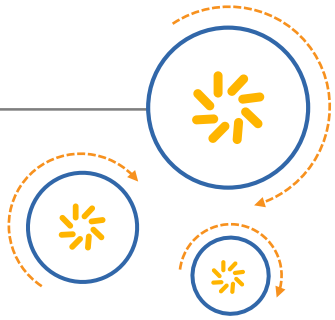
Qualcomm Technologies, Inc.



Camera Sensor Driver Development and Troubleshooting

摄像头模块驱动指南

80-NL239-32 G

January 16, 2015

Qualcomm Technologies, Inc. 5775 Morehouse Drive San Diego, CA 92121 U.S.A.
高通技术股份有限公司，美国加利福尼亚州圣地亚哥市莫豪斯路 5775 号，邮编 92121

Revision history

Revision	Date	Description
A	Apr 2014	Initial release
B	May 2014	Updated .dtsi file names
C	Jul 2014	[English version] Added Chapter 6 and Sections 2.3.4, 4.2.2, and 4.2.4. Updated Sections 2.3.1.1 and 4.2.1
D	Nov 2014	[English version] Updated title and Sections 2.3.1.2, 2.3.2.7.2, 5.1, 5.2.1, 5.2.2, and 7.1.2.2. Added Sections 3.2.2, 2.3.2, and Chapter 6
E	Dec 2014	[English version] Updated for MSM8909
F	Dec 2014	[English version] Updated Section 1.1
G	Jan 2015	Update for MSM8992 chipset

Contents

Figures

Tables

English Version

1 Introduction

1.1 Purpose

This document provides driver development guidelines for the camera sensor and associated modules, and describes how to bring up the camera. Much of the information in this document is generic to the Linux camera code on all MSM8x26/28, MSM8926/28, MSM8974, MSM8992, MSM8994, MSM8909, MSM8916, MSM8x36, andAPQ8084 Android platforms. Chipset-specific differences are covered in their separate sections.

Driver development guidelines and bringup procedures for other multimedia technologies are described in separate documents:

- *Multimedia Driver Development and Bringup Guide – Audio* (80-NU323-1)
- *Multimedia Driver Development and Bringup Guide – Display* (80-NU323-3)
- *Multimedia Driver Development and Bringup Guide – Video* (80-NU323-5)

NOTE: Camera driver development and bringup procedures are described in an identical document, *Multimedia Driver Development and Bringup Guide – Camera* (80-NU323-2), which will supersede this document in March 2015.

The camera sensor framework includes the configuration of the following components:

- Sensor
- CSIPHY
- CSID
- Camera Control Interface (CCI)
- Actuator
- Flash
- EEPROM
- Chromatix™

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, e.g., **`copy a:*.* b:`**.

If you are viewing this document using a color monitor, or if you print this document to a color printer, **red boldface** indicates code that is to be **added**, and ~~blue strikethrough~~ indicates code that is to be replaced or removed.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Pre-Bringup Guidelines

This chapter provides useful information on how to search existing PVL driver and understand the impact of non-PVL component selection on the overall camera schedule. It's recommended that the readers of this document review this information before proceeding with camera bring-up.

2.1 Guidelines for camera sensor selection and development timeframes

To understand the guidelines for camera selection and the development time/resource investment based on it, see Solution [00028471].

2.2 Accessing PVL drivers

For guidelines on how to access PVL (Preferred Vendor List) drivers, see *Qualcomm Createpoint Hardware Component Quick Start Guide (English)* (80-NC193-10) or *Qualcomm Createpoint Hardware Component Quick Start Guide (Chinese)* (80-NC193-10SC). Use these guidelines to find the list of available PVL camera drivers for a specific chipset, and download one or more drivers.

2.3 Useful resources

Review important documents listed under section "Sensor bring up" of Solution [00028470], and start sensor bring-up. If issues are encountered during the process, QTI Customer Engineering can help via a Salesforce case. For guidance on correct problem area identification, see Solution [00028523].

3 Sensor Driver Bringup

This chapter provides information necessary to bring up camera sensor hardware on the Android platform. The code samples in this chapter are based on an MSM8916 device.

NOTE: Unless specified otherwise, all information applies to Bayer sensors.

3.1 Reference drivers for YUV and Bayer sensors

This section shows the list of reference drivers for both Bayer and YUV sensors for MSM8916.

Bayer reference drivers

User space drivers are located in $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/.

- imx135_lib.c/h
- ov2680_lib.c
- ov2720_lib.c
- ov9724_lib.c
- s5k3l1yx_lib.c

YUV reference drivers

User space drivers are located in $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/.

- sp1628_lib.c
- SKUAA-Shengtai-hi256_lib.c
- ov5645_lib.c
- mt9m114_lib.c

Kernel drivers are located in kernel/drivers/media/platform/msm/camera_v2/sensor.

- sp1628.c
- hi256.c
- ov5645.c
- mt9m114.c

3.2 Files to be modified to add new driver

This section lists the files that must be modified to write a new sensor driver.

Bayer sensor

The device tree source file is <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi. Customers should use camera slots as shown here:

```
qcom,camera@0 {
        cell-index = <0>;
        compatible = "qcom,camera";
        . . .
}
```

Config

In vendor/qcom/proprietary/common/config/device-vendor.mk, make entry of the new libraries in the file to include in the build.

In $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/module/sensor_init.c, add sensor name in sensor_libs[] array.

NOTE: The kernel nodes are generic. There is no need to change them if the QTI schematics are followed.

NOTE: A change in this file must be made based on the customer's hardware design.

The user space sensor driver is <sensor>_lib.c/h and Android.mk in $(MM_CAMERA_DIR)/ mm-camera2/media-controller/modules/sensors/sensor_libs/, e.g., imx135_lib.c/h.

In addition to the driver file, Android.mk must be modified (refer to the reference makefile).

NOTE: For a 64-bit processor, use the arm64 directory instead of arm.

YUV sensor

The device tree source file is <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi. Customers should add a new entry in the .dtsi file as shown here:

```
qcom,camera@78 {
        compatible = "ovti,ov5645";
        . . .
|       }
```

The user space sensor driver is <sensor>_lib.c and Android.mk in $(MM_CAMERA_DIR)/ mm-camera2/media-controller/modules/sensors/sensor_libs/, e.g., sp1628_lib.c.

In addition to the driver file, Android.mk must be modified (refer to the reference makefile).

The kernel sensor driver is <sensor>.c and Makefile in kernel/drivers/media/platform/msm/camera_v2/sensor/.

Also, the CONFIG_<sensor> flag must be added in <target>_defconfig located in kernel/arch/arm/configs/.

Config

In vendor/qcom/proprietary/common/config/device-vendor.mk, make entry of the new libraries in the file to include in the build.

NOTE: A change in this file must be made based on the customer's hardware design.

NOTE: For a 64-bit processor, use the arm64 directory instead of arm.

As shown in the above example, for the Bayer sensor the concept of camera slots (0/1/2) is introduced, whereas in the YUV sensor a new entry must be aded in dtsi.

NOTE: Begin with reference files when writing a new driver.

3.3 Source code explanation

3.3.1 Kernel driver

This section provides information necessary for creating the kernel driver.

3.3.1.1 GPIO config

As shown below, the customer can configure sensor-specific GPIOs based on the target board. For explanations on each property, refer to documents in the following location:

kernel/Documentation/devicetree/bindings/media/video/

GPIO can be configured one of two ways, based on the software being used.

Using pinctrl

For the chipsets using pinctrl framework (e.g., MSM8909, MSM8916, MSM8936, MSM8939, MSM8992, MSM8994, etc.), pinctrl node entries in .dtsi can be used to configure GPIOs, e.g.:

```
pinctrl-names = "cam_default", "cam_suspend";
pinctrl-0 = <&cam_sensor_mclk0_default &cam_sensor_rear_default>;
pinctrl-1 = <&cam_sensor_mclk0_sleep &cam_sensor_rear_sleep>;
```

The phandles pointing at a pin configuration node, listed under pinctrl-XX entries above are defined in msmXXXX-pinctrl.dtsi. For MSM8916, it would be at kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi.

Using GPIO control

For chipsets not using pinctrl framework (e.g., MSM8x26/28, MSM8926/28, MSM8974,MSM8084, etc.), GPIO node entries in .dtsi can be used to configure GPIOs, e.g.:

```
gpios = <&msm_gpio 26 0>,
   <&msm_gpio 35 0>,
   <&msm_gpio 34 0>;
qcom,gpio-reset = <1>;
qcom,gpio-standby = <2>;
qcom,gpio-req-tbl-num = <0 1 2>;
qcom,gpio-req-tbl-flags = <1 0 0>;
qcom,gpio-req-tbl-label = "CAMIF_MCLK",
   "CAM_RESET1",
   "CAM_STANDBY";
```

For chipsets having a CCI hardware block, there are dedicated GPIOs for data and clock. Because these GPIOs are dedicated to the CCI master, the customer must use these settings if CCI is used for camera I2C. For example, the following pinctrl or GPIO control-based examples show dedicated GPIO pins 29 and 30 for CCI.

Using pinctrl

```
pinctrl-names = "cci_default", "cci_suspend";
pinctrl-0 = <&cci0_default>;
pinctrl-1 = <&cci0_sleep>;
```

The phandles pointing at a pin configuration node, listed under pinctrl-XX entries above are defined in msmXXXX-pinctrl.dtsi. For MSM8916, it would be at kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi.

Using GPIO control

```
gpios = <&msm_gpio 29 0>,
   <&msm_gpio 30 0>;
qcom,gpio-tbl-num = <0 1>;
qcom,gpio-tbl-flags = <1 1>;
qcom,gpio-tbl-label = "CCI_I2C_DATA0",
      "CCI_I2C_CLK0";
```

CCI has two independent masters (0 and 1, available). In most uses cases, using one CCI master is enough. However, in certain cases two CCI masters can be used. For example, if the front and rear cameras happen to have the same I2C slave address, connecting each camera to a different CCI master with independent physical GPIO pins for CCI masters can resolve the conflicting I2C slave address problem.

```
qcom,cci-master = <0>;
```

3.3.1.2 Clock-related settings

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

```
clocks = <&clock_gcc clk_mclk0_clk_src>,
               <&clock_gcc clk_gcc_camss_mclk0_clk>;
clock-names = "cam_src_clk", "cam_clk";
```

The ordering of the lists in the two properties is important. The nth clock-name will correspond to the nth entry in the clock's property. Thus, in the DT snippets above, cam_src_clk would correspond to clk_mclk0_clk_src, cam_clk would correspond to clk_gcc_camss_mclk0_clk, etc.

The customer does not need to change this, as it is parsed in the clock framework.

NOTE: For MSM8916, it is recommended that sensor driver settings be configured for 23.88 MHz MCLK input. Although drivers written with 24 MHz MCLK input assumption generally will not have functionality issues, some might show banding artifacts with light sources operating at variable frequency like 50 Hz.

3.3.1.3 Power handler

PMIC case

```
cam_vdig-supply = <&pm8916_s4>;
cam_vana-supply = <&pm8916_l17>;
cam_vio-supply = <&pm8916_l6>;
cam_vaf-supply = <&pm8916_l10>;
```

GPIO case

```
gpios = <&msm_gpio 27 0>,
            <&msm_gpio 28 0>,
            <&msm_gpio 33 0>,
            <&msm_gpio 114 0>,
            <&msm_gpio 110 0>;
qcom,gpio-reset = <1>;
qcom,gpio-standby = <2>;
qcom,gpio-vdig = <3>;
qcom,gpio-vana = <4>;
qcom,gpio-req-tbl-num = <0 1 2 3 4>;
qcom,gpio-req-tbl-flags = <1 0 0 0 0>;
qcom,gpio-req-tbl-label = "CAMIF_MCLK",
                          "CAM_RESET",
                          "CAM_STANDBY",
                          "CAM_VDIG",
                          "CAM_VANA";
```

- CAM_VANA – Supply voltage (analog)
- CAM_VDIG – Supply voltage (digital)

  CAM_VAF – Supply voltage (actuator voltage)
- CAM_VIO – Input/output voltage (digital)

3.3.1.4 I2C slave configuration

YUV sensor

In the .dtsi file:

```
qcom,camera@78 {
        compatible = "ovti,ov5645";
        reg = <0x78 0x0>;
        qcom,slave-id = <0x78 0x300a 0x5645>;
```

This is an 8-bit address where 7 MSB are I2C slave address and 1 LSB is write flag 0.

NOTE: For Bayer, see Section 3.3.2.3.

3.3.2 User space driver

This section provides information necessary for creating the user space driver.

3.3.2.1 Sensor init parameters

The sensor init parameters include the modes supported (2D/3D) and the position (FRONT/ BACK) of the camera in the device and the mount angle. If the mount angle is specified as 360, then the driver will pick up the mount angle specified in the kernel dtsi.

```
static struct msm_sensor_init_params sensor_init_params = {
  .modes_supported = CAMERA_MODE_2D_B,
  .position = BACK_CAMERA_B,
  .sensor_mount_angle = SENSOR_MOUNTANGLE_360,
};
```

3.3.2.2 Sensor output parameters

The sensor out parameter includes the output format, which specifies whether the sensor is Bayer or YUV. The connection mode specifies interface between the sensor and receiver (MIPI or parallel). The output size is specified in the raw_output.

```
static sensor_output_t sensor_output = {
  .output_format = SENSOR_BAYER,
  .connection_mode = SENSOR_MIPI_CSI,
  .raw_output = SENSOR_10_BIT_DIRECT,
};
```

3.3.2.3 Bayer slave configuration

The Sensor slave configuration information must provide the following information.

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {
  /* Camera slot where this camera is mounted */
  .camera_id = CAMERA_0,
  /* sensor slave address */
  .slave_addr = 0x20,
  /* sensor address type */
  .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
  /* sensor id info*/
  .sensor_id_info = {
    /* sensor id register address */
    .sensor_id_reg_addr = 0x0016,
    /* sensor id */
    .sensor_id = 0x0135,
  },
  /* power up / down setting */
  .power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
  },
. . .
```

This is an 8-bit address where 7 MSB are I2C slave address and 1 LSB is write flag 0.

NOTE: For YUV, see Section 3.3.1.4.

3.3.2.3.1 Sensor chip-id

The Sensor model ID/chip ID register must be identified from the data sheet and entered in the sensor_id_info.

```
.sensor_id_info = {
    /* sensor id register address */
    .sensor_id_reg_addr = 0x0016,
    /* sensor id */
    .sensor_id = 0x0135,
  },
```

3.3.2.3.2 Power-up/-down sequence

Sensor power sequence is added in an array using the msm_sensor_power_setting structure in each user space sensor driver.

```
static struct msm_sensor_power_setting power_setting[] = {
. . .
}
```

Both power-up and power-down sequences can be added in the msm_sensor_power_setting_array structure. If a power_down_setting/size_down member is not added as shown below, the power-down sequence will be the reverse of the power-up sequence.

```
.power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
 },
```

This power_setting will point to an array that has information on GPIO/CLK/VREG to be used to configure each sensor.

```
static struct msm_sensor_power_setting power_setting[] = {
  {
    .seq_type = SENSOR_VREG,
    .seq_val = CAM_VDIG,
    .config_val = 0,
    .delay = 0, //this delay is in ms
  },
  {
    .seq_type = SENSOR_VREG,
    .seq_val = CAM_VANA,
    .config_val = 0,
    .delay = 0, //this delay is in ms
  },
  . . .
```

This structure will be used in msm_camera_power_up() in the kernel to configure the sensor for the power-up sequence.

Refer to enums in kernel/include/media/msm_cam_sensor.h.

For YUV camera driver, msm_sensor_power_setting should be made in the kernel driver instead of the user space driver. For more details, refer to one of YUV reference drivers listed in Section 3.1.

Depending on the customer hardware design, the power source can be provided by PMIC or through GPIOs as shown below in the .dtsi file.

3.3.2.4 Dimensions table

A dimension table is added using the sensor_lib_out_info_t structure as shown in this example.

```
static struct sensor_lib_out_info_t sensor_out_info[] = {
{
/* full size @ 24 fps*/
.x_output = 4208,
.y_output = 3120,
.line_length_pclk = 4572,
.frame_length_lines = 3142,
.vt_pixel_clk = 360000000,
.op_pixel_clk = 360000000,
.binning_factor = 1,
.max_fps = 24.01,
.min_fps = 7.5,
.mode = SENSOR_DEFAULT_MODE,
},
```

- x_output – Active width
- y_output – Active height
- line_length_pclk – Width including blanking
- frame_length_lines – Height including blanking
- vt_pixel_clk(video timing clk value) – Virtual clock value used for calculating shutter time, and used by AEC for correcting banding artifacts

  vt_pixel_clk = line_length_pclk * frame_length_lines * frame rate
- op_pixel_clk – Represents how much data comes out of the camera over MIPI lanes to set the VFE clock

  op_pixel_clk = (total data rate from sensor)/bits-per-pixel

  For example, if the MIPI DDR clock value (speed of the clock lane of the MIPI camera sensor) is 300 MHz, and the sensor transmits on 4 lanes, each lane has a 600 MHz data rate. Thus, the total data rate is 2400 MHz. For 10 bits per pixel Bayer data, this translates to the op_pixel_clk value of 2400/10 = 240 MHz. These values must be filled in accordance with the sensor specifications.

  These values can be calculated based on the register settings configured for the camera sensor.

3.3.2.5 Chromatix parameters

The Chromatix parameters must be updated for each resolution with the proper header. Chromatix header generation is discussed in [Q4].

