

# **QCS610/QCS410 Linux Platform Development Kit**

## **Software Reference Manual**

80-PL631-100 Rev. M

July 6, 2021

Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

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

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

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to [DocCtrlAgent@qualcomm.com](mailto:DocCtrlAgent@qualcomm.com).

**Confidential Distribution:** Use or distribution of this item, in whole or in part, is prohibited except as expressly permitted by written agreement(s) and/or terms with Qualcomm Incorporated and/or its subsidiaries.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

All Qualcomm products mentioned herein are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm, Adreno, Chromatix, Hexagon, Kryo, MSM, QACT, QXDM Professional, and Snapdragon are trademarks or registered trademarks of Qualcomm Incorporated. QuRT and Quick Charge are trademarks of Qualcomm Incorporated. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

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

# Revision\_history

---

Revision	Date	Description
A	December 2019	Initial release
B	March 2020	Numerous changes were made to this document. It should be read in its entirety.
C	April 2020	Numerous changes were made to this document for ES release. It should be read in its entirety.
D	May 7, 2020	Updated the DisplayPort concurrency value in Section 5.2.2.4 <i>USB display</i>
E	May 20, 2020	Numerous changes were made to this document for FC release. It should be read in its entirety.
F	June 2020	Added the multiframe shot details in Section 6.4.6 and P-Iris details in Section 6.4.7
G	July 2020	Numerous changes were made to this document for CS release. It should be read in its entirety.
H	August 2020	<ul style="list-style-type: none"><li>■ Added <i>Qualcomm® Computer Vision SDK</i> details in Section 6.3.3.</li><li>■ Added <i>Audio Jack detection and implementation</i> details in Section 6.2.1</li><li>■ Added <i>PulseAudio audio server details</i> in Section 6.2.3</li><li>■ Added the <i>QCS410 LEPDK use cases</i> in Chapter 8.</li><li>■ Numerous updates have been made to the multimedia use cases in Chapter 7; read the chapter in its entirety</li></ul>
<i>Revision I was omitted in accordance with QTI document conventions.</i>		
J	October 2020	<ul style="list-style-type: none"><li>■ Updated the sensor configuration in Section 5.6</li><li>■ Added audio source tracking trio details in Section 6.2.4</li><li>■ Updated the sDHR version details for single and dual virtual connector, and added autofocus details in Section 6.4.2</li><li>■ Numerous changes were made to the use cases in Chapter 7; read the chapter content in its entirety.</li><li>■ Updated qtivdec pad properties in Appendix B.2</li><li>■ Updated qtioverlay pad properties in Appendix B.6</li><li>■ Added qtihexagonnn details in Appendix B.11</li></ul>
K	December 2020	<ul style="list-style-type: none"><li>■ Updated the LDC support information in Section 6.4.3</li><li>■ Updated the EIS data flow steps in Section 6.4.4</li><li>■ Added EIS with horizon leveling details in Section 6.4.5</li><li>■ Added ISP support in BPS path details in Section 6.4.11</li><li>■ Added JPEG DMA support details in Section 6.4.12</li></ul>

Revision	Date	Description
		<ul style="list-style-type: none"><li>■ Updated details on parameters that can be updated dynamically in Section 7.8.2</li><li>■ Updated the QCS410 multimedia use case details in Chapter 8</li></ul>
L	February 2021	Updated QCS410 use case details in Chapter 8
M	July 2021	<ul style="list-style-type: none"><li>■ Added dependency on LDC and EIS details in Section 6.4.5</li><li>■ Updated details on the IR mode feature in Section 6.4.11</li><li>■ Updated details on rate control in Section 6.5.1</li><li>■ Updated the pipeline options throughout Chapter 7</li><li>■ Updated the dynamic parameters in Section 7.8.2</li><li>■ Added details on ROI-based encoding in Section 7.11.3</li><li>■ Updated the gst-tracking-cam-app details in Section 7.17</li><li>■ Added the UVC and UAC architecture in Section 7.18</li><li>■ Updated the properties in Tables B-2, B-3, and B-7</li><li>■ Updated the gst-pipeline-app details in Appendix C</li></ul>

Qualcomm Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

# Contents

---

Revision_history .....	2
1 Introduction to QCS610/QCS410 chipset .....	17
1.1 Conventions .....	17
1.2 Technical assistance .....	17
2 Overview of Linux platform development kit .....	18
2.1 LEPDK for QCS610/QCS410 .....	18
3 QCS610/QCS410 HLOS software architecture – LEPDK framework .....	19
3.1 Source tree structure .....	19
3.2 QTI metadata layers .....	21
3.2.1 BSP .....	21
3.2.2 Data .....	26
3.3 Build optimizations .....	28
3.4 QTI-defined variables in PDK .....	28
4 QCS610/QCS410 feature overview .....	29
5 QCS610/QCS410 non-HLOS software architecture .....	32
5.1 Boot .....	32
5.1.1 Boot configuration .....	35
5.2 Peripherals .....	36
5.2.1 QUP v3 .....	36
5.2.2 USB .....	39
5.2.3 RGMII .....	48
5.3 Storage .....	48
5.3.1 Storage features .....	51
5.3.2 Storage configuration .....	52
5.4 PMIC .....	52
5.5 RPMh .....	52
5.5.1 Regulator management .....	55
5.5.2 Frequency management .....	59
5.5.3 Sleep management .....	63

5.5.4 SoC sleep .....	65
5.6 Sensors .....	66
5.6.1 LE software product-based platform .....	67
5.6.2 High-level platform overview .....	69
5.6.3 Sensors configuration .....	69
6 QCS610/QCS410 HLOS software architecture .....	71
6.1 Display .....	71
6.1.1 Display architecture .....	72
6.1.2 Display subsystem .....	73
6.1.3 Layer mixer .....	74
6.1.4 Destination surface processor .....	74
6.1.5 Display interface .....	75
6.1.6 Direct rendering manager and Kernel mode setting .....	75
6.2 Audio .....	77
6.2.1 Audio features .....	84
6.2.2 Audio configuration .....	91
6.2.3 PulseAudio .....	91
6.2.4 Source tracking trio .....	92
6.3 Hexagon Compute DSP .....	96
6.3.1 Hexagon cDSP with HVX .....	98
6.3.2 cDSP software architecture .....	99
6.3.3 Qualcomm Computer Vision SDK .....	101
6.3.4 Camera streaming .....	106
6.4 Camera .....	108
6.4.1 Camera features .....	110
6.4.2 sHDR overview .....	111
6.4.3 LDC overview .....	114
6.4.4 EIS overview .....	114
6.4.5 Dependency before running EIS and LDC .....	115
6.4.6 EIS with horizon leveling .....	116
6.4.7 Defog overview .....	116
6.4.8 Multiframe shot – MFNR .....	116
6.4.9 P-Iris .....	128
6.4.10 Live Tuning .....	130
6.4.11 IR mode .....	130
6.4.12 ISP support in BPS path .....	132
6.4.13 JPEG DMA support .....	132
6.4.14 Camera driver .....	133

6.5 Video .....	135
6.5.1 Video features .....	139
6.5.2 Video concurrency .....	144
6.6 Graphics .....	146
6.6.1 Graphics features .....	148
6.7 Security .....	151
6.7.1 TZ and secure application .....	152
6.7.2 Qualcomm TEE 5.0 .....	157
6.7.3 Secure boot and QFPROM .....	160
6.7.4 HLOS security .....	161
6.7.5 Device access control .....	163
6.7.6 Crypto core and software crypto modules .....	165
6.7.7 Secure storage .....	165
6.7.8 Secure device debugging .....	165
6.7.9 Crypto APIs .....	166
6.7.10 BSP binary protection .....	166
6.7.11 Root process audit .....	166
6.7.12 Full disk encryption .....	167
6.8 Thermal .....	172
6.8.1 Temperature sensors .....	172
6.8.2 Thermal management software .....	173
6.8.3 Boot thermal management .....	174
6.8.4 Thermal core framework .....	176
6.8.5 User space thermal engine .....	177
6.8.6 Thermal mitigation algorithms .....	179
6.8.7 Thermal engine configuration .....	182
6.8.8 Thermal zones .....	183
6.8.9 Other thermal software features .....	186
6.9 Power data flows .....	186
6.10 Performance and memory .....	187
6.11 ML .....	188
6.11.1 TensorFlow Lite .....	189
6.11.2 Qualcomm® Neural Processing SDK .....	190
6.12 Docker support .....	190
7 QCS610 LEPDK use cases .....	193
7.1 Prerequisites .....	193
7.2 Multimedia stack and control flow .....	193
7.2.1 Camera .....	194

7.2.2 Display and graphics .....	195
7.2.3 Video .....	196
7.2.4 Audio .....	199
7.2.5 GStreamer ML .....	200
7.3 Player– Single-stream playback .....	201
7.3.1 Start Weston .....	201
7.3.2 4K video file playback .....	202
7.3.3 1080p video file playback .....	202
7.4 Player – Multi-stream playback .....	203
7.4.1 Eight stream playback, seven 720p offline video, and one 720p live camera preview .....	203
7.4.2 Four 1080p stream side-by-side playback .....	205
7.4.3 Two 1080p stream picture in picture playback .....	206
7.5 Recorder – Two stream concurrencies .....	206
7.5.1 Two independent H264 streams both stored to file .....	207
7.5.2 Two independent H264 streams, one stored to file and another streamed over TCP .....	208
7.5.3 Two independent streams, one H264 stream stored to file and one live camera preview .....	209
7.6 Recorder – Three stream concurrencies .....	210
7.6.1 Three video streams all of them stored to file .....	210
7.6.2 Three video streams, one stored to file, another streamed over TCP, and another live preview on display .....	211
7.7 Recorder – Five stream concurrencies .....	212
7.7.1 Five H264/AVC encoded streams with sHDR enabled .....	212
7.7.2 Five H264/AVC encoded streams with sHDR and LDC enabled .....	214
7.8 Camera parameters .....	215
7.8.1 Static parameters .....	215
7.8.2 Dynamic parameters update using the gst-pipeline-app .....	216
7.9 Snapshot .....	219
7.9.1 JPEG and Bayer snapshot .....	220
7.10 Overlay .....	221
7.10.1 Live camera preview with overlay .....	221
7.10.2 Live video recording with overlay .....	223
7.11 Video encode .....	224
7.11.1 Static parameter update .....	224
7.11.2 Dynamic bitrate update .....	224
7.11.3 ROI-based encoding .....	226
7.12 Video composition .....	230
7.12.1 PiP – 1080p camera preview and 1080p recorded h264 video file playback .....	231
7.12.2 Video tiling – Multiple video streams tiled in one output .....	232

7.13 ML . . . . .	233
7.13.1 Single stream Qualcomm Neural Processing SDK inference with live camera . . . . .	233
7.13.2 Single stream Qualcomm Neural Processing SDK inference on offline video . . . . .	234
7.13.3 Single stream TensorFlow Lite inference on offline video and stored to file . . . . .	234
7.13.4 Single stream live camera TensorFlow Lite inference streamed over RTSP – SSD . . . . .	236
7.13.5 Two stream Qualcomm Neural Processing SDK inference with live camera – SSD . . . . .	238
7.13.6 Single stream live camera preview with DirectNN inferencing . . . . .	239
7.14 UBWC control . . . . .	241
7.14.1 Live preview using in memory compression with video record . . . . .	241
7.14.2 1080p video downscale to 720p . . . . .	242
7.15 Live streaming over RTSP . . . . .	243
7.15.1 1080p live streaming over RTSP . . . . .	243
7.15.2 Two H264 streams – one stored and another streamed over RTSP . . . . .	243
7.16 Parallel 4K encode and 1080p decode . . . . .	244
7.17 Video conferencing . . . . .	245
7.18 UVC and UAC architecture . . . . .	246
7.18.1 Run UVC + UAC . . . . .	248
8 QCS410 LEPDK use cases . . . . .	250
8.1 Record 1080p hardware-encoded H264 video . . . . .	250
8.2 Live snapshot – 1080p snapshot while a 1080p streaming is running . . . . .	250
9 Connectivity . . . . .	251
A System debugging and features . . . . .	252
A.1 Debug features and infrastructure . . . . .	252
A.1.1 On-target debug features . . . . .	252
A.1.2 Off-target debug features . . . . .	257
A.2 Debugging approaches . . . . .	259
A.2.1 On-target debugging . . . . .	259
A.2.2 Offline debugging . . . . .	259
A.3 Types of errors during software failure . . . . .	259
A.3.1 General error notification . . . . .	262
A.3.2 Software errors . . . . .	265
A.4 Watchdog . . . . .	266
A.4.1 Watchdog for AOP . . . . .	268
A.4.2 Watchdog for other subsystems . . . . .	268
A.4.3 Log analysis . . . . .	269
A.5 Erroneous transaction on bus (error and timeout) . . . . .	270
A.6 Hardware reset . . . . .	271

A.6.1 PMIC PON reset status .....	273
A.7 System hang during power collapse .....	274
B QTI GStreamer elements .....	276
B.1 qtiqmmfsrc .....	276
B.2 qtivdec .....	285
B.3 omxh264enc .....	286
B.4 waylandsink .....	289
B.5 qtivtransform .....	290
B.6 qtioverlay .....	292
B.7 qtimlesnpe .....	296
B.8 qtimletflite .....	299
B.9 qtivcomposer .....	302
B.10 omxaacenc .....	305
B.11 qtihexagonnn .....	307
C QTI GStreamer reference applications .....	310
C.1 QTI gst-pipeline-app .....	310
D References .....	313
D.1 Related documents .....	313
D.2 Acronyms and terms .....	314

# Tables

---

Table 3-1: Console distros.....	23
Table 3-2: Partition configuration file components.....	26
Table 5-1: Boot options and configuration.....	36
Table 5-2: QUP v3 GPIO assignment.....	37
Table 5-3: USB overview.....	39
Table 5-4: Linux USB features.....	39
Table 5-5: USB Type-C and USB PD software drivers.....	42
Table 5-6: USB analog audio software drivers.....	43
Table 5-7: USB charging software drivers.....	45
Table 5-8: DisplayPort concurrency orientation.....	46
Table 5-9: USB and DisplayPort concurrency software drivers.....	46
Table 5-10: USB controller software drivers.....	46
Table 5-11: USB Type-C configuration.....	47
Table 5-12: USB kernel configuration.....	47
Table 5-13: Optional configuration.....	47
Table 5-14: USB PD configuration.....	48
Table 5-15: Storage SD hardware/software specifications.....	49
Table 5-16: eMMC Linux storage features.....	51
Table 5-17: SD card Linux storage features.....	52
Table 5-18: SDIO Linux storage features.....	52
Table 5-19: SDHC kernel configuration.....	52
Table 5-20: Resource voters.....	54
Table 5-21: RPMh tasks.....	54
Table 5-22: QTI reference platform sensors.....	67

Table 5-23: Recommended SSC GPIOs and QUP usage.....	67
Table 5-24: GPIO numbers for different sensor interrupts.....	68
Table 5-25: Sensors power rails on QCS610.....	68
Table 5-26: QCS610/QCS410 sensor hardware specifications.....	69
Table 6-1: Features tested with APT levels.....	77
Table 6-2: Customization guidelines.....	91
Table 6-3: cDSP hardware specification.....	96
Table 6-4: cDSP features.....	97
Table 6-5: QCS610/QCS410 camera ISP features.....	110
Table 6-6: QCS610/QCS410 image quality (IQ) features.....	111
Table 6-7: QCS610 video format.....	137
Table 6-8: Adreno VPU433 codec specification for decoder.....	137
Table 6-9: Encoder specification.....	138
Table 6-10: Error resiliency features.....	142
Table 6-11: Multi-instance decode.....	144
Table 6-12: HFR encode.....	145
Table 6-13: Video encoder preprocessing.....	145
Table 6-14: Closed-source Adreno graphics libraries.....	147
Table 6-15: QCS610 Adreno graphics key specifications.....	148
Table 6-16: Supported HLOS security features.....	161
Table 6-17: Crypto core details.....	165
Table 6-18: Secure storage support for specific use cases.....	165
Table 6-19: Debug utility details.....	166
Table 6-20: FDE components.....	167
Table 6-21: Thermal software mitigation devices.....	179
Table 6-22: Power KPIs.....	186
Table 6-23: TensorFlow Lite performance numbers on QCS610.....	189
Table 7-1: ROI properties.....	227
Table 7-2: Use cases for video conferencing.....	246
Table 7-3: Formats and resolutions.....	247
Table A-1: Protection domain restart test cases.....	254

Table A-2: Force subsystem reset test cases.....	254
Table A-3: Types of watchdog.....	266
Table A-4: Hardware AOSS_CC_RESET_STATUS.....	272
Table A-5: PON reason.....	274
Table A-6: Hardware PMIC PON reason.....	274
Table A-7: Hardware warm reset reason.....	274
Table B-1: qtqmmfsrc pad templates.....	276
Table B-2: qtqmmfsrc properties.....	277
Table B-3: qtqmmfsrc video pad properties.....	282
Table B-4: qtivdec pad templates.....	285
Table B-5: qtivdec properties.....	286
Table B-6: omxh264enc pad templates.....	287
Table B-7: omxh264enc properties.....	287
Table B-8: waylandsink pad templates.....	289
Table B-9: waylandsink properties.....	290
Table B-10: qtivtransform pad templates.....	291
Table B-11: qtivtransform properties.....	291
Table B-12: qtioverlay pad templates.....	293
Table B-13: qtioverlay properties.....	293
Table B-14: qtimlesnpe pad templates.....	297
Table B-15: qtimlesnpe properties.....	298
Table B-16: qtimletflite pad templates.....	300
Table B-17: qtimletflite properties.....	301
Table B-18: qtivcomposer pad templates.....	303
Table B-19: qtivcomposer properties.....	304
Table B-20: qtivcomposer sink pad properties.....	304
Table B-21: omxaacenc pad templates.....	306
Table B-22: omxaacenc properties.....	306
Table B-23: qtihexagonnn pad templates.....	308
Table B-24: qtihexagonnn properties.....	308

# Figures

---

Figure 3-1: Linux PDK architecture.....	19
Figure 3-2: Directories in workspace after the source code is downloaded.....	20
Figure 3-3: Directories in workspace after the build is complete.....	21
Figure 5-1: Boot flow diagram.....	33
Figure 5-2: UEFI.....	35
Figure 5-3: QUP configuration example.....	37
Figure 5-4: SPI - serial communication between master and slave.....	39
Figure 5-5: USB configuration.....	41
Figure 5-6: USB analog audio.....	43
Figure 5-7: USB charging.....	44
Figure 5-8: USB and DisplayPort concurrency.....	45
Figure 5-9: eMMC block diagram.....	49
Figure 5-10: SD card block diagram.....	50
Figure 5-11: SDC 1.....	51
Figure 5-12: SDC 2.....	51
Figure 5-13: RPMh architecture.....	53
Figure 5-14: RPMh functionality.....	53
Figure 5-15: Regulator control in RPMh.....	55
Figure 5-16: Regulator control for vote on digital voltage domain.....	56
Figure 5-17: Regulator control for vote on non digital voltage domain.....	57
Figure 5-18: CPR micro adjustments for voltage and regulator domains.....	58
Figure 5-19: Frequency control in RPMh.....	59
Figure 5-20: RPMh frequency control through bandwidth votes.....	60
Figure 5-21: RPMh frequency control for memory and DDR subsystem.....	61

Figure 5-22: RPMh frequency control for memory through bandwidth votes.....	62
Figure 5-23: Sleep management for RPMh.....	63
Figure 5-24: RPMh workflow for entering Sleep state.....	64
Figure 5-25: RPMh workflow for exiting Sleep state.....	65
Figure 5-26: SEE software diagram.....	66
Figure 6-1: Display software architecture.....	72
Figure 6-2: Audio system overview (with Internal codec).....	77
Figure 6-3: Audio system overview (with external codec WCD9341).....	78
Figure 6-4: Jack detection call flow.....	79
Figure 6-5: Audio software architecture.....	80
Figure 6-6: Audio playback.....	81
Figure 6-7: Audio playback recording.....	83
Figure 6-8: Device and stream management.....	85
Figure 6-9: Split Bluetooth Advanced Audio Distribution Profile (A2DP).....	86
Figure 6-10: Split A2DP – data path.....	87
Figure 6-11: USB Digital Audio Tunnel Mode data path.....	88
Figure 6-12: Audio multicomponent architecture.....	89
Figure 6-13: Qualcomm Noise and Echo Cancellation Pro v2 source tracking UI display - sample.....	93
Figure 6-14: Sound Focus sector definition for Qualcomm Audio and Voice Communications suite.....	95
Figure 6-15: dspCV Library.....	102
Figure 6-16: Camera streaming – HVX tap points.....	107
Figure 6-17: Camera Memory-to-Memory Pre/Postprocessing.....	107
Figure 6-18: Camera streaming – computational camera.....	108
Figure 6-19: Camera high-level block diagram.....	109
Figure 6-20: sHDR 3.8 data flow for dual VC.....	112
Figure 6-21: sHDR 2.0 data flow for single VC.....	112
Figure 6-22: sHDR3.8 with autofocus.....	113
Figure 6-23: sHDR 2.0 with autofocus.....	113
Figure 6-24: LDC data flow.....	114
Figure 6-25: EIS data flow.....	114
Figure 6-26: MFNR processing – Prefiltering.....	119

Figure 6-27: MFNR processing – Blending.....	119
Figure 6-28: MFNR processing – Postfiltering.....	120
Figure 6-29: P-Iris gain and shutter speed.....	128
Figure 6-30: Configuration of an IPCamera with P-IRIS hardware.....	129
Figure 6-31: Spectral range of visible light and IR light.....	130
Figure 6-32: Normal mode and IR mode.....	131
Figure 6-33: Qualcomm Spectra 2xx Camera Driver.....	134
Figure 6-34: OpenMAX-based video decode – data flow.....	136
Figure 6-35: OpenMAX-based video encode – data flow.....	136
Figure 6-36: ROI video encoding.....	140
Figure 6-37: QP map programming for a frame.....	141
Figure 6-38: Output extra data enable sequence.....	142
Figure 6-39: Visual quality difference with PQ.....	143
Figure 6-40: Rate control convergence.....	144
Figure 6-41: Adreno 612 block diagram.....	146
Figure 6-42: Adreno 612 driver architecture.....	147
Figure 6-43: Product security overview.....	151
Figure 6-44: Qualcomm TEE v5.0 security framework architecture.....	153
Figure 6-45: Components overview.....	158
Figure 6-46: Emergency download/Fastboot flow chart.....	161
Figure 6-47: FDE component workflow.....	167
Figure 6-48: Thermal mitigation software architecture.....	172
Figure 6-49: Placement of on-die temperature sensors.....	173
Figure 6-50: BTM algorithm flow.....	175
Figure 6-51: Thermal core framework architecture.....	176
Figure 6-52: Software channels.....	178
Figure 6-53: Dynamic algorithm example.....	180
Figure 6-54: Threshold algorithm example.....	181
Figure 6-55: Memory map.....	188
Figure 6-56: TensorFlow Lite architecture.....	189
Figure 6-57: Generic virtualization layers.....	190

Figure 6-58: Meta-virtualization and QTI BSP layer meta-qtibsp.....	191
Figure 7-1: Camera functionality in LEPDK.....	194
Figure 7-2: Display and graphics functionality in LEPDK.....	195
Figure 7-3: Video encode functionality in LEPDK – Option 1.....	196
Figure 7-4: Video encode functionality in LEPDK – Option 2.....	197
Figure 7-5: Video decode functionality in LEPDK.....	198
Figure 7-6: Audio capture functionality in LEPDK.....	199
Figure 7-7: Audio playback functionality in LEPDK.....	200
Figure 7-8: High-level architecture of ML use case.....	200
Figure 7-9: Use case – ROI video encoding-based on ML inference results.....	226
Figure 7-10: roi-quant-rectangles property syntax.....	227
Figure 7-11: ROI and non-ROI video side by side pipeline structure.....	230
Figure 7-12: UVC-UAC high-level architecture.....	247
Figure A-1: MPSS software error fatal command on QXDM Professional.....	255
Figure A-2: MPSS software error fatal command on QPST.....	255
Figure A-3: QCAP.....	257
Figure A-4: Error routing overview.....	261
Figure A-5: Path for different type of errors.....	262
Figure A-6: Subsystem error on QCS610 APSS (SSR and PDR are disabled).....	263
Figure A-7: Subsystem error on QCS610 APSS (SSR is enabled).....	264
Figure A-8: Subsystem error on QCS610 APSS (PDR is enabled).....	265

# 1 Introduction to QCS610/QCS410 chipset

---

The Qualcomm® Vision Intelligence platform (QCS610/QCS410) is purpose-built for the unique requirements of the Internet of things (IoT). Also, they are optimized in application-specific configurations for today's smart camera and video solutions.

The platform features the first family of system-on-chips (SoC) built by Qualcomm Technologies, Inc. (QTI), specifically for IoT in an advanced 11 nm process. The Vision Intelligence platform (QCS610) is engineered to support exceptional power and thermal efficiency.

The Qualcomm Vision Intelligence platform (QCS610/QCS410) also features the most advanced image sensor processor (ISP) and digital signal processor (DSP) to date. In addition, it offers the cutting-edge CPU, GPU, camera processing software, connectivity, and security.

The chipset is engineered to provide the following capabilities:

- Multiple options for AI inference engines
- Heterogeneous computing (CPU/GPU/DSP) for on-device machine learning (ML)
- For QCS610, dual ISPs and 4K ultra HD video, and for QCS410, 1080p ultra HD video
- Real-time streaming protocol (RTSP) streaming and effortless connectivity using 4G LTE, Wi-Fi, and Bluetooth
- Native Ethernet (RGMII) interface

This document describes the device and associated peripherals and provides the integration, environment, functional description, and programming models for each peripheral and subsystem in the device.

For more information, see *QCS610/QCS410 Linux Platform Development Kit Quick Start Guide* (80-PL631-300) and *QCS610/QCS410 Linux Platform Development Kit Software Programming Guide* (80-PL631-200).

## 1.1 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `cp armcc armcpp`.

## 1.2 Technical assistance

For assistance or clarification on information in this document, submit a case to QTI at <https://createpoint.qti.qualcomm.com/>.

If you do not have access to the CDMA Tech Support website, register for access or send email to [support.cdmatech@qti.qualcomm.com](mailto:support.cdmatech@qti.qualcomm.com).

## 2 Overview of Linux platform development kit

---

The Linux software development on platform development kit (PDK) follows the principles of the Yocto Project. The core of PDK is derived from stable and tested Poky release made by the Yocto Project. The current PDK release is based on the Thud (v2.6) release of the Yocto Project. The PDK uses the GCC compiler (v8.2.0), and key third-party open source software such as glibc (v2.28), systemd (v1.239), OpenMP and so on from the Poky distribution created by the Yocto foundation. The 32-bit userspace has armhf, that is, hard-float ABI. PDK also uses the BitBake build system and some other metadata that is maintained by the Yocto Project.

QTI software integrated in PDK follows OpenEdge (OE) development principles. Each software component is integrated into the PDK via a recipe. The recipes in turn are organized into layers. In a PDK source tree, there are multiple layers with naming convention *meta-qt-i-*. These layers are written and maintained by the QTI engineers. The *meta-qt-i-* layers are specific to technology, for example, the *meta-qt-i-camera* layers contain recipes that are required to compile the camera libraries and daemons for the Linux user space. Most of these layers provide package groups that group the software recipes of the layer together.

For more information about the Yocto project, see the following link:

<https://www.yoctoproject.org/docs/2.6/overview-manual/overview-manual.html#what-is-the-yocto-project>.

### 2.1 LEPDK for QCS610/QCS410

The Linux Embedded Platform Development Kit (LEPDK) supports the Linux operating system (OS) on the Qualcomm® Snapdragon™ application processors. The goal of the LEPDK is to provide a simple and flexible development platform to rapidly build IoT products with the QTI chipsets.

The LEPDK provides the build framework, software components, tools, and sample applications to evaluate the platform and build software images for the target product. The LEPDK is based on the Yocto Project 2.6 release and follows several of its development principles. The Yocto Project is used by the LEPDK for the build framework. Through the LEPDK, QTI provides the capabilities for the user to define build configurations for specific customization needs.

The LEPDK has built-in support for the GStreamer multimedia framework. The GStreamer is an open-source multimedia framework that supports a wide variety of media-related components for video, audio, and image for its capture, playback, and streaming.

# 3 QCS610/QCS410 HLOS software architecture – LEPDK framework

The Apps complex provides a general computing platform for running a high-level operating system (HLOS). During normal operation, the HLOS is responsible for running the applications as directed by the user, and directing the operation of the various subsystems. The software architecture of the Apps complex is driven by the native HLOS software.



Figure 3-1 Linux PDK architecture

## 3.1 Source tree structure

To download the source code, see *QCS610/QCS410 Linux Platform Development Kit Quick Start Guide* (80-PL631-300).

After the source code is downloaded, a directory structure is displayed.

```
qcs610-le-1-0_ap_standard_oem
├── about.html
├── adsp_proc
├── aop_proc
├── apps_proc
├── boot_images
├── btfm_proc
├── cdsp_proc
├── common
├── contents.xml
├── modem_proc
├── trustzone_images
├── venus_proc
└── wdsp_proc
    └── wlan_proc
```

**Figure 3-2 Directories in workspace after the source code is downloaded**

Directory	Purpose
poky	It contains all the OE metadata layers, Yocto metadata layers, and QTI metadata layers
prebuilt_HY11	It contains prebuilt libraries that are used in the build
src	It is a directory where all the source projects are synched. All the new components must sync sources only under <workspace>/src/.

For build instructions, see *QCS610/QCS410 Linux Platform Development Kit Software Programmer's Guide* (80-PL361-200). After the build is complete, additional directories are displayed in the workspace.

```
qcs610-te-1-0_ap_standard_oem
├── about.html
├── adsp_proc
├── aop_proc
└── apps_proc/
    ├── build-qtি-distro-perf
    ├── downloads
    ├── poky
    ├── prebuilt_HY11
    ├── setup-environment -> poky/qtি-conf/set_bb_env.sh
    ├── src
    └── sstate-cache
        └── syncbuild.sh

└── boot_images
└── btfm_proc
└── cdsp_proc
└── common
└── contents.xml
└── modem_proc
└── trustzone_images
└── venus_proc
└── wdsp_proc
└── wlan_proc
```

Figure 3-3 Directories in workspace after the build is complete

Directory	Purpose
build-qtি-distro - debug	It contains the build artifacts for selected DISTRO
downloads	It is where the OE build system stores the downloaded packages
setup-environment	It is a symlink to <i>poky/qtি-conf/set_bb_env.sh</i> that aids environment setup
sstate-cache/	It contains the sstate-cache for the workspace

## 3.2 QTI metadata layers

QTI BSP metadata is a collection of meta layers that constitute both the general layers and the BSP layers. These layers are prefixed with *meta-qtি-* to differentiate them from the Yocto meta layers and the remaining upstream layers. The presence of a machine configuration file and a kernel recipe or an append file that leverages an existing kernel recipe makes a layer a BSP layer.

### 3.2.1 BSP

This section provides a description of PDK reference BSP layer *meta-qtি-bsp* in context of machine *qcs610-odk*.

The *meta-qti-bsp* layer provides metadata for the essential BSP that helps to compile minimal BSP for Snapdragon-based devices.

- Layer configuration

The priority of the layer is kept as the lowest among all the QTI BSP layers to allow other layers to override the variables defined here. *layer.conf* configuration file for this layer is as follows:

```
BBPATH .= ":${LAYERDIR}"  
  
BBFILE_COLLECTIONS += "qti-bsp"  
  
BBFILE_PRIORITY_qti-bsp = "10"  
  
BBFILE_PATTERN_qti-bsp := "^${LAYERDIR} /"  
  
LAYERSERIES_COMPAT_qti-bsp = "thud"  
  
LICENSE_PATH += "${LAYERDIR}/files/common-licenses"  
  
# Tools needed on the build host for usage within build tasks by recipes  
# of this layer.  
  
HOSTTOOLS_NONFATAL += "xgettext msgmerge msgfmt gmsgfmt java zip"  
  
BBFILES += " ${LAYERDIR}/recipes-*/*/*.bb \  
${LAYERDIR}/recipes-*/*/*.bbappend"
```

- Machine configuration

All the machines supported by PDK are maintained under the *conf/machine* directory. *qcs610-odk* is the default machine described in this section, which is based out of QCS610 system on chip (SoC) of QTI. *qcs610-odk* runs on linux-msm-4.14 kernel and has an active WCN3980 Wi-Fi module integrated with the SoC.

A sample machine configuration (Kernel configuration variables) for *qcs610-odk* is as follows:

```
KERNEL_IMAGETYPE = "Image.gz"  
  
KERNEL_DTB_NAMES = "qcom/qcs610-iot.dtb"  
  
KERNEL_BASE = "0x80000000"  
  
KERNEL_TAGS_OFFSET = "0x81e00000"  
  
  
KERNEL_CONFIG ?= "vendor/sdmsteppe_defconfig"  
KERNEL_CONFIG_qti-distro-debug ?= "vendor/sdmsteppe_defconfig"  
  
  
CONSOLE_PARAM ?= ""  
CONSOLE_PARAM_qti-distro-debug ?= "console=ttyMSM0,115200,n8"  
  
  
KERNEL_CMD_PARAMS ?= "noinitrd rootwait ${CONSOLE_PARAM}  
no_console_suspend=1 androidboot.hardware=qcom androidboot.console=ttyMSM0  
lpm_levels.sleep_disabled=1"
```

In the above snippet, *KERNEL\_DTB\_NAMES* is set to *qcom/qcs610-iot.dtb*. This device tree is included in the kernel image. This device tree can be found under *src/kernel/msm-4.14/arch/*

`arm64/boot/dts/qcom/qcs610-iot.dts`. This variable may be set to any new device tree defined by the user.

The defconfig to be selected is indicated by a KERNEL\_CONFIG variable. This defconfig file is in the source tree at `<path/to/workspace>/src/kernel/msm-4.14/arch/arm64/configs/vendor/sdmsteppe-perf_defconfig`. Update the KERNEL\_CONFIG variable to define and use custom defconfig. KERNEL\_CONFIG\_qti-distro-debug is an override that exists for KERNEL\_CONFIG variable. KERNEL\_CONFIG\_qti-distro-debug could be set to an alternate defconfig file, which may have more configuration items that are enabled for debugging the kernel.

The kernel command line is defined by the variable KERNEL\_CMD\_PARAMS. Any parameters added here are passed to the kernel by the bootloader.

Other variables in `qcs610-odk.conf` file:

```
PREFERRED_PROVIDER_virtual/bootloader = "edk2"  
  
# Conf with partition entries required for machine.  
  
MACHINE_PARTITION_CONF = "${COREBASE}/meta-qtibsp/conf/machine/partition/${MACHINE}-partition.conf"
```

The PREFERRED\_PROVIDER\_virtual/bootloader points to the recipe that should be used to build the bootloader, which is `<path/to/workspace>/poky/meta-qtibsp/recipes-kernel/edk2/edk2_git.bb`.

The MACHINE\_PARTITION\_CONF variable points to the file that defines the partition layout for the machine.

- Distro configuration

The default distributions supported by the PDK are maintained under the `conf/distro` directory. To address different development cycle needs, these distros are classified into the following types:

- debug
- perf
- user

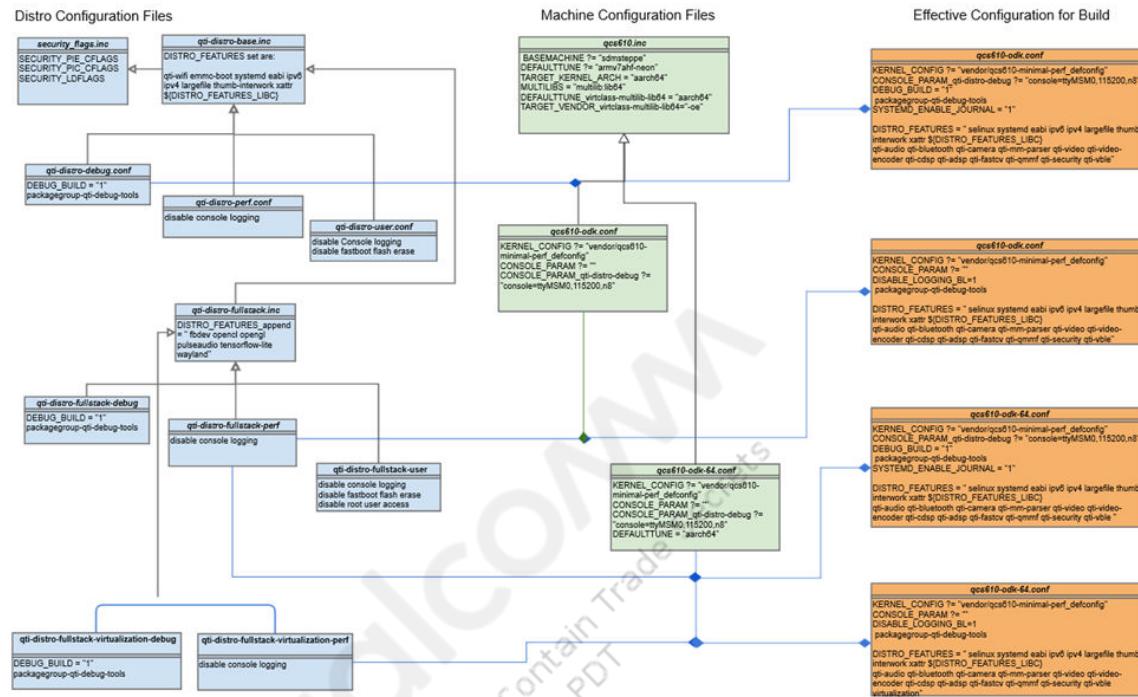
These distros are an amalgamation of upstream and proprietary board support packages, user space modules, and applications that a developer can extend.

The distros defined in the PDK are as follows:

**Table 3-1 Console distros**

qtidistro-debug	qtidistro-perf	qtidistro-user
<ul style="list-style-type: none"><li>▪ Default debug oriented builds</li><li>▪ Contains debug enabled kernel defconfig, extended logging support, and serial console support</li></ul>	<ul style="list-style-type: none"><li>▪ Performance oriented builds</li><li>▪ Limited logging support</li><li>▪ No serial console</li><li>▪ Kernel perf defconfig usage</li></ul>	<ul style="list-style-type: none"><li>▪ Product deployment oriented builds</li><li>▪ Contains all the logging related features as perf distro build; does not support root user privileges</li><li>▪ Debug utilities like diag and QXDM Professional™ tool support are absent</li></ul>

- Machine and distro configuration files – Relationship diagram



- Package groups

- packagegroup-android-utils: This package group brings recipes for supporting utilities such as adbd, binder, leproperties, and usb-composition.

- QTI image recipes

The meta-qtibsp layer defines image recipes, which in turn define the packages that should be a part of the rootfs image.

Image recipes generate a set of appsbootloader images, boot images, system images, overlays images, and firmware images. The device boots into userspace when `qcs610-odk` or any other derived machine is flashed with these images that are built from image recipes.

Image recipes inherit `qimage` class, a custom class inheriting `coreimage` and has commands to generate boot image and system image with auxiliary features when activated.

- qti-console-image: Console image includes package groups and recipes to create a rootfs image that is sufficient for booting the device to shell and connect over Wi-Fi.
- qti-multimedia-image: Multimedia image includes package groups and recipes to create a rootfs image that provides multimedia utilities and libraries such as PulseAudio, GStreamer, and Wayland in the generated rootfs image.

- QTI class – `qimage`

The `qimage` class is a QTI implementation to generate images suitable to be used with QTI boards and any derivative boards. `qimage` class serves the following key functions:

- Generates `boot.img` – `boot.img` is the image that contains devicetree, vmlinux, and kernel command line packed together
- Generates `rootfs` image – `rootfs` image contains user space libraries and binaries

- Generates overlayfs image – overlayfs image is the writeable image that is mounted on /etc and /data to make these directories of rootfs writeable
  - Generates partition binary – The partition binary when flashed using QFIL or QDL, partitions the flash as defined in MACHINE\_PARTITION\_CONF defined in machine configuration file.
- Any image recipe inherits this class to simplify and easily generate the started images.
- Main recipes
    - **firmware-sdmsteppe.bb**  
This recipe installs the firmware image files for Wi-Fi, Bluetooth, and video in the rootsfs. This recipe also deploys images for AOP, TrustZone, firehose programmer, SBL, and RPM in the deploy path.
    - **linux-msm\_4.14**  
This recipe configures and compiles the kernel sources at <workspace>/apps\_proc/src/kernel/msm-4.14. This recipe applies a few configuration fragments to the kernel defconfig defined in the machine.conf.  
For example, if DISTRO\_FEATURE contains “virtualization”, then *virtualization.cfg* is used during the configuration step. If COMBINED\_FEATURES contains fbdev, then *graphics\_fb.cfg* is applied to defconfig. The *.config* generated at the end of *do\_configure()* step is a combined output of defconfig defined by KERNEL\_CONFIG and configuration fragments applied by Linux-msm recipe.
    - **Edk2\_git**  
EDK2 recipe configures, compiles, and deploys the bootloader source available in tree under <workspace>/apps\_proc/src/bootable/bootloader/edk2.
    - **gen-partitions**  
LEPDK defines the disk partition layout using a partition configuration file. The partition configuration file lists all the partitions and respective partition sizes.  
For the qcs610-odk machine supported in this PDK, the partition configuration file is poky/meta-qtibsp/recipes-devtools/gen-partitions/gen-partitions-1.0/qcs610-odk-partition.conf.

The partition configuration file defines a few disk parameters and multiple partition entries as shown in the following table:

**Table 3-2 Partition configuration file components**

<b>Disk parameter</b>	--disk --type=emmc --size=4294967296 --write-protect-boundary=65536 --grow-last-partition --align-partitions=4096  --type should be set to one of emmc, nand or ufs. In this release only emmc is supported.  --size is disk size in bytes  --write-protect-boundary defines where the first partition boundary must begin. this is an optional parameter  --grow-last-partition is an optional parameter, when passed, the last partition is maxed to use all of disk size  --align-partitions is optional, when passed each partition on the disk starts at the specified boundary
<b>Partition entries</b>	--partition --name=sb11 --size=524288 --type-guid=DEA0BA2C-CBDD-4805-B4F9-F428251C3E98 --filename=sb11.mbn  --name partition name  --size partition size in bytes  --type-guid the partition type GUID for the partition  --filename file that should be flashed to this partition by the flashing tools QFIL/QDL

The partition configuration file is processed by the build when a user uses the command **bitbake qti-console-image**.

If the partition configuration file is successfully processed, the following files are generated under `build-qti-distro-debug/tmp-glibc/deploy/images/qcs610-odk-qti-distro-debug`:

- **Non-HLOS (NHLOS)**

Recipes like `firmware-sdmsteppe` provide instructions to BitBake for installing the firmware splitbins onto rootfs and NHLOS binaries such as `tz.mbn`, `rpm.mbm` into `$_DEPLOY_DIR_IMAGE`.

Before the build begins, manually copy `NHLOS_Binaries.zip` from the ChipCode to the file subdirectory under `recipes-firmware/firmware/qcs610-odk/`.

### 3.2.2 Data

Data layer is further split into the following open source and proprietary layers:

- `meta-qti-data` – Recipes for open source data packages
- `meta-qti-data-prop` – Recipes for QTI proprietary data packages

## Data layer interaction with DISTRO\_FEATURES and MACHINE\_FEATURES

Metadata layer	DISTRO_FEATURES	MACHINE_FEATURES
meta-qtidata	<ul style="list-style-type: none"> <li>▪ qti-wifi</li> <li>▪ ethernet</li> </ul>	<ul style="list-style-type: none"> <li>▪ qti-wifi</li> <li>▪ ethernet</li> <li>▪ qti-modem</li> </ul>
meta-qtidata-prop	<ul style="list-style-type: none"> <li>▪ qti-wifi</li> <li>▪ ethernet</li> </ul>	<ul style="list-style-type: none"> <li>▪ qti-wifi</li> <li>▪ ethernet</li> <li>▪ qti-modem</li> </ul>

### Package groups

meta-qtidata layer delivers packagegroup-qtidata, which consists of network tools like dnsmasq, iptables, and so on.

### Recipes

The recipe is meta-qtidata-prop\recipes\data\data\_git.bb

The following list describes working with the configuration features:

- **MACHINE\_FEATURES**

Specifies the list of machine capable hardware features. Along with features supported by Yocto metadata, there are additional machine features defined in the PDK machine configuration files to meet specific machine needs.

An example is `qti-wifi`, which indicates that the machine has QTI Wi-Fi solution. Disabling this feature turns off the Wi-Fi support.

For more information on how to update the `MACHINE_FEATURES`, see *Modify existing machine configurations section in QCS610/QCS410 Linux Platform Development Kit Software Programmer's Guide* (80-PL361-200).

- **DISTRO\_FEATURES**

Specifies the list of features that are enabled in a distribution. Along with features supported by Yocto metadata there are additional distro features defined in PDK to meet the distribution needs.

For the complete list of PDK-specific `DISTRO_FEATURES`, see *Distro configurations section in QCS610/QCS410 Linux Platform Development Kit Software Programmer's Guide* (80-PL361-200).

- **COMBINED\_FEATURES**

Some features should be controlled both at the machine and distribution configuration level. Such features are listed under `COMBINED_FEATURES`. This list is an intersection of `MACHINE_FEATURES` and `DISTRO_FEATURES`. Dropping a feature from either the machine feature or distro feature removes it from the combined feature list.

For example, the `qti-wifi` feature requires not only the hardware support but also the software enablement at the distribution level. If the hardware supports a feature, but you do not intend to use it, then check the combined features to manage appropriately.

For example, when `qti-wifi` string is appended to `MACHINE_FEATURES` and `DISTRO_FEATURES`, it leads to tools such as `iw`, `wlan-conf`, `wireless-tools`, `wpa_supplicant`, and

- WLAN kernel module driver. This also includes the Wi-Fi firmware in the rootfs image under path/*firmware*/.
- IMAGE\_FEATURES
  - These features control images generated by the OE build system.

## **3.3 Build optimizations**

A Yocto project build can take considerable build resources both in time and disk usage. There are methods to optimize this, for example, using a shared state cache (caches the state of the build) and preconfigured downloads directory (holds the downloaded packages).

### **Shared state cache**

These can be set in the *local.conf* file by adding a statement like `SSTATE_DIR="/local/mnt/workspace/PDK/sstate-cache"`. The sstate helps when multiple build directories are set, each of which uses a shared cache to minimize the build time.

### **Shared download directory**

A shared download directory minimizes the fetch time. Set `DL_DIR` in the *local.conf* file by adding a statement like: `DL_DIR="/local/mnt/workspace/PDK/download"`. Every package downloaded in the `DL_DIR` directory is marked with `.done`. To avoid fetching, check the `.done` file for the package name. To set up the download directory, see the [Yocto Project Reference Manual](#).

### **rm\_work class**

The OE build system uses a substantial amount of disk space during the build process. `rm_work` class deletes the temporary workdir and frees up the disk usage during builds. For every recipe, work file can be found under the  `${TMPDIR}/work` directory. After the build system generates the packages for a recipe, the work files for that recipe are no longer needed. However, by default, the build system preserves these files for inspection and possible debugging purposes. Deleting these files saves disk space as the build progresses.

## **3.4 QTI-defined variables in PDK**

- **BASEMACHINE:** It is the base SoC out of which machines like *qcs610-odk* are derived. The value of this variable is also included in `MACHINEOVERRIDES` to allow various overrides supported by OE build system.
- **SOC\_FAMILY:** It is a common QTI SoC group out of which SoC of the current machine is derived.

## 4 QCS610/QCS410 feature overview

---

The table provides the software capabilities of the QCS610 chipset:

Feature	QCS610/QCS410
<b>Processor</b>	
Applications processor	<p>64-bit Arm v-8 compliant applications processor, Qualcomm® Kryo™ 460 CPU, consisting of:</p> <ul style="list-style-type: none"> <li>■ QCS610 <ul style="list-style-type: none"> <li>□ Kryo Gold: Dual high-performance cores 2.2 GHz</li> <li>□ Kryo Silver: Six low-power cores 1.8 GHz</li> </ul> </li> <li>■ QCS410 <ul style="list-style-type: none"> <li>□ Kryo Gold: Dual high-performance cores up to 2.2 GHz</li> <li>□ Kryo Silver: Dual low-power cores up to 1.8 GHz</li> </ul> </li> </ul>
Neural processing engine	Qualcomm® Hexagon™ DSP with dual Hexagon Vector eXtensions (HVX), 1.1 GHz. For ML, integrated DNN for advanced voice activation and Snapdragon neural processing engine framework
Always-on system	<ul style="list-style-type: none"> <li>■ Always-on subsystem with always-on processor</li> <li>■ Hardware-based resource and power management (RPMh) with hardware accelerators for voltage control and regulation, clock management, and resource communication</li> </ul>
Low-power island	Low-power island with low-power DSP consists of Snapdragon sensor core and low-power audio subsystem
<b>Memory</b>	
System memory via EBI	Dual-channel non-PoP high-speed memory (discrete and eMCP) LPDDR4 is not supported.
External memory via	<p>UFS 2.1 gear 3 (one-lane), eMMC 5.1, and SD 3.0</p> <p>Note: UFS requires external LDO if eMMC is used</p>
<b>Wireless support</b>	
WLAN/Bluetooth	Yes (with WCN3980)
GNSS –Integrated Qualcomm Location Suite engine	<p>GPS, GLONASS, BeiDou, Galileo</p> <p><b>NOTE</b> Locations features like GPS and GNSS were validated separated with additional antenna hardware and calibration. They are not part of the default package.</p>
<b>Multimedia</b>	
Display support MIPI-DSI	One 4-lane; DSI D-PHY 1.2 with split link; supports resolutions up to FHD+
DisplayPort	DisplayPort 1.4

Feature	QCS610/QCS410
General display features	Color depth – 30-bit pp; TFT,LTPS, CSTN
Camera interface	Three 4-lane CSIs (4/4/4 or 4/4/2/1) D-DPHY is 2.1 Gbps, CPHY – 5.7 Gbps per trio MIPI CSI 3 × 4 lanes; C-PHY, D-PHY
Video applications performance encode	<ul style="list-style-type: none"> <li>▪ QCS610 <ul style="list-style-type: none"> <li>▫ Multi-format codec up to 4K30 video encode.</li> <li>▫ Multi-stream codec (4K30(HEVC/H.264) + 720p30 (YUV) + 480p30 (VA - YUV))</li> </ul> </li> <li>▪ QCS410 <ul style="list-style-type: none"> <li>▫ Multi-format codec up to 1080p90 video encode.</li> <li>▫ Multi-stream codec (1080p30 HEVC/H.264)+ 1080p30 (YUV) + 480p30 (VA - YUV))</li> </ul> </li> </ul>
Video applications performance decode	<ul style="list-style-type: none"> <li>▪ QCS610 – 4K30 HEVC, H264</li> <li>▪ QCS410 – 1080p90 HEVC, H264</li> </ul>
Graphics	<ul style="list-style-type: none"> <li>▪ Qualcomm® Adreno™ GPU 612, 3D graphics accelerator with 64-bit addressing</li> <li>▪ OpenCL 2.0 FP, C2D</li> </ul>
<b>Multimedia – Audio</b>	
Audio speaker	WCD9370/WCD9341 codec + WSA8810/WSA8815 speaker amplifier
Audio codec support	MP3;AAC; HE AAC v1, v2; WMA 9/Pro; FLAC, APE, ALAC, and AIFF
Audio interfaces SLIMbus	<ul style="list-style-type: none"> <li>▪ MI2S</li> <li>▪ SWR</li> <li>▪ Two port SLIMbus interface four ports</li> <li>▪ SoundWire interface</li> </ul>
Enhanced audio	<ul style="list-style-type: none"> <li>▪ Qualcomm Noise and Echo Cancellation v7 and Qualcomm Voice Suite v3</li> <li>▪ WSA8810/WSA8815 speaker protection</li> </ul>
<b>Peripherals</b>	
<ul style="list-style-type: none"> <li>▪ Qualcomm universal peripheral (QUP) ports</li> <li>▪ UART</li> <li>▪ I<sup>2</sup>C</li> <li>▪ SPI</li> <li>▪ Camera I<sup>2</sup>C</li> </ul>	<ul style="list-style-type: none"> <li>▪ 12 QUP ports; multiplexed serial interface functions</li> <li>▪ 6 QUP ports, multiplexed serial interface functions for LPI island</li> <li>▪ 4 UART interface – 2 applications side (Bluetooth and Console debug) + 2 LPI side (WCN + LPI debug)</li> <li>▪ 4 I<sup>2</sup>C interface – 2 Apps side (Legacy touch, Sensor hub) + 2 LPI side (Pressure, ALS Proximity, RGB sensor &amp; BLE)</li> <li>▪ 4 SPI interface – 3 Apps side (Fingerprint sensor, codec, and NFC) +1 LPI side (SPI sensor)</li> <li>▪ Two dedicated I<sup>2</sup>C interfaces for camera</li> </ul>
USB	USB 3.0; 1 USB 2.0/3.0 port
Secure digital interfaces	<ul style="list-style-type: none"> <li>▪ 8-bit port SDC1 for e-MMC5.1 and 4-bit SDC2 for SD card 3.0</li> <li>▪ SDC2 is dual-voltage</li> <li>▪ SD/MMC card; UFS</li> </ul>
Wireless connectivity	WCN3980 1 × 1 802.11a/b/g/n/ac RF
<b>Configurable GPIOs</b>	
Number of GPIO ports	120
Input configurations	Pull-up, pull-down, keeper, or no pull

Feature	QCS610/QCS410
Output configurations	Programmable drive current
Top-level mode multiplexer	Provides a convenient way to program groups of GPIOs
RGMII	One RGMII interface with MDIO for Ethernet with AVB (1.8 V only for RGMII and MDIO)
<b>Internal functions</b>	
General hardware security features	Secure boot, Haven Security Platform, secure debug, secure key provisioning, TrustZone, Qualcomm trusted execution
Crypto engines TrustZone services	Environment, hardware supported KeyStore Crypto engine v5 (CE5), DRBG/PRNG (FIPS-compliant), inline crypto engine (FIPS-compliant)
PLLs and clocks	<ul style="list-style-type: none"> <li>▪ Multiple clock regimes; watchdog and sleep timers</li> <li>▪ Input: 19.2 MHz from PMIC CXO</li> <li>▪ General-purpose outputs: M/N counter and PDM</li> </ul>
Debug	JTAG, QDSS, and ETM
<b>Chipset and RF interface features</b>	
SDR RF transceivers QLink digital interface	<ul style="list-style-type: none"> <li>▪ Three downlink lanes and one uplink lane</li> <li>▪ Improved layout, routing, package, and signal integrity</li> </ul>
Power management	<p>PM6150</p> <ul style="list-style-type: none"> <li>▪ Integrated switching charger with Qualcomm® Quick Charge™ technology v4.0 support</li> <li>▪ Five switching regulators and 19 LDOs</li> <li>▪ Eight switching regulators and 11 LDOs</li> </ul>
Wireless connectivity WLAN baseband data Bluetooth	I/Q differential pair interface UART interface
<b>Package</b>	
Non-PoP – small, thermally efficient package	806 PSP: 11.1 × 12 × 0.92 mm; 0.35 mm pitch

# 5 QCS610/QCS410 non-HLOS software architecture

---

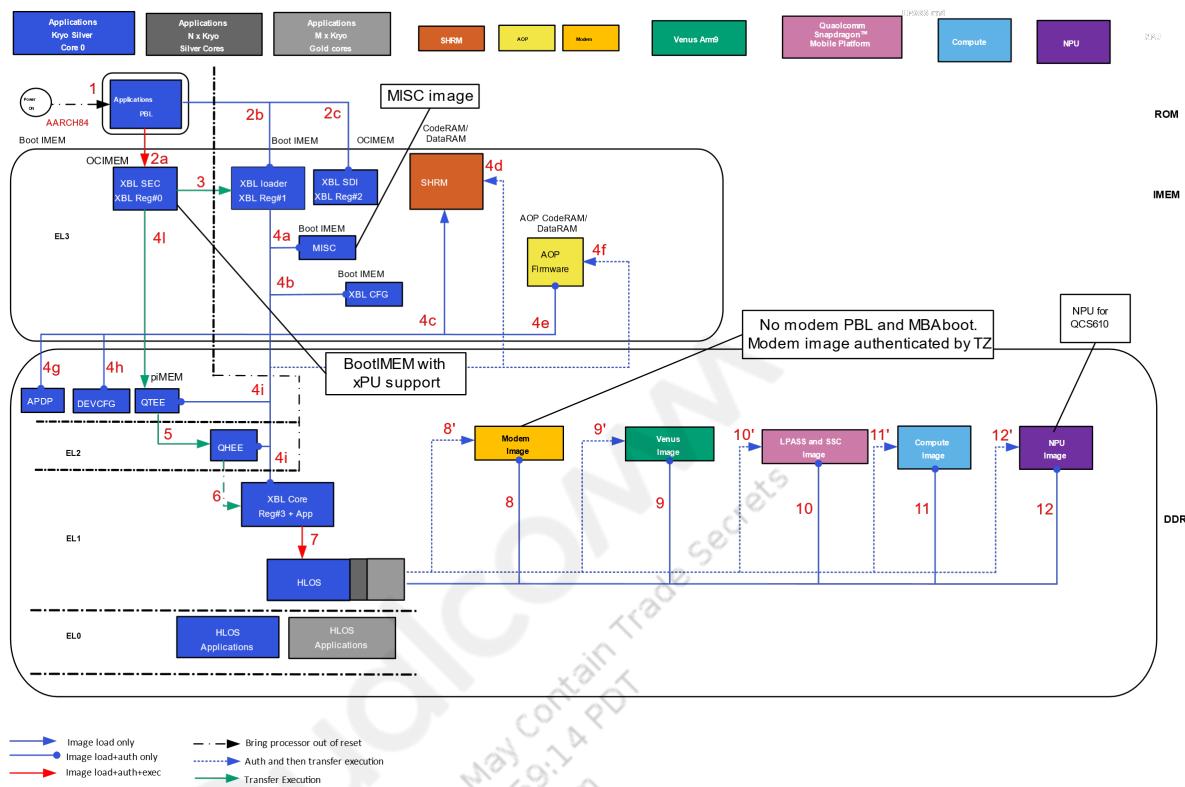
The chipset platform is built on intelligent subsystems that require a software image to operate. Some of these subsystems operate in loosely-coupled peer-to-peer paradigm and some operate in tightly-coupled CPU/co-processor paradigm.

1. In these subsystems, the applications processor boots the peer processors and directs overall system behavior.
2. The system boot starts with the RPM, which brings up the applications processor.
3. The applications processor allocates the memory and loads the software images for the other intelligent subsystems. It is also capable independently restarting each intelligent subsystem except RPM.

## 5.1 Boot

The boot sequence on the applications processor involves:

- Primary boot loader (PBL)
- eXtensible boot loader (XBL)
- Unified extensible firmware interface (UEFI)



**Figure 5-1 Boot flow diagram**

The following sequence describes the boot flow:

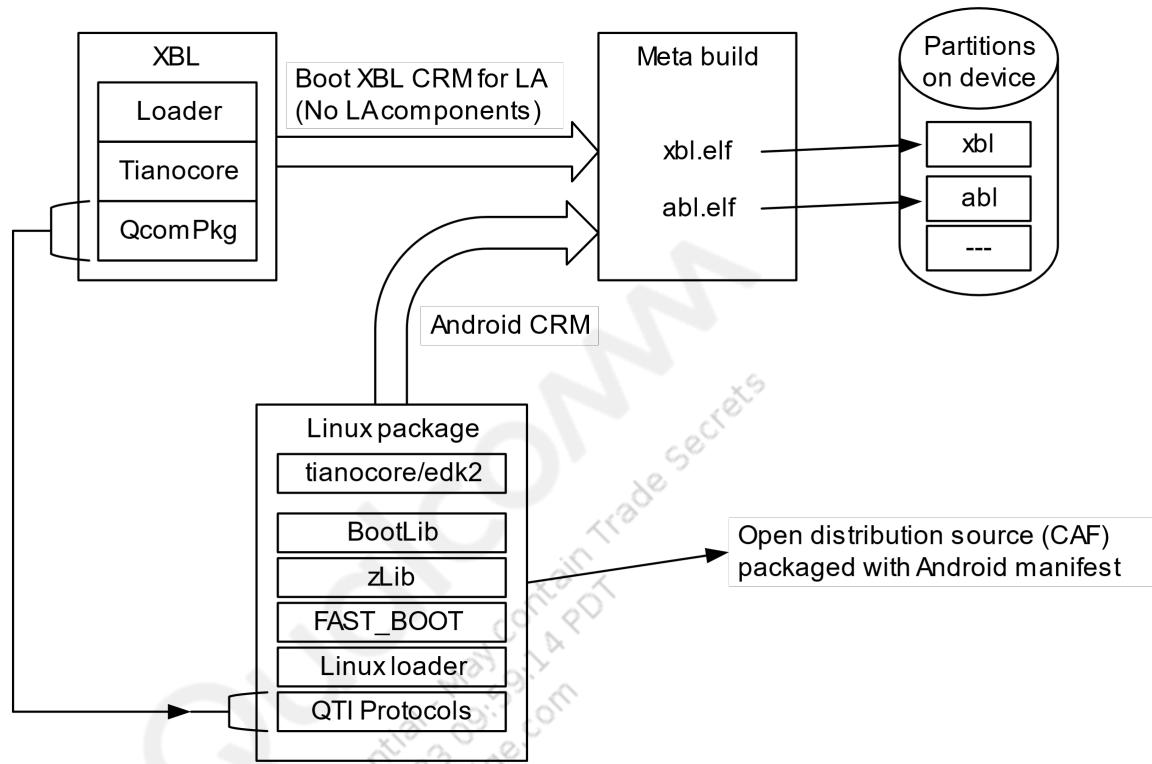
1. After reset, the Kryo Silver core 0 comes out of reset and then executes PBL.
- On Kryo Silver core 0, applications PBL initializes hardware (clocks, and so on), CPU caches and MMU, and detects the boot device as per the boot option configuration:
- Default boot option: eMMC > SD > USB
  - Default boot option: overridden by EDL cookie or force USB GPIO
2. Application processor PBL loads and authenticates the following:
    - a. XBL\_SEC (region #0) from the boot device to boot IMEM
    - b. XBL\_Loader (region #1) from the boot device to boot IMEM
    - c. XBL\_Debug (region #2) from the boot device to OCIMEM
    - d. Jumps to XBL\_SEC
  3. XBL\_SEC runs the security configuration in EL3 mode, and then executes the XBL\_loader in EL1 mode.
    - XBL-loader initializes hardware and firmware images, CPU caches, MMU, boot device, XBLConfig, PMIC driver, and DDR.
    - XBL-loader performs DDR training if applicable, executes an SCM call to XBL\_SEC to initialize PIMEM, and initializes clocks and configures the clock frequencies as per clock plan.

4. The application processor XBL does the following:
  - a. Loads and authenticates the MISC image.
  - b. Loads and authenticates the XBL CONFIG image.
  - c. Loads and authenticates the SHRM image from the boot device.
  - d. Brings SHRM out of reset.
  - e. Loads and authenticates AOP image from the boot device.
  - f. Brings the AOP processor out of reset.
  - g. Loads and authenticates the application processor debug policy (APDP) image from the boot device.  
If the DLOAD cookie is set, loads and authenticates XBL RAM dump, and then jumps to XBL RAM dump to collect the crash dump.
  - h. Loads and authenticates the TZ device configuration (DEVCFG) image from the boot device.
  - i. Loads and authenticates the Qualcomm® Trusted Execution Environment (Qualcomm TEE) image from the boot device; loads the SEC.dat (fuse blowing data) image from the boot storage if the image exists.
  - j. Loads and authenticates the QHEE image from the boot device.
  - k. Loads and authenticates the XBL\_CORE image from the boot device.
  - l. Executes an SCM call to XBL\_SEC to jump to the Qualcomm TEE cold boot.
5. Qualcomm TEE sets up the secure environment and executes the QHEE image.
6. QHEE executes the XBL\_CORE (or XBL region #3), and then the XBL\_CORE mounts and runs the UEFI application processor (ABL firmware volume (FV)).
7. Linux loader application (part of ABL FV) loads and authenticates the HLOS kernel with Verified Boot.
8. Peripheral image loader (PIL) driver in HLOS loads Venus to DDR and configures the clocks and power rails if necessary. The PIL driver in HLOS executes an SCM call to request a secure PIL driver, authenticates Venus, and brings it out of reset.
9. PIL driver in HLOS loads the LPASS and SSC firmware, and then executes an SCM call to request secure PIL driver, authenticates the firmware, and brings it out of reset.
10. PIL driver in HLOS loads the compute image, executes an SCM call to request secure PIL driver, authenticates the firmware, and brings it out of reset.

## **UEFI**

UEFI defines a software interface between an OS and platform firmware. The interface consists of data tables that contain platform-related information, and boot and run-time service calls that are

available to the OS and its loader. Together, these provide a standard environment for booting an OS and running preboot applications.



**Figure 5-2 UEFI**

QTI uses TianocoreEDK2 implementation for the UEFI specification. It is an open-source implementation available at <http://www.tianocore.org/edk2/>.

The UEFI is implemented in two parts:

- The XBL core contains chipset-specific core protocols (drivers) and core applications (such as charging). XBL core is part of the non-HLOS boot\_images code.
- The ABL contains chipset-independent applications such as fastboot.

### 5.1.1 Boot configuration

#### XBL configuration

The PMIC settings and DDR configuration are packed into the XBL configuration (XBLConfig). The XBLConfig separates the configuration from the XBL code sequence.

A single configuration file provides:

- Memory optimizations – The PMIC, DDR, and other drivers can break up the settings into multiple configuration binary items and load or use only the ones that are required.
- Flexibility and better management of configuration items per the hardware block version, chipset, and so on.
- Standalone tools – To convert the changed configuration binary items into executable and linking format (ELF) without going through compilation or build invocation.

For more information, see *XBL Configuration Guide* (80-PE663-1).

**Table 5-1 Boot options and configuration**

BOOT_CONFIG[4..1] (FAST_BOOT[3:0])	Boot sequence	To select boot option using fuses				To select boot option using GPIOs			
		OEM_CONFIG FUSE: FAST_BOOT_SELECT[3:0]				BOOT_CONFIG[4..1]			
		Bit 3	Bit 2	Bit 1	Bit 0	WLAN_SW_CTRL	GPIO [120]	GPIO [27]	GPIO [88]
0b000000	eMMC at SDC1 à SD at SDC2 à USB on USB 3.1	0	0	0	0	0	0	0	0
0b000001	SD at SDC2 à eMMC at SDC1	0	0	0	1	0	0	0	1
0b000010	SD at SDC2	0	0	1	0	0	0	1	0
0b000011	USB on USB 3.1	0	0	1	1	0	0	1	1
0b000100	UFS – HS-G1 mode	0	1	0	0	0	1	0	0
0b000101	SD at SDC2 à UFS	0	1	0	1	0	1	0	1

## 5.2 Peripherals

The Qualcomm® Universal Peripheral (QUP) v3 is supported on the QCS610/QCS410 chipset. The QUP v3 supports the I2C, UART, and SPI functionality.

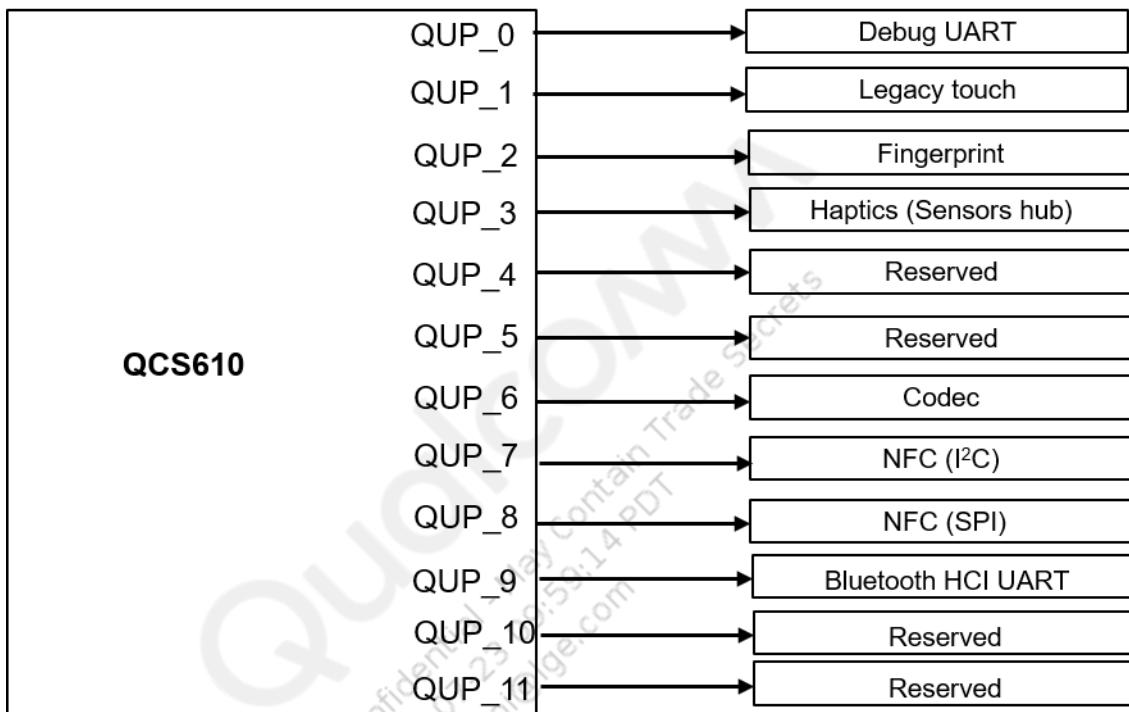
### 5.2.1 QUP v3

QUP v3 is a highly flexible and programmable module for supporting a wide range of serial interfaces. It supports access from multiple hardware entities in the system. Each entity has its own execution environment (EE), a separated address space, and an interrupt line.

A single QUP v3 module provides up to six serial interfaces using its internal serial engines. The firmware loaded to the serial engine determines the protocol that is supported by each interface. To

load the required protocol, in TZ, modify the `settings/buses/qup_accesscontrol/qupv3/config/6150/QUPAC_Access.xml` file.

A QUP v3 module provides up to six serial interfaces using its internal serial engines. The protocol supported by each interface is determined by the firmware loaded to the serial engine (SE). There are two QUP modules with six SEs, and a total of 12 QUPs/SEs available in the GPIOs. To load the required protocol, modify the `QUPAC_Access XML` file in TZ.



**Figure 5-3 QUP configuration example**

**Table 5-2 QUP v3 GPIO assignment**

GPIO numbers	QUP instance
16, 17	QUP_0
4, 5	QUP_1
0, 1, 2, 3	QUP_2
18, 19	QUP_3
20, 21, 22, 23	QUP_6
14, 15	QUP_7
6, 7, 8, 9	QUP_8
10, 11, 12, 13	QUP_9

The following are the QUP v3 operational modes:

- First In, First Out (FIFO) mode is intended for low-bandwidth applications. The interrupt is generated when the QUP output FIFO is empty and it has the programmed number of words in its buffer.
- Direct Memory Access (DMA) mode
- GSI mode

### **5.2.1.1 I<sup>2</sup>C**

The I<sup>2</sup>C core supports the following:

- 7-bit device addressing
- Standard (100 kHz), Fast (400 kHz), and Fast+ (1 MHz) modes

For more information, see */kernel/drivers/i2c/busses/i2c-qcom-gen1.c*.

### **5.2.1.2 UART**

The UART core supports the following:

- DMA
- Data rates up to 4 Mbps

For more information, see */kernel/drivers/tty/serial/msm\_gen1\_serial.c*.

For better performance, QTI recommends using high-speed UART drivers for higher baud rates and bigger data packages, for example, for the Bluetooth connectivity module.

The following are the different UART modes:

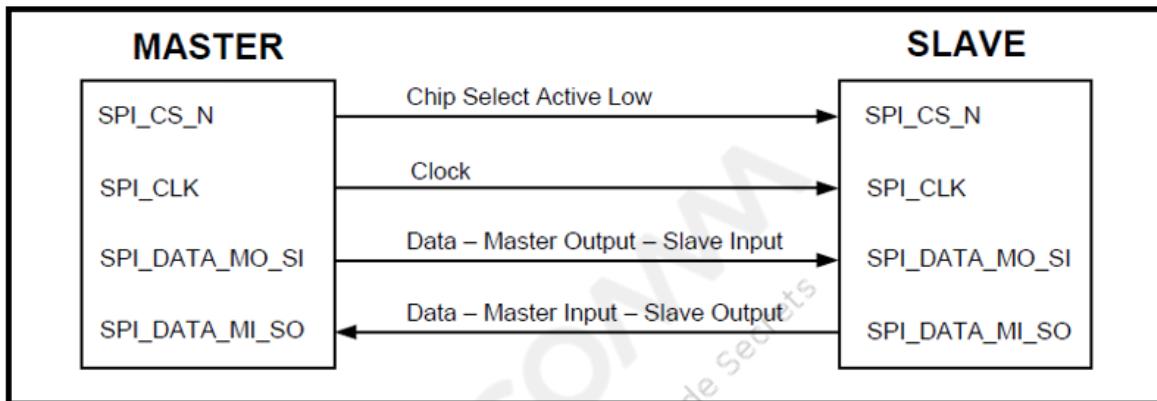
- High-speed mode
  - In High-speed mode, UART uses DMA to transfer data between its Rx and Tx buffers, and system memory.
  - Use high-speed UART if a large amount of data is transferred or for instances where a high-speed transfer is required.
- Low-speed mode
  - The low-speed UART driver supports small data transfers at slower rates, for example, for console debugging or for infrared data association (IrDA) transfers.
  - In Low-speed mode, UART uses FIFO to transfer data between its Rx and Tx buffers, and system memory.

### **5.2.1.3 SPI**

The SPI core allows full and half-duplex, synchronous, serial communication between a master and slave. There is no explicit communication framing, error checking, or defined data word length. Hence, the communication is strictly at the raw bit level.

The SPI core supports the following:

- Transfer rates up to 50 MHz
- 4 bits to 32 bits per word of transfer
- Spacing between byte and word transfers (SPI\_CS\_N WAIT)



**Figure 5-4 SPI - serial communication between master and slave**

For more information, see [/kernel/drivers/spi/spi-geni-qcom.c](#).

## 5.2.2 USB

**Table 5-3 USB overview**

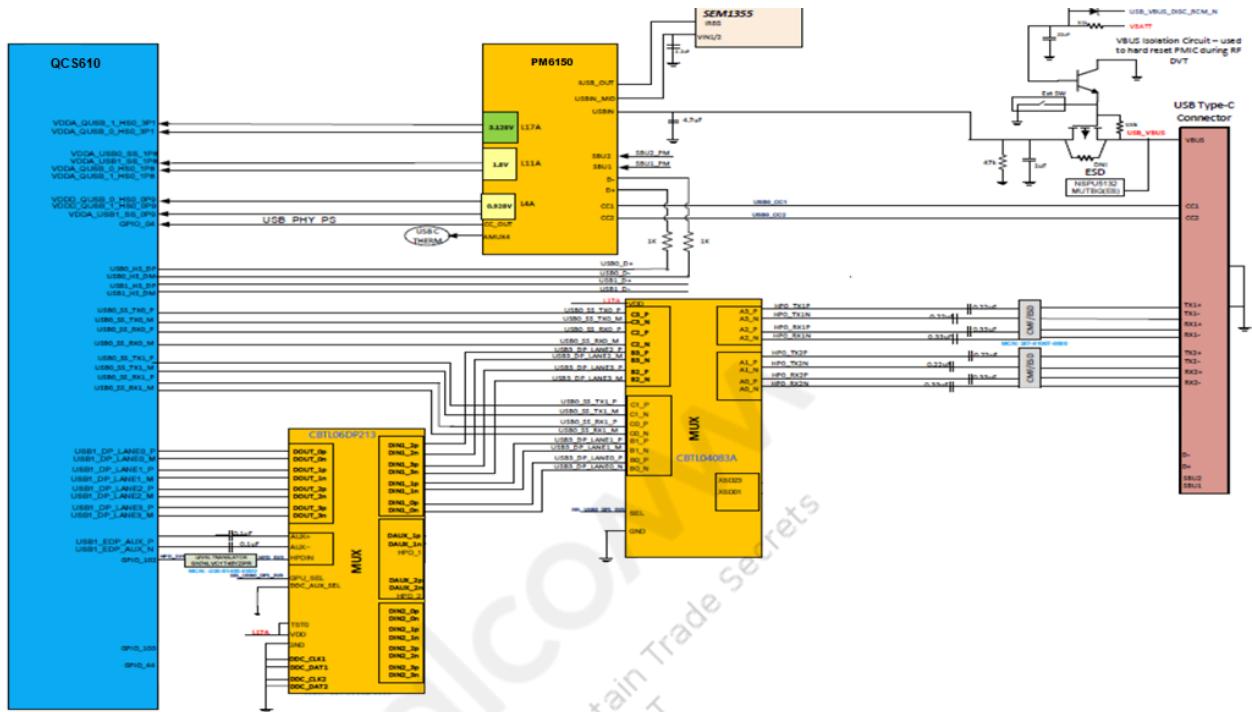
Feature	QCS610/QCS410
USB 3.0 cores	Supports one SS-USB controller
Additional USB cores	Supports one additional HS-USB core
VBUS indication	PMIC driver notifies USB PHY of VBUS indication
HSIC core	The HSIC core is not available in QCS610.
PHY access methods	Allows PHY access through the AHB2PHY path.
Integrated USB Type-C capability	Supported if PM6150 is used
Integrated SS-USB lane switch	Supported. No external switch is required to select the proper SS-USB transmit/receive (Tx/Rx pins) (for USB Type-C).

**Table 5-4 Linux USB features**

Feature	Supported	Comments
Peripheral ADB	Yes	–
Peripheral mass storage	Yes	–
Peripheral diag	Yes	–
Peripheral DUN	Yes	–
Peripheral QMI RmNet	Yes	Supports Cat12 speeds
Peripheral RNDIS	Yes	Supports Cat12 speeds

**Table 5-4 Linux USB features (cont.)**

Feature	Supported	Comments
Peripheral ECM	No	Not officially supported but can be enabled by OEMs (throughput results may differ)
Peripheral MBIM	No	–
Peripheral NCM	No	Only functional testing is done on the NCM interface
Peripheral MTP	Yes	–
Host and Peripheral digital audio(UAC)	Yes	–
Peripheral HSIC	No	–
QDSS	Yes	–
Host USB 3.0 driver (xHCI)	Yes	Both USB controllers support the xHCI architecture
USB attached SCSI protocol	No	UASP is not supported in any platform
Host HS USB	Yes	–
Host HSIC	No	–
Host HID/MS/Hub driver	Yes	–
Host digital audio driver	Yes	Implements low-power USB audio path, as well as support for autosuspend
Host and peripheral video driver (UVC)	Yes	--
Host HSIC link power management	No	–
Peripheral USB link power management	Yes	–
Accessory charger adapter (ACA)	No	Discontinued due to low market adaption
USB host PC charging (SDP)	Yes	BC 1.2 charger detection
USB wall charging (DCP)	Yes	BC 1.2 charger detection
OTG	No	Dual-role device support is possible (Host/Device mode support); no support for SRP/HNP
Quick Charge 2.0/3.0/4.0	Yes	Depends on if PM6150 is used
USB Type-C	Yes	Supports current charging, configuration channel (CC) logic, and SS-USB switch selection
USB Type-C Analog Audio	No	–
USB power delivery (PD) v2.0/3.0 charging	Yes	PD support is based on whether the PM6150 solution is used. This feature highlights only the PD 3.0 charging part of the PD specification; <i>not</i> fully compliant with the <i>USB Power Delivery 3.0 Specification</i> – implements certain features, such as PPS and PD 3.0 sink only mode.
USB Type-C display port	Yes	Supports SS-USB+DP operating concurrently (2 DP lanes)
USB proprietary charger detection	Yes	Supported by PM6150
USB floated charger detection	Yes	Supported by PM6150
USB3.1 Gen1	Yes	Gen2 is not supported at this time



## Figure 5-5 USB configuration

### **5.2.2.1 USB key performance indicators**

The following are the USB key performance indicators (KPIs):

- USB video class (UVC) v1.1
  - Test setup
  - Secondary boot image with Interactive mode
  - Switch DUT to UVC USB composition
  - setprop sys.usb.config diag,uvc,adb
  - Connect to Windows host PC, start video capture application (AMCap)
  - Use uvc-gadget to stream image over `/dev/videoX/data/uvc-gadget -d /dev/video1 -i /data/image-360.jpg -f 2`
  - Measure throughput using LeCroy analyzer

The UVC throughput measured at USB level (using LeCroy analyzer); may vary based on video application:

<b>Test case</b>	<b>Performance numbers</b>
Image Streaming Test	SS – 2.1 Gbps

- Host mass storage performance numbers
  - Test setup – Secondary boot image with Interactive mode

Test case	Performance numbers	
Copy of a single 1 GB file between internal memory and pen drive	HS	Read – 34.8 MB/sec Write – 30.8 MB/sec
	SS	Read – 141 MB/sec Write – 92 MB/sec

**NOTE** Although the QCS610 platform supports for UVC 1.5 and UAC 2.0 at kernel driver level, there is no support for an end-to-end user space application that executes the multimedia functions and share the results (like camera or audio data). OEM has to implement the user space components to realize the end-to-end use case.

### 5.2.2.2 USB Type-C

The following are the USB Type-C features:

- Display port
- Digital/analog audio
- PD
- Plug-in reversibility
- CC charging

PM6150 handles most of the USB Type-C PD detection, which includes the following:

- CC detection
- Charger type detection (APSD)
- PD negotiation

**Table 5-5 USB Type-C and USB PD software drivers**

Main drivers	Location	Description
qpnpc-smb5.c	<apps root>/drivers/power/supply/qcom/qpnpc-smb5.c	<ul style="list-style-type: none"> <li>▪ The PMIC driver handles the CC detection, charger type detection, and OTG VBUS regulator enablement.</li> <li>▪ The PMIC driver handles the power supplies through the power supply framework (that is, battery, USB, DC, and so on).</li> <li>▪ The driver contains the set_property() and get_property() callbacks for each supply.</li> </ul>
policy_engine.c	<apps root>/drivers/usb/pd/policy_engine.c	<ul style="list-style-type: none"> <li>▪ The USB driver handles the power supply changed events on the USB supply.</li> <li>▪ The driver contains the PD state machine to handle the different PD messages and states, as well as a USB state machine to handle what</li> </ul>

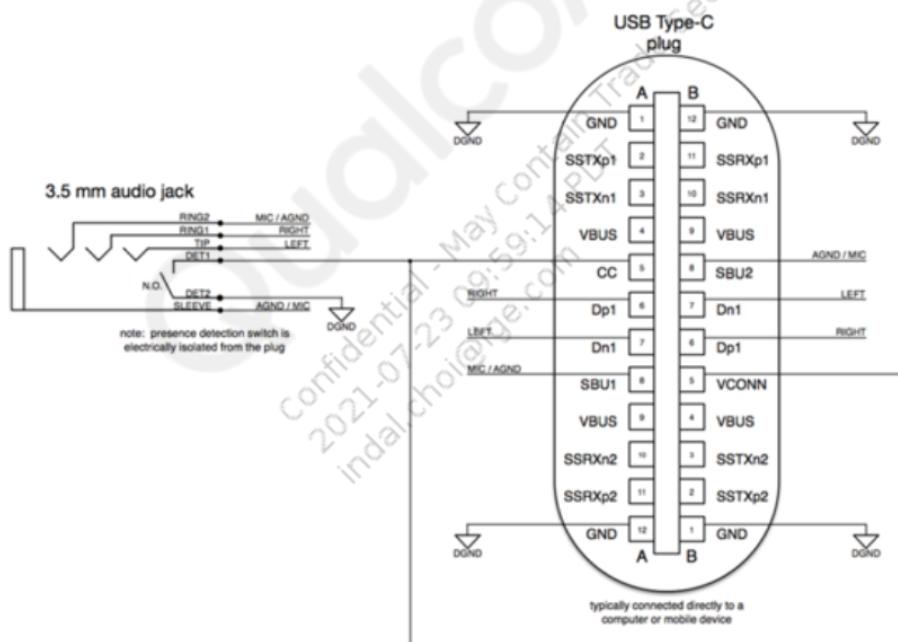
**Table 5-5 USB Type-C and USB PD software drivers (cont.)**

Main drivers	Location	Description
		type of notification to send to the DWC3 driver.
qpnp-pdphy.c	<apps root>/drivers/drivers/usb/pd/ qpnp-pdphy.c	The PMIC driver handles the interaction between the policy_engine and PMIC registers/hardware.

**USB analog audio**

The USB Type-C is capable of detecting audio adapter accessories. For example, 3.5 mm headset jack to USB Type-C adapter.

USB Type-C is useful for handsets that do not include a 3.5 mm headset jack. USB Type-C routes analog audio signals over USB D+/D- pins directly to the audio codec.

**Figure 5-6 USB analog audio****Table 5-6 USB analog audio software drivers**

Main drivers	Location	Description
qpnp-sm5.c	<apps root>/drivers/power/supply/qcom/ qpnp-smb5.c	The PMIC driver handles the CC detection for the Ra/Ra CC line resistance (analog audio adapter).
wcd-mbhc-v2.c	<apps root>/sound/soc/codecs/wcd-mbhc-v2.c	Sound codec driver that handles the switching of the external multiplexer

### 5.2.2.3 USB charging

The following are the USB Type-C CC charge rates:

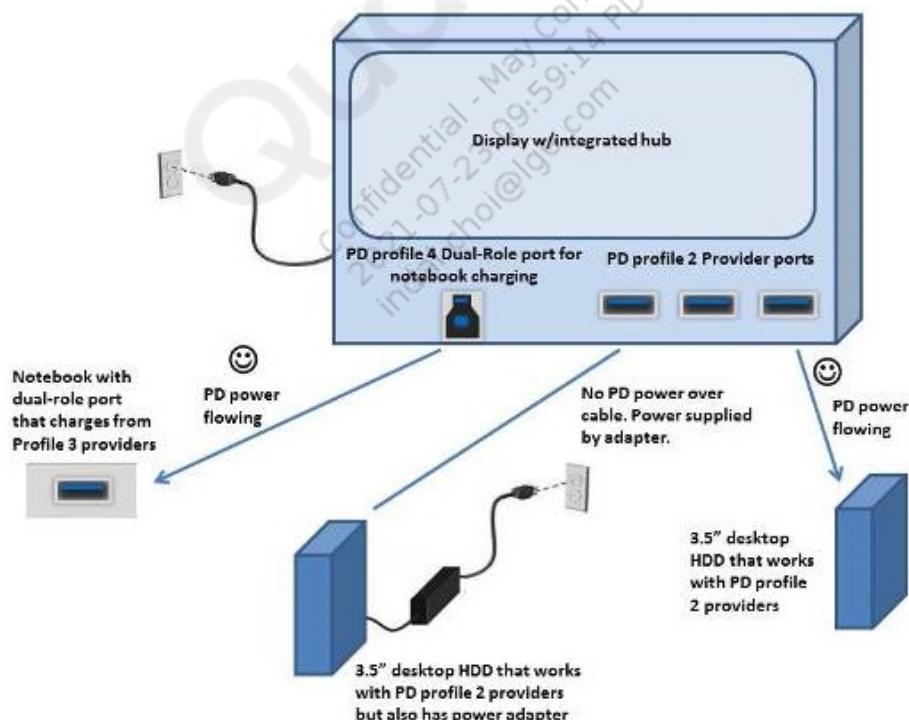
- 1.5 A at 5 V
- 3 A at 5 V

Depending on the capabilities of the PMI, USB PD 3.0 is configurable up to 5 A at 20 V. The USB host mode charging is supported. The following are the features of PD:

- PD adds increased power levels from the existing USB standards up to 100 W
- Power direction is no longer fixed; this enables the product with the power (host or peripheral) to provide power
- PD optimizes power management across multiple peripherals by negotiating different PD profiles
- Power delivery is compatible with pre-existing charging capabilities

The charging capabilities are as follows:

- BC 1.2
- Quick Charge 2.0/3.0/4.0



**Figure 5-7 USB charging**

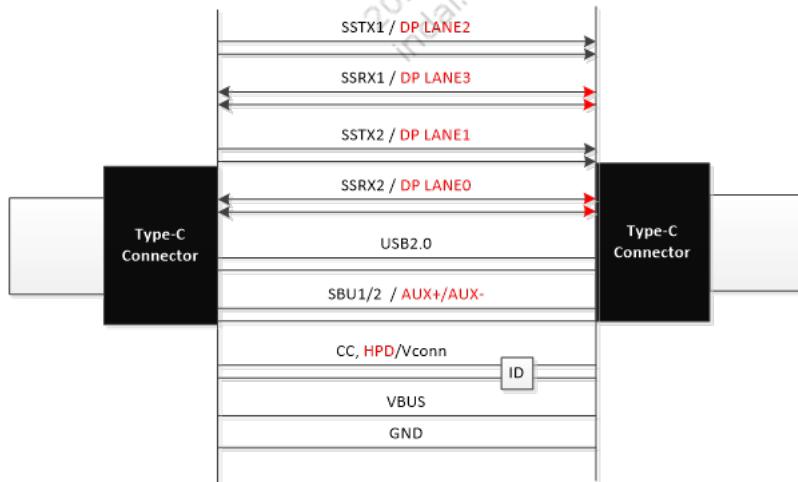
**Table 5-7 USB charging software drivers**

Main drivers	Location	Description
qpnp-smb5.c	<apps root>/drivers/power/supply/qcom/qpnp-smb5.c	<p>Handles charger type detection (BC 1.2, Qualcomm, USB Type-C) and sets input current limit.</p> <p>PMIC driver handles the power supplies through the power supply framework (that is, battery, USB, DC, and so on)</p> <p>Contains the set_property() and get_property() callbacks for each supply</p>
policy_engine.c	<apps root>/drivers/drivers/usb/pd/policy_engine.c	Handles negotiation of USB PD chargers and exposes a framework to request for PD profiles
qpnp-pdphy.c	<apps root>/drivers/drivers/usb/pd/qpnp-pdphy.c	Controls the physical communication channel for PD negotiation

#### 5.2.2.4 USB display

The USB and DisplayPort concurrency is as follows:

- 1920 × 1200 at 60 Hz DisplayPort over USB-C
- QMP SS-PHY can be programmed into several modes
- The following are the possible modes
  - SS-USB only
  - DisplayPort only

**Figure 5-8 USB and DisplayPort concurrency**

**Table 5-8 DisplayPort concurrency orientation**

Type-C receptacle pins	DisplayPort	
	portselect=0 (normal plug orientation)	portselect=1 (flipped plug orientation)
RX2+-	DP ML0	DP ML3
TX1+-	DP ML2	DP ML1
RX1+-	DP ML3	DP ML0
TX2+-	DP ML1	DP ML2
SBU1	DP_AUX_P	DP_AUX_P
SBU2	DP_AUX_N	DP_AUX_N

**Table 5-9 USB and DisplayPort concurrency software drivers**

Main drivers	Location	Description
phy-msm-ssusb-qmp.c	<apps root>/drivers/usb/phy/phy-msm-ssusb-qmp.c	Configures the USB PHY into DP mode, depending on the configuration selected
policy_engine.c	<apps root>/drivers/drivers/usb/pd/policy_engine.c	Discovers DP connections and sends notifications to the display driver
dp_usbpd.c	<apps root>/drivers/gpu/drm/msm/dp/dp_usbpd.c	Handles VDM PD messaging with the DP adapter and initializes the DP channel

**Table 5-10 USB controller software drivers**

Main drivers	Location	Description
dwc3-msm.c	<apps root>/drivers/usb/dwc3/dwc3-msm.c	DWC3 MSM™ driver for handling controller-specific sequences, such as suspending/resuming of the hardware, interrupt management, and so on
gadget.c	<apps root>/drivers/usb/dwc3/gadget.c	DWC3 USB device controller (UDC) driver that handles device mode operation and interacts with the ConfigFS framework
core.c	<apps root>/drivers/usb/dwc3/core.c	DWC3 core driver that initializes host and device mode parameters as defined by the upstream kernel
xhci-plat.c	<apps root>/drivers/usb/host/xhci-plat.c	DWC3 HCD driver that binds to the Linux XHCI stack

### 5.2.2.5 USB composition

The default USB composition is 0 × 90DB – diag + DUN + RmNet + DPL + QDSS + ADB.

Each interface is explained in the following table:

Interface	Description
Diag	Enables you to run the diagnostics data for QXDM Professional Tool or QPST
DUN	–
RmNet	Enables you to perform RmNet tethering
DPL	Enables you to do data protocol logging
QDSS	–
ADB	Android Debug Bridge

- The `sys.usb.config` value is `diag,serial_cdev,rmnet,dpl,qdss,adb`. The `sys.usb.configfs` value is 1, which enables the use of ConfigFS. All the available compositions are contained within `init msm.usb.configfs.rc`

### 5.2.2.6 USB peripherals configuration

**Table 5-11 USB Type-C configuration**

Configuration	Enabled	Comments
CONFIG_QPNP_SMB2	Yes	Enables the PMIC driver to handle USB Type-C events

**Table 5-12 USB kernel configuration**

Configuration	Enabled	Comments
CONFIG_USB	Yes	Main USB kernel configuration
CONFIG_USB_PHY	Yes	Enables the USB PHY framework; required if using separate PHY drivers
CONFIG_USB_MSM_SSUSB_QMP	Yes	Enables the SS-USB QMP PHY driver
CONFIG_MSM_QUSB_PHY	Yes	–
CONFIG_USB_DWC3_MSM	Yes	Enables the DWC3 controller driver
CONFIG_USB_LIBCOMPOSITE	Yes	Enables ConfigFS support
CONFIG_USB_GADGET	Yes	Enables USB gadget support; required for CONFIG_USB_LIBCOMPOSITE
CONFIG_USB_XHCI_HCD	Yes	Enables the XHCI host stack

**Table 5-13 Optional configuration**

Configuration	Enabled	Comments
CONFIG_USB_GADGET_DEBUG_FILES	Yes	Enables added debug files in the UDC layer
CONFIG_USB_GADGET_DEBUG_FS	Yes	Enables debug files when debugfs is mounted

**Table 5-14 USB PD configuration**

Configuration	Enabled	Comments
CONFIG_USB_PD_POLICY	Yes	Enables the PD policy engine
CONFIG_QPNP_USB_PDPHY	Yes	Enables the qnpp-pdphy driver to interact with the policy engine

### 5.2.3 RGMII

This section will be updated in a future revision of this document.

