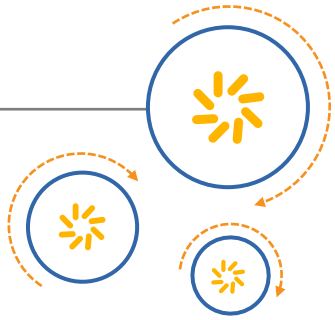
Qualcomm Technologies, Inc.



多媒体驱动程序开发和调通指南 – 摄像头

80-NU323-2SC 版本 H

2017 年 1 月 31 日

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

修订记录

版本	日期	说明
A	2014 年 11 月	初始版本
B	2015 年 1 月	针对 MSM8992 芯片组进行了更新
C	2015 年 2 月	更新了第 7 章
D	2015 年 4 月	针对 MSM8952 和 MSM8996 芯片组进行了更新
E	2015 年 12 月	更新了第 3.3.2.7.1 节
F	2016 年 5 月	更新了以下内容： ▪ 第 3.1 节 ▪ 第 7 章标题中包含了 MSM8998
G	2016 年 11 月	更新了第 3.3.2.4 节和第 3.3.2.7.1 节
H	2017 年 1 月	新增了第 3.3.2.7.2 节 更新了第 3.3.2.7.1 节 更新了第 3.3.2.4 节中的表 更新了第 9.2.1 节 更新了以下内容，以包括 SDM660/SDM630： ▪ 第 3.1 节中的注释 ▪ 第 7 章

目录

图

1 简介

1.1 用途

本文档为摄像头传感器和相关模块提供驱动程序开发指南，介绍如何在 MSM8x26/MSM8x28、MSM8926/MSM8928、MSM8974、APQ8084、MSM8992、MSM8994、MSM8996、MSM8909、MSM8916、MSM8952、MSM8936/MSM8939、MSM8998、SDM660 和 SDM630 Android 平台上调通摄像头。

其他多媒体技术的驱动程序开发指南以及调通步骤，将在以下各个文档中分别进行介绍：

- *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)

摄像头传感器框架包括以下各个组件的配置：

- 传感器
- CSIPHY
- CSID
- 摄像头控制接口 (CCI)
- 致动器
- 闪光灯
- EEPROM
- Chromatix™

本文档虽然是基于 MSM8916 代码库编写而成，但文档中的很多信息同样适用于所有 MSM8x26/MSM8x28、MSM8926/MSM8928、MSM8974、APQ8084、MSM8992、MSM8994、MSM8996、MSM8909、MSM8916、MSM8952 和 MSM8936/MSM8939 芯片组的 Linux 摄像头代码。芯片组之间特定的差异在各个章节中介绍。

1.2 约定

函数声明、函数名称、类型声明、属性以及代码示例以不同字体表示，例如 `#include`。

代码变量括在尖括号内，例如 `<number>`。

要输入的命令以不同字体显示，例如 **`copy a:*.* b:`**。

按钮和按键名称以粗体显示，例如，点击 **Save** 或按 **Enter** 键。

带阴影的部分表示本版本文档中新增的或已进行更改的内容。

1.3 技术协助

针对本文档中的信息，如需协助或说明，可通过 https://createpoint.qti.qualcomm.com/ 向 Qualcomm Technologies, Inc. (QTI) 提交用例。

如果您无法访问 CDMATech 支持网站，可在注册后进行访问，或者发送电子邮件至 support.cdmatech@qti.qualcomm.com。

2 预调通指南

本章介绍如何搜索现有 PVL 驱动程序以及了解非 PVL 组件选择对整个摄像头日程的影响。建议本文档读者在进行摄像头调通前先查阅此信息。

2.1 摄像头传感器选择和开发时间指南

要了解摄像头选择和基于该选择的开发时间/资源投入指南，可查看解决方案 [00028471]。

2.2 访问 PVL 驱动程序

有关如何访问 PVL（推荐供应商列表）驱动程序的指南，可参见 *Qualcomm CreatePoint Hardware Component Quick Start Guide（英文版）*(80-NC193-10) 或 *Qualcomm CreatePoint Hardware Component Quick Start Guide（中文版）*(80-NC193-10SC)。通过这些指南可以查询特定芯片组的 PVL 摄像头驱动程序列表，还可以下载一个或多个驱动程序。

