Qualcomm Linux Debug Guide - Addendum

80-70018-12A AB

April 10, 2025

Contents

1 Debug overview

This addendum provides supplementary information about the debugging features and tools for licensed developers with authorized access to Qualcomm® Linux® software.

Read this addendum along with the Qualcomm Linux Debug Guide.

Note: See Hardware SoCs that Qualcomm Linux supports.

2 Debug workflow

Identify the subsystem where the issue occurred first, as many subsystems are present on the Qualcomm SoC. This helps you debug the relevant subsystem.

The application processor as the primary subsystem detects when other subsystems crash. For example, if the aDSP subsystem crashes, the kernel log captures the subsystem restart (SSR) crash error log. Therefore, to identify the subsystems that need debugging, first verify the kernel debug messages.

2.1 Identify system issues from Qualcomm TEE logs

This section describes the sample logs that indicate the different types of errors in the Qualcomm® Trusted Execution Environment (TEE) diag logs. If the Qualcomm TEE logs show no errors, it indicates a secure watchdog bite issue. For more information about watchdog issues, see Hardware reset.

To extract the Qualcomm TEE diag logs, use the Qualcomm® Crash Analysis Portal (QCAP). For more information about QCAP, see Parse RAM dumps using QCAP.

To know more about debugging issues in Qualcomm TEE, see Qualcomm Linux Security Guide.

Nonsecure watchdog bite

Nonsecure code running on the application processor can get stuck and stop petting the nonsecure watchdog. Such a condition causes a nonsecure watchdog bite. The following log appears in the Qualcomm TEE diag when the application processor can't service the bark ISR, immediately after a watchdog bite.

```
Fatal Error: NON_SECURE_WDT
Encountered fatal FIQ error, CPU: 0, FIQ: 33
TZBSP_EC_MEM_DUMP_TRIGGER_S_WDOG_FROM_S_WORLD
```

AHB timeout error

The following sample log indicates the AHB timeout error:

```
ABT SNOC_1 ID: 0x0000e002
ABT SNOC_1 ADDR0: 0x15074000
ABT SNOC_1 ADDR1: 0x00000000
ABT SNOC_1 HREADY: 0xfffffffe
ABT SNOC_1 Slaves: 1
Fatal Error: AHB_TIMEOUT
```

xPU error

The following sample log indicates the xPU error:

```
xpu: ISR begin
XPU ERROR: Non Sec!!
XPU INTR 0:1 >> 00000000:00000800
xpu:>>> [1] XPU error dump, XPU id 3 (IPA_0_GSI_TOP)<<< xpu: uErrorFlags: 00000002
xpu: HAL_XPU2_ERROR_F_CLIENT_PORT
uBusFlags: 00000034
xpu: HAL_XPU2_BUS_F_ASHARED xpu: HAL_XPU2_BUS_F_APRIV xpu: HAL_XPU2_
BUS_F_APROTNS
xpu: uPhysicalAddressLower: 00024010 Upper:00000000 xpu: uMasterId: 00000000,
uAVMID : 00000000
xpu: uATID : 00000000, uABID : 00000000 xpu: uAPID : 00000000, uALen : 00000000
xpu: uASize : 00000000, uAPReqPriority : 00000000 xpu: uAMemType: 00000002
Fatal Error: XPU_VIOLATION
```

SMMU error

The following sample log indicates the SMMU error:

```
SMMU GLOBAL TCU NON-SEC FAULT: bit mask=0x00002000
SMMU:>> APPS_TCU NonSec Global Fault:
NSGFSR=0x00000002
NSGFAR=0x00000000917d9000
NSGFSYNR0=0x00000000
NSGFSYNR1=0x07230723
NSGFSYNR2=0x00000000
NSCR0=0x002f1c06
***** ENHANCED SMMU DEBUG *****
faultingSmmuBase = 0x 15000000
fsynr0 = 0x00000000 – faultingStage1CB = 0x 15040000
faultingStage2CB = 0x 15040000
>> smmu_tlb_dump_log : Starting with dump base_addr = 0x17ff05000 – Device = 0x0
<< smmu_tlb_dump_log : Returning : Wrote 0 bytes!
Fatal Error: SMMU_FAULT
```

NoC error

The system experiences many types of NoC errors, such as SNoC, CNoC, MEMNOC, and AggNOC. To identify the type of NoC error, verify the logs in the Qualcomm TEE diag.

The following sample log indicates the NoC error:

```
[79130a319b](NOC_FAULT_NAME_SBMS NOC_NAME: LPASS_AG_NOC SBM:0,
FAULTINSTATUS0_LOW = 0x80, FAULTINSTATUS0_HIGH = 0x0, FAULTINSTATUS1_LOW
= 0x0, FAULTINSTATUS1_HIGH = 0x0)
[7914810459](NOC_FAULT_NAME_SBMS NOC_NAME: GEM_NOC SBM:0,
FAULTINSTATUS0_LOW = 0x200, FAULTINSTATUS0_HIGH = 0x0, FAULTINSTATUS1_
LOW = 0x0, FAULTINSTATUS1_HIGH = 0x0)
[791482ce26](TZBSP_KRNL_FATAL_ERR_TYPE Fatal Error type:TZBSP_ERR_FATAL_
NOC_ERROR)
```

3 Debug Linux kernel space issues

It's recommended to use the `debug` build for debugging kernel issues. For more information about how to generate the `debug` build, see Qualcomm Linux Yocto Guide.

You can generate kernel logs using the `dmesg` command.

For information about kernel source configuration files, see Qualcomm Linux Kernel Guide.

3.1 Subsystem dump

To enable and capture the coredump of a subsystem, do the following:

1. Enable the `CONFIG_COREDUMP` kernel configuration option.

   `CONFIG_COREDUMP=y`

2. In the `/etc/initscripts/post_boot.sh` file, add the following command:

   ```
   echo enabled > /sys/kernel/debug/remoteproc/remoteprocN/coredump
   ```

   Here, N represents the subsystem for which you want to enable the coredump.

3. Ensure that the `subsystem_restart` binary is running.

Enabling coredump ensures that whenever the subsystem crashes, the device collects a coredump in the `/var/spool/crash` directory. The coredump filename is in the <rproc_base_addr>.remoteproc_<timestamp>.elf format.

For example, from the following logs, you can determine that the name of the coredump file is `3000000.remoteproc_xxyyyzzz.elf`. Here, the base address 0x3000000 belongs to the aDSP subsystem, according to `qcm6490.dtsi` (node: remoteproc_adsp: remoteproc@3000000).

```
0x000000000A27652C |   5198.790423:   qcom_q6v5_pas 3000000.
remoteproc: fatal error received: err_inject_crash.c:413:Crash
injected via Diag
0x000000000A276689 |   5198.801061:   remoteproc remoteproc2: crash
detected in 3000000.remoteproc: type fatal error
```

```
0x000000000A2767A1 |   5198.809602:   remoteproc remoteproc2:
handling crash #1 in 3000000.remoteproc
0x000000000A27688E |   5198.816837:   remoteproc remoteproc2:
recovering 3000000.remoteproc
0x000000000A276971 |   5198.823784:   qcom_q6v5_pas 8a00000.
remoteproc: subsystem event rejected
```

You can parse the subsystem coredump using the Qualcomm Crash Analysis Portal (QCAP).

3.2 Parse RAM dumps using QCAP

The QCAP is a powerful tool to parse logs from all subsystems and determine which subsystem crashed first. It's recommended to use QCAP for the initial triage.

The QCAP tool provides a consolidated system-level crash analysis report on a simple HTML interface. This report includes logs of various subsystems such as kernel logs (`dmesg`) of the application processor, Qualcomm TEE diag logs, and diag logs of other subsystems.

You can download the QCAP from Qualcomm Package Manager. For more information, see QCAP Systems Overview.

4 Debug non-HLOS issues

The following sections describe various tools, software, and debugging methods available to capture logs and debug non-HLOS subsystems.

4.1 Debug aDSP

You can use Diag or Qualcomm extensible diagnostic monitor (QXDM) Professional™ tool to debug aDSP. For more information, see Diag.

4.2 Debug Always-On-Processor (AOP)

To debug AOP-related issues, you can use the following tools on a Windows host computer:

- TRACE32 for on-device debugging
- Hansei parser to debug using RAM dump

Debug AOP using TRACE32

The scripts required to debug the AOP using TRACE32 are available in the AOP build at the `<aop build>\aop_proc\core\bsp\aop\scripts\` directory.

Using these scripts, you can perform debugging tasks such as loading a dump, and verifying the faults in a dump from the device. To debug AOP using TRACE32, do the following on a Windows host computer:

1. To save an AOP dump, in a live debug setup, stop the process and run the following command in the TRACE32 window:

   ```
   do <aop build>\aop_proc\core\bsp\aop\scripts\aop_dump.cmm
   ```

   The AOP dump is also part of the RAM dump collected after a system crash.

