Qualcomm Technologies, Inc.

Linux Android Bluetooth Debug and Configuration

User Guide

80-Y8113-9 Rev. J

March 7, 2022

For additional information or to submit technical questions, go to https://createpoint.qti.qualcomm.com

Confidential – Qualcomm Technologies, Inc. and/or its affiliated companies – May Contain Trade Secrets

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	September 2014	Initial release
B	October 2014	Restructured document to create a more task-based process flow; added a new Chapter 2 to include the following logging information: ■ WCN QXDM Logs ■ RAM dump log ■ Over-The-Air (OTA) sniffer log
C	February 2015	■ Included Logging details required for QCA61x4/QCA65x4/QCA937x ■ Changed the doc title to generic
D	March 2015	■ Included SMD Logging details
E	September 2015	■ Added Section PR3.0 Logging Enhancements
F	April 2016	■ Added Section Android Configuration
G	August 2018	Updated to conform to QTI standards; no technical content was changed in this document revision. Read the document in its entirety.
H	April 2020	Added the following chapters: ■ Chapter 11 Transport driver and issue triaging ■ Chapter 12 Bluetooth bring-up
Revision I was omitted in accordance with QTI document conventions.		
J	February 2022	Updated the following sections: ■ *Section 3.1 Bluetooth firmware dump* to update the log command ■ *Section 3.2 Ring buffer logs* to debug command

Contents

Tables

Figures

1 Android Bluetooth debug overview

Use this document to debug some of the common Bluetooth issues using the following:

- Android host logs (ADB logcat)
- Bluetooth logs
- Audio profile logs
- Data profile logs
- GAP and core stack logs
- BLE stack logs

This document can also be used to configure the Android properties and to debug SplitA2DP issues.

NOTE Corner cases are not included in this document.

2 Log Android host

Use the following table to debug the logs.

Table 2-1 Log debug table

	logcat	BTSnoop	Kmsg	QXDM	OTA
General IOT	Required	Required	Not required	Not required	Recommended
Bluetooth SoC	Not required	Recommended	Not required	Required	Required
Bluetooth on/off	Required For QCA61x4/ QCA65x4/ QCA937x, UART log is also required.	Not required	Not required	Required	Not required
SCO Audio	Recommended	Not required	Not required	Required	Required
Android crash	Required	Not required	Required	Not required	Not required
Firmware crash	Not required	Not required	Not required	Required	Recommended

WCNSS firmware crash requires RAMDUMP file, which is auto-generated in `/data/vendor/ssrdump/*`

Additional logs, which include the following, are required for Android crash:

- tombstone log (/data/tombstones/*)
- anr log (/data/anr/*)
- vmlinux binary file
- symbols files (<Source code>/out/target/product/MSM™*/symbols/*)

2.1 Android host logs in Linux

Android Debug Bridge (ADB) logcat captures Android system logs in the Linux user space.

The Android debug bridge logcat also captures the following items:

- Bluetooth-related applications such as device settings
- Framework such as A2DP service
- Bluedroid stack logs

2.2 Capture logs with timestamp enabled

Run the following command to capture the logcat logs with the timestamp enabled:

```
adb logcat -v threadtime
```

2.3 Enable and capture ADB logcat and BTsnoop logs

For any Bluetooth issue, capture the ADB logcat and BTSnoop (Bluedroid stack) logs.

The BTSnoop log determines whether the issue is related to application/Bluetooth host/Bluetooth SoC, or to a remote device.

To capture the logcat log with timestamp enabled, execute the following command:

```
#adb shell logcat –vtime
```

The process for snoop logging sends all the snoop packets to a new snoop process for dumping. By default, snoop logging is enabled from config file bt_stack.conf file.

Use the bt_stack.conf file to enable BTSnoop configuration. Snoop is enabled by default on userdebug builds. A generated BTSnoop log is at /sdcard/ hci_snoop**.cfa.

NOTE You must enable BTSnoop log by setting BtSnoopConfigFromFile to true. Enable BTSnoop override by setting BtSnoopConfigFromFile to False.

2.3.1 Enable Fluoride BTSnoop log

To enable Fluoride BTsnoop log, do the following:

1. Set the BtSnoopLogOutput to true in bt_stack.conf file, located in etc/bluetooth/…
2. Set the BtSnoopConfigFromFile flag to true.

   A generated BTSnoop log can be found at /sdcard/btsnoop_hci.log.

```
# Enable BtSnoop logging function
# valid value : true, false
BtSnoopLogOutput=true
# BtSnoop log output file
BtSnoopFileName=/sdcard/btsnoop_hci.log
```

2.4 Enable Fluoride BTSnoop log

To enable Fluoride BTsnoop log, do the following:

1. Set the BtSnoopLogOutput to true in bt_stack.conf file, located in etc/bluetooth/…
2. Set the BtSnoopConfigFromFile flag to true.

   A generated BTSnoop log can be found at /sdcard/btsnoop_hci.log.

```
# Enable BtSnoop logging function
# valid value : true, false
BtSnoopLogOutput=true
# BtSnoop log output file
BtSnoopFileName=/sdcard/btsnoop_hci.log
```

2.5 Capture UART kernel space logs

To capture a Linux kernel message containing UART kernel space log related to Bluetooth on/off cycle, execute the following commands:

```
adb root
adb remount
cd sys/module/MSM_serial_hs/parameters
echo 4 > debug_mask
Ctrl+C (exit)
adb shell
```

cat /sys/kernel/debug/ipc_logging/MSM_serial_hs/log_cont > c:\temp\uart.txt

2.5.1 Enable kernel space logging

Kernel space logging can be enabled for BlueZ and QCA61x4/QCA65x4/QCA937x based firmware.

- For BlueZ kernel space log related to Bluetooth on/off cycle, run the following command:
  `# adb shell cat /proc/kmsg`
- For QCA61x4/QCA65x4/QCA937x based firmware, run the following command:
  # adb root

```
# adb shell "cat /proc/kmsg" > c:\temp\kernellog.txt
```

2.6 Enable SMD logs

SMD channel names for Bluetooth include the following:

- APPS_RIVA_CTRL for HCI command packets
- APPS_RIVA_DATA for HCI ACL packets

1. Mount debugfs to get SMD channel table:

```
# mount –t debugfs none /sys/kernel/debug
# cat /sys/kernel/debug/smd/tbl
```

2. List the SMD ports that are currently allocated in the following format

   ```
   "name=WLAN_CTRL cid=8 ch type=6 xfer type=2 ref_count=1" shows that WLAN uses SMD channel 8 with name as WLAN_CTRL.
   # cat /sys/kernel/debug/smd/ch
   ```

3. Check the state of all the allocated SMD ports. The message is in the following format:

   ```
   "ch<cid>: <apps port state><read index/write index> <flags> <--> <remote port state><read index/write index> <flags>".
   ```

For example: "ch08: OPENED(3807/3807) DCCiwrsb <-> OPENED(1194/1194) DRCiwrsb: 2000" shows that the channel 8 used by WLAN and the read/write index match.

SMD channel names for Bluetooth include the following:

- adb shell cat /d/ipc_logging/smd_tty/log_cont > c:\temp\smd_tty_log.txt
- adb shell cat /d/ipc_logging/smd/log_cont > c:\temp\smd_log.txt

For targets that do not support IPC logging, do the following:

1. Enable SMD debug logging

   ```
   echo 1 > /sys/module/smd/parameters/debug_mask
   ```

2. Enable SMD INFO logging

   ```
   echo 4 > /sys/module/smd/parameters/debug_mask
   ```

3. Enable SMD INFO+DEBUG logging

   ```
   echo 5 > /sys/module/smd/parameters/debug_mask
   ```

3 Log Bluetooth firmware dump and related logs

If the Bluetooth firmware crashes, capture the Bluetooth firmware crash dump, along with UART IPC logs and host side ring buffer logs. UART IPC logs and ring buffer logs are not enabled by default. Set the specific property variables to capture them.

Following are the details of property variables. Set these property variables by default for user debug builds:

- If the SSR is enabled, and there is a Bluetooth crash, the host driver collects the Bluetooth firmware dump and other logs.
- If the SSR is disabled, the host triggers the system crash, after collecting firmware dump and logs. By default, the SSR is enabled in the builds.

For testing, connect to QXDM by USB from the command output window and type the following string into 'Command':

```
Software error fatal by ERR_FATAL()
Send_data 75 3 7 0 1 12 252 01 38
```

3.1 Bluetooth firmware dump

If Bluetooth firmware crashes, the Bluetooth dump is saved and one can retrieve the logs using the following command:

```
#adb pull /data/vendor/ssrdump/ramdump_bt_fw_crashdump_<timestamp>.bin
```

3.2 Ring buffer logs

It is difficult to conclude whether the data is corrupted at Bluetooth host, UART kernel, or Bluetooth SoC.

To debug the data corruption issue, enable the data sent from Bluetooth driver by setting the following command in the Kernel:

```
CONFIG_IPC_LOGGING
```

Run the following command to pull the logs along with crash dump:

```
#adb pull /data/vendor/ssrdump/ramdump_bt_uart_ipc_*
```

3.3 Disable SSR

Enter the following adb command to disable the SSR, which enables collecting the full system dump. Unlike, when SSR is enabled, in this case, the system is unresponsive and collects entire phone dump and reboots the phone.

```
#adb shell setprop persist.service.bdroid.ssrlvl 1
```

3.4 Enable SSR

To enable SSR back, use the following adb command:

```
#adb shell setprop persist.service.bdroid.ssrlvl 3
```

4 Log Bluetooth stack

The Bluetooth firmware debug logs are sent over a diagnostics interface from the SoC to the APSS diagnostics server using a dedicated SMD/UART channel.

Bluetooth firmware debug logs include the following:

- Power state (sleep/wakeup) – Known messages/Bluetooth.
- Error/INTR status – Known messages/Bluetooth.
- HCI packet logs – Known log items/Common/Bluetooth.
- LMP logs – Known Over-the-Air types/Bluetooth.
- Bluetooth Synchronous Connection-Oriented (SCO) audio packets (PCM and CVSD) – Known log items/Common/Bluetooth/Wireless connectivity subsystem for WCN only

Logs are captured using the following:

- QXDM with a USB connection.
- Device without a USB connection.

4.1 Enable firmware logging for QCA61x4/QCA65x4/QCA937x

To enable firmware logging:

1. Run the following commands:

```
# adb root
# adb shell setprop persist.service.bdroid.snooplog true
# adb shell setprop persist.service.bdroid.soclog true
```

2. Turn on Bluetooth

4.1.1 Capture QXDM logs

To capture QXDM logs, do the following:

1. Launch QXDM.
2. Select **Options** > **Communication** to select the target port to connect with Android Device Under Test (DUT). The Communications window appears.
3. Select the target port. Click **OK**.
4. Select **File** > **Load Configuration** to load the correct mask file (.dmc) with the required log mask enabled to begin the logging.

4.1.2 Create a logging mask file using QXDM

The on-device logging function logs diagnostic traffic to the device file system on the SD card.

To create a logging mask file using QXDM, do the following:

1. Open **QXDM.**
2. Go to **Filtered View.**
3. Right-click the item viewer and select **Config**.
4. Select the required items and enable **Save Diag** masks into DIAG.cfg.
5. Copy the mask file to the SD card.
6. Use adb push to copy the file to /sdcard/diag_logs.
7. Convert output file.

diag_mdlog is an application on the Android platform which writes data to the file in the .qmdl format.

QCAT can open the logs (.qmdl) to save as a .dlf file.

To convert .qmdl files to .isf files (.qmdl > .dlf > .isf), do the following:

1. Load the .qmdl file and save it as a .dlf file in APEX/QCAT.
2. Use DLFConverter (bundled with QXDM) to convert from .dlf to .isf.

4.1.3 Enable On-device logging for QCA61x4/QCA65x4/QCA937x

To enable on-device logging for QCA61x4/QCA65x4/QCA937, do the following:

1. Generate mask file (Diag_Qxdm.cfg) and execute the following commands:

```
# adb root
# adb remount
# adb push Diag_Qxdm.cfg /sdcard/diag_logs/Diag.cfg
# adb shell setprop persist.service.bdroid.soclog true
# adb shell setprop persist.service.bdroid.snooplog true (if snoop log is
required as part of QXDM)
```

2. Unplug the USB.
3. On **Qualcomm Settings > Enable On-Device logging**.
4. Reproduce the issue. After observing the issue, go to **Enable On-Device logging** setting and disable it.
5. Connect the USB and get the files from the following location:

adb pull /sdcard/diag_logs/

NOTE When on-device logging is selected, the USB port does not show up.

4.2 Enable RAM dump log

The RAM dump log is a snapshot of a memory dump created when the device crashes.

For example, the QPST memory debug application captures the crash log.

Prerequisites

- Enable the hardware watchdog.
- On the MTP/phone, set dip switch 1 to OFF.

The device goes into the emergency mode (DLOAD) when the system crashes.

To capture a RAM dump after the device crashes, do the following:

1. Plug in the USB if not already plugged in.
2. Check if the phone is in Download mode in the QPST configuration.
3. If the phone is in Download mode, start capturing RAM dumps.
4. Start the QPST Memory Debug application.
5. Click **Browse** to choose the QPST port, if not already selected.
6. Click **Get Regions**.
7. Select EBI1 memory region to capture the whole Android and WCNSS memory dump.
8. Click **Save To** and specify a folder to store the RAM dumps.

NOTE In QCA61x4/QCA65x4/QCA937x based firmware, RAM dump can be captured from the following location:

```
#adb pull /data/misc/bluetooth/
```

4.2.1 UART IPC logs

This captures the actual UART data that is put into UART hardware in case of Tx and just after the hardware read in case of Rx.

UART IPC logs are pulled by `#adb pull /data/misc/bluetooth/uart_ipclogs-xx.txt`

4.2.2 UART kernel logging through USB for QCA61x4/QCA65x4/QCA937x

UART logs are required for QCA61x4/QCA65x4/QCA937x-based Bluetooth firmware.

These logs are required for the turn on/off related issues.

Run the following commands on kernel version prior to 3.18:

```
# adb root
# adb remount
# adb shell
$ echo 4 > /sys/module/msm_serial_hs/parameters/debug_mask
$ Ctrl+c
# adb shell cat /sys/kernel/debug/ipc_logging/msm_serial_hs/log_cont > c:\temp
\uart.txt
```

Run the following command on kernel version 3.18:

```
# adb root
# adb remount
# adb shell
```

```
$ echo 4 > /sys/devices/soc/7570000.uart/debug_mask $ Ctrl+c
# adb shell cat /sys/kernel/debug/ipc_logging/7570000.uart/log_cont > c:\temp
\uart.txt
```

NOTE **<devid>**.uart is the device ID which varies based on target.

4.3 Capture OTA log

The Android host logs and Bluetooth firmware logs, for example, SMD, QXDM, or RAM dump logs, can be used to capture the following:

- Bluetooth stack/profile BTSnoop format logs above the HCI layer (hcidump).
- Plain text format LMP layer logs (QXDM).

Use OTA logs to capture the following:

- Profile/stack, above the HCI layer, LMP layer.
- SCO/eSCO packet
- Baseband packet (POLL/NULL).

OTA logs help us to do the following:

- Capture all Bluetooth traffic over a single ACL/SCO link between DUT and the remote device.
- Debug any Bluetooth SoC issues in connection with a remote device.

  An additional air sniffer USB dongle is required, for example, Frontline FTS4BT to capture the OTA log.

4.3.1 Capture OTA log for SSP-enabled remote device

For an SSP-enabled remote device, FTS4BT cannot use the pin code to decrypt the sniffing Bluetooth packet.

The Link Key stored in Android DUT must be used to decrypt the packet.

To capture an OTA log for SSP-enabled remote device, do the following:

1. Pair and connect with the remote device. Disconnect the link after the device is completely connected.
2. Switch DUT's directory in the ADB shell to data/misc/bluetoothd.
3. Use **ls** command to list the local bd address folder, then **cd** to this folder.
4. Execute 'cat linkkeys' to check the linkkeys for each paired device, e.g., get the link key as FE51A92603470D7BA7FBAF802422EA3C.
5. Execute the 'byte-order reversed' for the above link key in reversed order.
6. Execute the 'byte-order reversed' with the following result: 3CEA222480AFFBA77B0D470326A951FE

7. In FTS I/O settings:
   a. Select the appropriate master and slave.
   b. Select the pairing method to Link Key.
   c. Copy and paste the link key with the reversed order to the Link Key text field.
8. Click **OK** and start sniffing.
9. Wait for the master to reconnect with the slave.

   The master is DUT and the slave is the remote headset.
10. Capture and parse the sniffer log to the upper layer without any encryption parsing errors.

4.4 PR3.0 Logging Enhancements

The PR3.0 logging enhancements must follow certain guidelines that are explained in the following section.

4.4.1 Transfer firmware logs to the host via the HCI layer

To transfer firmware logs to the host via the HCI layer, consider the following guidelines:

- Allocate separate buffers in the firmware for logging
- If a buffer is full, firmware sends this buffer as a regular ACL data with a special connection handle to host.
- Host receives this packet as a regular connection data.
- After the host parses the packet for connection handle, the host identifies the logging packet and sends the data to BTSnoop module running on the HOST.
- BTSnoop tool records this data to SD Card with file name ‘hci_snoopxxxxxxxxxx.cfa’.
- If compiled with ‘user_debug’ build option, BTSnoop log is enabled by default.

  Host sends the vendor specific command to enable firmware logging with log level set to High, such that the firmware log available in BTSnoop, by default.

  NOTE FTS plug-in is required to understand this data in FTS viewer, which cannot be shared due to legal reasons.

4.4.2 Logging categories

The following are the different logging categories:

- LLM state transitions, LMP transactions per connection stats.
- Performance stats to measure performance of Inquiry, page etc.
- SLC logging to understand the link starvation and late dispatches due to priority or interrupt locking.
- BB level transactions such as CXM denials, CRC errors, no Rx.

- The data related to BB level transactions is sent to the host on any abnormal event and also periodically.
- Host can configure the following through HCI VS command:
  - Option to receive logging in real-time or only when host is awake
  - Drop the latest log or overwrite the existing log

    This is done as part of enabling this logging.

