Thundercomm
AIOT ENABLER

Thundercomm TurboX SOM Camx Architecture Camera Sensor Driver

Bringup Guide

Rev. V1.1
Nov 02, 2023

DN: tc-A-161711

Revision History

Revision	Date	Description
1.0	Feb 28, 2022	Initial release.
1.1	Nov 02, 2023	• Update Chapter 1. Introduction. • Fix format problems.

Table List

About This Document

- This document is applicable to the following products:
  - TurboX CT4350
  - TurboX D845
  - TurboX C865
  - TurboX C865C
  - TurboX C4290/CM4290
  - TurboX C2290/CM2290
  - TurboX S605
  - TurboX C410/C610
  - TurboX C6490
  - TurboX C6490P
- Illustrations in this documentation might look different from your product.
- Depending on the model, some optional accessories, features, and software programs might not be available on your device.
- Depending on the version of operating systems and programs, some user interface instructions might not be applicable to your device.
- Documentation content is subject to change without notice. Thundercomm makes constant improvements on the documentation of the products, including this guidebook.
- Function declarations, function names, type declarations, attributes, and code samples appear in a different format, for example, `cp armcc armcpp`.
- Code variables appear in angle brackets, for example, `<number>`.
- Button, tool, and key names appear in bold font, for example, click **Save** or press **Enter**.
- Commands to be entered appear in a different font, for example,

```
adb devices
```

- Part of the code that does not contain instructions appear in a different format, for example,

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", MODE="0777", GROUP="adm"
```

- Folders and files are formatted in italic, for example, *turbox_flash_flat.sh*.

Table of Contents

Chapter 1. Introduction

This document provides guidelines for driver development and describes how to bring up the camx architecture camera sensor driver on theLA (Linux Android) platform and the LE (Linux Embedded) platform. The sensor driver configuration of the same architecture is common on different platforms, but there are some differences in the path of the file, which will be marked below. The document also describes the terms used in the sensor driver and provides examples of the sensor configuration parameters.

If your code contains the path *vendor/qcom/proprietary/chi-cdk/*, it means that your camera sensor is of camx architecture.

Supported SOM platforms:

CT4350, CM2290/C2290, D845, C610/C410, C865, S605, CM4290/C4290, CT6490/C6490/C6490P.

➲ **NOTE:** On the C6490P platform, due to modifications made to the camx code, the bringup method for GMSL Camera can be referenced from that for a regular Camera.

Chapter 2. Camera Sensor Driver Bringup

2.1. Sensor bringup and debug

This section describes how to bring up a sensor. Follow the workflow for sensor bringup and debug.

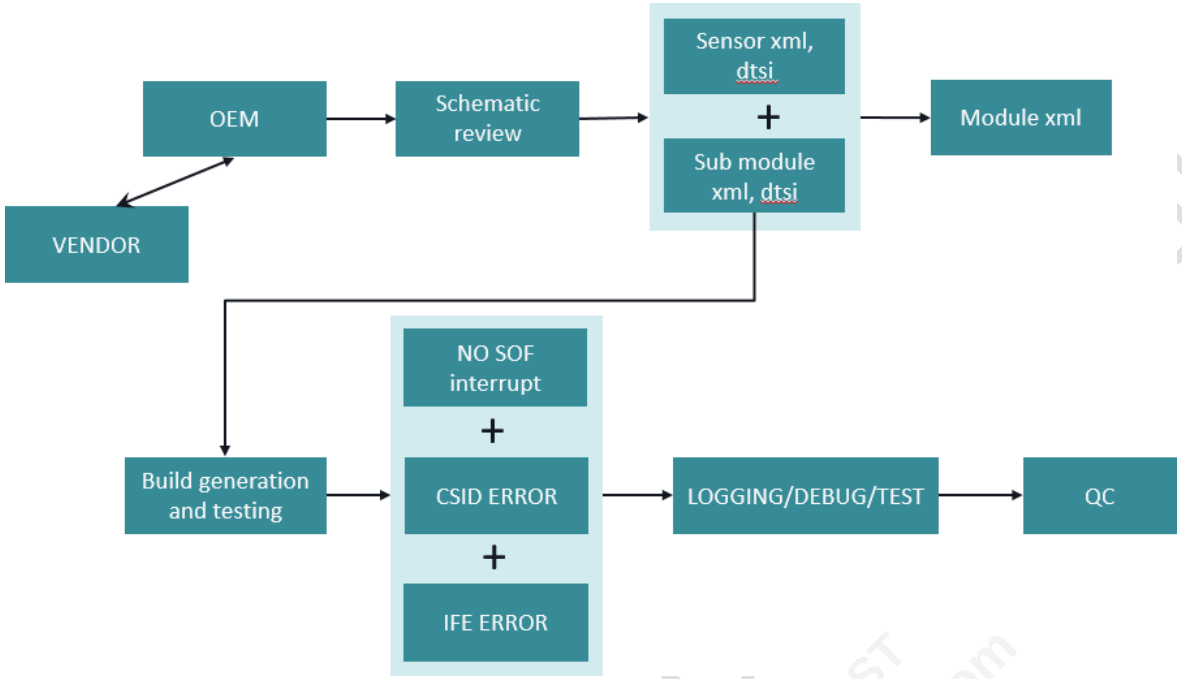

Figure 2-1. Sensor Bringup and Debug Workflow

Before software customization, review the schematic to understand the sensor module pins connection (for example, VDD_IO, CCI/CSI interface, etc.) as shown in Figure 2-2 below. Get the sensor power on/off timing sequence, sensor sub modules (OIS, actuator, etc.) integrated. Recommended MCLK: 19.2Mhz.

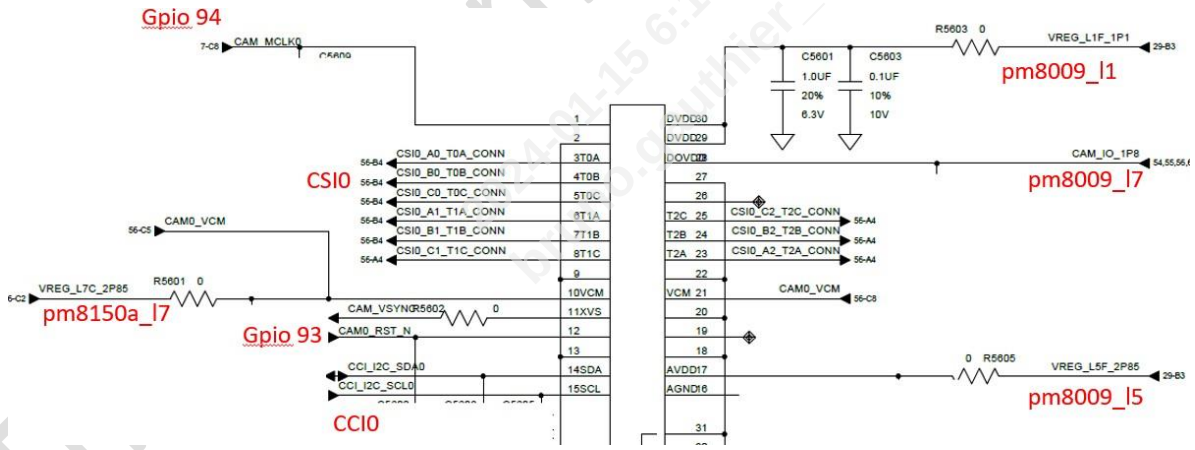

Figure 2-2. Pin Connection of Sensor Module

Follow the steps below to bring up the camera sensor driver:

Step 1. Locate the driver and module configuration XML files. Refer to file locations.

Step 2. Generate the binary files. Refer to Section 2.2.

Step 3. Compile the build.

File locations

The files at the following locations are modified during build compilation.

➲ **NOTE:** The file paths of different baselines are different, please refer to the baseline where you are located. The following paths are for reference only.

- Sensor driver XML files are at *proprietary/chi-cdk/oem/qcom/sensor/sensor_name/sensor_name_sensor.xml(for example, imx586/imx586_sensor.xml and imx586/imx586_pdaf.xml)*.
- Module configuration files are at *proprietary/chi-cdk/oem/qcom/module/module_name_module.xml* (for example, *semco_imx586_module.xml*).
- Kernel dts files are at *kernel/msm-4.19/arch/arm64/boot/dts/qcom/target_name-camera-sensor-platform.dtsi.*
- Some relatively new baselines kernel dtsi files are present in the following path: *proprietary/camera-devicetree/target_name-camera-sensor-platform.dtsi*.
- Submodule driver XML files are at *proprietary/chi-cdk/oem/qcom/sub-module_name/sub-module_name_sub-module.xml (for* example, *chi-cdk/oem/qcom/actuator/ak7374_actuator.xml*).
- The driver binary in the device vendor makefile to be included in the build is at *proprietary/chi-cdk/configs/product.mk* (for example, *PRODUCT_PACKAGES += com.qti.sensormodule.<sensor_name>.bin*).

2.2. Generate binary files

Prerequisite: Ensure that all register settings and power on/off sequences are aligned to data sheets for the bringup of the sensor/actuator and EEPROM is ready before driver creation.

Step 1. Run the fllowing script file in the root directory of souce code to build the debug version.

```
$ ./turbox_build.sh -a -l -v userdebug
```

The path of binary files varies with the platform.

- LA platform: *out/target/product/target_name/vendor/lib64/camera*.
- LE platform: taking CM2290 as an example, the sensor BIN files will be automatically generated under *tmp-glibc/work/qrbx210_rbx-oe-linux/qti-robotics-med-image/1.0-r0/rootfs/usr/lib/camera/*.

➲ **NOTE**: For some baselines, the bin file will not be automatically updated during compilation. To update it, proceed with the following steps.

1) Go to the *buildbins* directory.

```
$ cd {workspace}/vendor/qcom/proprietary/chi-cdk/tools/buildbins/
```

2) Execute the command to generate the bin file:

```
$ python buildbins.py --yaml-file-name buildbins_target_name.yaml
```

You can view more usage methods through `-help`.

```
$ python buildbins.py -help
usage: buildbins.py [-h] [--debug] [--verbose] [--force] [--check-deps-only]
            [--bin-path BIN_PATH]
            [--default-tuning-project DEFAULT_TUNING_PROJECT]
            [--target {all,code,bin}] [--clean]
            [--yaml-file-name YAML_FILE_NAME]
            [--gen-code-dir GEN_CODE_DIR]

Generate tuning binaries and their C++ parsers

optional arguments:
 -h, --help       show this help message and exit
 --debug         Show debugging output (you want --verbose?)
 --verbose        Show verbose output
 --force         Ignore timestamps & always force regeneration
 --check-deps-only   Do not generate anyting. For makefile dep gen
 --bin-path BIN_PATH  Comma separated list containing paths where compiled
             .bin files go
```

```
--default-tuning-project DEFAULT_TUNING_PROJECT
            A tuning poject folder, which will be used to create
            default tuning binary
--target {all,code,bin}
            Set the target will be generated by the script
--clean         Clean outputs
--yaml-file-name YAML_FILE_NAME
            yaml file name update for multiple yaml files usage
--gen-code-dir GEN_CODE_DIR
            Integration directory where the generated sources and
            mk files are outputed
```

The bin file will be generated in the *vendor/qcom/proprietary/chi-cdk/oem/qcom/bin/* path.

Step 2. Push the bin files.

1) Push the bin files to the */vendor/lib64/camera/* (for LA platform) or */usr/lib64/camera/* (for LE platform) folder. The target loads the xml files that were converted to binaries for camera software use.

2) Give the device proper permissions with the following example commands.

```
$ adb root
$ adb disable-verity
$ adb root
$ adb remount
```

3) Push the bin files to the specified directory.

```
$ adb push com.qti.sensormodule.sunny_ov13b10.bin /vendor/lib64/camera/
```

4) Restart the device.

```
$ adb reboot
```

NOTE: For a 32-bit processor, it is required to use the lib directory instead of lib64.

Chapter 3. Sensor Bringup Guidelines

3.1. Sensor software configuration

Every sensor has an associated configuration XML file that defines features, such as power settings, resolution, initialization settings, and exposure settings.

Complete settings are present at the *<sensorDriverData></sensorDriverData>* node of this XML file.

3.1.1. Sensor information nodes

- **Slave information node**

  The slave information node in the XML file contains the information that is used by the driver while probing the sensor.

Table 3-1. Sensor slave information node

Field	Description
slaveInfo	Contains the sensor slave information and power settings.
sensorName	Name of the sensor (example: imx230)
slaveAddress	8-bit or 10-bit slave address (example: 32) ➲ **NOTE:** For 0x20 slave address, the field value is 32. This will be used by sensor probe.
regAddrType	Register address/data size in bytes. Examples: • 1= Byte address • 2 = Word address • 3 = 3-byte address • 4 = Address type max
regDataType	Register address/data size in bytes. Examples: • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = Data type max
sensorIdRegAddr	Register address for sensor ID. Example: 22
sensorId	Sensor ID. Example: 560
sensorIDMask	Mask for sensor ID. Sensor ID may only be a few bits. Example: 4294967295.
i2cFrequencyMode	I2C Frequency mode of slave. Example: FAST Supported modes: • STANDARD (100 kHz) • FAST (400 kHz) • FAST_PLUS (1 MHz) • CUSTOM (custom frequency in DTSI)

Field	Description

powerUpSequence	Contains the power-up configuration sequence required to control the power to the device during power-on.
	Example:
	`<powerUpSequence>`
	`<powerSetting>`
	`<configType>RESET</configType>`
	`<configValue>0</configValue>`
	`<delayMs>0</delayMs>`
	`</powerSetting>`
	`</powerUpSequence >`

powerSetting	Contains power configuration type, value, and delay.
configType	Power configuration type. Example: MCLK Supported types: • MCLK • VANA • VDIG • VIO • VAF • RESET • STANDBY
configValue	• Power configuration type. Example: 19200000. • Recommended MCLK value: 19200000 (19.2Mhz)
delayMs	Delay in milliseconds. Example: 1

powerDownSequence	Contains the power-down configuration sequence that is required to control the power to the device during power-off.
	Example:
	`<powerDownSequence>`
	`<powerSetting>`
	`<configType>RESET</configType>`
	`<configValue>0</configValue>`
	`<delayMs>0</delayMs>`
	`</powerSetting>`
	`</powerDownSequence>`

- **Register information node**

The register information node contains the configuration register addresses for various sensor features such as to set gain, frame length lines, and test pattern generation.

Table 3-2. Register information node

Field	Description
regAddrInfo	Contains the information about the register addresses for various sensor settings.
xOutput	Controls the program width. Example: 844. If the register address is 0x34C, then the value of this field is 844.
yOutput	Controls the program height. Example: 846
frameLengthLines	Programs the frame length lines. Example: 832
lineLengthPixelClock	Programs the line length pixel clock. Example: 834

Field	Description
coarseIntgTimeAddr	Programs the coarse integration time. Example: 514
middleCoarseIntgTimeAddr	Programs the coarse integration time of middle exposure frame. Example: 400
shortCoarseIntgTimeAddr	Programs the coarse integration time of short exposure frame. Example: 350
globalGainAddr	Program the gain channel. Example: 516
middleGlobalGainAddr	Program the gain channel corresponding to middle exposure. Example: 14
shortGlobalGainAddr	Program the gain channel corresponding to short exposure. Example: 10
digitalGlobalGainAddr	Program the digital gain channel. Example: 3
middleDigitalGlobalGainAddr	Program the digital gain channel corresponding to middle exposure. Example: 2
shortDigitalGlobalGainAddr	Program the digital gain channel corresponding to short exposure. Example: 1
digitalGainRedAddr	Programs digital gain for the red channel. This address is optional but is required if supported by sensor. Example: 528
digitalGainGreenRedAddr	Programs digital gain for the green-red channel. This address is optional but is required if supported by sensor. Example: 526
digitalGainBlueAddr	Register address to program digital gain for the blue channel. This address is optional but is required if supported by sensor. Example: 530
digitalGainGreenBlueAddr	Programs digital gain for the green-blue channel. The address is optional but is required when supported by sensor. Example: 532
testPatternRAddr	Programs manual test pattern value for the red channel. Example: 1538
testPatternGRAddr	Programs manual test pattern value for the red channel. Example: 1540
testPatternBAddr	Programs manual test pattern value for the red channel. Example: 1542
testPatternGBAddr	Programs manual test pattern value for the red channel. Example: 1544

- **Resolution information node**

The resolution information node contains the information regarding resolution settings and configuration.

Table 3-3. Resolution information node

Field	Description
resolutionInfo	This node contains the configuration and settings for all the resolutions.
resolutionData	Contains the configuration data for one resolution. The number of sensors supported resolutions is equal to the number of resolutionData nodes. ➲ **NOTE:** First node of the resolutionData should always point to the full resolution configuration of the sensor.
colorFilterArrangement	Color filter arrangement of the sensor. Example: BAYER_RGGB Supported filter arrangements: • BAYER_BGGR • BAYER_GBRG • BAYER_GRBG • BAYER_RGGBn • BAYER_Y • YUV_UYVY • YUV_YUYV
streamInfo	Information related to the stream configuration. Example: <streamInfo> <streamConfiguration> <vc range="[0,3]">0</vc> <dt>43</dt> <frameDimension> <xStart>0</xStart> <yStart>0</yStart> <width>4608</width> <height>2592</height> </frameDimension> <bitWidth>10</bitWidth> <type>IMAGE</type> </streamConfiguration> </streamInfo>
streamConfiguration	Information related to the stream data.
vc	Virtual channel of the data Example: 2 Each unique combination of virtual channel (VC) and DT should map to a unique CID (Channel Identifier) value. Possible CID values for a given VC: • 0 – 0, 1, 2, 3 • 1 – 4, 5, 6, 7 • 2 – 8, 9, 10, 11 • 3 – 12, 13, 14, 15 Valid values for VC: 0, 1, 2, and 3.
dt	DT of the stream. Example: 43. Default value is 0x2B (10 bit RAW).
frameDimension	Frame dimension contains x and y start coordinates, and the total width and height of the image.
xStart	X coordinate of start of image
yStart	Y coordinate of start of image

Field	Description
width	Width of the image
height	Height of the image
bitWidth	Bit width of the data
type	Stream type. Example: IMAGE. Supported stream types: • BLOB • IMAGE: long exposure frame when there are multiple image frames generated and default image frame when there is single image frame. • IMAGE_SHORT: short exposure frame when there are multiple image frames generated. • PDAF: Phase Difference Auto Focus. PD (Phase Difference) is based on PDAF data generated by the sensor during same frame blanking period as image. • HDR: High Dynamic Range. Any histogram stat data generated by the sensor during the same frame blanking period as image. • META: Sensor frame meta stream output.
lineLengthPixelClock	Line length pixel clock of frame. Typically, this value is the active width + blanking width.
frameLengthLines	Frame length lines of frame. Typically, this value is the active height + blanking height.
minHorizontalBlanking	Minimum horizontal blanking interval in pixels
minVerticalBlanking	Minimum horizontal blanking interval in lines
outputPixelClock	This value should be obtained from the sensor vendor for a given mode of operation. Incorrect setting could cause PHY/CSID errors, incorrect IFE resource/clock selection behavior, or higher than required power.
horizontalBinning	Horizontal binning value
verticalBinning	Vertical binning value
framerate	Maximum frame rate
laneCount	The number of data lanes on which the sensor outputs data for a given mode of operation. The maximum data lane capability (given in the datasheet) of the sensor along with the sensor register settings configured in the driver determines the value.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>settleTimeNs</td><td>
Settle time in nanoseconds.
The value is configured, based on the output characteristics of the sensor, to ensure that the PHY transmitter of the sensor does not have sync issues with the PHY receiver of MSM chipsets.
DPHY:
settleTimeNs= ((85ns + 6UI)/T(Timer_clk)) – 10. where, TIMER_CLK refers to the operating frequency of PHY interface to which the camera sensor is connected (for example, CAMSS_PHY0_CSI0PHYTIMER_CLK for PHY0).
This value is set in the camera.dtsi file. In most cases, this value is 300Mhz but review the clock log. T(TIMER_CLK) is the duration of a clock cycle when the operating frequency is equal to TIMER_CLK, and is represented in nano second units.
Example:
UI (Unit Interval) is clock cycle time based on data rate per lane. For example, if the sensor outputs at the rate of 1000Mbps for each data lane, UI is 1ns.
TIMER_CLK= 300Mhz -> T(Timer_clk)=3.3ns The min value in xml is ((85+6)/3.3ns))-10=17
Refer to MIPI D-PHY v1.2 spec for more details C-PHY:
Min value: settleTimeNs= t3-prepare/Ttimer_clock+1 Max value: {t3-prepare+[t3-preamble]/2}/Ttimer_clock
If the Min value calculated does not work as expected, increase the value 1 by 1 and re-verify till it reaches the Max value calculated.
➲ Note: Work with vendor to set t3-prepare as minimum value as possible and refer to MIPI C-PHY v1.2 spec.
Ttimer_clock = PHY timer clk cycle time (for example, 3.33ns based on 300Mhz on SM8350 reference code).
Use correct value in case customized by customer based on sensor config and output data rate. OEM is requested to share details of such customization with Qualcomm for review. Unit: nanoseconds.
t3-prepare: confirm with sensor vendor. Unsure value must be measured via logic analyzer. The value can be different for each resolution mode setting. Unit: nanoseconds.
t3-preamble: confirm with sensor vendor. The value can be different for each resolution mode setting. Unit: nanoseconds.
Confirm with sensor vendor whether the following minimum requirements from receiver are met by the sensor. Note that UI here is calculated as cycle time for symbols throughput rate per trio. For example, for sensor outputting at the rate of Gsps per trio, UI is 1ns.
Timing parameter (only for 7nm and 5nm chipsets)
<table>
<tr><th>Timing parameter</th><th>Min (working)</th></tr>
<tr><td>Preamble</td><td>112UI (unit interval)</td></tr>
<tr><td>POST</td><td>112UI</td></tr>
<tr><td>LP001</td><td>50ns</td></tr>
<tr><td>LP111 (between data packets)</td><td>100ns</td></tr>
<tr><td>LP000</td><td>60ns</td></tr>
</table>
</td></tr>
<tr><td>is3Phase</td><td>Flag to know if the sensor is a three-phase sensor (C-PHY) or one- phase sensor (D-PHY).</td></tr>
</table>

Field	Description

resSettings	Sequence of register settings to configure the device with this resolution.
	➲ **NOTE:** Do not add stream on settings for the sensor in the initialization settings, as this enables the sensor streaming before real software stream on and may cause PHY to miss LP sequence.
	Example:
	`<resSettings>`
	`<regSetting>`
	`<registerAddr>0x114</registerAddr>`
	`<registerData>0x3</registerData>`
	`<regAddrType range="[1,4]">2</regAddrType>`
	`<regDataType range="[1,4]">1</regDataType>`
	`<operation>WRITE</operation>`
	`<delayUs>0x0</delayUs>`
	`</regSetting>`
	`<resSettings>`

regSetting	This node contains one register configuration and forms a unit of large-resolution register settings sequence.
registerAddr	Register address that is accessed.
registerData	• If the operation is WRITE, registerData is the data value to be written into the specified register address. • If the operation is READ, registerData is the number of bytes to be read from the specified register address.
regAddrType	Type of register address
regDataType	Type of register data
operation	Supported operation types: • WRITE • READ • POLL
delayUs	Delay in microseconds. ➲ **Note:** The delay is zero unless otherwise explicitly specified.

cropInfo	Crop information of the frame. Example:
	`<cropInfo>`
	`<left>0</left>`
	`<right>0</right>`
	`<top>0</top>`
	`<bottom>0</bottom>`
	`</cropInfo>`

left	Left crop pixel information
right	Right crop pixel information
top	Top crop pixel information
bottom	Bottom crop pixel information
exposureInfo	Resolution specific exposure control information. ➲ **NOTE:** Update if it is different than the common resolution exposure control information. If this information is not available, then info from common exposure control will be used.

Field	Description
exposureType	Information about the exposure type for which the exposure control information is being provided. Supported values: • DEFAULT: Corresponds to the long exposure when there are multiple exposures and default exposure in case of single exposure mode. • SHORT: Corresponds to the short exposure when there are multiple exposures. • MIDDLE: Corresponds to the middle exposure when there are multiple exposures.
coarseIntgTimeAddr	Register address to program the coarse integration time for this exposure type in this resolution.
globalGainAddr	Register address to program the gain channel for this exposure type in this resolution.
digitalGlobalGainAddr	Register address to program the digital gain channel for this exposure type in this resolution.
maxAnalogGain	Maximum analog gain supported by the sensor for this resolution.
minAnalogGain	Minimum analog gain supported by the sensor for this resolution.
maxDigitalGain	Maximum digital gain supported by the sensor for this resolution.
minDigitalGain	Minimum digital gain supported by the sensor for this resolution.
maxLineCount	Maximum Line Count supported by the sensor for this resolution.
minLineCount	Minimum Line Count supported by the sensor for this resolution.
verticalOffset	Minimum offset to be maintained between the line count and frame length lines for this resolution.
frameOffsetInfo	Contains addresses of various frame offset registers
middleFrameOffsetRegister	Register address to program the frame offset value of middle exposure frame.
shortFrameOffsetRegister	Register address to program the frame offset value of short exposure frame.
HDRExposureType	Information about the HDR exposure type of this resolution. This value must be filled only if the resolution is a type of HDR. **Supported values**: • TWOEXPOSURE: Indicates that two exposures are used in the HDR mode (Long & Short exposure) • THREEEXPOSURE: Indicates that three exposures are used in the HDR mode (Long, Middle & Short exposure)
capability	List of features/capabilities supported by the sensor. **Supported features:** • HFR • iHDR • INSENSOR_HDR • PDAF • QUADRA_CFA • RAW_HDR • ZIGZAG_HDR
ADCReadoutTime	Analog to digital conversion time for the sensor. Unit: milliseconds. Example: 2

If INSENSORZOOM is set, then the resolution information should be the same. For example, assuming mode 0 resolutionInfo is 8000*6000, mode 9 is the corresponding mode with INSENSORZOOM, and then mode 9 resolutionInfo should be same as mode 0 as there is no sensor reconfiguration during mode switch (meaning sensor hardware PLL clock settings will remain the same in both modes).

frameDimension is based on the sensor full output resolution. Refer to the following example: if frameDim = (0,517,4208,2104), fullResolutionWidth=4208 and fullResolutionHeight=3120, then 517 is invalid as 517*2+2104 is larger than the frame boundary 3120

In most cases, share the resolution information below when facing issues:

```
<resolutionInfo>
<resolutionData>
<colorFilterArrangement>BAYER_RGGB</colorFilterArrangement>
<vc range="[0,3]">0</vc>
<dt>43</dt>
<xStart>0</xStart>
<yStart>0</yStart>
<width>8000</width>
<height>6000</height>
<bitWidth>10</bitWidth>
<type>IMAGE</type>
<lineLengthPixelClock>9440</lineLengthPixelClock>
<frameLengthLines>6074</frameLengthLines>
<minHorizontalBlanking>678</minHorizontalBlanking>
<minVerticalBlanking>69</minVerticalBlanking>
<outputPixelClock>1586910000</outputPixelClock>
<horizontalBinning>1</horizontalBinning>
<verticalBinning>1</verticalBinning>
<frameRate>30.00</frameRate>
<laneCount>3</laneCount>
<settleTimeNs>14</settleTimeNs>
<is3Phase>1</is3Phase>
```

Refer to the Sensor XML and PDAF XML index mapping section for sensor XML and PDAF XML index mapping.

- **Exposure control information node**

The exposure control information node contains the exposure details such as maximum gain, maximum line count, and conversion formulas for gain manipulation.

Table 3-4. ***Exposure information***

Field	Description
exposureControlInfo	This node contains the information about the exposure details such as maximum gain, maximum line count, and conversion formulas for gain manipulation. Example: <exposureControlInfo> <maxAnalogGain>8</maxAnalogGain> <maxDigitalGain>2</maxDigitalGain> <verticalOffset>20</verticalOffset> <maxLineCount>65515</maxLineCount> <realToRegGain>512-(512/realGain)</realToRegGain> <regToRealGain>512/(512-regGain)</regToRealGain> </exposureControlInfo>
maxAnalogGain	Maximum analog gain supported by the sensor.
maxDigitalGain	Maximum digital gain supported by the sensor.
verticalOffset	Minimum offset to be maintained between the line count and frame length lines.
maxLineCount	Maximum line count supported by the sensor.
minLineCount	Minimum line count supported by the sensor.
middleMaxLineCount	Maximum line count supported by the sensor for middle exposure.

Field	Description
middleMinLineCount	Minimum line count supported by the sensor for middle exposure.
shortMaxLineCount	Maximum line count supported by the sensor for short exposure.
shortMinLineCount	Minimum line count supported by the sensor for short exposure.
realToRegGain	Real gain to register the gain equation. The equation must contain realGain in the equation.
regToRealGain	Register gain to real gain equation. The equation must contain regGain in the equation.

- **Stream on and stream off information node**

  Register settings to start streaming.

  Table 3-5. Stream on and stream off information

Field	Description
streamOnSettings	Sequence of register settings to configure the device to start streaming. In the sensor driver, the following settings are configured to match this guideline: Take the clock and data lanes from LP11 to HS TxState. Example: <streamOnSettings> <regSetting> <registerAddr>0x0100</registerAddr> <registerData>0x1</registerData> <regAddrType range="[1,4]">2</regAddrType> <regDataType range="[1,4]">1</regDataType> <operation>WRITE</operation> <delayUs>0</delayUs> </regSetting> </streamOnSettings>
regSetting	Register setting configuration. Contains: Register address, register data, register address type, register DT, operation, and delay in micro seconds.
streamOffSettings	Sequence of register settings to configure the device to stop streaming. Configure the following settings to match this guideline and take the clock and data lanes to LP11. Example: <streamOffSettings> <regSetting> <registerAddr>0x0100</registerAddr> <registerData>0x0</registerData> <regAddrType range="[1,4]">2</regAddrType> <regDataType range="[1,4]">1</regDataType> <operation>WRITE</operation> <delayUs>0</delayUs> </regSetting> </streamOffSettings>
regSetting	Register setting configuration. Contains: Register address, register data, register address type, register DT, operation, and delay in micro seconds.

1) Group hold settings

Register settings to configure the device in group hold settings.

Table 3-6. Group hold settings

Field	Description

groupHoldOnSettings	Sequence of register settings to configure the device in group hold settings.
	Example:
	`<groupHoldOnSettings>`
	`<regSetting>`
	`<registerAddr>0x0104</registerAddr>`
	`<registerData>0x1</registerData>`
	`<regAddrType range="[1,4]">2</regAddrType>`
	`<regDataType range="[1,4]">1</regDataType>`
	`<operation>WRITE</operation>`
	`<delayUs>0</delayUs>`
	`</regSetting>`
	`</groupHoldOnSettings>`

regSetting	Register setting configuration. Contains: Register address, register data, register address type, register data type, operation, and delay in micro seconds.

groupHoldOffSettings	Sequence of register settings to configure the device to stop group hold settings.
	Example:
	`<groupHoldOffSettings>`
	`<regSetting>`
	`<registerAddr>0x0104</registerAddr>`
	`<registerData>0x0</registerData>`
	`<regAddrType range="[1,4]">2</regAddrType>`
	`<regDataType range="[1,4]">1</regDataType>`
	`<operation>WRITE</operation>`
	`<delayUs>0</delayUs>`
	`</regSetting>`
	`</groupHoldOffSettings>`

regSetting	Register setting configuration. Contains: Register address, register data, register address type, register data type, operation, and delay in micro seconds.

2) Initial settings

Initial settings define the sequence of register settings to initialize the sensor.

Table 3-7. Initialization information

Field	Description

initSettings	Sequence of register settings to initialize the sensor.
	➲ **NOTE:** Do not add stream on settings for the sensor in the initialization settings, as this enables the sensor streaming before real software stream on and may cause PHY to miss LP sequence.
	Example:
	`<initSettings>`
	`<regSetting>`
	`<registerAddr>0x136</registerAddr>`
	`<registerData>0x18</registerData>`
	`<regAddrType range="[1,4]">2</regAddrType>`
	`<regDataType range="[1,4]">1</regDataType>`
	`<operation>WRITE</operation>`
	`<delayUs>0</delayUs>`
	`</regSetting>`
	`</initSettings>`

regSetting	Register setting configuration. Contains: Register address, register data, register address type, register data type, operation, and delay in micro seconds.

- **Test pattern information node**

  This node contains the information about the test pattern generation register settings.

Table 3-8. Test pattern information

Field	Description
testPatternInfo	Contains the information about the test pattern generation register settings. Example: <testPatternInfo> <testPatternData> <mode>OFF</mode> <settings> <regSetting> <registerAddr>0x136</registerAddr> <registerData>0x18</registerData> <regAddrType range="[1,4]">2</regAddrType> <regDataType range="[1,4]">1</regDataType> <operation>WRITE</operation> <delayUs>0</delayUs> </regSetting> </settings> </testPatternData> </testPatternInfo>

Table 3-9. Test pattern data

Field	Description
testPatternData	This node contains the register and mode settings of a particular test pattern.
Mode	Supported modes: • OFF • SOLID_COLOR • COLOR_BARS • COLOR_BARS_FADE_TO_GRAY • PN9 • CUSTOM1
settings	Sequence of register settings to configure the test pattern on the sensor.
regSetting	Register setting configuration. Contains: Register address, register data, register address type, register data type, operation, and delay in micro seconds.

- **Color level information node**

  The color level information node provides details about various channels in complete dark light.

Table 3-10. Color level information

Field	Description
colorLevelInfo	Color level information. The default level values in various channels in complete dark light. Examples: <colorLevelInfo> <whiteLevel>1023</whiteLevel> <rPedestal>64</rPedestal> <grPedestal>64</grPedestal> <bPedestal>64</bPedestal> <gbPedestal>64</gbPedestal> </colorLevelInfo>

Field	Description
whiteLevel	White level value.
rPedestal	Pedestal value for the red channel.
grPedestal	Pedestal value for the green-red channel.
bPedestal	Pedestal value for the blue channel.
gbPedestal	Pedestal value for the green-blue channel.

Table 3-11. Black region information

Field	Description

opticalBlackRegionInfo	Information about black regions.
	Multiple black regions are provided, if applicable. **Example:**
	`<opticalBlackRegionInfo>`
	`<dimension>`
	`<xStart>0</xStart>`
	`<yStart>0</yStart>`
	`<width>4608</width>`
	`<height>2592</height>`
	`</dimension>`
	`</opticalBlackRegionInfo>`

dimension	Frame dimension: Contains xStart, yStart, width, and height.
xStart	X start coordinate of the region.
yStart	Y start coordinate of the region.
width	Width of the region.
height	Height of the region.

Table 3-12. Pixel array information

Field	Description
pixelArrayInfo	Information about the pixel array. Active dimension and dummy pixels width are provided.

activeDimension	Width and height of the frame or subframe.
	Example:
	`<activeDimension>`
	`<xStart>0</xStart>`
	`<yStart>0</yStart>`
	`<width>4608</width>`
	`<height>2592</height>`
	`</activeDimension>`

xStart	Start of the X output in the frame.
yStart	Start of the Y output in the frame.
width	Total width of the frame in pixels.
height	Total height of the frame in pixels.

dummyInfo	Dummy pixels surrounding the active pixel array.
	Example:
	`<dummyInfo>`
	`<left>0</left>`
	`<right>0</right>`
	`<top>0</top>`
	`<bottom>0</bottom>`
	`</dummyInfo>`

left	Starting coordinate of the left dummy pixel.

Field	Description
right	Starting coordinate of the right dummy pixel.
top	Starting coordinate of the top dummy pixel.
bottom	Starting coordinate of the bottom dummy pixel.

Table 3-13. Delay information

Field	Description
appDelayInfo	Delay information.
linecount	Number of frames required to apply the line count.
gain	Number of frames required to apply the gain.
maxPipeline	Maximum pipeline delay in number of frames.
frameSkip	Number of initial bad frames to skip.

Table 3-14. Sensor property information

Field	Description

sensorProperty	Sensor property information. Example:
	`<sensorProperty>`
	`<pixelSize>0</pixelSize>`
	`<cropFactor>0</cropFactor>`
	`<sensingMethod>ONE_CHIP_COLOR_AREA</sensingMethod>`
	`</sensorProperty>`

pixelSize	Pixel size in micro meters.
cropFactor	Crop factor.
sensingMethod	Sensing method of sensor. Supported sensing methods: • UNDEFINED • ONE_CHIP_COLOR_AREA TWO_CHIP_COLOR_AREA • THREE_CHIP_COLOR_AREA • COLOR_SEQUENCE_AREA • TRILINEAR • COLOR_SEQUENCE_LINEAR

Table 3-15. Frame sync information

Field	Description

masterSettings	Sequence of register settings to configure the sensor to HW master mode.
	Note: It is valid only when HW sync is enabled. Example:
	`<masterSettings>`
	`<regSetting>`
	`<registerAddr>0x0350</registerAddr>`
	`<registerData>0x0</registerData>`
	`<regAddrType range="[1,4]">2</regAddrType>`
	`<regDataType range="[1,4]">1</regDataType>`
	`<operation>WRITE</operation>`
	`<delayUs>0</delayUs>`
	`</regSetting>`
	`</masterSettings>`

slaveSettings	Sequence of register settings to configure the sensor to HW slave mode. It is similar to masterSettings.

Field	Description
softwareSyncConfiguration	Flag to indicate if we need to compensate blanking for current sensor. It is only valid when SW frame sync is enabled and it is only valid on the SM8350 platform. Refer to KBA-200927214121 for details. Example: <softwareSyncConfiguration> <blankingCompensation>true</blankingCompensation> </softwareSyncConfiguration>

Table 3-16. Mode switch register information

Field	Description
modeSwitchRegInfo	Register settings and virtual channel register addresses to be programmed during seamless mode switch for the following data type. PDAFVCAddress, ImageLongVCAddress,ImageShortVCAddress, ImageMiddleVCAddress,BlobVCAddress, HDRVCAddress, MetaVCAddress Example: <modeSwitchRegInfo> <regAddrType>2</regAddrType> <regDataType>1</regDataType> <ImageLongVCAddress>0x0110</ImageLongVCAddress> <PDAFVCAddress>0x3076</PDAFVCAddress> </modeSwitchRegInfo>.

Table 3-17. Register setting group information

Field	Description
registerSettings	Register settings that need to be programmed as a part of this group.
registerSettingsType	Defines type of setting this register set indicates Supported flags are: • StreamOn • StreamOff StreamOn: This will indicate streamon settings forspecial use cases. StreamOff: This will indicate streamoff settings forspecial use cases.
registerSettingsFlag	Use this flag to indicate any settings specially applied for certain use cases. Supported flags are: PN9Test PN9Test: This indicates settings to be used during PN9 Pattern testing case.

3.1.2. Module configuration XML

This XML is used to store the camera module-specific information, including the lens information, mount angles, actuator, OIS, and flash-related information.

Table 3-18. Camera position

Field	Description
CameraPosition	Position of the sensor module. Supported values: • REAR • FRONT • REAR_AUX • FRONT_AUX • EXTERNAL

Table 3-19. Lens information

Field	Description
LensInformation	This node holds sensor lens-specific information.
focalLength	Focal length of the lens in millimeters.
fNumber	F-Number of the optical system.
minFocusDistance	Minimum focus distance in meters.
maxFocusDistance	Total focus distance in meters.
horizontalViewAngle	Horizontal view angle in degrees.
verticalViewAngle	Vertical view angle in degrees.
maxRollDegree	Maximum roll degree.
maxPitchDegree	Maximum pitch degree.
maxYawDegree	Maximum yaw degree.

Table 3-20. Lens information list

Field	Description
LensInformationList	This node holds sensor list of lens-specific information based on version number.
version	This corresponds to module version defined in EEPROM.
chromatixName	Chromatix name corresponding to this version.
lensInfo	Lens specific information.

Table 3-21. Module configuration

Field	Description
ModuleConfiguration	Module configuration.
cameraId	cameraId is the ID to which DTSI node is mapped. Typically, cameraId is the slot ID for Non-Combo mode.
moduleName	Name of the module integrator.
sensorName	Name of the sensor in the image sensor module.
sensorSlaveAddress	Sensor slave address to override the value present in the binary. Optional element. Do not use this entry if override is not required.
sensorI2CFrequencyMode	Sensor operation frequency mode. Supported types are: • STANDARD • FAST • FAST_PLUS • CUSTOM
actuatorName	Name of the actuator in the image sensor module. Optional element. Skip this element if actuator is not present.
oisName	Name of the OIS in the image sensor module. Optional element. Skip this element if OIS is not present.
eepromName	Name of the EEPROM in the image sensor module. Optional element. Skip this element if EEPROM is not present.
flashName	Flash name to open binary. Binary name is of form flashName_flash.bin Example: - pmic_flash.bin.

Field	Description
chromatixName	To open binary. Binary name is of the form sensor_model_chromatix.bin.
position	Camera position. Supported types: • REAR • FRONT • REAR_AUX • FRONT_AUX • EXTERNAL
CSIInfo	This holds CSI information like lane assign, combo mode and cphy- dphy combo mode flags
laneAssign	Indicates the value used to determine the CPHY and DPHY lanes that should be assigned to this sensor. Example: 0x2310
isComboMode	Flag to enable Combo mode. This flag is enabled if multiple sensors are using same CSI-PHY receiver: • First camera – Connected to PHY lanes 2:0. Clock lane is connected to PHY lane 1. Data lanes (up to 2) are connected to either of PHY lanes 0 or 2. • Second camera – Connected to PHY lanes 4:3. Clock lane is connected to PHY lane 4. Data lanes are connected to PHYlane 3.
cphydphyComboMode	Flag to enable CPhy-DPhy Combo mode. This flag is enabled if multiple sensors are being used in either combination: • Two sensors using CPHY receiver and One sensor with DPHY receiver • Two sensors using DPHY receiver and One sensor with CPHY receiver
pdafName	Name of the PDAF driver used to configure this image sensor module. Optional element. Skip this element if PDAF is not supported.
lensInfoList	List of lens information nodes. See Table 3-20.

Table 3-22. Module group

Field	Description
ModuleGroup	Module group can contain either one module or two modules. Dual camera, stereo camera use cases contain two modules in the group.
moduleConfiguration	Module configuration information. See Table 3-21.

Table 3-23. Camera module data

Field	Description
CameraModuleData	Camera module data.
module_version	Camera module version.
major_revision	Contains the driver revision.
minor_revision	Contains the minor revision number.
incr_revision	Contains incremental revision number.
moduleGroup	Specifies the module configuration data (see Table 3-21) count, ID, and the data.

For example:

```
<cameraId>4</cameraId>
<!--Name of the module integrator -->
<moduleName>semco</moduleName>
<!--Name of the sensor in the image sensor module -->
<sensorName>imx586</sensorName>
<!--Actuator name in the image sensor module
This is an optional element. Skip this element if actuator is not present -->
<actuatorName>lc898217xc</actuatorName>
<eepromName>cat24c64_imx586</eepromName>
<flashName>pmic</flashName>
<!--Chromatix name is used to used to open binary.
Binary name is of the form sensor_model_chromatix.bin -->
<chromatixName>semco_imx586</chromatixName>
```

3.2. Sensor hardware configuration

The hardware configuration of the camera sensor module is saved in the kernel DTSI files.

For more information on how to customize camera DTSI, refer to k*ernel/msm-4.19/Documentation/devicetree/bindings/media/video/* or *vendor/qcom/ proprietary/camera-devicetree/bindings/* (for relatively new baselines).

3.2.1. Sensor kernel nodes

Table 3-24. Camera_LDO

Field	Description
compatible	Data type to which this node corresponds.
reg	ID of the node.
regulator-name	LDO regulator name.
regulator-min-microvolt	Minimum voltage for this LDO.
regulator-max-microvolt	Maximum voltage for this LDO.
regulator-enable-ramp-delay	The time taken, in microseconds, for the supply rail to reach the target voltage, ± whatever tolerance the board design requires. This property describes the total system ramp time required due to the combination of internal ramping of the regulator itself, and board design issues such as trace capacitance and load on the supply.
enable-active-high	Enable with active high.
gpio	GPIO to be used with this node.
pinctrl-names	Pinctrl name.
pinctrl-0	Pinctrl handle for this GPIO.
vin-supply	Input voltage supply node name.

For example:

```
camera_ldo: gpio-regulator@2 { compatible = "regulator-fixed"; reg = <0x02 0x00>;
regulator-name = "camera_ldo"; regulator-min-microvolt = <1050000>;
regulator-max-microvolt = <1050000>;
regulator-enable-ramp-delay = <233>; enable-active-high;
gpio = <&pm8998_gpios 9 0>; pinctrl-names = "default";
pinctrl-0 = <&camera_dvdd_en_default>; vin-supply = <&pm8998_s3>;
};
```

Table 3-25. CCI-based sensor driver

Field	Description
cell-index	Points to the ID of the sensor node.
reg	Corresponds to the cell-index of the node.
compatible	Data type to which this node corresponds to.
qcom,csiphy-sd-index	Paired CSIPHY node for this DT.
qcom,sensor-position-roll	Sensor position roll angle.
qcom,sensor-position-pitch	Sensor position pitch angle.
qcom,sensor-position-yaw	Sensor position yaw angle.
qcom,led-flash-src	LED flash source node name.
qcom,actuator-src	Actuator source node name.
qcom,ois-src	OIS source node name.
qcom,eeprom-src	EEPROM source node name.
cam_vio-supply	I/O voltage supply source.
cam_vana-supply	Analog voltage source.
cam_vdig-supply	Digital voltage source.
cam_clk-supply	Camera clock source.
qcom,cam-vreg-name	Voltage regulator node names.
qcom,cam-vreg-min-voltage	Minimum voltage, in the above order.
qcom,cam-vreg-max-voltage	Maximum voltage, in the above order.
qcom,cam-vreg-op-mode	Operation mode of the voltage regulator, in above order.
regulator-names	Should contain names of all regulators needed.
rgltr-cntrl-support	This property is required if the SW control regulator parameters (e.g., rgltr-min-voltage).
pwm-switch	This property is required for regulator to switch into PWM mode.
rgltr-min-voltage	Should contain minimum voltage level for regulators mentioned in regulator-names property (in the same order).
rgltr-max-voltage	Should contain maximum voltage level for regulators mentioned in regulator-names property (in the same order).
rgltr-load-current	Should contain optimum voltage level for regulators mentioned in regulator-names property (in the same order).
qcom,gpio-no-mux	Whether GPIO muxing is enabled.
pinctrl-names	Pinctrl handle names for this GPIO if target uses pinctrl.
pinctrl-0	Binding to GPIO pin and function node.
pinctrl-1	Binding to GPIO pin and function node.
gpios	List of GPIO pins used.
qcom,gpio-reset	Should contain index to GPIO used by sensors reset_n.
qcom,gpio-vana	Should contain index to GPIO used by sensors analog vreg enable.
qcom,gpio-vaf	Should contain index to GPIO used by sensors of vreg enable.
qcom,gpio-req-tbl-num	Should contain index to GPIO specific to this sensor.
qcom,gpio-req-tbl-flags	Should contain direction of GPIO present in qcom, gpio- req-tbl-num property (in the same order).

Field	Description
qcom,gpio-req-tbl-label	Should contain name of GPIO present in qcom, gpio-req- tbl-num property (in the same order).
qcom,sensor-position	Mount angle of the sensor • 0 – Back Camera • 1 – Front Camera
qcom,sensor-mode	Supported sensor mode • 0 – back camera 2D • 1 – front camera 2D • 2 – back camera 3D • 3 – back camera int 3D
qcom,cci-master	I2C master used for this sensor • 0 – MASTER 0 • 1 – MASTER 1
status	Whether this node is enabled or disabled.
clocks	Clock node names.
clock-names	Name of the clocks required for the device.
qcom,clock-rates	Clock rate in Hz.

For the latest definition, refer to *kernel/msm-4.19/Documentation/devicetree/bindings/media/video/msm-cam-cci.txt* or *vendor/qcom/proprietary/camera-devicetree/bindings/msm-cam-cci.txt* (for relatively new baselines).

For example:

```
qcom,cam-sensor@0 { cell-index = <0>;
compatible = "qcom,cam-sensor"; reg = <0x0>;
qcom,csiphy-sd-index = <0>;
qcom,sensor-position-roll = <90>;
qcom,sensor-position-pitch = <0>;
qcom,sensor-position-yaw = <180>; qcom,led-flash-src = <&led_flash_rear>; qcom,actuator-
src = <&actuator_rear>; qcom,ois-src = <&ois_rear>;
qcom,eeprom-src = <&eeprom_rear>; cam_vio-supply = <&pm8998_lvs1>; cam_vana-supply =
<&pmi8998_bob>; cam_vdig-supply = <&camera_rear_ldo>; cam_clk-supply = <&titan_top_gdsc>;
qcom,cam-vreg-name = "cam_vio", "cam_vana", "cam_vdig", "cam_clk";
qcom,cam-vreg-min-voltage = <0 3312000 1050000 0>;
qcom,cam-vreg-max-voltage = <0 3600000 1050000 0>;
qcom,cam-vreg-op-mode = <0 80000 105000 0>;
qcom,gpio-no-mux = <0>;
pinctrl-names = "cam_default", "cam_suspend"; pinctrl-0 = <&cam_sensor_mclk0_active
&cam_sensor_rear_active>; pinctrl-1 = <&cam_sensor_mclk0_suspend
&cam_sensor_rear_suspend>; gpios = <&tlmm 13 0>,
<&tlmm 80 0>,
<&tlmm 79 0>;
qcom,gpio-reset = <1>;
qcom,gpio-vana = <2>;
qcom,gpio-req-tbl-num = <0 1 2>;
qcom,gpio-req-tbl-flags = <1 0 0>; qcom,gpio-req-tbl-label = "CAMIF_MCLK0",
"CAM_RESET0", "CAM_VANA";
qcom,sensor-mode = <0>;
qcom,cci-master = <0>; status = "ok";
clocks = <&clock_camcc CAM_CC_MCLK0_CLK>; clock-names = "cam_clk";
qcom,clock-rates = <24000000>;
};
```

Table 3-26. OIS node

Field	Description
cell-index	Points to the ID of this node.
reg	Corresponds to the cell-index of the node.
compatible	Data type to which this node corresponds to.
qcom,cci-master	CCI node that drives this node.
cam_vaf-supply	Should contain regulator from which AF voltage is supplied.
qcom,cam-vreg-name	Voltage regulator node names.
qcom,cam-vreg-min-voltage	Minimum voltage, in the above order.
qcom,cam-vreg-max-voltage	Maximum voltage, in the above order.
qcom,cam-vreg-op-mode	Operation mode of the voltage regulator, in above order.
status	Whether this node is enabled or disabled.

For example:

```
ois_rear: qcom,ois@0 { cell-index = <0>; reg = <0x0>;
compatible = "qcom,ois"; qcom,cci-master = <0>;
cam_vaf-supply = <&actuator_regulator>; qcom,cam-vreg-name = "cam_vaf"; qcom,cam-vreg-min-
voltage = <2800000>;
qcom,cam-vreg-max-voltage = <2800000>;
qcom,cam-vreg-op-mode = <0>; status = "disabled";
};
```

Table 3-27. EEPROM node

Field	Description
cell-index	Points to the node ID.
reg	Corresponds to the cell-index of the node.
compatible	Data type that this node corresponds to.
qcom,cci-master	CCI node that drives this node.
cam_vio-supply	Input/output voltage supply source.
cam_vana-supply	Analog voltage source.
cam_vdig-supply	Digital voltage source.
cam_clk-supply	Camera clock source.
qcom,cam-vreg-name	Voltage regulator node names.
qcom,cam-vreg-min-voltage	Minimum voltage, in the above order.
qcom,cam-vreg-max-voltage	Maximum voltage, in the above order.
qcom,cam-vreg-op-mode	Operation mode of the voltage regulator, in above order.
rgltr-load-current	-
qcom,gpio-no-mux	Whether GPIO muxing is enabled.
pinctrl-names	Pinctrl handle names.
pinctrl-0	Binding to GPIO pin and function node.
pinctrl-1	Binding to GPIO pin and function node.
gpios	List of GPIO pins used.
qcom,gpio-reset	Should contain index to GPIO used by sensors reset_n.
qcom,gpio-vana	Should contain index to GPIO used by sensors analog vreg enable.

Field	Description
qcom,gpio-vaf	Should contain index to GPIO used by sensors i2af vreg enable.
qcom,gpio-req-tbl-num	Should contain index to GPIO specific to this sensor.
qcom,gpio-req-tbl-flags	Should contain direction of GPIO present in qcom, gpio-req-tbl-num property (in the same order).
qcom,gpio-req-tbl-label	Should contain name of GPIO present in qcom, gpio-req-tbl-num property (in the same order).
qcom,sensor-position	Mount angle of the sensor: • 0 – Back camera • 1 – Front camera
qcom,sensor-mode	Supported sensor mode: • 0 – Back camera 2D • 1 – Front camera 2D • 2 – Back camera 3D • 3 – Back camera int 3D
qcom,cci-master	I2C master used for this sensor: • 0 – MASTER 0 • 1 – MASTER 1
status	Whether this node is enabled or disabled.
clocks	Clock node names.
clock-names	Name of the clocks required for the device.
qcom,clock-rates	Clock rate in Hz.

For the latest definition, refer to *kernel/msm-4.19/Documentation/devicetree/bindings/media/video/msm-cam-cci.txt* or *vendor/qcom/proprietary/camera-devicetree/bindings/msm-cam-cci.txt* (for relatively new baselines).

For example:

```
eeprom_rear: qcom,eeprom@0 { cell-index = <0>;
reg = <0>;
compatible = "qcom,eeprom"; cam_vio-supply = <&pm8998_lvs1>; cam_vana-supply =
<&pmi8998_bob>;
cam_vdig-supply = <&camera_rear_ldo>; cam_clk-supply = <&titan_top_gdsc>;
qcom,cam-vreg-name = "cam_vio", "cam_vana", "cam_vdig", "cam_clk";
qcom,cam-vreg-min-voltage = <0 3312000 1050000 0>;
qcom,cam-vreg-max-voltage = <0 3600000 1050000 0>;
qcom,cam-vreg-op-mode = <0 80000 105000 0>;
qcom,gpio-no-mux = <0>;
pinctrl-names = "cam_default", "cam_suspend"; pinctrl-0 = <&cam_sensor_mclk0_active
&cam_sensor_rear_active>; pinctrl-1 = <&cam_sensor_mclk0_suspend
&cam_sensor_rear_suspend>; gpios = <&tlmm 13 0>,
<&tlmm 80 0>,
<&tlmm 79 0>,
<&tlmm 27 0>;
qcom,gpio-reset = <1>;
qcom,gpio-vana = <2>;
qcom,gpio-vaf = <3>;
qcom,gpio-req-tbl-num = <0 1 2 3>;
qcom,gpio-req-tbl-flags = <1 0 0 0>; qcom,gpio-req-tbl-label = "CAMIF_MCLK0",
"CAM_RESET0", "CAM_VANA0", "CAM_VAF";
qcom,sensor-position = <0>;
qcom,sensor-mode = <0>;
qcom,cci-master = <0>; status = "ok";
clocks = <&clock_camcc CAM_CC_MCLK0_CLK>; clock-names = "cam_clk";
qcom,clock-rates = <24000000>;
};
```

3.2.2. CCI timing and debug

The I2C timing may vary with the quantity of slave devices on the I2C bus.

Make sure the characteristics of the SDA and SCL I/O stages (High to Low, Low to High, etc.) are within the specified range.

Refer to the table below for the most common CCI errors. Enable DUMP_CCI_REGISTERS in *cam_sensor_module/cam_cci/cam_cci_core.c.*

Table 3-28. IRQ status

IRQ status	Error reason	What to check
NACK_ERR	I2C command of I2C Master NACKed due to no ACK response from slave.	Check if any Slave devices are holding the bus due to improper power on/down, or board level noise.
CMD_ERR	An unsupported command word is programmed into HW.	Check if any Slave devices are holding the bus due to improper power on/down, or board level noise.
OVERFLOW	I2C command FIFO or I2C RD FIFO has overflowed.	Ensure we do not load command words more than the available space in I2C Master command FIFO.
UNDERFLOW	I2C RD FIFO has underflowed.	Ensure we do not read from I2C master RD FIFO when it is empty.

Common CCI error log:

```
if (irq_status0 & CCI_IRQ_STATUS_0_I2C_M0_ERROR_BMSK) { cci_dev-
>cci_master_info[MASTER_0].status = -EINVAL;
if (irq_status0 & CCI_IRQ_STATUS_0_I2C_M0_Q0_NACK_ERROR_BMSK) { CAM_ERR(CAM_CCI,
"Base:%pK, M0_Q0 NACK ERROR: 0x%x",
base, irq_status0);
complete_all(&cci_dev->cci_master_info[MASTER_0]
.report_q[QUEUE_0]);
}
if (irq_status0 & CCI_IRQ_STATUS_0_I2C_M0_Q1_NACK_ERROR_BMSK) { CAM_ERR(CAM_CCI,
"Base:%pK, M0_Q1 NACK ERROR: 0x%x",
base, irq_status0);
complete_all(&cci_dev->cci_master_info[MASTER_0]
.report_q[QUEUE_1]);
}
if (irq_status0 & CCI_IRQ_STATUS_0_I2C_M0_Q0Q1_ERROR_BMSK) CAM_ERR(CAM_CCI,
"Base:%pK, M0 QUEUE_OVER/UNDER_FLOW OR CMD ERR: 0x%x",
base, irq_status0);
if (irq_status0 & CCI_IRQ_STATUS_0_I2C_M0_RD_ERROR_BMSK) CAM_ERR(CAM_CCI,
"Base: %pK, M0 RD_OVER/UNDER_FLOW ERROR: 0x%x",
```

3.2.3. CCI operation speed configuration

Following the I2C specification, CCI (for chipsets where it is present) can be operated at the following frequencies:

- 100 kHz (STANDARD)
- 400 kHz (FAST)
- 1 MHz (FAST PLUS)

I2C frequency mode controls the operation speed.

For custom mode, the CCI tuning parameters setting information is in *kernel/msm-4.19/arch/arm64 /boot/dts/qcom/XXX-camera.dtsi* or *vendor/qcom/proprietary/camera/devicetree/XXX-camera.dtsi* (for relatively new baselines).

Example:

```
i2c_freq_custom: qcom,i2c_custom_mode {
qcom,hw-thigh = <38>;
qcom,hw-tlow = <56>;
```

```
qcom,hw-tsu-sto = <40>;
qcom,hw-tsu-sta = <40>;
qcom,hw-thd-dat = <22>;
qcom,hw-thd-sta = <35>;
qcom,hw-tbuf = <62>;
qcom,hw-scl-stretch-en = <1>;
qcom,hw-trdhld = <6>;
qcom,hw-tsp = <3>;
qcom,cci-clk-src = <37500000>;
status = "ok";
};
```

- If a custom CCI configuration is used, then the speed of I2C frequency information is calculated by the formula CCI clock = (src clock)/(hw_thigh+hw_tlow).

  As the CCI clock frequency is typically 19.2 MHz, the standard CCI frequency case is: CCI clock = 19.2 MHz/(78 + 114) = 100 kHz.

- If there is no CCI hardware, or when QUP is used for I2C, QUP speed is set to either 100 kHz or 400 kHz from the DTSI file.

➲ **NOTE**:

- It is recommended to use one of the supported operation frequencies instead of custom mode as they have been fully verified.
- The serial peripheral interface (SPI) is not supported for sensor configuration. It only supports the EEPROM driver.

3.2.4. Camera IFE clock configuration

This section discusses how to optimally configure the IFE clock for SM8150 and SM8250.

- **Factors affecting IFE clock**

The IFE clock rate is determined by the following sensor-based factors:

- Input frame dimensions for the sensor mode in use. Input dimensions are specified in terms of the pixel width and height of the frame being fed into the IFE from the sensor.
- The horizontal and vertical blanking periods for the sensor mode in use.
- The sensor output clock rate for the sensor mode in use.

The input dimensions and the blanking periods are specified in the corresponding sensor's XML configuration file. For example, a typical sensor may have a configuration for a particular mode that may look like this:

```
<resolutionData>
<streamInfo>
<streamConfiguration>
<vc range="[0,3]">0</vc>
<dt>43</dt>
<frameDimension>
<xStart>0</xStart>
<yStart>0</yStart>
<width>5488</width>
<height>4112</height>
</frameDimension>
<!--Minimum horizontal blanking interval in terms of the sensor's output pixel clock rate
-->
<minHorizontalBlanking>467</minHorizontalBlanking>
<!--Minimum vertical blanking interval in terms lines -->
<minVerticalBlanking>278</minVerticalBlanking>
<outputPixelClock>784800000</outputPixelClock>
…
```

The frame width, height, blanking information, and sensor output pixel clock rate are provided by the sensor vendor for each sensor mode. The blanking information provided by sensor vendors may not be the minimum blanking period, but the average blanking period. To calculate the IFE clock rate optimally and correctly, we cannot use the average blanking period as this can result in CAMIF overflows. Instead, the smallest horizontal and vertical blanking periods for a given frame must be specified.

- **Configuring the correct blanking values**

Frequently, the sensor vendor does not provide the smallest blanking period in their data sheets. In such cases, we can use the Spectra HW's ability to measure the smallest horizontal interval (HBI) and the smallest vertical interval (VBI) to get the correct values. To measure these values, enter the following commands:

➲ **NOTE:** Some baselines use **tfe* instead of **ife* as the node name.

```
adb shell "echo 0x80 > /sys/kernel/debug/camera_ife/ife_csid_debug"
adb shell "echo 0x1 > /sys/kernel/debug/camera_ife/ife_camif_debug"
adb logcat -b all -c && adb logcat -b kernel > kmd.log
```

The following is a sample log collected from a SM8150 based device showing HBI and VBI data:

```
12-31 20:07:12.806 0 0 E I[0:  AudioRecord:10818] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2256 Resource 4 HBI: 0x50d0362
12-31 20:07:12.806 0 0 E I[0:  AudioRecord:10818] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2258 Resource 4 VBI: 0x416471
12-31 20:07:12.839 0 0 E I[0:  SyncManager_0: 926] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2256 Resource 4 HBI: 0x50c0361
12-31 20:07:12.839 0 0 E I[0:  SyncManager_0: 926] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2258 Resource 4 VBI: 0x400cb1
12-31 20:07:12.873 0 0 E I[0:  logd.writer:   552] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2256 Resource 4 HBI: 0x50c0361
12-31 20:07:12.873 0 0 E I[0:  logd.writer:   552] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2258 Resource 4 VBI: 0x416471
12-31 20:07:12.906 0 0 E I[0:provider@2.4-se: 2182] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2256 Resource 4 HBI: 0x50c0361
12-31 20:07:12.906 0 0 E I[0:provider@2.4-se: 2182] CAM_INFO: CAM-ISP:
cam_ife_csid_get_hbi_vbi: 2258 Resource 4 VBI: 0x400cb2
```

As seen in the above example, there may be some small variation in the HBI and VBI values. It is very important to always use the smallest values observed over many seconds. Also note that this measurement must be done with the highest fps supported by the sensor each mode. To ensure highest fps, make sure device is being tested at around 1000 lux and verify that the highest fps is indeed being achieved. The Spectra HW reports the HBI and VBI intervals in terms of CSID clock rate.

SM8150: for the HBI, only bits 0 to 11 should be used to extract the minimum HBI. According to the log, the minimum HBI is 0x361. We need to convert this to the sensor's output pixel clock domain using this formula:

$$minHorizontalBanking = RoundUp(\frac{OutputPixClkRate * csidHBIcycles}{CSIDclockRate})$$

- *Output*PixClkRate is the sensor's output pixel clock rate in Hz.
- csidHBIcycles is the HBI value as reported by the Spectra HW. In the above logs, this value is 0x361.
- CSIDclockRate is the clock rate of the CSID block being used for this use case. This can be read using these commands while the use case is active:

```
adb shell cat /d/clk/cam_cc_ife_0_csid_clk/clk_measure
adb shell cat /d/clk/cam_cc_ife_1_csid_clk/clk_measure
```

The correct CSID clock rate should be used based on whether IFE_0 or IFE_1 is being used for the use case.

To calculate the minimum VBI, use the following formula:

$$minCSIDVerticalBanking = \frac{csidVBIcycles}{Ceiling((outputWidthPixels + csidHBIcycles), 32)}$$

minCSIDVerticalBlanking is in CSID clock domain. We need to convert this to the sensor's output pixel clock domain:

$$minVerticalBanking = RoundUp(\frac{OutputPixClkRate * minCSIDVerticalBlanking}{CSIDclockRate})$$

SM8250 IFE HW behavior is very similar to SM8150. However, it is capable of processing two pixels for every clock cycle. The IFE driver on SM8250 will account for increased throughput. So, the procedure for calculating *minHorizontalBlanking* and *minVerticalBlanking* are identical to the section above.

Check one of the following examples:

- CSIDclockRate = 399997872(400Mhz)
- Sensor Op=577650000 = 577Mhz
- csid_HBI_cycles = 0x5420129
- bit0 to bit11-> 0x129 = 297
- minHorizontalBlanking = (577*297)/400=429
- minVerticalBlanking = (OutputPixClkRate * csidVBIcycle) / (Ceiling ((outputWidthPixel + csidHBIcycles), 32) * CSIDclockRate)
- csid_VBIcycles=0x2518a0 = 2431136
- sensorOutputWidth = 4000
- minCSIDVerticlBlanking=2431136/(4000+429) = 549
- minVerticalBlanking=(577 * 549)/ 400 = 791

- **Frequently asked questions**

After setting the minHorizontalBlanking and minVerticalBlankingbased on instruction in preceeding sections of this document, the IFE clock rate seems to be too high. Can the IFE clock rate be optimized?

The output pixel clock rate utilized by the vendor may be higher than the data rate needed to transfer pixel data alone. The reasons for this are only known to the sensor vendor. A somewhat crude and risky way to check if the IFE can operate at a lower rate is by checking if it still works by forcing a lower clock rate than the value calculated based on minHorizontalBlanking and minVerticalBlanking. This can be done by using the setting ifeClockFrequencyMHz:

```
# First determine available rates (shown in Hz)
adb root && adb wait-for-device remount && adb wait-for-device adb shell cat
/d/clk/cam_cc_ife_0_clk_src/clk_list_rates
# Sample output on SM8150 when the above command is run: 400000000
558000000
637000000
847000000
950000000
# To force the clock to 558MHz, do this:
adb shell "echo ifeClockFrequencyMHz=558 >> /vendor/etc/camera/
camxoverridesettings.txt" adb reboot
```

Now try to run the camera with the sensor mode required. If CAMIF violation is reported in the KMD log (sample log below), then the clock rate is too low and a higher clock rate should be used.

```
12-14 03:14:01.854
IPP fifo over flow   0    0    E CAM_ERR : CAM-ISP: cam_ife_csid_irq: 3188 CSID:1
12-14 03:14:01.887   0    0    E CAM_ERR : CAM-ISP: cam_vfe_irq_err_top_half: 176
Encountered Error:   vfe:0:         Irq_status0=0x0 Status1=0x80
12-14 03:14:01.898   0    0    E CAM_ERR : CAM-ISP: cam_vfe_irq_err_top_half: 179
Stopping further IRQ processing from this HW index=0
12-14 03:14:01.909 0 0 I CAM_INFO: CAM-ISP: cam_vfe_irq_err_top_half: 213 Violation status
= 1
```

```
12-14 03:14:01.916 0 0 E CAM_ERR : CAM-ISP:
cam_vfe_bus_error_irq_top_half: 2495 Bus Err IRQ
```

If for example, 558MHz works and the calculated clock rate is 600MHz, we can try lowering the minHorizontalBlanking and minVerticalBlanking so that calculated rate matches 558MHz using the formulas above.

3.2.5. Power regulator configuration

The type of voltage to be applied is specified in driver XML for a given sensor.

Supported values:

- MCLK – mclock
- VANA – Analog supply voltage
- VDIG – Digital supply voltage
- VIO – I/O supply voltage
- VAF – Actuator supply voltage
- RESET – Reset GPIO
- STANDBY – Standby GPIO

The values map to specific voltage regulators using the DTSI sensor node.

- **PMIC-based sensor nodes**

For example, VIO maps to cam_vio-supply, VANA maps to cam_vana-supply, and MCLK maps to cam_clk-supply.

```
cam_vio-supply = <&pm8998_lvs1>; cam_vana-supply = <&pmi8998_bob>; cam_vdig-supply =
<&camera_rear_ldo>; cam_clk-supply = <&titan_top_gdsc>;
```

- **GPIO-based sensor nodes**

```
pinctrl-names = "cam_default", "cam_suspend"; pinctrl-0 = <&cam_sensor_mclk0_active
&cam_sensor_rear_active>; pinctrl-1 = <&cam_sensor_mclk0_suspend
&cam_sensor_rear_suspend>; gpios = <&tlmm 13 0>,
<&tlmm 80 0>,
<&tlmm 79 0>;
gpio-reset = <1>;
gpio-vana = <2>;
gpio-req-tbl-num = <0 1 2>;
gpio-req-tbl-flags = <1 0 0>;
gpio-req-tbl-label = "CAMIF_MCLK0","CAM_RESET0","CAM_VANA";
```

- **Regulator-based sensor nodes**

```
regulator-names = "cam_vio", "cam_vana", "cam_vdig","cam_clk"; rgltr-cntrl-support;
rgltr-min-voltage = <0 3312000 1050000 0>;
rgltr-max-voltage = <0 3600000 1050000 0>;
rgltr-load-current = <0 80000 105000 0>;
```

Customers can configure different power regulators and supply by using the DTSI node parameters.

3.2.6. Clock configuration

In the DTS file, for each sensor node, the customer can configure clock source as follows:

```
clocks = <&clock_camcc CAM_CC_MCLK0_CLK>; clock-names = "cam_clk";
clock-cntl-level = "turbo"; clock-rates = <24000000>;
```

The order of the lists in the two properties is important. The nth clock-name corresponds to the nth entry in the clock property. The customer does not need to change the list order, as it is parsed in the clock framework.

Support is provided for up to four CAM_MCLK running at 19.2 MHz by default.

- CAM_CLK peak-to-peak jitter < 400 ps with default frequency.
- Recommended CAM_MCLK for SM8150 is 19.2 MHz.
- Recommended CAM_MCLK for SDM845/SDM670/SDM710 is 24 MHz.
- Discuss with Qualcomm Customer Engineering before using any CAM_MCLK frequency other than 19.2 MHz to investigate if jitter is acceptable for such case.

3.3. Sensor library configuration

Sensor infrastructure provides an interface for OEM to write their own gain/exposure update functions. With this change, each sensor can have its own custom library, which implements the API exposed in camxsensordriverAPI.h. After the API methods are implemented, generate the library with the name *com.<vendor name>.sensor.<sensor name>.so*. This file should also be included in the *product.mk* file with the other library files that need to be generated.

Generated library is available at */vendor/lib64/camera/* (for LA) or */usr/lib64/camera/* (for LE).

Once the library with the specified syntax name in the specified path is available, sensor software module loads the binary and accesses the API methods to use for exposure-related functionality.

Sample library implementation is available for imx318 sensor at: *proprietary/chi-cdk/oem/qcom/sensor/imx318/*.

This infrastructure also allows data published using the vendor tag *org.quic.camera2.sensor_register_control* to read and pass it to sensor library as *SensorFillExposureData->additionalInfo*.

Library functions

It is possible to define an addition to the sensor driver using a library. This library can only contain gain-specific functions with the following definitions:

```
static double RegisterToRealGain(unsigned int regGain); static unsigned int
RealToRegisterGain(double realGain);
```

Examples:

```
static double RegisterToRealGain(unsigned int regGain){ double realGain;
if(regGain > SENSOR_MAkX_AGAIN_REG_VAL){ regGain = IMX318_MAX_AGAIN_REG_VAL;
}
realGain = 512.0 / (512.0 - regGain); return realGain;
}
static unsigned int RealToRegisterGain(double realGain){ unsigned int regGain = 0;
if (realGain < IMX318_MIN_AGAIN){ realGain = SENSOR_MIN_AGAIN;
}
regGain = (unsigned int)(512.0 - (512.0 / realGain)); return regGain;
}
void GetSensorLibraryAPIs(SensorLibraryAPI* pSensorLibraryAPI){ pSensorLibraryAPI-
>pRealToRegisterGain = RealToRegisterGain; pSensorLibraryAPI->pRegisterToRealGain =
RegisterToRealGain;
}
```

3.4. Blanking requirements

Refer to th3e following blanking requirements for ISP:

- Minimum horizontal blanking – 64
- Minimum vertical blanking – 32

Chapter 4. Actuator Bringup Guidelines

4.1. Actuator software configuration

4.1.1. Actuator specific XML

Every actuator has an associated configuration XML file that is used to define features such as, but not limited to, power settings, code step, and initialization settings.

Entire settings are enclosed in the < *actuatorDriver*></ *actuatorDriver*> node of this XML.

4.1.2. Actuator information node

This node contains the information related to the actuator driver configuration parameters.

Table 4-1. Actuator slave information

Field	Description
slaveInfo	Master node that stores slave information used to communicate with actuator.
actuatorName	Name of the actuator.
slaveAddress	Slave address used to talk to the actuator.
i2cFrequencyMode	I2C frequency mode of slave. Example: FAST Supported modes: • STANDARD (100 kHz) • FAST (400 kHz) • FAST_PLUS (1 MHz) • CUSTOM (custom frequency in DTSI)
actuatorType	Supported actuator types: • VCM • BIVCM
dataBitWidth	Data width in bits.
powerUpSequence	This node contains the power-up configuration sequence required to control the power to the device while closing it. Example: <powerDownSequence> <powerSetting> <configType>MCLK</configType> <configValue>0</configValue> <delayMs>0</delayMs> </powerSetting> </powerDownSequence>
powerSetting	This node contains power configuration type, value, and delay.
configType	Power configuration type. Example: MCLK Supported types are: • MCLK • VANA • VDIG • VIO • VAF • RESET • STANDBY

Field	Description
configValue	Power configuration type. Example: 24000000
delayMs	Delay in milliseconds. Example: 1
powerDownSequence	This node contains the power-down configuration sequence required to control the power to the device while closing it Example: <powerDownSequence> <powerSetting> <configType>RESET</configType> <configValue>0</configValue> <delayMs>0</delayMs> </powerSetting> </powerDownSequence>

Table 4-2. Actuator register information

Field	Description
registerConfig	This node holds sequence of register configurations.
registerParam	Actuator register parameter configuration.
regAddrType	Register address type/ data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3-byte address • 4 = Address type max
regDataType	Register data type / data size in bytes. Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
registerAddr	Register address that is accessed.
registerData	Register data to be programmed.
operation	Actuator operation to be performed on the register. Supported values: • WRITE_HW_DAMP • WRITE_DAC • WRITE • WRITE_DIR_REG • POLL • READ_WRITE
delayUs	Delay in micro seconds
hwMask	Hardware mask
hwShift	Number of bits to shift for the hardware
dataShift	Number of bits to shift for data

Table 4-3. Actuator init information

Field	Description
initSettings	Sequence of register settings to configure the device.
regSetting	Register setting configuration.
registerAddr	Register address that is accessed.
registerData	Data to be processed, read from, or written into the address.
regAddrType	Register address type/ data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3‑byte address • 4 = Address type max
regDataType	Register data type / data size in bytes. Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
Operation	Operation to be performed on the register. Supported values: • READ • WRITE • POLL
delayUs	Delay in micro seconds.

Table 4-4. Actuator tuning information

Field	Description
tunedParams	Actuator tuning parameters.
initialCode	Initial DAC value.
regionParams	Actuator region parameters for all regions.
region	Actuator region parameters for single region.
macroStepBoundary	Macro step boundary in the near/forward direction.
infinityStepBoundary	Infinity step boundary in the far/backward direction.
codePerStep	Code per step.
qValue	qvalue to convert float/double to integer format.
forwardDamping	Actuator scenario ringing and damping information in forward/near direction.
backwardDamping	Actuator scenario ringing and damping information in backward/far direction.
ringingScenario	Actuator ringing scenario value.
scenarioDampingParams	Actuator damping parameters for all scenarios element for dampingStep.
scenario	Actuator damping parameters for all regions element for dampingStep.
region	Actuator damping parameters for a single region.
dampingStep	Actuator damping step.
dampingDelayUs	Actuator damping delay in micro seconds is applied after programming damping step.
hwParams	Actuator hardware parameters.

Table 4-5. Actuator regulator

Field	Description
compatible	Data type that this node corresponds to.
reg	Points to the node ID.
regulator-name	LDO regulator name.
regulator-min- microvolt	Minimum voltage for this LDO.
regulator-max- microvolt	Maximum voltage for this LDO.
regulator-enable- ramp-delay	The time taken, in microseconds, for the supply rail to reach the target voltage, ± whatever tolerance the board design requires. This property describes the total system ramp time required due to the combination of internal ramping of the regulator itself and board design issues such as, trace capacitance and load on the supply.
enable-active- high	Enable with active high.
gpio	GPIO to be used with this node.
vin-supply	Input voltage supply node name.

4.2. Actuator hardware configuration

4.2.1. Kernel hardware configuration

Depending on the LED flash hardware, OEMs can decide which type of interface driver to configure. Some LED Flash hardware requires a power supply at input to turn it on or off. For such LED Flash hardware, OEMs can use a PMIC-based LED Flash driver to supply current or power from the PMIC IC. This driver is simple and just calls PMICAPIs to control the current or power level for different Flash states. Other LED Flash hardware is programmed with register settings to turn it on or off. For that hardware, OEMs can use either QUP or I2C-based LED Flash drivers.

The node entry in the device tree file changes based on the type of LED Flash driver – PMIC-based, I2C-based, or CCI-based.

For more details and explanation of each field in the device tree file, refer to *kernel/msm-4.19/Documentation/devicetree/bindings/media/video/msm-camera-flash.txt* or *vendor/qcom/proprietary/camera-devicetree/bindings/msm-camera-flash.txt* (for relatively new baselines).

4.2.2. Actuator node structure

Different actuator node structures are required for different interfaces. There is no support for SPI interface for actuator.

Table 4-6. SOC-based actuator driver

Field	Description
compatible	Data type that this node corresponds to.
reg	Should contain the I2C slave address of the actuator and length of data field, which is 0x0.
regulator-name	A string used as a descriptive name for regulator outputs.
regulator-min- microvolt	Smallest voltage to set.
regulator-max- microvolt	Largest voltage to set.
regulator-enable- ramp-delay	Ramp delay for regulator.

Field	Description
gpio	Contains phandle to GPIO controller node and array of #gpio-cells specifying a particular GPIO.
vin-supply	phandle to a regulator that powers this actuator.

For example:

```
actuator_regulator: gpio-regulator@0 { compatible = "regulator-fixed"; reg = <0x00 0x00>;
regulator-name = "actuator_regulator"; regulator-min-microvolt = <2800000>;
regulator-max-microvolt = <2800000>;
regulator-enable-ramp-delay = <100>; enable-active-high;
gpio = <&tlmm 27 0>;
vin-supply = <&pmi8998_bob>;
};
```

Table 4-7. CCI-based actuator driver

Field	Description
cell-index	Contains a unique identifier to differentiate between multiple actuators.
reg	Corresponds to the cell-index of the node.
compatible	Data type to which this node corresponds to.
qcom,cci-master	CCI node that drives this node.
cam_vaf-supply	Contains the regulator from which AF voltage is supplied.
qcom,cam-vreg- name	Contains the names of all regulators needed by this actuator.
qcom,cam-vreg- min-voltage	Contains the minimum voltage level in microvolts for regulators mentioned in qcom, cam-vreg-name property (in the same order).
qcom,cam-vreg- max-voltage	Contains the maximum voltage level in microvolts for regulators mentioned in qcom, cam-vreg-name property (in the same order).
qcom,cam-vreg-op- mode	Contains the maximum current in µA that is required from the regulators mentioned in the qcom, cam-vreg-name property (in the same order).

For example:

```
actuator_front: qcom,actuator@1 { cell-index = <1>;
reg = <0x1>;
compatible = "qcom,actuator"; qcom,cci-master = <1>;
cam_vaf-supply = <&actuator_regulator>; qcom,cam-vreg-name = "cam_vaf";
qcom,cam-vreg-min-voltage = <2800000>;
qcom,cam-vreg-max-voltage = <2800000>;
qcom,cam-vreg-op-mode = <0>;
};
```

Chapter 5. Flash Bringup Guidelines

5.1. Flash software configuration

5.1.1. Flash-specific XML

Flash has an associated configuration XML file used to define features such as, but not limited to the type of flash, max current, max duration, and flash related information.

Entire settings are enclosed in the < *flashDriverData* ></ *flashDriverData* > node of this XML.

5.1.2. Flash information node

Table 5-1. Flash driver data

Field	Description
flashDriverData	This node has all the information regarding the flash type and its parameters.
flashName	Name of the flash.
flashDriverType	Flash driver type. Supported values: • PMIC • I^2C
powerUpSequence	Sequence to power up the flash.
powerDownSequence	Sequence to power down the flash.
i2cInfo	Settings for an I^2C-based flash (see Table 5-2).
flashInformation	Flash-related trigger information (see Table 5-3).

Table 5-2. Flash I^2C information

Field	Description
slaveAddress	8-bit or 10-bit I^2C slave write address.
regAddrType	Register address type/data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3-byte address • 4 = Address type max
regDataType	Register data type or data size in bytes. Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
i2cFrequencyMode	I^2C Frequency mode of slave. Example: FAST Supported modes are: • STANDARD (100 kHz) • FAST (400 kHz) • FAST_PLUS (1 MHz) • CUSTOM (custom frequency in DTSI)
flashInitSettings	Sequence of I^2C register settings to initialize flash.

Field	Description
flashOffSettings	Sequence of I^{2}C Register settings to turn off flash.
flashLowSettings	Sequence of I^{2}C Register settings to turn on low flash.
flashHighSettings	Sequence of I^{2}C Register settings to turn on high flash.
regSetting	This node holds one register configuration and forms a unit of large flash-register-settings sequence.
registerAddr	Register address that is accessed.
registerData	• If the operation is WRITE, registerData is the data value to be written into the specified register address. • If the operation is READ, registerData is the number of bytes to be read from the specified register address.
regAddrType	Type of register address.
regDataType	Type of register data.
operation	Type of operation Supported values: • WRITE • READ • POLL
delayUs	Delay in microseconds. If not explicitly specified, the delay is zero.

Table 5-3. Flash trigger information

Field	Description
triggerInfo	Flash trigger information.
maxFlashCurrent	Maximum flash current of the trigger in mA.
maxTorchCurrent	Maximum Torch current of the trigger in mA.
maxFlashDuration	Maximum flash duration of the trigger in ms.

5.2. Flash hardware configuration

5.2.1. Kernel hardware configuration

Depending on the LED flash hardware, OEMs can decide which type of interface driver to configure. Some LED flash hardware requires a power supply at input to turn it on or off. For such LED flash hardware, OEMs use a PMIC-based LED flash driver to supply current or power from the PMIC IC. This driver is simple and just calls PMIC APIs to control the current and power level for different flash states. Other LED flash hardware is programmed with register settings to turn it on/off. For that hardware, OEMs can use either QUP or I2C-based LED flash drivers.

The node entry in the device tree file changes based on the type of LED Flash driver – PMIC-based, I2C-based, or CCI-based.

For more details and explanation of each field in the device tree file, refer to *kernel/msm-4.19/Documentation/devicetree/bindings/media/video/msm-camera-flash.txt* or *vendor/qcom/proprietary/camera-devicetree/bindings/msm-camera-flash.txt* (for relatively new baselines).

5.2.2. Flash node structure

Different flash structures are required for different interfaces. There is no support for SPI interface for flash.

Table 5-4. PMIC-based LED flash driver

Field	Description
cell-index	Specifies the node ID.
reg	Increments based on the cell index.
compatible	Data type that this node corresponds to.
flash-source	Should contain the array of phandles to flash source nodes.
torch-source	Should contain the phandle to Torch source node.
switch-source	Should contain the phandle to switch source node. Trigger dual led at same time to avoid sync issues.
status	Whether this node is enabled or disabled.

For example:

```
&soc {
led_flash_rear: qcom,camera-flash@0 {
cell-index = <0>;
reg = <0x00 0x00>;
compatible = "qcom,camera-flash";
flash-source = <&pmi8998_flash0 &pmi8998_flash1>;
torch-source = <&pmi8998_torch0 &pmi8998_torch1>;
switch-source = <&pmi8998_switch0>;
status = "ok";
};
};
```

Table 5-5. QUP/I2C-based LED flash driver

Field	Description
cell-index	Specifies the node ID.
reg	Increments based on the cell index.
qcom,slave-id	Slave address for the I^2C communication with flash hardware.
compatible	Data type to which the node corresponds to.
qcom,flash-name	Flash name.
qcom,flash-type	Flash type. Supported values: • LED flash • Strobe flash • Simple LED flash controlled by one GPIO
qcom,gpio-no-mux	Specifies the GPIO mux type. • 1 = GPIO mux is not available • 0 = Otherwise
gpios	Contains phandle to GPIO controller node and array of #gpio-cells specifying a GPIO (controller-specific).
qcom,gpio-flash-en	Contains the index to GPIO used by flash enable pin of flash.
qcom,gpio-flash-now	Contains the index to GPIO used by flash's "flash now" pin.
qcom,gpio-flash- reset	Contains the index to GPIO used by flash reset pin of flash.
qcom,gpio-req-tbl- num	Contains the index to GPIO-specific to this flash.

Field	Description
qcom,gpio-req-tbl- flags	Contains direction of GPIO present in qcom, gpio-req-tbl-num property (in the same order).
qcom,gpio-req-tbl- label	Contains name of GPIOs present in qcom, gpio-req-tbl-num property (in the same order).

For example:

```
&i2c {
led_flash0: qcom,led-flash@60 { cell-index = <0>;
reg = <0x60>;
qcom,slave-id = <0x60 0x00 0x0011>; compatible = "qcom,led-flash"; qcom,flash-name =
"adp1600"; qcom,flash-type = <1>;
qcom,gpio-no-mux = <0>;
gpios = <&msmgpio 18 0>,
<&msmgpio 19 0>;
qcom,gpio-flash-en = <0>;
qcom,gpio-flash-now = <1>;
qcom,gpio-req-tbl-num = <0 1>;
qcom,gpio-req-tbl-flags = <0 0>;
qcom,gpio-req-tbl-label = "FLASH_EN", "FLASH_NOW";
};
};
```

Table 5-6. CCI-based LED flash driver

Field	Description
cell-index	Specifies the node ID.
reg	Increments based on the cell index.
qcom, slave-id	Slave address for the I2C communication with flash hardware.
compatible	Data type to which this node corresponds to.
label	Contain unique flash name to differentiate from other flash.
qcom,flash-type	Supported flash type values: • LED flash • Strobe flash • Simple LED flash, controlled by one GPIO
pinctrl-names	Defines if a target uses pinctrl framework.
pinctrl-0	Pinctrl handle for pin0.
pinctrl-1	Pinctrl handle for pin1.
qcom,gpio-no-mux	Specifies the GPIO mux type. • 1 = GPIO mux is not available • 0 = Otherwise
gpios	Contain phandle to GPIO controller node and array of #gpio-cells specifying the GPIO (controller specific).
qcom,gpio-flash-en	Contains an index to GPIO used by flash enable pin of flash.
qcom,gpio-flash- now	Contains an index to GPIO used by flash now pin of flash.
qcom,gpio-flash- reset	Contains an index to GPIO used by flash reset pin of flash.
qcom,gpio-req-tbl- num	Contains an index to GPIOs specific to this flash.
qcom,gpio-req-tbl- flags	Contains the direction of GPIOs present in qcom, gpio-req-tbl-num property (in the same order).
qcom,gpio-req-tbl- label	Contains the name of GPIOs present in qcom, gpio-req-tbl-num property (in the same order).
qcom,cci-master	Contain I2C master ID to be used for this flash. Supported values: • 0 – MASTER_0 • 1 – MASTER_1

For example:

```
$cci {
led_flash0: qcom,led-flash@66 { cell-index = <0>;
reg = <0x66>;
qcom,slave-id = <0x66 0x00 0x0011>; compatible = "rohm-flash,bd7710"; label = "bd7710";
qcom,flash-type = <1>;
qcom,gpio-no-mux = <0>; qcom,enable_pinctrl;
pinctrl-names = "cam_flash_default", "cam_flash_suspend"; pinctrl-0 =
<&cam_sensor_flash_default>;
pinctrl-1 = <&cam_sensor_flash_sleep>; gpios = <&msm_gpio 36 0>,
<&msm_gpio 32 0>,
<&msm_gpio 31 0>;
qcom,gpio-flash-reset = <0>;
qcom,gpio-flash-en = <1>;
qcom,gpio-flash-now = <2>;
qcom,gpio-req-tbl-num = <0 1 2>;
qcom,gpio-req-tbl-flags = <0 0 0>; qcom,gpio-req-tbl-label = "FLASH_RST",
"FLASH_EN", "FLASH_NOW";
qcom,cci-master = <0>;
};
}
```

Chapter 6. EEPROM Bring-up Guidelines

Electrically erasable programmable read-only memory (EEPROM) is a non-volatile memory used to store the module calibration information. Module-to-module variations have an impact on the image quality of a camera system. The calibration data is used to make corrections for the module variations from a known reference (data from the golden module provided by the vendor).

Typical sources of module variation are:

- Lens placement accuracy (packaging or assembly tolerances)
- Color filter variations due to different batch in manufacturing
- IR (infrared) filter variations due to tolerance
- Electro-mechanical tolerances in autofocus actuator manufacturing
- Feature-specific calibration like PDAF, VHDR, dual camera

6.1. EEPROM software configuration

6.1.1. EEPROM-specific XML

EEPROM has an associated configuration XML file used to define calibration parameters such as PDAF, LSC, AF, dual camera, and WBC data.

The entire settings are enclosed in the *< EEPROMDriverData ></ EEPROMDriverData>* node of this XML.

6.1.2. EEPROM information node

- **slaveInfo node**

  slaveInfo contains the information that is used by the driver to power on/off the EEPROM device. Configuring this node is optional in kernel probe as the information is included in the kernel probe EEPROM device tree node.

Table 6-1. EEPROM slave information

Field	Description
slaveInfo	Contains the sensor slave information and power settings.
EEPROMName	Name of the EEPROM. Example: atmel_at24c32e
slaveAddress	8‑bit or 10‑bit slave address. Example: 0xa0
regAddrType	Register address/data size in bytes. Example 2: • 1 = Byte address • 2 = Word address • 3 = 3‑byte address • 4 = Address type max
regDataType	Register address/data size in bytes. Example 1: • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = Data type max

Field	Description
i2cFrequencyMode	I2C Frequency mode of slave. Example: FAST Supported modes: • STANDARD (100 kHz) • FAST (400 kHz) • FAST_PLUS (1 MHz) • CUSTOM (custom frequency in DTSI)

powerUpSequence	Contains the power-up configuration sequence required to control the power to the device while turning it on.
	powerSetting: Contains power configuration type, value, and delay. configType: Power configuration type
	Supported types:
	• MCLK
	• VANA
	• VDIG
	• VIO
	• VAF
	• RESET
	• STANDBY
	configValue: Value for the specified type or 0 to use default value delayMs: Delay in milliseconds.
	Example:
	`<powerUpSequence>`
	`<powerSetting>`
	`<configType>VIO</configType>`
	`<configValue>0</configValue>`
	`<delayMs>0</delayMs>`
	`</powerSetting>`
	`</powerUpSequence>`

powerDownSequence	Contains the power-down configuration sequence required to control the power to the device while turning it off. ➲ **NOTE:** Field descriptions are same as powerUpSequence.

- **MemoryMap node**

This memory map node contains the information that is needed by the driver to read the OTP data from EEPROM. Configuring this node is optional in kernel probe as the kernel probe EEPROM device tree node contains this information, but at minimum one set of nodes needs to be configured containing READ operation and total size of the data in registerData. To read total size of the data for multiple sets of regSetting nodes, information will be calculated as the sum off all the registerData values corresponding to the READ operation.

Table 6-2. EEPROM memoryMap

Field	Description
memoryMap	Contains regSetting node which contain details for read, write and poll operations. This can contain multiple sets of regSetting nodes.

regSetting	regSetting contains the slaveAddr, registerAddr, registerData, regAddrType, regDataType, operation, delayUS.
	Example:
	`<regSetting>`
	`<slaveAddr>0xa0</slaveAddr>`
	`<registerAddr>0x00</registerAddr>`
	`<registerData>700</registerData>`
	`<regAddrType >2</regAddrType>`
	`<regDataType>1</regDataType>`
	`<operation>READ</operation>`
	`<delayUs>0</delayUs>`
	`</regSetting>`
	`<regSetting>`
	`<slaveAddr>0xa1</slaveAddr>`
	`<registerAddr>0x00</registerAddr>`
	`<registerData>800</registerData>`
	`<regAddrType >2</regAddrType>`
	`<regDataType>1</regDataType>`
	`<operation>READ</operation>`
	`<delayUs>0</delayUs>`
	`</regSetting>`

slaveAddr	Slave address to communicate with the device. 8-bit or 10-bit slave address. Example: 0xa0
registerAddr	Register address that is accessed. Example: 0x00
registerData	Register data to be programmed/read. • For WRITE operation, registerData is the data value to be written into the specified register address. Example: 0x02 • For READ operation, registerData is the number of bytes to be read from the specified register address. Example: 1584 • For POLL operation, registerData is the value for which poll is being done. Example: 0x01
regAddrType	Register address type/data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3-byte address • 4 = Address type max
regDataType	Register data type/data size in bytes. Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
operation	Operation to be performed on EEPROM. Supported operations are: • READ • WRITE • POLL
delayUs	Delay in micro seconds. Example: 100

- **formatInfo node**

The format node contains the information needed to format the OTP data obtained from EEPROM. It is not necessary to have all the sub fields filled as each EEPROM can have some or all the type of the OTP data and only those, whose data is available can be configured. In the absence of format data information, raw (unformatted) data will be published to meta data pool and that can be used to format and obtain the required data by individual modules.

Refer to the requirements below for formatting the data buffer read from EEPROM:

- The position of each field also known as offset (with starting position reference as 0).
- Exact number of bits of data of that specific field, that is, size of the field represented in the form of mask and endianness to know the byte reading order.

In addition to the raw buffer formatting, this structure can also contain other constants that are not part of the buffer data to be configured.

Table 6-3. EEPROM format information

Field	Description
AF	Auto focus (AF) data information to format the AF OTP data.
autoFocusData	isAvailable – Indicates that AF data is available or not in the OTP data buffer and valid values are true/false. If this is false, all the remaining fields under AF node will be ignored. Example: true endianness – Type of the endianness. valid values are BIG or LITTLE Example: LITTLE
macro	offset – Position of the first byte of macro value in the buffer. Example: 22 mask – Represents the number of valid bits of the data read from offset Example: 0xFFF. This means 2 bytes of data to be read and in that 12 bits contain the valid data. Mask 0 indicates that particular data is not available in the buffer. The same explanation holds good for all other fields, which needs offset and mask information.
infinity	• offset – Position of the first byte of infinity value in the buffer. • mask – Represents the number of valid bits of the data.
hall	• offset – Position of the first byte of hall value in the buffer. • mask – Represents the number of valid bits of the data.
hallBias	• offset – Position of the first byte of hall bias value in the buffer • mask – Represents the number of valid bits of the data
hallRegisterAddr	Register address for hall and hall bias to be programmed. Example: 0x28
verticalMacro	• offset – Position of the first byte of vertical macro value in the buffer. • mask – Represents the number of valid bits of the data.
horizontalMacro	• offset – Position of the first byte of horizontal macro value in the buffer. • mask – Represents the number of valid bits of the data.
verticalInfinity	• offset – Position of the first byte of vertical infinity value in the buffer. • mask – Represents the number of valid bits of the data.
horizontalInfinity	• offset – Position of the first byte of horizontal infinity value in the buffer. • mask – Represents the number of valid bits of the data.

Field	Description
macroMargin	Margin to be extended towards macro region. Example: 0.3 that is, the new marco value is macro = macro + DACRange × 0.3 where, DACRange = macro - infinity
infinityMargin	Margin to be extended towards infinity region. Example: 0.15 that is, the new infinity value is infinity = infinity + DACRange × 0.15 where, DACRange = macro - infinity
WB	White balance (WB) data information to format WB data
WBData	isAvailable – Indicates that WB data is available or not in the OTP data buffer and valid values are true/false. If this is false, all the remaining fields under WB node will be ignored. Example: true endianness – Type of the endianness. valid values are BIG or LITTLE Example: LITTLE
dataType	Indicates whether the WB data is available in the form of RATIO or INDIVIDUAL. • If the data is available in the form of RATIO, then rOverGValue, bOverGValue, grOverGBValue will be configured. • If the data is available in the form of INDIVIDUAL, then rValue, grValue, bValue, gbValue will be configured.
lightInfo	Contains the type of the illuminant and corresponding color values based on dataType.
illuminantType	Type of the illuminant for which WB data is available. Supported illuminants are D65, TL84, A, D50, H.
rValue	• offset – Position of the first byte of rValue value in the buffer. • mask – Represents the number of valid bits of the data.
grValue	• offset – Position of the first byte of grValue value in the buffer. • mask – Represents the number of valid bits of the data.
bValue	• offset – Position of the first byte of bValue value in the buffer. • mask – Represents the number of valid bits of the data.
gbValue	• offset – Position of the first byte of gbValue value in the buffer. • mask – Represents the number of valid bits of the data.
rOverGValue	• offset – Position of the first byte of rOverGValue value in the buffer. • mask – Represents the number of valid bits of the data.
bOverGValue	• offset – Position of the first byte of bOverGValue value in the buffer. • mask – Represents the number of valid bits of the data.
grOverGBValue	• offset – Position of the first byte of grOverGBValue value in the buffer. • mask – Represents the number of valid bits of the data.
mirror	• offset – Position of the first byte of mirror value in the buffer. • mask – Represents the number of valid bits of the data.
flip	• offset – Position of the first byte of flip value in the buffer. • mask – Represents the number of valid bits of the data.
qValue	Factor to convert the obtained real values to required float values.
isInvertGROverGB	Set to true if GRoverGB needs to be inverted or else false.
LSC	LSC (Lens Shading Calibration) data needed to format LSC data.

Field	Description
LSCData	isAvailable – Indicates that LSC data is available or not in the OTP data buffer and valid values are true/false. If this is false, all the remaining fields under LSC node will be ignored. Example: true endianness – Type of the endianness. valid values are BIG or LITTLE Example: LITTLE
lightInfo	Contains the type of the illuminant and corresponding LSC gain values.
illuminantType	Type of the illuminant for which LSC data is available. Supported illuminants are D65, TL84, A, D50, H.
rGainMSB	• offset – Position of the MSB of the rGain value in the buffer. • mask – Represents the number of valid bits of the data.
rGainLSB	• offset – Position of the LSB of the rGain value in the buffer. • mask – Represents the number of valid bits of the data.
grGainMSB	• offset – Position of the MSB of the grGain value in the buffer. • mask – Represents the number of valid bits of the data.
grGainLSB	• offset – Position of the LSB of the grGain value in the buffer. • mask – Represents the number of valid bits of the data.
gbGainMSB	• offset – Position of the MSB of the gbGain value in the buffer. • mask – Represents the number of valid bits of the data.
gbGainLSB	• offset – Position of the LSB of the gbGain value in the buffer. • mask – Represents the number of valid bits of the data.
bGainMSB	• offset – Position of the MSB of the bGain value in the buffer. • mask – Represents the number of valid bits of the data.
bGainLSB	• offset – Position of the LSB of the bGain value in the buffer. • mask – Represents the number of valid bits of the data.
meshHWRollOffSize	Mesh hardware rolloff size.
rIncrement	Value to be incremented to find the next rGainMSB and rGainLSB value Example: 5 that is, if rGainMSB offset is 28, then the next rGainMSB offset will be 28+5 = 33. Same applies to other increments and if rGainLSB offset is 24, then the next rGainMSB offset will be 24+5 = 29. Same applies to other increments.
grIncrement	Value to be incremented to find next grGainMSB and grGainLSB values.
gbIncrement	Value to be incremented to find next gbGainMSB and gbGainLSB values.
bIncrement	Value to be incremented to find next bGainMSB and bGainLSB values.

6.2. EEPROM hardware configuration

6.2.1. Kernel hardware configuration

Refer to *kernel/msm-4.19/Documentation/devicetree/bindings/media/video/msm-cam-eeprom.txt* or *vendor/qcom/proprietary/camera-devicetree/bindings/msm-cam-eeprom.txt* (for relatively new baselines)for more details and explanation of each field in the device tree file.

6.2.2. EEPROM node structure

Different EEPROM structures are required for different interfaces. SPI interface is only supported for EEPROM. The hardware configuration of the EEPROM module is saved in the kernel DTSI files. See Table 3-25 for more details.

Chapter 7. PDAF Bring-up Guidelines

PDAF software configuration includes:

- Updating PDAF driver XML (see PDAF-specific XML and PDAF configuration data node)
- Updating sensor driver XML (see PDAF-driver example)
- Updating module configuration (based on Table 3-21)

7.1. PDAF-specific XML

Every PDAF driver has an associated configuration XML file that defines features such as, but not limited to buffer format, buffer pattern, block pattern, and native format.

Entire settings are enclosed in the *< PDConfigData></ PDConfigData>* node of this XML.

7.2. PDAF configuration data node

This node contains the information related to the PDAF driver configuration parameters. This information is common to the PDAF regardless of the PDAF supported sensor resolution being picked.

Table 7-1. Common PDAF information node

Field	Description
PDAFName	This is the string that contains the name of this PDAF driver.
PDOrientation	PD sensor orientation. Supported orientations are: • DEFAULT • MIRROR • FLIP • MIRRORANDFLIP
PDBlackLevel	Sensor black level information.
PDDefocusConfidenceThreshold	PD defocus confidence threshold.

The following node has the information about PDAF sensor native pattern to indicate the location ofPD pixels in the sensor array. Each node corresponds to one specific sensor resolution that supports PDAF nodes.

Table 7-2. PDAF mode information node

Field	Description
PDSensorNativePatternInfo	PDAF Native Pattern for this mode.
PDBufferBlockPatternInfo	PDAF Buffer block pattern for this mode.
PDType	PDAF type for this mode .PDAF type. Supported PDAF types are: • PDType2 • PDType3 • PDType2PD • PDTypeQPD

Field	Description

PDModeKey	This should be mapped to valid PDAF Mode Key. Each PDAF Mode is given user defined mode key to link sensor mode to correct PDAF mode.
	For example:
	`Sensor Driver`
	`Sensor ResolutionId  PDModeKey`
	`0  10`
	`PDAF Driver`
	`PDAF ResolutionId  PDModeKey`
	`1  10`
	In the example above, PDAF mode 1 corresponds to Sensor mode 0.
	Link between these modes is established using PDModeKey equal to 10.
	Max value of PDModeKey can be 99.

PDAFLibraryName	PD lib name to use.
PDSensorPDStatsFormat	Sensor PD stats format for Type1 sensor.
PDSensorOutputFormat	Sensor Output Buffer Format for Type1 to be used for buffer allocation. Default is UNPACKED16
lcrPDOffsetCorrection	LCR PD Offset Correction
PDPixelOrderType	Pixel order type. Supported order types: • LEFTTORIGHT • RIGHTTOLEFT For 2PD sensors: this field identifies the PD pixels order when all the pixels are PD pixels.
PDOffsetCorrection	This is a floating-point value that specifies the PD Offset correction.

Table 7-3. PDAF sensor native pattern node

Field	Description
PDBlockCountHorizontal	This holds the number of PD pixels in horizontal direction.
PDBlockCountVertical	This holds the number of PD pixels in vertical direction
PDBlockPattern	PD block pattern. See Table 7-5
PDCropRegion	In-sensor cropped region Table 7-8.
PDDownscaleFactorHorizontal	Horizontal downscale factor.
PDDownscaleFactorVertical	Vertical downscale factor.

The following node has the information about the PDAF buffer pattern that is streamed out of sensor. Each node corresponds to one specific sensor resolution that supports PDAF with maximum 14 supported nodes.

Table 7-4. PDAF buffer block pattern node

Field	Description
PDStride	This is the number of pixels in the PD stats buffer after which there is a jump to a newline.
PDBufferFormat	This is the data type of output stats buffer. Supported values: • MIPI8 • MIPI10 • MIPI12 • PACKED10 • UNPACKED16
PDBlockPattern	PDAF block pattern. See Table 7-5

Table 7-5. PDAF block pattern information

Field	Description
PDPixelCount	This is PDAF pixel number inside a window.
PDPixelCoordinates	Pixel 2D position, left_pixel, right_pixel. This should not contain the offsets see Table 7-6.
PDBlockDimensions	PD Block Dimensions see Table 7-7.
PDOffsetHorizontal	Horizontal offset that should be added back for correct skip pattern.
PDOffsetVertical	Vertical offset that should be added back for correct skip pattern.

The following node has the information about PDAF pixel coordinate in the PDAF block. A maximum of 256 PDAF pixel coordinate units are allowed in a block pattern.

Table 7-6. PDAF pixel coordinates

Field	Description
PDXCoordinate	This is PDAF pixel number inside a window.
PDYCoordinate	Pixel 2D position, left_pixel, right_pixel.
PDPixelShieldInformation	Pixel shield information. Supported shield patterns are: • LEFTDIODE • RIGHTDIODE • LEFTSHIELDED • RIGHTSHIELDED • UPDIODE • DOWNDIODE • UPSHIELDED • DOWNSHIELDED • TOPLEFTDIODE

Table 7-7. PDAF block dimensions

Field	Description
width	Width of the PD block dimension.
height	Height of the PD block dimension.

Table 7-8. PDAF crop region information

Field	Description
width	Width of the PD block dimension.
height	Height of the PD block dimension.
xStart	Starting coordinate of in-sensor crop width
yStart	Starting coordinate of in-sensor crop height

7.3. PDAF horizontal and vertical scale factors

For 2PD sensors, the vertical and horizontal downscale factors need to be calculated as follows:

- In full resolution, there is no binning or scaling. Therefore, two pixels in horizontal direction will bin to one left and one right PDAF pixel. Horizontal scale factor is 2 and vertical downscale factor is 4.
- For 1080p, since there is a downscale of every two pixels in sensor, both horizontal scale factor and vertical scale factor will be 4.
- For HDR mode, four pixels (horizontal direction) will bin to 1 left long, 1 left short, one right long, one right short resulting in horizontal scale factor of 4 and vertical scale factor of 4.

7.4. PDAF-driver example

Example

```
<!-- RES 1 4624x3472 @30fps -->
 <PDModeInfo>
  <PDModeKey>1</PDModeKey>
  <PDType>PDType2</PDType>
  <PDAFLibraryName>com.qti.stats.pdlib</PDAFLibraryName>
  <!--Sensor Native pattern infomation
   element for pdBlockCountHorizontal
   element for pdBlockCountVertical
   element for PD Block Pattern
   element for PD Crop Region
   element for downscale factor horizontal
   element for downscale factor vertical -->
  <PDSensorNativePatternInfo>
   <PDNativeBufferFormat>MIPI10</PDNativeBufferFormat>
   <!--Number of PD blocks in X direction
    2PD: PD Image Width -->
   <PDBlockCountHorizontal>574</PDBlockCountHorizontal>
   <!--Number of PD blocks in Y direction
    2PD: PD Image Height -->
   <PDBlockCountVertical>430</PDBlockCountVertical>
   <!--Block Pattern details of one block
    PDPixelCount: PDAF pixel number inside a window
    PDPixelCoordinates: Pixel 2D pos, left_pixel,right_pixel
    Should not contain the offset.
    Offset should add back for correct skip pattern.
    PD Block Pattern -->
   <PDBlockPattern>
    <PDPixelCount>8</PDPixelCount>
    <!--One pixel coordinate in a block -->
    <PDPixelCoordinates>
     <PDXCoordinate>19</PDXCoordinate>
     <PDYCoordinate>17</PDYCoordinate>
     <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
    </PDPixelCoordinates>
    <!--One pixel coordinate in a block -->
    <PDPixelCoordinates>
     <PDXCoordinate>20</PDXCoordinate>
     <PDYCoordinate>17</PDYCoordinate>
     <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
    </PDPixelCoordinates>
    <!--One pixel coordinate in a block -->
    <PDPixelCoordinates>
     <PDXCoordinate>17</PDXCoordinate>
     <PDYCoordinate>19</PDYCoordinate>
     <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
    </PDPixelCoordinates>
    <!--One pixel coordinate in a block -->
    <PDPixelCoordinates>
     <PDXCoordinate>18</PDXCoordinate>
     <PDYCoordinate>19</PDYCoordinate>
     <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
```

```
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
   <PDPixelCoordinates>
   <PDXCoordinate>21</PDXCoordinate>
   <PDYCoordinate>21</PDYCoordinate>
   <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>22</PDXCoordinate>
   <PDYCoordinate>21</PDYCoordinate>
   <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
  <PDXCoordinate>23</PDXCoordinate>
  <PDYCoordinate>23</PDYCoordinate>
  <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>24</PDXCoordinate>
   <PDYCoordinate>23</PDYCoordinate>
   <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--Width and height of the frame or subframe -->
  <PDBlockDimensions>
   <width>8</width>
   <height>8</height>
  </PDBlockDimensions>
  <PDOffsetHorizontal>17</PDOffsetHorizontal>
  <PDOffsetVertical>16</PDOffsetVertical>
 </PDBlockPattern>
 <!--Frame dimension: contains xStart, yStart, width and height      In-sensor Cropped
region -->
 <PDCropRegion>
  <xStart>0</xStart>
  <yStart>0</yStart>
  <width>4624</width>
  <height>3472</height>
 </PDCropRegion>
 <!--Horizontal Downscale factor -->
 <PDDownscaleFactorHorizontal>1</PDDownscaleFactorHorizontal>
 <!--Vertical Downscale factor -->
 <PDDownscaleFactorVertical>1</PDDownscaleFactorVertical>
</PDSensorNativePatternInfo>
<!--Block Pattern Info about all the blocks
 PDStride: This is the number of pixels in the PD stats buffer
 after which there is a jump to a new line.
 PDBufferDataFormat: This is the data type of output stats buffer. -->
<PDBufferBlockPatternInfo>
 <PDStride>2304</PDStride>
 <!--PDAF Buffer Data Format
 MIPI10:  compressed, [9:2] [9:2] [9:2] [9:2] [1:0][1:0][1:0][1:0]
 PACKED10: Q10 format -->
 <PDBufferFormat>UNPACKED16</PDBufferFormat>
 <!--Block Pattern details of one block
 PDPixelCount: PDAF pixel number inside a window
 PDPixelCoordinates: Pixel 2D pos, left_pixel,right_pixel
 Should not contain the offset.
 Offset should add back for correct skip pattern. -->
 <PDBlockPattern>
  <PDPixelCount>8</PDPixelCount>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>0</PDXCoordinate>
   <PDYCoordinate>0</PDYCoordinate>
   <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
```

```
   <PDXCoordinate>1</PDXCoordinate>
   <PDYCoordinate>0</PDYCoordinate>
   <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>0</PDXCoordinate>
   <PDYCoordinate>1</PDYCoordinate>
   <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>1</PDXCoordinate>
   <PDYCoordinate>1</PDYCoordinate>
   <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>0</PDXCoordinate>
   <PDYCoordinate>2</PDYCoordinate>
   <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>1</PDXCoordinate>
   <PDYCoordinate>2</PDYCoordinate>
   <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>0</PDXCoordinate>
   <PDYCoordinate>3</PDYCoordinate>
   <PDPixelShieldInformation>RIGHTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--One pixel coordinate in a block -->
  <PDPixelCoordinates>
   <PDXCoordinate>1</PDXCoordinate>
   <PDYCoordinate>3</PDYCoordinate>
   <PDPixelShieldInformation>LEFTSHIELDED</PDPixelShieldInformation>
  </PDPixelCoordinates>
  <!--Width and height of the frame or subframe -->
  <PDBlockDimensions>
   <width>2</width>
   <height>4</height>
  </PDBlockDimensions>
  <PDOffsetHorizontal>0</PDOffsetHorizontal>
  <PDOffsetVertical>0</PDOffsetVertical>
 </PDBlockPattern>
</PDBufferBlockPatternInfo>
</PDModeInfo
```

For QPD configuration:

```
<PDModeInfo>
 <PDModeKey>0</PDModeKey>
 <PDType>PDTypeQPD</PDType>
 <PDAFLibraryName>com.qti.stats.pdlib</PDAFLibraryName>
 <PDPixelOrderType>LEFTTORIGHT</PDPixelOrderType>
 <!--Sensor Native pattern information
  element for pdBlockCountHorizontal
  element for pdBlockCountVertical
  element for PD Block Pattern
  element for PD Crop Region
  element for downscale factor horizontal
  element for downscale factor vertical -->
 <PDSensorNativePatternInfo>
  <PDNativeBufferFormat>MIPI10</PDNativeBufferFormat>
  <!--Number of PD blocks in X direction
   2PD: PD Image Width -->
  <PDBlockCountHorizontal>4096</PDBlockCountHorizontal>
  <!--Number of PD blocks in Y direction2
   PD: PD Image Height -->
```