```
    static struct sensor_lib_chromatix_t imx135_chromatix[] = {
         {
.common_chromatix = IMX135_LOAD_CHROMATIX(common),
.camera_preview_chromatix = IMX135_LOAD_CHROMATIX(snapshot), /* RES0 */
.camera_snapshot_chromatix = IMX135_LOAD_CHROMATIX(snapshot), /* RES0 */
.camcorder_chromatix = IMX135_LOAD_CHROMATIX(default_video), /* RES0 */
.liveshot_chromatix = IMX135_LOAD_CHROMATIX(liveshot), /* RES0 */
          },
       }
```

3.3.2.6 Sensor register address

The register address fields for the exposure and output dimension must be filled with the proper values based on the sensor datasheet.

3.3.2.6.1 Exposure register address

```
static struct msm_sensor_exp_gain_info_t exp_gain_info = {
 .coarse_int_time_addr = 0x0202,
 .global_gain_addr = 0x0205,
 .vert_offset = 4,
};
```

- vert_offset – The integration line count should always be less than frame_length_lines by this margin when we configure to the max.

NOTE: Refer to the datasheet to get this offset (margin for coarse integration time).

3.3.2.6.2 Output register address

Register addresses for sensor cutout (x_output and y_output), frame_length_lines, and line_length_pclk must be configured based on the register specs in the camera sensor datasheet.

```
static struct msm_sensor_output_reg_addr_t output_reg_addr = {
  .x_output = 0x034C,
  .y_output = 0x034E,
  .line_length_pclk = 0x0342,
  .frame_length_lines = 0x0340,
};
```

3.3.2.7 MIPI receiver configuration

The sensor streams the imaging data in the MIPI CSI2 protocol. The QTI receiver receives it via MIPI CSI PHY and CSID.

3.3.2.7.1 CSI-PHY config

This structure shows CSI lane parameters.

```
static struct csi_lane_params_t csi_lane_params = {
  .csi_lane_assign = 0x4320,
  .csi_lane_mask = 0x1F,
  .csi_if = 1,
  .csid_core = { 0 },
  .csi_phy_sel = 0,
};

static struct msm_camera_csi2_params imx135_csi_params = {
  .csid_params = {
    .lane_cnt = 4,
    .lut_params = {
      .num_cid = ARRAY_SIZE(imx135_cid_cfg),
      .vc_cfg = {
        &imx135_cid_cfg[0],
        &imx135_cid_cfg[1],
        &imx135_cid_cfg[2],
      },
    },
  },
  .csiphy_params = {
    .lane_cnt = 4,
    .settle_cnt = 0x1B,
    .combo_mode = 1, //Present on recent builds
  },
};
```

- csi_lane_assign – Sometimes customer hardware may be designed with different pin mapping compared to the MSM™ chipset's reference pin map for camera data lanes, e.g., sensor data lane 0 may be connected to MSM data lane 4. The csi_lane_assign parameter can be configured to address such cases. This is a 16-bit value, with the meaning of each bit field presented as follows:

Bit position	Represents
15:12	MSM side PHY lane number, where sensor's data lane 3 is connected
11:8	MSM side PHY lane number, where sensor's data lane 2 is connected
7:4	MSM side PHY lane number, where sensor's data lane 1 is connected
3:0	MSM side PHY lane number, where sensor's data lane 0 is connected

NOTE: Lane 1 is reserved for the clock. Customers should not use this lane for mapping any data lanes.

- csi_lane_mask – This 8-bit field indicates which MIPI lanes are valid and enabled.

When a single camera is connected on the combo PHY, the value would be interpreted as:

Bit position	Represents
7:5	Reserved
4	Is data lane 3 valid? ▪ 0 – No ▪ 1 – Yes
3	Is data lane 2 valid? ▪ 0 – No ▪ 1 – Yes
2	Is data lane 1 valid? ▪ 0 – No ▪ 1 – Yes
1	Is clock lane valid? ▪ 0 – No ▪ 1 – Yes **Note:** This should always be set to 1.
0	Is data lane 0 valid? ▪ 0 – No ▪ 1 – Yes

For example, value 0x1F indicates that the camera has 4 valid data lanes and a clock lane.

When two cameras are connected (Cam0: 2 lanes camera connected to lanes 2:0; Cam1: 1 lane camera connected on lanes 4:3), the value would be interpreted as:

Bit position	Represents
7:5	Reserved
4	Is data lane 1 of Cam1 valid? ▪ 0 – No ▪ 1 – Yes
3	Is clock lane of Cam1 valid? ▪ 0 – No ▪ 1 – Yes **Note:** When camera is present, this should always be set to 1.
2	Is data lane 1 of Cam0 valid? ▪ 0 – No ▪ 1 – Yes
1	Is clock lane of Cam0 valid? ▪ 0 – No ▪ 1 – Yes **Note:** When camera is present, this should always be set to 1.
0	Is data lane 0 of Cam0 valid? ▪ 0 – No ▪ 1 – Yes

- csi_if – Not used
- csid_core – An index of the CSID hardware used by the sensor; two sensors operating concurrently cannot use the same value for this setting.
- csi_phy_sel – An index of the CSI-PHY core used by the sensor; should be unique for each sensor, unless an external MIPI bridge connects two sensors to the same PHY interface on the MSM. This value should be configured based on the customer's hardware schematic diagram.
- lane_cnt – The number of data lanes on which the sensor outputs data for a given mode of operation; this value is determined by the sensor's maximum data lanes capability (given in the datasheet) along with sensor register settings configured in the driver.
- combo_mode – Value 0 indicates that one camera is connected to the given PHY interface. Value 1 indicates that two cameras are sharing this PHY interface, in which case:
  - First camera – This is connected to PHY lanes 2:0. Clock lane is connected to PHY lane 1. Data lanes (up to 2) can be connected to either of PHY lanes 0 or 2.
  - Second camera – This is connected to PHY lanes 4:3. Clock lane is connected to PHY lane 3. Data lane can be connected to PHY lane 4.

- settle_cnt – This value must be configured, based on the sensor's output characteristics, to ensure the sensor's PHY transmitter does not have sync issues with the MSM's PHY receiver.

```
MIN [ Settle count * T(Timer clock)]  > T(HS_SETTLE)_MIN
MAX [ Settle count * T(Timer clock)]  < T(HS-PREPARE)+T(HS_ZERO) -
4*T(Timer clock)
```

Also, the customer can specify the settings for each operation mode separately as shown in the following example:

```
static struct msm_camera_csi2_params* csi_params[] = {
  &imx135_csi_params, /* RES 0 */
  &imx135_csi_params, /* RES 1 */
  &imx135_csi_params, /* RES 2 */
  &imx135_csi_params, /* RES 3 */
  &imx135_csi_params, /* RES 4 */
  &imx135_csi_params, /* RES 5 */
  &imx135_csi_params, /* RES 6 */
};
```

3.3.2.7.2 VFE clock rate calculation

Operating frequency of the clk lane of camera sensor (aka MIPI DDR clk) and number of active data lanes determine total data rate (throughput) from camera sensor for a given mode of operation. Data rate per lane is double the speed of MIPI DDR clk. For example, camera sensor operating at MIPI DDR clk of 200 MHz and 4 active data lanes would have total data rate of 1600 Mbps (200 * 2 = 400 Mbps dara rate per lane).

Data rate varies based on resolution of each frame, extra/dummy pixels/lines, H blanking, V blanking, MIPI packet overhead, bits per pixel, data format, whether there are multiple data streams interleaved within and data rate/overhead characteristics of each stream, etc. For initial camera bringup in a given mode of operation, calculate:

*X = Frame width * (Frame height * V blank) * bits per pixel * frames per second * (overhead from MIPI protocol and other streams)*

Find the closest value in VFE clk plan for the given MSM that is higher than X and use that as the initial value for VFE clk.

3.3.2.7.3 CSI-D configuration

```
static struct msm_camera_csid_vc_cfg imx135_cid_cfg[] = {
  { 0, CSI_RAW10, CSI_DECODE_10BIT },
  { 1, 0x35, CSI_DECODE_8BIT },
  { 2, CSI_EMBED_DATA, CSI_DECODE_8BIT}
};
```

The first entry for each row above is called CID (Channel ID). Each unique combination of Virtual Channel (VC) and Data Type (DT) should map to a unique CID value. Possible CID values for a given VC are:

- 0 – 0, 1, 2, 3
- 1 – 4, 5, 6, 7
- 2 – 8, 9, 10, 11
- 3 – 12, 13, 14, 15

In the imx135_cid_cfg example above, all three streams with data types CSI_RAW10, 0x35, CSI_EMBED_DATA are sent by the camera sensor in VC 0. Thus, the CID values are within the range 0 to 3.

3.3.2.8 Register settings

The camera sensor registers are configured for the streaming via I2C. The sensor can also be configured for other specific operations as described in this section.

3.3.2.8.1 Init settings

Perform a one-time register config at camera startup.

```
static struct msm_camera_i2c_reg_setting init_reg_setting[] = {
. . .
}
```

3.3.2.8.2 Grouphold on settings

The runtime updates for exposure settings are assigned to many registers (coarse integration time, framelengthlines, gain) and they must be changed within one frame period of the image. These registers are double buffered type and have the “grouped parameter hold” function to behave to be updated at once.When the “grouped parameter hold” register is set to 1, the transmitted data are held in the buffer registers.

```
static struct msm_camera_i2c_reg_setting groupon_settings = {
. . .
}
```

3.3.2.8.3 Grouphold off settings

When the "grouped parameter hold" register is set to 0, the exposure register values are updated as if they are transmitted at the same time and realize a smooth transition of parameter changes at the frame boundary.

```
static struct msm_camera_i2c_reg_setting groupoff_settings = {
. . .
}
```

3.3.2.8.4 Resolution settings

The sensor can be operated in multiple modes, e.g., Preview with quarter resolution and Snapshot with full resolution. The different resolutions can be configured as shown below.

```
static struct msm_camera_i2c_reg_setting res0 _settings[] = {
. . .
}
static struct msm_camera_i2c_reg_setting res1_settings[] = {
. . .
}
```

3.3.2.8.5 Exposure settings

For the AEC algorithm, real gain is used. For sensor hardware real gain must be converted to register gain in order to configure the sensor. The customer must implement the following functions for that purpose, based on the sensor datasheet. Refer to the analog and digital gain setting section in the sensor datasheet.

```
xxxx_real_to_register_gain()
xxxx_register_to_real_gain()
```

This function calculates the next exposure configuration for both exposure time and gain.

```
xxxx_calculate_exposure()
```

This function prepares the next exposure configure array.

```
xxxx_fill_exposure_array()
```

NOTE: Refer to reference drivers.

3.3.2.8.6 Start streaming settings

A MIPI data packet is bounded by Start of Transmission (SoT) and End of Transmission (EoT).

- SoT – LP11→LP01→LP00
- EoT – LP00→LP11

In the sensor driver, the following settings should be configured to match this guideline.

Take the clock and data lanes from LP11 to HS Tx state:

```
static struct msm_camera_i2c_reg_array start_reg_array[] = {
  { 0x0100, 0x01 },
};
```

3.3.2.8.7 Stop streaming settings

The stop streaming settings should put clock and data lanes of the sensor in the LP11 state. Not doing this correctly can cause an inconsistency in MIPI camera sync with the MSM.

```
static struct msm_camera_i2c_reg_array stop_reg_array[] = {
  { 0x0100, 0x00 },
};.
```

3.3.3 Using QUP/SPI

For the BLSP configuration, see [Q2]. Although this document is not specifically for MSM8916, the same concept is applicable. For chipsets that don't have CCI hardware, QUP is used to perform I2C transactions with camera.

NOTE: Customers needing to configure QUP/SPI on chipset with CCI hardware must contact QTI.

3.3.4 Updating CCI operation speed

Following I2C specification, CCI (for chipsets where it is present) can operate between 100 and 400 kHz and is controlled by a parameter named i2c_freq_mode within the structure msm_camera_sensor_slave_info. Refer to the IMX135 sensor sensor driver file $(MM_CAMERA_DIR)/ mm-camera2/media-controller/modules/sensors/sensor_libs/ imx135/imx135_lib.c for an example.

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {

  . . .

  /*sensor i2c frequency*/
.i2c_freq_mode = I2C_FAST_MODE,

. . .

};
```

The available I2C frequency modes are defined in kernel/include/media/msm_cam_sensor.h: standard (100 kHz), fast (400 kHz), and Custom mode.

```
enum i2c_freq_mode_t {
    I2C_STANDARD_MODE,
    I2C_FAST_MODE,
    I2C_CUSTOM_MODE,
    I2C_MAX_MODES,
};
```

For Custom mode, in the case of MSM8916, the CCI tuning parameters setting information can be found in kernel/arch/arm/boot/dts/qcom/msm8916-camera.dtsi.

```
&i2c_freq_custom {
    qcom,hw-thigh = <15>;
    qcom,hw-tlow = <28>;
    qcom,hw-tsu-sto = <21>;
    qcom,hw-tsu-sta = <21>;
    qcom,hw-thd-dat = <13>;
    qcom,hw-thd-sta = <18>;
    qcom,hw-tbuf = <25>;
    qcom,hw-scl-stretch-en = <1>;
    qcom,hw-trdhld = <6>;
    qcom,hw-tsp = <3>;
    status = "ok";
};
```

If a custom CCI configuration is used then the speed of I2C frequency information can be calculated by the formula:

CCI clock = (src clock) / (hw_thigh + hw_tlow)

Because the CCI clock frequency is typically 19.2 MHz, for the standard CCI frequency case, it 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 can be set to either 100 or 400 kHz from the .dtsi file. For more details, see entry qcom,clk-freq-out of node i2c_3 in file kernel/arch/arm/boot/dts/qcom/msm8909.dtsi at Section 8.1.

4 AF Actuator Driver

This chapter provides guidelines to customers who write their own AF actuator driver by looking at the reference AF actuator driver provided in the base build.

4.1 AF actuator driver directory structure

Customers who want to write a new AF actuator driver can refer to the following reference AF actuator driver. This example involves actuator dw9714, which is used by the imx135 sensor driver in the base build.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuator_libs/dw9714/
  - Android.mk
  - dw9714_actuator.c
  - dw9714_actuator.h
- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuators/0301/dw9714/
  - Android.mk
  - camcorder/
    - Android.mk
    - dw9714_camcorder.c
    - dw9714_camcorder.h
  - camera/
    - Android.mk
    - dw9714_camera.c
    - dw9714_camera.h
- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

4.2 Files to update/add

The following specified files must be updated or modified to add a new AF actuator driver.

4.2.1 Updating a device tree file

In the target's camera .dtsi file, e.g., msm8916-camera-sensor-mtp.dtsi, add an entry for the actuator node and assign qcom,actuator-src with actuator node.

- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

```
&cci{
        actuator0: qcom,actuator@6e {
                cell-index = <3>;
                reg = <0x6c>;
                compatible = "qcom,actuator";
                qcom,cci-master = <0>;
        };
        qcom,camera@0 {
                qcom,actuator-src = <&actuator0>;
        };
}
```

4.2.2 Setting up AF actuator power

Refer to each reference chipset platform DTSI file to specify the AF actuator power because the correct specification may be slightly different. In MSM8916, the cam_vaf-supply is specified with camera sensor node, however, in MSM8992/MSM8994, it is specified in MSM8992/MSM8994 actuator node:

Here is an example for MSM8916:

Note that the the power supply of AF is specified together with the camera sensor and it is the fourth entry in the list of each vreg name, type, min-voltage, max-voltage and op-mode.

```
&cci {
   ...
   qcom,camera@0 {
        ...
        cam_vdig-supply = <&pm8916_s4>;
        cam_vana-supply = <&pm8916_l17>;
        cam_vio-supply = <&pm8916_l6>;
        cam_vaf-supply = <&pm8916_l10>;
        qcom,cam-vreg-name = "cam_vdig", "cam_vio", "cam_vana",
                             "cam_vaf";
        qcom,cam-vreg-type = <0 1 0 0>;
        qcom,cam-vreg-min-voltage = <2100000 0 2850000 2800000>;
        qcom,cam-vreg-max-voltage = <2100000 0 2850000 2800000>;
```

```
        qcom,cam-vreg-op-mode = <200000 0 80000 100000>;
```

Here is an example for MSM8992/MSM8994:

The voltage regulator (PMIC) is specified under “qcom,actuator” device tree node.

```
&cci {
   actuator0: qcom,actuator@0 {
        ...
        cam_vaf-supply = <&pm8994_l23>;
        qcom,cam-vreg-name = "cam_vaf";
        qcom,cam-vreg-min-voltage = <2800000>;
        qcom,cam-vreg-max-voltage = <2800000>;
        qcom,cam-vreg-op-mode = <100000>;
   };
```

4.2.3 Adding optional GPIO control pin for actuator

In some cases, the GPIO pin can be used for the actuator power source instread of the default voltage regulator from PMIC. The voltage regulator node information is covered in the previous section and should be deleted first, then the following DTSI device tree properties need to be added:

Here is an example for MSM8916:

```
   &cci{
        qcom,camera@0 {
             qcom,actuator-src = <&actuator0>;
             ...
             qcom,actuator-vcm-pwd = <gpio number for VCM power>;
             qcom,actuator-vcm-enable = <gpio number for VCM eanble>;        };
   }
```

In case of MSM8916, the GPIO call sequence (i.e. gpio_set_value_cansleep) can be found in msm_actuator_power_up() / msm_actuator_power_down().