4.4.3 Interface to control logging at firmware

Different Host Controller Interface (HCI) VSC is available to control the logging at firmware:

Use the following command to configure the firmware enhanced logging over the HCI:

`HCI_VS_HOST_LOG_ENH_CONFIGS_SUBID`

Offset	Field	Size	Value	Description
0	OpCode	2	0xFC17	OGF = 0x3F, OCF = 0x17 HCI_VS_HOST_LOG_OPCODE
2	Parameter Length	1	0x05	Length of remaining data
3	SubOpcode	1	0x10	HCI_VS_HOST_LOG_ENH_CONFIGS_SUBID
4	Flags	2	Variable	Refer the below Logging Param Flags for each flag bit position.
6	Reserved	1	0x00	For Future Use
7	Logging Level	1	Variable	Default Logging level across all modules. 0x00 : Error 0x01 : High 0x02 : Med 0x03 : Low

Logging Param Flags

Field	Bit Pos	Description	Default Value
Enh Log Enable	0	0: Disable Logging 1: Enable Logging	0
Overwrite Prev Logs	1	0: Drop the latest logs when log buffers are not available. 1: De-queue the previous log buffer from Dbg Queue and use it for latest logs.	1
Dynamic Logging Enable	2	0: When HOST is in sleep don't send the logs immediately. Wait until HOST is awake. 1: Send out the Logs as when they are available	0
Reserved	3-15	For Future Use	0

Return Event

Offset	Field	Size	Value	Description
0	Event code	1	0x0E	HCI Command Complete
1	Parameter Length	1	0x05	Length of remaining data
2	Num_HCI_Command_Packets	1	0~255	The Number of HCI command packets which are allowed to be sent to the Controller from the Host.
3	Opcode	2	0xFC17	HCI_VS_HOST_LOG_OPCODE
5	Status	1	0xXX	0x00: succeeded 0x01~0xFF: Error code for HCI command
6	SubOpcode	1	0x10	HCI_VS_HOST_LOG_ENH_CONFIGS_SUBID

Example: VS commands to be sent from HOST

- ENABLE ERROR LEVEL LOGGING CMD: 17 FC 05 10 03 00 00 00
- ENABLE HIGH LEVEL LOGGING CMD : 17 FC 05 10 03 00 00 01
- DISABLE LOGGING CMD : 17 FC 05 10 02 00 00 00

To override the default logging level set using the Enhanced config command for each firmware module, use the following command:

`HCI_VS_HOST_LOG_ENH_LEVEL_OVR_SUBID`

Offset	Field	Size	Value	Description
0	OpCode	2	0xFC17	OGF = 0x3F, OCF = 0x17 HCI_VS_HOST_LOG_OPCODE
2	Parameter Length	1	0x01+N	Length of remaining data
3	SubOpcode	1	0x11	HCI_VS_HOST_LOG_ENH_LVL_OVR_SUBID
4	LogModule X	1	Variable	Log Module Index 0x00 : SLC 0x01 : HIF 0x02 : LLM 0x03 : OTA 0x04 : CxM 0x05 : APP 0x06 : STATS
5	LogLevel X	1	Variable	Log Level 0x00 : Error 0x01 : High 0x02 : Med 0x03 : Low When set other than 0x00 – 0x03, Override is ignored and Log level is set back to default log level which is set as part of ENH_CONFIGS command

Offset	Field	Size	Value	Description
6	LogModule Y	1	Variable	Log Module Index
7	LogLevel Y	1	Variable	Log Level

- Return event

A command complete event will be sent to Host.

Offset	Field	Size	Value	Description
0	Event code	1	0x0E	HCI Command Complete
1	Parameter Length	1	0x05	Length of remaining data
2	Num_HCI_Command_Packets	1	0~255	The Number of HCI command packets which are allowed to be sent to the Controller from the Host.
3	Opcode	2	0xFC17	HCI_VS_HOST_LOG_OPCODE
5	Status	1	0xXX	0x00: succeeded 0x01~0xFF: Error code for HCI command
6	SubOpcode	1	0x11	HCI_VS_HOST_LOG_ENH_LVL_OVR_SUBID

Example: VS commands to be sent from HOST

- Override LLM level to Low Level : 17 FC 03 11 02 03
- Override LLM (Med) and OTA (Low) Level : 17 FC 05 11 02 02 03 03
- Disable LLM Override : 17 FC 03 11 02 0F

The following command is used to FLUSH the current buffered logs available at controller, to be sent to HOST:

`HCI_VS_HOST_LOG_ENH_FLUSH_SUBID`

Offset	Field	Size	Value	Description
0	OpCode	2	0xFC17	OGF = 0x3F, OCF = 0x17 HCI_VS_HOST_LOG_OPCODE
2	Parameter Length	1	0x01	Length of remaining data
3	SubOpcode	1	0x12	HCI_VS_HOST_LOG_ENH_FLUSH_SUBID

- Return Event

A command complete event will be sent to Host.

Offset	Field	Size	Value	Description
0	Event code	1	0x0E	HCI Command Complete
1	Parameter Length	1	0x05	Length of remaining data
2	Num_HCI_Command_Packets	1	0~255	The Number of HCI command packets which are allowed to be sent to the Controller from the Host.
3	Opcode	2	0xFC17	HCI_VS_HOST_LOG_OPCODE

Offset	Field	Size	Value	Description
5	Status	1	0xXX	0x00: succeeded 0x01~0xFF: Error code for HCI command
6	SubOpcode	1	0x12	HCI_VS_HOST_LOG_ENH_FLUSH_SUBID

Example: VS commands to be sent from HOST: FLUSH: 17 FC 01 12

5 Split A2DP debugging

This chapter provides the Split A2DP debugging techniques.

5.1 Disable Split A2DP on Android

1. adb push <audio configuration file*> /etc/audio_policy.conf.
2. adb shell setprop persist.bt.enable.splita2dp false.
3. adb shell setprop persist.bt.a2dp_offload_cap false.
4. adb shell sync.
5. adb reboot.

5.2 Enable Split A2DP

1. adb shell setprop persist.bt.enable.splita2dp true
2. adb shell setprop persist.bt.a2dp_offload_cap <select codec**>
3. Codec choices: a) “sbc-aptx” b)“sbc-aptx-aptxhd” c)“sbc-aac”
4. adb shell sync
5. adb reboot

5.3 Integrate aptX Audio

1. adb push libaptXScheduler.so /system/vendor/lib/libaptXScheduler.so.
2. adb push libaptX-1.0.0-rel-Android21-ARMv7A.so /system/vendor/lib/libaptX-1.0.0-rel-Android21-ARMv7A.so.

5.4 Debugging Split A2DP

Bluetooth host and multimedia audio: Regular logcat log contains information of Bluetooth host stack and audio HAL logs.

Split A2DP debugging of Bluetooth SoC: Btsnoop logs provide details if there are any issues caused due to Bluetooth firmware audio data scheduling. The details include CxM denials, late Tx, and time since the last A2DP packet sent over the air and so on. Audio logging also takes place.

Audio data logging for LPASS: Need to enable audio data logging before and after the encoding through QXDM.

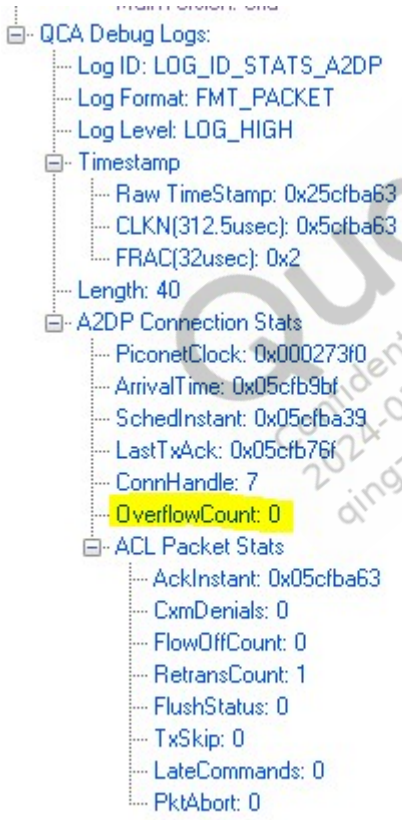
5.5 Split A2DP debugging of Bluetooth SoC

Overflow detection

- When A2DP ring buffer overflows, overflowCount is reported. This enhanced logging feature is enabled for user-debug build.
- OverflowCount is generated whenever every two successful A2DP TXs have a duration longer than the threshold.

Audio dump from snoop

PCM dump on FTS tool is not possible because Split A2DP is an interaction between audio DSP and 3990 and UART bandwidth is not sufficient.



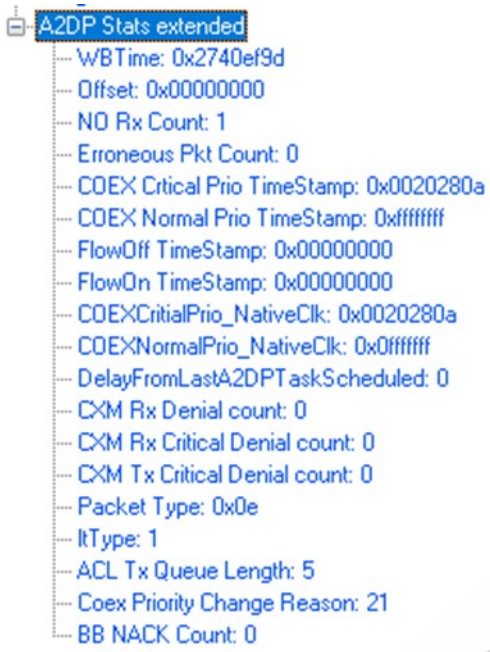
Overflow detection from LPASS

- OverflowCount in LOG_ID_STATS_A2DP counts the overflows from Bluetooth L2CAP buffers.
- For split-A2DP, audio sample is accumulated in a separate buffer. This buffer is filled by audio IOP. LOG_ID_A2DP_SAMPLE_BUF_OVERFLOW is created to monitor this buffer.

- LOG_ID_A2DP_SAMPLE_BUF_OVERFLOW is triggered whenever audio sample buffer overflows.

Bluetooth and WLAN CoEX

- This logging flag is useful to determine if issue is caused by Bluetooth coexistence. It lists the following useful attributes:
  - Erroneous Pkt Count: CRC or Header Error packets count.
  - FlowOff TimeStamp: Timestamp of remote sent a FLOWOFF.
  - FlowOn TimeStamp: Timestamp when remote is sent FLOWON.
  - CXM Rx Denial count: Rx denial counts caused by CxM module. However, this will be disabled by default.
  - ltType: Indicate which logical transport type used for Tx. Basic rate is “0”, and enhanced rate is “1”.



5.6 Delay from LPASS detection

If there is no audio data received from LPASS for at least 30 ms, this enhanced log message is generated by Bluetooth firmware. This is a normal occurrence when audio is paused or stopped.

5.7 A2DP flush event detection

When flush occurs, events are not generated for Split A2DP even though the flush timeout is configured by the host.

5.8 Audio data logging for LPASS

The Audio dump point of LPASS is as follows:

- PCM or compressed data can be dumped on each highlighted point.
- [0x1535] Encoder input – PCM.
- [0x1536] Encoder output – packetized compressed data.

- [0x152E] Decoder input – packetized compressed data.
- [0x1586] AFE logging – PCM.
- [0x14D2] aDSP log – Not audio dump, but required for debugging aDSP

Log messages

Search for underrun or overrun messages in the QXDM to find the underrun and overruns at runtime.

6 Log audio profiles

Run the following command to enable the capturing of logcat logs with the timestamp enabled, before logging audio profiles:

```
adb logcat -v threadtime
```

6.1 Capture and log audio profiles

To log audio profiles, do the following:

1. Capture frame numbers during A2DP playback, using the following command:

   ```
   adb root
   Adb pull /system/etc/bluetooth/bt_stack.conf
   Open    bt_stack.conf and add TRC_LATENCY_AUDIO = 7
       Adb push bt_stack.conf /system/etc/bluetooth/
       Btsnoop_hci.log file consists of the BT snoop logs.
   ```

2. Capture OTA logs by using one of the sniffer tools like Ellisy or FTS4BT.
3. Capture SYSTRACE logs by setting the required option in the following files:

   ```
       BT_AUDIO_SYSTRACE_LOG in audio_a2dp_hw.c
   btif_media_task.c
   ```

4. Capture PCM dump, enable the BT_AUDIO_SAMPLE_LOG option in the following file:

   ```
   audio_a2dp_hw.c
   ```

   This is the input to BT.

   Location of the pcm dump is /data/audio/output_sample.

5. Increase verbosity of hands-free profile by running the following:

   ```
   do adb shell, adb Setprop Handsfree VERBOSE
   ```

6.2 Troubleshoot A2DP source issues

Use the details in this section to troubleshoot the A2DP source issues related to the following:

- A2DP choppiness
- Audio heard on speaker
- A2DP connection failure

6.2.1 A2DP choppiness

The issue of A2DP choppiness is observed during A2DP testing, and could be due to the following:

- Delaying of A2DP packets OTA
- Dropping of packets in stack
- Receiving inputs from PCM by Bluetooth HAL with noise

Packets are being dropped at Bluetooth stack level

In this case, the following error log is displayed:

```
021759 ........ 01-02 00:00:28.009  1986  2060 W bt-btif : btif_media_aa_prep_2_send congestion buf count 24
```

Make the following changes in bt_stack.conf file for enhanced logging:

- TRC_L2CAP=6
- TRC_AVDT_SCB=6
- TRC_AVDT_CCB=6
- TRC_A2D=6
- TRC_BTAPP=6
- TRC_BTIF=6

Logcat logs

```
021745 ........ 01-02 00:00:27.999  1986  2108 D bt-l2cap: L2CA_FlushChannel()  flushed: 0 + 0,  num_left: 5
```

Num_left: 5 denotes that l2cap layer has already queued up five packets of AVDTP and further writing is not possible.

This situation could be due to the delay in num complete event received from SOC.

This condition is validated by snoop logs by arranging A2DP packets according to the delta.

Select the HCI tab to view the time difference between SBC data sent and Num complete packets received.

PCM input to Bluetooth HAL is not appropriate

To verify if the input to Bluetooth HAL is correct, enable PCM dump option in the external/bluetooth/bluedroid/audio_a2dp_hw/audio_a2dp_hw.c file, by doing the following:

```
Enable BT_AUDIO_SAMPLE_LOG
```

PCM dump location is: /data/audio/output_sample.pcm

The *.pcm file can be played over the gold wave tool to verity the audio as shown in the following figure:.

Figure 6-1 shows the noise heard.

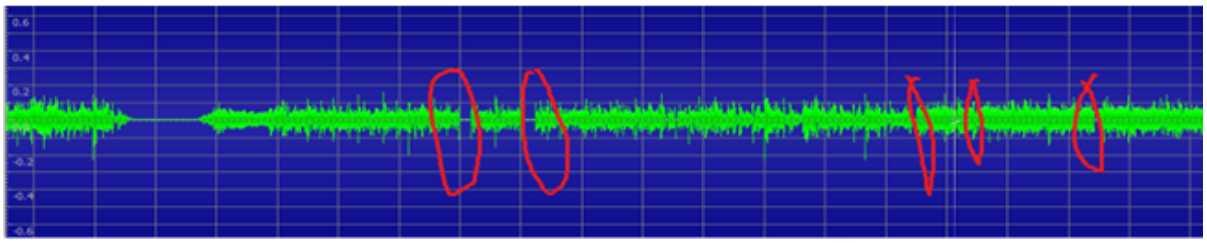


Figure 6-1 Noise heard

Audio choppiness due to a system issue

The sample issue is that a media task does not get scheduled on time.

Systrace can be captured to see if some task is straining the CPU.

Systrace information

The Systrace tool analyzes the performance of the application by capturing and displaying the execution times of the application processes and other Android system processes.

Use the Systrace tool to do the following:

- Combine data from the Android kernel such as the CPU scheduler, disk activity, and application threads.
- Generate a HTML report with the details related to the system process of an Android device for a given period of time.

Prerequisites to run Systrace

- Android SDK Tools 2.0 or later is installed.
- Python is installed and included in the computer execution path.
- Connect device to a running Android 4.1 (API Level 16) or higher using a USB debugging connection.

The Systrace tool can be run using one of the following options:

- Android SDK's graphical user interface tools.
- Command line interface.

To use Systrace on devices running Android 4.3 and higher, specify at least one trace category tag.

For example, to set trace tags and generate a trace from a connected device, execute the following commands:

```
    $ cd android-sdk/platform-tools/systrace
$ python systrace.py --time=10 -o mynewtrace.html sched gfx view wm
```

NOTE To see the names of tasks in the trace output, include the `sched` category in the command parameters.

To achieve the interpolations, where a possible error is captured and marked, enable the BT_AUDIO_SYSTRACE_LOG condition compilation flag, by doing the following:

- Uncomment //#define BT_AUDIO_SYSTRACE_LOG in the following files:
  - audio_a2dp_hw.c
  - btif_media_task.c

6.2.2 Audio heard on speaker

This issue might be the result of a Linux audio problems.

Capture A2DP state logs in logcat.

The following highlighted log shows that A2DP state machine is in a connected state.

```
023764 ..... 01-01 19:03:23.339  2217  2234 D HeadsetStateMachine: Connection state 00:23:78:EA:60:F3: 1->2
```

The following code sample shows the state of A2DP state machine:

```
// match up with btav_connection_state_t enum of bt_av.h
final static int CONNECTION_STATE_DISCONNECTED = 0;
final static int CONNECTION_STATE_CONNECTING = 1;
final static int CONNECTION_STATE_CONNECTED = 2;
final static int CONNECTION_STATE_DISCONNECTING = 3;
```

The following highlighted log shows the Playing state from logcat logs:

```
027926 ..... 01-01 19:05:04.029  2217  2235 D A2dpStateMachine: A2DP Playing state : device: 00:23:78:EA:60:F3 State:11->10
```

The following code sample shows the Playing state of A2DP state machine:

```
 */
public static final int STATE_PLAYING    =  10;

/**
 * A2DP sink device is NOT streaming music. This state can be one of
 * {@link #EXTRA_STATE} or {@link #EXTRA_PREVIOUS_STATE} of
 * {@link #ACTION_PLAYING_STATE_CHANGED} intent.
 */
public static final int STATE_NOT_PLAYING    =  11;
```

Use logcat logs to check if Linux audio is writing to Bluetooth HAL layer as shown in the following highlighted log.

```
027568 ..... 01-01 19:05:04.009  2217  2237 D bt-btif : a2dp-ctrl-cmd : A2DP_CTRL_CMD_START
```

Based on the preceding highlighted log, Bluetooth HAL receives A2DP_CTRL_CMD_START.

The important log to check is:

```
str_params: key: 'routing' value: '128'
AUDIO_DEVICE_OUT_BLUETOOTH_A2DP            = 0x80,
AUDIO_DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES = 0x100,
AUDIO_DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER    = 0x200,
AUDIO_DEVICE_OUT_AUX_DIGITAL               = 0x400,
AUDIO_DEVICE_OUT_ANLG_DOCK_HEADSET         = 0x800,
AUDIO_DEVICE_OUT_DGTL_DOCK_HEADSET         = 0x1000,
AUDIO_DEVICE_OUT_USB_ACCESSORY             = 0x2000,
```

```
AUDIO_DEVICE_OUT_USB_DEVICE                = 0x4000,
AUDIO_DEVICE_OUT_REMOTE_SUBMIX             = 0x8000,
```

6.2.3 A2DP connection failure

If the A2DP connection gets disconnected, the reasons are captured in logcat logs:

To debug the issue related to A2DP connection failure, create snoop and logcat logs, as shown in the highlighted log.

```
 9038 01-01 08:45:19.388 D/A2dpStateMachine( 2201): Connection state 94:CE:2C:FD:A6:9F: 1->0
```

Increase verbosity by making the following changes to the bt_stack.conf file:

```
TRC_L2CAP= 6
TRC_AVDT_SCB= 6
TRC_AVDT_CCB=6
TRC_AVDT=6
TRC_A2D=6
TRC_BTAPP=6
TRC_BTIF=6
```

6.3 Troubleshoot AVRCP problems using logs

The AVRCP application logs capture metadata information, by default.

The issues pertaining to metadata can be debugged by using the logcat logs.

Increase verbosity by making the following changes to the bt_stack.conf file:

```
TRC_AVRC=5
TRC_BTAPP=5
TRC_BTIF=5
TRC_AVCT=5
```

Capture the logcat logs as shown in the following figure.

```
108119 ........ 05-21 15:08:02.918  2250  2347 I BluetoothAvrcpServiceJni: btavrcp_get_element_attr_callback
108121 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : MESSAGE_GET_ELEM_ATTRS:numAttr=7
108123 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=1 str=Desi Girl
108125 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=2 str=Shankar Mahadevan, Sunidhi Chauhan
108127 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=3 str=Dostana @ Fmw11.com
108129 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=4 str=9
108131 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=5 str=30
108133 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=6 str=
108135 ........ 05-21 15:08:02.918  2250  2366 V Avrcp   : getAttributeString:attrId=7 str=307096
111809 ........ 05-21 15:08:03.878  2250  2366 V Avrcp   : MESSAGE_PLAY_INTERVAL_TIMEOUT
111811 ........ 05-21 15:08:03.878  2250  2366 V Avrcp   : position=116338
```

Play status logs are available in the logcat logs as shown in the following highlighted log.

```
042596 ........ 05-21 15:06:51.398  2250  2366 V Avrcp   : old state = 2, new state= 3
```

6.3.1 Play and pause commands not working

Logcat logs provide information if a valid file descriptor exists.