2.3 有用资源

查阅列于解决方案 [00028470] 的“传感器调通”部分下的重要文档，然后再开始传感器的调通工作。如果调通过程中遇到问题，QTI 客户工程部可通过 Salesforce 用例提供帮助。有关正确识别问题领域的指南，可参见解决方案 [00028523]。

3 传感器驱动程序调通

本章介绍调通 Android 平台上 MSM8916 器件摄像头传感器硬件的必要信息。

注： 除非另有说明，否则所有信息都适用于 Bayer 传感器。

3.1 YUV 和 Bayer 传感器的参考驱动程序

本节给出适用于 MSM8916 的 Bayer 和 YUV 传感器的参考驱动程序列表。

Bayer 参考驱动程序

用户空间驱动程序位于 $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/ 中。

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

YUV 参考驱动程序

用户空间驱动程序位于 $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/ 中。

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

内核驱动程序位于 kernel/drivers/media/platform/msm/camera_v2/sensor 中。

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

注： 在 MSM8996/MSM8998/SDM660/SDM630 的较新基线中，YUV 传感器驱动程序的内核实现移至用户空间；OV5645 传感器驱动程序可作为本次更新的参考。

3.2 添加新驱动程序所需修改的文件

本节列出了编写新传感器驱动程序所必须修改的文件。

Bayer 传感器

设备树源文件为 kernel/arch/arm/boot/dts/qcom/ 下的 <target>_camera*.dtsi，例如 msm8916-camera-sensor-mtp.dtsi。客户应使用如下所示的摄像头插槽：

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

配置

在 vendor/qcom/proprietary/common/config/device-vendor.mk 中，将新库条目添加在要包括在该版本的文件中。

在 $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/module/sensor_init.c 中的 sensor_libs[] 数组中添加传感器名称。

注： 内核节点是通用的。如果和 QTI 的原理图一致，则无需更改内核节点。

注： 必须根据客户的硬件设计更改该文件。

用户空间传感器驱动程序为 $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/ 下的 <sensor>_lib.c/h 和 Android.mk，例如 imx135_lib.c/h。

除驱动程序外，还需修改 Android.mk（参见参考 makefile）。

注： 对于 64 位处理器，使用 arm64 目录，而不是 arm 目录。

YUV 传感器

设备树源文件为 kernel/arch/arm/boot/dts/qcom/ 下的 <target>_camera*.dtsi，例如 msm8916-camera-sensor-mtp.dtsi。客户应在 .dtsi 文件中添加如下所示的新条目：

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

用户空间传感器驱动程序为 $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/sensor_libs/ 下的 <sensor>_lib.c 和 Android.mk，例如 sp1628_lib.c。

除驱动程序外，还需修改 Android.mk（参见参考 makefile）。

内核传感器驱动程序为 kernel/drivers/media/platform/msm/camera_v2/sensor/ 下的 <sensor>.c 和 Makefile。

同时，必须在 kernel/arch/arm/configs/ 下的 <target>_defconfig 中添加 CONFIG_<sensor> 标记。

配置

在 vendor/qcom/proprietary/common/config/device-vendor.mk 中，将新库条目添加在要包括在该版本的文件中。

注： 必须根据客户的硬件设计更改该文件。

注： 对于 64 位处理器，使用 arm64 目录，而不是 arm 目录。

如以上示例所示，Bayer 传感器引入了摄像头插槽 (0/1/2) 的概念。但对于 YUV 传感器，必须在 dtsi 中添加新条目。

注： 编写新驱动程序时可从参考文件开始。

3.3 源代码解释

3.3.1 内核驱动程序

本节介绍创建内核驱动程序的必须信息。

3.3.1.1 GPIO 配置

如下所示，客户可以根据目标板配置传感器特有的 GPIO。有关各个属性的解释，可参考以下目录下的文档：

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

基于正在使用的软件，可使用其中一个方法配置 GPIO。

使用 pinctrl

对于使用 pinctrl 框架的芯片组（例如 MSM8909、MSM8916、MSM8936、MSM8939、MSM8992、MSM8994 等），可使用 .dtsi 中的 pinctrl 节点条目配置 GPIO，例如：

```
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>;
```

