Qualcomm Linux Audio Guide - Addendum

80-70018-16A AA

March 19, 2025

Contents

1 Audio Addendum documentation

Use the audio subsystem to set up and customize audio devices.

1.1 Bring up audio

Verify aDSP is enabled

Verify that the PIL has loaded the application DSP image.

Verify sound card registration

Verify sound card registration to record and play back audio.

Verify record and playback at the driver level

Push an audio file to the device and play back.

1.2 Audio tools

Calibrate audio with QACT

Use the QACT tool to connect to a device, view files, and design and tune audio modules.

Enable diagnostic logging with QXDM

Use QXDM Professional™ to configure and filter audio logs.

1.3 Advanced audio customization

Sync and compile audio components

Extract the audio module source code and build the user space and kernel mode modules.

Custom module integration in DSP

Use the Qualcomm® Hexagon™ NPU SDK to add custom audio modules.

Connect third-party audio devices over MI2S/TDM

Use the MI2S and TDM interfaces to connect and transfer audio data to a device.

1.4 Tune audio

Calibrate offline

Change calibration files without connecting to a device.

Calibrate online

Change calibration files while in use by a device.

Tuning workflow

Follow these workflows to improve audio quality.

Cancel echo and suppress noise

Cancel the echo from the far-end and suppress the noise in the near-end audio signal.

Optimize playback quality

Improve the quality of the audio.

2 Bring up audio

QCS6490

The following diagram shows the audio bringup process.

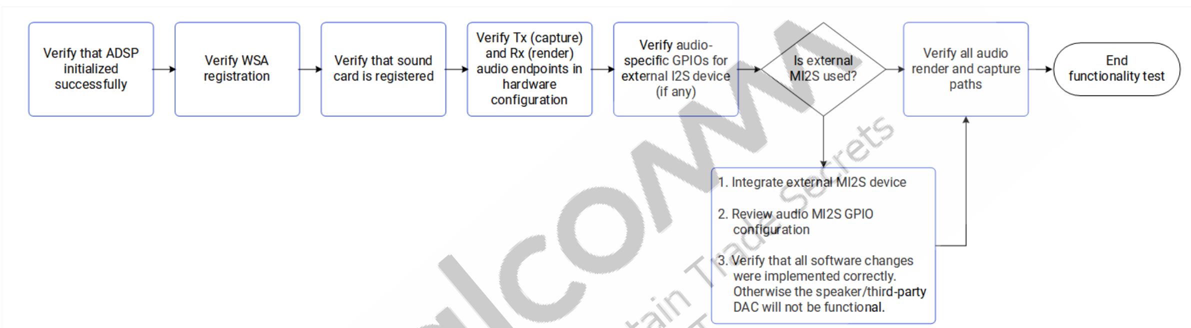


Figure1 Audio bringup process

2.1 Verify aDSP is enabled

Locate the following log marker to verify that the PIL has loaded the application DSP (aDSP) image successfully:

```
[    7.816174] remoteproc remoteproc1: Booting fw image qcom/
qcm6490/adsp.mdt, size 7332
[    8.063332] remoteproc remoteproc1: remote processor 3000000.
remoteproc is now up
[   18.581373] spf-core-platform soc@0:spf-core-platform: spf_
core_add_child_devices: apm is up
[   18.822671] qcom-soundwire 3250000.soundwire: Qualcomm
Soundwire controller v1.6.0 Registered
[   18.824072] wsa883x-codec sdw:0:0217:0202:00:2: WSA883X
Version 1_1, Variant: WSA8835_V2
[   18.829857] wsa883x-codec sdw:0:0217:0202:00:1: WSA883X
Version 1_1, Variant: WSA8835_V2
```

If the aDSP fails to load, consider re-flashing the device.

2.2 Verify WSA registration

1. Verify the codec registration:

```
ssh root@ip-addr
ls /sys/bus/soundwire/drivers/wsa883x-codec
```

2. Locate the following entries for the WSA:

```
sdw:0:0217:0202:00:1
sdw:0:0217:0202:00:2
```

If the WSA isn't registered, reconnect the speakers and restart the device.

2.3 Verify sound card registration

After mapping the digital audio interface links:

1. Verify sound card registration:

```
ssh root@ip-addr
cat /proc/asound/cards
```

2. Locate the sound card named `qcm6490-rb3-snd-card`.

If sound card registration fails, re-flash the device.

2.4 Verify record and playback

1. Verify record at the driver level:

```
ssh root@ip-addr
systemctl stop pulseaudio
tinymix set 'VA DMIC MUX0' DMIC0
tinymix set 'VA_AIF1_CAP Mixer DEC0' 1
tinymix set 'VA_DEC0 Volume' 100
agmcap /opt/test.wav -D 100 -d 101 -c 1 -r 48000 -b 16 -i
CODEC_DMA-LPAIF_VA-TX-0 -dkv 0xA3000004 -skv 0xB1000009 -
dppkv 0 -ikv 0 -T 10
```

2. To pull the recorded file, run the following command on the host PC:

```
scp root@[ip-addr]:/opt/test.wav
```

3. Push the .wav file to the device:

```
scp test.wav root@[ip-addr]:/opt/
```

4. Start playback:

```
ssh root@ip-addr
systemctl stop pulseaudio
scp test.wav root@[ip-addr]:/opt/
ssh root@ip-addr
tinymix set 'SpkrLeft PA Volume' 20
tinymix set 'WSA RX0 MUX' AIF1_PB
tinymix set 'WSA_RX0 INP0' RX0
tinymix set 'WSA_COMP1 Switch' 1
tinymix set 'SpkrLeft WSA MODE' 0
tinymix set 'SpkrLeft COMP Switch' 1
tinymix set 'SpkrLeft BOOST Switch' 1
tinymix set 'SpkrLeft DAC Switch' 1
tinymix set 'SpkrLeft VISENSE Switch' 0
tinymix set 'WSA_RX0 Digital Volume'  85
tinymix set 'SpkrRight WSA MODE' 0
tinymix set 'SpkrRight PA Volume' 20
tinymix set 'WSA RX1 MUX' AIF1_PB
tinymix set 'WSA_RX1 INP0' RX1
tinymix set 'WSA_COMP2 Switch' 1
tinymix set 'SpkrRight COMP Switch' 1
tinymix set 'SpkrRight BOOST Switch' 1
tinymix set 'SpkrRight DAC Switch' 1
tinymix set 'SpkrRight VISENSE Switch' 0
tinymix set 'WSA_RX1 Digital Volume'  85
agmplay /opt/test.wav -D 100 -d 100 -i CODEC_DMA-LPAIF_WSA-
RX-0
```

5. Start PulseAudio after verifying the above use cases:

```
ssh root@ip-addr
systemctl start pulseaudio
```

QCS9075

The following figure shows the audio bringup process.

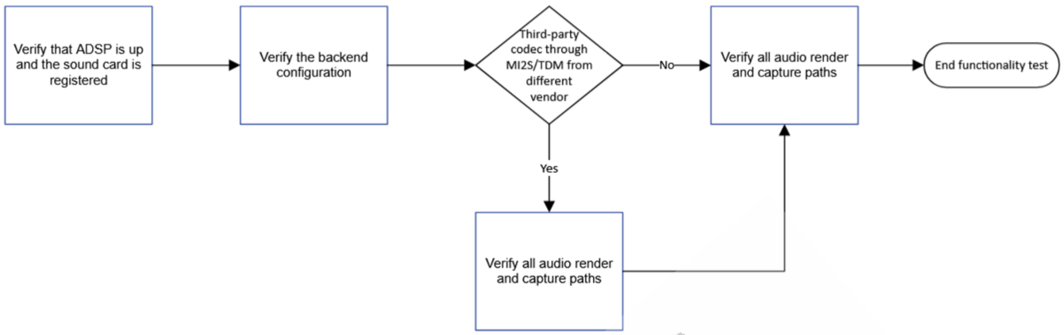


Figure2 Audio bringup process

2.5 Verify aDSP is enabled

Locate the following log marker to verify that the PIL has loaded the application DSP (aDSP) image successfully:

```
[   11.133283] remoteproc remoteproc0: 30000000.remoteproc is available
[   11.143106] remoteproc remoteproc0: powering up 30000000.remoteproc
[   11.143116] remoteproc remoteproc0: Booting fw image qcom/sa8775p/adsp.mbn, size 6643712
[   11.205694] remoteproc remoteproc0: remote processor 30000000.remoteproc is now up
[   11.555230] spf-core-platform soc@0:spf-core-platform: spf_core_add_child_devices: apm is up
```

If the aDSP fails to load, consider re-flashing the device.

2.6 Verify sound card registration

After mapping the digital audio interface links:

1. Verify sound card registration:

```
ssh root@ip-addr
cat /proc/asound/cards
```

2. Locate the sound card named `qcs9075-rb8-snd-card`.

If sound card registration fails, re-flash the device.

2.7 Verify record and playback

To verify record at the driver level, update the backend configuration.

1. Pull the backend configuration file:

```
scp root@[ip-addr]:/etc/backend_conf.xml .
```

2. Add the following device names to the backend configuration file:

```
<device name="MI2S-LPAIF_SDR-TX-TERTIARY" rate="48000" ch="1"
bits="16" />
<device name="MI2S-LPAIF_SDR-RX-PRIMARY" rate="48000" ch="2"
bits="16" />
```

3. Push the backend configuration file:

```
scp backend_conf.xml root@[ip-addr]:/etc/
```

4. Run the following commands:

```
ssh root@ip-addr
systemctl stop pulseaudio
agmcap /opt/test.wav -D 100 -d 101 -c 1 -r 48000 -b 16 -i
MI2S-LPAIF_SDR-TX-TERTIARY -dkv 0xA3000001 -skv 0xB1000009 -
dppkv 0 -ikv 0 -T 10
```

5. Pull the recorded file on the host PC:

```
scp root@[ip-addr]:/opt/test.wav .
```

6. Push the .wav file to the device:

```
scp test.wav root@[ip-addr]:/opt/
```

7. Start playback:

```
ssh root@ip-addr
systemctl stop pulseaudio
scp test.wav root@[ip-addr]:/opt/
ssh root@ip-addr
agmplay /opt/test.wav -D 100 -d 100 -i MI2S-LPAIF_SDR-RX-
PRIMARY
```

8. Start PulseAudio after verifying the above use cases:

```
ssh root@ip-addr
systemctl start pulseaudio
```

QCS8275

The following figure shows the audio bringup process.

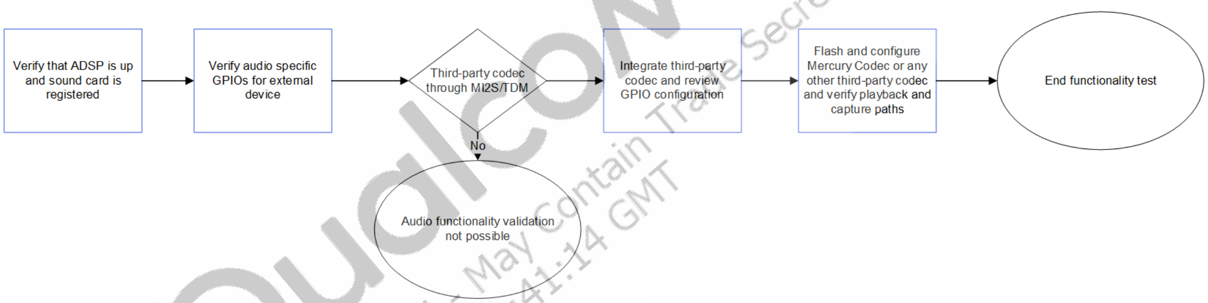


Figure3 Audio bringup process

The Mercury Codec in the Qualcomm Reference Platform requires firmware from NXP. You must get this directly from NXP due to licensing agreements.

2.8 Verify aDSP is enabled

Locate the following log marker to verify that the PIL has loaded the application DSP (aDSP) image successfully:

```
[    3.633740][  T679] remoteproc remoteproc0: 30000000.
remoteproc is available
[    3.642907][  T638] remoteproc remoteproc0: Booting fw image
qcom/sa8775p/adsp.mbn, size 6651904
[    3.705364][  T638] remoteproc remoteproc0: remote processor
30000000.remoteproc is now up
```

```
[    3.748622][  T637] spf-core-platform soc@0:spf-core-
platform: spf_core_add_child_devices: apm is up
```

If the aDSP fails to load, consider re-flashing the device.

2.9 Verify sound card registration

After mapping the digital audio interface links:

1. Verify sound card registration:

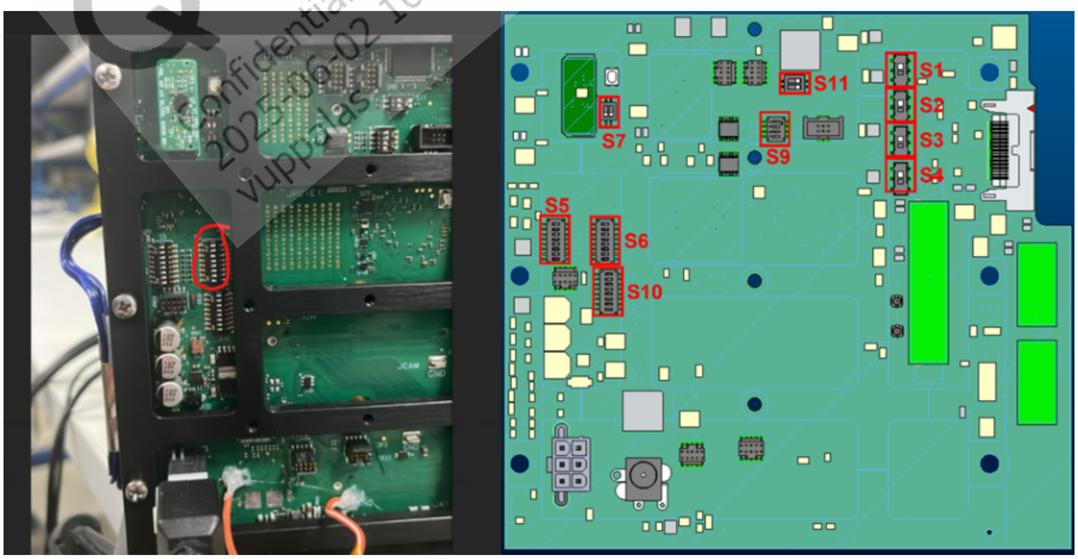
   ```
   ssh root@ip-addr
   cat /proc/asound/cards
   ```

2. Locate the sound card named `qcs8300-ridesx-snd-card`.

If sound card registration fails, re-flash the device.

2.10 Flash Mercury Codec

1. Power off the device.
2. Open the back panel of the device.
3. Set the DIP switch S6 bit7 to ON and the DIP switch S6 bit8 to OFF.



4. Power on the device.
5. Mount the device:

```
ssh root@ip-addr
mount -o rw,remount /
```

6. Push the Mercury firmware received from NXP to the following location:

```
ssh root@ip-addr
scp -r mercury_firmware\. root@[ip-addr]:/etc/firmware/
```

7. Set the device in Flashing mode. Select 1 (Flash Mercury) followed by 7 with `dac_mer_testapp`.

```
ssh root@ip-addr
dac_mer_testapp
```

8. Flash the firmware:

```
mercuryflasher -f
```

If the above command fails, check the output of the following commands for debug.

Check if the ping works:

```
mercuryflasher -p
```

Check if the read works:

```
mercuryflasher -r
```

Note: The commands should all return success or 0a.

The Mercuryflasher tool doesn't work directly on the first candidate build. See the latest tool for details.

9. Power off the device.
10. Set the DIP switch S6 bit7 to OFF. Use the preceding figure as reference.
11. Power on the device.

2.11 Verify record and playback

1. Set the playback and record mode:

```
ssh root@ip-addr
dac_mer_testapp
```

2. Select option 2 (SOC Mercury) followed by 1 (Enable Playback Setup) for playback and record use cases.

3. To pull the backend configuration file:

```
scp root@[ip-addr]:/etc/backend_conf.xml .
```

4. Add the following device names to the backend configuration file:

```
<device name="TDM-LPAIF_SDR-TX-PRIMARY" rate="48000" ch="1"
bits="16" />
<device name="TDM-LPAIF_SDR-RX-PRIMARY" rate="48000" ch="2"
bits="16" />
```

5. Push the backend configuration file:

```
scp backend_conf.xml root@[ip-addr]:/etc/
```

6. Run the following commands in a new command shell:

```
ssh root@ip-addr
systemctl stop pulseaudio
agmcap /opt/test.wav -D 100 -d 101 -c 1 -r 48000 -b 16 -i
TDM-LPAIF_SDR-TX-PRIMARY -dkv 0xA3000001 -skv 0xB1000009 -
dppkv 0 -ikv 0 -T 10
```

7. Pull the recorded file on the host PC:

```
scp root@[ip-addr]:/opt/test.wav .
```

8. Push the .wav file to the device:

```
scp test.wav root@[ip-addr]:/opt/
```

9. Start playback:

```
ssh root@ip-addr
systemctl stop pulseaudio
scp test.wav root@[ip-addr]:/opt/
ssh root@ip-addr
agmplay /opt/test.wav -D 100 -d 100 -i TDM-LPAIF_SDR-RX-
PRIMARY
```

10. Start PulseAudio after verifying the above use cases:

```
ssh root@ip-addr
systemctl start pulseaudio
```

2.12 Audio hardware interfaces

Hardware configuration details provide information about the supported audio hardware interfaces, speaker amplifier, and MI2S/TDM interfaces.

QCS6490

The audio hardware interfaces:

- Support WSA883x amplifier through SoundWire interface
- Support up to 4 digital microphones
- Connect WCN over SLIMbus interface for Bluetooth®
- Optionally connect third-party codec or speaker amplifier over MI2S/TDM interfaces

The following figure shows the supported interfaces:

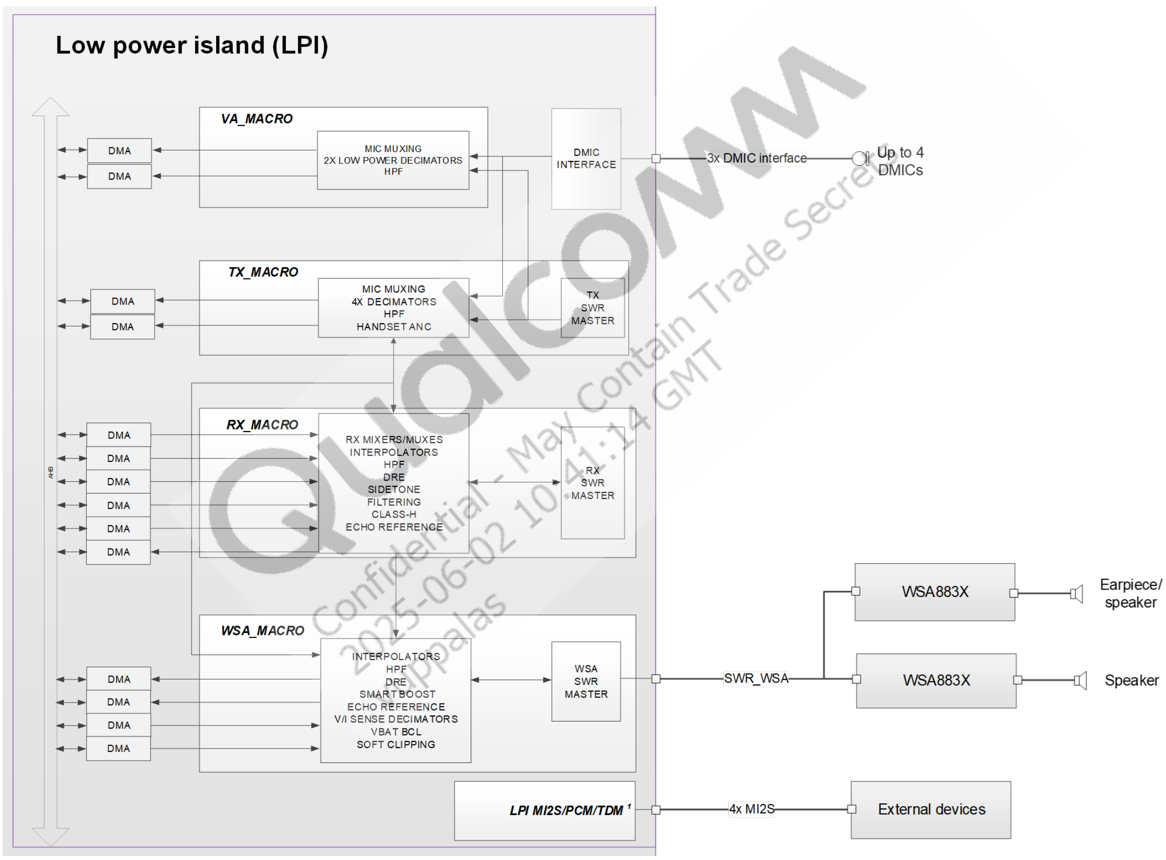


Figure4 Supported hardware interfaces

ALSA codec drivers

The codec class driver is a generic and hardware-independent ALSA compliant driver. It configures the codec and speaker amp to provide audio capture and playback.

Configure the device tree

The device tree is a data structure and language that describes hardware components on a System-on-Chip (SoC). It was created by the Open Firmware standards to unify the discovery of hardware devices connected to the SoC. The device tree abstracts hardware discovery from the Linux kernel source code, eliminating the need for hardcoded hardware information.

The device tree entries also have DAI links for PCM nodes on the SoC. The device tree defines the DAI links for PCM nodes. They are part of the sound node.

The following code shows this:

```
&sound {
      compatible = "qcom,qcm6490-sndcard";
      model = "qcm6490-rb3-vision-snd-card";

audio-routing =
      "SpkrLeft IN", "WSA_SPK1 OUT",
      "SpkrRight IN", "WSA_SPK2 OUT",
      "VA DMIC0", "vdd-micb",
      "VA DMIC1", "vdd-micb",
      "VA DMIC2", "vdd-micb",
      "VA DMIC3", "vdd-micb",
      "VA DMIC4", "vdd-micb",
      "VA DMIC5", "vdd-micb";

pinctrl-names = "default", "stub_aif1_active", "stub_aif1_sleep
";
pinctrl-0 = <&mi2s0_data0_sleep>, <&mi2s0_data1_sleep>, <&mi2s0_
mclk_sleep>,
            <&mi2s0_sclk_sleep>,<&mi2s0_ws_sleep>;
pinctrl-1 = <&mi2s0_data0>, <&mi2s0_data1>, <&mi2s0_mclk>, <&
mi2s0_sclk>, <&mi2s0_ws>;
pinctrl-2 = <&mi2s0_data0_sleep>, <&mi2s0_data1_sleep>, <&mi2s0_
mclk_sleep>,
            <&mi2s0_sclk_sleep>,<&mi2s0_ws_sleep>;

mi2s-capture-dai-link {
      link-name = "MI2S-LPAIF-TX-PRIMARY";

      cpu {
```

```
                sound-dai = <&q6apmbedai PRIMARY_MI2S_TX>;
        };

        codec {
                sound-dai = <&msm_stub_codec 1>;
        };
};

mi2s-playback-dai-link {
        link-name = "MI2S-LPAIF-RX-PRIMARY";

        cpu {
                sound-dai = <&q6apmbedai PRIMARY_MI2S_RX>;
        };

        codec {
                sound-dai = <&msm_stub_codec 0>;
        };
};
```

This code snippet defines the backend DAI of WSA. `cpu` denotes the entry for CPU. The platform DAI and `codec` node entry is for the codec DAI link. The name signifies a DAI-link stream name and is used by AGM to configure backends.

QCS9075

The audio hardware interfaces:

- Connect Maxim speaker amp over HS0 MI2S interface
- Connect MIC over HS2 MI2S interface
- Optionally connect other third-party codec or speaker amplifier over MI2S/TDM interfaces

The following figure shows the supported interfaces:

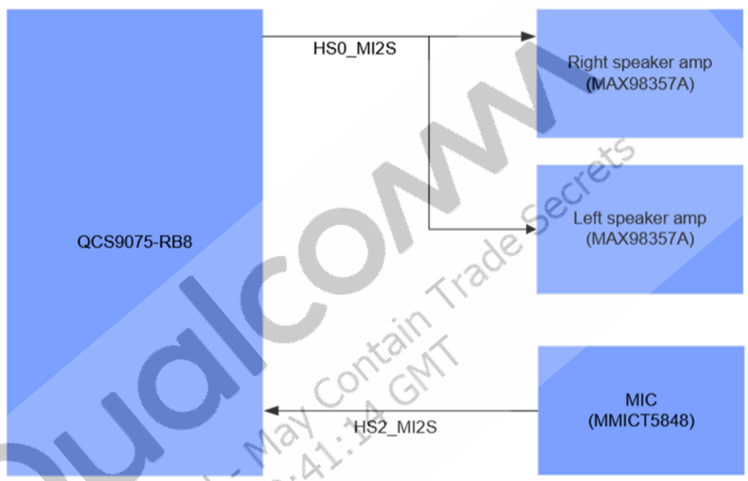


Figure5 Supported hardware interfaces

ALSA codec drivers

The codec class driver is a generic and hardware-independent ALSA compliant driver. It configures the codec and speaker amp to provide audio capture and playback.

Configure the device tree

The device tree is a data structure and language that describes hardware components on a System-on-Chip (SoC). It was created by the Open Firmware standards to unify the discovery of hardware devices connected to the SoC. The device tree abstracts hardware discovery from the Linux kernel source code, eliminating the need for hardcoded hardware information.

The device tree entries also have DAI links for PCM nodes on the SoC. The device tree defines the DAI links for PCM nodes. They are part of the sound node.