The AVRCP connection closure can be confirmed from the logcat logs and snoop logs.

Capture logcat logs are available, as shown in the following highlighted log.

```
11319 ........ 03-04 17:23:03.657  2016  2078 D bt-btif : send_key fd:-1 key:200 pressed:0
```

Increase verbosity at bt_stack.conf to capture various AVRCP events received, as shown in the following highlighted log.

```
10421 ....... 03-04 17:22:46.787  2016  2113 D bt-btif : AV nsm event=0x1233(AVRC_CLOSE)
```

Remote device does not send any AVRCP commands on key press

This issue occurs during Sniff mode.

To debug, capture and use the required OTA logs.

If the OTA logs captured using Ellysis show that the remote device does not send AVRCP, pause commands in Sniff mode.

For issues where DUT does not respond to AVRCP commands, check the OTA logs to see if the remote device has actually sent the AVRCP commands.

6.3.2 AVRCP metadata not available

Remote device does not understand AVRCP 1.5

The remote device does not understand AVRCP 1.5 version, which can be seen in snoop logs, and does not send any AVRCP 1.3 version registration commands.

If snoop logs are not available from logcat, the following highlighted log is not visible.

```
05511 ........ 03-04 17:22:34.847  2016  2113 D bt-avrp  : avrc_msg_cback handle:1, ctype:3, offset:11, len: 15
```

Ctype: 3 is for notification

Ctype: 1 is for status

If the remote device is unable to parse the AVRCP 1.5, it is not possible to send an AVRCP command for notification.

6.3.3 Shuffle/repeat not working when toggled from Carkit

Attempt toggling shuffle/repeat from carkit side.

PlayerApplicationSetting request is sent from Avrcp to music player for setting the new shuffle/repeat modes

Response is received back as Response received for SET_ATTRIBUTE_VALUES in the following code.

```
046195 01-01 01:04:34.590  1898  2035 I bt_btif : handle_rc_metamsg_cmd:
Passing received metamsg command to app. pdu: AVRC_PDU_SET_PLAYER_APP_VALUE
046196 01-01 01:04:34.590  1898  2035 D bt_btif : btif_rc_upstreams_evt pdu:
AVRC_PDU_SET_PLAYER_APP_VALUE handle: 0x0 ctype:0 label:0
046197 01-01 01:04:34.590  1898  2035 D bt_btif : btif_rc_upstreams_evt txn
label 0 enqueued to txn queue of pdu AVRC_PDU_SET_PLAYER_APP_VALUE, queue
size 1
046199 01-01 01:04:34.590  1898  2035 I BluetoothAvrcpServiceJni:
btavrcp_set_playerapp_setting_value_callback
046200 01-01 01:04:34.590  1898  2035 V Avrcp   : setPlayerAppSetting: number
of attributes1
```

```
046228 01-01 01:04:34.611  1898  2035 I Avrcp   : device found at index 0
046249 01-01 01:04:34.627  1898  1898 V Avrcp   : Response received for
SET_ATTRIBUTE_VALUES
046250 01-01 01:04:34.627  1898  1898 V Avrcp   : getResponse6
046251 01-01 01:04:34.628  1898  1898 V Avrcp   : updateLocalPlayerSettings
046252 01-01 01:04:34.628  1898  1898 V Avrcp   : ID: 1 Value: -1
046253 01-01 01:04:34.628  1898  1898 V Avrcp   : ID: 3 Value: 2
046254 01-01 01:04:34.628  1898  1898 V Avrcp   : ID: 2 Value: 1
046255 01-01 01:04:34.628  1898  1898 V Avrcp   : Respond to
SET_ATTRIBUTE_VALUES request
046256 01-01 01:04:34.628  1898  1898 V Avrcp   : checkPlayerAttributeResponse
046257 01-01 01:04:34.628  1898  1898 V Avrcp   : ID: 1 Value: -1
046258 01-01 01:04:34.628  1898  1898 V Avrcp   : ID: 3 Value: 2
046259 01-01 01:04:34.628  1898  1898 V Avrcp   : Pending SetAttribute
contains Shuffle
046260 01-01 01:04:34.628  1898  1898 V Avrcp   : ID: 2 Value: 1
046261 01-01 01:04:34.628  1898  1898 D bt_btif : -set_player_app_value_rsp
on index = 0
046262 01-01 01:04:34.628  1898  1898 D bt_btif : ## set_player_app_value_rsp
##
046263 01-01 01:04:34.628  1898  1898 I bt_btif : +send_metamsg_rsp:
rc_handle: 0, label: 0, code: 0x00, pdu: AVRC_PDU_SET_PLAYER_APP_VALUE
```

6.3.4 DUT does not respond to SetBrowsedPlayer

Remote sends setBrowsedPlayer after browsing l2cap channel connection and DUT does not respond

Request received in Bluedroid stack

```
091012 07-12 16:11:28.479  1869  2020 D bt_btif : btif_rc_upstreams_evt pdu:
AVRC_PDU_SET_BROWSED_PLAYER handle: 0x0 ctype:0 label:0
091013 07-12 16:11:28.479  1869  2020 I bt_btif : btif_rc_upstreams_evt()
AVRC_PDU_SET_BROWSED_PLAYER
091014 07-12 16:11:28.479  1869  2020 D bt_btif : btif_rc_upstreams_evt txn
label 0 enqueued to txn queue of pdu AVRC_PDU_SET_BROWSED_PLAYER, queue size 1
091015 07-12 16:11:28.479  1869  2020 I bt_btif : HAL bt_rc_callbacks-
>set_browsed_player_cb
```

Request received in Bluetooth apps

```
091016 07-12 16:11:28.479  1869  2020 I BluetoothAvrcpServiceJni:
btavrcp_set_browsed_player_callback
091017 07-12 16:11:28.479  1869  2020 I BluetoothAvrcpServiceJni: player id: 1
091018 07-12 16:11:28.479  1869  2020 V Avrcp   : setBrowsedPlayer: PlayerID:
1
091019 07-12 16:11:28.480  1869  2431 I Avrcp   : device found at index 0
091020 07-12 16:11:28.480  1869  2431 V Avrcp   : processSetBrowsedPlayer:
PlayerID: 1
091021 07-12 16:11:28.480  1869  2431 V Avrcp   : player found and available
```

```
091022 07-12 16:11:28.480  1869  2431 V Avrcp   : Player ID: 1is Browsable!
091023 07-12 16:11:28.480  1869  2431 V Avrcp   : Player ID: 1is Browsable
only when addressed!
091024 07-12 16:11:28.480  1869  2431 V Avrcp   : player addressed hence
browsable
```

Request sent to frameworks/Media player

```
091025 07-12 16:11:28.480  1869  2431 D TransportController:
setRemoteControlClientBrowsedPlayer in TransportControls
091027 07-12 16:11:28.481  1449  2465 D MediaSessionRecord:
setRemoteControlClientBrowsedPlayer in ControllerStub
091032 07-12 16:11:28.481  1449  2465 D MediaSessionRecord:
setRemoteControlClientBrowsedPlayer in SessionCb
091033 07-12 16:11:28.482  5044  5056 D MediaSession:
setRemoteControlClientBrowsedPlayer in CallbackStub
091049 07-12 16:11:28.483  5044  5044 D MediaSession: MSG_SET_BROWSED_PLAYER
received in CallbackMessageHandler
091054 07-12 16:11:28.483  5044  5044 D RemoteControlClient: setBrowsedPlayer
in RemoteControlClient
091067 07-12 16:11:28.491  5044  5044 D RemoteControlClient:
MSG_SET_BROWSED_PLAYER in RemoteControlClient
091070 07-12 16:11:28.491  5044  5044 D RemoteControlClient:
onSetBrowsedPlayer
091071 07-12 16:11:28.491  5044  5044 D RemoteControlClient:
mSetBrowsedPlayerListener.onSetBrowsedPlayer
091119 07-12 16:11:28.507  5044  5044 E RemoteControlClient:
updateFolderInfoBrowsedPlayer
091120 07-12 16:11:28.507  5044  5044 D RemoteControlClient:
updateFolderInfoBrowsedPlayerInt
091121 07-12 16:11:28.507  5044  5044 D MediaSession: MediaSession:
updateFolderInfoBrowsedPlayer
091122 07-12 16:11:28.508  1449  1916 D MediaSessionRecord: SessionStub:
updateFolderInfoBrowsedPlayer
091123 07-12 16:11:28.509  1449  1449 D MediaSessionRecord:
pushBrowsePlayerInfo
```

Response received in Bluetooth apps

```
091129 07-12 16:11:28.510  1869  2431 V Avrcp   :
onClientFolderInfoBrowsedPlayer: stringUri: content://media/external/audio/
media
091130 07-12 16:11:28.510  1869  2431 V Avrcp   : URI received: content://
media/external/audio/media
091138 07-12 16:11:28.511  1869  2431 V Avrcp   :
MSG_UPDATE_BROWSED_PLAYER_FOLDER
091139 07-12 16:11:28.511  1869  2431 V Avrcp   : updateBrowsedPlayerFolder:
numOfItems =  4 status = 4
```

Response received in Bluedroid stack

```
091147 07-12 16:11:28.511  1869  2431 D bt_btif : ## set_browseplayer_rsp ##
091148 07-12 16:11:28.511  1869  2431 D bt_btif : - set_browseplayer_rsp on
index = 0
```

See the details related to the request or response failure and debug accordingly.

6.4 Hands-free profile (HFP)

Run logcat with HeadsetStateMachine logs enabled, along with default stack logging to generate the HFP log file. Use this log file to troubleshoot HFP related issues.

Issues related HFP include the following:

- Audio routing
- Three-way calls
- Reconnections
- WBS and codec negotiation
- A2DP + call concurrency
- In-call LPM
- Virtual calls and voice recognition

6.4.1 Enable SOC audio logs for the WCN solution

To enable SOC audio logs for WCN, do the following:

1. From the QXDM menu select **Options > Log View Configuration >** > **Log Packets** tab.
2. Expand **Known Log Items > Common >** > **Wireless Connectivity Subsystem**.
3. Click [0x1558] Wireless Connectivity Subsystem Audio Data.
4. Click **OK**.
5. Select **Options > Message View Configuration >** > **Log Packets** tab.
6. Expand **Known Log Items > Common >** > **Wireless Connectivity Subsystem**.
7. Click **[0x1558] Wireless Connectivity Subsystem Audio Data**.
8. Click **OK**.

NOTE WCNSS Audio data logs cannot be captured for QCA61x4/QCA65x4/QCA937x-based Firmware.

6.4.2 Enable AFE audio logs

To enable AFE audio logs outside Bluetooth Firmware, do the following:

1. From the **QXDM menu select Options > Log View Configuration** > **> Log Packets** tab.
2. Expand **Known Log Items > Common**.
3. Click **[0x13B0] Audio Vocoder Data Paths**.

4. Click **OK**.
5. Select **Options > Message View Configuration >** > **Log Packets** tab.
6. Expand **Known Log Items > Common**.
7. Click **[0x13B0] Audio Vocoder Data Paths**.
8. Click **OK**.
9. Click **OK**.
10. Save the logs by selecting (from the QXDM menu) **File > Save Items**.
11. Save the .isf file to a known location.

NOTE Audio Vocoder Data Path logs can be captured even for QCA61x4/QCA65x4/QCA937x-based Firmware.

6.4.3 Create and analyze HFP logs

The HFP profile runs in state machine which can show the current state of HFP.

To create HFP logs, do the following:

- Set the following values:
  - DISCONNECTED: 0
  - PENDING: 1
  - CONNECTED: 2
  - DISCONNECTING: 3

These state movements can be observed in the UI as going from ‘Connecting’ to ‘Connected’ and ‘Disconnecting’ to ‘Disconnected’.

For example, see the following highlighted log:

```
)1595 ........ D/HeadsetStateMachine( 1438): make
)1603 ........ D/HeadsetStateMachine( 1438): BRSF_AG_CODEC_NEGOTIATION is enabled!
)1609 ........ D/HeadsetStateMachine( 1438): Enter Disconnected: -2
)1635 ........ D/HeadsetStateMachine( 1438): Proxy object connected
)1645 ........ D/HeadsetStateMachine( 1438): Disconnected process message: 10
)1649 ........ D/HeadsetStateMachine( 1438): Disconnected process message: 11
)2135 ........ D/HeadsetStateMachine( 1438): Disconnected process message: 1
)2137 ........ D/HeadsetStateMachine( 1438): Connection state 00:24:1C:CB:44:07: 0->1
)2149 ........ D/HeadsetStateMachine( 1438): Exit Disconnected: 1
)2153 ........ D/HeadsetStateMachine( 1438): Enter Pending: 1
)2227 ........ D/HeadsetStateMachine( 1438): Pending process message: 101
)2229 ........ D/HeadsetStateMachine( 1438): event type: 1
)2231 ........ D/HeadsetStateMachine( 1438): Connection state 00:24:1C:CB:44:07: 1->2
```

DUT does not send the latest call states

This log suggests that when an incoming connection happens, the state machine is moved to Connected, and it did not query the BluetoothPhoneService to get the latest call states.

As a result of this, the device failed to update the CIND indication.

Once the SLC is established, the DUT queries and tries to open the SCO connection.

For example, see the following logs:

```
)85313 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: Disconnected process message: 101, size: 0
)85315 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: event type: 1
)85317 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: processConnectionEvent state = 2, device = 00:0D:3C:B0:2D:6F
)85319 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: HFP Connected from Disconnected state
)85325 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: Incoming Hf accepted
)85327 ........ 05-29 09:33:21.454  1190  1190 D BluetoothPhoneService: handleMessage: 4
)85329 ........ 05-29 09:33:21.454  1190  1190 V BluetoothPhoneService: Call state Converted2: WAITING/ACTIVE -> 4
)85395 ........ 05-29 09:33:21.464  1670  1753 D HeadsetStateMachine: Connection state 00:0D:3C:B0:2D:6F: 0->2
)85441 ........ 05-29 09:33:21.464  1670  1753 D HeadsetStateMachine: device 00:0D:3C:B0:2D:6F is adding in Disconnected state
)85509 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: configAudioParameters for device:00:0D:3C:B0:2D:6F are: codec =0 nre
)85511 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: Exit Disconnected: 101
)85513 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: Enter Connected: 101, size: 1
)85515 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: Connected process message: 9, size: 1
)85517 ........ 05-29 09:33:21.474  1670  1753 E HeadsetStateMachine: terminateScoUsingVirtualVoiceCall:No present call to terminate
)87965 ........ 05-29 09:33:23.064  1670  1753 D HeadsetStateMachine: Connected process message: 101, size: 1
)87967 ........ 05-29 09:33:23.064  1670  1753 D HeadsetStateMachine: event type: 12event device : 00:0D:3C:B0:2D:6F
)88573 ........ 05-29 09:33:23.564  1670  1753 D HeadsetStateMachine: Connected process message: 101, size: 1
)88575 ........ 05-29 09:33:23.564  1670  1753 D HeadsetStateMachine: event type: 1event device : 00:0D:3C:B0:2D:6F
)88577 ........ 05-29 09:33:23.564  1670  1753 D HeadsetStateMachine: processConnectionEvent state = 3, device = 00:0D:3C:B0:2D:6F
)88583 ........ 05-29 09:33:23.574  1670  1753 D HeadsetStateMachine: Remote Brsf: 25 for device: 00:0D:3C:B0:2D:6F
)88587 ........ 05-29 09:33:23.574  1670  1753 D HeadsetStateMachine: Connected process message: 11, size: 1
)88729 ........ 05-29 09:33:23.674  1670  1753 D HeadsetStateMachine: Connected process message: 18, size: 1
)88731 ........ 05-29 09:33:23.674  1190  1190 D BluetoothPhoneService: handleMessage: 4
)88751 ........ 05-29 09:33:23.684  1190  1190 V BluetoothPhoneService: Call state Converted2: WAITING/ACTIVE -> 4
)88773 ........ 05-29 09:33:23.704  1670  1753 D HeadsetStateMachine: Connected process message: 9, size: 1
)88775 ........ 05-29 09:33:23.704  1670  1753 E HeadsetStateMachine: terminateScoUsingVirtualVoiceCall:No present call to terminate
```


```
)85313 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: Disconnected process message: 101, size: 0
)85315 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: event type: 1
)85317 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: processConnectionEvent state = 2, device = 00:0D:3C:B0:2D:6F
)85319 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: HFP Connected from Disconnected state
)85325 ........ 05-29 09:33:21.444  1670  1753 D HeadsetStateMachine: Incoming Hf accepted
)85327 ........ 05-29 09:33:21.454  1190  1190 D BluetoothPhoneService: handleMessage: 4
)85329 ........ 05-29 09:33:21.454  1190  1190 V BluetoothPhoneService: Call state Converted2: WAITING/ACTIVE -> 4
)85395 ........ 05-29 09:33:21.464  1670  1753 D HeadsetStateMachine: Connection state 00:0D:3C:B0:2D:6F: 0->2
)85441 ........ 05-29 09:33:21.464  1670  1753 D HeadsetStateMachine: device 00:0D:3C:B0:2D:6F is adding in Disconnected state
)85509 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: configAudioParameters for device:00:0D:3C:B0:2D:6F are: codec =0 nre
)85511 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: Exit Disconnected: 101
)85513 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: Enter Connected: 101, size: 1
)85515 ........ 05-29 09:33:21.474  1670  1753 D HeadsetStateMachine: Connected process message: 9, size: 1
)85517 ........ 05-29 09:33:21.474  1670  1753 E HeadsetStateMachine: terminateScoUsingVirtualVoiceCall:No present call to terminate
)87965 ........ 05-29 09:33:23.064  1670  1753 D HeadsetStateMachine: Connected process message: 101, size: 1
)87967 ........ 05-29 09:33:23.064  1670  1753 D HeadsetStateMachine: event type: 12event device : 00:0D:3C:B0:2D:6F
)88573 ........ 05-29 09:33:23.564  1670  1753 D HeadsetStateMachine: Connected process message: 101, size: 1
)88575 ........ 05-29 09:33:23.564  1670  1753 D HeadsetStateMachine: event type: 1event device : 00:0D:3C:B0:2D:6F
)88577 ........ 05-29 09:33:23.564  1670  1753 D HeadsetStateMachine: processConnectionEvent state = 3, device = 00:0D:3C:B0:2D:6F
)88583 ........ 05-29 09:33:23.574  1670  1753 D HeadsetStateMachine: Remote Brsf: 25 for device: 00:0D:3C:B0:2D:6F
)88587 ........ 05-29 09:33:23.574  1670  1753 D HeadsetStateMachine: Connected process message: 11, size: 1
)88729 ........ 05-29 09:33:23.674  1670  1753 D HeadsetStateMachine: Connected process message: 18, size: 1
)88731 ........ 05-29 09:33:23.674  1190  1190 D BluetoothPhoneService: handleMessage: 4
)88751 ........ 05-29 09:33:23.684  1190  1190 V BluetoothPhoneService: Call state Converted2: WAITING/ACTIVE -> 4
)88773 ........ 05-29 09:33:23.704  1670  1753 D HeadsetStateMachine: Connected process message: 9, size: 1
)88775 ........ 05-29 09:33:23.704  1670  1753 E HeadsetStateMachine: terminateScoUsingVirtualVoiceCall:No present call to terminate
```