```
 <PDBlockCountVertical>3072</PDBlockCountVertical>
 <!--Frame dimension: contains xStart, yStart, width and height
 In-sensor Cropped region -->
 <PDCropRegion>
  <xStart>0</xStart>
  <yStart>0</yStart>
  <width>8192</width>
  <height>6144</height>
 </PDCropRegion>
 <!--Horizontal Downscale factor -->
 <PDDownscaleFactorHorizontal>4</PDDownscaleFactorHorizontal>
 <!--Vertical Downscale factor -->
 <PDDownscaleFactorVertical>4</PDDownscaleFactorVertical>
</PDSensorNativePatternInfo>
<!--Block Pattern Info about all the blocks
 PDStride: This is the number of pixels in the PD stats buffer
 after which there is a jump to a new line.
 PDBufferDataFormat: This is the data type of output stats buffer. -->
 <PDBufferBlockPatternInfo>
  <PDStride>8192</PDStride>
  <!--PDAF   Buffer   Data   TypeRAW10PACKED:   compressed,   [9:2]   [9:2]   [9:2]   [9:2]
[1:0][1:0][1:0][1:0]
  RAW10LSB: Q10 format -->
  <PDBufferFormat>UNPACKED16</PDBufferFormat>
  <PDBlockPattern>
   <PDPixelCount>4</PDPixelCount>
   <!--One pixel coordinate in a block -->
   <PDPixelCoordinates>
    <PDXCoordinate>0</PDXCoordinate>
    <PDYCoordinate>0</PDYCoordinate>
    <PDPixelShieldInformation>TOPLEFTDIODE</PDPixelShieldInformation>
   </PDPixelCoordinates>
   <!--One pixel coordinate in a block -->
   <PDPixelCoordinates>
    <PDXCoordinate>1</PDXCoordinate>
    <PDYCoordinate>0</PDYCoordinate>
    <PDPixelShieldInformation>TOPRIGHTDIODE</PDPixelShieldInformation>
   </PDPixelCoordinates>
   <!--One pixel coordinate in a block -->
   <PDPixelCoordinates>
    <PDXCoordinate>0</PDXCoordinate>
    <PDYCoordinate>1</PDYCoordinate>
    <PDPixelShieldInformation>BOTTOMLEFTDIODE</PDPixelShieldInformation>
   </PDPixelCoordinates>
   <!--One pixel coordinate in a block -->
   <PDPixelCoordinates>
    <PDXCoordinate>1</PDXCoordinate>
    <PDYCoordinate>1</PDYCoordinate>
    <PDPixelShieldInformation>BOTTOMRIGHTDIODE</PDPixelShieldInformation>
   </PDPixelCoordinates>
   <!--Width and height of the frame or subframe -->
   <PDBlockDimensions>
    <width>2</width>
    <height>2</height>
   </PDBlockDimensions>
   <PDOffsetHorizontal>0</PDOffsetHorizontal>
   <PDOffsetVertical>0</PDOffsetVertical>
  </PDBlockPattern>
 </PDBufferBlockPatternInfo>
</PDModeInfo>
```

QPD configuration

For QPD configuration, the PDType property value should be set to PDTypeQPD. Supported pixel order type could be defined in PDPixelOrderType attribute as LEFTTORIGHT or RIGHTTOLEFT accordingly.

Refer to *chi-cdk/oem/qcom/sensor/imx766/imx766_pdaf.xml* for QPD sample driver configuration.

For QPD mode, two conversion coefficients (one horizontal and one vertical) are expected. And up and down gain along with right and left gain map are also expected in EEPROM settings.

For more information about configuration, refer to the following path:

chi-cdk/oem/qcom/eeprom/gt24p128c2csli_imx766_eeprom.xml

7.5. Sensor driver configuration for PDAF

This node contains the information related to sensor driver configuration to enable PDAF.

resolutionInfo node in the sensor driver XML contains all the resolutions supported by the sensor. Here, the *resolutionData* corresponding to the resolution that supports PDAF is modified to accommodate correct PDAF stream type configurations.

Also, the sensor register settings should be configured appropriately to enable PDAF data streaming based on the data sheet of the specific sensor and vendor settings.

Table 7-9. Sensor and vendor settings

Field	Description
streamInfo	This node holds all the stream configurations supported by this resolution.
streamConfiguration	This defines different stream configuration parameters based on the stream type.
vc	This is the CSID virtual channel on which we should expect the data from this stream.
dt	This is the data type of the data that is being streamed on this channel.
frameDimension	This indicates the width and height of the data being sent on the MIPI stream along with any analog cropping in the sensor, if any. If there is no crop, then xStart and yStart should be 0.
bitWidth	This indicates what is the MIPI output format of the stream. Example: This is 10 for MIPI10.
type	This indicated what is the stream type. Example: PDAF Supported values: • BLOB • IMAGE • PDAF • HDR • META

For example:

```
<streamInfo>
 <streamConfiguration>
  <vc range="[0,3]">0</vc>
  <dt>43</dt>
  <frameDimension>
   <xStart>0</xStart>
   <yStart>0</yStart>
   <width>5488</width>
   <height>4112</height>
  </frameDimension>
  <bitWidth>10</bitWidth>
```

```
 <type>IMAGE</type>
</streamConfiguration>
<streamConfiguration>
 <vc range="[0,3]">0</vc>
 <dt>0x36</dt>
 <frameDimension>
  <xStart>0</xStart>
  <yStart>0</yStart>
  <width>5488</width>
  <height>2</height>
 </frameDimension>
 <bitWidth>10</bitWidth>
 <type>PDAF</type>
</streamConfiguration>
</streamInfo>
```

7.6. Sensor XML and PDAF XML index mapping

The sensor XML is used to configure the stream information with respect to each VC/DT and resolution. The PDAF XML is used to configure the PD hardware block. If the sensor is outputting PDAF stream, then there should be a corresponding PD hardware configuration described in the PDAF XML. *PDSensorMode* is a field used in the PDAF XML for the mapping between PDAF XML and sensor XML of the resolution information corresponding to the PDAF stream. It is 1 to 1 mapped.

Table 7-10. PDSensorMode

Field	Description
PDSensorMode(deprecated)	This node holds the index value of the resolutioninformation in the sensor XML. The resolutioninformation in the sensor XML is indexed sequentiallyfrom 0 to n-1 where 'n' represents the number ofresolution data information. Deprecated for SM8450 and onwards, use PDModeKey.

PDModeKey	This should be mapped to valid PDAF Mode Key. Each PDAF Mode is given user defined mode key to link sensor mode to correct PDAF mode.
	For example:
	`Sensor Driver`
	`Sensor ResolutionId  PDModeKey`
	`  2  10`
	`PDAF Driver`
	`PDAF ResolutionId  PDModeKey`
	`3  10`
	In the example above, PDAF mode 1 corresponds to Sensor mode 0.
	Link between these modes is established usingPDModeKey equal to 10.
	Max value of PDModeKey can be 99.

Example of common mistakes made when new resolution information is added in sensor XML:

Assume that in the old resolution information, there are 7 resolution information parameters: res0, res1, res2, res3, res4, res5, res6 (where res5 and res6 support both IMAGE and PDAF stream). Therefore, in PDAF XML one should see PDSensorMode value for 5 and 6 correspondingly. If one adds a new resolution (only IMAGE) information at index 3 of the sensor XML, there are now a total of 8 resolution information parameters: res0, res1, res2, res3 (newly added), res4 (previously res3), res5(previously res4), res6 (previously res5), res7 (previously res6).

Since the indexing for the resolution information from 3 onwards incremented by one, in the PDAFXML the PDSensorMode value must now be changed to 6 (previously 5) and 7 (previously 6) correspondingly to align the mapping.

Table 7-11. Common mistake

1. Original Driver Configuration

Sensor Driver			PDAF Driver	
Sensor Mode (Auto-generated Index)	**PDModeIndex (User defined key element)**		**PDAF Mode (Auto-generated Index)**	**PDModeIndex (User defined key element)**
0	10		0	10
1			1	20
2	20		2	30
3	30			
4				

2. Add a new sensor mode in the middle of the driver.

Sensor Driver			PDAF Driver	
Sensor Mode (Auto-generated Index)	**PDModeIndex (User defined key element)**		**PDAF Mode (Auto-generated Index)**	**PDModeIndex (User defined key element)**
0	10		0	10
1			1	20
2	20		2	30
3’				
3	30			
4				

3. Add a new PD Mode for the newly added sensor mode.

Sensor Driver			PDAF Driver	
Sensor Mode (Auto-generated Index)	**PDModeIndex (User defined key element)**		**PDAF Mode (Auto-generated Index)**	**PDModeIndex (User defined key element)**
0	10		0	10
1			1	20
2	20		2	30
3’	40		3	40
3	30			
4				

4. Remove an existing sensor mode from the middle of the sensor driver.

Sensor Driver			PDAF Driver	
Sensor Mode (Auto-generated Index)	**PDModeIndex (User defined key element)**		**PDAF Mode (Auto-generated Index)**	**PDModeIndex (User defined key element)**
0	10		0	10
1			1	30
3’	40		2	40
3	30			
4				

5. Allow one PDAF mode to be linked to multiple sensor modes (for instance, Quadra CFA sensors with RAW stream can share same PD settings with Remosaic stream)

Sensor Driver			PDAF Driver	
Sensor Mode (Auto-generated Index)	**PDModeIndex (User defined key element)**		**PDAF Mode (Auto-generated Index)**	**PDModeIndex (User defined key element)**
0	10		0	10
1			1	30
3’	40		2	40
3	30			
4				

Chapter 8. OIS Driver Bring-up Guidelines

8.1. OIS-specific XML

Every OIS has an associated configuration XML that defines features such as, but not limited to, powersettings, code step, and initialization settings.

Entire settings are enclosed in the *< OISDriver></ OISDriver>* node of this XML.

8.1.1. OIS information node

This node contains the information related to the OIS driver configuration parameters.

Table 8-1. OIS slave information

Field	Description
slaveInfo	Master node that stores slave information used to communicate with the OIS.
OISName	Name of the OIS.
slaveAddress	Slave address used to talk to the OIS.
i2cFrequencyMode	I2C frequency mode of slave. Example: FAST Supported modes are: • STANDARD (100 kHz) • FAST (400 kHz) • FAST_PLUS (1 MHz) • CUSTOM (custom frequency in DTSI)
Firmware flag	Flag to know if OIS supports firmware. The following firmware coefficients need to be programmed in the xml: • Prog • Coeff • Peripheral • memory
OIS calibrationflag	Flag to enable if OIS has the calibration data.
powerUpSequence	This node contains the power-up configuration sequence required to control the power to the device while closing it. Example: <powerDownSequence> <powerSetting> <configType>MCLK</configType> <configValue>0</configValue> <delayMs>0</delayMs> </powerSetting> </ powerDownSequence>
powerSetting	This node contains power configuration type, value, and delay.

Field	Description
configType	Power configuration type. Example: MCLK Supported types are: • MCLK • VANA • VDIG • VIO • VAF • RESET • STANDBY
configValue	Power configuration type. Example: 24000000
delayMs	Delay in milliseconds. Example: 1
powerDownSequence	This node contains the power-down configuration sequence required to control the power to the device while closing it. Example: <powerDownSequence> <powerSetting> <configType>RESET</configType> <configValue>0</configValue> <delayMs>0</delayMs> </powerSetting> </ powerDownSequence>

Table 8-2. OIS register information

Field	Description
registerConfig	This node holds sequence of register configurations.
registerParam	OIS register parameter configuration.
regAddrType	Register address type/ data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3‑byte address • 4 = Address type max
regDataType	Register data type/data size in bytes Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
registerAddr	Register address that is accessed.
registerData	Register data to be programmed.

Field	Description
operation	OIS operation to be performed on the register. Supported values: • WRITE_HW_DAMP • WRITE_DAC • WRITE • WRITE_DIR_REG • POLL • READ_WRITE
delayUs	Delay in micro seconds.
hwMask	Hardware mask.
hwShift	Number of bits to shift for the hardware.
dataShift	Number of bits to shift for data.

Table 8-3. OIS init information

Field	Description
initSettings	Sequence of register settings to configure the device.
regSetting	Register setting configuration.
registerAddr	Register address that is accessed.
registerData	Data to be processed, read from, or written into the address.
regAddrType	Register address type/data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3‑byte address • 4 = Address type max
regDataType	Register data type/data size in bytes. Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
Operation	Operation to be performed on the register. Supported values: • READ • WRITE • POLL
delayUs	Delay in micro seconds.

Table 8-4. OIS Mode information

Field	Description
modeSettings	OIS supports different modes pantilt mode, Movie mode, video The driver must write setting based on the mode. The settings are predefined in the xml driver and configured based on the metadata event.
regSetting	Register setting configuration.
registerAddr	Register address that is accessed.
registerData	Data to be processed, read from, or written into the address.
regAddrType	Register address type/data size in bytes. Example: 2 • 1 = Byte address • 2 = Word address • 3 = 3 - byte address • 4 = Address type max
regDataType	Register data type/data size in bytes. Example: 1 • 1 = Byte data • 2 = Word data • 3 = Double word data • 4 = DT max
Operation	Operation to be performed on the register. Supported values: • READ • WRITE • POLL
delayUs	Delay in micro seconds.

Table 8-5. OIS regulator

Field	Description
Compatible	Data type that this node corresponds to.
Reg	Points to the node ID.
regulator-name	LDO regulator name.
regulator-min-microvolt	Minimum voltage for this LDO.
regulator-max-microvolt	Maximum voltage for this LDO.
regulator-enable-ramp-delay	The time taken, in microseconds, for the supply rail to reach the target voltage, ± the tolerance the board design requires. This property describes the total system ramptime required due to the combination of internal ramping of the regulator itself andboard design issues such as, trace capacitance and load on the supply.
enable-active-high	Enable with active high.
Gpio	GPIO to be used with this node.
vin-supply	Input voltage supply node name.

OIS lens position read information

In order to read the OIS lens position data and format information, *OISLensPositionReadInfo* structure needs to be updated. This is optional information and when OIS FW supports lens positionread information and if that information needs to be read for EIS algorithm to use, this structure can be updated. If this information is not updated in the OIS driver xml file, it is assumed that there is no support for OIS lens position data.

This structure also supports information to update *OISLensPositionFormatInfo* and this can be updated if the raw OIS lens position data is according to the standard OIS lens position data format. If the raw data is different from the standard format, then the OIS library can be developed to implementcustom function definition to format and provide the data to CAMX.

Refer to the following common structure to update any of the field which needs to be formatted *inOISLensPositionFormatInfo*.

Table 8-6. FormatInfo

Field	Description
offset	Indicates the position of the value to be formatted inthe raw data read. Raw data index starts with 0.
sizeInBytes	Size of the value in bytes.
offsetIncrementBytes	Indicates size in bytes to increment to go to the next offset value from the current offset.

Structure definition provides information needed to format the raw data. If OEM has raw data that is not as per this standard format, they can implement OIS library function *pFormatOISLensPositionData* as per the API declarations in *camxoisdriverapi.h*. If OIS library implements this function, this will take precedence and internal formatting function will not be used. This structure is not required to update when OEMs implement OIS library format function.

Table 8-7. OISLensPositionFormatInfo

Field	Description
positionValidSamples	Information to find and format number of valid lensposition samples in the raw data read. Update this asper the FormatInfo structure details. One sample isthe set of X, Y lens position data at a particular time.
positionX	Information to find and format X position of the OISlens position data in the raw data read. Update this asper the FormatInfo structure details.
positionY	Information to find and format Y position of the OISlens position data in the raw data read. Update this asper the FormatInfo structure details.
timestamp	Information to find and format timestamp of the OISlens position data in the raw data read. Update this asper the FormatInfo structure details. If the timestamp is available only for the last valid lens position sample, update offsetIncrementBytes as 0.
isPerSampleTimestamp	Set it to "true" if the timestamp is available for each sample. Set it to "false" if the timestamp is available for only last valid sample
timerClockFrequency	Frequency of the timer clock which can be used to convert internal clock to nano seconds capacitance and load on the supply. This is optional parameter to configure and needed when time stamp unit is internal clock.
timestampUnit	Unit of the time stamp associated with the sample. Set it to one of the valid values from these MILLISECONDS, MICROSECONDS, NANOSECONDS, INTERNALCLOCKTICKS.

This structure defines information needed to read the OIS lens position data and write system time to OIS FW, if FW supports it and for formatting the data. If this structure is not available as part OIS driver, then framework will treat it as no OIS lens position data available.

Table 8-8. OISLensPositionReadInfo

Field	Description
sampleFrequencyInHZ	Specifies the rate (in Hertz) at which OIS lens position data is captured and cached in OIS FW. This field is used to determine time difference between each lens position data and to determine OIS driver internal buffer sizes.
totalSamples	Specifies total number of samples in each read buffer including the dummy samples. This is a fixed value based on how many OIS samples OIS FW can cache to read.
readIntervalInMS	Specifies the interval in milli seconds at which len sposition data buffer can be read. This value should be less than ((1000 /sampleFrequencyInHZ) * totalSamples) Reduce this vale by considering software and CCI delays.
readSettings	Register settings to read the OIS lens position data.
writeTimeSettings	Element for writing reference time stamp register settings for system time synchronization. This is optional configuration and need to be configured if writing system to OIS FW is supported. ➲ **Note:** Write time register data should be updated to FF FF FF FF FF FF FF FF as these 8 bytes will be updated while with the current Qtimers value while writing time to OIS FW.
formatInfo	Format information to format the read data. Format information defined in *OISLensPositionFormatInfo*.

8.2. OIS hardware configuration

8.2.1. Kernel hardware configuration

For more details and explanation of each field in the device tree file, refer to *kernel/msm-4.19/ Documentation/devicetree/bindings/media/video/msm-cam-cci.txt* or *vendor/qcom/proprietary/camera-devicetree/bindings/msm-cam-cci.txt*(for relatively new baselines).

8.2.2. OIS node structure

The hardware configuration of the OIS module is saved in the kernel DTSI file. See Table 3-26 for details.

8.3. OIS library configuration

OIS infrastructure provides an interface for OEMs to write their own functions to implement custom methods. With this change, each OIS can have its own custom library, which implements the API exposed in camxoisdriverAPI.h. After the API methods are implemented, generate the library with the name *com.<vendor name>.ois.<ois name>.so*. This file should also be included in the device-vendor.mk file with the other library files that need to be generated.

Generated library is available at */vendor/lib64/camera/ (for LA) or /usr/lib64/camera/* (for LE).

Once the library with the specified syntax name in the specified path is available, OIS software module loads the binary and accesses the API methods to use them as needed.

Library functions

It is possible to define additional methods to the OIS driver using OIS library. This library can only expose functions with the following definitions.

Structures defined:

```
/// @brief LensPosition to hold the formated lens position data
typedef struct LensPosition
{
  FLOAT  shiftX;     ///< OIS lens position shift in X direction
  FLOAT  shiftY;     ///< OIS lens position shift in Y direction
  UINT64 timeStampInNS; ///< time stamp of the sample w.r.t Qtimer in nano seconds
}LensPosition;

/// @brief LensPositionDataInfo to hold the formated lens position and also
info needs to format the raw data
typedef struct LensPositionDataInfo
{
  LensPosition*  pLensPos;       ///< OIS lens position data to be filled in after format
(Output)
  UINT       validSamples;    ///< number valid samples in the formatted data (Output)
  UINT32      totalSamples;    ///< Total samples lensPos can hold (Input)
  UINT8*      pRawLensPosData;  ///< Pointer to raw lens position data (Input)
  UINT32      rawDataSize;     ///< Size of the raw lens position data (Input)
  UINT64         QTimerInNS;      ///< QTimer values in nano seconds at the time of lens
position read (Input)
  }LensPositionDataInfo;
```

Example:

```
static bool FormatOISLensPositionData(LensPositionDataInfo* pLensPosData)
{
// Implement the format function to format the raw data and output the
formatted data as per the define ouput structure LensPosition.
}
void GetOISLibraryAPIs(OISLibraryAPI* pOISLibraryAPI){ pOISLibraryAPI->
pFormatOISLensPositionData = FormatOISLensPositionData;
}
```

Chapter 9. Camera Debug Log

9.1. Camera user mode driver log

9.1.1. Camera user mode override settings

For debug purposes, there are two methods to override default camera driver setttings:

Step 1. Push a configuration file to */vendor/etc/camera/camxoverridesettings.txt*.

Step 2. Set Android properties:

```
adb shell setprop <setting> <value>
```

A device reboot may be required after changing override settings if the camera server has already loaded the UMD instance.

Table 9-1. Override settings

Description	Name	Default Value	Setprop
Verbose Log Mask	logVerboseMask	0	persist.vendor.camera.logVerboseMask
Entry/Exit Log Mask	logEntryExitMask	0	persist.vendor.camera.logEntryExitMask
Info Log Mask	logInfoMask	0xFFFFFFFF	persist.vendor.camera.logInfoMask
Warning Log Mask	logWarningMask	0xFFFFFFFF	persist.vendor.camera.logWarningMask
System Log Enable	systemLogEnable	TRUE	persist.vendor.camera.systemLogEnable

- **Example 1**

  Controlled debug log messages determine which groups of the camera system output logs for each log level.

  To enable verbose and entry or exit log settings, add the following to the camxoverridesettings.txt file:

```
logVerboseMask=0xFFFFFFFF
logEntryExitMask=0xFFFFFFFF
```

  With a rooted device, use the following commands as a quick way to write these values to the override file:

```
adb root
adb remount
adb shell "echo logVerboseMask=0xFFFFFFFF >> /vendor/etc/camera/camxoverridesettings.txt"
adb shell "echo logEntryExitMask=0xFFFFFFFF >> /vendor/etc/camera/camxoverridesettings.txt"
```

- **Example 2**

  To enable verbose logs for HAL CSL groups only, enable warnings from sensor, HAL, and CSL, and disable system logging:

```
logVerboseMask=0x00000480
logWarningMask=0x482
systemLogEnable=FALSE
```

9.1.2. Camera user mode logging groups

The driver is split into various logical components known as groups. Using a specific group allows you to reduce the amount of log messages sent to the component(s) you are interested in obtaining information from. This is accomplished with a bitmask to selectively choose which logs to enable from different components of the camera system.

Table 9-2.Override settings

Driver Group Name	Value	Description
ogGroupNone	(1 << 0)	Generic default group
ogGroupSensor	(1 << 1)	Sensor
CamxLogGroupIFace	(1 << 2)	IFace
CamxLogGroupISP	(1 << 3)	ISP
CamxLogGroupPProc	(1 << 4)	Post processor
CamxLogGroupHAL	(1 << 7)	HAL
CamxLogGroupJPEG	(1 << 8)	JPEG
CamxLogGroupStats	(1 << 9)	3A algorithms
CamxLogGroupCSL	(1 << 10)	Camera service layer
CamxLogGroupUtils	(1 << 12)	Utilities
CamxLogGroupSync	(1 << 13)	Synchronization/mutex/fences
CamxLogGroupMemSpy	(1 << 14)	Memory tracker
CamxLogGroupCore	(1 << 16)	Core camera system
CamxLogGroupHWL	(1 << 17)	Hardware layer
CamxLogGroupChi	(1 << 18)	Camera HAL interface
CamxLogGroupDRQ	(1 << 19)	Deferred request queue
CamxLogGroupFD	(1 << 20)	Face detection

9.2. Camera kernel mode driver logs

KMD log messages are enabled by default and appear in the kernel logs. Camera KMD logs all start with the CAM prefix. Additional debugging logs are available in the KMD and are controlled through groups.

Table 9-3.KMD logging groups

KMD Group	Value	Description
CAM_CDM	(1 << 0)	Camera data mover
CAM_CORE	(1 << 1)	Camera core
CAM_CPAS	(1 << 2)	Camera peripherals and support
CAM_ISP	(1 << 3)	Image signal processor
CAM_CRM	(1 << 4)	Camera request manager
CAM_SENSOR	(1 << 5)	Sensor
CAM_SMMU	(1 << 6)	Shared memory management unit
CAM_SYNC	(1 << 7)	Synchronization
CAM_ICP	(1 << 8)	Image control processor
CAM_JPEG	(1 << 9)	JPEG

KMD Group	Value	Description
CAM_FD	(1 << 10)	Face detection
CAM_LRME	(1 << 11)	Low resolution motion estimation
CAM_FLASH	(1 << 12)	Flash
CAM_ACTUATOR	(1 << 13)	Actuator
CAM_CCI	(1 << 14)	Camera control interface
CAM_CSIPHY	(1 << 15)	Camera serial interface
CAM_EEPROM	(1 << 16)	Electronically erasable programmable read-Only memory
CAM_UTIL	(1 << 17)	Utilities
CAM_HFI	(1 << 18)	Host-firmware interface
CAM_CTXT	(1 << 19)	Camera context
CAMX_OIS	(1 << 20)	Optical image stabilization
CAM_RES	(1 << 21)	Camera shared resource API
CAM_MEM	(1 << 22)	For camera memory manager
CAM_IRQ_CTRL	(1 << 23)	For events in IRQ controller
CAM_REQ	(1 << 24)	Tracks a request submitted to KMD
CAM_PERF	(1 << 25)	Used for performance (clock, BW, etc.) logs
CAM_CUSTOM	(1 << 26)	For custom hardware node
CAM_OPE	(1 << 28)	For SM4250/SM6115 OPE module debug

Follow the steps below to enable camera kernel mode driver logs.

Step 1. If the platform supports camera DLKM, change */sys/module* in the following path to *sys/module/camera*. To enable CAM_SENSOR and CAM_ICP debug logs in the KMD, run the following commands:

```
adb root
adb remount
adb shell "echo 0x120 > /sys/module/cam_debug_util/parameters/debug_mdl"
adb shell cat /proc/kmsg > name_of_kmd_logs.txt
```

Step 2. To enable CSID IRQ logs dynamically, use the appropriate mask value(s) with the following command:

```
adb shell "echo MASKVALUE > /sys/kernel/debug/camera_ife/ife_csid_debug"
```

For example:

- To enable SOF, EOF, SOT, EOT IRQs:

```
adb shell "echo 0xf > /sys/kernel/debug/camera_ife/ife_csid_debug"
```

- To set the only short packet and long packet capture IRQs:

```
adb shell "echo 0x30 > /sys/kernel/debug/camera_ife/ife_csid_debug"
```

The information is logged to kmsg and can be obtained using the following command:

```
adb shell cat /proc/kmsg > name_of_kmd_logs.txt
```

Table 9-4.CSID debug groups

CSID Debug Group	Value	Description
CSID_DEBUG_ENABLE_SOF_IRQ	(1 << 0)	Start of frame
CSID_DEBUG_ENABLE_EOF_IRQ	(1 << 1)	End of frame
CSID_DEBUG_ENABLE_SOT_IRQ	(1 << 2)	Start of transmission
CSID_DEBUG_ENABLE_EOT_IRQ	(1 << 3)	End of transmission
CSID_DEBUG_ENABLE_SHORT_PKT_CAPTURE	(1 << 4)	Short packet capture
CSID_DEBUG_ENABLE_LONG_PKT_CAPTURE	(1 << 5)	Long packet capture
CSID_DEBUG_ENABLE_CPHY_PKT_CAPTURE	(1 << 6)	CPHY packet capture

Chapter 10. Troubleshooting

This chapter introduces the log analysis for troubleshooting.

- **Sensor reference DTSI configuration and DTSI parsing log**

```
qcom,cam-sensor4 {
cell-index = <4>; //this is to align sensor module xml
compatible = "qcom,cam-sensor"; //this should be same as cam_sensor_driver_dt_match
csiphy-sd-index = <0>; //pair to csiphy0
actuator-src = <&actuator_triple_wide>; //including actuator
led-flash-src = <&led_flash_triple_rear>; //using LED based as Flash eeprom-src =
<&eeprom_triple_wide>; //including eeprom
cam_vio-supply = <&pm8009_l7>; cam_bob-supply = <&pm8150a_bob>; cam_vana-supply =
<&pm8009_l5>; cam_vdig-supply = <&pm8009_l1>; cam_clk-supply = <&titan_top_gdsc>;
regulator-names = "cam_vio", "cam_vana", "cam_vdig", "cam_clk", "cam_bob";
rgltr-cntrl-support; pwm-switch;
rgltr-min-voltage = <1800000 2800000 1104000 0 3008000>;
rgltr-max-voltage = <1800000 3000000 1104000 0 3960000>;
rgltr-load-current =<120000 80000 1200000 0 2000000>;
gpio-no-mux = <0>;
pinctrl-names = "cam_default", "cam_suspend"; //cam_default means when running, pinctrl-
0 = <&cam_sensor_mclk0_active
&cam_sensor_active_rear>; //defined in /arm64/boot/dts/vendor/qcom/holi-pinctrl.dtsi
pinctrl-1 = <&cam_sensor_mclk0_suspend
&cam_sensor_suspend_rear>;
gpios = <&tlmm 94 0>, //using GPIO 94 as mclk0
<&tlmm 93 0>;//using GPIO 93 as reset
gpio-reset = <1>;//using the gpios[1] as reset, which is GPIO 93 gpio-req-tbl-num = <0
1>;
gpio-req-tbl-flags = <1 0>; //0 indicates GPIO 93 as GPIOF_DIR_OUT gpio-req-tbl-label =
"CAMIF_MCLK0",
"CAM_RESET0";
sensor-mode = <0>;
cci-master = <0>; status = "ok";
clocks = <&clock_camcc CAM_CC_MCLK0_CLK>; clock-names = "cam_clk";
clock-cntl-level = "turbo"; clock-rates = <24000000>;
};
CAM-UTIL: cam_soc_util_get_dt_properties: 1253 cam_soc_util_get_dt_properties for:
ac4f000.qcom,cci:qcom,cam-sensor4
CAM-UTIL: cam_soc_util_get_dt_regulator_info: 1182 cam_soc_util_get_dt_regulator_info
for: ac4f000.qcom,cci:qcom,cam-sensor4
CAM-UTIL: cam_soc_util_get_dt_regulator_info: 1204 rgltr_name[0] = cam_vio CAM-UTIL:
cam_soc_util_get_dt_regulator_info: 1204 rgltr_name[1] = cam_vana CAM-UTIL:
cam_soc_util_get_dt_regulator_info: 1204 rgltr_name[2] = cam_vdig CAM-UTIL:
cam_soc_util_get_dt_regulator_info: 1204 rgltr_name[3] = cam_clk CAM-UTIL:
cam_soc_util_get_dt_regulator_info: 1204 rgltr_name[4] = cam_bob
CAM-UTIL: cam_soc_util_get_dt_clk_info: 742 cam_soc_util_get_dt_clk_info for:
ac4f000.qcom,cci:qcom,cam-sensor4
CAM-UTIL: cam_soc_util_get_dt_clk_info: 745 No shared clk parameter defined
CAM-UTIL: cam_soc_util_get_dt_clk_info: 754 E: dev_name = ac4f000.qcom,cci:qcom,cam-
sensor4 count = 1
CAM-UTIL: cam_soc_util_get_dt_clk_info: 772 clock-names[0] = cam_clk CAM-UTIL:
cam_soc_util_get_dt_clk_info: 820 [0] : turbo 7
CAM-UTIL: cam_soc_util_get_dt_clk_info: 840 soc_info->clk_rate[7][0] = 24000000 CAM-
UTIL: cam_soc_util_get_dt_clk_info: 848 No src_clk_str found
CAM-UTIL: cam_soc_util_get_gpio_info: 1071 gpio count 2
CAM-UTIL: cam_soc_util_get_gpio_info: 1079 gpio_array[0] = 1194
CAM-UTIL: cam_soc_util_get_gpio_info: 1079 gpio_array[1] = 1193
CAM-UTIL: cam_soc_util_get_dt_gpio_req_tbl: 1006 cam_gpio_req_tbl[0].gpio = 1194
CAM-UTIL: cam_soc_util_get_dt_gpio_req_tbl: 1006 cam_gpio_req_tbl[1].gpio = 1193
CAM-UTIL: cam_soc_util_get_dt_gpio_req_tbl: 1019 cam_gpio_req_tbl[0].flags = 1
CAM-UTIL: cam_soc_util_get_dt_gpio_req_tbl: 1019 cam_gpio_req_tbl[1].flags = 0
CAM-UTIL: cam_soc_util_get_dt_gpio_req_tbl: 1031 cam_gpio_req_tbl[0].label = CAMIF_MCLK0
CAM-UTIL: cam_soc_util_get_dt_gpio_req_tbl: 1031 cam_gpio_req_tbl[1].label = CAM_RESET0
CAM-UTIL: cam_soc_util_get_dt_properties: 1338 cam_soc_util_get_dt_properties end:
ac4f000.qcom,cci:qcom,cam-sensor4
```

- **Slot index described in module XML and DTSI**

  As part of camxhal3module initialize, the sensors will be probed based on the number of sensor module BIN files in */vendor/lib64/camera* and during each probe. It is based on the slot index described in module xml and DTSI.

```
CamX : [ INFO][SENSOR ] camximagesensormoduledatamanager.cpp:226
CreateAllSensorModuleSetManagers() total:8
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 0 = /vendor/lib64/camera/
com.qti.sensormodule.truly_imx476.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 1 = /vendor/lib64/camera/
com.qti.sensormodule.sunny_imx519.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 2 = /vendor/lib64/camera/
com.qti.sensormodule.pmd_irs2381c.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 3 = /vendor/lib64/camera/
com.qti.sensormodule.semco_imx481.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 4 = /vendor/lib64/camera/
com.qti.sensormodule.ofilm_imx563.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 5 = /vendor/lib64/camera/
com.qti.sensormodule.semco_s5k3m5.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 6 = /vendor/lib64/camera/
com.qti.sensormodule.liteon_imx362.bin
CamX : [ VERB][SENSOR ] camximagesensormoduledatamanager.cpp:239
CreateAllSensorModuleSetManagers() 7 = /vendor/lib64/camera/
com.qti.sensormodule.semco_imx586.bin
CAM_INFO: CAM-SENSOR: cam_sensor_match_id: 634 read id: 0x586 expected id 0x586:
CAM_INFO: CAM-SENSOR: cam_sensor_driver_cmd: 730 Probe
success,slot:4,slave_addr:0x34,sensor_id:0x586
CamX : [ INFO][SENSOR ] camximagesensormoduledata.cpp:736 Probe() Probe success results
- Detected: 1, DeviceIndex: 27,sensorname:imx586
CAM_INFO: CAM-SENSOR: cam_sensor_driver_cmd: 730 Probe
success,slot:6,slave_addr:0x34,sensor_id:0x481
CamX : [ INFO][SENSOR ] camximagesensormoduledata.cpp:736 Probe() Probe success results
- Detected: 1, DeviceIndex: 28,sensorname:imx481
CAM_INFO: CAM-SENSOR: cam_sensor_driver_cmd: 730 Probe
success,slot:5,slave_addr:0x20,sensor_id:0x30d5
CamX : [ INFO][SENSOR ] camximagesensormoduledata.cpp:736 Probe() Probe success results
-
Detected: 1, DeviceIndex: 29,sensorname:s5k3m5 semco_imx586_module.xml
<!--Module configuration -->
<moduleConfiguration description="Module configuration">
<!--CameraId is the id to which DTSI node is mapped. Typically CameraId is the slot Id
for non combo mode. -->
<cameraId>4</cameraId> holi-camera-sensor-qrd.dtsi qcom,cam-sensor4 {
cell-index = <4>;
compatible = "qcom,cam-sensor"; csiphy-sd-index = <0>;
sensor-position-roll = <90>;
```

- **Loading the Sensor sub modules listed in module XML**

```
CamX : [ INFO][SENSOR ] camxeepromdata.cpp:104 EEPROMData() Data read success for:
cat24c64_imx586
CamX : [ INFO][SENSOR ] camximagesensormoduledata.cpp:812 CreateSensorSubModules()
Actuator device found on camera 4, name:lc898217xc
CamX : [ INFO][SENSOR ] camximagesensormoduledata.cpp:876 CreateSensorSubModules() Flash
device found on camera 4.
<actuatorName>lc898217xc</actuatorName>
<eepromName>cat24c64_imx586</eepromName>
<flashName>pmic</flashName>
```

- **PHY data rate selection**

  The data rate configuration is based on sensor stream format and output pixel clock.

```
CAM_INFO: CAM-CSIPHY: cam_csiphy_cphy_data_rate_config: 286 required data rate :
1596333333
CAM_DBG : CAM-CSIPHY: cam_csiphy_cphy_data_rate_config: 303: table[0] BW : 5700000000
Selected // data_rate_delta_table_1_2 in drivers/cam_sensor_module/cam_csiphy/include
/Cam_csiphy_1_2_hwreg defines the PHY data rate and Table[0] is to support up to 2.5Gsps
```

Appendix 1. References

A1.1. Related documents

Table A1-1. Related documents

Title	DCN or URL
Resources	-
D-PHY	https://mipi.org/specifications/d-phy

A1.2. Acronyms and terms

Table A1-2. Acronyms and terms

Acronym or term	Definition
CID	Channel identifier
DT	Data type
I2C	Inter-integrated circuit
SPI	Serial peripheral interface
VC	Virtual channel

Appendix 2. Notices

Thundercomm may have patents or pending patent programs covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries to service@thundercomm.com.

THUNDERCOMM PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Changes are made periodically to the information herein; these changes will be incorporated in new editions of the publication. To provide better service, Thundercomm reserves the right to improve and/or modify the products and software programs described in the manuals, and the content of the manual, at any time without additional notice.

The software interface and function and hardware configuration described in the manuals included with your development board or system on module might not match exactly the actual configuration of that you have purchased. For the configuration of the product, refer to the related contract (if any) or product packing list, or consult the distributor for the product sales. Thundercomm may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Thundercomm product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Thundercomm or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

The information of this document should not be as any invitation for offer or any advice to the visitors. Please consult the professional comments from the sales consultant prior to do any actions of investment or purchase.

Thundercomm may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Thundercomm Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Thundercomm product, and use of those Web sites is at your own risk. Thundercomm shall not be responsible for the content of the third party.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

E-mail messages sent to Thundercomm via the Internet are not guaranteed to be completely secure. Thundercomm shall not be liable for any loss incurred by the surfer when transmitting any information over the Internet or for any loss incurred by Thundercomm when sending any information over the Internet at your request.

Thundercomm has all rights under other relevant exemptions provided by laws and regulations, and Thundercomm's failure to claim or delay in claiming such rights shall not be deemed to be a waiver of such rights by Thundercomm.

Thundercomm reserves the right of final interpretation of this document.

Appendix 3. Trademarks

Thundercomm and Thundercomm TurboX are trademarks of Thundercomm Corporation or its associate companies in China and/or other countries. Intel, Intel SpeedStep, Optane, and Thunderbolt are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Microsoft, Windows, Direct3D, BitLocker, and Cortana are trademarks of the Microsoft group of companies. MDP (Mini DisplayPort), DisplayPort, and VESA are trademarks of the Video Electronics Standards Association. The terms HDMI and HDMI High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC in the United States and other countries. Wi-Fi, Wi-Fi Alliance, WiGig, and Miracast are registered trademarks of Wi-Fi Alliance. USB-C is a registered trademark of USB Implementers Forum. All other trademarks are the property of their respective owners.