**

6.1.5 Bluetooth Reset: Set TX flatness Value

If you need to calibrate TX power for each frequency, then must set flatness value to 0 first. If platform is Windows, please set it through BT DLL of Realtek.

The TX flatness Value is a 2's complement.

Mode	Index	length	mParamData				
			0	1	2	3	4
2 Byte	TX_POWER_G AIN_K	3	0x02 (Set)	Flatness Value LSB	Flatness Value MSB	X	X
4 Byte		5	0x02 (Set)	Flatness Value LSB		Flatness Value MSB	

Table 11 Set flatness Value command table

The flatness settings of APIs are defined below:

A simple example to use “API_ThermalPowerTracking” of RF dynamic link library is as follows:

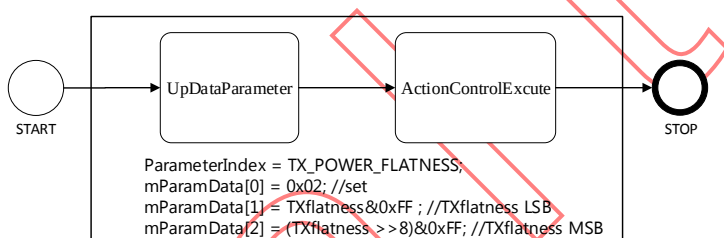


Table 12 Set flatness Value flow

pParam->ParameterIndex = TX_POWER_FLATNESS;

pParam->mParamData[0] = 0x02; //set

2 Byte mode	pParam->mParamData[1] = TXflatness &0xFF ;//LSB; pParam->mParamData[2] = (TXflatness >>8) &0xFF; //MSB;
4 Byte mode	pParam->mParamData[1] = TXflatness &0xFF ;//LSB; pParam->mParamData[2] = (TXflatness >>8) &0xFF; //LSB; pParam->mParamData[3] = (TXflatness>>16) &0xFF ;//MSB; pParam->mParamData[4] = (TXflatness >>24) &0xFF; //MSB;

pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);

if (pBluetoothModule->ActionControlExcute(pBluetoothModule) !=0){

goto error;

}

6.1.6 Bluetooth Reset::Set TX path loss module Value

If you need to calibrate TX power to target tx power , then must set the TX path loss module value.

The TX path loss module value is a 2's complement.

Parameter Index	Param Data length	mParamData	
		0	
TX_POWER_GAIN_K	2	0x03 (Set)	TX path loss module Value

Table 13 Set TX path loss module Value Table

The settings APIs define below:

```

pParam->ParameterIndex = TX_PATH_LOSS_MODULE;
pParam->mParamData[0] = 3; //set value
pParam->mParamData[1] = TX_path_loss_module Value; //set value
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
if (pBluetoothModule->ActionControlExcute(pBluetoothModule) !=0){
    goto error;
}

```

Note: Often in mass production, the RF of the board end is equal to the rf end of the chip, and this can be set to 0

6.2 Bluetooth calibration Frequency

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.

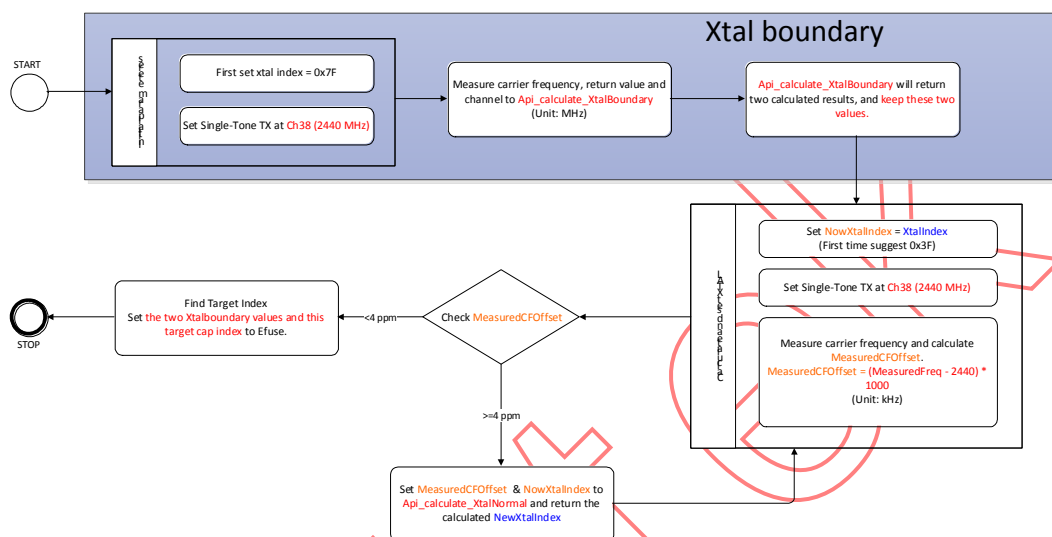


Figure 15 Search Crystal Capacitance Index Flow

Parameter Description of flow
XtalIndex : The default value is 0x3F. Index range is 0x0 to 0x7F.
NowXtalIndex : The current xtal cap index.
NewXtalIndex : The calculated xtal cap index from API.
MeasuredCFOffset : (Carrier frequency measured by instrument - Ideal carrier frequency) * 1000 (kHz). Target range Abs. MeasuredCFOffset value 4 ppm in 2.4GHz band is about ±10 kHz.
40M_boundary_1_cal : 40MHz boundary integer value calculated by Api_calculate_XtalBoundary from measured carrier frequency at ch38 (2440MHz) with xtal cap index = 0x7F

Note: The method of calibration depends on different chips.

Through the Realtek's dll, to complete the calibration process. These APIs are defined in later chapters.

6.2.1 Set Calibrates Crystal (Xtal) Cap

If need to set crystal cap then use this function to set it.