In the following log, even though SCO is opened, there was no audio on the headset, as it did not have complete information about how the call became active.

```
088779 ........ 05-29 09:33:23.704  1670  1753 I bt-btif : BTHF: btif_hf_check_if_slc_connected: slc connected for idx = 0
088781 ........ 05-29 09:33:23.704  1670  1753 D bt-btif : phone_state_change: num_active=1 [prev: 0]  num_held=0[prev: 0] call_setup=BTHF_CALL_STATE_INCOMING [prev: BTH
088785 ........ 05-29 09:33:23.704  1670  1753 D bt-btif : phone_state_change: Call setup states changed. old: BTHF_CALL_STATE_IDLE new: BTHF_CALL_STATE_INCOMING
089347 ........ 05-29 09:33:24.054  1670  1753 D HeadsetStateMachine: Connected process message: 101, size: 1
089349 ........ 05-29 09:33:24.054  1670  1753 D HeadsetStateMachine: event type: 6event device : 00:0D:3C:B0:2D:6F
089377 ........ 05-29 09:33:24.064  1670  1753 D HeadsetStateMachine: Connected process message: 7, size: 1
089575 ........ 05-29 09:33:24.224  1670  1753 D HeadsetStateMachine: Connected process message: 101, size: 1
089577 ........ 05-29 09:33:24.224  1670  1753 D HeadsetStateMachine: event type: 2event device : 00:0D:3C:B0:2D:6F
089579 ........ 05-29 09:33:24.224  1670  1753 D HeadsetStateMachine: Set sample rate: 8000 for device:00:0D:3C:B0:2D:6F
089649 ........ 05-29 09:33:24.234  1670  1753 D HeadsetStateMachine: Audio state 00:0D:3C:B0:2D:6F: 11->12
089651 ........ 05-29 09:33:24.234  1670  1753 D HeadsetStateMachine: Enter AudioOn: 101, size: 1
```

The following logs are for call states updated from telephony to Bluetooth HFP.

```
041131 ........ 05-29 09:32:43.304  1190  1190 D BluetoothPhoneService:  handlePreciseCallStateChange: foreground: DIALING background: IDLE ringing: IDLE
041133 ........ 05-29 09:32:43.304  1190  1190 E BluetoothPhoneService: Could not get a handle on Connection object for the call
041137 ........ 05-29 09:32:43.314  1190  1190 D BluetoothPhoneService: update the call states, active: 0held0
041139 ........ 05-29 09:32:43.314  1190  1190 D BluetoothPhoneService: update the headset
041141 ........ 05-29 09:32:43.314  1190  1190 V BluetoothPhoneService: Call state Converted2: IDLE/DIALING -> 2
041561 ........ 05-29 09:32:43.434  1190  1190 D BluetoothPhoneService: handleMessage: 1
041563 ........ 05-29 09:32:43.434  1190  1190 D BluetoothPhoneService:  handlePreciseCallStateChange: foreground: DIALING background: IDLE ringing: IDLE
041565 ........ 05-29 09:32:43.434  1190  1190 E BluetoothPhoneService: Could not get a handle on Connection object for the call
041567 ........ 05-29 09:32:43.434  1190  1190 D BluetoothPhoneService: update the call states, active: 0held0
041881 ........ 05-29 09:32:43.574  1190  1190 D BluetoothPhoneService: handleMessage: 1
041883 ........ 05-29 09:32:43.574  1190  1190 D BluetoothPhoneService:  handlePreciseCallStateChange: foreground: DIALING background: IDLE ringing: IDLE
041885 ........ 05-29 09:32:43.574  1190  1190 E BluetoothPhoneService: Could not get a handle on Connection object for the call
041887 ........ 05-29 09:32:43.574  1190  1190 D BluetoothPhoneService: update the call states, active: 0held0
045037 ........ 05-29 09:32:45.764  1190  1190 D BluetoothPhoneService: handleMessage: 1
045039 ........ 05-29 09:32:45.764  1190  1190 D BluetoothPhoneService:  handlePreciseCallStateChange: foreground: ALERTING background: IDLE ringing: IDLE
045041 ........ 05-29 09:32:45.764  1190  1190 E BluetoothPhoneService: Could not get a handle on Connection object for the call
045043 ........ 05-29 09:32:45.764  1190  1190 D BluetoothPhoneService: update the call states, active: 0held0
045045 ........ 05-29 09:32:45.764  1190  1190 D BluetoothPhoneService: update the headset
045047 ........ 05-29 09:32:45.764  1190  1190 V BluetoothPhoneService: Call state Converted2: IDLE/ALERTING -> 3
045705 ........ 05-29 09:32:46.154  1190  1190 D BluetoothPhoneService: handleMessage: 1
045709 ........ 05-29 09:32:46.154  1190  1190 D BluetoothPhoneService:  handlePreciseCallStateChange: foreground: ACTIVE background: IDLE ringing: IDLE
```

6.4.4 WBS and codec negotiation

Issues related to WBS and codec negotiation can include the following:

- ‘Remote unresponsive’ for codec
- Failure of the SCO connection
- No audio over the WBS SCO connection.

Snoop or OTA logs show the behavior for codec negotiation and WBS.

To view the Snoop or OTA logs, do the following:

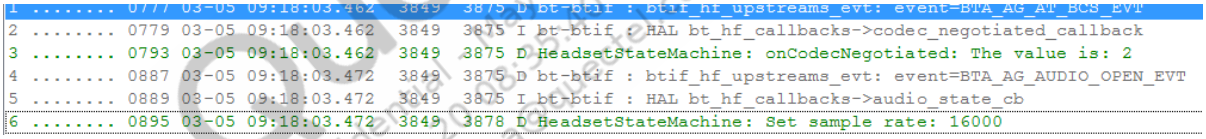
- Create Snoop or OTA logs.

  At SLC, the following exchange must be present:

L2CAP:	106	S (HF)	1	AT+BAC=1,2.	23	00:00:00.011312	3/5/2014 9:17:18.053489 AM
	107	M (AG)	1	..OK..	19	00:00:00.000470	3/5/2014 9:17:18.053959 AM

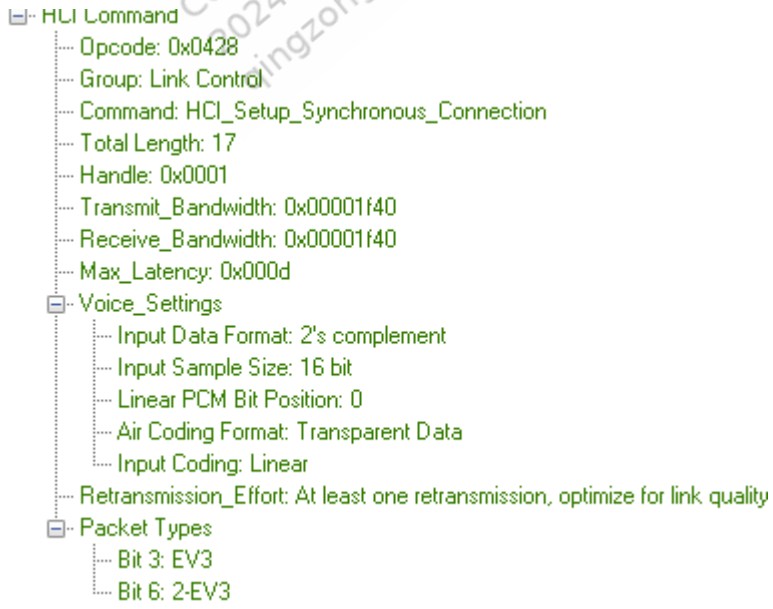
Before opening SCO, DUT should always negotiate the codec once in SLC life cycle until remote rejects or starts renegotiation.

Addr	301	M (AG)	1	..+CIEV: 2,2..	26	00:00:44.868389	3/5/2014 9:18:03.366481 AM
PDU	303	M (AG)	1	..+BCS: 2..	23	00:00:00.000506	3/5/2014 9:18:03.366987 AM
Char	304	M (AG)	1	..+CIEV: 2,3..	26	00:00:00.001319	3/5/2014 9:18:03.368306 AM
RFCOMM	305	M (AG)	1	..+CIEV: 1,1..	26	00:00:00.001646	3/5/2014 9:18:03.369952 AM
Role	306	M (AG)	1	..+CIEV: 2,0..	26	00:00:00.000802	3/5/2014 9:18:03.370754 AM
Addr	315	S (HF)	1	AT+BCS=2.	21	00:00:00.095428	3/5/2014 9:18:03.466182 AM
Addr	320	M (AG)	1	..OK..	19	00:00:00.004323	3/5/2014 9:18:03.470505 AM



```
1 ........ 0777 03-05 09:18:03.462  3849  3875 D bt-btif : btif_hf_upstreams_evt: event=BTA_AG_AT_BCS_EVT
2 ........ 0779 03-05 09:18:03.462  3849  3875 I bt-btif : HAL bt_hf_callbacks->codec_negotiated_callback
3 ........ 0793 03-05 09:18:03.462  3849  3875 D HeadsetStateMachine: onCodecNegotiated: The value is: 2
4 ........ 0887 03-05 09:18:03.472  3849  3875 D bt-btif : btif_hf_upstreams_evt: event=BTA_AG_AUDIO_OPEN_EVT
5 ........ 0889 03-05 09:18:03.472  3849  3875 I bt-btif : HAL bt_hf_callbacks->audio_state_cb
6 ........ 0895 03-05 09:18:03.472  3849  3878 D HeadsetStateMachine: Set sample rate: 16000
```

- SCO connection must always be as follows:



```
HCI Command
    Opcode: 0x0428
    Group: Link Control
    Command: HCI_Setup_Synchronous_Connection
    Total Length: 17
    Handle: 0x0001
    Transmit_Bandwidth: 0x00001f40
    Receive_Bandwidth: 0x00001f40
    Max_Latency: 0x000d
    Voice_Settings
        Input Data Format: 2's complement
        Input Sample Size: 16 bit
        Linear PCM Bit Position: 0
        Air Coding Format: Transparent Data
        Input Coding: Linear
    Retransmission_Effort: At least one retransmission, optimize for link quality
    Packet Types
        Bit 3: EV3
        Bit 6: 2-EV3
```

- For a non-WBS headset, the following log should not be present:

```
set sample rate: 16000
```

- For an SCO audio connection, the Bluetooth stack maintains the SCO state machine.

  This machine drives the SCO and codec state movements.

SCO states

```
/* sco states */
enum
{
    BTA_AG_SCO_SHUTDOWN_ST,     /* no sco listening, all sco connections
closed */
    BTA_AG_SCO_LISTEN_ST,       /* sco listening */
#if (BTM_WBS_INCLUDED == TRUE )
    BTA_AG_SCO_CODEC_ST,        /* sco codec negotiation */
#endif
    BTA_AG_SCO_OPENING_ST,      /* sco connection opening */
    BTA_AG_SCO_OPEN_CL_ST,      /* opening sco connection being closed */
    BTA_AG_SCO_OPEN_XFER_ST,    /* opening sco connection being transferred */
    BTA_AG_SCO_OPEN_ST,         /* sco open */
    BTA_AG_SCO_CLOSING_ST,      /* sco closing */
    BTA_AG_SCO_CLOSE_OP_ST,     /* closing sco being opened */
    BTA_AG_SCO_CLOSE_XFER_ST,   /* closing sco being transferred */
    BTA_AG_SCO_SHUTTING_ST      /* sco shutting down */
};
```

SCO events

```
/* sco events */
enum
{
    BTA_AG_SCO_LISTEN_E,        /* listen request */
    BTA_AG_SCO_OPEN_E,          /* open request */
    BTA_AG_SCO_XFER_E,          /* transfer request */
#if (BTM_WBS_INCLUDED == TRUE )
    BTA_AG_SCO_CN_DONE_E,       /* codec negotiation done */
    BTA_AG_SCO_REOPEN_E,        /* Retry with other codec when failed */
#endif
    BTA_AG_SCO_CLOSE_E,         /* close request */
    BTA_AG_SCO_SHUTDOWN_E,      /* shutdown request */
    BTA_AG_SCO_CONN_OPEN_E,     /* sco open */
    BTA_AG_SCO_CONN_CLOSE_E,    /* sco closed */
    BTA_AG_SCO_CI_DATA_E        /* SCO data ready */
};
```

```
2487 ........ 03-05 09:18:14.182  3849  3921 I bt-btif : BTA ag sco evt (hdl 0xffff): State 7, Event 8
```

SCO event log checks for the current SCO state and the event that it is going to process, based on the logcat logs.

Mapping to this log, any SCO state machine issue can be identified.

SCO states must be appropriate to handle new SCO connections or proper disconnections.

Inappropriate SCO state movements are observed during stress tests on audio connections.

This log information helps to analyze them quickly.

6.4.5 A2DP + Call concurrency

If an A2DP playback and a call state change happens together, A2DP is always suspended in Bluetooth until the call process is complete.

This ensures that the remote headset state is correct and ready to handle the audio codecs (A2DP and SCO).

This log can also be used to resolve IOT issues.

Remote device moves to Bad state if the audio is not suspended in time.

- The following highlighted log must be observed in the call scenarios.

```
041131 ........ 05-29 09:32:43.304  1190  1190 D BluetoothPhoneService:  handlePreciseCallStateChange: foreground: DIALING background: IDLE ringing: IDLE
041133 ........ 05-29 09:32:43.304  1190  1190 E BluetoothPhoneService: Could not get a handle on Connection object for the call
041137 ........ 05-29 09:32:43.314  1190  1190 D BluetoothPhoneService: update the call states, active: 0held0
041139 ........ 05-29 09:32:43.314  1190  1190 D BluetoothPhoneService: update the headset
041141 ........ 05-29 09:32:43.314  1190  1190 V BluetoothPhoneService: Call state Converted2: IDLE/DIALING -> 2
041145 ........ 05-29 09:32:43.314  1670  1753 D HeadsetStateMachine: Connected process message: 9, size: 1
041149 ........ 05-29 09:32:43.314  1670  1753 E HeadsetStateMachine: terminateScoUsingVirtualVoiceCall:No present call to terminate
041153 ........ 05-29 09:32:43.324   235   235 V SRS_Proc: ParamSet string: A2dpSuspended=true
041155 ........ 05-29 09:32:43.324   235   235 D audio_hw_primary: adev_set_parameters: enter: A2dpSuspended=true
041161 ........ 05-29 09:32:43.324   235   235 V listen_hw: handle_set_parameters: Enter kvpairs=A2dpSuspended=true
041171 ........ 05-29 09:32:43.324   235   235 I str_params: key: 'A2dpSuspended' value: 'true'
```

- After the call is disconnected, the headset state machine updates the A2DP device to be ready for streaming, as shown in the following log.

```
097841 ........ 05-29 09:33:30.674   235 10129 V SRS_Proc: ParamSet string: A2dpSuspended=false
097843 ........ 05-29 09:33:30.674   235 10129 D audio_hw_primary: adev_set_parameters: enter: A2dpSuspended=false
097849 ........ 05-29 09:33:30.674   235 10129 V listen_hw: handle_set_parameters: Enter kvpairs=A2dpSuspended=false
097859 ........ 05-29 09:33:30.674   235 10129 I str_params: key: 'A2dpSuspended' value: 'false'
```

If the above log is not generated, then A2DP will not resume on the connected headset.

This confirms that Bluetooth states are correct and the issue is probably related to non-playback.

6.4.6 Virtual call scenario

Android Bluetooth provides support for opening SCO audio connections for VOIP applications using the virtual voice call interface.

Any similar applications must request audio service to use the Bluetooth SCO using the following Bluetooth SDK API:

`startScoUsingVirtualVoiceCall and stopScoUsingVirtualVoiceCall.`

While debugging an audio issue, check the highlighted logs for the APIs.

```
0279 ........ 03-05 09:18:03.352  1196  1721 D BluetoothHeadset: startScoUsingVirtualVoiceCall()
0283 ........ 03-05 09:18:03.352   215  1981 V SRS_Proc: ParamSet string: A2dpSuspended=true
0285 ........ 03-05 09:18:03.352   215  1981 D audio_hw_primary: adev_set_parameters: enter: A2dpSuspended=true
0291 ........ 03-05 09:18:03.352   215  1981 V listen_hw: handle_set_parameters: Enter kvpairs=A2dpSuspended=true capture=0
0301 ........ 03-05 09:18:03.352   215  1981 I str_params: key: 'A2dpSuspended' value: 'true'
0793 ........ 03-05 09:18:03.462  3849  3875 D HeadsetStateMachine: onCodecNegotiated: The value is: 2
0895 ........ 03-05 09:18:03.472  3849  3878 D HeadsetStateMachine: Set sample rate: 16000
2371 ........ 03-05 09:18:14.132  1196  1848 D BluetoothHeadset: stopScoUsingVirtualVoiceCall()
2415 ........ 03-05 09:18:14.142   215  1981 V SRS_Proc: ParamSet string: A2dpSuspended=false
2417 ........ 03-05 09:18:14.142   215  1981 D audio_hw_primary: adev_set_parameters: enter: A2dpSuspended=false
2423 ........ 03-05 09:18:14.142   215  1981 V listen_hw: handle_set_parameters: Enter kvpairs=A2dpSuspended=false capture=0
2435 ........ 03-05 09:18:14.142   215  1981 I str_params: key: 'A2dpSuspended' value: 'false'
```

In-call low-power mode

Issues were observed in audio routing (or no audio playback) due to the following:

- Failure of synchronization of sniff mode
- Open/close of SCO connections along with sending HFP indicators while in sniff mode

Bad sniff values impact the SCO disconnect from the host as the host does not exit sniff for the disconnection of SCO.

If the link is in sniff mode, this delay can impact the A2DP resume after the call ends.

Use the OTA and snoop logs to analyze these issues.

6.5 Change dynamic audio profile version

By default, the audio and video remote control profile (AVRCP) version is 1.3.

- Change the local AVRCP version based on the remote AVRCP version. To enable the advanced features, unpair and pair the remote device.

Similarly, for HFP AG, by default, the version is 1.6.

Change HFP AG version based on the remote HFP device. To enable this advanced feature, unpair and pair the remote device.

7 Log data profiles

Prerequisites to capture log data profiles

Run the following command to enable the capturing of logcat logs with the timestamp enabled, before generating a profile:

```
adb logcat -v threadtime
```

7.1 Enable logs for Object Exchange (OBEX) profiles

To generate log for OBEX profiles, do the following:

1. Boot the Android device.
2. Go to **Settings**Turn on **Bluetooth**.
3. Place the device in Discoverable mode.
4. Use a remote corresponding client (or server) profile to connect to server (or client) on DUT.

   Example: Use MAP client [MCE] to connect to the MAP server [MSE].

By default, only the DEBUG info or logs are enabled in logcat.

VERBOSE logging from application layer runtime can be controlled through adb shell.

1. To enable VERBOSE logging from adb shell, use the following command:

   ```
   $adb shell setprop log.tag.<"Profile Specific LOG_TAG"> VERBOSE
   ```

2. Turn **Off/On Bluetooth** from Settings.
3. To disable VERBOSE logging, use the following command:

   ```
   $adb shell setprop log.tag.<"Profile Specific LOG_TAG"> DEBUG
   ```

4. Turn **Off/On Bluetooth** from Settings.
5. Replace the following “LOG_TAG” for corresponding profiles:
   - BluetoothMap > MAP Server
   - BluetoothPbap > PBAP Server
   - BluetoothSap > SAP Server
   - BluetoothOpp > Opp Server and OPP Client
   - BluetoothObex > OBEX Server and OBEX Client
   - BluetoothPan > PANU and NAP
   - BluetoothHid > Hid Host

- BluetoothFtp > FTP Server
- BluetoothDun > DUN Server

7.2 Use cases for OBEX MAP server profile logs

The following sections describe use cases for OBEX MAP server profile logs.

7.2.1 Multiple recipients not displayed in carkit TO and CC columns

Root cause and description

In Get Message Response from MSE, the To and Cc email addresses composed with triangular braces '<' and '>' were not supported for carkit.