Here is an example for MSM8992/MSM8994:

```
   &cci{
           qcom,camera@0 {
                 ...
                 gpios = <&msm_gpio 27 0>, ...
                 ...
                 qcom,gpio-vaf = <0>; /* The first array item */
                 ...
                 qcom,gpio-req-tbl-label = "CAM_VAF",
                 ...
```

```
        };
}
```

For MSM8992/MSM8994, the GPIO for actuator ("qcom,gpio-vaf") is referred by the index of in the GPIO list of camera sensor node. In the camera sensor power sequence, for example, msm_camera_power_up/down), the actuator power will be enabled or disabled.

4.2.4 Updating a sensor driver file

Considering the imx135_lib.c driver file as an example, the actuator_name field in the sensor_lib_t structure must be populated to associate an actuator driver with the sensor.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/imx135/imx135_lib.c

```
static sensor_lib_t sensor_lib_ptr = {
  /* sensor slave info */
  .sensor_slave_info = &sensor_slave_info,
  /* sensor init params */
  .sensor_init_params = &sensor_init_params,
  /* sensor actuator name */
.actuator_name = "dw9714",
```

4.2.5 Adding AF actuator files

The following <actuator>_actuator.c and <actuator>_actuator.h files must be added for a new actuator driver.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuator_libs/<actuator>/
  - Android.mk
  - <actuator>_actuator.c
  - <actuator>_actuator.h

The <actuator>_actuator.c file is a simple layer that returns actuator_lib_ptr.

```
#include "actuator_driver.h"

static actuator_driver_ctrl_t actuator_lib_ptr = {
#include "<actuator>__actuator.h"
};

void *actuator_driver_open_lib(void)
{
  return &actuator_lib_ptr;
}
```

The <actuator>_actuator.h file should have the following params defined for actuator tuning. Some params in this file are derived from the actuator datasheet and some are a result of the AF tuning performed.

```
{
    /* actuator_params_t */
    {
      /* module_name */
      "abico",
      /* actuator_name */
      "dw9714",
      /* 8 bit i2c_addr */
      0x18,
      /* i2c_data_type. Check actuator data sheet for i2c data type */
      MSM_ACTUATOR_BYTE_DATA/MSM_ACTUATOR_WORD_DATA,
      /* i2c_addr_type Check actuator data sheet for i2c addr type*/
      MSM_ACTUATOR_BYTE_ADDR/MSM_ACTUATOR_WORD_ADDR,
      /* act_type */
      ACTUATOR_VCM/ACTUATOR_PIEZO,
      /* data_size */
      10,
      /* af_restore_pos */
      0,
      /* msm_actuator_reg_tbl_t */
      {
        /* reg_tbl_size */
        1,
        /* msm_actuator_reg_params_t */ Check the actuator data sheet for
eeprom current programing method.
        {
          /* reg_write_type;hw_mask; reg_addr; hw_shift >>; data_shift << */
          {MSM_ACTUATOR_WRITE_DAC, 0x0000000F, 0xFFFF, 0, 4},
        },
      }
      /* init_setting_size */
      0,
      /* init_settings */
      {
   /* Check actuator data sheet for init_setting register write sequence .
It shall look like the e.g. given below*/
   /* reg_addr, addr_type, reg_data, data_type, i2c_operation, delay/*
        {0xEC, MSM_ACTUATOR_BYTE_ADDR, 0xA3, MSM_ACTUATOR_BYTE_DATA,
MSM_ACT_WRITE, 0},
      },
    }, /* actuator_params_t */
```

```
Following actuator_tune_params_t are derived from the AF tuning.
    /* actuator_tuned_params_t */
    {
      /* scenario_size */
      {
        /* scenario_size[MOVE_NEAR] */
        2,
        /* scenario_size[MOVE_FAR] */
        3,
      },

      /* ringing_scenario */
      {
        /* ringing_scenario[MOVE_NEAR] */
        {
          4,
          41,
        },
        /* ringing_scenario[MOVE_FAR] */
        {
          8,
          22,
          41,
        },
      },

      /* intial_code */ This represent the initial step after which actuator
move the lens position linearly.
      0,
      /* region_size */ This number represent into how many region the
total lens step sizes are divided into.
      2

      /* region_params */ Following regions are devided into number specified
in region_size.
      {
        /* step_bound[0] - macro side boundary */
        /* step_bound[1] - infinity side boundary */
        /* Region 1 */
        {
          .step_bound = {2, 0},
          .code_per_step = 85,
        },
        /* Region 2 */
        {
```

```
    .step_bound = {41, 2},
    .code_per_step = 9,
  },
}
{
  /* damping */
  {
    /* damping[MOVE_NEAR] */
    {
      /* scenario 1 */
 /* Number of scenarios depends on the size defined in scenario_size */
      {
        /* region 1 */
        {
          .damping_step = 0x1FF,
          .damping_delay = 7000,
          .hw_params = 0x0000000C,
        },
        /* region 2 */
        {
          .damping_step = 0x1FF,
          .damping_delay = 7000,
          .hw_params = 0x7,
        }
        ,
      }
      ,
    }
  },

  {
    /* damping[MOVE_FAR] */
    {
      /* scenario 1 */
      {
        /* region 1 */
        {
          .damping_step = 0x1FF,
          .damping_delay = 7000,
          .hw_params = 0x0000000C,
        },
        /* region 2 */
        {
          .damping_step = 0x1FF,
```

```
                .damping_delay = 4500,
                .hw_params = 0x0000007,
              },
            },
          },
        },
      },
    }, /* actuator_tuned_params_t */
  },
```

4.2.6 Adding AF algorithm tuning files

The following AF algorithm tuning files must be added for Camera and Camcorder modes. For information on AF algorithm tuning, see *Presentation: Camera Auto Focus Tuning Guide* (80-N8459-1).

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuators/0301/<actuator>/
  - Android.mk
  - camcorder/
    - Android.mk
    - <actuator>_camcorder.c
    - <actuator>_camcorder.h
  - camera/
    - Android.mk
    - <actuator>_camera.c
    - <actuator>_camera.h

5 LED Flash Driver

This chapter provides guidelines to customers who write their own LED Flash driver by looking at a reference LED Flash driver provided in the base build.

5.1 LED Flash driver directory structure

The reference LED Flash driver can be PMIC-based, QUP/I2C-based or CCI-based (for chipsets having CCI hardware block). This example uses the LED Flash driver adp1660.c file.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/led_flash/
  - led_flash.c
  - led_flash.h
- kernel/drivers/media/platform/msm/camera_v2/sensor/flash/
  - Makefile
  - adp1660.c – QUP/I2C-based LED Flash driver
  - msm_led_flash.c – Generic LED Flash driver that handles user space IOCTLs
  - msm_led_flash.h – Generic LED Flash driver header file
  - msm_led_i2c_trigger.c – QUP/I2C- or CCI-based LED Flash interface driver
  - msm_led_torch.c – PMIC-based LED torch driver
  - msm_led_trigger.c – PMIC-based LED Flash driver
- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/,e.g., msm8916-camera-sensor-mtp.dtsi

5.2 Files to be modified

The following specified files must be updated or modified to add a new LED Flash driver.

5.2.1 Updating a device tree file

In the target camera .dtsi file, e.g., msm8916-camera-sensor-mtp.dtsi, add an entry for led_flash node and assign qcom,led-flash-src with led_flash node.

- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/e.g., msm8916-camera-sensor-mtp.dtsi

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

Node entry in the device tree file will change based on the type of LED Flash driver – PMIC-based, I2C-based, or CCI-based.

For more details and explanation of each field in device tree file, refer to the kernel at kernel/Documentation/devicetree/bindings/media/video$ vi msm-camera-flash.txt.

PMIC-based LED Flash driver

```
&soc {
    led_flash0: qcom,camera-led-flash {
         cell-index = <0>;
         compatible = "qcom,camera-led-flash";
         qcom,flash-type = <1>;
         qcom,torch-source = <&pm8941_torch>;
         qcom,flash-source = <&pm8941_flash0 &pm8941_flash1>;
    };
};
```

QUP-/I2C-based LED Flash driver

```
&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";
};
};

&cci {
```

CCI-based LED Flash driver

```
    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>;
    };
```

NOTE: For any of the three LED Flash drivers above, the following entry must be added to associate it with the sensor:

```
qcom,camera@0 {
        qcom,led-flash-src = <&led_flash0>;
};
};
```

For QUP-/I2C-based entry into the camera .dtsi file, the &i2c node shall be defined earlier in the target .dtsi file, e.g., msm8916-mtp.dtsi. The following snippet of the example code shows which QUP node is connected to &i2c. To create a new QUP device node, see *Guide to Configuring and Debugging BAM Low-speed Peripherals for Linux Kernel for MSM8974, MSM8x26, MDM9x25, APQ8084 Chipsets* (80-NE436-1).

```
i2c: i2c@f9928000 { /* BLSP1 QUP6 */
        cell-index = <6>;
        compatible = "qcom,i2c-qup";
        #address-cells = <1>;
        #size-cells = <0>;
        reg-names = "qup_phys_addr";
        reg = <0xf9928000 0x1000>;
        interrupt-names = "qup_err_intr";
        interrupts = <0 100 0>;
        qcom,i2c-bus-freq = <100000>;
        qcom,i2c-src-freq = <19200000>;
        qcom,sda-gpio = <&msmgpio 16 0>;
        qcom,scl-gpio = <&msmgpio 17 0>;
        qcom,master-id = <86>;
};
```

For a PMIC-based LED Flash driver, flash-source and torch-source params in the camera .dtsi file supplied by PMIC IC and their handler are defined in kernel/arch/arm/boot/dts/<target>-leds.dtsi, e.g., msm8916-leds.dtsi.

```
qcom,leds@d300 {
   status = "okay";
   pm8941_flash0: qcom,flash_0 {
        qcom,max-current = <1000>;
        qcom,default-state = "off";
        qcom,headroom = <3>;
        qcom,duration = <1280>;
        qcom,clamp-curr = <200>;
        qcom,startup-dly = <3>;
        qcom,safety-timer;
        label = "flash";
        linux,default-trigger =
             "flash0_trigger";
        qcom,id = <1>;
        linux,name = "led:flash_0";
        qcom,current = <625>;
   };
```

```
        pm8941_flash1: qcom,flash_1 {
             qcom,max-current = <1000>;
             qcom,default-state = "off";
             qcom,headroom = <3>;
             qcom,duration = <1280>;
             qcom,clamp-curr = <200>;
             qcom,startup-dly = <3>;
             qcom,safety-timer;
             linux,default-trigger =
                  "flash1_trigger";
             label = "flash";
             qcom,id = <2>;
             linux,name = "led:flash_1";
             qcom,current = <625>;
        };

        pm8941_torch: qcom,flash_torch {
             qcom,max-current = <200>;
             qcom,default-state = "off";
             qcom,headroom = <0>;
                  qcom,startup-dly = <1>;
                  linux,default-trigger =
                       "torch_trigger";
                  label = "flash";
                  qcom,id = <2>;
                  linux,name = "led:flash_torch";
                  qcom,current = <200>;
                  qcom,torch-enable;
             };
        };
    . . .
    };
```

5.2.2 Changing GPIO pin number for CCI-based case

If GPIO pin numbers must be changed, change the following two places:

- kernel/arch/arm/boot/dts/<target>-camera-sensor-mtp.dtsi (e.g., msm8916-camera-sensor-mtp.dtsi):

```
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>;
```

The following configuration can be changed based on the hardware schematic design (GPIO 36, 31, 32):

```
gpios = <&msm_gpio 36 0>,
  <&msm_gpio 32 0>,
  <&msm_gpio 31 0>;
```

- kernel/arch/arm/boot/dts/<target>-camera-sensor-mtp.dtsi (e.g., msm8916-camera-sensor-mtp.dtsi):

```
cam_sensor_flash {
    /* FLSH_RESET,FLASH_EN,FLASH_NOW */
    qcom,pins = <&gp 36>, <&gp 31>,<&gp 32> ;
    qcom,num-grp-pins = <3>;
    qcom,pin-func = <0>;
    label = "cam_sensor_flash";
    /* active state */
    cam_sensor_flash_default: default {
         drive-strength = <2>; /* 2 MA */
         bias-disable = <0>; /* No PULL */
    };
    /*suspended state */
    cam_sensor_flash_sleep: sleep {
         drive-strength = <2>; /* 2 MA */
         bias-pull-down = <0>; /* PULL DOWN */
    };
};
```

The following configuration can be changed based on the hardware schematic design (GPIO 36, 31, 32):

```
qcom,pins = <&gp 36>, <&gp 31>,<&gp 32> ;
```

5.2.3 Adding an LED Flash driver file

For QUP/I2C-based and CCI-based LED Flash drivers, add the <led>.c file, e.g., adp1660.c, in the following directory:

- FILE – kernel/drivers/media/platform/msm/camera_v2/sensor/flash/
  - <led>.c

The following data structures and functions must be implemented in the <led>.c driver file.

For a QUP-/I2C-based LED Flash driver, the i2c_driver structure and probe function must be defined as follows:

```
static struct i2c_driver <led>_i2c_driver = {
        .id_table = <led>_i2c_id,
        .probe  = msm_flash_adp1660_i2c_probe,
        .remove = __exit_p(msm_flash_<led>_i2c_remove),
        .driver = {
                .name = FLASH_NAME,
                .owner = THIS_MODULE,
                .of_match_table = <led>_trigger_dt_match,
        },
};

static int msm_flash_<led>_i2c_probe(struct i2c_client *client,
                const struct i2c_device_id *id)
{
        if (!id) {
                pr_err("msm_flash_<led>_i2c_probe: id is NULL");
                id = <led>_i2c_id;
        }

        return msm_flash_i2c_probe(client, id);
}
```

For a CCI-based LED Flash driver, the platform_driver structure and probe function must be defined as follows:

```
static struct platform_driver <led>_platform_driver = {
        .probe = msm_flash_<led>_platform_probe,
        .driver = {
```

```
                .name = "qcom,led-flash",
                .owner = THIS_MODULE,
                .of_match_table = <led>_trigger_dt_match,
        },
};

static int msm_flash_<led>_platform_probe(struct platform_device *pdev)
{
        const struct of_device_id *match;
        match = of_match_device(<led>_trigger_dt_match, &pdev->dev);
        if (!match)
                return -EFAULT;
        return msm_flash_probe(pdev, match->data);
}
```

On successful probe, the msm_led_flash_ctrl_t structure is fully populated and a v4l2 node is created for an LED Flash driver.

```
static struct msm_led_flash_ctrl_t fctrl = {
        .flash_i2c_client = &<led>_i2c_client,
        .reg_setting = &<led>_regs,
        .func_tbl = &<led>_func_tbl,
};
```

The following function table should be initialized with appropriate functions from the msm_led_i2c_trigger.c LED Flash interface driver. This function table is the same for QUP-/I2C-based and CCI-based LED Flash drivers.

```
static struct msm_flash_fn_t <led>_func_tbl = {
        .flash_get_subdev_id = msm_led_i2c_trigger_get_subdev_id,
        .flash_led_config = msm_led_i2c_trigger_config,
        .flash_led_init = msm_flash_led_init,
        .flash_led_release = msm_flash_led_release,
        .flash_led_off = msm_flash_led_off,
        .flash_led_low = msm_flash_led_low,
        .flash_led_high = msm_flash_led_high,
};
```

- .flash_get_subdev_id() – This function returns the subdev_id of the client.
- .flash_led_config () – This is a handler function to handle a user space command to configure LED with different states, e.g., LOW, HIGH, OFF, RELEASE, etc.

- .flash_led_init() – In this function, LED Flash init_settings are written to the LED hardware. The LED Flash datasheet provides the sequence of register settings needed to write to the LED hardware.

  The following is an example init_setting struct. Refer to the LED Flash datasheet for reg_init setting, size of the init_array, addr_type, data_type, and delay.

```
static struct msm_camera_i2c_reg_setting <led>_init_setting = {
        .reg_setting = <led>_init_array,
        .size = ARRAY_SIZE(<led>_init_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
        .delay = 0,
};
```

  In case of CCI-based LED Flash, CCI hardware is also initialized in this function.

- .flash_led_release() – This function is called during the camera close sequence. All the GPIOs are turned off here, and in case of a CCI-based LED Flash driver, it will deinit the CCI h/w.
- .flash_led_off() – This function is called to turn off the LED from a LOW or HIGH state. In this function LED Flash driver off_settings are written to the LED hardware. For off_setting, refer to the LED Falsh driver datasheet.