The following code shows this:

```
sound {
             compatible = "qcom,qcs9075-rb8-sndcard";
             model = "qcs9075-rb8-snd-card";

             clocks = <&q6prmcc LPASS_HW_MACRO_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>,
                         <&q6prmcc LPASS_HW_DCODEC_VOTE LPASS_
CLK_ATTRIBUTE_COUPLE_NO>;
             clock-names = "macro", "dcodec";

             pinctrl-names = "default", "mi2s_aud_out_active",
"mi2s_aud_out_sleep";
             pinctrl-0 = <&hs1_mi2s_data0_sleep>, <&mi2s_mclk_
sleep>, <&hs0_mi2s_data0_sleep>,
                         <&hs0_mi2s_sclk_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_ws_sleep>,
                         <&hs1_mi2s_sclk_sleep>, <&hs1_mi2s_
data1_sleep>, <&hs1_mi2s_ws_sleep>,
                         <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                         <&hs2_mi2s_ws_sleep>, <&lpass_quad_clk_
sleep>, <&lpass_quad_data_sleep>,
                         <&lpass_quad_ws_sleep>;
             pinctrl-1 = <&hs1_mi2s_data0>, <&mi2s_mclk>, <&hs0_
mi2s_data0>,
                         <&hs0_mi2s_sclk>, <&hs0_mi2s_data1>, <&
hs0_mi2s_ws>,
                         <&hs1_mi2s_sclk>, <&hs1_mi2s_data1>, <&
hs1_mi2s_ws>,
                         <&hs2_mi2s_data0>, <&hs2_mi2s_data1>,
<&hs2_mi2s_sck>,
                         <&hs2_mi2s_ws>, <&lpass_quad_clk>, <&
lpass_quad_data>,
                         <&lpass_quad_ws>;
             pinctrl-2 = <&hs1_mi2s_data0_sleep>, <&mi2s_mclk_
sleep>, <&hs0_mi2s_data0_sleep>,
                         <&hs0_mi2s_sclk_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_ws_sleep>,
                         <&hs1_mi2s_sclk_sleep>, <&hs1_mi2s_
data1_sleep>, <&hs1_mi2s_ws_sleep>,
                         <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                         <&hs2_mi2s_ws_sleep>, <&lpass_quad_clk_
sleep>, <&lpass_quad_data_sleep>,
```

```
                        <&lpass_quad_ws_sleep>;

                hs0-mi2s-playback-dai-link {
                        link-name = "MI2S-LPAIF_SDR-RX-PRIMARY";

                        cpu {
                                sound-dai = <&q6apmbedai PRIMARY_
SDR_MI2S_RX>;
                        };

                        codec {
                                sound-dai = <&max98357a>;
                        };
                };

                hs2-mi2s-capture-dai-link {
                        link-name = "MI2S-LPAIF_SDR-TX-TERTIARY";

                        cpu {
                                sound-dai = <&q6apmbedai TERTIARY_
SDR_MI2S_TX>;
                        };

                        codec {
                                sound-dai = <&dmic_codec>;
                        };
                };
        };
```

QCS8275

The supported audio hardware interfaces:

- Connects Mercury Codec over MI2S interface
- Connects DAC to Mercury Codec
- Optionally connects other third-party codec or speaker amplifier over MI2S/TDM interfaces

The following figure shows the supported interfaces:

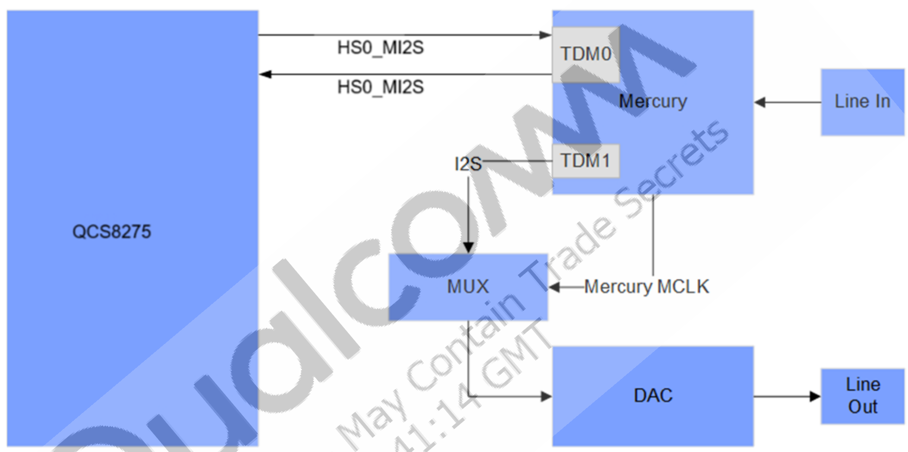


Figure6 Supported hardware interfaces

ALSA codec drivers

The codec class driver is a generic and hardware-independent ALSA compliant driver. It configures the codec and speaker amp to provide audio capture and playback.

Configure the device tree

The device tree is a data structure and language that describes hardware components on a System-on-Chip (SoC). It was created by the Open Firmware standards to unify the discovery of hardware devices connected to the SoC. The device tree abstracts hardware discovery from the Linux kernel source code, eliminating the need for hardcoded hardware information.

The device tree entries also have DAI links for PCM nodes on the SoC. The device tree defines the DAI links for PCM nodes. They are part of the sound node.

The following code shows this:

```
sound {
            compatible = "qcom,qcs8300-sndcard";
            model = "qcs8300-ridesx-snd-card";

            clocks = <&q6prmcc LPASS_HW_MACRO_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>,
                     <&q6prmcc LPASS_HW_DCODEC_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>;
            clock-names = "macro", "dcodec";

            pinctrl-names = "default", "stub_aif1_active",
"stub_aif1_sleep",
                            "stub_aif2_active", "stub_aif2_
sleep", "stub_aif3_active",
                            "stub_aif3_sleep", "stub_aif4_
active", "stub_aif4_sleep";
            pinctrl-0 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>, <&lpass_i2s1_clk_
sleep>,
                        <&lpass_i2s1_data_sleep>, <&lpass_i2s1_
ws_sleep>, <&mi2s_mclk_sleep>,
                        <&hs0_mi2s_data0_sleep>, <&hs0_mi2s_
sck_sleep>, <&hs0_mi2s_data1_sleep>,
                        <&hs0_mi2s_ws_sleep>;
            pinctrl-1 = <&mi2s1_data0>, <&mi2s1_data1>, <&
mi2s1_sck>, <&mi2s1_ws>;
            pinctrl-2 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>;
            pinctrl-3 = <&lpass_i2s1_clk>, <&lpass_i2s1_data>,
<&lpass_i2s1_ws>;
            pinctrl-4 = <&lpass_i2s1_clk_sleep>, <&lpass_i2s1_
data_sleep>,
                        <&lpass_i2s1_ws_sleep>;
            pinctrl-5 = <&hs0_mi2s_data0>, <&hs0_mi2s_data1>,
<&hs0_mi2s_sck>, <&mi2s_mclk>,
                        <&hs0_mi2s_ws>;
            pinctrl-6 = <&hs0_mi2s_data0_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_sck_sleep>,
                        <&mi2s_mclk_sleep>,<&hs0_mi2s_ws_sleep>
;
            pinctrl-7 = <&mi2s1_data0>, <&mi2s1_data1>, <&
mi2s1_sck>, <&mi2s_mclk>,
```

```
                        <&mi2s1_ws>;
            pinctrl-8 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>, <&mi2s_mclk_sleep>;

            tdm0-capture-dai-link {
                    link-name = "TDM-LPAIF_SDR-TX-PRIMARY";

                    cpu {
                            sound-dai = <&q6apmbedai PRIMARY_
SDR_TDM_TX_0>;
                    };

                    codec {
                            sound-dai = <&msm_stub_codec 5>;
                    };
            };

            tdm0-playback-dai-link {
                    link-name = "TDM-LPAIF_SDR-RX-PRIMARY";

                    cpu {
                            sound-dai = <&q6apmbedai PRIMARY_
SDR_TDM_RX_0>;
                    };

                    codec {
                            sound-dai = <&msm_stub_codec 4>;
                    };
            };
     };
```

3 Tools

3.1 QACT for audio calibration

QACT configures and calibrates audio use cases and audio software features.

Launcher View appears when starting QACT. It provides an interface to open qwsp/acdb files or connect to devices. QACT capabilities and presentation depend on the connected product.

Figure1 QACT Launcher View

QACT View helps you design graphs and tune modules. The following figure shows the QACT UI components.

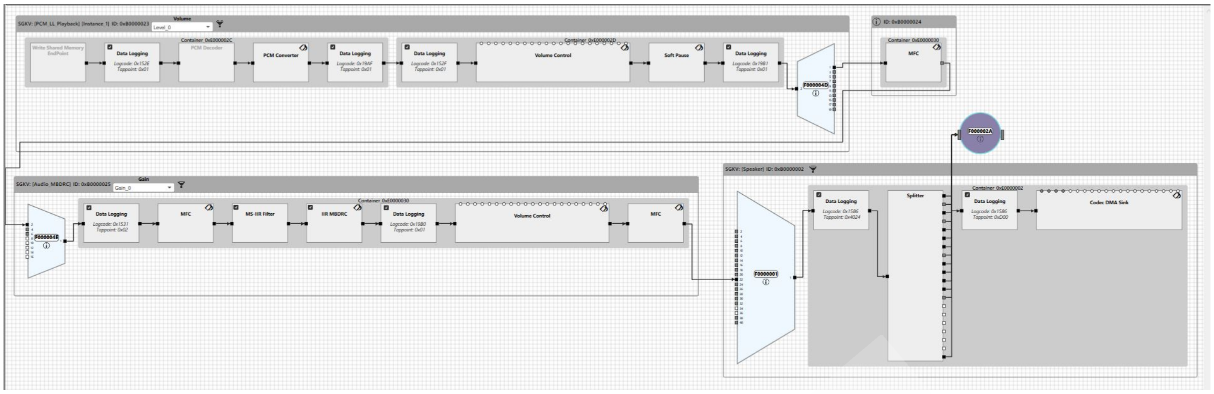


Figure2 QACT UI

Next steps

- To get started, download QACT.
- To learn more, see Tune audio.

3.2 QXDM for DMC diagnostics

QXDM Professional™ (QXDM Pro) provides a fast prototyping platform for new diagnostic clients. It provides a GUI to visualize data transmitted.

To enable logging, select *File -> Manage Configuration* and select the required use case DMC.

The following figure shows how to select and apply the required DMC log mask.

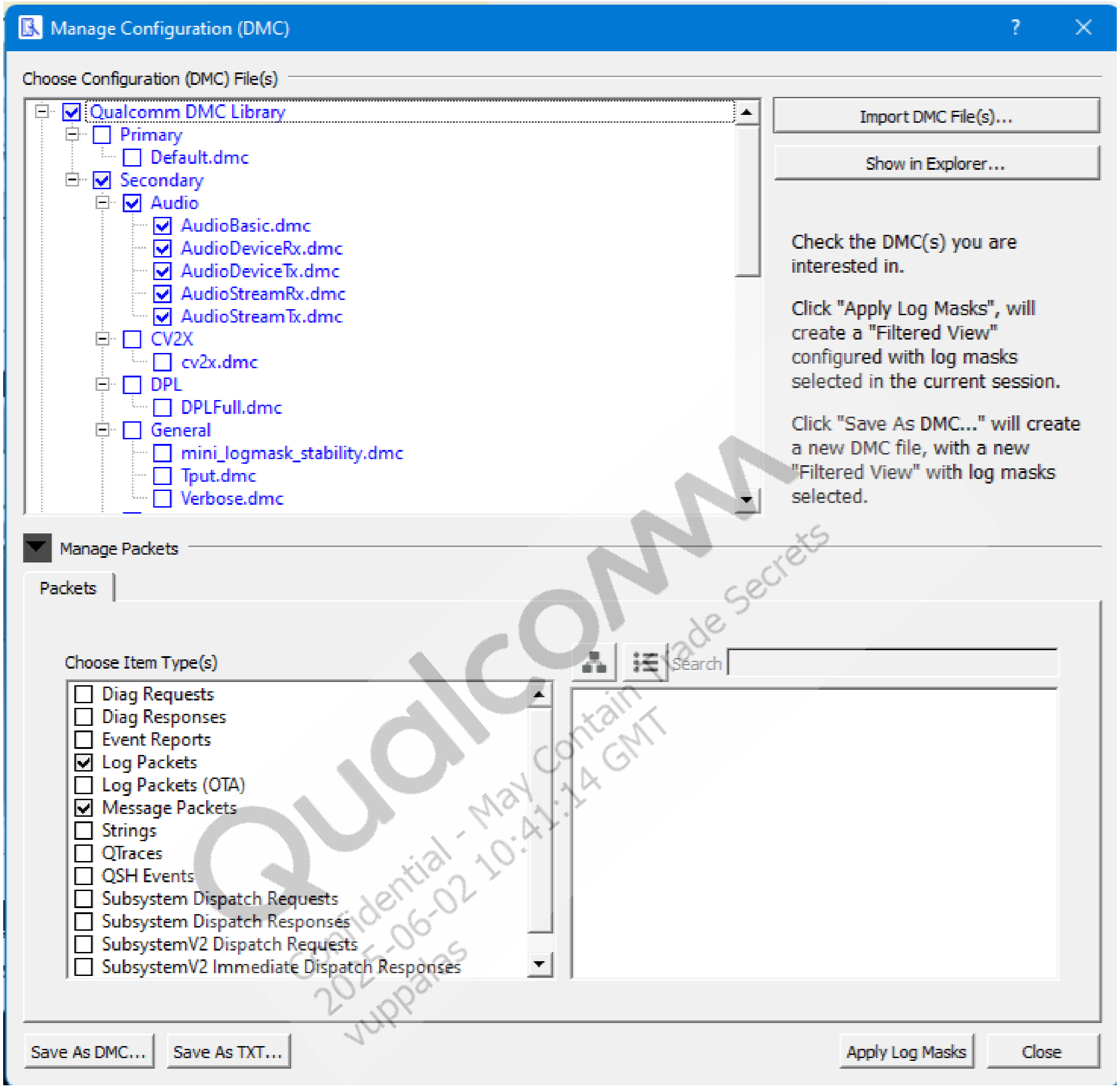


Figure3 QXDM DMC log mask

Next steps

- To get started, download QXDM.
- To learn more, see Tune audio.

4 Advanced audio customization

4.1 Sync and compile audio components

The audio software uses the user space and kernel space modules, which are in the Linux-enabled audio software directory.

The audio user space and kernel module source trees extract to the `<workspace>/build-qcom-wayland/workspace/sources` path.

The following steps show how to use the devtool Linux utility to get and extract the audio module source code and build the user space and kernel mode modules.

Note: Go to the workspace (`<workspace>/build-qcom-wayland$`) to access the source code trees using devtool.

Sync PulseAudio

1. Extract the source tree:

   ```
   devtool modify pulseaudio
   ```

   The PulseAudio source tree extracts to `build-qcom-wayland/workspace/sources/pulseaudio`.

2. Build the source tree:

   ```
   devtool build pulseaudio
   ```

Sync PAL

1. Extract the source tree:

```
devtool modify qcom-pal
```

   The PAL source tree extracts to `build-qcom-wayland/workspace/sources/qcom-pal/opensource/arpal-lx`.

2. Build the source tree:

```
devtool build qcom-pal
```

Sync TinyALSA

1. Extract the source tree:

```
devtool modify tinyalsa
```

   The TinyALSA source tree extracts to `build-qcom-wayland/workspace/sources/tinyalsa`.

2. Build the source tree:

```
devtool build tinyalsa
```

4.2 Configure MI2S/TDM interfaces

MI2S and TDM interfaces allow devices to connect and transfer audio data.

MI2S

MI2S is a serial bus interface that connects many digital audio devices. It's a simple data interface without any form of address or device selection.

MI2S consists of only one bus controller, one transmitter, and one receiver. The transmitter or receiver can be a bus controller. The MI2S bus carries two or more audio channels on the data lines. Each data line carries two-channel data, the left and right channels, which carry stereo audio data streams. The data alternates between the left and right channels, as controlled by a word select (WS) signal from the bus controller.

The MI2S feature supports:

MI2S support

Configurations	Controller and Target mode
Data formats	16-bit, 24-bit left-aligned, 32-bit
Sample rates	• 8, 16, 32, 44.1, 48, 88.2, 96, 176.4, 192, 352.8 and 384 kHz in Controller mode • All standard sample rates in Target mode
Bit clock	Maximum of 24.586 MHz
Serial data lines	Configurable serial data lines for Rx/Tx. Each data line can carry 2 channels of data (stereo data). The MI2S data line supports a maximum of 4 channels in Half-duplex mode (Rx or Tx), or supports stereo in Full-duplex mode.

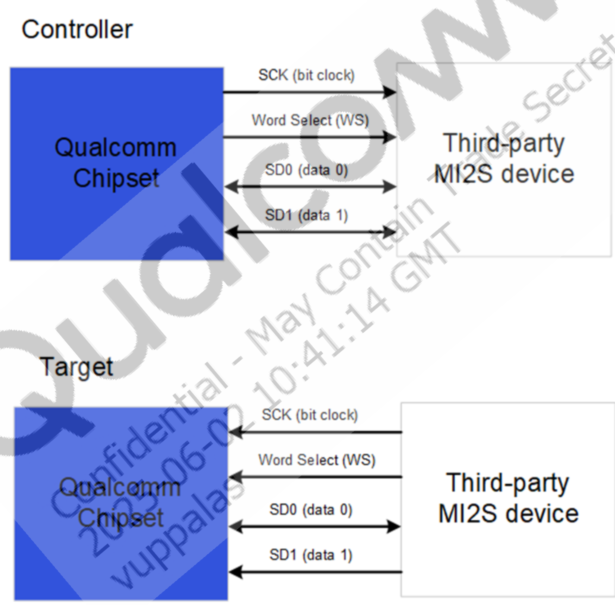


Figure1 MI2S interface

TDM

The TDM interface commonly transfers channels of audio data between devices in a system. The TDM interface has two control clocks: a frame synchronization pulse (FSYNC) and a serial clock (SClk), also known as the bit clock (BCLK). It also has two or more serial audio data lines (SDATA or SD).

The TDM feature supports:

TDM support

Configurations	Controller and Target mode
Data formats	16-bit, 24-bit left-aligned, 32-bit
Sample rates	• 8, 16, 32, 44.1, 48, 88.2, 96, 176.4, 192, 352.8 and 384 kHz in Controller mode • All standard sample rates in Target mode
Bit clock	Maximum of 24.586 MHz
Serial data lines	• Data sent per slot can be smaller or equal to slot size – Tx – If data size is less than the slot size, it pads the remaining bits with zeros – Rx – If data size is less than the slot size, it ignores all received data after data size • Any one of the transit or receive slots are available as the active slot - Maximum use of all 32 active slots per frame
Frame/Slot size	• 1 to 512 bits per frame • 1 to 32 bits per slot (width of serial data in and out can be different) • 1 to 32 slots per frame, dependent on the programmed bit rate and number of bits per slot

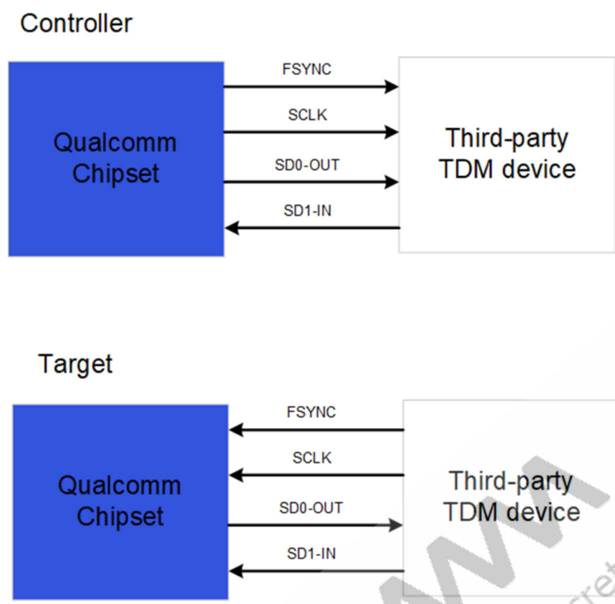


Figure2 TDM interface

4.3 MI2S/TDM interfaces

QCS6490

MI2S/TDM overview

MI2S is a serial bus interface that connects multiple digital audio devices. It is a simple data interface without any form of address or device selection.

The TDM interface transfers many channels of audio data between devices within a system.

The TDM interface has two control clocks:

- Frame synchronization pulse (FSYNC)
- Serial clock (SCLK), also known as the bit clock (BCLK)

The following figure shows the MI2S process flow:

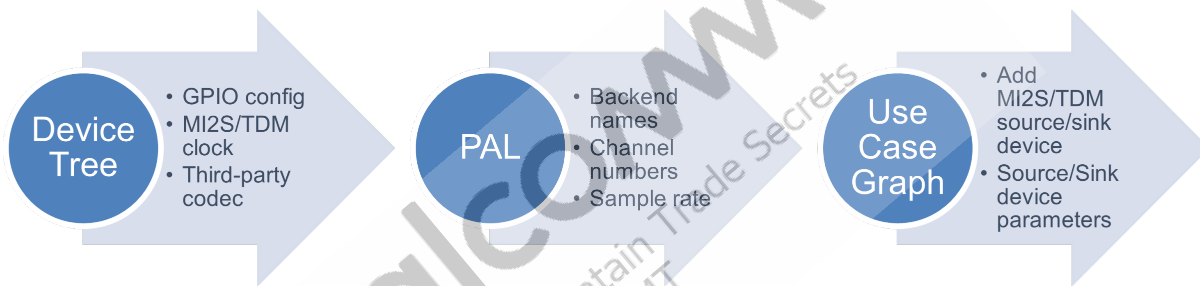


Figure3 MI2S flow

GPIOs to connect third-party devices

The MI2S interfaces use GPIOs for signal and data transmission. The following table lists the GPIOs that can connect third-party codec or speaker amps. Use these GPIOs even for TDM configuration.

GPIOs for MI2S configuration

Interface mapping	Backend name	TLMM GPIO	LPASS GPIO
Primary (mi2s0)	MI2S: • RX - MI2S-LPAIF-RX-PRIMARY • TX - MI2S-LPAIF-TX-PRIMARY TDM: • RX - TDM-LPAIF-RX-PRIMARY • TX - MI2S-LPAIF-TX-PRIMARY	• GPIO_97 CLK • GPIO_100 SYNC • GPIO_98 DATA0 • GPIO_99 DATA1	–
Secondary (MI2S1)	MI2S • RX - MI2S-LPAIF-RX-SECONDARY • TX - MI2S-LPAIF-TX-SECONDARY TDM • RX - TDM-LPAIF-RX-SECONDARY • TX - TDM-LPAIF-TX-SECONDARY	• GPIO_106 CLK • GPIO_108 SYNC • GPIO_107 DATA0 • GPIO_105 DATA1	–

Interface mapping	Backend name	TLMM GPIO	LPASS GPIO
Tertiary (MI2S2)	MI2S • RX -MI2S-LPAIF_AUD-RX-SECONDARY • TX - MI2S-LPAIF_AUD-TX-SECONDARY TDM: • RX - TDM-LPAIF_AUD-RX-SECONDARY • TX - TDM-LPAIF_AUD-TX-SECONDARY	• GPIO_101 CLK • GPIO_103 SYNC • GPIO_102 DATA0 • GPIO_104 DATA1	–
Quaternary (lpi-qua-mi2s)	MI2S • RX - MI2S-LPAIF_RXTX-RX-PRIMARY • TX - MI2S-LPAIF_RXTX-TX-PRIMARY TDM • RX - TDM-LPAIF_RXTX-RX-PRIMARY • TX - TDM-LPAIF_RXTX-TX-PRIMARY	• GPIO_144 CLK • GPIO_145 SYNC • GPIO_146 DATA0 • GPIO_147 DATA1 • GPIO_148 DATA2 • GPIO_149 DATA3	• LPASS_GPIO_0 CLK • LPASS_GPIO_1 SYNC • LPASS_GPIO_2 DATA0 • LPASS_GPIO_3 DATA1 • LPASS_GPIO_4 DATA2 • LPASS_GPIO_5 DATA3

Interface mapping	Backend name	TLMM GPIO	LPASS GPIO
Quinary (lpi-i2s1)	MI2S • RX - MI2S-LPAIF_VA-RX-PRIMARY • TX - MI2S-LPAIF_VA-TX-PRIMARY TDM • RX - TDM-LPAIF_VA-RX-PRIMARY • TX - TDM-LPAIF_VA-TX-PRIMARY	• GPIO_150 CLK • GPIO_151 SYNC • GPIO_152 DATA0 • GPIO_153 DATA1	• LPASS_GPIO_6 CLK • LPASS_GPIO_7 SYNC • LPASS_GPIO_8 DATA0 • LPASS_GPIO_9 DATA1

There are three MCLKs capable of generating up to 512 ×, 48 kHz (24.576 MHz) each. Despite what the GPIO alternate function name suggests, all three MCLK outputs are independent of I2S and HS-I2S interfaces. They can be used with all interfaces.

MCLKs for independent interface configuration

MCLK	TLMM GPIO	LPASS GPIO
pri_mi2s_mclk	GPIO_96	–
sec_mi2s_mclk	GPIO_105	–
EXT_MCLK1	GPIO_149 GPIO_153 GPIO_157	LPASS_GPIO_5 LPASS_GPIO_9 LPASS_GPIO_13

Set GPIO and clock configuration logic for MI2S

Configure device tree files

The machine driver enables/disables the MI2S GPIOs whenever capture starts over the MI2S interface.

Specify the MI2S GPIO configuration in the device tree file of the platform and its pinctrl file.

GPIO pinctrl entries are in the following files based on the form-factor of the platform:

- `arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dts`
- `arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2-video-mezz.dts`

- arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2-vision-mezz.dts

Example GPIO configuration for primary I2S

Add MI2S GPIO Active/Sleep configurations in the customer target .dtsi file.

Ensure that no other interface uses these MI2S interface GPIOs for another purpose. Remove all other entries from the device tree.