Issue

The remote carkit does not support message parsing format as per MSE standards.

If multiple recipients are present in the To field, the same names are not displayed in the carkit.

Solution

The FORD SYNC CARKIT-specific fix does not include "<" and ">" in email address information sent to MSE.IOT.

7.2.2 New lines for SMS message body not displayed

Root cause and description

A Get Message Response from MSE must append a <CRLF> carriage return line feed for each line as per MAP Spec 3.1.3. However, this is not supported on FORD SYNC CARKIT.

Issue

Carkit is not able to read new lines for SMS.

Solution

Perform a generalized fix to solve similar IOT issues, which should solve the similar issues on other carkits.

Therefore, the plan is to maintain a 'blacklist' for such carkits (i.e., a list of specific issues with specific carkits).

7.2.3 Host fails to register for MAP service after Airplane mode

Root cause and description

After a quick airplane mode On/Off, if the MAP SDP cannot listen or register, host fails to register for MAP service.

Log analysis

- MAP service successfully registered on Bluetooth ON.
- Server socket listener started on both MAS ID 0 and MAS ID 1.

Hence, both entries in MAP SDP service record are available, as shown in the following highlighted log:

```
14:24:17.151 V/BluetoothMapService( 1640): mMasId is 1
```

After subsequent Bluetooth TURN OFF and ON trials, MAP service socket listening and registration fails.

Therefore, MAP SDP addition fails as shown in the following highlighted log:

```
14:24:48.041 E/BluetoothMapService( 1640): Error to create listening socket after 10 try
```

Issue

After a quick airplane mode on/off, MAP service is not advertised.

Solution

This is a cleanup issue, while handling Bluetooth TURN OFF state.

The code changes have closed the listen server socket and updated proper flags to a avoid resource leak.

The issue also causes subsequent RFCOMM socket listening trials (10 times) to fail.

7.3 Use cases for OBEX PBAP server profile logs

The following sections describe use cases for OBEX PBAP server profile logs.

7.3.1 Functionality searching

Remote device searches for a particular number in DUT and response is sent twice from DUT

Analyze the issue based on the following highlighted log:

```
01 ........ 06-04 17:15:38.402  1539  4410 D BluetoothPbapObexServer: OnGet type is x-bt/vcard-listing; name is null
```

Issue

The response is sent twice, while the remote device was searching for a particular number in DUT.

Solution

Add the check to limit the search.

As a result of this, if the contact name is sorted through number, then the same contact is not searched more than once.

7.3.2 Functionality parsing

Issue

Unable to make call when selecting contact from PBAP. The hands-free call fails due to extra characters.

Solution

This issue is fixed by removing extra spaces from the telephone number in the Vcard entry formation.

8 Log GAP/Core stack

Prerequisites

Run the following command to enable the capturing of logcat logs with the timestamp enabled, before enabling the log for Generic Access Profile (GAP)/Core stack:

```
adb logcat -v threadtime.
```

8.1 Enable logging for GAP/core stack

To enable the log for GAP/core stack, do the following:

1. Run the following commands:

   ```
   adb root
   Adb pull /etc/bluetooth/bt_stack.conf
   ```

2. Open bt_stack.conf file and add the following values:

   ```
   TRC_BTM=5
   TRC_HCI=5
   TRC_L2CAP=5
   TRC_RFCOMM=5
   TRC_SDP=5
   TRC_BTAPP=5
   TRC_BTIF=5
   ```

3. Run the following adb command:

   ```
   adb push bt_stack.conf /etc/bluetooth/
   ```

8.2 Troubleshoot BTIF layer issues

Most of the GAP-related functions are implemented in the btif_dm.c file for BTIF layer.

Use the btif_storage.c file to set the adapter/device properties.

All the information is placed at /data/misc/bluedroid/bt_config.conf of DUT.

Frequent issues related to BTIF layer include the following:

- Stuck in pairing.
- Adapter/remote device information that does is not updated immediately.
- File Descriptor (FD) leak-related issues.

- GAP state machine issues due to wrong handling of events.
- Turning bluetooth on or off.

8.2.1 Stuck in pairing

In the Bluedroid stack design, the pairing process includes the following sequence:

- After the authentication is successful, the application waits until the completion of SDP search
- After SDP search is completed, it posts the actual BONDED event to the upper layers. This is done even though Bluedroid stack design is actually paired from the lower layers.

SDP itself fails even after retrial of the SDP search. In this case, SDP does not post the BONDED event, which leads to pop-up the dialog box showing **Pairing to the device xxx**.

The process is recovered only when the Bluetooth is turned off and on.

Issue

Even though pairing is successful, the device does not show up in the paired list, after the Bluetooth is turned off or on.

Solution

Handle the discovery complete event properly.

Reset the SDP variables properly, and post the bonded event to the upper layers.

Log analysis

If paring is completed from the snoop log (Frame #50), the host receives the link key notification.

Before the SDP search request is received from the host, the remote device disconnects the link and there is no SDP search initiated from the host side.

This is the next step from the host side after completion of the authentication procedure.

NOTE The bonded device information must be sent to the upper layers.

Use the following highlighted log for analysis.

```
03:01:18.829  3314  3811 D bt-btif : btif_dm_search_services_evt Remote Service SDP done. Call bond_state_changed_cb BONDED
```

For the pairing process, this log should appear if the device is going to be paired.

8.2.2 Adapter/remote device information not updated immediately

From Android KitKat onwards, Google has introduced a delayed configuration file update to avoid frequent updates to the bt_config.xml file in the following location:

/data/misc/bluedroid/

This delay has been introduced to avoid the back-to-back updating of the configuration file during the inquiry due to multiple inquiry results.

Due to the waiting time, there is a possibility that critical information is not updated and it could include the following:

- Bonded device information
- Discoverability timeout
- Alias name

System events like battery removal (adb reboot) result in waiting time.

The mechanism is to wait for updates every three seconds.

If no update is received within three seconds, the system directly writes it to the configuration file. If an update is received within three seconds, the system caches the current changes and waits for another three seconds.

The system waits for only five minutes.

For more details, see the following highlighted log:

```
D/bt-btif (13097): btif_config_save(L342): processing_save_cmd:1, cached change:88
```

The system accumulates 95% of the changes that are ready for an update.

8.2.3 File descriptor (FD) leak-related issues

The current Bluetooth process allows a maximum of 1024 opened files.

If any module tries to open a new file beyond the maximum limit, the **Error 24** alert message is displayed.

The following are some of the system calls that open a file:

- Open
- Socket pair that is used by btif layer for rfcomm socket
- Socket

The local and Bluetooth server sockets are used in the Bluetooth on/off path.

The following FD leak issues are present on the Google base code:

- The local server socket connection has an FD leak issue. As part of the Bluetooth process, the local server connection is used for the BTC application layer and it does not close the FD server.
- The Bluetooth server socket connection also has the same FD leak issue, and it is used for all application layer profiles.

```
--------------------------------------------------------------------------
05-01 20:55:25.019  9731 12815 F Looper  : Could not create wake pipe.
errno=24
--------------------------------------------------------------------
01-06 22:55:59.948 E/bt-btif ( 1836): sock accept failed (Too many open
files)
01-06 22:55:59.948 I/bt-btif ( 1836): NEW FD -1
--------------------------------------------------------------------------
```

8.2.4 Bluetooth on/off issues

Bluetooth on/off issues include the following:

- SOC failures
- Framework (for example, delay in intent\binder)
- JNI (for example, initNative) not called
- Lower layer (for example, callbacks) not received on time.
- Timing/concurrency

8.2.4.1 SOC failures

After the SOC firmware is downloaded, the firmware prepares all the layers and initializes SMD. The vendor then downloads the firmware and gives a callback to HCI of preload event complete.

SOC download fails display the following message:

max-retry:0)...

8.2.4.2 Framework issues

Use case: Delay in intent/binder failure

The following timeout occurs when binding fails on Bluetooth on/off path.

MESSAGE_TIMEOUT_BIND

The following timeouts occur in case of intent failure:

```
CURRENT_STATE=PENDING, MESSAGE = START_TIMEOUT, isTurningOn= True
CURRENT_STATE=PENDING, MESSAGE = ENABLE_TIMEOUT, isTurningOn=
CURRENT_STATE=PENDING, MESSAGE = STOP_TIMEOUT, isTurningOn=
```

8.2.4.3 JNI issues

Use case: initNative not called

This issue may occur due to a problem with the framework, such as delay in intent/binder failure, leading to improper cleanup of resources at JNI.

To analyze the issue, use the following highlighted log file:

```
19:43:22.627 E/BluetoothServiceJni( 2773): Error while setting the callbacks
```

8.2.4.4 Lower layer

Use case: Callbacks not received, or callbacks not received on time.

While stack init bt_hc_worker_thread starts, it loops infinitely and exits.

HCI resets callbacks/timeout or cleanup.

This thread exits only if it gets the following event: HC_EVENT_EPILOG or HC_EVENT_EXIT.

9 Log BLE stack/profiles

This section describes the log capture and initial analysis for the connect and disconnect sequences for Bluetooth Low Energy (BLE) stacks or profiles.

9.1 Prerequisites

The prerequisites for BLE stack profile log configurations includes the following:

- Set Bluedroid logcat level to 6.
- Logcat must have timestamps.
- Provide approximate timestamps in the CR details to clarify the location of the problem.
- Make FTS HCI trace available.
- Set bookmarks for FTS HCI trace to indicate the problem.
- For example, a connection is not going through, and it puts a bookmark showing that the HOST initiated the LE_Create_Connection, but the Controller does not react.
- If any unexpected disconnection is noticed, a bookmark **unexpected disconnection** is put in FTS trace.
- Logging for BLE profiles
- Offloads:
  - Navigate to the /device/qcom/common/bdroid_buildcfg.h file
  - Enable the following BLE Vendor Specific Capabilities flag on the Host side:
  - `#define BLE_VND_INCLUDED TRUE`
  - Enabled the following BLE Offloads flag to work from Bluetooth host side:

    ```
    #define BLE_VND_INCLUDED   TRUE
    ```
- If a controller does not support the Google Offload mode, and `BLE_VND_INCLUDED` flag is enabled from host, then there are vendor specific commands to read the controller capabilities and for the host to act accordingly.

9.1.1 Capture logs for BLE profiles

To capture logcat logs with the timestamp enabled, use the following command:

```
adb logcat -v threadtime.
```

To capture logs in the Bluetooth stack, use the following commands:

```
adb root
Adb pull /system/etc/bluetooth/bt_stack.conf
```

The following are the BLE-related configurations in bt_stack.conf and add log trace level 6:

```
TRC_BTM=6
TRC_HCI=6
TRC_L2CAP=6
TRC_GATT=6
TRC_SMP=6
TRC_BTAPP=6
TRC_BTIF=6
TRC_GAP=6
TRC_HID_HOST=6
Adb push bt_stack.conf /system/etc/bluetooth/
```

Apart from these logging portions, PTS Helpers are added in the latest code for passing some of the PTS cases as they are not possible to hit with the normal operations from the apps side, the list of PTS cases per configuration below can be separately provided.

To enable PTS Helpers on the bt_stack.conf file, do the following:

```
# PTS testing helpers
# Secure connections only mode.
# PTS_SecurePairOnly=true
# Disable LE Connection updates
#PTS_DisableConnUpdates=true
# Disable BR/EDR discovery after LE pairing to avoid cross key derivation
errors
#PTS_DisableSDPOnLEPair=true
# SMP Pair options (formatted as hex bytes) auth, io, ikey, rkey, ksize
# PTS_SmpOptions=0xD,0x4,0xf,0xf,0x10
# SMP Certification Failure Cases
# Fail case number range from 1 to 9 will set up remote device for test
# case execution. Setting PTS_SmpFailureCase to 0 means normal operation.
# Failure modes:
#  1 = SMP_CONFIRM_VALUE_ERR
#  2 = SMP_PAIR_AUTH_FAIL
#  3 = SMP_PAIR_FAIL_UNKNOWN
#  4 = SMP_PAIR_NOT_SUPPORT
#  5 = SMP_PASSKEY_ENTRY_FAIL
#  6 = SMP_REPEATED_ATTEMPTS
#  7 = PIN generation failure?
#  8 = SMP_PASSKEY_ENTRY_FAIL
#  9 = SMP_NUMERIC_COMPAR_FAIL;
#PTS_SmpFailureCase=0
```

9.2 Examples of logs for BLE profiles

The following section describes examples of logs for BLE profiles.

9.2.1 BLE GATT client application issues direct connection

Log analysis

The following are the upper layer log messages for a direct connection request from the BLE client application:

```
01-02 00:12:19.889  3285  3285 D BluetoothGatt: connect() - device:
44:13:19:02:E1:C5, auto: false
01-02 00:12:19.889  3285  3285 D BluetoothGatt: registerApp()
01-02 00:12:19.899  1896  1957 D BtGatt.GattService: clientConnect() -
address=44:13:19:02:E1:C5, isDirect=true
……
```

The following are the Bluetooth stack log messages for a direct connection from the BLE client application.

```
…….
01-02 00:12:39.819  1896  2018 I bt-btm  : btm_ble_connected
…….
01-02 00:12:39.819  1896  2018 D bt-att  : GATT   ATT protocol channel with
BDA: 44131902e1c5 is connected
…..
01-02 00:12:39.859  1896  1957 D bt-btif : BTA_DM_LINK_UP_EVT. Sending
BT_ACL_STATE_CONNECTED
……
01-02 00:12:39.859  1896  1957 D BtGatt.GattService: onConnected() -
clientIf=7, connId=7, address=44:13:19:02:E1:C5
01-02 00:12:39.869  3285  3297 D BluetoothGatt: onClientConnectionState() -
status=0 clientIf=7 device=44:13:19:02:E1:C5
```

9.2.2 BLE server application initiates BLE connection request

Log analysis

The following are the upper layer log messages for a BLE connection request from the GATT server application:

```
01-02 00:31:19.509  4869  4869 D BluetoothGattServer: connect() - device:
44:13:19:02:E1:C5, auto: false
01-02 00:31:19.509  1896  3335 D BtGatt.GattService: serverConnect() -
address=44:13:19:02:E1:C5
```

The following are the Bluetooth stack log messages for a BLE connection from the GATT server application:

```
01-02 00:31:36.469  1896  2018 I bt-btm  : btm_ble_connected
……..
01-02 00:31:36.469  1896  2018 D bt-att  : GATT   ATT protocol channel with
BDA: 44131902e1c5 is connected
……..
```

```
01-02 00:31:36.479  1896  1957 D BtGatt.GattService: onConnected() connId=5,
address=44:13:19:02:E1:C5, connected=true
……..
01-02 00:31:36.489  4869  4880 D BluetoothGattServer:
onServerConnectionState() - status=0 serverIf=5 device=44:13:19:02:E1:C5
01-02 00:31:36.489  1896  1957 D bt-btif : BTA_DM_LINK_UP_EVT. Sending
BT_ACL_STATE_CONNECTED
```

9.2.3 BLE GATT server application initiates disconnection

Log analysis

The following are the upper layer log messages for disconnection request from GATT server application.

```
01-02 00:25:09.589  4707  4707 D BluetoothGattServer: cancelConnection() -
device: 44:13:19:02:E1:C5
01-02 00:25:09.589  1903  1919 D BtGatt.GattService: serverDisconnect() -
address=44:13:19:02:E1:C5, connId=5
01-02 00:25:09.589  1903  1919 D BtGatt.btif: btif_gatts_close
```

There may be debug prints like onConnectionStateChange with state disconnected which implies that the LE disconnect request was sent successfully to the lower layers by the upper layer applications.

This does not imply that the actual ACL disconnection happened.

The stack triggers actual disconnection only when none of the applications hold the BLE link.

The following lower layer prints show that the physical BLE link has been disconnected.

This will result in ACTION_ACL_DISCONNECTED intent sent by the lower layers.

```
01-02 00:25:19.059  1903  4846 D bt-att  : GATT   ATT protocol channel with
BDA: 44131902e1c5 is disconnected
……..
01-02 00:25:19.069  1903  4809 D bt-btif : BTA_DM_LINK_DOWN_EVT. Sending
BT_ACL_STATE_DISCONNECTED
```

The reason for disconnection is **Local User Initiated Disconnect**.

9.2.4 BLE GATT client application initiates disconnection

Log analysis

The following are the upper layer log messages for disconnection request from GATT client application.

```
01-02 00:33:04.449  2847  2847 D BluetoothGatt: close()
01-02 00:33:04.449  2847  2847 D BluetoothGatt: unregisterApp() - mClientIf=5
01-02 00:33:04.449  1185  1240 D BtGatt.GattService: unregisterClient() -
clientIf=5
01-02 00:33:04.449  1185  2977 I bt-att  : GATT_Disconnect conn_id=5
```

Or

```
01-02 02:14:35.669 D/BluetoothGatt( 6095): cancelOpen() - device:
45:9C:9D:5E:1B:A201-02 02:14:35.669 D/BtGatt.GattService( 1858):
clientDisconnect() - address=45:9C:9D:5E:1B:A2, connId=601-02 02:14:35.669 D/
BtGatt.btif( 1858): btif_gattc_close
……
```

There may be debug prints like onConnectionStateChange with state disconnected which only means that the LE disconnect request was sent successfully to the lower layers by the upper layer application.

This does not imply that the actual ACL disconnection happened.

The actual disconnection will be triggered by the stack only when none of the applications hold the BLE link.

The following lower layers prints show that the physical BLE link has been disconnected.

This will result in ACTION_ACL_DISCONNECTED intent sent by the lower layers.

```
01-02 00:25:19.059  1903  4846 D bt-att  : GATT   ATT protocol channel with
BDA: 44131902e1c5 is disconnected
……..
01-02 00:25:19.069  1903  4809 D bt-btif : BTA_DM_LINK_DOWN_EVT. Sending
BT_ACL_STATE_DISCONNECTED
```

The reason for disconnection is **Local User Initiated Disconnect**.

9.2.5 BLE GATT Application receives unexpected disconnection

The DUT application did not initiate this disconnection.

The following Bluetooth stack log message prints are present WITHOUT the disconnection API prints of the upper layers for remote initiated disconnection.

```
01-02 01:53:37.409  1968  5622 D bt-att  : GATT   ATT protocol channel with
BDA: 00a0c6814dac is disconnected
……….
01-02 01:53:37.419  1968  5622 D bt-att  : exit gatt_cleanup_upon_disc
01-02 01:53:37.419  1968  5622 D bt-att  : ATT disconnected
01-02 01:53:37.419  1968  5622 I bt-smp  : SMDBG l2c smp_connect_cback
01-02 01:53:37.419  1968  5622 D bt-btm  : btm_acl_removed ……
01-02 01:53:37.419  1968  5622 D bt-btm  : BTM_BLI_ACL_DOWN_EVT
```

The reason is not **Local User Initiated Disconnect**. There is NO disconnect request from the local HOST here.

If the FTS HCI logs contain an unexpected disconnection frame because of a connection timeout, LMP timeout, a failure to establish connection (0x3E), or an MIC error, it is a controller/RF issue that the controller team must look into.

If the FTS HCI logs contains an unexpected disconnection frame because of **Remote User Terminated Connection,** then the remote user issues the disconnection.

10 Android Configuration

This chapter details the Android Bluetooth profile lists and the steps for Bluetooth configuration on Android.

10.1 Enable Bluetooth profile

Bluetooth packages have flags to turn on/off profile service.

To manually enable or disable Bluetooth profiles, do the following:

1. Go to packages/apps/Bluetooth/res/values/
2. Open config.xml.
3. Change values as appropriate.
4. Recompile Bluetooth.apk and reinstall apk on the target device.