## 5.3 Storage

### eMMC

The eMMC is an embedded storage, which uses a multimedia card (MMC) interface. The following are the eMMC features:

- Flashes the device image, using different tools. For example, QFIL, JTAG, fastboot, and QPST.
- Boots the device by reading the image from eMMC. The boot images are read, for example, SBL, recovery, kernel, RPM, and TrustZone.
- Stores the user data in the user data partition in the HLOS.
- Stores secure TrustZone data in the RPM partition.

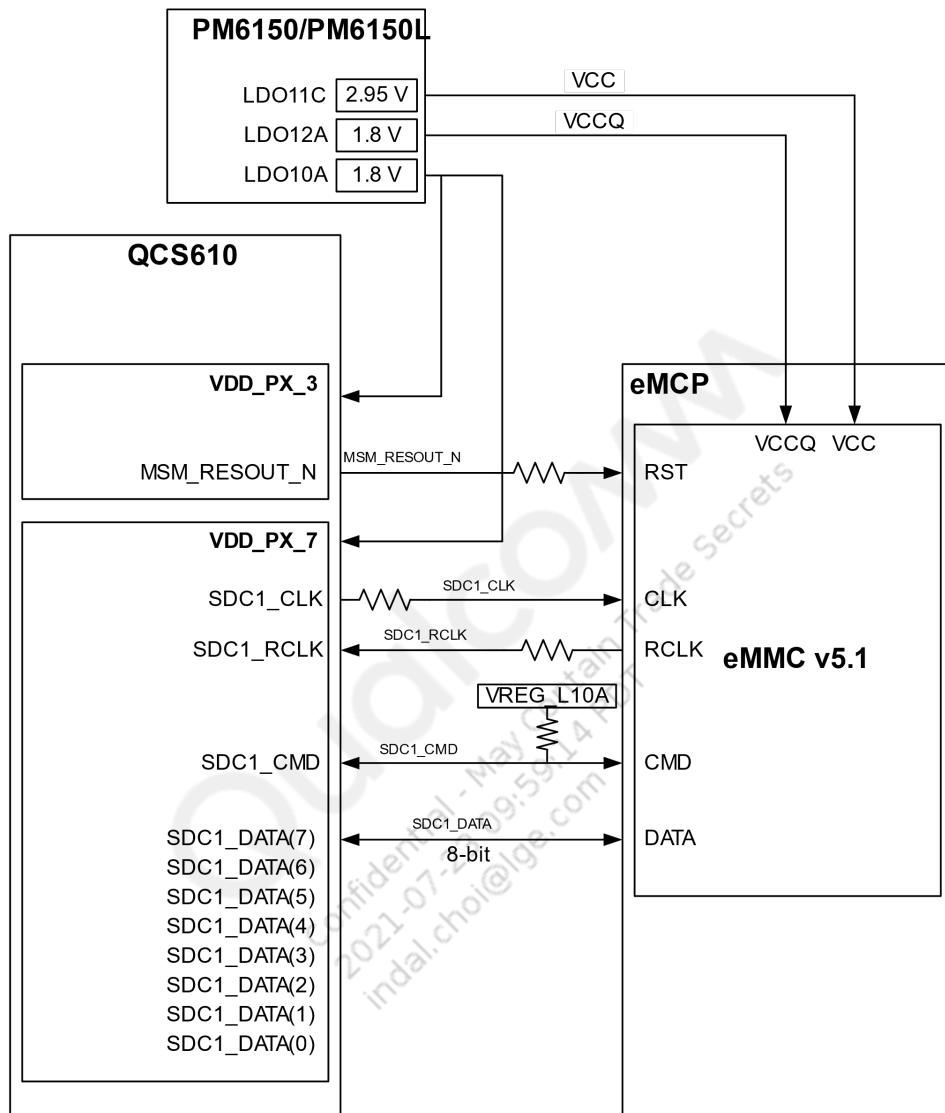


Figure 5-9 eMMC block diagram

### SD card

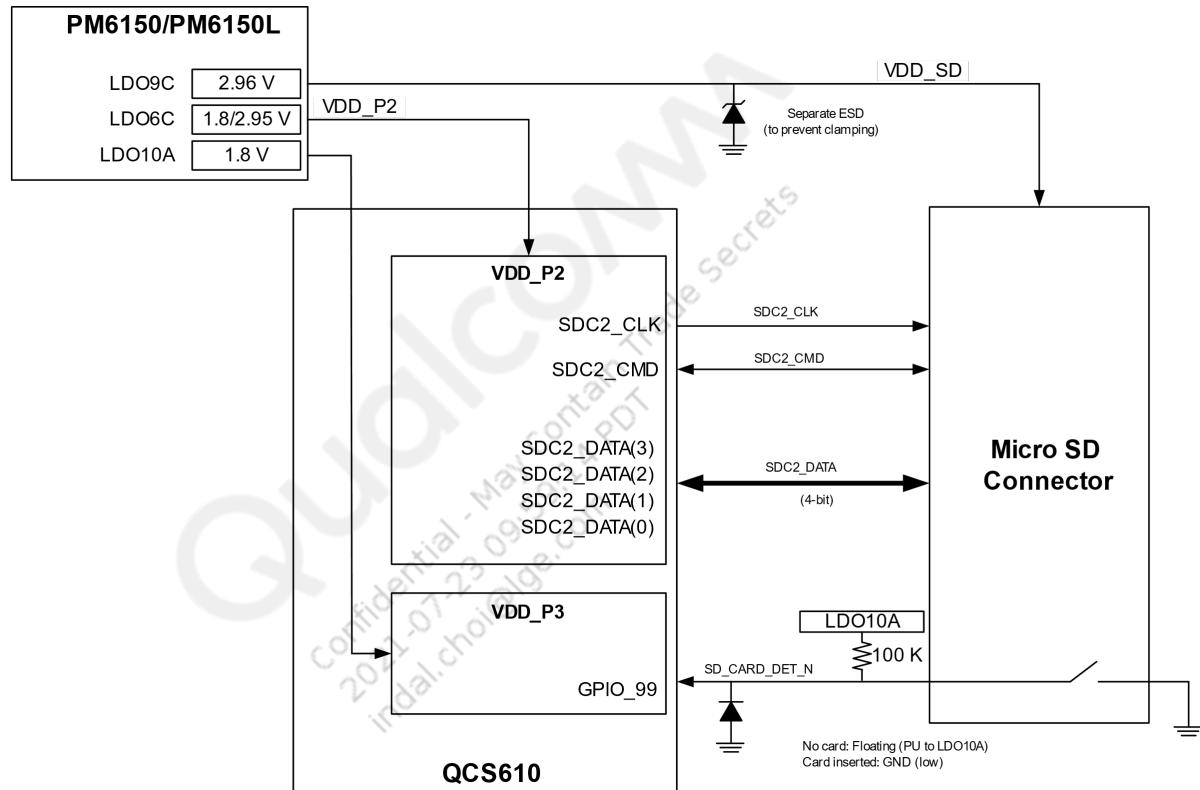
Table 5-15 Storage SD hardware/software specifications

Hardware/software features	QCS610/QCS410 support	Description
SDCC5* core	2 cores	<ul style="list-style-type: none"> <li>SDC1 – 8 bit</li> <li>SDC2 – 4 bit</li> </ul>
Clock speeds supported	192 MHz 100 MHz	<ul style="list-style-type: none"> <li>SDC1 at HS400</li> <li>SDC2 at SDR104</li> </ul>
SD hot-plug detect	Yes	GPIO_96 is used in the QTI reference design

**Table 5-15 Storage SD hardware/software specifications (cont.)**

Hardware/software features	QCS610/QCS410 support	Description
Low-power mode (LPM)	Yes	Only VDD (VCC) goes to 0 V on runtime suspend and system suspend

\*The SDCC5 is the fifth generation of the secure digital card controller (SDCC) core.

**Figure 5-10 SD card block diagram**

The SDCC memory address is shown in the following diagram.

0x007E0000	
0x007D0000	Unused (64 KB)
0x007C8000	PERIPH_SS_SDC1_SDCC_ICE (32 KB) read-write
0x007C6000	Unused (8 KB)
0x007C5000	PERIPH_SS_SDC1_SDCC_SDCC5_HC_CMDQ (4 KB) read-write
0x007C4000	PERIPH_SS_SDC1_SDCC_SDCC5_HC (4 KB) read-write
0x007C0000	Unused (16 KB)

**Figure 5-11 SDC 1**

0x08820000	
0x08805000	Unused (108 KB)
0x08804000	PERIPH_SS_SDC2_SDCC_SDCC5_HC (4 KB) read-write
0x08800000	Unused (16 KB)

**Figure 5-12 SDC 2**

### 5.3.1 Storage features

**Table 5-16 eMMC Linux storage features**

Feature	Description
Boot and storage	Driver support for boot/storage from an 8-bit eMMC card
eMMC RPMB	Driver support for read/write access to RPMB
eMMC HS200	Supports eMMC HS200 bus Speed mode SDR data sampling with frequency up to 200 MHz and data rate up to 200 Mbps.
eMMC HS400	Supports the eMMC HS400 bus speed mode DDR data sampling with frequency up to 200 MHz, and data rate up to 400 Mbps (from eMMC 5.0).
eMMC 4.51	Software support for the eMMC 4.5 power-off notification, packed command, cache command, HPI, discard, and sanitize.
eMMC 5.1	Software support for the eMMC 5.1, firmware update, and command queuing features, enhanced data strobe, RPMB write throughput, secure write protection, cache flush policy/cache barrier, and AUTO BKOPS.

**Table 5-17 SD card Linux storage features**

Feature	Description
SD card driver	Driver supports boot from SD card, collects RAM dump, and uses as external storage.
SD SDR104	Support for SDR104 bus speed mode, a frequency up to 208 MHz, and data rate up to 104 Mbps
GPIO-based SD card detection	The secure digital high capacity (SDHC) driver uses external GPIO as an interrupt signal to detect SD card insertion and removal.

**Table 5-18 SDIO Linux storage features**

Feature	Description
Secure digital input/output (SDIO) for WLAN	Driver support for SDIO to use with WLAN

### 5.3.2 Storage configuration

**Table 5-19 SDHC kernel configuration**

Configuration	Enabled	Description
CONFIG_MMC	Yes	Enables MMC driver
CONFIG_MMC_CLKGATE	Yes	Enables clock gating
CONFIG_MMC_PARANOI_SD_INIT	Yes	Multiple attempts for SD card to resume/attach/setup, and so on
CONFIG_MMC_PERF_PROFILING	Yes	Enables driver performance profiling
CONFIG_MMC_DEBUG	No	Enables driver debug messages
CONFIG_MMC_SDHCI	Yes	Detects SDHCI driver
CONFIG_MMC_SDHCI_MSM	Yes	Detects SDHCI driver for MSM platforms
CONFIG_MMC_TEST	Yes	mmc_test.c compiled  <b>NOTE</b> mmc_test is compiled as a module

## 5.4 PMIC

For an overview of UEFI PMIC software drivers, battery management applications, and battery charging configurations, see *UEFI PMIC Software User Guide* (80-P2484-42).

## 5.5 RPMh

The power and clocks for any shared resources that need to be managed on a fine grained dynamic basis are managed either directly in hardware or by the RPM. For example, clocks and power for shared system buses should be managed either directly in hardware or by RPM.

RPMh is a hardware-based solution with software assistance. With the hardware-accelerated transitions, RPMh achieves significant improvement in transition latency.

RPMh solution provides the following benefits:

- Reduces existing transition timelines by making the systems atleast 3 to 10 times faster
- Improves days-of-use (DoU) by two-and-a-half percent
- Uses always-on processor (AOP) to facilitate debugging

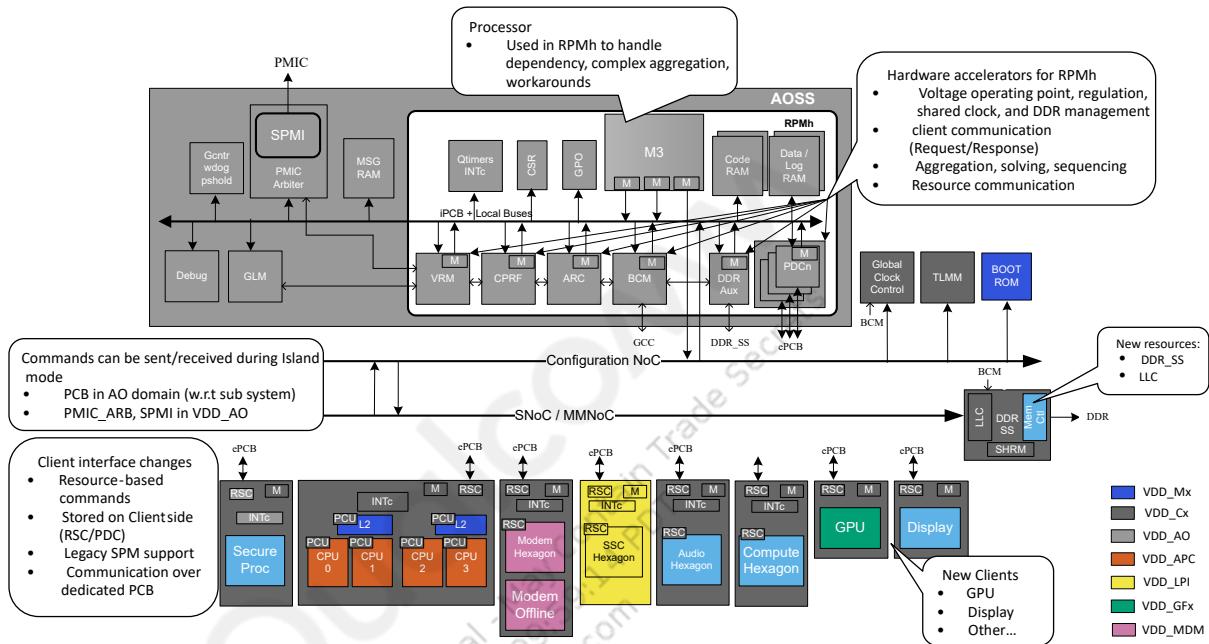


Figure 5-13 RPMh architecture

	RPMh	Functionality
Local	Solver (sensor and display only) (SLVR)	Sleep mode decision
	Resource state coordinator (RSC)	Subsystem resource-controlled Sleep and awake states
	Power domain controller (PDC)	Subsystem sleep and wake-up including interrupt management
	Aggregated resource controller (ARC)	Shared voltage operating point control
	Bus clock manager (BCM)	Bus and memory clock control
	Voltage regulator manager (VRM)	Regulator, crystal oscillator (XO) buffer, and crystal control
	Power control bus (PCB)	Resource and power-controlled message passing
	AOP	Complex aggregation or dependency handling, and workarounds

Figure 5-14 RPMh functionality

Direct resource voter (DRV) – RPMh-based masters:

- Applications (application processor)
- TZ
- Hypervisor
- HLOS
- LPASS
- Sensor
- Secure processor
- AOP
- GPU
- Display

DRV caters to resource dependencies for digital rails. For example, MX >= CX

Resources must vote using the address of the corresponding accelerators.

The CmdDB driver supports resource name to address mapping for an accelerator. CmdDB is in the DDR-shared memory, and each subsystem has CmdDB APIs for the required information.

**Table 5-20 Resource voters**

Resource	RPMh-based
CX, MX	ARC-handled resource
XO clock	ARC-handled resource
GFX	ARC-handled resource
Modem subsystem (MSS)	ARC-handled resource
Shared clocks (NoC, DDR)	BCM-handled resource
PMIC resources (SMPS, LDO)	VRM-handled resource

**Table 5-21 RPMh tasks**

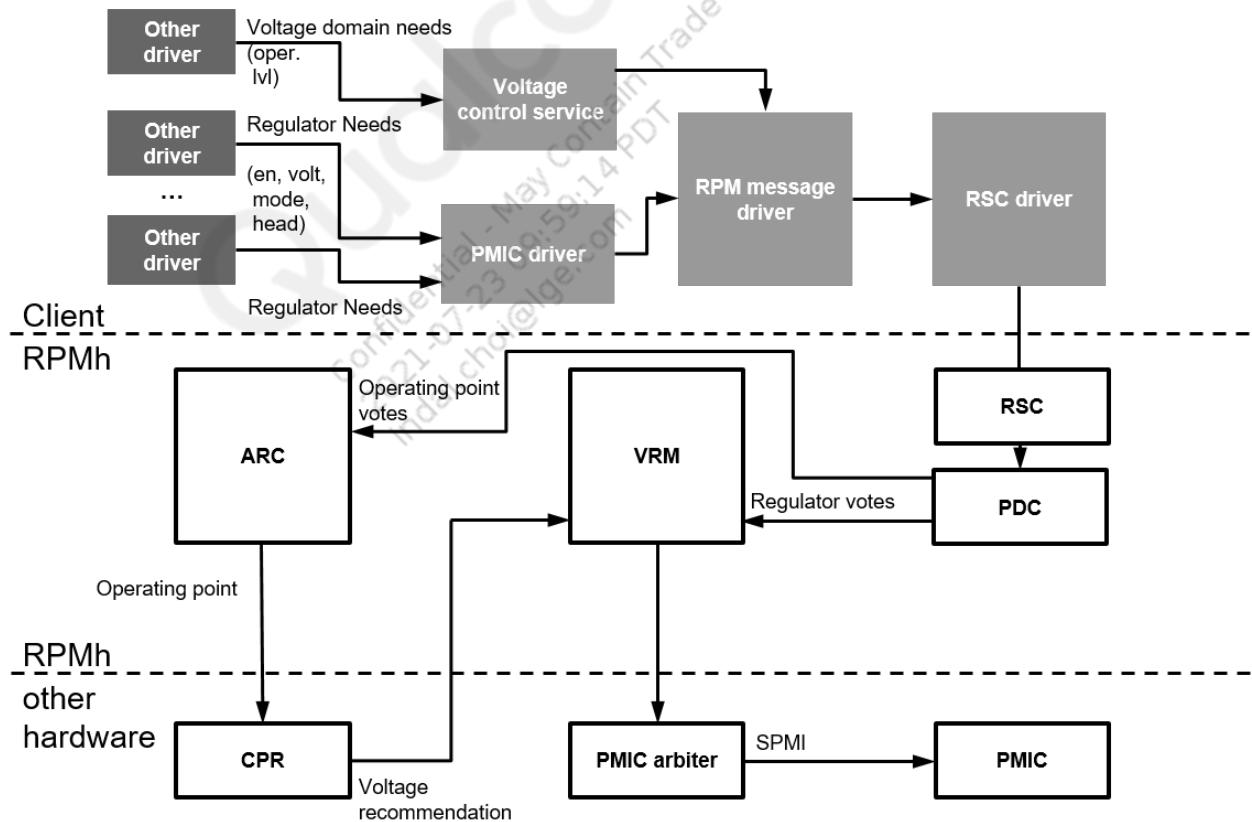
Task	RPMh
Immediate request for a resource	Write to RSC TSC register and send packet to AOP (RPMh driver in each subsystem)
Shut down or wake-up request	Subsystem RSC > subsystem PDC
Wakeup-capable interrupt support	PDC interrupt controller
Active or sleep set information	Stored at the subsystem end (RPMh driver in each subsystem)
Railway driver	PDC of subsystem receives the packet and routes it to ARC Hardware-based ARC aggregation and handling
CPR driver	Hardware-based CPR interrupt handling
Clock or bus driver	Hardware-based BCM aggregation and handling (RSC > PDC > BCM)

**Table 5-21 RPMh tasks (cont.)**

Task	RPMh
PMIC driver	Hardware-based VRM aggregation and handling (RSC > PDC > VRM)
DDR low-power modes	Hardware-based DDR_AUX notifications to SHRM
Latest state of resource	Hardware accelerator register configuration (ARC, CPRF, BCM, and VRM)
Vdd minimization	Hardware-based ARC aggregation and handling
Always-on subsystem (AOSS) sleep	AOP software handling

### 5.5.1 Regulator management

The figure provides an overview of RPMh-augmented regulator control, and describes on a high level, the flow of votes through different drivers to address the voltage and regulator needs.

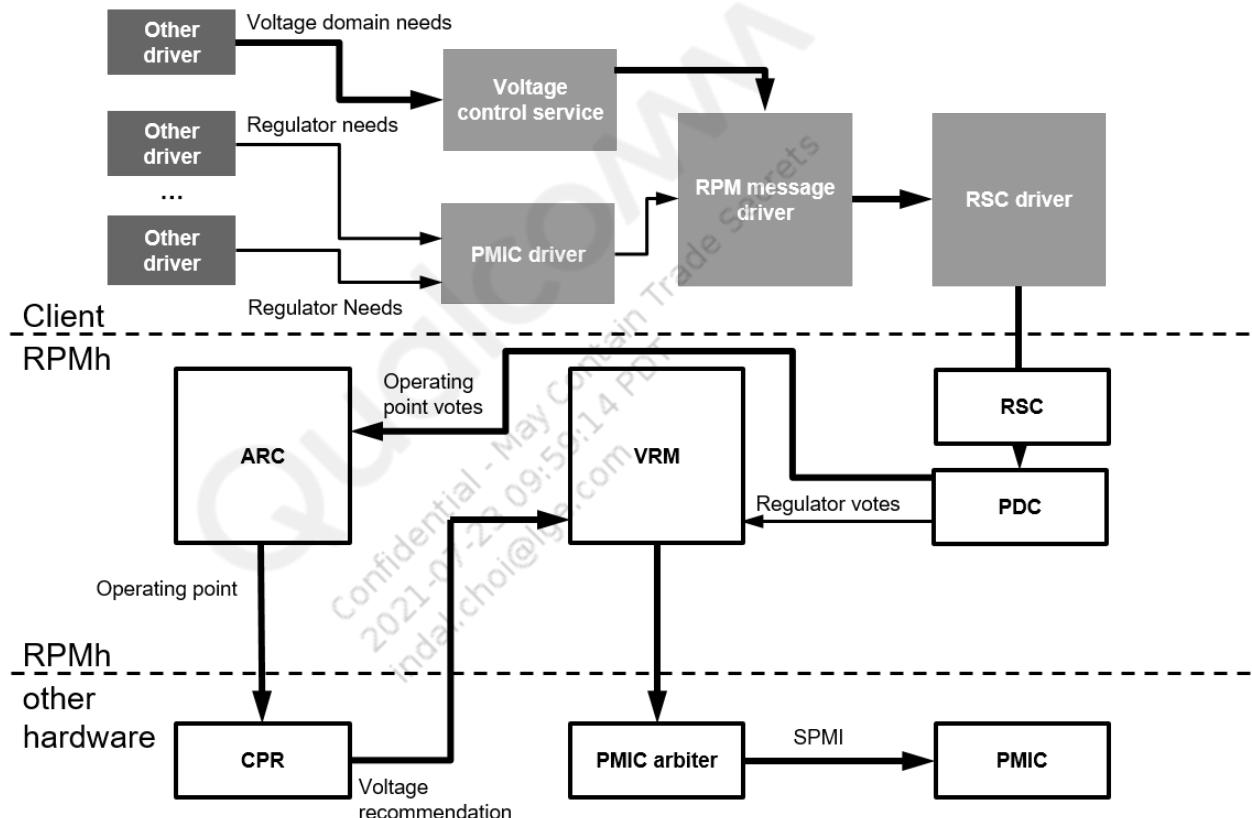
**Figure 5-15 Regulator control in RPMh**

In RPMh, the client side stays the same. The RPM message driver communicates through the RSC driver and PDC over to the hardware components. The RSC driver programs the RSC hardware, after which, the RSC block communicates with PDC through ePCB.

The voting requests are routed to VRM either through PDC, or ARC and CPR. VRM handles the aggregation and communicates the voltage updates to PMIC through the PMIC arbiter.

- If it is a regulator vote, the request is passed to VRM through PDC.
- If it is an operating point vote, the request is passed on to ARC, which communicates with CPR for the recommended voltage. CPR then maps operating points to voltage.
- If there is a voltage change, then CPR directly interacts with VRM.

### 5.5.1.1 Vote on digital voltage domain

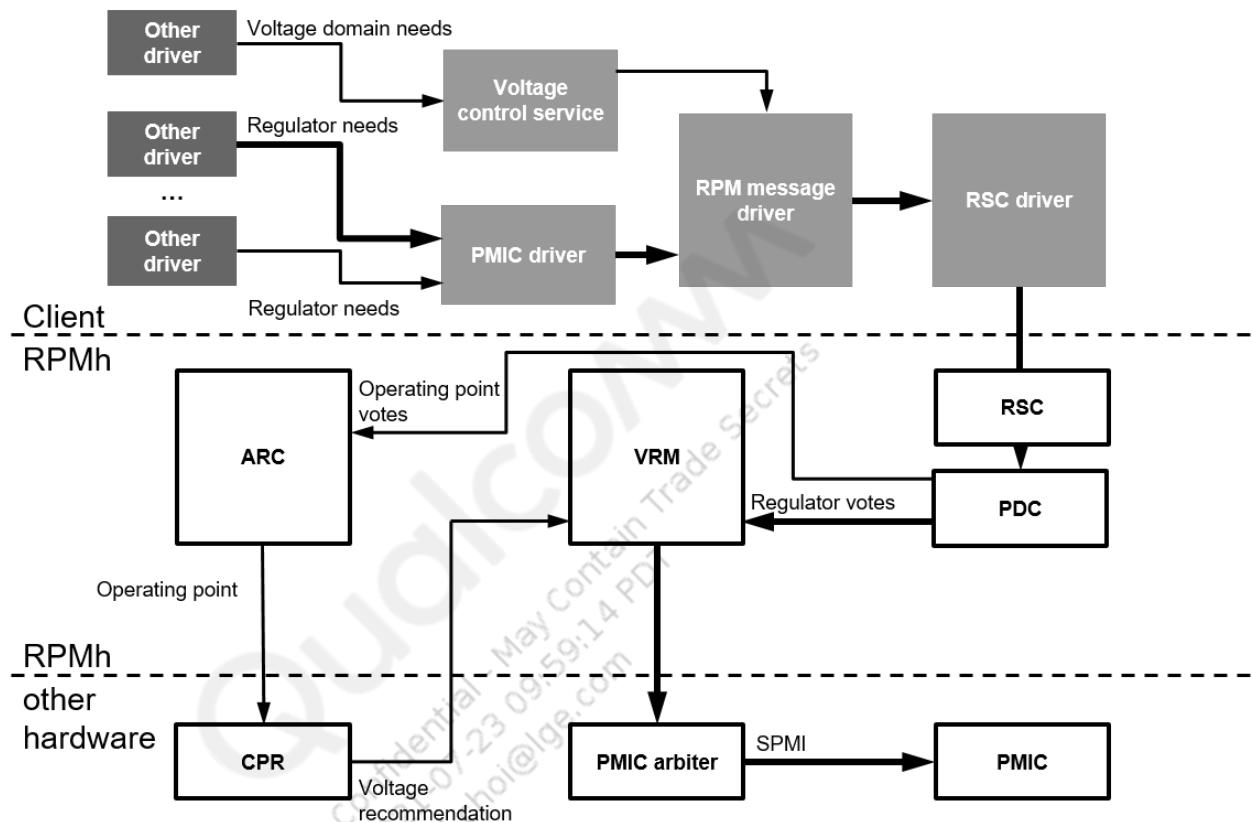


**Figure 5-16 Regulator control for vote on digital voltage domain**

1. The voltage request goes through the voltage control service, RPM message driver, and RSC driver, which programs RSC hardware, and the RSC block communicates with PDC through ePCB.
2. PDC, through communication protocol, sends the request for operating point to ARC.
3. ARC aggregates the operating point to a new value, and communicates the value to CPR.
4. CPR converts the value to a safe voltage, and passes the value to VRM.
5. VRM does the aggregation, and then programs PMIC through the PMIC arbiter.

### 5.5.1.2 Vote on regulator

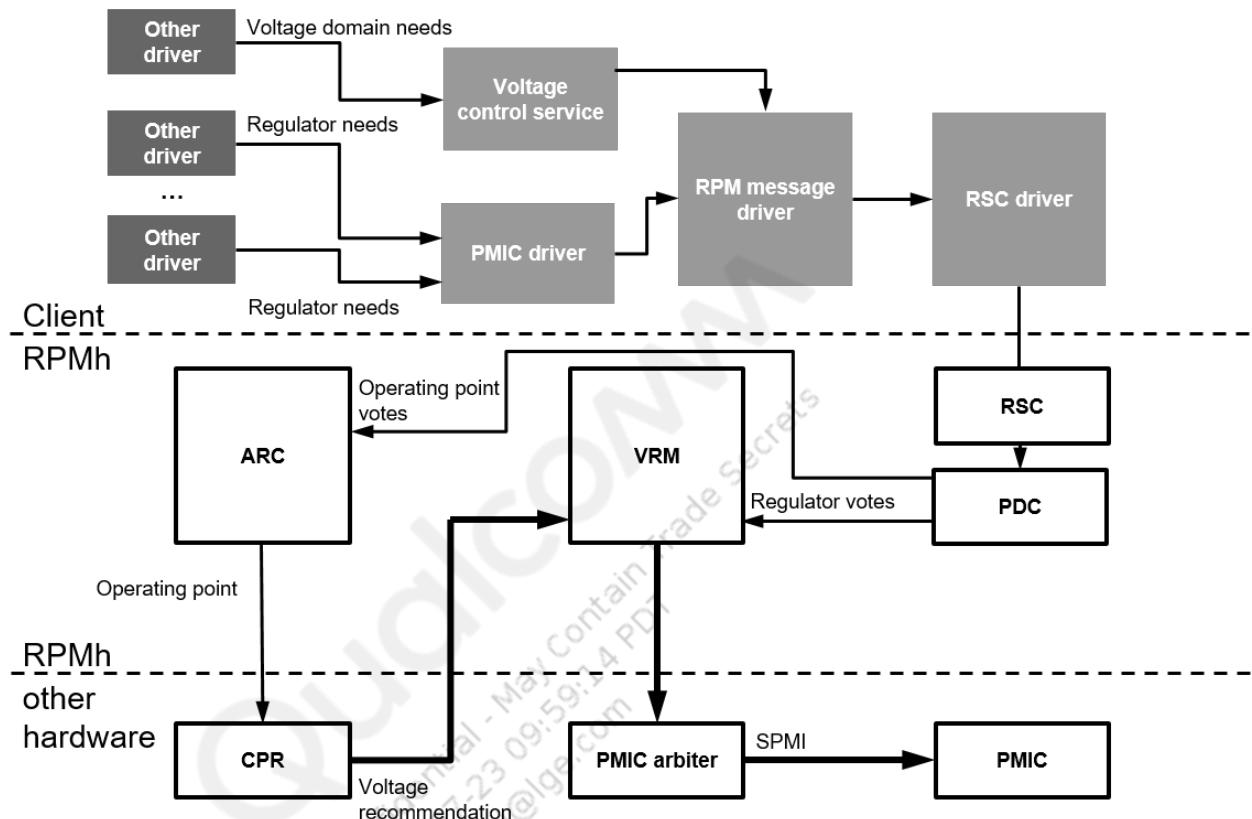
The figure depicts flow of votes for regulator that is not voltage domain. For example, an RF regulator.



**Figure 5-17 Regulator control for vote on non digital voltage domain**

1. The request goes through the PMIC driver, RPM message driver, and RSC driver.  
The RSC driver programs RSC hardware, and the RSC block communicates with PDC through ePCB.
2. PDC targets the regulator votes through VRM.
3. VRM aggregates the votes, sends the aggregation to the PMIC arbiter.  
Through the system power management interface (SPMI), the PMIC arbiter programs PMIC.

### 5.5.1.3 CPR micro adjustment

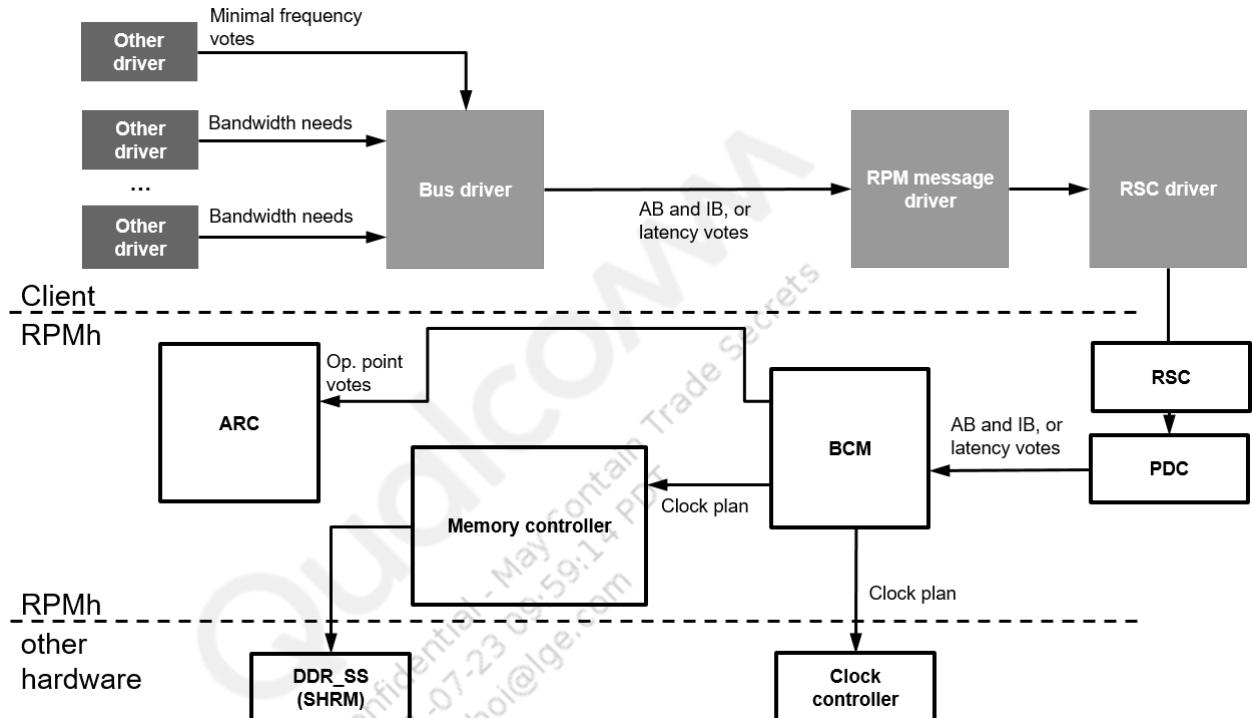


**Figure 5-18 CPR micro adjustments for voltage and regulator domains**

1. CPR detects the micro adjustment needs of the software and communicates the updated voltage to VRM.
2. VRM aggregates the new voltage and sends it to the PMIC arbiter to program PMIC.

## 5.5.2 Frequency management

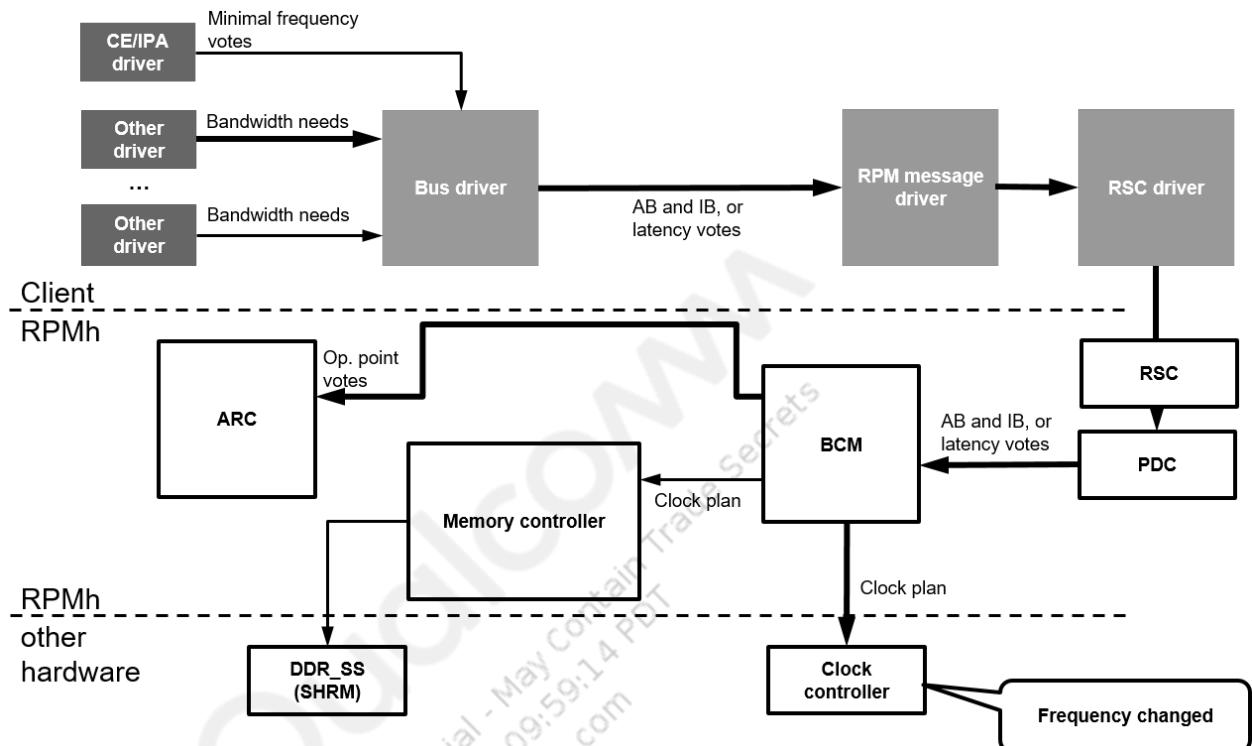
The figure provides an overview of frequency control and on a high level, describes the flow of votes through different drivers to address the changes, which take place either through ARC or memory controller.



**Figure 5-19 Frequency control in RPMh**

1. In RPMh frequency management, as there are no minimal frequency votes, the votes cannot be sent to the clock driver for frequency changes due to the deprecated API.
2. The bus driver passes the average bandwidth (AB) and instantaneous bandwidth (IB), or latency votes to BCM.
3. The BCM is responsible for aggregating and mapping the settings, coordinating with clock controller and DDR to facilitate the frequency change.
4. It also interacts with ARC for any voltage dependency (different operating levels such as CX, MX).

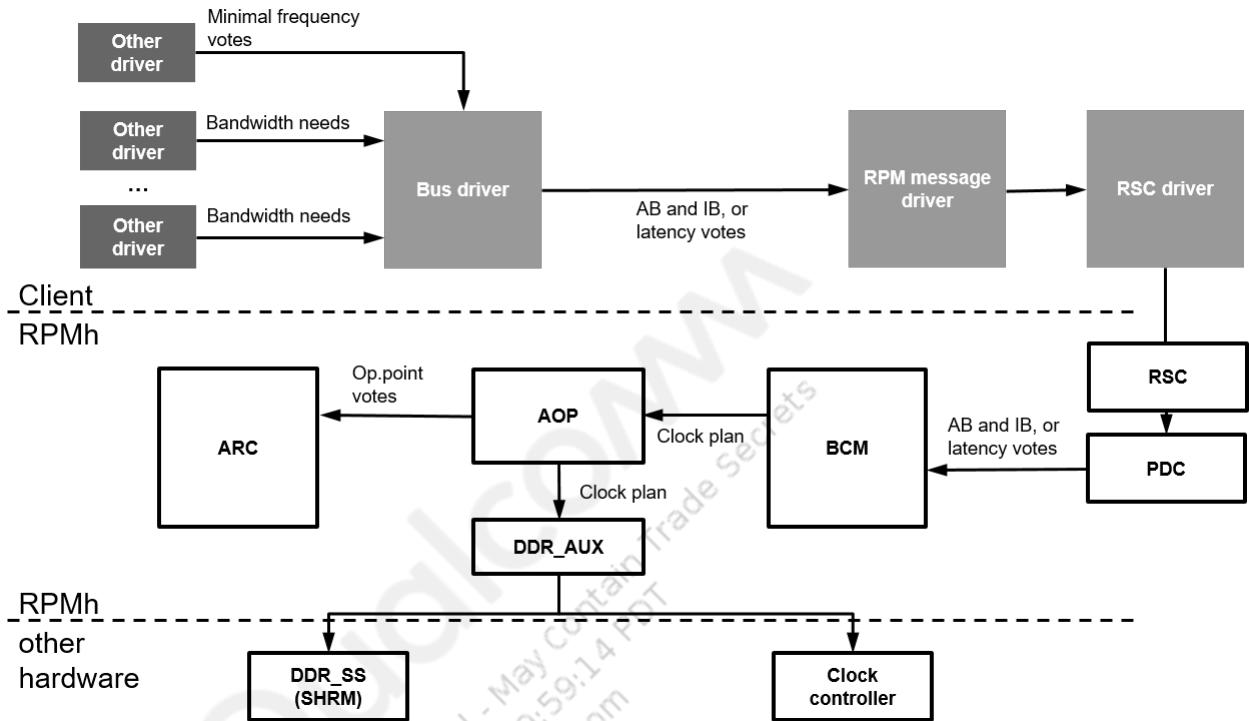
### 5.5.2.1 Frequency control – bandwidth vote



**Figure 5-20 RPMh frequency control through bandwidth votes**

1. The bus driver passes the AB and IB, or latency votes to BCM.
2. BCM aggregates the AB, IB values to determine and map them to a clock plan (CP), which is indicated by the CP index that provides the bandwidth range.
3. It determines the voltage dependencies for the clock plan where the index is available and sends the dependencies to ARC to set the operating points. The dependencies are managed in regulator management. For more information, see [Regulator management](#).
4. BCM handshakes with the clock controller to switch the frequency. The clock controller updates the frequency.

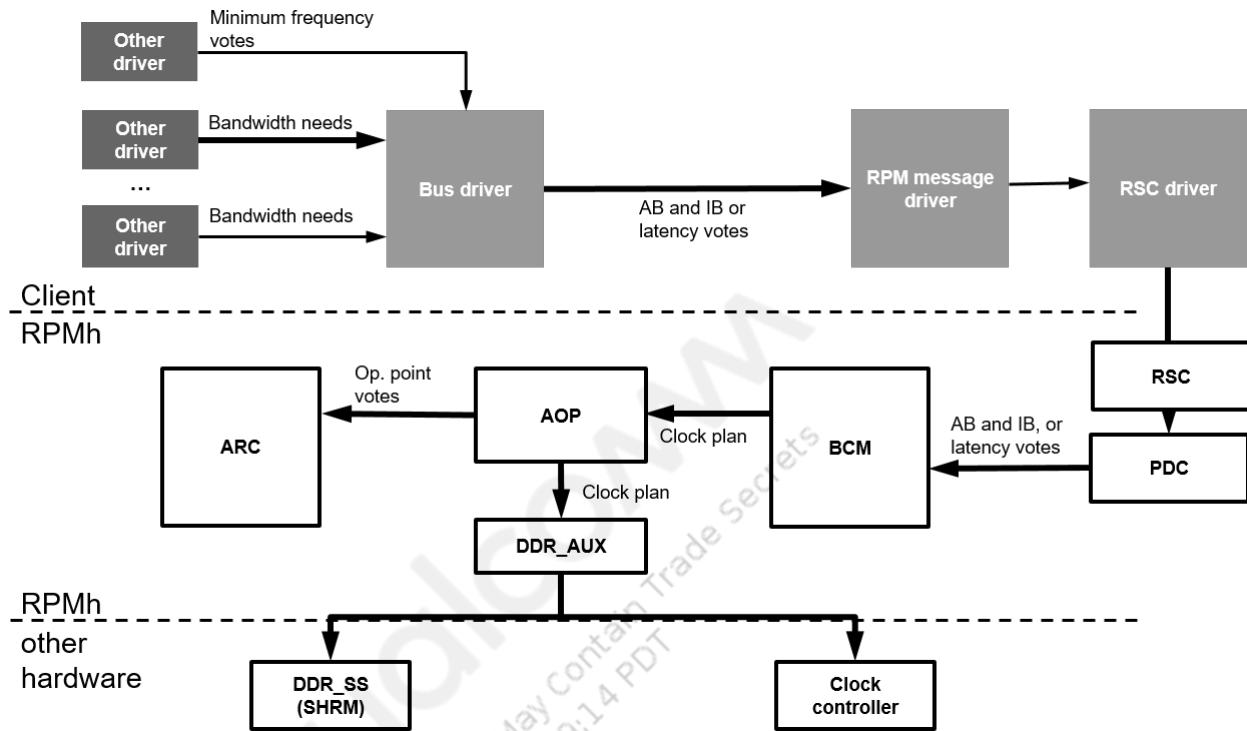
### 5.5.2.2 Memory controller



**Figure 5-21 RPMh frequency control for memory and DDR subsystem**

1. BCM aggregates the two-clock domains associated with DDR\_SS. AOP is leveraged to handle memory needs.
2. BCM sends the appropriate clock plan index to AOP, which is responsible for coordinating the unique requirements to be able to handle voltage domain changes and regulators, handling intermediate frequencies, and other so on.
3. For voltage-related changes (for voltages like CX, MX and VDDa), AOP makes the dependency request to ARC.
4. For memory-related updates, AOP uses DDR-AUX as a sequencer offloader to sequence the frequency changes with DDR\_SS and clock controller.

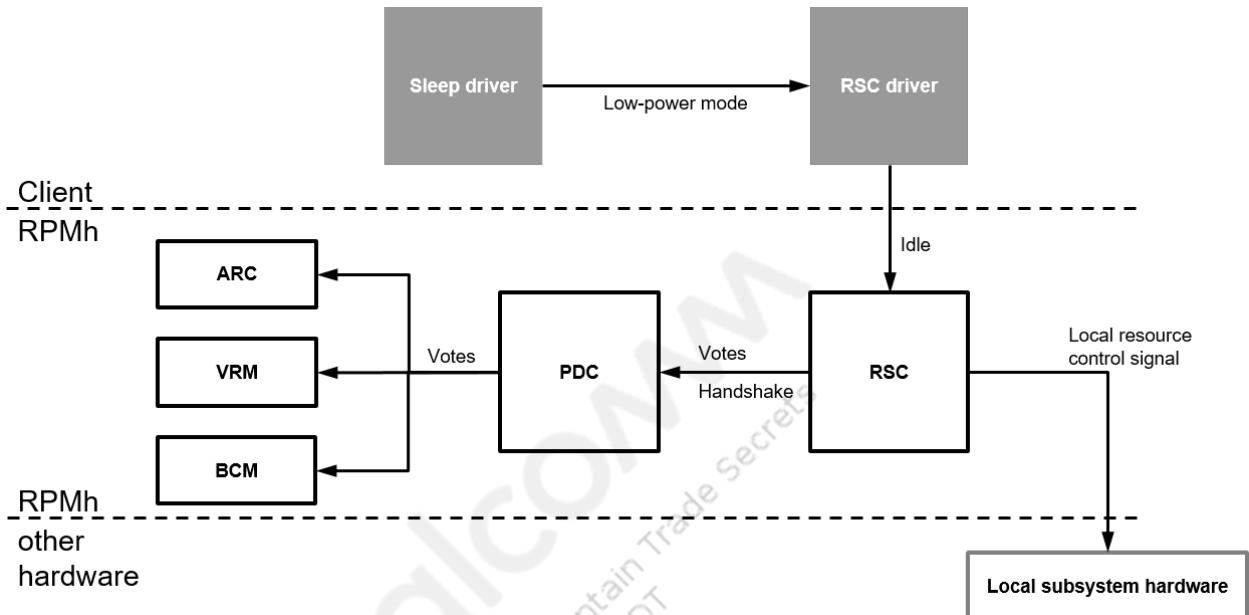
### Memory controller – bandwidth vote



**Figure 5-22 RPMh frequency control for memory through bandwidth votes**

1. BCM aggregates the clock plan.
2. BCM provides the clock plan index to the AOP, which handles the dependencies including frequency jumps.
3. AOP sends votes of the dependencies to the ARC.
4. AOP uses the DDR\_AUX to facilitate the frequency change.

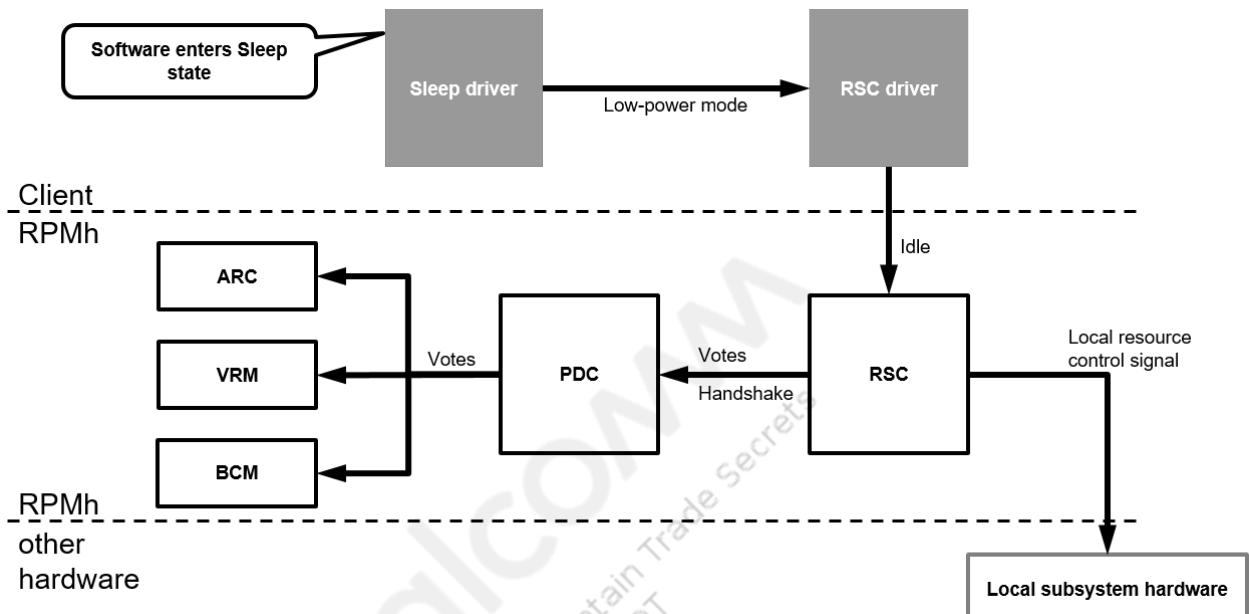
### 5.5.3 Sleep management



**Figure 5-23 Sleep management for RPMy**

1. Sleep driver picks the Low-power mode and communicates with the RSC driver.
2. While SPM was responsible for sequencing for local resources in RPM, RSC performs this function in RPMy.
3. RSC votes on other resources. This is where the stored votes are triggered later. These votes conceptually represent the Sleep and Wake states.
4. RSC votes over to PDC, which is responsible for voting the resources to make RSC functional. Cx, Mx, and XO are the typical minimum number of votes that are placed within the PDC.
5. PDC also hosts the always-on interrupt controller and always-on timer.

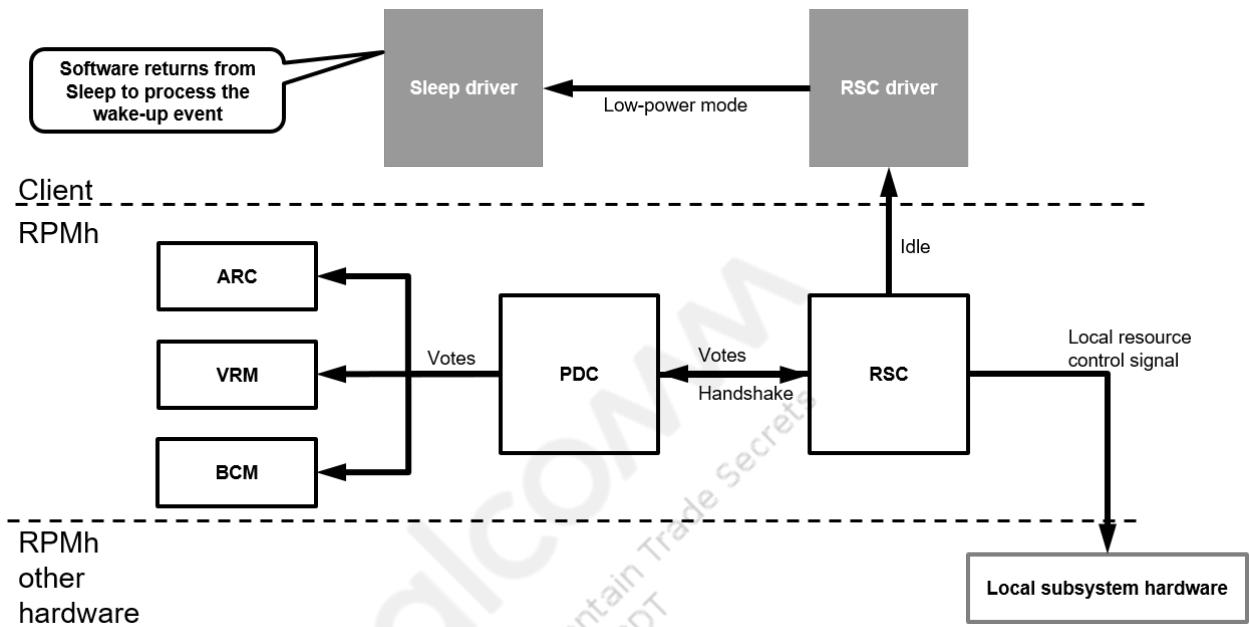
### 5.5.3.1 Sleep enter



**Figure 5-24 RPMh workflow for entering Sleep state**

1. Sleep driver enters the Sleep state.
2. RSC driver programs the Low-power mode. The Client enters the Idle state.
3. The RSC does the local resource control, which includes resource control for local subsystem hardware.
4. RSC votes for resources to Low-power mode. For example, in modem case, turning off RF resource, or regulators.
5. RSC handshakes with PDC.
6. PDC votes resources to Low-power mode. These votes only go to ARC because the votes are for cx, mx, and XO, which are ARC-based.
7. PDC waits for timer to expire or wake up interrupt to occur.

### 5.5.3.2 Sleep exit



**Figure 5-25 RPMh workflow for exiting Sleep state**

1. Timer expires or wake-up interrupt occurs.
2. PDC votes resources back to operational. For example, it brings cx, mx, xo from collapse retention off to Nominal ON.
3. PDC handshakes with RSC.
4. RSC votes on resources to become operational. This provides voting bandwidth for the processor to be functional again. In modem, it maybe turning on all RF regulators.
5. RSC does local resource control.
6. Processor starts running again and software exits the Sleep state.

### 5.5.4 SoC sleep

The system sleep is tightly-coupled into the following sleep modes:

- XO Shutdown
- Vdd minimization (Vdd\_Min)

The SoC resource sleep states are independent. In RPMh, there are no sleep modes. The following sleep states are available for a resource:

- XO – off
- Vdd\_CX
  - Retention
  - Collapse (off)

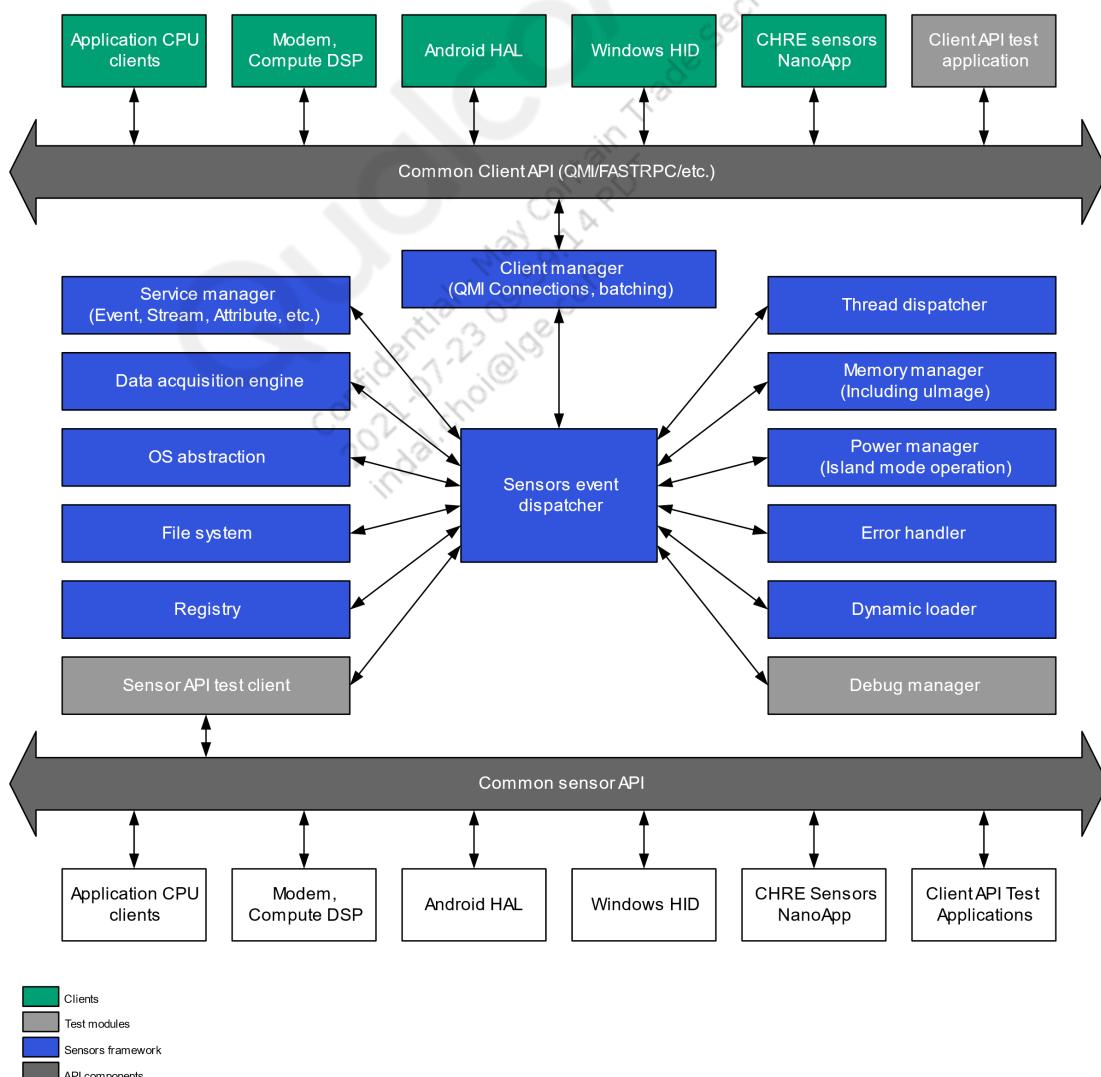
- Vdd\_MX – Retention
- AOSS – The sleep state occurs when the subsystem goes into a low-power state

## 5.6 Sensors

The following diagram shows the sensors execution environment (SEE) software architecture. It includes the new SEE architecture, replacing the device driver framework (DDF)/sensors manager (SMGR) architecture that was used earlier. The Qualcomm Snapdragon Sensors Core (SSC) drivers and algorithms must be revised to comply with the new SEE APIs.

For additional information on SEE, see *Sensors Execution Environment (SEE) Sensors Deep Dive* (80-P9301-35).

**NOTE** On LE software product, Sensor HAL is not available. SEE APIs and native utilities (as reference) can be used to develop sensor client in HLOS side.



**Figure 5-26 SEE software diagram**

## 5.6.1 LE software product-based platform

### Low-power mode support

The accelerometer/gyroscope driver is supported in the Island mode.

The following algorithms are supported in Island mode:

- AMD
- SMD
- Pedometer
- Tilt
- Gravity/linear acceleration
- Game rotation vector
- PMD (for stationary/motion detect)
- Device orientation
- Gyroscope calibration (QGyroCal)

### Hardware system architecture

The sensors present on QCS610/QCS410 QTI reference platforms are enabled by default. For more information on the reference platform, see *QCS610/QCS410 Linux Platform Development Kit Quick Start Guide* (80-PL631-300).

**Table 5-22 QTI reference platform sensors**

Sensor type	Part (Vendor)	Interface
Accelerometer/Gyroscope	BMI160 (Bosch)	SPI

Sensors are connected to BLSP 5 on the QCS610/QCS410 QTI reference platform.

**Table 5-23 Recommended SSC GPIOs and QUP usage**

QUP	SSC GPIO	GPIO usage	Sensors connected
QUP0	SSC_0	SSC_I2C1_SDA	Not used
	SSC_1	SSC_I2C1_SCL	
QUP1	SSC_2	SSC_SPI1_MISO	Accelerometer/Gyroscope
	SSC_3	SSC_SPI1_MOSI	
	SSC_4	SSC_SPI1_CLK	
	SSC_5	SSC_SPI1_CS0	
	SSC_6	Not used	
	SSC_7		
QUP2	SSC_8	Not used	
	SSC_9		

**Table 5-23 Recommended SSC GPIOs and QUP usage (cont.)**

QUP	SSC GPIO	GPIO usage	Sensors connected
	SSC_10		
	SSC_11		
QUP3	SSC_12	SSC_UART1_TX	Not used
	SSC_13	SSC_UART1_RX	
QUP4	SSC_14	SSC_UART2_TX	Not used
	SSC_15	SSC_UART2_RX	
QUP5	SSC_16	SSC_UART3_TX_DBG	Not used
	SSC_17	SSC_UART3_RX_DBG	

Sensors interrupt GPIOs for QCS610/QCS410 QTI reference platform:

- Any available (MPM wake-able) GPIO can be routed as an interrupt to the SSC subsystem.
- QTI recommends using the same GPIO configuration as specified in the table.

**Table 5-24 GPIO numbers for different sensor interrupts**

MSM GPIO	Use on QTI reference platform
85	ACCEL_INT
86	GYRO_INT*
87	MAG_DRDY_INT*
84	ALSPG_INT_N*
50	HALL_INT_N*

<sup>1</sup>

**Table 5-25 Sensors power rails on QCS610**

Sensor type	Power rails
Accelerometer/Gyroscope	VDD/VDDIO – LDO8 (1.8 V)

QTI strongly recommends that OEMs use the same power rails. If OEMs want to use different power rails, review and receive approval from QTI hardware, PMIC, and Sensors teams in advance of designing a hardware platform. For more information, see [Technical assistance](#).

<sup>1</sup> \*These interrupt pins are not used on QTI references platforms. These pins are reserved for purpose noted in the table. OEMs can use these pins for respective sensors.

## 5.6.2 High-level platform overview

### Host architecture

The SEE framework runs on Hexagon processor V66K. The Hexagon processor is a general-purpose digital signal processor designed for high performance and low power across a wide variety of multimedia and modem applications.

**Table 5-26 QCS610/QCS410 sensor hardware specifications**

Feature	Specification
Processor	<ul style="list-style-type: none"><li>▪ Hexagon V66K – 864 MHz nominal, 998.4 MHz Turbo</li><li>▪ Two hardware threads</li><li>▪ Shared with audio</li><li>▪ Low Power Island</li></ul>
Cache	<ul style="list-style-type: none"><li>▪ L1 16I/16D</li><li>▪ Island memory – 768 KB</li></ul>
QUP FIFO bus interface	<ul style="list-style-type: none"><li>▪ QUPv3</li><li>▪ 5 dedicated QUPs (1 I2C, 2 SPI, 2 UART)</li><li>▪ 32 LPI GPIOs (shared for sensors and audio)</li><li>▪ 18 dedicated GPIOs for sensors</li><li>▪ Supports Island mode</li></ul>
Features	All prior features supported

### Firmware architecture

SEE does not contain any platform level firmware routine or code.

### System concurrency impact

Sensors are enabled only when client requests for the sensor data. The client may reside on any system like application processor/HLOS, ADSP, SLPI, mDSP, and so on. The SEE framework by design supports concurrent clients possibly spread across different systems.

### Tools for sensor support

The test tools/applications are available to execute different functionalities for the available sensors. For additional information on test tools, see *Sensors Execution Environment (SEE) Sensors Deep Dive* (80-P9301-35).

## 5.6.3 Sensors configuration

The following sensor software customization can be done for sensors drivers and algorithms.

### Driver customization

The sensor vendor tests the drivers according to the QTI driver acceptance checklist and distributes them.

If a sensor is not listed in the QTI reference design, the sensor vendor can develop a driver with SEE. Most vendors have access to OpenSSC 5.x tools.

To integrate the sensor driver into the SSC, see *Sensors Execution Environment (SEE) Sensors Deep Dive* (80-P9301-35).

### **Algorithm customization**

Use the sample template algorithm (OEM1) at `adsp_proclssc\sensors\oem1` for algorithm development. For more details, see *Adding a Custom Sensors Algorithm with Sensors Execution Environment (SEE)* (80-P9301-67).

Qualcomm Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

# 6 QCS610/QCS410 HLOS software architecture

---

This chapter provides an architecture, features, and support information of the following QCS610 functionalities:

- Display
- Audio
- Camera
- Video
- Graphics
- Thermal
- Docker

## 6.1 Display

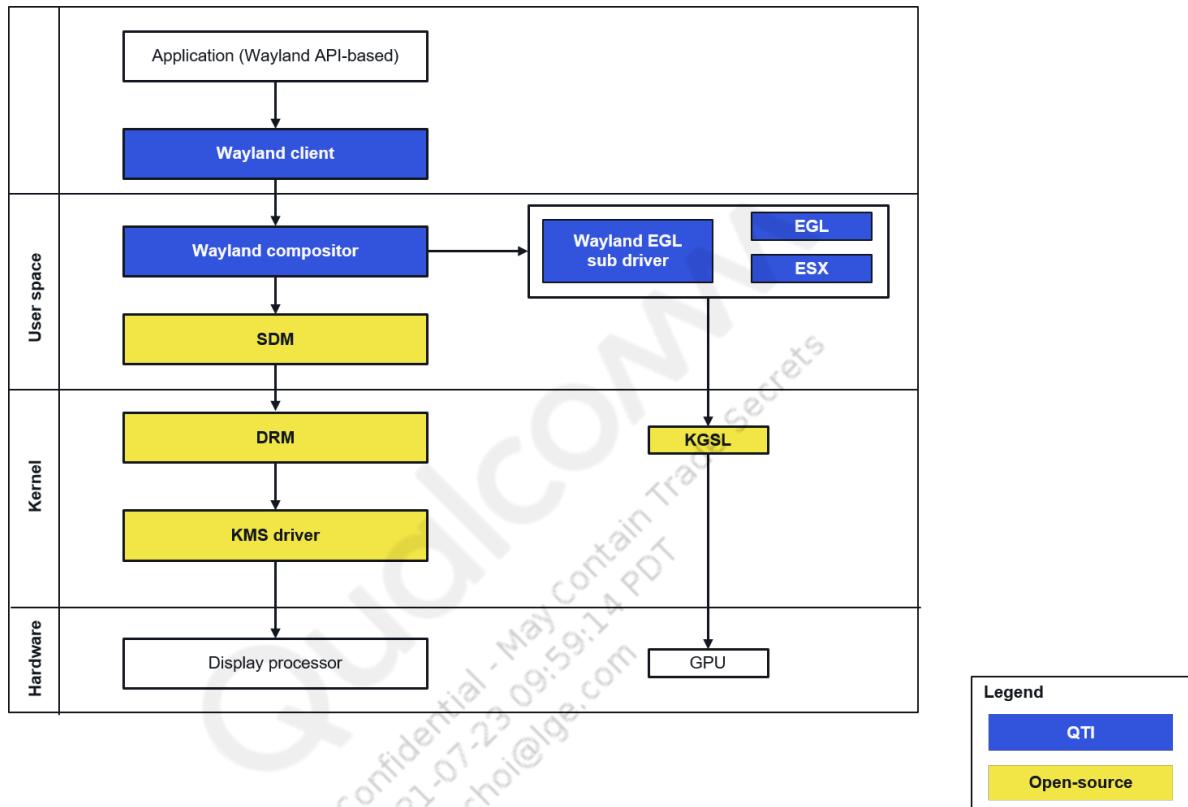
The QCS610/QCS410 chipset supports the following display features:

- Weston Wayland framework for UI
- Interfaces for primary display
- DSI0 – One 4-lane MIPI DSI port, supports up to 1920 × 1200 at 60 Hz or 1080 × 2520 at 60 Hz
- HDMI (DSI-HDMI bridge-LT9611) supported up to 1920 × 1080 at 60 fps for external display

No display support in UEFI or boot loader

## 6.1.1 Display architecture

As shown in the figure, the display software architecture implements the Wayland display framework.



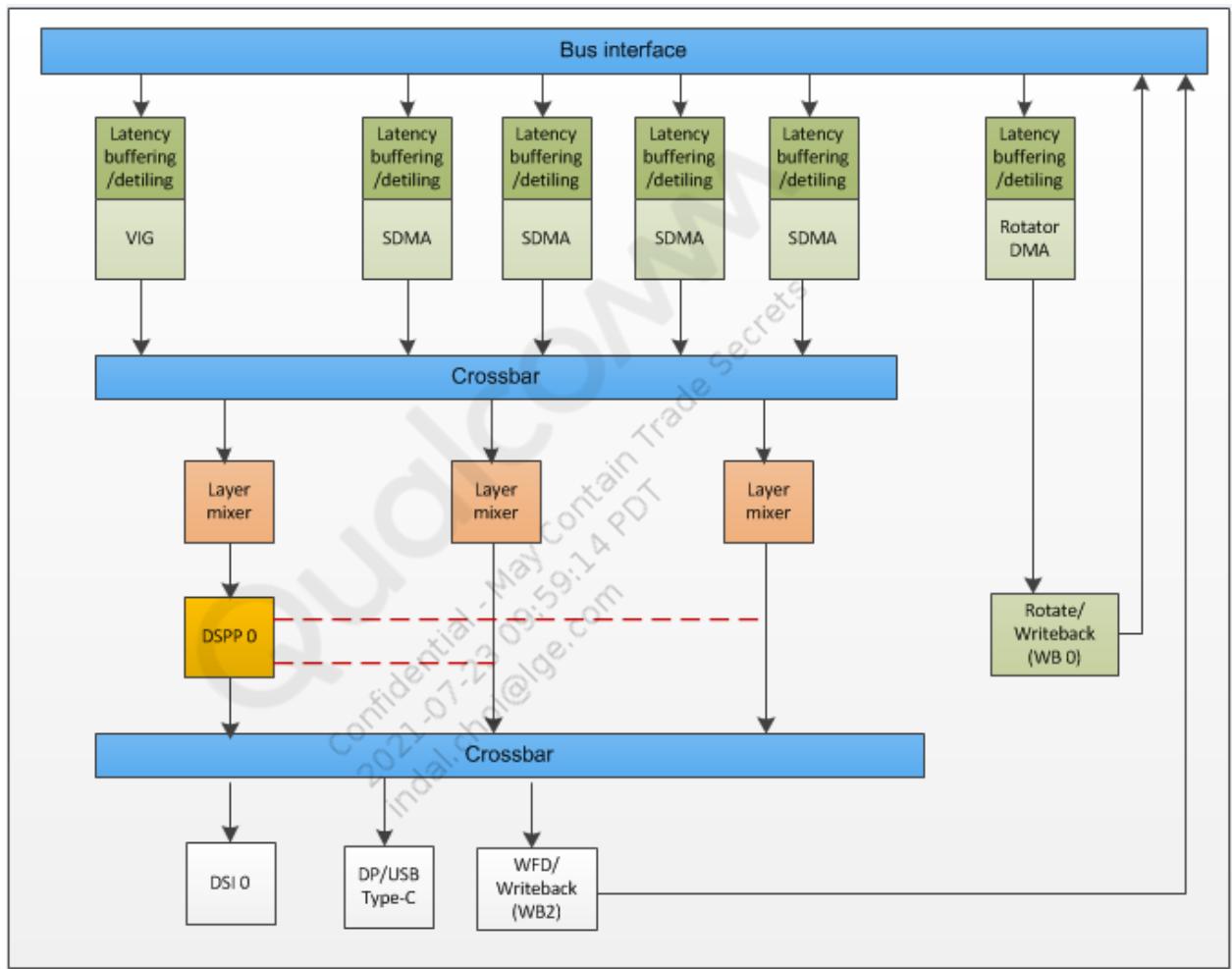
**Figure 6-1 Display software architecture**

The following are the display architecture components:

- Wayland framework: Wayland is a display server protocol.  
A display server using the Wayland protocol is called a Wayland compositor. The reference implementation of a Wayland compositor is called Weston.  
The Wayland protocol follows a client-server model:
  - The clients are the graphical applications requesting the display of pixel buffers on the panel.
  - The server (compositor) is the service provider controlling the display of these buffers.
- QTI hardware abstraction layer (HAL)  
The QTI HAL is called Qualcomm Snapdragon display manager (SDM). It consists of composition manager, display interface, overlays, generic buffer management (GBM), and hardware composer.
  - Composition manager: A QTI proprietary software framework that manages display processor hardware resources and selects the best composition strategy for each connected display.
  - Display interfaces: An interface that is unique to each connected display. It controls and pushes data created by the display processor on to the panel.

The user application client interacts with the Wayland server and performs abstraction at the framework level. Wayland communicates with the SDM to select an appropriate composition strategy and allocate display processor hardware resources for the layer stack received from the client.

## 6.1.2 Display subsystem



The display processor consists of the following major modules:

- Adreno display processing unit (DPU) provides hardware-accelerated image processing, rapid transfer of image data to display interfaces, and enhanced on-screen image quality.
- Source surface processor (VIG and DMA source surface processor pipes (SSPP)) enables format conversion and quality improvements for source surfaces. For example, video and graphics.
- Layer mixer (LM) blends and mixes source surface together
- Destination surface processor pipes (DSPP) enables conversion, correction, and adjustment based on panel characteristics
- Write back/rotation writes back to memory and rotates, if needed.
- Display interface acts as a timing generator and interface connecting to the display peripheral.

- SDMA allows two RGB source layers with a single pipe:
  - Available on VIG and DMA pipes
  - Supports source split
  - Supports both horizontal flip and vertical flip
  - Supports overlapping layers, maximum two layers
  - Supports different formats for two rectangles
- Parallel Fetch mode is used when the rectangles share a scanline. The maximum rectangle width is half of the SSPP width.
- Serial Fetch mode/Time Multiplex mode is used when one rectangle needs to be processed before the second rectangle:
  - Maximum width supported is same as the SSPP width
  - Rectangle R1 starts at least two linear lines or two Qualcomm® Universal Bandwidth Compression (UBWC) tile rows (eight lines) after Rectangle R0

The processor adds an exclusive rectangle. A portion of the rectangle is excluded from hardware fetching of the source rectangle.

The following are the restrictions:

- Only ARGB formats are supported
- No scaling

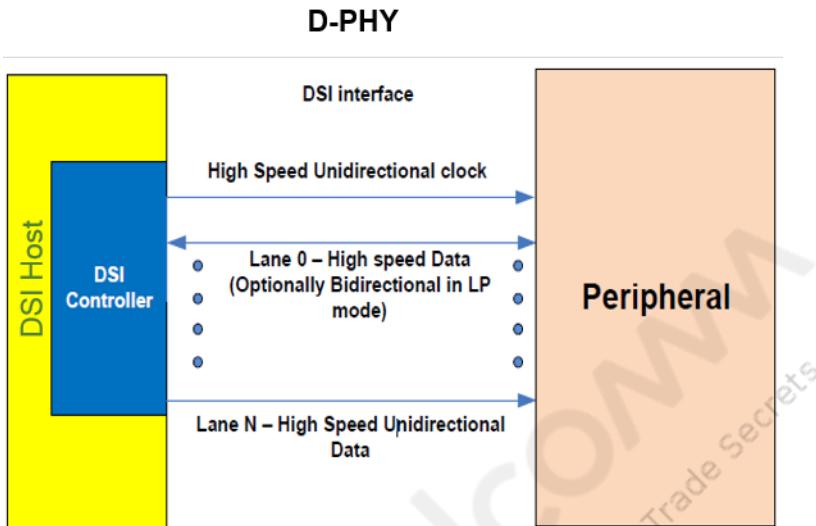
### 6.1.3 Layer mixer

The layer mixer blends the output of SSPP before sending the output to the display device. The layer mixer blends different video and graphics layers as per transparency specified by the parameter alpha.

### 6.1.4 Destination surface processor

The destination surface processor is composed of one or more DSPP. The DSPP performs picture adjustment, inverse gamma correction, gamut mapping, color correction, and panel correction LUT.

### 6.1.5 Display interface



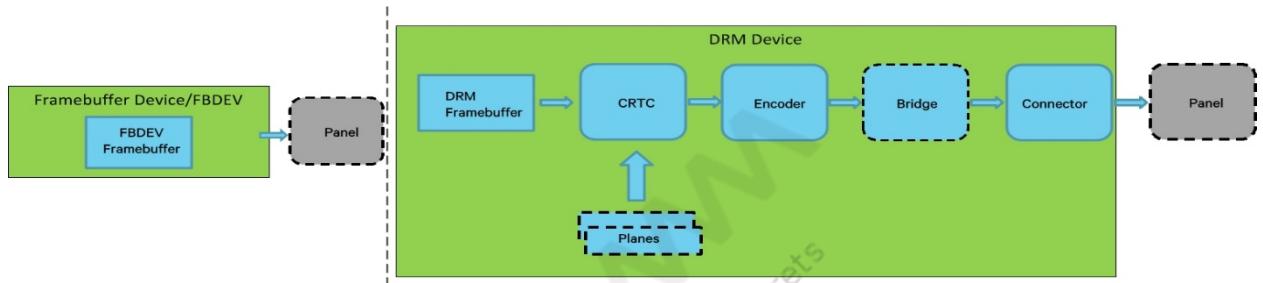
- DSI interface is implemented to support the MIPI Alliance standard for DSI. The device Supports DSI 1.2 with data rates up to 2.5 Gbps per lane with D-PHY.
- In D-PHY mode, each DSI interface supports one differential clock lane and up to four differential data lanes.
- DSI-compliant peripherals operate in Command mode and Video mode.

### 6.1.6 Direct rendering manager and Kernel mode setting

- Direct rendering manager (DRM) is introduced to deal with graphic cards embedding GPUs
- Kernel mode setting (KMS) is a sub part of the DRM API, sets mode on the given display
- Rendering and mode setting are split into two different APIs and are accessible through /dev/dri/renderX and /dev/dri/controlDX
- KMS provides a way to configure the display pipeline of a graphic card (or an embedded system)
- KMS is an alternative to frame buffer dev (FBDEV)

Feature	FBDEV	DRM/KMS
Upstream	<ul style="list-style-type: none"> <li>▪ Less actively maintained</li> <li>▪ Does not provide all the features like overlays, cursors</li> <li>▪ Developers are now encouraged to move to DRM/KMS</li> </ul>	<ul style="list-style-type: none"> <li>▪ Actively maintained by open source</li> <li>▪ Better control on display pipeline</li> <li>▪ Widely used standardized architecture for Graphics – Display protocol</li> <li>▪ Full set of advanced features</li> </ul>
User/Kernel APIs	Custom overlay/atomic commit APIs	Standard (kernel upstream) APIs
Source pipes management	<ul style="list-style-type: none"> <li>▪ Managed through enum index and atomic commit IOCTL</li> <li>▪ Features exposed through DPU driver capabilities</li> </ul>	<ul style="list-style-type: none"> <li>▪ Plane objects managed through properties and atomic mode set commit</li> <li>▪ Features exposed through plane properties</li> </ul>

Feature	FBDEV	DRM/KMS
Layer mixer management	Handled within driver, difficult for user mode to know what is going on	Default allocation in driver. User mode can query and override, virtualization is taken care in CRTC
Panel management	Managed through several sysfs nodes	Managed as connector object with properties



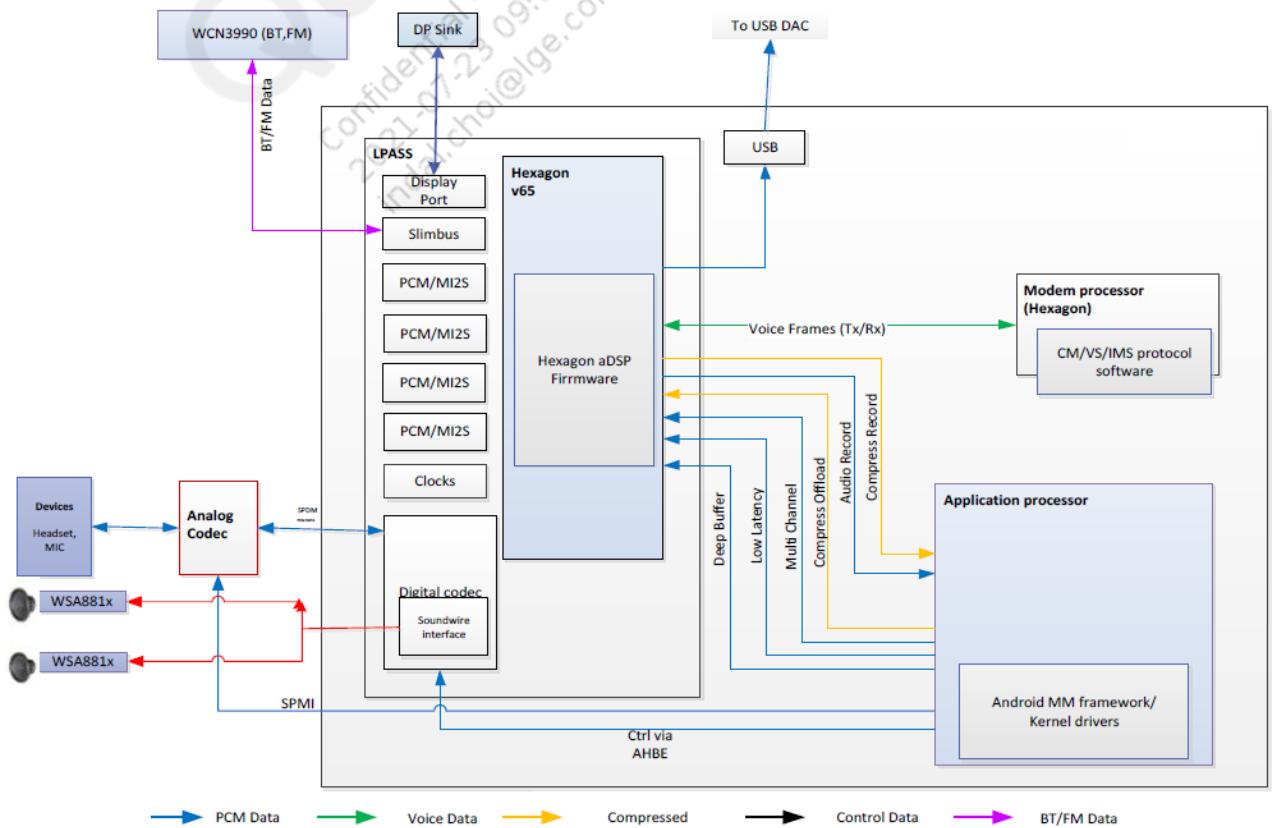
- **Frame buffers (struct drm\_framebuffer)**  
Frame buffers are abstract memory objects that provide a source of pixels to scan out to a CRTC. Implementation depends on the memory manager used and the IOMMU capabilities
- **Planes (struct drm\_plane)**
  - A plane represents an image source that can be blended with or overlaid on top of a CRTC during the scan out process.
  - Planes are associated with a frame buffer to crop a portion of the image memory (source) and optionally scale it to a destination size. The result is then blended with or overlaid on top of a CRTC.
- **CRTC (struct drm\_crtc)**
  - CRT controller (CRTC) is not related only to CRT displays. It configures the appropriate display settings.
  - Display timings/resolution, scans out frame buffer content to one or more displays, and so on.
- **Encoder (struct drm\_encoder)**  
Takes pixels data from CRTC and converts it to the format suitable for any attached connectors.
- **Connectors (struct drm\_connector)**  
Represents display interface (HDMI, DisplayPort, DSI, and VGA), transmits signal to display, detects display, exposes mode, and so on.
- **Bridge**  
Associated with an encoder, participates in mode set, device power management, connection detection, and so on.

## 6.2 Audio

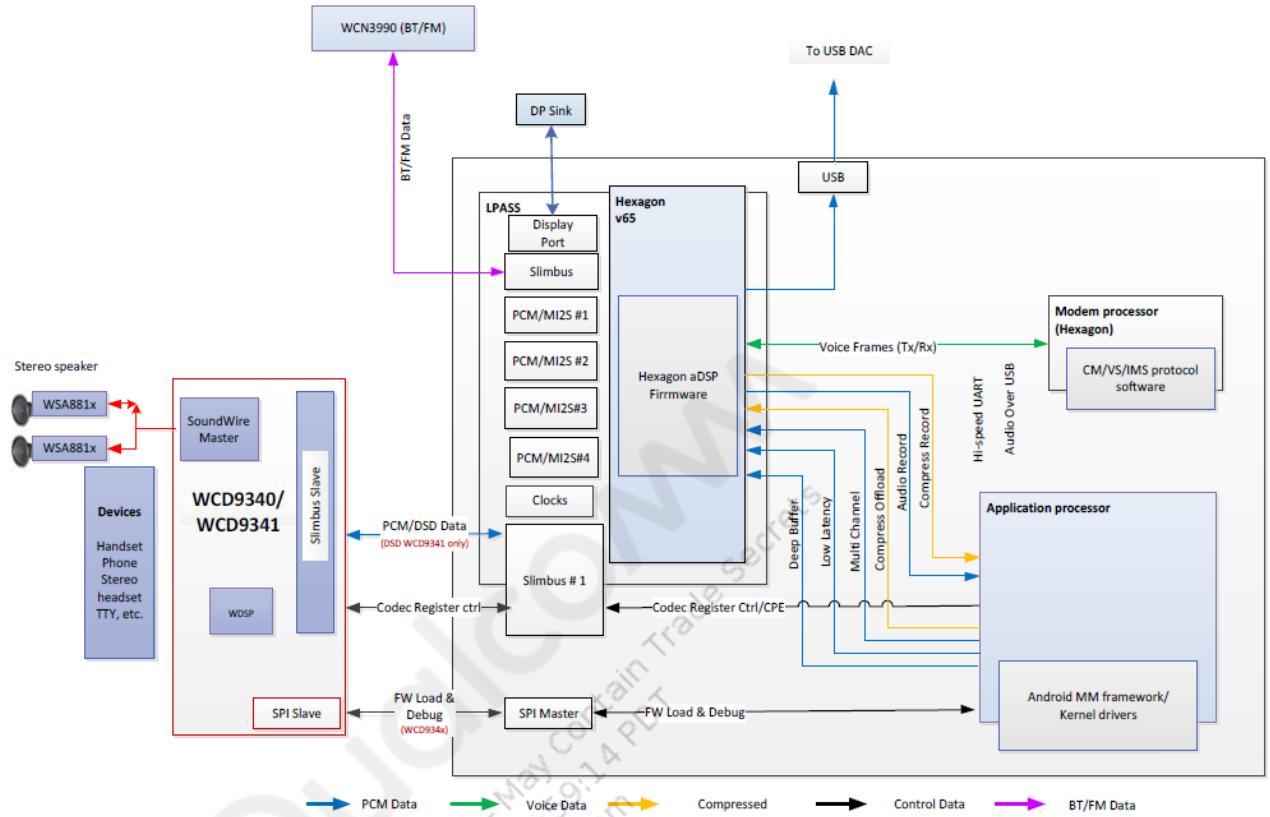
**Table 6-1 Features tested with APT levels**

Feature	Purpose	APT level for testing
■ Source Tracking ■ Sound Focus ■ Audio Zoom	Audio recording	HAL and PulseAudio
Single mic ECNS	Audio recording	HAL and PulseAudio
Dual mic ECNS	Audio recording	HAL and PulseAudio
Three mic ECNS	Audio recording	HAL and PulseAudio
Quad mic ECNS	Audio recording	HAL and PulseAudio
Voice UI single mic ECNS	Voice activation	HAL and PulseAudio
Voice UI dual mic ECNS	Voice activation	HAL and PulseAudio
Voice UI three mic ECNS	Voice activation	HAL and PulseAudio
Voice UI quad mic ECNS	Voice activation	HAL and PulseAudio
Barge in	Voice activation	HAL and PulseAudio

The figures show the audio system workflow with internal and external codec.



**Figure 6-2 Audio system overview (with Internal codec)**



**Figure 6-3 Audio system overview (with external codec WCD9341)**

#### Separate audio Line-in and Line-out through WCD9370

Jack detection for the following ports can be supported by pulse audio provided they are supported by the hardware.

- Line in/Line out
- Headset
- Headphone
- HDMI
- SPDIF

#### Jack detection

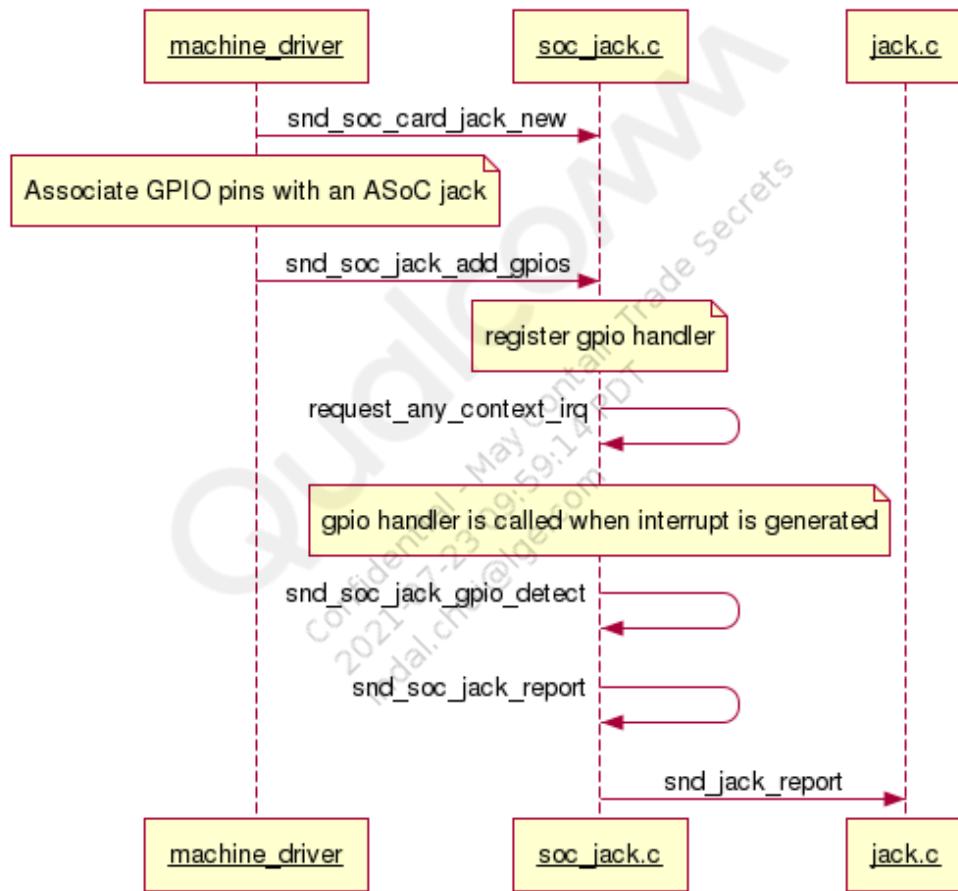
Jack detection is based on the jack type supplied by the module and availability of the port to clients. Since this is a GPIO-based jack detection, only detection or removal of event is triggered, and the jack type cannot be determined.

If the jack is inserted, the GPIO value will be 0 and if the jack is removed the GPIO value will be 1.

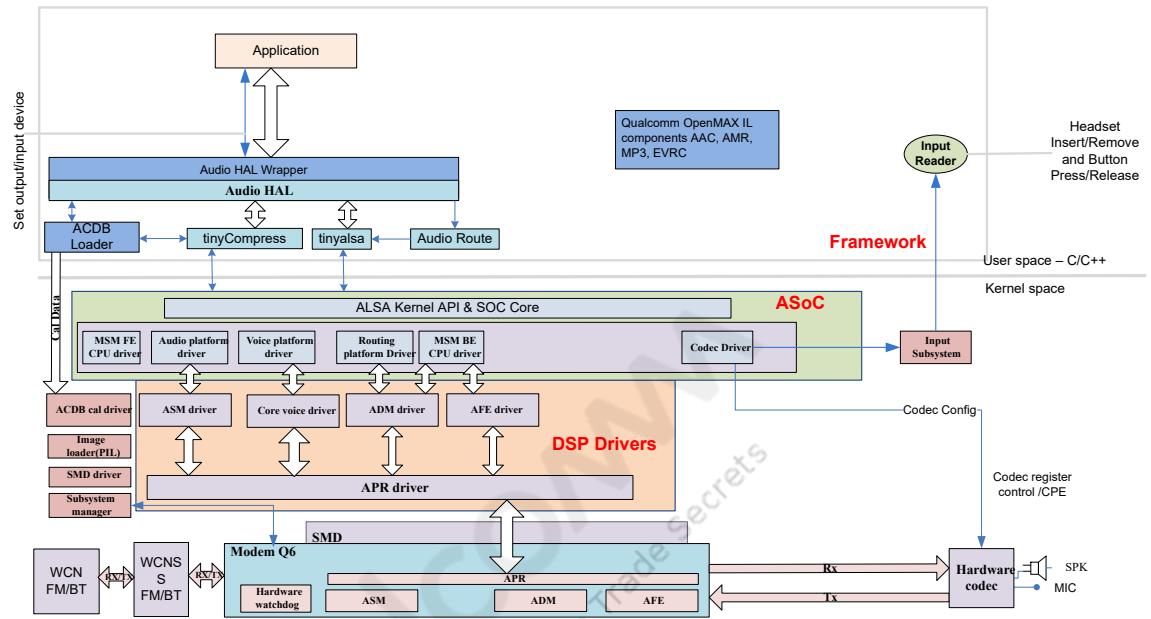
The implementation of jack detection is based on detection using the input interface (Line-in/Line-out) event-based devices. The following are the main tasks:

- Scans all the input files on the /dev/input directory.
- Opens the file that are for WCD jack/button detection.
- Reports any plug-in or plug-out to module.