2. To load the AOP dump into a TRACE32 simulator, run the following command in the TRACE32 window:

   ```
   do <aop build>\aop_proc\core\bsp\aop\scripts\aop_load_dump.cmm
   <dump path>
   ```

3. To parse AOP uLog reports, use either of the following methods:

   - From the AOP TRACE32 live or TRACE32 simulator, run the following command:

     ```
     do <aop build>\aop_proc\core\power\ulog\scripts\aop_
     ulogdump.cmm <output path>
     ```

   - From the command prompt, run the following command:

     ```
     <aop build>\aop_proc\core\bsp\aop\scripts\aop_log_hfam.
     py -f "<path>\AOP External Log.ulog" -tbl <aop build>\
     aop_proc\core\api\debugtrace\tracer_event_tbl.h > AOP_
     ulog_parsed.txt
     ```

Debug AOP using Hansei parser

The AOP build includes the Hansei scripts, which are available in the `aop_proc\core\bsp\aop\scripts\hansei\` directory.

To debug AOP using the Hansei parser, do the following on a Windows host:

1. To install the Hansei parser tool, do the following:

    a. Download Python 2.7.x.

    Note: Qualcomm Linux supports any version of Python 2.7. Python 3.0 or later versions aren't supported.

    ---

    b. Install the pyelftools library that supports the Arm® compiler.

    ```
    python setup.py install
    ```

2. To parse the RAM dump, run the Hansei scripts from the AOP build.

    ```
    aop_proc\core\bsp\aop\scripts\hansei\hansei.py --elf <aop>. elf
    path> -o <output_directory_path>. -t <target_name> dumpfile
    <ramdump directory>
    ```

The script generates the following files that provide high-level debugging information:

File/Directory	Description
aop-summary.txt	This file provides the platform information and state of AOP firmware. For example, whether the AOP firmware was running or in a fatal scenario. If the fatal scenario is due to bus usage, hard, or memory management faults, the summary file provides the fault details.
BIN	This directory includes the following: • Required binaries such as `CODERAM.BIN`, `DATARAM.BIN`, and `MSGRAM*.BIN` for `backup.AOP` elf file • CMM scripts load binaries for further debugging
Requests_By_Master	This directory has a separate file for each subsystem that includes votes on each resource. For example, `AOP_drv6.txt`, `APPS_drv2.txt`.

File/Directory	Description
Requests_For_ Resource	This directory includes the following text files: • `arc_vt.txt`: Provides the list of votes for railway resources such as CX, MX, and the state of each resource at the time of dump collection • `bcm_vt.txt`: Provides the list of votes for clock resources such as DDR, SNoC, and the state of each clock at the time of dump collection • `vrm_vt.txt`: Provides the list of votes for PMIC resources such as s1a, s2a, l2a, and state of each SPMS/LDO at the time of dump collection • `cprf.txt`: Provides the CPRF control settings and voltage table for each rail • `rpmh_summary.txt`: Provides the busy or idle state of each RPMh resource manager
task_info.txt	This file includes the list of tasks running in the AOP firmware, their priority, wait events, wait signals, and state (suspended or not suspended).
cmd_db.txt	This file includes the list of resources that RPMh manages and their voting addresses.
sleep_stats.txt	This file includes AOP or AOSS Low-Power mode statistics such as the following: • `sleep_count`: Number of times that the mode was in the Low-Power mode (LPM) • Total accumulated duration that the system spent in the LPM: – `last_entered_at` – `last_exited_at` – `accumulated_duration` • Sleep parameters: – AOSS shutdown (AOSD): AOSS deep sleep – CX shutdown (CXSD): CX collapse

<table>
<tr><th>File/Directory</th><th>Description</th></tr>
<tr><td>PDC</td><td>This directory includes a separate file for each subsystem, which provides the power domain controller (PDC) state or configuration (CFG) details of the corresponding subsystem. For example: PDC_AOP.txt and PDC_APPS.txt. These files contain the following parameters:
• PDC_MODE_STATUS: This parameter indicates whether the subsystem is active (Pass through mode) or in sleep (Sequencer mode).
• ENABLE_PDC: This parameter indicates whether a PDC is enabled.
• TIMER_MATCH_VALUE_HI and TIMER_MATCH_VALUE_Lo: These parameters indicate the next wake-up time for the corresponding subsystem.
• IRQ_CFG, IRQ_ENABLE, and IRQ_STATUS: These PDC monitors wake up interrupts on behalf of the subsystem, when the subsystem interrupt controller isn't functional. Some of them are general-purpose interrupts that are visible to every PDC, whereas the others are specific to this particular PDC.
– IRQ_CFG: This PDC monitor configures the sensitivity of the interrupts. For example, level, edge, raising, or falling triggered.
– IRQ_ENABLE: This PDC monitor allows the interrupts that PDC can monitor and wakes up the subsystem.
– IRQ_STATUS: This PDC monitor indicates whether the interrupts occurred or not.

Note: The PDC directory isn't generated when the SDI fails to back up the RPMh binaries.</td></tr>
</table>

4.3 Diag services

Diagnostic (Diag) is a software feature that captures the diagnostic packets such as F3s, logs, and events from the different subsystems. The Diag also supports sending and receiving commands and responses between the device, saving logs on the device, and logging information through the QXDM Professional.

The Diag provides the following services:

Table : Diag services

Service	Description
Request/response service	This service allows clients to process inbound request packets from the diag service and return a response packet to the external device.
Debug message service	This service debugs messaging that's conditionally compiled into the code, based on the default build mask or on a custom build mask.
Event reporting service	This service reports events from the subsystems to indicate actions, such as changes in the state and configuration, related to the operating standards of the system.
Logging service	This service allows clients to send information to the external device when it's available, instead of sending it when the information is requested from the external device.

The Qualcomm Linux devices support the following Diag features:

- On-device logging
- Diag callback
- Diag socket logging

On-device logging

The on-device logging feature allows you to save diagnostic traffic to a memory device such as an SD card or an eMMC. On-device logging helps debug devices during field testing, when connecting the device to a workstation or laptop is difficult. Later, you can use host PC tools to process the logged item files.

The diag_mdlog is the application that allows the on-device logging feature. The diag_mdlog application uses circular logging, which allows logging to continue even when the device runs out of memory. When the device runs out of memory, the application deletes the oldest log file to create free space and continues logging. Whenever the application deletes a file to create space, it prints the log filename and size of the deleted log file (in kB). The deleted logs aren't recoverable.

Run diag_mdlog application

Use the following command to run the diag_mdlog application with the default parameters:

```
diag_mdlog -f <mask_file name> -o <output dir> -s <size in MB> -c
```

The `-s`, `-n`, and `-o` options manage the files created by the diag_mdlog application. For more information about `-s`, `-n`, and `-o` options, see Table : Options to use when running diag_mdlog application.

To run the diag_mdlog application in the background, use the `&` option. The diag_mdlog application creates a directory and files in the `Directory with date_time and file diag_logs_date_time.qmdl` format.

For example, `20240226_112226/diag_log_20240226_1122261708946546575.qmdl`

The following table lists the options that are available to use with the diag_mdlog application:

Table : Options to use when running diag_mdlog application

Option	Description
`-f`	Path and mask filename for MSMT and APQ targets.
`-m`	Path and mask filename for MDM targets.
`-l`	Name of the file that has the list of mask files.
`-o`	Name of the output directory.
`-s`	Maximum size (in MB) of log file. When the specified file size is reached, a new file is created.
`-w`	Waiting for directory.
`-n`	Maximum number of log files that can be created. • If the number of files isn't specified, the diag_mdlog application continues to create files until the storage space is full. • If the number of files is specified, the oldest file is deleted when the file limit is reached.
`-k`	Kill the existing instance of diag_mdlog.
`-c`	Send mask cleanup to modem at exit. Use this option while launching the diag_mdlog application and not while killing it.
`-d`	Disable console messages.
`-e`	Run using a wake-up source to keep the application processor on.
`-b`	Have peripherals buffer data and send data in nonreal-time.
`-h`	Usage help; prints a list of these arguments.
`-a`	Disable HDLC encoding. Use this option if you are using cfg2 file for non-HDLC mode.

Option	Description
-p	Peripheral: • 16 = SLPI • 15 = Others except SLPI
-r	Rename directory or filenames, according to the time when they were closed.
-t	Configure Peripheral Tx mode: • 0: Streaming mode • 1: Threshold mode • 2: Circular Buffering mode
-u	Guid-diagid mapping in qmdl2 header.
-x	Peripheral ID bitmask for the peripherals interested in Buffering mode.
-q	Use Qualcomm debug system (QDSS) mode.
-g	PD selection.
-i	Enable Accelerated data protocol logging (ADPL) mode.
-j	Select the Proc mask for logging.
-y	etr buffer size.

Methods to capture logs

You can use either of the following methods to capture logs:

- **From SSH**

  To enable on-device logging from the SSH, do the following:

  1. Use the `scp` command to push the mask configuration file (`.cfg`) to the device at the `/data` directory and set permissions on the mask configuration file.