10.1.1 Profile lists

The following Bluetooth profiles can be enabled on the device:

Table 10-1 List of profiles

Profiles	Name	Mutual exclusive	Counterpart profile
A2DP source	profile_supported_a2dp	Yes	A2DP_sink
A2DPSink	profile_supported_a2dp_sink	Yes	A2DP source
AVRCP controller	profile_supported_avrcp_controller	Yes	
Hands-free HF	profile_supported_hfpclient	Yes	Hands-free AG
Hans-free AG	profile_supported_hs_hfp	Yes	Hands-free HF
Health device	profile_supported_hdp	No	
HID Host	profile_supported_hid	No	
OPP	profile_supported_opp	No	
PAN BNAP	profile_supported_pan	No	
PBAP Server	profile_supported_pbap	Yes	PBAP Client
PBAP Client	profile_supported_pbapclient	Yes	PBAP Server
GATT server/client	profile_supported_gatt	No	
MAP server	profile_supported_map	No	
SAP server	profile_supported_sap	No	

Table 10-1 List of profiles (cont.)

Profiles	Name	Mutual exclusive	Counterpart profile
NAP and PANU	profile_supported_pan	No	
FTP Server	profile_supported_ftp	No	
DUN Server	profile_supported_dun	No	

NOTE If mutually exclusive is true, you cannot enable counterpart profile on the same device.

10.1.2 Overlay configuration

Due to overall system-wise configuration, default setting from package/apps/Bluetooth can be overwritten during compilation. This can be determined by checking the config.xml file in the following location:

`device/qcom/common/device/overlay/packages/apps/Bluetooth/res/values/config.xml`

10.2 Fluoride stack configuration

It is recommended not to change the Fluoride stack configuration.

If an update to the Fluoride stack is required, do the following:

1. Go to System/bt/include.
2. Open bt_target.h.
3. Change values as appropriate.
4. Recompile Fluoride stack and reinstall binary on target device.

The following table describes the configuration parameters in the Fluoride stack and their corresponding default values:

Table 10-2 List of configurations

Configuration	Default values
BTA_FS_INCLUDED	TRUE
BTA_AV_SINK_INCLUDED	TRUE
BTM_SAFE_REATTEMPT_ROLE_SWITCH	TRUE
L2CAP_DESIRED_LINK_ROLE	HCI_ROLE_MASTER
LE_L2CAP_CFC_INCLUDED	TRUE
BLE_SC_INCLUDED	TRUE
AVDT_NUM_SEPS	5
BTA_AV_CO_CP_SCMS_T	FALSE
SDP_AVRCP_1_6	TRUE
AVCT_COVER_ART_INCLUDED	TRUE

Table 10-2 List of configurations (cont.)

Configuration	Default values
AVCT_BROWSE_INCLUDED	TRUE
AVRC_ADV_CTRL_INCLUDED	TRUE

10.3 System properties

Some of the system properties can be customized by OEMs to change the debug level settings or log settings on each devices.

By default, all the settings are false.

10.3.1 Set system properties

To set the system properties, do the following:

1. Open ADB shell window with root permission.
   a. c:\> adb root
   b. c:\> adb shell

2. Set the system properties in the window.

c:\> adb shell setprop *system_property_value* true

10.3.2 Properties list

Property value	Default values	Comments
wc_transport.force_special_byte	FALSE	Collect special bytes in UART ring buffer. Recommend to enable it if UART debug is required
wc_transport.in_ringbuf_log	FALSE	Inbound ring-buffer log on UART side. Recommend to enable it if UART debug is required
wc_transport.out_ringbuf_log	FALSE	Outbound ring-buffer log on UART side. Recommend to enable it if UART debug is required
persist.service.bdroid.soclog	FALSE	Controller verbose log flag. For QXDM logging, enable it to get SOC log on QXDM application
persist.service.bdroid.fwsnoop	FALSE	If soclog is enabled and fwsnoop is on, controller verbose log will be going up to Bluetooth stack to save it onto btsnoop file. It has a benefit to get firmware log without QXDM attachment
bt.sap.pts	FALSE	Flag to enable/disable Immediate disconnection support in SAP Server. Enable this flag for passing SAP PTS test case TC_Server_DCN_BV_03

Property value	Default values	Comments
bt.avrcpct-passthrough.pts	FALSE	Flag to send pass PTS TCs for Vol up/down passthrough without using Bluetooth test application for passing PTS TCs - PTS TCs TC_CT_PTT_BV_02_I and TC_CT_PTH_BV_01_C
bluetooth.pts.force_a2dp_abort	FALSE	Property to send Abort command to PTS as required. Not to be enabled, by default.
persist.bt.max.a2dp.connections	FALSE	Number of simultaneous A2DP connections supported together. Allowed values: 1 and 2.
persist.bt.enable.multicast	FALSE	Property to allow multi-streaming on dual A2DP connection configuration between HS of similar capabilities. To be enabled only in conjunction with the persist.bt.max.a2dp.connections property.
persist.bt.max.hs.connections	FALSE	Number of simultaneous HF connections supported together. Allowed values: 1 and 2.
bt.pts.certification	FALSE	Set this property to TRUE to pass HFP AG 1.7 PTS test cases and to pass HSP PTS test case TC/AG/RAV/BV-02. Set this to FALSE once PTS testing is completed.
persist.service.bt.hfp.client	FALSE	Set this property to TRUE to enable HF client.
ro.bluetooth.hfp.ver	FALSE	Set this property to 1.6 or 1.7 to enable WBS in HFP AG or HFP client.
persist.service.bt.a2dp.sink	FALSE	Set this property to TRUE to enable A2DP sink.

10.4 NVM configuration

Based on the customer's hardware specification, modify the NVM parameters to fit the hardware performance requirements.

For more details on modifying NVM parameters, see *QCA9377 Bluetooth Configuration Parameters* (80-WL431-26).

10.5 Write Bluetooth address onto the fluoride stack

The following are the two ways in which Bluetooth address can be written onto the fluoride stack on Qualcomm chips.

Write OTP area for Bluetooth address

Due to hardware configuration, certain module makers have an added memory bank called one-time programmable (OTP) read-only memory to write Bluetooth address. Upon a factory process, the module makers write BD address onto an OTP area. After the BD address is written to an OTP area, the BT stack is not required to write Bluetooth address in the middle of rampatch or an NVM download because the chipset checks the OTP area to set local BT address during bringup.

To write OTP for Bluetooth address, contact the Qualcomm hardware customer engineering (CE) team because the procedure is different for each chipset.

Set BT address using the NVM way for MCC customers

The Bluetooth address can be set using either of the following two ways:

Using persist NV area:

By using the persist NV area, the physical path is set as `/persist/bluetooth/.bt_nv.bin`.

Based on the OEM's hardware design, if an OEM prepares a persistent area in /persist, this area can be used for reading Bluetooth address if an OS upgrade does not overwrite this area.

- btnvtool -- board-address xx.xx.xx.xx.xx.xx".
- Check the file creation in /persist/bluetooth/.bt_nv.bin.
- After Bluetooth off/on from UI, the chipset receives the correct Bluetooth address during initialization.

Using system property

- `ro.vendor.bt.bdaddr_path` —Set by kernel module after reading Bluetooth address from kernel area. This property is set to "/sys/module/bdaddress/parameters/bdaddress".
- `ro.vendor.bt.boot.macaddr` —Set BT address in a customized way. Enter the `setprop ro.vender.bt.boot.macaddr xx:xx:xx:xx:xx:xx` command. After Bluetooth off/on from UI, the chipset receives the correct Bluetooth address during initialization.

11 Transport driver and issue triaging

Android contains Bluetooth transport driver between Fluoride stack and Bluetooth SoC chip. This chapter provides instructions on how to debug transport driver issues.

11.1 Transport driver overview

Bluetooth transport driver controls the Bluetooth traffic and communicates between Bluetooth stack and SoC on top of UART interface.

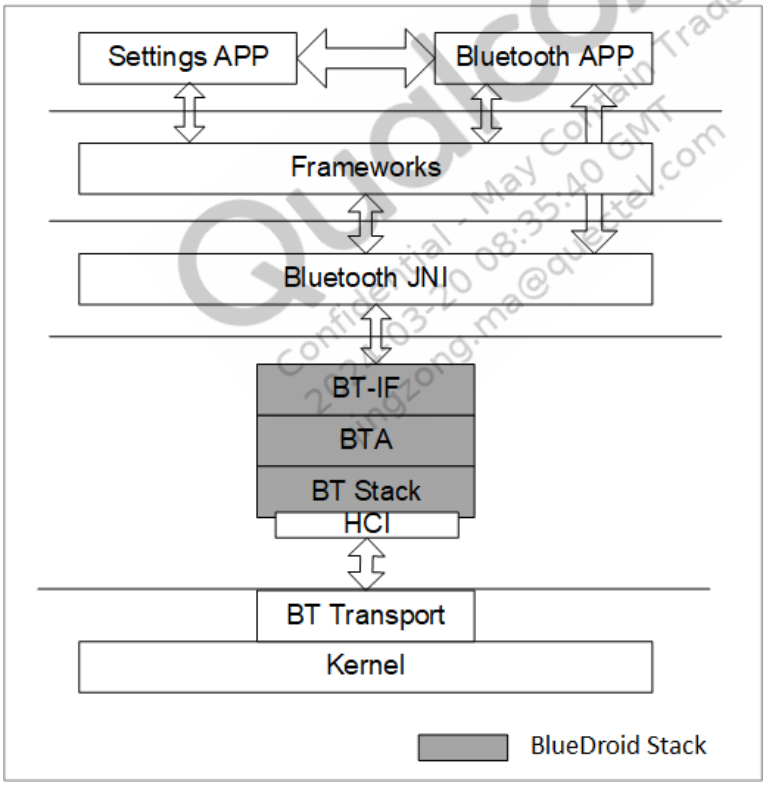


Figure 11-1 Bluetooth transport driver

11.2 HIDL software overview

HIDL is middle layer to serve upper services like Bluetooth/ANT/FM. It provides logical separation to support multiple upper services at the same time.

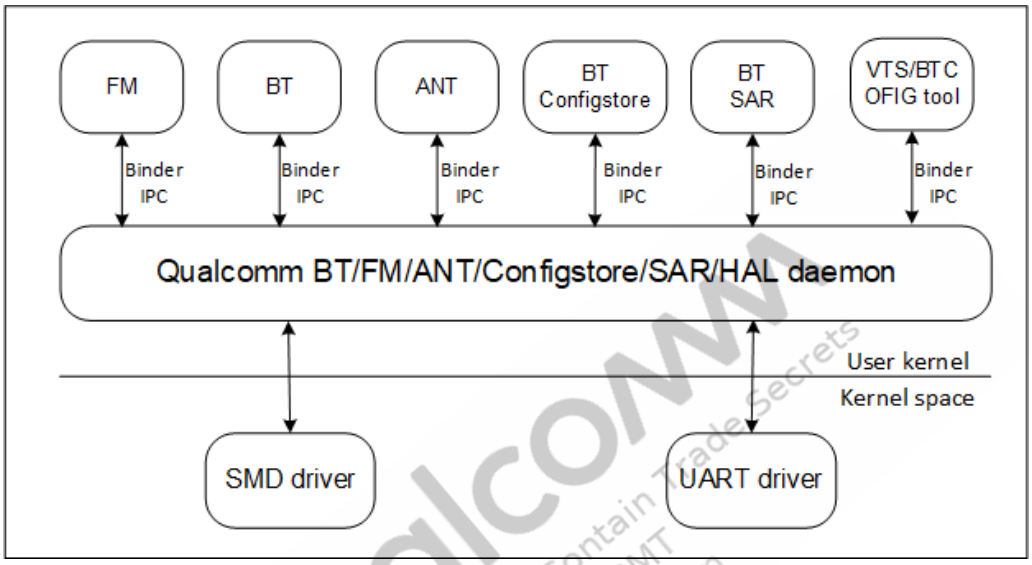


Figure 11-2 HIDL software

11.3 Logging

Bluetooth host transport driver supports the following logs for triaging of issues:

- Logcat: All Bluetooth Transport driver logs are prefixed with tag “vendor.qti.bluetooth”.
- HIDL snoop: tramdump_bt_snoop_2019-06-27_07-18-07.cfa
  - Snoop logs are generated during transport driver close initiated due to internal issue or when SoC crash’s itself. These logs capture all the data sent/received by the transport driver to Bluetooth SoC.
  - Snoop is also generated in normal close operation when persist.vendor.service.bdroid.dump_ringbuff is set to true.
  - Snoop is viewed in FTS tool.
  - All the IBS data is also logged in snoop as commands (when HOST sends IBS bytes) or events (when IBS bytes received)
- IPC logs: UART IPC logs are pulled as part of every SoC crash and populated in persistent filesystem (/data/vendor/ssrdump). UART IPC logs provide:
  - The sequence of bytes sent/received over UART to/from SoC
  - UART-related states

    Sample name:

    ```
    ramdump_bt_uart_ipc_pwr_2019-06-27_07-18-07.log
    ramdump_bt_uart_ipc_state_2019-06-27_07-18-07.log
    ```

```
ramdump_bt_uart_ipc_tx_2019-06-27_07-18-07.log
ramdump_bt_uart_ipc_rx_2019-06-27_07-18-07.log
```

- Bluetooth state log file: State logs are mini logcat state of Bluetooth Transport driver. The file name includes the timestamp also. For example, dump_bt_state_2019-06-27_07-18-07.log

  Few mandatory logs in stats:

```
13:26:02:873-Last Tx packet and timestamp 01 57 FD 02 00 01
             Actual last Tx pkt len = 6
13:26:03:066-Last Rx packet before SSR and timestamp 04 10 01 8F
13:26:03:378-Last Rx packet and timestamp
04 FF 1E FC 00 02 00 8F 00 53 75 6E 20 4A 75 6C 20 20 37 20 31
Actual last Rx pkt len = 33
```

  Primary reason for SoC Crash: SoC crashed with diag initiated SSR

  Secondary reason for SoC Crash: SSR CMD at time: Sun Jul 7 13:26:03 2019

- Kernel logs: These are necessary to isolate other system level issues.
- Mapping TS between tombstone, logcat and IPC logs and confirming if all proper logs are available for issue triaging.

Filename	What to check	Comments
tombstone_xx	Timestamp: 2019-09-21 23:01:28-0700 09-21 23:01:28.783 7454 8375 D vendor.qti.bluetooth@1.0-btstateinfo: TS System 06:01:28:743 TS kernel 10528:220467154	Give the system time when forced abort is triggered in HIDL Check the time of logcat and kernel log TS, which needs to be mapped in IPC, logs
ramdump_bt_state_2019-09-21_23-01-28.log	ramdump_bt_state_2019-09-21_23-01-28.log	Check if the shared ramdump files have matching TS as per tombstone logs
ramdump_bt_uart_ipc_pwr_2019-09-21_23-01-28.log ramdump_bt_uart_ipc_state_2019-09-21_23-01-28.log ramdump_bt_uart_ipc_tx_2019-09-21_23-01-28.log ramdump_bt_uart_ipc_rx_2019-09-21_23-01-28.log	IPC Tx Logs: [10528.208138167/ 0x4671fcc60] DMA Tx[0x0000000000:1]: fd [10528.218186031/ 0x46722bdf8] DMA Tx[0x0000000000:1]: fd [10528.228346344/ 0x46725b7fc] DMA Tx[0x0000000000:1]: fd	Check if shared IPC logs have logs around the kernel TS mentioned in tombstone logs

11.4 SSR level and crashes

The following table captures the Bluetooth transport driver behavior during Bluetooth SoC crashes and other issues:

Reasons	SSR Level = 1 or 2		SSR Level = 3	
	User Debug Build	User Build	User Debug Build	User Build
Command timeout	Abort	Kill	Abort	Kill
Invalid bytes from SoC	Abort	Kill	Abort	Kill
SoC crashed	Kernel panic	Kernel panic	Abort	Kill
Rx thread stuck	Abort	Kill	Abort	Kill
Unable to wakeup SoC	Abort	Kill	Abort	Kill
Spurious wake	Abort	Kill	Abort	Kill
Diag SSR	Kernel panic	Kernel panic	Abort	Kill
Bluetooth Init timeout	Abort	Kill	Abort	Kill

11.5 Triaging issue – tombstone

11.5.1 Rx thread stuck

If the upper layer stack (Bluetooth/FM/ANT) takes more than 1 second to return the callback from Bluetooth Transport driver (due to logging, alarm-related issues), then Bluetooth transport driver timer to monitor the Rx thread expires and results in abort.

To debug any Rx thread stuck issue, logcat and kernel logs prior until issue TS are mandatory. Contact the host team for such issues, after basic triaging is performed with matching logs.

Rx thread stuck due to upper layer client stack issues

Logs	What to look for
Logcat and Bluetooth state log	In Bluetooth state logs and logcat, the following prints are observed: 1. Primary reason for SoC Crash: Rx Thread Stuck Secondary reason for SoC Crash: Default at time: Fri Jun 7 06:06:18 2019 2. The following logs are shown in logcat and the state logs too: TS of pre_stack_event_call_back 10:40:38:919-event callback--> TS of post_stack_event_call_back 10:40:38:909-event callback<-- TS of pre_stack_acl_call_back 10:40:38:692-data callback--> TS of post_stack_acl_call_back 10:40:38:692-data callback<-- When the following is true, then issue is not with the HIDL but upper layer client (Bluetooth Stack, ANT Stack, FM Stack). ■ Post stack event timestamp is less than < pre stack event timestamp ■ Post stack ACL timestamp is less than < pre stack ACL timestamp

Rx thread stuck due to low memory killer

This issue is caused by Android system running low on memory resources. When this happens, Android tries to get rid of some tasks. If these tasks are Bluetooth transport driver clients, it can result in Rx thread stuck issue. To debug any Rx thread stuck issue, logcat and kernel logs prior until issue TS is mandatory. Triage team must check for low memory killers in logcat and update the same in initial triage to host team.

Logs	What to look for
Logcat	// Low Memory killer killed Bluetooth PID abruptly Line 268341: 07-24 01:15:55.871 817 817 E lowmemorykiller: Kill 'com.android.bluetooth' (14501), uid 1002, oom_adj 0 to free 111864 KB Line 268342: 07-24 01:15:55.871 817 817 I lowmemorykiller: Reclaimed 111864kB at oom_adj 0 Line 268444: 07-24 01:15:56.009 3711 7059 I ActivityManager: Process com.android.bluetooth (pid 14501) has died: psvc PER In addition, the prints in Bluetooth state logs are also dumped into the logcat file.
Logcat and Bluetooth state Log	In Bluetooth state logs and logcat, the following prints are observed: 1. Primary reason for SoC Crash: Rx Thread Stuck. Secondary reason for SoC Crash: Default at time: Fri Jun 7 06:06:18 2019 2. The following logs are show in logcat and the state logs too: TS of pre_stack_event_call_back 10:40:38:919-event callback--> TS of post_stack_event_call_back 10:40:38:909-event callback<-- TS of pre_stack_acl_call_back 10:40:38:692-data callback--> TS of post_stack_acl_call_back 10:40:38:692-data callback<-- When the following is true, then issues are not with the HIDL. ■ Post stack event timestamp is less than < pre stack event timestamp ■ Post stack ACL timestamp is less than < pre stack ACL timestamp

Rx thread stuck due to user space tasks freezing

Logs	What to look for
Logcat	Logcat does not show any logs for many seconds around the timestamp at which primary crash reason is printed in logcat/state logs. For example, as per the logcat at below location, there are no logs for 20 seconds: \\waterfall\axiomdumps\prd\20190921\870c95e4-bd12-4605-aafa-6a95c9f4737c\Logs 09-21 21:10:38.150 10332 10332 E QCommandServer: Written to client 12 09-21 21:10:58.197 594 611 D DrmLibTime: got the req here! ret=0
Kernel logs	09-21 21:10:38.173 0 0 I : Freezing user space processes ... 09-21 21:10:58.191 0 0 E : Freezing of tasks failed after 20.017 seconds (3 tasks refusing to freeze, wq_busy=0) or other sample print - 06-25 06:47:18.582 0 0 I : Freezing user space processes ... 06-25 06:47:38.604 0 0 I : Restarting tasks ...
Logcat and Bluetooth state log	In Bluetooth state logs and logcat, the following prints are observed: 1. Primary reason for SoC Crash: Rx thread stuck. Secondary reason for SoC Crash: Default at time: Fri Jun 7 06:06:18 2019 2. The following logs are show in logcat and the state logs too: TS of pre_stack_event_call_back 10:40:38:919-event callback--> TS of post_stack_event_call_back 10:40:38:909-event callback<-- TS of pre_stack_acl_call_back 10:40:38:692-data callback--> TS of post_stack_acl_call_back 10:40:38:692-data callback<-- When the following is true, then issues are not with the HIDL. ■ Post stack event timestamp is less than < pre stack event timestamp ■ Post stack ACL timestamp is less than < pre stack ACL timestamp

11.5.2 Spurious wakeup timer

Spurious wakeup timer issue happens when SoC sends WakeUp request to Bluetooth transport driver, and then it:

- Does not send any event/data for 10 seconds OR
- Does not send Sleep request to Bluetooth transport driver for 10 seconds after sending the last event/data.