## JACK detection with ALSA framework



**Figure 6-4 Jack detection call flow**



**Figure 6-5 Audio software architecture**

### APSS audio software architecture

- TinyALSA and lib/audioroute
  - TinyALSA is a small library to interface with advanced Linux sound architecture (ALSA) in the Linux kernel. It provides the basic PCM and mixer API used for an audio HAL implementation.
  - libaudioroute abstracts the mixer controls from ALSA kernel drivers using mixer\_paths.xml. It then provides APIs to enable and disable routes and paths in the aDSP or WCD (resulting in device and stream management) or sets mixer controls independently.
  - Mixer controls are categorized into:
    - Machine-specific
    - Platform-specific
    - DSP routing-specific
    - Codec-specific
- Audio HAL
  - Talks to the hardware drivers directly
  - Opens and closes drivers for rendering or capturing
  - Enables and disables the input and output devices and DSP routing using libaudioroute
  - Handles volume control in the aDSP
  - Interacts with the ACDB loader to pass the ACDB ID of a device for which the calibration data has to be sent

- Mixer control
  - Acts as a control variable exposed from ALSA kernel drivers to the user space
  - User space can pass specific parameters to the mixer control to exercise particular functions in the kernel
- ACDB and ACDB loader
  - ACDB files contain calibration data for devices that are sent to the ACDB driver in the kernel by the ACDB loader.
  - Calibration data for the WCD codec is also stored in the ACDB files for ANC and AANC, MBHC, and Snapdragon Voice Activation features.
- ASoC driver – Comprised of the machine driver, FE and BE drivers, routing driver, platform drivers, codec driver, and WDSP driver (for Snapdragon Voice Activation)
- SLIMbus driver – Communicates between the aDSP and WCD codecs
- ACDB kernel driver – Allocates memory for storing the calibration data for a device and sends it to the aDSP
- aDSP drivers – Comprised of the ASM driver, ADM, AFE, voice driver and so on.

## Audio Playback/Recording architecture

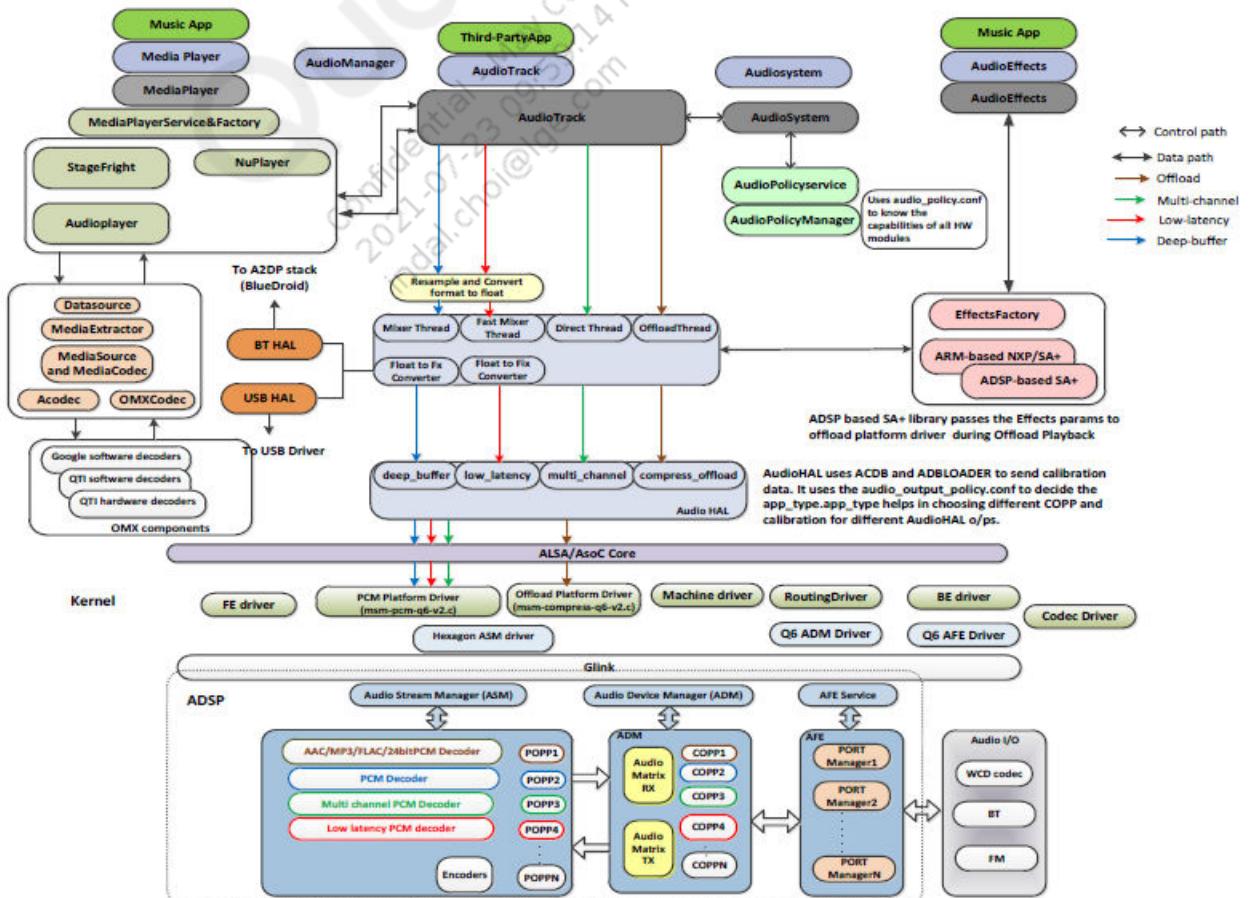
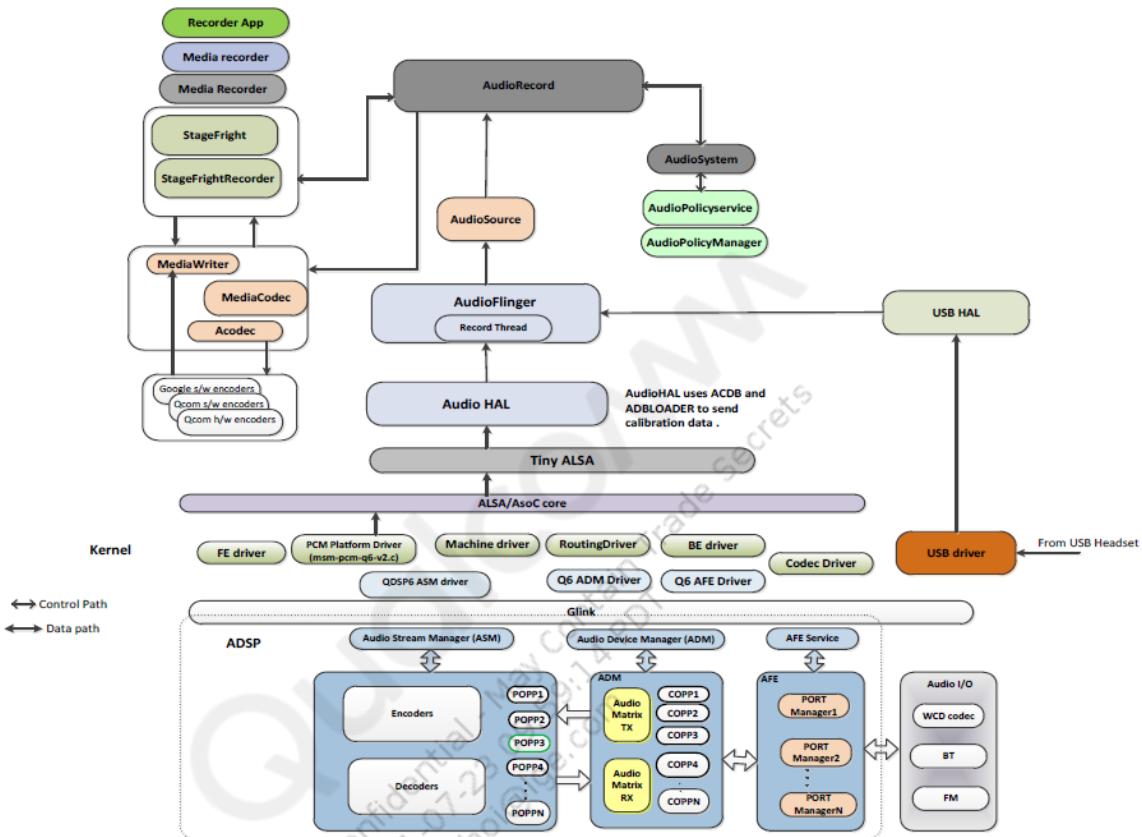


Figure 6-6 Audio playback

One-to-one mapping between POPP and COPP instance is not needed. More than one POPP output can go through one COPP. The output from more than one COPP can go to a single AFE port. Output of one COPP to multiple AFE ports is not supported.

- Offload playback
  - A large-sized buffer is sent to the aDSP and the APSS goes to sleep. The aDSP decodes, applies post processing effects, and outputs the PCM data to the physical sound device. Before the aDSP decoder input runs out-of-data, it interrupts the APSS to wake up and send the next set of buffers
  - Supported formats – MP3, AAC, 24-bit PCM, 16-bit PCM, FLAC
  - Sampling rates in kHz – 8, 11.025, 16, 22.05, 32, 44.1, 48, 64, 88.2, 96, 176.4, 192
  - Flags to be set in AudioTrack – AUDIO\_OUTPUT\_FLAG\_DIRECT | AUDIO\_OUTPUT\_FLAG\_COMPRESS\_OFFLOAD | AUDIO\_OUTPUT\_FLAG\_NON\_BLOCKING
  - Supported channels – 1, 2, 2.1, 4, 5, 5.1, 6, 7.1
- Deep buffer playback
  - The PCM data is sent to the aDSP, postprocessed, and rendered to an output sound device. The audio effects are also applied in the ARM or aDSP
  - Use cases – Ringtone, audio/video playback, audio streaming, YouTube streaming, and so on
  - Supported format – PCM
  - Sampling rates – 44.1 kHz and 48 kHz
  - Flag to be set in AudioTrack – AUDIO\_OUTPUT\_FLAG\_PRIMARY
  - Supported channel – Stereo
- Low latency
  - Playback mode is similar to Deep Buffer mode. It uses a smaller buffer size and minimal to no postprocessing in the aDSP so that the PCM stream is rendered to the output sound device.
    - Use cases – Touchtone, gaming audio, and so on
    - Supported format – PCM
    - Sampling rates – 44.1 kHz and 48 kHz
    - Flag to be set in AudioTrack – AUDIO\_OUTPUT\_FLAG\_FAST
    - Supported channel – Stereo
- Multichannel
  - Playback mode where the PCM output of the multichannel decoder is sent to the aDSP, postprocessed, and rendered at the output device
    - Use cases – AAC 5.1 channel
    - Supported format – PCM
    - Sampling rates – 44.1 kHz and 48 kHz

- Flag to be set in AudioTrack – AUDIO\_OUTPUT\_FLAG\_DIRECT
- Number of channels – Six (default); changes dynamically



**Figure 6-7 Audio playback recording**

- Compress mode – A recording mode where encoded packets are received by the APSS directly from the aDSP; it supports the AMR WB format only.
- Non-tunnel mode – A recording mode where the PCM data from the mic is preprocessed in DSP and received by the APSS, which then encodes the PCM to the required encoding format by using the DSP-based or software-based encoder.

Examples include camcorder recording and in-call recording

- Multichannel mode – Used for capturing more than two channels of the PCM stream and encoding them into a multichannel codec format like Dolby AC-3.

Examples include surround sound camcorder recording, 4 to 6 channel up sampling for 5.1 channel encoding

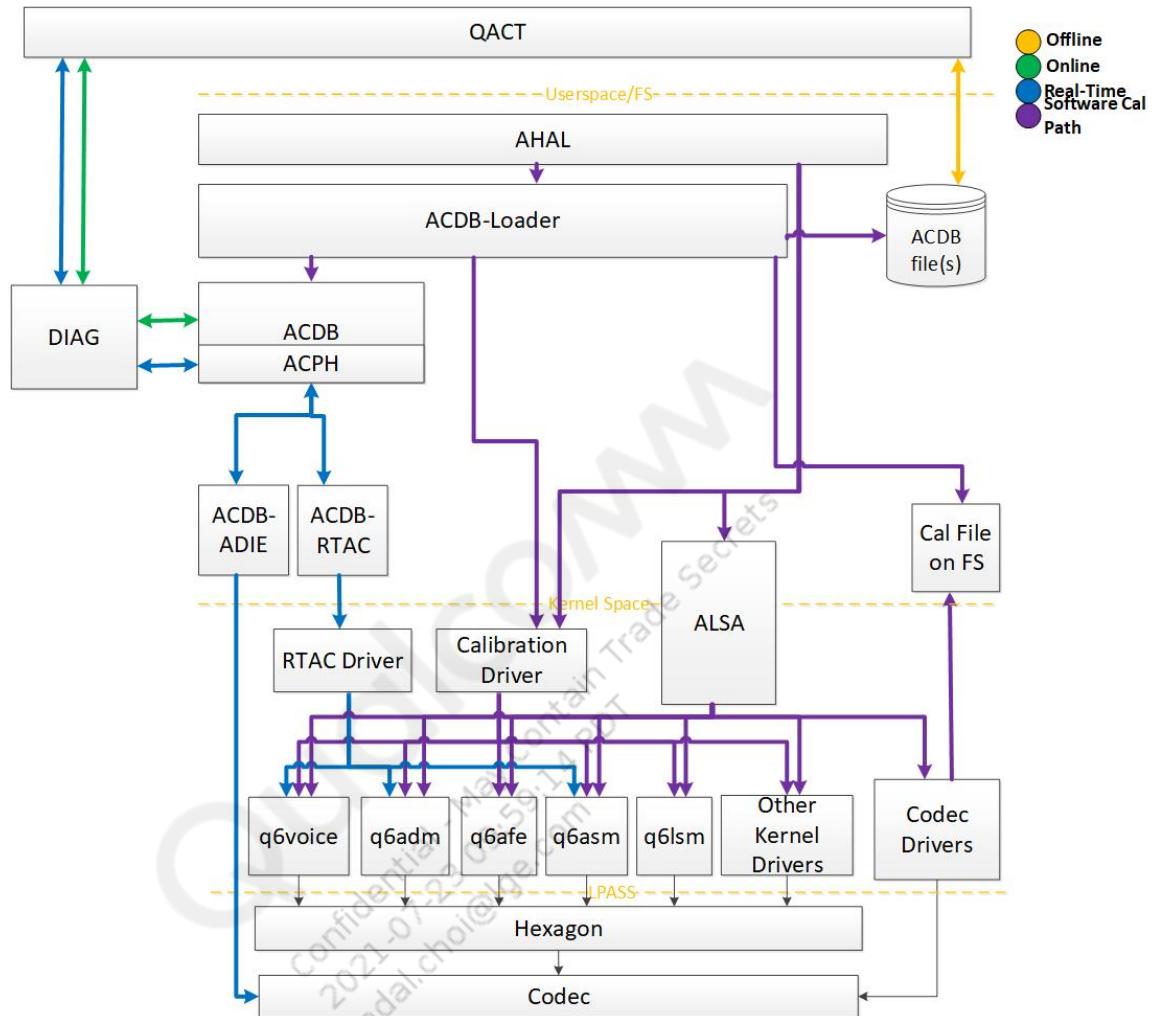
## 6.2.1 Audio features

This section describes audio features supported on the QCS610/QCS410 platform, as well as various software and hardware components designed and developed to support the following features:

- Audio calibration
  - The audio calibration database (ACDB) is a static database on the application processor. It contains all the tuning and calibration parameters for the LPASS and WCD analog codec.
  - Calibration data for various audio devices, for example, handset and speaker, are organized in an \*.acdb file format that is edited usinga PC-based tool called QACT™ Platform and placed on the phone file system in the /etc folder.

Based on device categories, ACDB data files are split into the following files:

- Global\_cal.acdb
- General\_cal.acdb
- Speaker\_cal.acdb
- Handset\_cal.acdb
- Headset\_cal.acdb
- Bluetooth\_cal.acdb
- adsp\_avs\_config.acdb
- During device switch, the audio route module in the audio HAL queries the ACDB database with a given device ID and pushes the device calibration data to kernel physical memory.

**Figure 6-8 Device and stream management**

- Split Bluetooth A2DP – Enables audio offload for A2DP.
  - Application processor – Bluetooth host communicates with the Bluetooth SoC to start and stop streaming services. Sets up the data path between the aDSP and Bluetooth SoC for transmitting encoded frames over SLIMbus.
  - LPASS aDSP converts the audio samples into SBC encoded frames and sends encoded data over a SLIMbus interface to WCN3990.
  - WCN3990 – Bluetooth SoC implements AVDTP and L2CAP encapsulation for encoded frames and sends data over the air.

- Key advantages – Low power A2DP playback

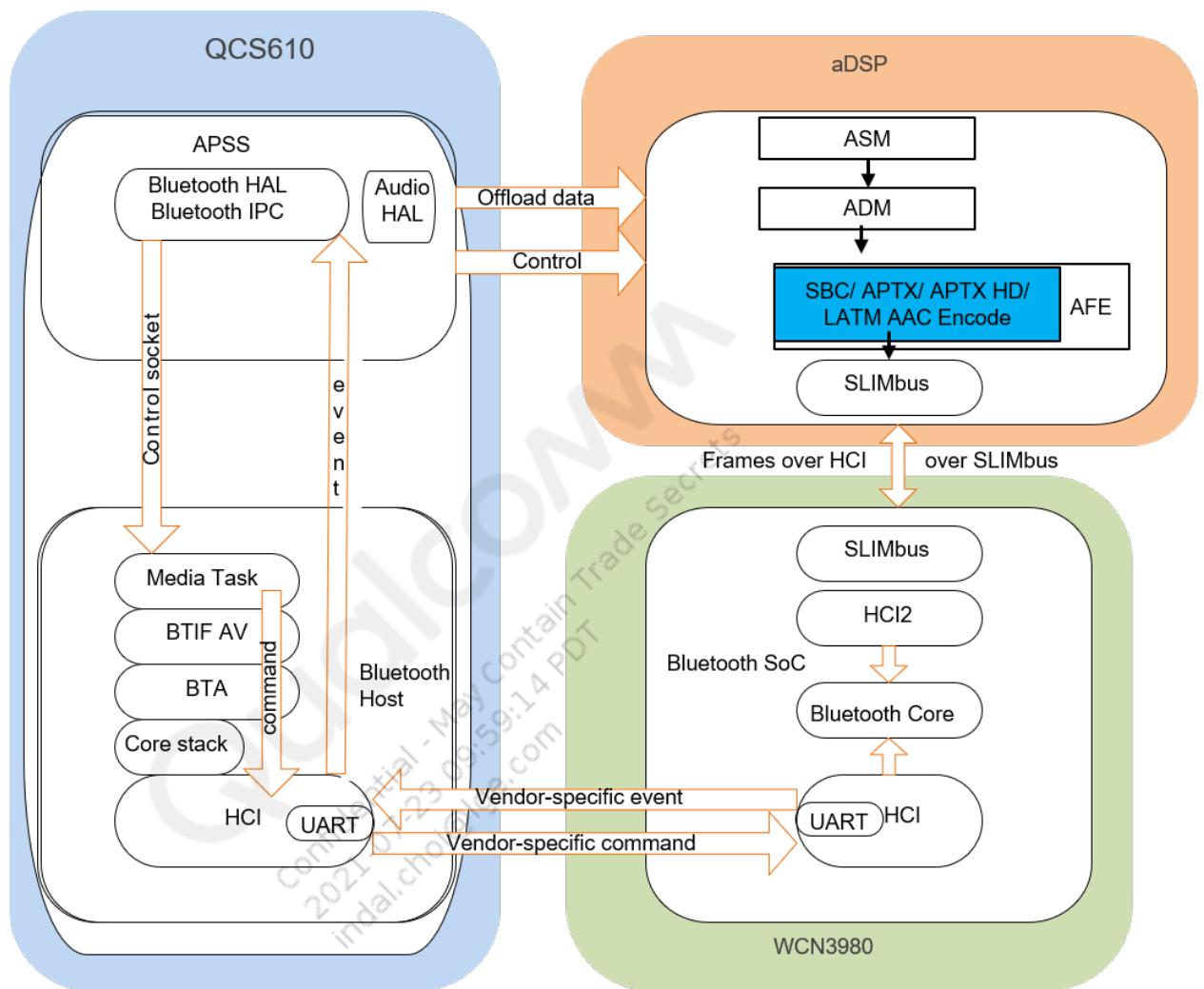
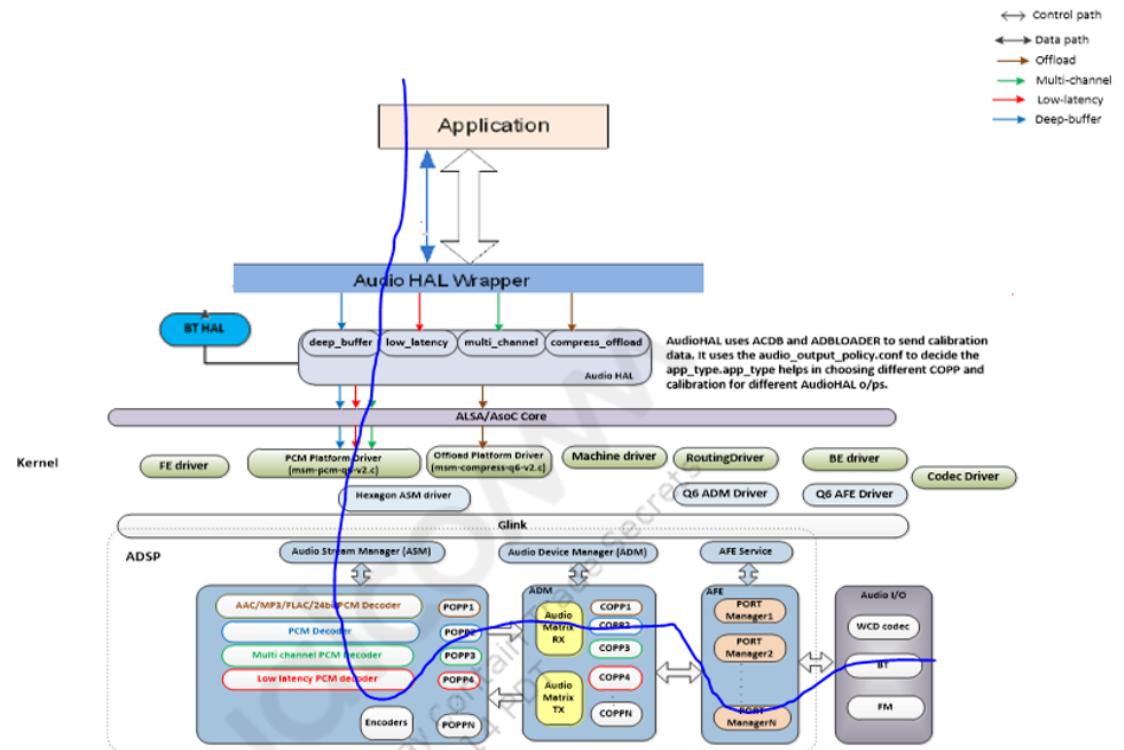


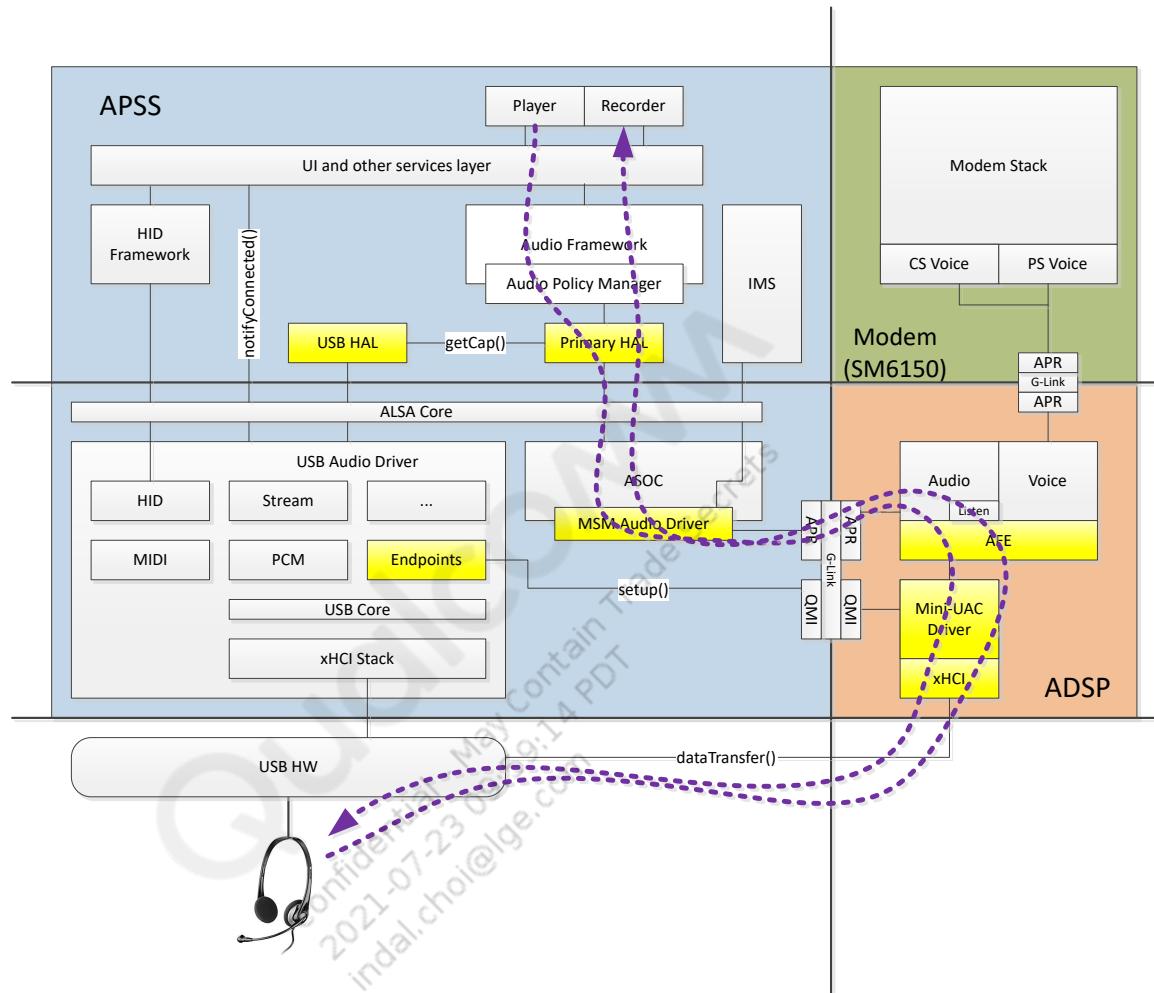
Figure 6-9 Split Bluetooth Advanced Audio Distribution Profile (A2DP)



**Figure 6-10 Split A2DP – data path**

- USB Digital Audio Tunnel mode: This mode is designed to support low-power audio playback over USB Type-C headsets.
  - Tunnels audio and voice data directly from the LPASS to the USB controller
  - APSS sets up the controller before LPASS takes over for managing the data path
- Typically, the USB playback was handled through APSS (a high-power solution).
- Key advantages:
  - Lower power
  - Lower voice call RTD

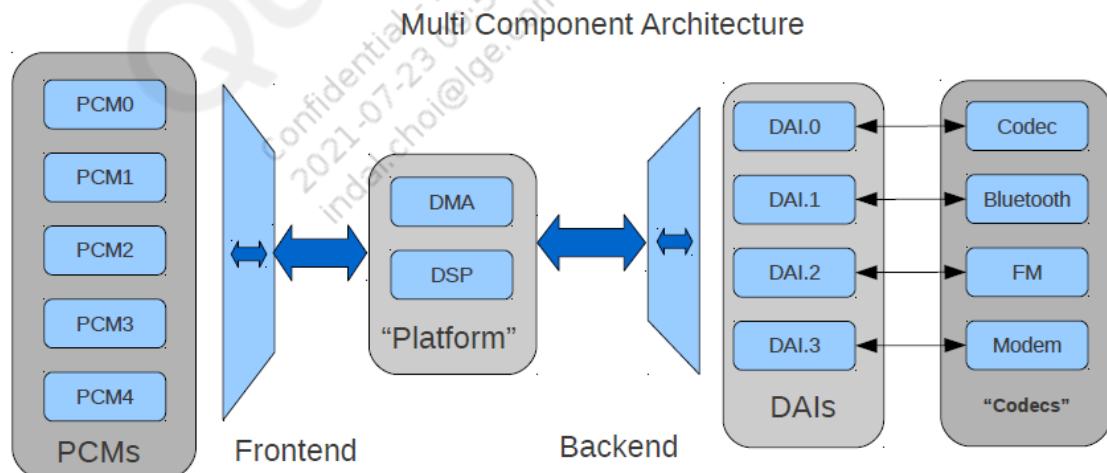
- Modes supported – USB 2.1 audio profile with hi-resolution audio (up to 24-bit/192 kHz)



**Figure 6-11 USB Digital Audio Tunnel Mode data path**

- Snapdragon voice activation – Snapdragon voice activation is used for waking up the device from sleep by speaking a predefined/user-defined keyword.
- Two modes
  - WDSP mode – Keyword is detected in the WCD and the application processor is woken up
  - aDSP mode – WCD wakes up the aDSP by determining the sound pressure level (SPL) of the surroundings against a predefined threshold. The keyword detection algorithm in the aDSP determines if a keyword was spoken.
- Power consumption is < 0.5 mA for Snapdragon voice activation due to WDSP mode.
- Debug tools for audio
  - A QACT PC-based tool used for editing audio/voice calibration data
  - Opens the ACDB data file from the build file system (`vendor\qcom\proprietary\mm-audio\audcal\family-b\acdbdata\<Chipset>`) or the device file system (`/etc`)

- Performs offline, online, and real-time calibration
- QXDM Professional is used to collect aDSP logs for voice and audio use cases
- Logs collected through the QXDM Professional are stored in .isf format
- QCAT helps in extracting PCM data vocoder packets from the .isf file and analyzes the payload of each packet.
- Volume controls – ALSA-compliant audio architecture in kernel space, codec driver, and so on.
  - ASoC is the lowest layer within the ALSA stack. It has a component-based architecture. The components are as follows:
    - Codec driver
    - Platform driver
    - Machine driver
    - CPU driver
  - ASoC core marshals all ALSA PCM calls to all components
  - Supports front-end and back-end DAIs
  - Supports complex routing between front-end and back-end DAIs
  - Dynamic audio power management (DAPM) module in ASoC minimizes power consumption.



**Figure 6-12 Audio multicomponent architecture**

Other audio features:

- 44.1 kHz playback support
- Native 44.1 kHz playback from MSM
  - No resampling in the application processor, DSP, or codec
  - Audio is played with a native 44.1 kHz/48 kHz/88.2 kHz/96 kHz/176.4 kHz/192 kHz/352.8 kHz sample rate

- Compressed and PCM offload path used for native playback support
  - Simultaneous 44.1 kHz and 48 kHz playback support DSP AFE down sample 48 kHz
  - No mixing of voice with 44.1 kHz music playback in the codec, no SRC in the DAC path
- Internal codec supports true native at only 44.1 kHz but does not support multiples of 44.1 kHz like 88.2 kHz/ 176.4 kHz /352.8 kHz
- WCD9335/WCD9340/WCD9341 supports multiples of 44.1kHz, that is 88.2 kHz/176.4 kHz/352.8 kHz
- When native playback is not active, media format converter (MFC) of the non-native paths becomes a pass-through
- During the concurrent scenario, 44.1 kHz stream is rendered at true native and alert tones and notifications are resampled to 44.1 kHz
- DSD playback – Philips and Sony developed the following DSD format:
  - DSD64 –  $44.1 \text{ kHz} \times 64 = 2822.4 \text{ kHz} = 2.8224 \text{ MHz}$
  - DSD128 –  $44.1 \text{ kHz} \times 128 = 5644.8 \text{ kHz} = 5.6448 \text{ MHz}$
- To send DSD content over SLIMbus, the content is packetized in the following PCM format:
  - DSD64 is converted to 64 DoP =  $44.1 \text{ kHz} \times 64 / 16 = 176.4 \text{ kHz}$
  - DSD128 is converted to 128 DoP =  $44.1 \text{ kHz} \times 128 / 16 = 352.8 \text{ kHz}$
- DSF file format is from Sony contains data in the DSD format
- DSDIFF file format is from Philips contains data in DSD format or direct stream transfer (DST) format and DST format decoding is not supported
- Both DSD native and DSD non-native configurations are supported to handle concurrency scenarios and playback of DSD content over various devices
- DSD playback – Native and non-native configuration
  - DSD – Native playback
    - Supports playback of first DSD stream over HPH/lineout.
    - DSD Native play back is supported only on the WCD9341 codec.
    - DSD64/DSD128PDM data is packetized in an ASM component in the aDSP, resulting in a 64 DoP/128 DoP stream, which is routed to the AFE without any PCM processing. The WCD codec has a dedicated DoP depacketizer, which depacketizes the 64DoP/128DoP and, renders the PDM data.
    - Volume control is done in the WCDcodec
    - DSD-Native mode is not supported on the speaker path
  - DSD non-native playback – Non-native DSD path is used in the following scenarios:
    - Playback of a second DSD stream in concurrency with another
    - Playback of DSD stream over the speaker path
    - Uses the primary and mixing path of the codec depending on the device and the concurrency scenario.

- The DSD stream is decoded and PCM data is sent to the aDSP, where the PCM processing and resampling are done based on the AFE configuration. The PCM stream from the AFE is sent to the codec, where the PCM data is routed to the end device using the primary or mixing path
- DSD Non-native playback is supported in an Internal codec also
- Device and stream management
- ALSA-compliant audio architecture in kernel space, codec driver, and so on

## 6.2.2 Audio configuration

**Table 6-2 Customization guidelines**

Module	Description	Customization guideline
APM	Manages various input and output device interfaces, chooses, and defines the appropriate routing strategy based on the stream mode and method, and manages volume/mute settings per stream (as they become active or inactive)	<ul style="list-style-type: none"> <li>▪ Licensees can modify the device priority and matrix to suit their use case of their choice</li> <li>▪ To reduce development and test effort, remain as close as possible to software-tested configuration</li> </ul>
Audio HAL	The hardware abstraction layer that maps AudioFlinger calls to ASoC drivers	Add changes related to new device and new audio routes
XML	Contains audio device route and device mixer controls	Add new device routes and devices
Codec driver	Configures hardware codec registers	Do not modify
Machine driver	Specific to the machine implementation or customer board	External power AMP widgets and voltage block-related changes are made to the machine driver as they are customer-specific
DeviceTree	Codec/SLIMbus-specific platform data configurations	Codec routing map changes (mic bias internal/external, and so on), voltage blocks-related changes, and codec/SLIMbus platform data configurations are customer board-specific

## 6.2.3 PulseAudio

PulseAudio is an audio server. The role of an audio server in any platform is to do the following:

- Audio routing
  - In PulseAudio, audio routing is controlled from `/etc/pulse/system.pa` file.
  - The user can define the default sink and source for any stream. The routing framework enables an application to explicitly choose the destination *sink/source*.

- Audio processing
  - The sink and source devices come with pre-configured audio parameters such as sample specification, channel mapping, and so on.
  - For an incoming stream to be played on sink, the source and sink specifications should match. In case of a mismatch, PulseAudio bridges the gap by converting audio to the desired codec specification.
- Stream management handles the creation and deletion of streams. It also manages the handling of multiple streams at the same time.

### PulseAudio support for Sound Trigger

- PulseAudio is intended to run as a middleware between the applications and HAL or the hardware device.
- PulseAudio's Dbus interface are used to configure SVA and to detect keywords.
- PulseAudio requires clients to make peer-to-peer connection and not route through system/server dbus daemon.
- PulseAudio and other lower level details are abstracted from the test application and voice UI applications. For applications, the interface will be similar to the existing one qsthw\_api without any need to be middleware specific.
- Qsthw PA module takes module ID as argument and does STHAL load or unload as part of init and done. A global interface is exposed to call methods exposed.

## 6.2.4 Source tracking trio

The source tracking trio comprises the following features:

- Source tracking: Beamforming techniques to localize sound coming from different directions.
- Sound focus: Beamforming and masking techniques to preserve or suppress sound coming from specific directions.
- Audio zoom: Zoom-in/Zoom-out of audio speech. Strength of incoming audio from the 'desired direction' can be varied.

### Source tracking

The Qualcomm Audio and Voice Communications suite v5.5 broadside topology and the Qualcomm Audio and Voice Communications Pro v2 topology solutions employ beamforming techniques to localize the sound coming from different directions.

- Qualcomm Audio and Voice Communications suite can scan the 360-degree plane parallel to the ground and identify sounds coming from any direction.
- At any given time, Qualcomm Audio and Voice Communications suite can return the direction of arrival of the incoming sound so that the user can continuously track the audio sources in the environment.

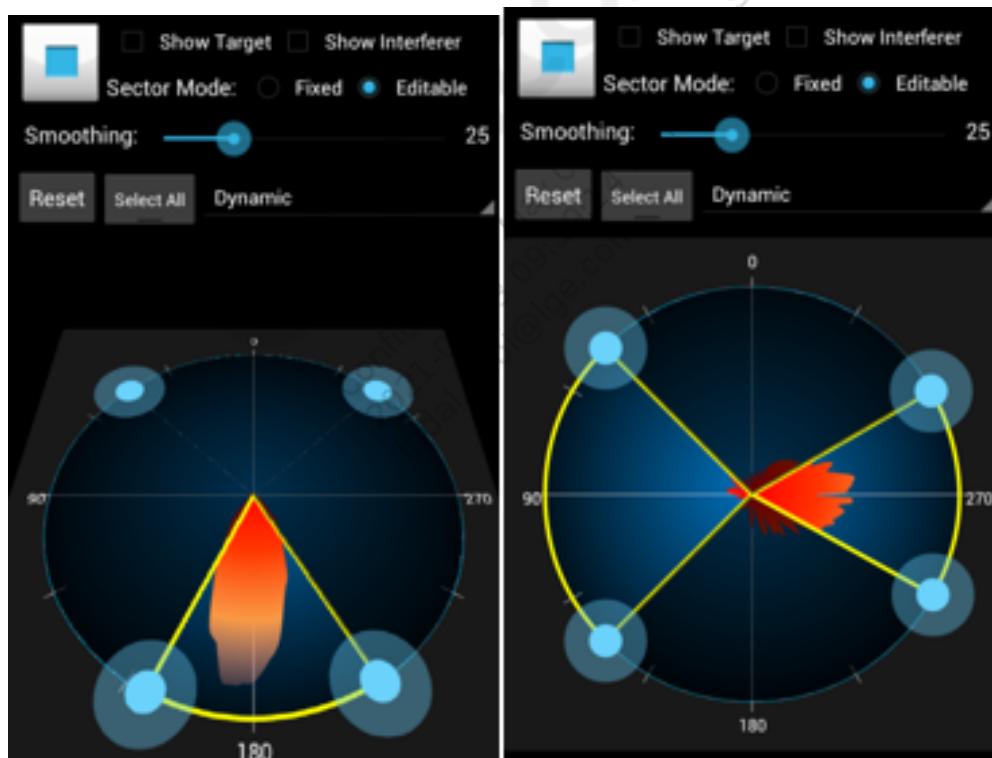
This is called source tracking and it is a benefit of Qualcomm Audio and Voice Communications suite beamforming and source localization capability.

Source tracking is different in Qualcomm Audio and Voice Communications suite v5.5 (two-mic) and Qualcomm Audio and Voice Communications suite Pro v2 (three 3-mic or four-mic) solutions.

- With two-mic solutions, source localization can be done only in the 180-degree angular range. In this case, the source localization plot is symmetric around the microphone array axis (0 to 180 degree axis) as the beamformers cannot discriminate between sounds coming in the 0 to 180 degree directions vs. 180 to 360-degree directions.
- With three-mic or four -mic solutions, source localization can be done in the full 0 to 360-degree angular range and there are no limitations to the source tracking functionality.

The user can receive the following additional information about the environment from Qualcomm Audio and Voice Communications suite. HLOS receives the information from DSP and provides it to the user space to be displayed on the UI.

- Voice activity detection (VAD)
- Polar activity, which is the relative strength of various audio sources in the environment
- Direction of arrival (DOA) of the dominant talker and the dominant interferer in the environment



**Figure 6-13 Qualcomm Noise and Echo Cancellation Pro v2 source tracking UI display - sample**

The figure shows a sample source tracking UI display for Qualcomm Audio and Voice Communications suite Pro V2 audio recording use case. The polar activity (directionality) of the talkers in the environment is shown by the orange flame display. The sector angles are shown by the blue dots around the 360-degree plane.

### Sound focus

The Qualcomm Audio and Voice Communications suite employs fixed beamforming and masking techniques to preserve or suppress sound coming from specific directions. The 360 degree plane

parallel to the ground is divided into 4 angular sectors (the number of sectors can be extended) and the end user can configure the angular definition of these 4 sectors and choose which sectors shall preserve the incoming sound and which sectors shall suppress the incoming sound. The sound focus feature gives the user the capability to configure the directionality of the Qualcomm Audio and Voice Communications suite beamformer so that they can record sound from any direction of their preference.

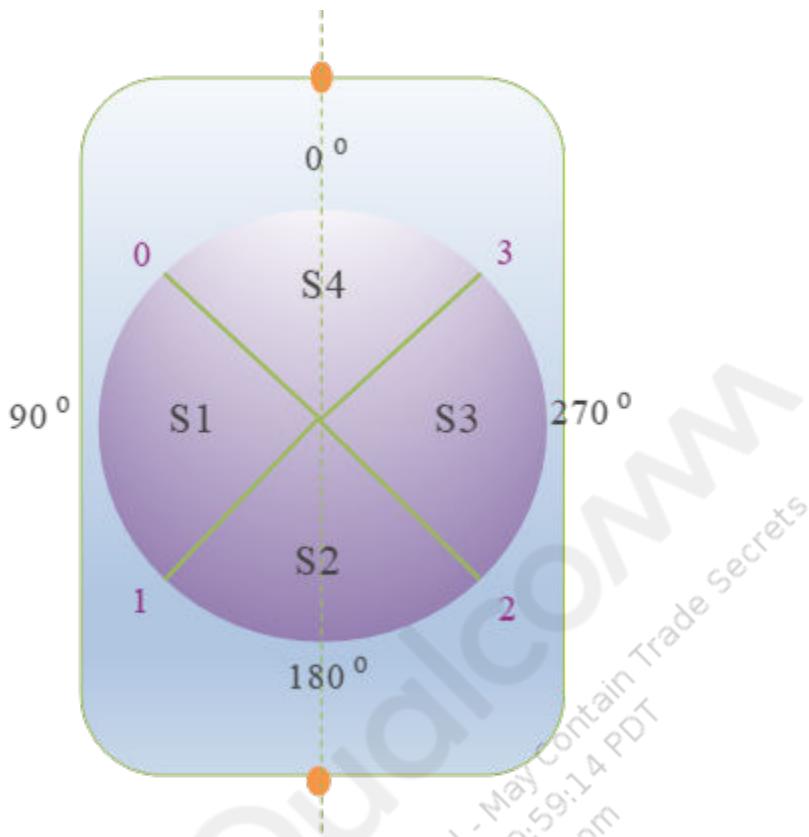
The behavior of the sound focus feature is different between Qualcomm Audio and Voice Communications suite V5.5 (2-mic) and Qualcomm Audio and Voice Communications suite Pro V2 (3-mic/4-mic) solutions. With 2-mic solutions, directionality discrimination can be done only in the 180-degree angular range. In this case, the 4 angular sectors cannot be independently configured. The sectors should be symmetric around the microphone array axis (typically, 0-180 deg axis). If the user tries to configure arbitrary recording sectors for the 2-mic solution, then Qualcomm Audio and Voice Communications suite V5 will automatically correct the sector definition to reflect the symmetry around 0-180 deg axis.

With 3-mic/4-mic solutions, directionality discrimination can be done in the full 0- 360-degree angular range and there are no limitations to the sound focus functionality.

With the Sound Focus feature, the Qualcomm Audio and Voice Communications suite can take additional information about the user preference and then set it to the DSP for dynamically modifying the behavior. HLOS can take the information from the user through the UI and then set it to the DSP for customizing the Qualcomm Audio and Voice Communications suite.

Information available for Sound Focus setting:

- Sector definition: Starting angles of the sectors covering the 360-degree plane parallel to the ground.
- Sector enablement: Enable/disable information for each of the sectors.



**Figure 6-14 Sound Focus sector definition for Qualcomm Audio and Voice Communications suite**

The convention for angle measurement is as shown, in the anti-clockwise fashion. The 4 sectors spanning the 360-degree plane are shown as S1, S2, S3 and S4. The two microphones on the device are shown by orange dots.

### Audio Zoom

The audio zoom feature is built on the sound focus feature and it is applicable only for camcorder recording and video call applications. During camera + audio usecases, when the camera is zoomed in or zoomed out, the audio can also be zoomed in or zoomed out correspondingly so that the user gets an integrated audio + video experience. The audio zoom processing is done as follows.

- Camera zoom in:
  - When the camera zooms in, the field of view becomes narrower.
  - Audio recording sector is made narrower to correspond to the camera field of view.
  - Audio coming from the camera subject is recorded and other audio sources are suppressed.
  - Gain of the recorded audio is increased proportionally as the camera zooms in.
- Camera zoom out:
  - When the camera zooms out, the field of view becomes wider.
  - Audio recording sector is made wider to correspond to the camera field of view.

- Audio coming from a wide range of angles is recorded.
- Gain of the recorded audio is decreased proportionally as the camera zooms out.

The audio zoom feature thus ties the sound focus sector definition and the audio gain to the camera zoom so that both audio and video get proportionally zoomed with the camera operation.

With the audio zoom feature, the UI application needs to take the camera zoom step information and then map it to the corresponding audio zoom step. The audio zoom step is defined by the appropriate sector angles and the audio gain for the zoom step.

Information available for audio zoom setting:

- Sector definition: Starting angles of the sectors covering the 360-degree plane parallel to the ground.
- Sector enablement: Enable/disable information for each of the sectors.
- Audio Gain: Gain value of the Qualcomm Audio and Voice Communications suite processed signal.

For more information, see *Fluence Pro V2 Overview* (80-NK880-7).

## 6.3 Hexagon Compute DSP

The QCS610/QCS410 chipset has a dedicated compute DSP (cDSP). The OEMs can offload the computation intensive operations to cDSP to boost the performance. cDSP ensures improved image processing, computer vision, camera streaming, and ML.

The following are the primary use cases:

- Image enhancement for camera, still video
- Computer vision and extended reality
- Video
- Camera streaming
- ML

**Table 6-3 cDSP hardware specification**

Parameter	Hexagon V66K	Hexagon V66A HVX
Threads/Core	2 threads	+ 2 128-byte vector threads
Processor Clock	864 MHz Nominal (1 GHz Turbo)	AA SKU – 800 MHz (Turbo) AB SKU – 1.1 GHz (Turbo)
Arithmetic	Fixed and floating point	Fixed point
Caches	<ul style="list-style-type: none"> <li>▪ L1 Instruction: 16 K</li> <li>▪ L1 Data: 16 K</li> <li>▪ L2: 512 K</li> </ul>	<ul style="list-style-type: none"> <li>▪ Same L2 as Core</li> <li>▪ 256 KB VTCM</li> </ul>
Performance	Performance	Performance (Dual-HVX)
Peak OPS	<ul style="list-style-type: none"> <li>▪ 8 MMAC/MHz (16-bit fixed point)</li> </ul>	<ul style="list-style-type: none"> <li>▪ + 64 MMAC/MHz (16-bit fixed point)</li> </ul>
Dhrystone	<ul style="list-style-type: none"> <li>▪ 16 MMAC/MHz (8-bit fixed point)</li> </ul>	<ul style="list-style-type: none"> <li>▪ + 512 MMAC/MHz (8-bit fixed point)</li> </ul>
Coremark	<ul style="list-style-type: none"> <li>▪ 4 MFLOP (SP)/MHz</li> </ul>	<ul style="list-style-type: none"> <li>▪ + 256 Mops/MHz (16-bit fixed point)</li> </ul>

**Table 6-3 cDSP hardware specification (cont.)**

Parameter	Hexagon V66K	Hexagon V66A HVX
	<ul style="list-style-type: none"> <li>■ 7.2 DMIPS/MHz</li> <li>■ 14.4 Coremark/MHz</li> </ul>	
Architecture	Simultaneous multi-threading (SMT)	–
Instruction/System Enhancements	Multimedia enhanced Co-processor capability Automatic power collapse (APC)	HVX Dual-clusters Scatter/Gather

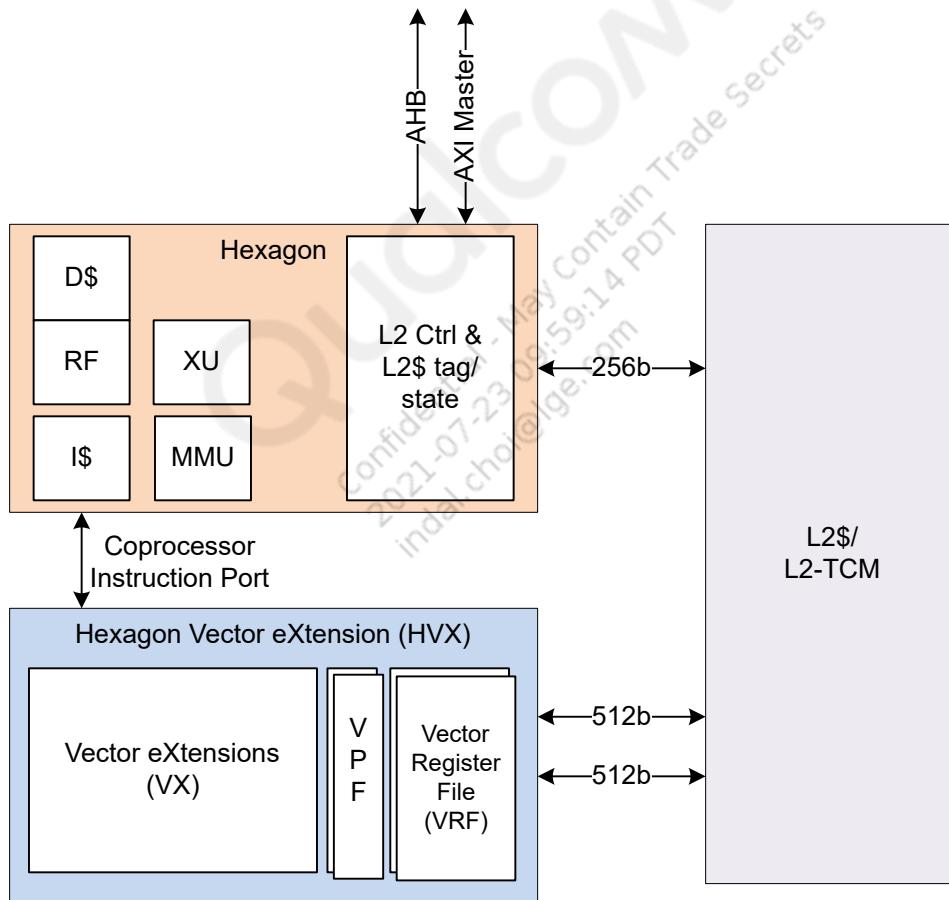
**Table 6-4 cDSP features**

Feature	Description
Vector TCM (VTCM)	Local TCM (256 KB) added in Hexagon V66
Scatter-Gather support	New HVX instructions and usage examples for scatter-gather support Provides support for non-linear vectors
HVX contexts and concurrency	RTOS management of HVX contexts and concurrency: No need to reserve or lock HVX contexts RTOS can pre-empt and context-save HVX contexts 128-byte mode is the default and algorithms can start calling HVX instructions
Dedicated high-speed bus for data transfer from the DDR to the cDSP	cDSP is directly connected to the DDR
IO coherency	Low RPC latency with input/output coherence support
Qualcomm® Computer Vision SDK software development kit with HVX acceleration	Qualcomm Computer Vision SDK APIs are accelerated using HVX
Unsigned protection domain	Support for signature-free dynamic shared objects
Secure protection domain	Facilitates secure communication
Feature	Description
Migrating to 128-byte mode1	Migrating to 128-byte mode is essential because 64-byte/512-bit mode is no longer supported
Powering ON HVX is not required	Hexagon Access API to power on HVX is no longer required By default, HVX power is ON unless cDSP is power collapsed or is idle Current Hexagon Access APIs do support HVX power ON for backward compatibility, but power ON is not required Issuing a request either way does not result in any issues
SDK 3.4	Provides examples and a development environment SDK 3.4 is used for development

**Table 6-4 cDSP features (cont.)**

Feature	Description
Camera streaming – four tap points	Additional four tap points in camera pipeline to leverage HVX camera streaming functionality for Preview and Video Stream use cases Data can be inserted and output at multiple points in an image front-end engine (IFE) pipeline
Dedicated bus for camera streaming on cDSP	Enables preprocessing on camera streams before the image signal processor (ISP) Supports a dual camera use case involving two sensors and ISPs

### 6.3.1 Hexagon cDSP with HVX



The following are the features of Hexagon cDSP with HVX:

- Provides interleaved multithread VLIW DSP
- Combines best of DSP with general purpose processor ease-of-programming
- Provides excellent control code and signal processing
- Focuses on throughput and efficiency instead of peak single-thread performance
- HVX is optional component of Hexagon DSP

- Adds wide-vector (1024-bit) SIMD support
- Instruction extensions to Hexagon V66 processor architecture support vector operations on 1024-bit wide data
- Operations are implemented in a coprocessor
- Suitable for high performance imaging, compute, and ML applications

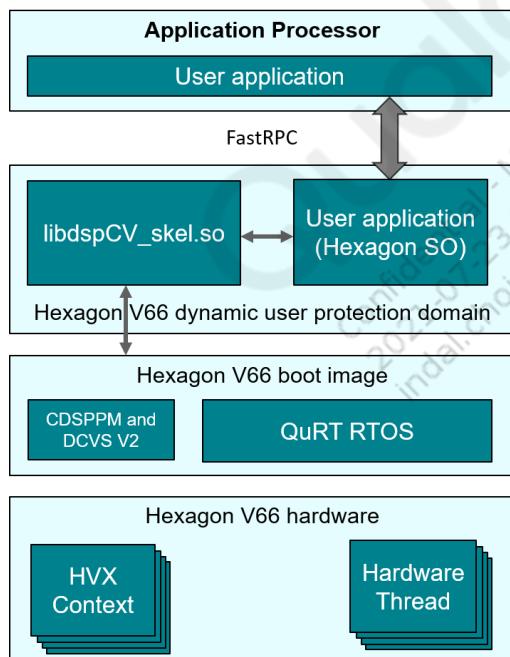
### 6.3.2 cDSP software architecture

The basic compute software architecture is maintained from previous non-HVX to HVX targets.

In the programming model, the assembly code can contain HVX instructions and C code can contain HVX intrinsics. The HVX context is automatically locked to software thread when that thread issues HVX instructions.

The HVX register set is saved and restored by QuRT™ Software during context switch. The libdspCV\_skel.so file provides multi-threading support that can be directly invoked from the DSP side. The DCVS v2 APIs are provided for clock and power management.

The HVX context must be locked to a hardware thread by RTOS.



## QuRT RTOS

The QuRT RTOS assigns HVX contexts to software threads (upon request), and software threads to hardware threads.

- Schedules software threads onto hardware threads based on thread priority
- Provides APIs for HVX control
  - Reserves a number of HVX contexts for user protection domain (held until reservation is cancelled)
  - Locks/unlocks an HVX context to a requesting software thread, with 1024-bit mode
    - All active HVX contexts must be in same mode at a given time
    - If two threads request locks for two different modes, one is blocked until the first thread unlocks
  - Queries for HVX capabilities of target

## cDSP power management and DCVS

The cDSP power management (CDSPPM) and DCVS v2 aggregate clock votes and HVX power votes and drive hardware accordingly

- Aggregate client votes for Hexagon core clock and bus clock to DDR
- Aggregate client votes to power on/off HVX units
- Monitor load in real time, and adjusts clocks accordingly

The cDSP and bus clocks are managed within cDSP by a combination of voting (CDSPPM) plus background DCVS task. The voting establishes initial clocks, and floor clocks are to be guaranteed throughout voting use case life cycle. CDSPPM sets default clock to NOM (nominal) frequency.

DCVS monitors processor loading and might raise or lower clocks to meet real-time needs with lowest possible power

The Hexagon access power APIs provide access for setting clock and bus votes along with other DCVS parameters

The DCVS v2 APIs use cDSP DCVS v2 to set cDSP clocks. The following DCVS v2 configurable parameters can be used with `Hap_power_set_dcvs_2()` function:

- DCVS enable/disable
- DCVS options to instruct DCVS algorithm to use a predefined set of thresholds and operation logic, based on a selected option
- Set sleep latency vote in microseconds
- DCVS parameters to set upper and lower DCVS thresholds, and vote for core and bus clocks using voltage corner

For DCVS v2 usage, see Hexagon SDK documentation at <Hexagon\_SDK\_Root>\<version>\docs\Hap\_power\_set\_dcvs\_2.html

For SDK example, see <Hexagon\_SDK\_Root>\<version>\examples\compute\benchmark\src\_dsp\benchmark\_imp.c

The function is `benchmark_setClocks()`.

### User application

The user application is partitioned across processors via the Hexagon SDK.

User-space application on applications processor invokes C-callable functions implemented in user application Hexagon SO file through FastRPC

### User application – Hexagon SO

The user application is partitioned between application processor and Hexagon.

- The Hexagon/HVX functions are built with toolchain in Hexagon SDK and placed on the device file system
- The Hexagon SO is loaded dynamically when invoked by user application on application processor
- Invokes Hexagon Access APIs with DCVS v2 parameters for required clock corner, DDR bandwidth, enable/disable DCVS, power on/off HVX units
- Supports C and C++ applications

### `libdspCV_skel.so`

The `libdspCV_skel.so` library is a Hexagon SDK deliverable, which wraps multi-threading callback interface. It is used by FastCV applications to implement multi-threading and wraps required APIs from Hexagon boot image. It supplies worker threads and callback interface for user application multi-threading.

### FastRPC

The FastRPC interfaces between user application and Hexagon libraries with HVX implementations.

- Enables the User mode and Kernel mode drivers to enable user application to call functions in Hexagon SO file
- Upon first RPC invocation from user application, FastRPC automatically does the following:
  - Creates a user process on cDSP that services this and subsequent calls from the caller process
  - Marshals function arguments back and forth for each RPC call
  - Tears down cDSP process when an application process dies
- Provides cache IO coherency to reduce RPC latency

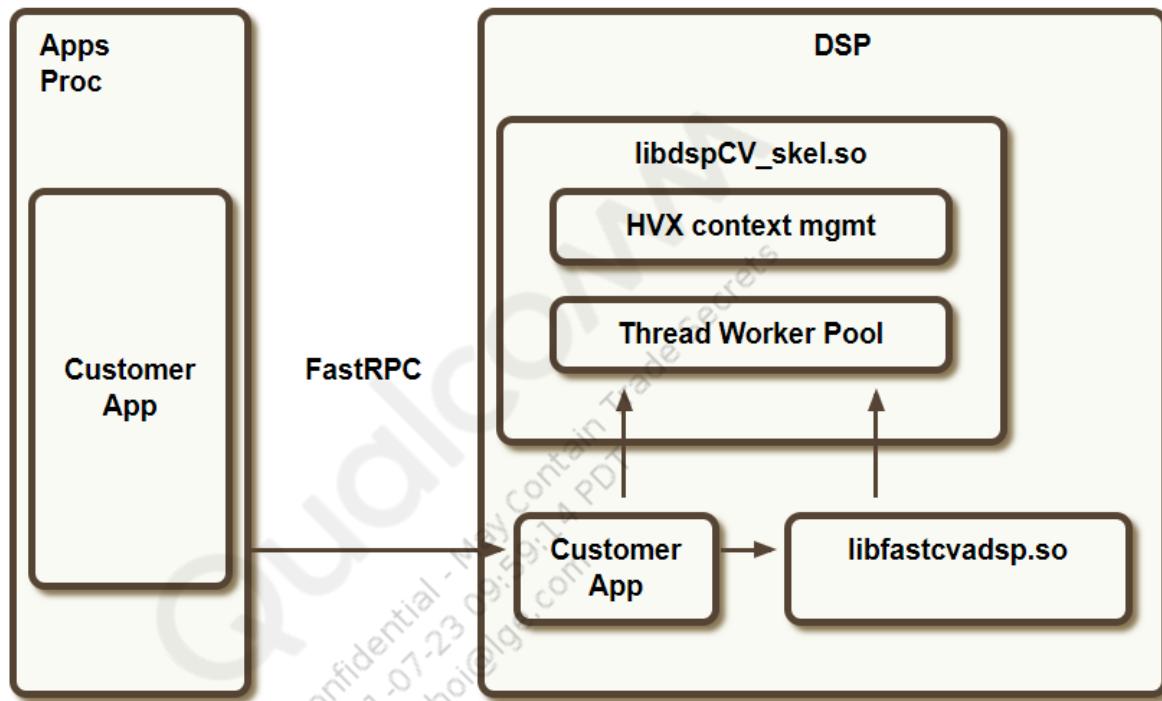
### 6.3.3 Qualcomm Computer Vision SDK

The Qualcomm Computer Vision SDK is a variant of Open Computer Vision (OpenCV) that accelerates some of the computer vision primitives in the CPU, GPU or DSP.

The Qualcomm Computer Vision SDK has API and library that enables mobile devices to run CV applications efficiently. It enables acceleration of CV applications through hardware and is analogous to OpenGL ES in rendering domain and has a clean modular library.

Customers can develop DSP-accelerated CV applications on Hexagon V5 and later products via dynamic loading and FastRPC. The dspCV utility library abstracts the DSP runtime environment for any compute application (not necessarily just CV) to effectively use DSP resources.

The Qualcomm Computer Vision SDK library contains many image processing APIs, optimized for the Hexagon DSP. The Hexagon SDK includes several examples demonstrating usage of both of these libraries in the typical usage model shown in the figure.



**Figure 6-15 dspCV Library**

The dspCV library in \${HEXAGON\_SDK\_ROOT}/lib/fastcv/dspCV aims to abstract as much of the DSP runtime environment as possible to reduce effort in offloading compute processing to the DSP. It offers APIs to perform such functions as clock/power voting, multi-threaded callbacks, concurrency checking, and HVX resource management.

Separation of the cDSP from the aDSP and other new features in relevant software frameworks have reduced the need for some of dspCV APIs.

The evolution of frameworks and recommendations for each supported target are detailed in the Hexagon/HVX Overview in the Hexagon SDK documentation. To summarize that information, it is no longer recommended to use dspCV for clock voting, or even to explicitly call `dspCV_init_with_attributes()`. The `dspCV_concurrency` APIs are no longer needed.

The dspCV can be linked and its worker pool initialized via static constructor at loading time.

## dspCV remoted APIs

The following remoted APIs (to be called from the application processor) are fully documented inline in the  `${HEXAGON_SDK_ROOT}/lib/fastcv/dspCV/inc/dspCV.idl` header file.

- `dspCV_initQ6_with_attributes()` boosts the DSP and SNOC bus clocks to values indicated by the attributes specified.
  - For information on the power/performance trade-off options available to the calling application, see the inline documentation in  `${HEXAGON_SDK_ROOT}/lib/fastCV/dspCV/inc/dspCV.idl` for details.
  - This function also instantiates and reference-counts a singleton worker thread pool per process on the DSP for subsequent use by any multi-threaded DSP functions within the established RPC process.
  - Calls to `dspCV_deinitQ6()` revokes the clock boosts and tears down the worker pool (once reference count reaches 0).
- `dspCV_getQ6_concurrency_attributes()` allows the application to perform instantaneous queries for information on the DSP concurrency level.

`dspCV_initQ6_with_attributes()` and `dspCV_deinitQ6()` are meant to be called at the beginning and end of a use case respectively. In between those calls, DSP applications can use the multi-threading callback APIs in  `${HEXAGON_SDK_ROOT}/lib/fastcv/dspCV/<flavor>/dspCV_worker.h`.

The application can submit jobs to the worker pool where each job is pair of a function callback pointer and a data pointer to pass to that callback, in a worker thread context.

## dspCV C-callable APIs from within the DSP

- `dspCV_worker.h` HVX APIs

The  `${HEXAGON_SDK_ROOT}/lib/fastCV/dspCV/inc/dspCV_worker.h` header contains APIs for utilizing the worker thread pool utility.

The key APIs are documented inline in that file.

```

// signature of callbacks to be invoked by worker threads
typedef void ( *dspCV_worker_callback_t )( void* );
// descriptor for requested callback
typedef struct
{
    dspCV_worker_callback_t fptr;      // function pointer
    void* dptr;                      // data pointer
} dspCV_worker_job_t;

// initialize a synchronization token for the number of jobs to be
// submitted (done by master)
void dspCV_worker_pool_syncToken_init(dspCV_syncToken_t *token, unsigned
int njobs);

// submit a callback + data ptr job to the worker pool (done by master)
int dspCV_worker_pool_submit(dspCV_worker_job_t job);

// release a synchronization token (done by the worker callback)
void dspCV_worker_pool_syncToken_jobdone(dspCV_syncToken_t *token);

// wait until all synchronization tokens have been released (done by
// master)
void dspCV_worker_pool_syncToken_wait(dspCV_syncToken_t *token);

// utility to atomically increment a counter. Useful for letting workers
// compete for numbered jobs
unsigned int dspCV_atomic_inc_return(unsigned int *target)

// utility to atomically decrement a counter. Useful for letting workers
// compete for numbered jobs
unsigned int dspCV_atomic_dec_return(unsigned int *target)

```

- **dspCV\_hvx.h HVX APIs**

The dspCV library contains helper functions to assist with HVX resource logic in \$ {HEXAGON\_SDK\_ROOT}/lib/fastCV/dspCV/inc/dspCV\_hvx.h.

The APIs it contains are documented inline in that file.

```
int dspCV_hvx_reserve(unsigned int num_units);
void dspCV_hvx_unreserve(void);

int dspCV_hvx_num_reserved(void);

int dspCV_hvx_power_on(void);

void dspCV_hvx_power_off(void);

int dspCV_hvx_lock(dspCV_hvx_mode_t mode, unsigned int block);

void dspCV_hvx_unlock(void);

void dspCV_hvx_prepare_mt_job(dspCV_hvx_config_t *hvx_config);

void dspCV_hvx_cleanup_mt_job(dspCV_hvx_config_t *hvx_config);

void dspCV_hvx_disable(void);

void dspCV_hvx_enable(void);

void dspCV_hvx_set_default_mode(dspCV_hvx_mode_t mode);
```

- **dspCV\_concurrency.h Concurrency APIs**

In addition to being remoted to the application processor, the concurrency information may also be queried by the DSP compute application implementation from with the DSP.

The APIs are available with inline documentation in \${HEXAGON\_SDK\_ROOT}/lib/fastCV/dspCV/inc/dspCV\_concurrency.h.

```
void dspCV_concurrency_query(dspCV_ConcurrencyAttribute* attrib, int attribLen);

void dspCV_concurrency_set_audio_mpps_1_hvx_threshold(int threshold);

void dspCV_concurrency_set_audio_mpps_2_hvx_threshold(int threshold);
```

One of the attributes available for the `dspCV_concurrency_query()` API is `COMPUTE_RECOMMENDATION`. Requesting this attribute will return `COMPUTE_RECOMMENDATION_OK` or `COMPUTE_RECOMMENDATION_NOT_OK`, depending on the current concurrency.

A compute application should make this query at convenient times, at least every 30 msec, and halt DSP processing upon receiving `COMPUTE_RECOMMENDATION_NOT_OK`. Other attributes (listed in \${HEXAGON\_SDK\_ROOT}/lib/fastCV/dspCV/inc/dspCV.idl) are also available for gathering additional information.

## Qualcomm Computer Vision SDK library

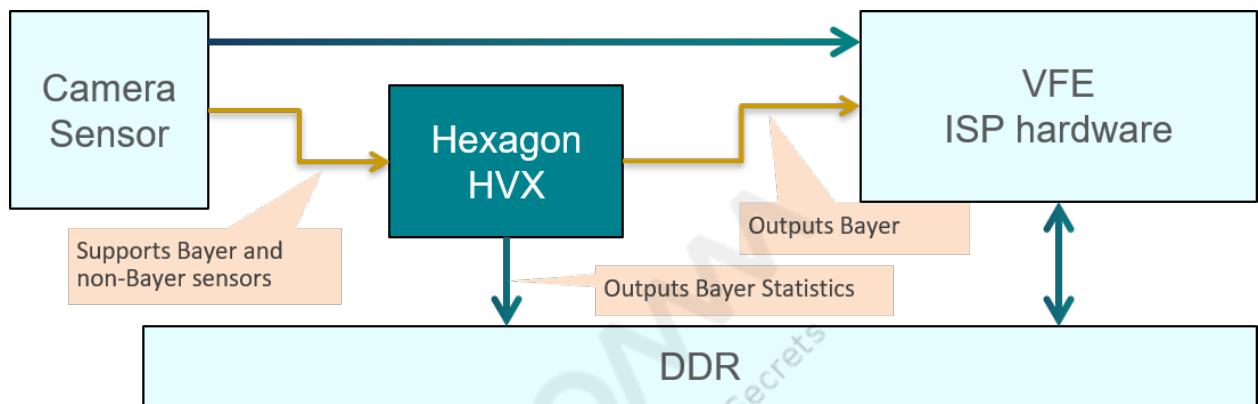
Hexagon SDK contains the Hexagon Qualcomm Computer Vision SDK libraries conforming to version 1.7.1 of the Qualcomm Computer Vision SDK API. These libraries are found at \${HEXAGON\_SDK\_ROOT}/lib/fastcv/fastcv and are available as static libraries (for linking into simulator executables), and as dynamic libraries (for on-target usage).

The list of Qualcomm Computer Vision SDK 1.7.1 APIs that have been hand-optimized for Hexagon V5x and later are listed in the *Hexagon SDK documentation*.

Hexagon V6x is backward-compatible to benefit from all the V5-optimized functions. Additionally, functions that have been reoptimized specifically for HVX are listed in *Hexagon SDK documentation*.

For more information, see the *Qualcomm Computer Vision SDK APIs in QCS610/QCS410 Software Programming Guide* (80-PL631-200).

### 6.3.4 Camera streaming



A special bus enables access between camera sensor and hardware.

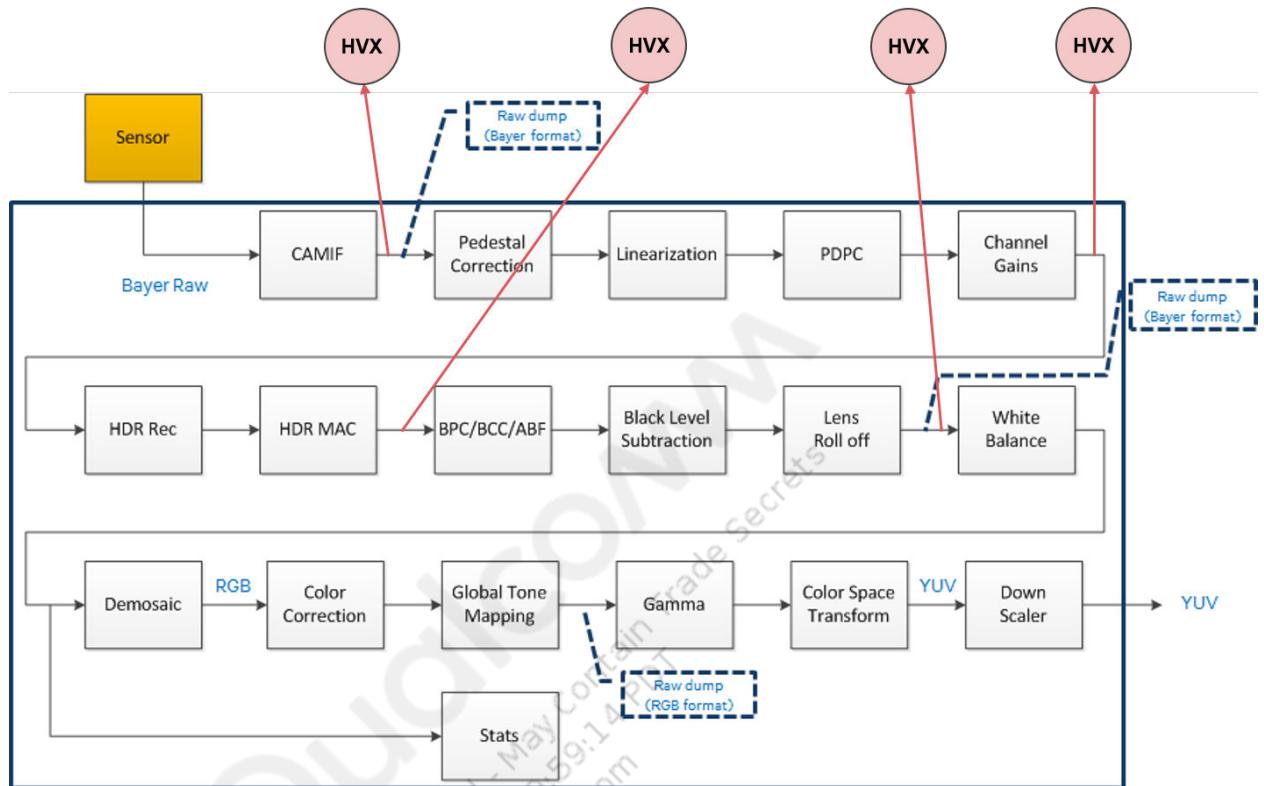
The following are the camera streaming use cases:

- OEM differentiating camera feature
  - Opportunity to reduce BOM by removing external chips
  - Variation in camera sensors (non-Bayer, different resolution) requires programmability
- Direct streaming interface avoids DDR memory accesses
  - Hexagon cDSP can be inserted between the camera sensor and video front end (VFE) (QTI ISP)
- Example algorithm/use case: Custom Bayer statistics for auto-focus/auto-exposure
  - Bayer Focus (region line max, sharpness, sum, num)
  - Bayer Exposure statistics (num, sum)

For more information, see the following documentation:

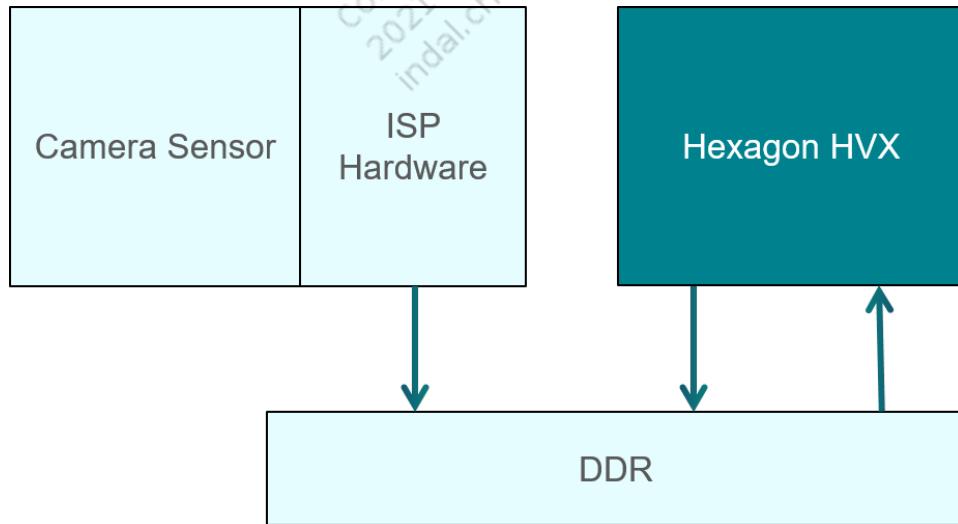
- *Qualcomm Hexagon HVX Streaming Customization for Camera* (80-NF772-36)
- *Hexagon SDK 3.4, examples/camera\_streaming*

The figure shows the HVX tap points. For more details on enabling the tap points, contact QTI support.



**Figure 6-16 Camera streaming – HVX tap points**

The figure shows the use case for memory to memory before ISP or after ISP:

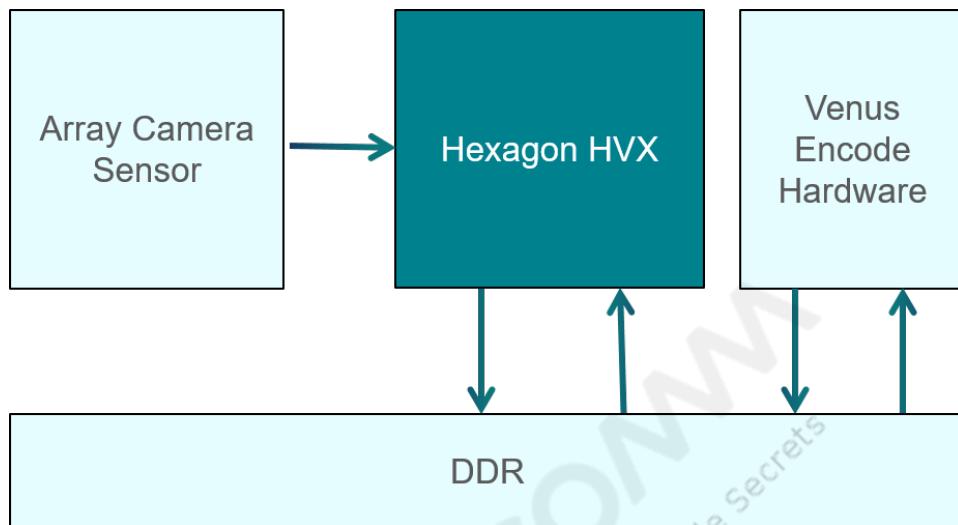


**Figure 6-17 Camera Memory-to-Memory Pre/Postprocessing**

As an example, the application can be done for wavelet-based denoiser and CV.

For more information, see *Hexagon SDK 3.4, examples/compute/downscaleBy2*.

The figure shows the computational computational photography using array cameras for next generation use cases such as augmented reality and virtual reality.



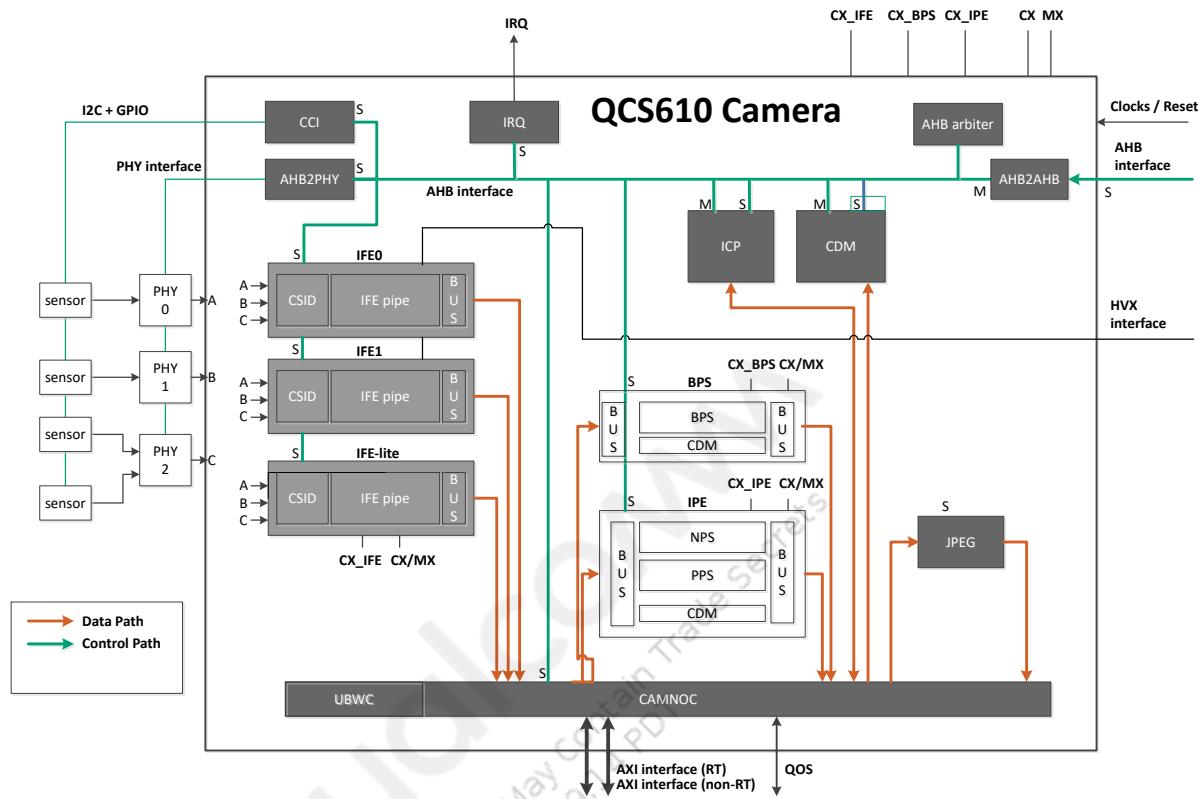
**Figure 6-18 Camera streaming – computational camera**

A non-traditional camera cannot use ISP hardware. All ISP functions must be done in the software.

## 6.4 Camera

The camera core is based on the Spectra 2xx. The Spectra 2xx camera subsystem includes the following:

- Image front-end (IFE)
- Bayer processing segment (BPS)
- Image processing engine (IPE)
- Camera peripheral and support (CPAS)



**Figure 6-19 Camera high-level block diagram**

- Physical layer (PHY) – The mobile industry processor interface (MIPI)-camera serial interface (MIPI-CSI)-based sensors are mounted on the PHY interface. QCS610/QCS410 has three combination PHYs, which support both C-PHY and D-PHY interfaces.
- Camera serial Interface decoder (CSID) – A MIPI-CSI based sensor sends the CSI-2 encoded frames to the PHY serially. The PHY turns this data into byte data, which the CSID decodes into unprocessed pixels.
- Image front-end (IFE) – IFE block supports bayer processing of preview and video. It can also generate bayer stats that can be used by I/Q blocks further downstream. For more information, see *Qualcomm Spectra 2XX Deep Dive* (80-P9301-60).
- QCS610/QCS410 has two instances of the IFEs and hence supports simultaneous streaming of at most two bayer cameras.
- IFE-Lite – QCS610/QCS410 also supports one instance of the IFE-Lite block. The IFE-Lite can support a maximum of 2 MP streams at 30 fps. It does not support bayer processing and can only support raw data interface (RDI) streams. It can only handle YUV or mono cameras.
- Bayer processing segment (BPS) – The BPS is an offline engine that is mostly used for enhanced IQ processing in the bayer domain. This is not a realtime engine and hence typically linked for additional noise processing snapshot streams.
- Image processing engine (IPE) – The IPE block is mainly used for post processing of preview, snapshot and video frames, followed by the bayer processing. This is another offline processing engine. For more information on its in-built IQ blocks, see *Qualcomm Spectra 2XX Deep Dive* (80-P9301-60).

## 6.4.1 Camera features

The following are the key camera features:

- Motion compensated temporal filter (MCTF)
- Sensor formats support
- Manual 2A settings
- Adaptive dynamic range compression (ADRC)
- IR-Mode
- Staggered high dynamic range (sHDR)
- Lens compensation/De-warp
- Objective and subjective image quality verification

The camera ISP offers enhanced image and video quality for multiple high-FPS sensors with lower power, less bandwidth, and better hardware-software sync.

The tables provide the camera ISP and image quality capabilities of the QCS610/QCS410 chipset:

**Table 6-5 QCS610/QCS410 camera ISP features**

Feature	Camera ISP
Number of IFE	Three (two IFEs + one IFE lite); 16 + 16 + 2 MP
Maximum video performance	4k@30
Number of offline engines	One IPE and one BPS
Snapshot/video pipeline	Discrete pipelines for video and snapshot
Sensor	Bayer (RAW10, RAW12), zzHDR, PDAF, and YUV
Image quality features	<ul style="list-style-type: none"> <li>▪ Better NR – HNR, multipass ANR</li> <li>▪ MCTF</li> <li>▪ Better processing order – NR &gt; local tone-mapping (LTM)/color processing</li> <li>▪ 14/10-bit internal, IPE can output only 8-bit YUV</li> <li>▪ Multiframe processing – MFNR and MFSR</li> <li>▪ 2D color LUT</li> <li>▪ Bayer processing, ASF, and LTM improvements</li> </ul>
Power/bandwidth	<ul style="list-style-type: none"> <li>▪ UBWC at IPE, IFE, and reference output</li> <li>▪ Single input multiple outputs (SIMO)</li> </ul>
Camera i/f	<ul style="list-style-type: none"> <li>▪ D-PHY1.2 – 4/4/4; 2.5 Gbps/lane</li> <li>▪ C-PHY1.0 – 3T/3T/3T; 5.7 Gbps/T (3T = 17.1 Gbps)</li> </ul>

**Table 6-6 QCS610/QCS410 image quality (IQ) features**

Feature	Description
Spatial noise processing	<ul style="list-style-type: none"> <li>▪ The frequency-based denoiser and multiresolution denoiser reduce noise in low-light.</li> <li>▪ The noise reduction is done before LTM and color processing</li> <li>▪ Grain: A fine random noise is added to the Y channel and the subjective quality of images is improved after NR</li> </ul>
Multiframe super resolution	<ul style="list-style-type: none"> <li>▪ Blends several images into one of a greater resolution.</li> <li>▪ Determines whether to run multiframe super resolution based on zoom ratio</li> <li>▪ Chooses the anchor image and blend all images into one YCbCr 4:2:0 image</li> </ul>
High-quality low-power temporal noise filter	<ul style="list-style-type: none"> <li>▪ Motion compensated temporal filtering (MCTF) is an I/Q enhancement feature that blends frames between static areas and helps reducing noise.</li> <li>▪ High-quality low-power temporal noise filter</li> <li>▪ Blends between frames in static areas thus reduce noise</li> <li>▪ Uses ICA2 and TF</li> </ul>
High-quality hue and saturation control	<ul style="list-style-type: none"> <li>▪ 2D LUT on hue (H) and saturation (S)</li> <li>▪ <math>24 \times 16</math> (<math>H \times S</math>)</li> </ul>
Supported formats	Bayer, RCCB, MONO, zzHDR, and PDAF (sparse and 2PD)

For more information, see *Qualcomm Spectra™ 2xx Camera ISP Overview* (80-P9301-14).

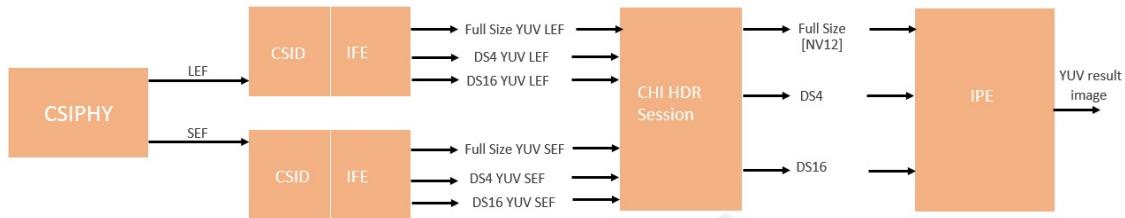
#### 6.4.2 sHDR overview

The sHDR mode is a sensor feature that outputs frames with different exposure times.

- The first one is the long exposure frame (LEF) and the second one is the short exposure frame (SEF).
- The sensor outputs a pair of two lines as one unit. The frame of LEF and frame of SEF are output alternately in the pair of these two lines.
- The rolling shutter readout is staggered (row interleaved) so that the short integration starts immediately (within the same frame) after sampling of the long integration. It is also called Digital Overlap mode.
- The sHDR sensors are also capable of outputting LEF and SEF frames separately where each frame is sent out on different virtual channel (VC)/DT.
- QCS610 supports two sHDR algorithms, sHDR v2.0 and sHDR v3.8.
  - The default version supported is sHDR v3.8. sHDR 3.8 solution requires dual VC support from sensor which outputs LEF and SEF frames on separate VC.
  - The sHDR 2.0 solution requires sensor to output both LEF and SEF on single VC in an interleaved manner.

The sHDR feature is not supported on two concurrent, independently running cameras. For more information, see *Staggered HDR Overview and Tuning Guide* (80-PF105-40).

### 6.4.2.1 sHDR 3.8 dual VC with two-frame exposures



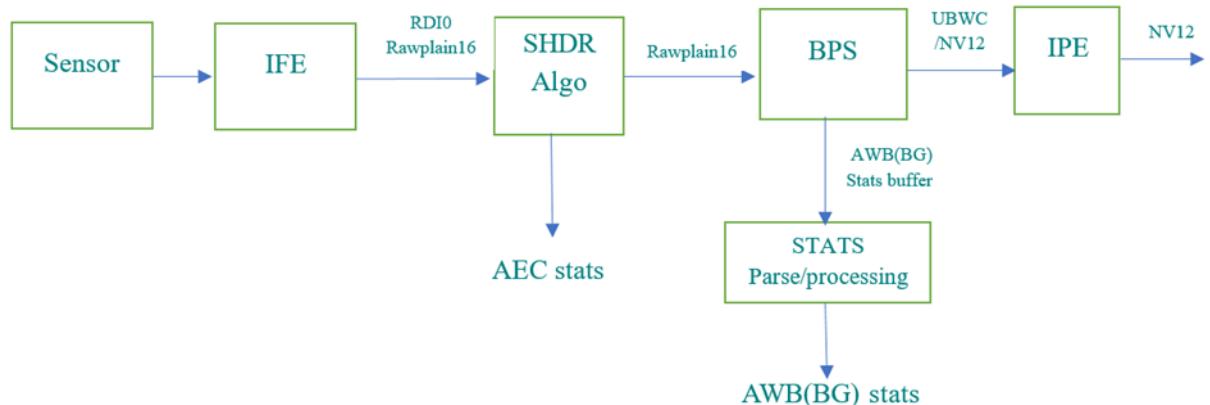
**Figure 6-20 sHDR 3.8 data flow for dual VC**

The sHDR 3.8 solution supports sensors that can send the exposure frames (LEF and SEF) on different CSI VCs.

1. The long and short exposure frames are transmitted on different VCs to different IFEs for YUV processing.
2. Both short and long frames are processed in individual IFE block before fusion.
3. Each IFE outputs full YUV image and scaled images (DS4 and DS16), which is fed to sHDR fusion algorithm for HDR processing.
4. The fusion is done in YUV domain.
5. The sHDR algorithm outputs full frame, DS4, and DS16 frames that are fed to IPE for color and noise processing.

### 6.4.2.2 sHDR 2.0 single VC with two-frame exposures

The sHDR 2.0 requires sensor to output both LEF and SEF on single VC in an interleaved manner. It is called as digital overlap mode (DOL). The rolling shutter readout is staggered (row interleaved) so that the short integration starts immediately (within the same frame) after sampling of the long integration. There is a vertical offset between the SEF and LEF, which is instinct for the sHDR algorithm.

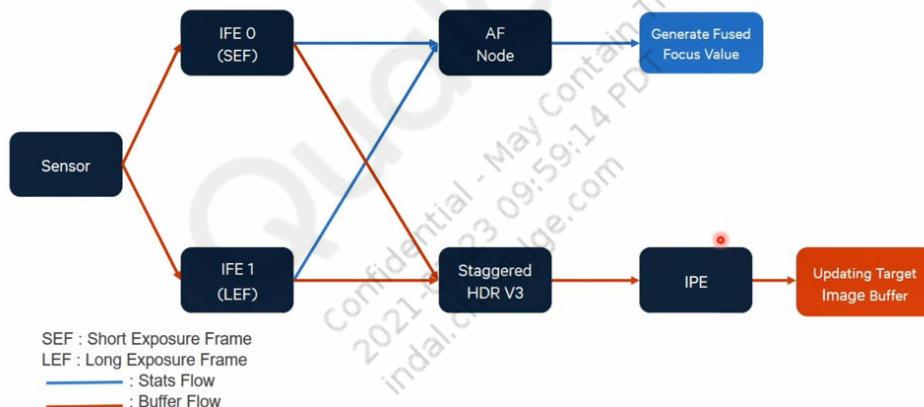


**Figure 6-21 sHDR 2.0 data flow for single VC**

- The sensor outputs raw frame, which is a two-frame DOL HDR (LEF + SEF), transmits on a single VC to IFE.
- The IFE outputs this raw frame in plain 16 format without any Bayer processing (through its RDI port). This sHDR raw image is given to the sHDR 2.0 algorithm. The LEF and SEF offset values of the sHDR frame are also passed to sHDR algorithm. The long and short exposures are fused by the sHDR algorithm on the GPU.
- The sHDR algorithm also generates the AEC stats, which are software-based. The raw output image data from the sHDR algorithm is given to the BPS.
- The Bayer data processing happens on the BPS and outputs the YUV/UBWC frame. The BPS also outputs AWB (BG) stats that are parsed/processed by the stats module.
- The YUV/UBWC frame data is processed by the IPE to perform noise reduction, color and detail enhancements and so on, which are passed to the application/framework.

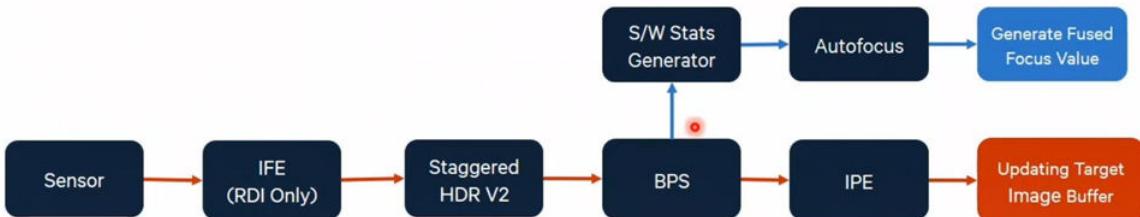
#### 6.4.2.3 sHDR with autofocus

The QCS610 platform supports the autofocus feature on both sHDR 2.0 and sHDR 3.8.



**Figure 6-22 sHDR3.8 with autofocus**

- In the sHDR 3.8 use case, the hardware stats (buffer flow) are generated by two IFEs, one with SEF and other with LEF.
- They are used by the AutoFocus node to generate the final focus value to achieve better autofocus.



**Figure 6-23 sHDR 2.0 with autofocus**

- In the sHDR 2.0 use case, IFE is running RDI mode, therefore, there are no available hardware stats.
- The BPS registration image buffer output is processed to generate BF stats in software, which are used by the Autofocus node to generate the focus value.

### 6.4.3 LDC overview

The LDC is the process to correct the distortion that is introduced due to fish-eye lens, which makes a straight line in a scene to be captured as a curved line due to the distortion of the lens.