  The following is an example of an off_setting struct. Refer to the LED Flash driver datasheet for off_settings, size of the off_array, addr_type, data_type, and delay.

```
static struct msm_camera_i2c_reg_setting <led>_off_setting = {
        .reg_setting = <led>_off_array,
        .size = ARRAY_SIZE(<led>_off_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
        .delay = 0,
};
```

- .flash_led_low() – This function is called to set the LED to a LOW state. In this function, LED Flash driver low_settings are written to the LED hardware. For low_setting, refer to the LED Flash driver datasheet.

  The following is an example of a low_setting struct. Refer to the LED Flash driver datasheet for low_settings, size of the low_array, addr_type, data_type, and delay.

```
static struct msm_camera_i2c_reg_setting <led>_low_setting = {
        .reg_setting = <led>_low_array,
        .size = ARRAY_SIZE(<led>_low_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
```

```
        .delay = 0,
};
```

- .flash_led_high() – This function is called to set the LED to a HIGH state. In this function, LED Flash driver high_settings are written to the LED hardware. For high_setting, refer to the LED Flash driver datasheet.

  The following is an example of a high_setting struct. Refer to the LED Flash datasheet for high_settings, size of the high_array, addr_type, data_type, and delay.

```
static struct msm_camera_i2c_reg_setting <led>_high_setting = {
        .reg_setting = <led>_high_array,
        .size = ARRAY_SIZE(<led>_high_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
        .delay = 0,
};
```

5.2.4 Adding PWM-based flash drivers

For adding PWM-based LED flash drivers, check the case system solution #00028369 – "How to use GPIO to generate PWM on MSM8926". For different targets, implement the changes mentioned in the case system solution #00028369 in a similar fashion.

6 EEPROM Driver

This chapter provides guidelines to customers who write their own EEPROM driver by looking at a reference EEPROM driver provided in the base build.

6.1 EEPROM driver directory structure

This example uses the EEPROM driver sunny_q5v22e.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/eeprom/
  - eeprom.c
  - eeprom.h
- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/eeprom_libs/sunny_q5v22e/
  - Android.mk
  - sunny_q5v22e_eeprom.c
- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

6.2 Files to be modified

The following specified files must be updated or modified to add a new EEPROM driver.

6.2.1 Updating a device tree file

In the target's camera .dtsi file, e.g., msm8916-camera-sensor-mtp.dtsi, add an entry for EEPROM node and assign qcom,eeprom-src with EEPROM node.

- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

The <target>_camera*.dtsi file must be updated with the following .dtsi fields. For explanations of each field in the device tree file, refer to the kernel at kernel/Documentation/devicetree/bindings/media/video/msm-eeprom.txt.

```
&cci {
                eeprom3: qcom,eeprom@6c {
                cell-index = <3>;
                reg = <0x6c>;
                qcom,eeprom-name = "sunny_q5v22e";
                compatible = "qcom,eeprom";
```

```
qcom,slave-addr = <0x20>;
qcom,cci-master = <0>;
qcom,num-blocks = <4>;
qcom,page0 = <1 0x0100 2 0x01 1 1>;
qcom,poll0 = <0 0x0 2 0 1 1>;
qcom,mem0 = <0 0x0 2 0 1 0>;
qcom,page1 = <1 0x3d84 2 0xc0 1 1>;
qcom,poll1 = <0 0x0 2 0 1 1>;
qcom,mem1 = <0 0x3d00 2 0 1 0>;
qcom,page2 = <1 0x3d88 2 0x7010 2 1>;
qcom,poll2 = <0 0x0 2 0 1 1>;
qcom,mem2 = <0 0x3d00 2 0 1 0>;
qcom,page3 = <1 0x3d8A 2 0x70F4 2 1>;
qcom,pageen3 = <1 0x3d81 2 0x01 1 10>;
qcom,poll3 = <0 0x0 2 0 1 1>;
qcom,mem3 = <228 0x7010 2 0 1 1>;

cam_vdig-supply = <&pm8226_l5>;
cam_vana-supply = <&pm8226_l19>;
cam_vio-supply = <&pm8226_lvs1>;
qcom,cam-vreg-name = "cam_vdig","cam_vana", "cam_vio";
qcom,cam-vreg-type = <0 1 2>;
qcom,cam-vreg-min-voltage = <1200000 2850000 0>;
qcom,cam-vreg-max-voltage = <1200000 2850000 0>;
qcom,cam-vreg-op-mode = <200000 80000 0>;
qcom,gpio-no-mux = <0>;
gpios = <&msmgpio 26 0>,
        <&msmgpio 37 0>,
        <&msmgpio 36 0>;
qcom,gpio-reset = <1>;
qcom,gpio-standby = <2>;
qcom,gpio-req-tbl-num = <0 1 2>;
qcom,gpio-req-tbl-flags = <1 0 0>;
qcom,gpio-req-tbl-label = "CAMIF_MCLK",
        "CAM_RESET1",
        "CAM_STANDBY";
qcom,cam-power-seq-type = "sensor_vreg","sensor_vreg",
        "sensor_vreg", "sensor_clk",
        "sensor_gpio", "sensor_gpio";
qcom,cam-power-seq-val = "cam_vdig","cam_vana",
        "cam_vio", "sensor_cam_mclk",
        "sensor_gpio_reset",
        "sensor_gpio_standby";
qcom,cam-power-seq-cfg-val = <1 1 1 24000000 1 1>;
qcom,cam-power-seq-delay = <1 1 1 5 5 10>;
```

```
        };


        qcom,camera@20 {
                qcom,eeprom-src = <&eeprom3>;
       };
};
```

6.2.2 Updating a sensor driver file

Considering the ov5648_lib.c driver file as an example, the eeprom_name field in the sensor_lib_t structure must be populated to associate a EEPROM driver with the sensor.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/ov5648_q5v22e/ ov5648_q5v22e_lib.c

```
static sensor_lib_t sensor_lib_ptr = {
  /* sensor slave info */
  .sensor_slave_info = &sensor_slave_info,
  /* sensor init params */
  .sensor_init_params = &sensor_init_params,
 /* sensor eeprom name */
  .eeprom_name = "sunny_q5v22e",
```

6.2.3 Adding a EEPROM driver file

The following <eeprom>.c file must be added for a new EEPROM driver.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/eeprom_libs/eeprom/
  - Android.mk
  - <eeprom>.c

  Any new <eeprom>.c file should have the following function pointers mapped and defined in it. Any of these functions not defined in that EEPROM driver must be set to NULL.

```
static eeprom_lib_func_t <eeprom>_lib_func_ptr = {

  .get_calibration_items = NULL,
  .format_calibration_data = NULL,
  .do_af_calibration = NULL,
  .do_wbc_calibration = NULL,
  .do_lsc_calibration = NULL,
  .do_dpc_calibration = NULL,
  .get_dpc_calibration_info = NULL,
  .get_raw_data = NULL,
};
```

- .get_calibration_items() – This function should return the configuration supported by the EEPROM module. Based on the configuraton supported in EEPROM, that specific flag is set to TRUE or FALSE.

```
void <eeprom>_get_calibration_items(void *e_ctrl)
{
  sensor_eeprom_data_t *ectrl = (sensor_eeprom_data_t *)e_ctrl;
  eeprom_calib_items_t *e_items = &(ectrl->eeprom_data.items);
  e_items->is_insensor = TRUE;
  e_items->is_afc = FALSE;
  e_items->is_wbc = TRUE;
  e_items->is_lsc = TRUE;
  e_items->is_dpc = FALSE;
}
```

  - Is_insensor – This flag is set to TRUE if the EEPROM configuration is supported in the sensor module itself. No external EEPROM is available.
  - Is_afc – This flag is set to TRUE if it supports the AF calibration.
  - Is_wbc – This flag is set to TRUE if it supports the white balance calibration.
  - Is_lsc – This flag is set to TRUE if it supports the lens shading calibration.
  - Is_dpc – This flag is set to TRUE if it supports defect pixel correction.

- .format_calibration_data() – This function is used to format the data that can be written to the eeprom/sensor module. Check the eeprom/sensor datasheet for addr_type, data_type, and register settings. At the end of this function, reg_setting should have the data for wb, lsc, afc, dpc, etc. in eeprom/sensor-understandable format.

```
void <eeprom>_format_calibration_data(void *e_ctrl) {
  SLOW("Enter");
  sensor_eeprom_data_t *ectrl = (sensor_eeprom_data_t *)e_ctrl;
  uint8_t *data = ectrl->eeprom_params.buffer;

  g_reg_setting.addr_type = MSM_CAMERA_I2C_WORD_ADDR;
  g_reg_setting.data_type = MSM_CAMERA_I2C_BYTE_DATA;
  g_reg_setting.reg_setting = &g_reg_array[0];
  g_reg_setting.size = 0;
  g_reg_setting.delay = 0;
  <eeprom>_format_wbdata(ectrl);
  <eeprom>_format_lensshading(ectrl);

  SLOW("Exit");
}
```

- .do_af_calibration() – This function should handle any AF-related calibration operations, formatting data and writing to the EEPROM for AF calibration.
- .do_wbc_calibration() – This function should handle any white balance-related calibration operations, formatting data and writing to the EEPROM for white balance calibration.
- .do_lsc_calibration() – This function should handle any lens shading correction-related calibration operations, formatting data and writing to the EEPROM for lens shading calibration.
- .do_dpc_calibration() – This function should handle any defect pixel correction-related calibration operations, formatting data and writing to the EEPROM for defect pixel correction calibration.

7 Updates for MSM8992/MSM8994

This chapter describes the different sensor driver structure in the new MSM8992/MSM8994 code.

7.1 User space changes

In MSM8992/MSM8994.LA.1.1, the sensor driver structure has been reorganized. The main structure and the driver configuration remain largely same, but some naming conventions of structure members have been changed, notably for “array” type with postfix “_a”.

Also, the driver file style has been changed so that most of the data information has moved to its header file. For example, file ov4688_lib.c is now a short file with a handle of functions only, e.g.:

- sensor_real_to_register_gain()
- sensor_register_to_real_gain()
- sensor_calculuate_exposure()
- sensor_fill_exposure_array()
- sensor_open_lib()

Most of the data structure mentioned in Section 3.3.2 moved to its corresponding header files. Check file ov4688_lib.h. In addition to the I2C register array structures, now it has a single consolidated data structure, sensor_lib_ptr. In fact, the nested structure can be constructed by multiple assignments like the previous driver, which has equivalent data. For the minor change of the naming changes, refer to file vendor/qcom/proprietary/mm-camerasdk/sensor/includes/sensor_lib.h

```
typedef struct {
  /* sensor slave info */
  struct msm_camera_sensor_slave_info sensor_slave_info;
. . .
  /* resolution config table */
  struct sensor_res_cfg_table_t sensor_res_cfg_table;

  struct sensor_lib_out_info_array     out_info_array;
  struct sensor_lib_csi_params_array   csi_params_array;
  struct sensor_lib_crop_params_array  crop_params_array;
  struct sensor_lib_chromatix_array    chromatix_array;
```

For kernel header changes, refer the header file kernel/include/media/msm_camsensor_sdk.h.

```
struct msm_sensor_power_setting_array {
    struct msm_sensor_power_setting  power_setting_a[MAX_POWER_CONFIG];
    struct msm_sensor_power_setting *power_setting;
    uint16_t size;
    struct msm_sensor_power_setting
power_down_setting_a[MAX_POWER_CONFIG];
    struct msm_sensor_power_setting *power_down_setting;
    uint16_t size_down;
};

struct msm_camera_csid_lut_params {
    uint8_t num_cid;
    struct msm_camera_csid_vc_cfg vc_cfg_a[MAX_CID];
    struct msm_camera_csid_vc_cfg *vc_cfg[MAX_CID];
};
```

NOTE: New array type members power_setting_a and vc_cfg_a have been added.

8 Updates for MSM8909

This chapter describes the different driver structure in the MSM8909 code.

8.1 No CCI hardware

The major difference on MSM8909 is the absence of a CCI hardware block compared to other chipsets, MSM8916, MSM8974, etc. This means all camera driver-specific enries in dtsi file, e.g., kernel/arch/arm/boot/dts/qcom/msm8909-camera-sensor-mtp.dtsi, would be listed under legacy i2c node. For example:

```
&i2c_3 {

    actuator0: qcom,actuator@0 {
    . . .
    };

    eeprom0: qcom,eeprom@6c{
    . . .
    };

    led_flash0: qcom,led-flash@60 {
    . . .
    };

    qcom,camera@0 {
    . . .
    };

    qcom,camera@1 {
    . . .
    };

};
```

Reference to the node &i2c_3 indicates that BLSP1 QUP3 is used for camera operations on MSM8909. This node is defined in kernel/arch/arm/boot/dts/qcom/msm8909.dtsi:

```
i2c_3: i2c@78b7000 { /* BLSP1 QUP3 */
       compatible = "qcom,i2c-msm-v2";
       #address-cells = <1>;
       #size-cells = <0>;
       reg-names = "qup_phys_addr", "bam_phys_addr";
       reg = <0x78b7000 0x1000>,
             <0x7884000 0x23000>;
       interrupt-names = "qup_irq", "bam_irq";
       interrupts = <0 97 0>, <0 238 0>;
       clocks = <&clock_gcc clk_gcc_blsp1_ahb_clk>,
                <&clock_gcc clk_gcc_blsp1_qup3_i2c_apps_clk>;
       clock-names = "iface_clk", "core_clk";
       qcom,clk-freq-out = <100000>; //NOTE: This can be set to 400000 if
400 KHz frequency is required
       qcom,clk-freq-in  = <19200000>;
       pinctrl-names = "i2c_active", "i2c_sleep";
       pinctrl-0 = <&i2c_3_active>;
       pinctrl-1 = <&i2c_3_sleep>;
       qcom,noise-rjct-scl = <0>;
       qcom,noise-rjct-sda = <0>;
       qcom,bam-pipe-idx-cons = <8>;
       qcom,bam-pipe-idx-prod = <9>;
       qcom,master-id = <86>;
};
```

8.2 Reference drivers

MSM8909 MTP uses rear camera OV8858 (module part q8v19w). Thus, for driver code references, refer to ov8858_q8v19w_lib.c. For EEPROM, refer to sunny_ov8858_q8v19w_eeprom.c.

9 Troubleshooting

This chapter describes the debugging processes to follow should you encounter any errors upon creating or configuring the components for camera bringup. Troubleshooting should be performed in the order presented in this chapter.

9.1 Sensor troubleshooting

This section describes the various processes to follow when sensor module mount/probe failures are present.

9.1.1 Module mount

9.1.1.1 Bayer sensor

1. Open the sensor file.
2. Note the sensor_slave_info array camera_id value.

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {
        /* Camera slot where this camera is mounted */
        .camera_id = CAMERA_0,
        /* sensor slave address */
        .slave_addr = 0x20,
        /* sensor address type */
        .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
        /* sensor id info */
```

3. Note the value for the cell-index value in the sensor dtsi file.

```
qcom,camera@0 {
        cell-index = <0>;
        compatible = "qcom,camera";
        reg = <0x0>;
        qcom,csiphy-sd-index = <0>;
        qcom,csid-sd-index = <0>;
        qcom,mount-angle = <90>;
        qcom,actuator-src = <&actuator0>;
        qcom,led-flash-src = <&led_flash0>;
        cam_vdig-supply = <&pm8916_s4>;
        cam_vana-supply = <&pm8916_l17>;
        cam_vio-supply = <&pm8916_l6>;
        cam_vaf-supply = <&pm8916_l10>;
        qcom,cam-vreg-name = "cam_vdig", "cam_vio", "cam_vana",
                                                    "cam_vaf";
```

4. The value for sensor_slave_info array camera_id should be the same as the cell-index value in the sensor dtsi file.

9.1.1.2 YUV sensor

1. Add the following sensor dtsi settings to the board camera dtsi file (arch/arm/boot/dts).

   If you are viewing this document using a color monitor, or if you print this document to a color printer, **red boldface** indicates code that is need to be **added**.

```
qcom,camera@78 {
     compatible = "qcom,ABC";
     reg = <0x78>;
     qcom,slave-id = <0x78 0x0016 0x0135>;
     XXXXXX
    };
```

   The value in **red boldface** in the example is a unique value. It represents the camera id at the dtsi tree. Here, we use 78, so if you bring up another camera , you cannot use 78 again.

2. Add camera clk source to arch/arm/mach-msm/clock-xxxx.c
   - For MSM8974, MSM8x26/28, and MSM8926/28

```
CLK_LOOKUP("cam_src_clk", mclk0_clk_src.c, "78.qcom,camera"),
CLK_LOOKUP("cam_clk", camss_mclk0_clk.c, "78.qcom,camera"),
```

   - For MSM8916/MSM8936/MSM8909/MSM8084/MSM8992/MSM8994 and later, clk_mclk1_clk_src and clk_gcc_camss_mclk1_clk settings are specified in the .dtsi file. See kernel/arch/arm/boot/dts/qcom/msm8916-camera-sensor-mtp.dtsi for an example.

9.1.2 Module probe

9.1.2.1 Power up/down sequence

1. Define a macro to enable the log in the file “msm_camera_dt_util.c” as shown.