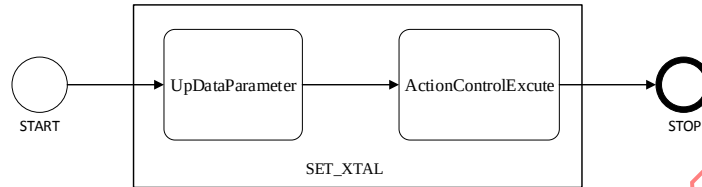


Figure 16 Set Calibrates Crystal (Xtal) Cap value flow

```

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

```

6.2.2 Single-Tone TX API sample

(1): Start Single-Tone Tx

```

pBtParam->mChannelNumber=(unsigned char)Channel&0xFF;
pParam->ParameterIndex = FW_SINGLETONE_TX_START;
UpDataParameter(pBtModule,pBtParam);
ActionControlExcute(pBtModule);

```

(2): Stop Single-Tone Tx.

```

pParam->ParameterIndex = FW_SINGLETONE_TX_STOP;
UpDataParameter(pBtModule,pBtParam);
ActionControlExcute(pBtModule);

```

6.2.3 Api_calculate_XtalBoundary

API define below:

Api_calculate_XtalBoundary	
Calculate the integer value of the xtal carrier frequency boundary.	
Syntax	
<pre>RTKBTMP_API int Api_calculate_XtalBoundary(BASE_BTMPDLL_MODULE *pDLLBtBaseModule, CAL_PARAMETER *pParameter, int cal_freq_MHz, double MeasureFreq_MHz, unsigned int *pData);</pre>	
Parameters	
* pDLLBtBaseModule	Structure BASE_BTMPDLL_MODULE point
* pParameter	Structure CAL_PARAMETER point
cal_freq_MHz	Integer , Array length of Patchcode
MeasureFreq_MHz	Double Value, Frequency deviation measured by the instrument.
pData	Return Array
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
1.Please include "bt_mp_calibration_base.h". 2.The return array length by different chip. 3. structure CAL_PARAMETER refer Appendix.	

Table 14 Api_calculate_XtalBoundary API

6.2.4 Api_calculate_XtalNormal

API define below:

Api_calculate_XtalNormal	
Calculate the integer value of the xtal carrier frequency boundary. This api is only supported by some chips.	
Syntax	
RTKBTMP_API int Api_calculate_XtalNormal (BASE_BTMPDLL_MODULE *pDLLBtBaseModule, CAL_PARAMETER *pParameter, double MeasureFreqOffset_KHz, unsigned int pInitCFCapValue, unsigned int *pNewCFCapValue);	
Parameters	
* pDLLBtBaseModule	Structure BASE_BTMPDLL_MODULE point
* pParameter	Structure CAL_PARAMETER point
cal_freq_MHz	Measured center frequency
MeasureFreq_MHz	Double Value, Frequency deviation measured by the instrument.
pInitCFCapValue	Current cap value
pNewCFCapValue	Retrun new Cap value
Return Value	
Status	0: The operation completed successfully. Other:Error
Remakes	
1.Please include "bt_mp_calibration_base.h". 2. structure CAL_PARAMETER refer Appendix.	

Table 15 Api_calculate_XtalNormal API

6.2.5 Bluetooth crystal boundary cap calibration

To adjust the integer value of the carrier frequency boundary measured at the upper limit of xtal (index = 0x7F). The flow show as:

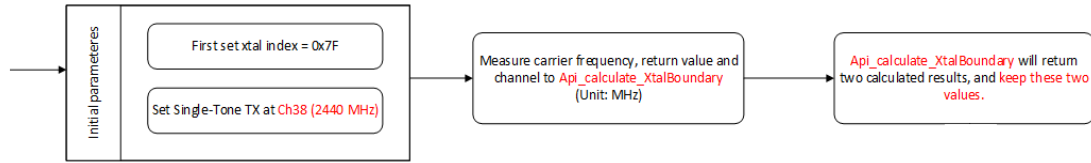


Figure 17 Bluetooth crystal boundary cap calibration flow

6.2.6 Bluetooth crystal cap calibration

If crystal source of Bluetooth comes from module, so customers just have to double check crystal source accuracy. The flow of crystal Accuracy Check is shown as below:

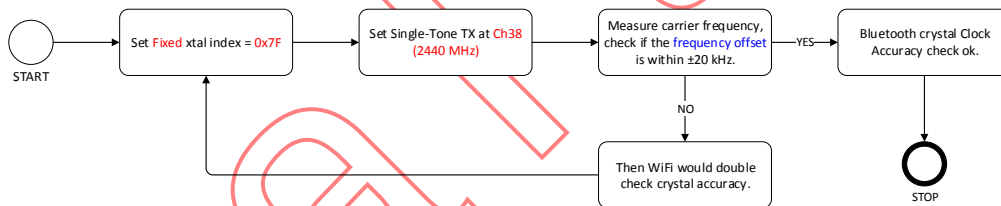


Figure 18 Crystal Accuracy Check Flow (Crystal Source Comes From module) flow

6.3 Bluetooth calibration Tx Power

Calibration Tx Power environment architecture for this process is defined.

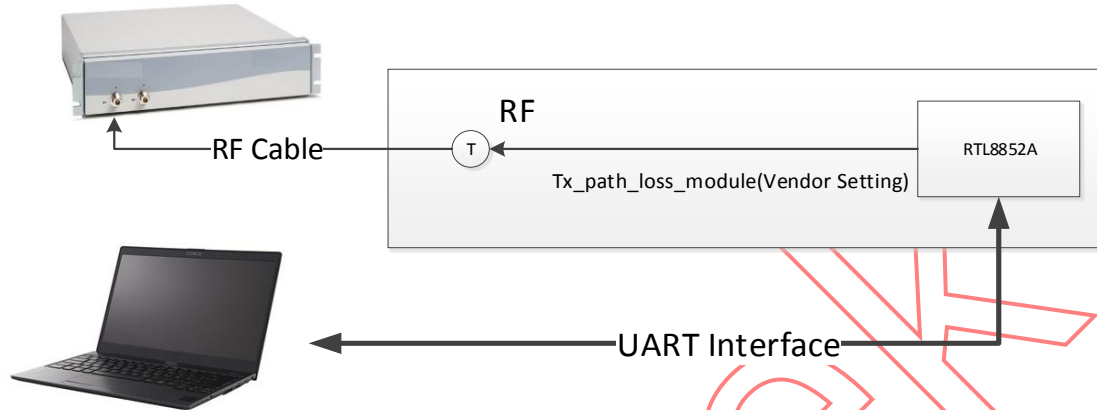


Figure 19 Bluetooth calibration Tx Power schematic

This simple diagrammatic sketch is as follows:

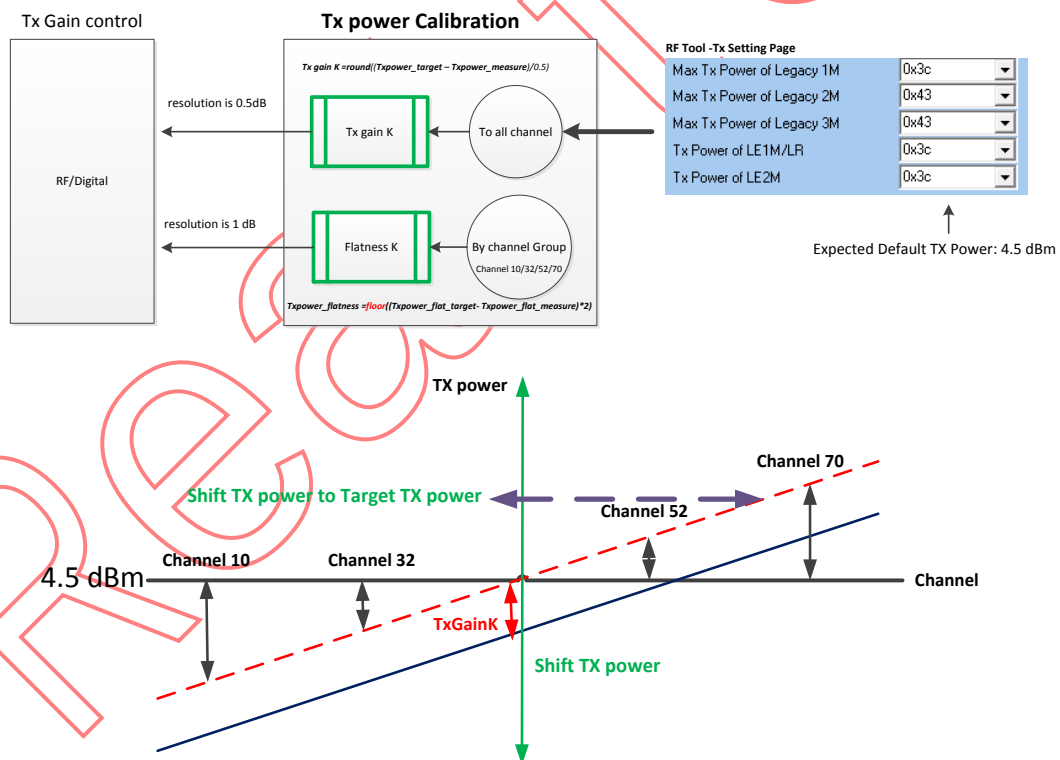


Figure 20 Bluetooth TX Power Calibration Diagram

6.3.1 Set Power TX Level

First, to shift the Tx power level to target power. The flow show as:

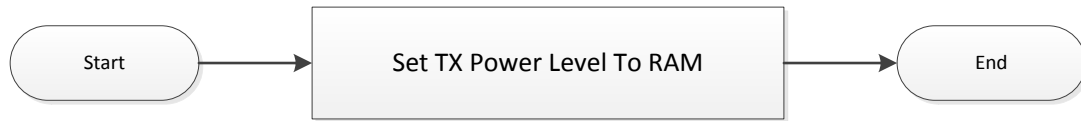


Figure 21 Set Power Tx Level flow

When the calibration is completed, please write the set target tx power value to the device.

6.3.2 Calibration TX gain K

Here demonstrate the flow about calibrating TX power to target power. Following plot adapted power target TX power as an example but **we suggest using desired target power as calibrated power in this step.**

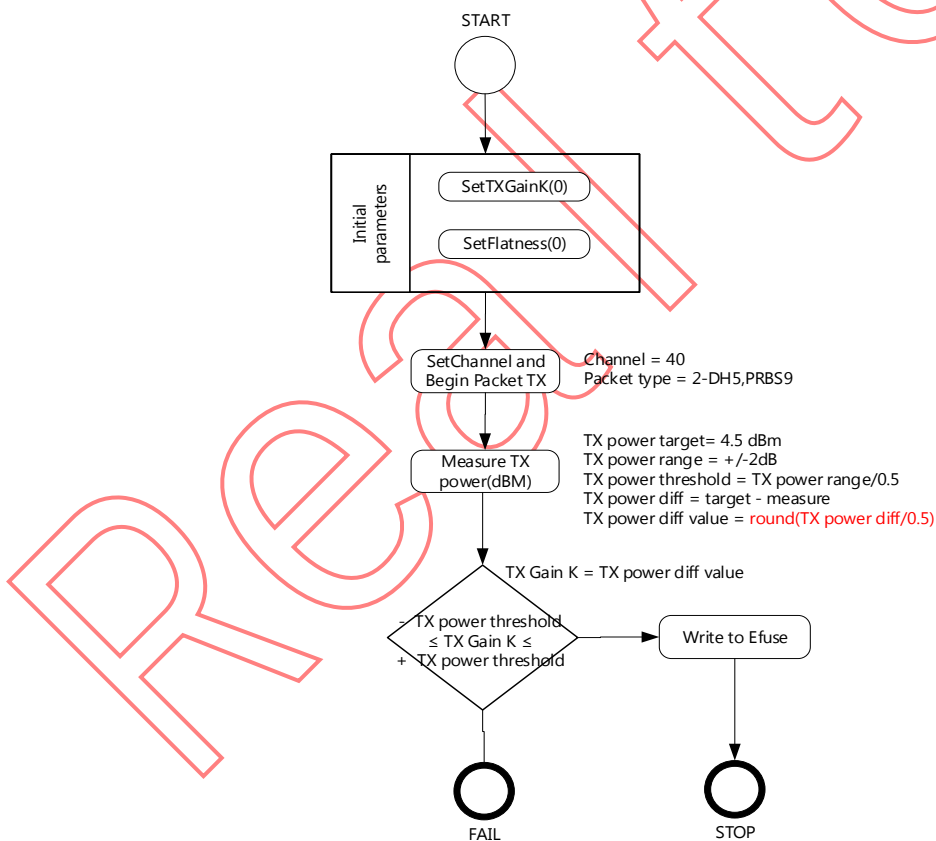


Figure 22 Calibration to Target Power

6.3.3 Calibration TX flatness K

Following plot adapted target tx power as an example but **we also suggest using desired target power as calibrated power in this step.**