LDC is part of the use case xml, as a separate use case with output of IPE node passed to Image Warp CHI node. The Image Warp CHI node uses the static mesh that is generated using lens calibration. The CHI node calls the iWarp (image Warping) library APIs internally. iWarp module is a software feature that runs on Qualcomm Adreno GPU. It applies warping to the given input image based on input static mesh. iWarp library uses OpenGL api for warping.

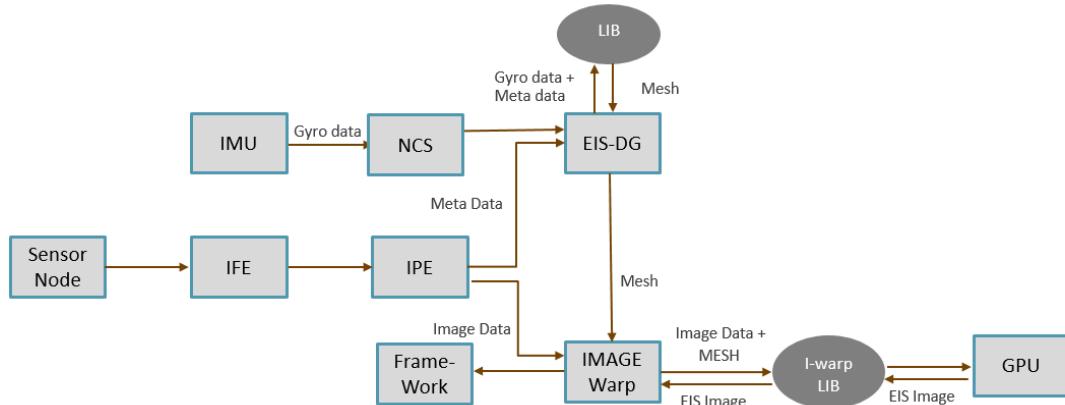


**Figure 6-24 LDC data flow**

For LDC calibration procedure, see *360 Camera Calibration* (80-P9894-1) and *IoT Lens Distortion Correction* (80-PN984-7).

### 6.4.4 EIS overview

EIS is an image enhancement technique using electronic processing. EIS minimizes blurring and compensates for device shake, often a camera. EIS takes the motion data from IMU sensor and generate the transformation matrix to compensate for device moment in all three direction. Gyro sensor provides motion in pitch, yaw, and roll.



**Figure 6-25 EIS data flow**

1. EIS-DG CHI node takes image metadata from the vendor tags published by sensor, IFE and IPE. It also fetches the Gyro data from NCS interface.
2. The CHI node configures the EIS library with image metadata and Gyro sample to generate the transformation mesh. Library generates transformation mesh for each frame to be applied on the image by GPU.
3. The iWarp CHI node receives the image data (YUV) from IPE and transformation mesh from the EIS DG node, and performs the warping of the frame using GPU.
4. The GPU generates the final warped output YUV image, which is passed to the framework/encoder.
5. Image Warp CHI node calls the iWarp library APIs internally. The iWarp library uses OpenGL API for warping.
6. LDC is enabled by default with EIS. The iWarp CHI node is capable of supporting EIS + LDC warp simultaneously.

#### 6.4.5 Dependency before running EIS and LDC

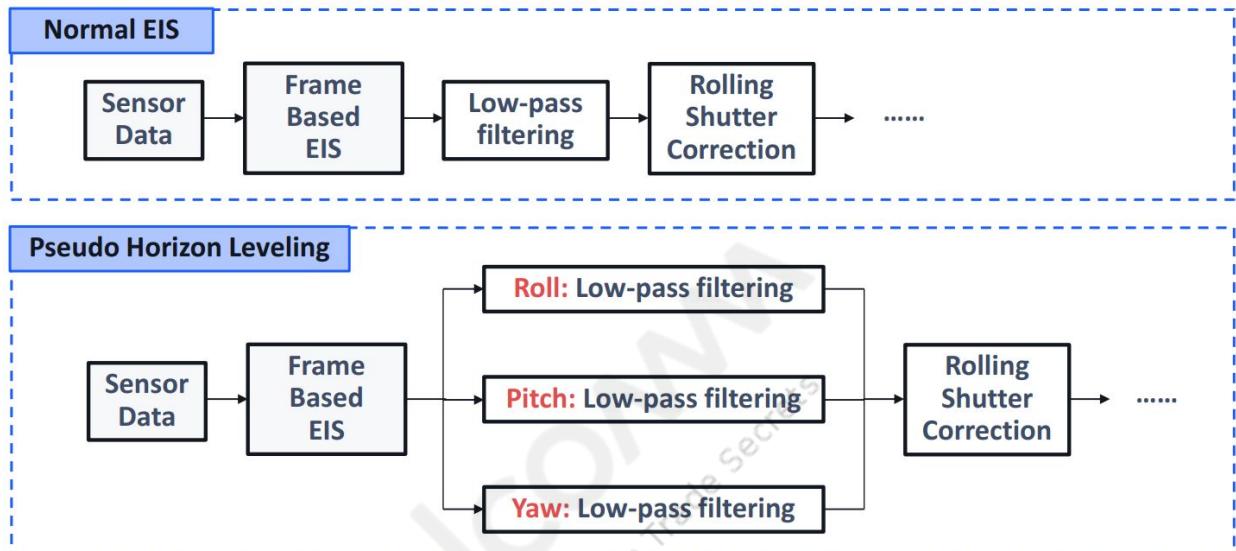
The OpenGL ES based implementation of LDC and EIC operates on GBM buffers, which need Weston server that runs manually before running the use case. The Weston server is responsible for allocating GBM buffers using EGL driver.

The EGL driver operates on DRM device node /dev/dri/card0. Ensure that the /dev/dri/card0 path is available before launching Weston. This node is created when the display hardware is present.

The following options enable you to run the Weston server:

- DSI panel  
./weston
- DSI with the HDMI bridge chip  
./weston -device=hDMI
- If the /dev/dri/card0 node is not present, that is, if there is no physical display hardware available, this option enables you to run Weston with simulated or dummy display panel:  
/weston --device=displaydummy

### 6.4.6 EIS with horizon leveling



This feature is intended for drone use cases, drone movement is more in Yaw (Horizontal) direction and less in Roll/Pitch(Up/Down) directions.

In Normal EIS, we use single Low-pass filtering for all directions Roll, Pitch and Yaw.

In Horizon Levelling EIS feature, we apply individual tuning of low-pass filtering for each orientation Roll, Pitch and Yaw. We apply bigger low-pass filtering in Yaw direction (as drone movement is more in Horizontal direction) and weaker low-pass filtering in Roll and Pitch directions. EIS-DG node takes care of Horizon Levelling with individual low-pass filtering for each direction.

### 6.4.7 Defog overview

Defog is a fog detection and removal technique. Defog is a video processing function that restores details and color to obtain accurate and natural image or video. Defog is also the common imaging technique that maintains clarity in videos that are captured in poor weather conditions such as rain, smog, haze, or fog.

The defog library performs the defog operation with the stats and interpolation data, and generates new tables to the IQ modules to apply them for the next frame.

For defog verification, see *QCS610/QCS410 Software Programmer's Guide* (80-PL631-200).

### 6.4.8 Multiframe shot – MFNR

Multiframe algorithms blend information from multiple images into a single image. This makes the final single image better than each of the initial images:

- Less noise (MFNR): MFNR blends several images into one. A typical number of input images is 8 (OEMs may use 6 images or sometimes 3 in their solution).
- Higher resolution (MFSR): MFSR blends several images into one of a greater resolution. Typical upscaling factors are 2 and 4. A typical number of images is 4.

To blend the images, multiframe algorithms use warping – a transformation of each image, which aligns them to a chosen one, called the anchor image. The warping parameters are calculated by the registration algorithm.

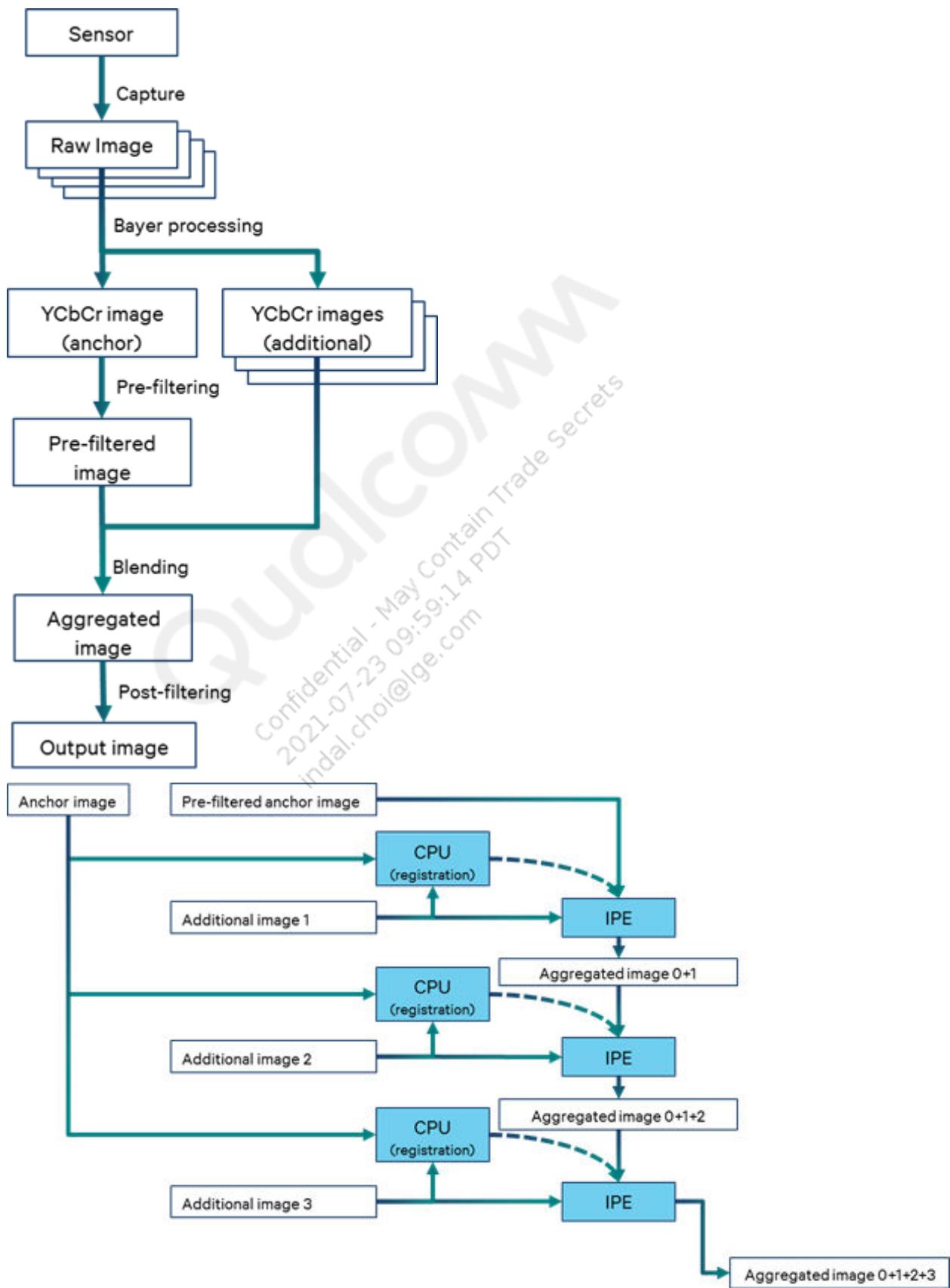
Moving objects are captured in a different state or location in each input image; for moving objects, the final image should show their state as it was in the anchor image. The algorithm that determines which parts are moving objects, is called “ghost detection”. For these moving parts one approach is to turn off (or reduce) blending of input images.

Multiframe algorithm performs the following tasks:

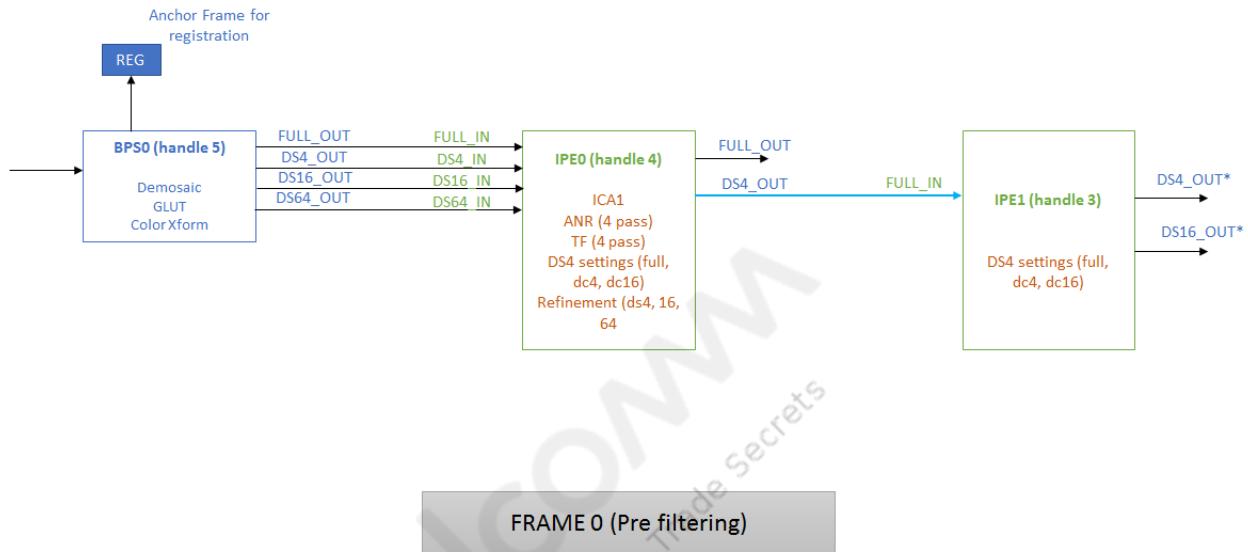
1. Determine whether to run any multi-frame algorithm
2. Capture the required number of images in raw format
3. Choose the anchor image, and determine processing order
4. Convert each image from raw to YCbCr 4:2:0 format
5. Filter the anchor image
6. Blend all images into one YCbCr 4:2:0 image
7. Do additional processing on the image

For the multiframe usecase, select a raw image as an anchor image first. OEMs can put anchor selection logic in CHI override, using a custom node.

Multiframe processing usecases have three-stage processing. First is "Prefiltering," which is done on the anchor image. Second stage is the "Blending" operation of all remaining raw images. The last stage is "Postfiltering," which is done on last raw image.

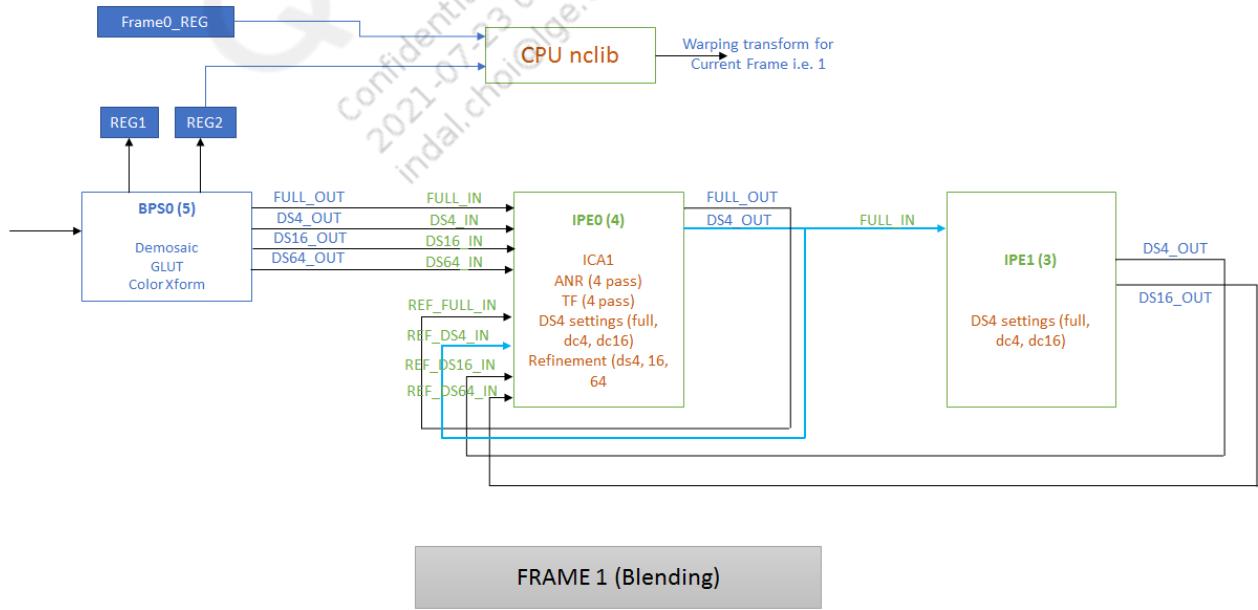


### 6.4.8.1 Multiframe processing



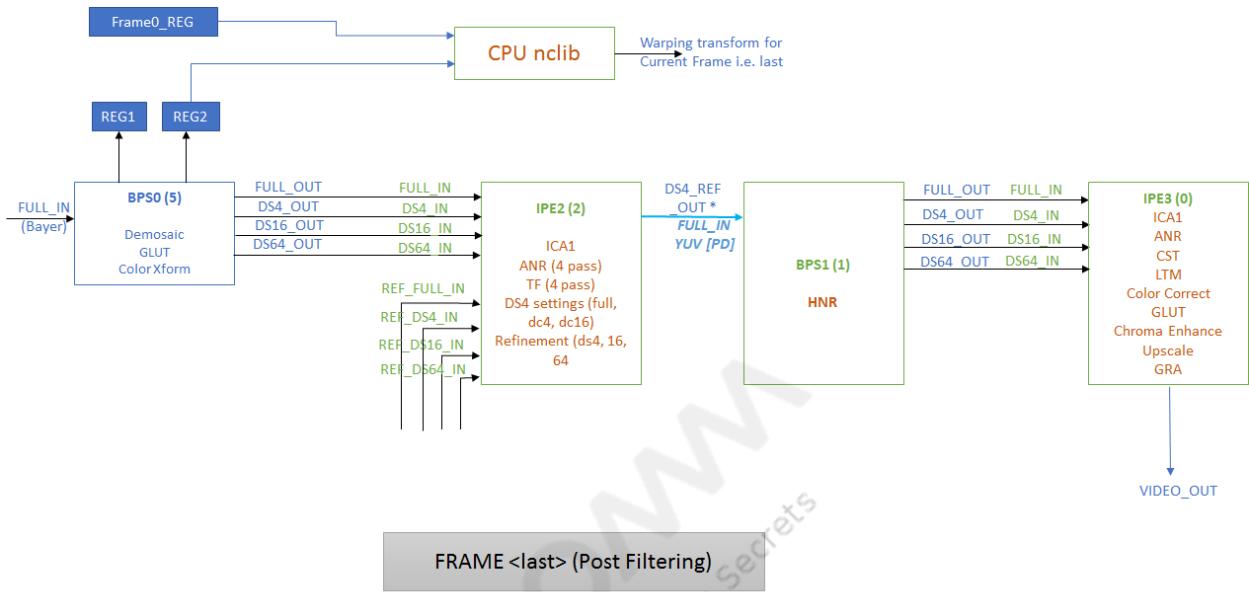
Note: \*DS4\_OUT and DS16\_OUT are the ports configured, but actual data coming out of them is 1:16 and 1:64 of the original input image (i.e. output of BPS0)

**Figure 6-26 MFNR processing – Prefiltering**



Note: \*DS4\_OUT and DS16\_OUT are the ports configured, but actual data coming out of them is 1:16 and 1:64 of the original input image (i.e. output of BPS0)

**Figure 6-27 MFNR processing – Blending**



**Figure 6-28 MFNR processing – Postfiltering**

This feature is implemented by using CHI override. To enable MFNR snapshot mode, CHI needs a hint (capture intent). In regular snapshot mode, CHI sends a raw frame from ZSL queue to offline topology (BPS → IPE). In case of MFNR mode, CHI needs to pick multiple raw images for processing. It then passes them through series of pipelines to perform; prefiltering → blending → postfiltering operations. The final YUV is given to JPEG for encoding. This is assuming the realtime preview pipeline is capturing normal, preview along with RDI raw, which is the same as the ZSL usecase (realtime streams i.e., sensor→IFE→IPE and with RDI raw).

When CHI override receives a hint about enabling MFNR, it will create four pipelines:

- Prefiltering [BPS + IPE (NPS)]
- Blending stage [BPS + registration + IPE (NPS)]
- Postfiltering [BPS + registration + IPE (NPS) + BPS (HNR) + IPE (PPS)]
- Scaling only [IPE (scale only)]

When pipelines are constructed, the session can do Acquire Device, buffer negotiation, etc., thus keeping it ready for streaming. When the capture request on the snapshot stream is received, the CHI layer will pick up the raw image from the ZSL queue as the anchor image. There are different criteria/selection logic for selecting a good anchor image.

To make the images easy to blend, all input images should be captured with fixed parameters:

- Gain/gamma
- Exposure
- White balance gain
- Color matching parameters

The anchor image is passed through the BPS, and REG\_OUT is generated, which will be the anchor for the registration process. It will also pass other BPS output through IPE (NPS only) to generate reference outputs for TF.

Registration output from the BPS is used to compute warping transform for ICA modules. Registration output is 3x downsampled image of the original.

First, BPS REG\_OUT buffer is used as the anchor image to registration process, and second frame onwards, with the help of current REG\_OUT output. The anchor frame registration algorithm generates the warping transform and confidence value. These output values will be used to configure ICA1 in the IPE (NPS only).

From the second frame, the blending stage begins. During the blending stage, IPE (NPS) node does temporal filtering. It receives the input reference buffers from the prefilter stage.

From the third frame onwards, TF will receive reference inputs from previous frames in a loop back manner. Thus, the blending stage starts preparing aggregated reference images that are used in postfilter stage.

For last raw frame, it passes through BPS and then through IPE (similar to the blending stage). After that, DS4\_REF\_OUT of the IPE will generate a full size PD yuv, which goes into the BPS for HNR processing and then finally through the IPE again with PPS enabled.

This stage has the following important points to consider:

- IPE is getting reference buffers from the previous pipeline (topology).
- IPE's DS4\_REF\_OUT is actually generating a full size image (not downsampled)
- IPE is connected with the BPS (as the BPS can understand yuv in PD format only)
- There are multiple instances of BPS and IPE that are doing different tasks, so we need to pass this information to specific node instances to achieve this.

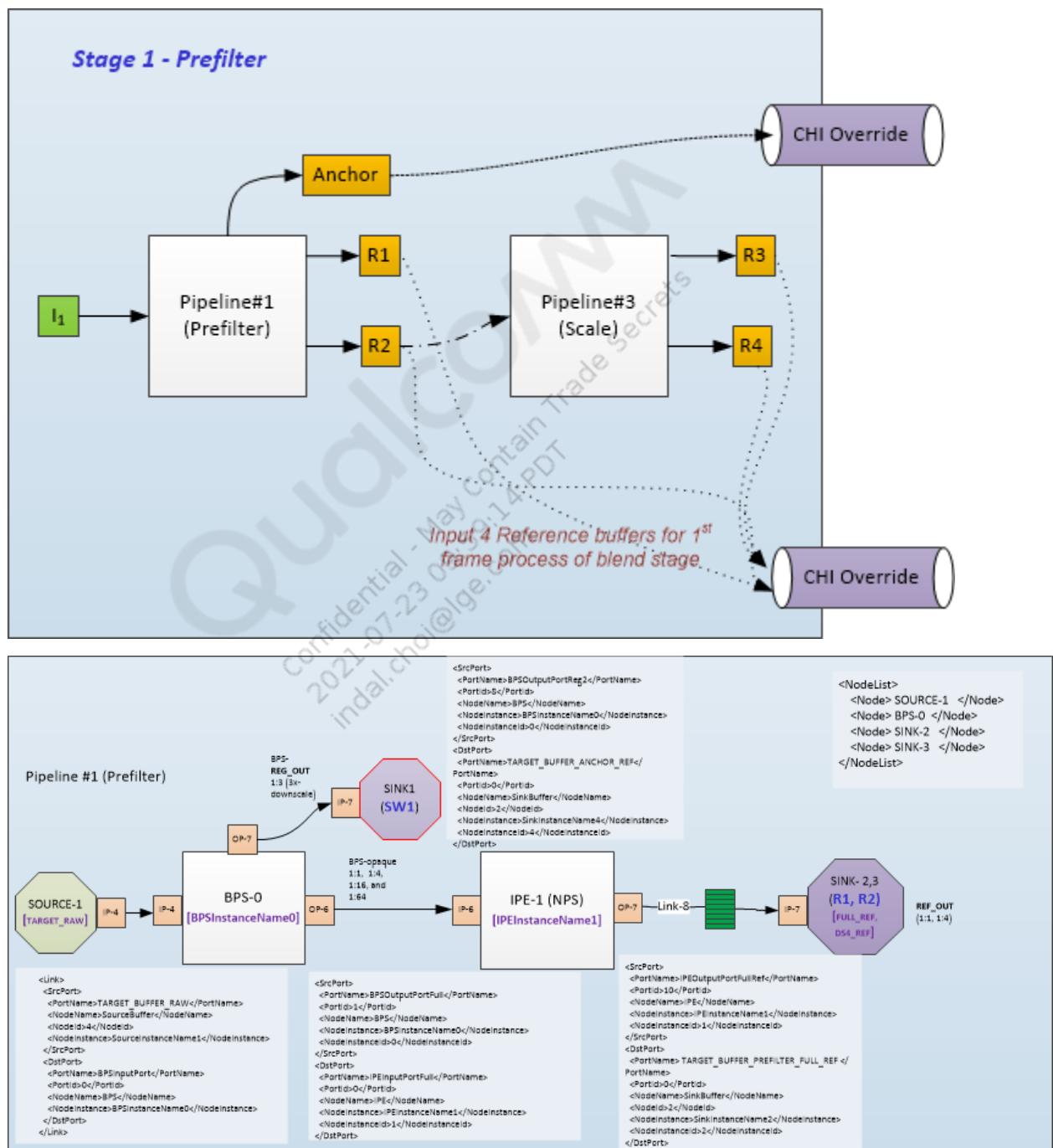
The CHI override/native application needs to create at least four pipelines to realize the MFNR usecase. Later we may need to divide some pipelines into smaller ones as part of optimization.

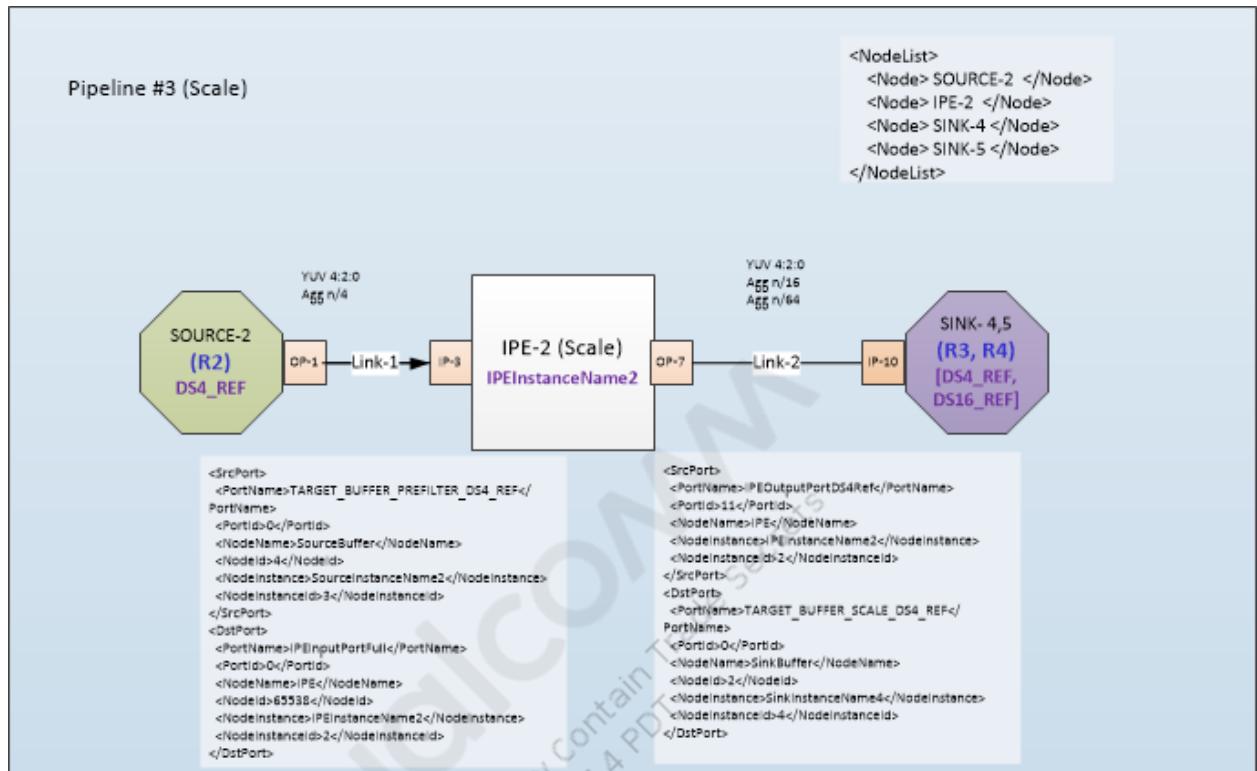
In summary, the first pipeline will be for the prefilter stage, to generate the anchor image and two reference images. There is a scale only pipeline (number 4) that will take DS4 reference buffer as input and its output will be two reference buffers (DS16 and DS64). Blending has similar pipelines with the addition of the registration process plus reference inputs. The CHI override layer needs to manage buffers across multiple pipelines. There is no buffer negotiation between different pipelines.

The CHI override layer will create CHIPIPELINEREQUEST, which consists of multiple CHICAPTUREREQUESTs. Each CHICAPTUREREQUEST contains a request for a single pipeline. If memory/buffer is not restricted, the CHI override layer can load the entire sequence of MFNR in one shot to the driver. The driver needs fences to wait on dependencies before triggering the hardware.

### 6.4.8.2 Multiframe processing stages and pipelines

#### Stage 1 – prefilter



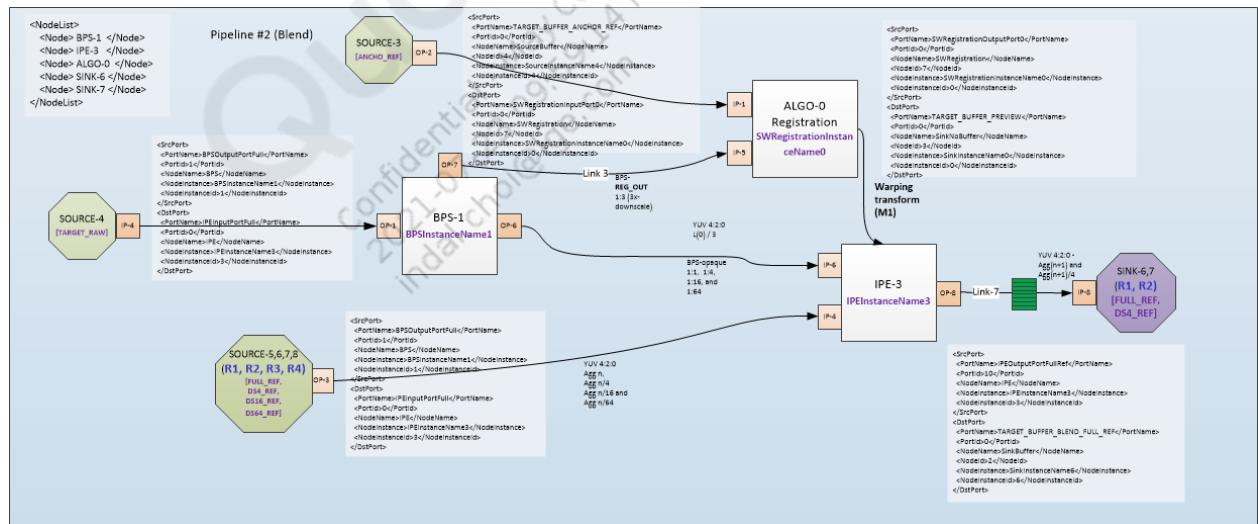
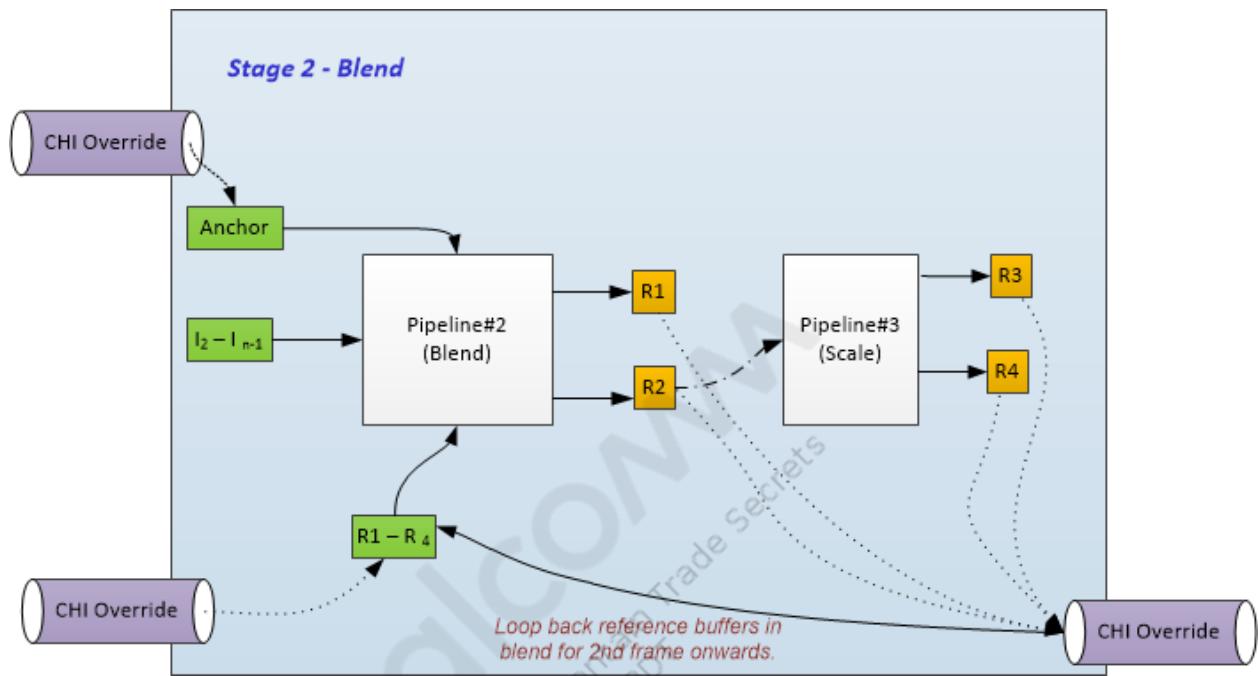


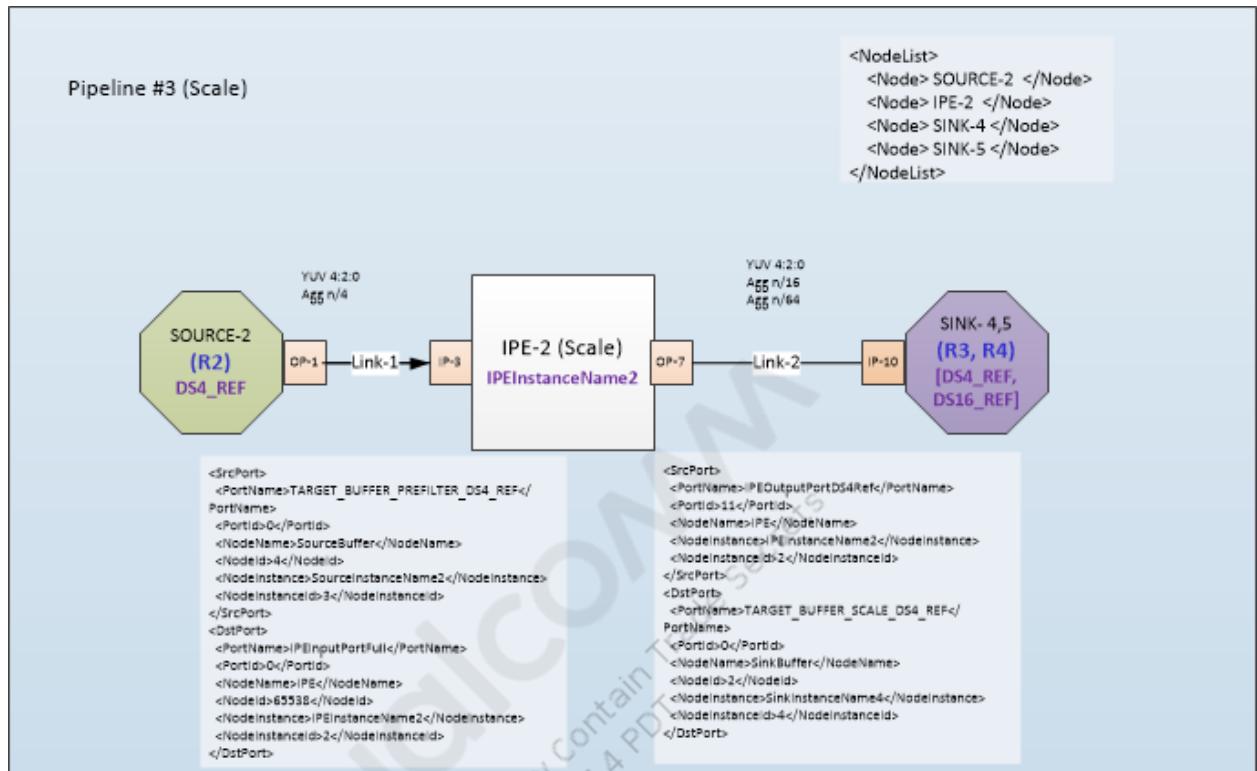
For prefetching, the CHI override will create two CHICAPTUREREQUEST requests. One for pipeline 1 and a second for pipeline 4. R2 from pipeline 1 is input to pipeline 4. If fence support is available, the override can submit a single pipeline request with input fence information to pipeline 4.

For the first frame, the input reference ports of the IPE0 instance will not get any buffer to begin with. The prefetching stage will generate two components:

- Anchor frame registration image buffer
- Reference temporal output buffers (full DS4, DS16, and DS64)

## **Stage 2 – Blend**





For blending, the CHI override will create two CHICAPTUREREQUEST requests. One for pipeline 2 and the second for pipeline 4. R2 from pipeline 2 is input to pipeline 4. If fence support is available, the override can submit a single pipeline request with input fence information to pipeline 4. Pipeline 2 requires input reference buffers that are fed in by the CHI override. Those reference buffers come from the prefilter stage or are used from the output of previous blend requests.

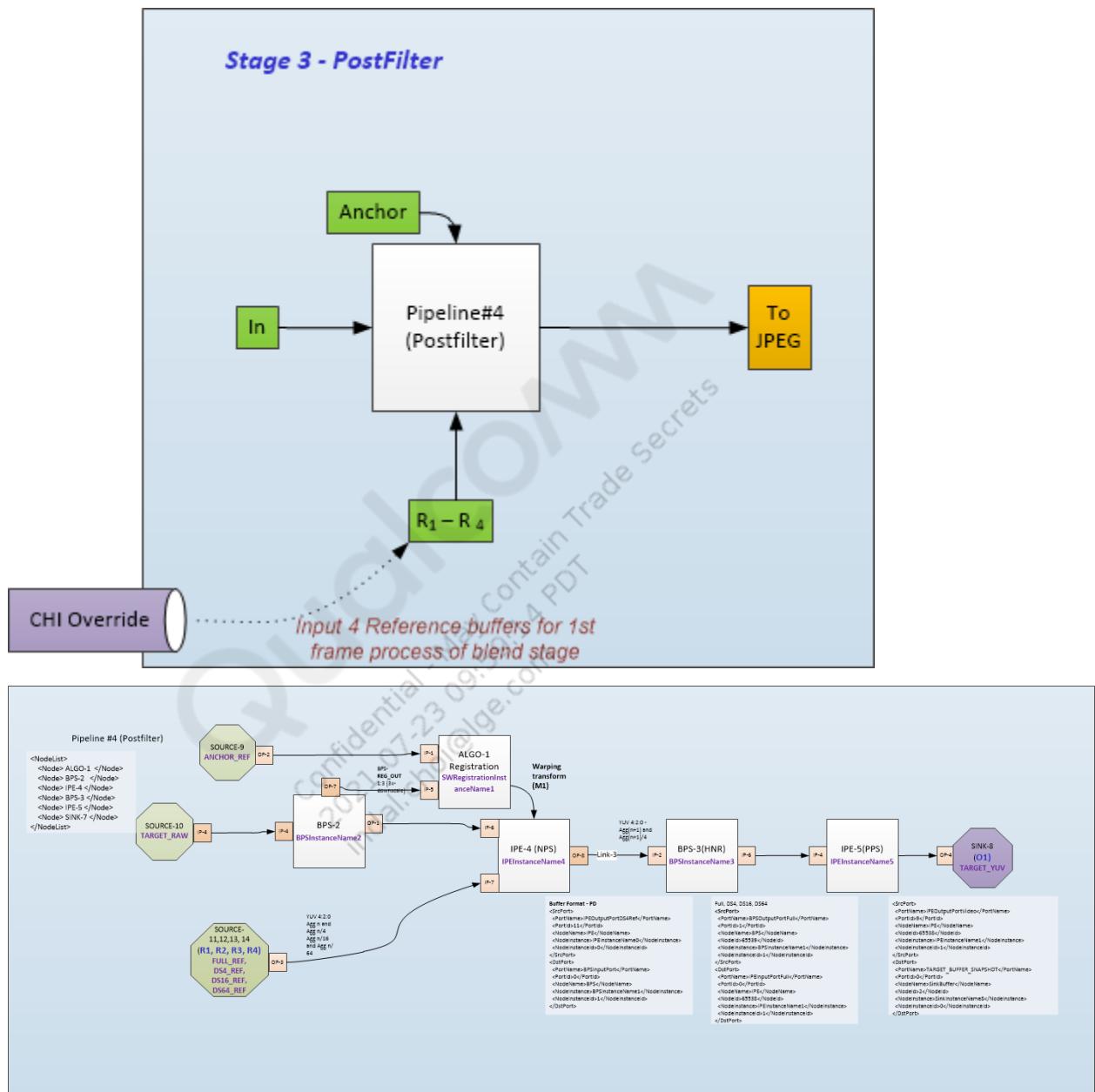
The blend request sequence is repeated for (n-2) times, where n is the total number of MFNR frames.

In the blending stage, reference ports of the IPE0 instance are configured using reference outputs from the previous ProcessCapture. Meaning reference buffers are fed into IPE0 with a delay of 1. In this stage, registration process kicks in, which uses the anchor frame as one of the input, and current reg\_out of BPS as the second input. The registration process is a software algorithm run in a separate CamX software node. The input to this node is the anchor image and current image, and its output is the warping transform for ICA2.

Thus, the blending stage prepares aggregated images. To make the aggregate image easily mixable with other images, the PPS part of the IPE is disabled.

The blending stage will generate reference temporal output buffers (full DS4, DS16, and DS64).

### Stage 3 – postfilter



For postfiltering, the CHI override will create one CHICAPTUREREQUEST request. Pipeline 3 requires input reference buffers that are fed in by the CHI Override. Those reference buffers come from the output of the last blend request. CHI also needs to hold on to the anchor image till PostFilter is done.

For the last frame, CHI triggers the second pipeline. The second pipeline has two BPS instances and two IPE instances. As shown above, the BPS1 instance does HNR (luma noise reduction available in V2), and it takes DS4\_REF\_OUT (full size yuv in PD format).

The postfiltering stage will generate one component:

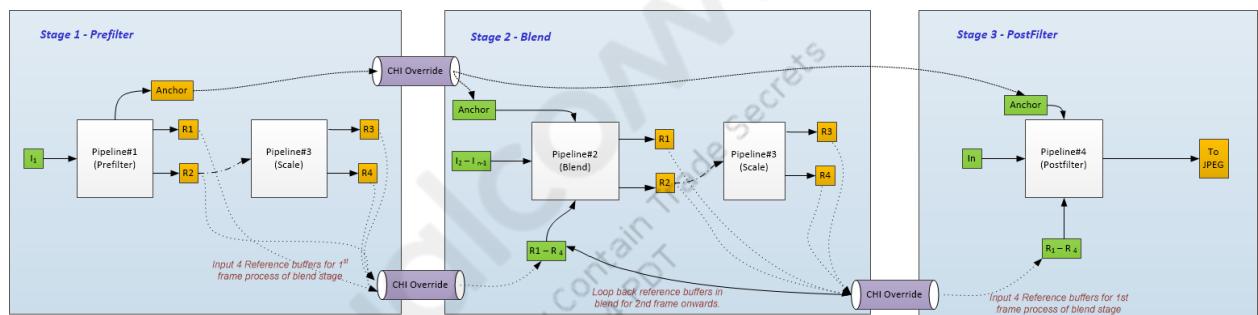
Final yuv to JPEG or if JPEG is connected, a snapshot.

The reason for using DS4\_REF\_OUT instead of FULL\_REF\_OUT is that the BPS can take only PD as the input format, and only DS4\_REF\_OUT can produce that. The last IPE instance shown in the diagram above finally does PPS and generates video out, which will be given to JPEG.

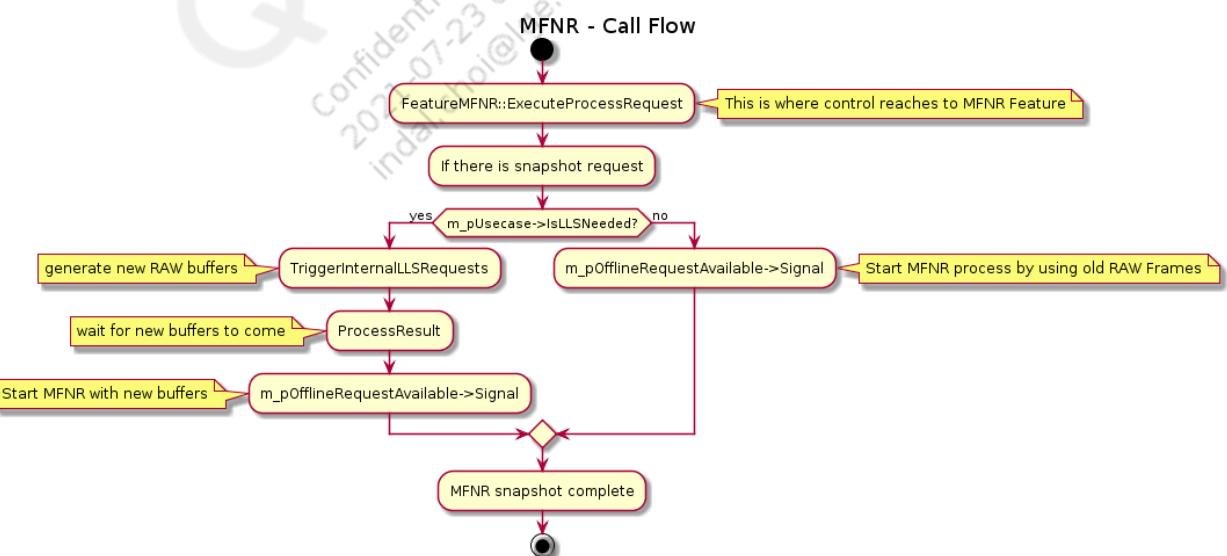
If hardware can support the PD format on FULL\_REF\_OUT, it would have been easier for the software to use the blending pipeline even for the postfiltering stage. BPS0(5) → IPE2(2) is being created in the diagram above because of hardware limitations. Ideally, if FULL\_REF\_OUT supports the PD format or the BPS supports another format other than PD, we could have only used the HNR and PPS stages from postfiltering.

In the postfilter IPE1, DS4\_REF\_OUT of the IPE is connected as the input to BPS2. Even though it is a DS4 port, the IPE is not really doing the downscale.

## All stages



## Summary of CHI override steps



1. Create five pipelines (realtime preview, prefilter, blend, postfilter, scale).
2. Allocate buffers (based on the total number of MFNR frames) for the reference aggregation (full, DS4, DS16, DS64).
3. Whenever the snapshot request is received from App, pick a RAW as anchor.

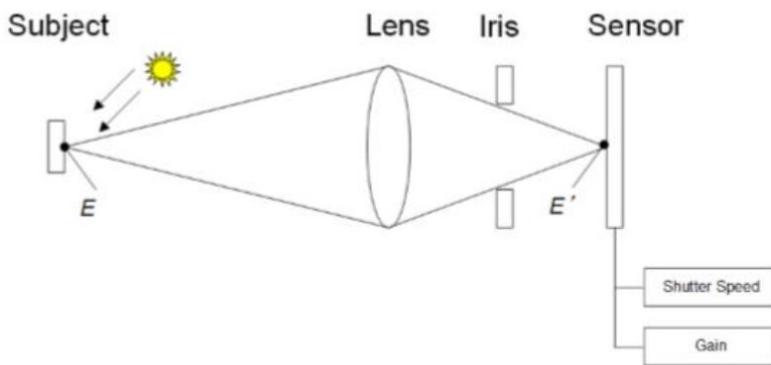
4. Begin the prefilter stage:
  - a. Submit a capture request for the prefilter pipeline (input – raw; output – 1 anchor, 2 reference buffers).
  - b. Submit a capture request for the scale pipeline (input – DS4\_ref; output – 2 refs).
5. Start the blending stage:
  - Submit a capture request for the blend pipeline (input – raw, anchor, 4 refs; output – 2 refs).
  - Submit a capture request for the scale pipeline (input – DS4\_ref; output – 2 refs)
6. Repeat the blending stage as many times as needed.
7. Start the postfilter stage:
  - Submit a capture request for the postfilter pipeline (input – raw, anchor, 4 refs; output – 1 yuv or JPEG).

There are multiple instances of BPS and IPE, and they have different functionality (NPS only, PPS only, downscale only, BPS HNR only, etc.). The <NodeProfile> in xml/xsd that will contain information about assigning appropriate profile to node instance when pipeline is created. The IPE/BPS node need to create IQ modules based on the profile selected.

Valid values for the IPE node profile are all, NPS, PPS, and scale. Valid values for the BPS node profile are all and HNR.

Nodes need to know which processing stage that are in (i.e., prefiltering, blending, or postfiltering) as they have to do specific things in each stage. The CHI override can put this information in the process capture request. If possible, it shall be considered to have 3A trigger-based switch between ZSL and MFNR.

#### 6.4.9 P-Iris



**Figure 6-29 P-Iris gain and shutter speed**

For QCS610, which has a moving Iris, the QTI lux\_index calculated from gain and shutter speed corresponds to E. To know the value of E, compensate the lux\_index value by F No. ratio.

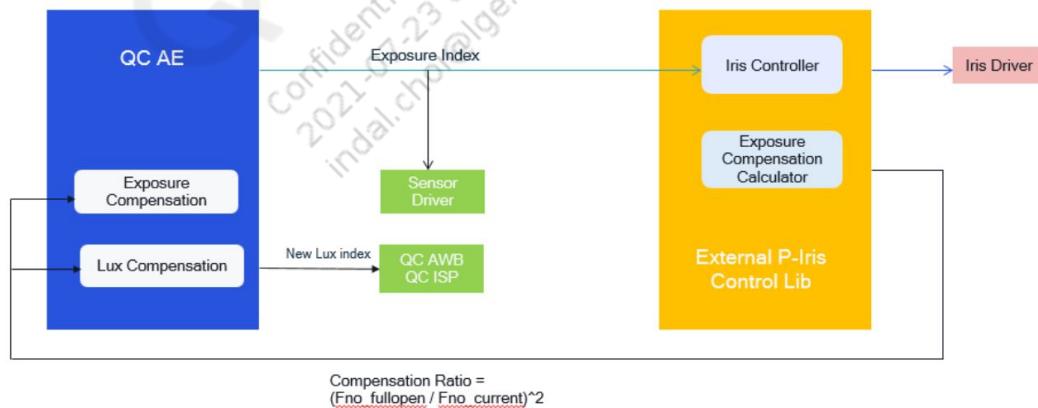
- The P-Iris driver/controller calculates the compensation ratio and sends it as a vendor tag to the QTI AEC algorithm.
- The AEC algorithm calculates the new exposure index based on this received compensation ratio.

### Prerequisites for P-Iris implementation

QTI P-Iris solution requires the OEM to send the compensation ratio based on the opening and closing of the Iris aperture. Specifically, they will have to implement the following

- P-Iris driver/controller should calculate the compensation ratio and send the ratio as camera metadata.
  - Compensation Ratio =  $(Fno\_fullopen / Fno\_current)^2$
  - Vendor tag – org.codeaurora.qcamera3.compensation\_ratio", "ratio"
  - Vendor tag data – Compensation ratio in float
- Implement the P-Iris controller and integrate it as part of the camera pipeline, and then send the compensation ratio as the vendor tag.

The QCS610 camera stack has support for P-Iris hardware on IPCamera. The AEC algorithm can be modulated by sending a compensation ratio from the P-Iris driver.



**Figure 6-30 Configuration of an IPCamera with P-Iris hardware**

1. The vendor tag is read and interpreted by AEC STATS processor as HAL param and passed to the AEC engine.
2. The AEC Engine then passes this ratio to AEC algorithm using AECAalgoSetParamCompensationRatio parameter.
3. The AEC algorithm considers the compensation ratio to calculate new exposure index.

To validate the P-Iris feature, do the following:

- Log string to confirm: (Enable AEC Verbose logs).
- Search with “compensation\_ratio.”
- Dummy values can be sent with the vendor tag from application or from some other nodes.

**NOTE** Iris works by limiting the amount of light that hits the image sensor. Too much light can wash out your video, and too little light can make everything dark. Therefore, it is recommended to have an Iris that is specifically suited for your camera location.

The supported compensation ratio is between 1 to 256, anything out of this range will be clipped.

#### 6.4.10 Live Tuning

- The Live Tuning feature allows a user to replace the QualcommChromatix™ Camera Calibration Tool tuning binaries without the need to restart the camera sessions.  
One can recreate the binaries with updated settings and push the files using adb push and the effects become visible immediately in a couple of frames.
- The Live Tuning feature depends on the chromatix.bins created using ParameterParser v2.2 or later.  
If the customers have generated the tuning binaries using an older version, QTI recommends to regenerate the binaries with the version present on the current build.  
Live Tuning is disabled by default. It can be enabled using the following override.  
`adb shell "echo enableLiveTuning=1 >> /etc/camera/camxoverridesettings.txt"`
- This feature can be validated by enabling/disabling some of the key features in the Chromatix7 binary file and pushing it to see the effect.  
For example, push the tuning binary file that has some IQ blocks disabled, like Gamma correction block. Then, check the video streams for effect (The stream should become dark).  
To revert to the original state, push the original binary file (Gamma-enabled). This doesn't need restarting the camera.

#### 6.4.11 IR mode

Infrared (IR) is invisible radiant energy, electromagnetic radiation with longer wavelengths than those of visible light, extending from the nominal red edge of the visible spectrum. Image sensors could see more spectral range than human eyes. Hence, to imitate human eyes, every camera has an IR filter that removes IR spectrum or light above 750 nm.

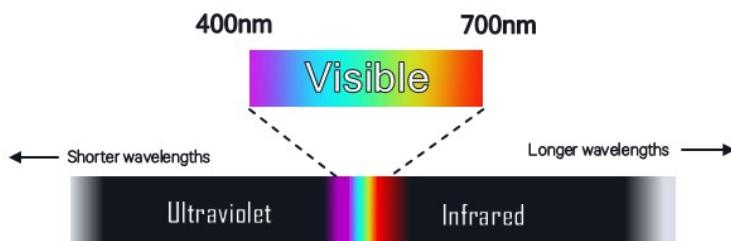


Figure 6-31 Spectral range of visible light and IR light

For an IR camera, the IR cut filter can be mechanically controlled.

- When IR cut filter is enabled, IR camera works in normal mode to block the IR wavelengths and then works like a normal camera.
- When IR cut filter is removed, camera works in IR mode since sensor is exposed to IR light in addition to visible light. This could increase the brightness in dark light conditions.

In addition to the IR cut filter, IR-LEDs are added to the IR camera to illuminate the target better and improve the dark light image quality.

As since the IR wavelength is visible to human eyes, the IR image look abnormal. Therefore, the IR camera is used to change the color of IR mode to mono.



**Figure 6-32 Normal mode and IR mode**

### IR modes setting

The camera software supports two modes for IR camera operations: OFF and ON.

- IR mode off – Normal camera operational mode
- IR mode on – Always on IR camera operational mode

IR mode can be controlled from application using the below vendor tag;

"org.codeaurora.qcamera3.ir\_led" with entries: "ir\_modes\_supported" and "mode"

#### Use case- IR mode OFF

In this mode, the camera works the same way as the smartphone camera mode. Irrespective of the lighting conditions, the camera always operates in normal mode. Camera output will have all the three components (YUV) of color and all the other features. In IR mode off, IR Cut filter is always in closed mode and removes all lights on IR spectrum and sensor behaves same as human eyes.

#### Use case- IR mode ON

In this mode, the camera works always in IR mode irrespective of the lighting conditions. Camera outputs in mono mode and will have input image information only on Y component, and UV component have a fixed color information. In IR mode on, IR Cut filter is always in opened mode and allows all lights on the IR spectrum and the sensor becomes responsive to dark light conditions.

## IR LED control

For IR camera equipped with IR LED, AEC dynamically controls the LED intensity to make proper exposure brightness. There are some parameters that can help AEC control the IR LED convergence.

See *Infrared Mode Tuning Guidelines* (80-PN984-21) for more info on infrared mode feature overview and tuning guidelines.

## 6.4.12 ISP support in BPS path

In the current CamX design, the number of active real time cameras is limited to the number of IFE instances that can handle data. As only two IFE modules exist in the current hardware revisions. This can make it difficult to support more than two Bayer camera modules.

A solution of this problem in the current hardware design can be achieved by using BPS for real time processing, with an IFE-lite utilized for dumping the RAW camera data into memory. This will allow at least three camera instances to operate in parallel.



BPS will generate the following:

- Processed image data (FULL, DS4, DS16 and DS64) - for downstream image processing. The image data will be forwarded to IPE.
- HDRHist and AWB statistics - for AE and AWB algorithm input.

As there is no ability to generate HDRBE stats in the BPS hardware, the AWBBG stats will be sent as HDRBE. A new 3A input port, StatsInputPortBPSAWBBG, is added to support BPS AWB and HDRBE stats generation.

## 6.4.13 JPEG DMA support

There are two instances of hardware in JPEG core are supported – a JPEG-Encoder and a JPEG-DMA. Each JPEG instance is controlled independently and has its own set of registers for control and hardware operation, and its own core clock. They can be treated as separate core hardware devices.

- JPEG-Encoder:  
This device supports offline mode Encoding of image data, where source and sink of data is external memory.
- JPEG-DMA: This device can be used in two modes
  - DMA Transfer mode can transfer image data in read/write burst length of maximum 256 bytes.
  - DMA Image Downscale mode can downscale image up to 128x.

Since there is direct memory access here, JPEG DMA is very less power consuming solution for downscaling and copy operations

JPEG DMA is part of the use case xml as a CamX node. Both Input and Output formats supported for DMA node are YUV NV12.



DMA node shall be inserted in use cases where we need to get downscaled buffers or memcpy buffers. If the downscale is already completed by the previous nodes of IPE etc, then DMA node will work as a simple DMA memcpy node, else the DMA node will perform the downscale needed.

#### 6.4.14 Camera driver

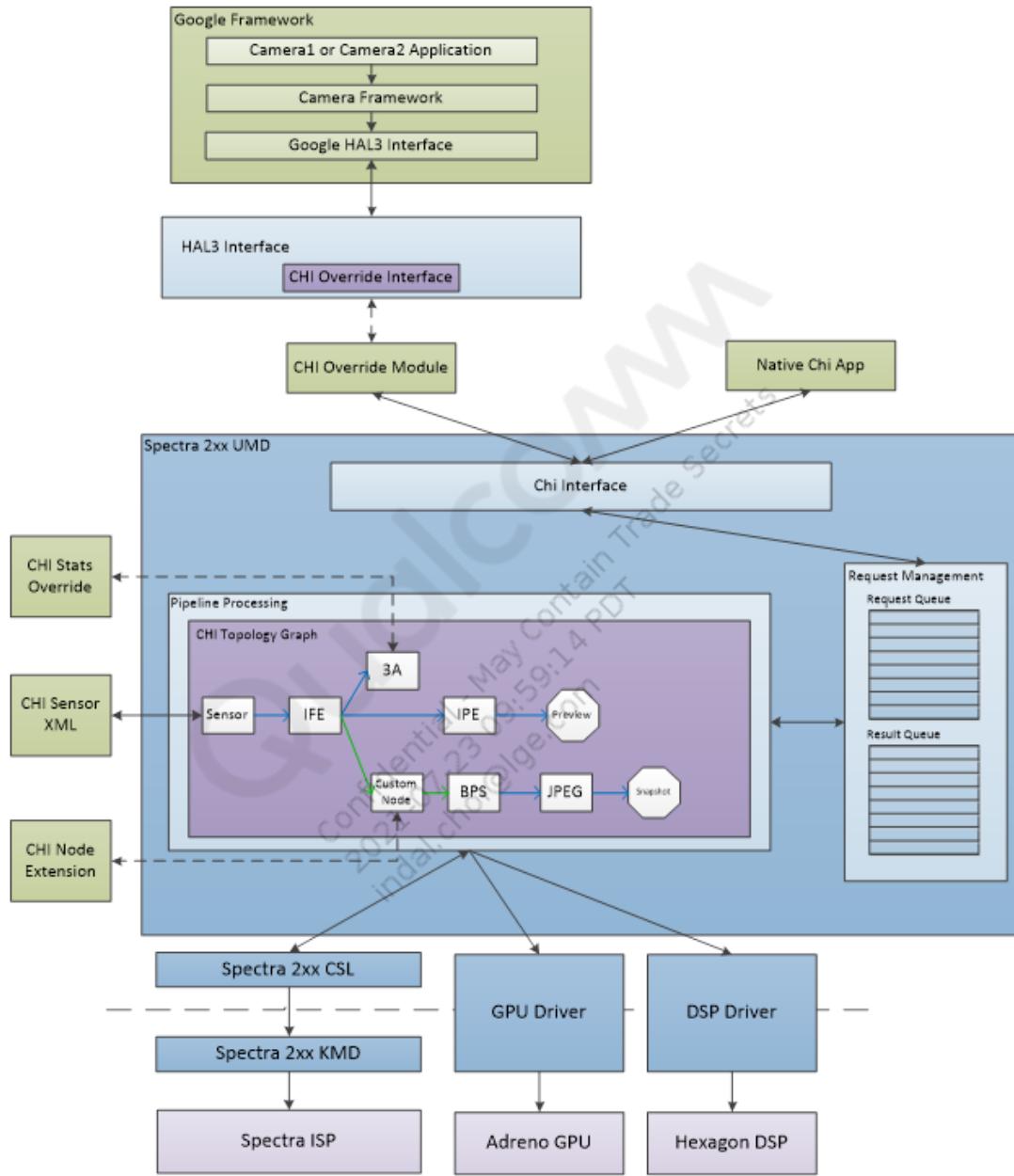
The CHI API aims to separate the camera2/HAL3 interface for using a camera, and supplement it with a fully flexible image processing driver. The CHI image processing driver is accessible as a standalone processing entity, with the ability to consume image data from either a sensor, or memory.

To allow Spectra to maintain backward-compatibility with the existing camera2/HAL3 interface, a thin HAL3 driver converts HAL3 calls into the appropriate CHI calls. The HAL3 driver is where all work to generate a camera use case is implemented; and the use case is implemented via CHI API calls.

An interface is provided that allows an OEM to modify the default Qualcomm Camera use case implementation. This modification allows OEMs to use a Camera2 or HAL3 application and still have access to the flexibility provided by CHI. These overrides are intended to allow an OEM to implement any type of image processing on any available engine, with low latency, and without the need to coordinate synchronization across multiple engines.

The CHI API builds upon the existing flexibility of the Google HAL3 Camera Interface. HAL3 is designed around explicit per-request control of the camera pipeline, to give full processing control to the end-user, but only at request boundaries. CHI is intended to give finer grained control, along with access to processing engines within the ISP. This interface allows OEMs and end-users to take

advantage of the low latency hardware and software paths that QTI supplies on the Snapdragon processor platforms for both camera and generic image processing applications.



**Figure 6-33 Qualcomm Spectra 2xx Camera Driver**

The Qualcomm Spectra camera driver has five key customizable components, which enable OEMs to take full advantage of CHI for camera applications:

- CHI override module supplements the Google HAL3 interface to allow for explicit image processing pipeline generation, explicit engine selection, and multiframe control for any HAL3-compliant camera application.
- CHI pipeline allows for an arbitrary compute pipeline to be constructed to process images. The pipeline can consist of fixed function ISP blocks (FF-ISP), provided by QTI, or extended nodes, which are controlled outside of the camera driver stack. For Camera2/HAL3 implementations, an

- XML helper file specifies the graph, which matches the existing Camera2/HAL3 use cases. Calls directly into CHI can programmatically generate pipelines as well.
- CHI node extensions are a further extension of CHI, which provide convenient hooks to streamline processing on the CPU, GPU (via OpenCL, OpenGL ES), or DSP (via OpenDSP, Qualcomm Computer Vision SDK, or custom programming). Custom nodes can specify the private vendor tags to be used by the application, and interact with CHI FF-ISP nodes or other extended nodes.
- CHI stats overrides, including 3A, allow mechanisms to override any of QTI default stats algorithms, without the need for driver changes. External stats algorithms can store private data, which is also accessible by custom nodes.
- CHI sensor XML allows device manufacturers to define parameter-driven drivers for their specific hardware components including the camera module, the image sensor, actuators, electronically erasable programmable read-only memory (EEPROM), and flash components.

## 6.5 Video

The QCS610/QCS410 chipset video core uses OpenMAX (OMX) integration layer (IL) to access the video hardware.

The following are the OMX IL video decoder/encoder components:

- OMX IL wrapper provides a standard interface for hardware encoder and decoder
- OMX IL video decoder/encoder is suitable for V4L2 interface
- OMX IL source code is located at *hardware/qcom/media/mm-video-v4l2/vidc/*

The following are the kernel video driver components:

- Video4Linux (second version) (V4L2) framework is a video API and framework for Linux. V4L2 handles events, callbacks, and buffer management.
- MSM\_vidc is a common driver for encoder and decoder. MSM\_vidc uses chipset-specific plugin for V4L3 framework. MSM\_vidc handles video engine, buffer allocation, and clocks/bus management.
- MSM\_vidc source code is located at *kernel/drivers/media/platform/msm/vidc/*
- HFI is the lowest software abstraction that interacts with video engine. HFI generates communication packets, manages shared queues, and handles interrupts.
- HFI source code is located at *kernel/drivers/media/platform/msm/vidc/*

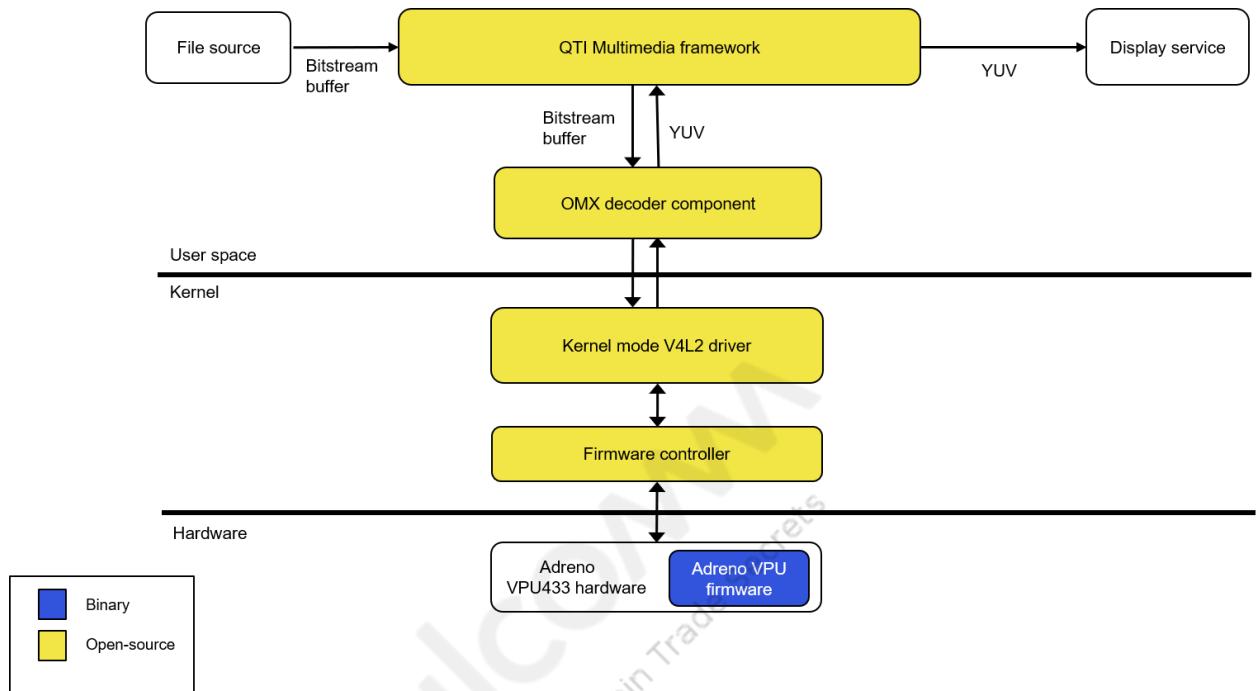


Figure 6-34 OpenMAX-based video decode – data flow

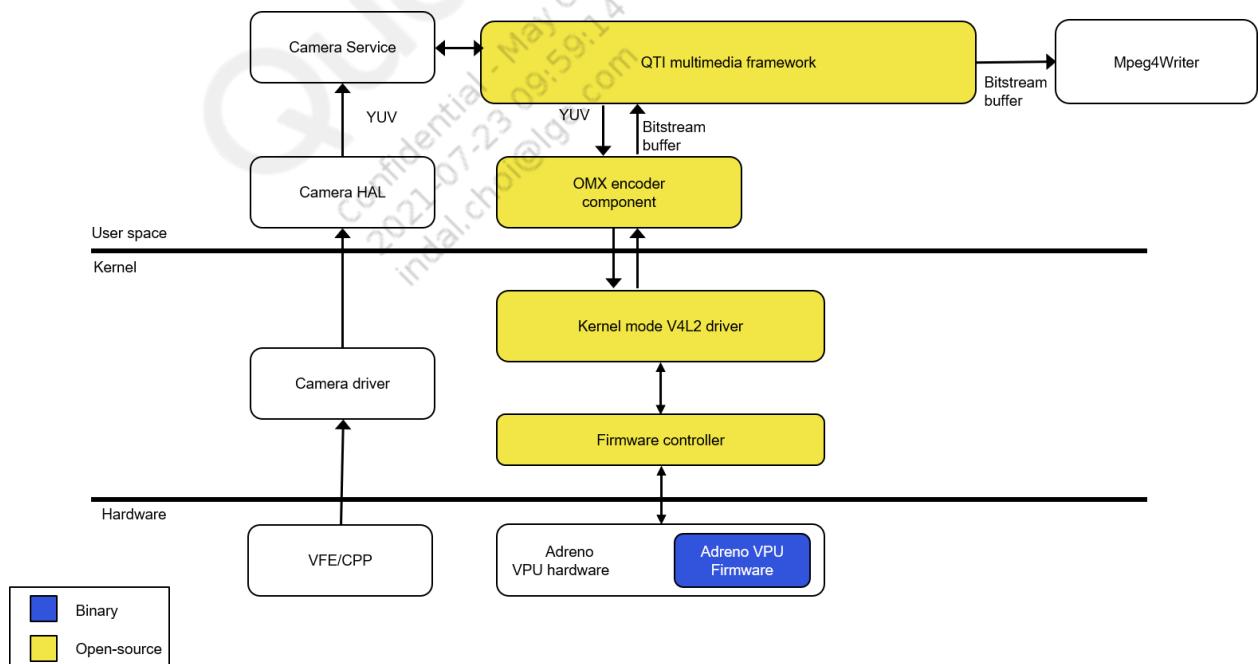


Figure 6-35 OpenMAX-based video encode – data flow

**Table 6-7 QCS610 video format**

Feature	Description
Video engine	Adreno VPU433
Decoder	H.264 BP/MP/HP MPEG-2 MP VP8 VP9 HEVC Main
Encoder	H.264 BP/MP/HP VP8 HEVC Main
Concurrent video sessions	16x
Maximum performance	Decoder: 3840 × 2160 at 30 fps 4096 × 2160 at 24 fps Encoder: 3840 × 2160 at 30 fps + 480 at 30 fps 4096 × 2160 at 24 fps

**Table 6-8 Adreno VPU433 codec specification for decoder**

Decoder standard	Supported profile and level	Minimum/maximum resolution, maximum frame rate, and maximum bit rate	Supported resolution, frame rate, bit rate	Limitations/tools not supported
HEVC	Main profile 8-bit, up to level 5.1	<ul style="list-style-type: none"> <li>▪ Minimum resolution: 96 × 96</li> <li>▪ Maximum resolution: 4096 × 2160 or 2160 × 4096</li> <li>▪ Maximum fps: 240</li> <li>▪ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ 1280 × 720 at 240 fps, 100 Mbps</li> <li>▪ 1920 × 1088 at 120 fps, 100 Mbps</li> <li>▪ 3840 × 2160 at 30 fps, 100 Mbps</li> <li>▪ 4096 × 2160 at 24 fps, 100 Mbps</li> </ul>	Maximum 10 slices per frame
H.264	Constrained baseline, baseline, main, high, constrained high profiles; up to level 5.1	<ul style="list-style-type: none"> <li>▪ Minimum resolution: 96 × 96</li> <li>▪ Maximum resolution: 3840 × 2160 or 4096 × 2160</li> <li>▪ Maximum fps: 240</li> <li>▪ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ 1280 × 720 at 240 fps, 100 Mbps</li> <li>▪ 1920 × 1088 at 120 fps, 60 Mbps</li> <li>▪ 3840 × 2160 at 30 fps, 100 Mbps</li> <li>▪ 4096 × 2160 at 24 fps, 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ Flexible macroblock order (FMO)</li> <li>▪ Arbitrary slice ordering (ASO)</li> <li>▪ Redundant slices (RS)</li> <li>▪ Data partition</li> <li>▪ Maximum 10 slices per frame</li> <li>▪ Interlaced content up to 1920 × 1088</li> </ul>

**Table 6-8 Adreno VPU433 codec specification for decoder (cont.)**

<b>Decoder standard</b>	<b>Supported profile and level</b>	<b>Minimum/maximum resolution, maximum frame rate, and maximum bit rate</b>	<b>Supported resolution, frame rate, bit rate</b>	<b>Limitations/tools not supported</b>
MPEG-2	Main profile	<ul style="list-style-type: none"> <li>▪ Minimum resolution: <math>96 \times 96</math></li> <li>▪ Maximum resolution: <math>1920 \times 1088</math></li> <li>▪ Maximum fps: 30</li> <li>▪ Maximum bit rate: 40 Mbps</li> </ul>	$1920 \times 1088$ at 30 fps, 40 Mbps	<ul style="list-style-type: none"> <li>▪ Multiresolution processing inside video decoder</li> <li>▪ Range mapping (information is returned as meta data for external postprocessing)</li> </ul>
VP8	Profile 0 (Main), version 0, version 1, version 2, and version 3	<ul style="list-style-type: none"> <li>▪ Minimum resolution: <math>96 \times 96</math></li> <li>▪ Maximum resolution: <math>3840 \times 2160</math> or <math>2160 \times 3840</math></li> <li>▪ Maximum fps: 120</li> <li>▪ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ <math>1280 \times 720</math> at 120 fps, 100 Mbps</li> <li>▪ <math>1920 \times 1088</math> at 60 fps, 20 Mbps</li> <li>▪ <math>3840 \times 2160</math> at 30 fps, 100 Mbps</li> </ul>	None
VP9	Profile 0; 8-bit, up to level 5.1	<ul style="list-style-type: none"> <li>▪ Minimum resolution: <math>96 \times 96</math></li> <li>▪ Maximum resolution: <math>4096 \times 2160</math> or <math>2160 \times 4096</math></li> <li>▪ Maximum fps: 240</li> <li>▪ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ <math>1280 \times 720</math> at 240 fps, 100 Mbps</li> <li>▪ <math>1920 \times 1088</math> at 120 fps, 100 Mbps</li> <li>▪ <math>3840 \times 2160</math> at 30 fps, 100 Mbps</li> <li>▪ <math>4096 \times 2160</math> at 24 fps, 100 Mbps</li> </ul>	None

**NOTE** The level indicates the capacity to parse a bit stream encoded at this level. The maximum supported bit rate is indicated under supported resolution, frame rate, and bit rate column.

**Table 6-9 Encoder specification**

<b>Encoder standard</b>	<b>Supported profile and level</b>	<b>Minimum/Maximum resolution, maximum frame rate, and maximum bit rate</b>	<b>Supported resolution, frame rate, bit rate</b>	<b>Limitations/tools not supported</b>
H.264	Baseline, main, high profiles; up to level 5.1	<ul style="list-style-type: none"> <li>▪ Minimum resolution: <math>96 \times 96</math></li> <li>▪ Maximum resolution: <math>4096 \times 2160</math></li> <li>▪ <math>2160 \times 4096</math></li> <li>▪ Maximum fps: 240</li> <li>▪ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ <math>1280 \times 720</math> at 240 fps, 100 Mbps</li> <li>▪ <math>1920 \times 1088</math> at 120 fps, 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>▪ Interlaced tools</li> <li>▪ More than two multiple-reference frames</li> <li>▪ Weighted prediction</li> </ul>

**Table 6-9 Encoder specification (cont.)**

Encoder standard	Supported profile and level	Minimum/Maximum resolution, maximum frame rate, and maximum bit rate	Supported resolution, frame rate, bit rate	Limitations/tools not supported
			<ul style="list-style-type: none"> <li>■ 3840 × 2160 at 30 fps, 100 Mbps</li> <li>■ 4096 × 2160 at 24 fps, 100 Mbps</li> </ul>	
HEVC	Main profile 8-bit, up to level 5.1	<ul style="list-style-type: none"> <li>■ Minimum resolution: 96 × 96</li> <li>■ Maximum resolution: 4096 × 2160 or 2160 × 4096</li> <li>■ Maximum fps: 240</li> <li>■ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>■ 1280 × 720 at 240 fps, 100 Mbps</li> <li>■ 1920 × 1088 at 120 fps, 100 Mbps</li> <li>■ 3840 × 2160 at 30 fps, 100 Mbps</li> <li>■ 4096 × 2160 at 24 fps, 100 Mbps</li> </ul>	None
VP8	Profile 0 (main), version 0, version 1, version 2, and version 3	<ul style="list-style-type: none"> <li>■ Minimum resolution: 96 × 96</li> <li>■ Maximum resolution: 3840 × 2160 or 2160 × 3840</li> <li>■ Maximum fps: 30</li> <li>■ Maximum bit rate: 100 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>■ 3840 × 2160 at 30 fps, 100 Mbps</li> <li>■ 1920 × 1088 at 30 fps, 20 Mbps</li> </ul>	<ul style="list-style-type: none"> <li>■ MB level change of QP</li> <li>■ DB parameters</li> <li>■ MB pitch</li> </ul>

The codec configuration performances mentioned in the table are currently being validated. The performance might increase during the final release.

### 6.5.1 Video features

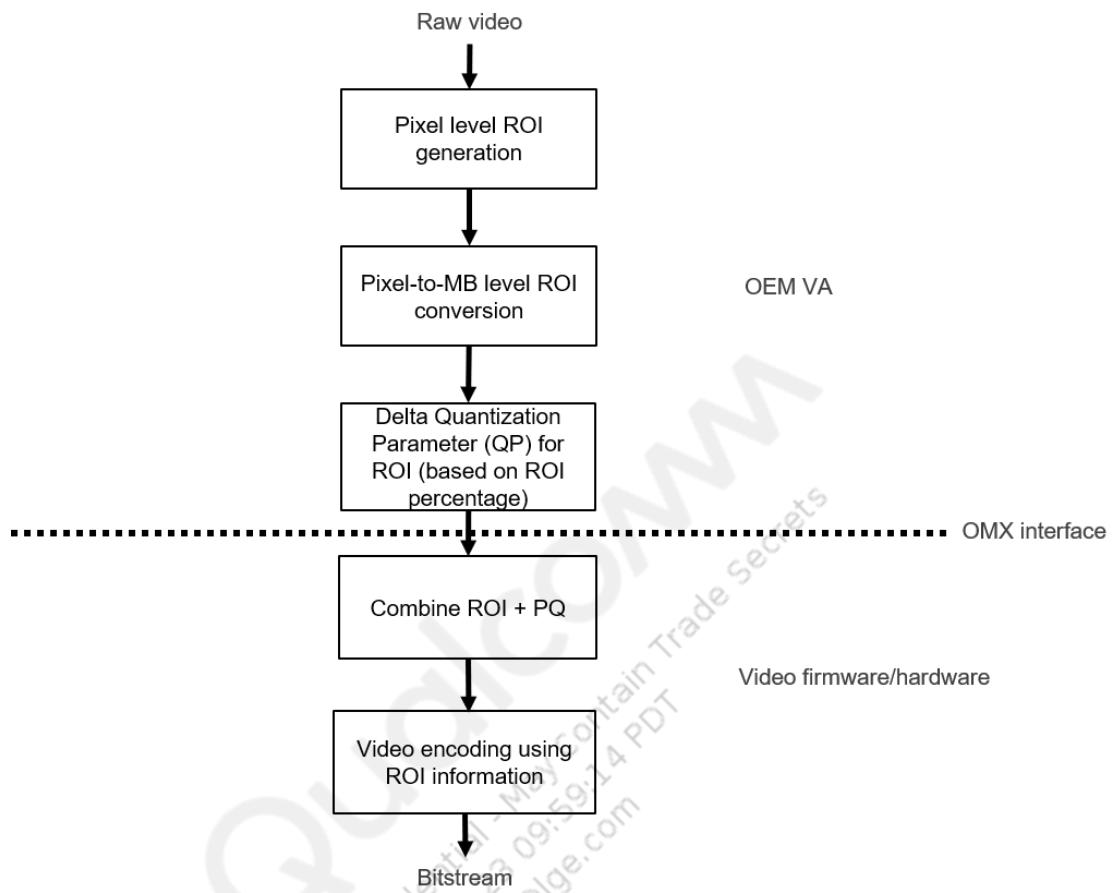
#### Region of interest video encoding

The region of interest (ROI) is similar to OMX\_CONFIG\_FOCUSREGIONTYPE, which defines nine focus measurement regions—center, left, right, top, bottom, top left, top right, bottom left, bottom right. ROI allows the definition of the nine focus measurement regions to be more discrete, and potentially overlapping, prioritized regions of interests.

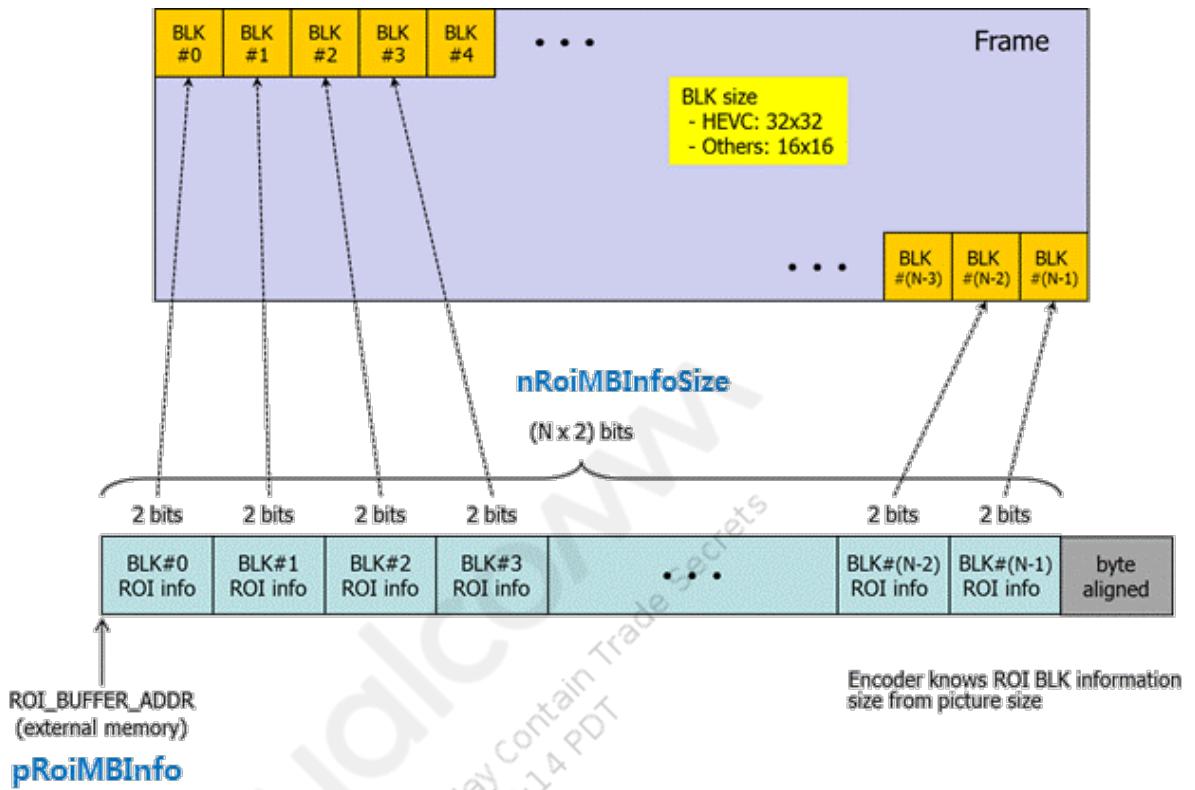
The ROI bounding box is generated by the client application that is invoking the OMX API.

The APIs need to be defined on VA-generated ROI before passing it to the video encoder. Multiple ROIs can be simultaneously supported by the video encoder. The video encoder would prefer higher quality for the ROI at the expense of non-ROI region. The ROI quality improvement/non-ROI quality degradation depends on the size of ROI. Rate-control maintains the bit rate restrictions.

**NOTE** Currently, ROI encoding is at a prototype/simulation stage. Further updates will be provided during the CS timeframe.



**Figure 6-36 ROI video encoding**



**Figure 6-37 QP map programming for a frame**

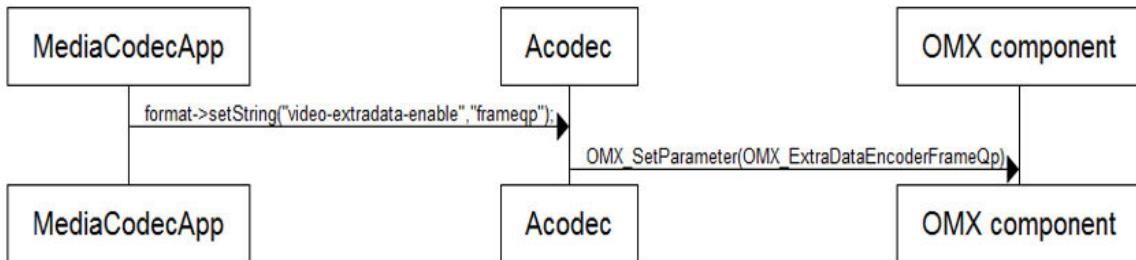
The following sequence describes the QP map programming:

1. The frame average QP is implemented as video extra data. The extra data output is generated at `FillBufferDone`. A parsing logic extracts the average QP information of encoded frame.
2. The supported extra data is OpenMAX compliant. This is propagated as an additional payload information with each output buffer using the `OMX_OTHER_EXTRADATATYPE` structure.
3. To access the extra data information from the MediaCodec client, the video-extradata-enable key has to be set as `frameqp` in the format object passed at `MediaCodec::configure()` API. The OMX framework allocates the extra data buffer and sets the OMX component as shown in the following code snippet:

```
format->setString("video-extradata-enable","frameqp");
```

The following code snippet shows how to enable frame QP output extra data in OMX IL:

```
QOMX_INDEXEXTRADATATYPE extra_data; // OMX_QcomIndexParamIndexExtraDataType
OMX_INIT_STRUCT(&extra_data, QOMX_INDEXEXTRADATATYPE);
extra_data.nPortIndex = (OMX_U32)PORT_INDEX_OUT;
extra_data.nIndex = OMX_ExtraDataEncoderFrameQp;
extra_data.bEnabled = OMX_TRUE;
OMX_SetParameter(m_hEncoder,
(OMX_INDEXTYPE)OMX_QcomIndexParamFrameInfoExtraData,
(OMX_PTR)&extra_data);
```



**Figure 6-38 Output extra data enable sequence**

### Error resiliency

The error resiliency feature enables to use streaming/video chat/videotelephony applications. It helps in reducing the loss of video quality caused by network congestion factors such as frame corruption caused by packet loss, or jitter and jerkiness caused by irregular and regular frame drops.

**Table 6-10 Error resiliency features**

Features	Support information
Intra refresh/gradual decoder refresh (GDR)	Cyclic/random/GDR are supported
HP/SVC temporal	Up to 6-layers are currently supported, framework can support up to 15 layers
LTR frames	Up to 4 LTR frames, manual/periodic/automatic marking
HP with LTR	HP with base layer LTR support
Dynamic HP layers and/or slice sizes	Ability to react to changing network conditions Supports dynamic HP layers with rate control enabled

The error resiliency feature on encoder for video chat-based applications is supported as a static configuration (enabled per session). The IL client specifies the period to refresh or the number of MBs per frame to be refreshed.

The encoder chooses the order in which the MBs are randomly refreshed. It also ensures that it covers all the MBs within N number of frames, which is enabled by setting constrained\_intra to true.

The IR helps to reduce channel loss for cases like streaming or casting applications that favor constant bit rate.

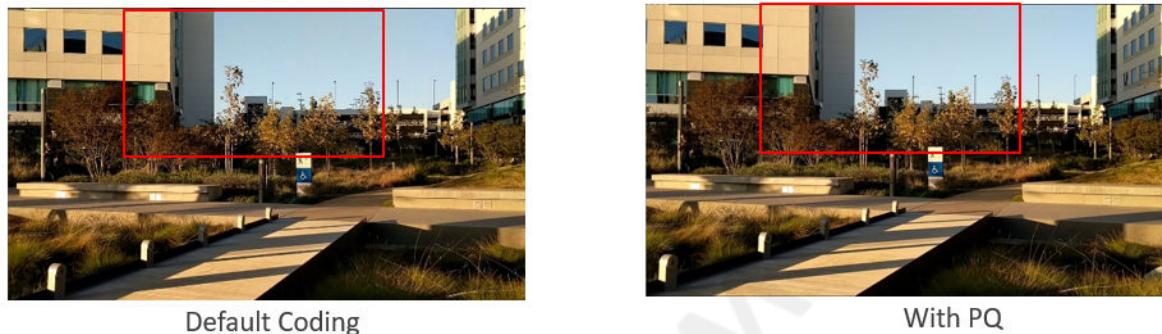
### Perceptual quality (PQ)

The PQ feature improves the video quality through smarter allocation of bits based on scene analysis, without impacting memory and performance. Previously, video encoder was designed to generate the best quality video for each frame. Spatial video quality was assumed to be similar for an image, however, users weigh differently on different part of the images.

PQ uses human visual system model and allocates more bits to regions where visual perception is high. For example, flat regions like sky.

It allocates fewer bits to regions where visual perception is low. For example, fast moving objects or high textured regions like trees.

The algorithm used by PQ processes the input YUV to calculate the block level statistics for video encoder.



**Figure 6-39 Visual quality difference with PQ**

### Rate control

The following rate control features are enhanced for codecs:

- Variable bit rate (VBR): Bit rate within 10% with low-quality fluctuation (consider average bitrate at the end of stream)
- Constant bit rate (CBR): Bit rate within 5% and 500 ms convergence time (consider average bitrate at the end of stream, 500 ms convergence time is an ideal condition and can only be achieved in case of MP4 or YUV file as source)
- Maximum bit rate: Streaming use case with low delay, content dependent peak bit rate, not overshooting above the target bit rate
- Peak constrained RC peak bit rate limited to as low as 1.5x target bit rate
- Dynamic bit rate, frame rate, IDR insertion: Fast convergence tied to RC mode (For example: 500 ms for CBR)
- Initial delay/Buffer size: Allows user to control latency – as low as 100 ms
- Slice-based encoding (MB-based): Support for up to 10 slices
- Hierarchical B-frames for HEVC: Up to 4-layers (default is 3-layers)

The following are the requirements for different video use cases:

- Camcorder
- Steady quality with limited overshoot
- Videotelephony – Steady bit rate with quick convergence
- Low end-to-end delay
- Ability to adapt quickly during session
- IP Camera streaming
- Low latency
- High quality for near static videos

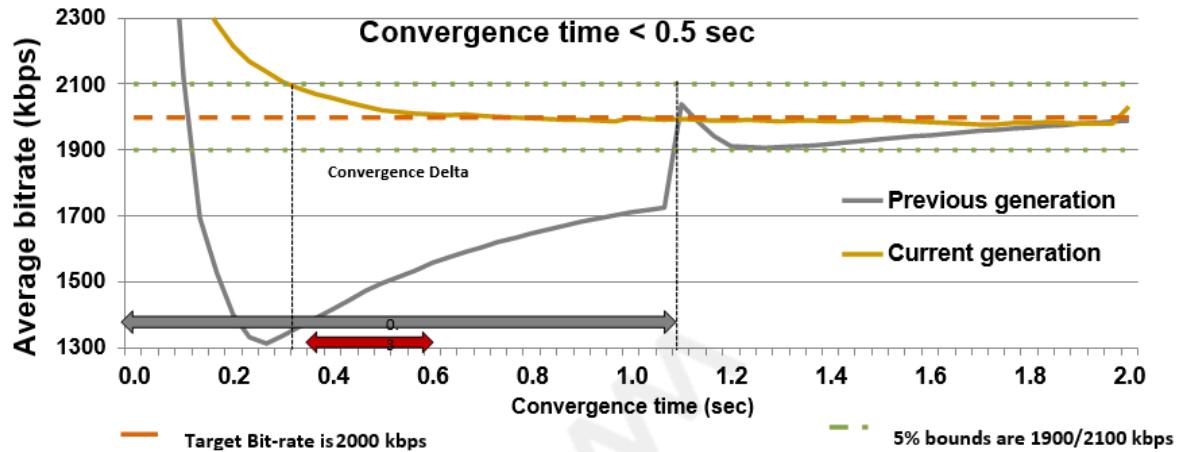


Figure 6-40 Rate control convergence

### Other video features

Features	QCS610
Encoder native input color format	<ul style="list-style-type: none"> <li>■ Venus_NV12/NV21</li> <li>■ Venus_NV12_UBWC (default)</li> <li>■ RGBA8888</li> <li>■ UBWC_RGBA8888</li> </ul>
Decoder native output color format	<ul style="list-style-type: none"> <li>■ Venus_NV12/NV21</li> <li>■ Venus_NV12_UBWC (default)</li> </ul>
Multichannel support	Up to 16 sessions
Perceptual quantization	Yes (part of video core)
HFR	<ul style="list-style-type: none"> <li>■ <math>1920 \times 1088</math> at 120 fps</li> <li>■ <math>1280 \times 720</math> at 240 fps</li> </ul>
Widevine secure playback	Yes
Snapdragon UBWC support	UBWC version 2.0

### 6.5.2 Video concurrency

Table 6-11 Multi-instance decode

Use case	Capability
Single instance playback at UHD resolution	$3840 \times 2160$ at 30 fps
Maximum number of playback instances of HD and FHD resolutions	
Maximum number of playback instances in high definition (HD) and full high definition (FHD) resolutions	$8 \times 1280 \times 720$ at 30 fps $4 \times 1920 \times 1088$ at 30 fps

**Table 6-12 HFR encode**

QCS610	Capability
High Frame Rate (HFR) camcorder for H.264/HEVC	$1920 \times 1088$ at 120 fps
	$1280 \times 720$ at 240 fps

**NOTE** The validated use cases mentioned might not include all supported concurrencies. If there are different scenarios, contact the QTI support team to confirm if these scenarios are supported.