```
--- a/drivers/media/platform/msm/camera_v2/sensor/io/msm_camera_dt_util.c
+++ b/drivers/media/platform/msm/camera_v2/sensor/io/msm_camera_dt_util.c
@@ -18,7 +18,7 @@

 #define CAM_SENSOR_PINCTRL_STATE_SLEEP "cam_suspend"
 #define CAM_SENSOR_PINCTRL_STATE_DEFAULT "cam_default"
-/*#define CONFIG_MSM_CAMERA_DT_DEBUG*/
+#define CONFIG_MSM_CAMERA_DT_DEBUG*/
 #undef CDBG
 #ifdef CONFIG_MSM_CAMERA_DT_DEBUG
 #define CDBG(fmt, args...) pr_err(fmt, ##args)
```

2. Search for the keyword “msm_camera_power_up” from the kernel log. The log will point out the failed sequence type.
3. Ensure the power up/down setting array meets the sensor specification.

   The following code snip is an example of where in the code that the power up/down sequence is configured.

```
static struct msm_sensor_power_setting power_setting[] = {
     {
          .seq_type = SENSOR_VREG,
          .seq_val = CAM_VDIG,
          .config_val = 0,
          .delay = 0,
     },
     {
          .seq_type = SENSOR_VREG,
          .seq_val = CAM_VANA,
          .config_val = 0,
          .delay = 0,
     },
     {
          .seq_type = SENSOR_VREG,
          .seq_val = CAM_VIO,
          .config_val = 0,
          .delay = 0,
     },
```

   More examples can be found in the following file:
   Vendor/qcom/proprietary/mm-camera/mm-camera2/
   media-controller/modules/sensors/sensor_libs/xxxx/ xxxx_lib.c

9.1.2.2 CCI

The CCI performs I2C operation for camera slave devices. Some hardware components cannot run I2C Speed mode, which may cause issues. This section describes how to change the I2C Speed mode.

1. Enable logging in the CCI release source file (kernel/drivers/media/platform/msm/camera_v2/sensor/cci/msm_cci.c) to validate register settings.
2. Change the CCI I2C speed by changing the i2c_freq_mode value:
   - I2C_FAST_MODE – 400 KHz
   - I2C_STANDARD_MODE – 100 KHz

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {
  /* Camera slot where this camera is mounted */
  .camera_id = CAMERA_0,
  /* sensor slave address */
  .slave_addr = 0x20,
  /* sensor address type */
  .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
  /* Slave I2C Frequency */
  .i2c_freq_mode = I2C_FAST_MODE,
  /* sensor id info*/
  .sensor_id_info = {
    /* sensor id register address */
    .sensor_id_reg_addr = 0x0016,
    /* sensor id */
    .sensor_id = 0x0135,
  },
  /* power up / down setting */
  .power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
    .power_down_setting = power_down_setting,
    .size_down = ARRAY_SIZE(power_down_setting),
  },
};
```

3. Enabling CCI clock stretch:

   For the sensors that mandate the I2C master (CCI) clock stretch feature to operate properly, try setting the property “qcom,hw-scl-stretch-en” to 1 in dtsi file:

```
. . .
qcom,hw-scl-stretch-en = <1>;
. . .
```

   Although this change should not cause functional issues, I2C speed would be reduced, as the master (CCI) now must listen to the slow rising SCL signals instead of determining all the timing based on the internal CCI clock. It is recommended that customers do thorough stress

testing of I2C functionality when making this change, and update QTI via Salesforce case in case of any issues.

9.1.2.3 MCLK

MCLK is set to 24 MHz by default in the msm_sensor.c file.

```
static struct msm_cam_clk_info cam_xxxx_clk_info[] = {
     [SENSOR_CAM_MCLK] = {"cam_src_clk", 24000000},
     [SENSOR_CAM_CLK] = {"cam_clk", 0},
};
```

The MCLK value can be specified, for example 19.2 MHz in the sensor power setting array.

```
static struct msm_sensor_power_setting abc_power_setting[] = {
   {
        .seq_type = SENSOR_CLK,
        .seq_val = SENSOR_CAM_MCLK,
        .config_val = 19200000,
        .delay = 1,
   },
}
```

For MSM8916 and MSM8939, the recommended MCLK is 23.88 MHz. It is acceptable to configure the sensor with the assumption of 24 MHz input MCLK though. To set this for MSM8916, set the following entry in the file, kernel/arch/arm/boot/dts/qcom/msm8916-camera-sensor-mtp.dtsi

```
qcom,mclk-23880000 = <1>;
```

9.2 ISP troubleshooting

9.2.1 SOF IRQ timeout

1. Open mm-camera/mm-camera2/media-controller/mct/bus/mct_bus.c
2. Search for the mct_bus_sof_thread_run function and check the if receive log “SOF freeze; Sending error message.”
   - If it is yes, it means that the ISP does not receive the SOF IRQ from the kernel. We must check if something is wrong at CSID/CSIPHY/CAMIF.
   - If no , it means that the ISP receives frames from the sensor.

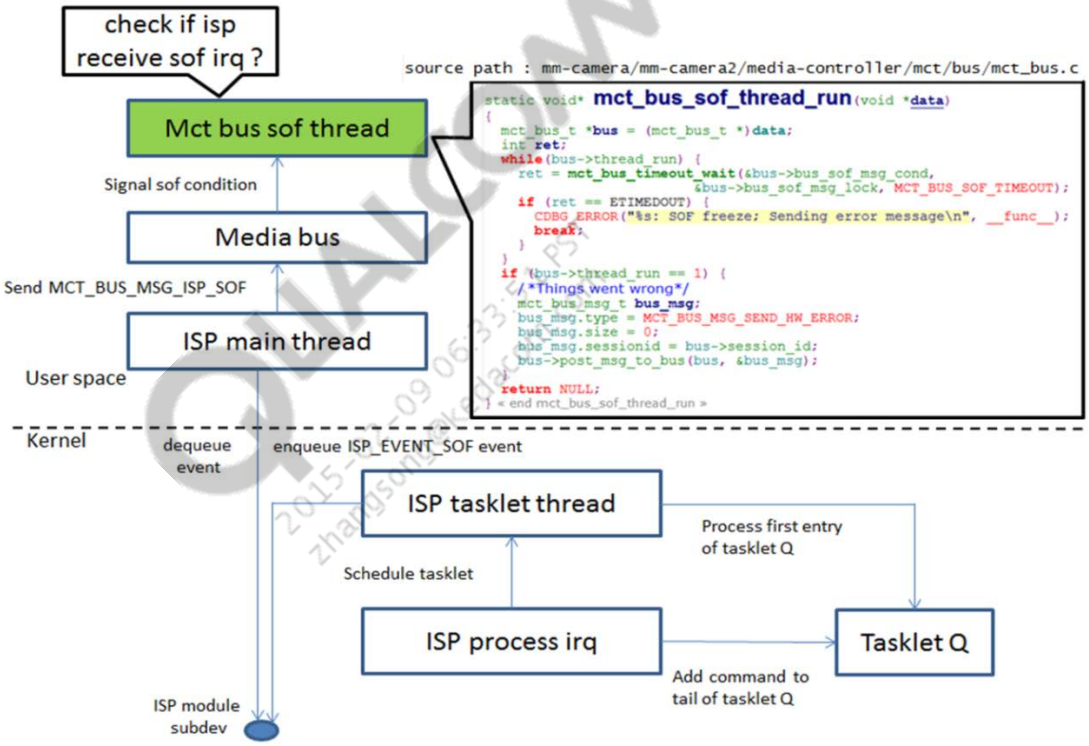


Figure 9-1 SOF IRQ timeout

9.2.2 VFE overflow

Overflows occur when the VFE clock is set less than the output MIPI data rate of the sensor. Figure 9-2 illustrates an overflow due to VFE clock less than the sensor output MIPI data rate.

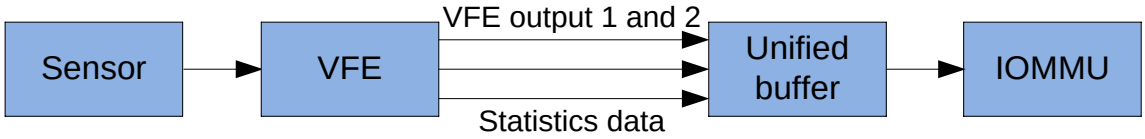


Figure 9-2 VFE overflow

To determine if an overflow has occurred, examine the overflow log files located at:

- kernel/drivers/media/platform/msm/camera_v2/isp/ msm_ispxx.c
- kernel/drivers/media/platform/msm/camera_v2/ispif/ msm_ispif.c.

The typical overflow log entry will appear as:

```
<3>[2019.674774] msm_ispif_read_irq_status: pix intf 0 overflow.
<3>[2019.700866] msm_ispif_read_irq_status: pix intf 0 overflow.
```

Increasing the VFE clock will prevent overflow. To change the VFE clock, open the file at Vendor/qcom/proprietary/mm-camera/mm-camera2/media-controller/modules/sensors/sensor_libs/xxxx/ xxxx_lib.c and edit the op_pixel_clk parameter.

9.2.3 CAMIF error status

A CAMIF error occurs when there is a mismatch between the declared sensor output size and the actual sensor output that is received by the VFE.

1. To verify the settings, perform a dump of the CAMIF settings with the isp_hw_camif_dump_cfg function in the isp_hw.c file.

```
static void isp_hw_camif_dump_cfg(struct msm_vfe_pix_cfg *pix_cfg)
{
  CDBG("%s: =====dump Camif cfg for PIX interface====\n", __func__);
  CDBG("%s: camif input type = %d\n", __func__,
    pix_cfg->camif_cfg.camif_input);
  CDBG("%s: camif pixel pattern = %d\n", __func__, pix_cfg->pixel_pattern);
  CDBG("%s: camif input_format= %d\n", __func__, pix_cfg->input_format);

  CDBG("%s: camif first_pix = %d\n", __func__, pix_cfg->camif_cfg.first_pixel);
  CDBG("%s: camif last_pix = %d\n", __func__, pix_cfg->camif_cfg.last_pixel);
  CDBG("%s: camif first_line = %d\n", __func__, pix_cfg->camif_cfg.first_line);
  CDBG("%s: camif last_line = %d\n", __func__, pix_cfg->camif_cfg.first_line);
  CDBG("%s: camif pixels per line = %d\n",
     __func__,  pix_cfg->camif_cfg.pixels_per_line);
  CDBG("%s: camif lines per frame = %d\n", __func__,
    pix_cfg->camif_cfg.lines_per_frame);
}
```

2. Compare the frame size indicated in the debug messages against the ISP sensor frame size.

In the following CAMIF error example, the error status, 0x9a70a00 indicates an ISP receive frame size of 2471x2560 (0x9a7 = 2471 , 0xa00 = 2560).

```
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: =====dump
Camif cfg for PIX interface====
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
input type = 3
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
pixel pattern = 6
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
input_format= 0
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
first_pix = 0
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
last_pix = 6527
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
first_line = 0
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
last_line = 0
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
pixels per line = 6528
01-01 08:07:20.175 E/mm-camera(  302): isp_hw_camif_dump_cfg: camif
lines per frame = 2448
01-01 08:07:24.335 E/klogd   (  640): [   81.563301]
msm_vfe40_process_error_status: camif error status: 0x9a70a00
```

3. If there is a mismatch, the following may be the root cause:
   - Check if sensor settings are correct and meet the resolution size. For example, the sensor is configured to output 12 MB size but the ISP is configured to receive 8 MB size.
   - It is possible that some sensors cannot guarantee that the very first frame after the new resolution settings are sent to the sensor is of the requested size. In these cases, work with the sensor vendor to fix the issue."

9.3 CSID troubleshooting

In order to troubleshoot CSID, all CSID IRQs must be enabled to check if CSID receives mipi data or error bit IRQs.

If you are viewing this document using a color monitor, or if you print this document to a color printer, **red boldface** indicates code that is to be **added**, and ~~blue strikethrough~~ indicates code that is to be replaced or removed.

```
--- a/drivers/media/platform/msm/camera_v2/sensor/csid/msm_csid.c
+++ b/drivers/media/platform/msm/camera_v2/sensor/csid/msm_csid.c
@@ -25,12 +25.15 @@
 #define CSID_VERSION_V30                0x30000000
 #define MSM_CSID_DRV_NAME               “msm_csid”

 #define DBG_CSID      0
 #define DBG_CSID      1

 #define TRUE          1
 #define FALSE    0

 #undef CDBG
 #define CONFIG_MSMB_CAMERA_DEBUG
 #ifdef CONFIG_MSMB_CAMERA_DEBUG
 #define CDBG(fmt, args...) pr_err(fmt, ##args)
 #else
@@ -83,8 +89,8 @@ static void msm_csid_set_debug_reg(void __iomem
*csidbase,
 {
   uint32_t val = 0;
```

```
    val = ((1 << csid_params->lane_cnt – 1) << 20;
    msm_camera_io_w(0x7f010800 | val, csidbase + CSID_IRQ_MASK_ADDR);
    msm_camera_io_w(0x7f010800 | val, csidbase + CSID_IRQ_CLEAR_CMD_ADDR);
    msm_camera_io_w(0xFFFFFFFF | val, csidbase + CSID_IRQ_MASK_ADDR);
    msm_camera_io_w(0xFFFFFFFF | val, csidbase + CSID_IRQ_CLEAR_CMD_ADDR);
  }
#else
static void msm_csid_set_debug_ref(void __iomem *csidbase,
```

Checking to see if CSID receives MIPI data

To see if CSID receives MIPI data, examine IRQ bits 0 through 7.

- CAMSS_A_CSID_X_CSID_IRQ_STATUS bit 0-3 (where X is the bit number) – These IRQs fire when the CSID core receives an SOT on PHY data lines 0-3
- CAMSS_A_CSID_X_CSID_IRQ_STATUS bit 4-7 (where X is the bit number) – These IRQs fire when the CSID core receives an EOT on PHY data lines 0-3

The following log example shows a normal 4-lane sensor CSID IRQ status log.

```
<3>[ 272.271075] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.276101] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.281099] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.286133] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.291164] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.296193] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.301232] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.306248] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.311300] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
<3>[ 272.316342] msm_csid_irq CSID0_IRQ_STATUS_ADDR = 0xdd
```

If any mipi signal error IRQs fired, see [Q5] for the CSID IRQ bit definition, check with the sensor vendor, or modify the sensor settle count.

9.4 DPHY troubleshooting

This section describes two methods of DPHY troubleshooting.

Checking the DPHY debug logs

In order to troubleshoot DPHY, DPHY debug logs must be enabled to check if the hardware register, CAMSS_A_CSI_PHY_**X**_MIPI_CSIPHY_INTERRUPT_STATUS**Y**, receives any IRQ errors.

```
--- a/drivers/media/platform/msm/camera_v2/sensor/csiphy/msm_csiphy.c
+++ b/drivers/media/platform/msm/camera_v2/sensor/csiphy/msm_csiphy.c
@@ -21,7 +21.7 @@
 #include “msm_sd.h”
 #include “msm_csiphy_hwreg.h”
 #include “msm_camera_io_util.h”
 #define DBG_CSIPHY          0
 #define DBG_CSIPHY          1

 #define V4L2_IDENT_CSIPHY                  50003
 #define CSIPHY_VERSION_V22                 0x01
@@ -30,6 +30,7 @@
 #define MSM_CSIPHY_DRV_NAME                “msm_csiphy”

 #undef CDBG
 #define CONFIG_MSMB_CAMERA_DEBUG
 #ifdef CONFIG_MSMB_CAMERA_DEBUG
 #define CDBG(fmt, args...) pr_err(fmt, ##args)
 #else
```

The following log example shows a normal CSIPHY IRQ status log.

```
<3>[ 1027.268003] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS0 = 0x0
<3>[ 1027.268020] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS1 = 0x0
<3>[ 1027.268036] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS2 = 0x0
<3>[ 1027.268052] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS3 = 0x0
<3>[ 1027.268068] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS4 = 0x0
<3>[ 1027.268084] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS5 = 0x0
<3>[ 1027.268100] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS6 = 0x0
<3>[ 1027.268116] c0 18360 [CAM] msm_csiphy_irq MIPI_CSIPHY1_INTERRUPT_STATUS7 = 0x0
```

If any CSIPHY signal error IRQs fired, see [Q5] for the CSID IRQ bit definition, check with the sensor vendor, or modify the sensor settle count.

Checking the DPHY index and hardware layout relationship

If there are issues with the DPHY, there may be a mismatch between the DPHY index and hardware layout. Compare the values in the sensor dtsi settings with the corresponding values in the sensor's xxxx_lib.c file. The values must match for proper operation. The items in bold indicate the current parameter and its corresponding hardware index.