The following is an example configuration for the primary MI2S interface. Add similar configurations for the primary MI2S interfaces if the customer plans to use them.

```
&sound {
    pinctrl-names = "default", "stub_aif1_active", "stub_aif1_
sleep";
    pinctrl-0 = <&mi2s0_data0_sleep>, <&mi2s0_data1_sleep>, <&
mi2s0_mclk_sleep>,
            <&mi2s0_sclk_sleep>,<&mi2s0_ws_sleep>;
    pinctrl-1 = <&mi2s0_data0>, <&mi2s0_data1>, <&mi2s0_mclk>,
<&mi2s0_sclk>, <&mi2s0_ws>;
    pinctrl-2 = <&mi2s0_data0_sleep>, <&mi2s0_data1_sleep>, <&
mi2s0_mclk_sleep>,
            <&mi2s0_sclk_sleep>,<&mi2s0_ws_sleep>;
    mi2s-capture-dai-link {
        link-name = "MI2S-LPAIF-TX-PRIMARY";
        cpu {
            sound-dai = <&q6apmbedai PRIMARY_MI2S_TX>;
        };
        codec {
            sound-dai = <&msm_stub_codec 1>;
        };
          };

&tlmm {
       mi2s0_data0_sleep: mi2s0-data0-sleep {
          pins = "gpio98";
          function = "gpio";
          drive-strength = <2>;
          bias-pull-down;
          input-enable;
   };

   mi2s0_data1_sleep: mi2s0-data1-sleep {
          pins = "gpio99";
          function = "gpio";
          drive-strength = <2>;
```

```
            bias-pull-down;
            input-enable;
    };

    mi2s0_mclk_sleep: mi2s0-mclk-sleep {
            pins = "gpio96";
            function = "gpio";
            drive-strength = <2>;
            bias-pull-down;
            input-enable;
    };

    mi2s0_sclk_sleep: mi2s0-sclk-sleep {
            pins = "gpio97";
            function = "gpio";
            drive-strength = <2>;
            bias-pull-down;
            input-enable;
    };

    mi2s0_ws_sleep: mi2s0-ws-sleep {
            pins = "gpio100";
            function = "gpio";
            drive-strength = <2>;
            bias-pull-down;
            input-enable;
    };
};

&mi2s0_data0 {
            drive-strength = <8>;
            bias-disable;
};

&mi2s0_data1 {
             drive-strength = <8>;
             bias-disable;
};

&mi2s0_mclk {
             drive-strength = <8>;
             bias-disable;
             output-high;
};
```

```
&mi2s0_sclk {
          drive-strength = <8>;
          bias-disable;
          output-high;
};

&mi2s0_ws {
          drive-strength = <8>;
          bias-disable;
          output-high;
};
```

MI2S clock configuration logic – Controller mode

In Controller mode, the sample rate, number of channels, and the bit width configure the bit clock frequency.

For example, bit clock frequency = 48000 (Hz) x 2 (stereo) x 16 (bit width) = 1.536 MHz.

The Qualcomm chipset provides both the bit clock and WS clock. The bit clock sources the WS clock. The WS clock is set equal to the sample rate. WS doesn't require clock configuration.

MI2S clock configuration logic – Target mode

In Target mode, a third-party MI2S device provides both bit clock and WS clock. LPASS handles the MI2S interface clock control, number of channels, number of serial data lines, dataline direction bit-width, and sample rate. The apps processor reads the MI2S interface configuration from acdb and pushes it to LPASS while starting the use case over MI2S interface. For Target mode, in acdb, you must set the clock source for the bit clock as external.

By default, release builds have complete configurations for MI2S hardware interfaces. These changes must be present in the .dtsi files.

To enable the MI2S audio hardware interface:

1. Verify that the GPIOs map to the correct MI2S hardware interface
2. Remove GPIO configuration settings if another hardware module is using the same GPIOs as MI2S.

Modify the resource manager xml file for PAL configuration

Update the `resourcemanager.xml` file by replacing the existing codec device backend names. The following sections describe example file changes made to the `/etc/resourcemanager_qcm6490_rb3.xml` file on the device for specific use cases.

The following code shows the changed backend for the `PAL_DEVICE_IN_SPEAKER_MIC` device. It configures the channels as mono.

```
<id>PAL_DEVICE_IN_SPEAKER_MIC</id>
<back_end_name>MI2S-LPAIF-TX-PRIMARY</back_end_name>
<max_channels>4</max_channels>
<samplerate>48000</samplerate>
<channels>1</channels>
```

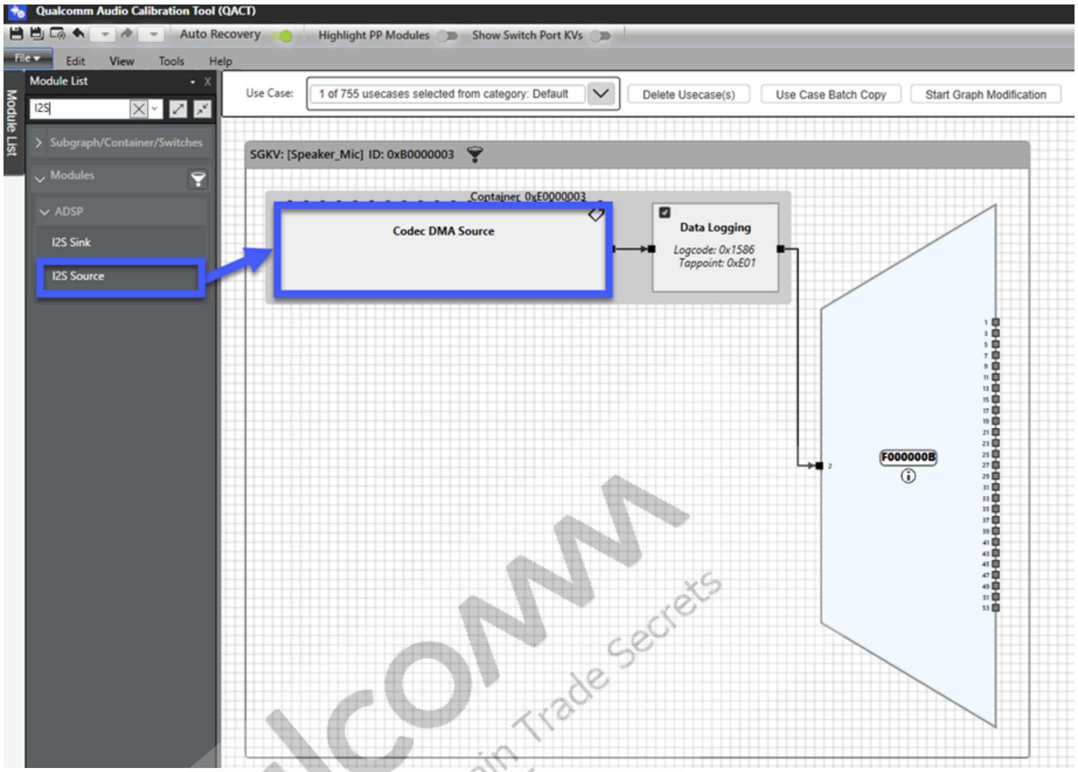
In this code snippet:

- `<max_channels>` – Maximum number of channels supported over MI2S
- `<channels>` – Default number of channels used while executing the use case.

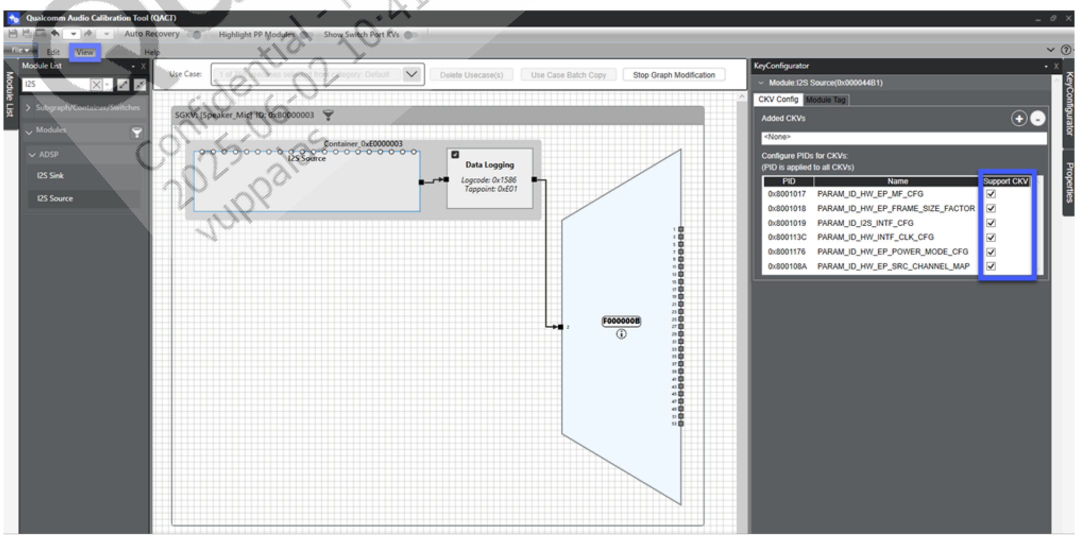
Modify the use case graph

After making the necessary changes in the `resourcemanager.xml` file, follow these steps in acdb to replace the WCD source devices with MI2S source devices. Note that replacing a source module in a device subgraph affects all use cases where the device is used.

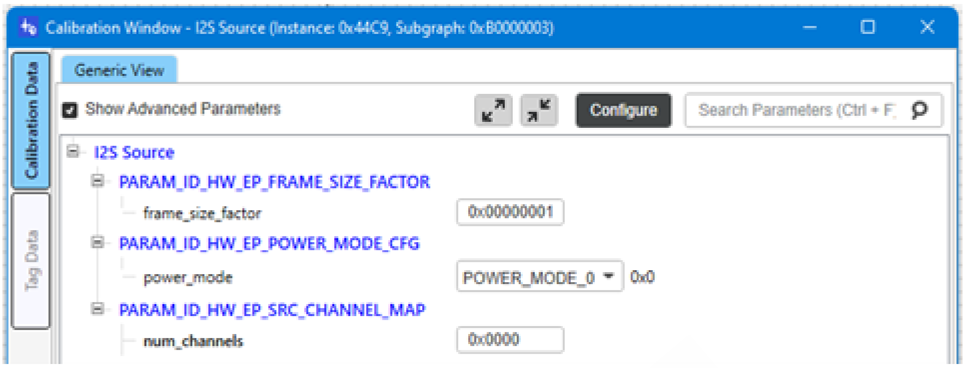
1. In QACT, navigate to a use case containing the device subgraph.
2. Select *Start Graph Modification*.
3. Choose the existing source module and then choose the DeviceTX subgraph.
4. Replace the code DMA Source with the I2S Source by dragging the I2S Source from the module list and dropping it onto the existing DMA Source.



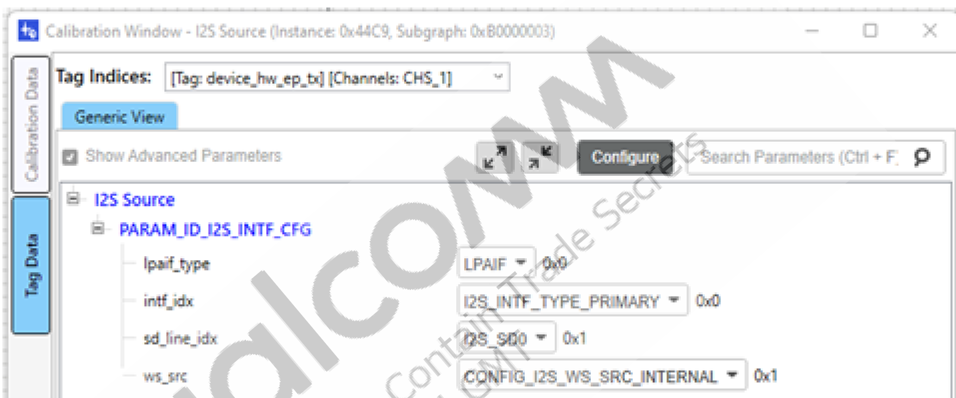
5. Select *View* → *Key Configurator*.
6. Select the *Support CKV`* checkboxes associated with the relevant CKVs.



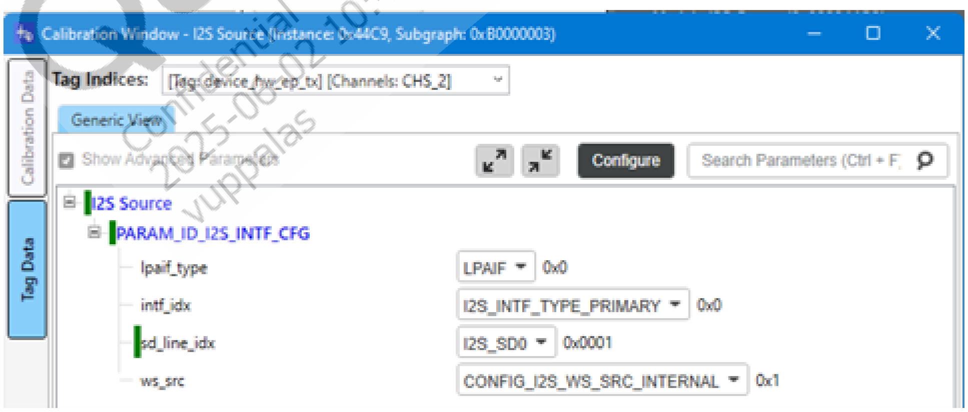
7. Double-tap the I2S source module to open the properties window. Ensure to properly configure the following parameters:
   - Calibration data should be set as follows:



- Tag data of the I2S Source for the mono channel should be set as follows:



- Tag data of the I2S Source for the stereo channel should be set as follows:



8. Assign the appropriate module tags. Typically, a module tag controls the channel count and corresponding channel mask of a source module. To modify module tags:
9. Ensure the I2S source is selected, Key Configurator is open, and the graph is still in the modification state.
10. In *Key Configurator*, select the *Module Tag* tab and click *+*.
11. In the *Select Tags* list, choose the correct module tag. In this example, it is `device_`

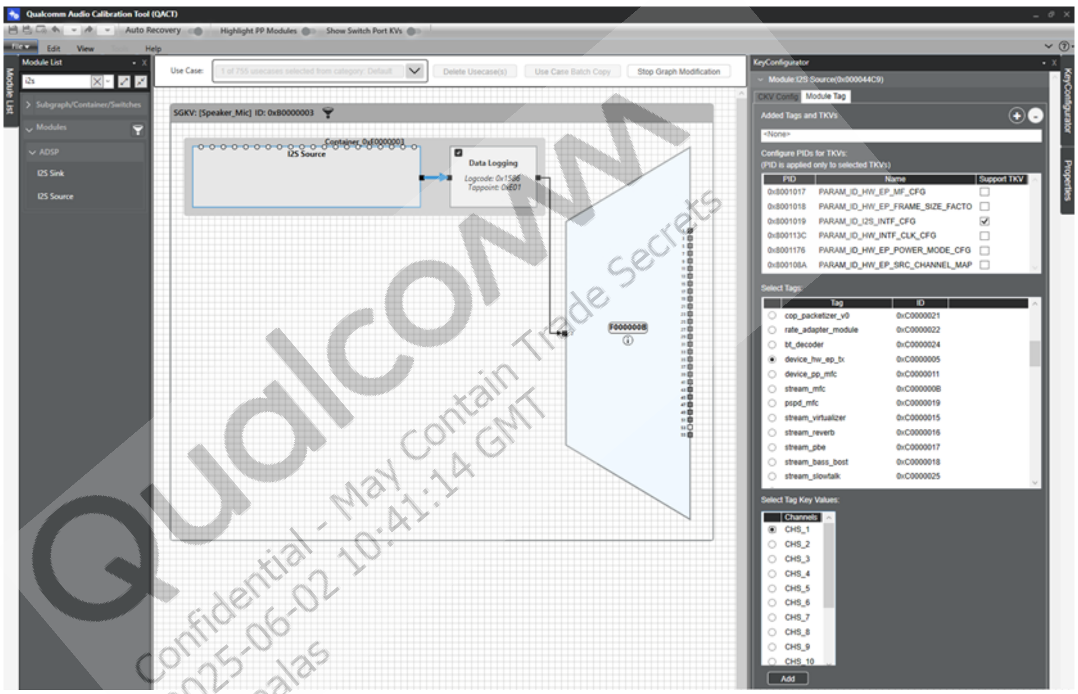
`hw_ep_tx`.

12. Once the tag is selected, a list of PIDs appears. In this list, select the *Support TKV* checkbox associated with the correct PID. In this example, it is `PARAM_ID_I2S_INTF_CFG`.

13. In the *Select Tag Key Values* list, select the correct tag key value. In this example, it is `CHS_1` for mono channel.

14. Select *Add*.

15. Repeat this process for CHS_2 for stereo channel support.



9. Once all channel configurations are added, select *Stop Graph Modification* and save the acdb files.

Set GPIO and clock configuration logic for TDM

Device tree configuration

The machine driver enables/disables the TDM GPIOs whenever playback or capture starts over the TDM interface.

Specify the TDM GPIO configuration in the device tree file of the platform and its pinctrl file.

GPIO pinctrl entries are in the following files based on the form-factor of the platform:

- `arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi`
- `arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2-video-mezz.dts`
- `arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2-vision-mezz.dts`

Example GPIO configuration for primary TDM

Add TDM GPIO Active/Sleep configurations in the customer target .dtsi file.

Ensure that no other interface uses these TDM interface GPIOs for another purpose. Remove all other entries from the device tree.

The following is an example configuration for the primary TDM interface. Add similar configurations for the primary TDM interfaces if the customer plans to use them.

```
&sound {

    pinctrl-names = "default", "stub_aif1_active", "stub_aif1_
sleep";
    pinctrl-0 = <&mi2s0_data0_sleep>, <&mi2s0_data1_sleep>, <&
mi2s0_mclk_sleep>,
                <&mi2s0_sclk_sleep>, <&mi2s0_ws_sleep>;
    pinctrl-1 = <&mi2s0_data0>, <&mi2s0_data1>, <&mi2s0_mclk>,
<&mi2s0_sclk>, <&mi2s0_ws>;
    pinctrl-2 = <&mi2s0_data0_sleep>, <&mi2s0_data1_sleep>, <&
mi2s0_mclk_sleep>,
                <&mi2s0_sclk_sleep>, <&mi2s0_ws_sleep>;
    tdm-capture-dai-link {
            link-name = "TDM-LPAIF-TX-PRIMARY";
            cpu {
                    sound-dai = <&q6apmbedai PRIMARY_TDM_TX_0>;
            };
            codec {
                    sound-dai = <&msm_stub_codec 1>;
            };
    };
```

```
    tdm-playback-dai-link {
            link-name = "TDM-LPAIF-RX-PRIMARY";
            cpu {
                    sound-dai = <&q6apmbedai PRIMARY_TDM_RX_0>;
            };
            codec {
                    sound-dai = <&msm_stub_codec 0>;
            };
    };
```

Modify the resource manager xml file for PAL configuration

Update the `resourcemanager.xml` file by replacing the existing codec device backend names. The following sections describe example file changes made to the `/etc/resourcemanager_qcm6490_idp.xml` file on the device for specific use cases.

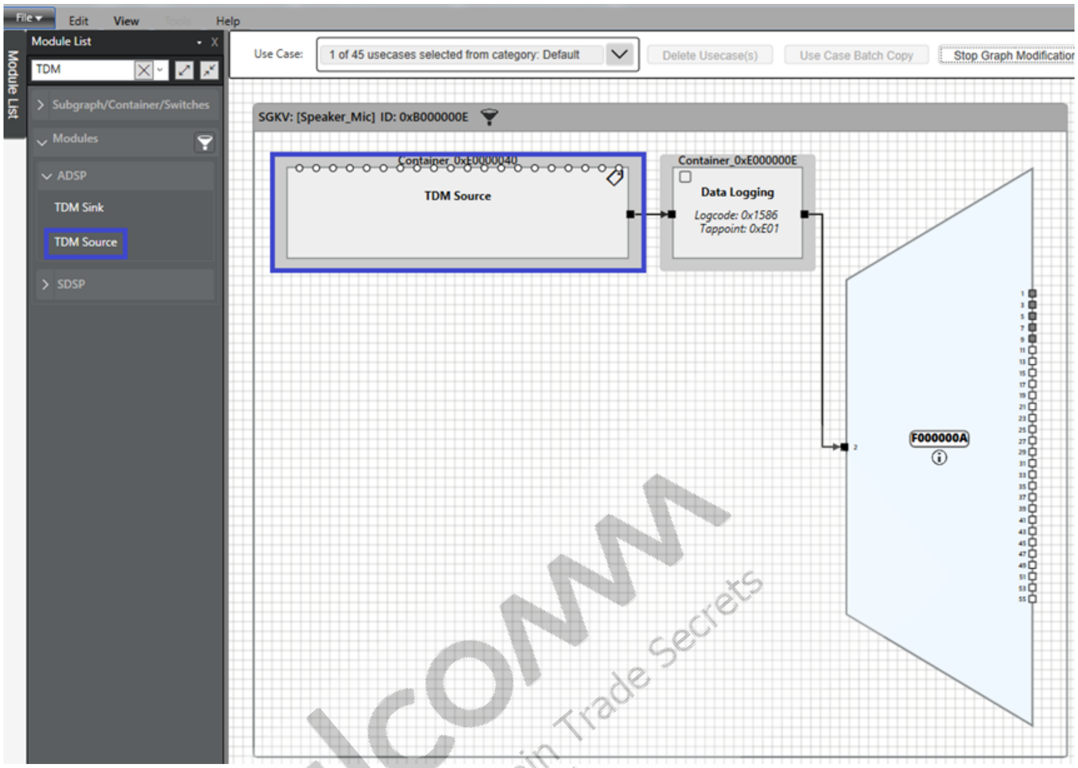
The following code shows the changed backend for the `PAL_DEVICE_IN_SPEAKER_MIC` device. It configures the channels as 2.

```
<id>PAL_DEVICE_IN_SPEAKER_MIC</id>
<back_end_name>TDM-LPAIF-TX-PRIMARY</back_end_name>
<max_channels>4</max_channels>
<samplerate>48000</samplerate>
<channels>2</channels>
```

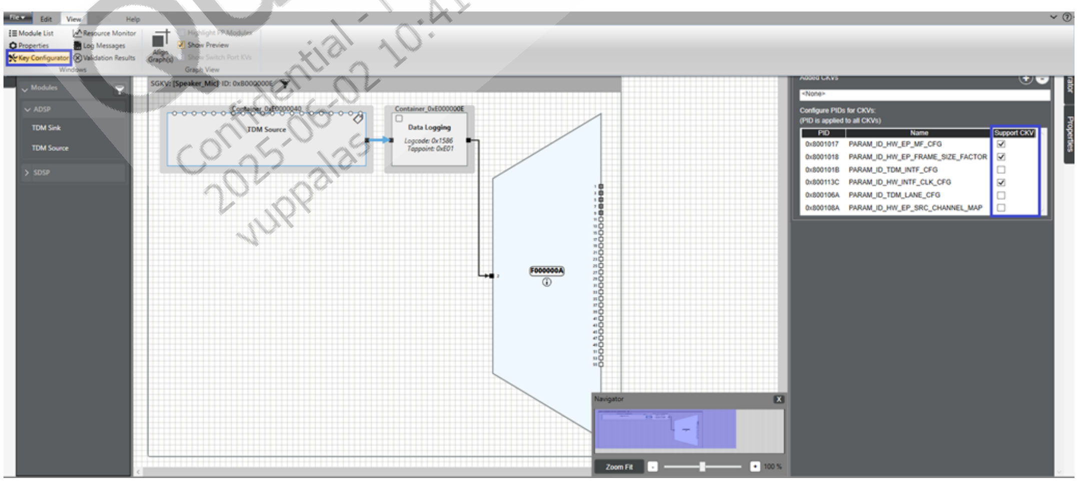
Modify the use case graph

After making the necessary changes in the `resourcemanager.xml` file, follow these steps to replace the source devices with TDM source devices in acdb. Note that replacing a source module in a device subgraph affects all use cases where the device is used.

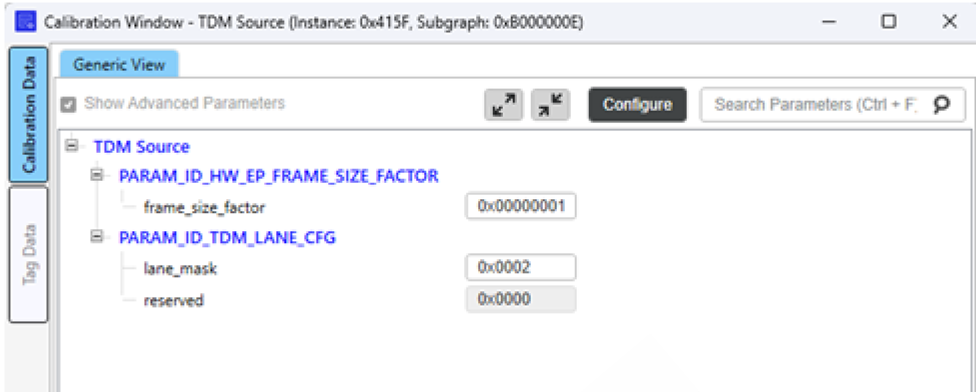
1. In QACT, navigate to a use case containing the desired device subgraph.
2. Select *Start Graph Modification*.
3. Choose the existing source module and then choose the DeviceTX subgraph.
4. Replace the Source with the TDM Source by dragging the TDM Source from the module list and dropping it onto the existing Source.