Bluetooth transport driver collects crash dump from SoC by sending special buffer (FBFB… 1100 times) and aborts post collection of SoC dumps. IPC logs are mandatory for triaging spurious wake issues, as the issue is either in BluetoothSoC/UART or external module like TZ is busy.

NOTE SoC team must be the primary triage team for such issues falling in case 1. For case 2, triage TAM must check kernel logs and route Jira to LSS team.

Spurious wakeup timer expired due to SoC/UART issues

Logs	What to look for
Kernel logs	No issue observed.
Logcat	09-21 21:10:58.200 844 3037 E vendor.qti.bluetooth@1.0-ibs_handler: ibs_spurious_wake_timeout: expired 09-21 21:10:58.200 844 3037 D vendor.qti.bluetooth@1.0-uart_controller: SsrCleanup: SSR triggered due to 17 sending special buffer
Logcat and Bluetooth state log	In Bluetooth state logs and logcat, the following prints are observed: 1. Primary reason for SoC Crash: SSR due to spurious wakeup

Spurious wakeup timer due to user space tasks freezing

Logs	What to look for
Kernel logs	09-21 21:10:38.173 0 0 I : Freezing user space processes ... 09-21 21:10:58.191 0 0 E : Freezing of tasks failed after 20.017 seconds (3 tasks refusing to freeze, wq_busy=0):
Logcat and Bluetooth state log	In Bluetooth state logs and logcat, the following prints are observed: 1. Primary reason for SoC Crash: SSR due to spurious wakeup
Logcat	09-21 21:10:58.200 844 3037 E vendor.qti.bluetooth@1.0-ibs_handler: ibs_spurious_wake_timeout: expired 09-21 21:10:58.200 844 3037 D vendor.qti.bluetooth@1.0-uart_controller: SsrCleanup: SSR triggered due to 17 sending special buffer

11.5.3 SoC wakeup issue

Before sending any data/command to SoC, Bluetooth transport driver tries to wake up SoC (if SoC was sent out FE previously). For this, it keeps sending WakeUp (FD) request to SoC 150 times (with interval of 10 ms) expecting WakeACK (FC) is received for every retrial. If no WakeUp ACK is received from SoC, Bluetooth Transport driver tries to collect crash dump from SoC by sending special buffer (FBFB… 1100 times) and aborts post collection of SoC dumps. IPC logs are mandatory for triaging Unable to Wake Up ACK issues, as the issue is in BluetoothSoC/UART modules. SoC team must be the primary triage team for such issues.

SoC wakeup issue: Hardware problems

Various bad hardware configurations causes SoC wakeup issues. For example, QCA639x.

- Confirm that the metal crossbars from the back of QCA639x card are removed before it is plugged into MTP. This results in various difficult to debug issues due to possible short circuit.
- For clock, Timex is used (not XO). Time is POR.

SoC wakeup issue: SoC/UART problems

Logs	What to look for
Kernel logs	No issue.
Logcat	09-12 17:43:38.858 21300 21353 D vendor.qti.bluetooth@1.0-ibs_handler: WakeRetransTimeout: Failed to get wake ACK from the SoC

Logs	What to look for
	09-12 17:43:38.858 21300 21353 D vendor.qti.bluetooth@1.0-ibs_handler: SerialClockVote: vote for UART CLK OFF 09-12 17:43:38.859 21300 21300 E vendor.qti.bluetooth@1.0-ibs_handler: DeviceWakeUp:SoC not responding, stop sending wake byte 09-12 17:43:38.859 21300 21300 E vendor.qti.bluetooth@1.0-ibs_handler: DeviceWakeUp: Failed to wake SoC ……. 09-12 17:43:46.897 21300 29428 D vendor.qti.bluetooth@1.0-btstateinfo: Primary reason for SoC Crash: Unable to wake SoC 09-12 17:43:46.904 21300 29428 E vendor.qti.bluetooth@1.0-uart_controller: Aborting daemon as SSR is completed!
Logcat and Bluetooth state log	Primary reason for SoC Crash: Unable to wake SoC
IPC logs	UART IPC Tx log shows the WakeRequests from Host to SoC (0XFDs), but UART IPC Rx log does not show any response (0xFC): IPC Tx Logs: [952.728186031952.718138167/ 0x46722bdf80x4671fcc60] DMA Tx[0x0000000000:1]: fd [952.738346344952.728186031/ 0x46725b7fc0x46722bdf8] DMA Tx[0x0000000000:1]: fd [952.748523636952.738346344/ 0x46728b34b0x46725b7fc] DMA Tx[0x0000000000:1]: fd [952.748523636/ 0x46728b34b] DMA Tx[0x0000000000:1]: fd IPC Rx Logs: Do not show any “FC” after 952.718.

11.5.4 SoC Initialization failed

When Bluetooth turns on, Bluetooth transport driver initializes the SoC. This includes executing the power-on sequence, UART calls, download Bluetooth firmware files (NVM and patches), performing HCI reset, starting read threads, and diag initialization. If Bluetooth SoC init is not performed in 2.9 seconds, Bluetooth stack triggers close on the Bluetooth transport driver, which triggers abort. IPC logs are mandatory for triaging these issues.

SoC initialization failed: SoC in bad state

Logs	What to look for
Kernel logs	No issue.
Logcat	bt_hci (5550): startup_timer_expired
Logcat and Bluetooth state log	// Sending and reading Get version cmd have valid timestamps. But Get Version resp rcvd TS is all 0 s. 09:49:15:760-Sending Get Version CMD to SoC 09:49:15:760-Reading Get Version CMD RSP from SoC 00:00:00:000- Get Version CMD RSP not rcvd from SoC ……, Primary reason for SoC Crash: SoC init failed during patch downloading

Logs	What to look for
	Secondary reason for SoC Crash: Default at time: Thu Sep 12 17:49:18 2019
IPC logs	IPC Tx logs show get version command sent, but corresponding IPC Rx logs do not exist. This is because GetVersion() is the first command sent by Bluetooth transport driver to Bluetooth SoC during initialization. [1744.273033951/ 0x911ded62a] DMA Tx[0x0000000000:5]: 01 00 fc 01 19

SoC initialization failed: First UART write blocked

Logs	What to look for
Kernel logs	No issue.
Logcat	bt_hci (5550): startup_timer_expired
Logcat and Bluetooth state log	// Get version req cmd sending has valid timestamp. But reading Get Version cmd rsp TS is all 0 s. 09:49:15:760- Sending Get Version CMD to SoC 00:00:00:000- Get Version CMD RSP not yet read 00:00:00:000- Get Version CMD RSP not rcvd from SoC ……, Primary reason for SoC Crash: SoC init failed during patch downloading Secondary reason for SoC Crash: Default at time: Thu Sep 12 17:49:18 2019
IPC logs	IPC Tx logs do not show any get version command.

SoC initialization failed: UART API delays

Logs	What to look for
Kernel logs	No issue.
Logcat	bt_hci (5550): startup_timer_expired …. // From logcat logs, we see some UART APIs taking long time. For example, we see UART opening and CLK on API took approx. 1.5 seconds, which leads to blocking Bluetooth transport driver for same time. Line 2591: 2019-09-08 19:25:14.996298 1002 730/22292 (0:main) I vendor.qti.bluetooth@1.0-uart_transport: InitTransport: opening /dev/ttyHS0 Line 2728: 2019-09-08 19:25:16.446886 1002 730/22292 (0:main) D vendor.qti.bluetooth@1.0-uart_transport: userial clock on
Logcat and Bluetooth state log	// Not even started to send the Get version cmd to soc. 00:00:00:000- Get Version CMD not sent to SoC 00:00:00:000- Get Version CMD rsp not yet read 00:00:00:000- Get Version CMD RSP not rcvd from SoC ……, Primary reason for SoC Crash: SoC init failed during patch downloading Secondary reason for SoC Crash: Default at time: Thu Sep 12 17:49:18 2019
IPC logs	IPC Tx logs do not show any get version command.

SoC initialization failed: Diag Init delayed

Logs	What to look for
Kernel logs	No issue.
Logcat	bt_hci (5550): startup_timer_expired …. // From logcat logs, we see that init thread TID 2577 has not completed and we received the close call before the Diag API could complete. 09-27 03:00:19.891 730 2577 I vendor.qti.bluetooth@1.0-patch_dl_manager: HciReset: HCI RESET 09-27 03:00:19.895 730 2577 D vendor.qti.bluetooth@1.0-patch_dl_manager: ReadCmdCmplEvent: Cmd Cmpl received for opcode c03 09-27 03:00:19.895 730 2577 D vendor.qti.bluetooth@1.0-uart_controller: add on features read true 09-27 03:00:19.976 730 2795 I vendor.qti.bluetooth@1.0-ibs_handler: ProcessIbsCmd: Received IBS_SLEEP_IND: 0xFE 09-27 03:00:20.396 730 2576 D vendor.qti.bluetooth@1.0-wake_lock: Release wakelock is released 09-27 03:00:21.048 730 2577 E Diag_Lib: diag: In do_mask_sync, mask sync error, count: 1001 09-27 03:00:21.893 730 730 E Diag_Lib: BluetoothDeathRecipient: Calling HAL close 09-27 03:00:21.893 730 730 W vendor.qti.bluetooth@1.0-bluetooth_hci: BluetoothHci::close() 09-27 03:00:21.893 730 730 W vendor.qti.bluetooth@1.0-data_handler: DataHandler::CleanUp() 09-27 03:00:26.249 730 2577 E vendor.qti.bluetooth@1.0-diag_interface: Init:Failed to Init diag
Bluetooth state log	Primary reason for SoC Crash: SoC init failed during patch downloading Secondary reason for SoC Crash: Default at time: Mon Sep 23 17:22:08 2019 TS System 17:22:08:787 TS kernel 46:790965396 17:22:01:456-HCI initialize rcvd from client type = 0 17:22:04:497-HCI Close rcvd from client type = 0 17:22:01:763- Sending Get Version CMD to SoC 17:22:01:783- Reading Get Version CMD RSP from SoC 17:22:01:783-Get Version rsp rcvd, num bytes in last rsp = 5 17:22:02:447- SoC initialization successful. Bluetooth SoC firmware SU build info: BTFM.CHE.3.2.0-00224-QCACHROM-2 17:22:02:447-Read thread was started: SUCCESS 17:22:08:710-Diag Init successful -> Diag init successful after close received Controller Init not completed -> Init callback not sent within 2.9 seconds
IPC logs	Nothing suspicious after receiving the HCI reset response.

11.5.5 Command timeout issues

Whenever Bluetooth stack sends any command to SoC, it starts an internal 2 seconds timer. Once the timer expires, Bluetooth stack sends debug info command to crash SoC, which internally converts HIDL to special buffer (FBFB…1100 times). For triaging command timeout issues, IPC logs are mandatory, and SoC team must be the initial triage team.

Command timeout - no response from SoC

Logs	What to look for
Kernel logs	No issue.
Logcat	Primary reason for SoC Crash: SSR due to command timed out
Logcat and Bluetooth state log	Primary reason for SoC Crash: SSR due to command timed out
IPC logs	[87263.110569835/ 0x186d860371e] DMA Tx[0x0000000000:1]: 01 [87263.110619106/ 0x186d8603ad0] DMA Tx[0x0000000000:5]: 0a fc 02 05 00 << VS_A2DP_Cmd SubOpcode: HCI_VS_STOP_A2DP_MEDIA [87263.114383116/ 0x186d8615521] DMA Rx[0x0000000000:7]: 04 05 04 00 0a 00 08 << Not command complete event, but disconnection complete event due to connection timeout [87264.11071358887263.153656866/ 0x186d98539e60x186d86cd6a6] DMA Rx[0x0000000000:1]: fe [87264.110713588/0x186d98539e6] DMA Tx[0x0000000000:1]: fe [87264.110768796/0x186d9853e0a] vote_clock_offPOSIX timer 4 ioctl 0 usage_count 1 [87264.255223797/0x186d9af902b] msm_geni_serial_runtime_suspend: [87264.255250047/0x186d9af9222] msm_geni_wakeup_isr: Edge-Count 0 [87265.117830518/0x186daac47ab] msm_geni_serial_runtime_resume: [87265.117840674/0x186daac486e] vote_clock_onbluetooth@1.0-s ioctl 1 usage_count 0 [87265.118009841/0x186daac551e] DMA Tx[0x0000000000:1]: fd [87265.119844789/0x186daacdebf] DMA Rx[0x0000000000:1]: fc // Host crashes SoC as no command complete event received for HCI_VS_STOP_A2DP_MEDIA for 2 seconds [87265.120163956/0x186daacf6ad] DMA Tx[0x0000000000:184]: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb [87265.120807758/0x186daad26f7] DMA Tx[0x0000000000:916]: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb

Command timeout - delayed response from SoC

Logs	What to look for
Kernel logs	No issue.
Logcat	Primary reason for SoC Crash: SSR due to command timed out
Logcat and Bluetooth state log	Primary reason for SoC Crash: SSR due to command timed out
IPC logs	[662.266883843/0x309a547b9] msm_geni_serial_handle_dma_rx: -5 -> some issue in UART, which might be cause of the issue

Logs	What to look for

	//CMD sent
	[662.681812542/0x30a1ed75e] DMA Tx[0x0000000000:1]: 01
	[662.681910042/0x30a1edeb0] DMA Tx[0x0000000000:5]: 57 fd 02 00 01
	[663.682115879/0x30b43e61e] DMA Tx[0x0000000000:1]: fe
	[664.861274164/0x30c9d5afd] DMA Tx[0x0000000000:1]: fd
	[664.861450257/0x30c9d6834] DMA Rx[0x0000000000:1]: fc
	// No response for 2 seconds, leading host sending special buffer to crash SoC
	[664.861707080/0x30c9d7b75] DMA Tx[0x0000000000:453]: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
	[664.863301143/0x30c9df309] DMA Tx[0x0000000000:647]: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
	[665.211488644/0x30d03f515] DMA Rx[0x0000000000:1]: fd
	// Response for cmd seen now
	[665.212048175/0x30d041f0c] DMA Rx[0x0000000000:9]: 04 0e 06 01 57 fd 00 00 01

11.5.6 SoC crash issues

Whenever SoC crashes on its own, ramp dump is collected by transport driver along with IPC logs, and secondary reason is set to reason for SoC crash as retrieved from VS event before receiving SoC Dump. Primary reason is set to “SoC crashed”. SoC dump file is mandatory for SoC team to check the reason for crash along with IPC logs found in data/vendor/ssrdump folder. SoC team must be primary team to triage these issues.

Logs	What to look for
Kernel logs	No issue.
Logcat	Primary reason for SoC Crash: SoC crashed
Bluetooth state log	Primary reason for SoC Crash: : SoC crashed Secondary reason for SoC Crash: Assert at time: Mon Sep 23 21:23:05 2019

11.5.7 ANR in com.android.bluetooth

Application No Response (ANR) is an issue in Common Bluetooth package (com.android.bluetooth).

ANR due to blockage of main thread in HIDL

HIDL transport driver has only 1 thread for all binder calls from upper layer client (Bluetooth/FM/ANT). If the main thread is blocked, the binder calls from upper layer client will not reach HIDL and hence the calling thread will be blocked leading to ANR’s. If the main thread is blocked for any reason in HIDL due to deadlock, or kernel call stuck, based on the previous experiences, it is an unrecoverable issue, and only device reboot can fix it or if the HIDL PID is killed forcefully for auto restart of HIDL daemon.

For triaging ANR issue, use the following steps:

1. Check the TS of ANR crash mapping to Jira and the backtrace to understand the cause of main thread blocked.
2. Check the logcat logs before the reported TS to check if main thread is blocked for HIDL, which might lead to ANR subsequently.

Logs	What to look for
Kernel logs	No issue.
ANR Trace	----- pid 24612 at 2019-09-24 09:15:49 ----- Cmd line: com.android.bluetooth "main" prio=5 tid=1 Native | group="main" sCount=1 dsCount=0 flags=1 obj=0x726d5e78 self=0x79662bec00 | sysTid=24612 nice=0 cgrp=default sched=0/0 handle=0x7967827ed0 | state=S schedstat= (125626974 49188861 128) utm=4 stm=7 core=6 HZ = 100 | stack=0x7fe3b9d000-0x7fe3b9f000 stackSize=8192 KB | held mutexes= kernel: (could not read /proc/self/task/24612/stack) native: #00 pc 00000000000d1084 /apex/com.android.runtime/lib64/bionic/libc.so (__ioctl+4) native: #01 pc 000000000008b6b0 /apex/com.android.runtime/lib64/bionic/libc.so (ioctl+132) native: #02 pc 000000000009dc84 /system/lib64/libhidlbase.so (android::hardware::IPCThreadState::transact(int, unsigned int, android::hardware::Parcel const&, android::hardware::Parcel*, unsigned int)+1772) native: #03 pc 0000000000099170 /system/lib64/libhidlbase.so (android::hardware::BpHwBinder::transact(unsigned int, android::hardware::Parcel const&, android::hardware::Parcel*, unsigned int, std::__1::function<void (android::hardware::Parcel&)>) +72) native: #04 pc 000000000008fd68 /system/lib64/libhidlbase.so (android::hidl::base::V1_0::BpHwBase::_hidl_interfaceChain(android::hardware::IInterface*, android::hardware::details::HidlInstrumentor*, std::__1::function<void (android::hardware::hidl_vec<android::hardware::hidl_string> const&)>)+216) native: #05 pc 0000000000091d04 /system/lib64/libhidlbase.so (android::hidl::base::V1_0::BpHwBase::interfaceChain(std::__1::function<void (android::hardware::hidl_vec<android::hardware::hidl_string> const&)>)+140) native: #06 pc 000000000005b3c0 /system/lib64/libhidlbase.so (android::hardware::details::canCastInterface(android::hidl::base::V1_0::IBase*, char const*, bool) +396) native: #07 pc 000000000005e5f8 /system/lib64/libhidlbase.so (android::hardware::details::getRawServiceInternal(std::__1::basic_string <char, std::__1::char_traits<char>, std::__1::allocator<char>> const&, std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>> const&, bool, bool)+1144) native: #08 pc 000000000000e4ac /product/lib64/vendor.qti.hardware.btconfigstore@1.0.so (_ZN7android8hardware7details18getService InternalIN6vendor3qti8hardware13btconfigstore4V1_017BpHwBTConfigStoreENS7_14 IBTConfigStoreEvvEENS_2spIT0_EERKNSt3__112basic_stringIcNSD_11char_traitsIcEENSD_9allocatorIcEEEEbb+204) native: #09 pc 00000000000030c8 /system/lib64/libbtconfigstore.so (getVendorProperties(unsigned int, std::__1::vector<vendor_property_t, std::__1:: allocator<vendor_property_t>>&)+148) native: #10 pc 000000000004cb74 /system/lib64/libbluetooth_jni.so (android::initNative(_JNIEnv*, _jobject*)+92) Above call stack shows that main thread is stuck in getVendorProperties HIDL binder call

Logs	What to look for
Logcat log	Line 79790: 09-24 09:12:19.779 814 19201 I vendor.qti.bluetooth@1.0-ibs_handler: ProcessIbsCmd: Received IBS_WAKE_IND: 0xFD Line 79791: 09-24 09:12:19.779 814 19201 I vendor.qti.bluetooth@1.0-ibs_handler: ProcessIbsCmd: Writing IBS_WAKE_ACK Line 79792: 09-24 09:12:19.780 814 19201 W vendor.qti.bluetooth@1.0-data_handler: OnPacketReady: Received event for command sent internally: 08 fc Line 79793: 09-24 09:12:19.780 814 814 D vendor.qti.bluetooth@1.0-uart_controller: UartController::Cleanup, soc_need_reload_patch=1 Line 79794: 09-24 09:12:19.780 814 19201 E vendor.qti.bluetooth@1.0-async_fd_watcher: ThreadRoutine: End of AsyncFdWatcher::ThreadRoutine Line 79795: 09-24 09:12:19.783 814 814 W vendor.qti.bluetooth@1.0-async_fd_watcher: StopThread: stopped the work thread Line 79796: 09-24 09:12:19.794 814 24222 D vendor.qti.bluetooth@1.0-diag_interface: BT_Diag_LSM_DeInit // HIDL main thread is blocked, no further logs for vendor.qti.bluetooth after above line until device rebooted. ANR logs seen periodically Line 89276: 09-24 09:15:51.417 1174 1558 E ActivityManager: ANR in com.android.bluetooth Line 99615: 09-24 09:19:14.938 1174 1558 E ActivityManager: ANR in com.android.bluetooth Line 108267: 09-24 09:22:38.697 1174 1558 E ActivityManager: ANR in com.android.bluetoo

11.5.8 Low memory issues

The amount of memory available/free is printed every 5 minutes in the logs for stability CRs.