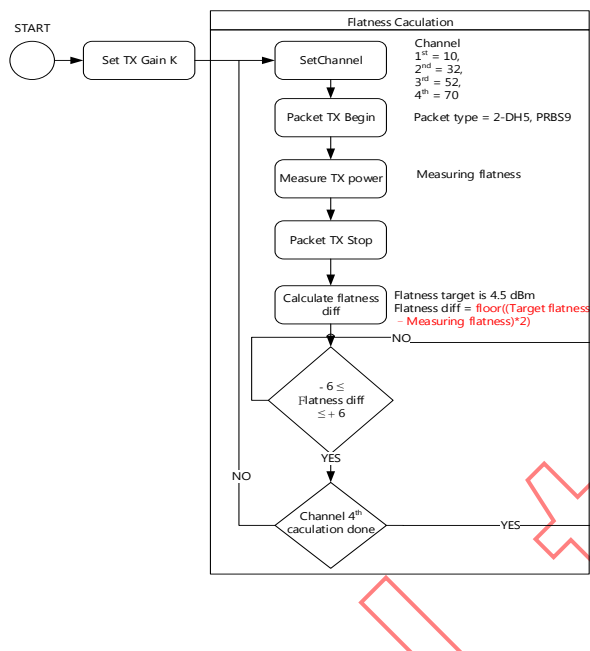


Figure 23 Bluetooth TX Power Flatness Calibration flow

The description of the process parameters:

TXpower_flat_measure[0~3]: Measure TX power.

TXpower_flat_target : target flatness power which is set to target tx power.

TXpower_flat_diff[0~3]: Measure each channel's difference between TX power measurement and target flatness TXpower. Defined as :

$$TXpower_flat_diff[0\sim3] = \text{floor}((TXpower_flat_target - TXpower_flat_measure[0\sim3]) * 2)$$

Calculate TX power flatness value:

➤ If flatness is 2 byte mode, then step is 0.5 dBm.

The TXpower_flatness value is defined as:

Bits	0~3	4~7	8~11	12~15
define	Low Flatness, ch0 ~ 21	Channel MidLow Flatness Ch22~ch43	Channel MidHigh Flatness ch44 ~ 61	Channel High Flatness Ch62 ~ 78

Table 16 The TX power flatness value define table(2byte)

For example:

Channel	Measured TX power	TX power flat_diff Calculate	TX power flat_diff	Value	Flatness Value
10	-0.6	$\text{floor}((0 - (-0.6)) * 2)$	1	0x1	0xDE01
32	-0.3	$\text{floor}((0 - (-0.3)) * 2)$	0	0x0	
52	0.7	$\text{floor}((0 - (0.7)) * 2)$	-2	0xE	
70	1.2	$\text{floor}((0 - (1.2)) * 2)$	-3	0xD	

Table 17 The TX power flatness value define sample table (2byte)

➤ If flatness is 4 byte mode, then step is 0.25 dBm.

The TXpower_flatness value is defined as:

Bits	0~7		8~15		16~23		24~31	
define	Low	Channel	MidLow	Channel	MidHigh	Channel,	High	Channl
	Flatness,		Flatness		Flatness		Flatness	
	ch0 ~ 21		Ch22~ch43		ch44 ~ 61		ch62 ~ 78	

Table 18 The TX power flatness value define table(4byte)

For example:

Channel	Measured TX power	TX power flat_diff Calculate	TX power flat_diff	Value	Flatness Value
10	-0.6	$\text{floor}((0 - (-0.6)) * 2)$	1	0x01	0x0D0E0001
32	-0.3	$\text{floor}((0 - (-0.3)) * 2)$	0	0x00	
52	0.7	$\text{floor}((0 - (0.7)) * 2)$	-2	0x0E	
70	1.2	$\text{floor}((0 - (1.2)) * 2)$	-3	0x0D	

Table 19 The TX power flatness value define sample table (4byte)

6.3.4 Bluetooth Calibrates Thermal (Tmeter) vaule

Normal driver will load thermal meter to do power tracking. So this value must be filled on correct eFuse location. Use the command below to read Bluetooth thermal meter once. If platform is Windows, please through BT DLL of Realtek to obtain raw thermal value. Please refer below sample code of use DLL:

```
/*-----*/

BT_DEVICE_REPORT *pModuleBtReport=&mBT_DEVICE_REPORT;
//Read Tmeter
pBluetoothModule->ActionReport(pBluetoothModule,REPORT_THERMAL,pModuleBtReport);
{
    printf(">>Get Thermal is Fail...!!\n");
    return;
}
printf(">>Thermal=(%d)\n",pModuleBtReport->CurrThermalValue);

/*-----*/
```

Note: The Thermal Value is RAW Data of Realtek define.

6.4 Bluetooth verify flow

6.4.1 Verify Bluetooth Legacy Tx Performance

To measure the DUT Bluetooth legacy TX power/initial Carrier offset/modulation characteristics to check Tx performance is ok or not. **The DUT parameter setting is determined by the pattern of the instrument.**

For Verify Bluetooth performance, there is a simple example of parameter setting below:

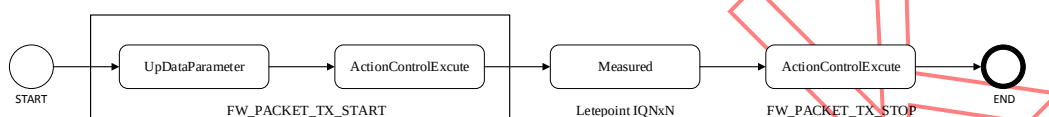


Figure 24 Verify Bluetooth Legacy Tx Performance flow

Bluetooth legacy 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
Verify Tx 3DH1	Initial Carrier Frequency Error		DH1	Low (CH6-2408MHZ)	-20KHz ~ 20KHz
				Middle (CH42-2444MHZ)	
				High (CH70-2472MHZ)	
	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 20 Bluetooth legacy TX criterion

step by step command of use api's:

(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=0x80; mTxGainIndex=0x7; mPacketHeader=0x33820; mHitTarget=0x0000009e8b33
DH1	Delta F1	mChannelNumber=0x06; mPacketType=0x00; mPayloadType=0x05; mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x33820; mHitTarget=0x0000009e8b33
DH1	Delta F2	mChannelNumber=0x06; mPacketType=0x00; mPayloadType=0x02; mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x33820; mHitTarget=0x0000009e8b33
3DH1	ALL	mChannelNumber=0x06; mPacketType=0x06; mPayloadType=0x07; mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x15C40; mHitTarget=0x0000009e8b33

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

		mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x33820; mHitTarget=0x0000009e8b33
DH1	Delta F1	mChannelNumber=0x46; mPacketType=0x00; mPayloadType=0x05; mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x33820; mHitTarget=0x0000009e8b33
DH1	Delta F2	mChannelNumber=0x46; mPacketType=0x00; mPayloadType=0x02; mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x33820; mHitTarget=0x0000009e8b33
3DH1	ALL	mChannelNumber=0x46; mPacketType=0x06; mPayloadType=0x07; mTxPacketCount=0x00; mWhiteningCoeffValue=0x80; mTxGainIndex=0x7; mPacketHeader=0x15C40; 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

6.4.2 Verify Bluetooth BLE4.0 Tx Performance

To measure the DUT BLE TX power and modulation index to check BLE TX performance is ok or not.