Sensor's dtsi settings	Sensor's xxxx_lib.c file

qcom,camera@78 {
compatible = "qcom,ABC";
XXXXXX
qcom,csiphy-sd-index = <1>;
// DPHY index
qcom,csid-sd-index = <**3**>;
// CSID index
XXXXXX
	static struct csi_lane_params_t
	csi_lane_params = {
	.csi_lane_assign = 0x4320,
	.csi_lane_mask = 0x7,
	// 1 lane : 0x3 ,
	2 lane : 0x7 ,
	4 lane : 0x1F
	.csi_if = 1,
	.csid_core = {3}, // CSID index
	.csi_phy_sel = **1**, // DPHY index

A References

A.1 Related documents

Documents	
Qualcomm Technologies, Inc.	
Multimedia Driver Development and Bringup Guide – Audio	80-NU323-1
Multimedia Driver Development and Bringup Guide – Display	80-NU323-3
Multimedia Driver Development and Bringup Guide – Video	80-NU323-5
Presentation: Chromatix™ 6 Camera Tuning	80-NK872-2
Presentation: Camera Auto Focus Tuning Guide	80-N8459-1
BAM Low-speed Peripherals for Linux Kernel Configuration and Debugging Guide	80-NE436-1
Qualcomm Createpoint Hardware Component Quick Start Guide (English)	80-NC193-10
Qualcomm Createpoint Hardware Component Quick Start Guide (Chinese)	80-NC193-10SC

A.2 Acronyms and terms

Term	Definition
AF	Auto Focus
GCDB	Global Components Database
IRQ	Interrupt Request
QRD	Qualcomm Reference Design
SOF	Start of Frame

Chinese Version

1 简介

1.1 目的

本文对高通平台的 camera 驱动开发进行了详细的介绍，主要包含以下几部分:

- Sensor
- CSIPHY
- CSID
- CCI
- Actuator
- Flash
- EEPROM
- Chromatix™

本文讨论为点亮摄像头而如何创建/配置各部分代码。其中第二章涵盖摄像头传感器驱动开发. 第三章介绍自动对焦功能的马达驱动开发. 第四章介绍 LED 闪光灯驱动开发. 第五章介绍 EEPROM 驱动开发.

1.2 范围

虽然本文是基于 MSM8916 的代码而写的, 本文中很多信息同样适用于 MSM8974, APQ8084, 和 MSM8x26 (B-Family)平台 Linux 摄像头代码。

1.3 约定

函数声明，函数名称，类型声明以及代码举例需要使用不同的字体，例如`#include`。

代码变量应出现在角括号中，例如`<number>`。

在本文中，$(MM_CAMERA_DIR)代表/vendor/qcom/proprietary/mm-camera.

1.4 参考

参考文件，其中可能包括高通的文档、标准和资源，具体列在表 1-1。对于不再适用参考的文档已从本表中删除，因此，参考文档的引用顺序可能不连续。

表 **1-1** 参考文档和标准

编号	文档	
高通技术文档		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Guide to Configuring and Debugging BAM Low-speed Peripherals for Linux Kernel for MSM8974, MSM8x26, MDM9x25, APQ8084 Chipsets*	80-NE436-1
Q3	*Presentation: Camera Auto Focus Tuning Guide*	80-N8459-1
Q4	*Presentation: Chromatix™ 6 Camera Tuning*	80-NK872-2

1.5 技术支持

如果您想得到关于本指南的支持细节，请登陆至 https://support.cdmatech.com/。

如果您无法登陆 CDMA 技术支持服务网站，请注册或发送电子邮件至 support.cdmatech@qualcomm.com。

1.6 缩写词

术语和缩略词的定义请参考文档[Q1]。

2 摄像头传感器驱动

这个章节介绍如何在 MSM8916 Android 平台编写驱动点亮摄像头传感器。

请注意除非特别说明，下面提到的摄像头传感器都是 Bayer 类型.

2.1 YUV和Bayer Sensor参考驱动

这个段落罗列 MSM8916 当前主线代码包含的 Bayer 和 YUV sensor 参考驱动。

Bayer 参考驱动

用户空间驱动位于$(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/

- imx135_lib.c/h
- ov2680_lib.c
- ov2720_lib.c
- ov9724_lib.c
- s5k3l1yx_lib.c

YUV 参考驱动

用户空间驱动位于$(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/

- sp1628_lib.c
- SKUAA-Shengtai-hi256_lib.c
- ov5645_lib.c
- mt9m114_lib.c

内核空间驱动位于 kernel/drivers/media/platform/msm/camera_v2/sensor

- sp1628.c
- hi256.c
- ov5645.c
- mt9m114.c

2.2 需要修改的文件

这个段落列出移植一个新的摄像头传感器驱动需要修改的文件.

Bayer sensor

Dtsi 文件为 kernel/arch/arm/boot/dts/qcom/目录下的<target>_camera*.dtsi。比如 msm8916-camera-sensor-mtp.dtsi. 客户需使用如下对应的条目:

```
qcom,camera@0 {
        cell-index = <0>;
        compatible = "qcom,camera";
        . . .
}
```

配置

在 vendor/qcom/proprietary/common/config/device-vendor.mk 文件中, 需为新的驱动加入一个条目.

在$(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/module/sensor_init.c 文件中的 sensor_libs[] 数组中需添加 sensor 名称.

内核代码是通用的，如果和 QTI 的原理图一致，那么无需修改内核代码.

对于该文件的修改必须基于客户的硬件设计.

用户空间驱动是$(MM_CAMERA_DIR)/ mm-camera2/media-controller/modules/sensors/sensor_libs/<snesor>目录下<sensor>_lib.c/h, 比如 imx135_lib.c/h.

除了驱动文件，位于驱动文件目录下的 Android.mk 文件也需修改 (请参考现有的其他驱动的 Makefile).

NOTE: 对应 64 位处理器，使用 arm64 目录而不是 arm 目录

YUV Sensor

dts 文件是位于 kernel/arch/arm/boot/dts/qcom/目录下的<target>_camera*.dtsi 文件,

比如 msm8916-camera-sensor-mtp.dtsi，客户需要在该 dtsi 文件加入类似于如下例子的一个新条目。

```
qcom,camera@78 {
        compatible = "ovti,ov5645";
        . . .
}
```

用户空间驱动是$(MM_CAMERA_DIR)/
mm-camera2/media-controller/modules/sensors/sensor_libs/<snesor>目录下<sensor>_lib.c/h,
比如 sp1628_lib.c/h.

除了驱动文件，位于驱动文件目录下的 Android.mk 文件也需修改 (请参考现有的其他驱动的 Makefile).

内核空间驱动是在 kernel/drivers/media/platform/msm/camera_v2/sensor/目录下有<sensor>.c 文件，该目录下的 Makefile 文件也要修改。

同时, 需要在 kernel/arch/arm/configs 目录下<target>_defconfig 文件中添加
CONFIG_<sensor> 定义

配置

在 vendor/qcom/proprietary/common/config/device-vendor.mk 文件中, 需为新的驱动加入一个条目.

对于该文件的修改必须基于客户的硬件设计.

如果是 64 位处理器，使用 arm64 目录而不是 arm 目录

在上面的例子中我们可以看到 Bayer sensor 引入了 camera slots (0/1/2)的概念。但是对于 YUV sensor, 不同的 sensor 仍需在 dtsi 文件中添加新的条目

在写新的驱动时请以现有的驱动文件为模板

2.3 源代码解释

2.3.1 内核驱动

本节介绍创建内核驱动的相关信息.

2.3.1.1 GPIO配置

如下所示，客户可以根据自己的硬件设置配置 sensor 特有的 GPIO.
关于各个属性的解释，可以参考以下目录下的 msm-cci.txt：

kernel/Documentation/devicetree/bindings/media/video/

```
gpios = <&msmgpio 15 0>,
            <&msmgpio 90 0>,
            <&msmgpio 89 0>;
qcom,gpio-reset = <1>;
qcom,gpio-standby = <2>;
```

```
qcom,gpio-req-tbl-num = <0 1 2>;
qcom,gpio-req-tbl-flags = <1 0 0>;
qcom,gpio-req-tbl-label = "CAMIF_MCLK",
                                        "CAM_RESET1",
                                         "CAM_STANDBY";
qcom,gpio-set-tbl-num = <1 1>;
qcom,gpio-set-tbl-flags = <0 2>;
qcom,gpio-set-tbl-delay = <1000 30000>;
```

对于 CCI, 在下面 MSM8916 专有的 GPIO 用于数据和时钟传输.因为这两个 GPIO 只用于 CCI, 如果客户使用 CCI 为 I2C 通信，请保持现有配置.

```
gpios = <&msm_gpio 29 0>,
            <&msm_gpio 30 0>;
qcom,gpio-tbl-num = <0 1>;
qcom,gpio-tbl-flags = <1 1>;
qcom,gpio-tbl-label = "CCI_I2C_DATA0",
                                    "CCI_I2C_CLK0";
```

2.3.1.2 时钟相关设置

在 dts 文件中，对于每一个 sensor 节点，客户可以如下设置时钟源:

```
clocks = <&clock_gcc clk_mclk0_clk_src>,
              <&clock_gcc clk_gcc_camss_mclk0_clk>;
clock-names = "cam_src_clk", "cam_clk";
```

这两个属性的顺序是相关的. clock-name 属性的第 n 个值对应 clocks 属性的第 n 个值. 因此在上面的 dt 片段中, cam_src_clk 对应 clk_mclk0_clk_src, cam_clk 对应 clk_gcc_camss_mclk0_clk.

以上设置被时钟模块代码引用，客户一般无需修改设置.

2.3.1.3 电源设置

PMIC 供电

```
cam_vdig-supply = <&pm8916_s4>;
cam_vana-supply = <&pm8916_l17>;
cam_vio-supply = <&pm8916_l6>;
cam_vaf-supply = <&pm8916_l10>;
```

GPIO 控制外部 LDO 供电

```
gpios = <&msm_gpio 27 0>,
            <&msm_gpio 28 0>,
            <&msm_gpio 33 0>,
            <&msm_gpio 114 0>,
            <&msm_gpio 110 0>;
qcom,gpio-reset = <1>;
qcom,gpio-standby = <2>;
qcom,gpio-vdig = <3>;
qcom,gpio-vana = <4>;
qcom,gpio-req-tbl-num = <0 1 2 3 4>;
qcom,gpio-req-tbl-flags = <1 0 0 0 0>;
qcom,gpio-req-tbl-label = "CAMIF_MCLK",
                          "CAM_RESET",
                          "CAM_STANDBY",
                          "CAM_VDIG",
                          "CAM_VANA";
```

- CAM_VANA – Supply voltage (模拟)
- CAM_VDIG – Supply voltage (数字)
- CAM_VAF – Supply voltage (Actuator 电压)
- CAM_VIO – Input/output voltage (IO 数字)

2.3.1.4 I2C slave 地址配置

YUV sensor

在 dtsi 文件中:

```
qcom,camera@78 {
        compatible = "ovti,ov5645";
        reg = <0x78 0x0>;
        qcom,slave-id = <0x78 0x300a 0x5645>;
```

以上设置的 I2C slave 地址是八位地址，其中高 7 位为 I2C slave 地址，最低位是写标记 0.

NOTE: Bayer sensor 的情况，请参考章节 3.3.2.3.

2.3.2 用户空间驱动

该节介绍创建用户空间驱动的相关信息.

2.3.2.1 Sensor初始化参数

Sensor 初始化参数包括支持模式 (2D/3D)，安装位置 (前置/后置)和安装角度. 如果安装角度设置成 360 度，内核的 dtsi 文件中安装角度将用于驱动中.

```
static struct msm_sensor_init_params sensor_init_params = {
  .modes_supported = CAMERA_MODE_2D_B,
  .position = BACK_CAMERA_B,
  .sensor_mount_angle = SENSOR_MOUNTANGLE_360,
};
```

2.3.2.2 Sensor输出配置

Sensor 输出配置包括输出格式（Bayer 或者 YUV）. 连接模式 (MIPI 或者 parallel, MSM8916 只支持 MIPI)和 raw 格式类型.

```
static sensor_output_t sensor_output = {
  .output_format = SENSOR_BAYER,
  .connection_mode = SENSOR_MIPI_CSI,
  .raw_output = SENSOR_10_BIT_DIRECT,
};
```

2.3.2.3 Bayer slave配置

Sensor slave 配置信息必须提供下面信息.

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {
  /*该 sensor 安装在哪个物理接口上*/
  .camera_id = CAMERA_0,
  /* I2C slave 地址 */
  .slave_addr = 0x20,
  /* sensor 地址类型 */
  .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
  /* sensor id 信息*/
  .sensor_id_info = {
    /* sensor id 寄存器地址 */
    .sensor_id_reg_addr = 0x0016,
    /* sensor id */
    .sensor_id = 0x0135,
```

```
  },
  /* 上下点设置 */
  .power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
  },
. . .
```

以上设置的 I2C slave 地址是八位地址，其中高 7 位为 I2C slave 地址，最低位是写标记 0.

NOTE: YUV sensor 请参考章节 3.3.1.4.

2.3.2.3.1 Sensor 芯片id

Sensor 模组/芯片 ID 寄存器设置必须参考 data sheet，然后写入 sensor_id_info 数据结构.

```
.sensor_id_info = {
    /* sensor id register address */
    .sensor_id_reg_addr = 0x0016,
    /* sensor id */
    .sensor_id = 0x0135,
  },
```

2.3.2.3.2 上下电时序

Sensor 的上下电时序以数组的格式写于用户空间驱动的 msm_sensor_power_setting 数据结构中.

```
static struct msm_sensor_power_setting power_setting[] = {
. . .
}
```

上电时序和下电时序都可以分别加在 msm_sensor_power_setting_array 数组中. 如果 power_down_setting/size_down 数据成员没有添加,下电时序会使用上电时序的反时序.

```
.power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
 },
```

power_setting 包含 GPIO/CLK/VREG 拉高/拉低还有延时的序列

```
static struct msm_sensor_power_setting power_setting[] = {
  {
```

```
    .seq_type = SENSOR_VREG,
    .seq_val = CAM_VDIG,
    .config_val = 0,
    .delay = 0, //this delay is in ms
  },
  {
    .seq_type = SENSOR_VREG,
    .seq_val = CAM_VANA,
    .config_val = 0,
    .delay = 0, //this delay is in ms
  },
  . . .
```

该数据结构被内核空间的 msm_camera_power_up()调用来配置 sensor 的上电时序.

请参考 kernel/include/media/msm_cam_sensor.h 的枚举类型.

对应 YUV 摄像头驱动 msm_sensor_power_setting 设置需写在内核空间驱动而不是用户空间驱动. 详情请参考 3.1 节枚举的现有 YUV 参考驱动

根据客户硬件设置的不同, 供电可以来自 dtsi 文件配置的 PMIC 或者 GPIO.

2.3.2.4 输出尺寸表

输出尺寸表定义在如下的 sensor_lib_out_info_t 数据结构中, 例如

```
static struct sensor_lib_out_info_t sensor_out_info[] = {
{
/* full size @ 24 fps*/
.x_output = 4208,
.y_output = 3120,
.line_length_pclk = 4572,
.frame_length_lines = 3142,
.vt_pixel_clk = 360000000,
.op_pixel_clk = 360000000,
.binning_factor = 1,
.max_fps = 24.01,
.min_fps = 7.5,
.mode = SENSOR_DEFAULT_MODE,
},
```

- x_output – sensor 输出有效宽度
- y_output – sensor 输出有效高度
- line_length_pclk –包含 blanking 的宽度值
- frame_length_lines –包含 blanking 的高度值

- vt_pixel_clk(video timing clk value) –该虚拟时钟值用于曝光时间计算，用于 AEC 算法的 banding artifacts 纠正，vt_pixel_clk 的计算如下:

  vt_pixel_clk = line_length_pclk * frame_length_lines * frame rate

- op_pixel_clk – VFE 时钟，表示每秒 VFE 处理的数据量(in pixel).

  op_pixel_clk = (sensor 输出实际比特率)/bits-per-pixel

  比如，如果 MIPI DDR 时钟值 (sensor MIPI 的时钟 lane 频率) 为 300Mhz, 同时 sensor 使用 4 个 lane 传输数据, 每一个 lane 的数据率是 300*2 = 600Mhz. 因此, 总数据率为 600*4 = 2400Mhz. 对于 10bit 的 bayer sensor, op_pixel_clk 值可设置为 2400/10 = 240Mhz.

  这些值可以从 sensor 的寄存器设置中技算出来。.

2.3.2.5 Chromatix 参数

每一种分辨率都必须有对应的 chromatix 头文件。

Chromatix 头文件的生成方法在[Q4]讨论。

```
    static struct sensor_lib_chromatix_t imx135_chromatix[] = {
         {
.common_chromatix = IMX135_LOAD_CHROMATIX(common),
.camera_preview_chromatix = IMX135_LOAD_CHROMATIX(snapshot), /* RES0 */
.camera_snapshot_chromatix = IMX135_LOAD_CHROMATIX(snapshot), /* RES0 */
.camcorder_chromatix = IMX135_LOAD_CHROMATIX(default_video), /* RES0 */
.liveshot_chromatix = IMX135_LOAD_CHROMATIX(liveshot), /* RES0 */
          },
       }
```

2.3.2.6 Sensor 寄存器地址

必须参考 sensor 规格书填写正确的控制曝光和输出大小的寄存器地址

2.3.2.6.1 曝光设定寄存器地址

```
static struct msm_sensor_exp_gain_info_t exp_gain_info = {
 .coarse_int_time_addr = 0x0202,
 .global_gain_addr = 0x0205,
 .vert_offset = 4,
};
```

- vert_offset – 曝光行数上限的边界值，曝光行数任何情况下都应该小于 frame_length_lines 减去 vert_offset.

NOTE: 根据规格书中的描述来决定这个值 (粗曝光设定的边界值).

2.3.2.6.2 输出控制寄存器地址

必须根据 sensor 规格书中 寄存器描述填写正确的输出控制寄存器地址

```
static struct msm_sensor_output_reg_addr_t output_reg_addr = {
  .x_output = 0x034C,
  .y_output = 0x034E,
  .line_length_pclk = 0x0342,
  .frame_length_lines = 0x0340,
};
```

2.3.2.7 MIPI 接收器配置

Sensor 通过 MIPI CSI2 传输图像，高通接收器通过 MIPI CSI PHY 和 CSID 结束相应数据。

2.3.2.7.1 CSI-PHY config

该数据结构设置 CSI lane 参数.