     ```
     chmod 777 /data/Diag.cfg
     ```

  2. Create an output directory.

     ```
     mkdir /data/diag_logs
     ```

     ```
     chmod 777 /data/diag_logs
     ```

  3. Run the logging application.

     ```
     /usr/bin/diag_mdlog -f /data/Diag.cfg -o /data/diag_
     logs/test1 -c
     ```

```
/usr/bin/diag_mdlog -f /data/Diag.cfg -o /data/diag_
logs/test1 -c -b
```

The application creates the `/data/diag_logs/test1` directory and generate log files in this directory.

- **From a user space application**

  To start the diag_mdlog application from another application, make a system call and invoke the diag_mdlog application by running the following command:

  ```
  system("/usr/bin/diag_mdlog & ");
  ```

Note: To save logs on an SD card, the application must have root access and other read/write access to the SD card.

Diag_mdlog output

The diag_mdlog application generates the output file in `.qmdl` or `.qmdl2` format. You can open the `.qmdl` files in the QXDM Professional Tool.

The following table describes the qmdl2 header format:

Table : qmdl 2 header format

Field	Length (bytes)	Description

`HeaderLength`	4	Number of bytes reserved for the header. Data in the QMDL2; starts after the header.
		`v1 header length = 4+1+1+4+(16* GUIDEntryCount)`
		`v2 header length = 4+1+1+4+(16* GUIDEntryCount)+4+( 47*DiagIDEntryCount)`
		`v3 header length = 4+1+1+4+(16* GUIDEntryCount)+4+( 47@DiagIDEntryCount)+(1+( Keycount*keyinfoSize))`

Field	Length (bytes)	Description
`Version`	1	Version value of the header. • 1: Includes GUID entry information • 2: Indicates DIAG ID <—>GUID mapping is present in the header, following the version 1 GUID entry information. • 3: Indicates that the wrapped key information is present in the header, following the version 2 DIAG ID <—>GUID mapping.
`HDLCDataType`	1	Type of encoding for data in the buffer. • 0: Indicates that HDLC encoding is removed • 1: Indicates that HDLC encoding is enabled • 2: Indicates that a file is a QDSS file
`GuidListEntryCount`	4	Number of GUIDs available to read.

`(DIAG ID, DIAG ID string, GUID) [DiagIDEntryCount]`	47*(DiagIDEntryCount)	An array of sets of Diag ID (1 byte), Diag ID string (30 bytes), and GUID (16 bytes).
		`Diag ID1, Diag ID1 string, GUID1,`
		`DIAG ID2, Diag ID2 string, GUID2 ..`

`KeyCount`	1	Number of keys to read.
`KeyInfo (Version, Classifier, PublicType, Reserved, PublicLen, WrappedLen, PublicKey, WrappedKey) [KeyCount]`	(1 + 1 + 1 + 1 + 2 + 2 + PublicLen + WrappedLen) * KeyCount	An array of key information. The size of individual key entries may vary and depends on the `PublicLen` and `WrappedLen` fields. The `Version` field may also impact the size and contents of the entries. The current information listed here reflects `version 1` for the `KeyInfo` field.

The following figures show the difference between the QMDL and QMDL2 file formats:

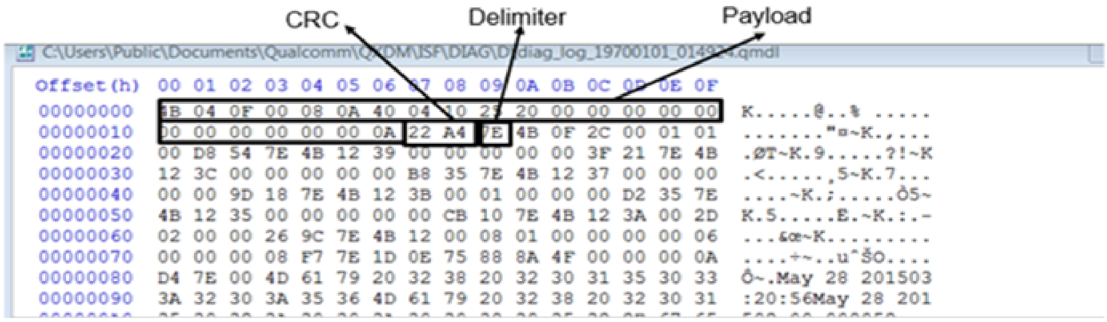


Figure : QMDL file format

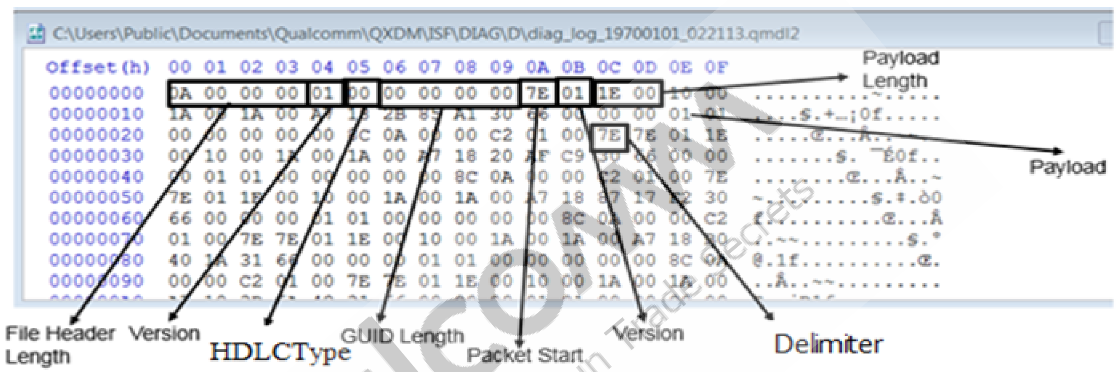


Figure : QMDL2 file format

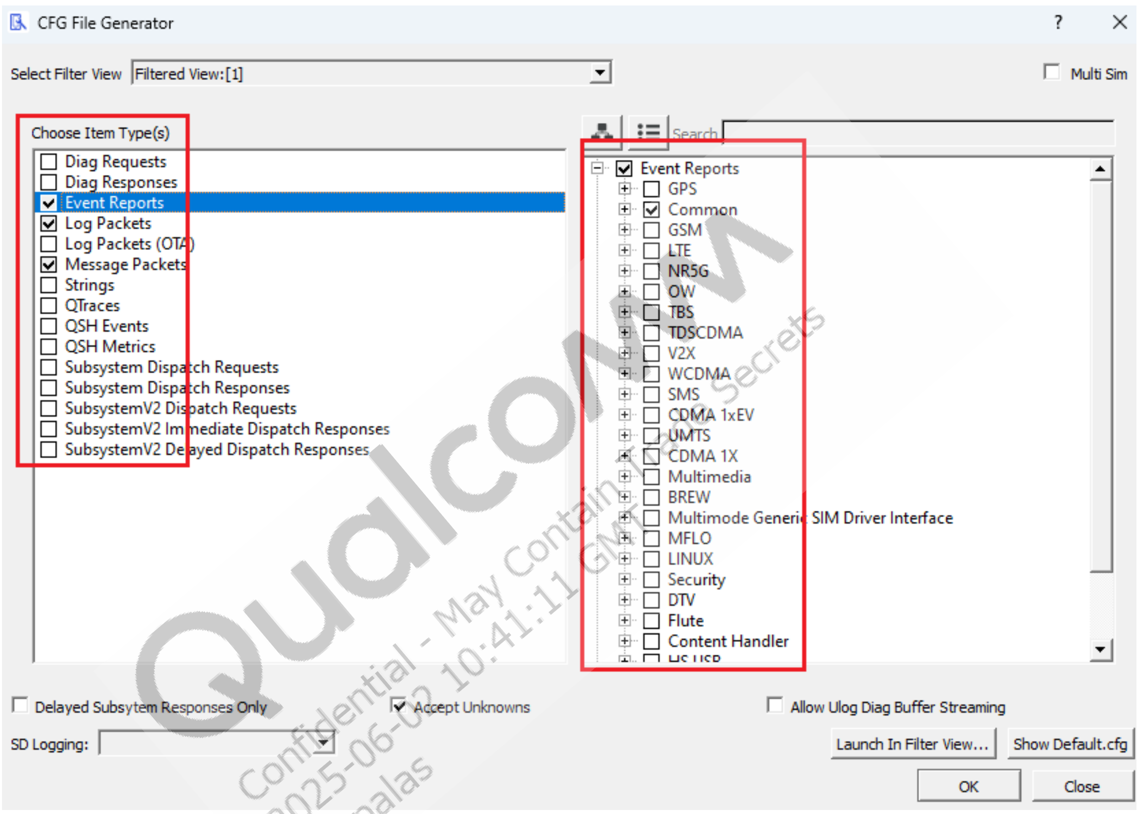
Diag masks

To enable on-device logging for the required subsystem, you must generate a file containing information of all masks from the QXDM Professional, and place this file in the `diag_logs` directory in the output directory. This section describes how to generate, change, read, and use a mask configuration file for on-device logging.