Bluetooth BLE TX criterion is shown as below:

	Test Item	Sub Test Item	Payload Type	Channel	Criterion
					Bluetooth Spec.
Verify BLE Tx	BLE Output Power	Average Power	PRBS9	Low (CH0-2402MHZ)	> 0dBm
				Middle (CH19-2440MHZ)	> 0dBm
				High (CH34-2470MHZ)	> 0dBm
	Modulation Characteristics	Delta F1 Avg.	BT_PAYLOAD_TYPE_1111_0000	2440MHZ	225 kHz ~ 275 kHz
		Delta F2 Max.	BT_PAYLOAD_TYPE_1010	CH19-	≥ 185 kHz
		Modulation Index	None		≥ 0.8

Table 21 Bluetooth BLE TX criterion(BT4.0)

For example, the step by step:

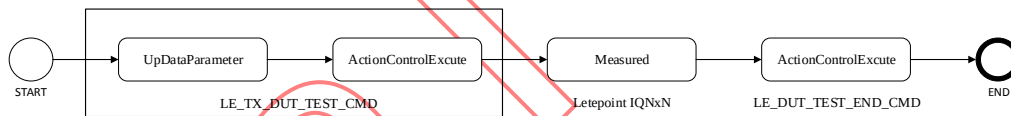


Figure 25 Verify Bluetooth LE TX Performance flow(BT4.0)

If interface of device is UART, then the step by step commands:

- (1) Enter MP Mode and download patch code
- (2) Set Parameter :

Test Item		adb command
	Test Item	Channel = 6
BLE	Average Power	mChannelNumber=0x00; //(change channel :0x19,0x34) mPayloadType=0x00; //Payload type = PRBS9 mTxGainIndex=0x6;
BLE	Delta F1	mChannelNumber=0x00; //(change channel :0x19,0x34) mPayloadType=0x01; //Payload type = 11110000 mTxGainIndex=0x6;
BLE	Delta F2	mChannelNumber=0x00; //(change channel :0x19,0x34) mPayloadType=0x02; //Payload type =10101010 mTxGainIndex=0x6;

6.4.3 Verify Bluetooth BLE5.0 Tx Performance

To measure the DUT BLE 5.0 TX power and modulation index to check BLE TX performance is ok or not. Bluetooth BLE 5.0 TX criterion is shown as below. **For vender's TX verification, please adapt target power value as your own spec.** Bluetooth Target tx power spec depends on different chips. Bluetooth BLE 5.0 TX criterion is shown as below:

PHY Spec	Test Item	Sub Test Item	Payload Type	Channel	Criterion Bluetooth Spec.
2M	BLE Output Power	Average Power	PRBS9	0/3/19/22/36/39	2.5 dBm ~ 6.5 dBm NOTE: Max variation = Target power value +/- 2 dB*
LR S2					
LR S8					
2M	Carrier freq. offset & drift	None	PRBS9	0/3/19/22/36/39	2M: < 20 kHz
LR S2			PRBS9		S8: < 19.2 kHz
LR S8			PRBS9		
2M	Modulation Characteristics	Delta F1 Avg.	BT_PAYLOAD_TY PE_1111_0000		2M: 450 ~ 550 kHz S8: 225 ~ 275 kHz
LR S2		Delta F2 Max.	BT_PAYLOAD_TY PE_1010		2M: > 370 kHz
LR S8		F2avg/F1avg	None		2M: > 0.8

Table 22 Bluetooth BLE 5.0 TX criterion

***Max variation spec only available after doing TX calibration flow**

For this test must to setting current tx power by Calibrates TX power. For example, the step by step 錯誤!
找不到參照來源。:

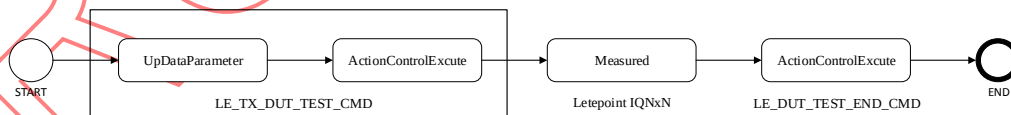


Figure 26 Verify Bluetooth LE TX Performance flow(BT5.0)

For example, the step by step :

(1) Set Parameter :

Test Item	Channel = 0/3/19/22/36/39
Average Power	pParam ->ParameterIndex =LE_TX_DUT_TEST_CMD pParam ->mChannelNumber=0x00;//channel pParam ->mPayloadType=0x00; //Payload type = PRBS9 pParam->PHY = 2 ; // 1:LE 1M 2:LE 2M 3: LE Coded PHY pParam->ModulationIndex = 1;

	// 0: standard modulation 1: stable modulation
Delta F1	pParam ->ParameterIndex =LE_TX_DUT_TEST_CMD pParam ->mChannelNumber=0x00;//channel pParam ->mPayloadType=0x01; //Payload type =00001111 pParam->PHY = 2 ; // 1:LE 1M 2:LE 2M 3: LE Coded PHY pParam->ModulationIndex = 1; // 0: standard modulation 1: stable modulation
Delta F2	pParam ->ParameterIndex =LE_TX_DUT_TEST_CMD pParam ->mChannelNumber=0x00;//channel pParam ->mPayloadType=0x02; //Payload type =10101010 pParam->PHY = 2 ; // 1:LE 1M 2:LE 2M 3: LE Coded PHY pParam->ModulationIndex = 1; // 0: standard modulation 1: stable modulation

pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);

(2)Execute active job - Run LE DUT TX

pBluetoothModule->ActionControlExcute(pBluetoothModule);

(3) Measured by Bluetooth test instrument(ex. Letepoint IQNxxN)

(4) Execute active job - Run Packet Stop

Ex.