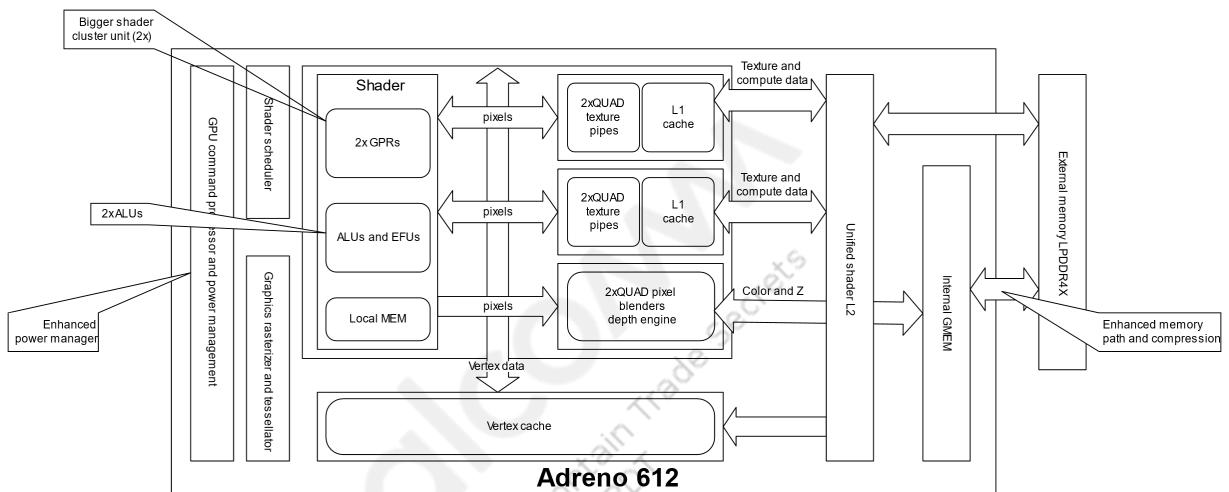
All the preprocessing algorithms run on the VPU. The table provides the video encoder preprocessing configuration.

**Table 6-13 Video encoder preprocessing**

Algorithm	Configuration	Codecs	Color formats
Downscale	Up to 1/8 scaling ratio	<ul style="list-style-type: none"> <li>■ H.264</li> <li>■ HEVC</li> <li>■ VP8</li> </ul>	<ul style="list-style-type: none"> <li>■ NV12</li> <li>■ NV21</li> <li>■ NV12_UBWC</li> </ul>
Rotation	90/180/270 degree rotation	<ul style="list-style-type: none"> <li>■ H.264</li> <li>■ HEVC</li> <li>■ VP8</li> </ul>	<ul style="list-style-type: none"> <li>■ NV12</li> <li>■ NV21</li> <li>■ NV12_UBWC</li> </ul>
Color space conversion (CSC)	ITU Rec 601 to Rec 709 conversion	<ul style="list-style-type: none"> <li>■ H.264</li> <li>■ HEVC</li> <li>■ VP8</li> </ul>	<ul style="list-style-type: none"> <li>■ NV12</li> <li>■ NV21</li> <li>■ NV12_UBWC</li> </ul>
Blur effect	–	<ul style="list-style-type: none"> <li>■ H.264</li> <li>■ HEVC</li> </ul>	<ul style="list-style-type: none"> <li>■ NV12</li> <li>■ NV21</li> <li>■ NV12_UBWC</li> </ul>
Flip	Horizontal flip/Vertical flip	<ul style="list-style-type: none"> <li>■ H.264</li> <li>■ HEVC</li> <li>■ VP8</li> </ul>	<ul style="list-style-type: none"> <li>■ NV12</li> <li>■ NV21</li> <li>■ NV12_UBWC</li> </ul>

## 6.6 Graphics

The Adreno 612 GPU is three-dimensional (3D) graphics accelerator with 64-bit addressing. The GPU includes dedicated GMEM for the graphics subsystem for Fast Z, color, and stencil rendering. The GPU supports UBWC 2.0.



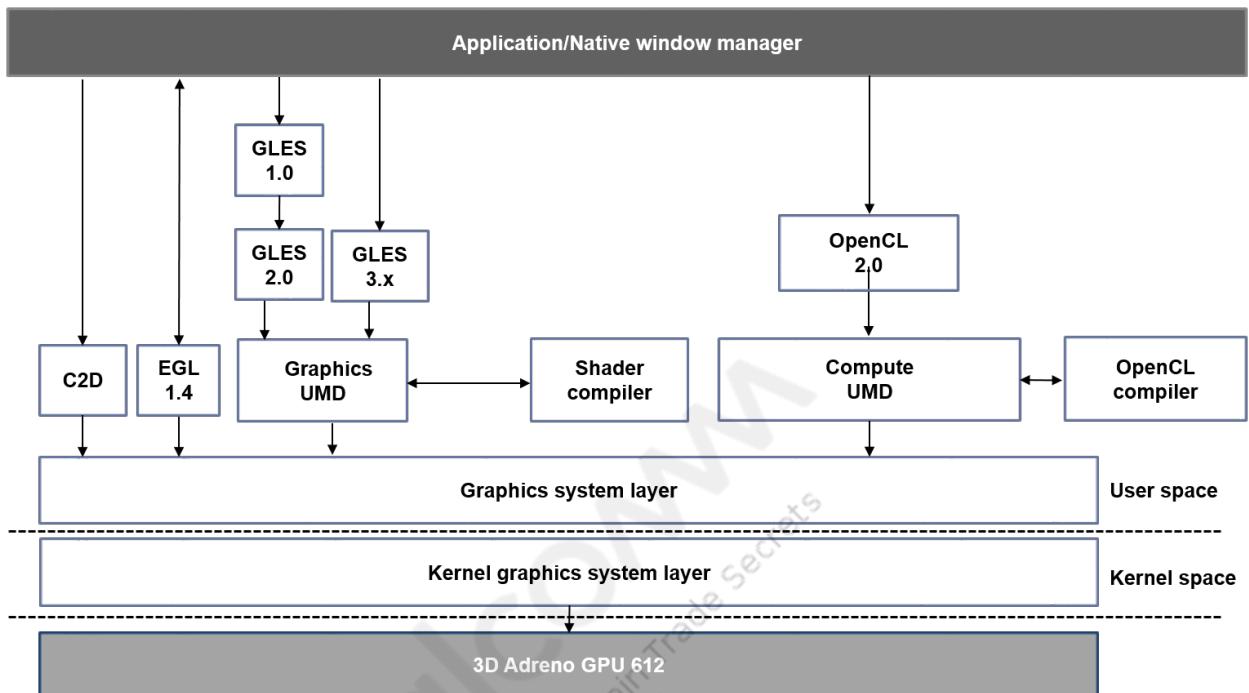
**Figure 6-41 Adreno 612 block diagram**

Adreno 612 GPU architecture:

- Delivers latest in Graphics and compute APIs
  - OpenGL ES 3.2
  - OpenCL 2.0 FP

C2D Adreno 612 – High-level overview

- Architectural improvements for performance and power
  - More efficient, wider shader micro-architecture
  - Significant resource (ALU) increase in shader processors
  - Increased texture performance
- Memory bandwidth reduction and efficiency improvements
  - 2D rasterization patterns
  - Improved hidden surface removal to reduce work
  - Better lossless compression to reduce memory bandwidth and power

**Figure 6-42 Adreno 612 driver architecture****Table 6-14 Closed-source Adreno graphics libraries**

Library	Description
libEGL_adreno.so	EGL driver
libGLESv1_CM_adreno.so	OpenGL ES 1.1 driver
libGLESv2_adreno.so	OpenGL ES 2.0/3.2 driver
libeglSubDriverWayland	EGL subdriver
libadreno_utils.so	Utility library for the driver
libgsl.so	GSL library
libC2D2.so	C2D wrapper on C2D30 – determines current chipset
libc2d30.so	C2D30 driver (logic) – loads appropriate Hardware abstraction layer (HAL)
libc2d30-bltlib.so	Blit library
libOpenCL.so	OpenCL library
libCB.so	Compute Boy library
libl LLVM-qcom.so	CL compiler library
Libl LLVM-glnext.so	Shader compiler library

### Kernel mode driver

- Kernel graphics system layer (KGSL) driver
  - 64-bit compatible device driver for QCS610
  - Available in the build at *kernel/drivers/gpu/msm/*
  - Change in the Linux kernel version of QCS610 to 4.14

## 6.6.1 Graphics features

The following are the graphics hardware features:

- UBWC 2.0 for reduced bandwidth
- Support P010, TP10, NV12-4R compressed, MIPI packed Raw, packed Bayer and unpacked Bayer formats

**Table 6-15 QCS610 Adreno graphics key specifications**

Component	QCS610				
Graphics Processing Unit (GPU)	Adreno 612				
GPU clock	845 MHz				
On-Chip memory (GMEM)	132 KB				
Shader processor	One enhanced shader processor for higher efficiency with double the number of ALUs compared to Adreno 512				
Power management	GPMU is available				
Reduced bandwidth requirement	UBWC 2.0				

RGB									
	A	I	X	R	G	B	Per pixel		
Linear (Uncompressed)									
Interleaved RGB								In	Out
RGB565	0	5	6	5	16			✓	✓
RGB888	0	8	8	8	24			✓	✓
ARGB8888	8	8	8	8	32			✓	✓
RGBA8888	8	8	8	8	32			✓	✓
XRGB8888	8	8	8	8	32			✓	✓
RGBX8888	8	8	8	8	32			✓	✓
ARGB1555	1	5	5	5	16			✓	✓
RGBA5551	1	5	5	5	16			✓	✓
XRGB1555	1	5	5	5	16			✓	✓
RGBX5551	1	5	5	5	16			✓	✓
ARGB4444	4	4	4	4	16			✓	✓
RGBA4444	4	4	4	4	16			✓	✓
RGBX4444	4	4	4	4	16			✓	✓
XRGB4444	4	4	4	4	16			✓	✓
ARGB2 10 10 10	2	10	10	10	32			✓	✓
XRGB2 10 10 10	2	10	10	10	32			✓	✓
RGBA10 10 10 2	2	10	10	10	32			✓	✓
RGBX10 10 10 2	2	10	10	10	32			✓	✓

RGB									
	A	I	X	R	G	B	Per pixel		
Linear (Uncompressed)									
Interleaved RGB							In	Out	
RGB16 16 16	0	16	16	16		48	✓	□	
ARGB16 16 16 16	16	16	16	16		64	✓	✓	
RGBA16 16 16 16	16	16	16	16		64	✓	✓	
ARGB16 16 16 16 float	16	16	16	16		64	✓	✓	
RGBA16 16 16 16 float	16	16	16	16		64	✓	✓	
RGB 32 32 32	-	32	32	32		96	✓	□	
RGBA32 32 32 32	32	32	32	32		128	✓	✓	
BGR565	0	5	6	5		16	✓	✓	
BGR888	0	8	8	8		24	✓	✓	
ABGR8888	8	8	8	8		32	✓	✓	
BGRA8888	8	8	8	8		32	✓	✓	
BGRX8888	8	8	8	8		32	✓	✓	
XBGR8888	8	8	8	8		32	✓	✓	
ABGR1555	1	5	5	5		16	✓	✓	
BGRA5551	1	5	5	5		16	✓	✓	
XBGR1555	1	5	5	5		16	✓	✓	
BGRX5551	1	5	5	5		16	✓	✓	
ABGR4444	4	4	4	4		16	✓	✓	
BGRA4444	4	4	4	4		16	✓	✓	
BGRX4444	4	4	4	4		16	✓	✓	
XBGR4444	4	4	4	4		16	✓	✓	
XBGR2 10 10 10	2	10	10	10		32	✓	✓	
BGRA10 10 10 2	2	10	10	10		32	✓	✓	
BGRX10 10 10 2	2	10	10	10		32	✓	✓	
BGR16 16 16	0	16	16	16		48	✓	□	
ABGR16 16 16 16	16	16	16	16		64	✓	✓	
BGRA16 16 16 16	16	16	16	16		64	✓	✓	
ABGR16 16 16 16 Float	16	16	16	16		64	✓	✓	
BGRA16 16 16 16 Float	16	16	16	16		64	✓	✓	
ABGR 32 32 32	-	32	32	32		96	✓	□	
BGRA32 32 32 32	32	32	32	32		128	✓	✓	
<b>UBWC 1.x\2.0 Lossless</b>									
RGBA8888	8	8	8	8	-		✓	✓	
RGBA10 10 10 2	2	10	10	10	-		✓	✓	

RGB							
	A\X	R	G	B	Per pixel		
Linear (Uncompressed)							
Interleaved RGB						In	Out
RGBA16 16 16 16	16	16	16	16	64	✓	✓
RGB565	0	5	6	6	-	✓	✓
BGRX4444	4	4	4	4	16	✓	✓
<b>YUV</b>							
Bits per component							
	A	Y	U	V	Average per pixel		
Linear (Uncompressed)							
YUV420							
YUV planar	0	8	8	8	12	✓	✓
YYU planar	0	8	8	8	12	✓	✓
YUV semi-planar	0	8	8	8	12	✓	✓
YVU semi-planar	0	8	8	8	12	✓	✓
YUV semi-planar 10-bit	0	10	10	10	-	✓	✓
YUV tightly packed 10-bit	0	10	10	10	-	✓	□
Y - Luma Only	0	8	0	0	-	✓	□
YUV422							
YUYV interleaved	0	8	8	8	16	✓	✓
YYVU interleaved	0	8	8	8	16	✓	✓
YUYV interleaved	0	8	8	8	16	✓	✓
UYVY interleaved	0	8	8	8	16	✓	✓
Compressed							
UBWC 1.x\2.0 lossless							
YUV420 8-bit	0	8	8	8		✓	✓
YUV420 10-bit	0	10	10	10		✓	✓
YUV420 semi-planar 8-bit	0	8	8	8		✓	✓
YUV420 semi-planar 10-bit	0	10	10	10		✓	✓
Bayer							
Linear (Uncompressed)							
MIPI RAW 10-bit						✓	□
MIPI RAW 12-bit						✓	□

**NOTE** Support 10-bit out by CL.

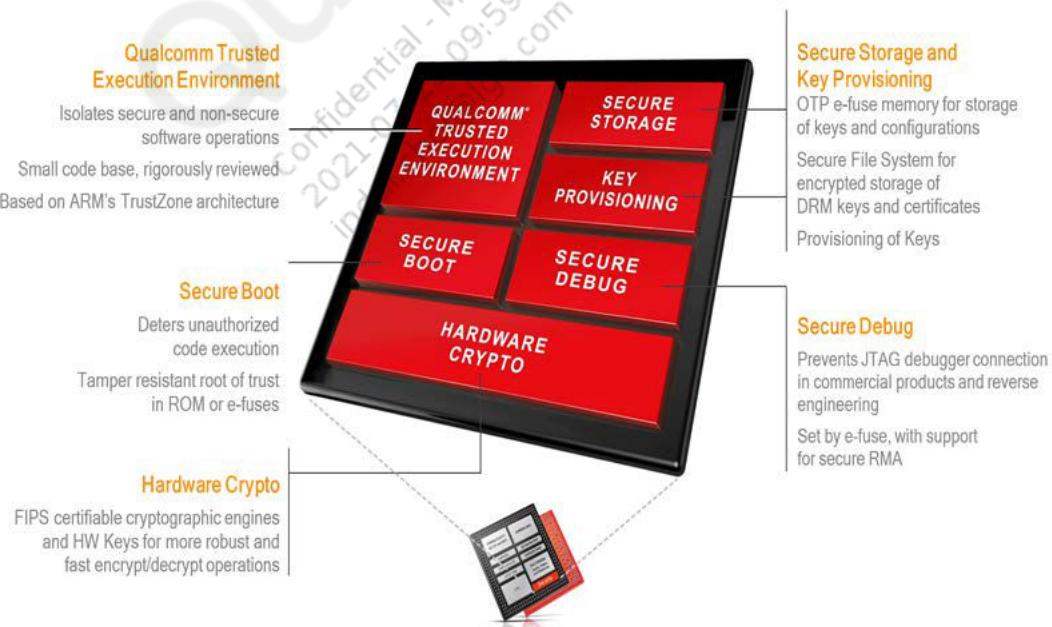
## 6.7 Security

The QTI product security comprises of a security feature set, which provides a security framework that enables a device to counter security threats from both a software and hardware side.

TZ is the core of product security and it facilitates secure execution environment for most of the product security features. TZ is built on Arm core TZ technology and relies on Secure Execution mode of Arm core.

The product security feature set majorly comprises following security components:

- TZ and secure application
- Qualcomm TEE 5.0 overview
- Secure boot and QFPROM
- HLOS security
- Device access control
- Crypto core and software crypto modules
- Secure storage
- Secure device debugging



**Figure 6-43 Product security overview**

The chipset provides a comprehensive and robust, all round security solution which comprises major security building blocks such as – TZ, Qualcomm TEE 5.0, secure boot, crypto cores, secure storage, access control measures and also facilitates method for secure device debugging.

## TZ

The TZ is the core of the chipset's product security and provides the necessary framework for implementation and execution of these security features, such as secure boot, HLOS security, access control, and so on. TZ is the secure OS which executes in secure mode of Arm core and separates the non-secure world (HLOS) from secure world (TZ).

### Secure boot

The secure boot ensure that device bootup with only authorized software and protects the bootup sequence against any attempt of bringing in malicious software. For more information, see *QCS610 Secure Boot Enablement User Guide* (80-PL631-42).

### HLOS Security

The HLOS security modules in the chipset ensure the safe loading and execution of HLOS components. The chipset has Verified Boot, device mapper verity (dm-verity), Security-Enhanced Linux (SELinux), kernel module signing and so on to ensure HLOS security.

### Crypto modules

The chipset supports both software and hardware crypto modules, which are available in both HLOS and TZ, to support all crypto use cases.

### Secure storage

The chipset provides secure storage for safeguarding any data/keys, which need to be kept protected, such as provisioning keys (Keymaster and so on) and trusted application data (Fingerprint, Widevine, and so on).

### Hardware support

The chipset has hardware support (xPU) for access control which is used to protect range of memory resources (register, memory regions, GPIO, and so on) against unauthorized access.

### Debugging

The chipset has provision for debugging secure devices in a controlled environment, to access this, QTI provides debug features such as validated image programming (VIP) and debug policy.

### Overall product security

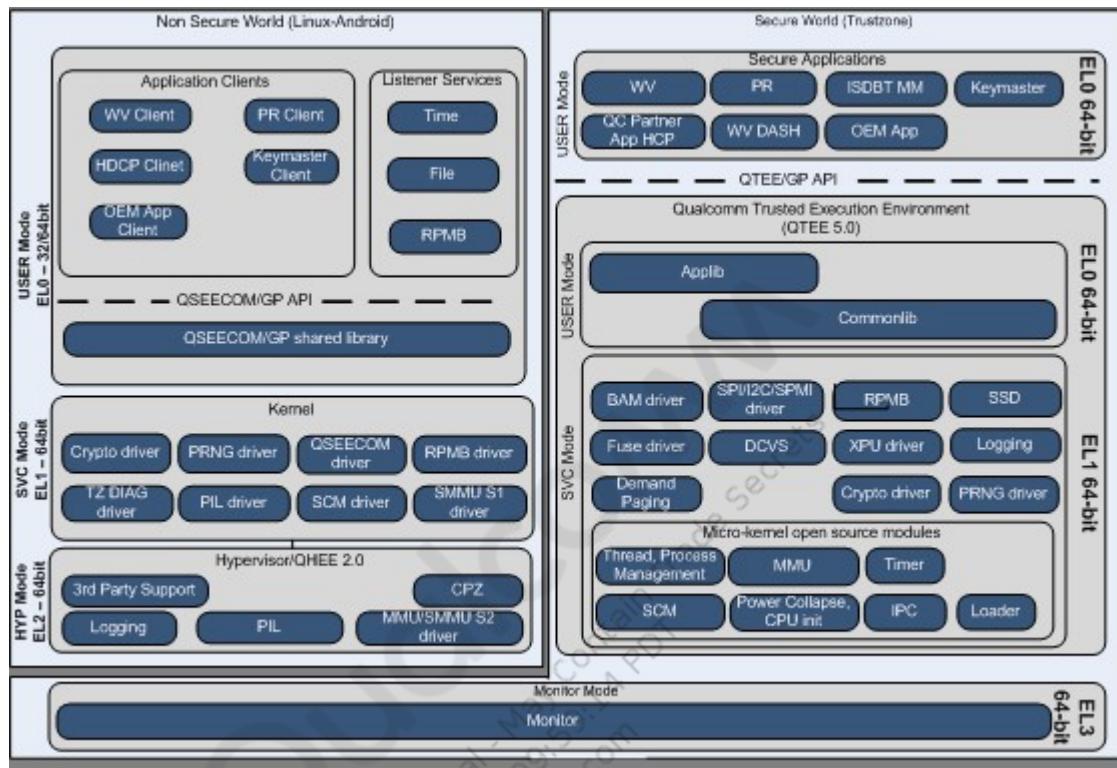
The chipset has overall product security in-place, which enforces security measures from the boot to the secure and controlled debugging and protects it from the malicious attacks.

## 6.7.1 TZ and secure application

The chipset has 64-bit Arm, v8-A processor system in which Arm cores have two modes of execution:

- Non-Secure mode – HLOS executes in this mode of Arm core
- Secure mode – Secure OS (TZ) executes in secure mode of Arm core.

The Secure mode of Arm core is the core of TZ Technology which provides a hardware-based security environment for secure OS and separates secure world from non-secure world.



**Figure 6-44 Qualcomm TEE v5.0 security framework architecture**

TZ comprises of following major components:

- Trusted application space
- TZ board support package (TZBSP)
- Secure Monitor
- Device Configuration
- Qualcomm TEE 5.0

### Trusted application space

Trusted applications are run in this space. The TZ gives freedom to the OEM to develop and execute their own trusted application in TZ user space for any secure use case.

**NOTE**    QTI provides sample application code to help the OEM develop their own trusted application.

## TZBSP

The TZBSP provides software support for the chipset security. It exposes hardware abstraction layer (HAL) APIs for the chipset security functions, such as, Crypto, fuse block, pseudo random number generator (PRNG).

- It initializes system security environment for software and hardware during bootup and wake-up from power collapse.
- It provides memory, other subsystem protection, and services during runtime.

## Qualcomm TEE 5.0

The Qualcomm TEE provides security services, such as, image loading, authentication, cache management, crypto, logging, and Qualcomm fuse-programmable read-only memory (QFPROM) to TZ secure applications.

The XBL loads the TZ software image during the initial device bootup process.

## Device Configuration

There are some platform-dependent customizations possible which is configurable via device configuration (devcfg) files. OEMs can do the configuration changes and build devcfg.mbn image with new customizations.

## TZ functionalities

Major TZ functions are as follows:

- Cold boot:
  - a. The boot sequence starts from a secure root of trust (application processor PBL), the boot process begins in on-chip ROM (application processor PBL).
  - b. Signed software images (including XBL and TZ) from flash memory are loaded into DDR and authenticated before execution.
  - c. Application processor PBL loads and authenticates XBL, XBL loads and authenticates TZ.
  - d. TZ programs different xPUs in the system to protect different resources.
  - e. TZ also programs VMIDMT tables and the masters that have access to the tables.
- TZ access control – Sets up access control for the registers of following hardware blocks:
  - BAM
  - CPU registers
  - Qualcomm generic interrupt controller (QGIC)
  - Hardware crypto engine
  - QFPROM
    - TZ can write/blow QFPROM bits in application processor region
    - All other subsystems have read access, but no write access
  - PRNG

- Warm boot:
  - a. TZ runs within DDR partition allocated for it and application processor CPU starts in Secure mode
  - b. Power collapse terminates in TZ and it programs application processor CPU to start from TZ reset vector
- TZ warm boot – Performs the following initializations during warm boot:
  - a. CPU core system initialization, which configures CPU to a known state
  - b. Initializes CPU security
  - c. If it is a power collapse, initialize L2 cache
  - d. Switches to Non-Secure mode
- Secure peripheral image loader (PIL):
  - a. TZ Secure PIL authenticates different subsystem images and configures related xPUs to protect the subsystem memory regions
  - b. During initialization, the PIL authentication service in TZBSP protects the registers responsible for taking the peripheral processors out of reset
  - c. HLOS loads the ELF file for subsystem image at a 4-byte aligned location in DDR RAM
  - d. HLOS requests the PIL authentication service in TZBSP to authenticate the images securely
  - e. HLOS cannot authenticate the images as it can potentially be tempered
  - f. TZBSP protects the memory areas used by the subsystem images with the MPU
  - g. TZBSP initializes the registers of a processor and necessary clocks
  - h. Once an image is validated, the PIL authentication service resets the respective subsystem core so it can boot
- Power collapse:
  - Some xPUs are power collapsible and they are mostly related to multimedia functionalities.
- Crypto BAM:
  - TZ configures every BAM pipe crypto to prevent HLOS from using OEM and QTI hardware keys.
- Secure debug:
  - The OEM can disable the JTAG debug functionality by using QFPROM; which is re-enabled in TZ using software OVERRIDE registers
  - The hardware keys replace dummy keys when debug is enabled; TZ software then, use the dummy keys when debug is re-enabled
  - Watchdog debug

## Major BSP drivers in TZ

- QFPROM:
  - QFPROM blowing framework is now part of TZ
  - OEMs can use the following APIs to read and write QFPROM from their TZ application
    - qsee\_fuse\_read()/qsee\_fuse\_write()
  - API prototype is in \ trustzone\_images\ssg\api\securemsm\trustzone\qsee\qsee\_fuse.h
- Crypto engine:
  - Two general-purpose crypto engines (one each for modem and TZ)
  - Performance for CE – pending
- Random number generator (RNG):
  - Configured by TZ only
  - Both TZ and HLOS can access the generated random numbers
- SPI driver:
  - Used to interface with external devices (that is, fin Qualcomm® Fingerprint Sensors)
  - Configured by TZ and is disabled by default
  - To enable the SPI driver, uncomment the definition of BAM\_TZ\_DISABLE\_SPI in the \core\hwengines\bam\build\builds.scons build file

## Secure watchdog

Only secure watchdog timer (SWDT) can reset the system. All other WDTs generate an interrupt on bite. HLOS (subsystem restart) handles the subsystem WDT bites. TZ (context dump/reset) handles the HLOS WDT bite. Secure WDT bite then, reset device to the first stage of reset.

The longest bark/bite timeout in the system, maximum of 32 seconds. It is configured and accessible only by TZ. Enabled in cold boot after TZ enables interrupts. TZ pet secure WDT on bark.

Debug APIs:

- Disable secure WDT for debug purposes – tzbsp\_wdt\_disable()
- Trigger secure WDT for debug purposes – tzbsp\_wdt\_trigger()

## Secure Monitor

Secure Monitor executes at the highest privilege level (EL3) of Arm core and facilitates the transition between NON-Secure world to Secure worlds.

## Device Configuration

The TZ has multiple platform dependent customizable features such as:

- SFS storage path
- Key derivation features

- Access control configuration
- xPU enable disable configuration
- Hypervisor configuration
- Key re-provisioning configuration and so on.

Multiple configuration files are available for controlling different functionalities.

#### Ex. `oem_config.xml`

```
<props name="OEM_sec_wdog_bark_time" type="0x00000002">      6000      </props>
<props name="OEM_sec_wdog_bite_time" type=DALPROP_ATTR_TYPE_UINT32>
22000      </props>
```

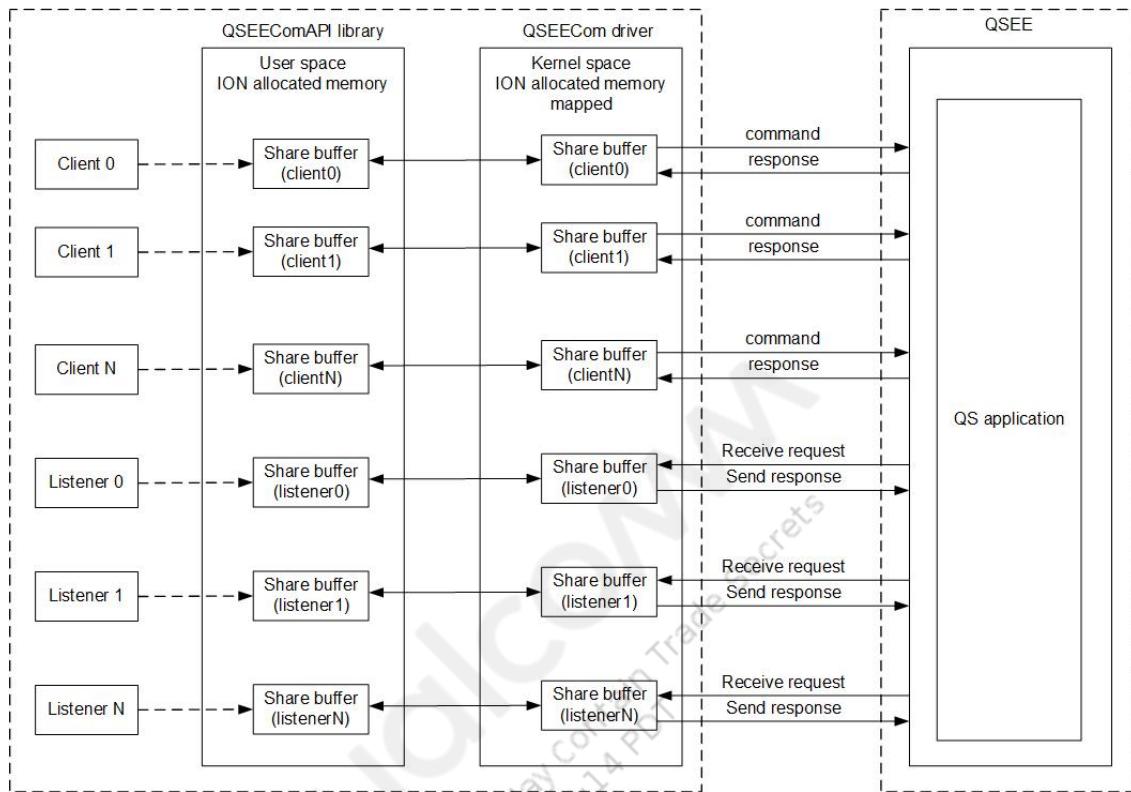
OEMs can change the configuration & build new device config(`devcfg.mbn`) images for use.

For information on TZ/Qualcomm TEE 5.0, see *Qualcomm Secure Execution Environment Version 5.0 User Guide* (80-NH537-4).

## 6.7.2 Qualcomm TEE 5.0

Major QSEE components for OEMs use cases are as follows:

- HLOS components:
  - HLOS client application to communicate with Trusted Applications in TZ.
  - Listener service – Listens for any requests coming from TZ of Trusted Application in TZ such as QSAPP, sample app client, Keystream and so on.
    - For example, RPMB Listener, FS Listener and so on.
  - QSEEComAPI library – Qualcomm TEE interface for HLOS to communicate to TZ via QSEECom kernel driver
  - In kernel space the components are:
    - QSEECom driver
    - SCM driver
- Secure world components
  - TZ – Secure OS running in Secure Mode of 64-bit Armv8-A Arm core.
  - QSAPP – Trusted application executing in user space of TZ, intended to serve the request of peer HLOS Client applications. For example, sample application.



**Figure 6-45 Components overview**

Transition from Non-Secure to Secure mode occurs via a Secure Monitor in Qualcomm TEE executing at EL3 privilege level.

Qualcomm TEE facilitates a way for HLOS to communicate with TZ via QSEECom.

### QSEECom client

The QSEECom client, also referred to as the QSAPP client, is the HLOS application that initiates all requests to QSAPPS. Initially, it is the client that issues a request to load the secure application by invoking `QSEECom_start_app()` and retrieving the handle to the QSEECom character driver. When the secure application is loaded, the client can use the retrieved handle to send commands/requests to the secure application.

### QSEECom listener

QSEECom listener, also referred to as QSAPP listener, is the HLOS service module that serves requests originating from Qualcomm TEE. These requests are mostly originated by a client/service (QSAPP) residing in Qualcomm TEE.

QSEE listeners are registered with QSEECom by invoking the `QSEECom_register_listener()` call. This function in turns calls `QSEECOM_IOCTL_REGISTER_LISTENER_REQ`. Upon successful registration, the QSEECom driver stores the listener ID in a listener service queue. The current QSEECom design handles registration of multiple listener services.

After registering with the QSEECom driver, each listener service must call `QSEECom_register_listener`, which in turn calls `QSEECOM_IOCTL_RCV_REQ` to start the listener service. One thread from each service is blocked after calling `QSEECOM_IOCTL_RCV_REQ`. The thread is signaled when QSEECom receives a command containing a particular listener service ID.

All listener services are initiated by a daemon called qseecomd that is run during start-up. The daemon lists all the available listener services. The purpose of this daemon is to register the available listed listeners and initiate the listener service thread that services all requests to the respective listener. Any new listener can be added to the list defined in the qseecomd daemon and the service must be implemented accordingly.

### **QSEECom driver**

Qualcomm TEE communicator (QSEECom) is a kernel module responsible for all HLOS communication with QUALCOMM TEE. QSEECom is a platform device driver developed specifically for Linux-based HLOS such as Android. It provides IOCTLs for the user space applications to communicate with QSEE. The driver also exports APIs for kernel space applications to communicate with QSEE on the secure end. The QSEECom driver uses the SCM interface to switch the context between the HLOS and QSEE. This driver also enables QSEE to use the services of the HLOS via a listener service interface.

### **User space access to QSEE**

All functions discussed in this section are invoked by APIs from the QSEEComAPI library context. The following operations are performed based on the open/close and IOCTL calls associated with a character device. These open/close and IOCTL calls should not be called directly by any HLOS client or listener. The purpose of listing these calls in this section is for documentation completion and to provide a better understanding of the QSEEComAPI library interface to the QSEECom driver. To communicate with the QSEECom driver, the user space client/listener must use the APIs exposed in the QSEEComAPI library.

#### **qseecom\_open**

The open() command on the QSEE communicator device translates into the qseecom\_open() kernel call to initialize the context of the QSEECom driver for the specific handle. This open() command is called by the HLOS QSEEComAPI library on two occasions:

- To start a secure application on QSEE
- To register a listener ID from QSEE, indicating the listener service now exists

**NOTE** Each handle (created as a result of the open() command) is unique per process. Handles cannot be shared between processes.

#### **qseecom\_close**

The close() command on the QSEE communicator device translates into the qseecom\_release() kernel call to release the context of the QSEECom driver for the specific handle. This close() command is called by the HLOS QSEEComAPI library on two occasions:

- To shut down a secure application on QSEE
- To deregister a listener ID from QSEE, indicating the listener service no longer exists

#### **qseecom\_ioctl**

When the secure application is started and the driver is opened, all other requests (data or control) are communicated via IOCTLs. The QSEECom driver provides these IOCTLs for the HLOS services/client:

- QSEECOM\_IOCTL\_REGISTER\_LISTENER\_REQ – Register the HLOS listener service
- QSEECOM\_IOCTL\_UNREGISTER\_LISTENER\_REQ – Deregister the HLOS listener service

- QSEECOM\_IOCTL\_SEND\_CMD\_REQ – Send a command request to QSAPP
- QSEECOM\_IOCTL\_SEND\_MODFD\_CMD\_REQ – Send a modified command request to QSAPP
- QSEECOM\_IOCTL\_RCV\_REQ – Wait for a command from QSAPP
- QSEECOM\_IOCTL\_SEND\_RESP\_REQ – Send response to the last incomplete command on QSAPP executive
- QSEECOM\_IOCTL\_ENABLE\_PERF\_REQ – Step up crypto clocks to improve crypto operation performance on secure applications
- QSEECOM\_IOCTL\_DISABLE\_PERF\_REQ – Step down crypto clocks to normal crypto operation mode
- QSEECOM\_IOCTL\_LOAD\_EXTERNAL\_ELF\_REQ – Load an application on a secure domain
- QSEECOM\_IOCTL\_UNLOAD\_EXTERNAL\_ELF\_REQ – Unload an application to a secure domain
- QSEECOM\_IOCTL\_APP\_LOAD\_QUERRY\_REQ – Check if an application is already loaded on the secure end

### **Kernel space access to QSEE**

All functions discussed in this section are exported APIs available to any kernel space modules to communicate with Qualcomm TEE.

#### **QSEEComAPI library**

QSEEComAPI is the library that exposes an API for the HLOS client/listener to send and receive data to/from Qualcomm TEE through the QSEECom driver. The APIs allow access to the QSEECom driver to issue commands and receive responses from the Qualcomm TEE. All HLOS clients and listeners must use the API exposed in this library to communicate with QSAPPS.

For more information, see *Qualcomm Trusted Execution Environment (QTEE) Version 5.0 User Guide* (80-NH537-4).

## **6.7.3 Secure boot and QFPROM**

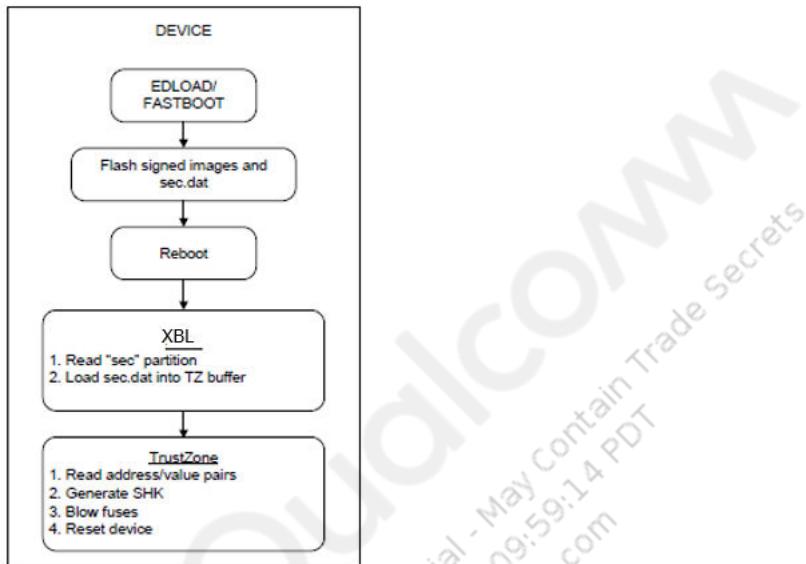
For information, see *QCS610 Secure Boot Enablement User Guide* (80-PL631-42).

### **QFPROM and fuse programming**

QFPROM is one-time programmable memory which keeps the configuration fuses. As a part of the secure boot enablement process OEMs do need to configure these fuses to create the secure state of device.

- The configurations are done in sectool configuration files:
  - sm6150\_secimage.xml
  - sm6150\_fuseblower\_OEM.xml
  - sm6150\_fuseblower\_QTI.xml and so on.
- Secure boot's root of trust is stored in QFPROM which is one-time programmable memory.
- QFPROM has several fuses to configure different security features and device security state.

- OEM configures QFPROM fuses in sectool fuse configuration xml files and then uses sectools scripts to generate sec.dat.
- The generated sec.dat file is programmed to the sec storage partition
- TZ parses the loaded sec.dat file from DDR, and programs the list of fuses one by one
- There are mandatory fuses, which must be blown to enable secure boot. For example, Debug disable fuses, image authentication enable fuse, rollback protection fuse, and so on.



**Figure 6-46 Emergency download/Fastboot flow chart**

For more information, see QCS610 QFPROM Programming Reference Guide (80-PL052-97).

#### 6.7.4 HLOS security

**Table 6-16 Supported HLOS security features**

Security feature	Description
Verified Boot LE	Ensures cryptographic authentication of HLOS images such as boot image.
DM-verity	Provides read-only transparent integrity checking of block devices.
Disk encryption	Full Disk Encryption (FDE) is supported.
SELinux	Provides added kernel security.
Kernel module signing	Kernel module signing is supported
Peripheral image loader	Authenticates and loads peripheral firmware, and resets the peripheral hardware.

The LEPDK provides the following security features:

- DM-verity:

The device mapper is an infrastructure in the Linux kernel that provides a generic way to create virtual layers of block devices. The dm-verity target

The dm-verity target provides read-only transparent integrity checking of block devices using kernel crypto API.

The dm-verity documentation can be found at the following path:

*KERNEL\_SRC/Documentation/device-mapper/verity.txt*

The complete specification of kernel parameters and on-disk metadata format is available at the cryptsetup project's wiki page: <https://gitlab.com/cryptsetup/cryptsetup/wikis/DMVerity>.

- SELinux: Enables enhanced access control over the system.

#### 6.7.4.1 DM-verity

Dm-verity uses a tree of hashes to sign the entire system partition with a single signature. This scheme allows data to be verified as it is read, rather than requiring the entire partition to be verified before it can be used. If the block of data is verified, the read succeeds and if the data is not verified, an input/output (I/O) error is generated as if the block was physically corrupt.

##### Feature delivery

The following combinations regarding dm-verity feature are applicable only for LEPDK based software products (SPs).

dm-verity feature is delivered as a distro feature `DISTRO_FEATURES += "dm-verity"`.

- LEPDK + UEFI bootloader combination

For this combination, choose dm-verity kernel driver and pass the driver specific parameters as required.

In the kernel configurations file, dm-verity kernel driver is added by using the configuration `CONFIG_DM_VERITY=y`.

By default, the system partition is signed with a private key, that is in PKCS #8 syntax, which is a standard syntax for storing private key information.

The private key that is in PKCS #8 syntax is available at `poky/meta-qtibsp/recipes-devtools/verity-utils/verity.pk8`.

##### Mandatory checkpoint

Do the following mandatory steps:

- LEPDK and UEFI bootloader combination
  - OEMs should generate own private key that is in PKCS #8 syntax:  
`verity.pk8`
  - Replace the existing private key in the following path:  
`poky/meta-qtibsp/recipes-devtools/verity-utils/verity.pk8`

### Impact of storage, memory, and performance

- **Storage:** 32 KB (metadata) + around 3 MB (for hash storage of 3 GB system partition)
- **Memory:** Recovery of lost memory up to 16-24 MB of consecutive blocks anywhere on a typical 2-3 GB. System partition is 0.8%. Memory overhead and no performance impact unless corruption is detected.
- **Performance (only if corruption is detected):** On a 2 GB partition, performance impact includes reading extra 2000 blocks that have encoded (corrected) data.

### Generate verity keys

To generate the verity keys, do the following:

- Generate the `verity.priv` private key.

```
# openssl genrsa -f4 2048 > verity.priv
```

Do not disclose or distribute this key.

- Generate the `verity.priv` public key certificate.

```
# openssl req -new -x509 -sha256 -key verity.priv -out verity.x509.pem -days 10000 -subj "/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com"
```

OEMs should change the `-subj` option as per their own string.

- Create the `verity.pk8` file in the PKCS #8 syntax of the `verity.priv` private key.

```
# openssl pkcs8 -in verity.priv -topk8 -outform DER -out verity.pk8 -nocrypt
```

#### 6.7.4.2 SELinux

SELinux is a security enhancement to Linux, which allows users and administrators to have more control over access to system. It was created by the United States National Security Agency ( NSA) as a set of patches to Linux Kernel using the Linux kernel module, and later, it was adopted by Linux kernel.

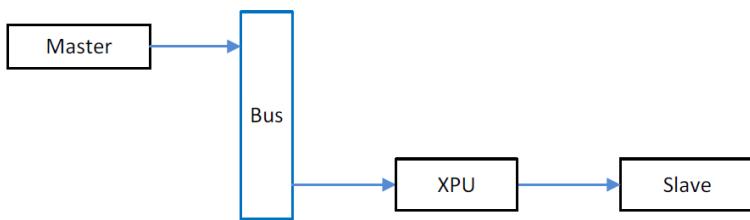
- It provides a mandatory access control (MAC) system, which defines access control for users or processes. Thus, improving the security of the system.
- After SELinux is enabled in the Enforcing mode, the corresponding SELinux policies are enabled across the modules of the HLOS software.
- When this software is executed, disallowed access is prevented, and the kernel logs an attempted access violation as an accessVector cache (AVC) denial message to dmesg and logcat.
- By writing new SELinux policies or redefining the existing policies, the user can use these AVC denials to refine the HLOS software.

For more information, see <https://www.redhat.com/en/topics/linux/what-is-selinux>.

#### 6.7.5 Device access control

The QCS610 chipset has hardware xPU support for enforcing slave side access control for register/fixed address/dynamic memory regions.

xPU stand for extended protection unit, It is a hardware block monitoring memory resources access at slave side.



It is the combination of multiple security blocks known as protection units, which protect memory resources from unauthorized access.

- The xPU units are:
  - Memory Protection Unit (MPU) – Protects a software defined memory region.
  - Address Protection Unit (APU) – Protects a pre-decoded address region.
  - Register Protection Unit (RPU) – Protects a group of register(s).
- The xPU definition is configurable at compile time.
- Master conditionally granted access to resource based on security attribute, such as:
  - VMID
  - xMssSelfAuth
- The xPU configuration follows Arm security extensions:
  - Some configuration registers are for secure access only
  - Non-secure configuration access controlled by secure side
  - Support for MSA feature. xMssSelfAuth identifies MSA traffic

The xPU is configurable via device configuration files and these configurations get compiled as part of devcfg.mbn image which later parsed and programmed in xPU registers by TZ.

The xPU can be enabled or disabled via device configuration in *xpu\_config.xml*.

```

<device id="/ac/xpu">
  <props name="disable_xpu_ac" type=DALPROP_ATTR_TYPE_UINT32> 0</props>
</device>
  
```

## 6.7.6 Crypto core and software crypto modules

The QCS610/QCS410 chipset has both software and hardware crypto support, both in HLOS and TZ. It has all the standard cryptographic algorithm support, such as, AES-128/256, DES/Triple-DES, HASH-SHA-256/SHA-384, HMAC, ECC, and so on.

**Table 6-17 Crypto core details**

Crypto core	Details
Inline crypto engine (ICE)	Crypto core in line with eMMC/UFS device storage for accelerating storage device encryption and decryption. Major used in FDE/FBE disk encryption process.
General purpose crypto engine (GPCE)	Crypto core available for general purpose use cases.
HWPRNG	Pseudo random number generator

For information on crypto interfaces in TZ, see *Qualcomm Trusted Execution Environment Version 5.0 User Guide* (80-NH537-4).

For details on crypto interface in HLOS, see the documentation in Linux file system of your meta build at Documentation/crypto/msm/\*.

## 6.7.7 Secure storage

**Table 6-18 Secure storage support for specific use cases**

Secure storage	Details
Qualcomm fuse programmable read only memory (QFPROM)	QFPROM along with configurable fuses provides OEM programmable spare region. OEM uses this to keep their permanent keys or configuration data.
Replay Protected memory block (RPMB)	RPMB has limited storage space and is mostly used for keeping version information of device images, such as, SFS files, device key files, Keymaster keyblob, and so on. OEMs use it for storage of configuration requiring significantly limited storage size.
Secure file system (SFS)	SFS provides encrypted secure storage for TZ, secure application and modem.

For information on secure storage interfaces in TZ, see *Qualcomm Trusted Execution Environment Version 5.0 User Guide* (80-NH537-4).

## 6.7.8 Secure device debugging

The secure devices cannot be debugged like non-secure devices. All the logs and dump collection are disabled on secure device and multiple tools such as, QFIL and ADB do not work on it.

The secure devices debugging needs specific utilities and prior work of customized image generation to get the debug information out of secure device and flash the debug images in the device.

**Table 6-19 Debug utility details**

Debug utility	Details
Debug policy	Debug policy images with right configurations are needed to be flashed in the device via VIP. It enables the logs and dumps collection on the secure device.
VIP	VIP utility is used to flash signed images on the secure device.
System debug interface (SDI)	SDI needs to be properly configured in order to collect the dump.

For information on debug policy, see *Sectools: Debug Policy Tool User Guide* (80-NM248-6). For information on VIP, see *Qualcomm Firehose Validated Image Programming Guide* (80-P9116-1).

## 6.7.9 Crypto APIs

The Linux kernel crypto APIs headers are at *kernel/msm-5.4/include/crypto/* and the source is at *kernel/msm-5.4/drivers/crypto/msm/*. For more information, see the documentation at *kernel/msm-5.4/Documentation/devicetree/bindings/crypto/msm/*.

For TZ software and hardware crypto APIs, see *Qualcomm Trusted Execution Environment (QTEE) Version 5.0 User Guide* (80-NH537-4).

## 6.7.10 BSP binary protection

It is recommended to enable BSP binary protection for any new binary that is being added. All the QTI binaries are compliant with few exceptions, and some of the OSS binaries are not compliant.

The following are the binary protection techniques:

- Relocation read-only (RELRO): Provides a memory corruption mitigation technique
- Canary: Provides stack buffer overflow protection
- Position independent executable or position independent code (PIE or PIC): A PIE binary and all of its dependencies are loaded into random locations within virtual memory each time the application is executed. This makes executing the return oriented programming (ROP) attacks difficult
- Non-execute or Non-executable (NX) segment: The application when loaded in memory does not allow any of its segments to be both writable and executable

## 6.7.11 Root process audit

QTI recommends that any new process or daemon should run non-root. If there is an security breach for process or daemon that has root permissions, an attacker can gain unnecessary access to the system.

All the QTI processes or daemons are compliant with few exceptions. And, some of the OSS processes or daemons are not compliant.

### 6.7.12 Full disk encryption

Full Disk Encryption (FDE) in Linux embedded provides a mechanism to protect used data. It encrypts the user in process of protecting it from any illegal access.

It is hardware-based encryption mechanism, which uses ICE to accelerate the encryption/decryption of user partition. FDE uses symmetric key encryption.

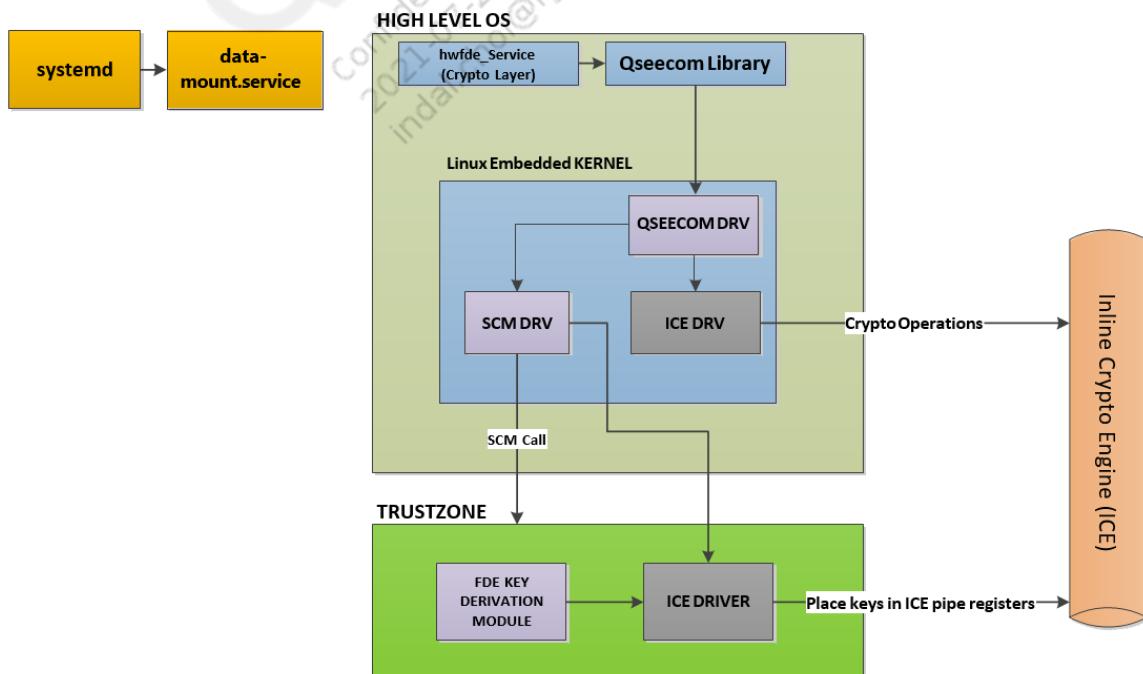
As per the current configuration, the FDE application is limited to protecting the *userdata* partition in the device. It does not provide encryption facility for any other partition.

The user data build image is `usrfs.ext4`.

To enable, configure, and verify FDE, see *QCS610/QCS410 Linux Platform Development Kit Software Programming Guide* (80-PL631-200).

**Table 6-20 FDE components**

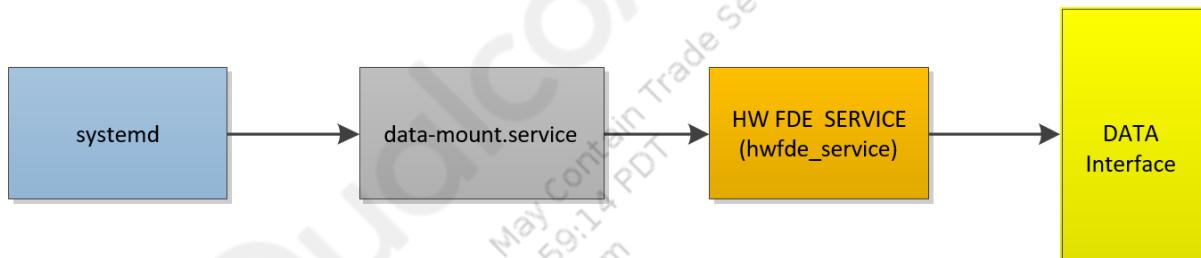
Component	Description
data-mount.service	Starts the <code>hwfde_service</code> on device boot up when it is called from <code>systemd</code> .
hwfde_service	Manages the configuration and derives the disk encryption process.
Qseecom	Qseecom facilitates the communication between HLOS FDE component and TZ FDE components.
ICE	Performs the encryption and decryption
TZ	Secure OS, which inhabits the secure FDE component that does the key derivations and ICE configuration task.



**Figure 6-47 FDE component workflow**

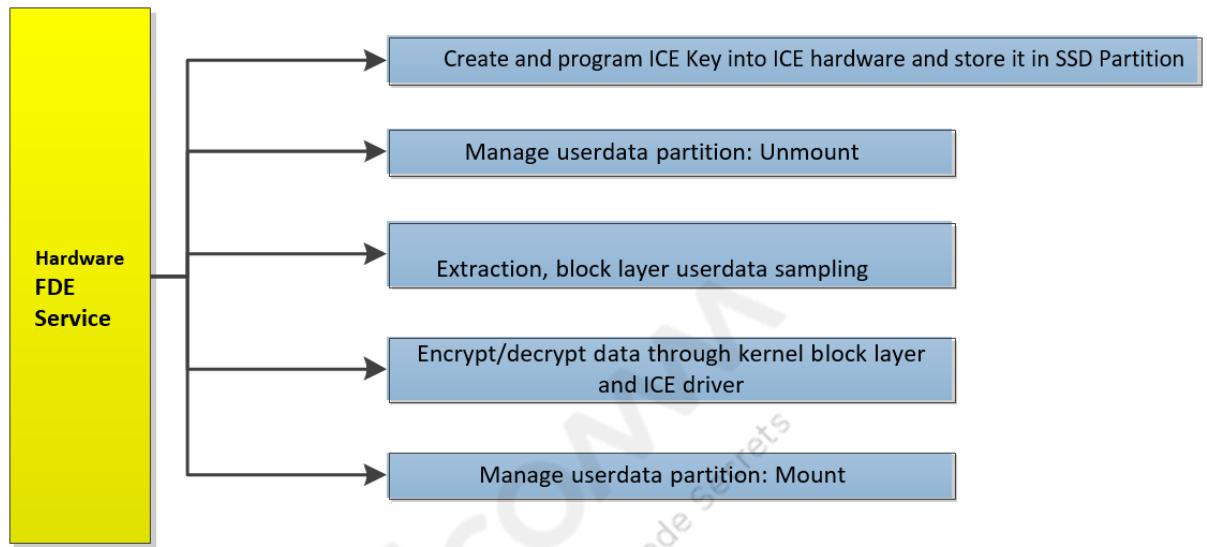
1. FDE takes place during the first boot of device after flash. Later, every I/O operation to userdata partition passes through the inline crypto engine hardware and is encrypted or decrypted for write/read operation before being read from or written back to the disk.
2. The hardware FDE service makes a call in cryptfs to configure inline crypto engine.
3. In response, cryptfs via kernel block layer driver, sends a request to inline crypto engine driver in TZ to configure ICE. Additionally, cryptfs makes request to TZ to derive a master key and program it in ICE pipes to be used by ICE hardware.
4. 12b bit Advanced Encryption Standard (AES) with cipher-block chaining (CBC) and ESSIV:SHA-256 is used for user data encryption.
5. After ICE has the master key in its pipe and FDE configuration in place. It is ready for FDE.
6. The hardware FDE service initiates block by block, encryption and decryption sequence begins on user data partition.

#### FDE trigger at device bootup



- systemd executes the data-mount.service.
- data-mount.service executes the hardware FDE service to start disk encryption.
- Hardware FDE service calls the crypto interfaces to configure ICE and trigger the encrypt operation.
- At this stage, the hardware FDE is triggered and hwfde\_service processes the rest.

## Hardware FDE service data operations



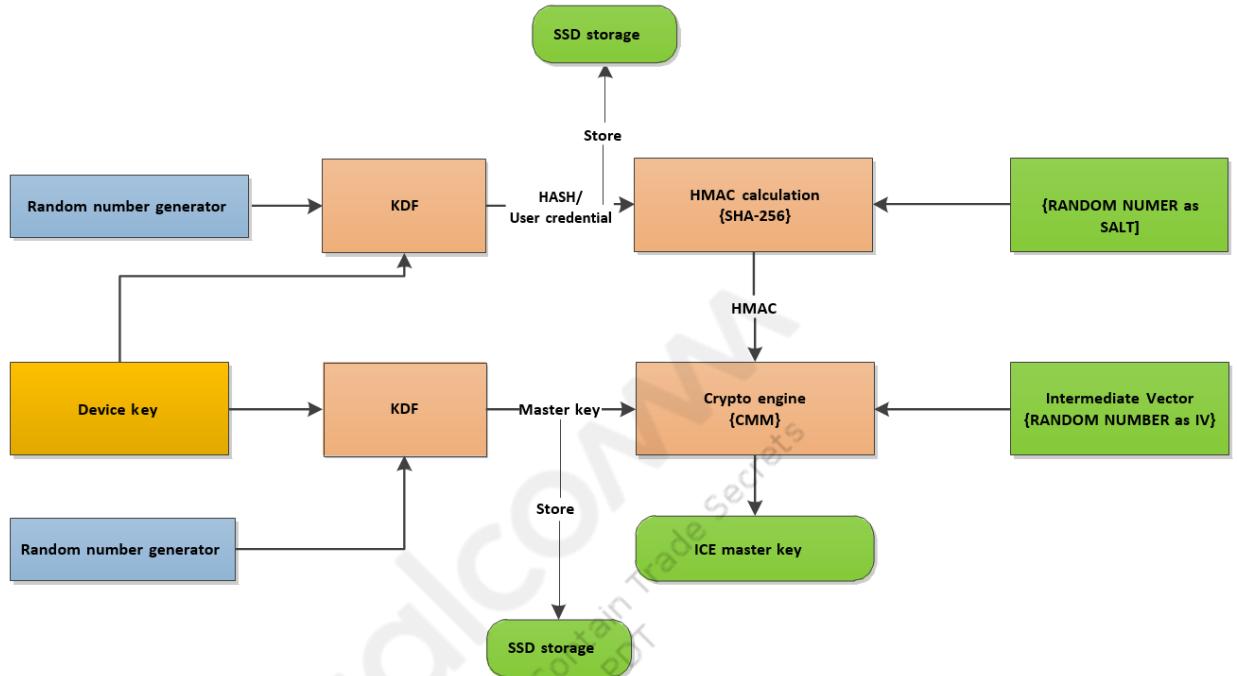
The hardware FDE service does the following:

- Performs the following key management functions with help of QSECOMM, ICE driver (Both in kernel and TZ) and TZ.
  - Key creation
  - Key storage in secure storage
  - Programming secure key in ICE pipes
- Unmounts the userdata partition to encrypt it.
- Uses EXT4 FS layer interface and utils to collect the userdata partition information and decide on the sampling rate of userdata for encryption.
- Perform in-place encryption of userdata blocks through hardware ICE.

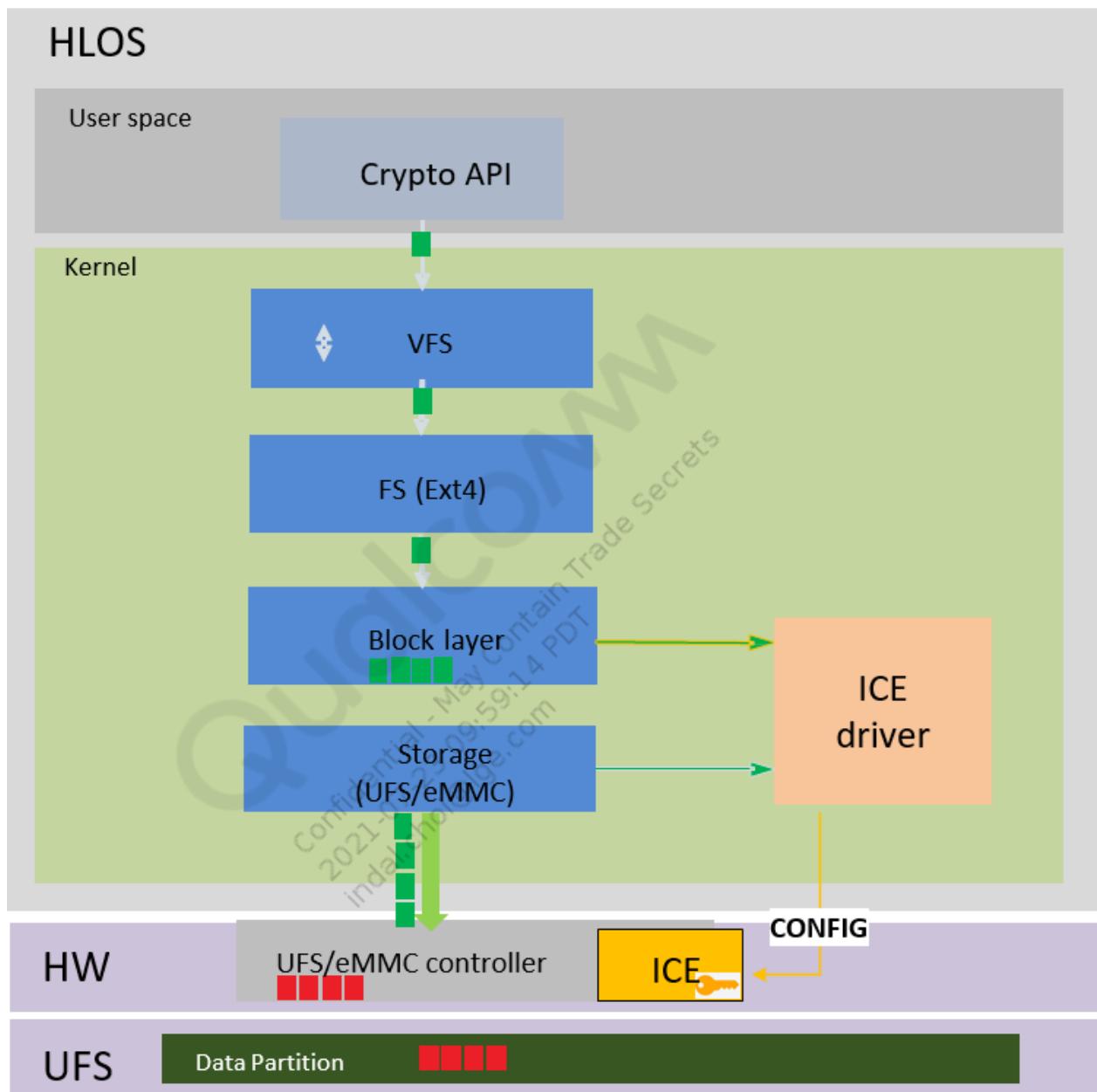
After the userdata encryption is successful, it is represented via `/dev/x`, where x can be any number, and it is mounted at `/data` mount point.

- `getprop ro.crypto.type` should be encrypted.
- `getprop ro.crypto.state` should be block.

## Key generation and storage



- Key derivation function (KDF) of random number and device key is calculated and stored in the SSD partition via TZ KeyStore module and SSD Listener in HLOS.
- This storage process is well-secured and HLOS will never have access to the data that is already stored or being stored. HMAC is calculated over this HASH and SALT (input).
- Again, KDF of random number and device key is calculated and in stored in SSD partition. This is the Master key.
- HMAC, intermediate vector (IV) and master key are passed to crypto engine, which processes it through CMM algorithm and generates the ICE key.
- TZ programs the ICE key in ICE register. This key is not stored, it is generated every time.
- Device Key is programmed in QFPROM and default password and master key are stored in SSD partition; only TZ has access to these locations.



- The hardware FDE service is invoked by the INIT script.
- Read/write calls are made to kernel from hardware FDE service.
- File system/block layer prepare the bio/requests.
- Block layer marks the requests for encryption/decryption. The actual encryption/decryption takes place through ICE.
- ICE driver in HLOS does the basic hardware configuration. The data is encrypted/decrypted by ICE engine based on the configuration done in storage controller.

## 6.8 Thermal

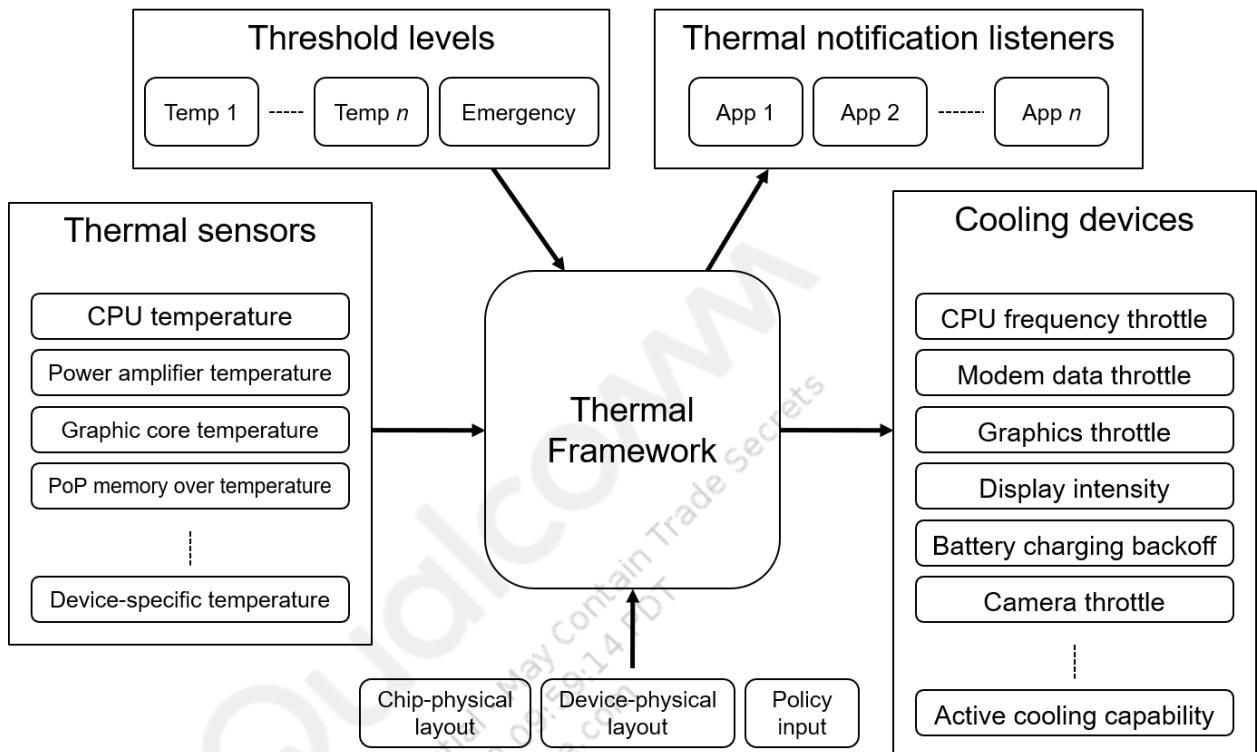


Figure 6-48 Thermal mitigation software architecture

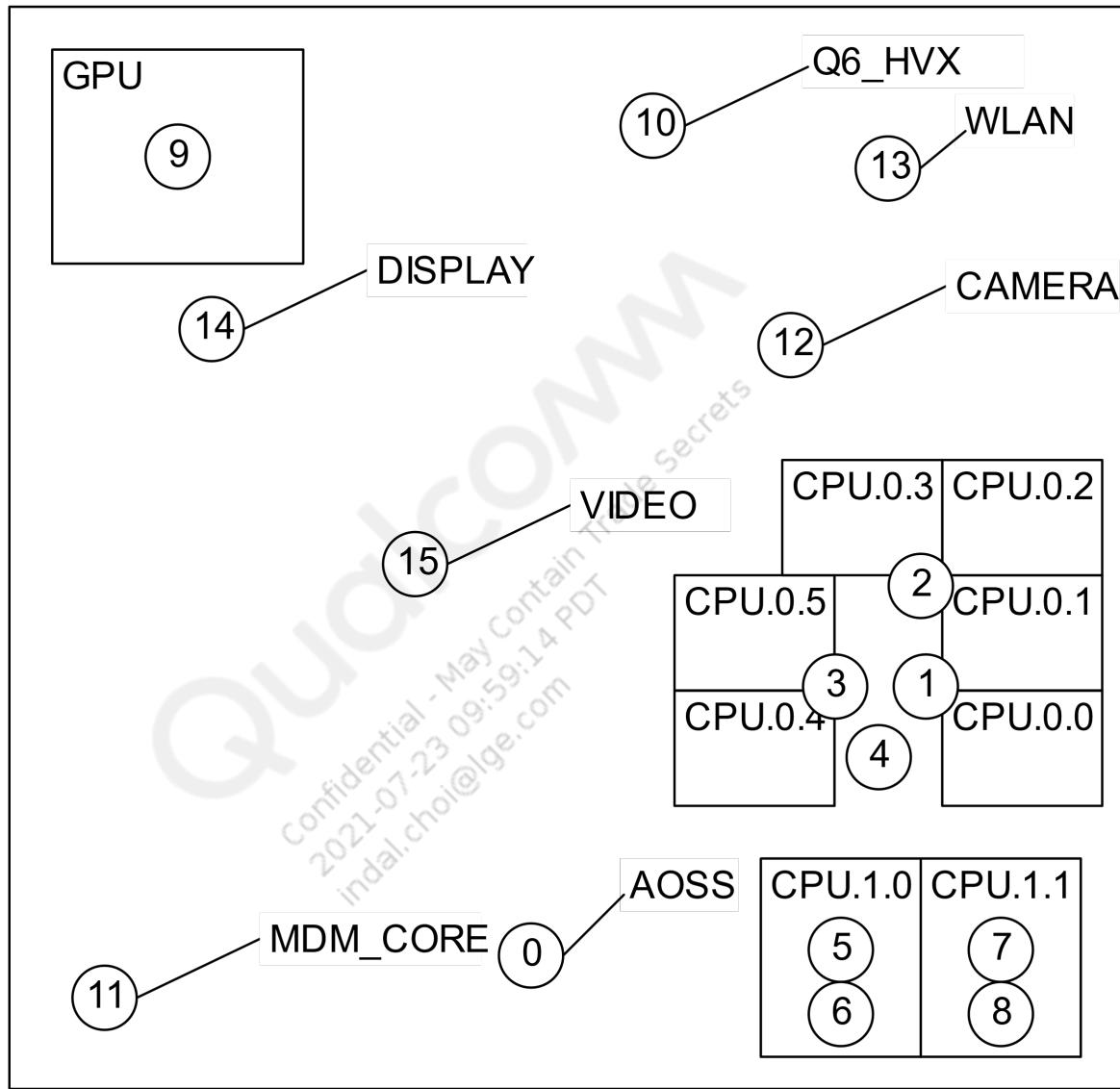
### 6.8.1 Temperature sensors

Thermal management software helps to manage chip temperature limits and external device skin temperature limits. Thermal sensors are placed on the chipset die. Board thermistors such as PMIC, PA, XO are used.

The QCS610 chipset has 16 on-die sensors. The sampling rate is 2.72 msec for more responsive thermal actions. Hardwired `tsens_reset` occurs to protect the device when predefined critical high or low threshold in SBL1 exceeds.

Thermal management devices enable passive cooling, which is applied by reducing performance. The device selection and threshold configuration is done to tune for ID variability through a configuration file.

Additional sensors (thermistors) are placed on the PCB, near the PAs.



**Figure 6-49 Placement of on-die temperature sensors**

### 6.8.2 Thermal management software

The thermal management software enables managing the limits for chip temperature and external device skin temperature. The software manages sensors that are on the chipset die and board thermistors such as PMIC, PA, XO, and so on.

The management devices apply passive cooling by reducing performance. The device selection and threshold are configurable by the OEM to tune for ID variability through a configuration file.

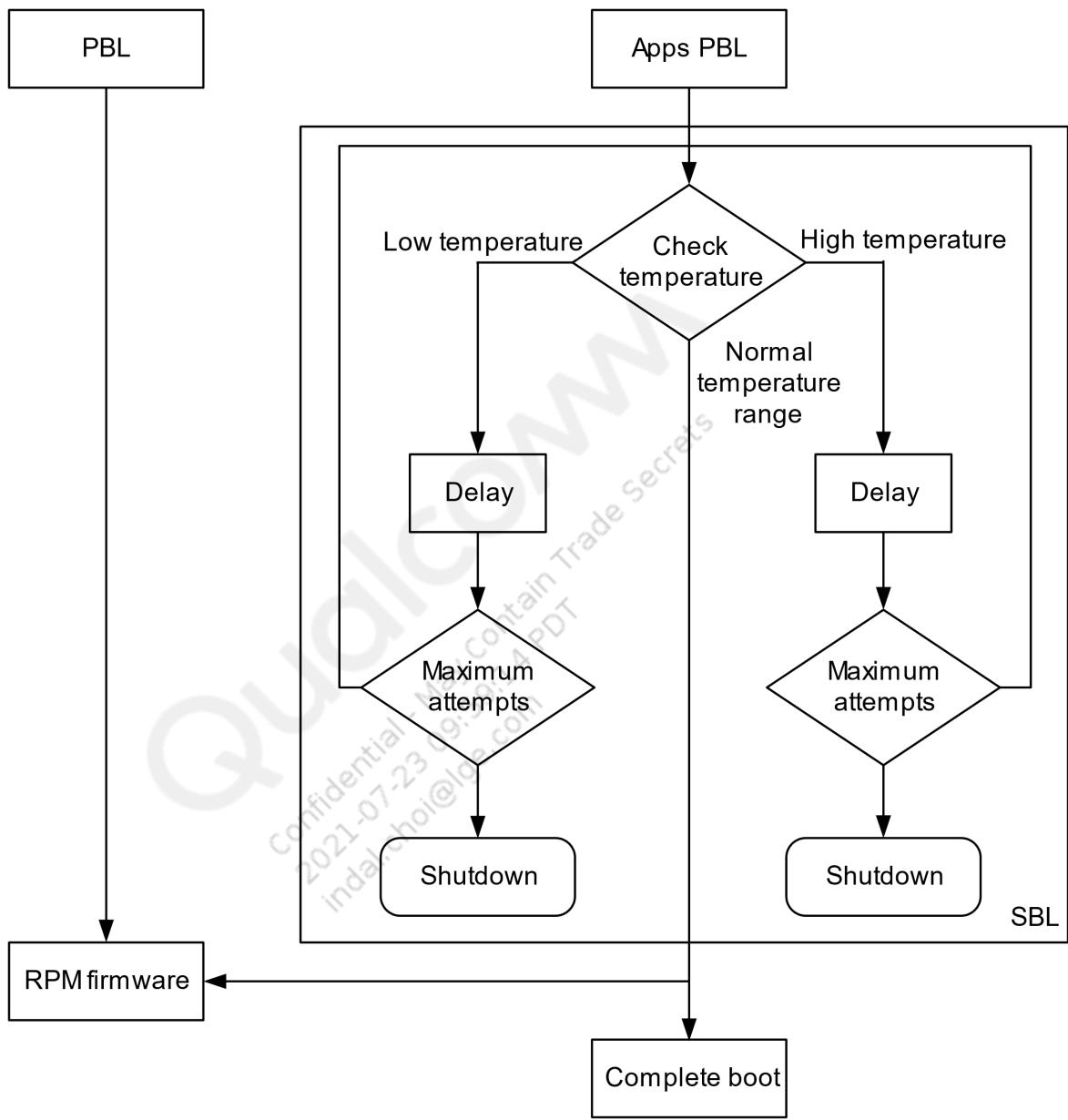
The software architecture comprises the user space, which has custom  $T_{skin}$  rules and application-level awareness. The kernel enables core isolation. The software also limits hardware for time critical mitigation for high junction temperature and/or current.

### 6.8.3 Boot thermal management

The following are the boot voltage restrictions:

- The boot thermal management (BTM) algorithm ensures that the temperature is in a valid operating range before allowing the device to boot.
- The temperature thresholds, delays, and maximum attempts are configured in the boot loader build.

Qualcomm Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com



**Figure 6-50 BTM algorithm flow**

The BTM is disabled by default. The bootup is deferred if the current temperature is higher or lower than the corresponding threshold values.

- .nLowerThresholdDegC is set to -150°C
- .nUpperThresholdDegC is set to 150°C
- The bootup can fail if the number of retries exceeds the predefined limit.

The BTM algorithm is implemented in the boot loader library at *QcomPkg/Library/BootTempCheckLib/BootTempCheck.c*

## 6.8.4 Thermal core framework

For QCS610, thermal core is the replacement for KTM. The following are the key functions of the thermal core:

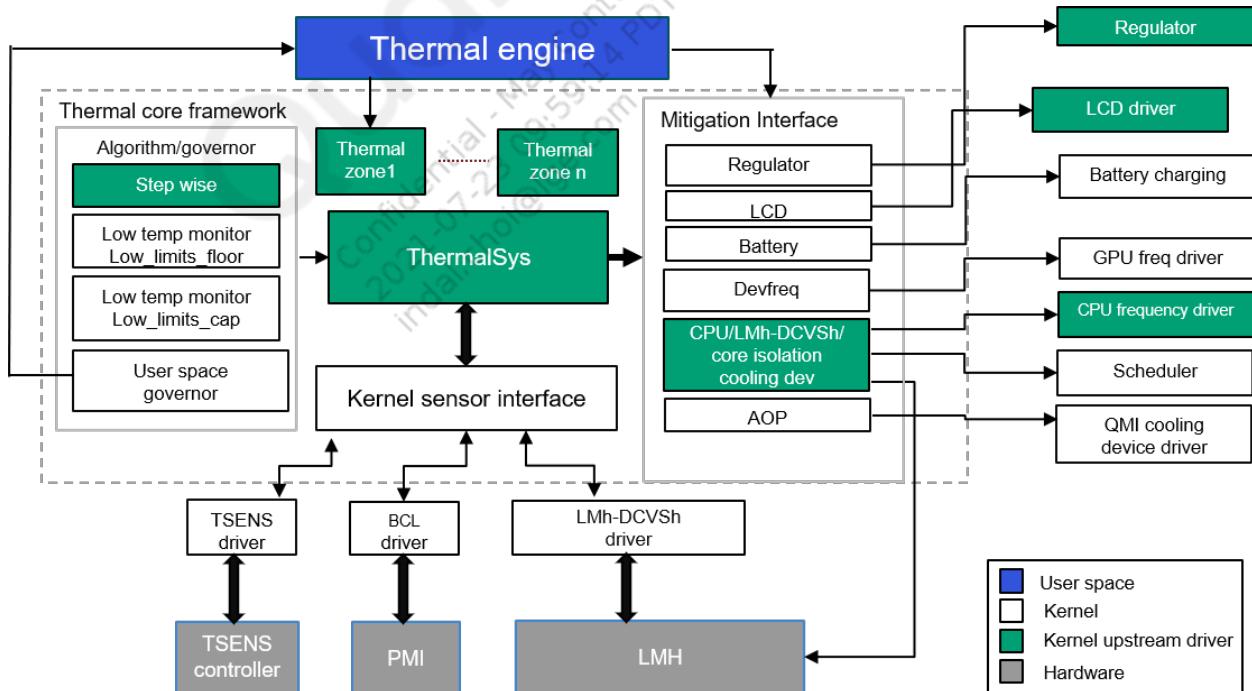
- Core isolation (hotplugging)
- CPU frequency mitigation
- GPU T<sub>j</sub> management
- Low temperature management, that is, VDD restriction

Thermal core is part of the upstream Linux solution for thermal management.

For more information, see *Thermal Core Overview* (80-P9301-113).

The thermal core framework is represented as follows within the context of QTI limits management solution. The user space engine is used to tune the thermal configuration during the development stage. After configuration is finalized, OEMs must move the tuned configuration to a dtsi file.

The figure indicates the plan to upstream many of the kernel-level thermal management features and drivers:



**Figure 6-51 Thermal core framework architecture**

### Thermal core framework – Device tree entries

One of the key functions of the thermal core framework is core hotplugging.

The following is an example device tree entry for hotplugging CPU0 when ibat reaches 2.3 A and clears the mitigation when ibat goes less than 2.1 A.

Assuming that there are 22 frequency mitigation steps for cluster0 (indexed from 0), step 22 is the hotplugging step.

```
ibat {
    polling-delay-passive = <100>;
    polling-delay = <0>;
    thermal-governor = "step_wise";
    thermal-sensors = <&bcl_sensor 0>;
    trips {
        low_ibat_00: low-ibat {
            temperature = <2300>;
            hysteresis = <200>;
            type = "passive";
        };
    };
    cooling-maps {
        ibat_map0 {
            trip = <&low_ibat_00>;
            cooling-device = <&CPU0 22 22>;
        };
    };
};
```

**NOTE** Since the upper and lower mitigation values are the same, the rule behaves like a monitor rule, staying at Level 22, until hysteresis is reached.

## LMH

LMH is a hardware-based protection circuit. LMH enhances reliability and robustness of the device in the following ways:

- Manages the peak current consumed by the CPU subsystem within the specified capability of the PMIC supply rail.
- Provides fast thermal management response if any CPUs are overheated (this applies to performance clusters only).
- HLOS (thermal core framework) can vote for frequency changes to the LMH, this request is made directly to the hardware for a faster response

For more information, see *Limits Management Hardware (LMH) Overview* (80-NM328-108).

### 6.8.5 User space thermal engine

The thermal engine has altered thermal daemon to enable integration with the sensor manager and allows for multiple algorithms.

- Thermal engine – It supports legacy and advanced dynamic algorithms running in parallel; OEMs are able to choose the existing algorithm or the new algorithm in the thermal engine configuration file
- Dynamic algorithm – Dynamic thermal management (DTM) exhibits significantly reduced tuning effort and improved average DMIPS
- Virtual sensor – It is a combination of more than two sensor readings with associated weights to accurately correlate skin temperature
- Dynamic parameter update – Important set of thermal engine configuration parameters can be updated at runtime for better OEM-specific DTM

QCS610-specific LMH features are as follows:

- Peak current management is applied to Gold CPU cluster
- Thermal management is applied to each CPU cluster, that is, Gold and Silver
- Junction temperature management is done using the SS algorithm

Other thermal software features are as follows:

- Battery current limiting – CPU mitigation based on state-of-charge level of battery
- Speaker coil calibration – Automatic calibration of speaker coil resistance vs. temperature enables audio codec to protect against speaker coil damage at high temperatures and high-power output
- Voltage restriction – Voltage restriction enables low operating voltage above 0°C by adjusting the required minimum voltage at temperature extremes
- Dynamic parameter update – Important parameter sets can be updated at runtime for better OEM-specific DTM

<b>SW</b>			
<b>Channel</b>	<b>Controller Number</b>	<b>Name</b>	
0	0	0	AOSS
1	0	1	CPUSS.0
2	0	2	CPUSS.1
3	0	3	CPUSS.2
4	0	4	CPUSS.3
5	0	5	CPU.1.0
6	0	6	CPU.1.1
7	0	7	CPU.1.2
8	0	8	CPU.1.3
9	0	9	GPU
10	0	10	Q6_HVX
11	0	11	MDM_CORE
12	0	12	CAMERA
13	0	13	WLAN
14	0	14	DISPLAY
15	0	15	VIDEO

**Figure 6-52 Software channels**

**Table 6-21 Thermal software mitigation devices**

Mitigation device	Function
CPUx	Adjustment of maximum allowed operating frequency per cluster
GPU	Adjustment of maximum allowed operating frequency
Shutdown	Safety mechanism to ensure CPU cores shut off before junction temperature limits are exceeded
Modem	Adjustment of peak data rates, maximum Tx power, and data call termination
LCD	Adjustment of maximum backlight intensity
Battery	Adjustment of maximum allowable charge rate
Camera and Camcorder	Adjustment of fps rate

## 6.8.6 Thermal mitigation algorithms

The following are the thermal mitigation algorithms, which you can configure in the thermal configuration file:

- DTM – algo\_type ss
- Monitor – algo\_type monitor

### DTM

DTM uses algo\_type ss algorithm. You can specify a single temperature at which to maintain and the algorithm dynamically throttles the user-specified action (CPU frequency) accordingly. The algo\_type ss configuration is used for skin temperature and on-die temperature control. This configuration requires less tuning effort than the monitor.

Enables stricter enforcement of temperature limits. Only works for CPU and GPU mitigation (such as lowering maximum frequencies). Threshold changes for the DTM algorithm are made via *set\_point* and *set\_point\_clr* in the *thermal-engine.conf* file.

The DTM algorithm is strongly recommended over monitor for CPU and GPU mitigation as it greatly relieves the tuning effort, boosts performance, and strictly maintains temperature.

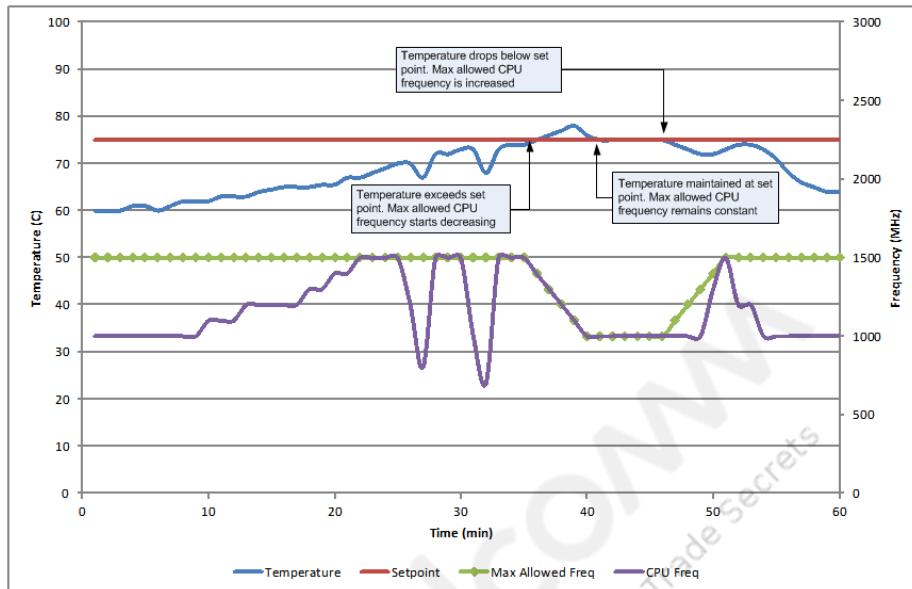
### Monitor

The algorithm is applicable to LCD, modem, and battery mitigation, and uses algo\_type monitor configuration. You must specify a series of temperature thresholds and a precise corresponding action per the threshold.

The monitor algorithm is not recommended for CPU and GPU mitigation as it requires extensive tuning to find each set point.

The threshold changes for the monitor algorithm should be made via thresholds and *threshold\_clr* in *thermal-engine.conf*.

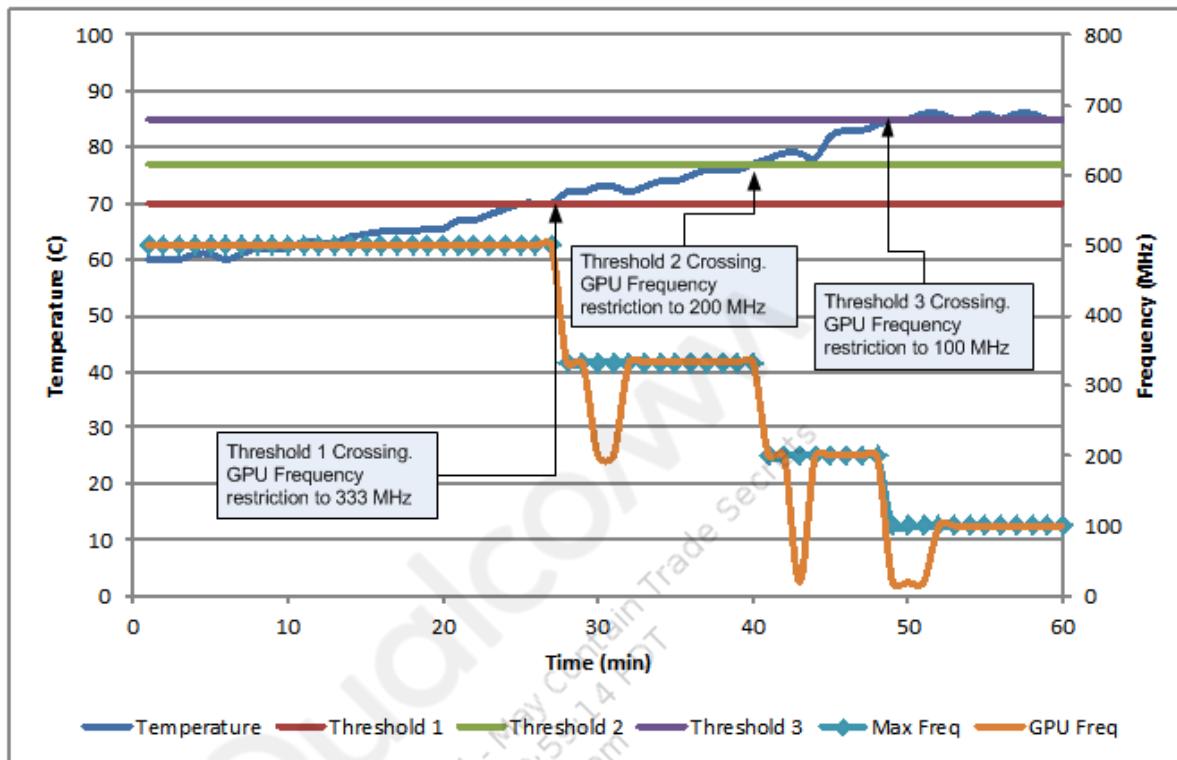
### Dynamic algorithm example – estimated response



**Figure 6-53 Dynamic algorithm example**

- When the temperature crosses the set point ( $75^{\circ}\text{C}$ ), the performance reduces until the temperature stabilizes.
- The Polling mode is enabled based on the sampling parameter defined on the rule. As the temperature falls below  $75^{\circ}\text{C}$ , the maximum allowed frequency is restored.
- If the temperature drops further below the set point clear ( $50^{\circ}\text{C}$ ), the Interrupt mode is re-enabled. The dynamic algorithm is used for the CPU and GPU control.

### Threshold algorithm example



**Figure 6-54 Threshold algorithm example**

The measured temperature crosses a predefined threshold, and then sets a predefined mitigation level. When the sensor temperature reaches 70°C, the GPU frequency is reduced from 500 MHz to 333 MHz. The final sensor temperature is maintained at 85°C.