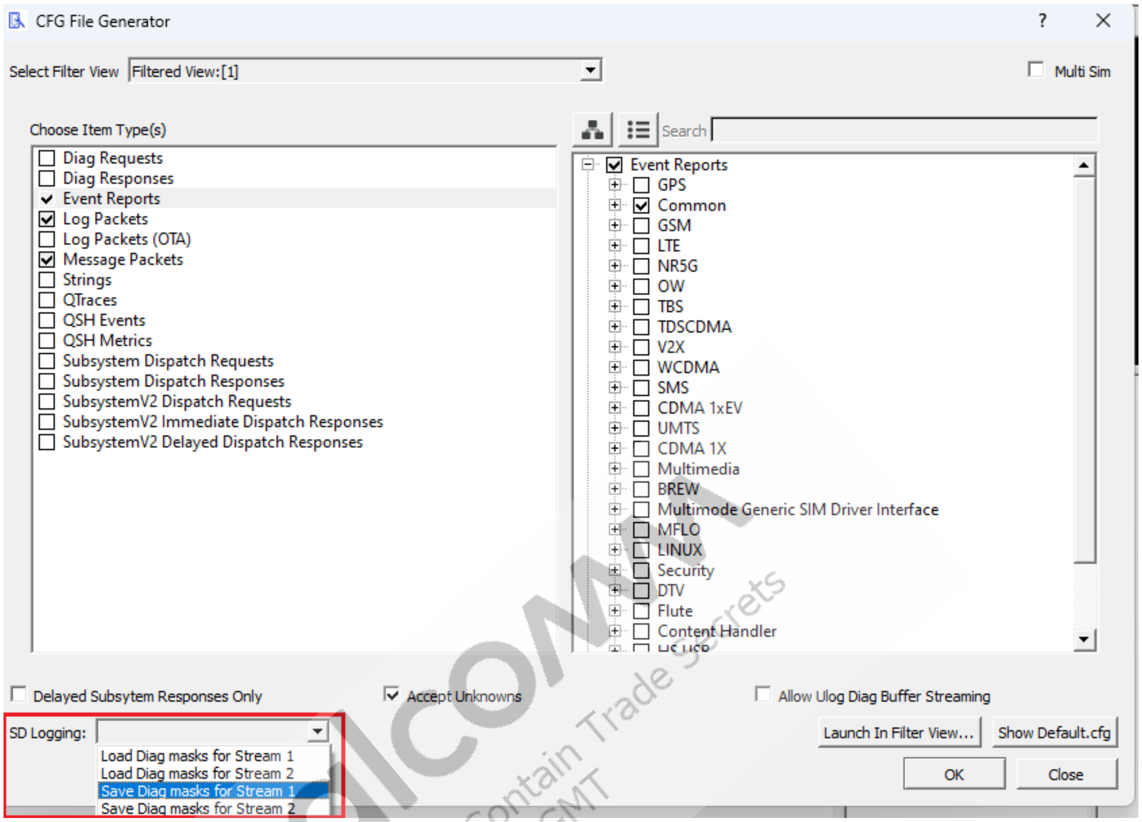
Generate mask configuration file

To generate a mask configuration file (`.cfg`/`.cfg2`), do the following on your Windows host computer:

1. Start the QXDM Professional.
2. Go to **Tools** > **CFG File Generator** and select the relevant masks.



3. Select the appropriate value in the **SD Logging** list box.
   - To generate the `.cfg` configuration file for the HDLC mode, select **Save Diag masks for Stream 1**.
   - To generate the `.cfg2` configuration file for the Non-HDLC mode, select **Save Diag masks for Stream 2**.

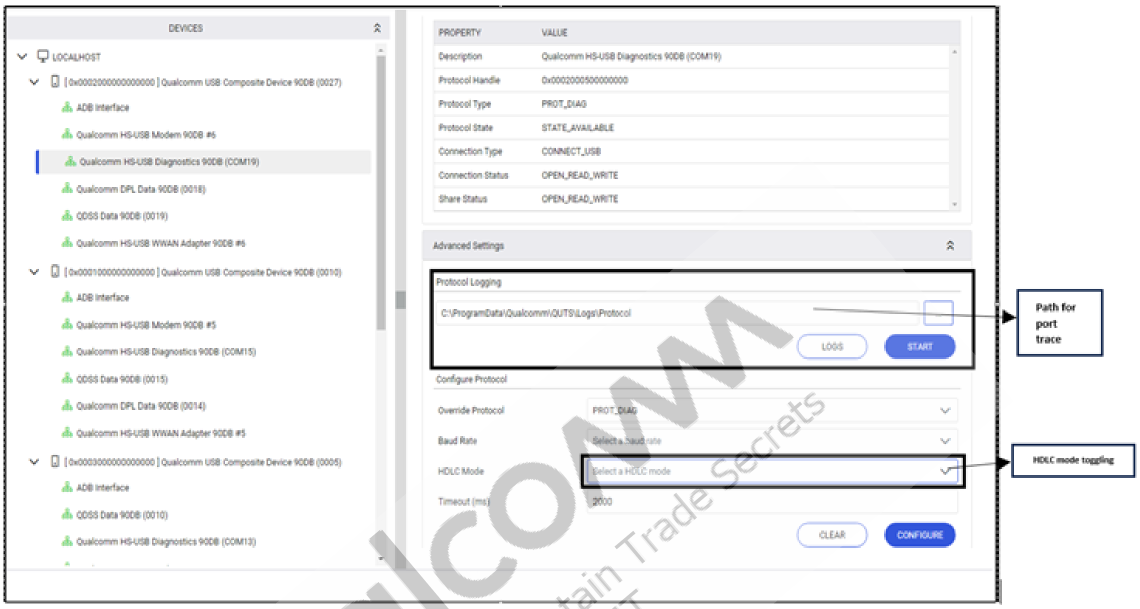


4. Name the file as `Diag.cfg`, and save the file at an appropriate location.

Change mask configuration file

To insert a command in the mask configuration file (`.cfg`/`.cfg2`), update the file as follows:

1. Open the Qualcomm Unified Tools Service (QUTS) application.



   a. In the left panel, select port.

   b. In the right panel, under the Protocol Logging section, specify the path to save the log files.

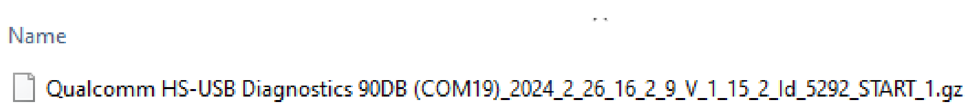
   c. To collect the port trace logging, select **START**.

   d. After the debugging scenario is complete, to stop trace logging, select **STOP**.

   e. To insert a command in the `.cfg` file, send the command through the QXDM Professional.

   f. Disable the port trace logging.

   After stopping the port trace logging, the system generates the `.gz` file with the port name at `C:\ProgramData\Qualcomm\QUTS\Logs\Protocol`. The following is an example of a `.gz` file.



2. Open the generated `.gz` file and search for the command that you inserted using the QXDM Professional.

You must find both the command and the response in the `.gz` file.

3. Copy only the command and append it at the end in the `.cfg/.cfg2` file.

   You can open the `.cfg` file in the hexadecimal editor.

   - **For .cfg file**

     a. To insert a cDSP stress test command for F3s in the `.cfg` file, send the following command through QXDM Professional:

     ```
     send_data 75 18 01 36 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0
     0 0 0 0 0
     ```

     b. In the port trace log file (`.gz` file), search for the hexadecimal representation of the command that you have sent through the QXDM Professional. You must find the following:

     ```
     4B 12 01 24 01 00 00 00 01 01 01 00 00 01 00 00 00
     00 00 00 00 00 00 00 37 B5 7E
     ```

     HDLC format: Command + CRC + Delimiter

     For more information on `.cfg` file formats, see Read HDLC encoded configuration file.

     c. Copy the hexadecimal representation of the command. Append it at the end in the `.cfg` file and save the file.

   - **For .cfg2 file**

     a. To insert a cDSP stress test command for F3s in the `.cfg` file, send the following command through QXDM Professional:

     ```
     send_data 75 18 01 36 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0
     0 0 0 0 0
     ```

     b. In the port trace log file (`.gz` file), search for the hexadecimal representation of the command that you have sent through the QXDM Professional. You must find the following:

     ```
     7E 01 18 00 4B 12 01 24 01 00 00 00 01 01 01 00 00
     01 00 00 00 00 00 00 00 00 00 00 7E
     ```

     Non-HDLC format: Delimiter + Version + Length of payload (To be read in little endian) + Payload (Command) + Delimiter

     For more information about `.cfg` file formats, see Read non-HDLC encoded configuration file.

c. Copy the hexadecimal representation of the command. Append it at the end on the `.cfg2` configuration file and save the file.