ParameterIndex = FW_PACKET_TX_STOP

pBluetoothModule->ActionControlExcute(pBluetoothModule);

6.4.4 Verify Bluetooth Legacy Rx Performance

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 IQN_xN). Bluetooth Rx criterion is shown as below:

Verify Bluetooth Rx	Test Item	Packet Type	Criterion
	Sensitivity	DH1 or 3DH5	Bluetooth Spec
			< -70dBm

Table 23 Verify Bluetooth Legacy Rx Performance Spec

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:

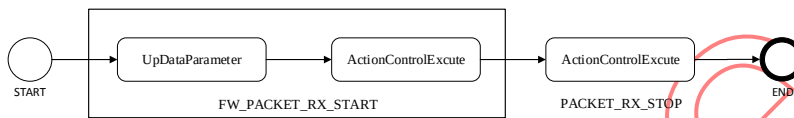


Figure 27 Verify Bluetooth Legacy Rx api's flow

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 28 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=0x80; mPacketHeader=0x33820; mHitTarget=0x000000c6967e
42	DH1	mChannelNumber=0x2a; mPacketType=0x00; mPayloadType=0x07; mWhiteningCoeffValue=0x80; mPacketHeader=0x33820; mHitTarget=0x000000c6967e
70	DH1	mChannelNumber=0x46; mPacketType=0x00;

		mPayloadType=0x07; mWhiteningCoeffValue=0x80; mPacketHeader=0x33820; mHitTarget=0x000000c6967e
6	3DH1	mChannelNumber=0x06; mPacketType=0x06; mPayloadType=0x07; mWhiteningCoeffValue=0x80; mPacketHeader=0x33820; mHitTarget=0x000000c6967e
42	3DH1	mChannelNumber=0x2a; mPacketType=0x06; mPayloadType=0x07; mWhiteningCoeffValue=0x80; mPacketHeader=0x15C40; mHitTarget=0x000000c6967e
70	3DH1	mChannelNumber=0x46; mPacketType=0x06; mPayloadType=0x07; mWhiteningCoeffValue=0x80; mPacketHeader=0x15C40; 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.

6.4.5 Verify Bluetooth BLE4.0 Rx Performance

Measure the DUT BLE Rx sensitivity to check Rx performance is ok or not. The Bluetooth Rx criterion is shown as below:

Channel	Payload Type	BLE sensitivity limit	Criterion Bluetooth Spec
Low (CH0-2402MHZ)	PRBS9	PER<= 30.800 %	< -70dBm
Middle (CH19-2440MHZ)			
High (CH34-2470MHZ)			

Table 24 The recommended test items of Bluetooth BLE Rx

To calculate the Packet Error Rate(PER)

$$\text{PER}\% = 100 * (1 - (\text{Packets Received} / \text{Packets Send}))$$

The step by step list below :

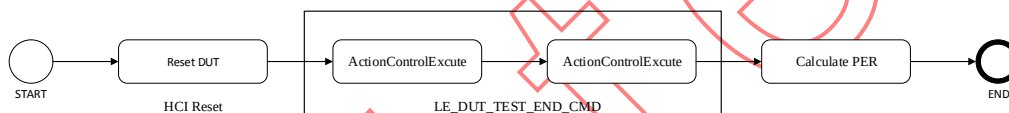


Figure 29 The recommended test items of Bluetooth BLE Rx flow

(1) Enter MP Mode and download patch code

(2) Set Parameter :

Test Item		adb command
Item	Payload type	
Sensitivity	PRBS9	mChannelNumber=0x00; (change channel :0x19/0x34)
		mPayloadType=0x00; //PRBS9

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

(4) Run LE Packet Rx

(5) Stop LE Packet Rx and to obtain the receive packet count

(6) Calculate PER

If (PER < limit) , goto step (3) , else Stop

6.4.6 Verify Bluetooth BLE5.0 Rx Performance

To calculate the Packet Error Rate(PER)

$$\text{PER}\% = 100 * (1 - (\text{Packets Received} / \text{Packets Send}))$$

Measure the DUT BLE Rx sensitivity to check Rx performance is ok or not. The Bluetooth Rx criterion is shown as below:

Channel	Spec	Payload Type	BLE sensitivity limit	Criterion
	PHY Spec			Bluetooth Spec
0/3/19/22/36/39	1M	PRBS9	PER<= 30.800 %	< -70 dBm
	2M	PRBS9	PER<= 30.800 %	< -70 dBm
	LRS2	PRBS9	PER<= 30.800 %	< -75 dBm
	LRS8	PRBS9	PER<= 30.800 %	< -82 dBm

Table 25 The recommended test items of Bluetooth BLE 5.0 Rx

The simple sample code of RF dll show as:

The step by step list below :

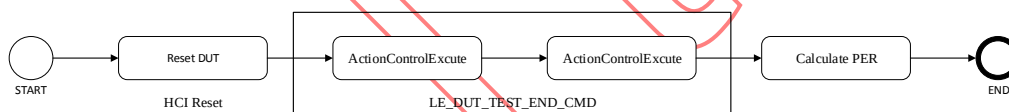


Figure 30 Verify Bluetooth BLE5.0 Rx

- (1) Enter Hci reset to reset the DUT.
- (2) Set Parameter & Run LE Packet Rx

The LE RX sample code below:

```

pParam->ParameterIndex = LE_RX_DUT_TEST_CMD; //23
pParam->mChannelNumber= 0; //channel :0~39
pParam->PHY = 2; // 1:LE 1M 2:LE 2M 3: LE Coded PHY
pParam->ModulationIndex = 1; // 0: standard modulation 1: stable modulation
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule

```

- (3) To setting Parameter with the Bluetooth test instrument ,then start TX until the end.
- (4) Stop LE Packet Rx and to obtain the receive packet count

```

pParam->ParameterIndex = LE_DUT_TEST_END_CMD; //23
pBluetoothModule->UpDataParameter(pBluetoothModule,pParam);
pBluetoothModule->ActionControlExcute(pBluetoothModule

```

- (1) Calculate PER:

If (PER< limit) , goto step (2) , else to To finish.

7. Write EFUSE Data to Bluetooth Device(BT Controller)

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
RTL8761B series

Table 26 Write EFUSE support chips

7.1 Write BT MAC Address to EFUSE

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

- 1) Byte0 and Byte 1 can't be 0xFF at the same time.
- 2) Byte2 and Byte 3 can't be 0xFF at the same time.
- 3) Byte4 and Byte5 can't be 0xFF at the same time.