Property	Level 1	Level 2	Level 3
Threshold set	70°C	77°C	85°C
Threshold clear	65°C	72°C	80°C
GPU frequency	333 MHz	200 MHz	100 MHz

## 6.8.7 Thermal engine configuration

The following is a sample configuration:

```
#debug mode and default sampling
debug
sampling 1000
#Monitor algorithm instance example
[CPU5_HOTPLUG_MONITOR]
algo_type monitor
sampling 1000
sensor cpu3
thresholds 90000
thresholds_clr 70000
actions hotplug_3
action_info 1
#Dynamic algorithm instance example
[SS-CPU0]
algo_type ss
sampling 10
sensor cpu0
device cluster0
set_point 85000
set_point_clr 65000
time_constant 0
```

## 6.8.8 Thermal zones

### Thermal\_zonesdtsiFiles

Thermal\_zone can be one of the following:

- Simple thermal sensor information
- Thermal sensor and thermal configuration instance used by thermal core framework

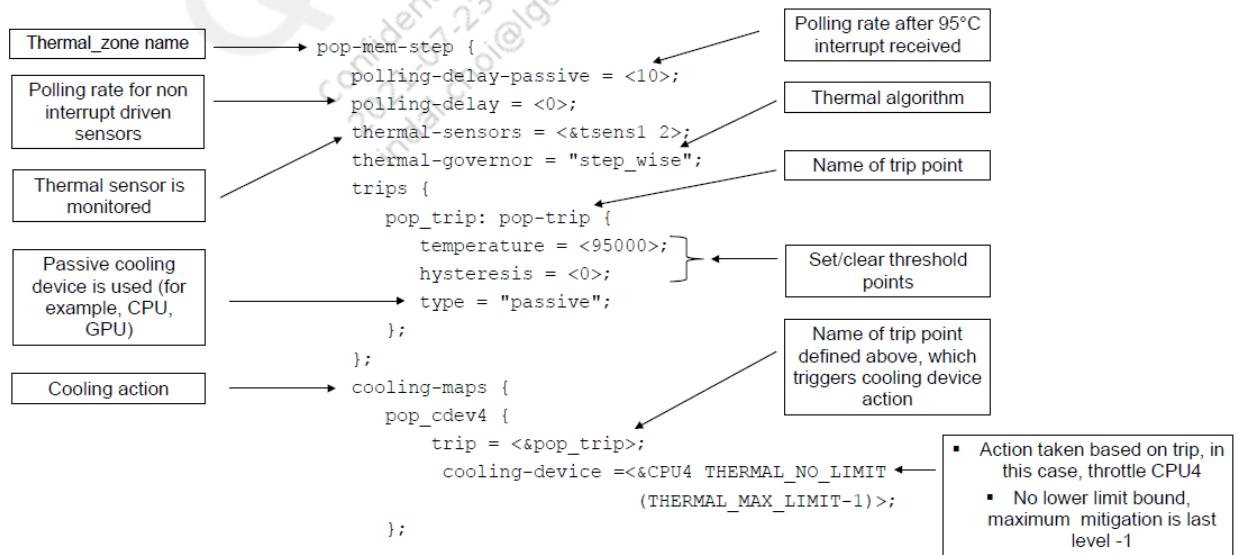
Thermal sensor <>target>-thermal.dtsi example:

```
cpul-gold-usr {
    polling-delay-passive = <0>;
    polling-delay = <0>;
    thermal-sensors = <&tsens0 8>;
    thermal-governor = "user_space";
    trips {
        active-config0 {
            temperature = <125000>;
            hysteresis = <1000>;
            type = "passive";
        };
    };
};
```

Sensor and thermal configuration:

```
pop-mem-step {
    polling-delay-passive = <10>;
    polling-delay = <0>;
    thermal-sensors = <&tsens1 2>;
    thermal-governor = "step_wise";
    trips {
        pop_trip: pop-trip {
            temperature = <95000>;
            hysteresis = <0>;
            type = "passive";
        };
    };
    cooling-maps {
        pop_cdev4 {
            trip = <&pop_trip>;
            cooling-device = <&CPU4 THERMAL_NO_LIMIT
                (THERMAL_MAX_LIMIT-1)>;
        };
    };
};
```

### Thermal\_zones parameters



**NOTE** The full listing of parameters is available in kernel documentation */kernel/msm-4.14/Documentation/devicetree/bindings/thermal/thermal.txt*

## Thermistor sensor thermal\_zones

Config rule for skin temperature control is recommended to be placed in dtsi with the new thermal core framework.

dtsi file	Purpose of thermal_zones	PCB thermistor
<<target>>-mtp.dtsi or pm6150.dtsi (or OEM's platform dtsi)	PCB thermistor exposure	xo-therm-adc msm-therm-adc pa-therm1-adc quiet-therm-adc emmc_therm-adc

xo\_therm/quiet-therm-adc entry example:

```

xo-therm-adc {
    polling-delay-passive = <0>;
    polling-delay = <0>;
    thermal-sensors = <&pm8998_adc_tm
0x4c>;
    thermal-governor = "user_space";
    trips {
        active-config0 {
            temperature = <65000>;
            hysteresis = <1000>;
            type = "passive";
        };
    };
};

quiet-therm-adc {
    polling-delay-passive = <0>;
    polling-delay = <0>;
    thermal-sensors = <&pm660_adc_tm 0x51>;
    thermal-governor = "user_space";
    trips {
        active-config0 {
            temperature = <65000>;
            hysteresis = <1000>;
            type = "passive";
        };
    };
};

```

## Thermal zone sysfs

Thermal zone sysfs exposes every sensor available in the system.

**Example:** cd sys/class/thermal/thermal\_zone23

```
sdm845:/sys/class/thermal # cd thermal_zone23
sdm845:/sys/class/thermal/thermal_zone23 # ls
available_policies    integral_cutoff   offset          temp
cdev0                 k_d               policy         trip_point_0_hyst
cdev0_lower_limit     k_i               power          trip_point_0_temp
cdev0_trip_point      k_po              slope          trip_point_0_type
cdev0_upper_limit     k_pu              subsystem     type
cdev0_weight          mode             sustainable_power uevent

Type      : sensor name
temp      : sensor reading
Mode      : monitoring is enabled or disabled
Policy    : currently configured thermal governor
available_policies   : Thermal governors available for this target
trip_point_*_temp     : trip temperature for trip number
trip_point_*_hyst     : trip temperature hysteresis for trip number
trip_point_*_type      : trip type for the trip number
Cdev*/type            : cooling device name
Cdev*_trip_point      : trip number that this cooling device is
associated to.
```

The mitigation level is exposed through cooling\_device node in sysfs. To read the mitigation level from sysfs, go to sys/class/thermal/cooling\_deviceX/cur\_state.

The cooling\_deviceX/cur\_state GPU mitigation level is exposed, where X is the cooling\_device number that represents devfreq (node for GPU). The cur\_state reflects the mitigation index and not the actual value. If there is no mitigation, cur\_state reads as 0 and values above 0 represent incrementally deeper mitigation.

The following is the node to check LMH final aggregated request send to OSM block:

```
/sys/bus/platform/drivers/msm_lmh_dcvs/17d41000.qcom,cpucc:qcom,limits-dcvs@0/
lmh_freq_limit
/sys/bus/platform/drivers/msm_lmh_dcvs/17d41000.qcom,cpucc:qcom,limits-dcvs@1/
lmh_freq_limit
```

The actual CPU running frequency is

Perf cluster	- /d/clk/perfc1_clk/clk_measure
Power cluster	- /d/clk/pwrcl_clk/clk_measure

## 6.8.9 Other thermal software features

Feature	Description
Battery current limiting	CPU mitigation based on state-of-charge level of battery For more information, see <i>Battery Current Limit (BCL) Overview and Tuning</i> (80-NM328-709)
Voltage restriction	Voltage restriction enables low operating voltage above 0°C by adjusting the required minimum voltage at temperature extremes
Dynamic parameter update	Important parameter sets can be updated at runtime for better OEM-specific dynamic thermal management

## 6.9 Power data flows

The power and thermal use case data flows will be updated in a future revision of this document.

**Table 6-22 Power KPIs**

Test Case	FPS (Observed)	Battery adjusted (mA)	Battery adjusted (mW)
4K Encode @30 FPS +TNR	~30	316.23	1163.45
1080pEncode @30FPS +TNR	~30	154.36	568.17
1080pAVC30+480pAVC30+720pYUV30	~30	174.60	642.73
4kAVC15+480pAVC15+720pYUV15	~15	223.09	821.05
1080pAVC30+480pAVC30+720pYUV30_SHDR3.8+TNR	~30	550.10	2030.14
4kAVC15+480pAVC15+720pYUV15_SHDR3.8+TNR	~15	756.97	2799.05

## 6.10 Performance and memory

Test case	Use case	Data point	QCS610.LE.1.0
Boot (In seconds)	BootTime	Until recorder Init	6.0.6 sec
WarmBoot (In seconds)	NA	Total	1.25 sec
MemoryMap (in MB)	AfterBoot	NonHlos	244.34 MB
		Kernel	70.3 MB
		ION	1.09 MB
		Non-Ion	354.25 MB
		FreeMemory	1378.01 MB
	4k 30 FPS	NonHlos	244.34 MB
		Kernel	70.3 MB
		ION	399.33 MB
		Non-Ion	420.63 MB
		FreeMemory	913.39 MB
Single_1080p30fpsStorage_480pLive_720Yuv_SHDR3.8		NonHlos	244.34 MB
		Kernel	70.32 MB
		ION	200 MB
		Non-Ion	466.49 MB
		FreeMemory	1066.96 MB
RecLatencies (In seconds)	SingleCamera_4k30fps	Connect to FirstVidFrame	4.32 sec
Glass to Glass Latency	4k Live streaming over Wi-Fi	Latency	2.5 sec
	4k Live streaming over LAN	Latency	2.7 sec
	4k Live Camera Preview (YUV) to HDMI display	Latency	175 msec
CPU	4k15_AVC_720p15_AVC_720p10_AVC_480p10_AVC_240p10_AVC_SHDR_EIS_LDC	% Utilization	Overall 40% CPU0-5 @ 1.2 GHz to 1.36 GHz CPU6-7 @ 0.77 GHz
GPU	4k1530_AVC_720p15_AVC_720p10_AVC_480p10_AVC_240p10_AVC_SHDR_EIS_LDC	% Utilization	88.9% @ 845 GHz



**Figure 6-55 Memory map**

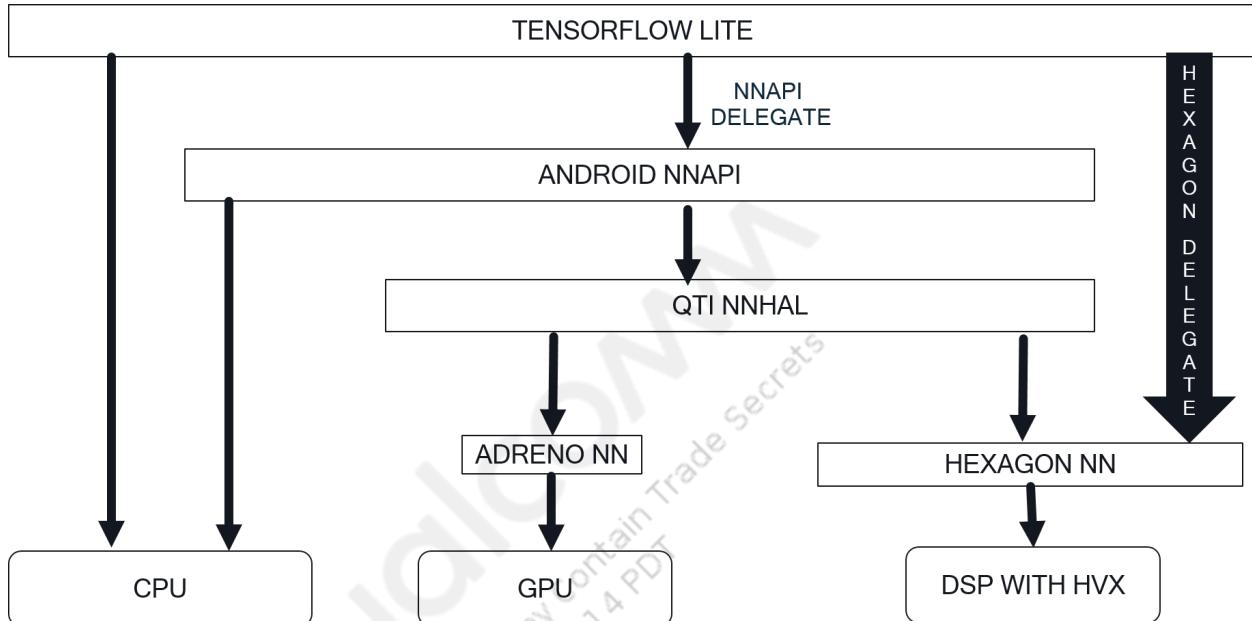
## 6.11 ML

QCS610/QCS410 provides the following inference engines for running the ML application:

- TensorFlow Lite – Available in the LE source code
- Qualcomm® Neural Processing SDK – Available at <https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk>

### 6.11.1 TensorFlow Lite

TensorFlow Lite is an open-source deep learning framework for on-device inference from Google.



**Figure 6-56 TensorFlow Lite architecture**

TensorFlow Lite v.2.2 with hardware acceleration on chipset is shipped with QCS610.

The hardware acceleration is based on NNAPI v1.2 found on Android devices (Android 10, API level 29).

For more information see the following resources:

- TensorFlow Lite – <https://www.tensorflow.org/lite>
- NNAPI drivers – <https://source.android.com/devices/neural-networks>
- NN APIs – <https://developer.android.com/ndk/guides/neuralnetworks/>

To evaluate TensorFlow Lite and test any model after the test app is included in Linux, use `label_image` app. For more information on testing using `label image`, see *QCS610/QCS410 Software Programmer's Guide* (80-PL631-200).

For performance evaluation use benchmark app, see *QCS610/QCS410 Software Programmer's Guide* (80-PL631-200).

For testing supported operations, use VTS. To enable VTS on build, set `PACKAGECONFIG ??= "vts"` in the `poky/meta-qti-ml/recipes/nn-framework/nn-framework.bb` file.

**Table 6-23 TensorFlow Lite performance numbers on QCS610**

Models Names	CPU (2 threads) in msec	NNAPI Delegate in msec	Hexagon Delegate in msec
efficientnet_lite4_classification_int8_2.tflite	155.668	39.0557	300.643
mobilenet_v1_1.0_224_quantized_default_1.tflite	28.9058	8.76232	4.82874

**Table 6-23 TensorFlow Lite performance numbers on QCS610 (cont.)**

Models Names	CPU (2 threads) in msec	NNAPI Delegate in msec	Hexagon Delegate in msec
mobilenet_v2_1.0_224_quantized_default_1.tflite	25.0502	9.50814	5.35499
inception_v1_quant_default_1.tflite	70.7006	14.426	7.58071
inception_v2_quant_default_1.tflite	105.471	150.865	53.4459
inception_v3_quant_default_1.tflite	261.752	30.0935	20.7952
inception_v4_quant_default_1.tflite	456.236	58.1565	40.7786
ssd_mobilenet_v1_default_1.tflite	56.0377	22.673	19.204

## 6.11.2 Qualcomm® Neural Processing SDK

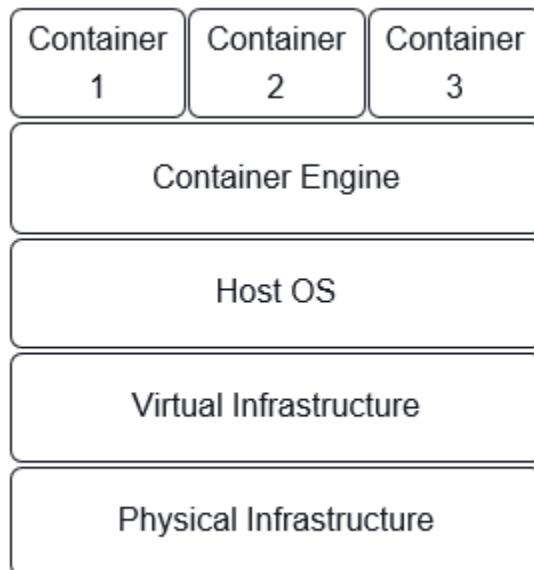
The Qualcomm Neural Processing SDK for artificial intelligence (AI) is designed to help developers run one or more neural network models trained in Caffe/Caffe2, ONNX, or TensorFlow on Snapdragon mobile platforms like CPU, GPU or DSP.

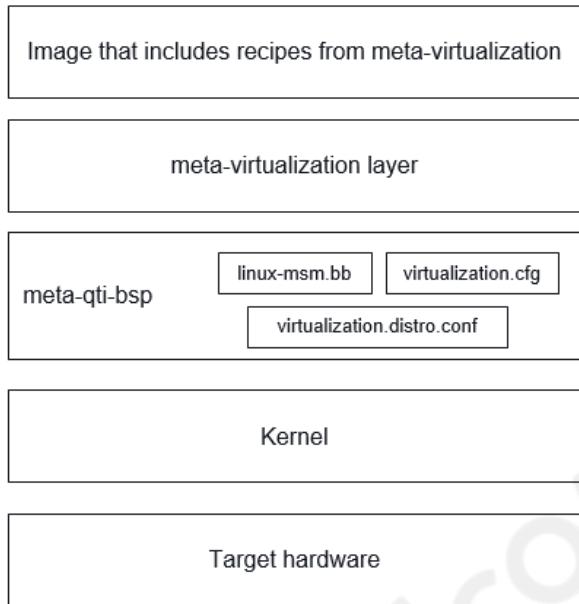
QCS610/QCS410 LE support is added in Qualcomm Neural Processing SDK v1.36 and later. For more information, see <https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk>

## 6.12 Docker support

Docker is a containerization technology that relies on the Linux kernel to provide an isolated execution environment and host resources for applications without having to run a full-fledged virtual machine. For more information, see <https://www.docker.com/>.

If a developer wants to work with Docker on an emulator environment, Docker is installed on the host. The required use case image is built by enabling meta-virtualization bitbake layer.

**Figure 6-57 Generic virtualization layers**



**Figure 6-58 Meta-virtualization and QTI BSP layer meta-qtibsp**

- A suitable target is chosen to run the image. Kernel rests upon the target hardware.
- The kernel-space frameworks required for the user-space virtualization libraries to function are enabled by the virtualization-distros defined by meta-qtibsp.
- The distro enables the drivers required for virtualization if the feature is appended to the DISTRO\_FEATURES variable. This uses the meta-virtualization layer.
- The built image will include recipes from meta-virtualization.

### Integration of meta-virtualization and QTI software

In QTI software releases, meta-virtualization is provided as a dedicated BitBake layer. This layer “\${WORKSPACE}/poky/meta-virtualization” is included in the corresponding bblayers.conf file. Also, “virtualization” is appended to DISTRO\_FEATURES variable for distros that support virtualization. Any one of these being missed would fail in building the distro with virtualization support.

Among the distros provided by QTI, only qti-distro-fullstack-virtualization-debug and qti-distro-fullstack-virtualization-perf support virtualization. To check whether a distro supports virtualization,

1. Run the following commands:

```
$ cd path/to/workspace
$ cd poky/meta-qtibsp/conf/distro
```

2. Open the <distro-name>.conf file and search if DISTRO\_FEATURES += "virtualization" is present.

Not all images include recipes from meta-virtualization. To check whether building an image includes recipes from meta-virtualization, check the packagegroup-<image\_name>.bb file. If “virtualization” is appended to REQUIRED\_DISTRO\_FEATURES variable, the image includes meta-virtualization.

The virtualization.cfg file includes a list of kernel config items, which enable kernel side framework for virtualization. For virtualization user space software libraries built by meta-virtualization layer to function, Kernel must be built with these config items.

The virtualization.cfg file should be included only when virtualization feature is enabled. virtualization.cfg can be found at *poky/meta-qtibsp/recipes-kernel/linux-msm/files/*.

The *linux-msm.inc* file at *poky/meta-qtibsp/recipes-kernel/linux-msm* uses the function *bb.utils.contains* to check if the DISTRO\_FEATURES has virtualization. If DISTRO\_FEATURES has virtualization, then it includes virtualization.cfg file.

```
SRC_URI = "${@bb.utils.contains('DISTRO_FEATURES', 'virtualization', 'file://virtualization.cfg', '', d)}"
```

The *linux-msm.inc* file is included in all *linux-msm\_<version>.bb* files.

For information on testing virtualization on target, see *QCS610/QCS410 Linux Platform Development Kit Software Programmer's Guide* (80-PL631-200).

Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

# 7 QCS610 LEPDK use cases

---

This chapter provides the LEPDK multimedia framework use case data flow. Additionally, it describes the use cases on multi-stream concurrencies.

QTI recommends using GStreamer-based interface to execute the use cases.

For information on QTI GStreamer elements, see [QTI GStreamer elements](#). For information on upstream GStreamer plugins and elements, see the GStreamer documentation at [https://gstreamer.freedesktop.org/documentation/plugins\\_doc.html](https://gstreamer.freedesktop.org/documentation/plugins_doc.html).

## 7.1 Prerequisites

The device must be flashed to execute the following use cases. For the flashing procedure, see *QCS610/QCS410 Software Programmer's Guide* (80-PL631-200).

1. Prepare the device to run the multimedia use cases:

- Set up the camera module:

```
adb shell mkdir -p /etc/camera  
adb shell touch /etc/camera/camxoverridesettings.txt  
adb shell "echo IFEDualClockThreshold=600000000 >> /etc/camera/  
camxoverridesettings.txt"
```

- Set up the display log level:

```
adb shell mkdir -p /data/misc/display  
adb shell "echo 0 > /data/misc/display/gbm_dbg_cfg.txt"  
adb shell sync
```

2. Run adb reboot to reboot the device.

3. Set up GStreamer environment:

```
adb shell source /etc/gstreamer1.0/set_gst_env.sh
```

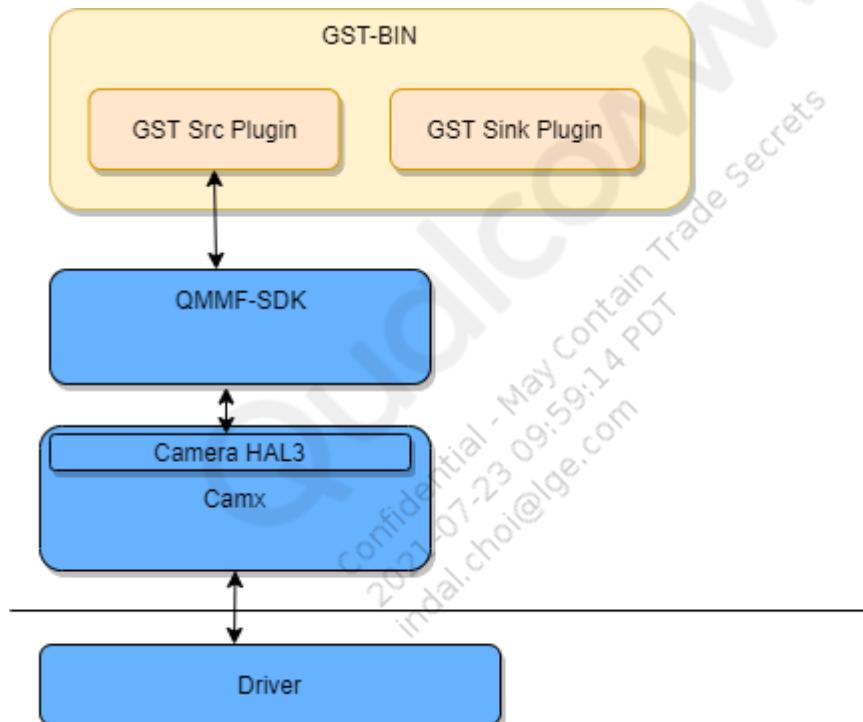
## 7.2 Multimedia stack and control flow

For each of these multimedia subsystems, the following sections describe the stacks and call flows with end-to-end use cases:

This section:

- Camera
- Display and graphics
- Video
- Audio
- GStreamer ML

### 7.2.1 Camera

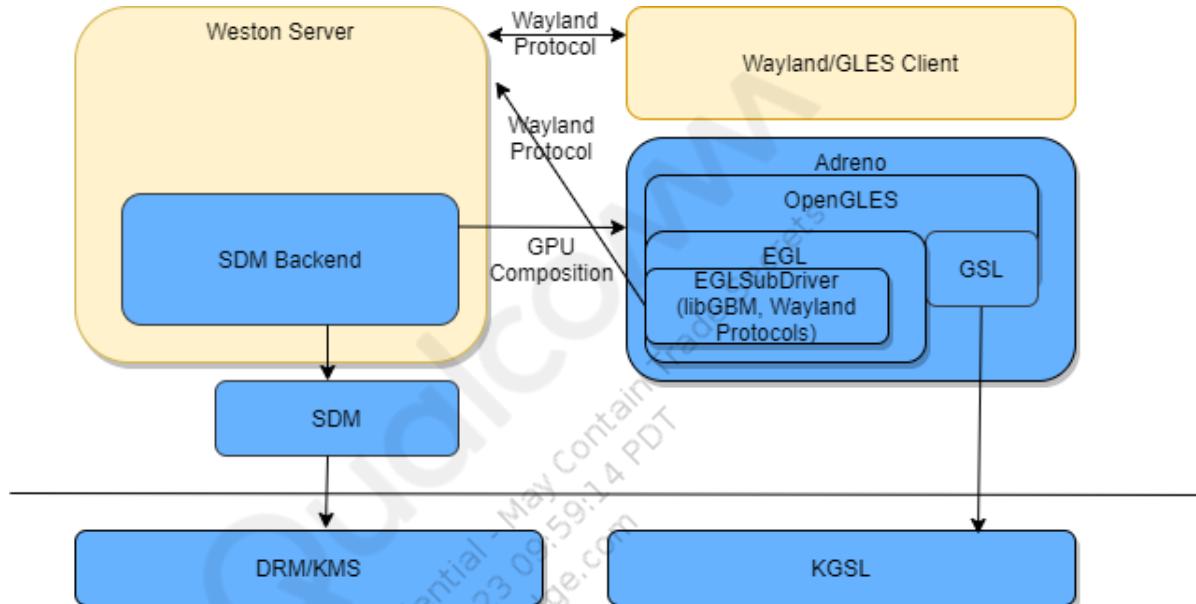


**Figure 7-1 Camera functionality in LEPDK**

- GST SRC plugin (qtiqmmfsrc) is capable of providing multiple encoded (AVC/HEVC) bitstreams and YUV streams, number of streams depends on HW capability.
- GST SRC plugin is a client to QMMF server, QMMF server runs as a daemon in the system, and provides easy RPC APIs to implement camera/recorder use cases.
- The encoded bitstream provided by GST SRC plugin can be used by different types of sink elements such as filesink to save bitstream into the file system, and also can be used by streaming plugins (RTP over TCP or UDP) to send the stream to network.
- Raw YUV stream can be used by waylandsink element to render the camera frames to a physical display to achieve a live camera preview use case, or it can be used by post-processing elements to improve quality or it can be used by ML inferencing elements to do inferencing on live camera.

- QMMF-SDK interacts with HAL3 and further HAL3 interacts with camera backend (Camx) and camera driver to configure sensor and to get a stream from camera sensor.
- Buffers for each camera stream are allocated by QMMF-SDK using libGBM and submitted to HAL3, same buffers are circulated with video subsystem if stream type is AVC/HEVC or with the application if stream type is YUV.

## 7.2.2 Display and graphics



**Figure 7-2 Display and graphics functionality in LEPDK**

- Weston server is a system-level compositor, it caters to the composition/rendering needs, and it runs as a separate process in the system.
- Weston server has a SDM backend that uses SDM (display HAL) to interact with Display hardware.
- SDM has multiple platform-dependent implementations, one of them is for DRM/KMS.
- LibGBM has ION backend and it is used for zero-copy buffer sharing between display and graphics.
- EGL platform-specific driver (EGL sub-driver) to deal with libGBM and Wayland protocols to communicate to Weston compositor.

## 7.2.3 Video

### Video encode

The following are the options to achieve video encode use case:

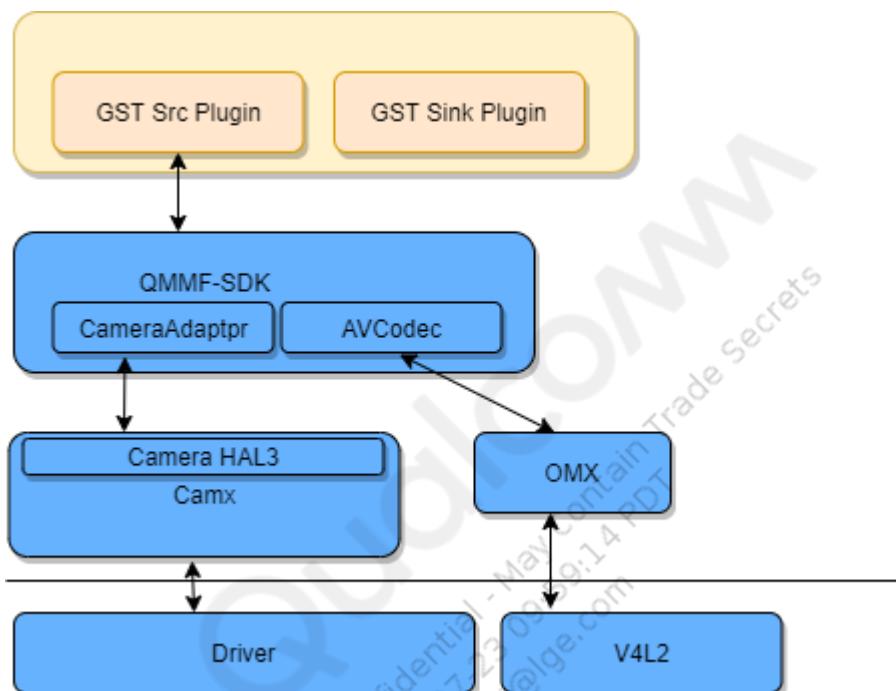
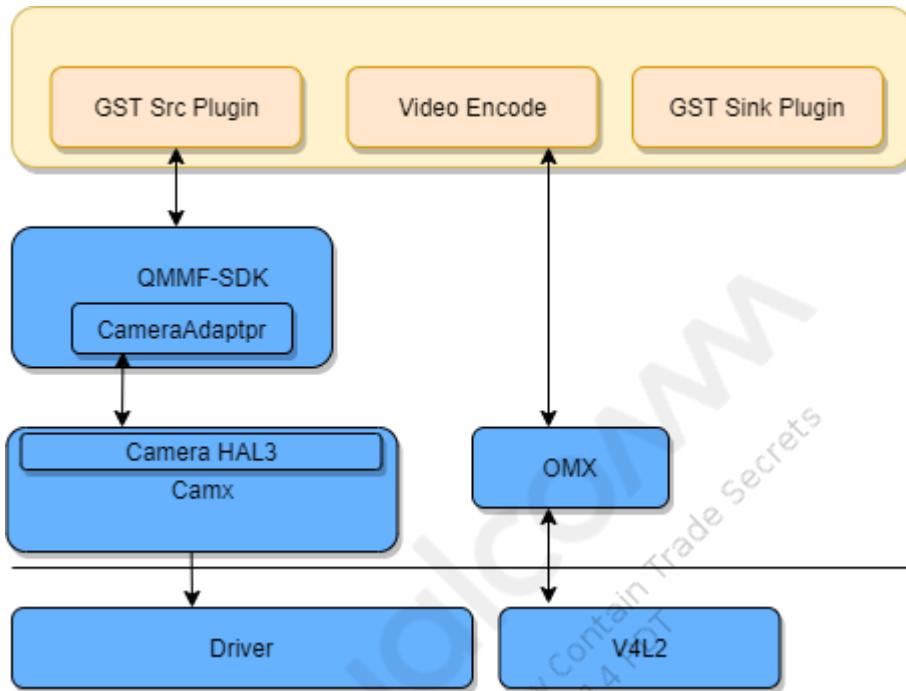


Figure 7-3 Video encode functionality in LEPDK – Option 1

In this option, GST Src plugin (qtinqmmfsrc) provides encoded bitstream stream, this stream further can be used by different types of sink or muxer plugins based on use case.



**Figure 7-4 Video encode functionality in LEPDK – Option 2**

In the second option, GST Src plugin (qtinqmmfsrc) provides RAW YUV stream, this stream further can be used by different types of plugins to build different pipelines such as:

- Different types of post-processing to improve video quality
- ML
- waylandsink to show live camera preview

If there is a need to encode YUV stream then GST OMX encode (omxh264enc or omxh265enc) plugin can be used in the pipeline, as shown in the above diagram.

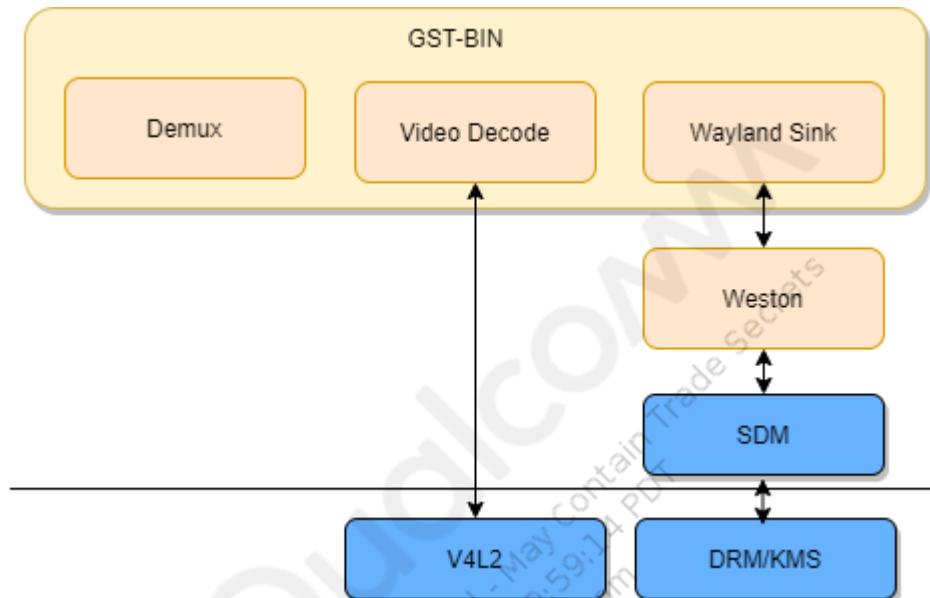
The following are the additional details for both options:

- qtinqmmfsrc GST plugin uses QMMF-SDK client APIs to get multiple camera streams, and these streams can be of YUV or H264/5 format.
- QMMF-SDK (Qualcomm Multimedia framework) can provide 'N' number of RAW YUV or H264/5 encoded streams, 'N' is based on camera & video hardware capability.
- QMMF runs as a server (daemon) in the system, and its clients use binder IPC mechanism to communicate, QMMF exposes helper client APIs which under the hood takes care of doing Binder RPC to the server.
- All kinds of buffer communication (YUV, JPEG, Bitstream) between server and clients are zero-copy.
- AVCCodec adaptation layer within QMMF-SDK framework uses OMX APIs to encode video, and further OMX implementation uses V4L2 interface to encode video.
- qtinqmmfsrc GST plugin sitting on top of QMMF-SDK is capable of providing multiple streams in parallel (multiple src pads), and each stream can be of different formats.

For example, three parallel streams, one stream for local storage, and another YUV stream for live camera preview on a local display, and another one for network streaming.

The next plugin in the pipeline connected to qtiqmmfsrc can be a muxer (mp4mux or mpegs), filesink or a network streaming plugin like tcpserversink.

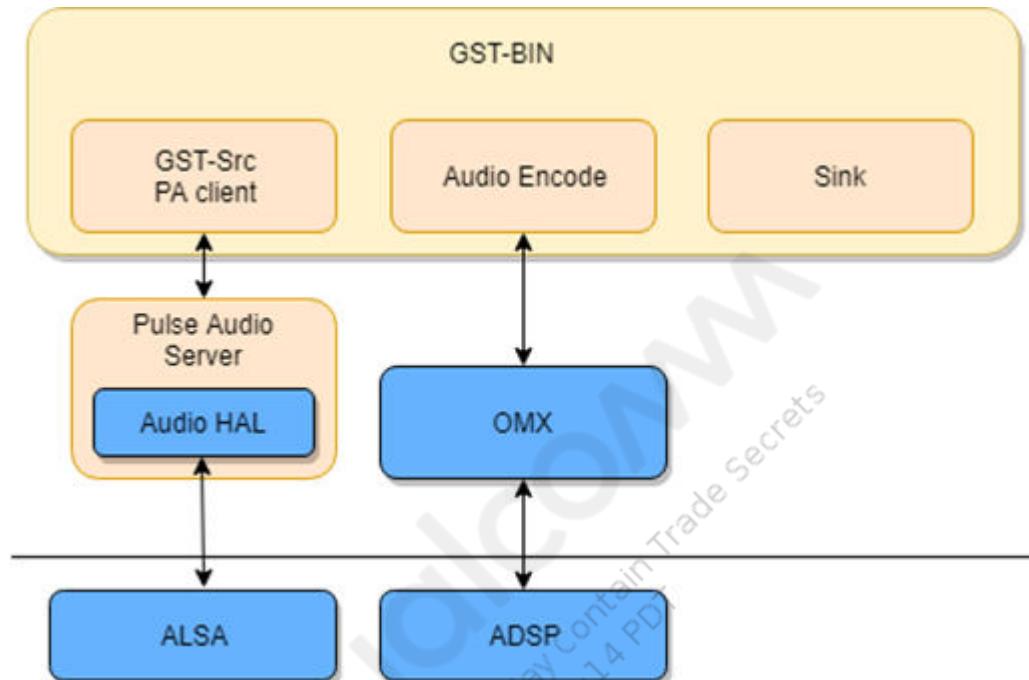
### Video decode



**Figure 7-5 Video decode functionality in LEPDK**

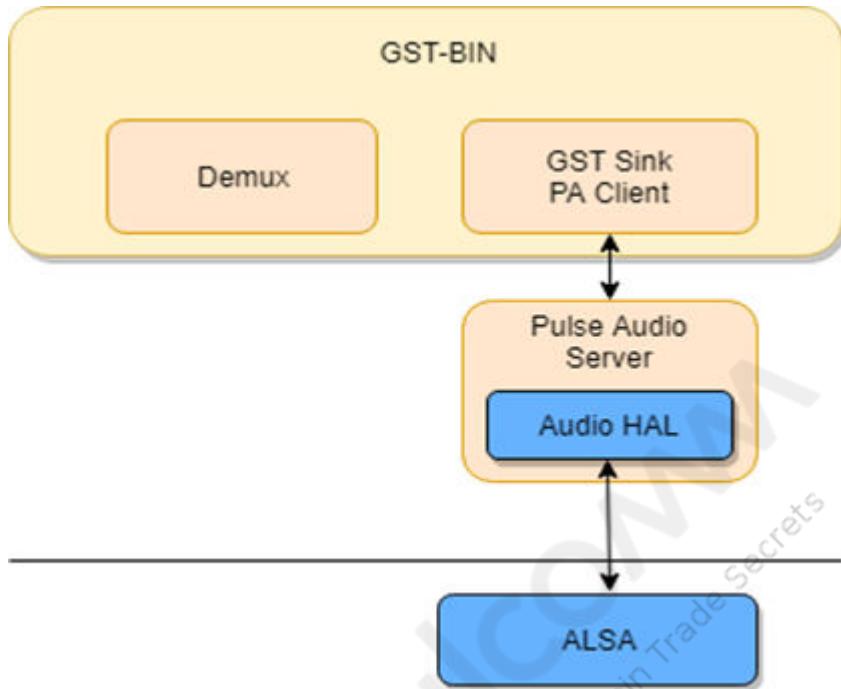
- Video Decode plugin (qtivdec) GST plugin uses V4L2 IOCTLs to decode H264/H265 bitstream.
- This plugin is mainly used in playback and transcode use-cases.
- For playback use case, waylandsink receives GBM buffers (output of decoder), further waylandsink passes GBM buffers to the weston server via wayland protocol for composition. GBM buffers shared between three components - video decode, waylandsink and weston compositor are zero-copy.
- For transcode use cases the output of video decoder is processed by encoder element either JPEG or OMX encode.
- And for transform (rotate/scale/flip) use cases the output of video decoder is processed by transform element and then further by encoder element.

### 7.2.4 Audio



**Figure 7-6 Audio capture functionality in LEPDK**

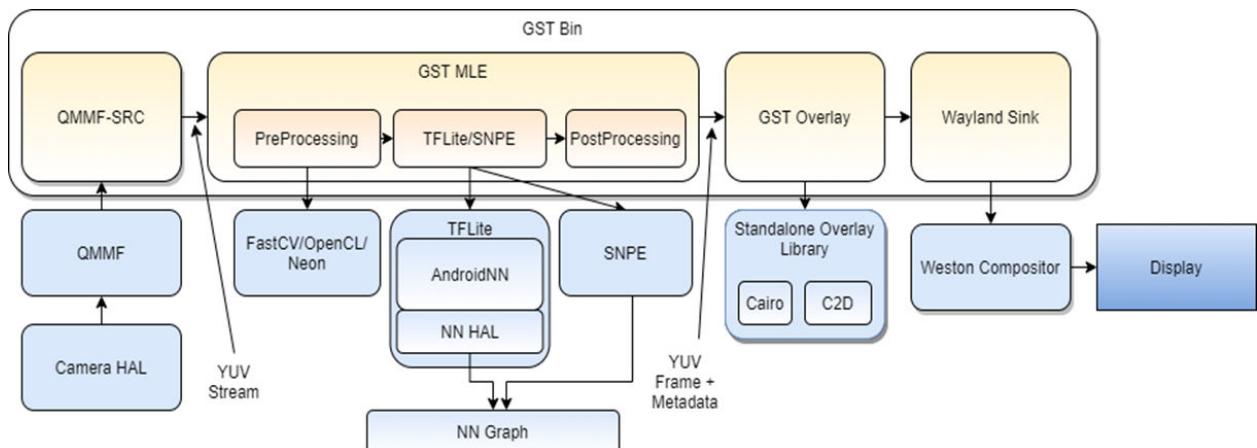
- GST pulsesrc element uses PulseAudio clients APIs to interact with PulseAudio server to get PCM audio samples.
- PulseAudio server under the hood uses Audio HAL as a pluggable module to interact with ALSA driver.
- Audio Encode plugin can be SW (CPU) or HW (ADSP) accelerated.
- Sink Element can be a file sink, muxer or network streaming element to stream audio to network depending on the use case.



**Figure 7-7 Audio playback functionality in LEPDK**

- There are two types of pulse sink elements, one is pulsesink and another one is pulsedirectsink, both plugins are client to PulseAudio server.
- pulsesink element is only capable of rendering PCM samples, audio decode plugin needs to be present in the pipeline before the sink element if a user wants to build a pipeline to play encoded audio stream.
- pulsedirectsink element is capable of rendering compressed audio (compressed offload to ADSP), audio decoder plugin is not needed in pipeline if pulsedirectsink element is used as a sink element.
- PulseAudio supports pass through compressed audio to HAL and HAL further passes to ADSP.

### 7.2.5 GStreamer ML



**Figure 7-8 High-level architecture of ML use case**

- The source or provider for the GST ML engine (MLE) could be a live camera source via the `qtiqmmfsrc` plugin or a offline video via the `filesrc` plugin.
- The `qtiqmmfsrc` plugin can provide the YUV stream directly, whereas in case of offline video additional plugins are needed to obtain the YUV stream
- GST MLE plugins are a set of ML plugins that allow multiple simultaneous inferencing on different inference backends like Qualcomm Neural Processing SDK, TensorFlow Lite, and Qualcomm DSP Hexagon Neural Network (HexagonNN) accelerator.
- It has three main stages – preprocessing, inferencing and postprocessing:
  - The preprocessing of input stream involves color convert, downscale, mean subtraction and padding. It is done using Qualcomm Computer Vision SDK, OpenCL, or Arm Neon.
  - The inferencing is done using Qualcomm Neural Processing SDK, TensorFlow Lite or HexagonNN based on the plugin used. The compute engine/core for the backend can be DSP or CPU and is configurable from the plugin.
  - The postprocessing involves processing the inferencing result and supports types such as classification, detection, segmentation, and pose estimation. The postprocessing data is then attached as `MLMeta` to the GST buffer and passed on to the `qtioverlay` plugin.
- The `qtioverlay` plugin interprets the ML metadata and draws appropriate overlay on the buffer which can be classification label, detection bounding box, segmentation map, or pose graph.
- The stream with overlay can then be rendered on to the display or encoded, and stored or streamed over network.

For more information, see [QTI GStreamer elements](#).

## 7.3 Player– Single-stream playback

This section provides information on executing few playback use cases from `gst-launch`. The **qtivdec gstreamer** video decoder plug-in is a V4L2 video decoder which is used for hardware-accelerated video decoding. For more information, see [qtivdec](#).

### 7.3.1 Start Weston

Run the following command to check if Weston is running:

```
/# ps | grep weston
```

The following output is displayed.

```
xxxx root      1h07 weston --idle-time=4294967 --log=/tmp/weston.log
```

If the preceding output is not displayed, ensure that the external display monitor (1080p) is connected via HDMI, and then run the following commands to start Weston.

```
adb root
adb shell
export XDG_RUNTIME_DIR=/dev/socket/weston
/usr/bin/weston --tty=1 --idle-time=0 &
```

For further assistance, contact the QTI support team.

### 7.3.2 4K video file playback

Ensure that Weston is running. For more information, see [Start Weston](#).

- Run the following commands to generate 4K mp4 file for playback.

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_4k_avc.mp4"
```

- The commands uses the [qtiqmmfsrc](#) plugin to generate the hardware-encoded H264 stream.
- The stream parameters such as width, height, framerate, and format are specified as a capsfilter.
- This stream is multiplexed using the mp4mux plug-in and dumped using the filesink plug-in.

- Run the following commands to start video playback.

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_4k_avc.mp4 ! qtdemux name=demux demux. ! queue !
h264parse ! qtivdec ! video/x-raw\ (memory:GBM\),compression=ubwc !
waylandsink fullscreen=true
```

- The commands use the filesrc plug-in to read the file.
- The file is demultiplexed by the qtdemux plugin, parsed by the h264parse plug-in, decoded by the qtivdec plug-in.
- The decoded YUV buffers are rendered to HDMI display by the waylandsink plug-in, which is client to Weston server.

### 7.3.3 1080p video file playback

Ensure that Weston is running. For more information, see [Start Weston](#).

- Run the following commands to generate 1080p video file for playback:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_avc.mp4"
```

- The commands use the qtiqmmfsrc plugin to generate the hardware-encoded H264 stream.
- The stream parameters such as width, height, framerate, and format are specified as a capsfilter.
- This stream is then multiplexed using the mp4mux plug-in and dumped using the filesink plug-in.

- Run the following command to start video playback:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_avc.mp4 ! qtdemux name=demux demux. ! queue !
```

```
h264parse ! qtivdec ! video/x-raw\ (memory:GBM\),compression=ubwc !
waylandsink fullscreen=true
```

- The commands use the filesrc plug-in to read the file.
- The file is demultiplexed by the qtdemux plugin, parsed by the h264parse plugin, decoded by the qtivdec plugin.
- The decoded YUV buffers are rendered to HDMI display by the waylandsink plugin, which is client to Weston server.

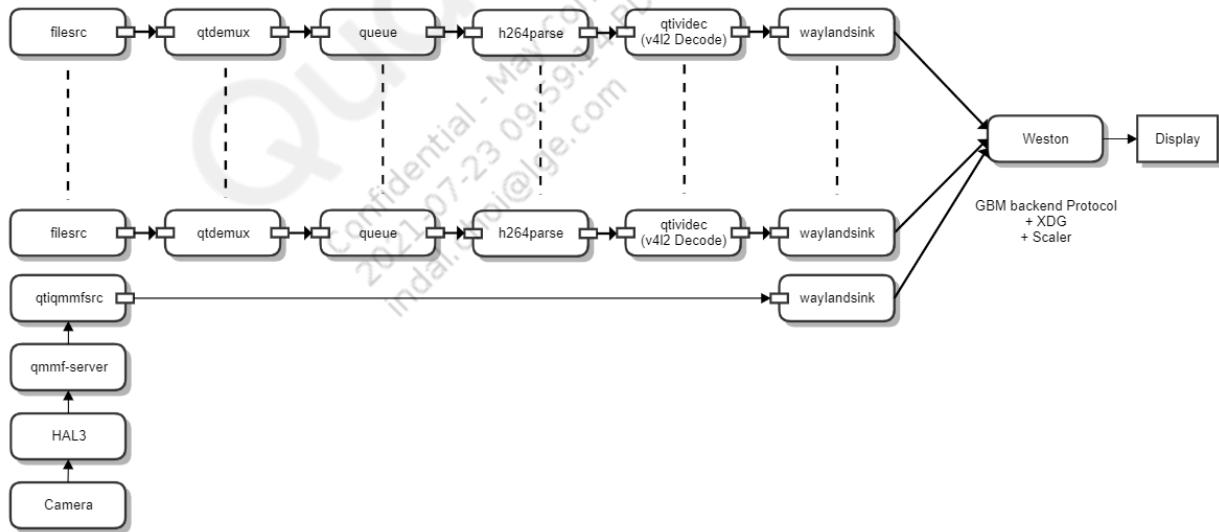
## 7.4 Player – Multi-stream playback

For information on QTI GStreamer elements, see [QTI GStreamer elements](#).

### 7.4.1 Eight stream playback, seven 720p offline video, and one 720p live camera preview

#### Prerequisites:

Ensure that Weston is running. For more information, see [Start Weston](#).



1. Run the following command to create a test video. Press **Ctrl + C** to stop the recording.

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1280,height=720,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_720p.mp4"
```

2. Create `7streamsAnd1LiveCamera.sh`. Copy the following commands to `7streamsAnd1LiveCamera.sh`:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=960 y=0 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=0 y=0 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=960 y=270 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=0 y=270 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=960 y=540 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=0 y=540 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_720p.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=960 y=810 width=960 height=270 &

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 -e qtiqmmfsrc
name=qmmf ! "video/x-
raw(memory:GBM),format=NV12,camera=0,width=640,height=480,framerate=30/1" !
waylandsink sync=false x=0 y=810 width=960 height=270 enable-last-
sample=false &
```

3. To run the use case, do the following:

- a. Push `7streamsAnd1LiveCamera.sh` on the device

```
adb push 7streamsAnd1LiveCamera.sh /data/
```

- b. Run the script

```
adb shell
```

```
sh /data/7streamsAnd1LiveCamera.sh
```

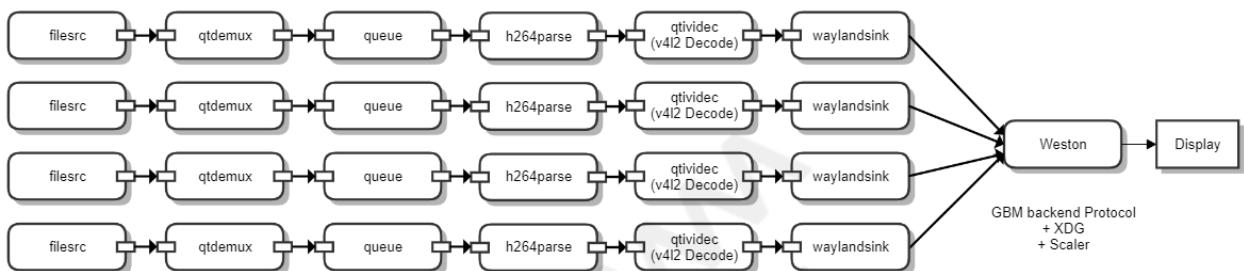
To stop the use case, from another terminal window, run the following command:

```
adb shell killall gst-launch-1.0
```

#### **7.4.2 Four 1080p stream side-by-side playback**

## Prerequisite:

Ensure that Weston is running. For more information, see [Start Weston](#).



1. Run the following command to create a test video. Press **Ctrl + C** to stop the recording.

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux_1080p_avc.mp4"
```

2. Create 4streams.sh. Copy the following commands to 4streams.sh.

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc  
location=/data/mux_1080p_avc.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !  
waylandsink x=960 y=0 width=960 height=540 &  
  
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc  
location=/data/mux_1080p_avc.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !  
waylandsink x=0 y=0 width=960 height=540 &  
  
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc  
location=/data/mux_1080p_avc.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !  
waylandsink x=960 y=540 width=960 height=540 &  
  
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc  
location=/data/mux_1080p_avc.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !  
waylandsink x=0 y=540 width=960 height=540 &
```

3. To run the use case, do the following:

- a. Push 4streams.sh on the device

```
adb push 4streams.sh /data/
```

- #### b. Run the script

```
adb shell  
sh /data/4streams.sh
```

To stop the use case, from another terminal window, run the following command:

```
adb shell killall qst-launch-1.0
```

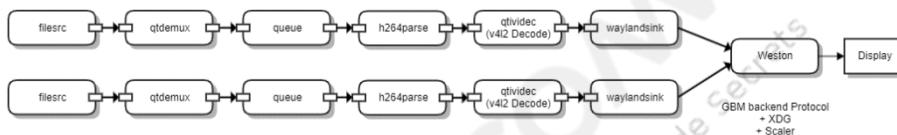
### 7.4.3 Two 1080p stream picture in picture playback

The display windows can be placed anywhere on the screen within the screen boundaries. Each window can have different destination rectangles. These rectangles can be overlapping and non-overlapping with each other. The destination rectangle can be smaller than the source rectangle, which means the downscale is done by Weston compositor.

This example illustrates two 1080p resolution streams picture in picture (PiP) playback, one stream is overlapped on top of the other and shown in 480x270 destination rectangle.

#### Prerequisites:

Ensure that Weston is running. For more information, see [Start Weston](#).



- Run the following command to create a test video. Press **Ctrl + C** to stop the recording.

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_1080p_avc.mp4"
```

- Create `pictureInPicture.sh`. Copy the following commands to `pictureInPicture.sh`.

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_1080p_avc.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink fullscreen=true &
sleep 0.5
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_1080p_avc.mp4 ! qtdemux ! queue ! h264parse ! qtivdec !
waylandsink x=1440 y=810 width=480 height=270 &
```

- To run the use case, do the following:

- Push `pictureInPicture.sh` on the device

```
adb push pictureInPicture.sh /data/
```

- Run the script

```
adb shell
sh /data/pictureInPicture.sh
```

To stop the use case, from another terminal window, run the following command:

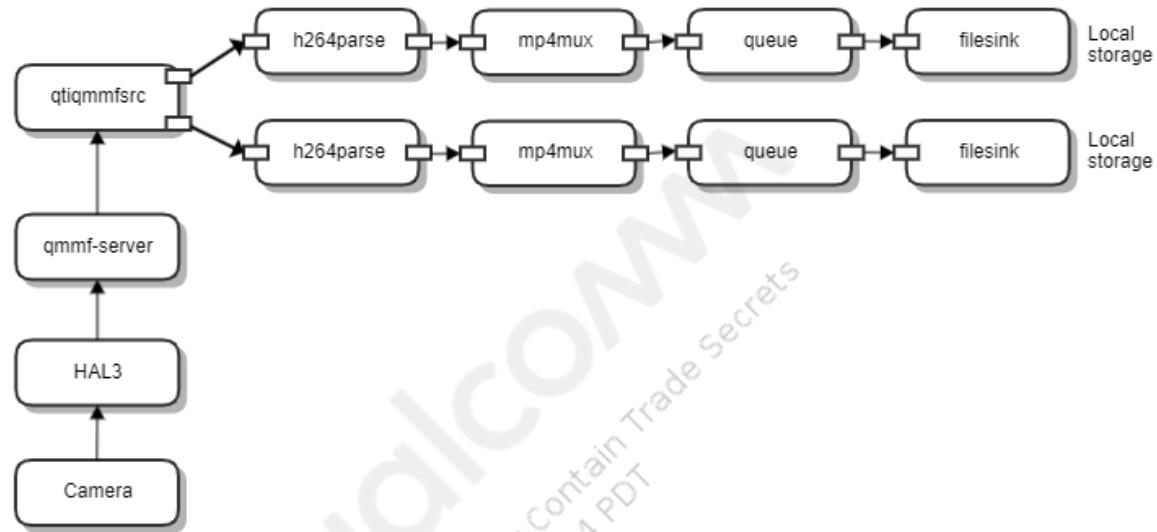
```
adb shell killall gst-launch-1.0
```

## 7.5 Recorder – Two stream concurrencies

Before you execute any of the commands, complete the [Prerequisites](#).

For information on QTI GStreamer elements, see [QTI GStreamer elements](#). For information on upstream GStreamer plugins and elements, see the GStreamer documentation at [https://gstreamer.freedesktop.org/documentation/plugins\\_doc.html](https://gstreamer.freedesktop.org/documentation/plugins_doc.html).

### 7.5.1 Two independent H264 streams both stored to file



The following `gst-launch` pipeline provides simultaneous capture, encode, and storage of two H264/AVC encoded video streams, one is 4K resolution and the other is 480p resolution:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux_4k.mp4" qmmf. ! video/x-
h264,format=NV12,width=640,height=480,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux_480.mp4"
```

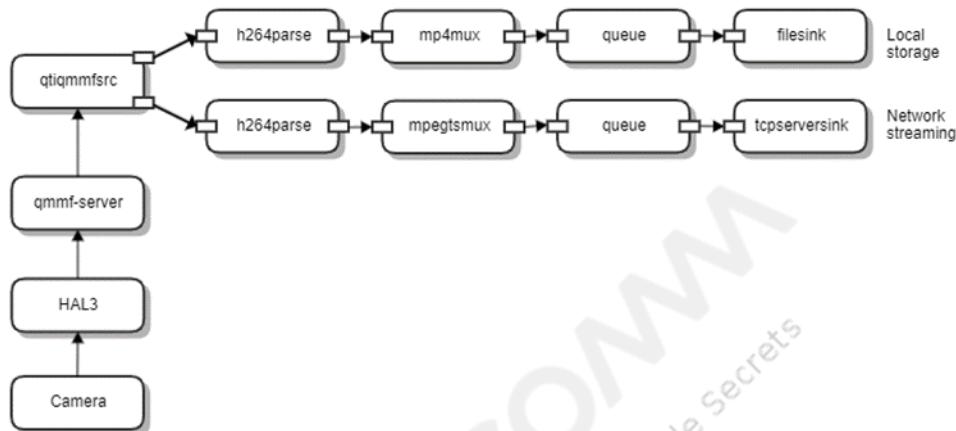
1. The `qtinqmmfsrc` element is used to capture and encode both the video streams.
2. The `h264parse` and `mp4mux` elements process the buffers and prepare them for storage.
3. The `queue` element makes sure each path/track runs independently of the other.
4. The `filesink` elements stores the buffers in the corresponding file.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following `adb pull` command, and then play content on host PC.

```
adb pull /data/mux_4k.mp4
adb pull /data/mux_480.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

## 7.5.2 Two independent H264 streams, one stored to file and another streamed over TCP



The following `gst-launch` pipeline provides the capture of two 1080p resolution H264/AVC encoded video streams. One stream is stored to file while the other is streamed over network(TCP).

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux.mp4" qmmf. ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse config-
interval=1 ! mpegtsmux name=muxer ! queue ! tcpserversink port=8900
host=127.0.0.1
```

1. The `qtiqmmfsrc` element is used to capture and encode both video streams.
2. The `h264parse` and `mp4mux` elements process the buffers from the first stream and prepare them for storage.
3. The `filesink` elements stores the buffers from first stream in the file.
4. The `h264parse` and `mpegtsmux` elements process the buffers from the second stream and prepare them for streaming.
5. The `tcpserversink` element then sends the buffers on the network. The `queue` element makes sure each path/track runs independently of the other.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following `adb pull` command, and then play content on host PC.

```
adb pull /data/mux.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

### 7.5.2.1 View the TCP stream on the host machine (PC)

**Prerequisites:**

1. Install adb and VLC Media Player on the host machine.
2. Set the binary/executable paths in the environment variables.

For more information, contact QTI support.

On Linux host, run the following commands:

```
adb forward tcp:8900 tcp:8900
vlc -vvv tcp://127.0.0.1:8900
```

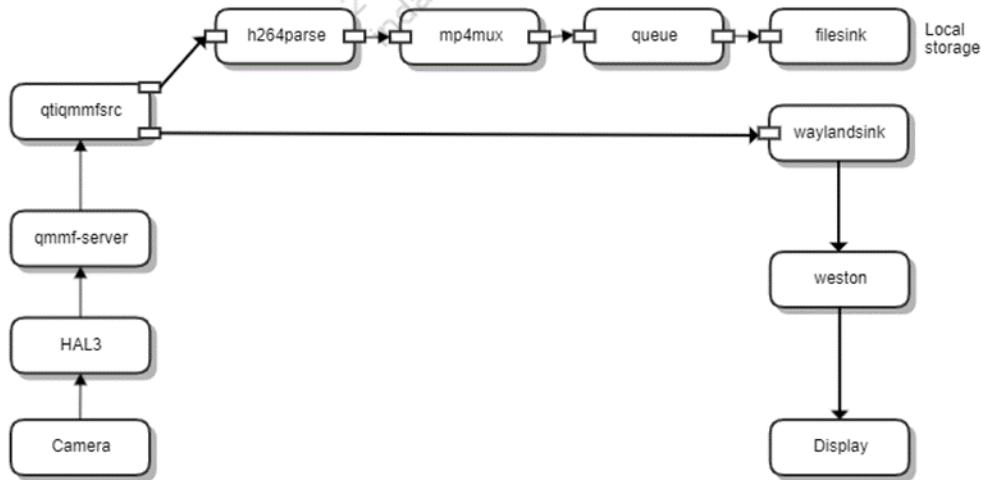
On Windows host:

1. Run adb forward tcp:8900 tcp:8900.
2. Start the VLC Media Player.
3. Select Media/Open Network Stream... (or press **CTRL + N**)
4. Enter the network URL – *tcp://127.0.0.1:8900*
5. Click **Play**.

### 7.5.3 Two independent streams, one H264 stream stored to file and one live camera preview

**Prerequisites:**

Ensure that Weston is running. For more information, see [Start Weston](#).



The following `gst-launch` pipeline provides simultaneous record and live preview on display use case:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 -e qtqmmfsrc
name=qmmf ! "video/x-
raw(memory:GBM),format=NV12,width=1920,height=1080,framerate=30/1" !
```

```
waylandsink sync=false fullscreen=true enable-last-sample=false qmmf. !
video/x-h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux1.mp4"
```

1. The `qtiqmmfsrc` element is used to generate 4K resolution encoded video streams and 1080p resolution YUV stream.
2. The `h264parse` and `mp4mux` elements process the encoded video stream
3. The `filesink` element stores the stream in a file on the device.

The YUV stream is rendered on the display panel by the `waylandsink` element.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following `adb pull` command, and then play content on host PC.

```
adb pull /data/mux1.mp4
```

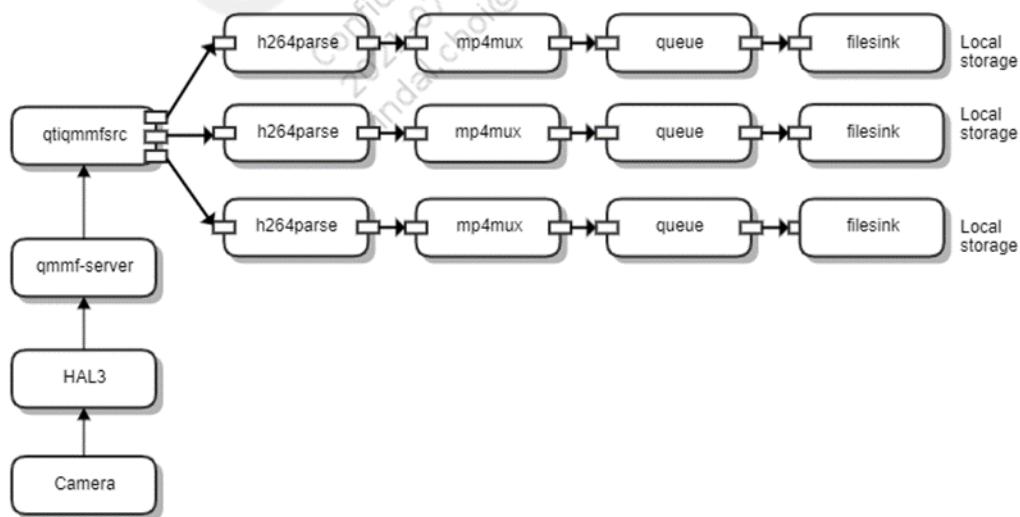
To play the content on the device, see [Player– Single-stream playback](#).

## 7.6 Recorder – Three stream concurrencies

Before you execute any of the following commands, complete the [Prerequisites](#).

For information on QTI GStreamer elements, see [QTI GStreamer elements](#). For information on upstream GStreamer plugins and elements, see the GStreamer documentation at [https://gstreamer.freedesktop.org/documentation/plugins\\_doc.html](https://gstreamer.freedesktop.org/documentation/plugins_doc.html).

### 7.6.1 Three video streams all of them stored to file



The following `gst-launch` pipeline provides simultaneous 1080p video record use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux1.mp4" qmmf. ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux2.mp4" qmmf. ! video/x-
```

```
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux3.mp4"
```

1. The qtiqmmfsrc element is used to capture and encode all three video streams.
2. The h264parse, mp4mux elements process the buffers and prepare them for storage.
3. The queue element makes sure each path/track runs independently of the other.
4. The filesink elements stores the buffers in the corresponding file.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following adb pull command, and then play content on host PC.

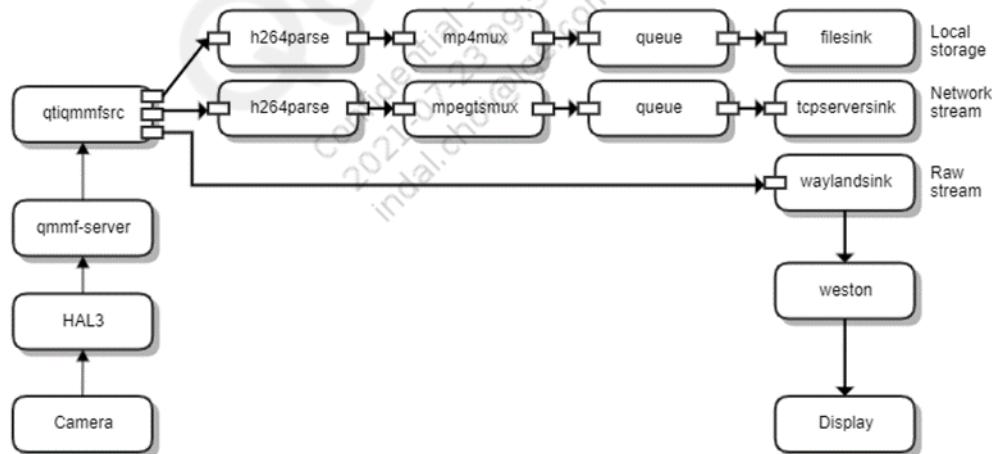
```
adb pull /data/mux1.mp4
adb pull /data/mux2.mp4
adb pull /data/mux3.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

## 7.6.2 Three video streams, one stored to file, another streamed over TCP, and another live preview on display

### Prerequisites:

Ensure that Weston is running. For more information, see [Start Weston](#).



The following gst-launch pipeline provides simultaneous 4K record, 1080p live preview on display, and 480p streaming use case.

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 -e qtiqmmfsrc
name=qmmf qmmf.video_0 ! video/x-
h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse ! mp4mux !
queue ! filesink location="/data/mux1.mp4" qmmf.video_1 ! video/x-
h264,format=NV12,width=640,height=480,framerate=30/1 ! h264parse config-
interval=1 ! mpegtsmux name=muxer ! queue ! tcpserversink port=8900
host=127.0.0.1 qmmf.video_2 ! "video/x-
raw(memory:GBM),format=NV12,width=1920,height=1080,framerate=30/1" !
waylandsink sync=false fullscreen=true enable-last-sample=false
```

1. The `qtiqmmfsrc` element is used to generate two encoded video streams (4K and 480p resolution) and one 1080p YUV stream.
2. The first encoded video stream is processed by `h264parse` and `mp4mux` elements.
3. The `filesink` element stores it in a file on the device.
4. The second encoded stream is processed by `h264parse` and `mpegtsmux` elements, and streamed over TCP by the `tcperversink` element.
5. The third raw YUV stream is rendered on the display panel by the `waylandsink` element.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following adb pull command, and then play content on host PC.

```
adb pull /data/mux1.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

To view the TCP stream on the host machine, see [View the TCP stream on the host machine \(PC\)](#).

## 7.7 Recorder – Five stream concurrencies

Before you execute any of the following commands, complete the [Prerequisites](#).

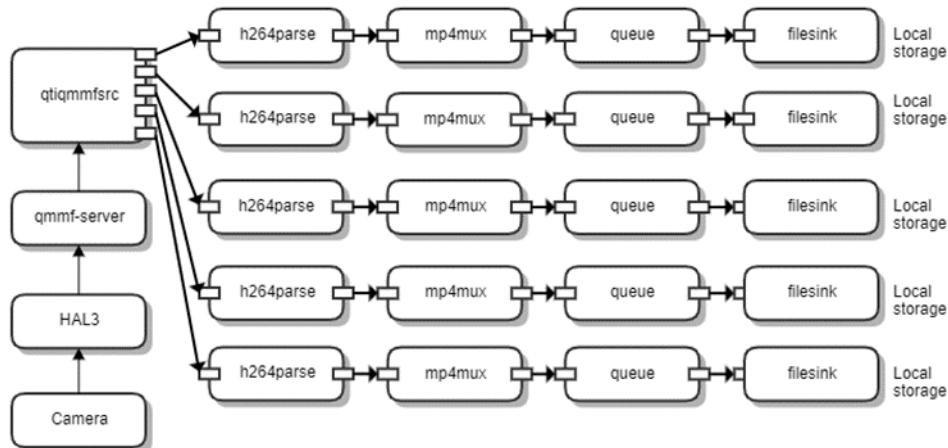
For information on QTI GStreamer elements, see [QTI GStreamer elements](#). For information on upstream GStreamer plugins and elements, see the GStreamer documentation at [https://gstreamer.freedesktop.org/documentation/plugins\\_doc.html](https://gstreamer.freedesktop.org/documentation/plugins_doc.html).

### 7.7.1 Five H264/AVC encoded streams with sHDR enabled

#### Prerequisites:

If `disableAFDStatsProcessing` is not present in the `/etc/camera/camxoverridesettings.txt`, run the following command to ensure that there is camera override setting for sHDR:

```
adb shell "echo disableAFDStatsProcessing=TRUE >> /etc/camera/camxoverridesettings.txt"
adb shell pkill qmmf-server
```



The following `gst-launch` pipelines provides simultaneous five stream video record use cases with sHDR enabled:

- One 4K, two 720p, one 480p, and one 240p, with sHDR

```
gst-launch-1.0 -e qtiqmmfsrc video_2::source-index=1 video_3::source-
index=1 video_4::source-index=1 name=qmmf shdr=TRUE ! video/x-
h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink async=true location="/data/mux1.mp4" qmmf. !
video/x-h264,format=NV12,width=1280,height=720, framerate=15/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux2.mp4"
qmmf. ! video/x-h264,format=NV12,width=1280,height=720, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux3.mp4"
qmmf. ! video/x-h264,format=NV12,width=720, height=480, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux4.mp4"
qmmf. ! video/x-h264,format=NV12,width=320, height=240, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux5.mp4"
```

- One 1080p, two 720p, one 480p, and one 240p, with sHDR

```
gst-launch-1.0 -e qtiqmmfsrc video_2::source-index=1 video_3::source-
index=1 video_4::source-index=1 name=qmmf shdr=TRUE ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink async=true location="/data/mux1.mp4" qmmf. !
video/x-h264,format=NV12,width=1280,height=720, framerate=15/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux2.mp4"
qmmf. ! video/x-h264,format=NV12,width=1280,height=720, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux3.mp4"
qmmf. ! video/x-h264,format=NV12,width=720, height=480, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux4.mp4"
qmmf. ! video/x-h264,format=NV12,width=320, height=240, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux5.mp4"
```

- The `qtiqmmfsrc` element captures and encodes one 4K resolution and two 720p resolution streams from the camera, and generates the remaining 480p and 240p video streams from it.

1. The `h264parse`, `mp4mux` elements process the buffers and prepare them for storage.
2. The `queue` element makes sure each path/track runs independently of the other.
3. The `filesink` elements stores the buffers in the corresponding file.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following adb pull command, and then play content on host PC.

```
adb pull /data/mux1.mp4
adb pull /data/mux2.mp4
adb pull /data/mux3.mp4
adb pull /data/mux4.mp4
adb pull /data/mux5.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

## 7.7.2 Five H264/AVC encoded streams with sHDR and LDC enabled

### Prerequisites:

- If disableAFDStatsProcessing is not present in the /etc/camera/camxoverridettings.txt, run the following command to ensure that there is cameraoverride setting for sHDR:

```
adb shell "echo disableAFDStatsProcessing=TRUE >> /etc/camera/
camxoverridettings.txt"
adb shell pkill qmmf-server
```

The following gst-launch pipelines provides simultaneous five stream video record use cases with sHDR and LDC enabled.

- One 4K, two 720p, one 480p, and one 240p, with sHDR and LDC

```
gst-launch-1.0 -e qtiqmmfsrc video_2::source-index=1 video_3::source-
index=1 video_4::source-index=1 name=qmmf shdr=TRUE ldc=TRUE ! video/x-
h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink async=true location="/data/mux1.mp4" qmmf. !
video/x-h264,format=NV12,width=1280,height=720, framerate=15/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux2.mp4" qmmf. !
video/x-h264,format=NV12,width=1280,height=720, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux3.mp4" qmmf. !
video/x-h264,format=NV12,width=720, height=480, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux4.mp4" qmmf. !
video/x-h264,format=NV12,width=320, height=240, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux5.mp4"
```

- One 1080p, two 720p, one 480p, and one 240p, with sHDR and LDC

```
gst-launch-1.0 -e qtiqmmfsrc video_2::source-index=1 video_3::source-
index=1 video_4::source-index=1 name=qmmf shdr=TRUE ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink async=true location="/data/mux1.mp4" qmmf. !
video/x-h264,format=NV12,width=1280,height=720, framerate=15/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux2.mp4" qmmf. !
video/x-h264,format=NV12,width=1280,height=720, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux3.mp4" qmmf. !
video/x-h264,format=NV12,width=720, height=480, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux4.mp4" qmmf. !
video/x-h264,format=NV12,width=320, height=240, framerate=10/1 !
h264parse ! mp4mux ! queue ! filesink async=true location="/data/mux5.mp4"
```

- The qtiqmmfsrc element captures and encodes one 4K resolution and two 720p resolution streams from the camera, and generates the remaining 480p and 240p video streams from it.
- The h264parse, mp4mux elements process the buffers and prepare them for storage.
- The queue element makes sure each path/track runs independently of the other.
- The filesink elements stores the buffers in the corresponding file.

To stop the use case, press **CTRL + C**, pull the recorded content from device using the following adb pull command, and then play content on host PC.

```
adb pull /data/mux1.mp4
adb pull /data/mux2.mp4
adb pull /data/mux3.mp4
adb pull /data/mux4.mp4
adb pull /data/mux5.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

## 7.8 Camera parameters

This section covers the static and dynamic camera parameters.

### 7.8.1 Static parameters

#### LDC

To run the LDC use case , run the following command:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ldc=TRUE qmmf.video_0 ! video/x-h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux.mp4"
```

To stop the use case, press **CTRL + C**, pull the recorded content out from device using the following adb pull command, and then play content on host PC.

```
adb pull /data/mux.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

#### sHDR

##### Prerequisites:

If disableAFDStatsProcessing is not present in the /etc/camera/camxoverridesettings.txt, run the following command to ensure that cameraoverride is enabled for sHDR:

```
adb shell "echo disableAFDStatsProcessing=TRUE >>/etc/camera/camxoverridesettings.txt"
adb shell pkill qmmf-server
```

To enable sHDR from the qtiqmmfsrc element, run the following gst-launch command:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf shdr=TRUE qmmf.video_0 ! video/x-h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux.mp4"
```

#### EIS

##### Prerequisites:

A gyro sensor should be connected. For more information, contact QTI support. To enable EIS from the qtiqmmfsrc element, run the following gst-launch command:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf eis=TRUE qmmf.video_0 ! video/x-h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux.mp4"
```

To stop the use case, press **CTRL + C**, pull the recorded content out from device using the following adb pull command, and then play content on host PC.

```
adb pull /data/mux.mp4
```

To play the content on the device, see [Player– Single-stream playback](#).

## 7.8.2 Dynamic parameters update using the gst-pipeline-app

The QTI gst-pipeline-app is the helper tool that exposes the same capability as GStreamer's gst-launch-1.0 tool. It also provides an option to set runtime properties for GST elements. For more information, see [QTI gst-pipeline-app](#).

### Prerequisites:

Ensure that Weston is running. Fore more information, see [Start Weston](#).

To update camera properties of the qtiqmmfsrc plugin at runtime and observe the changes on the display, do the following:

1. Run the following pipeline through gst-pipeline-app.

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-pipeline-app qtiqmmfsrc name=qmmf ! "video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1" ! waylandsink sync=false fullscreen=true enable-last-sample=false
```

2. Set the pipeline to playing state, select the **PLAYING** option
3. Choose **Plugin Mode**.
4. Type the name of the plugin to be controlled or enter the option number:

```
qmmf
```

All the parameters that can be updated dynamically are displayed:

```
#####
##### MENU #####
#####
===== Plugin Properties =====
( 0) adrcc : Automatic Dynamic Range Compression
( 1) effect : Effect applied on the camera frames
( 2) scene : Camera optimizations depending on the scene
( 3) antihanding : Camera antihanding routine for the current illumination condition
( 4) sharpness : Image Sharpness Strength
( 5) contrast : Image Contrast Strength
( 6) saturation : Image Saturation Strength
( 7) iso-mode : ISO exposure mode
( 8) manual-iso-value : Manual exposure ISO value. Used when the ISO mode is set to 'manual'
( 9) exposure-mode : The desired mode for the camera's exposure routine.
(10) exposure-lock : Locks current camera exposure routine values from changing.
(11) exposure-metering : The desired mode for the camera's exposure metering routine.
(12) exposure-compensation : Adjust (Compensate) camera images target brightness. Adjustment is measured as a count of steps.
(13) manual-exposure-time : Manual exposure time in nanoseconds. Used when the Exposure mode is set to 'off'.
(14) custom-exposure-table : A GstStructure describing custom exposure table
(15) white-balance-mode : The desired mode for the camera's White balance routine.
(16) white-balance-lock : Locks current White Balance values from changing. Affects only non-manual white balance modes.
(17) manual-wb-settings : Manual White Balance settings such as color correction temperature and R/G/B gains. Used in manual white balance modes.
(18) focus-mode : Whether auto-focus is currently enabled, and in what mode it is.
(19) noise-reduction : Noise reduction filter mode
(20) noise-reduction-tuning : A GstStructure describing noise reduction tuning
(21) zoom : Camera zoom rectangle ('<X, Y, WIDTH, HEIGHT >') in sensor active pixel array coordinates
(22) defog-table : A GstStructure describing defog table
(23) ltm-data : A GstStructure describing local tone mapping data
(24) infrared-mode : Infrared Mode
(25) active-sensor-size : The active pixel array of the camera sensor ('<X, Y, WIDTH, HEIGHT >') and it is filled only when the plugin is in READY or above state
video_0_Pad -----
(26) framerate : Target framerate in frames per second for displaying
(27) bitrate : Target bitrate in bits per second for compressed streams
(28) idr-interval : IDR interval for compressed streams
(29) crop : Crop rectangle ('<X, Y, WIDTH, HEIGHT >'). Applicable only for JPEG and YUV2 formats
=====
===== Plugin Signals =====
(30) capture-image
=====
===== Other =====
(b) Back : Return to the previous menu
```

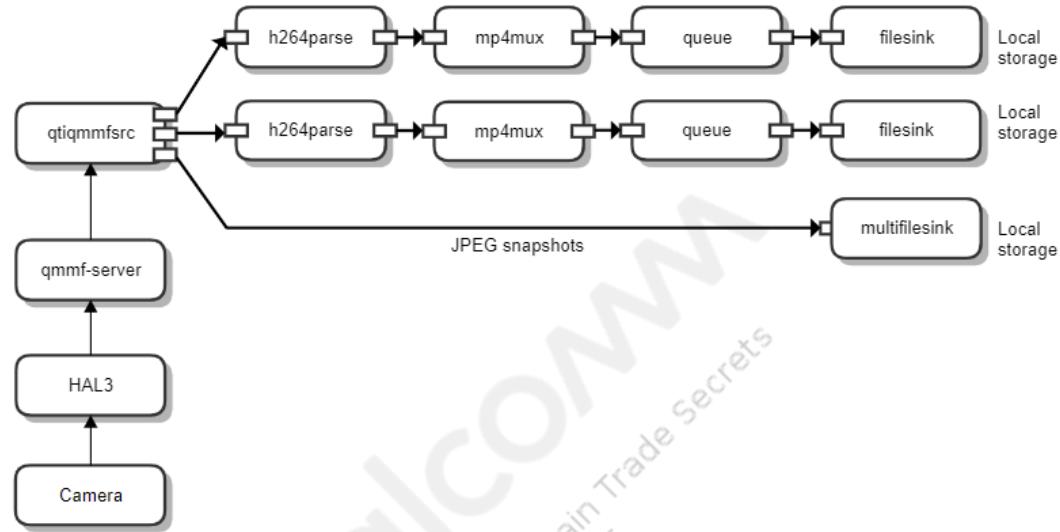
- Choose the required option and provide inputs to on/off or set the appropriate value of that feature.
- The effect can viewed on the live preview.

The table lists some of the important parameters. For more information, see [qtqmmfsrc](#).

Feature	Action
AF modes	Select <b>focus-mode</b>
Exposure Metering	Select <b>exposure-metering</b>
ISO Setting	Select <b>iso-mode</b>
Manual Exposure	<ol style="list-style-type: none"> <li>Disable AE mode by selecting <b>exposure-mode</b>, and then <b>off</b>.</li> <li>Select <b>manual-exposure-time</b>.</li> </ol>
Digital Zoom	<ol style="list-style-type: none"> <li>Select <b>zoom</b>.</li> <li>Check the selected to Sensor mode adbilogcat   grep SelectedMode Based on the selected sensor mode, adjust the input value to see the desired zoom effect. For example: If the selected sensor mode is 3840 x 2160 <ul style="list-style-type: none"> <li>Top left zoom <ul style="list-style-type: none"> <li>For 1x zoom, input value should be &lt;0,0,3840, 2160&gt;</li> <li>For 2x zoom, sensor mode width and height should be divided by 2. The input value would be &lt;0,0,1920,1080&gt;.</li> </ul> </li> <li>Center zoom</li> </ul> For 2x zoom, specify the location(x,y) as the first two arguments. So the input would be &lt;960,540,1920,1080&gt;. This (x,y) value is obtained by dividing the final resolution by 2, since this is a 2x zoom, the final resolution would be 1920x1080 and center of the frame would be (960,540)</li> </ol>
AWB Modes	Select <b>white-balance-mode</b>

Feature	Action
Dynamic Exposure Tables	<p>Select <b>custom-exposure-table</b></p> <p>The input should be in the following format:</p> <pre>org.codeaurora.qcamera3.exposuretable, isValid=true, sensitivityCorrectionFactor=1.0, kneeCount=1, gainKneeEntries=&lt;1.0, 1.0, 32.0&gt;, expTimeKneeEntries=(int)&lt;10040, 33333333, 60000000&gt;, incrementPriorityKneeEntries=&lt;1, 1, 1&gt;, expIndexKneeEntries=&lt;0.0, 274.2918, 434.8758&gt;, thresAntiBandingMinExpTimePct=0.97;</pre>
Defog	<p>Choose the <b>defog-table</b> option</p> <p>The input should be in the following format:</p> <pre>org.quic.camera.defog, enable=true, algo_type=0, algo_decision_mode=1, strength=1.0, convergence_speed=3, lp_color_comp_gain=1.0, abc_en=true, acc_en=true, afsd_en=true, afsd_2a_en=true, defog_dark_thres=10, defog_bright_thres=40, abc_gain=2.0, acc_max_dark_str=2.0, acc_max_bright_str=2.0, dark_limit=255, bright_limit=0, dark_preserve=10, bright_preserve=50, trig_params=&lt;0.0, 1.0, 0, 1.5, 4.0, 100, 4.5, 8.0, 0, 0.0, 50.0, 0, 80.0, 450.0, 100, 510.0, 900.0, 0, 0.0, 2000.0, 0, 2300.0, 4500.0, 0, 5000.0, 8500.0, 100, 10000.0, 20000.0, 0, 0.0, 1.0, 100, 2.0, 64.0, 100&gt;;</pre>
IR	Select <b>infrared-mode</b>
LTM (Dynamic Contrast)	<p>Select <b>ltm-data</b></p> <p>The input should be in the following format:</p> <pre>org.quic.camera.ltmDynamicContrast, ltmDynamicContrastStrength=10.0, ltmDarkBoostStrength=20.0, ltmBrightSupressStrength=30.0;</pre>
Automatic dynamic range compression (ADRC)	Can be enabled or disabled.
Advanced noise reduction (ANR)	<p>Select <b>ltm-data</b></p> <p>The input should be in the following format:</p> <pre>org.quic.camera.anr_tuning, anr_intensity=50.0, anr_motion_sensitivity=70.0;</pre>
sharpness	Change sharpness value
framerate	Change framerate value
bitrate	Change bitrate value
idr-interval	Change idr-interval value
saturation	Change image saturation value. The expected value is in the range [0,10]. The default is 5.
contrast	Change image contrast value. The expected value is in the range [-100, 100]. The default value is 5.

## 7.9 Snapshot



In this example 1080p snapshot are captured while two 1080p video recordings are in progress. For more information on `gst-pipeline-app`, see [QTI `gst-pipeline-app`](#).

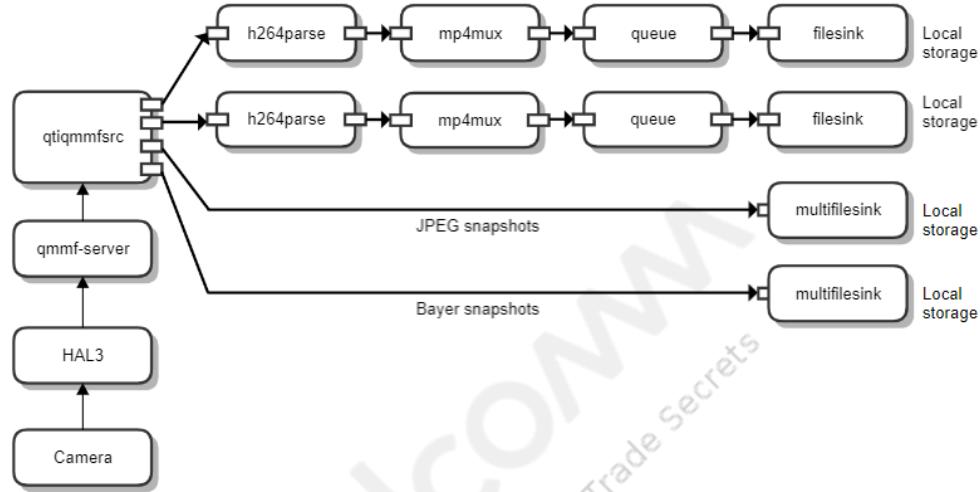
- Run the following command:

```
gst-pipeline-app -e qtqmmfsrc name=qmmf ! video/x-h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux1.mp4" qmmf. ! video/x-h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4" qmmf.image_2 ! "image/jpeg,width=1920,height=1080,framerate=30/1" ! multifilesink location=/data/frame%d.jpg sync=true async=false
```

A menu is displayed on the cmd line terminal.

- To set the pipeline in playing state, select **PLAYING**.
- Choose **Plugin Mode**.
- Type the plugin name that is to be controlled:  
qmmf
- Select the **capture-image** option to capture the image.  
The recorded MP4 files and the JPEG snapshots are stored in /data/.
- To pull snapshot and recorded video, run the following command:  
`adb pull /data/<filename>`

### 7.9.1 JPEG and Bayer snapshot



In this use case, JPEG snapshot stream of 1080p and maximum resolution of Bayer stream are captured while two 1080p video recording are in progress.

- Run the following command to execute the use case.

```
gst-pipeline-app -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux1.mp4" qmmf. ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux2.mp4" qmmf.image_2 ! "image/
jpeg,width=1920,height=1080,framerate=30/1" ! multifilesink location=/data/
frame%d.jpg sync=true async=false qmmf.image_3 ! "video/x-
bayer,format=gbrg,bpp=(string)10,width=3864,height=2180,framerate=30/1" !
multifilesink location=/data/frame%d.gbry sync=true async=false
```

A menu is displayed on the command line terminal.

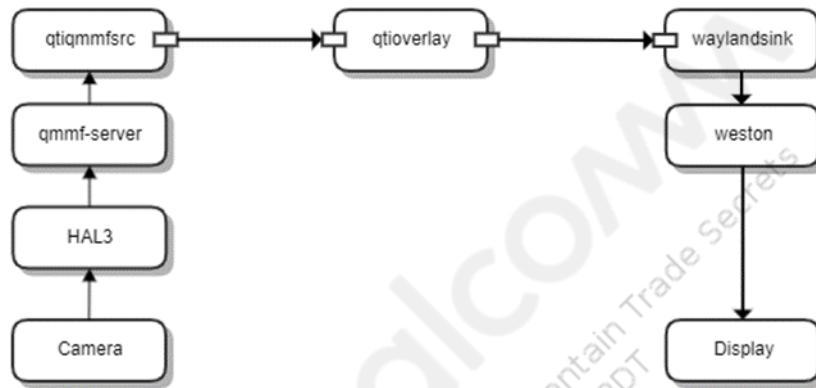
- To set the pipeline to playing state, select **PLAYING**.
- Choose **Plugin Mode**.
- Type the plugin name that is to be controlled:  
qmmf
- Select **capture-image** to take snapshots.  
The recorded MP4 files and the snapshots are stored in /data/.
- To pull snapshots and recorded video, run the following command:  
adb pull /data/<filename>

## 7.10 Overlay

### Prerequisites:

Ensure that Weston is running. For more information, see [Start Weston](#).

#### 7.10.1 Live camera preview with overlay



##### 7.10.1.1 1080p video preview with user text overlay

To apply user text overlay on 1080p live video preview, run the following command:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtqmmfsrc !
video/x-raw\ (memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1, camera=0 ! qtioverlay overlay-text="text0, text=\\"Qualcomm\\
Intelligence\\", color=(uint)0xFFFF00FF, dest-rect=<160, 624, 944, 50>;" !
waylandsink x=0 y=0 width=1920 height=1080 sync=false
```

1. The qtqmmfsrc element is used to generate the 1080p resolution YUV stream.
2. The YUV stream is received by the qtioverlay plugin, which applies user text given as overlay-text property value on to the stream, and then passes it to waylandsink element for rendering on display.

To stop the use case, press **CTRL + C**.

##### 7.10.1.2 1080p video preview with date and time overlay

To apply date and time overlay on 1080p live video preview, run the following command:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtqmmfsrc !
video/x-raw\ (memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1, camera=0 ! qtioverlay overlay-date="date0, date-format=
\"MMDDYYYY\\", time-format=\"HHMMSS_24HR\\", color=(uint)0xFF00FFFF, dest-
rect=<0, 0, 256, 80>;" ! waylandsink x=0 y=0 width=1920 height=1080 sync=false
```

1. The qtiqmmfsrc element is used to generate one 1080p resolution YUV stream.
2. The YUV stream is received by the qtioverlay plugin, which applies date and time given as overlay-date property value on to the stream, and then passes it to waylandsink element for rendering on display.

To stop the use case, press **CTRL + C**.

#### 7.10.1.3 1080p video preview with inverse circular privacy mask overlay

To apply inverse privacy mask overlay on 1080p live video preview, run the following command:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtiqmmfsrc !
video/x-raw\ (memory:GBM\), format=NV12, width=1920, height=1080,
framerate=30/1, camera=0 ! qtioverlay overlay-mask="mask0, circle=<960, 540,
540>, inverse=true, dest-rect=<0, 0, 1920, 1080>, color=0x202020FF" !
waylandsink x=0 y=0 width=1920 height=1080 sync=false
```

1. The qtiqmmfsrc element is used to generate one 1080p resolution YUV stream.
2. The YUV stream is received by the qtioverlay plugin, which applies the inverted privacy mask, meaning only the region specified in the circle is visible rest of the region is masked out by the color specified, and then passes it to waylandsink element for rendering on display.

To stop the use case, press **CTRL + C**.

#### 7.10.1.4 1080p video preview with static image and circular privacy mask overlay

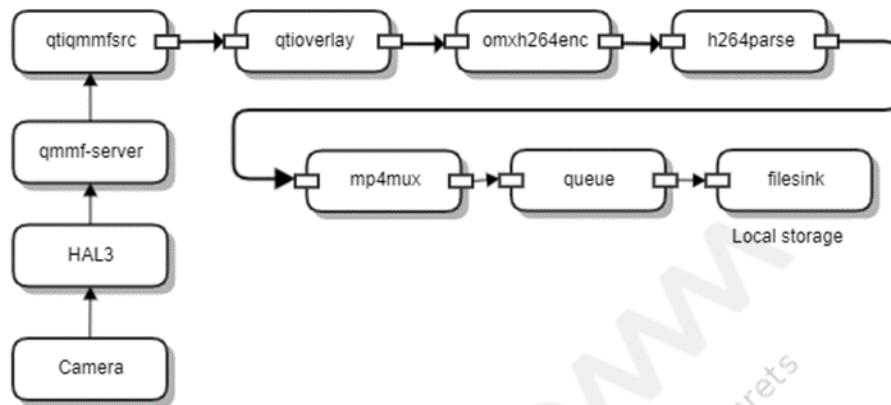
To apply static image overlay with the privacy mask overlay on 1080p live video preview, run the following command:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 -e qtiqmmfsrc !
video/x-raw\ (memory:GBM\), format=NV12, width=1920, height=1080,
framerate=30/1, camera=0 ! qtioverlay overlay-simg="image0, image=/data/
misc/qmmf/overlay_test_464_109.rgba\", resolution=<464, 109>, dest-
rect=<1300, 800, 464, 109>; overlay-mask="mask0, circle=<960, 540, 400>,
dest-rect=<0, 0, 1920, 1080>, color=0x202020FF" ! waylandsink x=0 y=0
width=1920 height=1080 sync=false
```

1. The qtiqmmfsrc element is used to generate one 1080p resolution YUV stream.
2. The YUV stream is received by the qtioverlay plugin, which applies the privacy mask, meaning only the region specified in the circle is masked out by the color specified, also lays the static image at the specified location.
3. The qtioverlay plugin then passes frames with overlay on it to waylandsink element for rendering on display.

To stop the use case, press **CTRL + C**.