以上 pinctrl-XX 条目下列出的、指向管脚配置节点的 phandle 指针在 msmXXXX-pinctrl.dtsi 中定义。对于 MSM8916 是指 kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi。

使用 GPIO 控制

对于不使用 pinctrl 框架的芯片组（例如 MSM8x26/MSM8x28、MSM8926/MSM8928、MSM8974、MSM8084、MSM8952、MSM8996 等），可使用 .dtsi 中的 GPIO 节点条目配置 GPIO，例如：

```
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";
```

对于具有 CCI 硬件模块的芯片组，存在专用于数据和时钟的 GPIO。由于这些 GPIO 专用于 CCI 主器件，因此当 CCI 用于摄像头 I2C 时，客户必须使用这些设置。例如，以下 pinctrl 或基于 GPIO 控制的示例显示 GPIO 管脚 29 和管脚 30 专用于 CCI。

使用 pinctrl

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

以上 pinctrl-XX 条目下列出的、指向管脚配置节点的 phandle 指针在 msmXXXX-pinctrl.dtsi 中定义。对于 MSM8916 是指 kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi。

使用 GPIO 控制

```
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 拥有两个独立主器件（0 和 1 可用）。在多数用例中，使用一个 CCI 主器件便已足够。不过，有时可使用两个 CCI 主器件。例如，在前置和后置摄像头恰好拥有同一个 I2C 从器件地址时，将每个摄像头连接至不同的 CCI 主器件（使用与 CCI 主器件相连的独立 GPIO 物理管脚实现）可解决 I2C 从器件地址冲突问题。

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

3.3.1.2 时钟相关设置

对于 .dts 文件中的每个传感器节点，客户可以如下配置时钟源：

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

这两个属性中的排列顺序很重要。clock-name 属性的第 n 个值对应于 clocks 属性的第 n 个条目。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，依此类推。

以上设置在时钟框架中解析，客户无需修改设置。

注： 对于 MSM8916，建议对传感器驱动程序设置进行配置，以使用 23.88 MHz MCLK 输入。尽管通常默认以 24 MHz MCLK 输入写入驱动程序不会出现功能问题，但使用在可变频率（如 50 Hz）下工作的光源时可能出现带状干扰。

3.3.1.3 电源句柄

PMIC 情形

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

GPIO 情形

```
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 – 电源电压（模拟）
- CAM_VDIG – 电源电压（数字）
- CAM_VAF – 电源电压（致动器电压）
- CAM_VIO – 输入/输出电压（数字）

3.3.1.4 I2C 从器件配置

YUV 传感器

在 .dtsi 文件中：

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

这是一个 8 位地址，其中 7 个 MSB 为 I2C 从器件地址，1 个 LSB 是写标志 0。

注： 有关 Bayer 的信息，可参见章节 3.3.2.3。

3.3.2 用户空间驱动程序

本节介绍创建用户空间驱动程序的必要信息。

3.3.2.1 传感器初始化参数

传感器初始化参数包括支持的模式 (2D/3D) 以及设备中摄像头的安装位置（前/后）和安装角度。如果将安装角度指定为 360 度，则驱动程序将获取在内核 dtsi 中指定的安装角度。

```
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 传感器输出参数

传感器输出参数包括输出格式，用于指定是 Bayer 还是 YUV 传感器。连接模式指定传感器和接收器之间的接口（MIPI 或并联）。输出大小在 raw_output 中指定。

```
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 从器件配置

传感器从器件配置信息必须提供以下信息。

```
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),
  },
. . .
```