Byte Number	5	4	3	2	1	0
Range	0~0xFFFE		0~0xFFFE		0~0xFFFE	

If you want to use 0xFFFF as Bluetooth MAC address, you must first write a non-0xFFFF value and then write 0xFFFF.

For write to efuse 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.

Chip	Offset	Default value	Example value	Description
RTL8761A	0x3C-0x42	0x00E04C829987	0x00E023345678	BD_ADDR
RTL8761B	0x30-0x35	0x00E04C829987	0x00E023345678	BD_ADDR

The efuse offset by chip defined.

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]);

```

7.2 Write Power On Function to EFUSE

The function only support RTL8761A Chip. 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]	
Power on Level Array	0x01	0x30	0x00	0x02	0x40[0x52]	0x52[0x50]

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) ;
```

7.3 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.

Chip	Offset	Default value	Example value	Description
RTL8761A	0x01E6	0x20	0x00~0x3F	XTAL_K Value Bit[5:0], Xi, Xo Range
RTL8761B	TBD	TBD	TBD	TBD

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
CHIP	Description	Sub-Index	Offset (Low)	Offset (High)	Length	Xtal value
Rtl8761A	Crystal Cap Calibration Array	0x01	0xE6	0x01	0x01	0~63
RTL8761B						

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];
```

7.4 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) ;

```

8. Appendix: Define

8.1 INTERFACE_TYPE

Interface	Index	Note
USB	0	
UART	1	
FILTER UART	2	Realtek Special mode
SDIO	3	
WINUSB	4	
ADB UART	5	Realtek Special mode
ADB USB	6	
SOCET UART	7	
SOCET USB	8	
BUMBLE BEE UART	9	

Table 27 Interface type index

The enum define is :

```
enum
{
    TYPE_USB = 0,
    TYPE_UART,
    TYPE_FILTER_UART,
    TYPE_SDIO,
    TYPE_WINUSB,
    TYPE_ADB_UART,
    TYPE_ADB_USB,
    TYPE_SOCKET_DEVICE_UART,
    TYPE_SOCKET_DEVICE_USB,
    TYPE_BUMBLE_BEE_USB,
    TYPE_NUMBER
}INTERFACE_TYPE;
```

8.4 BT_ACTIONCONTROL_TAG

Command	Index	Support Chip									
		RTL8723B RTL8761A RTL8821A	RTL8723D	RTL8822B	RTL8821C	RTL8763B	NBIO7/Ameba	RTL8822C	RTL8761B	lastt	
HCI_RESET	0	■	■	■	■	■	■	■	■	■	
TEST_MODE_ENABLE	1	■	■	■	■	■	■	■	■	■	
WRITE_EFUSE_DATA	2	■ (RTL8761A)							■		
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	■	■	■					■		
LE_CONTINUE_TX_START	28	■									
LE_CONTINUE_TX_STOP	29	■									
FW_PACKET_TX_START	30		■	■	■	■		■	■	■	
FW_PACKET_TX_STOP	31		■	■	■	■		■	■	■	

FW_PACKET_RX_START	32		■	■	■	■	■	■	■	■
FW_PACKET_RX_STOP	33		■	■	■	■	■	■	■	■
FW_CONTINUE_TX_START	34		■	■	■	■	■	■	■	■
FW_CONTINUE_TX_STOP	35		■	■	■	■	■	■	■	■
FW_LE_CONTINUE_TX_START	36		■	■	■	■	■	■	■	■
FW_LE_CONTINUE_TX_STOP	37		■	■	■	■	■	■	■	■
FW_READ_TX_POWER_INFO	38		■	■	■	■	■	■	■	■
SET_GPIO3_0	39	■ (RTL8761A)	■							
SET_ANT_INFO	40		■					■		
SET_ANT_DIFF_S0S1	41		■							
TX_POWER_TRACKING	42		■	■	■	■	■	■	■	■
SET_K_TX_CH_PWR	43		■	■				■		
WRITE_FLASH_CONFIG	44					■				
TX_POWER_GAIN_K	45					■	■	■	■	■
TX_POWER_FLATNESS	46					■	■	■	■	■
TX_PATH_LOSS_MODULE	47					■	■	■	■	■
CONFIG_EXTEND(Flash write)	48					■				TBD
READ_FLASH_CONFIG	49									TBD
UNLOCK_8822C	50							■		
SetTxPower_8822C	51					■	■	■	■	■

Table 28 BT_ACTIONCONTROL define index

Note :Function support is dependent on different CHIPS. Please ask FAE.

- *GET_MAX_TXPOWER_INDEX*

GET_MAX_TXPOWER_INDEX is a special Vendor command. This command is able to get device maximum Tx power index(eg. MAX_TXPOWER_INDEX).

- *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 “0x80”, 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 *MAX_TXPOWER_INDEX*. Only when TxGainIndex is out of range, TxGainValue is meaningful.

Realtek Bluetooth Chip Number	<i>MAX_TXPOWER_INDEX</i>
RTL8723B series	7
RTL8761A series	7
RTL8821A series	7
RTL8703B series	7
RTL8723D series	<i>GET_MAX_TXPOWER_INDEX Command</i>
RTL8822B series	<i>GET_MAX_TXPOWER_INDEX Command</i>
RTL8821C series	<i>GET_MAX_TXPOWER_INDEX Command</i>
RTL8822C series	<i>GET_MAX_TXPOWER_INDEX Command</i>
RTL8852A series	<i>GET_MAX_TXPOWER_INDEX Command</i>
RTL8723F	<i>GET_MAX_TXPOWER_INDEX Command</i>
Other	<i>GET_MAX_TXPOWER_INDEX Command</i>

8.5 PKT_TYPE

The packet types are defined in Table PKT_TYPE:

Legacy:

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
BT_PKT_TYPE_NULL	10	None
BT_PKT_LE_2M	11	0~255
BT_PKT_LE_CODED_S8	12	0~255
BT_PKT_LE_CODED_S2	13	0~255

Table 29 PKT_TYPE for BDE/EDR,Hopping index

BLE 5.0 Tx :NAME	INDEX
BLE5_TX_1M_PHY	1
BLE5_TX_2M_PHY	2
LE5_TX_CODED_PHY_S8	3
LE5_TX_CODED_PHY_S2	4
BLE 5.0 RX NAME	INDEX
BLE5_TX_1M_PHY	1
BLE5_TX_2M_PHY	2
LE5_TX_CODED_PHY_LR	3