## 7.10.2 Live video recording with overlay



### 7.10.2.1 1080p video with user text overlay stored to file

To apply user text overlay on 1080p video, run the following command:

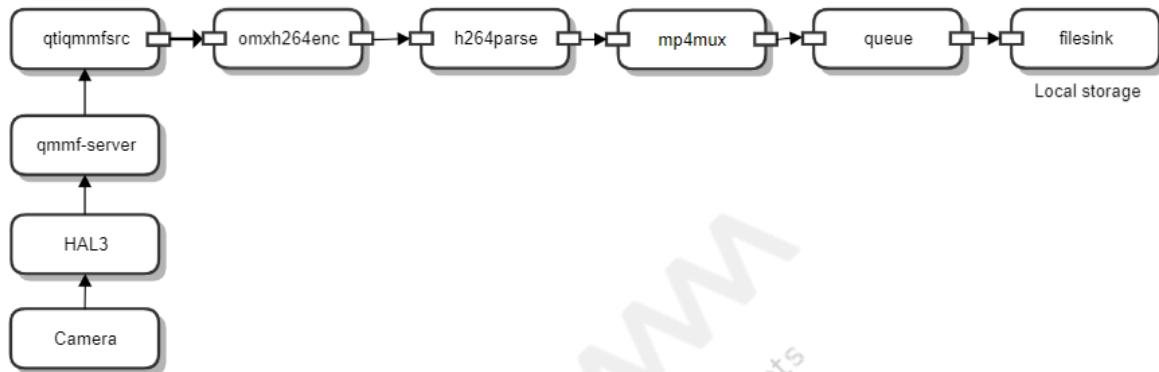
```
gst-launch-1.0 -e qtqmmfsrc name=qmmf ! "video/x-raw(memory:GBM),format=NV12,camera=0,width=1920,height=1080,framerate=30/1" ! qtioverlay overlay-text="text0, text=\"Qualcomm\ Intelligence\", color=(uint)0xFFFF00FF" ! omxh264enc target-bitrate=6000000 ! h264parse ! mp4mux ! queue ! filesink location=/data/mux.mp4
```

1. The qtqmmfsrc element is used to generate one 1080p resolution YUV stream.
2. The YUV stream is received by the qtioverlay plugin, which applies user text given as overlay-text property value on to the stream.
3. The YUV stream with overlay is then processed by h264parse and encoded by the omxh264enc plugin.
4. The mp4mux plugin packages the encoded stream in MP4 container.
5. The filesink element stores it in a file on the device.

To stop the use case, press **CTRL + C**, and pull the recorded content out from device using the following adb pull command, and then play content on host PC.

```
adb pull /data/mux.mp4
```

## 7.11 Video encode



The `omxh264enc` plugin supports various video encode parameters. This parameter can be updated statically during launch and changed dynamically when the use case is running. For complete list of video encode parameters supported by the plugin, see [omxh264enc](#).

### 7.11.1 Static parameter update

Run the following command to set the bitrate, rate control, and I-frame interval through the `omxh264enc` plugin:

```
gst-launch-1.0 -e qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1,camera=0 ! omxh264enc
target-bitrate=6000000 control-rate=2 interval-intraframes=29 ! h264parse !
mp4mux ! queue ! filesink location=/data/mux.mp4
```

1. The `qtiqmmfsrc` element captures 1080p resolution stream from camera.
2. The `omxh264enc` plugin encodes it with bitrate to 6 Mbps, bitrate control set to constant, and I-frame interval set to 29, that is, every 30th frame will be I-frame.
3. The `h264parse` and `mp4mux` elements process the bitstream and prepare them for storage.
4. The `queue` element ensures that each path and track run independently of the other.
5. The `filesink` elements store the buffers in the corresponding file.

### 7.11.2 Dynamic bitrate update

1. Run the use case using QTI `gst-pipeline-app` command line tool to set bitrate through the `omxh264enc` plugin dynamically when the pipeline is in PLAYING state. For information on `gst-pipeline-app`, see [QTI gst-pipeline-app](#).

```
gst-pipeline-app -e qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1,camera=0 !
omxh264enc ! h264parse ! mp4mux ! queue ! filesink location=/data/mux.mp4
```

The following options are displayed:

```
#####
##### MENU #####
=====
===== Pipeline Controls =====
(0) NULL : Set the pipeline into NULL state
(1) READY : Set the pipeline into READY state
(2) PAUSED : Set the pipeline into PAUSED state
(3) PLAYING : Set the pipeline into PLAYING state
=====
===== Other =====
(p) Plugin Mode : Choose a plugin which to control
(q) Quit : Exit the application

Choose an option: 3
```

Set the pipeline in **PLAYING** state.

- Choose **Plugin Mode** to see the plugins whose properties can be modified.

```
#####
##### MENU #####
=====
===== Pipeline Controls =====
(0) NULL : Set the pipeline into NULL state
(1) READY : Set the pipeline into READY state
(2) PAUSED : Set the pipeline into PAUSED state
(3) PLAYING : Set the pipeline into PLAYING state
=====
===== Other =====
(p) Plugin Mode : Choose a plugin which to control
(q) Quit : Exit the application

Choose an option: p
```

- Type `omxh264enc-omxh264enc0` to change the bitrate. A menu listing the name of the property and its description is displayed:

```
-----
( 1) filesink0
( 2) queue0
( 3) mp4mux0
( 4) h264parse0
( 5) omxh264enc-omxh264enc0
( 6) capsfilter0
( 7) qmmfsrc0
-----

Enter plugin name or its index (or press Enter to return): 5

Enter name of the plugin which will be controlled: omxh264enc-omxh264enc0

#####
##### MENU #####
=====
===== Properties =====
( 0) target-bitrate : Target bitrate in bits per second (0xffffffff=component default)
=====
===== Other =====
(q) Quit
```

4. Type 0 to select target-bitrate. The current value of the target-bitrate followed by the range of acceptable values is displayed. Type the bitrate value.

The screenshot provides an example where the value is set to 6000000 for 6 Mbps bitrate.

```
#####
# MENU #####
=====
==== Plugin Properties =====
( 0) target-bitrate : Target bitrate in bits per second (0xffffffff=component default)
----- sink Pad -----
----- src Pad -----
===== Plugin Signals =====
===== Other =====
(b) Back : Return to the previous menu

Choose an option: 0
-----
Current value: 4294967295, Range: 0 - 4294967295
-----

Enter value (or press Enter to keep current one): 6000000

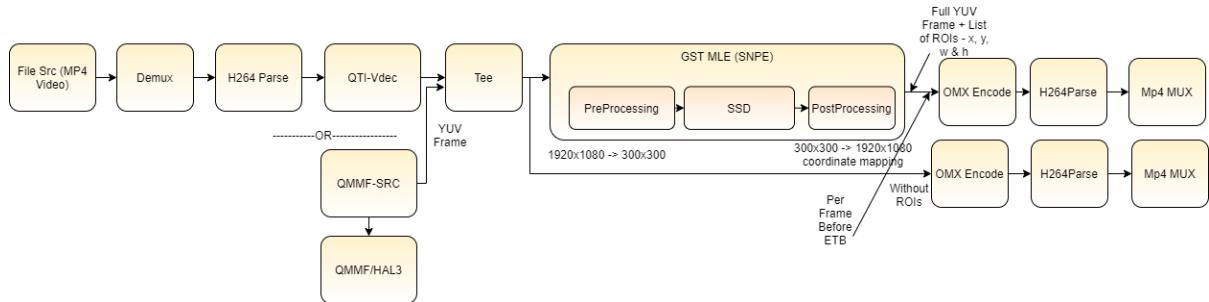
#####
# MENU #####
=====
==== Plugin Properties =====
( 0) target-bitrate : Target bitrate in bits per second (0xffffffff=component default)
----- sink Pad -----
----- src Pad -----
===== Plugin Signals =====
===== Other =====
(b) Back : Return to the previous menu

Choose an option: 0
-----
Current value: 6000000, Range: 0 - 4294967295
-----
```

To pull the recorded video, run the `adb pull /data/mux.mp4` command

### 7.11.3 ROI-based encoding

The QTI video encoder supports ROI-based encoding. The changes were made to the GStreamer omxh264enc plugin to enable ROI-based video encoding through manual settings or from inference results of ML engine.



**Figure 7-9 Use case – ROI video encoding-based on ML inference results**

- The source of the video can be either a live camera source (`qtqmmfsrc` plugin) or an offline video source (`filesrc` plugin).
- The video frames are fed to the ML engine (`qtimle*` plugins) for processing.

- The ML engine generates the inference results (rectangle regions) which are then used by the omxh264enc plugin for ROI based encoding.
- ROI encoded video is then stored in the file after muxing.
- Optionally we can use tee plugin as shown above to generate 2 streams from the video stream obtained from the video source. One stream can be encoded using ROI and one without it for comparing the output.

The ROI region can also be specified manually via omxh264enc property, for this case the GST MLE block shown in the usecase flow above will not be part of the pipeline.

The features are enabled by introducing the following properties in the omxh264enc GStreamer plugin.

**Table 7-1 ROI properties**

Property	Description
roi-quant-delta	<ul style="list-style-type: none"> <li>■ The quantization parameter are added to the frame QP values that are defined in the ROI macroblocks.</li> <li>■ A negative value indicates better quality while positive values indicate worst quality.</li> <li>■ Integer; Range: -31 to 30; default: -15</li> </ul>
roi-quant-mode	<p>Adjust quantization parameter according to ROI</p> <ul style="list-style-type: none"> <li>■ <b>(0): disabled:</b> Let the encoder change the QP for each coding unit according to its content</li> <li>■ <b>(1): metadata:</b> Adjust QP according to the regions of interest defined on each frame. Must be set to handle ROI metadata.</li> <li>■ <b>(2): custom:</b> User-defined ROI area via the 'roi-quant-area' property. The ROI metadata is ignored.</li> </ul>
roi-quant-rectangles	<p>Manually, set ROI rectangles (example: "&lt;&lt;X, Y, W, H&gt;, &lt;X, Y, W, H&gt;&gt;") to be used when the mode is set to custom, that is, "roi-quant-mode=custom". The syntax should be written with mandatory quotes as shown in the roi-quant-rectangles property syntax figure.</p>

```

1st ROI Rectangle      2nd ROI Rectangle
|                   |
|                   |
|                   |---> More rectangle can be added with ',' separator
"<<0, 0, 400, 400>, <760, 340, 400, 400>>" 
|   |   |   |
|   |   |   |--> Rectangle height
|   |   |   |--> Rectangle width
|   |   |
|   |   |--> Y axis coordinate begining from top left corner
|   |   --> X axis coordinate begining from top left corner

```

**Figure 7-10 roi-quant-rectangles property syntax**

## Manual mode

- ROI QP encoding can be enabled via the roi-quant-mode property of the omxh264enc plugin. The ROI macroblocks QP delta can be set via roi-quant-delta property, and single or multiple ROI rectangles can be set via property roi-quant-rectangles.
  - The pipeline needs omxh264enc connected right after the source plugin with roi-quant-mode set to custom mode and optionally roi-quant-delta its value can be set between -31 to 30 (default value is -15) where, negative numbers represent better quality while positive numbers represent lower quality.
  - The roi-quant-rectangles property is used to specify the ROI rectangles with the syntax specified in the property table with mandatory quotes.
- To disable ROI QP encoding, set the roi-quant-mode property to disabled.

## ML Metadata mode

- ROI QP encoding can be enabled via the roi-quant-mode property of the omxh264enc plugin and ROI macroblocks QP delta set via the roi-quant-delta property. For it to work in conjunction with MLE the following setup is required:
  - The pipeline that is going to create needs to have a MLE plugin with a detection model and postprocessing detection enabled.
  - The pipeline also needs omxh264enc connected right after the MLE plugin with roi-quant-mode set to metadata mode and optionally roi-quant-delta may be set to a value between -31 to 30 (default value is -15) where, negative numbers represent better quality while positive numbers represent lower quality.
- To disable ROI QP encoding, set the roi-quant-mode property to disabled.

To perform a test, ensure that the ML configuration and model file is available on the target device for running the ML-based ROI video encoding use case.

For example, for tflite detection mode, run the following commands:

```
adb push labelmap.txt /data/misc/camera/
adb push detect.tflite /data/misc/camera/
```

The following list provides the test cases and the respective commands:

- **Manual ROI better quality with constant rate control**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! queue ! omxh264enc roi-quant-mode=custom roi-
quant-rectangles="<<0, 0, 400, 400>, <760, 340, 400, 400>>" roi-quant-
delta=-30 control-rate=constant target-bitrate=5000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse config-interval=-1 ! queue !
filesink location=/data/VideoWithCustomROI_CBR_5Mbps_QP[-30].h264
```
- **Manual ROI poor quality with constant rate control**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! queue ! omxh264enc roi-quant-mode=custom roi-
quant-rectangles="<<0, 0, 400, 400>, <760, 340, 400, 400>>" roi-quant-
delta=15 control-rate=constant target-bitrate=5000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse config-interval=-1 ! queue !
filesink location=/data/VideoWithCustomROI_CBR_5Mbps_QP[15].h264
```

- **Manual ROI with constant rate control**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! queue ! omxh264enc roi-quant-mode=custom roi-
quant-rectangles="<<0, 0, 400, 400>, <760, 340, 400, 400>>" roi-quant-
delta=-15 control-rate=constant target-bitrate=5000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse config-interval=-1 ! queue !
filesink location=/data/VideoWithCustomROI_CBR_5Mbps_QP[-15].h264
```

- **ML ROI better quality with constant rate control**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! qtimletflite model=/data/misc/camera/
detect.tflite labels=/data/misc/camera/labelmap.txt
postprocessing=detection ! queue ! omxh264enc roi-quant-mode=metadata roi-
quant-delta=-30 control-rate=constant target-bitrate=5000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse config-interval=-1 ! queue !
filesink location=/data/VideoWithROI_CBR_5Mbps_QP[-30].h264
```

- **ML ROI poor quality with constant rate control**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! qtimletflite model=/data/misc/camera/
detect.tflite labels=/data/misc/camera/labelmap.txt
postprocessing=detection ! queue ! omxh264enc roi-quant-mode=metadata roi-
quant-delta=15 control-rate=constant target-bitrate=5000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse config-interval=-1 ! queue !
filesink location=/data/VideoWithROI_CBR_5Mbps_QP[15].h264
```

- **ML ROI with constant rate control**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! qtimletflite model=/data/misc/camera/
detect.tflite labels=/data/misc/camera/labelmap.txt
postprocessing=detection ! queue ! omxh264enc roi-quant-mode=metadata roi-
quant-delta=-15 control-rate=constant target-bitrate=5000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse config-interval=-1 ! queue !
filesink location=/data/VideoWithROI_CBR_5Mbps_QP[-15].h264
```

- **ROI disabled**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! queue ! omxh264enc roi-quant-mode=disabled
control-rate=constant target-bitrate=5000000 interval-intraframes=29
periodicity-idr=1 ! h264parse config-interval=-1 ! queue ! filesink
location=/data/VideoWithoutROI_CBR_5Mbps.h264
```

- **ROI disabled with ML**

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),width=1920,height=1080 ! qtimletflite model=/data/models/detect.tflite
labels=/data/models/labelmap.txt postprocessing=detection ! queue !
omxh264enc roi-quant-mode=disabled control-rate=constant target-
bitrate=5000000 interval-intraframes=29 periodicity-idr=1 ! h264parse
```

```
config-interval=-1 ! queue ! filesink location=/data/
VideoWithoutROI_ML_CBR_5Mbps.h264
```

To verify the output, do the following:

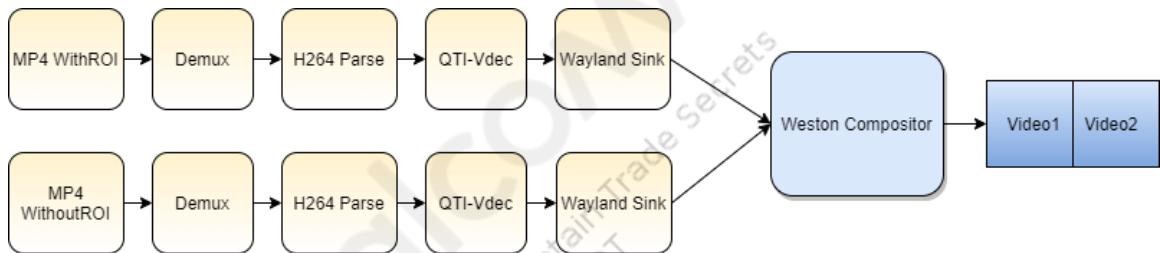
- On host verification

The video file can be with and without ROI encoding and be pulled from the target device, and then played back side-by-side on the host PC. Play the file in your favorite media player side-by-side:

```
adb pull /data/<video file>
```

- On target verification

The decode video file with and without ROI encoding on the target device and rendering on the HDMI display. The figure shows the pipeline structure:



**Figure 7-11 ROI and non-ROI video side by side pipeline structure**

Ensure that Weston is running. For more information, see [Start Weston](#). Run the following commands:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/<video file#1> ! queue ! h264parse ! qtivdec ! waylandsink
x=0 y=0 width=960 height=540

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/<video file#2> ! queue ! h264parse ! qtivdec ! waylandsink
x=960 y=0 width=960 height=540
```

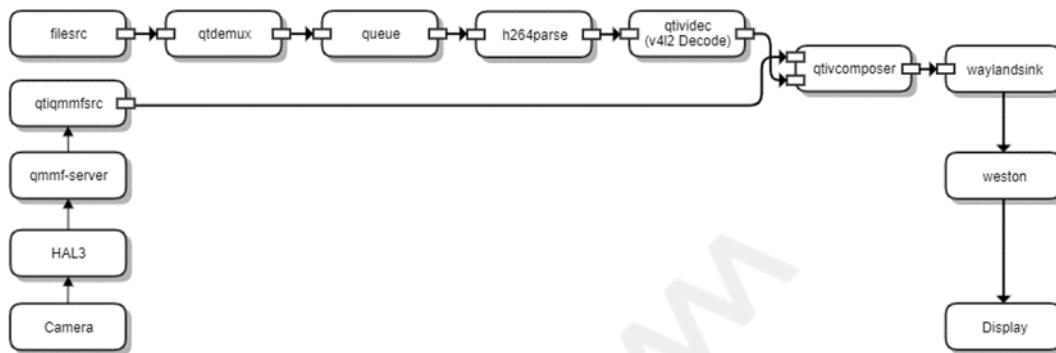
## 7.12 Video composition

**Prerequisites:**

- Ensure that Weston is running. For more information, see [Start Weston](#).
- Run the following command to create a test video. Press **Ctrl + C** to stop the recording.

```
gst-launch-1.0 -e qtqmmfsrc name=qmmp ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_1080p_avc.mp4"
```

### 7.12.1 PiP – 1080p camera preview and 1080p recorded h264 video file playback



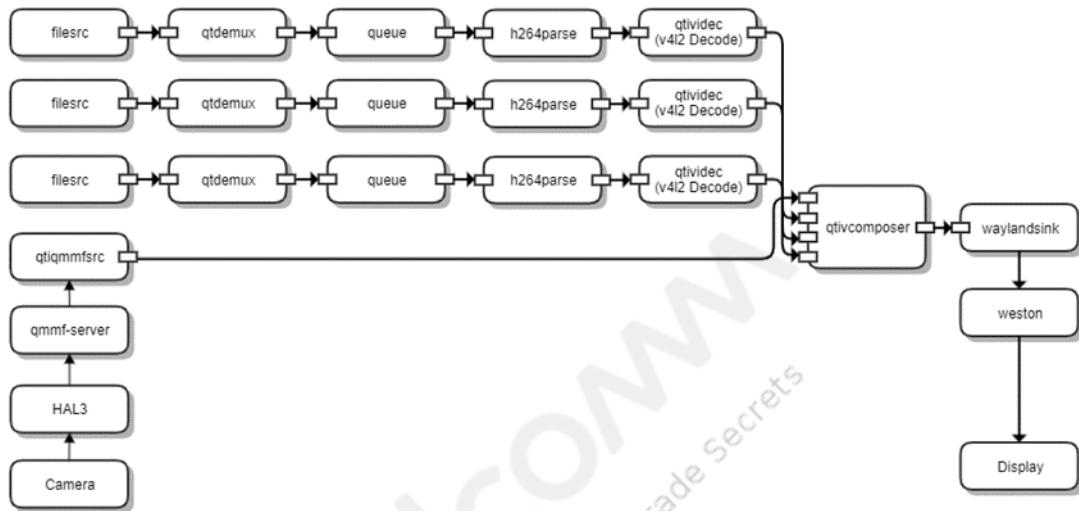
For a 1080p live preview from camera with 1080p recorded video playback in  $640 \times 360$  destination window in the bottom right corner, run the following command:

```

export XDG_RUNTIME_DIR=/dev/socket/weston_socket && gst-launch-1.0 qtqmmfsrc
name=qmmf ! video/x-raw\ (memory:GBM\),width=1920,height=1080 ! qtivcomposer
name=mix sink_0::position="<0, 0>" sink_0::dimensions="<1920, 1080>" 
sink_1::position="<1280, 720>" sink_1::dimensions="<640, 360>" mix. ! video/x-
raw\ (memory:GBM\),format=NV12 ! queue ! waylandsink sync=false
fullscreen=true filesrc location=/data/mux_1080p_avc.mp4 ! qtdemux !
h264parse ! qtivdec ! mix.
  
```

1. The qtqmmfsrc element captures 1080p resolution stream from camera and passes it to the qtivcomposer element.
2. The other 1080p stream is read from stored file by filesrc element and passed to qtivcomposer element
3. The qtivcomposer element downscals the stream from camera to  $640 \times 480$  and composes it on top of buffer obtained from filesrc. The composed buffer is then passed to waylandsink.
4. The waylandsink element renders the composed buffer on the display.

## 7.12.2 Video tiling – Multiple video streams tiled in one output



For a 1080p output constructed from four inputs placed in a  $2 \times 2$  grid, where, one input stream is 1080p live preview for camera and remaining three streams are 1080p recorder videos, run the following command:

```

export XDG_RUNTIME_DIR=/dev/socket/weston_socket && gst-launch-1.0 qtqmmfsrc
name=qmmf ! video/x-raw\ (memory:GBM\),width=1920,height=1080 ! qtivcomposer
name=mix sink_0::position="<0, 0>" sink_0::dimensions="<960, 540>"
sink_1::position="<960, 0>" sink_1::dimensions="<960, 540>"
sink_2::position="<0, 540>" sink_2::dimensions="<960, 540>"
sink_3::position="<960, 540>" sink_3::dimensions="<960, 540>" mix. ! video/x-
raw\ (memory:GBM\),format=NV12 ! queue ! waylandsink sync=false
fullscreen=true filesrc location=/data/mux_1080p_avc.mp4 ! qtdeMUX !
h264parse ! qtivdec ! mix. filesrc location=/data/mux_1080p_avc.mp4 !
qtdeMUX ! h264parse ! qtivdec ! mix. filesrc location=/data/
mux_1080p_avc.mp4 ! qtdeMUX ! h264parse ! qtivdec ! mix.

```

1. The qtqmmfsrc element captures the 1080p resolution stream from camera and passes it to the qtivcomposer element.
2. The other three 1080p stream are read from stored file by filesrc element and passed to qtivcomposer element.
3. The qtivcomposer element downscales the stream buffers to  $960 \times 540$  and composes them in a  $2 \times 2$  grid. The composed buffer is then passed to waylandsink.
4. The waylandsink element renders the composed buffer on the display.

## 7.13 ML

### Prerequisites:

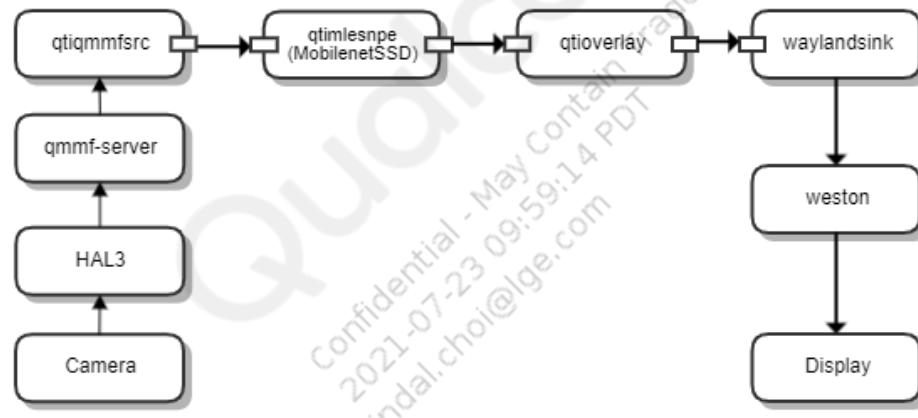
- Ensure that Weston is running. For more information, see [Start Weston](#).
- Ensure that the corresponding models, labels, and test video files are pushed on the target device. Run the following command to push the files on the target:

```
adb push <file_name> /data/misc/camera/
```

### 7.13.1 Single stream Qualcomm Neural Processing SDK inference with live camera

In this use case, the video analytics is done on 720p live camera stream.

The object detection is done using the Qualcomm Neural Processing SDK detection model (MobileNet Single Shot MultiBox Detector (SSD)) and bounding box overlay is drawn over the detected objects.



Run the following command to execute the use case.

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtqmmfsrc !
video/x-raw\ (memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1, camera=0 ! qtimlesnpe config=/data/misc/camera/
mle_snpe.config postprocessing=detection ! queue ! qtioverlay ! waylandsink
x=960 y=0 width=960 height=540 async=true sync=false enable-last-sample=false
```

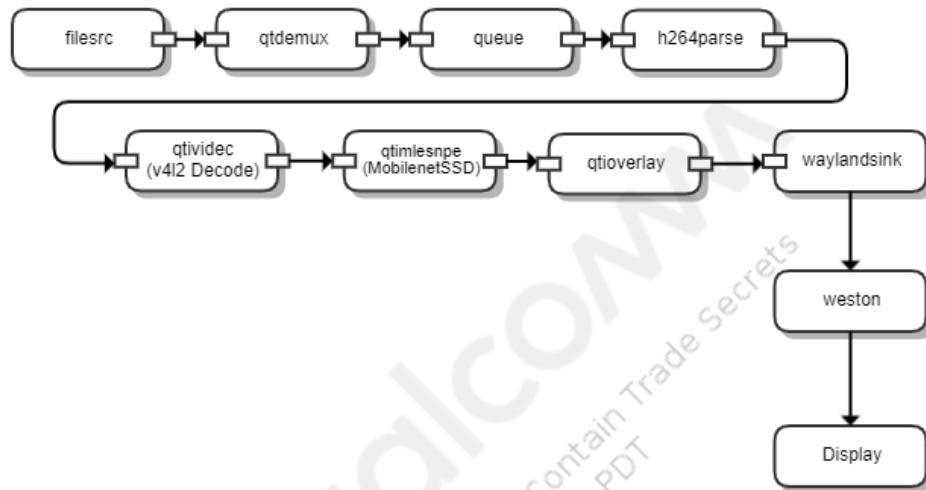
1. The qtqmmfsrc plugin generates a 720p YUV frames
2. The qtimlesnpe plugin processes the input YUV frames.
3. Based on the configuration, qtimlesnpe loads and executes the model on Snapdragon compute core (CPU, GPU, DSP, and AIP), and generates the inference results.
4. The YUV frames and inference results are processed by qtioverlay plugin, which draws the bounding box overlay based on inference result on the frames.
5. The frames with overlay are rendered onto the HDMI display by the waylandsink plugin.

To stop the use case, press **CTRL + C**.

## 7.13.2 Single stream Qualcomm Neural Processing SDK inference on offline video

In this use case, the video analytics is done on 720p offline pre-recorded video stream.

The object detection is done using the Qualcomm Neural Processing SDK detection model (MobileNet SSD), and bounding box overlay is drawn over the detected objects.



Run the following command to execute the use case:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc location=/data/test_video.mp4 ! qt demux name=demux demux. ! queue ! h264parse !
qtivdec ! qt imlesnpe config=/data/misc/camera/mle_snpe.config
postprocessing=detection ! queue ! qt overlay ! waylandsink x=960 y=0
width=960 height=540 async=true sync=false enable-last-sample=false
```

1. The filesrc plugin reads the video frames from the test file.
2. The qt demux and h264parse plugins demultiplex and process the frame.
3. qtivdec decodes the AVC encoded bitstream and generates YUV frames for qt imlesnpe.
4. The qt imlesnpe plugin processes the input YUV frames.
5. Based on configuration, qt imlesnpe loads and executes the model on Snapdragon compute core (CPU, GPU, DSP, and AIP), and generates inference results.
6. The YUV frames and inference results are processed by qt overlay plugin, which draws bounding box overlay based on inference result on the frames.
7. The frames with bounding box overlay are rendered onto the HDMI display by the waylandsink plugin.

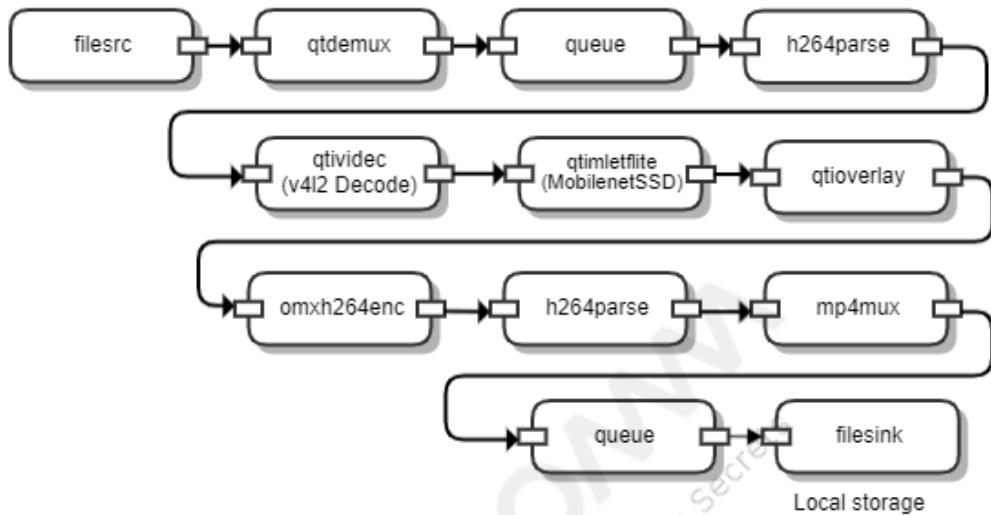
To stop the use case, press **CTRL + C**.

## 7.13.3 Single stream TensorFlow Lite inference on offline video and stored to file

In this use case, the video analytics is done on 720p offline pre-recorded video stream.

The object detection is done using the TensorFlow Lite detection model (MobileNet SSD), and bounding box overlay is drawn over the detected objects.

The video stream with inference overlay is stored to a file.



Run the following command to execute the use case:

```
gst-launch-1.0 filesrc location=/data/test_video.mp4 ! qtdemux name=demux
demux. ! queue ! h264parse ! qtivdec ! qtimletflite config=/data/misc/camera/
mle_tflite.config model=/data/misc/camera/detect.tflite labels=/data/misc/
camera/labelmap.txt postprocessing=detection ! queue ! qtioverlay !
omxh264enc target-bitrate=6000000 ! h264parse ! mp4mux ! queue ! filesink
location=/data/out_mux.mp4
```

The `filesrc` plugin reads the video frames from the test file.

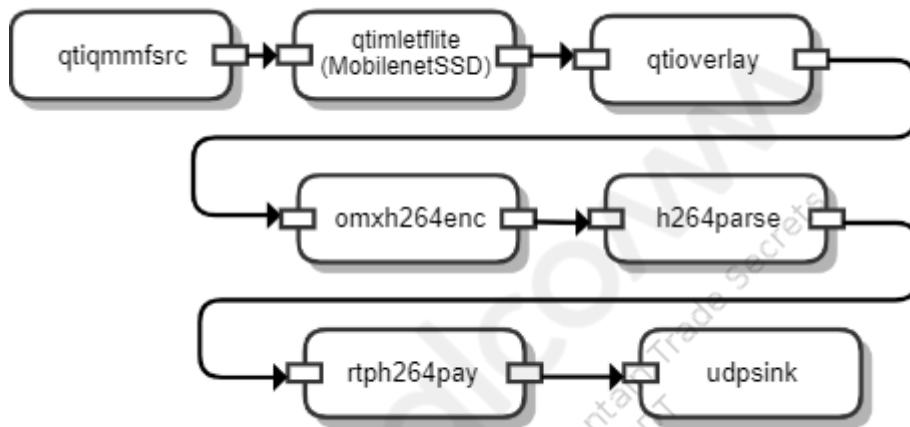
1. The `qtdemux` and `h264parse` plugins demultiplex and process the frame.
2. `qtivdec` decodes the AVC encoded bitstream and generates YUV frames for `qtimletflite`.
3. The `qtimletflite` plugin processes the input YUV frames.
4. Based on configuration, `qtimletflite` loads and executes the model on Snapdragon compute core (CPU or DSP), and generates the inference results.
5. The YUV frames and inference results are processed by the `qtioverlay` plugin, which draws the bounding box overlay based on inference result on the frames.
6. The frames with bounding box overlay are processed by `h264parse` and encoded by `omxh264enc` plugin.
7. The `mp4mux` plugin packages the encoded stream in MP4 container.
8. The `filesink` element stores it in a file on the device.

To stop the use case, press **CTRL + C**, pull the recorded content out from device using the following `adb pull` command, and then play content on host PC.

```
adb pull /data/out_mux.mp4
```

### 7.13.4 Single stream live camera TensorFlow Lite inference streamed over RTSP – SSD

In this use case, the video analytics is done on 720p live camera video stream. The object detection is done using TensorFlow Lite detection model (MobileNet SSD) and bounding box overlay is drawn over the detected objects. The video stream with inference overlay is streamed over RTSP.



- Run RTSP server on target with udpsrc

```
gst-rtsp-server -p 8900 -m "( udpsrc name=pay0 port=8554 caps=\\"application/x-rtp,media=video,clock-rate=90000,encoding-name=H264,payload=96\\" )"
```

The RTSP server is started on port number 8900. This server receives the video stream using udpsrc, and then processes the RTP payload and streams it over network at port number 8900.

- On the console, run the following command:

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\ (memory:GBM\), format=NV12, width=1280, height=720, framerate=30/1, camera=0 ! qtimletflite config=/data/misc/camera/mle_tfLite.config model=/data/misc/camera/detect.tflite labels=/data/misc/camera/labelmap.txt postprocessing=detection ! queue ! qtioverlay bbox-color=0xFF0000FF ! omxh264enc periodicity-idr=1 interval-intraframes=29 ! h264parse config-interval=-1 ! rtph264pay pt=96 ! udpsink host=127.0.0.1 port=8554
```

- The qtiqmmfsrc plugin generates a 720p YUV frames.
- The qtimletflite plugin processes the input YUV frames.
- Based on the configuration, qtimletflite loads and executes the model on Snapdragon Compute core (CPU and DSP), and generates inference results.
- YUV frame and inference result are processed by qtioverlay plugin, which draws the bounding box overlay based on inference result on the frames.
- The frames with bounding box overlay are encoded by omxh264enc plugin and processed by h264parse plugin.

- The encoded buffer stream with inference overlay is then transformed into RTP payload data by the rtpb264pay plugin.
- The RTP packets are dumped on the UDP port 8554 by udpsink plugin.
- The RTP packets with payload data are received by the UDP server started earlier using udpsrc plugin.

#### 7.13.4.1 View RTSP stream on the host PC

##### Prerequisites:

- Install ADB and VLC Media Player on the host machine.
- Set the binary or executable paths in the environment variables.

For more information, contact QTI support.

On Linux host, run the following commands:

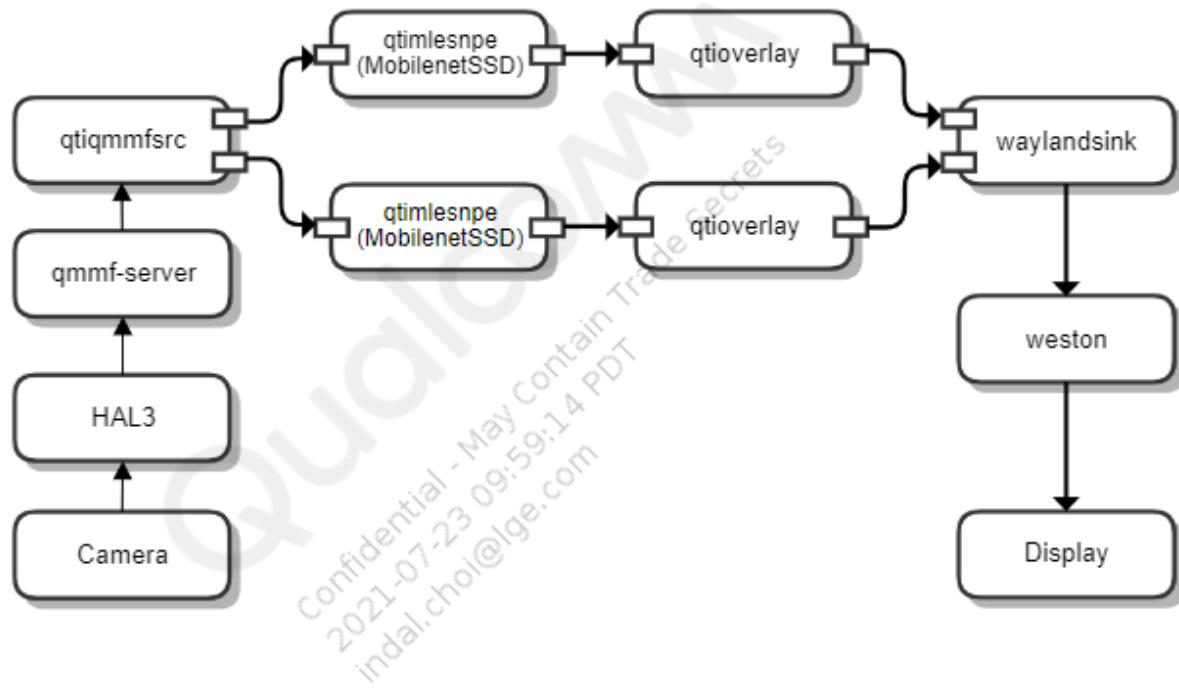
```
adb forward tcp:8900 tcp:8900  
vlc -vvv rtsp://127.0.0.1:8900/live
```

On Windows host, do the following:

1. Run adb forward tcp:8900 tcp:8900.
2. Start VLC Media Player.
3. Select **Media/Open Network Steam...** (or press **CTRL + N**)
4. Enter the network URL – rtsp://127.0.0.1:8900/live
5. Click **Play**.

### 7.13.5 Two stream Qualcomm Neural Processing SDK inference with live camera – SSD

In this use case, the video analytics is done on two separate 720p live camera streams. On both streams, object detection is done using Qualcomm Neural Processing SDK detection model (MobileNet SSD) and bounding box overlay is drawn over the detected objects.



Run the following command to execute the use case:

```

export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtqmmfsrc
name=qmmf ! video/x-raw\ (memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1, camera=0 ! qtimlesnpe config=/data/misc/camera/
mle_snpe.config postprocessing=detection ! queue ! qtiooverlay ! waylandsink
x=960 y=0 width=960 height=540 sync=false enable-last-sample=false qmmf. !
video/x-raw\ (memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1, camera=0 ! qtimlesnpe config=/data/misc/camera/
mle_snpe.config postprocessing=detection ! queue ! qtiooverlay ! waylandsink
x=0 y=0 width=960 height=540 sync=false enable-last-sample=false

```

1. The qtqmmfsrc plugin generates two 720p YUV streams.
2. The qtimlesnpe plugin then processes the input YUV frames.
3. Based on configuration, the qtimlesnpe plugin loads and executes the model on Snapdragon compute core (CPU, GPU, DSP, and AIP), and generates inference results.

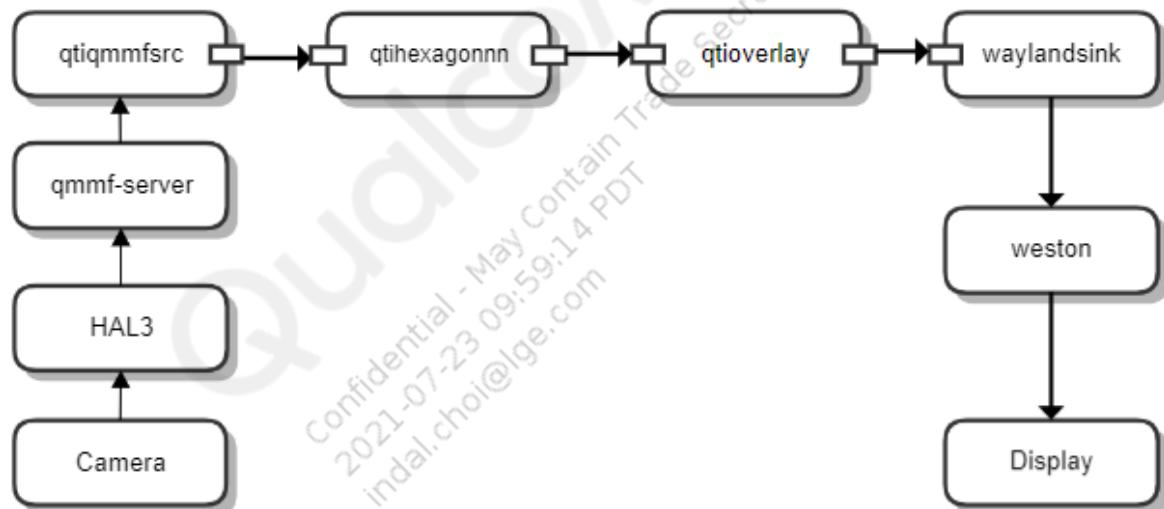
4. The YUV frames along with inference result are processed by qtioverlay plugin, which draws bounding box overlay based on inference result on the frames.
5. The frames with overlay are rendered onto the HDMI display by the waylandsink plugin.

To stop the use case, press **CTRL + C**.

### 7.13.6 Single stream live camera preview with DirectNN inferencing

The qtihexagonnn GStreamer plugin can load and execute neural network models directly on DSP hexagon neural network accelerator. It supports the following models out of the box:

- MobileNet SSD (v1)
- Semantic segmentation (DeepLab v3)
- PoseNet (posenet\_mobilenet\_v1\_075)



#### Prerequisites

1. Push the skel file on the device

```
adb push libhexagon_nn_skel_dp.so in /usr/lib/rfsa/adsp/
```

The device must be signed.

2. Push the label file on the device

```
adb push <label-file> /data/misc/camera/
```

For more information on obtaining the skel file and signing of the device, contact QTI support team.

#### MobileNet SSD

In this use case, video analytics is done on the 720p live camera stream. The object detection is done using MobileNet SSD v1 model and bounding box overlay is drawn over the detected objects.

Run the following command to execute the use case:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtiqmmfsrc
name=qmmf ! video/x-raw, format=NV12, width=1280, height=720, framerate=30/1,
camera=0 ! qtihexagonnn model-name=mnetssd label-file=labels.txt ! queue !
qtioverlay ! waylandsink fullscreen=true async=true
```

1. The qtiqmmfsrc plugin generates the 720p YUV frames.
2. The qtihexagonnn plugin processes the input YUV frames, runs the Mobile Net SSD v1 model on the DSP hexagon neural network accelerator, and generates inference results.
3. The inference results are attached to the GST buffer as GST MLMeta.
4. The YUV frames and inference results are processed by the qtioverlay plugin, which draws the bounding box based on the inference result on the frames.
5. The frames with overlay are rendered onto the HDMI display by the waylandsink plugin.

To stop the use case, press **CTRL + C**.

### Semantic segmentation

In this use case, video analytics is done on 720p live camera stream. Semantic segmentation is done using the segmentation (DeepLab v3) model, and resulting segmentation mask is drawn as overlay.

Run the following command to execute the use case:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtiqmmfsrc
name=qmmf ! video/x-raw, format=NV12, width=1280, height=720, framerate=30/1,
camera=0 ! qtihexagonnn model-name=segmentation ! queue ! qtioverlay !
waylandsink fullscreen=true async=true
```

1. The qtiqmmfsrc plugin generates the 720p YUV frames.
2. The qtihexagonnn plugin processes the input YUV frames, runs the semantic segmentation model on the DSP hexagon neural network accelerator, and generates inference results.
3. The inference results are attached to the GST buffer as GST MLMeta.
4. The YUV frames and inference results are processed by the qtioverlay plugin, which overlays the segmentation mask based on the inference result on the frames.
5. The frames with segmentation overlay are rendered onto the HDMI display by the waylandsink plugin.

To stop the use case, press **CTRL + C**.

### PoseNet

In this use case, the video analytics is done on the 720p live camera stream. Pose detection is done using PoseNet (posenet\_mobilenet\_v1\_075) model and pose graph is drawn over the objects.

Run the following command to execute the use case:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 qtiqmmfsrc
name=qmmf ! video/x-raw, format=NV12, width=1280, height=720, framerate=30/1,
camera=0 ! qtihexagonnn model-name=posenet ! queue ! qtioverlay !
waylandsink fullscreen=true async=true
```

1. The qtiqmmfsrc plugin generates the 720p YUV frames.
2. The qtihexagonnn plugin processes the input YUV frames, runs the PoseNet model on the DSP hexagon neural network accelerator, and generates inference results.
3. The inference results are attached to the GST buffer as GST MLMeta.
4. The YUV frames and inference results are processed by the qtioverlay plugin, which draws the pose graph based on inference result on the frames.
5. The frames with overlay are rendered onto the HDMI display by the waylandsink plugin.

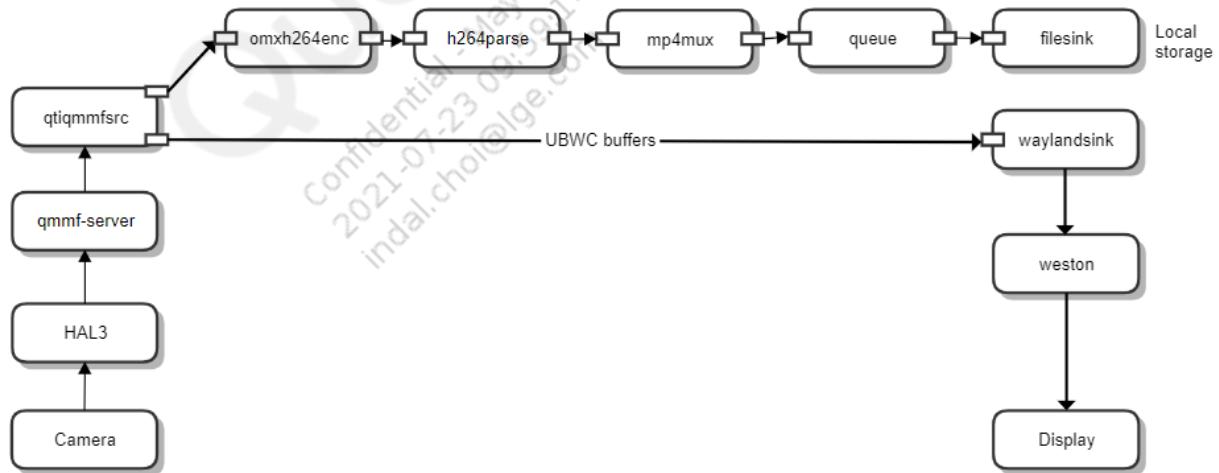
To stop the use case, press **CTRL + C**.

## 7.14 UBWC control

The sample use cases enable you to control UBWC per video stream. UBWC helps reduce the memory bandwidth and power requirement of the use case. All major modules of the platform like camera, video, display, and so on support UBWC and enable efficient end-to-end use case.

### 7.14.1 Live preview using in memory compression with video record

Ensure that Weston is running. For more information, see [Start Weston](#).



Run the following command to execute the use case:

```

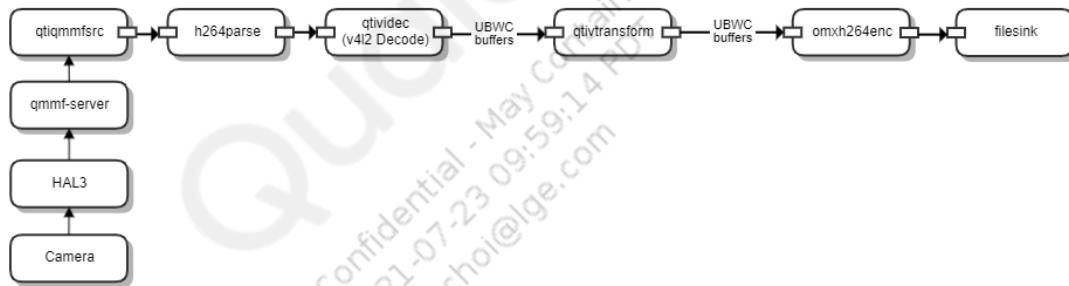
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 -e qtiqmmfsrc
name=qmmf ! "video/x-
raw(memory:GBM),format=NV12,compression=ubwc,width=1920,height=1080,framerate=
30/1" ! waylandsink sync=false fullscreen=true enable-last-
sample=false qmmf. ! "video/x-
raw(memory:GBM),format=NV12,width=1920,height=1080,framerate=30/1,camera=0" !
omxh264enc ! h264parse ! mp4mux ! queue ! filesink location=/data/mux.mp4
  
```

1. The **qtiqmmfsrc gstreamer** plug-in generates one in memory compressed YUV stream (UBWC) and one YUV stream without in memory compression (non-UBWC).
2. The stream parameters such as width, height, framerate, and format are specified as a capsfilter.
3. The **compression** parameter in the capsfilter is used to control UBWC, when it is set to "**compression=ubwc**", UBWC is enabled for that particular stream. The two connecting plugins then use UBWC buffers.
4. The compressed stream is processed and rendered by the **waylandsink gstreamer** plug-in on to the HDMI display.
5. The uncompressed stream is encoded by **omxh264enc** plugin, processed by **h264parse**, wrapped in mp4 container by **mp4mux** plugin, and dumped in file by the **filesink** plugin.

To stop the use case, press **CTRL + C**. Pull the recorded content out from device using `adb pull /data/mux.mp4` command, and play content on host device.

#### 7.14.2 1080p video downscale to 720p

In this use case, a 1080p AVC video stream is decoded to YUV, downscaled to 720p, and then dumped to file after AVC encoding.



Run the following command to execute the use case:

```
gst-launch-1.0 qtiqmmfsrc ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
qtivdec ! video/x-raw\ (memory:GBM\),format=NV12, compression=ubwc ! queue !
qtivtransform ! video/x-raw\ (memory:GBM\),format=NV12,
compression=ubwc,width=1280,height=720 ! omxh264enc ! filesink location=/data/
video.h264
```

1. The qtiqmmfsrc gstreamer plugin generates h264-encoded 1080p buffers.
2. h264parse parses the buffer and qtivdec decodes the buffer to YUV.
3. qtivdec shares the compressed (UBWC) YUV buffer to qtivtransform.
4. qtivtransform downscals the buffer to 720p using hardware acceleration and passes it to omxh264enc.
5. omxh264enc encodes the buffer to h264 using hardware acceleration and passes it to filesink.
6. filesink then stores the encoded stream to file.

To stop the use case, press **CTRL + C**, pull the recorded content out from device using the following `adb pull` command, and then play content on host PC.

```
adb pull /data/video.h264
```

## 7.15 Live streaming over RTSP

The following use cases are provided in this section:

- An AVC-encoded 1080p video stream at 30 fps is streamed over RTSP from the device.
- Two AVC-encoded 1080p video stream at 30 fps. One stream is stored on the device after muxing, while other stream is streamed over the network over RTSP.

### 7.15.1 1080p live streaming over RTSP

In this use case, a AVC-encoded 1080p video stream at 30 fps is streamed over RTSP from the device.

To execute the use case, do the following:

1. Run RTSP server on target with udpsrc:

```
gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
\"application/x-rtp,media=video,clock-rate=90000,encoding-
name=H264,payload=96\" )"
```

The RTSP server is started on port number 8900. This server receives the video stream using udpsrc, and then processes the RTP payload and streams it over network at port number 8900.

2. Run the following command to create video stream for the RTSP server:

```
gst-launch-1.0 -e qtiqmmfsrc ! video/x-
h264,width=1920,height=1080,framerate=30/1 ! h264parse config-
interval=-1 ! rtph264pay pt=96 ! udpsink host=127.0.0.1 port=8554
```

- The qtiqmmfsrc gstreamer plugin creates a 1080p AVC-encoded video stream.
- h264parse processes the stream and the buffers are packed as RTP payload by rtph264pay for streaming.
- The udpsink element sends this data to the gst-rtsp-server.

### 7.15.2 Two H264 streams – one stored and another streamed over RTSP

To execute the use case, do the following:

1. Run RTSP server on target with udpsrc

```
gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
\"application/x-rtp,media=video,clock-rate=90000,encoding-
name=H264,payload=96\" )"
```

The RTSP server is started on port number 8900. This server receives the video stream using udpsrc, processes the RTP payload, and then streams it over network at port number 8900.

2. Run the following command to create video stream for the RTSP server:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux.mp4" qmmf. ! video/x-
```

```
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse config-
interval=1 ! rtph264pay pt=96 ! udpsink host=127.0.0.1 port=8554
```

- The qtiqmmfsrc gstreamer plugin creates both 1080p AVC-encoded video streams.
- The first stream is processed by h264parse, packaged in MP4 container by the mp4mux plugin, and dumped to a file by the filesink plugin.
- The second stream is processed by h264parse, rtph264pay packages the buffers into RTP payload for streaming, and the udpsink element sends this data to the gst-rtsp-server.

To view the RTSP stream on the host PC, see [View RTSP stream on the host PC](#).

## 7.16 Parallel 4K encode and 1080p decode

This section provides a use case to decode and playback 1080p H264-encoded video on HDMI display while 4K H264-encoded video is captured and stored in file.

Here, the video hardware performs both encode and decode parallelly.

### Prerequisites:

1. Ensure that Weston is running. For more information, see [Start Weston](#).
2. Record a 1080p test video for playback or push a pre-recorded video.

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_avc.mp4"
```

To execute the use case, do the following:

1. To start 4K H264 recording, run the following command in one terminal:
 

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-
h264,format=NV12,width=3840,height=2160,framerate=30/1 ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_4k_avc.mp4"
```

  - a. The qtiqmmfsrc element is used to capture and encode the video streams.
  - b. The h264parse and mp4mux elements process the buffers and prepare them for storage.
  - c. The filesink element stores the buffers in the file.
2. To play back the recorded 1080p video on HDMI, run the following command in another terminal:
 

```
export XDG_RUNTIME_DIR=/dev/socket/weston && gst-launch-1.0 filesrc
location=/data/mux_avc.mp4 ! qtdemux name=demux demux. ! queue !
h264parse ! qtivdec ! video/x-raw\ (memory:GBM\),compression=ubwc !
waylandsink fullscreen=true
```

  - The filesrc element reads the pre-recorded 1080p video.
  - The qtdemux element demultiplexes the video and h264parse processes the video.
  - The qtivdec element decodes the encoded video and generates YUV video for display.
  - The waylandsink element renders the YUV video on the HDMI display.

## 7.17 Video conferencing

The gst-tracking-cam-app is a QTI reference application to demonstrate two-stream video conferencing use case using GStreamer framework, which consists of an main video stream and a video stream with ML enabled on it.

- The application demonstrates the creation of two streams – main and ML and the usage of the appsink element to take the ML ROIs out from pipeline into the application. It shows the real time camera tracking of an object using ML within full FOV and applying crop on the main video stream.
- The application has custom built-in filter algorithm, which is used to generate smooth ROIs for a smooth digital pan experience. The algorithm takes many parameters as input such as speed, position, dimension threshold, and dimension margin. Based on the parameters, it generates a series of smooth ROIs (bounding boxes). All the parameters are exposed to user and can be tuned based on the scene.
- The application has support for stream types – YUY2, MJPEG, and NV12.
- Currently, waylandsink is used as sink plugin for the user to see output immediately on Display/HDMI (available for NV12 and YUY2 format only). It has support for file save/mux, the same stream will be saved to mp4.

Parameter	Function	Description
-c	The crop type selected	The crop type used: <ul style="list-style-type: none"> <li>▪ 0 - GST, using qtivtransform plugin</li> <li>▪ 1 - CamX, using vendor tags</li> </ul>
-d	The dimensions threshold of the crop	The size of the crop window will be updated only if the size of the detected object crosses this threshold.
-f	The mainstream format	The format of the mainstream: <ul style="list-style-type: none"> <li>▪ 0 - YUY2</li> <li>▪ 1 - MJPEG</li> <li>▪ 2 - NV12</li> </ul>
-h	The mainstream height	The height of the main camera stream
-m	The dimensions margin added to the calculated crop	Additional pixels added to all sides of the final crop window.
-p	The position threshold of the crop	The position of the crop window will be updated only if the position of the detected object crosses this threshold.
-s	The speed of movement to the final crop rectangle	The speed for a smooth transition from the current coordinates of the crop to the final destination.
-w	The mainstream width	The width of the main camera stream

1. Run the following commands:

```

adb root
adb disable-verity
adb reboot
adb root
adb shell mount -o remount,rw

```

2. Push the detect.tflite and labelmap.txt files.

```
adb push labelmap.txt /data/misc/camera/
adb push detect.tflite /data/misc/camera/
```

3. Apply the following properties:

```
adb shell "echo outputFormat=0 >> /etc/camera/camxoverridesettings.txt"
```

```
##### For YUY2 #####
adb shell setprop persist.vendor.camera.mjpeg.usecase 0
adb shell setprop persist.vendor.camera.yuy2.usecase 2
```

```
##### For MJPEG #####
adb shell setprop persist.vendor.camera.mjpeg.usecase 2
adb shell setprop persist.vendor.camera.yuy2.usecase 0
```

```
##### For NV12 #####
adb shell setprop persist.vendor.camera.mjpeg.usecase 0
adb shell setprop persist.vendor.camera.yuy2.usecase 0
```

4. Run Weston.

```
adb shell ". /etc/profile ; export XDG_RUNTIME_DIR=/dev/socket/weston ;
chmod 0700 $XDG_RUNTIME_DIR; cd /usr/bin/ ; ./weston --tty=1 --idle-time=0"
```

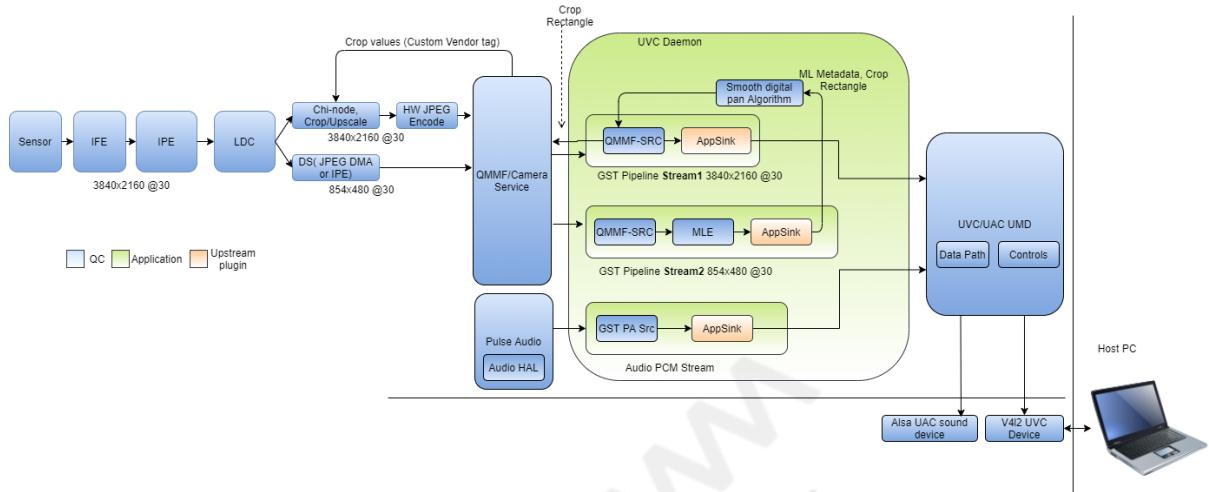
**Table 7-2 Use cases for video conferencing**

Use case	Execution command	Validation method
YUY2 (Camera crop)	export XDG_RUNTIME_DIR=/dev/socket/weston && gst-tracking-cam-app -p 10 -d 10 -m 5 -s 5 -f 0 -c 1 -w 3840 -h 2160	/data/mux.mp4 should be available. The video should be visible on the screen and object should be tracked.
NV12 (GST crop)	export XDG_RUNTIME_DIR=/dev/socket/weston && gst-tracking-cam-app -p 10 -d 10 -m 5 -s 5 -f 2 -c 0 -w 3840 -h 2160	/data/mux.mp4 should be available. The video should be visible on the screen and object should be tracked.
MJPEG (Camera crop)	export XDG_RUNTIME_DIR=/dev/socket/weston && gst-tracking-cam-app -p 10 -d 10 -m 5 -s 5 -f 1 -c 1 -w 3840 -h 2160	/data/mjpeg.avi should be available and the object should be tracked.

## 7.18 UVC and UAC architecture

The gst-umd-daemon application uses the GStreamer framework to demonstrate the following:

- UVC and UAC capabilities of the system with the help of umd gadget library
- ML capability to track objects based on ROI with the help of auto framing algorithm



**Figure 7-12 UVC-UAC high-level architecture**

## UVC

**Table 7-3 Formats and resolutions**

Formats/ resolutions	360p	720p	1080p	4k
YUY2	Supported	Supported	Supported	Not supported
MJPEG	Supported	Supported	Supported	Supported

Control	Support
Brightness	Supported
Contrast	Supported
Saturation	Supported
Sharpness	Supported
White Balance Temperature	Supported
Auto-Exposure Mode	Supported
Exposure Time	Supported
Zoom	Supported
Pan	Supported
Tilt	Supported
Power Line Frequency	Supported
Backlight Compensation	Supported
Gain	Not supported
Auto-Exposure Priority (Low light Compensation)	Not supported

## UAC

- **Formats and resolutions:** 16-bit (S16LE), two-channel, and 48 kHz PCM
- **Audio controls:** Volume and mute

### 7.18.1 Run UVC + UAC

#### Prerequisites:

Ensure that the following settings are set up:

- Basic:

```
$ adb root
$ adb disable-verity
$ adb reboot
$ adb root
$ adb shell mount -o remount,rw /
$ adb shell "echo outputFormat=0 >> etc/camera/camxoverridesettings.txt"
```

- MPJPEG:

```
$ adb shell setprop persist.vendor.camera.mjpeg.usecase 2
$ adb shell
```

- Wi-Fi

- a. Edit the `wpa_supplicant.conf` file with known network credentials:

```
$ pkill hostapd && rmmod wlan && modprobe wlan
$ wpa_supplicant -B -c/etc/misc/wifi/wpa_supplicant.conf -Dnl80211 -
iwlan0
$ wpa_cli -I wlan0 -p /etc/misc/wifi -s /etc/misc/wifi
$ dhcpcd wlan0
```

- b. Note the IP address of the device.

To enable UVC, do the following:

1. From a new terminal, connect adb over Wi-Fi and start Weston server.

```
$ adb tcpip 5556
$ adb connect <device IP address>:5556 (IP address from above step)
$ adb -s <device IP address>:5556 shell
$ setenforce 0
$ . /etc/profile ; export XDG_RUNTIME_DIR=/dev/socket/weston ; chmod 0700
$XDG_RUNTIME_DIR; cd /usr/bin/ ; ./weston --tty=1 --device=hdmi --idle-
time=0
```

2. Change the default USB composition to UVC + UAC.

The cmd line app to select the usb modes is `usb_composition`.

```
$ adb -s <device IP address>:5556 shell
$ usb_composition
$ Pid number : 9113 (This selects the DIAG + ADB + UVC Mode)
$ Choose core: y - hsic, n - hsusb ? (y/n) n
```

```
$ Would you like it to be the default composition ? (y/n) y
$ Would you like the composition to change immediately? (y/n) y
$ Are you performing the composition switch from adbd? (y/n) y )
```

The device is ready to stream over UVC.

3. Run the UVC daemon: \$ source /etc/profile && chown -R qmmfsrv:qmmfsrv /dev/socket/weston

gst-umd-daemon has two modes of operations:

- A default one, which is without ML

```
$ source /etc/profile && chown -R qmmfsrv:qmmfsrv /dev/socket/weston
```

- The following parameters can be set via interactive menu to enable ML stream or direct cmd with tuning parameters:

```
$ gst-umd-daemon
-l, --ml-auto-framing-enable Enable ML based auto framing (default: false)
-p, --ml-framing-position-threshold The acceptable delta (in percent), between previous ROI position and current one, at which it is considered that the ROI has moved (default: 8)
-d, --ml-framing-dimensions-threshold The acceptable delta (in percent), between previous ROI dimensions and current one, at which it is considered that the ROI has being resized (default: 16)
-m, --ml-framing-margins=MARGINS Used to additionally increase the final size of the ROI rectangle (default: 10)
-s, --ml-framing-speed=SPEED Used to specify the movement speed of the ROI rectangle (default: 10)
-c, --ml-framing-crop-type=[0 - internal / 1 - external] The type of cropping (internal or external) used for the ROI rectangle (default: internal)
```

- With ML (using autoframing) from command line:\$ source /etc/profile && chown -R qmmfsrv:qmmfsrv /dev/socket/weston

After running either of the commands on your device, the QCS610 device is detected as UVC camera.

4. To start streaming, start AMCap or Microsoft teams, or any other similar application. Select **UVC Camera** for video and **Capture input terminal** for audio.

For the ML use case, the camera tracks the person (smooth digital pan) within FOV.

**NOTE** Currently, the gst-umd-daemon operates only on 9113 (DIAG + UAC1 + UVC + ADB) mode.

# 8 QCS410 LEPDK use cases

---

**NOTE** All the use cases in the [QCS610 LEPDK use cases](#) chapter are applicable to the QCS410 chipset, however, with the following limitations:

- Max encode capability is 1080p@90 fps
- Max playback capability is 1080p@90 fps
- sHDR will be enabled in a future software release
- EIS is not supported in QCS410 ODK. However, QCS410 SOC will support EIS in a device in which gyro sensor for EIS is enabled.

QTI recommends using GStreamer-based interface to execute the use cases.

For prerequisites to run the QCS410 LEPDK use cases, see [Prerequisites](#).

For multimedia stack and control flow end-to-end use cases, see the use cases in [Multimedia stack and control flow](#). Additionally, few examples have been provided in the subsequent sections.

## 8.1 Record 1080p hardware-encoded H264 video

Run the following command to record 1080p hardware-encoded H264 video:

```
gst-launch-1.0 -e qtiqmmfsrc name=qmmf ! video/x-h264,format=NV12,width=1920,height=1080,framerate=30/1 ! h264parse ! mp4mux ! queue ! filesink location="/data/mux_avc.mp4"
```

The command uses the `qtiqmmfsrc` to generate the hardware-encoded H264 stream. The stream parameters such as width, height, framerate, and format are specified as a capsfilter.

This stream is then multiplexed using the `mp4mux` plugin, and then dumped using the `filesink` plugin.

To stop the use case, press **CTRL + C**, pull recorded content out from device using the following `adb pull` command, and then play content on host PC.

```
adb pull /data/mux_avc.mp4
```

## 8.2 Live snapshot – 1080p snapshot while a 1080p streaming is running

For information, see [Snapshot](#).

## 9 Connectivity

---

For information on connectivity, see the following documents:

- *WCN39xx Connectivity (WLAN/Bluetooth) Software Reference Manual* (80-WL050-1)
- *WCN39xx Connectivity (WLAN/Bluetooth) Software Programmer's Guide* (80-WL050-2)
- *WFA Certification for Linux MDM Platforms Test Guide* (80-N8207-2)

Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

# A System debugging and features

---

To enhance overall debugging capability, QTI provides the required hardware as part of the debug architecture and dedicated software debug packages.

## A.1 Debug features and infrastructure

The following are the types of debug related features:

- On-target debug features: The features are available on the chipset platform and can be used in run time.
- Off-target debug features: The features are available for online or offline debugging.

### A.1.1 On-target debug features

#### Qualcomm Debug Subsystem

The Qualcomm Debug Subsystem (QDSS) is a hardware block based on Arm CoreSight products.

QDSS provides the capability to trace various hardware events and software events from multiple trace sources into a common interleaved data format with a universal high frequency trace timestamp clock. This capability helps trace systemwide activities at a specific time.

OEMs can also generate a systemwide cross-trigger fatal event to halt all masters at the same time to ensure that the snapshot of a system closely matches the failure. The QDSS block is also configurable to route the trace events into the QDSS internal buffer, DDR memory, or USB port.

For more information, see *QDSS Debug User Guide* (80-NF515-8).

#### QTimer

During a crash dump analysis, the subsystem in which an error has been first generated needs to be identified. The QTimer hardware block is introduced in each subsystem to have a universal timestamp across the subsystems.

All Qtimers are clocked by XO (19.2 MHz). They are automatically in sync while the system is running so that all the subsystems can use a universal time stamp. Using QTimer, the OEM can compare the error messages in each subsystem and determine which error was generated first by comparing the QTimer time stamp.

The only exception is the timestamp in HLOS, where dmesg uses a kernel time tick, which is not in sync with QTimer. However, RTB log on the QCS610 chipset prints the both kernel tick and QTimer.

#### Example 1

**(RTB log)**

```
89669c1 [2473.500766] [47667252507] : LOGK_IRQ interrupt 13 handled from addr
fffffff9add081744 gic_handle_irq (unknown info for address 0xffffffff9add081744)
```

2473.500766 here is kernel tick and 47667252507 is QTimer counter, which is in sync with other subsystems and TZ diag log timestamp.

If the user needs to correlate specific kernel time tick and QTimer, the following equation can be used:

```
kernel time (in nanosec) = cd.epch_ns+((Qttime * 19200000)-
cd.epoch_cyc)&sched_clock_mask)*cd.mult>>cd.shift
```

**Example 2**

Translate 494804.425664 (Qtimer) to kernel time tick:

**[Subsystem log(Qtimer)]**

```
494804.425664: rpm_abort_interrupt_received (application processor NON SECURE
WD BITE) ... aborting
```

**[Kernel log(kernel time tick)]**

```
[494786.962316] FG: fg_get_battery_cycle: Get battery cycle = 35
[494786.984876] FG: update_sram_data: soc:[100], soc_raw[10000], voltage:
[4335149], ocv:[4315312], current:[152], batt_temp:[461], charge_raw
[3414000 / 3431000]
```

```
\vmlinux\sched_clock\cd =
wrap_kt = (tv64 = 3131746996224),
epoch_ns = 491684280882043,
epoch_cyc = 9440396789438,
seq = (sequence = 318),
rate = 19200000,
mult = 109226667,
shift = 21,
suspended = FALSE)

sched_clock_mask = 0x00FFFFFFFFFFFFF

kernel time (in nanosec) = cd.epch_ns+((RPM time * 19200000)-
cd.epoch_cyc)&sched_clock_mask)*cd.mult>>cd.shift
= 494801373814316
= 494801.373814316 sec // this is when rpm_abort_interrupt event happened
in kernel tick scale.
```

The rpm\_abort\_interrupt event happened after FG: update\_sram\_data: soc event.

**Subsystem restart**

The subsystem restart (SSR) feature lets the system run even after a subsystem crash. QTI recommends to use the SSR feature on a commercial device. SSR is disabled by default as QTI requires a full memory dump for debugging issues.

## Protection domain restart

Similar to SSR, the protection domain restart can be enabled for a particular firmware (FW). In QCS610, WLAN FW and audio FW is subject to protection domain restart.

Once this is enabled, even with FW running within the crashed protection domain, other software running outside of protection domain can keep running.

The protection domain restart is disabled by default, but protection domain restart dump is enabled if protection domain restart is enabled.

**Table A-1 Protection domain restart test cases**

Test case	Sensor protection domain	Audio protection domain	WLAN protection domain
Software Error Fatal	send_data 75 37 03 128 00	send_data 75 37 03 144 00	send_data 75 37 03 176 00
Software Exception (NULL ptr)	send_data 75 37 03 128 02	send_data 75 37 03 144 02	send_data 75 37 03 176 02

For more information, see *PD Restart Overview* (80-P9301-114).

## Force subsystem reset

For test and debug purposes, the OEM can force the subsystem to crash using the DIAG command.

```
adb shell "echo c > /proc/sysrq-trigger"
```

The following table lists the commands:

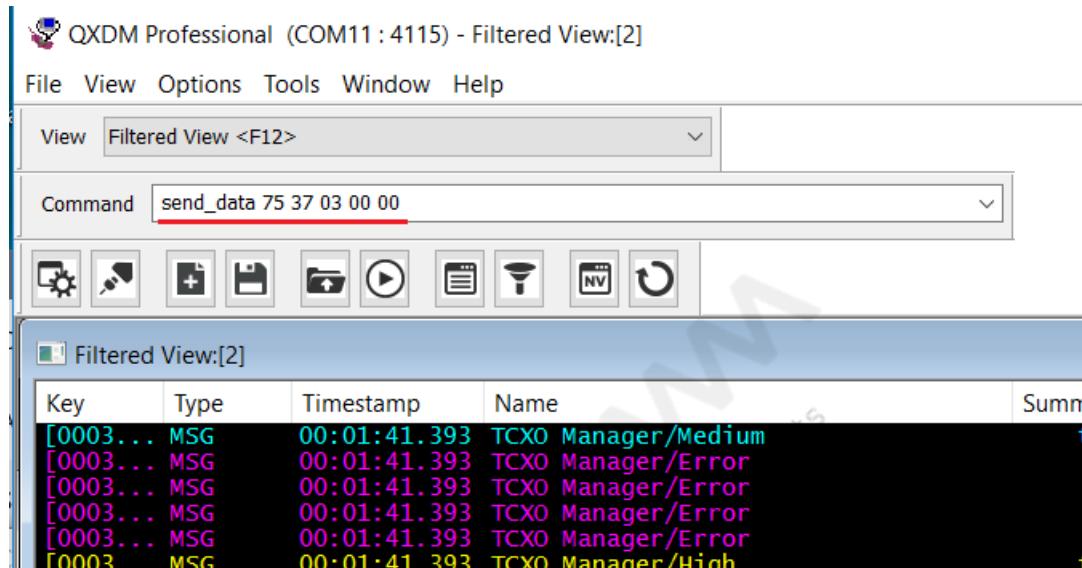
**Table A-2 Force subsystem reset test cases**

Test case	MPSS	LPASS (ADSP)	CDSP
Software Error Fatal	03 00 00	03 48 00	03 96 00
WDOG Bite (in STM)	03 00 01	03 48 01	03 96 01
Software Exception (NULL ptr)	03 00 02	03 48 02	03 96 02
WDOG Bite (NOT in STM)	03 00 06	03 48 06	03 96 06

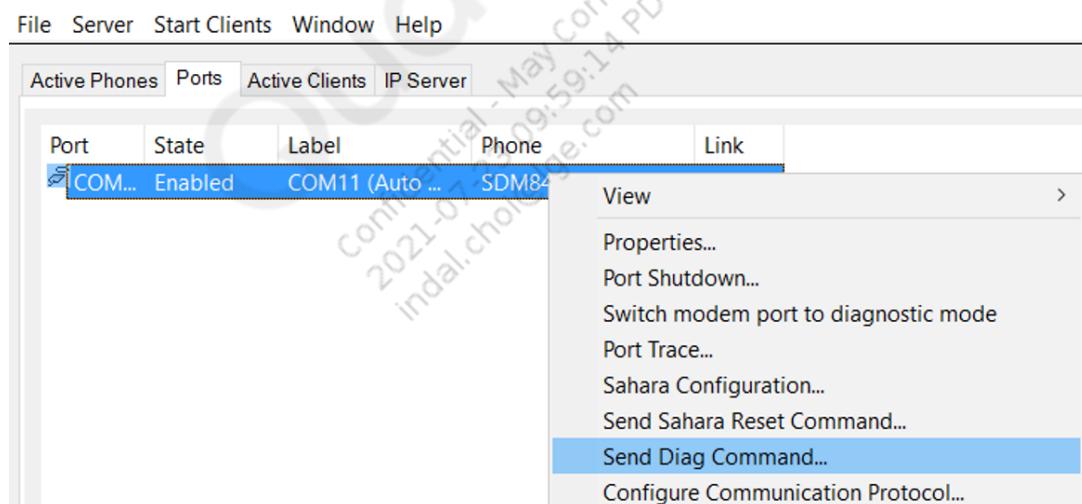
All commands start with 75 37. For example, for MPSS software error fatal, the command is

75 37 03 00 00.

The figures show the working of the software error fatal commands on QXDM Professional tool and QPST:



**Figure A-1 MPSS software error fatal command on QXDM Professional**



**Figure A-2 MPSS software error fatal command on QPST**

### Data capture and compare

Data capture and compare (DCC) is a debug-related DMA engine that is triggered in a catastrophic failure or software-based cross-trigger event. It performs either of the following operations:

- Saves register and memory space dumps to internal RAM or any QDSS sink – QDSS sink can be ETB, DDR, or USB.
- Performs a CRC check over register or memory space dumps and compares it with previously calculated and stored CRC values

This feature is helpful to save the state of any memory mapped hardware block registers at the time of failure to help understand hardware block behavior. For more information, see *Data Capture and Compare User Guide* (80-P1727-1).

## Debug policy for secure fuse devices

Some of the debug features will be unavailable once device is secured (when secure fuse blown on the devices). If OEM wants to have same debug coverage on these devices, OEMs must reflash the device with properly signed software and, in some cases, even modify the code to reenable functionalities. In certain situations, such an approach is not easily applicable.

The debug policy offers a flexible way of enabling development and debugging on fuse-blown devices, without modifying the code or reflashing existing images. The following are scenarios where it is most applicable:

- Support of in-house development on a potentially large number of secure fuse-blown devices
- Enable debugging of a commercial device that was returned under a return material authorization (RMA)

For more information on debug policy for production devices with secure fuse blown, see *Sectools: Debug Policy Tool User Guide* (80-NM248-6).

## ECC

This information will be provided in a future revision of this document.

## JTAG on AO domain

The JTAG logic can stay attached to AOSS during CX power collapse.

The target will go to certain levels of power collapse depending on the processor attached.

- Apps0 Attached – CX is held up (SYSPWRUPREQ)
- Only AOP attached – AOP held out of retention
- SYS.M.PREPARE on AOP – AOSS debug busses held on, but AOP can go to retention

For more information, see *RPM hardening and debugging overview* (80-P9301-16).

## A.1.2 Off-target debug features

### Qualcomm crash analysis portal

The Qualcomm crash analysis portal (QCAP) is a consolidated, system-level crash analysis web portal, with a simple, up-to-date, interface for OEM engineers to retrieve crash signatures independently and to identify duplicates. For more information, see <http://cap.qti.qualcomm.com/>.

The screenshot shows a web-based report titled "QCAP REPORT". At the top, there is a message: "To file a Qualcomm case, please follow the link [HERE](#)". Below this is a table with four columns: "User Name", "Report Id", "Duration", and "Timestamp". A single row is shown with "shinjin" in "User Name", "3a109cd4-3c21-4b27-adc2-46909ee29193" in "Report Id", "6 min, 6 sec." in "Duration", and "6/23/2017 4:59:06 PM" in "Timestamp". To the left of the table is a sidebar titled "Summary" containing a tree view of chipsets: SDM845 (+ Chipset, + APPS, + TZ, + MODEM, + ADSP, + CDSP, + SLPI). To the right of the table are three sections: "Root Cause Analysis" (APPs Crash - Page fault Execute error: FSR = 0x96000045 LR = sysrq\_handle\_crash+0x14/0x34), "Log Location" (\\\varv-shinjin\public\Temp\Port\_COM1), and "Build Information".

**Figure A-3 QCAP**

QCAP enables analysis using internal tools and facilities, significantly reducing crash analysis turnaround time. QCAP provides an interface for OEM engineers to retrieve crash signatures independently and identify duplicates. If further analysis is required, a QTI case can be filed directly from QCAP.

QCAP also helps with the following:

- Support for slow, unreliable internet connections
- Minimum dependency on IT environment
- Dump and ELF matching check

For more information, see *QCAP Start-up Guide* (80-NR964-54).

### Qualcomm smart debug guide

When OEMs raise a SalesForce case, QTI internal debug automation system parses and analyzes it, and then updates the case with the first triage information. It suggests the reference document, a point of contact, and cause of the issue.

This is intended to accelerate the debug so OEMs can revisit their dump/analysis for more information.

The following is the example and part of smart debug guide message.

*Comment added by: sys admin.*

*Dear Customer,*

*QSDA\_SmartDebugGuide found your dump had matching pattern of common failure. and we have debug guide for this failure symptom. please refer below for further debugging*

=====  
Issue Title: Modem  
MemHeap Corruption Issue

Description: This means corruption is detected in Memheap Block header

Common Root Cause: Most common root cause is other task is overwriting heap block header

Recommended Problem Area: BSP/HLOS > Stability > Modem Stability

Recommended Actions for further debugging

Related Modules/Source files:

Further Reference(s): 80-NF515-10 MPSS Debug Manual : 6.6 Heap walker extraction 80-NJ839-1  
Heap Debug Guide : 3.1 Scripts to debug memheap issues

Thanks

- QSDA\_SmartDebugGuide

### Qualcomm DDR USB test tool

The [Qualcomm DDR USB test tool](#) (QDUTT) is a DDR test PC tool that integrates several DDR tests for the QCS610 chipset. It simulates DDR-related signals eye diagram and reviews OEMs hardware design.

For more information, see *Qualcomm® DDR USB Test Tool (QDUTT) Tool* (80-P2163-1).

### DDR debug

The DDR debug image (DDI) is a DDR verification and test image that runs on the target device. DDI uses a USB diag port to communicate with QDUTT running on a PC. DDI runs on core0, and can sweep the settings such as drive strength of both controller and DDR part, write calibrated delay circuit (CDC), and read CDC configuration and more.

The test logs are generated through USB Diag during testing, which helps engineers check the marginality in both DDR read/write. For more information, see *Qualcomm DDR USB Test Tool (QDUTT) Tool* (80-P2163-1).

### Qualcomm memory validation suite

The Qualcomm memory validation suite (QMVS) is a DDR validation package including Qualcomm Memory Stress Application (QMesa) in order to develop formal memory verification workflow to ensure that the memory setting runs optimally on customer board and to identify memory problems at an early design stage.

QMVS itself is a PC tool which can work with QMesa. The common usage is QMVS (frequency switching dynamic +suspend/resume) + QMesa (DDR and cache stress test) mix test.

For more information, see *Memory Verification Guidelines for OEM* (80-P7202-4) and *QMVS User Guide* (80-P3255-38).

### QMesa

QMesa is a performance-oriented cache and memory stress testing application which is part of QMVS. It uses the C-library functions and custom, performance optimized assemblies that can achieve the highest possible memory bandwidth that the CPUs can attain.

QMesa stresses the system timings, clocking, voltage, simultaneous switching, and cooling of the CPU, caches, buses, and DDR memory. For more information, see *Qualcomm Open Lab Validation Test Suite* (80-P7820-1) and *QMESA: Qualcomm Memory Stress Application User Guide* (80-NH759-1).

## A.2 Debugging approaches

This section describes general concepts and use cases of major system debug features.

There are two different approaches in debugging. On-target (online and live) debugging and offline debugging.

### A.2.1 On-target debugging

On-target debugging is the traditional way of debugging software issues; it is powerful as most of the information is still accessible from the device. However, this requires that the issue be easily reproducible. It also requires additional hardware components like the host PC, USB cable, and JTAG.

There are various debug tools and mechanisms based on failure symptom, scenario, OS, module, or cores, for example, using JTAG or QXDM Professional tool for subsystems debugging. JTAG is preferred for Linux kernel-level debugging, while Eclipse is widely used for application debugging.

The subsystem debugging can be done by JTAG or QXDM Professional tool. JTAG is preferred for low-level debugging, while QXDM Professional is for protocol-layer debugging.

### A.2.2 Offline debugging

The offline debugging uses captured logs instead of an actual device, is convenient and efficient, as sharing logs from OEMs to QTI is easier than sharing actual devices. This debugging approach is preferred and widely used.

The debugging is done by memory dump and/or QXDM log but various types of logs can be used for offline debugging. The other log files saved in EFS can also be used.

The QXDM Professional tool logs are captured while testing from the device and can be used for many protocol-related debugging issues. When you need extra information, extra debug code can be added any time to collect another set of logs.

Since RAM dump captures most of a memory region but very limited hardware register information, debugging hardware related issue is more challenging. Adding debug code is often needed in this case.

In the case of RAM dump, symbol information is required. The debug symbols for subsystem is stored in an .elf file.

## A.3 Types of errors during software failure

When a device crashes, it falls into one of the following categories:

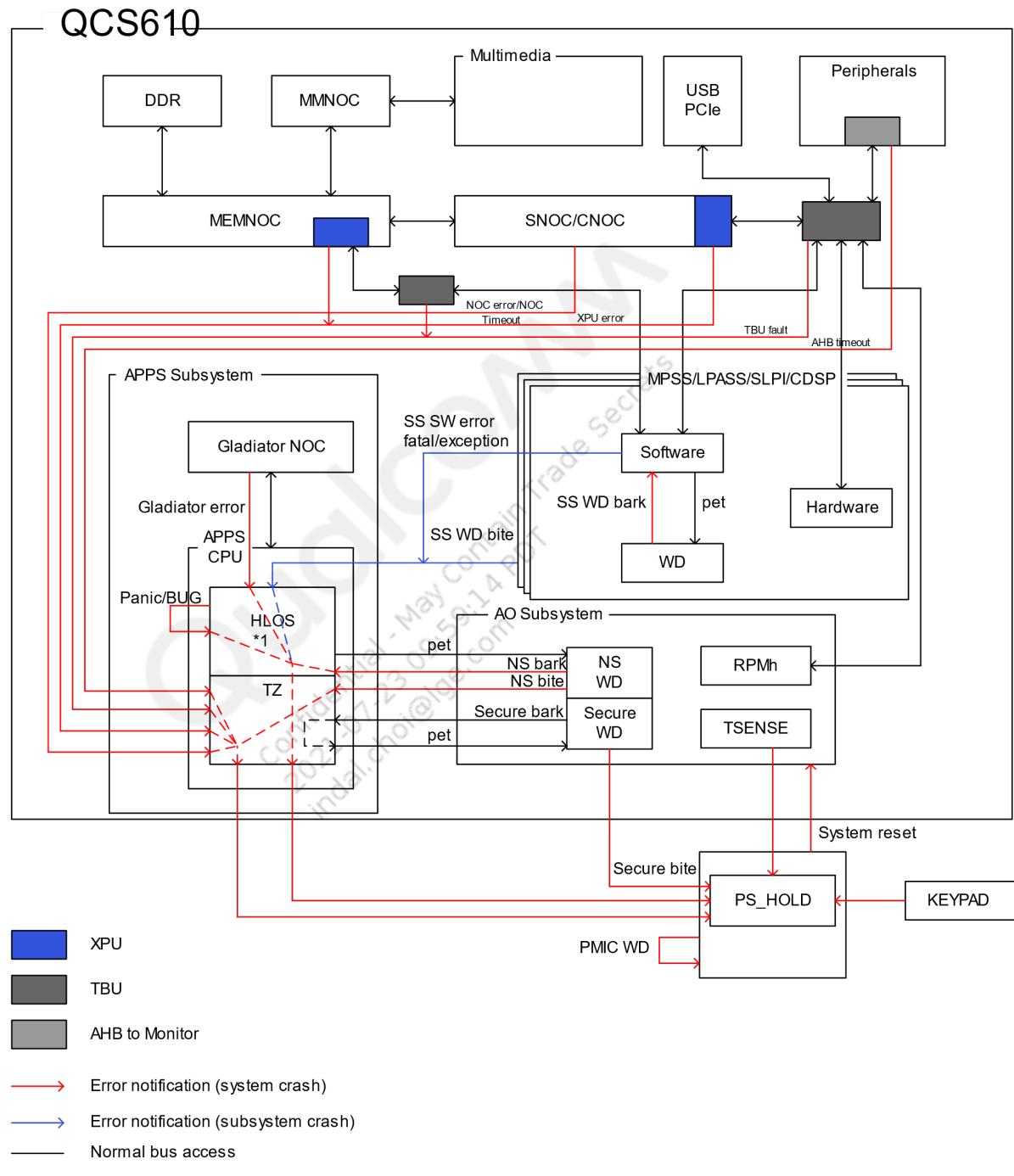
- **Software errors:** Any failures that software can detect fall into this category. It includes error fatal/exception in subsystems. Usually each subsystem has its own error handlers for these types of

failures. Error handlers collect needed information for later debugging. Therefore, it is often easy to triage.

If SSR is enabled, the subsystem errors are considered as recoverable errors from an application processor perspective.

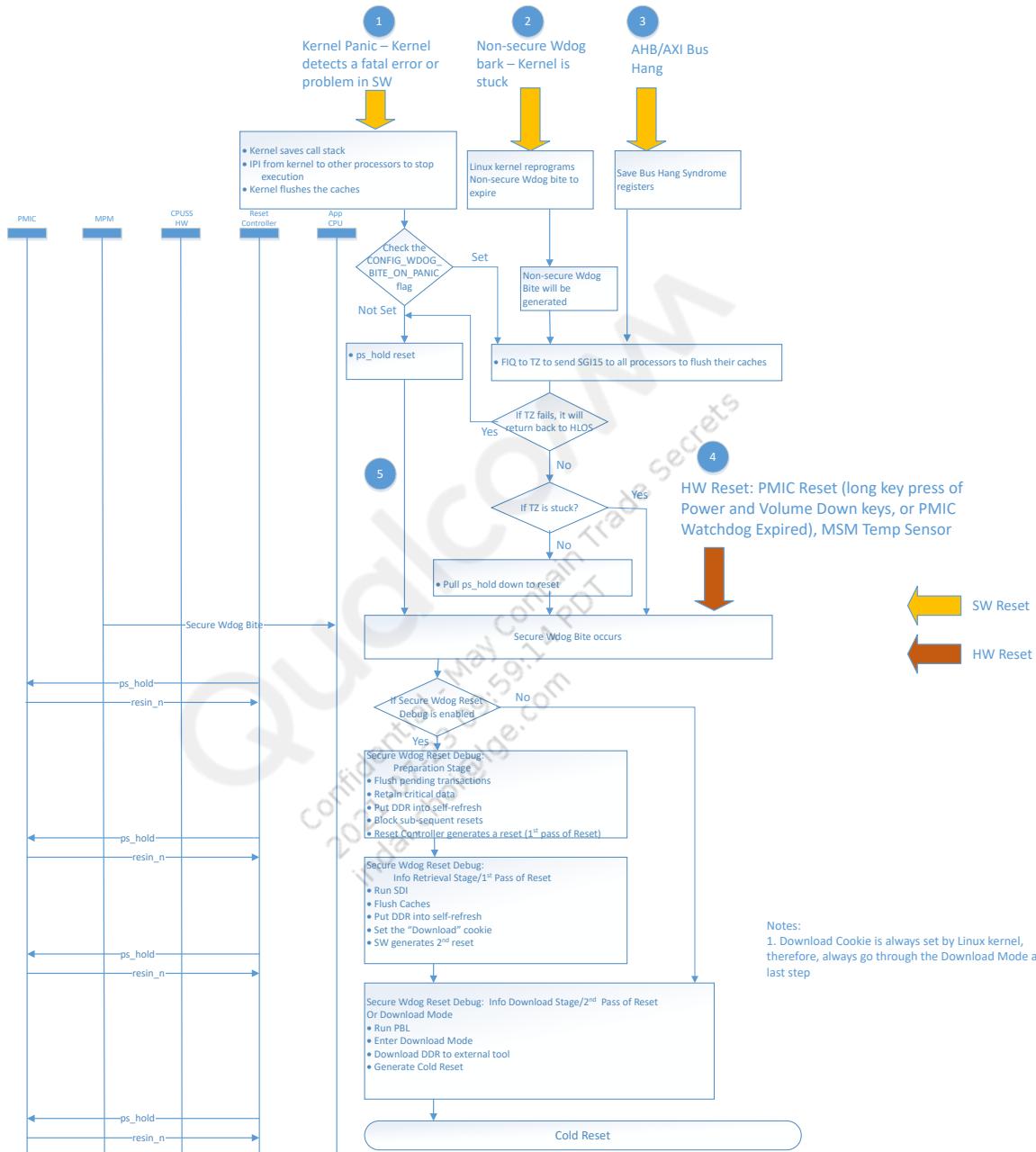
- **Watchdog:** To ensure each subsystem is not being stuck, watchdog is used. After the core fails to service (pet) the watchdog, the watchdog bark/bite sets off an alarm.  
If SSR is enabled, watch dog bark/bite on subsystems are considered as recoverable errors from an application processor perspective.
- **Bus Timeout / Error:** SNOC, CNOC, XPU, TBU and AHB are system infrastructure within QCS610. They are responsible for bus transaction, address translation and memory protection and so on. A failure or timeout may occur on these infrastructures, which results in a system error and is reported to TZ.
- **Hardware error:** Hardware glitches can cause hardware errors. For example, if the voltage to the CPU core is not stable enough, a hardware glitch may occur.  
These glitches usually result in random resets or being stuck. The PMIC WD timeout, TSENSE error, and so on are considered hardware errors from HLOS perspective as the error is neither routed to HLOS nor TZ.
- **System hang:** If the CPU hangs, the hardware watchdog catches the error. However, there are some corner cases where the watchdog or error handler is disabled, and the CPU is stuck.

The following figure shows the types of error generated and error handlers:



**Figure A-4 Error routing overview**

The following figure shows the sequence of each type of reset.



**Figure A-5 Path for different type of errors**

### A.3.1 General error notification

On QCS610, the APSS is the master for the whole system failure, therefore, the software failures from other subsystems are routed to APSS. This routing is done through SMSM and/or interrupt. APSS also sends error notifications to other subsystems through SMSM or GLINK.

When a subsystem gets errors or exceptions, its own error handler first flushes cache, saves the registers into a reserved location of memory, and changes the SMSM state to notify the APSS. RPMh

is using an intraprocessor interrupt directly to the applications processor instead of using SMSM. RPMh does not have cache and RPM saves register values into its own memory.

After APSS receives the message through SMSM or a dedicated interrupt, it invokes the kernel panic handler or SSR module based on configuration. If kernel panic is invoked, it is routed to other subsystems, such as ADSP, again through SMSM, so that other subsystems process their own error handlers to flush cache and save register values. Caches from all subsystems are flushed by this logic.