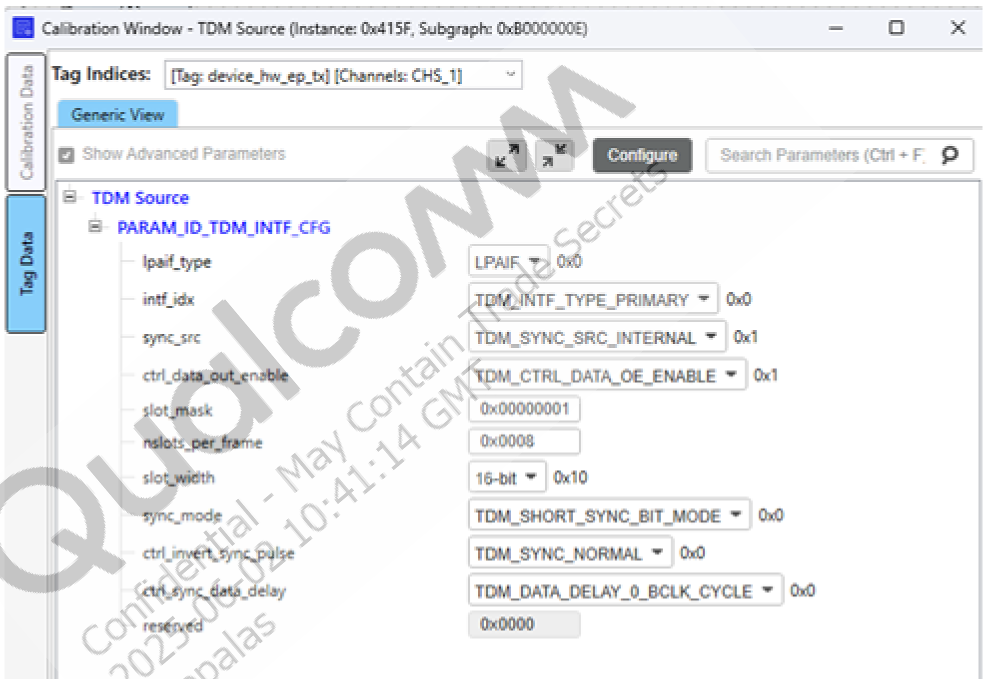


5. Select *View* → *Key Configurator*.
6. Select the *Support CKV* checkboxes associated with the relevant CKVs.



7. Select the Module Tag associated checkbox with relevant tags.
8. Double-tap the TDM source module to open the properties window. Ensure to properly configure the following parameters:
    - Calibration data and tag data should be configured as follows:





9. Assign the appropriate module tags. Typically, a module tag controls the channel count and corresponding channel mask of a source module. To modify module tags:

10. Ensure the TDM source is selected, Key Configurator is open, and the graph is still in the modification state.

11. In *Key Configurator*, select the *Module Tag* tab and click +.

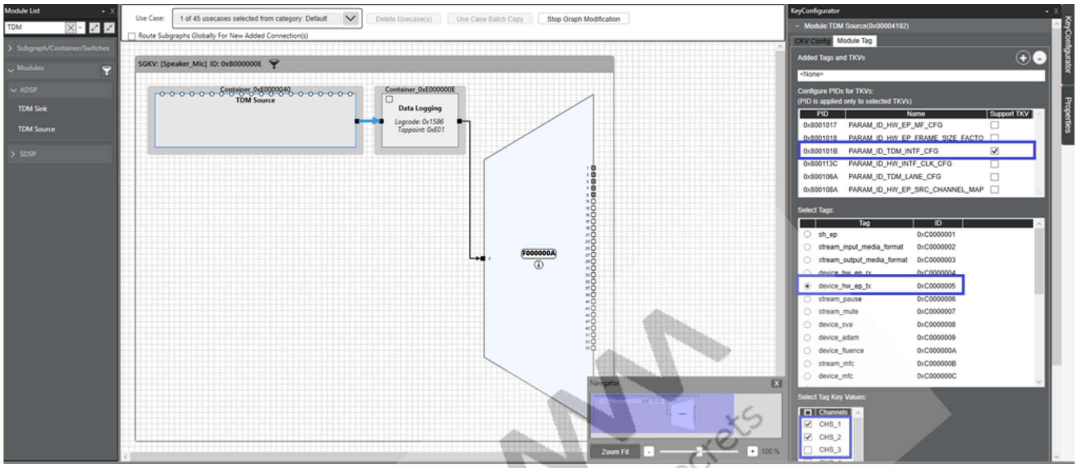
12. In the *Select Tags* list, select the correct module tag. In this example, it is `device_hw_ep_tx`.

13. Once the tag is selected, a list of PIDs appears. In this list, select the *Support TKV* checkbox associated with the correct PID. In this example, it is `PARAM_ID_TDM_INTF_CFG`.

14. In the *Select Tag Key Values* list, select the correct tag key value. In this example, it is

`CHS_1` for mono channel.

15. Select *Add*.
16. Repeat this process for CHS_2 for stereo channel support.



10. Once all channel configurations are added, click *Stop Graph Modification* and save the acdb files.

Troubleshoot and validate MIS2/TDM

This section checks that the sound card is properly registered. It also verifies I2S is recording from the PulseAudio level.

Note: Connect to the device console using SSH. See Use SSH for instructions.

Verify MI2S/TDM source capture

Check that the newly-added backend DAI registers the sound card:

```
cat /proc/asound/pcm
```

```
00-05: MI2S-LPAIF-TX-PRIMARY msm-stub-aif1-tx-5 :  : capture 1
```

If the backend DAI doesn't appear in the backend list, there may have been an issue adding the DAI links. Check the kernel logs to make sure there were no errors related to DAI links.

For TDM, the backend name will be `TDM-LPAIF-TX-PRIMARY`.

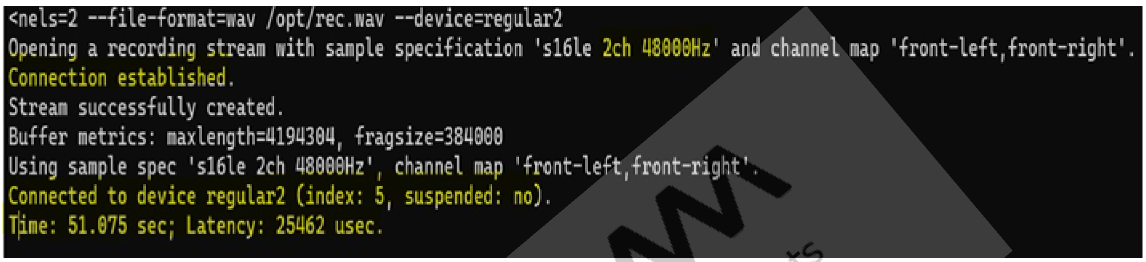
Verify the device is recording from the PulseAudio level. This example is for a 48 kHz, stereo, 16-bit recording.

1. Update the *channels tab* in the `resourcemanager.xml` file to *2*.

2. Reboot the target.
3. Run the following command from a shell prompt. It should generate a file at `/opt/rec.wav` with the audio sent from the I2S device.

```
parec -v --rate=48000 --format=s16le --channels=2 --file-
format=wav /opt/rec.wav --device=regular2
```

The command shell should show a message similar to the following:



```
<nels=2 --file-format=wav /opt/rec.wav --device=regular2
Opening a recording stream with sample specification 's16le 2ch 48000Hz' and channel map 'front-left,front-right'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, fragsize=384000
Using sample spec 's16le 2ch 48000Hz', channel map 'front-left,front-right'.
Connected to device regular2 (index: 5, suspended: no).
Time: 51.075 sec; Latency: 25462 usec.
```

Figure4 Command shell example for verifying PulseAudio recording

Analyze logs

When capture for 48k/stereo/16-bit is done, the following logs should be seen.

- The I2S source has been modified for the speaker_mic device. The device open should be called for the speaker mic as follows:

```
Apr 29 15:50:46 pulseaudio[1001]: open: 469: Enter.
deviceCount 0 for device id 28 (PAL_DEVICE_IN_SPEAKER_MIC)
```

- The backend used for the capture is MI2S-LPAIF-TX-PRIMARY, with SR as 48000 kHz, and the number of channels as 2:

```
Apr 29 15:50:46 pulseaudio[1001]: setDeviceMediaConfig: 1056:
MI2S-LPAIF-TX-PRIMARY rate ch fmt data_fmt 48000 2 2 1
```

- The device open should exit with status 0 and a proper deviceCount for the speaker mic device:

```
Apr 29 15:50:46 pulseaudio[1001]: open: 504: Exit.
deviceCount 1 for device id 28 (PAL_DEVICE_IN_SPEAKER_MIC),
exit status: 0
```

- The hardware endpoints should be configured properly for 48k/stereo/16-bit as follows:

```
Apr 29 15:50:46 pulseaudio[1001]: configure_hw_ep_media_
config: 664 rate 48000 bw 16 ch 2, data_fmt 1
```

QCS9075

MI2S/TDM interfaces

MI2S/TDM overview

MI2S is a serial bus interface that connects multiple digital audio devices. It is a simple data interface without any form of address or device selection.

The TDM interface transfers many channels of audio data between devices within a system.

The TDM interface has two control clocks:

- Frame synchronization pulse (FSYNC)
- Serial clock (SCLK), also known as the bit clock (BCLK)

The following figure shows the MI2S process flow:

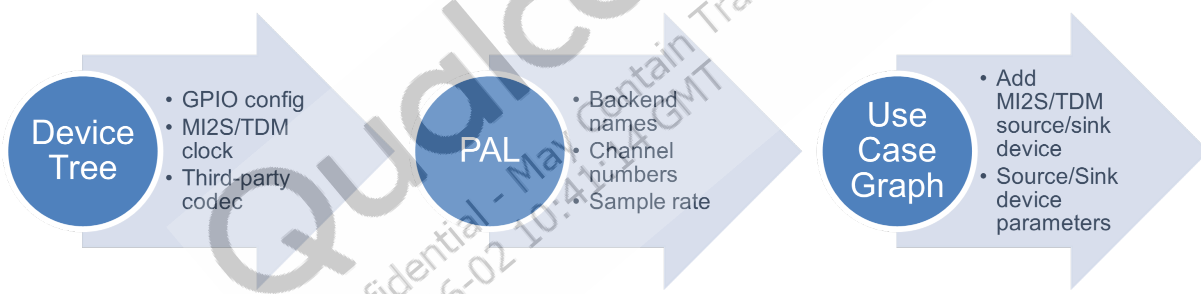


Figure5 MI2S flow

GPIOs to connect third-party devices

The MI2S interfaces use GPIOs for signal and data transmission. The following table lists the GPIOs that can connect third-party codec or speaker amps. Use these GPIOs even for TDM configuration.

GPIOs for MI2S configuration

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Primary (LPI-LS-MI2S4)	MI2S: • RX - MI2S-LPAIF-RX-PRIMARY • TX - MI2S-LPAIF-TX-PRIMARY TDM: • RX - TDM-LPAIF-RX-PRIMARY • TX - TDM-LPAIF-TX-PRIMARY	• GPIO_141 CLK • GPIO_142 SYNC • GPIO_143 DATA0 • GPIO_144 DATA1	• LPASS_GPIO_12 • LPASS_GPIO_13 • LPASS_GPIO_17 • LPASS_GPIO_18
Secondary (LS-MI2S1)	MI2S • RX - MI2S-LPAIF-RX-SECONDARY • TX - MI2S-LPAIF-TX-SECONDARY TDM • RX - TDM-LPAIF-RX-SECONDARY • TX - TDM-LPAIF-TX-SECONDARY	• GPIO_106 CLK • GPIO_107 SYNC • GPIO_108 DATA0 • GPIO_109 DATA1	–

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Tertiary (LS-MI2S2)	MI2S • RX - MI2S-LPAIF-RX-TERTIARY • TX - MI2S-LPAIF-TX-TERTIARY TDM: • RX - TDM-LPAIF-RX-TERTIARY • TX - TDM-LPAIF-TX-TERTIARY	• GPIO_110 CLK • GPIO_111 SYNC • GPIO_112 DATA0 • GPIO_113 DATA1	–
Quaternary (LPI-LS-MI2S0)	MI2S • RX - MI2S-LPAIF_RXTX-RX-PRIMARY • TX - MI2S-LPAIF_RXTX-TX-PRIMARY TDM • RX - TDM-LPAIF_RXTX-RX-PRIMARY • TX - TDM-LPAIF_RXTX-TX-PRIMARY	• GPIO_126 CLK • GPIO_127 SYNC • GPIO_128 DATA0 • GPIO_129 DATA1 • GPIO_130 DATA2 • GPIO_131 DATA3	• LPASS_GPIO_0 • LPASS_GPIO_1 • LPASS_GPIO_2 • LPASS_GPIO_3 • LPASS_GPIO_4 • LPASS_GPIO_5

Audio interface mapping	**Backend name**	**TLMM GPIOs**	**LPASS GPIOs**
Quinary (LPI-LS-MI2S1)	MI2S • RX - MI2S-LPAIF_VA-RX-PRIMARY • TX - MI2S-LPAIF_VA-TX-PRIMARY TDM • RX - TDM-LPAIF_VA-RX-PRIMARY • TX - TDM-LPAIF_VA-TX-PRIMARY	• GPIO_132 CLK • GPIO_133 SYNC • GPIO_134 DATA0 • GPIO_135 DATA1	• LPASS_GPIO_6 • LPASS_GPIO_7 • LPASS_GPIO_8 • LPASS_GPIO_9
Senary (LPI-LS-MI2S2)	MI2S • RX - MI2S-LPAIF_WSA-RX-PRIMARY • TX - MI2S-LPAIF_WSA-TX-PRIMARY TDM • RX - TDM-LPAIF_WSA-RX-PRIMARY • TX - TDM-LPAIF_WSA-TX-PRIMARY	• GPIO_136 CLK • GPIO_137 SYNC • GPIO_138 DATA0 • GPIO_139 DATA1	• LPASS_GPIO_10 • LPASS_GPIO_11 • LPASS_GPIO_15 • LPASS_GPIO_16

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Septenary (LPI-LS-MI2S3)	MI2S • RX - MI2S-LPAIF_AUD-RX-PRIMARY • TX - MI2S-LPAIF_AUD-TX-PRIMARY TDM • RX - TDM-LPAIF_AUD-RX-PRIMARY • TX - TDM-LPAIF_AUD-TX-PRIMARY	• GPIO_145 CLK • GPIO_146 SYNC • GPIO_147 DATA0 • GPIO_148 DATA1	• LPASS_GPIO_19 • LPASS_GPIO_20 • LPASS_GPIO_21 • LPASS_GPIO_22
HS_IF0 (HS-MI2S0)	MI2S • RX - MI2S-LPAIF_SDR-RX-PRIMARY • TX - MI2S-LPAIF_SDR-TX-PRIMARY TDM • RX - TDM-LPAIF_SDR-RX-PRIMARY • TX - TDM-LPAIF_SDR-TX-PRIMARY	• GPIO_114 CLK • GPIO_115 SYNC • GPIO_116 DATA0 • GPIO_117 DATA1	–

Audio interface mapping	**Backend name**	**TLMM GPIOs**	**LPASS GPIOs**
HS_IF1 (HS-MI2S1)	MI2S • RX - MI2S-LPAIF_SDR-RX-SECONDARY • TX - MI2S-LPAIF_SDR-TX-SECONDARY TDM • RX - TDM-LPAIF_SDR-RX-SECONDARY • TX - TDM-LPAIF_SDR-TX-SECONDARY	• GPIO_118 CLK • GPIO_119 SYNC • GPIO_120 DATA0 • GPIO_121 DATA1	–
HS_IF2 (HS-MI2S2)	MI2S • RX - MI2S-LPAIF_SDR-RX-TERTIARY • TX - MI2S-LPAIF_SDR-TX-TERTIARY TDM • RX - TDM-LPAIF_SDR-RX-TERTIARY • TX - TDM-LPAIF_SDR-TX-TERTIARY	• GPIO_122 CLK • GPIO_123 SYNC • GPIO_124 DATA0 • GPIO_125 DATA1	–

There are three MCLKs capable of generating up to 512 $\times$, 48 kHz (24.576 MHz) each. Despite what the GPIO alternate function name suggests, all three MCLK outputs are independent of I2S and HS-I2S interfaces. They can be used with all interfaces.

MCLKs for independent interface configuration

MCLK	GPIO
MI2S_MCLK0	GPIO_105
MI2S_MCLK1	GPIO_117
EXT_MCLK1	GPIO_131 GPIO_135 GPIO_142 GPIO_140 GPIO_148

Set GPIO and clock configuration logic for MI2S

Configure device tree files

The machine driver enables/disables the MI2S GPIOs whenever capture starts over the MI2S interface.

Specify the MI2S GPIO configuration in the device tree file of the platform and its pinctrl file.

GPIO pinctrl entries are in the following files based on the form-factor of the platform:

- `arch/arm64/boot/dts/qcom/qcs9075-addons-rb8.dtsi`

Example GPIO configuration for primary I2S

Add MI2S GPIO Active/Sleep configurations in the customer target .dtsi file.

Ensure that no other interface uses these MI2S interface GPIOs for another purpose. Remove all other entries from the device tree.

The following is an example configuration for the primary MI2S interface. Add similar configurations for the primary MI2S interfaces if the customer plans to use them.

```
&sound {
            compatible = "qcom,qcs9075-rb8-sndcard";
            model = "qcs9075-rb8-snd-card";

            clocks = <&q6prmcc LPASS_HW_MACRO_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>,
                        <&q6prmcc LPASS_HW_DCODEC_VOTE LPASS_
CLK_ATTRIBUTE_COUPLE_NO>;
            clock-names = "macro", "dcodec";

            pinctrl-names = "default", "mi2s_aud_out_active",
"mi2s_aud_out_sleep";
            pinctrl-0 = <&hs1_mi2s_data0_sleep>, <&mi2s_mclk_
sleep>, <&hs0_mi2s_data0_sleep>,
                        <&hs0_mi2s_sclk_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_ws_sleep>,
```

```
                        <&hs1_mi2s_sclk_sleep>, <&hs1_mi2s_
data1_sleep>, <&hs1_mi2s_ws_sleep>,
                        <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                        <&hs2_mi2s_ws_sleep>, <&lpass_quad_clk_
sleep>, <&lpass_quad_data_sleep>,
                        <&lpass_quad_ws_sleep>;
            pinctrl-1 = <&hs1_mi2s_data0>, <&mi2s_mclk>, <&hs0_
mi2s_data0>,
                        <&hs0_mi2s_sclk>, <&hs0_mi2s_data1>, <&
hs0_mi2s_ws>,
                        <&hs1_mi2s_sclk>, <&hs1_mi2s_data1>, <&
hs1_mi2s_ws>,
                        <&hs2_mi2s_data0>, <&hs2_mi2s_data1>, <&
hs2_mi2s_sck>,
                        <&hs2_mi2s_ws>, <&lpass_quad_clk>, <&
lpass_quad_data>,
                        <&lpass_quad_ws>;
            pinctrl-2 = <&hs1_mi2s_data0_sleep>, <&mi2s_mclk_
sleep>, <&hs0_mi2s_data0_sleep>,
                        <&hs0_mi2s_sclk_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_ws_sleep>,
                        <&hs1_mi2s_sclk_sleep>, <&hs1_mi2s_
data1_sleep>, <&hs1_mi2s_ws_sleep>,
                        <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                        <&hs2_mi2s_ws_sleep>, <&lpass_quad_clk_
sleep>, <&lpass_quad_data_sleep>,
                        <&lpass_quad_ws_sleep>;
          lpi-mi2s1-capture-dai-link {
                  link-name = "MI2S-LPAIF_VA-TX-PRIMARY";
                  cpu {
                          sound-dai = <&q6apmbedai PRIMARY_
MI2S_TX>;
                  }:
             };
};
```

MI2S clock configuration logic – Controller mode

In Controller mode, the sample rate, number of channels, and the bit width configure the bit clock frequency.

For example, bit clock frequency = 48000 (Hz) x 2 (stereo) x 16 (bit width) = 1.536 MHz.

The Qualcomm chipset provides both the bit clock and WS clock. The bit clock sources the WS clock. The WS clock is set equal to the sample rate. WS doesn't require clock configuration.

MI2S clock configuration logic – Target mode

In Target mode, a third-party MI2S device provides both bit clock and WS clock. LPASS handles the MI2S interface clock control, number of channels, number of serial data lines, dataline direction bit-width, and sample rate. The apps processor reads the MI2S interface configuration from acdb and pushes it to LPASS while starting the use case over MI2S interface. For Target mode, in acdb, you must set the clock source for the bit clock as external.

By default, release builds have complete configurations for MI2S hardware interfaces. These changes must be present in the .dtsi files.

To enable the MI2S audio hardware interface:

1. Verify that the GPIOs map to the correct MI2S hardware interface
2. Remove GPIO configuration settings if another hardware module is using the same GPIOs as MI2S.

Modify the resource manager xml file for PAL configuration

Update the `resourcemanager.xml` file by replacing the existing codec device backend names. The following sections describe example file changes made to the `/etc/resourcemanager_qcs9075-rb8.xml` file on the device for specific use cases.

The following code shows the changed backend for the `PAL_DEVICE_IN_SPEAKER_MIC` device. It configures the channels as mono.

```
<id>PAL_DEVICE_IN_SPEAKER_MIC</id>
<back_end_name>MI2S-LPAIF-TX-PRIMARY</back_end_name>
<max_channels>4</max_channels>
<samplerate>48000</samplerate>
<channels>1</channels>
```

In this code snippet:

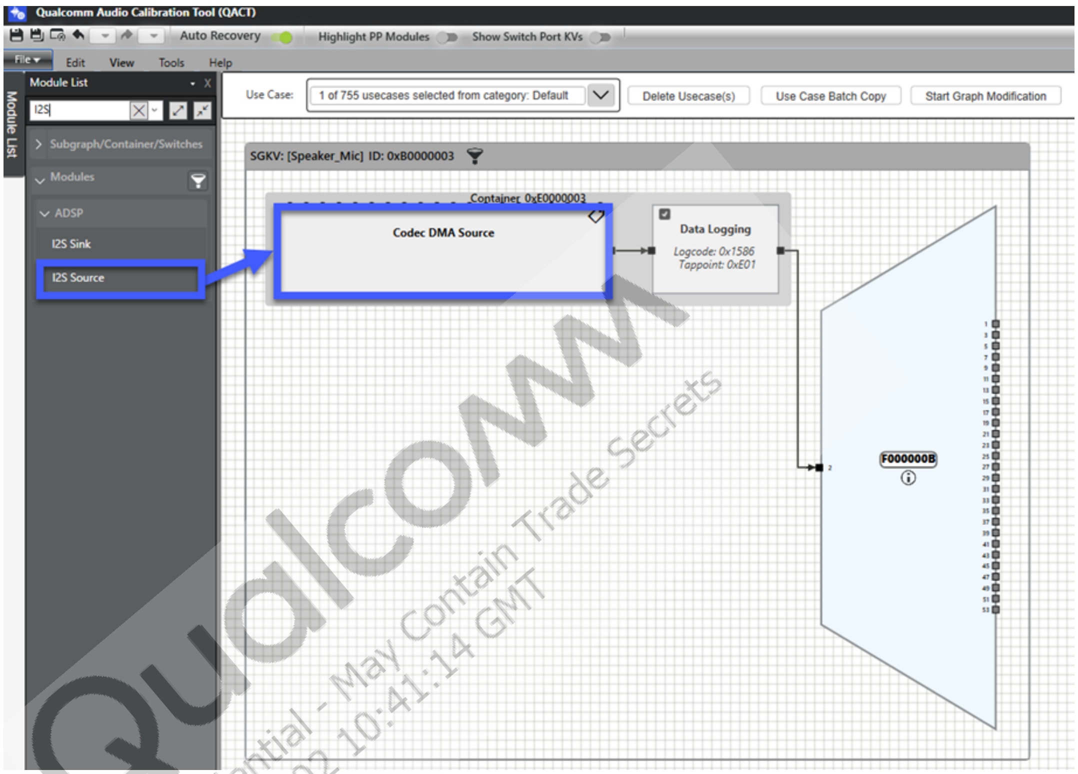
- `<max_channels>` – Maximum number of channels supported over MI2S
- `<channels>` – Default number of channels used while executing the use case.

Modify the use case graph

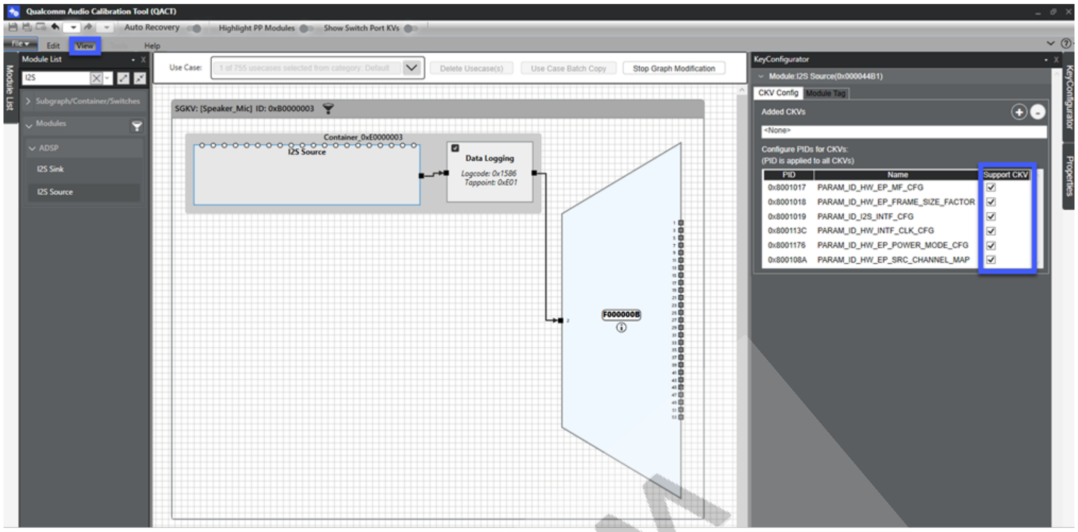
After making the necessary changes in the `resourcemanager.xml` file, follow these steps in acdb to replace the WCD source devices with MI2S source devices. Note that replacing a source module in a device subgraph affects all use cases where the device is used.

1. In QACT, navigate to a use case containing the device subgraph.

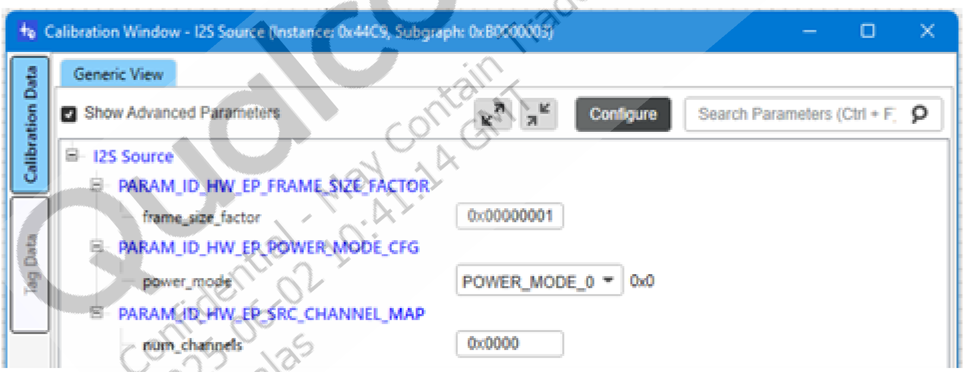
2. Select *Start Graph Modification*.
3. Choose the existing source module and then choose the DeviceTX subgraph.
4. Replace the code DMA Source with the I2S Source by dragging the I2S Source from the module list and dropping it onto the existing DMA Source.