Table 30 BLE PKT_TYPE index

8.6 BLE_MODULATION_TYPE

NAME	INDEX	
STANDARD_MODULATION_INDEX	0	
STABLE_MODULATION_INDEX	1	

Table 31 BLE_MODULATION_TYPE

8.7 PAYLOAD_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 32 PAYLOAD_TYPE index for BDR/EDR

NAME	INDEX
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

Table 33 PAYLOAD_TYPE index for BLE

8.8 PACKET_HEADER

Packet type	Payload(Bits)	Packet Header Hex
DH1	216	33820
DH3	1464	39858
DH5	2712	A078
2DH1	432	33820
2DH3	2936	C050
2DH5	5432	3F870
3DH1	664	15C40
3DH3	4416	39858
3DH5	8168	A078

Table PACKET_HEADER

8.9 Report ActiceItem

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

Table 34 Report Actice Item index

8.11 TX Gain K Define Table

TX gain K value (The unit is 0.5 dB per step.)	Value (2's complement)
12	0x0C
...	...
4	0x04
3	0x03
2	0x02
1	0x01
0	0x00 (default)

Table 35 TX Gain K Define Table

8.12 Flatness Define Table

TXpower_flat_diff_value	Efuse Flatness value
6	0x6
5	0x5
4	0x4
3	0x3
2	0x2
1	0x1
0	0x0
-1	0xFF
-2	0xFE
-3	0xFD
-4	0xFC
-5	0xFB
-6	0xFA

Table 36 Flatness Define Table

9. Appendix: Structure

9.1 BASE_INTERFACE_MODULE :Interface Base structure

```
#define MAX_DATA_LEN 20
```

```
// Base interface module structure  
struct BASE_INTERFACE_MODULE_TAG
```

```
{
```

```
    BASE_FP_OPEN Open;
```

```
    BASE_FP_SEND Send;
```

```
    BASE_FP_RECV Recv;
```

```
    BASE_FP_CLOSE Close;
```

```
    BASE_FP_WAIT_MS WaitMs;
```

```
    BASE_FP_SET_USER_DEFINED_DATA_POINTER SetUserDefinedDataPointer;
```

```
    BASE_FP_GET_USER_DEFINED_DATA_POINTER GetUserDefinedDataPointer;
```

```
    unsigned char InterfaceType;
```

```
    // User defined data
```

```
    unsigned long UserDefinedData;
```

```
    //for usb , uart
```

```
    unsigned char PortNo;
```

```
    //for uart
```

```
    unsigned long Baudrate;
```

```
    unsigned char bUartProtocol;
```

```
    unsigned char pData[MAX_DATA_LEN];
```

```
};
```

9.2 BT_PARAMETER Structure

```

struct BT_PARAMETER_TAG
{
    unsigned long ParameterIndex;

    unsigned char mPGRawData[MAX_USERAWDATA_SIZE];
    unsigned char mParamData[MAX_USERAWDATA_SIZE];
    unsigned char mChannelNumber;
    unsigned long mPacketType;
    unsigned char mTxGainIndex;
    unsigned char mTxGainValue;
    unsigned long mTxPacketCount;
    unsigned long mPayloadType;
    unsigned long mPacketHeader;
    unsigned char mWhiteningCoeffValue;
    unsigned char mTxDAC;
    ULONG64 mHitTarget;

    unsigned char TXGainTable[MAX_TXGAIN_TABLE_SIZE];
    unsigned char TXDACTable[MAX_TXDAC_TABLE_SIZE];

    unsigned char bHoppingFixChannel;
    unsigned long Rtl8761Xtal;
    unsigned char ExeMode;
    unsigned char PHY; //for Le Enhanced bt 5.0
    unsigned char ModulationIndex; //for Le Enhanced bt 5.0
};

```

Name	Value Range	Description
ParameterIndex	See BT_ACTION_CONTROL MAX Value	See BT_ACTION_CONTROL_TAG
mPGRawData	Raw data	
mChannelNumber	0~78	None
mPacketType	0~8	See PKT_TYPE
mTXGainIndex	0x00 (for default TX power)	
mTXPacketCount (for packet TX)	0~0xFFFFFFFF	0 : infinite TX packet count
mPayloadType	0~7	See BR/EDR

		PAYLOAD_TYPE and BLE
		PAYLOAD_TYPE
mPacketHeader	0x0~0x3FFFF	See PACKET_HEADER
mWhiteningCoeffValue	0x80 or 0x7F	0x7F : Enable Whitening 0x80 : Disable Whitening
mHitTarget	6 bytes	Instrument
bHoppingFixChannel	0 : Disable 1 : Enable Fix Channel	BD_ADDRESS For Hopping mode parameter
Rtl8761Xtal	0~0x7F (default 0x3F)	Crystal Cap Value

Table 37 UpDataParameter define: BT_PARAM

9.3 BASE_CAL_PARAMETER

```
#define MAX_BYTES_ARRAY_SIZE 4096
#define MAX_DOUBLE_ARRAY_SIZE 128

typedef struct BASE_CAL_ARRAY_TAG{
    unsigned Array8bit[MAX_BYTES_ARRAY_SIZE];
    double    ArrayDouble[MAX_DOUBLE_ARRAY_SIZE];
}BASE_CAL_ARRAY_TAG;

typedef struct BASE_CAL_RTK_TXPOWERK_PARAMETER_TAG{
    unsigned char TxGainKVlaue;
    unsigned int  TxFlatnessKValue;
}BASE_CAL_RTK_TXPOWERK_PARAMETER_TAG;

struct BASE_CAL_PARAMETER_TAG
{
    int Index;
    int chipID;
    BASE_CAL_ARRAY_TAG input;
    BASE_CAL_ARRAY_TAG output;
    BASE_CAL_RTK_TXPOWERK_PARAMETER_TAG txpowK;
    //////////////////////////////////////
    // Function point define
    BASE_FP_CAL_MEASUREMENT_CW fpMeasurementCW;
    BASE_FP_CAL_MEASUREMENT_TX fpMeasurementTX;
    void *pUserdefinePoint;
    void **ppUserdefineFuncPoint;
};
#endif //end #ifndef BASE_CAL_DEFINE
```

Realtek