Use the mask configuration file

To use the mask configuration file (`.cfg` or `.cfg2`), do the following:

1. Ensure that the diag_mdlog application has created the `/sdcard/diag_logs/` directory, and you have the appropriate file permissions.
   - To avoid any file permission-related errors, run the diag_mdlog application and kill this instance using the `diag_mdlog -k` command.
   - If the `/sdcard/diag_logs/` directory isn't available, create the directory and set the permissions:

     ```
     chmod 777 sdcard/diag_logs/
     ```

     ```
     chmod 777 sdcard
     ```

2. Using the `scp` command push the generated configuration file to the device at `/sdcard/diag_logs`.
3. Run the diag_mdlog application:

   ```
   diag_mdlog
   ```

   The diag_mdlog application uses the configuration file that's pushed to the device. You can push the configuration file to a different location using the `-f` option. For example,

   ```
   diag_mdlog -f /sdcard/new_logs/Diag.cfg
   ```

Read HDLC encoded configuration file

Open the mask configuration file in a hexadecimal editor.

```
Field            Length (in bits)   Description
-------          ----------------   -----------
Information      Variable           ICD Packet or Message
Frame Check      16                 CRC-CCITT standard 16-bit CRC
Ending Flag      8                  Ending character "0x7E"
```

The following is a part of the hexadecimal dump of a `DIAG.CFG` file saved from the **QXDM Filtered View dialog**.

```
1D 1C 3B 7E 00 78 F0 7E 7C 93 49 7E 1C 95 2A 7E 0C 14 3A 7E 63 E5 A1
7E 4B 0F 00 00 BB 60 7E 4B 09 00 00 62 B6 7E 4B 08 00 00 BE EC 7E 4B
08 01 00 66 F5 7E 4B 04 00 00 1D 49 7E 4B 04 0F 00 D5 CA 7E
```

In the HDLC encoded configuration file, `7e` delimiters seperate commands. The two bytes immediately before `7e` represents the CRC. The rest of the bytes between the two `7e` represent the data.

```
4b 0f 0a 00 00 // Info
bb 60          // CRC
7e             // DELIMITER
```

Using the end flags to split the messages, you can decode the messages according to the interface control document (ICD):

```
1d 1c 3b 7e  // 0x1D == Time Stamp Request
00 78 f0 7e  // 0x00 == Version Request
7c 93 49 7e  // 0x7C == Extended Build ID Request
1c 95 2a 7e  // 0x1C == DIAG Version Request
0c 14 3a 7e  // 0x0c == Status Request
63 e5 a1 7e  // 0x6C == Phone State Request
4b 0f 1a 00 00 bb 60 7e // 0x4B 0x0F 0x001A == Call Manager Subsystem
Sys Select (80-V1294-7)
4B 09 00 00 62 B6 7E // 0x4B 0x09 0x0000 == UMTS Subsystem Version
Request (80-V2708-1
4b 08 00 00 be ec 7e    // 0x4B 0x08 0x0000 == GSM Subsystem Version
Request (80-V5295-1)
4b 08 01 00 66 f5 7e // 0x4B 0x0F 0x0001 == GSM Subsystem Status
Request (80-V5295-1)
4b 04 00 00 1d 49 7e    // 0x4B 0x04 0x0000 == WCDMA Subsystem
Version Request (80-V2708-1)
4b 04 0f 00 d5 ca 7e    // 0x4B 0x04 0x000F == WCDMA Subsystem
Additional Status Request (80-V2708-1)
```

The subsystem `CMD_CODE(75)` is defined in the subsystem dispatch (75/0x4B).

```
4b    // CMD_CODE 75 DIAG command identifier
0f    // Subsystem ID for Call Manager commands, refer to 80-V1297-7
```

After identifying the subsystem ID, use the subsystem ICD to decode the subsystem message.

```
1a 00 // Subsystem CMD_CODE 0x001a (26d) for System Selection
Preference of Current Subscription
   00    // 0 = No active subscription ID
```

For the HDLC-encoded format (`.cfg` file), the format of the file is as follows:

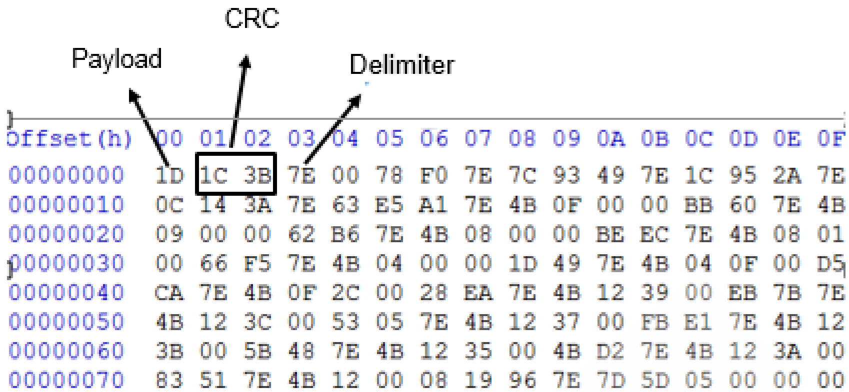


Figure : HDLC encoded format

Read non-HDLC encoded configuration file

Open the `.cfg2` configuration file in a hexadecimal editor.

```
Field                Length (in bits)   Description
-------              ----------------   -----------
Start              8           This is the start of packet, 0x7E
Version      8          Version
Payload Length  16         Payload length
Payload      Variable      This is the actual data
Packet End          8             Ending character "0x7E"
```

For a non-HDLC encoded format, the format of the `.cfg2` configuration file is as follows:

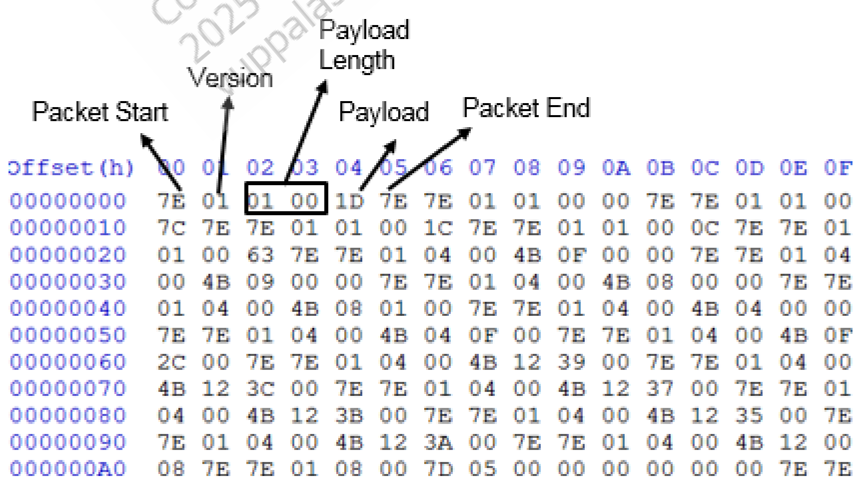


Figure : Non-HDLC encoded format

Diag callback

The Diag provides APIs that allow the client application to receive the logs, events, and F3 messages originating from different subsystems directly. The client can register for a callback to receive Diag packets and can implement their logic in the callback.

The following APIs are available to implement the callback:

API	Parameters	Description
`void diag_register_callback(int (*cb_func_ptr)(unsigned char *ptr, int len, void *context_data), void *context_data);`	• `cb_func_ptr`: Callback function • `ptr`: Incoming Diag data • `len`: Length of the incoming data • `context_data`: Data supplied by the client while registering the second parameter	Registers a client with the Diag to call back traffic from the current primary processor
`void diag_register_remote_callback(int (*client_rmt_cb_func_ptr)(unsigned char *ptr, int len, void *context_data), int proc, void *context_data);`	• `client_rmt_cb_func_ptr`: The callback function • `ptr`: Incoming Diag data • `len`: Length of the incoming data • `context_data`: Data supplied by the client while registering the third parameter • `proc`: The remote processor that the client is interested in	Registers a client with the Diag to receive callback traffic from a remote processor

Use Diag callback

To use the Diag callback, the client application must do the following:

1. Open a port to Diag with the following command, which gives a handle to the Diag driver.

```
Diag_LSM_Init()
```

2. Register callbacks with the `diag_register_callback` and `diag_register_remote_callback` APIs.
3. To receive data, switch the logging mode using the following command:

```
diag_switch_logging(CALLBACK_MODE, NULL);
```

   This turns off the Diag port that logs to the QXDM Professional.
4. Read the mask file using the `diag_read_mask_file();` command.
   This command updates the mask information from the config file (`Diag.cfg`) to the Diag driver.
5. Deregister from the Diag using the following command:

```
Diag_LSM_DeInit()
```

6. When done with Callback mode, switch to the USB mode by calling the following API:

```
diag_switch_logging(USB_MODE, NULL);
```

Diag logging: Libdiag API usage by clients

API	Parameters	Description
`int diag_lsm_comm_open(void);`	None	Opens a Unix socket fd and connects to that socket address
`int diag_lsm_comm_ioctl(int fd, unsigned long request, void *buf, unsigned int len)`	• `fd`: Socket fd • `request`: Request id • `buf`: Buffer to hold the data • `len`: Length of the buffer passed	Performs input/output control operations • 0: Success • Negative value: Failure
`int diag_lsm_comm_write(int fd, unsigned char buf[], int bytes, int flags);`	• `fd`: Socket fd • `buf`: Buffer to be sent • `bytes`: Length of the data to be passed • `flags`: Set `MSG_DONTWAIT` or 0	Sends DIAG commands and control masks to peripherals or subsystems

Diag socket logging

The socket logging feature provides diagnostic traffic over Ethernet and Wi-Fi. If the device has Ethernet or Wi-Fi enabled, it connects to the QUTS and QXDM Professional tools to fetch the Diag packets, similar to other logging methods. Some of the factory testing relies on the wireless scenario. The PC tools such as QXDM Professional can also work with the Diag module in this condition. The diag_socket_log is the application that allows the Diag socket logging feature.