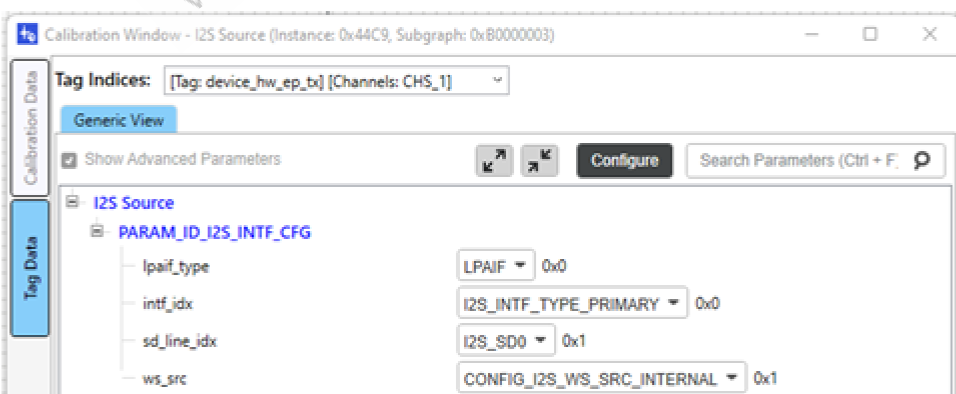
5. Select *View* → *Key Configurator*.
6. Select the *Support CKV* checkboxes associated with the relevant CKVs.



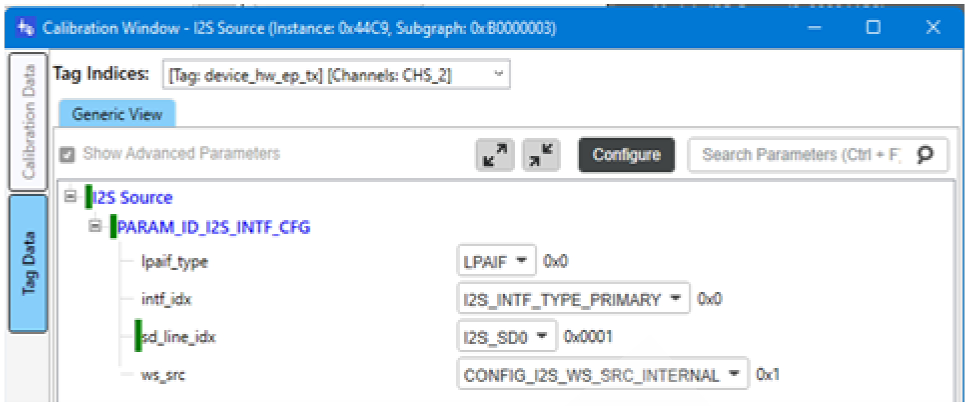
7. Double-tap the I2S source module to open the properties window. Ensure to properly configure the following parameters:
   - Calibration data should be set as follows:



   - Tag data of the I2S Source for the mono channel should be set as follows:



   - Tag data of the I2S Source for the stereo channel should be set as follows:



8. Assign the appropriate module tags. Typically, a module tag controls the channel count and corresponding channel mask of a source module. To modify module tags:

9. Ensure the I2S source is selected, Key Configurator is open, and the graph is still in the modification state.

10. In *Key Configurator*, select the *Module Tag* tab and click +.

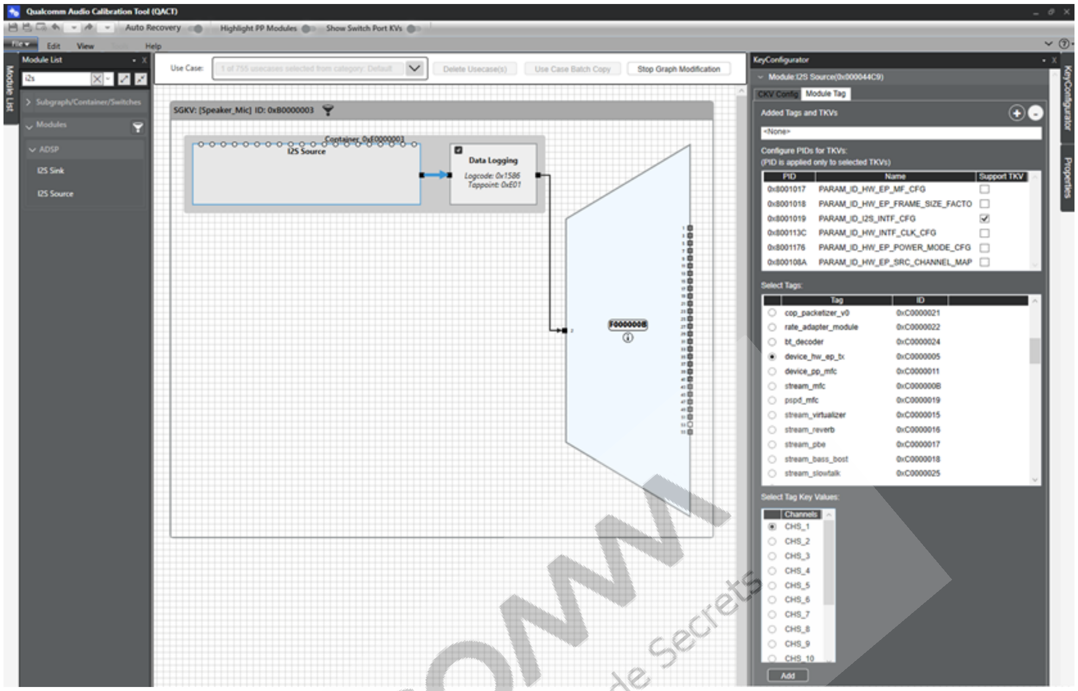
11. In the *Select Tags* list, choose the correct module tag. In this example, it is `device_hw_ep_tx`.

12. Once the tag is selected, a list of PIDs appears. In this list, select the *Support TKV* checkbox associated with the correct PID. In this example, it is `PARAM_ID_I2S_INTF_CFG`.

13. In the *Select Tag Key Values* list, select the correct tag key value. In this example, it is `CHS_1` for mono channel.

14. Select *Add*.

15. Repeat this process for CHS_2 for stereo channel support.



9. Once all channel configurations are added, select *Stop Graph Modification* and save the acdb files.

Set GPIO and clock configuration logic for TDM

Device tree configuration

The machine driver enables/disables the TDM GPIOs whenever playback or capture starts over the TDM interface.

Specify the TDM GPIO configuration in the device tree file of the platform and its pinctrl file.

GPIO pinctrl entries are in the following files based on the form-factor of the platform:

- `arch/arm64/boot/dts/qcom/qcs9075-addons-rb8.dtsi`

Example GPIO configuration for primary TDM

Add TDM GPIO Active/Sleep configurations in the customer target .dtsi file.

Ensure that no other interface uses these TDM interface GPIOs for another purpose. Remove all other entries from the device tree.

The following is an example configuration for the primary TDM interface. Add similar configurations for the primary TDM interfaces if the customer plans to use them.

```
&sound {
        compatible = "qcom,qcs9075-rb8-sndcard";
```

```
            model = "qcs9075-rb8-snd-card";

            clocks = <&q6prmcc LPASS_HW_MACRO_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>,
                        <&q6prmcc LPASS_HW_DCODEC_VOTE LPASS_
CLK_ATTRIBUTE_COUPLE_NO>;
            clock-names = "macro", "dcodec";

            pinctrl-names = "default", "mi2s_aud_out_active",
"mi2s_aud_out_sleep";
            pinctrl-0 = <&hs1_mi2s_data0_sleep>, <&mi2s_mclk_
sleep>, <&hs0_mi2s_data0_sleep>,
                        <&hs0_mi2s_sclk_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_ws_sleep>,
                        <&hs1_mi2s_sclk_sleep>, <&hs1_mi2s_
data1_sleep>, <&hs1_mi2s_ws_sleep>,
                        <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                        <&hs2_mi2s_ws_sleep>, <&lpass_quad_clk_
sleep>, <&lpass_quad_data_sleep>,
                        <&lpass_quad_ws_sleep>;
            pinctrl-1 = <&hs1_mi2s_data0>, <&mi2s_mclk>, <&hs0_
mi2s_data0>,
                        <&hs0_mi2s_sclk>, <&hs0_mi2s_data1>, <&
hs0_mi2s_ws>,
                        <&hs1_mi2s_sclk>, <&hs1_mi2s_data1>, <&
hs1_mi2s_ws>,
                        <&hs2_mi2s_data0>, <&hs2_mi2s_data1>, <&
hs2_mi2s_sck>,
                        <&hs2_mi2s_ws>, <&lpass_quad_clk>, <&
lpass_quad_data>,
                        <&lpass_quad_ws>;
            pinctrl-2 = <&hs1_mi2s_data0_sleep>, <&mi2s_mclk_
sleep>, <&hs0_mi2s_data0_sleep>,
                        <&hs0_mi2s_sclk_sleep>, <&hs0_mi2s_
data1_sleep>, <&hs0_mi2s_ws_sleep>,
                        <&hs1_mi2s_sclk_sleep>, <&hs1_mi2s_
data1_sleep>, <&hs1_mi2s_ws_sleep>,
                        <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                        <&hs2_mi2s_ws_sleep>, <&lpass_quad_clk_
sleep>, <&lpass_quad_data_sleep>,
                        <&lpass_quad_ws_sleep>;
            lpi-tdm1-capture-dai-link {
```

```
                    link-name = "TDM-LPAIF_VA-TX-PRIMARY";

                    cpu {
                            sound-dai = <&q6apmbedai PRIMARY_
TDM_TX_0>;
                    };

                    codec {
                            sound-dai = <&msm_stub_codec 1>;
                    };
            };

            lpi-tdm1-playback-dai-link {
                    link-name = "TDM-LPAIF_VA-RX-PRIMARY";

                    cpu {
                            sound-dai = <&q6apmbedai PRIMARY_
TDM_RX_0>;
                    };

                    codec {
                            sound-dai = <&msm_stub_codec 0>;
                    };
            };
```


Modify the resource manager xml file for PAL configuration

Update the `resourcemanager.xml` file by replacing the existing codec device backend names. The following sections describe example file changes made to the `/etc/resourcemanager_qcm6490_idp.xml` file on the device for specific use cases.

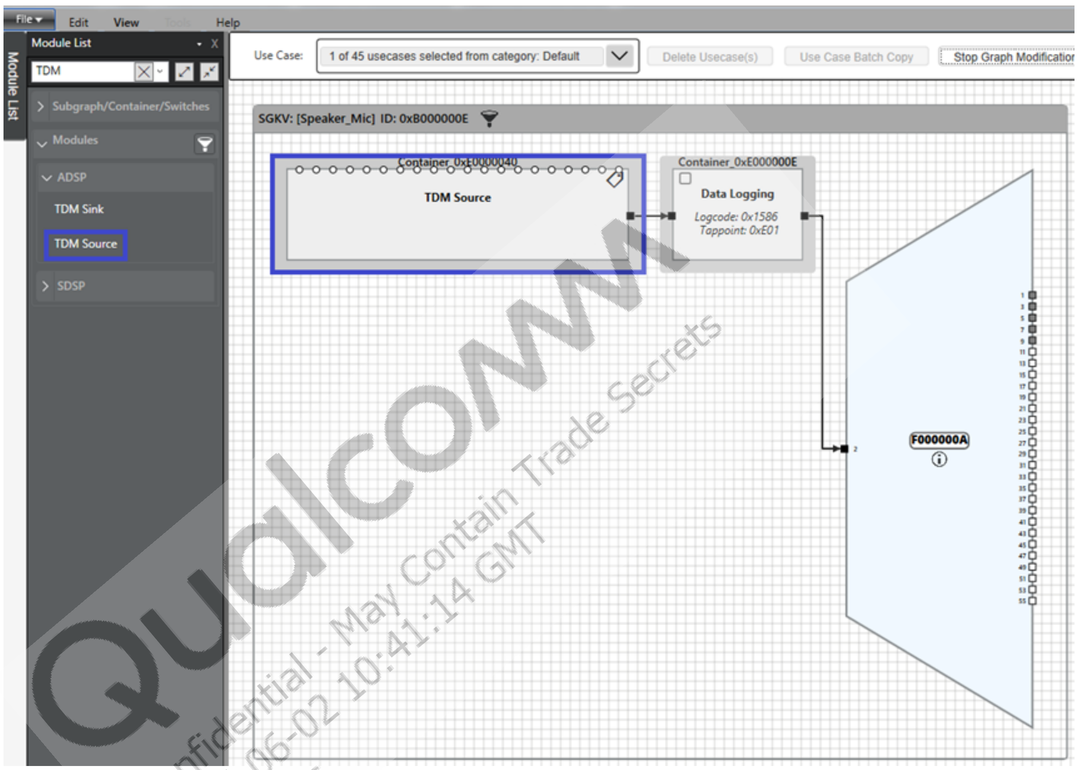
The following code shows the changed backend for the `PAL_DEVICE_IN_SPEAKER_MIC` device. It configures the channels as 8.

```
<id>PAL_DEVICE_IN_SPEAKER_MIC</id>
<back_end_name> TDM-LPAIF_VA-TX-PRIMARY</back_end_name>
<max_channels>8</max_channels>
<channels>8</channels>
<samplerate>48000</samplerate>
```

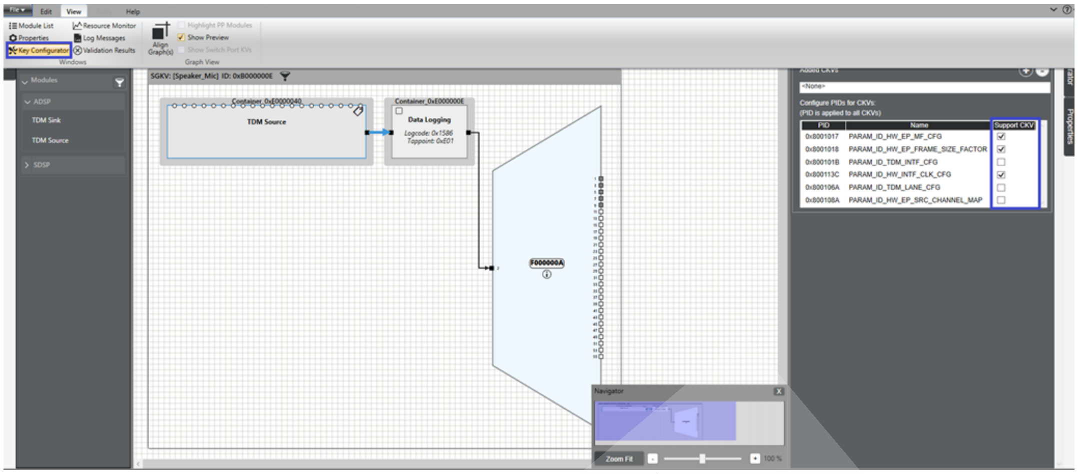
Modify the use case graph

After making the necessary changes in the `resourcemanager.xml` file, follow these steps to replace the source devices with TDM source devices in acdb. Note that replacing a source module in a device subgraph affects all use cases where the device is used.

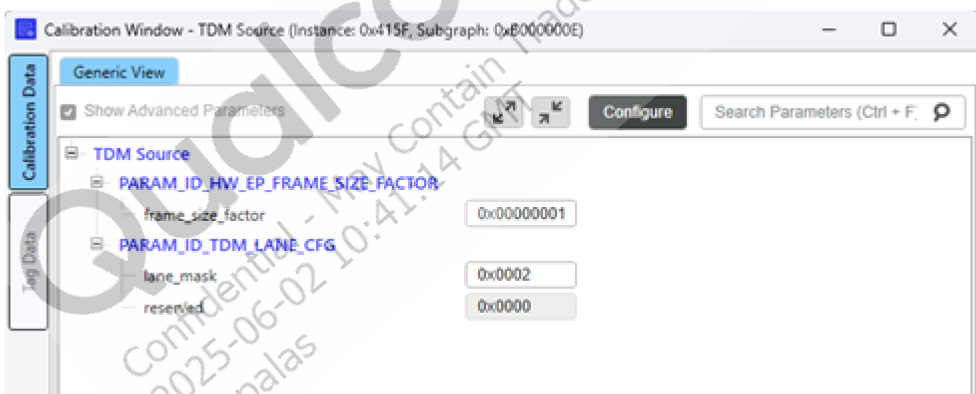
1. In QACT, navigate to a use case containing the desired device subgraph.
2. Select *Start Graph Modification*.
3. Choose the existing source module and then choose the DeviceTX subgraph.
4. Replace the Source with the TDM Source by dragging the TDM Source from the module list and dropping it onto the existing Source.

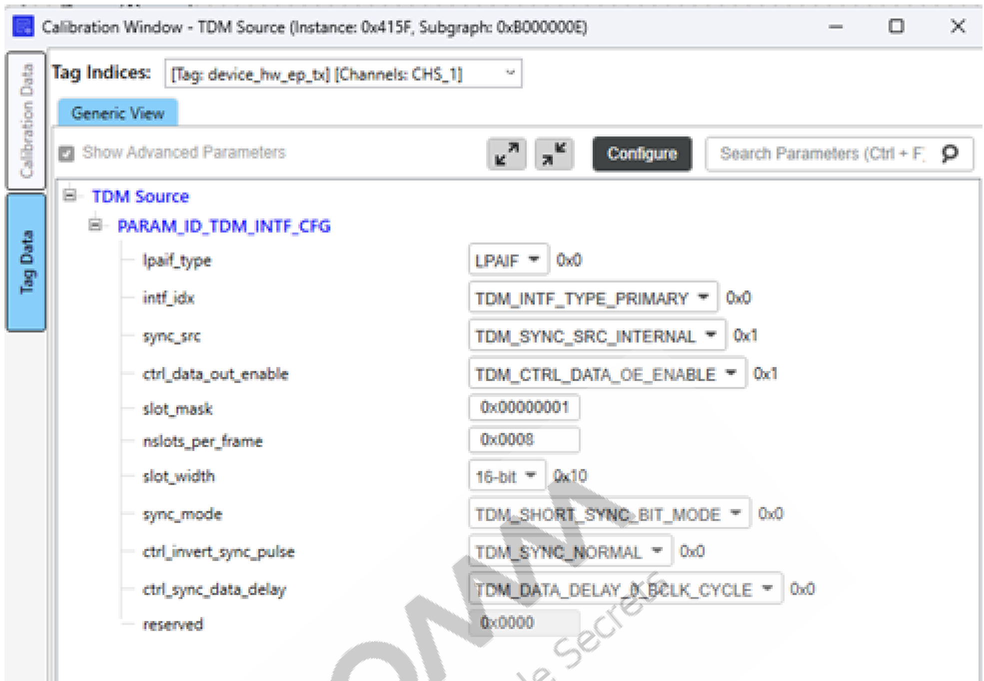


5. Select *View* → *Key Configurator*.
6. Select the *Support CKV* checkboxes associated with the relevant CKVs.



7. Select the Module Tag associated checkbox with relevant tags.
8. Double-tap the TDM source module to open the properties window. Ensure to properly configure the following parameters:
   - Calibration data and tag data should be configured as follows:

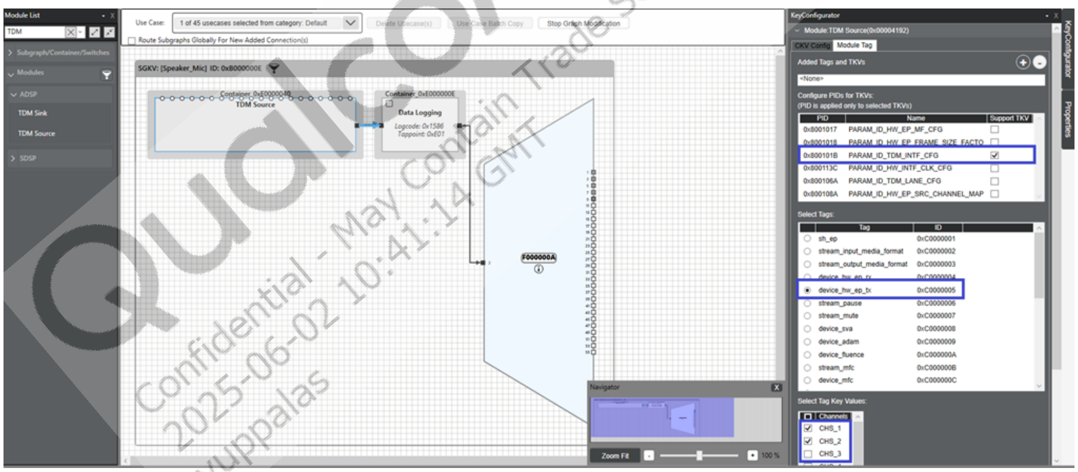




The following table lists each configuration detail.

Configuration	Details
lpaif_type	Selects the interface type for TDM interface. For example, LPAIF, LPAIF_VA, etc.
intf_idx	Selects the interface index for the TDM interface. For example, TDM_INTF_TYPE_PRIMARY, TDM_INTF_TYPE_SECONDARY, etc.
sync_src	Sync source. – Select TDM_SYNC_SRC_INTERNAL for MSM Controller mode configuration. – Select TDM_SYNC_SRC_EXTERNAL for MSM Target mode configuration.
ctrl_data_out_enable	TDM block shares data out signal to driver with other controllers.
slot_mask	Specifies the active slots for channels in 32-bit format. – 0x00000001 – One active slot/channel at position 0. – 0x00000002 – One active slot/channel at position 1. – 0x00000003 – Two active slots/channels at position 0 and 1. – 0x0000000F – Four active slots/channels at position 0, 1, 2, and 3.
nslots_per_frame	Specifies the number of slots per frame. The slot number should be greater than or equal to the number of channels. – 0x0001 – Total number of slots – 1 – 0x0002 – Total number of slots – 2 – 0x0008 – Total number of slots – 8 – 0x0010 – Total number of slots – 16
slot_width	Specifies the bitwidth of the slot. The slot bitwidth should be greater than or equal to the number of the channel bitwidth. Supported bitwidths are 16, 24, and 32 bit.
sync_mode	TDM synchronization mode settings should be configured according to the hardware requirements of third-party devices. Supported modes are TDM_SHORT_SYNC_BIT_MODE, TDM_LONG_SYNC_MODE, and TDM_SHORT_SYNC_SLOT_MODE.
ctrl_invert_sync_pulse	Indicates whether to invert synchronization. These settings should be configured according to the hardware requirements of third-party devices. Supported modes are TDM_SYNC _NORMAL and TDM_SYNC_INVERT.
ctrl_sync_data_delay	Indicates the number of bit clock cycles for delaying data synchronization, supporting up to 3-bit clock delays. These settings should be configured according to the hardware

9. Assign the appropriate module tags. Typically, a module tag controls the channel count and corresponding channel mask of a source module. To modify module tags:
10. Ensure the TDM source is selected, Key Configurator is open, and the graph is still in the modification state.
11. In *Key Configurator*, select the *Module Tag* tab and click *+*.
12. In the *Select Tags* list, select the correct module tag. In this example, it is `device_hw_ep_tx`.
13. Once the tag is selected, a list of PIDs appears. In this list, select the *Support TKV* checkbox associated with the correct PID. In this example, it is `PARAM_ID_TDM_INTF_CFG`.
14. In the *Select Tag Key Values* list, select the correct tag key value. In this example, it is `CHS_1` for mono channel.
15. Select *Add*.
16. Repeat this process for CHS_2 for stereo channel support.



10. Once all channel configurations are added, click *Stop Graph Modification* and save the acdb files.

Troubleshoot and validate MIS2/TDM

This section checks that the sound card is properly registered. It also verifies I2S is recording from the PulseAudio level.

Note: Connect to the device console using SSH. See Use SSH for instructions.

Verify MI2S/TDM source capture

Check that the newly-added backend DAI registers the sound card:

```
cat /proc/asound/pcm
```

```
00-05: MI2S-LPAIF-TX-PRIMARY msm-stub-aif1-tx-5 :  : capture 1
```

If the backend DAI doesn't appear in the backend list, there may have been an issue adding the DAI links. Check the kernel logs to make sure there were no errors related to DAI links.

For TDM, the backend name will be `TDM-LPAIF_VA-TX-PRIMARY`.

Verify the device is recording from the PulseAudio level. This example is for a 48 kHz, stereo, 16-bit recording.

1. Update the *channels tab* in the `resourcemanager.xml` file to *2*.
2. Reboot the target.
3. Run the following command from a shell prompt. It should generate a file at `/opt/rec.wav` with the audio sent from the I2S device.

   ```
   parec -v --rate=48000 --format=s16le --channels=2 --file-
   format=wav /opt/rec.wav --device=regular2
   ```

The command shell should show a message similar to the following:

```
<nels=2 --file-format=wav /opt/rec.wav --device=regular2
Opening a recording stream with sample specification 's16le 2ch 48000Hz' and channel map 'front-left,front-right'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, fragsize=384000
Using sample spec 's16le 2ch 48000Hz', channel map 'front-left,front-right'.
Connected to device regular2 (index: 5, suspended: no).
Time: 51.075 sec; Latency: 25462 usec.
```

Figure6 Command shell example for verifying PulseAudio recording

Analyze logs

When capture for 48k/stereo/16-bit is done, the following logs should be seen.

- The I2S source has been modified for the speaker_mic device. The device open should be called for the speaker mic as follows:

```
Apr 29 15:50:46 pulseaudio[1001]: open: 469: Enter.
deviceCount 0 for device id 28 (PAL_DEVICE_IN_SPEAKER_MIC)
```

- The backend used for the capture is MI2S-LPAIF-TX-PRIMARY, with SR as 48000 kHz, and the number of channels as 2:

```
Apr 29 15:50:46 pulseaudio[1001]: setDeviceMediaConfig: 1056:
MI2S-LPAIF-TX-PRIMARY rate ch fmt data_fmt 48000 2 2 1
```

- The device open should exit with status 0 and a proper deviceCount for the speaker mic device:

```
Apr 29 15:50:46 pulseaudio[1001]: open: 504: Exit.
deviceCount 1 for device id 28 (PAL_DEVICE_IN_SPEAKER_MIC),
exit status: 0
```

- The hardware endpoints should be configured properly for 48k/stereo/16-bit as follows:

```
Apr 29 15:50:46 pulseaudio[1001]: configure_hw_ep_media_
config: 664 rate 48000 bw 16 ch 2, data_fmt 1
```

QCS8275

MI2S/TDM interfaces

MI2S/TDM overview

MI2S is a serial bus interface that connects multiple digital audio devices. It is a simple data interface without any form of address or device selection.

The TDM interface transfers many channels of audio data between devices within a system.