For example:

\\heath\axiomdumps\prd\20191005\053b63dc-b1f8-4376-9c93-3ade3ea54f5b\Logs

axiom_meminfo-20191005-102735.log

Last part of the logfile name is the timestamp, which corresponds to timestamp in the logcat that is, close to following print in logcat:

10-05 10:27:35.986 1069 12460 W ActivityManager: Scheduling restart of crashed service com.qualcomm.qti.callenhancement/.CallEnhancementService in 30934 ms

Axiom meminfo file captures memory information as follows:

```
MemTotal:        5347372 kB
MemFree:         1435568 kB
MemAvailable:    1827780 kB
Buffers:            1972 kB
''''
```

The following prints are just debug/info prints and do not point to real issue with memory:

```
 10-07 01:53:01.888   954   954 D lowmemorykiller: Zone Normal: free:180407
high:16643 cma:72321 reserve:(0 4096 0) anon:(31726 31586) file:(6960 5301)

10-07 01:53:01.888   954   954 D lowmemorykiller: Zone Movable: free:119914
high:1838 cma:0 reserve:(0 0 0) anon:(0 8366) file:(0 0)

10-07 01:53:01.889   954   954 I lowmemorykiller: Ignoring pressure since per-
zone watermarks ok
```

The following prints point to the low memory issue:

Logcat logs

https://opengrok.qualcomm.com/source/xref/LA.UM.9.12/system/core/lmkd/lmkd.c

```
1636     ALOGE("Kill '%s' (%d), uid %d, oom_adj %d to free
%ldkB", taskname, pid, uid, procp->oomadj,
1637           tasksize * page_k);
```

The preceding print indicates that some cleanup is performed by lowmemorykiller but even this does not mean that device is out of memory. When lowmemorykiller cannot clean up enough memory and device is dangerously low on memory, out of memory killer(oom-killer) kicks in. At this point, the apps behavior is unpredictable:

Kernel logs

https://opengrok.qualcomm.com/source/xref/LA.UM.9.12/kernel/msm-4.19/mm/oom_kill.c#524

```
524   pr_warn("%s invoked oom-killer: gfp_mask=%#x(%pGg), nodemask=%*pbl,
order=%d, oom_score_adj=%hd\n",
525     current->comm, oc->gfp_mask, &oc->gfp_mask,
526     nodemask_pr_args(oc->nodemask), oc->order,
```

12 Bluetooth bring-up

This chapter provides the information about Bluetooth bring up including key host changes.

12.1 WCN685x bring-up on SM8350

The following information is written based on SM8350 MTP default configuration. This information might change the configuration/folder structure according to the platform.

12.1.1 Bluetooth software architecture on Android

Qualcomm uses old/mature O-MR1 stack with Bluetooth application due to the following reasons:

- Disrupting the stability and interoperability of the Bluetooth stack due to significant code re-architecture changes from Google.
- Adapting Qualcomm value-add features on top of a new baseline within a short upgrade cycle.
- Discarding existing Qualcomm features when the same feature is available from Google.
- Not applying all known fixes on the new baseline.

For more details, see *Android Bluetooth Feature Specification* (80-Y7674-180).

12.1.2 Bluetooth power-on sequence

The following figure represents the Bluetooth power-on sequence for both host and controller. Once the power on sequence is completed properly, it is ready to initialize the Bluetooth host.

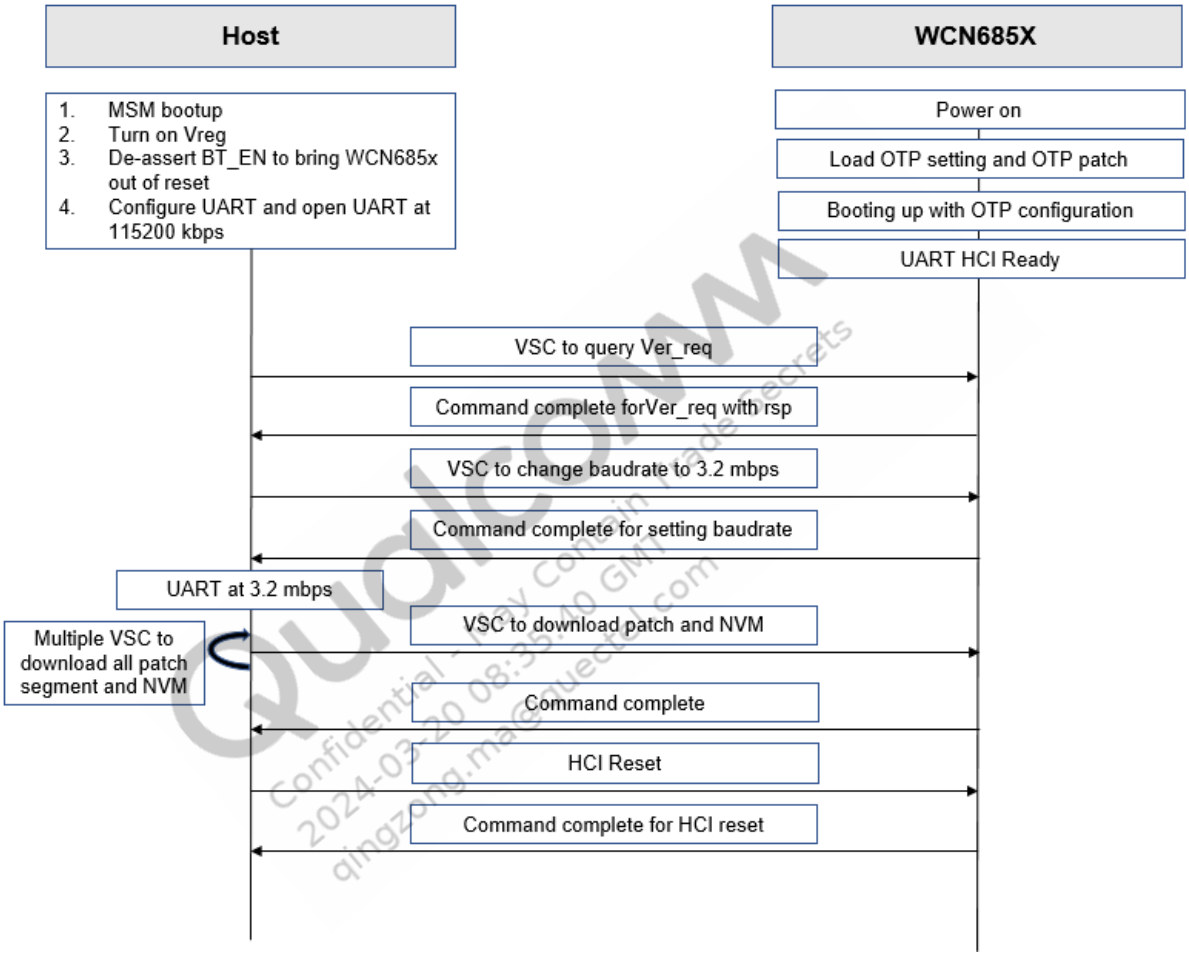


Figure 12-1 Bluetooth power-on sequence

12.1.3 WCN685x power up sequence and power grid diagram

The following figure shows the WCN685x power-up sequence and power grid between PMIC and WCN685x.

For the defined timing parameters, see *WCN6850 Dual-Band 2X2 MIMO DBS 802.11AX + Bluetooth Milan Data Sheet* (80-WL531-1).

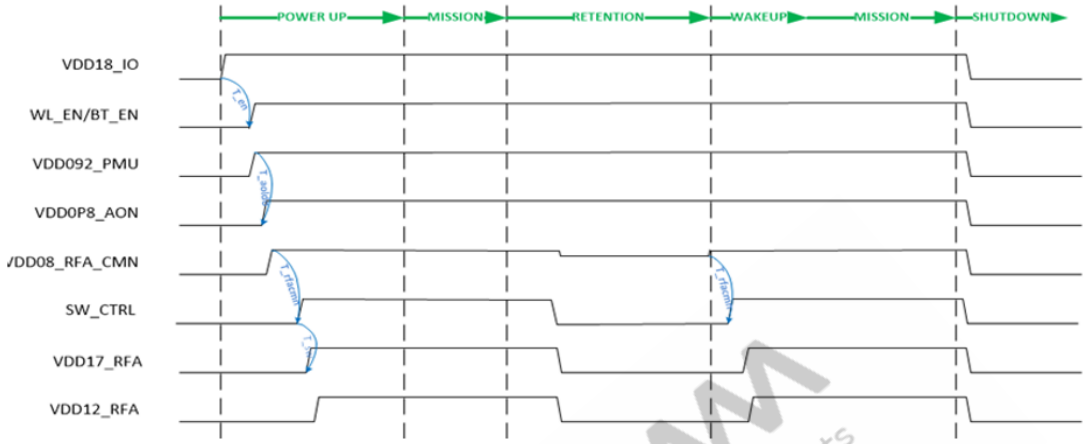


Figure 12-2 WCN685X power up sequence

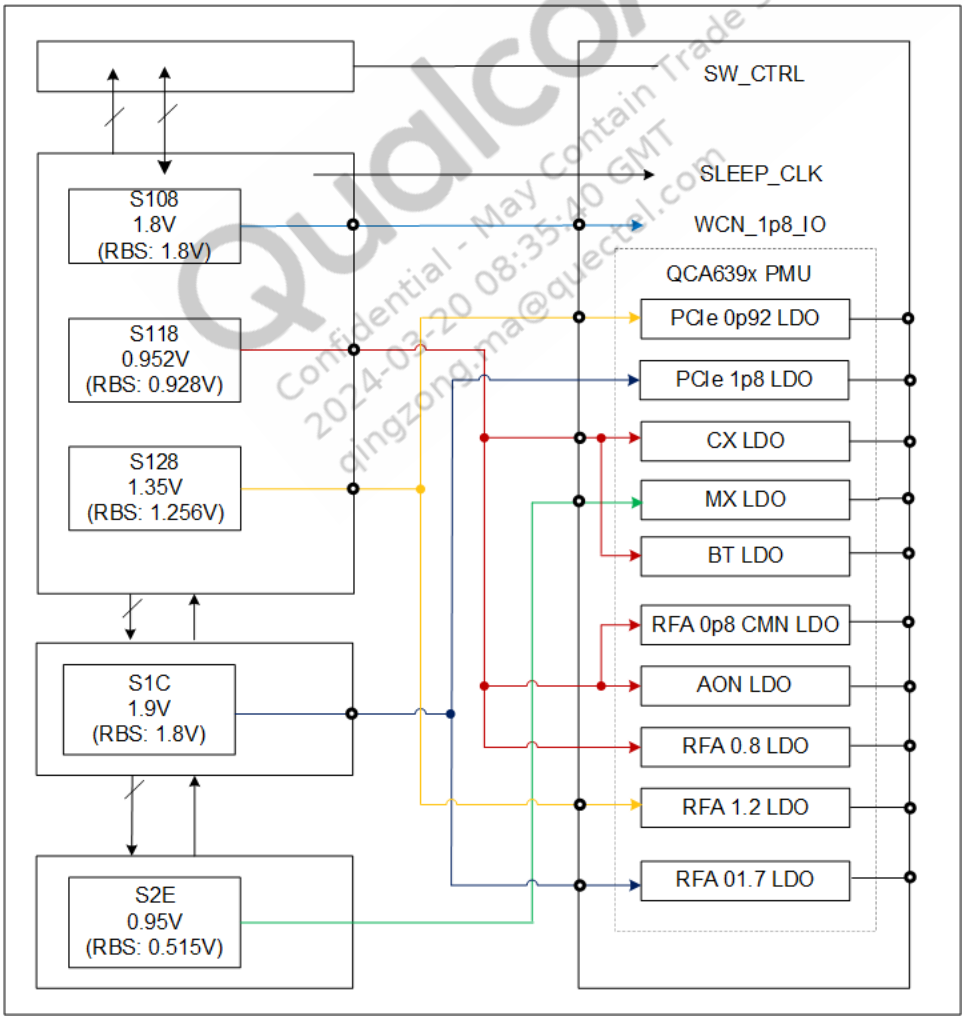


Figure 12-3 WCN685X power grid

12.1.4 Bluetooth source folder tree

Bluetooth stack	/vendor/qcom/opensource/commonsys/system/bt	Fluoride stack
Bluetooth apps	/vendor/qcom/opensource/commonsys/packages/apps/Bluetooth	Each profile
Bluetooth HAL	/vendor/qcom/proprietary/bluetooth/hidl_transport/bt/1.0/default	Bluetooth host transport driver
Kernel driver	/kernel/msm-[ver]/drivers/bluetooth	- Bluetooth power driver - SLIMbus driver
FTM application	/vendor/qcom/proprietary/ftm	- Ftmdaemon - wdsdaemon
Device tree	/vendor/qcom/proprietary/devicetree/qcom/	- Bluetooth power configuration - GPIO configuration

12.1.5 Key host changes to enable WCN685x

This section describes the required host changes to enable WCN685x.

Bluetooth power driver changes

This is name change in the kernel driver for the new chip.

Changes in btpower.c

```
static const struct of_device_id bt_power_match_table[] = {
    {    .compatible = "qca,ar3002" },
    {    .compatible = "qca,qca6174" },
    {    .compatible = "qca,wcn3990" },
    {    .compatible = "qca,qca6390" },
    {        .compatible = "qca,qca6490" },
    {}
};
```

Device tree (DTSI) changes

Power configuration for Bluetooth is mapped with power grid.

Changes in lahaina.dtsi

```
bluetooth: bt_qca6490 {
        compatible = "qcom,qca6490";
        pinctrl-names = "default";
        pinctrl-0 = <&bt_en_sleep>;
        qcom,bt-reset-gpio = <&tlmm 65 0>; /* BT_EN */
        qcom,bt-vdd-aon-supply = <&S11B>;
        qcom,bt-vdd-dig-supply = <&S11B>;
        qcom,bt-vdd-rfa1-supply = <&S1C>;
        qcom,bt-vdd-rfa2-supply = <&S12B>;
```

```
        qcom,bt-vdd-aon-config = <950000 950000 0 1>;
        qcom,bt-vdd-dig-config = <950000 950000 0 1>;
        qcom,bt-vdd-rfa1-config = <1880000 1880000 0 1>;
        qcom,bt-vdd-rfa2-config = <1350000 1350000 0 1>;
};
```

GPIO for BT_EN in lahaina-pinctrl.dtsi

```
bt_en_sleep: bt_en_sleep {
        mux {
            pins = "gpio65";
            function = "gpio";
        };
```

WCN685x logical address for SLIMbus slave change

One major change from a previous solution is the SLIMbus slave address. It is modified for WCN685x.

Changes in lahaina.dtsi

```
        /* Slimbus Slave DT for QCA6490 */
        btfmslim_codec: qca6490 {
            compatible = "qcom,btfmslim_slave";
            elemental-addr = [00 01 21 02 17 02];
            qcom,btfm-slim-ifd = "btfmslim_slave_ifd";
            qcom,btfm-slim-ifd-elemental-addr = [00 00 21 02 17 02];        };
```

WCN685x SoC ID to transport driver change

Transport driver keeps mapping the proper rampatch and NVM files based on SoC ID.

Changes in patch_dl_manager.cpp

```
void PatchDLManager::LoadPatchMaptable() {
…
  PatchPathInfoMap_.insert(std::make_pair<uint64_t,
PatchPathManager>(GENOA_VER_1_0,
                           PatchPathManager("GEN1_0", "gnbtfw10.tlv",
"gnnv10.bin")));
  PatchPathInfoMap_.insert(std::make_pair<uint64_t,
PatchPathManager>(GENOA_VER_2_0,
                           PatchPathManager("GEN2_0", "gnbtfw20.tlv",
"gnnv20.bin")));
  PatchPathInfoMap_.insert(std::make_pair<uint64_t,
PatchPathManager>(GENOA_VER_2_0_0_2,
                           PatchPathManager("GEN2_0", "gnbtfw20.tlv",
"gnnv20.bin")));
  PatchPathInfoMap_.insert(std::make_pair<uint64_t,
PatchPathManager>(HSP_VER_1_0,
                           PatchPathManager("HSP1_0", "hpbtfw10.tlv",
"hpnv10.bin")));
  PatchPathInfoMap_.insert(std::make_pair<uint64_t,
```

```
PatchPathManager>(HSP_VER_2_0,
                          PatchPathManager("HSP2_0", "hpbtfw20.tlv",
"hpnv20.bin")));
}
```

Changes in patch_dl_manager.h

```
enum {
  PROD_ID_ROME = 0x08,
  PROD_ID_CHEROKEE = 0x0A,
  PROD_ID_COMANCHE = 0x0F,
  PROD_ID_HASTINGS = 0x10,
  PROD_ID_GENOA = 0x12,
  PROD_ID_HSP = 0x13,
  PROD_ID_APACHE = PROD_ID_CHEROKEE
};

enum {
  HSP_BUILD_VER_0100 = 0x0100,
  HSP_BUILD_VER_0200 = 0x0200,
};

enum {
  QCA_HSP_SOC_ID_0100 = 0x400C0100,
  QCA_HSP_SOC_ID_0200 = 0x400C0200,
};

enum {
  HSP_VER_1_0   = QCA_BT_VER(QCA_HSP_SOC_ID_0100, PROD_ID_HSP,
HSP_BUILD_VER_0100),
  HSP_VER_2_0   = QCA_BT_VER(QCA_HSP_SOC_ID_0200, PROD_ID_HSP,
HSP_BUILD_VER_0200),
};
```