The following figure shows the architecture of the Diag socket logging feature:

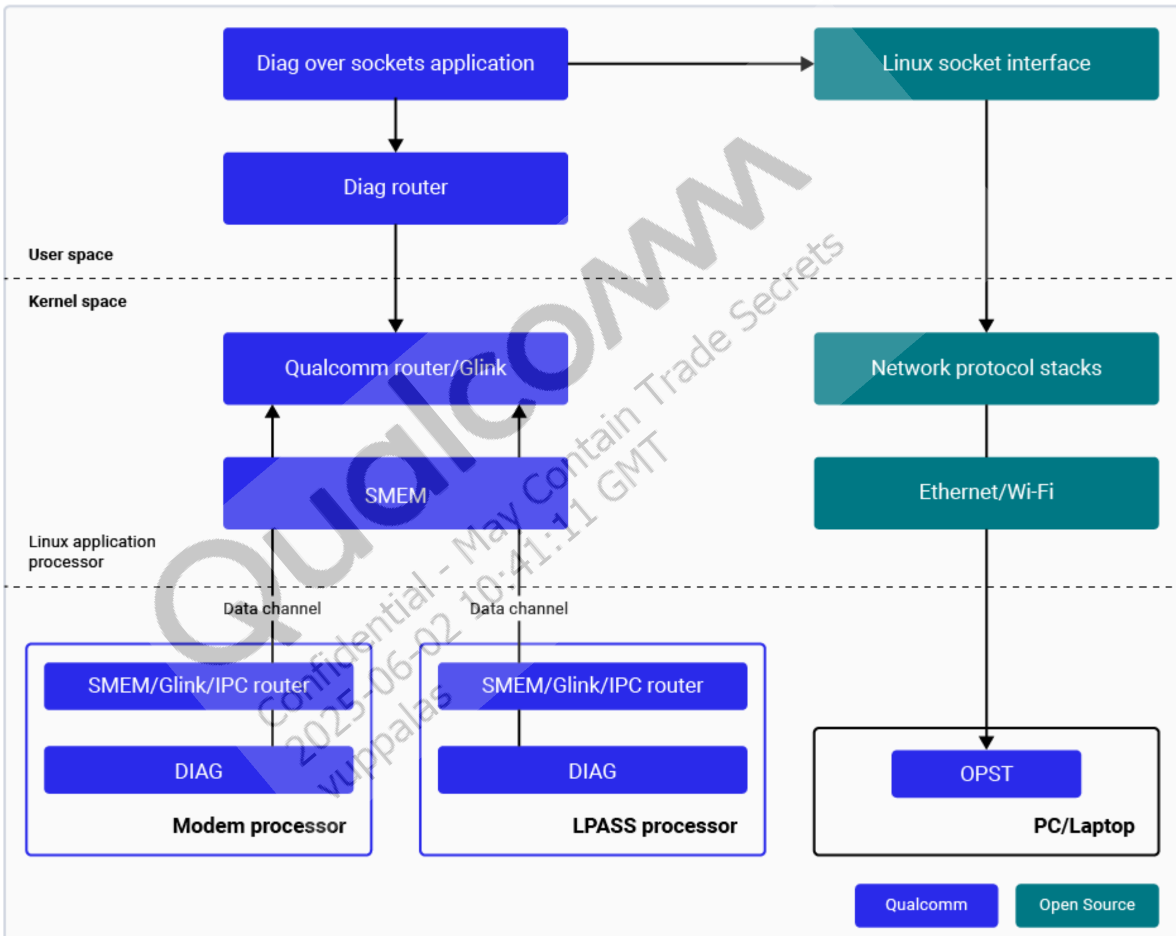


Figure : Architecture of Diag socket logging feature

Enable Diag socket logging

To enable Diag socket logging, do the following:

1. Set up the network configuration.

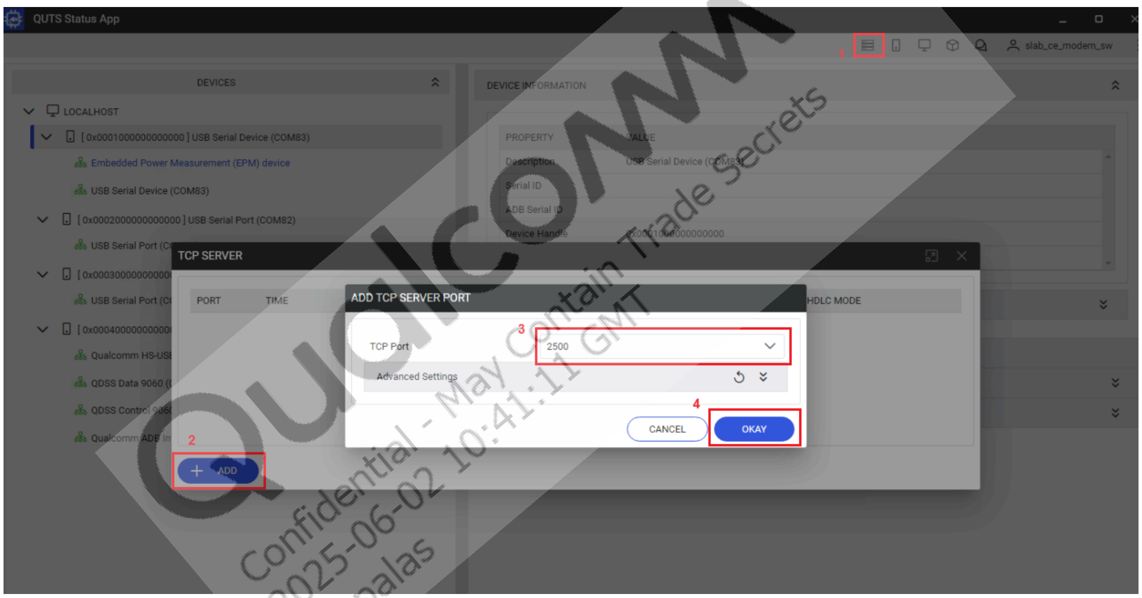
    - On PC

```
> ipconfig
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : lan
   Link-local IPv6 Address . . . . . : fe80::d51b:c96e:d96d:5777%19
   IPv4 Address. . . . . . . . . . . : 192.168.2.104
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.2.1
```

- On device

```
> adb root
> adb shell ifconfig

wlan0     Link encap:Ethernet  HWaddr 2a:6b:0a:3b:64:84  Driver icnss
          inet addr:192.168.2.233  Bcast:192.168.2.255  Mask:255.255.255.0
```

2. Set up the QUTS tool.



3. Start the diag_socket_log application.

   a. Plug in the device using USB.

   b. Open the SSH and run the following command:

```
diag_socket_log -a <IP address of PC> -p <port number> -r
<number of retries to connect> &
```

The following table lists the options that are passed as arguments to the diag_socket_log application:

Table : Arguments for diag_socket_log application

Argument	Description
`-a`	IP address. This is a mandatory argument.
`-p`	Port number; default is 2500. This is an optional argument.
`-r`	Number of retries for connect; default is 10000, with a 2 seconds wait between tries. This is an optional argument.
`-k`	Kills existing instance of diag_socket_log.
`-h`	Usage help; prints the list of arguments that are passed to the application.
`-c`	Device mask bits. • 1: MSM (for application processor) • 2: MDM (SDX attaches) • 4: MDM_2 (WLAN attaches)

Note: Appending *&* to the command causes the application to run in the background, even when the USB cable isn't connected.

Stop diag_socket_log application

To stop the diag_socket_log application, determine the process ID of the application and kill it.

```
ps
```

```
kill -9 <diag_socket_log process id>
```

When the diag_socket_log application exits, the Diag functionality returns to the default USB configuration.