The TDM interface has two control clocks:

- Frame synchronization pulse (FSYNC)
- Serial clock (SCLK), also known as the bit clock (BCLK)

The following figure shows the MI2S process flow:

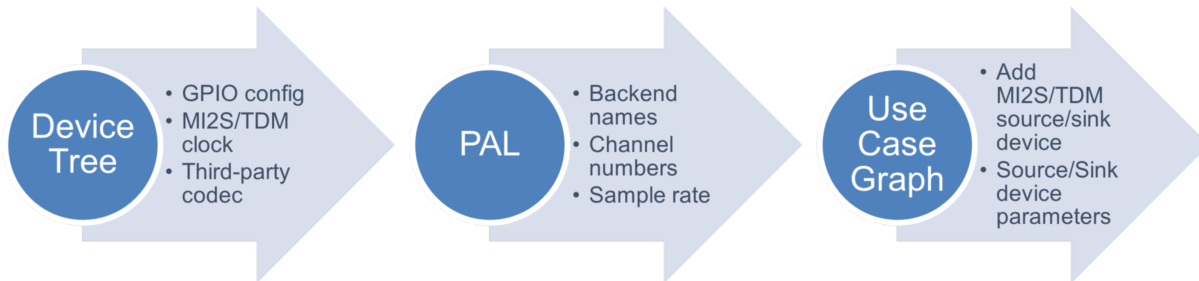


Figure7 MI2S flow

GPIOs to connect third-party devices

The MI2S interfaces use GPIOs for signal and data transmission. The following table lists the GPIOs that can connect third-party codec or speaker amps. Use these GPIOs even for TDM configuration.

GPIOs for MI2S configuration

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Primary (LPI-LS-MI2S4)	MI2S: • RX - MI2S-LPAIF-RX-PRIMARY • TX - MI2S-LPAIF-TX-PRIMARY TDM: • RX - TDM-LPAIF-RX-PRIMARY • TX - TDM-LPAIF-TX-PRIMARY	• GPIO_125 CLK • GPIO_126 SYNC • GPIO_127 DATA0 • GPIO_128 DATA1	• LPASS_GPIO_12 • LPASS_GPIO_13 • LPASS_GPIO_17 • LPASS_GPIO_18

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Secondary (LS-MI2S1)	MI2S • RX - MI2S-LPAIF-RX-SECONDARY • TX - MI2S-LPAIF-TX-SECONDARY TDM • RX - TDM-LPAIF-RX-SECONDARY • TX - TDM-LPAIF-TX-SECONDARY	• GPIO_98 CLK • GPIO_99 SYNC • GPIO_100 DATA0 • GPIO_101 DATA1	–
Tertiary (LS-MI2S2)	MI2S • RX - MI2S-LPAIF-RX-TERTIARY • TX - MI2S-LPAIF-TX-TERTIARY TDM: • RX - TDM-LPAIF-RX-TERTIARY • TX - TDM-LPAIF-TX-TERTIARY	• GPIO_102 CLK • GPIO_103 SYNC • GPIO_104 DATA0 • GPIO_105 DATA1	–

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Quaternary (LPI-LS-MI2S0)	MI2S • RX - MI2S-LPAIF_RXTX-RX-PRIMARY • TX - MI2S-LPAIF_RXTX-TX-PRIMARY TDM • RX - TDM-LPAIF_RXTX-RX-PRIMARY • TX - TDM-LPAIF_RXTX-TX-PRIMARY	• GPIO_110 CLK • GPIO_111 SYNC • GPIO_112 DATA0 • GPIO_113 DATA1 • GPIO_114 DATA2 • GPIO_115 DATA3	• LPASS_GPIO_0 • LPASS_GPIO_1 • LPASS_GPIO_2 • LPASS_GPIO_3 • LPASS_GPIO_4 • LPASS_GPIO_5
Quinary (LPI-LS-MI2S1)	MI2S • RX - MI2S-LPAIF_VA-RX-PRIMARY • TX - MI2S-LPAIF_VA-TX-PRIMARY TDM • RX - TDM-LPAIF_VA-RX-PRIMARY • TX - TDM-LPAIF_VA-TX-PRIMARY	• GPIO_116 CLK • GPIO_117 SYNC • GPIO_118 DATA0 • GPIO_119 DATA1	• LPASS_GPIO_6 • LPASS_GPIO_7 • LPASS_GPIO_8 • LPASS_GPIO_9

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
Senary (LPI-LS-MI2S2)	MI2S • RX - MI2S-LPAIF_WSA-RX-PRIMARY • TX - MI2S-LPAIF_WSA-TX-PRIMARY TDM • RX - TDM-LPAIF_WSA-RX-PRIMARY • TX - TDM-LPAIF_WSA-TX-PRIMARY	• GPIO_120 CLK • GPIO_121 SYNC • GPIO_122 DATA0 • GPIO_123 DATA1	• LPASS_GPIO_10 • LPASS_GPIO_11 • LPASS_GPIO_15 • LPASS_GPIO_16
Septenary (LPI-LS-MI2S3)	MI2S • RX - MI2S-LPAIF_AUD-RX-PRIMARY • TX - MI2S-LPAIF_AUD-TX-PRIMARY TDM • RX - TDM-LPAIF_AUD-RX-PRIMARY • TX - TDM-LPAIF_AUD-TX-PRIMARY	• GPIO_129 CLK • GPIO_130 SYNC • GPIO_131 DATA0 • GPIO_132 DATA1	• LPASS_GPIO_19 • LPASS_GPIO_20 • LPASS_GPIO_21 • LPASS_GPIO_22

Audio interface mapping	**Backend name**	**TLMM GPIOs**	**LPASS GPIOs**
HS0_IF (HS0-MI2S)	MI2S • RX - MI2S-LPAIF_SDR-RX-PRIMARY • TX - MI2S-LPAIF_SDR-TX-PRIMARY TDM • RX - TDM-LPAIF_SDR-RX-PRIMARY • TX - TDM-LPAIF_SDR-TX-PRIMARY	• GPIO_106 CLK • GPIO_107 SYNC • GPIO_108 DATA0 • GPIO_109 DATA1	–
HS1_IF (HS1-MI2S)	MI2S • RX - MI2S-LPAIF_SDR-RX-SECONDARY • TX - MI2S-LPAIF_SDR-TX-SECONDARY TDM • RX - TDM-LPAIF_SDR-RX-SECONDARY • TX - TDM-LPAIF_SDR-TX-SECONDARY	• GPIO_45 CLK • GPIO_46 SYNC • GPIO_47 DATA0 • GPIO_48 DATA1	–

Audio interface mapping	Backend name	TLMM GPIOs	LPASS GPIOs
HS2_IF (HS2-MI2S)	MI2S • RX - MI2S-LPAIF_SDR-RX-TERTIARY • TX - MI2S-LPAIF_SDR-TX-TERTIARY TDM • RX - TDM-LPAIF_SDR-RX-TERTIARY • TX - TDM-LPAIF_SDR-TX-TERTIARY	• GPIO_49 CLK • GPIO_50 SYNC • GPIO_51 DATA0 • GPIO_52 DATA1	–

There are three MCLKs capable of generating up to 512 $\times$, 48 kHz (24.576 MHz) each. Despite what the GPIO alternate function name suggests, all three MCLK outputs are independent of I2S and HS-I2S interfaces. They can be used with all interfaces.

MCLKs for independent interface configuration

MCLK	GPIO
MI2S_MCLK0	GPIO_97
MI2S_MCLK1	GPIO_109
EXT_MCLK1	GPIO_115 GPIO_119 GPIO_126 GPIO_124 GPIO_132

Set GPIO and clock configuration logic for MI2S

Configure device tree files

The machine driver enables/disables the MI2S GPIOs whenever capture starts over the MI2S interface.

Specify the MI2S GPIO configuration in the device tree file of the platform and its pinctrl file.

GPIO pinctrl entries are in the following files based on the form-factor of the platform:

- `arch/arm64/boot/dts/qcom/qcs8300-addons-ride.dtsi`

Example GPIO configuration for primary I2S

Add MI2S GPIO Active/Sleep configurations in the customer target .dtsi file.

Ensure that no other interface uses these MI2S interface GPIOs for another purpose. Remove all other entries from the device tree.

The following is an example configuration for the primary MI2S interface. Add similar configurations for the primary MI2S interfaces if the customer plans to use them.

```
&sound {
            compatible = "qcom,qcs8300-sndcard";
            model = "qcs8300-ridesx-snd-card";

            clocks = <&q6prmcc LPASS_HW_MACRO_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>,
                        <&q6prmcc LPASS_HW_DCODEC_VOTE LPASS_
CLK_ATTRIBUTE_COUPLE_NO>;
            clock-names = "macro", "dcodec";

            pinctrl-names = "default", "stub_aif0_active",
"stub_aif0_sleep",
                            "stub_aif1_active", "stub_aif1_sleep
", "stub_aif2_active",
                            "stub_aif2_sleep", "stub_aif3_active
", "stub_aif3_sleep";
            pinctrl-0 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>, <&lpass_i2s1_clk_
sleep>, <&lpass_i2s1_data_sleep>,
                        <&lpass_i2s1_ws_sleep>, <&hs2_mi2s_
data0_sleep>, <&mi2s_mclk_sleep>,
                        <&hs2_mi2s_sck_sleep>, <&hs2_mi2s_data1_
sleep>,<&hs2_mi2s_ws_sleep>;
            pinctrl-1 = <&lpass_i2s1_clk>, <&lpass_i2s1_data>,
<&lpass_i2s1_ws>;
            pinctrl-2 = <&lpass_i2s1_clk_sleep>, <&lpass_i2s1_
data_sleep>,
                        <&lpass_i2s1_ws_sleep>;
            pinctrl-3 = <&mi2s1_data0>, <&mi2s1_data1>, <&mi2s1_
sck>, <&mi2s1_ws>;
            pinctrl-4 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>;
            pinctrl-5 = <&hs2_mi2s_data0>, <&hs2_mi2s_data1>, <&
hs2_mi2s_sck>, <&mi2s_mclk>,
                        <&hs2_mi2s_ws>;
            pinctrl-6 = <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
```

```
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                        <&mi2s_mclk_sleep>,<&hs2_mi2s_ws_sleep>;
            pinctrl-7 = <&mi2s1_data0>, <&mi2s1_data1>, <&mi2s1_
sck>, <&mi2s_mclk>,
                        <&mi2s1_ws>;
            pinctrl-8 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>, <&mi2s_mclk_sleep>;
lpi-mi2s1-capture-dai-link { link-name = "MI2S-LPAIF_VA-TX-
PRIMARY"; cpu { sound-dai = <&q6apmbedai PRIMARY_MI2S_TX>; }
```

MI2S clock configuration logic – Controller mode

In Controller mode, the sample rate, number of channels, and the bit width configure the bit clock frequency.

For example, bit clock frequency = 48000 (Hz) x 2 (stereo) x 16 (bit width) = 1.536 MHz.

The Qualcomm chipset provides both the bit clock and WS clock. The bit clock sources the WS clock. The WS clock is set equal to the sample rate. WS doesn't require clock configuration.

MI2S clock configuration logic – Target mode

In Target mode, a third-party MI2S device provides both bit clock and WS clock. LPASS handles the MI2S interface clock control, number of channels, number of serial data lines, dataline direction bit-width, and sample rate. The apps processor reads the MI2S interface configuration from acdb and pushes it to LPASS while starting the use case over MI2S interface. For Target mode, in acdb, you must set the clock source for the bit clock as external.

By default, release builds have complete configurations for MI2S hardware interfaces. These changes must be present in the .dtsi files.

To enable the MI2S audio hardware interface:

1. Verify that the GPIOs map to the correct MI2S hardware interface
2. Remove GPIO configuration settings if another hardware module is using the same GPIOs as MI2S.

Modify the resource manager xml file for PAL configuration

Update the `resourcemanager.xml` file by replacing the existing codec device backend names. The following sections describe example file changes made to the `/etc/resourcemanager_qcs8300_ridesx.xml` file on the device for specific use cases.

The following code shows the changed backend for the `PAL_DEVICE_IN_SPEAKER_MIC` device. It configures the channels as mono.

```
<id>PAL_DEVICE_IN_SPEAKER_MIC</id>
<back_end_name>MI2S-LPAIF-TX-PRIMARY</back_end_name>
<max_channels>4</max_channels>
<samplerate>48000</samplerate>
<channels>1</channels>
```

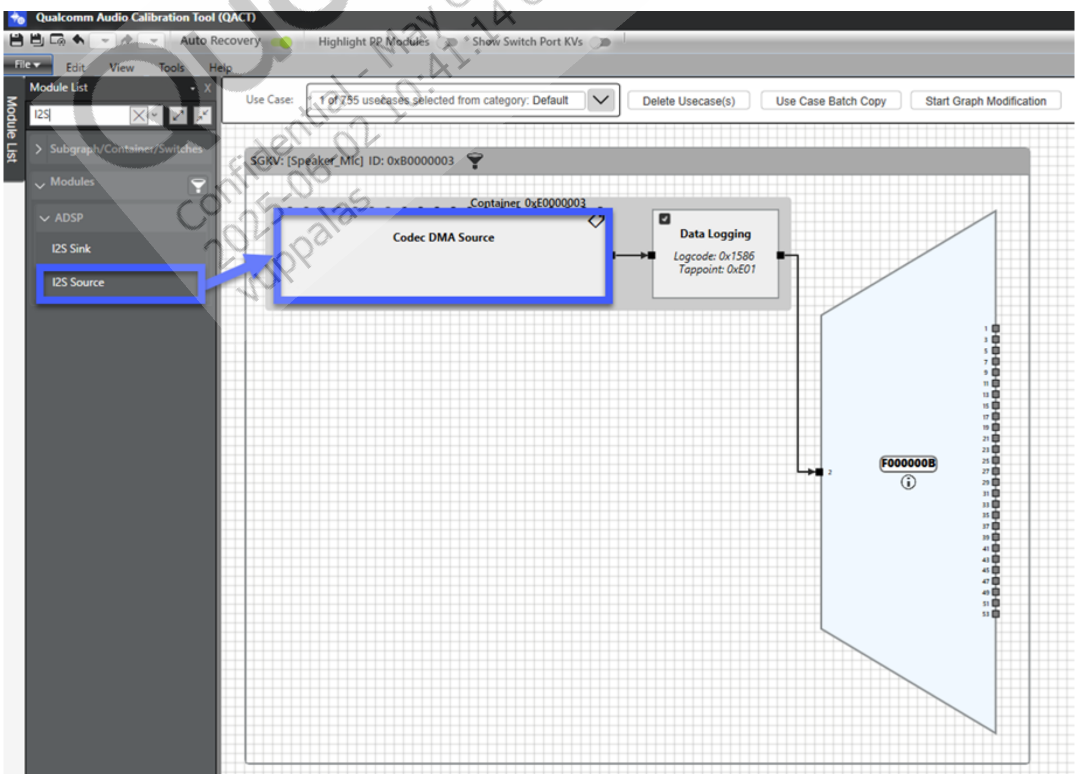
In this code snippet:

- `<max_channels>` – Maximum number of channels supported over MI2S
- `<channels>` – Default number of channels used while executing the use case.

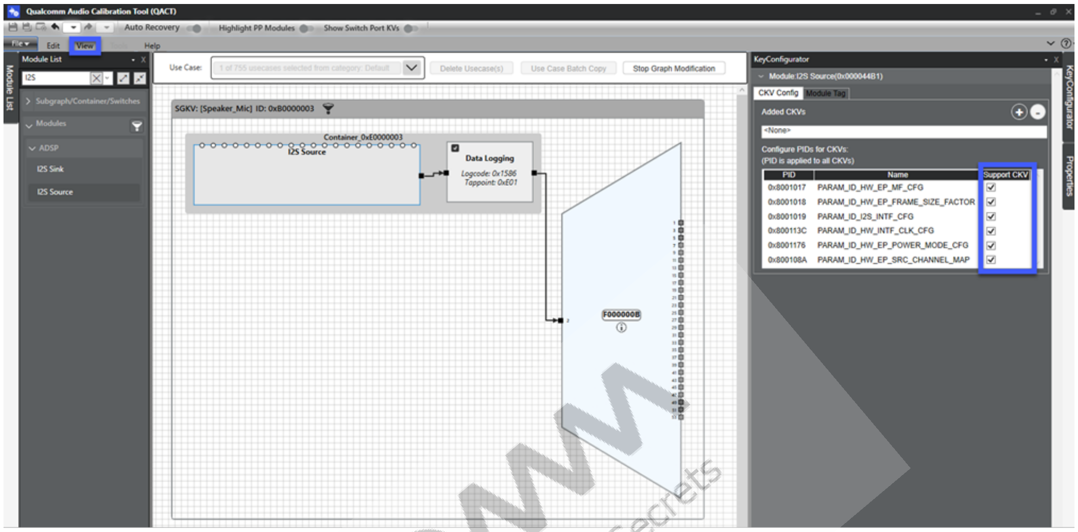
Modify the use case graph

After making the necessary changes in the `resourcemanager.xml` file, follow these steps in acdb to replace the WCD source devices with MI2S source devices. Note that replacing a source module in a device subgraph affects all use cases where the device is used.

1. In QACT, navigate to a use case containing the device subgraph.
2. Select *Start Graph Modification*.
3. Choose the existing source module and then choose the DeviceTX subgraph.
4. Replace the code DMA Source with the I2S Source by dragging the I2S Source from the module list and dropping it onto the existing DMA Source.

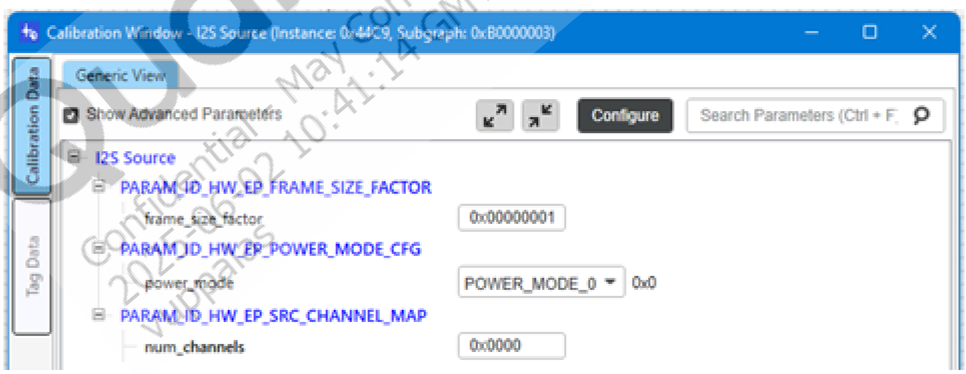


5. Select *View* → *Key Configurator*.
6. Select the *Support CKV* checkboxes associated with the relevant CKVs.

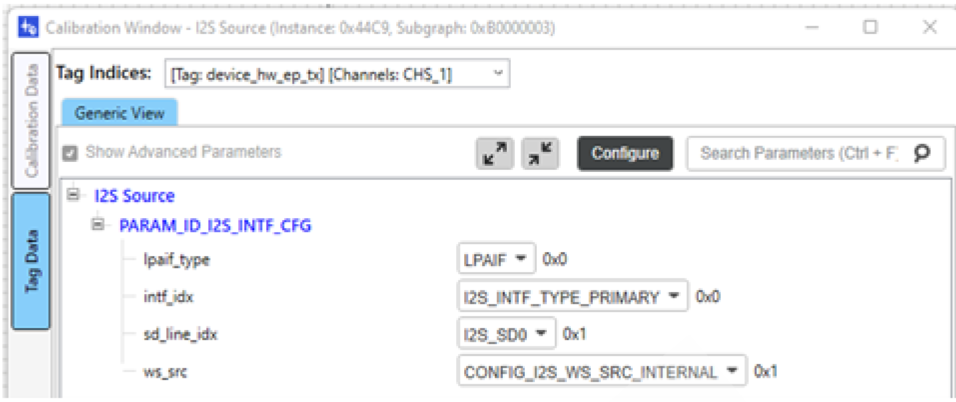


7. Double-tap the I2S source module to open the properties window. Ensure to properly configure the following parameters:

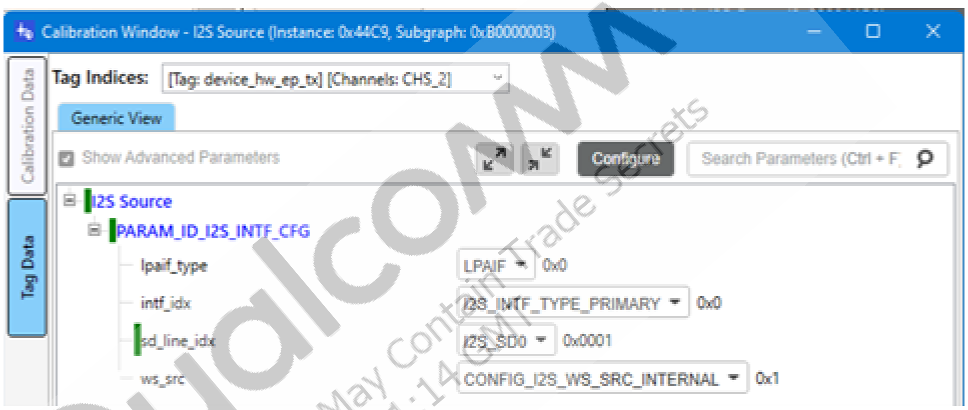
   - Calibration data should be set as follows:



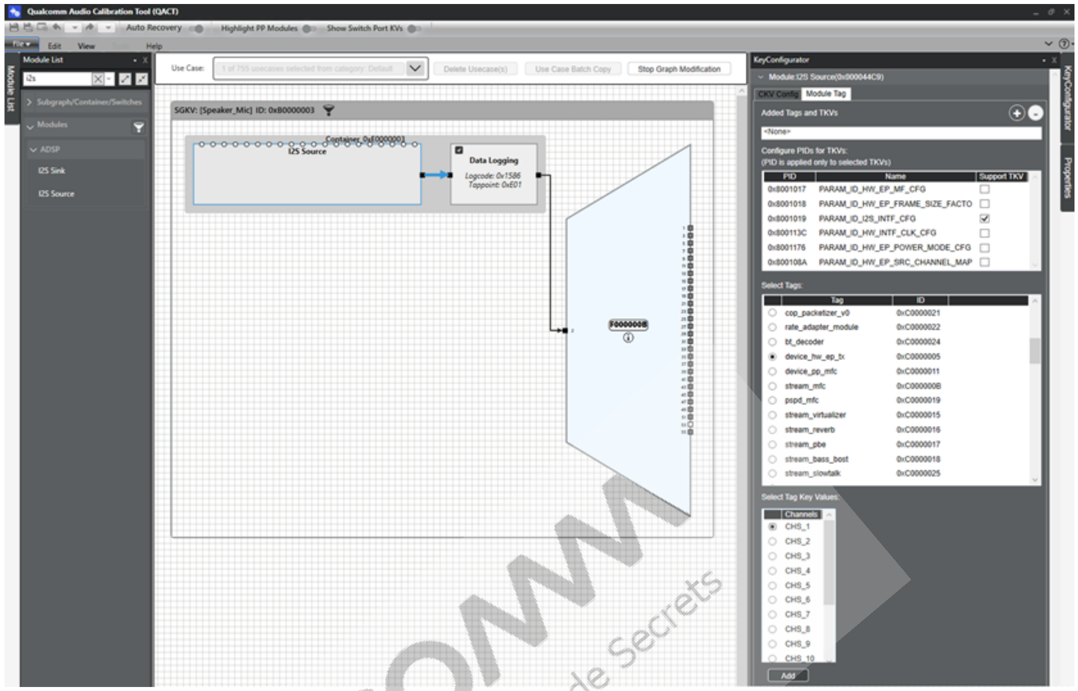
   - Tag data of the I2S Source for the mono channel should be set as follows:



- Tag data of the I2S Source for the stereo channel should be set as follows:



8. Assign the appropriate module tags. Typically, a module tag controls the channel count and corresponding channel mask of a source module. To modify module tags:
9. Ensure the I2S source is selected, Key Configurator is open, and the graph is still in the modification state.
10. In *Key Configurator*, select the *Module Tag* tab and click *+*.
11. In the *Select Tags* list, choose the correct module tag. In this example, it is `device_hw_ep_tx`.
12. Once the tag is selected, a list of PIDs appears. In this list, select the *Support TKV* checkbox associated with the correct PID. In this example, it is `PARAM_ID_I2S_INTF_CFG`.
13. In the *Select Tag Key Values* list, select the correct tag key value. In this example, it is `CHS_1` for mono channel.
14. Select *Add*.
15. Repeat this process for CHS_2 for stereo channel support.



9. Once all channel configurations are added, select *Stop Graph Modification* and save the acdb files.

Set GPIO and clock configuration logic for TDM

Device tree configuration

The machine driver enables/disables the TDM GPIOs whenever playback or capture starts over the TDM interface.

Specify the TDM GPIO configuration in the device tree file of the platform and its pinctrl file.

GPIO pinctrl entries are in the following files based on the form-factor of the platform:

- `arch/arm64/boot/dts/qcom/qcs8300-addons-ride.dtsi`

Example GPIO configuration for primary TDM

Add TDM GPIO Active/Sleep configurations in the customer target .dtsi file.

Ensure that no other interface uses these TDM interface GPIOs for another purpose. Remove all other entries from the device tree.