```
static struct csi_lane_params_t csi_lane_params = {
  .csi_lane_assign = 0x4320,
  .csi_lane_mask = 0x1F,
  .csi_if = 1,
  .csid_core = { 0 },
  .csi_phy_sel = 0,
};

static struct msm_camera_csi2_params imx135_csi_params = {
  .csid_params = {
    .lane_cnt = 4,
    .lut_params = {
      .num_cid = ARRAY_SIZE(imx135_cid_cfg),
      .vc_cfg = {
        &imx135_cid_cfg[0],
        &imx135_cid_cfg[1],
        &imx135_cid_cfg[2],
      },
    },
  },
  .csiphy_params = {
    .lane_cnt = 4,
    .settle_cnt = 0x1B,
  },
};
```

- csi_lane_assign –有时候用户的 mipi lanes 可能使用不同于 MSM 参考设置的端口映射. 比如, sensor 的 lane 0 连接到 MSM 的数据 lane 4 等等. 对于该种情况, csi_lane_assign 参数能设置正确的端口映射. csi_lane_assign 是一个 16bit 的值，每位的含义参见下表

Bit position	Represents
15:12	连接 MSM 端数据 lane 4 的 sensor MIPI 输入 lane 序号
11:8	连接 MSM 端数据 lane 3 的 sensor MIPI 输入 lane 序号
7:4	连接 MSM 端数据 lane 2 的 sensor MIPI 输入 lane 序号
3:0	连接 MSM 端数据 lane 0 的 sensor MIPI 输入 lane 序号

NOTE: lane1 用用于 MIPI 时钟，客户不可用它来映射到任何数据 lane..

- csi_lane_mask –用于表示那些 lane 被使用，这个一个 8 位值, 每一位含义如下:

Bit position	Represents
4	数据 lane 4 是否使用: - 0: 不 - 1: 是
3	▪ 数据 lane 3 是否使用: - 0: 不 - 1: 是
2	▪ 数据 lane 2 是否使用: - 0: 不 - 1: 是
1	时钟 lane 1 是否使用: - 0: 不 - 1: 是 ▪ 注意: 该位必须设置为 1
0	数据 lane 0 是否使用: - 0: 不 - 1: 是
4	▪ 数据 lane 4 是否使用: - 0: 不 - 1: 是

  比如 0x1F 表示 4 条数据 lane 和 时钟 lane 都被使用。

- csi_if – 暂不使用
- csid_core –设置哪个 CSID 硬件被该 sensor 使用。两个并发的 sensor 不能使用同一个 CSID 硬件。

- csi_phy_sel –设置哪个 CSI-PHY 硬件被该 sensor 使用。对于每一个 sensor 来是必须是独一无二的，非有额外的 MIPI 桥连接两个 sensor 到同一个 PHY 接口上. 该值必须参照客户的硬件设置来设置。
- lane_cnt –有多少数据 lane 用于数据传输. 该值必须在 sensor 最大能力范围内，而且 sensor 寄存器设置必须与该 lane 数匹配.
- settle_cnt –该值须和 sensor 的特性匹配, 保证 sensor 的 MIPI 传输和 MSM 的 MIPI 接收能同步.

```
MIN [ Settle count * T(Timer clock)]  > T(HS_SETTLE)_MIN
MAX [ Settle count * T(Timer clock)]  < T(HS-PREPARE)+T(HS_ZERO) -
4*T(Timer clock)
```

同时客户可以对不同的 RES 模式 使用不同 csi 参数设置:

```
static struct msm_camera_csi2_params* csi_params[] = {
  &imx135_csi_params, /* RES 0 */
  &imx135_csi_params, /* RES 1 */
  &imx135_csi_params, /* RES 2 */
  &imx135_csi_params, /* RES 3 */
  &imx135_csi_params, /* RES 4 */
  &imx135_csi_params, /* RES 5 */
  &imx135_csi_params, /* RES 6 */
};
```

2.3.2.7.2 CSI-D 配置

```
static struct msm_camera_csid_vc_cfg imx135_cid_cfg[] = {
  { 0, CSI_RAW10, CSI_DECODE_10BIT },
  { 1, 0x35, CSI_DECODE_8BIT },
  { 2, CSI_EMBED_DATA, CSI_DECODE_8BIT}
};
```

每一行设置的第一列被称为 CID (channel ID). Virtual Channel (VC)和 Data Type (DT)独一无二的组合映射到唯一的 CID 值. 下表列出对于特定的 VC 可能的 CID 值.

- 0 – 0, 1, 2, 3
- 1 – 4, 5, 6, 7
- 2 – 8, 9, 10, 11
- 3 – 12, 13, 14, 15

在上面的 imx135_cid_cfg 例子中, 我们可以看到有 CSI_RAW10, 0x35, CSI_EMBED_DATA 三种数据类型在 VC 0 中. 因此, CID 的范围是 0 到 3.

2.3.2.8 寄存器设定

相机 sensor 的寄存器通过 I2C 来配置使 sensor 输出数据，本节中还会介绍通过其他特别的操作方式来实现配置 sensor

2.3.2.8.1 初始化设定

表现为在相机启动时一组一次性写入的寄存器

```
static struct msm_camera_i2c_reg_setting init_reg_setting[] = {
. . .
}
```

2.3.2.8.2 Grouphold on 设定

sensor 工作时更新曝光设定需要操作许多寄存器（曝光时间，每帧行数，增益），他们必须在同一帧完成更新。这些寄存器都有双 buffer，并具有按组更新的功能。表现为所有相关寄存器一起完成更新。当设定 grouped parameter hold 为 1 时，写入的寄存器数据被暂存的 buffer 寄存器中。

```
static struct msm_camera_i2c_reg_setting groupon_settings = {
. . .
}
```

2.3.2.8.3 Grouphold off 设定

当设定 grouped parameter hold 为 0 时， 曝光寄存器的值会被同时更新，参数的变化会在同一帧生效。

```
static struct msm_camera_i2c_reg_setting groupoff_settings = {
. . .
}
```

2.3.2.8.4 分辨率设定

Sensor 可以有多种操作模式，例如按四分之一大小预览和全尺寸拍照，不同的分辨率可以按如下方式配置，

```
static struct msm_camera_i2c_reg_setting res0 _settings[] = {
. . .
}
static struct msm_camera_i2c_reg_setting res1_settings[] = {
. . .
}
```

2.3.2.8.5 曝光设定

AEC 算法中实际 gain 用于曝光计算, 实际 gain 必须转换成寄存器 gain 去设置 sensor。客户需要参考 sensor 的 datasheet 去编写正确的 gain 转换函数.

```
xxxx_real_to_register_gain()
xxxx_register_to_real_gain()
```

下面函数基于上面的转换函数计算曝光时间和 `gain.`

```
xxxx_calculate_exposure()
```

下面函数基于曝光时间和 gain 的计算值填写正确的寄存器设置.

```
xxxx_fill_exposure_array()
```

NOTE: 参考示例驱动程序.

2.3.2.8.6 启动输出设定

MIPI 数据包必须以在 Start of Transmission (SoT) 和 End of Transmission (EoT)之间发送

- SoT – LP11→LP01→LP00
- EoT – LP00→LP11

在 sensor 驱动, 下面的参数必须根据向导指示进行配置.

找出 clock 和 data lanes 从 LP11 到 HS Tx 的状态:

```
static struct msm_camera_i2c_reg_array start_reg_array[] = {
  { 0x0100, 0x01 },
};
```

2.3.2.8.7 停止输出配置

停止输出配置应该把 clock 和 data lane 置为 LP11 的状态，如果没有正确执行会导致相机和 MSM 平台同步问题

```
static struct msm_camera_i2c_reg_array stop_reg_array[] = {
  { 0x0100, 0x00 },
};.
```

2.3.3 Using QUP/SPI

关于 BLSP 的配置请参考文档 80-NE436-1. 尽管该文档不是专门针对 MSM8916，但是相同的概念可以通用.

NOTE: 如果客户需要在 MSM8916 上配置 QUP/SPI 连接的 sensor，请联系 Qualcomm.

3 AF 马达驱动

本章将通过介绍基线代码中 AF 驱动来讲解客户自己编写 AF 马达驱动的基本原则

3.1 AF 马达驱动代码目录结构

客户可参考现有 AF 马达驱动代码编写新的 AF 马达驱动。我们以某 imx135 模组的 dw9714 驱动为例进行说明.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuator_libs/dw9714/
  - Android.mk
  - dw9714_actuator.c
  - dw9714_actuator.h
- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuators/0301/dw9714/
  - Android.mk
  - camcorder/
    - Android.mk
    - dw9714_camcorder.c
    - dw9714_camcorder.h
  - camera/
    - Android.mk
    - dw9714_camera.c
    - dw9714_camera.h
- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

3.2 需要添加和修改的文件

下面介绍添加新的 AF 马达驱动需要添加和修改的文件.

3.2.1 更新 device tree 文件

在对应的摄像头 dtsi 文件中, 比如 msm8916-camera-sensor-mtp.dtsi, 为 AF 马达添加一个新的项目.

- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

```
&cci{        actuator0: qcom,actuator@6e {
                cell-index = <3>;
                reg = <0x6c>;
                compatible = "qcom,actuator";
                qcom,cci-master = <0>;
        };
        qcom,camera@0 {
                qcom,actuator-src = <&actuator0>;
        };
}
```

3.2.2 更新用户空间设备驱动文件

以 imx135_lib.c 驱动为例, sensor_lib_t 数据结构的 actuator_name 数据成员设置正确的马达名称.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/imx135/imx135_lib.c

```
static sensor_lib_t sensor_lib_ptr = {
  /* sensor slave info */
  .sensor_slave_info = &sensor_slave_info,
  /* sensor init params */
  .sensor_init_params = &sensor_init_params,
  /* sensor actuator name */
.actuator_name = "dw9714",
```

3.2.3 添加马达驱动

新的马达驱动需要添加以下文件 <actuator>_actuator.c 和 <actuator>_actuator.h .

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuator_libs/<actuator>/
  - Android.mk
  - <actuator>_actuator.c
  - <actuator>_actuator.h

The <actuator>_actuator.c 文件会返回 actuator_lib_ptr.

```
#include "actuator_driver.h"

static actuator_driver_ctrl_t actuator_lib_ptr = {
#include "<actuator>__actuator.h"
};

void *actuator_driver_open_lib(void)
{
  return &actuator_lib_ptr;
}
```

<actuator>_actuator.h 文件中应该由以下参数来调节 AF 效果. 下面有些参数调节效果 其他一些参数调节性能.

```
{
    /* actuator_params_t */
    {
      /* module_name */
      "abico",
      /* actuator_name */
      "dw9714",
      /* 8 bit i2c_addr */
      0x18,
      /* i2c_data_type. Check actuator data sheet for i2c data type */
      MSM_ACTUATOR_BYTE_DATA/MSM_ACTUATOR_WORD_DATA,
      /* i2c_addr_type Check actuator data sheet for i2c addr type*/
      MSM_ACTUATOR_BYTE_ADDR/MSM_ACTUATOR_WORD_ADDR,
      /* act_type */
      ACTUATOR_VCM/ACTUATOR_PIEZO,
      /* data_size */
```

```
      10,
      /* af_restore_pos */
      0,
      /* msm_actuator_reg_tbl_t */
      {
        /* reg_tbl_size */
        1,
        /* msm_actuator_reg_params_t */ Check the actuator data sheet for
eeprom current programing method.
        {
          /* reg_write_type;hw_mask; reg_addr; hw_shift >>; data_shift << */
          {MSM_ACTUATOR_WRITE_DAC, 0x0000000F, 0xFFFF, 0, 4},
        },
      }
      /* init_setting_size */
      0,
      /* init_settings */
      {
   /* Check actuator data sheet for init_setting register write sequence .
It shall look like the e.g. given below*/
   /* reg_addr, addr_type, reg_data, data_type, i2c_operation, delay/*
        {0xEC, MSM_ACTUATOR_BYTE_ADDR, 0xA3, MSM_ACTUATOR_BYTE_DATA,
MSM_ACT_WRITE, 0},
      },
    }, /* actuator_params_t */
Following actuator_tune_params_t are derived from the AF tuning.
    /* actuator_tuned_params_t */
    {
      /* scenario_size */
      {
        /* scenario_size[MOVE_NEAR] */
        2,
        /* scenario_size[MOVE_FAR] */
        3,
      },

      /* ringing_scenario */
      {
        /* ringing_scenario[MOVE_NEAR] */
        {
          4,
          41,
        },
        /* ringing_scenario[MOVE_FAR] */
        {
```

```
          8,
          22,
          41,
        },
      },

      /* intial_code */ This represent the initial step after which actuator
move the lens position linearly.
      0,
      /* region_size */ This number represent into how many region the
total lens step sizes are divided into.
      2

      /* region_params */ Following regions are devided into number specified
in region_size.
      {
        /* step_bound[0] - macro side boundary */
        /* step_bound[1] - infinity side boundary */
        /* Region 1 */
        {
          .step_bound = {2, 0},
          .code_per_step = 85,
        },
        /* Region 2 */
        {
          .step_bound = {41, 2},
          .code_per_step = 9,
        },
      }
      {
        /* damping */
        {
          /* damping[MOVE_NEAR] */
          {
            /* scenario 1 */
       /* Number of scenarios depends on the size defined in scenario_size */
            {
              /* region 1 */
              {
                .damping_step = 0x1FF,
                .damping_delay = 7000,
                .hw_params = 0x0000000C,
              },
              /* region 2 */
              {
```

```
                .damping_step = 0x1FF,
                .damping_delay = 7000,
                .hw_params = 0x7,
              }
              ,
            }
            ,
          }
        },

        {
          /* damping[MOVE_FAR] */
          {
            /* scenario 1 */
            {
              /* region 1 */
              {
                .damping_step = 0x1FF,
                .damping_delay = 7000,
                .hw_params = 0x0000000C,
              },
              /* region 2 */
              {
                .damping_step = 0x1FF,
                .damping_delay = 4500,
                .hw_params = 0x0000007,
              },
            },
          },
        },
      },
    }, /* actuator_tuned_params_t */
  },
```

3.2.4 调节AF 算法

以下文件 Camera and Camcorder 模式都需要添加. 算法调节方法参考 [Q3].

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/actuators/0301/<actuator>/
  - Android.mk
  - camcorder/
    - Android.mk
    - <actuator>_camcorder.c
    - <actuator>_camcorder.h
  - camera/
    - Android.mk
    - <actuator>_camera.c
    - <actuator>_camera.h

4 闪光灯驱动

此部分提供如何添加闪光灯驱动的说明。

4.1 闪光灯驱动的目录结构

闪光灯驱动可以基于 PMIC, QUP/I2C 或者 CCI. 以下以 adp1660.为例说明.

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/led_flash/
  - led_flash.c
  - led_flash.h
- kernel/drivers/media/platform/msm/camera_v2/sensor/flash/
  - Makefile
  - adp1660.c – QUP/I2C-based LED Flash driver
  - msm_led_flash.c – Generic LED Flash driver that handles user space IOCTLs
  - msm_led_flash.h – Generic LED Flash driver header file
  - msm_led_i2c_trigger.c – QUP/I2C- and CCI-based LED Flash interface driver
  - msm_led_torch.c – PMIC-based LED torch driver
  - msm_led_trigger.c – PMIC-based LED Flash driver
- <target>_camera*.dtsi 在如下目录 kernel/arch/arm/boot/dts/qcom/,如, msm8916-camera-sensor-mtp.dtsi

4.2 需要客制化的文件

添加闪光灯驱动时以下文件需要进行客制化修改。

4.2.1 更新device tree 文件

找到 camera .dtsi 文件, 如 msm8916-camera-sensor-mtp.dtsi, 添加 led_flash 节点，分配 qcom,led-flash-src 到 led_flash 节点。

- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/e.g., msm8916-camera-sensor-mtp.dtsi

客户需要基于闪光灯硬件接口选择驱动程序配置. 某些闪光灯需要 PMIC 电源来控制闪光灯的打开和关闭. 这种类型的驱动只需要调用 PMIC 的接口来控当前闪光状态就可以了。另外一些闪光灯则需要编程读写寄存器的方式来控制。

device tree 的节点入口需要根据闪光灯驱动修改。 PMIC, I2C 和 CCI 有不同的配置方式.

更多的解释参考 device tree 文件, 目录如下:
kernel/Documentation/devicetree/bindings/media/video$ vi msm-camera-flash.txt.

基于 PMIC 的闪光灯驱动

```
&soc {
   led_flash0: qcom,camera-led-flash {
        cell-index = <0>;
        compatible = "qcom,camera-led-flash";
        qcom,flash-type = <1>;
        qcom,torch-source = <&pm8941_torch>;
        qcom,flash-source = <&pm8941_flash0 &pm8941_flash1>;
   };
};
```

基于 QUP-/I2C 的闪光灯驱动

```
&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";
};
};

&cci {
```

基于 CCI 的闪光灯驱动

```
led_flash0: qcom,led-flash@60 {
       cell-index = <0>;
       reg = <0x60>;
       qcom,slave-id = <0x60 0x00 0x0011>;
       compatible = "qcom,led-flash";
       label = "adp1660";
       qcom,flash-type = <1>;
       qcom,cci-master = <0>;
       gpios = <&msmgpio 23 0>,
              <&msmgpio 24 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";
};
```

提示：任何一种闪光灯都需要配置一下参数与摄像头关联。

```
qcom,camera@0 {
       qcom,led-flash-src = <&led_flash0>;
};
};
```

对于 QUP-/I2C 工作方式的.dtsi 配置来说，&i2c 节点需要再开发板 .dtsi 文件中配置, 如., msm8916-mtp.dtsi. 以下参考代码显示了 QUP 节点如何与 i2c 设备的连接。 关于如何建立 QUP 设备节点，参考 [Q2].