QXDM Professional

Use QXDM Professional to debug various subsystems. Route the diag packets from a subsystem to the QXDM Professional.

For instructions on how to download and install QXDM Professional, see the User Guide - Qualcomm® Package Manager 3 Documentation.

For more information about QXDM Professional, see QXDM Professional Tool v5 for Windows OS User Guide.

Note: All QXDM commands start with `75 37`. For example, `75 37 03 00 00` is the command to trigger crash on the mDSP, and `75 37 03 48` is the command to trigger crash on the aDSP.

5 Debug common system issues

Watchdog timeout, bus hang, timeout error, and hardware reset are some common system issues. The following sections provide the information on how to identify and debug such system issues.

5.1 Watchdog issues

A watchdog (WD) is a fixed-length counter that allows a system to recover from an unexpected hardware or software catastrophe. Unless the system periodically pets the watchdog timer, the watchdog timer assumes a catastrophe and resets the subsystem or the entire system, depending on the watchdog that's fired.

Different types of watchdog implementations are:

- Hardware watchdog
- Software watchdog
- Bark
- Bite

The following table summarizes various types of watchdog implementation:

Types of watchdogs	Timeout duration	Owner	When expired	Result
Nonsecure WD bark	11 s	HLOS	IRQ to Qualcomm TEE	HLOS falls to Panic
Nonsecure WD bite	12 s	HLOS	Fast interrupt request (FIQ) to Qualcomm TEE	Qualcomm TEE asserts PS_HOLD
Secure WD bark	6 s	Qualcomm TEE	FIQ to Qualcomm TEE	Qualcomm TEE just pets secure WD
Secure WD bite	22 s	Qualcomm TEE	Asserting PS_HOLD	PMIC resets the system
AOP hardware WD bark	10 ms	AOP	IRQ to AOP	AOP falls to fatal error

Types of watchdogs	Timeout duration	Owner	When expired	Result
AOP hardware WD bite	30 ms	AOP	IRQ to application processor	HLOS falls to panic
Subsystem hardware WD bark	2.25 s	Dog task on subsystem	FIQ to error handler	Subsystem error fatal/pet WD [1]
Subsystem nonmaskable interrupt (NMI) due to hardware WD	2.4 s	Dog task on subsystem	NMI to subsystem	NMI on subsystem [2]
Subsystem hardware WD bite	2.5 s	Dog task on subsystem	IRQ to HLOS	Subsystem hardware reset [2]

[1]On aDSP and cDSP, watchdog bark FIQ pets the hardware watchdog. It's a fatal error.

[2]Error fatal or NMI or hardware reset on the subsystem leads to or restarts a subsystem, depending on the remoteproc configuration. For more information about the remoteproc driver, see Qualcomm Linux Kernel Guide.

The system uses both software and hardware watchdogs. For example, mDSP implements both software and hardware watchdogs. The hardware watchdog module ensures that the processor is active and consists of a timer that counts down from a predetermined value. If the timer isn't reset by the corresponding CPU core, it eventually counts to 0 and triggers a watchdog timeout.

Watchdog for application processor CPU

Nonsecure hardware watchdog

- Every 10 seconds, the HLOS triggers a timer event to pet the nonsecure hardware watchdog. If the HLOS doesn't pet the nonsecure watchdog for 11 seconds, the nonsecure watchdog bark fires and the HLOS must handle it. If the HLOS can't handle it, the HLOS falls into panic.
- If the HLOS is unable to handle nonsecure watchdog bark, it triggers a nonsecure watchdog bite and sends it to Qualcomm TEE, causing the Qualcomm TEE to fall into a fatal error.
- You can customize the watchdog pet and bark time using the kernel configuration. For example, the following configuration sets the bark time to 13 seconds and pet time to 11 seconds.

  ```
  CONFIG_QCOM_WATCHDOG_BARK_TIME=13000
  ```

  ```
  CONFIG_QCOM_WATCHDOG_PET_TIME=11000
  ```

Secure hardware watchdog

- Every 6 seconds, Qualcomm TEE triggers a secure watchdog bark as a fast interrupt request (FIQ). The FIQ handler in the Qualcomm TEE pets the secure hardware Watchdog. This isn't an error or a fatal issue.
- If Qualcomm TEE can't handle the secure watchdog bark for 22 seconds, the secure watchdog bite expires. Then, the PMIC asserts the PS_HOLD pin, and eventually, the entire system is reset.

Watchdog for Always On Processor (AOP)

Software watchdog

The AOP doesn't include a software watchdog.

Hardware watchdog

- Software on AOP must pet AOP watchdog hardware. If AOP doesn't pet it within 10 ms, AOP watchdog bark is triggered and software on AOP must handle it. This issue eventually causes an AOP fatal error.
- If software on AOP can't handle the AOP Watchdog bark, AOP Watchdog bite fires. The watchdog hardware resets AOP and sends a notification to the Qualcomm TEE on the application processor. Then, the application processor falls into panic.
- Watchdog hardware automatically stops when JTAG is attached to AOP.

Watchdog for other subsystems

Software watchdog

The mDSP includes a software watchdog. Each user task registers to the watchdog task. All tasks must ping a watchdog task within 10 seconds. If the ping doesn't happen, the watchdog task detects the ping failure and falls into a fatal software error.

Hardware watchdog

- On mDSP, the watchdog task pets the subsystem watchdog hardware every 100 ms. If the watchdog task doesn't pet the subsystem watchdog hardware within 2.25 seconds, the subsystem watchdog triggers bark FIQ, and the subsystem falls into a fatal software error.
- On aDSP and cDSP, a hardware watchdog bark (FIQ) is triggered every 2.25 seconds, and the FIQ handler pets the hardware watchdog. This error is a fatal condition.
- If the hardware watchdog bark isn't handled properly, that is, if the software didn't fall into fatal error, NMI is triggered in 2.4 seconds.
- If the hardware watchdog isn't pet within 2.5 seconds, the hardware watchdog bite is triggered and the subsystem is reset. Also, the hardware watchdog bite is notified to HLOS on the application processor. The corresponding NMI that's sent to the DSP saves the registers and flushes the cache.

5.2 Bus hang and timeout error

The SNoC, CNoC, xPU, TBU, and AHB are the system infrastructure components on the device, which are responsible for operations such as:

- Bus transaction
- Address translation
- Memory protection

Failures or timeouts on these system infrastructure components can cause system errors, which are then reported to the Qualcomm TEE.

A bus hang occurs when a bus primary, processor, or nonprocessor accesses a subsystem that may have one of the key clocks off in such a way that the bus waits indefinitely to access the subsystem. The processor waits indefinitely and the only way for the processor to recover is to restart the system using a watchdog.

To debug bus hang issues, most buses implement the following monitors:

- NoC timeout monitor
- AHB timeout monitor

Another common issue is invalid access to DDR from an unintended primary without appropriate permission. To avoid this issue, introduce the xPU and TBU (SMMU) to build a more secure system and simplify debugging.

NoC error and timeout

The Qualcomm TEE generates an FIQ after detecting the NoC and timeout errors. You can parse the NoC error with the following decoder on a Windows host computer: `trustzone_imagescoresystemdriversicbscriptsnocerrorhNoC_error_decode.py`

```
python NoC_error_decode.py kodiak system_noc 0x104 0x1f 0xa5 0x0 0x0
0x0 0x0 0x0 0x1 0x0 0x0 0x0
```

Sample output:

```
SYSTEM_NOC ERROR: ERRLOG0_LOW = 0x00000104
SYSTEM_NOC ERROR: ERRLOG0_HIGH = 0x0000001f
SYSTEM_NOC ERROR: ERRLOG1_LOW = 0x000000a5
TZBSP_EC_MEM_DUMP_SECURE_WDOG_RESET
SYSTEM_NOC ERROR: SBM0 FAULTINSTATUS0_LOW = 0x00000001
```

AHB timeout

A few AHBs interconnect within the system, primarily in the peripheral subsystem. If an access timeout occurs on a transaction, for reporting an AHB timeout, FIQ is sent to Qualcomm TEE.

xPU violations

xPU violations occur when the application processor subsystem tries to access a system resource, such as a memory region or device controller register for which either appropriate access permission isn't set, or the device controller isn't functional, due to the unavailability of the clocks.

During the development and testing phase, configure the xPU violations as fatal to ensure capturing all unexpected issues. The Qualcomm TEE diag captures the xPU logs.

The following table lists the fields in the xPU dump:

Table : Fields in xPU dump

Field	Description
`XPU ID`	ID to identify a device controller or hardware module.
`uPhysicalAddress`	Address that was read/written with reference to xPU ID.
`BID or Bus ID`	SNoC, PNoC, or CNoC.
`PID` or `Port ID`	Port number on the bus identified with `BID` from which the transaction originated.
`MID`	Primary ID that initiated the transaction.
`VMID`	Virtual Master ID. This ID can be the same as MID but uniquely identifies a master.