The following is an example configuration for the primary TDM interface. Add similar configurations for the primary TDM interfaces if the customer plans to use them.

```
&sound {
        compatible = "qcom,qcs8300-sndcard";
```

```
            model = "qcs8300-ridesx-snd-card";

            clocks = <&q6prmcc LPASS_HW_MACRO_VOTE LPASS_CLK_
ATTRIBUTE_COUPLE_NO>,
                        <&q6prmcc LPASS_HW_DCODEC_VOTE LPASS_
CLK_ATTRIBUTE_COUPLE_NO>;
            clock-names = "macro", "dcodec";

            pinctrl-names = "default", "stub_aif0_active",
"stub_aif0_sleep",
                            "stub_aif1_active", "stub_aif1_sleep
", "stub_aif2_active",
                            "stub_aif2_sleep", "stub_aif3_active
", "stub_aif3_sleep";
            pinctrl-0 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>, <&lpass_i2s1_clk_
sleep>, <&lpass_i2s1_data_sleep>,
                        <&lpass_i2s1_ws_sleep>, <&hs2_mi2s_
data0_sleep>, <&mi2s_mclk_sleep>,
                        <&hs2_mi2s_sck_sleep>, <&hs2_mi2s_data1_
sleep>,<&hs2_mi2s_ws_sleep>;
            pinctrl-1 = <&lpass_i2s1_clk>, <&lpass_i2s1_data>,
<&lpass_i2s1_ws>;
            pinctrl-2 = <&lpass_i2s1_clk_sleep>, <&lpass_i2s1_
data_sleep>,
                        <&lpass_i2s1_ws_sleep>;
            pinctrl-3 = <&mi2s1_data0>, <&mi2s1_data1>, <&mi2s1_
sck>, <&mi2s1_ws>;
            pinctrl-4 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>;
            pinctrl-5 = <&hs2_mi2s_data0>, <&hs2_mi2s_data1>, <&
hs2_mi2s_sck>, <&mi2s_mclk>,
                        <&hs2_mi2s_ws>;
            pinctrl-6 = <&hs2_mi2s_data0_sleep>, <&hs2_mi2s_
data1_sleep>, <&hs2_mi2s_sck_sleep>,
                        <&mi2s_mclk_sleep>,<&hs2_mi2s_ws_sleep>;
            pinctrl-7 = <&mi2s1_data0>, <&mi2s1_data1>, <&mi2s1_
sck>, <&mi2s_mclk>,
                        <&mi2s1_ws>;
            pinctrl-8 = <&mi2s1_data0_sleep>, <&mi2s1_data1_
sleep>, <&mi2s1_sck_sleep>,
                        <&mi2s1_ws_sleep>, <&mi2s_mclk_sleep>;
```

```
            lpi-tdm1-capture-dai-link {
                    link-name = "TDM-LPAIF_VA-TX-PRIMARY";

                    cpu {
                            sound-dai = <&q6apmbedai PRIMARY_
TDM_TX_0>;
                    };

                    codec {
                            sound-dai = <&msm_stub_codec 1>;
                    };
            };

            lpi-tdm1-playback-dai-link {
                    link-name = "TDM-LPAIF_VA-RX-PRIMARY";

                    cpu {
                            sound-dai = <&q6apmbedai PRIMARY_
TDM_RX_0>;
                    };

                    codec {
                            sound-dai = <&msm_stub_codec 0>;
                    };
            };
```


Modify the resource manager xml file for PAL configuration

Update the `resourcemanager.xml` file by replacing the existing codec device backend names. The following sections describe example file changes made to the `/etc/resourcemanager_qcm6490_idp.xml` file on the device for specific use cases.

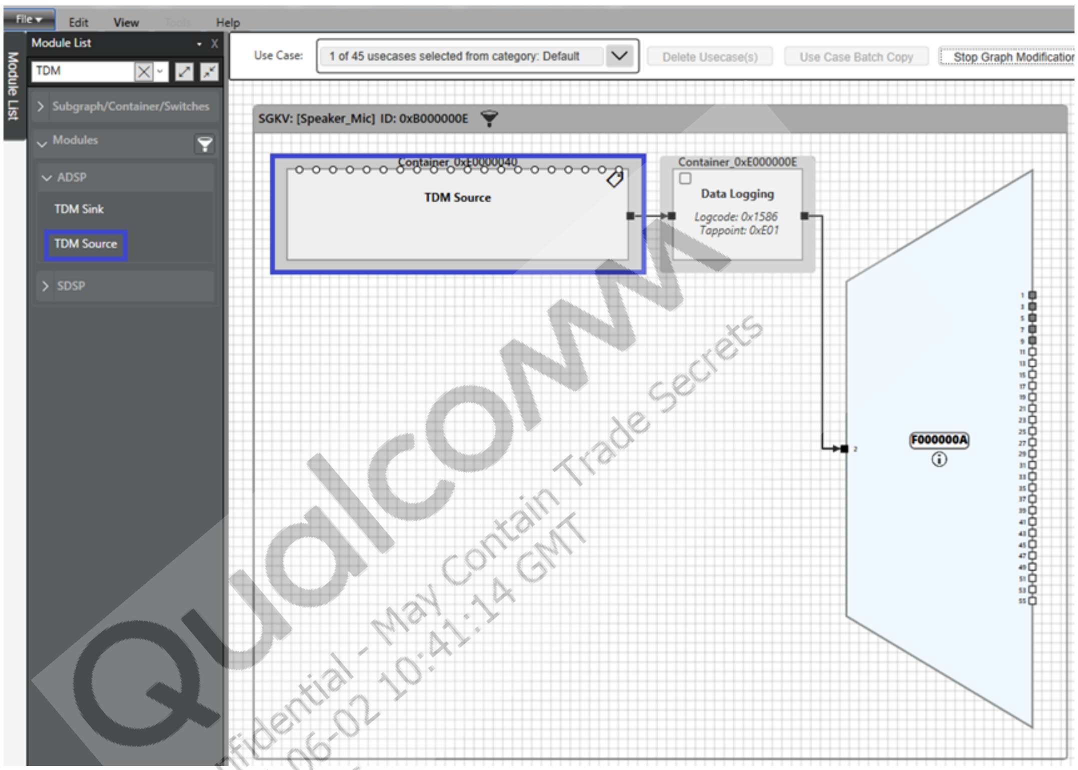
The following code shows the changed backend for the `PAL_DEVICE_IN_SPEAKER_MIC` device. It configures the channels as 8.

```
<id>PAL_DEVICE_IN_SPEAKER_MIC</id>
<back_end_name> TDM-LPAIF_VA-TX-PRIMARY</back_end_name>
<max_channels>8</max_channels>
<channels>8</channels>
<samplerate>48000</samplerate>
```

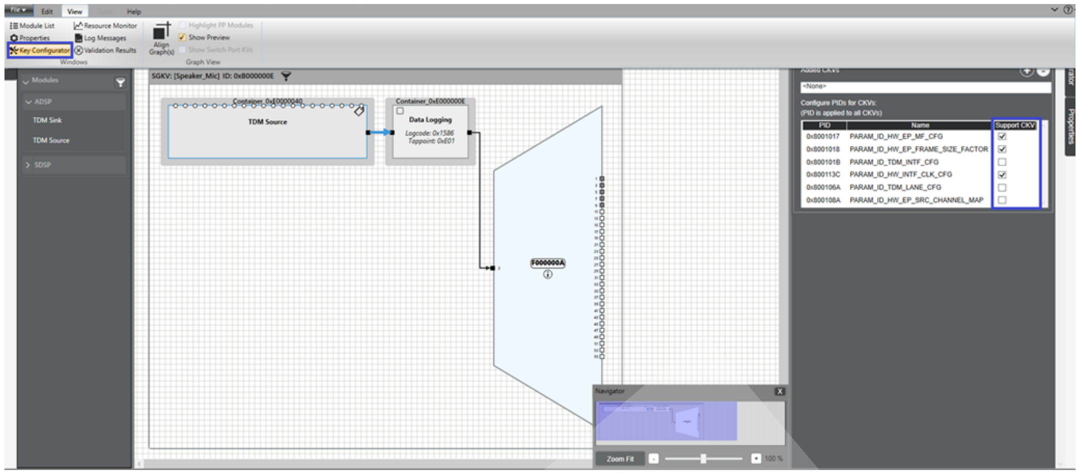
Modify the use case graph

After making the necessary changes in the `resourcemanager.xml` file, follow these steps to replace the source devices with TDM source devices in acdb. Note that replacing a source module in a device subgraph affects all use cases where the device is used.

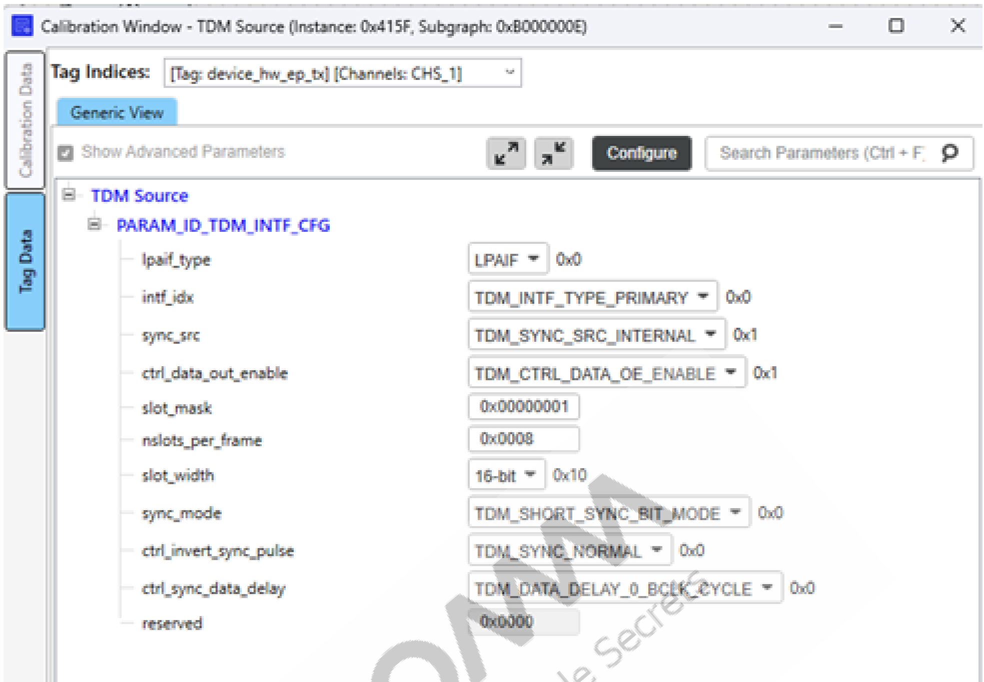
1. In QACT, navigate to a use case containing the desired device subgraph.
2. Select *Start Graph Modification*.
3. Choose the existing source module and then choose the DeviceTX subgraph.
4. Replace the Source with the TDM Source by dragging the TDM Source from the module list and dropping it onto the existing Source.



5. Select *View* → *Key Configurator*.
6. Select the *Support CKV* checkboxes associated with the relevant CKVs.



7. Select the Module Tag associated checkbox with relevant tags.
8. Double-tap the TDM source module to open the properties window. Ensure to properly configure the following parameters:
   - Calibration data and tag data should be configured as follows:



The following table lists each configuration detail.

Configuration	**Details**
lpaif_type	Selects the interface type for TDM interface. For example, LPAIF, LPAIF_VA, etc.
intf_idx	Selects the interface index for the TDM interface. For example, TDM_INTF_TYPE_PRIMARY, TDM_INTF_TYPE_SECONDARY, etc.
sync_src	Sync source. – Select TDM_SYNC_SRC_INTERNAL for MSM Controller mode configuration. – Select TDM_SYNC_SRC_EXTERNAL for MSM Target mode configuration.
ctrl_data_out_enable	TDM block shares data out signal to driver with other controllers.
slot_mask	Specifies the active slots for channels in 32-bit format. – 0x00000001 – One active slot/channel at position 0. – 0x00000002 – One active slot/channel at position 1. – 0x00000003 – Two active slots/channels at position 0 and 1. – 0x0000000F – Four active slots/channels at position 0, 1, 2, and 3.
nslots_per_frame	Specifies the number of slots per frame. The slot number should be greater than or equal to the number of channels. – 0x0001 – Total number of slots – 1 – 0x0002 – Total number of slots – 2 – 0x0008 – Total number of slots – 8 – 0x0010 – Total number of slots – 16
slot_width	Specifies the bitwidth of the slot. The slot bitwidth should be greater than or equal to the number of the channel bitwidth. Supported bitwidths are 16, 24, and 32 bit.
sync_mode	TDM synchronization mode settings should be configured according to the hardware requirements of third-party devices. Supported modes are TDM_SHORT_SYNC_BIT_MODE, TDM_LONG_SYNC_MODE, and TDM_SHORT_SYNC_SLOT_MODE.
ctrl_invert_sync_pulse	Indicates whether to invert synchronization. These settings should be configured according to the hardware requirements of third-party devices. Supported modes are TDM_SYNC _NORMAL and TDM_SYNC_INVERT.
ctrl_sync_data_delay	Indicates the number of bit clock cycles for delaying data synchronization, supporting up to 3-bit clock delays. These settings should be configured according to the hardware

9. Assign the appropriate module tags. Typically, a module tag controls the channel count and corresponding channel mask of a source module. To modify module tags:

10. Ensure the TDM source is selected, Key Configurator is open, and the graph is still in the modification state.

11. In *Key Configurator*, select the *Module Tag* tab and click *+*.

12. In the *Select Tags* list, select the correct module tag. In this example, it is `device_hw_ep_tx`.

13. Once the tag is selected, a list of PIDs appears. In this list, select the *Support TKV* checkbox associated with the correct PID. In this example, it is `PARAM_ID_TDM_INTF_CFG`.

14. In the *Select Tag Key Values* list, select the correct tag key value. In this example, it is `CHS_1` for mono channel.

15. Select *Add*.

16. Repeat this process for CHS_2 for stereo channel support.

10. Once all channel configurations are added, click *Stop Graph Modification* and save the acdb files.

Troubleshoot and validate MIS2/TDM

This section checks that the sound card is properly registered. It also verifies I2S is recording from the PulseAudio level.

Note: Connect to the device console using SSH. See Use SSH for instructions.

Verify MI2S/TDM source capture

Check that the newly-added backend DAI registers the sound card:

```
cat /proc/asound/pcm
```

```
00-05: MI2S-LPAIF-TX-PRIMARY msm-stub-aif1-tx-5 :  : capture 1
```

If the backend DAI doesn't appear in the backend list, there may have been an issue adding the DAI links. Check the kernel logs to make sure there were no errors related to DAI links.

For TDM, the backend name will be `TDM-LPAIF_VA-TX-PRIMARY`.

Verify the device is recording from the PulseAudio level. This example is for a 48 kHz, stereo, 16-bit recording.

1. Update the *channels tab* in the `resourcemanager.xml` file to *2*.
2. Reboot the target.
3. Run the following command from a shell prompt. It should generate a file at `/opt/rec.wav` with the audio sent from the I2S device.

```
parec -v --rate=48000 --format=s16le --channels=2 --file-
format=wav /opt/rec.wav --device=regular2
```

The command shell should show a message similar to the following:

```
<nels=2 --file-format=wav /opt/rec.wav --device=regular2
Opening a recording stream with sample specification 's16le 2ch 48000Hz' and channel map 'front-left,front-right'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, fragsize=384000
Using sample spec 's16le 2ch 48000Hz', channel map 'front-left,front-right'.
Connected to device regular2 (index: 5, suspended: no).
Time: 51.075 sec; Latency: 25462 usec.
```

Figure8 Command shell example for verifying PulseAudio recording

Analyze logs

When capture for 48k/stereo/16-bit is done, the following logs should be seen.

- The I2S source has been modified for the speaker_mic device. The device open should be called for the speaker mic as follows:

```
Apr 29 15:50:46 pulseaudio[1001]: open: 469: Enter.
deviceCount 0 for device id 28 (PAL_DEVICE_IN_SPEAKER_MIC)
```

- The backend used for the capture is MI2S-LPAIF-TX-PRIMARY, with SR as 48000 kHz, and the number of channels as 2:

```
Apr 29 15:50:46 pulseaudio[1001]: setDeviceMediaConfig: 1056:
MI2S-LPAIF-TX-PRIMARY rate ch fmt data_fmt 48000 2 2 1
```

- The device open should exit with status 0 and a proper deviceCount for the speaker mic device:

```
Apr 29 15:50:46 pulseaudio[1001]: open: 504: Exit.
deviceCount 1 for device id 28 (PAL_DEVICE_IN_SPEAKER_MIC),
exit status: 0
```

- The hardware endpoints should be configured properly for 48k/stereo/16-bit as follows:

```
Apr 29 15:50:46 pulseaudio[1001]: configure_hw_ep_media_
config: 664 rate 48000 bw 16 ch 2, data_fmt 1
```

4.4 Audio module source code

If you have full access to the proprietary software shipped with Qualcomm Linux, view the audio module source code at:

Audio module	source code
PulseAudio	`build-qcom-wayland/workspace/sources/pulseaudio`
PAL	`build-qcom-wayland/workspace/sources/qcom-pal/opensource/arpal-lx`
TinyALSA	`build-qcom-wayland/workspace/sources/tinyalsa` `build-qcom-wayland/workspace/sources/tinycompress`
acdb files	`build-qcom-wayland/workspace/sources/qcom-acdbdata/opensource/audioreach-conf/ar-acdb/acdbdata`
ARGS	`build-qcom-wayland/workspace/sources/qcom-args/opensource/args`

4.5 Add custom audio modules

The Qualcomm® Hexagon™ SDK audio add-on is an audio plug-in that helps build and integrate audio modules into the SPF.

Next steps

- To get started, download the Hexagon SDK.
- To build and validate custom modules, read the Hexagon SDK documentation at `<Hexagonsdk_directory>/5.5.x/addons/audio/docs/spf/Module_Development/Module_Development.html`

  Note that eclipse-based IDE may not be available in all Hexagon SDKs. The preferred development option is command line.

5 Tune audio

Tune the Qualcomm aDSP module using the following required tools.

- QACT
- Qualcomm USB Drivers
- Qualcomm Unified Tools Service (QUTS)
- Qualcomm extensible Diagnostic Monitor (QXDM)

Download all tools from the Qualcomm Software Center.

For more details, see *QACT V8.1 User Guide* (80-VM407-21).

5.1 QACT overview

QACT helps to configure and calibrate voice and audio use cases and features that are available in the audio software architecture.

After installing QACT v8.1, configure the tool by following the steps in the user guide available at: `C:\\Users\\<User ID>\\AppData\\Local\\Qualcomm\\QACT 8.1\\80-VM407-21.pdf`.

5.2 QACT tuning modes and features

Offline calibration mode

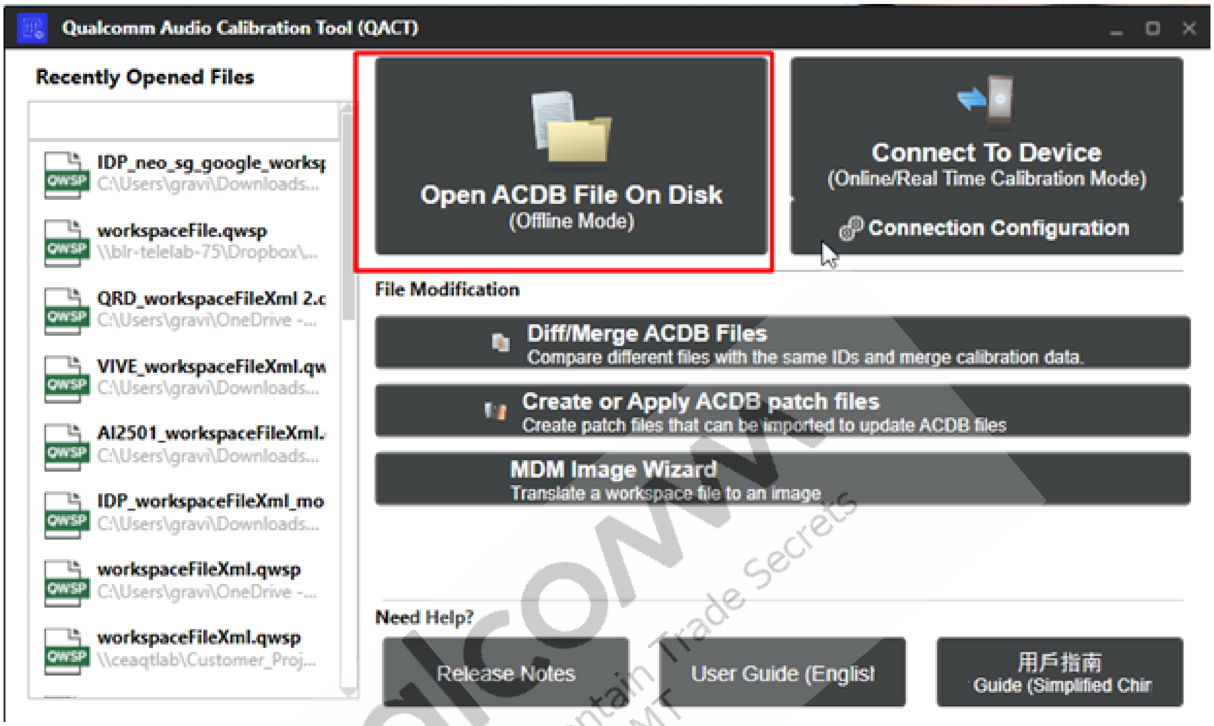


Figure1 Offline calibration mode

- Open and change calibration files without connecting to a target device.
- Push saved settings to a target device later.
- Loading updated parameters into the device memory requires a power cycle of the target device.

Online calibration mode

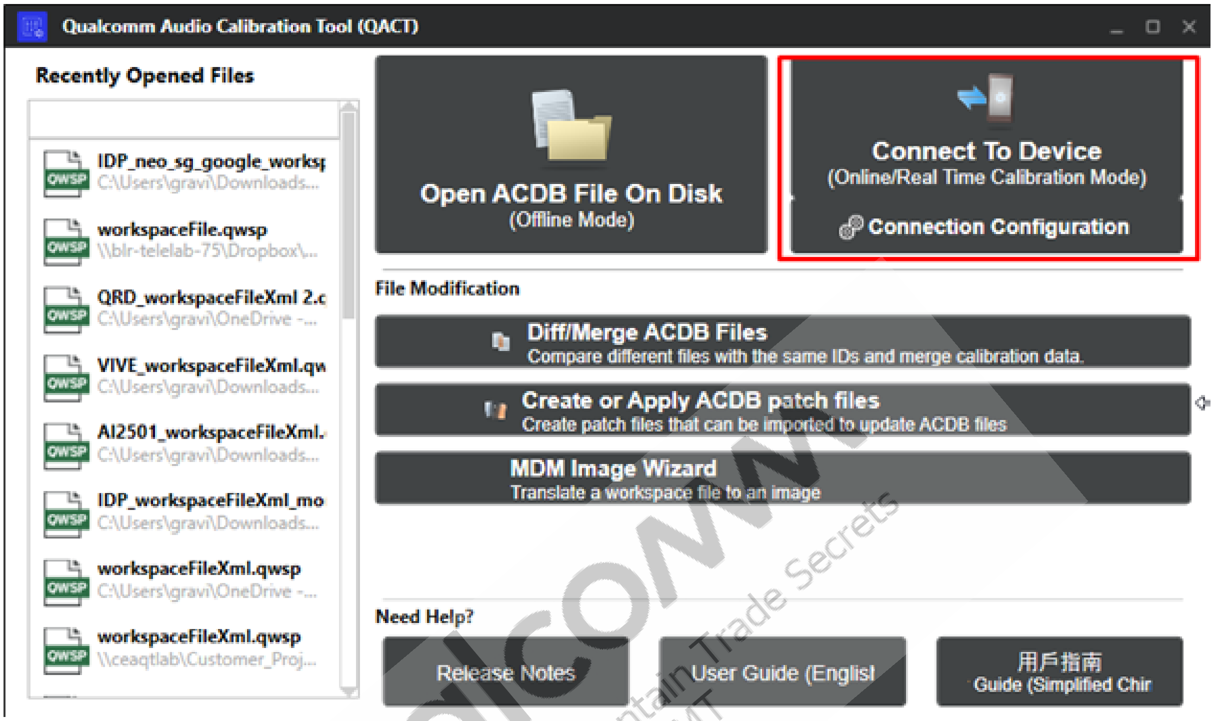


Figure2 Online calibration mode

- Calibrate acdb data currently present and in use by a device.
- Open, change, and save acdb data on the connected device using QACT for PC.
- Apply changes immediately until the device is power-cycled.
- If the device is already in a voice call, the changes are effective from the next voice call.
- For changes to persist, store the data as a database on the PC that you can later flash to the device.

Real-time calibration mode

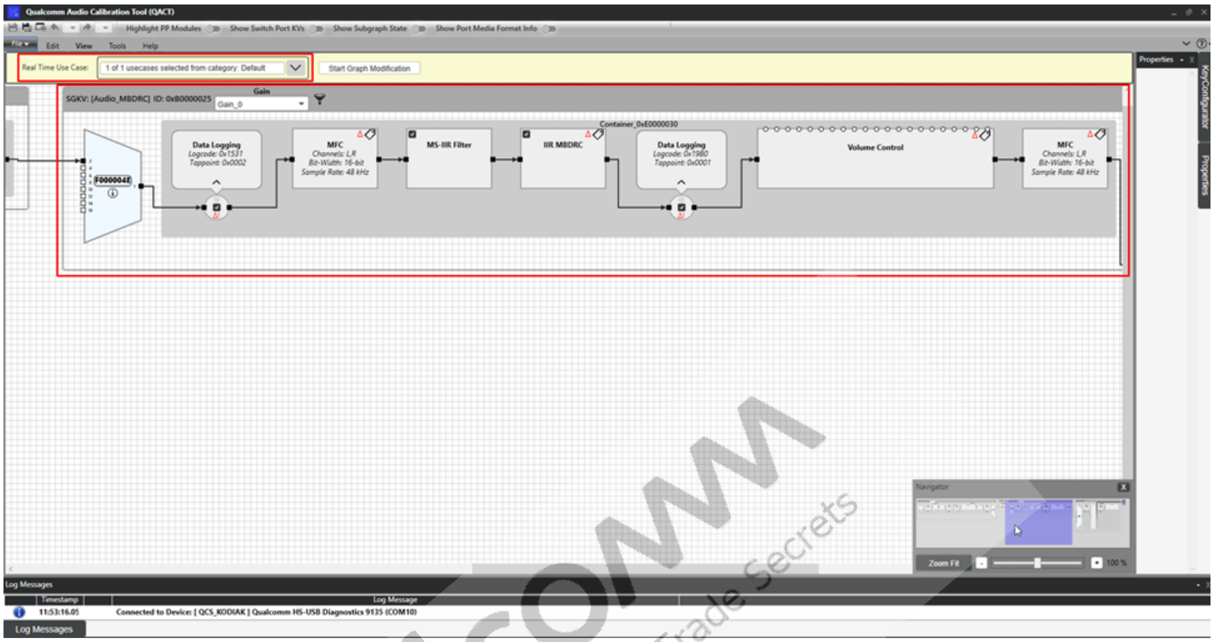


Figure3 Real-time calibration mode

- Change calibration data currently in use by the DSP of a connected target device.
- Real-time calibration (RTC) doesn't provide access to the entire calibration file, it only accesses calibration data that's currently in use by the DSP.
- It's *not* necessary to push updated calibration data to the target or force a device switch to load the updated calibration data.