The system behavior depends on SSR and PD restart configuration. The following figures show how a subsystem notifies the APSS for unrecovered errors and how APSS treats it.

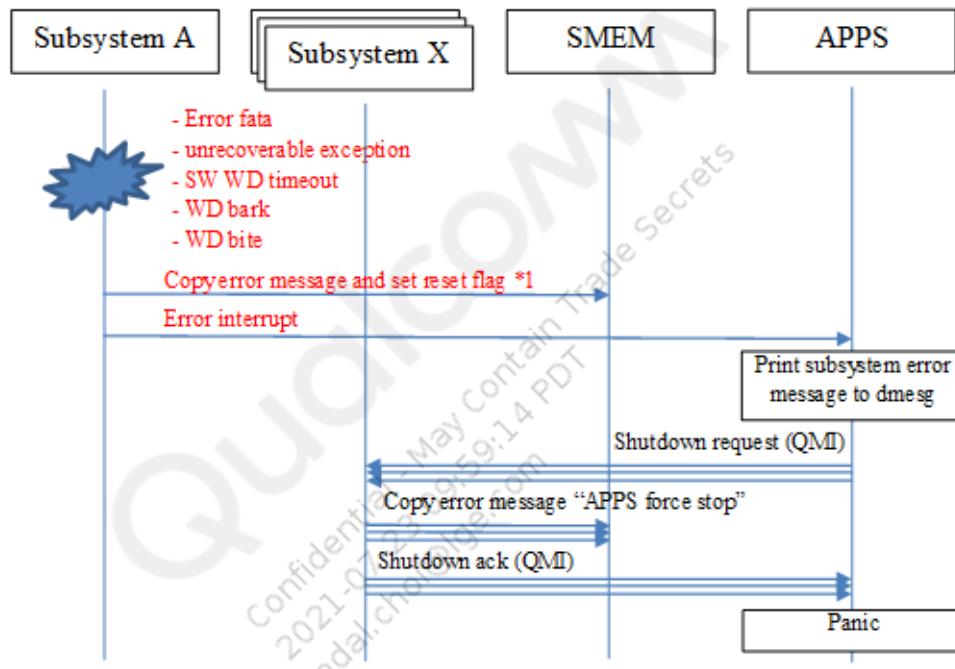


Figure A-6 Subsystem error on QCS610 APSS (SSR and PDR are disabled)

\*1 If WD bite occurs, then the error message and reset flag will not be set to SMEM

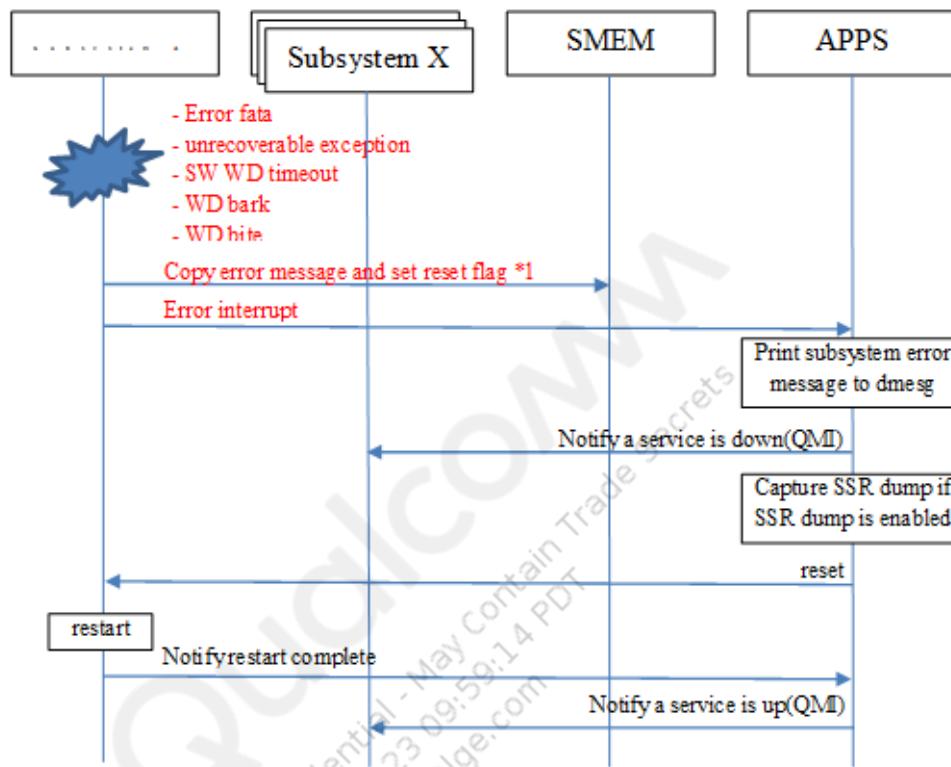
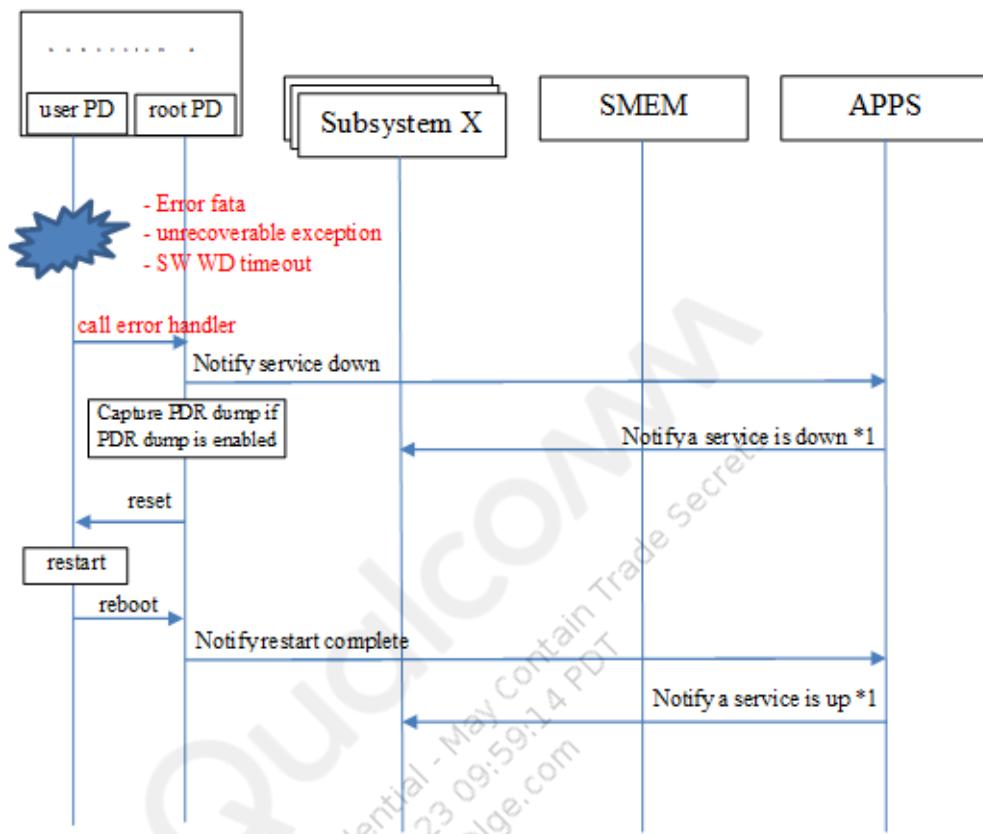


Figure A-7 Subsystem error on QCS610 APSS (SSR is enabled)

\*1 If the WD bite occurs, then the error message and reset flag will not be set to SMEM.



**Figure A-8 Subsystem error on QCS610 APSS (PDR is enabled)**

\*1: In QCS610 no subsystems register, the APSS receives a WLAN service down notification, so this message is not sent by default.

### A.3.2 Software errors

Each core should have its own dedicated error handler, which is expected to handle any failure that the software can detect. These may cover failure cases such as the following:

- Software failures – The programmer has expected an error situation, therefore, it usually links to error handler directly, for example, ASSERT(), error fatal, panic. You may have some hardware-assisted error detection to trigger a software failure as well, for example, watchdog bark.
- Exception – Exceptions are detected by the CPU directly, and it invokes a registered exception handler. The implementation on the exception handlers may vary based on OS. Not all the exceptions are fatal error, for example, TLB miss often happens on both application processor and xDSP but it's often intentional and recoverable by mapping new memory region to TLB/Page table entry by OS. However, if the exception is non-recoverable then it falls to fatal error.

Each subsystem may need to be aware of another subsystem's failure. For the error notifications from subsystem to application processor, SMSM is used.

## A.4 Watchdog

A watchdog (WD or wdog) is a fixed-length counter that enables a system to recover from an unexpected hardware or software catastrophe. Unless the system periodically resets the watchdog timer, the watchdog timer assumes a catastrophe and reset the subsystem or the entire system depending on which watch dog fires.

In general, there are multiple types of watchdog implementations, hardware watchdog and software watchdog, bark, bite, and so on.

**Table A-3 Types of watchdog**

Type of watchdog	Timeout duration	owner	when expired	result
non secure (NS) WD bark	11 sec	HLOS	IRQ to HLOS	HLOS falls to Panic
non secure (NS) WD bite	12 sec	HLOS	FIQ to TZ	TZ asserts PS_HOLD
Secure WD bark	6 sec	TZ	FIQ to TZ	TZ just pets secure WD
Secure WD bite	22 sec	TZ	asserting PS_HOLD	PMIC resets the system
AOP hardware WD bark	10 ms	AOP	IRQ to AOP	AOP falls to error fatal
AOP hardware WD bite	30 ms	AOP	IRQ to application processor	HLOS falls to Panic
SW WD timeout	10 sec	user tasks on SS	calls Error fatal	SS Error fatal <sup>1,2</sup>
(SS) hardware WD bark	2.25 sec	dog task on SS	FIQ to error handler	SS Error fatal/pet WD <sup>1,2</sup>
(SS) NMI due to HW WD	2.4 sec	dog task on SS	NMI to SS	NMI on SS <sup>1</sup>
SS) hardware WD bite	2.5 sec	dog task on SS	IRQ to HLOS	SS hardware reset <sup>1</sup>

1. Error fatal or NMI or hardware reset on subsystem may be falling to panic or just SSR depending on SSR configuration.  
 2. Software WD is available on ADSP and CDSP.

Both software and hardware watchdogs are used in the system.

- The software watchdog is not implemented in all of the subsystems, e.g., RPM does not have software watchdogs, while LPASS and MPSS implement software watchdog.
- The hardware watchdog module is a piece of hardware that is used to ensure that the processor is not stuck or overloaded, and consists of a timer that counts down from a predetermined value. If the timer is not reset (also known as a dog\_kick or petting the dog or servicing the dog) by the corresponding CPU core, it eventually counts to 0 and triggers a watchdog timeout.

## Watchdog for application processor CPU

- Non secure hardware watch dog
  - A timer event fires on HLOS every 10 seconds to pet non-secure WD hardware. If HLOS doesn't pet it for 11 seconds, non-secure WD bark fires and HLOS needs to handle it. If HLOS can handle it, HLOS will fall into panic.
  - If HLOS can't handle non-secure WD bark then non-secure WD bite fires to TZ. TZ then falls into fatal error.
- Secure hardware watchdog
  - Secure WD bark fires every 6 seconds to TZ as FIQ, and the FIQ handler in TZ pets the secure WD hardware. This isn't any error or fatal.
  - If TZ can't handle secure WD bark for 22 seconds, then secure WD bite expires. It asserts PS\_HOLD in PMIC and eventually the entire system is reset.
- Other hardware watchdog characteristics
  - hardware WD is automatically stopped when all cores on application processor CPU go into sleep or JTAG is attached to core0.
  - hardware WD can be disabled by one of the following three methods. This disables both Secure and non-Secure watchdog. (Watchdog enabled: enable=1, Watchdog disabled: enable=0)
    - Add the kernel command line option  
`msm_watchdog.enable=0.`
    - Turn off the kernel feature CONFIG\_MSM\_WATCHDOG (make kernelconfig, system type submenu, down toward the bottom).
    - If the watchdog is enabled at bootup (enable=1):
 

```
echo 1 > /sys/bus/platform/drivers/msm_watchdog/17C10000.qcom,wdt/
              disable
```

## MSM watchdog test module

The MSM watchdog test module is a kernel module (compiled as a .ko file) that can be used to trigger apps processor watchdog barks, nonsecure apps processor watchdog bites, and secure apps processor watchdog bites on demand. It is mainly used to verify crash dump collection in these three reset scenarios.

The module is named as `msm_watchdog_test_module.ko` and is delivered externally as part of the APSS release,

The `msm_watchdog_test_module` and `msm_watchdog_test.sh` script should reside in the following directory –

`build/vendor/qcom/proprietary/kernel-tests-internal/watchdog`

1. Push the module to a desired location on the device to test by using the following ADB command:

```
>adb root
>adb push msm_watchdog_test_module.ko /data/msm_watchdog_test_module.ko
>adb shell insmod /data/msm_watchdog_test_module.ko
```

2. Invoke the module using one of the following commands:

- Non secure watchdog bark:  
echo 1 > /sys/module/msm\_watchdog\_test\_module/parameters/apps\_wdog\_bark
- Non secure watchdog bite:  
echo 1 > /sys/module/msm\_watchdog\_test\_module/parameters/apps\_wdog\_bite
- Secure watchdog Bite:  
echo 1 > /sys/module/msm\_watchdog\_test\_module/parameters/sec\_wdog\_bite
- Secure watchdog bite from TZ by scm call:  
echo 1 > /sys/module/msm\_watchdog\_test\_module/parameters/sec\_wdog\_scm

#### A.4.1 Watchdog for AOP

There is no software watchdog for AOP, the following are the features of the hardware watchdog:

- The software on AOP has to pet AOP WD hardware. If AOP doesn't pet it for 10 ms, AOP WD bark fires and software on AOP needs to handle it then it's eventually AOP error fatal.
- If the software on AOP can't handle AOP WD bark then AOP WD bite fires. AOP is reset by WD hardware and notification is sent to HLOS on application processor. Application processor then falls into panic.
- The hardware WD is automatically stopped when JTAG is attached to AOP.

#### A.4.2 Watchdog for other subsystems

##### Software watchdog

There is software watchdog on MPSS, ADSP, CDSP. Each user tasks register to the dog task. All tasks have to ping to dog task within 10 seconds. If it doesn't do so, the dog task detects it and falls into software error fatal.

##### Hardware watchdog

- On MPSS, ADSP, CDSP, the dog task pets SS WD hardware every 100ms. If the dog task doesn't pet it for 500 ms, SS WD bark FIQ fires and it falls into software error fatal.
- On ADSP and CDSP, hardware WD bark (FIQ) fires every 500 ms seconds and FIQ handler pets hardware WD. This is not error fatal condition.
- If hardware WD bark was not properly handled (i.e., if the software didn't fall into error fatal) then NMI fires in 700 ms. QuRT traps it and flushes the cache, saves TCM, saves register values etc., then just waits for hardware WD bite.
- If hardware WD was not pet within 800 ms, hardware WD bite fires and resets the subsystem. It then notifies "WD fired" to HLOS in application processor. In this case, cache, register values, contents of TCM, etc. may not be saved.

### A.4.3 Log analysis

#### **Non-secure apps watchdog bark**

On subsystems, the following log is displayed:

NMI from apps

The following logs are printed by the application processor kernel:

```
<6>[    26.924581] Watchdog Bark! Now = 26.924571
<6>[    26.927638] Watchdog last pet at 14.628269
<6>[    26.931719] cpu alive mask from last pet 0-3
<0>[    26.935971] Kernel panic - not syncing: Apps watchdog Bark received!
```

#### **Non-secure apps watchdog bite**

On subsystems, the following log is displayed:

NMI from apps

When apps logs are extracted, the following is displayed:

```
<6> apps watchdog bite
```

The following is displayed on TZ diag:

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

#### **Secure watchdog bark**

Secure watchdog bark is a non-fatal FIQ, therefore, a log is not printed to either dmesg or TZ diag.

#### **Secure watchdog bite**

Since secure watchdog bite asserts PS\_HOLD directly without notifying HLOS or TZ, an error message is not displayed for dmesg or TZ diag.

#### **Watchdog bark from subsystem**

For example, in case of MPSS watchdog bark, logs similar to the following are printed by the application processor kernel:

```
SMSM: Modem SMSM state changed to SMSM_RESET.
Fatal error on the modem.

modem subsystem failure reason: dog.c:1225:Watchdog Bark timeout.

subsys-restart: subsystem_restart_dev(): Restart sequence requested for
modem, restart_level = SYSTEM.

Kernel panic - not syncing: subsys-restart: Resetting the SoC - modem crashed.
```

### Watchdog bite from subsystem

For example, in case of MPSS watch dog bite, the following logs are printed by the kernel.

```
<3>[ 110.996166] Watchdog Bite received from modem software!
<3>[ 111.000452] modem subsystem failure reason: SFR Init: WatchDog or
kernel error suspected..
<6>[ 111.008325] subsys-restart: subsystem_restart_dev(): Restart sequence
requested for modem, restart_level = RELATED.
<6>[ 111.020432] subsys-restart: subsystem_shutdown(): [ee055500]: Shutting
down modem
```

## A.5 Erroneous transaction on bus (error and timeout)

A bus hang occurs when a bus master, processor or nonprocessor, accesses a slave that may have one of the key clocks off, such that an access to this slave waits indefinitely and never returns. If the access that hung was from a processor, the processor waits indefinitely, and the only way for the processor to recover is to restart the system by watchdog.

To monitor and control such scenarios, bus hang monitors are implemented in all the buses in the system, which is NOC timeout monitor and AHB timeout monitor.

To overcome the challenge of having invalid access to DDR from unintended master without permission, XPU and TBU (SMMU) are introduced to build a secure system for debug.

### NOC error and timeout

NOC\_ERROR and timeout are fatal errors when NOC detected decode error or unclocked bus access, and so on. The NOC generates FIQ to TZ.

### AHB timeout

There are couple of AHBs are used within the QCS610 system as an interconnection in peripherals. If access timeout occurs on a transaction, it generates FIQ to TZ to report AHB timeout.

### xPU violations

The xPU violations occur when a master subsystem tries to access a system resource. For example, memory region, device controller registers, and so on, for which either proper access permission is not set or the device controller is not functional due to unavailability of the clocks.

The xPU violations should be configured as fatal during the development and testing phase to ensure capturing all of the unexpected issues. To configure the XPU violation as a fatal error, the Linux kernel config should set CONFIG\_MSM\_XPU\_ERR\_FATAL=y.

The various fields in the XPU dump can be interpreted as follows:

- XPU ID – Refers to an ID to identify a device controller or hardware module
- uPhysicalAddress – Address that was read/write with reference to XPU ID
- BID or Bus ID – SNOC, PNOC, and CNOC
- PID or Port ID – Port number on the bus identified with BID from which the transaction originated

- Master ID (MID) – MID that initiated the transaction
- Virtual Master ID (VMID) – May be the same as MID, but will uniquely identify a master

### TBU (SMMU) error

- TBU is used for a system-level MMU. It supports address translation from an input address to an output address, based on address mapping and memory attribute information held in translation tables.
- SMMU error is a fatal error, such as when a subsystem or peripheral tries to access an invalid address or with invalid attribute.
- SMMU error is routed to TZ.

For more information, see *Arm System Memory Management Unit Architecture Specification*, [IH10062D](#).

## A.6 Hardware reset

A hardware reset can be caused by a secure watchdog, TSENSE, PMIC or SMPL. Figuring out the cause is crucial for reset debugging. A further debugging approach might be different based on this information.

Any of error handlers in software may not run for the reset issues. Therefore, the user may not save any data before a reset happens. This results in lack of information that for finding out the root cause of the issues.

On QCS610, watchdog reset debugging is improved by debug-through-watchdog. It reserves power rail to CPU core and system cache when a secure watchdog reset happens, allowing SDI and DCC to flush the cache and save debug information upon hardware reset before resetting the system.

RST\_STAT.BIN and PMIC\_PON.BIN are generated when a reset occurs, and these hold the AOSS\_CC\_RESET\_STATUS value and PMIC power on status register dump.

The following figure is an example:

Registers	
Reset Registers (SBL) :	
GCC_RESET_STATUS	: 0x03
PMIC Registers (SBL) :	
PON_REASON1	: 0x21
ON_REASON	: 0x40
WARM_RESET_REASON1	: 0x02
POFF_REASON1	: 0x02
OFF_REASON	: 0x80

The following is the general warm reset sequence between QCS610 and PMIC.

1. MSM pull PS\_HOLD low.
2. PMIC pull PON\_RESET\_N low to keep MSM in reset.

3. PMIC executes the warm reset.
4. PMIC pull PON\_RESET\_N high to take MSM out of reset.

### SDM reset status

After a reset comes into QCS610 by watchdog and other reason. The hardware register AOSS\_CC\_RESET\_STATUS (which is used to be called GCC\_RESET\_STATUS) holds reset status. The following table shows a description of the hardware register AOSS\_CC\_RESET\_STATUS. In general, writing to the status bits clears the status.

**Table A-4 Hardware AOSS\_CC\_RESET\_STATUS**

Bits	Name	Type	Description
9	PS_HOLD_STATUS	Read-write	Set when PS_HOLD drops. If this bit is set together with another status bit, then the status should be interpreted as PS_HOLD caused by the condition indicated by the status bit. 1 – PS_HOLD dropped
8	RESERVED	–	–
7	RESERVED	–	–
6	PMIC_ABNORMAL_RESIN_RESET_STATUS	Read-write	■ 1 – pmic resin_n occurred because of PMIC abnormal conditions and not because of PS_HOLD
5	MSM_TSENSE1_RESET_STATUS	Read-write	Set when temperature sensor activates ■ 1 – tsens1 triggered reset
4	PROC_HALT_CTL_STATUS	Read-write	■ 0 – NO_HALT ■ 1 – HALT
3	SRST_RESET_STATUS	Read-write	Set high when any SRST triggered ■ 1 – srst triggered ■ Reset state – 0x0
2	MSM_TSENSE0_RESET_STATUS	Read-write	Set when temperature sensor activates 1 – tsens0 triggered reset
1	PMIC_RESIN_RESET_STATUS	Read-write	Set when PMIC resetin occurs ■ 1 – pmic resin_n triggered ■ Reset state – 0x0
0	SECURE_WDOG_EXPIRE_RESET_STATUS	Read-write	Helps track watchdog reset loops ■ 1 – Watchdog triggered ■ Reset state – 0x0

Most of system crashes including software errors like panic, BUG, or TZ error eventually assert PS\_HOLD (which causes PMIC\_RESIN) and then falls into secure WD reset by design. Therefore, bit0 and bit1 are displayed, which are set on most of crash dump.

## A.6.1 PMIC PON reset status

We used to use PMIC\_PON.bin to see PMIC reset reason, but because SDI hardware and software execute PS\_HOLD warm resets before the RAM dump occurs, it is preferred to use PMON\_HIS.BIN over PMIC\_PON.BIN.

The following is a PMIC\_PON screenshot from PON\_HSI.bin dump:

There are three logs in the dump. It starts from offset +0x0C from the top. Each entry is 8 bytes and can be decoded like the following.

Note that the third log has the oldest RTC time stamp and first log is the latest, which means the third log was recorded first. The logging starts from the bottom and grows to the top side. So the third (oldest) log is provides details regarding the reset.

Register Name	First reset	Second reset	Third reset	Description
PON_PON_REASON1	80	80	80	0x80: Triggered from KPDPWR press
PON_WARM_RESET_REASON1	02	02	02	0x0 : PS_HOLD
PON_ON_REASON	80	40	40	0x40: warm reset 0x80: normal powering-on
PON_POFF_REASON1	00	00	00	N/A
PON_OFF_REASON	04	04	04	0x04: RAW_DVDD_RB_OCCURED
PON_FAULT_REASON1	00	00	00	N/A
PON_FAULT_REASON2	00	00	00	N/A
Coded RTC time	7D	7D	82	RTC time

In this example, PON\_ON\_REASON of the first reset was 0x80, which is normal powering-on. This is the usual reset caused by PS\_HOLD. This reset is followed by twice the amount of 0x40:warm resets because a couple of warm resets happen before entering the dump mode.

**Table A-5 PON reason**

Bit	Name	Description
7	PON_SEQ	<ul style="list-style-type: none"> <li>■ PMIC enters ON state because of a power-on sequence</li> <li>■ 0x1: Trigger received</li> </ul>
6	WARM_SEQ	<ul style="list-style-type: none"> <li>■ PMIC enters ON state because of a warm-reset sequence</li> <li>■ 0x1: Trigger received</li> </ul>

**Table A-6 Hardware PMIC PON reason**

Bit	PON_REASON1 (0x808)	Type	Description
7	KPDPWR_N	Read	Keypad power button press
6	CBLPWR_N	Read	USB cable insertion
5	PON1	Read	
4	USB_CHG	Read	USB charger insertion
3	DC_CHG	Read	Wall charger insertion
2	RTC	Read	RTC alarm/calendar event
1	SMPL	Read	Power loss
0	Hard reset	Read	

**Table A-7 Hardware warm reset reason**

Bit	WARM_RESET_REASON1 (0x80A)	Type	Description
7	KPDPWR_N	Read	Keypad power button press
6	RESIN_N	Read	RESET IN
5	KPDPWR_AND_RESIN	Read	Keypad and resin
4	GP2	Read	General purpose use
3	GP1	Read	General purpose use
2	PMIC_WD	Read	PMIC watchdog
1	PS_HOLD	Read	PS_HOLD pulled low
0	SOFT	Read	Software trigger reset

For more information, see *HLOS PMIC PON Software User Guide* (80-P2484-40).

## A.7 System hang during power collapse

The power consumption is saved by putting cores into Power Collapse mode whenever possible. Entering Power Collapse requires clock and voltage manipulation. Some abnormal behaviors may be observed during such operations. The Power Collapse mode is supported in most of the cores to maximize the device standby time.

Being stuck while waking up from power collapse is an issue since an error handler is not expected to cover the situation. Multiple hardware components are involved during the operations.

For example, while APSS wakes up from power collapse, MPM, RPMh, SPM, and TZ are all involved since RPMh wakes up APSS, and the TZ image runs on APSS upon wake-up. Many clock and power rails are accessed during the operation, which may cause abnormal behavior of CPU cores.

Identifying which stage of power collapse the device was running on is crucial. The RPMh log and TZ counter are used to identify the image level first. After the module is identified, the QDSS hardware events and software events can be used. Major events on the device can be tracked and monitored by those.

For additional information on power collapse/wake-up debugging, see *Resource Power Manager (RPM.BF) Debug Manual* (80-P9301-16).

Qualcomm Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

# B QTI GStreamer elements

This appendix provides details on QTI-added and modified GStreamer elements.

For more information, see the documentation at <https://gstreamer.freedesktop.org/documentation>.

## B.1 qtiqmmfsrc

The qtiqmmfsrc plugin can be used to capture video frames via the QMMF service.

The plugin consists of the main class called GstQtiqmmfsrc that acts as a wrapper on top of the QMMF Recorder Client with separate pads for video and image streams.

The pads store the creation time parameters (passed as GstCaps during pipeline creation) for the particular stream while the GstQtiqmmfsrc takes that information, translates it to the QMMF Recorder Client parameters and calls the necessary APIs on each state transition of the element. For video and image pads, the camera device ID, which will be used for this instance of the plugin can be set via the "camera" property (by default this ID is 0).

1. During transitioning between NULL and READY state, the plugin opens and initializes the camera device with the given ID.
2. When transitioning to PAUSED state from READY, the plugin translates the set pad parameters and makes calls to the QMMF service in order to create the source streams for each pad.
3. The session streams are started when transition is done to PLAYING state.

When a frame is received in the main class, it will create a GstBuffer and send it to the relevant pad buffer queue. The pad will push the buffer to its linked sink pad from the next plugin. Buffer allocation takes place inside the QMMF service while the plugin only ensures that the buffers are returned to the service when they are no longer in use.

For video, buffers are sent by the QMMF service when the plugin state is PLAYING, but for image pad a "capture-image" signal must be sent. For each "capture-image" a single buffer will be sent from the QMMF service.

**Table B-1 qtiqmmfsrc pad templates**

Pad name	Template
video_%u (output) ■ Presence – on request ■ Presence – src	video/x-h264 profile: { (string)baseline, (string)main, (string)high } level: { (string)1, (string)1.3, (string)2, (string)2.1, (string)2.2, (string)3, (string)3.1, (string)3.2, (string)4, (string)4.1, (string)4.2, (string)5, (string)5.1, (string)5.2 } width: [ 16, 3840 ] height: [ 16, 2160 ] framerate: [ 0/1, 30/1 ] video/x-h264(memory:GBM) profile: { (string)baseline, (string)main, (string)high } level: { (string)1, (string)1.3, (string)2, (string)2.1, (string)2.2, (string)3, (string)3.1,

**Table B-1 qtiqmmfsrc pad templates (cont.)**

Pad name	Template
	<pre>(string)3.2, (string)4, (string)4.1, (string)4.2, (string)5, (string)5.1, (string)5.2 } width: [ 16, 3840 ] height: [ 16, 2160 ] framerate: [ 0/1, 30/1 ] video/x-h265 profile: { (string)main } level: { (string)3, (string)4, (string)5, (string)5.1, (string)5.2 } width: [ 16, 3840 ] height: [ 16, 2160 ] framerate: [ 0/1, 30/1 ] video/x-h265(memory:GBM) profile: { (string)main } level: { (string)3, (string)4, (string)5, (string)5.1, (string)5.2 } width: [ 16, 3840 ] height: [ 16, 2160 ] framerate: [ 0/1, 30/1 ] video/x-raw format: { (string)NV12 } width: [ 16, 3840 ] height: [ 16, 2160 ] framerate: [ 0/1, 30/1 ] video/x-raw(memory:GBM) format: { (string)NV12 } width: [ 16, 3840 ] height: [ 16, 2160 ] framerate: [ 0/1, 30/1 ]  image/jpeg  width: [ 16, 4096 ] height: [ 16, 4096 ] framerate: [ 0/1, 120/1 ]</pre>
image_%u (output)	<pre>image/jpeg width: [ 16, 4000 ] height: [ 16, 4000 ] framerate: [ 0/1, 30/1 ]  image/jpeg(memory:GBM) width: [ 16, 4000 ] height: [ 16, 4000 ] framerate: [ 0/1, 30/1 ] video/x-bayer format: { (string)RAW8 (string)RAW10, (string)RAW12, (string)RAW16 } width: [ 16, 4000 ] height: [ 16, 4000 ] framerate: [ 0/1, 30/1 ] video/x- bayer(memory:GBM) level: { (string)RAW8 (string)RAW10, (string)RAW12, (string)RAW16 } width: [ 16, 4000 ] height: [ 16, 4000 ] framerate: [ 0/1, 30/1 ] video/x-raw format: { (string)NV21 } width: [ 16, 4000 ] height: [ 16, 4000 ] framerate: [ 0/1, 30/1 ] video/x-raw(memory:GBM) format: { (string)NV21 } width: [ 16, 4000 ] height: [ 16, 4000 ]</pre>

The following table provides some of the main properties of qtiqmmfsrc.

To view the effect of some of the following properties, the camera sensor must be tuned.

**Table B-2 qtiqmmfsrc properties**

Property	Description
name	<p>The name of the object.</p> <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: "qtiqmmfsrc0"</li> </ul>
camera	<p>The camera device ID to be used by video/image pads.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0</li> </ul>
slave	<p>Sets camera as a slave device. This is used for multi-stream use cases.</p> <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>

**Table B-2 qtiqmmfsrc properties (cont.)**

Property	Description
ldc	Sets the LDC imaging <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
shdr	Sets the super high dynamic range imaging <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
eis	Image Stabilization technology to reduce the effects of camera shake. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
effect	The effect applied to the camera frames. <ul style="list-style-type: none"> <li>▪ type: Enum "GstCameraEffectMode"</li> <li>▪ access: read/write <ul style="list-style-type: none"> <li>▫ (0): off – No color effect will be applied.</li> <li>▫ (1): mono – A 'monocolor' effect, where the image is mapped into a single color.</li> <li>▫ (2): negative – A 'photo-negative' effect, where the image's colors are inverted.</li> <li>▫ (3): solarize – A 'solarisation' effect (Sabattier effect), where the image is wholly or partially reversed in tone.</li> <li>▫ (4): sepia – A 'sepia' effect, where the image is mapped into warm gray, red, and brown tones.</li> <li>▫ (5): posterize – A 'posterization' effect, where the image uses discrete regions of tone rather than a continuous gradient of tones.</li> <li>▫ (6): whiteboard – A 'whiteboard' effect, where the image is typically displayed as regions of white with black or grey details.</li> <li>▫ (7): blackboard – A 'blackboard' effect, where the image is typically displayed as regions of black, with white or grey details.</li> <li>▫ (8): aqua – An 'aqua' effect where a blue hue is added to the image.</li> </ul> </li> <li>▪ default: 0, "off"</li> </ul>
antibanding	The camera antibanding routine for the current illumination condition. <ul style="list-style-type: none"> <li>▪ type: Enum "GstAntibandingMode"</li> <li>▪ access: read/write <ul style="list-style-type: none"> <li>▫ (0): off – The camera device will not adjust exposure duration to avoid banding problems.</li> <li>▫ (1): 50 Hz – The camera device will adjust exposure duration to avoid banding problems with 50 Hz illumination sources.</li> <li>▫ (2): 60 Hz – The camera device will adjust exposure duration to avoid banding problems with 60 Hz illumination sources.</li> <li>▫ (3): auto – The camera device will automatically adapt its antibanding routine to the current illumination condition.</li> </ul> </li> <li>▪ default: 3, "auto"</li> </ul>

**Table B-2 qtiqmmfsrc properties (cont.)**

Property	Description
exposure-compensation	<p>Adjust (Compensate) camera images target brightness. Adjustment is measured as a count of steps.</p> <ul style="list-style-type: none"> <li>■ type: Integer</li> <li>■ access: read/write default: 0</li> <li>■ range: -12 - 12</li> </ul>
exposure-lock	<p>Locks current camera exposure routine values from changing.</p> <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
white-balance-mode	<p>The desired mode for the camera's white balance routine.</p> <ul style="list-style-type: none"> <li>■ type: Enum "GstCameraWiteBalanceMode"</li> <li>■ access: read/write <ul style="list-style-type: none"> <li>□ (0):off – The camera device's auto-white balance routine is disabled.</li> <li>□ (1): manual-cc-temp – The auto-white balance routine is inactive and manual color correction temperature is used, which is set via the 'manual-wb-settings' property.</li> <li>□ (2): manual-rgb-gains – The auto-white balance routine is inactive and manual R/G/B gains are used which are set via the 'manual-wb-settings' property.</li> <li>□ (3):auto – The camera device's auto-white balance routine is active.</li> <li>□ (4):shade – The camera device uses shade light as the assumed scene illumination for white balance.</li> <li>□ (5): incandescent – The camera device uses incandescent light as the assumed scene illumination for white balance.</li> <li>□ (6): fluorescent – The camera device uses fluorescent light as the assumed scene illumination for white balance.</li> <li>□ (7): warm-fluorescent – The camera device uses warm fluorescent light as the assumed scene illumination for white balance.</li> <li>□ (8): daylight – The camera device uses daylight light as the assumed scene illumination for white balance.</li> <li>□ (9):cloudy-daylight – The camera device uses cloudy daylight light as the assumed scene illumination for white balance.</li> <li>□ (10):twilight – The camera device uses twilight light as the assumed scene illumination for white balance.</li> </ul> </li> <li>■ default: 3, "auto"</li> </ul>
white-balance-lock	<p>Locks current White Balance values from changing. Affects only non-manual white balance modes.</p> <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
exposure-mode	<p>The desired mode for the camera's exposure routine.</p> <ul style="list-style-type: none"> <li>■ type: Enum "GstCameraAEMode"</li> <li>■ access: read/write <ul style="list-style-type: none"> <li>□ (0):off – The auto exposure routine is disabled. Manual exposure time will be used.</li> <li>□ (1): auto - The auto exposure routine is active.</li> </ul> </li> <li>■ default: 1, "auto"</li> </ul>

**Table B-2 qtiqmmfsrc properties (cont.)**

Property	Description
manual-exposure-time	<p>Manual exposure time in nanoseconds. Used when the Exposure mode is set to 'off'.</p> <ul style="list-style-type: none"> <li>■ type: 64 bit Integer</li> <li>■ access: read/write</li> <li>■ default: 33333333</li> <li>■ Range: 0 - 9223372036854775807</li> </ul>
exposure-metering	<p>The desired mode for the camera's exposure metering routine.</p> <ul style="list-style-type: none"> <li>■ type: Enum " GstCameraExposureMetering"</li> <li>■ access: read/write <ul style="list-style-type: none"> <li>□ (0): average – The camera device's exposure metering is calculated as average from the whole frame.</li> <li>□ (1): center-weighted – The camera device's exposure metering is calculated from the center region of the frame.</li> <li>□ (2): spot – The camera device's exposure metering is calculated from a chosen spot.</li> <li>□ (6): custom – The camera device's exposure metering is calculated from a custom metering table.</li> </ul> </li> <li>■ default: 0, "average"</li> </ul>
focus-mode	<p>Whether auto-focus is currently enabled, and in what mode it is.</p> <ul style="list-style-type: none"> <li>■ type: Enum "GstCameraFocusMode"</li> <li>■ access: read/write <ul style="list-style-type: none"> <li>□ (0):off – The auto focus routine is disabled.</li> <li>□ (1):auto – The auto focus routine is active.</li> <li>□ (2):macro – In this mode, the auto focus algorithm is optimized for focusing on objects very close to the camera.</li> <li>□ (3):continuous – In this mode, the AF algorithm modifies the lens position continually to attempt to provide a constantly-in-focus image stream.</li> <li>□ (4):edof – The camera device will produce images with an extended depth of field automatically; no special focusing operations need to be done before taking a picture.</li> </ul> </li> <li>■ default: 0, "off"</li> </ul>
adrc	<p>Automatic dynamic range compression</p> <p>type: Boolean</p> <p>access: read/write default: false</p>
iso-mode	<p>Sets the ISO Exposure mode.</p> <ul style="list-style-type: none"> <li>■ type: Enum "GstCameraISOMode"</li> <li>■ access: read/write <ul style="list-style-type: none"> <li>□ (0):auto – The ISO exposure mode will be chosen depending on the scene.</li> <li>□ (1):deblur – The ISO exposure sensitivity set to prioritize motion deblur.</li> <li>□ (2):100 – The ISO exposure sensitivity set to prioritize level 100.</li> <li>□ (3):200 – The ISO exposure sensitivity set to prioritize level 200.</li> <li>□ (4): 400 – The ISO exposure sensitivity set to prioritize level 400.</li> <li>□ (5): 800 – The ISO exposure sensitivity set to prioritize level 800.</li> </ul> </li> </ul>

**Table B-2 qtiqmmfsrc properties (cont.)**

Property	Description
	<ul style="list-style-type: none"> <li>□ (6): 1600 – The ISO exposure sensitivity set to prioritize level 1600.</li> <li>□ (7): 3200 – The ISO exposure sensitivity set to prioritize level 3200.</li> <li>■ default: 0, "auto"</li> </ul>
infrared-mode	<p>Sets the Infrared mode.</p> <ul style="list-style-type: none"> <li>■ type: Enum ""GstCameralRMode"</li> <li>■ access: read/write</li> <li>□ (0):off – The infrared is disabled.</li> <li>□ (1): on – The infrared is active until canceled.</li> <li>□ (2): auto – The infrared is turned OFF or ON depending on the conditions.</li> <li>■ default: 0, "off"</li> </ul>
noise-reduction	<p>Enables the Noise Reduction Filter mode.</p> <ul style="list-style-type: none"> <li>■ type: Enum ""GstCameraNoiseReduction"</li> <li>■ access: read/write</li> <li>□ (0): off – No noise reduction filter is applied.</li> <li>□ (1): fast – Temoral Noise Reduction (TNR) fast mode.</li> <li>□ (2): hq – TNR High Quality mode.</li> <li>■ default: 0, "off"</li> </ul>
zoom	<p>The camera zoom rectangle ('&lt;X, Y, WIDTH, HEIGHT &gt;') in sensor active pixel array coordinates.</p> <ul style="list-style-type: none"> <li>■ type: GstValueArray of GValues of type Integer</li> <li>■ access: read/write</li> <li>■ default: NONE</li> </ul>
custom-exposure-table	<p>A GstStructure describing custom exposure table</p> <ul style="list-style-type: none"> <li>■ type: String (Either the exposure table in string form or path for file with exposure table values)</li> <li>■ access: read/write</li> <li>■ default: "org.codeaurora.qcamera3.exposuretable;"</li> </ul>
defog-table	<p>A GstStructure describing the defog table.</p> <ul style="list-style-type: none"> <li>■ type: String (Either the exposure table in string form or path for file with defog table values)</li> <li>■ access: read/write</li> <li>■ default: "org.quic.camera.defog;"</li> </ul>
itm-data	<p>Describes the local tone mapping data</p> <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: "org.quic.camera.l tmDynamicContrast;"</li> </ul>
noise-reduction-tuning	<p>Describes noise reduction tuning</p> <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: NONE</li> </ul>

**Table B-2 qtiqmmfsrc properties (cont.)**

Property	Description
sharpness	Controls sharpness strength value <ul style="list-style-type: none"> <li>■ type: Integer</li> <li>■ access: read/write</li> <li>■ default: 2</li> <li>■ Range: 0-6</li> </ul>
contrast	Image contrast strength <ul style="list-style-type: none"> <li>■ type: Integer</li> <li>■ access: read/write</li> <li>■ default: 5</li> <li>■ Range: -100-100</li> </ul>
saturation	Image contrast strength <ul style="list-style-type: none"> <li>■ type: Integer</li> <li>■ access: read/write</li> <li>■ default: 5</li> <li>■ Range: 0-10</li> </ul>

**Table B-3 qtiqmmfsrc video pad properties**

Property	Description
source-index	The index of the source video pad to which the pad is linked. <ul style="list-style-type: none"> <li>■ type: Integer</li> <li>■ access: read/write</li> <li>■ default:-1</li> </ul>
framerate	Target framerate in frames per second for displaying. <ul style="list-style-type: none"> <li>■ type: Double</li> <li>■ access: read/write</li> <li>■ default: 30.0</li> <li>■ range: 0 - 30</li> </ul>
bitrate	Target bitrate in bits per second for compressed streams. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 6000000</li> </ul>
bitrate-control	Bitrate control method for compressed streams. <ul style="list-style-type: none"> <li>■ type: Enum (GstVideoControlRate)</li> <li>■ access: read/write</li> <li>■ default: 3,</li> <li>■ "maxbitrate" Available Values:<ul style="list-style-type: none"> <li>□ (0): disable – Disable (1): variable - Variable</li> <li>□ (2): constant - Constant</li> <li>□ (3): maxbitrate – Max Bitrate</li> <li>□ (4): constant-skip-frames – Variable Skip Frames</li> </ul></li> </ul>

**Table B-3 qtiqmmfsrc video pad properties (cont.)**

Property	Description
	<ul style="list-style-type: none"> <li>□ (5): variable-skip-frames - Constant Skip Frames</li> <li>□ (6): maxbitrate-skip-frames – Max Bitrate Skip Frames</li> </ul>
quant-i-frames	Quantization parameter on I-frames for compressed streams. This parameter is applicable for initial qp values. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 27</li> </ul>
quant-p-frames	Quantization parameter on P-frames for compressed streams. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 28</li> </ul>
quant-b-frames	Quantization parameter on B-frames for compressed streams. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 28</li> </ul>
min-qp	Minimum QP value allowed during rate control for compressed streams. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 10</li> <li>■ range : 0 - 51</li> </ul>
max-qp	Max QP value allowed during rate control for compressed streams. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 51</li> <li>■ range : 0 - 51</li> </ul>
min-qp-i-frames	Minimum QP value allowed on I-Frames during rate control for compressed streams. This parameter is applicable for session qp values. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 10</li> <li>■ range : 0 - 51</li> </ul>
max-qp-i-frames	Max QP value allowed on I-Frames during rate control for compressed streams. This parameter is applicable for session qp values. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 51</li> <li>■ range : 0 - 51</li> </ul>
min-qp-p-frames	Minimum QP value allowed on P-Frames during rate control for compressed streams. This parameter is applicable for session qp values. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> </ul>

**Table B-3 qtiqmmfsrc video pad properties (cont.)**

Property	Description
	<ul style="list-style-type: none"> <li>▪ default: 10</li> <li>▪ range : 0 - 51</li> </ul>
max-qp-p-frames	<p>Max QP value allowed on P-Frames during rate control for compressed streams. This parameter is applicable for session qp values.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 51</li> <li>▪ range : 0 - 51</li> </ul>
min-qp-b-frames	<p>Minimum QP value allowed on B-Frames during rate control for compressed streams. This parameter is applicable for session qp values.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 10</li> <li>▪ range : 0 - 51</li> </ul>
max-qp-b-frames	<p>Max QP value allowed on B-Frames during rate control for compressed streams. This parameter is applicable for session qp values.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 51</li> <li>▪ range : 0 - 51</li> </ul>
idr-interval	<p>IDR interval for compressed streams</p> <p>Max QP value allowed on B-Frames during rate control for compressed streams.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 1</li> <li>▪ range : 0 – 4294967295</li> </ul>
crop	<p>Crop rectangle ('&lt;X, Y, WIDTH, HEIGHT&gt;'). Applicable only for JPEG and YUY2 formats.</p> <ul style="list-style-type: none"> <li>▪ type: GstValueArray of GValues of type Integer</li> <li>▪ access: read/write</li> <li>▪ default: &lt;0,0,0,0&gt;</li> </ul>
extra-buffers	<p>Number of additional buffers that will be allocated.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write (NULL or READY state)</li> <li>▪ default: 0</li> <li>▪ range : 0 – 4294967295</li> </ul>

## B.2 qtivdec

The qtivdec GStreamer element is V4L2-based video decoder that uses QTI video hardware cores to decode video. The qtivdec plugin is derived from the GstVideoDecoder GStreamer base class for video decoders.

1. GstVideoDecoder calls start when the element is activated.
2. GstVideoDecoder calls setformat to notify qtivdec of the input configurations, and hands over the frame to the qtivdec.
3. qtivdec processes and notifies the base class that the frame processing is complete.
4. After the data is processed, the base class calls stop on qtivdec to notify that processing has stopped.

For more information on GstVideoDecoder class and subclasses, see [GstVideoDecoder in Installing GStreamer Manual](#).

**Table B-4 qtivdec pad templates**

Pad name	Template
sink (input) <ul style="list-style-type: none"> <li>■ Presence – always</li> <li>■ Direction – sink</li> </ul>	video/x-h264 stream-format: { (string)byte-stream } alignment: { (string)au } video/x-h265 stream-format: { (string)byte-stream } alignment: { (string)au } video/mpeg mpegversion: { (int)1, (int)2, (int)4 } systemstream: false parsed: true
source (output) <ul style="list-style-type: none"> <li>■ Presence – always</li> <li>■ Direction – src</li> </ul>	video/x-raw format: NV12 width: [ 32, 4096 ] height: [ 32, 4096 ] video/x-raw(memory:GBM) format: NV12 width: [ 32, 4096 ] height: [ 32, 4096 ]

**Table B-5 qtivdec properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: "qtivdec0"</li> </ul>
silent	Controls the verbose output. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
decode-order	Sets the output decode order. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
skip-frames	Controls frame skipping. It is used in trick play. When enabled, only I frames are decoded whereas P and B frames are skipped. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
turbo	Runs the hardware video decoder in Performance/Turbo mode. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>

### B.3 omxh264enc

The omxh264enc GStreamer element is part of the gst-omx plugin. The gst-omx plugin is an adapter for the available OpenMAX IL components and maps the OpenMAX APIs and states to the GStreamer APIs and states. The omxh264enc element is an OpenMAX H264 video encoder that provides hardware-accelerated H264 video encoding using the OpenMAX component on platforms that support it.

QTI has added support for its OpenMAX video encode component in the gst-omx plugin to enable hardware-accelerated video encode on its video core. The omxh264enc element is based on GstVideoEncoder base class for video encoders.

For more information on the GstVideoEncoder methods and API call flow, see [GstVideoEncoder in Installing GStreamer Manual](#).

**Table B-6 omxh264enc pad templates**

Pad name	Template
sink (input) <ul style="list-style-type: none"> <li>■ Presence – always</li> <li>■ Direction – sink</li> </ul>	video/x-raw <ul style="list-style-type: none"> <li>width: [ 1, 2147483647 ]</li> <li>height: [ 1, 2147483647 ]</li> <li>framerate: [ 0/1, 2147483647/1 ]</li> </ul>
source (output) <ul style="list-style-type: none"> <li>■ Presence – always</li> <li>■ Direction – src</li> </ul>	video/x-h264 <ul style="list-style-type: none"> <li>width: [ 16, 4096 ]</li> <li>height: [ 16, 4096 ]</li> </ul>

**Table B-7 omxh264enc properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: "omxh264enc-omxh264enc0"</li> </ul>
qos	Handles quality-of-service events from downstream. <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
control-rate	The bitrate control method. This property can be updated only in NULL or READY state. <ul style="list-style-type: none"> <li>■ type: Enum "GstOMXVideoEncControlRate"</li> <li>■ access: read/write</li> <li>■ default: -1, "default"</li> <li>■ possible values: <ul style="list-style-type: none"> <li><input type="checkbox"/> (0): disable – Disable</li> <li><input type="checkbox"/> (1): variable – Variable</li> <li><input type="checkbox"/> (2): constant – Constant</li> <li><input type="checkbox"/> (3): variable-skip-frames – Variable Skip Frames</li> <li><input type="checkbox"/> (4): constant-skip-frames – Constant Skip Frames</li> <li><input type="checkbox"/> (-1): default – Component Default</li> </ul> </li> </ul>
target-bitrate	The target bitrate in bits per second. This property can be updated in all states. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 4294967295</li> <li>■ range: 0 - 4294967295</li> </ul>
quant-i-frames	The quantization parameter for I-frames. This property can be updated only in NULL or READY state. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> </ul>

**Table B-7 omxh264enc properties (cont.)**

Property	Description
	<ul style="list-style-type: none"> <li>▪ default: 4294967295</li> <li>▪ range: 0 - 4294967295</li> </ul>
quant-p-frames	<p>The quantization parameter for P-frames. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 4294967295</li> <li>▪ range: 0 - 4294967295</li> </ul>
quant-b-frames	<p>The quantization parameter for B-frames. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 4294967295</li> <li>▪ range: 0 - 4294967295</li> </ul>
periodicity-idr	<p>The periodicity of IDR frames. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 4294967295</li> <li>▪ range: 0 - 4294967295</li> </ul>
interval-intraframes	<p>The interval of coding Intra frames. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 4294967295</li> <li>▪ range: 0 - 4294967295</li> </ul>
b-frames	<p>The number of B-frames between two consecutive I-frames. This property can be updated only in NULL or READY state. In current code, value 1 is only supported to enable b-frames.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 4294967295</li> <li>▪ range: 0 - 4294967295</li> </ul>
entropy-mode	<p>The Entropy mode for encoding process. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>▪ type: Enum "GstOMXH264EncEntropyMode"</li> <li>▪ access: read/write</li> <li>▪ default: -1, "default"</li> <li>▪ possible values: <ul style="list-style-type: none"> <li>▫ (0): CAVLC – CAVLC entropy mode</li> <li>▫ (1): CABAC – CABAC entropy mode</li> <li>▫ (-1): default – Component Default</li> </ul> </li> </ul>

**Table B-7 omxh264enc properties (cont.)**

Property	Description
constrained-intra-prediction	If enabled, the prediction only uses residual data and decoded samples from neighboring coding blocks, which are coded using the Intra-Prediction modes. This property can be updated only in NULL or READY state. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
loop-filter-mode	Enables or disables the deblocking filter. This property can be updated only in NULL or READY state. <ul style="list-style-type: none"> <li>▪ type: Enum "GstOMXH264EncLoopFilter"</li> <li>▪ access: read/write</li> <li>▪ default: -1, "default"</li> <li>▪ possible values: <ul style="list-style-type: none"> <li>▫ (0): enable – Enable deblocking filter</li> <li>▫ (1): disable – Disable deblocking filter</li> <li>▫ (2): disable-slice-boundary – Disable deblocking filter on slice boundary</li> <li>▫ (-1): default – Component Default</li> </ul> </li> </ul>

## B.4 waylandsink

The waylandsink element is a video sink element that uses the Wayland Weston compositor implementation. It maps the Weston client APIs and states to the appropriate GStreamer APIs and states. It is based on the GStreamer GstVideoSink base class for video sinks.

For more information on the methods and call flow of GstVideoSink, see [GstVideoSink in Installing GStreamer Manual](#).

QTI has extended the waylandsink implementation and added support for window positioning and proprietary GBM-based buffer backend for zero-copy.

For information on pad templates and properties supported by the upstream waylandsink element, see [waylandsink in Installing GStreamer Manual](#)

**Table B-8 waylandsink pad templates**

Pad name	Template
sink (input)	<ul style="list-style-type: none"> <li>▪ Presence – always</li> <li>▪ Direction – sink</li> </ul> <p>video/x-raw(memory:GBM)  format: { (string)BGRx, (string)BGRA, (string)RGBx, (string)xBGR,  (string)xRGB, (string)RGBA, (string)ABGR, (string)ARGB,  (string)RGB, (string)BGR, (string)RGB16, (string)BGR16,  (string)YUY2, (string)YVYU, (string)UYVY, (string)AYUV,  (string)NV12, (string)NV21, (string)NV16, (string)YUV9,  (string)YVU9, (string)Y41B, (string)I420, (string)YV12,  (string)Y42B, (string)v308 }  width: [ 1, 2147483647 ]  height: [ 1, 2147483647 ]  framerate: [ 0/1, 2147483647/1 ]</p>

**Table B-9 waylandsink properties**

Property	Description
xdg-shell	Enables XDG shell protocol for the display. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: true</li> </ul>
x	X position for the content. <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0</li> <li>▪ range: 0 - 1920</li> </ul>
y	Y position for the content. <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0</li> <li>▪ range: 0 - 1080</li> </ul>
width	The destination width for the content. <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 640</li> <li>▪ range: 0 - 1920</li> </ul>
height	The destination height for the content. <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 640</li> <li>▪ range: 0 - 1920</li> </ul>

## B.5 qtivtransform

This GStreamer plugin has the capability to resize, flip, rotate, and color covert incoming the YUV or RGB video frames.

The main class of the plugin is called GstVideoTransform, which is responsible for the following:

- Facilitating capability negotiations between GstVideoTransform plugin and any other plugin connected to it
- Allocating output buffers that contain the transformed video frame

The C2D library, which is exposed by Adreno, is used for video frame transformation.

This library is wrapped inside the GstC2dVideoConverter class and few APIs are defined to create, configure, and process the incoming and outgoing buffers.

The output buffers are allocated by a buffer pool class called GstImageBufferPool. This pool can allocate either GBM or ION buffers depending on the negotiated capabilities between the plugins. The

GBM allocation is done via the libgbm, and ION allocation is done using the IOCTL commands to the kernel.

1. When a frame is received into the main class it will try to get a free output buffer from the pool and if the free buffer is unavailable, a buffer will be allocated.
2. Both the input and output buffers are passed to the GstC2dVideoConverter for processing.
3. After processing, the input buffer is dereferenced while the output buffer is sent to the next plugin in the pipeline.

**Table B-10 qtivtransform pad templates**

Pad name	Template
sink (input)	<pre> video/x-raw format: { (string)BGRA, (string)RGBA, (string)BGR, (string)RGB, (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] video/x-raw(ANY) format: { (string)BGRA, (string)RGBA, (string)BGR, (string)RGB, (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] </pre>
source (output)	<pre> video/x-raw format: { (string)BGRA, (string)RGBA, (string)BGR, (string)RGB, (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] video/x-raw(ANY) format: { (string)BGRA, (string)RGBA, (string)BGR, (string)RGB, (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] </pre>

**Table B-11 qtivtransform properties**

Property	Description
name	<p>The name of the object.</p> <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: "qtivtransform0"</li> </ul>
qos	<p>Handles the quality-of-service events.</p> <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>

**Table B-11 qtvtransform properties (cont.)**

Property	Description
flip-horizontal	Flips the video image horizontally. <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
flip-vertical	Flips the video image vertically. <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
rotate	Rotates the video images. <ul style="list-style-type: none"> <li>■ type: Enum "GstVideoTransformRotate"</li> <li>■ access: read/write <ul style="list-style-type: none"> <li>□ (0): none – No rotation</li> <li>□ (1):90 CW – Rotate 90 degrees clockwise</li> <li>□ (2): 90 CCW – Rotate 90 degrees counter-clockwise</li> <li>□ (3):180 – Rotate 180 degrees</li> </ul> </li> <li>■ default: 0, "none"</li> </ul>
crop-x	Sets the Pixels to crop starting from x-axis coordinate. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 0</li> </ul>
crop-y	Sets the Pixels to crop starting from y-axis coordinate. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 0</li> </ul>
crop-width	Sets the width of the crop rectangle. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 0</li> </ul>
crop-height	Sets the height of the crop rectangle. <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 0</li> </ul>

## B.6 qtioverlay

The qtioverlay gstreamer element is a hardware accelerated in-place image draw and blit element for drawing overlays on top of YUV images. The overlays can be configured through properties and also through buffer metadata attached to the input buffers.

It supports blitting of static images, bounding boxes, custom user text, data/time overlays, privacy mask through properties. It also supports blitting of detection (bounding box), segmentation (semantic

mask/mask image), classification (label/user text), and pose graph overlay through metadata attached to the input buffer.

**Table B-12 qtioverlay pad templates**

Pad name	Template
sink (input)	<pre> video/x-raw format: { (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] video/x-raw(ANY) format: { (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] </pre>
source (output)	<pre> video/x-raw format: { (string)BGRA, (string)RGBA, (string)BGR, (string)RGB, (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] video/x-raw(ANY) format: { (string)BGRA, (string)RGBA, (string)BGR, (string)RGB, (string)NV12, (string)NV21 } width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 2147483647/1 ] </pre>

**Table B-13 qtioverlay properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: "qtioverlay0"</li> </ul>
qos	Handles the quality-of-service events. <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: true</li> </ul>
overlay-text	The user text overlay. If this property is set, the user text is enabled and user text value from property is displayed. <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state</li> <li>▪ access: read/write</li> <li>▪ default: None</li> </ul>

**Table B-13 qtioverlay properties (cont.)**

Property	Description
	<p>The value format is "&lt;name&gt;, text=\"&lt;user text&gt;\", color=(unit) 0xRRGGBAA, dest-rect=&lt;x, y, w, h&gt;;"</p> <p>For example: "text0, text=\"Qualcomm\ Intelligence\", color=(uint) 0xFFFF00FF, dest-rect=&lt;160, 624, 944, 50&gt;;"</p> <p>The example format creates a user text overlay with name text0 with text "Qualcomm Intelligence" in yellow color (0xFFFF00FF) at position 160×624 (xxy), and dimensions 944×50 (wxh).</p>
overlay-date	<p>The overlay date.</p> <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul> <p>The value format is "&lt;name&gt;, date-format=\"&lt;date format&gt;\", time-format=\"&lt;time format&gt;\", color=(uint) 0xRRGGBAA, dest-rect=&lt;x, y, w, h&gt;;"</p> <p>Supported date formats</p> <ul style="list-style-type: none"> <li>▪ "MMDDYYYY"</li> <li>▪ "YYYYMMDD"</li> </ul> <p>Supported time formats</p> <ul style="list-style-type: none"> <li>▪ "HHMMSS_24HR"</li> <li>▪ "HHMMSS_AMPM"</li> <li>▪ "HHMM_24HR"</li> <li>▪ "HHMM_AMPM"</li> </ul> <p>For example: "date0, date-format=\"MMDDYYYY\", time-format=\"HHMMSS_24HR\", color=(uint) 0xFF00FFFF, dest-rect=&lt;0, 0, 256, 80&gt;;"</p> <p>The example format generates a date time overlay with name date0 in the date format "MMDDYYYY", time format "HHMMSS_24HR", and purple color (0xFF00FFFF) at position 0x0 (xxy) with dimensions 256×80 (wxh).</p>
overlay-simg	<p>The static image overlay. If this property is set, the static image is enabled and static image from the path specified is rendered on video stream.</p> <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state</li> <li>▪ access: read/write</li> <li>▪ default: None</li> </ul> <p>The value format is "&lt;name&gt;, image=\"&lt;rgba image full path with filename&gt;\", resolution=&lt;w, h&gt;, dest-rect=&lt;x, y, w, h&gt;;"</p> <p>For example: "image0, image=\"/data/misc/qmmf/overlay_test_464_109.rgba\", resolution=&lt;464, 109&gt;, dest-rect=&lt;1300, 800, 464, 109&gt;;"</p> <p>The example format will blit the static image at /data/misc/qmmf/overlay_test_464_109.rgba with resolution of 464×109 (wxh) at position 1300×800 (xxy), and dimension 464×109 (wxh).</p>

**Table B-13 qtioverlay properties (cont.)**

Property	Description
overlay-bbox	<p>The bounding box overlay. If this property is set, the bounding box is enabled and bounding box of the specified shape is rendered along with the label on the video stream.</p> <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state</li> <li>▪ access: read/write</li> <li>▪ default: None</li> </ul> <p>The value format is "&lt;name&gt;, bbox=&lt;x, y, w, h&gt;, label=&lt;text label&gt;, color=(uint)0xRRGGBAA;"</p> <p>For example:"bbox0, bbox=&lt;140, 130, 100, 100&gt;, label="Pet", color=(uint)0x0000FFFF;"</p> <p>The example creates a bouding box with name bbox0 at position 140×130 (xxy), of dimension 100×100(w×h), with label "Pet", and in blue color (0x0000FFFF)</p>
overlay-mask	<p>The privacy mask overlay. If this property is set, the privacy mask is enabled and privacy mask of the specified shape is rendered on the video stream.</p> <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state</li> <li>▪ access: read/write</li> <li>▪ default: None</li> </ul> <p>The value format is "&lt;name&gt;, circle=&lt;x, y, r&gt;, dest-rect=&lt;x, y, w, h&gt;, color=0xRRGGBAA"</p> <p>For example:"mask0, circle=&lt;960, 540, 400&gt;, dest-rect=&lt;0, 0, 1920, 1080&gt;, color=0x202020FF"</p> <p>The example creates a circular mask with name mask0 at location 960×540 (xxy) of radius 400(r) on top of frame with dimensions 1920×1080(w×h) and of color greyish black (0x202020FF).</p> <p>There is a option to create a inverse mask, where the only the area defined as circle is not masked and rest of the area described in the dest-rect is masked out with the color specified. To create a inverse mask, the user needs to add inverse = true after the circle definition in the value string.</p>
bbox-color	<p>The bounding box overlay color. The color format is RGBA.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0xCCFF</li> </ul>
date-color	<p>The date overlay color. The color format is RGBA.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0xFF0000FF</li> </ul>
text-color	<p>The text overlay color. The color format is RGBA.</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0xFFFF00FF</li> </ul>

**Table B-13 qtioverlay properties (cont.)**

Property	Description
pose-color	The pose overlay color. The color format is RGBA. <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0x33CC00FF</li> </ul>
dest-rect-ml-text	<ul style="list-style-type: none"> <li>▪ The destination rectangle params for ML detection overlay.</li> <li>▪ The Start-X, Start-Y, Width, and Height of the destination rectangle format is &lt;X, Y, WIDTH, HEIGHT&gt;.</li> </ul> <p>type: GstValueArray of GValues of type "gint" access: read/write</p>

## B.7 qtimlesnpe

The qtimlesnpe element exposes the Qualcomm Neural Processing SDK capabilities to GStreamer. It can load and execute Qualcomm Neural Processing SDK models. It supports the following functionalities:

- Preprocessing supports downscale, color convert, mean subtraction and padding
- Postprocessing supports the most popular model types as classification, detection, and segmentation. The postprocessing result is attached as MLMeta to GST buffer.

The qtimlesnpe element supports the following configurations:

- Config file in JSON format
- GST properties

The parameters can be set using either of the configurations. However, GST properties has higher priority.

**Table B-14 qtimlesnpe pad templates**

Pad name	Template
sink (input)	<pre> video/x-raw format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ]  height: [ 1, 2147483647 ]  framerate: [ 0/1, 2147483647/1 ]  video/x-raw (ANY)  format: { (string)NV12, (string)NV21 }  width: [ 1, 2147483647 ]  height: [ 1, 32767 ]  framerate: [ 0/1, 2147483647/1 ] </pre>
source (output)	<pre> video/x-raw format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ]  height: [ 1, 2147483647 ]  framerate: [ 0/1, 2147483647/1 ]  video/x-raw (ANY)  format: { (string)NV12, (string)NV21 }  width: [ 1, 2147483647 ]  height: [ 1, 2147483647 ]  framerate: [ 0/1, 2147483647/1 ] </pre>

**Table B-15 qtimlesnpe properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: "qtimlesnpe0"</li> </ul>
qos	Handles the quality-of-service events. <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
config	Provides the path to JSON configuration file. For example: /data/misc/camera/mle_snpe_config.json <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> </ul>
model	The model file name. The default location is /data/misc/camera. The filename extension should be .dlc. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> </ul>
labels	The name of the labels file. The default location is /data/misc/camera. The filename extension should be .txt. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> </ul>
input-format	Sets the inference input format. <ul style="list-style-type: none"> <li>■ Supported formats:<ul style="list-style-type: none"> <li>□ 0 – RGB</li> <li>□ 1 – BGR</li> <li>□ 2 – RGBFloat</li> <li>□ 3 – BGRAFloat</li> </ul></li> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 3</li> </ul>
postprocessing	The supported postprocessing. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> <li>■ supported values: "classification", "detection", "singlessd", "segmentation"</li> </ul>
mean	The channel mean subtraction values ('<B, G, R>'). For example, mean=<128.0,128.0,128.0> <ul style="list-style-type: none"> <li>■ type: GstValueArray of GValues of type "gdouble" access:</li> <li>■ read/write</li> <li>■ default: 128.0</li> </ul>

**Table B-15 qtimlesnpe properties (cont.)**

Property	Description
sigma	The channel divisor values ('<B, G, R>'). For example, sigma="<128.0,128.0,128.0>". <ul style="list-style-type: none"> <li>▪ type: GstValueArray of GValues of type "gdouble"</li> <li>▪ access: read/write</li> <li>▪ default: 128.0</li> </ul>
runtime	<ul style="list-style-type: none"> <li>▪ The Qualcomm Neural Processing SDK runtime.</li> <li>  ▫ 0 – CPU</li> <li>  ▫ 1 – DSP</li> </ul> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 1</li> </ul>
output-layers	The model output layers separated by a comma. It should be set if the model has more than one output. <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: None</li> </ul>
preprocess-type	Controls the pre-processing aspect ratio (AR) maintenance. <ul style="list-style-type: none"> <li>▪ Possible values: 0-kKeepARCrop, 1-kKeepARPAd, 2-kDirectDownscale.</li> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 1</li> </ul>
conf-threshold	The confidence threshold value. <ul style="list-style-type: none"> <li>▪ type: Float</li> <li>▪ access: read/write</li> <li>▪ default: 0.5</li> </ul>

## B.8 qtimletflite

The qtimletflite element exposes the TensorFlow Lite (TFLite) capabilities to GStreamer. It can load and execute the TFLite models.

It supports both preprocessing and postprocessing functionalities.

- The preprocessing supports downscale, color convert, mean subtraction and padding.
- The postprocessing supports the most popular model types as classification, detection, and segmentation. The postprocessing result is attached as MLMeta to GST buffer.

The qtimletflite element supports the following configurations:

- Config file in JSON format
- GST properties

The parameters can be set using either of the configurations. However, GST properties has higher priority.

**Table B-16 qtimletflite pad templates**

Pad name	Template
sink (input)	<p>video/x-raw</p> <ul style="list-style-type: none"> <li>■ Presence – always</li> <li>■ Direction – sink</li> </ul> <pre> format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ]  height: [ 1, 2147483647 ]  framerate: [ 0/1, 2147483647/1 ] </pre> <p>video/x-raw (ANY)</p> <pre> format: { (string)NV12, (string)NV21 }  width: [ 1, 2147483647 ]  height: [ 1, 32767 ]  framerate: [ 0/1, 2147483647/1 ] </pre>
source (output)	<p>video/x-raw</p> <ul style="list-style-type: none"> <li>■ Presence – always</li> <li>■ Direction – src</li> </ul> <pre> format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ]  height: [ 1, 2147483647 ]  framerate: [ 0/1, 2147483647/1 ] </pre> <p>video/x-raw (ANY)</p> <pre> format: { (string)NV12, (string)NV21 }  width: [ 1, 2147483647 ]  height: [ 1, 32767 ]  framerate: [ 0/1, 2147483647/1 ] </pre>

**Table B-17 qtimletflite properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: " qtimletflite0"</li> </ul>
qos	Handles the quality-of-service events. <ul style="list-style-type: none"> <li>■ type: Boolean</li> <li>■ access: read/write</li> <li>■ default: false</li> </ul>
config	Provides the path to JSON file. For example:/data/misc/camera/mle_tflite_config.json <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> </ul>
model	The name of model file. The default location is /data/misc/camera. The filename extension should be .tflite. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> </ul>
labels	The name of the labels file. The default location is /data/misc/camera. The filename extension should be .txt. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> </ul>
input-format	Sets the inference input format. <ul style="list-style-type: none"> <li>■ Supported formats:<ul style="list-style-type: none"> <li>□ 0 – RGB</li> <li>□ 1 – BGR</li> <li>□ 2 – RGBFloat</li> <li>□ 3 – BGRAFloat</li> </ul></li> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 3</li> </ul>
postprocessing	The supported postprocessing. <ul style="list-style-type: none"> <li>■ type: String</li> <li>■ access: read/write</li> <li>■ default: None</li> <li>■ supported values: "classification", "detection", "singlessd", "segmentation"</li> </ul>
mean	The channel mean subtraction values ('<B, G, R>'). For example, mean=<128.0,128.0,128.0> <ul style="list-style-type: none"> <li>■ type: GstValueArray of GValues of type "gdouble"</li> <li>■ access: read/write</li> <li>■ default: 128.0</li> </ul>

**Table B-17 qtimletflite properties (cont.)**

Property	Description
sigma	The channel divisor values ('<B, G, R>'). For example, sigma="<128.0,128.0,128.0>". <ul style="list-style-type: none"> <li>▪ type: GstValueArray of GValues of type "gdouble"</li> <li>▪ access: read/write</li> <li>▪ default: 128.0</li> </ul>
preprocess-type	Controls the pre-processing aspect ration (AR) maintenance. <ul style="list-style-type: none"> <li>▪ Possible values: 0-kKeepARCrop, 1-kKeepARPAd, 2-kDirectDownscale.</li> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 1</li> </ul>
conf-threshold	The confidence threshold value. <ul style="list-style-type: none"> <li>▪ type: Float</li> <li>▪ access: read/write</li> <li>▪ default: 0.5</li> </ul>
delegate	The supported TFLite delegates: "default" - use CPU; "dsp" - use DSP. <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: "default"</li> </ul>
num-threads	The number of threads that are to be used for inference. <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 2</li> </ul>

## B.9 qtivcomposer

The qtivcomposer GStreamer plugin takes one or more input video streams (sink pads) and depending on their dimensions and user-provided parameters, generates a composed output stream containing those mixed streams.

The main class of the plugin is `GstVideoComposer` and it is responsible for capability negotiations between this plugin and each of the requested sink pad. It will compare the input geometry, color format and frame rate to define the output parameters.

If no composition parameters have been explicitly set, the output video frames will have the geometry of the biggest incoming video stream and the frame rate of the fastest incoming video stream. If there is a discrepancy between the input streams and output stream video formats, a color conversion will be done.

For video frame transformation, the C2D library exposed by the Adreno is used. This library is wrapped inside the `GstC2dVideoConverter` class and few APIs are defined to create, configure and process the incoming and outgoing buffers.

The output buffers are allocated by a buffer pool class called `GstImageBufferPool`. This pool can allocate either GBM or ION buffers depending on the negotiated capabilities between the plugins. The GBM allocation is done via the `libgbm` and ION allocation is done through IOCTL commands to the kernel.

The individual parameters for each input stream can be configured on the video composer pads:

**Table B-18 qtivcomposer pad templates**

Pad name	Template
sink (input)	<pre> video/x-raw format: { (string)BGRA, (string)RGBA,           (string)BGR, (string)RGB, (string)NV12,           (string)NV21 }  width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 255/1 ]  video/x-raw(ANY) format: { (string)BGRA,           (string)RGBA, (string)BGR, (string)RGB,           (string)NV12, (string)NV21 }  width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 255/1 ] </pre>
source (output)	<pre> video/x-raw format: { (string)BGRA, (string)RGBA,           (string)BGR, (string)RGB, (string)NV12,           (string)NV21 }  width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 255/1 ]  video/x-raw(ANY) format: { (string)BGRA,           (string)RGBA, (string)BGR, (string)RGB,           (string)NV12, (string)NV21 }  width: [ 1, 32767 ] height: [ 1, 32767 ] framerate: [ 0/1, 255/1 ] </pre>

**Table B-19 qtivcomposer properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: "qtivcomposer0"</li> </ul>
parent	The parent of the object. <ul style="list-style-type: none"> <li>▪ type: GstObject</li> <li>▪ access: read/write</li> </ul>
latency	The additional latency in Live mode to allow upstream to take longer to produce buffers for the current position (in nanoseconds). <ul style="list-style-type: none"> <li>▪ type: 64 bit Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0</li> </ul>
start-time-selection	Decides the start time output. <ul style="list-style-type: none"> <li>▪ type: Enum GstAggregatorStartTimeSelection"</li> <li>▪ access: read/write</li> <li>□ (0): zero – Start at 0 running time(default)</li> <li>□ (1): first – Start at first observed input running time</li> <li>□ (2): set – Set start time with start-time property</li> <li>▪ default: 0, "zero"</li> </ul>
start-time	The start time to use if start-time-selection=set. <ul style="list-style-type: none"> <li>▪ type: 64 bit Integer</li> <li>▪ access: read/write</li> <li>▪ default: 18446744073709551615</li> <li>▪ range: 0 - 18446744073709551615</li> </ul>
background	The background color in hexadecimal format. <ul style="list-style-type: none"> <li>▪ type: 64 bit Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0xFFFFFFFF</li> <li>▪ range: 0 – 0xFF808080</li> </ul>

**Table B-20 qtivcomposer sink pad properties**

Property	Description
zorder	The Z-axis order, default will be order of creation. <ul style="list-style-type: none"> <li>▪ type: Integer</li> <li>▪ access: read/write</li> <li>▪ default: -1</li> <li>▪ range: -1 - 2147483647</li> </ul>
crop	The crop rectangle ('<X, Y, WIDTH, HEIGHT >'). <ul style="list-style-type: none"> <li>▪ type: GstValueArray of GValues of type "gint"</li> <li>▪ access: read/write</li> </ul>

**Table B-20 qtivcomposer sink pad properties (cont.)**

Property	Description
position	The X and Y coordinates of the destination rectangle top left corner ('<X, Y>'). <ul style="list-style-type: none"> <li>▪ type: GstValueArray of GValues of type "gint"</li> <li>▪ access: read/write</li> </ul>
dimensions	The destination rectangle width and height, if left as '0' they will be the same as input dimensions ('<WIDTH, HEIGHT>'). <ul style="list-style-type: none"> <li>▪ type: GstValueArray of GValues of type "gint"</li> <li>▪ access: read/write</li> </ul>
alpha	The alpha channel value. <ul style="list-style-type: none"> <li>▪ type: Double</li> <li>▪ access: read/write</li> <li>▪ default: 1</li> <li>▪ range: 0 - 1</li> </ul>
Flip-horizontal	Flip video horizontally <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
flip-vertical	Flip video vertically <ul style="list-style-type: none"> <li>▪ type: Boolean</li> <li>▪ access: read/write</li> <li>▪ default: false</li> </ul>
rotate	Rotate video <ul style="list-style-type: none"> <li>▪ type: Enum "GstVideoComposerRotate"</li> <li>▪ access: read/write <ul style="list-style-type: none"> <li>▫ (0): none – No rotation</li> <li>▫ (1): 90CW – Rotate 90 degrees clockwise</li> <li>▫ (2): 90CCW – Rotate 90 degrees counter-clockwise</li> <li>▫ (3): 180 – Rotate 180 degrees</li> </ul> </li> <li>▪ default: 0, "none"</li> </ul>

## B.10 omxaacenc

The omxaacenc GStreamer element is part of the gst-omx plugin. The gst-omx plugin is an adapter for the available OpenMAX IL components. It maps the OpenMAX APIs and states to the Gstreamer APIs and states.

The omxaacenc element is an OpenMAX AAC audio encoder that provides hardware accelerated AAC audio encoding using the OpenMAX component on platforms that support it.

QTI has added support for its OpenMAX audio encode component in the gst-omx plugin to enable hardware-accelerated audio encode on its audio core. The omxaacenc element is based on GstAudioEncoder base class for audio encoders.

For more information on the GstAudioEncoder methods and API call flow, see <https://gstreamer.freedesktop.org/documentation/audio/gstaudioencoder.html?gi-language=c>.

**Table B-21 omxaacenc pad templates**

Pad name	Template
sink (input) ■ Presence – always ■ Direction – sink	audio/x-raw rate: [ 1, 2147483647 ] channels: [ 1, 16 ] format: { (string)S8, (string)U8, (string)S16LE, (string)S16BE, (string)U16LE, (string)U16BE, (string)S24LE, (string)S24BE, (string)U24LE, (string)U24BE, (string)S32LE, (string)S32BE, (string)U32LE, (string)U32BE }
source (output) ■ Presence – always ■ Direction – src	audio/mpeg mpegversion: { (int)2, (int)4 } stream-format: { (string)raw, (string)adts, (string)adif, (string)loas, (string)latm }

**Table B-22 omxaacenc properties**

Property	Description
name	The name of the object. ■ type: String ■ access: read/write ■ default: "omxaacenc-omxaacenc0"
parent	The parent of the object. ■ type: GstObject ■ access: read/write
perfect-timestamp	Favour perfect timestamps over tracking upstream timestamps. ■ type: Boolean ■ access: read/write ■ default: false
mark-granule	Apply granule semantics to buffer metadata (implies perfect-timestamp) ■ type: Boolean ■ access: read ■ default: false
hard-resync	Perform clipping and sample flushing upon discontinuity. ■ type: Boolean ■ access: read/write ■ default: false
tolerance	Consider discontinuity if timestamp jitter/imperfection exceeds tolerance (nanosecond). ■ type: Signed Integer (64-bit) ■ access: read/write

**Table B-22 omxaacenc properties (cont.)**

Property	Description
	<ul style="list-style-type: none"> <li>■ default: 40000000</li> <li>■ range: 0 - 9223372036854775807</li> </ul>
bitrate	<p>The bitrate for audio stream. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>■ type: Unsigned Integer</li> <li>■ access: read/write</li> <li>■ default: 128000</li> <li>■ range: 0 - 4294967295</li> </ul>
aac-tools	<p>The allowed AAC tools. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>■ type: Flags "GstOMXAACTools"</li> <li>■ access: read/write</li> <li>■ default: 0x0000001f, "ltp+pns+tns+is+ms"</li> <li>■ possible-values: <ul style="list-style-type: none"> <li>□ (0x00000001): ms – Mid/side joint coding</li> <li>□ (0x00000002): is – Intensity stereo</li> <li>□ (0x00000004): tns – Temporal noise shaping</li> <li>□ (0x00000008): pns – Perceptual noise substitution</li> <li>□ (0x00000010): ltp – Long term prediction</li> </ul> </li> </ul>
aac-error-resilience-tools	<p>The allowed AAC error resilience tools. This property can be updated only in NULL or READY state.</p> <ul style="list-style-type: none"> <li>■ type: Flags "GstOMXAACERTools"</li> <li>■ access: read/write</li> <li>■ default: 0x00000000, "(none)"</li> <li>■ possible-values: <ul style="list-style-type: none"> <li>□ (0x00000001): vcb11 – Virtual code books</li> <li>□ (0x00000002): rvlc – Reversible variable length coding</li> <li>□ (0x00000004): hcr – Huffman codeword reordering</li> </ul> </li> </ul>

## B.11 qtihexagonnn

The qtihexagonnn GStreamer plugin enables running of end-to-end ML tasks on QTI Silicon-powered edge compute platform. It loads and executes neural network models directly on the DSP hexagon neural network accelerator. It supports MobileNet SSD (v1), Semantic Segmentation (DeepLab v3), and PoseNet (posenet\_mobilenet\_v1\_075) models out of the box.

The model can be selected using the model-name property of the plugin. For Segmentation and PoseNet models, no other configuration is required whereas for MobileNet SSD, the user needs to

push the label file on the device, and specify **label-file** and **data-folder** (If the label file is not pushed at the default location) properties.

**Table B-23 qtihexagonnn pad templates**

Pad name	Template
sink (input) Presence – always Direction – sink	video/x-raw format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ] height: [ 1, 2147483647 ] framerate: [ 0/1, 2147483647/1 ] video/x-raw(ANY) format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ] height: [ 1, 2147483647 ] framerate: [ 0/1, 2147483647/1 ]
source (output) Presence – always Direction – src	video/x-raw format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ] height: [ 1, 2147483647 ] framerate: [ 0/1, 2147483647/1 ] video/x-raw(ANY) format: { (string)NV12, (string)NV21 } width: [ 1, 2147483647 ] height: [ 1, 2147483647 ] framerate: [ 0/1, 2147483647/1 ]

**Table B-24 qtihexagonnn properties**

Property	Description
name	The name of the object. <ul style="list-style-type: none"><li>▪ type: String</li><li>▪ access: read/write</li><li>▪ default: "hexagonnn0"</li></ul>
parent	The parent of the object. <ul style="list-style-type: none"><li>▪ type: GstObject</li><li>▪ access: read/write</li></ul>
model-name	Specifies the model type to execute. Currently supported models are: <ul style="list-style-type: none"><li>▪ "segmentation": semantic segmentation through deeplab v3 model</li><li>▪ "posenet": pose estimation using posenet_mobilenet_v1_075 model</li><li>▪ "mnetssd": object detection using mobilenet SSD v1 model</li><li>▪ type: String</li><li>▪ access: read/write</li><li>▪ default: segmentation</li></ul>
execution-mode	Specifies the operating mode of execution.

**Table B-24 qtihexagonnn properties (cont.)**

Property	Description
	<p>Currently, the supported modes are:</p> <ul style="list-style-type: none"> <li>▪ "default" : no frames would be skipped</li> <li>▪ "auto" : number of frames skipped is decided by the plugin</li> <li>▪ "frame_skip_count" : numbers of frames to be skipped is specified in skip-count property.</li> </ul> <p>▪ type: String      ▪ access: read/write      ▪ default: default</p>
skip-count	<p>The number of frames to be skipped between two processed frames when the execution-mode property is set to "frame_skip_count"</p> <ul style="list-style-type: none"> <li>▪ type: Unsigned Integer</li> <li>▪ access: read/write</li> <li>▪ default: 0</li> <li>▪ range: 0 - 30</li> </ul>
data-folder	<p>Specifies the folder where the label files are stored. This location should preferably be in the /data/ folder.</p> <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: /data/misc/camera</li> </ul>
label-file	<p>Specifies the name of the label file to be used from the path specified in data-folder property.</p> <p>This required when the model-name property is set to "mnetssd".</p> <ul style="list-style-type: none"> <li>▪ type: String</li> <li>▪ access: read/write</li> <li>▪ default: default</li> </ul>

# C QTI GStreamer reference applications

This appendix provides details on QTI-authored GStreamer reference applications. For more information on GStreamer applications, see [GStreamer Application Development Manual](#).

## C.1 QTI gst-pipeline-app

The QTI gst-pipeline-app is a helper tool does the following:

- Exposes the same capability as GStreamer gst-launch-1.0 tool.
- Provides an option to set runtime properties for any GST element in the pipeline that allows property change in PLAYING state.

The syntax is `gst-pipeline-app [OPTION?] DESCRIPTION`.

- [OPTION?]: The supported options are `-e` or `--eos-on-exit`, which sends EOS event before shutting the pipeline down.
- DESCRIPTION is a list of elements separated by exclamation marks (!). The properties can be appended to elements in the form `property = value`.

When a pipeline is run, the `gst-pipeline-app` lists all the elements in the pipeline. If element name was specified, then the element appears with that name. Else, the element is displayed with the default name.

1. Run the video encode pipeline to list all the elements.

```
gst-pipeline-app -e -e qtiqmmfsrc ! video/x-raw\ (memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1,camera=0 !
omxh264enc ! h264parse ! mp4mux ! queue ! filesink location=/data/mux.mp4
```

2. Set the pipeline in **PLAYING** state.

```
#####
##### MENU #####
=====
===== Pipeline Controls =====
(0) NULL : Set the pipeline into NULL state
(1) READY : Set the pipeline into READY state
(2) PAUSED : Set the pipeline into PAUSED state
(3) PLAYING : Set the pipeline into PLAYING state
=====
===== Other =====
(p) Plugin Mode : Choose a plugin which to control
(q) Quit : Exit the application

Choose an option: 3
```

3. Choose **Plugin Mode** to see the plugins whose properties can be modified.

```
#####
# MENU #####
=====
==== Pipeline Controls =====
(0) NULL : Set the pipeline into NULL state
(1) READY : Set the pipeline into READY state
(2) PAUSED : Set the pipeline into PAUSED state
(3) PLAYING : Set the pipeline into PLAYING state
=====
==== Other =====
(p) Plugin Mode : Choose a plugin which to control
(q) Quit : Exit the application

Choose an option: p
```

4. Select the element for which the property should be modified.

All the properties that can be updated are listed with description. For the mentioned example, if "omxh264enc-omxh264enc0" is selected, the options shown in the figure are listed:

```
-----
( 1) filesink0
( 2) queue0
( 3) mp4mux0
( 4) h264parse0
( 5) omxh264enc -omxh264enc0
( 6) capsfilter0
( 7) qmmfsrc0
-----
Enter plugin name or its index (or press Enter to return): 5
```

List of properties that can be updated dynamically are displayed.

- To choose the property, type the option number and enter the desired value.
- To check if the value is set, select the property again and check the current value field.

```
#####
# MENU #####
=====
==== Plugin Properties =====
( 0) target-bitrate : Target bitrate in bits per second (0xffffffff=component default)
----- sink Pad -----
----- src Pad -----
=====
==== Plugin Signals =====
=====
==== Other =====
(b) Back : Return to the previous menu

Choose an option: 0
-----
Current value: 4294967295, Range: 0 - 4294967295
-----
Enter value (or press Enter to keep current one): 6000000

#####
# MENU #####
=====
==== Plugin Properties =====
( 0) target-bitrate : Target bitrate in bits per second (0xffffffff=component default)
----- sink Pad -----
----- src Pad -----
=====
==== Plugin Signals =====
=====
==== Other =====
(b) Back : Return to the previous menu

Choose an option: 0
-----
Current value: 6000000, Range: 0 - 4294967295
```

For more information on the usage of `gst-pipeline-app`, see [Dynamic parameters update using the `gst-pipeline-app`](#) and [Snapshot](#).

Qualcomm  
Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com

# D References

## D.1 Related documents

Title	Number
<b>Qualcomm Technologies, Inc.</b>	
<i>QCS610 Linux Platform Development Kit Quick Start Guide</i>	80-PL631-300
<i>QCS610 Linux Platform Development Kit Software Programmer's Guide</i>	80-PL631-200
<i>QCS610 Release Notes</i>	RNO-200630134155
<i>QCS610 Secure Boot Enablement User Guide</i>	80-PL631-42
<i>QCS610 QFPROM Programming Reference Guide</i>	80-PL052-97
<i>Fluence Pro V2 Overview</i>	80-NK880-7
<i>UEFI PMIC Software User Guide</i>	80-P2484-42
<i>XBL Configuration Guide</i>	80-PE663-1
<i>Thermal Core Overview</i>	80-P9301-113
<i>Battery Current Limit (BCL) Overview and Tuning</i>	80-NM328-709
<i>WCN39xx Connectivity (WLAN/Bluetooth) Software Reference Manual</i>	80-WL050-1
<i>WCN39xx Connectivity (WLAN/Bluetooth) Software Programmer's Guide</i>	80-WL050-2
<i>Sensors Execution Environment (SEE) Sensors Deep Dive</i>	80-P9301-35
<i>Adding a Custom Sensors Algorithm with Sensors Execution Environment (SEE)</i>	80-P9301-67
<i>QDSS Debug User Guide</i>	80-NF515-8
<i>PD Restart Overview</i>	80-P9301-114
<i>Data Capture and Compare User Guide</i>	80-P1727-1
<i>Sectools: Debug Policy Tool User Guide</i>	80-NM248-6
<i>QCAP Start-up Guide</i>	80-NR964-54
<i>Qualcomm® DDR USB Test Tool (QDUTT) Tool</i>	80-P2163-1
<i>Memory Verification Guidelines for OEM</i>	80-P7202-4
<i>QMVS User Guide</i>	80-P3255-38
<i>Qualcomm Open Lab Validation Test Suite</i>	80-P7820-1
<i>QMESA: Qualcomm Memory Stress Application User Guide</i>	80-NH759-1
<i>HLOS PMIC PON Software User Guide</i>	80-P2484-40
<i>Resource Power Manager (RPM.BF) Debug Manual</i>	80-P9301-16
<i>Qualcomm Spectra™ 2xx Camera ISP Overview</i>	80-P9301-14

Title	Number
<i>Qualcomm Spectra 2XX Deep Dive</i>	80-P9301-60
<i>Staggered HDR Overview and Tuning Guide</i>	80-PF105-40
<i>360 Camera Calibration</i>	80-P9894-1
<i>IOT Lens Distortion Correction</i>	80-PN984-7
<i>Qualcomm Hexagon HVX Streaming Customization for Camera</i>	80-NF772-36
<i>Qualcomm Trusted Execution Environment (QTEE) Version 5.0 User Guide</i>	80-NH537-4
<b>Standards</b>	
<i>Arm System Memory Management Unit Architecture Specification</i>	IHI0062D
Resources	
<b>CreatePoint</b>	
<a href="http://cap.qti.qualcomm.com/">http://cap.qti.qualcomm.com/</a>	
<a href="http://www.tianocore.org/edk2/">http://www.tianocore.org/edk2/</a>	
<b>Yocto Project Reference Manual</b>	
<a href="https://gstreamer.freedesktop.org/documentation">https://gstreamer.freedesktop.org/documentation</a>	
<a href="https://gitlab.com/cryptsetup/cryptsetup/wikis/DMVerity">https://gitlab.com/cryptsetup/cryptsetup/wikis/DMVerity</a>	
<a href="https://www.redhat.com/en/topics/linux/what-is-selinux">https://www.redhat.com/en/topics/linux/what-is-selinux</a>	
<a href="https://www.docker.com/">https://www.docker.com/</a>	

## D.2 Acronyms and terms

Acronym or term	Definition
ADRC	Adaptive dynamic range compression
AES	Advanced encryption standard
ALSA	Advanced Linux sound architecture
AOP	Always-on processor
APDP	Application processor debug policy
APU	Address protection unit
AVC	Access vector cache
BTM	Boot thermal management
CBC	Cipher-block chaining
CBR	Constant bit rate
CC	Configuration channel
DDI	DDR debug image
DEVCFG	Device configuration
DMA	Direct memory access
DM-verify	Device mapper verify
DoU	Days-of-use
DRM	Direct rendering manager

Acronym or term	Definition
DRV	Direct resource voter
DSP	Digital signal processor
DTM	Dynamic thermal management
EDK	Embedded development kit
EE	Execution environment
EIS	Electronic image stabilization
ELF	Executable and linking format
FDE	Full disk encryption
FIFO	First In, First Out
FW	Firmware
GPCE	General purpose crypto engine
HLOS	High-level operating system
ICE	Inline crypto engine
IFE	Image front-end
IPE	Image processing engine
IrDA	Infrared data association
IR	Infrared
ISP	Image sensor processor
IV	Intermediate vector
iWarp	Image warping
KDF	Key derivation function
KGSL	Kernel graphics system layer
KMS	Kernel mode setting
LDC	Lens distortion correction
LEF	Long exposure frame
LEPDK	Linux Embedded Platform Development Kit
LTM	Local tone-mapping
MAC	Mandatory access control
MCTF	Motion compensated temporal filter
MFNR	Multiframe noise reduction
MIPI-CSI	Mobile industry processor interface camera serial interface
MPU	Memory protection unit
NFC	Near field communicator
NX	Non executable
OE	OpenEdge
OMX IL	OpenMAX integration layer
PBL	Primary boot loader
PD	Power delivery

Acronym or term	Definition
PDK	Platform development kit
PIC or PIE	Position independent executable or position independent code
PHY	Physical layer (PHY)
PIL	Peripheral image loader
PQ	Perceptual quality
QCAP	Qualcomm crash analysis portal
QDSS	Qualcomm Debug Subsystem
QDUTT	Qualcomm DDR USB test tool
QFPROM	Qualcomm fuse programmable read only memory
QGIC	Qualcomm generic interrupt controller
QMMF	Qualcomm multimedia framework
QMVS	Qualcomm memory validation suite
QUP	Qualcomm® Universal Peripheral
RELRO	Relocation read-only
RMA	Return material authorization
ROI	Region of interest
ROP	Return oriented programming
RPMh	Resource power management hardening
RPMB	Replay protected memory block
RPU	Register protection unit
RSC	Rolling shutter correction
RTSP	Real-time Streaming Protocol
SDI	System debug interface
sHDR	Staggered high dynamic range
SE	Serial engine
SEF	Short exposure frame
SE-Linux	Security-enhanced Linux
SFS	Secure file system
SIMO	Single input multiple outputs
SoC	System-on-chip
SPL	Sound pressure level
SSR	Subsystem restart
SSD	<ul style="list-style-type: none"> <li>■ Storage: Solid-state drive</li> <li>■ Camera – Object detection: Single Shot MultiBox Detector</li> </ul>
SWDT	Secure watchdog timer
TF	Temporal filtering
TZ	TrustZone
UDE	User-defined effect

Acronym or term	Definition
UEFI	Unified extensible firmware interface
VC	Virtual channel
VIP	Validated image programming
VBR	Variable bit rate
WD	Watchdog
XBL	eXtensible boot loader

Qualcomm Confidential - May Contain Trade Secrets  
2021-07-23 09:59:14 PDT  
indal.choi@lge.com