The xPU error messages are formatted as follows:

```
(1001b6 <xpu_Name> <xpu_err_count> <xpu_id> <errorFlags> <busFlags>
<PhysicalAddressLower32> <PhysicalAddressUpper32> <uMasterID>
<uAVMID>
<uATID> <uABID> <uAPID> <uALen> <uASize> <uAReqPriority> <uAMemType>)
```

Sample log:

```
[0000000015b33a9f] xpu: ISR begin
[0000000015b33db0] XPU ERROR: Non Sec!!
[0000000015b4210c] xpu: uPhysicalAddressLower: fde54000 Upper:
00000000
[0000000015b42554] xpu: uMasterId: 0000008c, uAVMID : 00000000
[0000000015b42935] xpu: uATID : 0000000e, uABID : 00000002
[0000000015b42d1d] xpu: uAPID : 00000006, uALen : 00000000
```

In this sample log, `0xfde54000` is the error syndrome register and represents the address for which the client is requesting access.

Translation buffer unit error

The system-level MMU uses the translation buffer unit (TBU/SMMU). The TBU supports address translation from an input address to an output address, based on address mapping and memory attribute information held in translation tables.

When a subsystem or peripheral tries to access an invalid address or with an invalid attribute, the SMMU error occurs. This fatal error is routed to the Qualcomm TEE.

For more information, see [Arm System Memory Management Unit Architecture Specification, IHI0062D](#).

5.3 Hardware reset

A secure watchdog, temperature sensor (TSENS), or PMIC issue can cause a hardware reset. The debugging approach for hardware reset issues depends on the cause of the hardware reset. To identify the cause of hardware reset, it's necessary to collect the RAM dump because it includes the hardware registers dump.

When a hardware reset occurs, the system generates `RST_STAT.BIN` and `PMIC_PON.BIN` files. These files contain the `AOSS_CC_RESET_STATUS` register value and the PMIC power-on status register dump. The `AOSS_CC_RESET_STATUS` hardware register indicates the reset status. The `AOSS_CC_RESET_STATUS` is also known as `GCC_RESET_STATUS` register.

The following are a few sample values for `AOSS_CC_RESET_STATUS` register:

- 0x03 and 0x23: Secure watchdog bite
- 0x1B: Temperature sensor triggered reset
- 0x42: PMIC abnormal reset

The QCAP report also parses hardware register information as shown in the following screenshot.

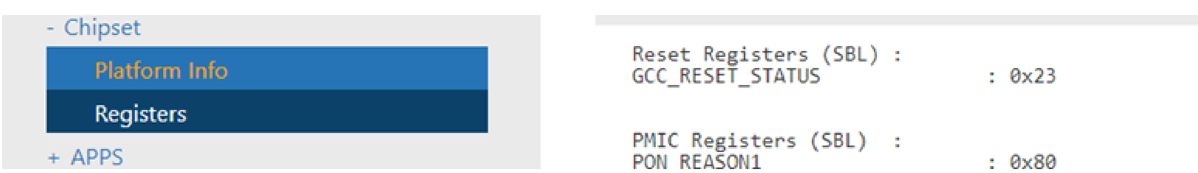


Figure : QCAP parsing hardware register information

The following is the general warm reset sequence between the device and the PMIC:

1. The device pulls `PS_HOLD` low.
2. PMIC pulls `PON_RESET_N` low to keep the device in Reset mode.

3. PMIC performs the warm reset.
4. PMIC pulls `PON_RESET_N` high to take the device out of reset.

PMIC reset

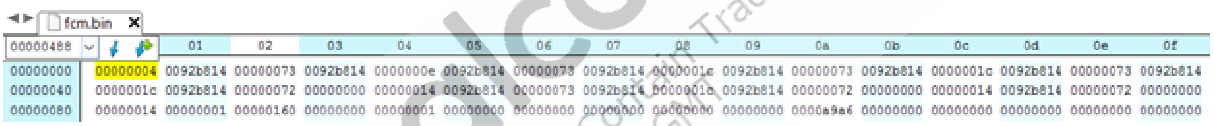
The SDI hardware and software perform a `PS_HOLD` warm reset before the RAM dump occurs. Therefore, analyzing the `PMON_HIS.BIN` file helps to find the cause of the PMIC reset. For more information on PMIC reset, see *HLOS PMIC PON Software User Guide* (80-P2484-40).

CPU hang register dump

One of the common reasons for the secure watchdog bite is a CPU hang. You can check the status of the CPU hang in the `fcm.bin` file, which the SDI collects. The RAM dump parser extracts the `fcm.bin` file using the `--parse-debug-image` option.

First core hang

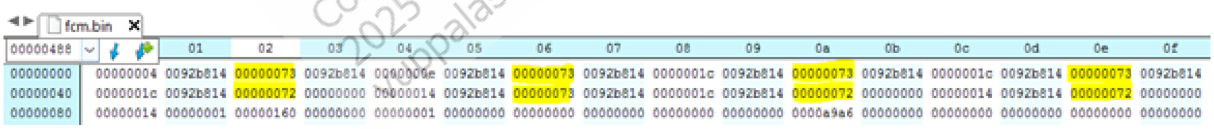
The first four bytes of the `fcm.bin` indicate the first core hang. In the following screenshot, core 2 is the first core hang.



The first core hang causes the secure watchdog bite. The value 0x0 indicates that there is no core hang.

Core hang status

The highlighted values in the following screenshot indicate the status of core hang for each core:



Bit 0 indicates the actual core hang status. A value of 1 indicates that the specific core is hung. In the following example, cores 0, 1, 2, 3, and 5 are hung.

```
Core0 (offset 0x8) : 0x00000073
Core1 (offset 0x18) : 0x00000073
Core2 (offset 0x28) : 0x00000073
Core3 (offset 0x38) : 0x00000073
Core4 (offset 0x48) : 0x00000072
Core5 (offset 0x58) : 0x00000073
Core6 (offset 0x68) : 0x00000072
Core7 (offset 0x78) : 0x00000072
```

6 References

6.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Linux Debug Guide	80-70018-12
Qualcomm Linux Kernel Guide	80-70018-3
Qualcomm Linux Security Guide	80-70018-11
Qualcomm Linux Yocto Guide	80-70018-27
QCAP Systems Overview	80-NR964-54
QXDM Professional Tool v5 for Windows OS User Guide	80-V1241-25
Resources	
Qualcomm® Package Manager 3 Documentation	
Arm System Memory Management Unit Architecture Specification, IHI0062D	

6.2 Acronyms and terms

Acronym or term	Definition
aDSP	Application digital signal processor
AHB	Advanced High-performance Bus
AOP	Always-on-processor
APSS	Application processor subsystem
cDSP	Compute DSP
CMA	Contiguous memory allocator
CNoC	Config network-on-chip
DCC	Data capture and compare
DDR	Double data rate

Acronym or term	Definition
FAR	Fault address register
FIQ	Fast interrupt request
FSR	Fault status register
GDB	GNU debugger
GFP	Get free pages
HLOS	High-level OS
LPASS	Low-power audio subsystem
mDSP	Modem DSP
MPM	Mobile Station modem power manager
NMI	Nonmaskable interrupt
NoC	Network-on-chip It's a bus connecting many subsystems on the SoC. There are many types of NoCs such as System NoC (SNoC), Config NoC (CNoC), Multimedia NoC (MMNoC).
PCB	Printed circuit board
PDC	Power domain controller
PMIC	Power management IC PMIC is the power supply block of the chipset.
PS_HOLD	It's a power supply hold signal line from the chipset to the PMIC.
QCAP	Qualcomm crash analysis portal
QDSS	Qualcomm debug subsystem
QXDM	Qualcomm extensible diagnostic monitor
SDI	Software debug image This is a debug feature in the Qualcomm TEE to capture context of core for all subsystems in the RAM dump during an abnormal reset. For more information about RAM dump, see RAM dump.
SNoC	System network-on-chip
SPM	Subsystem power manager
SMMU	System memory management unit
TRACE32	Lauterbach TRACE32 software
TBU	Translation buffer unit. Arm SMMU IP component.
TCM	Tightly coupled memory
TSENS	Temperature sensor It captures the junction temperature of the chipset.
TZ	TrustZone

Acronym or term	Definition
WD	Watchdog A watchdog is a fixed-length counter that allows a system to recover from an unexpected hardware or software catastrophe.
WD bark	Watchdog timeout that results into a bark interrupt and kernel panic.
WB bite	Watchdog timeout that occurs if a watchdog isn't petted even after a WD bark, resulting in a bite interrupt in secure mode. This issue further leads to a system reset.
WPSS	Wireless local area network processor subsystem
XBL	eXtensible Boot Loader
xPU	External protection unit It's a module in the Qualcomm TEE for protecting memory regions, addresses, and registers.

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.