Batch copy feature

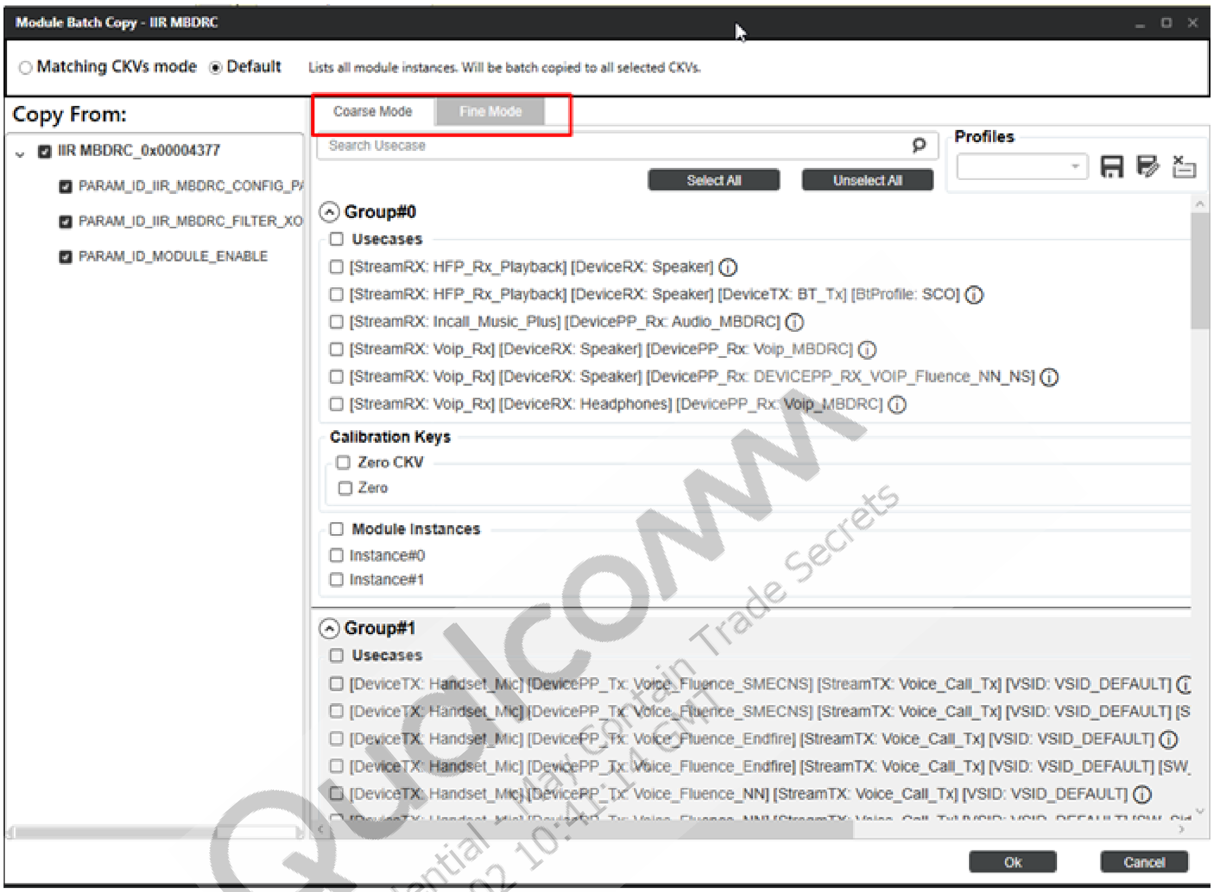


Figure4 Batch copy operations

QACT supports batch copy operations. These operations reduce the manual work of copying the tuning data from one use case to other use cases.

QACT supports the following operations:

- Model batch copy
- Use case batch copy

See *QACT V8.1 User Guide* (80-VM407-21), Section 5.4 for detailed information about this feature.

Diff/Merge feature

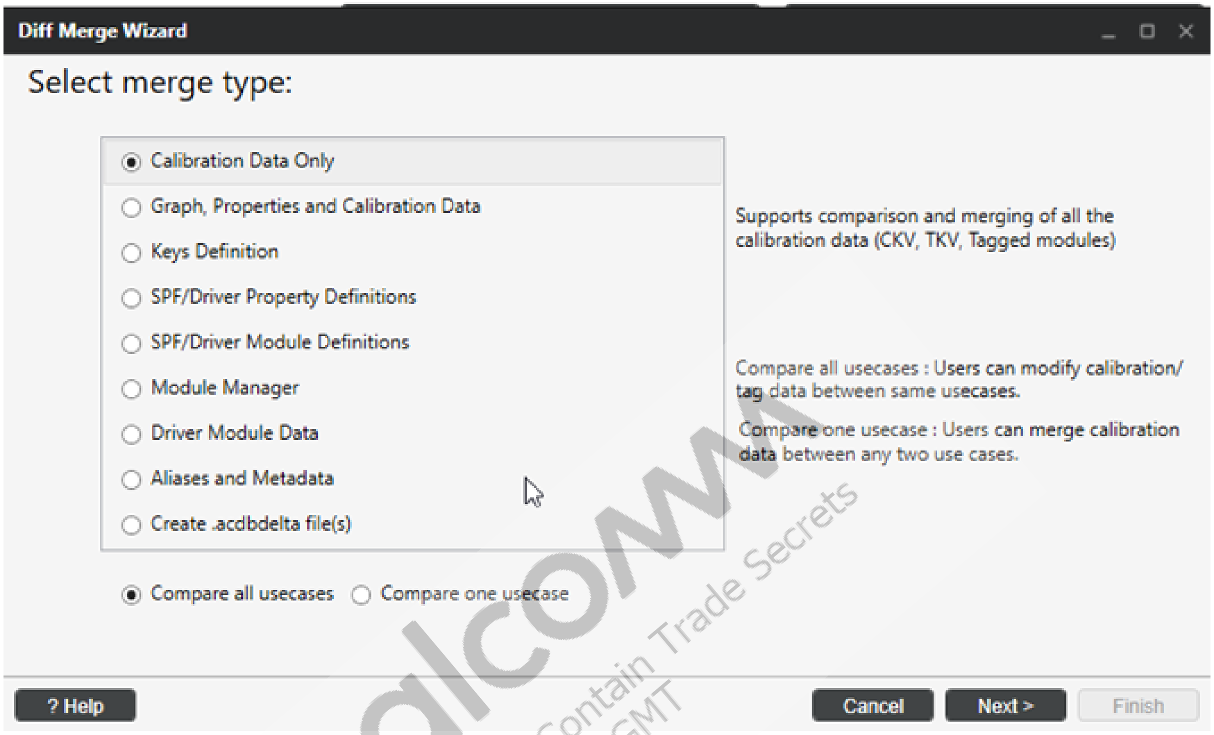


Figure5 Diff/Merge feature

Diff/Merge features help you find the differences between calibration, graphs, keys, and definitions between two acdb files. These features also help to add, delete, and update the supported use cases and definitions in acdb files.

See *QACT V8.1 User Guide* (80-VM407-21), Section 4.11 for detailed information about this feature.

5.3 Audio fine-tuning workflows

Workflows provide fine-tuning to improve audio.

Playback/RX_Path tuning workflow

The following example shows the default playback topology that has MS-IIR Filter and MBDRC modules.

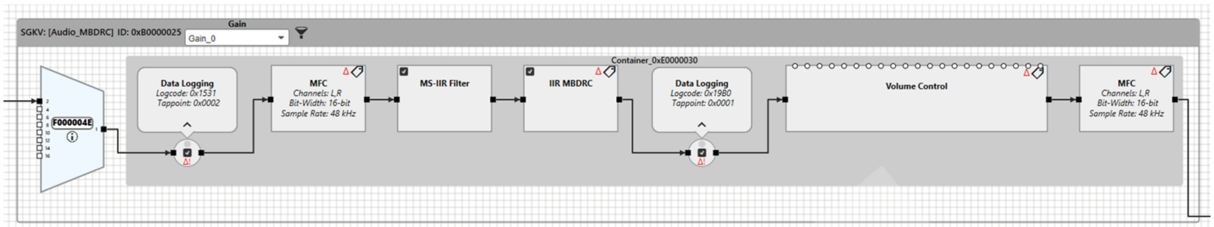


Figure6 Playback/RX_Path tuning workflow example

Playback tuning workflow

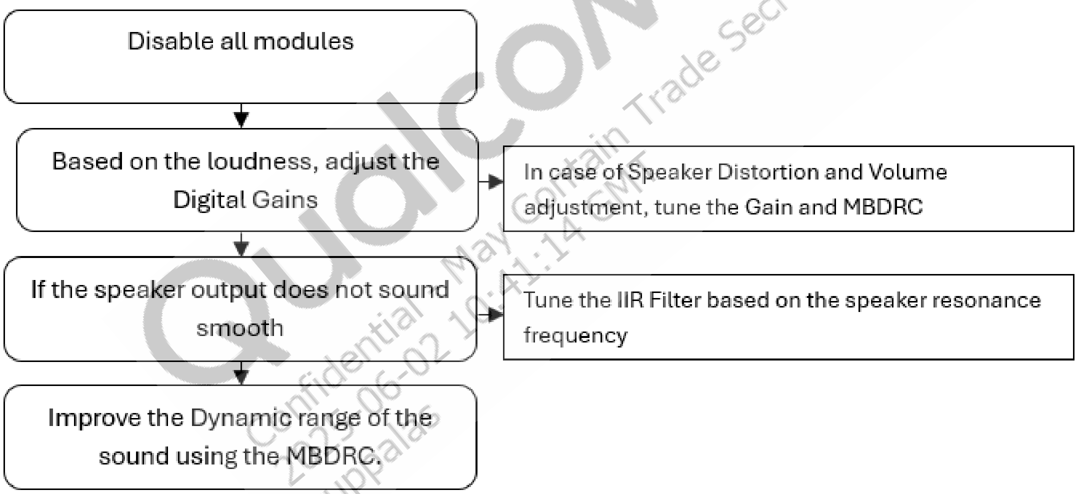


Figure7 Playback tuning workflow example

Recording tuning workflow

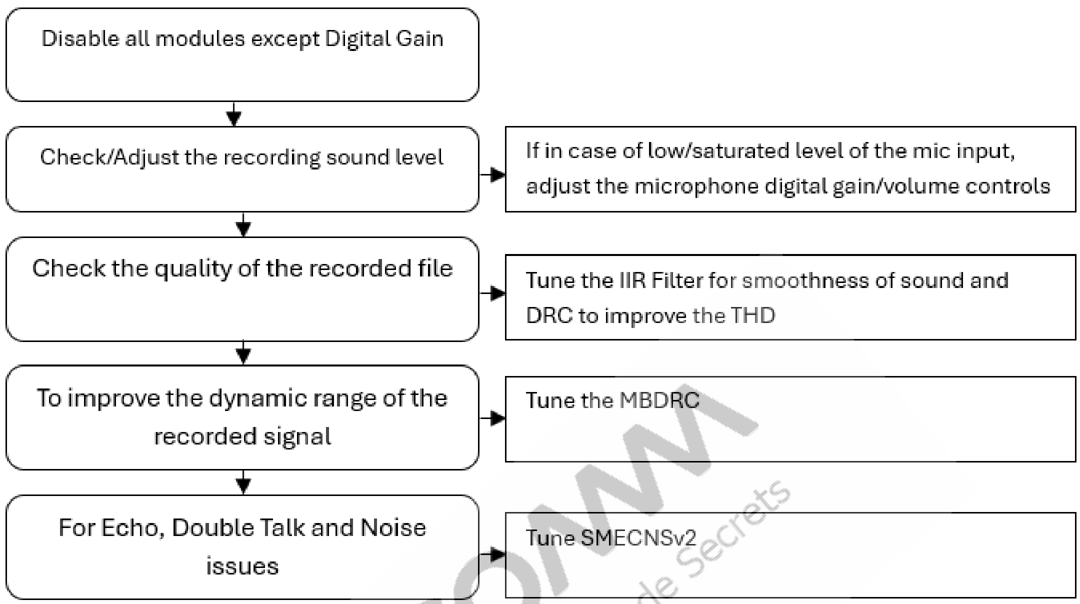


Figure8 Recording tuning workflow example

VoIP tuning workflow

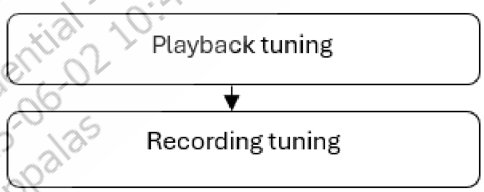


Figure9 VoIP tuning workflow example

5.4 Key audio modules

Key audio modules include SMECNS and MBDRC.

SMECNS_v2 for echo and noise

Single mic echo canceller and noise suppressor (SMECNS) helps cancel the echo from far-end and suppress the noise in the near end audio signal.

SEMCNS includes:

- Linear echo canceller – Cancel the linear echo
- Noise suppressor – Suppress the noise from the near end signal
- ECPP – Cancel the nonlinear echo

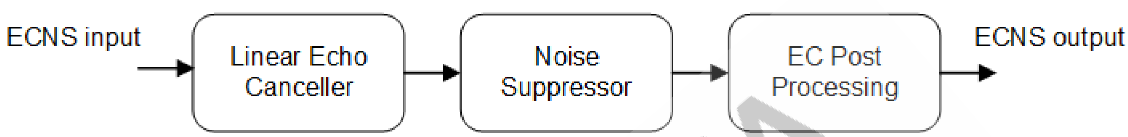


Figure10 SEMCNS process flow

SEMCNS supports the following audio devices and platforms:

- Handset
- Speaker
- Bluetooth

Use cases include:

- VoIP call uplink
- Audio recording

The SMECNS algorithm includes:

- Linear echo suppression (LEC)
- Nonlinear EC (ECPP)
- Noise suppression (NS)

LEC tuning

- Consists of a continuously adapting background (BG) filter and a stable foreground (FG) filter that only updates in favorable conditions (no divergence allowed).
- Evaluates BG coefficients for EC capability, robustness, and stability. Updates the FG filter with qualified BG coefficients. The strictness of the update is tunable.
- During double-talk, the BG filter diverges, while the FG filter retains qualified coefficients from an older frame (wherever the last update occurred).

Echo path change detection (EPCD)/double-talk detection (DTD)

- DTD analyzes the performance of both adaptive filters (BG and FG) to determine the presence of double-talk.
- EPCD analyzes and compares several signals to detect a change in the acoustic echo path. This is necessary in mobile devices where an echo path change can happen due to significant variations in device usage by end users.
- Tune the detectors and LEC in conjunction to get the best possible performance with the acoustic design of the device under test (DUT).

echo_path_delay_L16 (Q0)

- Adjust to capture the biggest peak in the echo path response as early as possible.
- Fine tune for both mobile originated (MO) and mobile terminated (MT), and tune this parameter for the worst case scenario.

Tune AF coefficients amplitude

- `AF_Taps_Qfac` – Increase up to 4 in steps of 1 until the coefficient amplitude is under 8192.
  - If increasing it doesn't produce a noticeable change on the filter coefficients or on the output of the LEC, then check:
    - If echo is saturated already
    - Digital gain before SMECNS V2
    - Analog gain
- `AF_BG_ERLE_Thresh` – Increase this parameter to relax the update criteria and update the FG more often. This also improves the convergence of the filter. For Handset/Headset modes, don't increase above 0x4E20.
- `AF_DT_Thresh` – Verify that there are no false double-talk detections that may prevent updates of the FG filter. In all modes, don't increase above 0x7D00.

ECPP tuning

Tune for far-end single-talk/double talk performance

- `PP_Gamma_e_High` – Applies more echo muting gain during far-end speech. A higher value means more muting.
- `DENS_gamma_e_low` – Applies more echo muting gain during near-end speech. A higher value means more muting.
- `PP_Gamma_e_DT` – Attenuates residual echo energy after LEC during double-talk.
  - Typically kept at the same value as `PP_Gamma_e_High`.
  - Decrease if attenuating the intended near-end speech during double-talk.
- `PP_AggQ` – Adjusts the Q-factor for `PP_Gamma_e_High` and `PP_Gamma_e_DT` to increase

or decrease the echo suppression aggressiveness as required.

- PP_Gamma_NL – Overestimates the nonlinear echo estimate and can be used in situations where the amount of nonlinear echo is high.
- PP_CNI_Level – Increasing this parameter increases the amount of comfort noise injected.
- DENS_tail_alpha – Represents the decay in energy of the echo tail of the impulse response.
- DENS_tail_portion – Represents the power estimated of the echo tail.

NS tuning

- Voice activity detection (VAD)
  - SMECNS_V2 employs an SNR-based VAD, which provides more exact single-channel speech detection compared to traditional energy-based detectors.
  - Detection of speech allows control of noise reference updates, as well as postprocessing.
- Noise reference
  - Combines different types of noise references to provide a stronger noise estimate.
  - Individually turn on/off and scale each type of noise reference to provide the best overall noise reference.
- NS postprocessing
  - Individually control subtraction in the low, mid, and high frequencies to provide greater control of NS aggressiveness.
- NS CNI
  - Increase CNI_Level to increase the amount of comfort noise.
  - This parameter is important for EC_CNI to work. EC_CNI tries to match the noise floor at the NS output.
- DENS_gamma_n – Noise subtraction factor in spectral gain function. Recommended range: 0x200 to 0x320.
- DENS_NFE_blockSize – Higher value results in lower noise suppression. Recommended range: 0x96 to 0x190.
- DENS_limit_NS – Controls maximum noise suppression LEVEL. A lower value results in more noise suppression. Recommended range: 0xC00 to 0x4000.
- fnsSalpha – Increase this parameter value to increase the NS aggressiveness. Recommended range: 0x1000 to 0x2000.
- fnsTargetNS – Increase this parameter value to increase the noise suppression level. Recommended range: 0x600 to 0x1400.

- `fnsSNblock` – Decrease this parameter value if the noise suppression convergence is too slow. Recommended range: 0x28 to 0x4B.

MBDRC for multiband dynamic rate control

The multiband dynamic rate control (MBDRC) feature provides DRC in more than one band.

DRC allows tracking changes dynamically in the audio signal and normalizing output gain level to improve the perceived loudness without affecting overall signal level. MBDRC provides this in multifrequency bands.

Apply MBDRC for different frequency regions to avoid low-frequency distortion on small speakers and retain or boost natural-sounding content on other frequency regions. Well-tuned MBDRC modifies the dynamic range of the signal in different frequency bands without distorting the speaker in the low frequencies. With make up gain, the overall audio can sound louder than the original while preventing playback distortion.

DRC units are nonlinear operations that adjust the dynamic range of an audio signal. This changes the perceived loudness, noise level, and some subtle musical/artistic characteristics according to your design.

The DRC module has a downward compressor (attenuates loud signal), an upward compressor (amplifies quiet signals), and a downward expander (soft noise gate). Signal RME energy estimation triggers these signals. MBDRC is available in both Tx and Rx paths.

DRC changes the signal's dynamic range by automatically adjusting the gain based on the signal's short term RMS level estimated in real time.

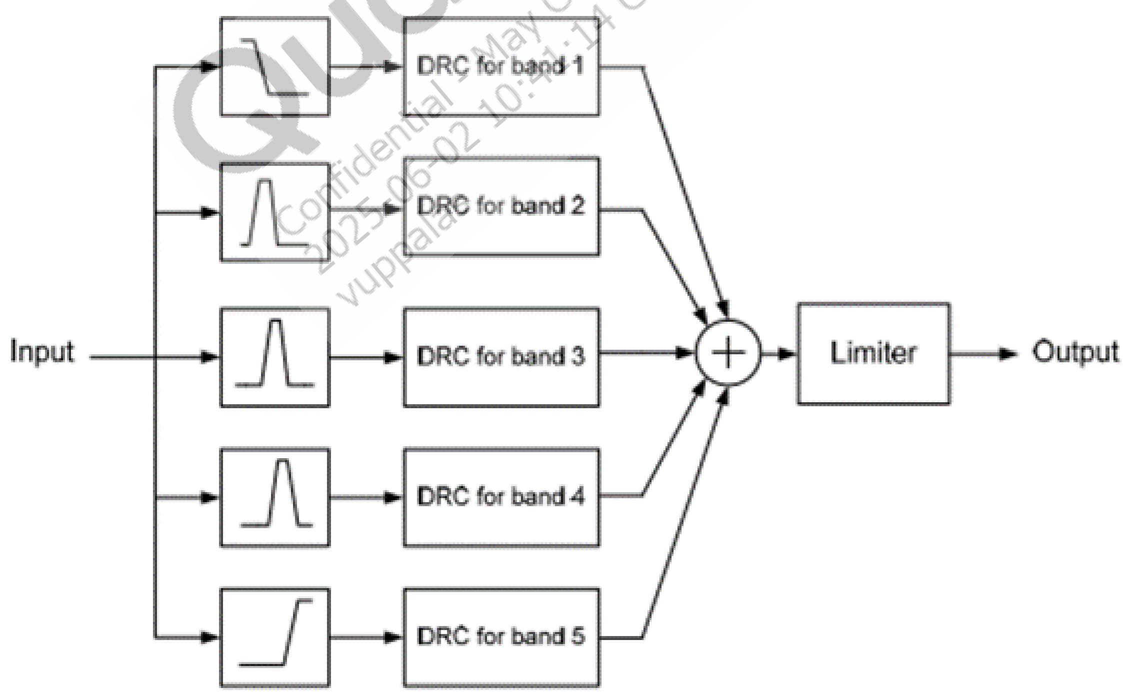


Figure11 DRC process flow

The following thresholds split the signal into four ranges:

- dwCT – Downward compression threshold
- uwCT – Upward compression threshold
- dwET – Downward expansion threshold

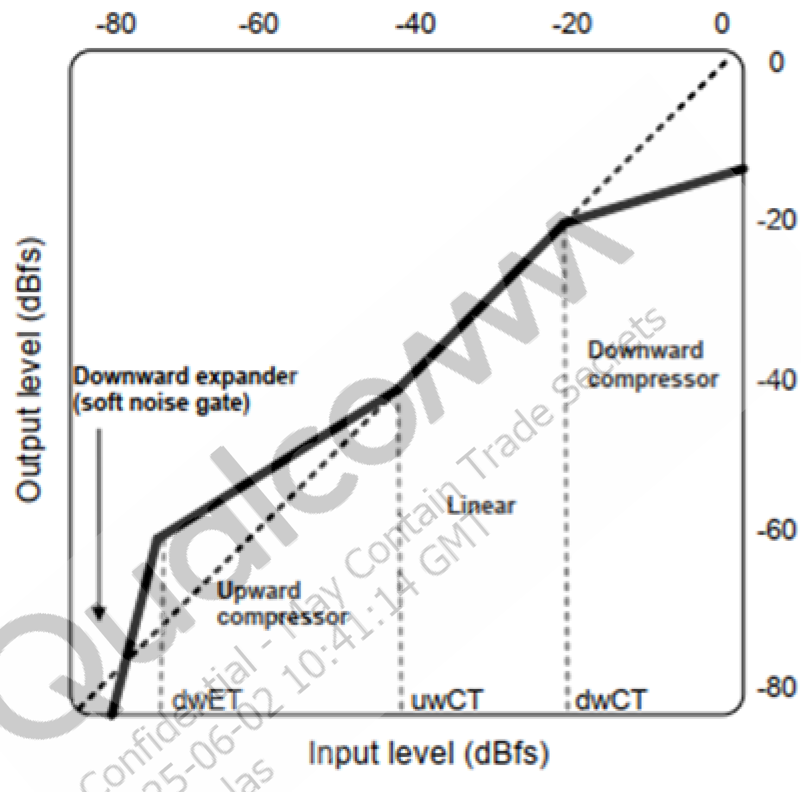


Figure12 Signal range thresholds

- Depending on the input signal level (RMS), the following scenarios are possible:
  - dwCT < RMS level < 0 dB – The signal is too loud, and the DRC applies compression to the loud signal. The preferred gain to apply to the signal is negative.
  - uwCT < RMS level < dwCT – The linear range, where the DRC doesn't change the input. It's recommended to have the signal in the linear region as much as possible to minimize nonlinear distortions.
  - dwET < RMS level < uwCT – This region applies to signals that carry information, but are soft, such as notes from a musical instrument that you can't hear clearly. The preferred gain to apply to the signal is positive.
  - RMS level < dwET – The signal is small and noise-like. Therefore, applies a negative gain to the signal.

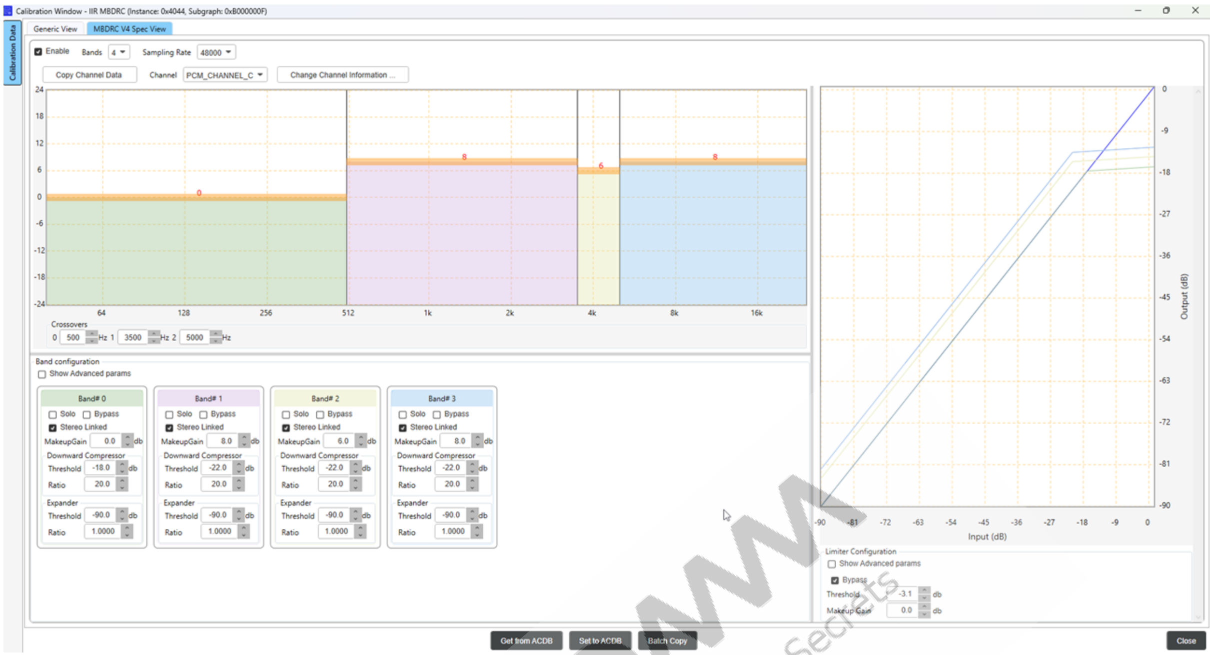


Figure13 MBDRC calibration view

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.