```
i2c: i2c@f9928000 { /* BLSP1 QUP6 */
        cell-index = <6>;
        compatible = "qcom,i2c-qup";
        #address-cells = <1>;
        #size-cells = <0>;
```

```
        reg-names = "qup_phys_addr";
        reg = <0xf9928000 0x1000>;
        interrupt-names = "qup_err_intr";
        interrupts = <0 100 0>;
        qcom,i2c-bus-freq = <100000>;
        qcom,i2c-src-freq = <19200000>;
        qcom,sda-gpio = <&msmgpio 16 0>;
        qcom,scl-gpio = <&msmgpio 17 0>;
        qcom,master-id = <86>;
};
```

对于 PMIC 的闪光灯驱动, flash-source 和 torch-source 参数如下文件配置在 camera .dtsi, 他们的句柄在 kernel/arch/arm/boot/dts/<target>-leds.dtsi 定义, 如： msm8916-leds.dtsi.

```
qcom,leds@d300 {
   status = "okay";
   pm8941_flash0: qcom,flash_0 {
        qcom,max-current = <1000>;
        qcom,default-state = "off";
        qcom,headroom = <3>;
        qcom,duration = <1280>;
        qcom,clamp-curr = <200>;
        qcom,startup-dly = <3>;
        qcom,safety-timer;
        label = "flash";
        linux,default-trigger =
             "flash0_trigger";
        qcom,id = <1>;
        linux,name = "led:flash_0";
        qcom,current = <625>;
   };

   pm8941_flash1: qcom,flash_1 {
        qcom,max-current = <1000>;
        qcom,default-state = "off";
        qcom,headroom = <3>;
        qcom,duration = <1280>;
        qcom,clamp-curr = <200>;
        qcom,startup-dly = <3>;
        qcom,safety-timer;
        linux,default-trigger =
             "flash1_trigger";
        label = "flash";
        qcom,id = <2>;
```

```
        linux,name = "led:flash_1";
        qcom,current = <625>;
};

   pm8941_torch: qcom,flash_torch {
        qcom,max-current = <200>;
        qcom,default-state = "off";
        qcom,headroom = <0>;
             qcom,startup-dly = <1>;
             linux,default-trigger =
                  "torch_trigger";
             label = "flash";
             qcom,id = <2>;
             linux,name = "led:flash_torch";
             qcom,current = <200>;
             qcom,torch-enable;
        };
   };
. . .
};
```

4.2.2 添加闪光灯驱动文件

对于 QUP/I2C 型闪光灯和 CCI 型闪光灯，需要在如下目录添加<led>.c 文件，例如 adp1660.c：

- FILE – kernel/drivers/media/platform/msm/camera_v2/sensor/flash/
  - <led>.c

在该<led>.c 中，需要实现下列数据结构和函数。

对于 QUP/I2C 型闪光灯，i2c_driver 结构体和检测函数必须按如下定义：

```
static struct i2c_driver <led>_i2c_driver = {
        .id_table = <led>_i2c_id,
        .probe  = msm_flash_adp1660_i2c_probe,
        .remove = __exit_p(msm_flash_<led>_i2c_remove),
        .driver = {
                .name = FLASH_NAME,
                .owner = THIS_MODULE,
                .of_match_table = <led>_trigger_dt_match,
        },
};

static int msm_flash_<led>_i2c_probe(struct i2c_client *client,
```

```
                const struct i2c_device_id *id)
{
        if (!id) {
                pr_err("msm_flash_<led>_i2c_probe: id is NULL");
                id = <led>_i2c_id;
        }

        return msm_flash_i2c_probe(client, id);
}
```

对于 CCI 型闪光灯， platform_driver 结构体和检测函数必须按如下定义：

```
static struct platform_driver <led>_platform_driver = {
        .probe = msm_flash_<led>_platform_probe,
        .driver = {
                .name = "qcom,led-flash",
                .owner = THIS_MODULE,
                .of_match_table = <led>_trigger_dt_match,
        },
};

static int msm_flash_<led>_platform_probe(struct platform_device *pdev)
{
        const struct of_device_id *match;
        match = of_match_device(<led>_trigger_dt_match, &pdev->dev);
        if (!match)
                return -EFAULT;
        return msm_flash_probe(pdev, match->data);
}
```

如果该闪光灯被成功检测到， msm_led_flash_ctrl_t 结构体会被填入相应信息，相应的 v4l2 结点也会被创建。

```
static struct msm_led_flash_ctrl_t fctrl = {
        .flash_i2c_client = &<led>_i2c_client,
        .reg_setting = &<led>_regs,
        .func_tbl = &<led>_func_tbl,
};
```

下面的函数表需要指向 msm_led_i2c_trigger.c 定义的相应函数。这个函数表对 QUP/I2C 型闪光灯和 CCI 型闪光灯是一样的。

```
static struct msm_flash_fn_t <led>_func_tbl = {
        .flash_get_subdev_id = msm_led_i2c_trigger_get_subdev_id,
        .flash_led_config = msm_led_i2c_trigger_config,
        .flash_led_init = msm_flash_led_init,
        .flash_led_release = msm_flash_led_release,
        .flash_led_off = msm_flash_led_off,
        .flash_led_low = msm_flash_led_low,
        .flash_led_high = msm_flash_led_high,
};
```

- .flash_get_subdev_id() – 该函数返回 subdev_id。
- .flash_led_config () – 该函数处理来自用户空间的闪光灯状态设置命令，如 LOW、HIGH、OFF、RELEASE 等。
- .flash_led_init() – 该函数会把闪光灯寄存器序列 init_settings 写入闪光灯硬件。闪光灯 datasheet 会提供相应的寄存器设置序列。

  下面给出了 init_setting 结构体的一个例子。请根据所使用闪光灯的 datasheet 配置 reg_init、 init_array、addr_type、data_type 以及 delay。

  ```
  static struct msm_camera_i2c_reg_setting <led>_init_setting = {
          .reg_setting = <led>_init_array,
          .size = ARRAY_SIZE(<led>_init_array),
          .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
          .data_type = MSM_CAMERA_I2C_BYTE_DATA,
          .delay = 0,
  };
  ```

  对于 CCI 型闪光灯，CCI 硬件也是在该函数中初始化。

- .flash_led_release() – 该函数会在 camera 关闭时被调用。所有的 GPIO 都在这里被关闭。如果是 CCI 型闪光灯，CCI 硬件也在这里被关闭。
- .flash_led_off() – 该函数被用于将闪光灯从 LOW 或 HIGH 状态下关闭。在该函数中，闪光灯寄存器序列 off_settings 会被写入 LED 硬件。请根据所使用闪光灯的 datasheet 配置 off_setting。

下面给出了 off_setting 结构体的一个例子。请根据所使用闪光灯的 datasheet 配置 off_settings、off_array、addr_type、data_type 和 delay。

```
static struct msm_camera_i2c_reg_setting <led>_off_setting = {
        .reg_setting = <led>_off_array,
        .size = ARRAY_SIZE(<led>_off_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
        .delay = 0,
};
```

- .flash_led_low() – 该函数被用于将闪光灯置为 LOW 状态。在该函数中，闪光灯寄存器序列 low_settings 会被写入闪光灯硬件。请根据所使用闪光灯的 datasheet 配置 low_setting。

  下面给出了 low_setting 结构体的一个例子。请根据所使用闪光灯的 datasheet 配置 low_setting、low_array、addr_type、data_type 和 delay。

```
static struct msm_camera_i2c_reg_setting <led>_low_setting = {
        .reg_setting = <led>_low_array,
        .size = ARRAY_SIZE(<led>_low_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
        .delay = 0,
};
```

- .flash_led_high() –该函数被用于将闪光灯置为 HIGH 状态。在该函数中，闪光灯寄存器序列 high_settings 会被写入闪光灯硬件。请根据所使用闪光灯的 datasheet 配置 high_setting。

  下面给出了 high_setting 结构体的一个例子。请根据所使用闪光灯的 datasheet 配置 high_setting、high_array、addr_type、data_type 和 delay。

```
static struct msm_camera_i2c_reg_setting <led>_high_setting = {
        .reg_setting = <led>_high_array,
        .size = ARRAY_SIZE(<led>_high_array),
        .addr_type = MSM_CAMERA_I2C_BYTE_ADDR,
        .data_type = MSM_CAMERA_I2C_BYTE_DATA,
        .delay = 0,
};
```

5 EEPROM 驱动

本节以基线中已有 EEPROM 驱动为例介绍了实现新驱动的方法。

5.1 EEPROM 驱动目录结构

以 sunny_q5v22e 为例：

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/eeprom/
  - eeprom.c
  - eeprom.h
- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/eeprom_libs/sunny_q5v22e/
  - Android.mk
  - sunny_q5v22e_eeprom.c
- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

5.2 需要修改的文件

在添加新 EEPROM 驱动时，下面给出的文件都需要被更新或修改。

5.2.1 更新device tree文件

在平台相对应的 camera .dtsi 文件中，例如 msm8916-camera-sensor-mtp.dtsi，添加一个新的 EEPROM 项，并且将 qcom,eeprom-src 指向该 EEPROM 项。

- <target>_camera*.dtsi in kernel/arch/arm/boot/dts/qcom/, e.g., msm8916-camera-sensor-mtp.dtsi

在该文件中，下列.dtsi 参数必须做相应修改。kernel/Documentation/devicetree/bindings/media/video/msm-eeprom.txt 给出了这些参数的说明。

```
&cci {
                eeprom3: qcom,eeprom@6c {
                cell-index = <3>;
                reg = <0x6c>;
```

```
qcom,eeprom-name = " sunny_q5v22e";
compatible = "qcom,eeprom";
qcom,slave-addr = <0x20>;
qcom,cci-master = <0>;
qcom,num-blocks = <4>;
qcom,page0 = <1 0x0100 2 0x01 1 1>;
qcom,poll0 = <0 0x0 2 0 1 1>;
qcom,mem0 = <0 0x0 2 0 1 0>;
qcom,page1 = <1 0x3d84 2 0xc0 1 1>;
qcom,poll1 = <0 0x0 2 0 1 1>;
qcom,mem1 = <0 0x3d00 2 0 1 0>;
qcom,page2 = <1 0x3d88 2 0x7010 2 1>;
qcom,poll2 = <0 0x0 2 0 1 1>;
qcom,mem2 = <0 0x3d00 2 0 1 0>;
qcom,page3 = <1 0x3d8A 2 0x70F4 2 1>;
qcom,pageen3 = <1 0x3d81 2 0x01 1 10>;
qcom,poll3 = <0 0x0 2 0 1 1>;
qcom,mem3 = <228 0x7010 2 0 1 1>;

cam_vdig-supply = <&pm8226_l5>;
cam_vana-supply = <&pm8226_l19>;
cam_vio-supply = <&pm8226_lvs1>;
qcom,cam-vreg-name = "cam_vdig","cam_vana", "cam_vio";
qcom,cam-vreg-type = <0 1 2>;
qcom,cam-vreg-min-voltage = <1200000 2850000 0>;
qcom,cam-vreg-max-voltage = <1200000 2850000 0>;
qcom,cam-vreg-op-mode = <200000 80000 0>;
qcom,gpio-no-mux = <0>;
gpios = <&msmgpio 26 0>,
        <&msmgpio 37 0>,
        <&msmgpio 36 0>;
qcom,gpio-reset = <1>;
qcom,gpio-standby = <2>;
qcom,gpio-req-tbl-num = <0 1 2>;
qcom,gpio-req-tbl-flags = <1 0 0>;
qcom,gpio-req-tbl-label = "CAMIF_MCLK",
        "CAM_RESET1",
        "CAM_STANDBY";
qcom,cam-power-seq-type = "sensor_vreg","sensor_vreg",
        "sensor_vreg", "sensor_clk",
        "sensor_gpio", "sensor_gpio";
qcom,cam-power-seq-val = "cam_vdig","cam_vana",
        "cam_vio", "sensor_cam_mclk",
        "sensor_gpio_reset",
```

```
                        "sensor_gpio_standby";
                qcom,cam-power-seq-cfg-val = <1 1 1 24000000 1 1>;
                qcom,cam-power-seq-delay = <1 1 1 5 5 10>;
        };


        qcom,camera@20 {
                qcom,eeprom-src = <&eeprom3>;
       };
};
```

5.2.2 更新摄像头驱动文件

以 ov5648_q5v22e_lib.c 为例，sensor_lib_t 结构体的 eeprom_name 字段需要设置成相应的 EEPROM 注册名。

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/ov5648_q5v22e/ ov5648_q5v22e_lib.c

```
static sensor_lib_t sensor_lib_ptr = {
  /* sensor slave info */
  .sensor_slave_info = &sensor_slave_info,
  /* sensor init params */
  .sensor_init_params = &sensor_init_params,
 /* sensor eeprom name */
  .eeprom_name = "sunny_q5v22e",
```

5.2.3 添加EEPROM驱动文件

对于新的 EEPROM 驱动，需要添加下列 <eeprom>.c 文件。

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/eeprom_libs/eeprom/
  - Android.mk
  - <eeprom>.c

  每个新的<eeprom>.c 文件都 必须设置下面的函数表，并且实现相应函数。如果有某些函数没有定义，则需要将其设置为 NULL。

```
static eeprom_lib_func_t <eeprom>_lib_func_ptr = {

  .get_calibration_items = NULL,
  .format_calibration_data = NULL,
  .do_af_calibration = NULL,
  .do_wbc_calibration = NULL,
```

```
  .do_lsc_calibration = NULL,
  .do_dpc_calibration = NULL,
  .get_dpc_calibration_info = NULL,
  .get_raw_data = NULL,
};
```

- .get_calibration_items() – 该函数需要返回 EEPROM 支持的功能项。根据实际支持情况，每项都会相应置为 TRUE 或 FALSE。

```
void <eeprom>_get_calibration_items(void *e_ctrl)
{
  sensor_eeprom_data_t *ectrl = (sensor_eeprom_data_t *)e_ctrl;
  eeprom_calib_items_t *e_items = &(ectrl->eeprom_data.items);
  e_items->is_insensor = TRUE;
  e_items->is_afc = FALSE;
  e_items->is_wbc = TRUE;
  e_items->is_lsc = TRUE;
  e_items->is_dpc = FALSE;
}
```

  - Is_insensor – 如果 Sensor 内置 EEPROM，该项为 TRUE。
  - Is_afc – 如果支持 AF 校准，该项为 TRUE。
  - Is_wbc – 如果支持白平衡校准，该项为 TRUE。
  - Is_lsc – 如果支持 Lens Shading 校准，该项为 TRUE。
  - Is_dpc – 如果支持坏点校正，该项为 TRUE。

- .format_calibration_data() – 该函数被用于格式化将要写入 EEPROM/摄像头模组的数据。请根据 EEPROM/摄像头 datasheet 确定 addr_type、 data_type 和寄存器设置。在该函数结尾处， reg_setting 将存有符合 EEPROM/摄像头格式要求的 wb、lsc、afc、dpc 等数据。

```
void <eeprom>_format_calibration_data(void *e_ctrl) {
  SLOW("Enter");
  sensor_eeprom_data_t *ectrl = (sensor_eeprom_data_t *)e_ctrl;
  uint8_t *data = ectrl->eeprom_params.buffer;

  g_reg_setting.addr_type = MSM_CAMERA_I2C_WORD_ADDR;
  g_reg_setting.data_type = MSM_CAMERA_I2C_BYTE_DATA;
  g_reg_setting.reg_setting = &g_reg_array[0];
  g_reg_setting.size = 0;
  g_reg_setting.delay = 0;
  <eeprom>_format_wbdata(ectrl);
  <eeprom>_format_lensshading(ectrl);
```

```
  SLOW("Exit");
}
```

- .do_af_calibration() – 该函数处理 AF 校准相关操作。
- .do_wbc_calibration() – 该函数处理白平衡校准相关操作。
- .do_lsc_calibration() – 该函数处理 Lens Shading 校准相关操作。
- .do_dpc_calibration() – 该函数处理坏点校正相关操作。