这是一个 8 位地址，其中 7 个 MSB 为 I2C 从器件地址，1 个 LSB 是写标志 0。

注： 有关 YUV 的信息，可参见章节 3.3.1.4。

3.3.2.3.1 传感器芯片 ID

必须从数据表中找到传感器型号 ID/芯片 ID 寄存器，然后写入 sensor_id_info 中。

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

3.3.2.3.2 上电/掉电时序

传感器上电/掉电时序以数组格式添加在每个用户空间传感器驱动程序的 msm_sensor_power_setting 结构中。

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

上电和掉电时序都可添加在 msm_sensor_power_setting_array 结构中。如果 power_down_setting/ size_down 数组成员没有添加，掉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),
 },
```

该 power_setting 将指向包含用于配置每个传感器的 GPIO/CLK/VREG 信息的数组。

```
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
  },
  . . .
```

在内核的 msm_camera_power_up() 中使用该结构配置传感器上电时序。

参见 kernel/include/media/msm_cam_sensor.h 中的枚举类型。

对于 YUV 摄像头驱动程序，msm_sensor_power_setting 应添加在内核驱动程序而不是用户空间驱动程序中。有关详细信息，可参见章节 3.1 中列出的其中一个 YUV 参考驱动程序。

根据客户硬件设计，电源可来自 .dtsi 文件中配置的 PMIC 或 GPIO。

3.3.2.4 尺寸表

如本例所示，使用 sensor_lib_out_info_t 结构添加尺寸表。

```
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 – 有效宽度
- y_output – 有效高度
- line_length_pclk – 包含消隐的宽度

注： 对于某些传感器，通过程序设定到传感器寄存器中的水平消隐值可能与传感器输出的实际消隐值不同。因此，建议在计算 line_length_pclk 参数值时使用测得的消隐值（实际值）。

- frame_length_lines – 包含消隐的高度
- vt_pixel_clk（视频计时时钟值）– 虚拟时钟值用于计算快门时间，并且 AEC 使用该值纠正带状干扰

  vt_pixel_clk = line_length_pclk * frame_length_lines * frame rate
- op_pixel_clk – 表示要设置 VFE 时钟，需要通过 MIPI 通道从摄像头获取的数据量

  op_pixel_clk = (传感器总数据传输速率) / 每个像素的位数

  例如，如果 MIPI DDR 时钟值（MIPI 摄像头传感器时钟通道的速度）为 300 MHz，且传感器在 4 条通道上传输数据，每个通道的数据传输速率为 600 MHz。从而，总的数据传输速率为 2400 MHz。对于每像素 10 位的 Bayer 数据，这相当于 op_pixel_clk 值为 2400/10 = 240 MHz。这些值必须按照传感器技术规范赋值。

  这些值可基于为摄像头传感器配置的寄存器设置进行计算。

注： 以下为最小消隐时间的要求：

芯片组	最低水平消隐要求	最低垂直消隐要求
MSM8x26/MSM8926/MSM8974	32	64
MSM8916/MSM8939	32	64
MSM8952/MSM8956/MSM8976	64	64
MSM8992/MSM8994	64	64
MSM8996/MSM8998/SDM660/SDM630	64	64

假设在任何用例的 VFE 输出上配置的最高分辨率为 1280 x 720。如果客户摄像头传感器不满足这些消隐要求或者用例的 VFE 输出上配置的最高分辨率小于 1280 x 720，可联系 QTI 的摄像头客户工程团队并附上用例及相关的详细信息。

3.3.2.5 Chromatix 参数

每种分辨率下为 Chromatix 参数更新适当的头文件。*Chromatix 6 Camera Tuning* (80 NK872-2) 中介绍了 Chromatix 头文件生成。

```
    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 传感器寄存器地址

必须根据传感器数据表正确赋值曝光和输出尺寸的寄存器地址字段。

3.3.2.6.1 曝光寄存器地址

```
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，曝光行数任何情况下都应小于 frame_length_lines 减去 vert_offset。

注： 参考数据表获取此偏移量（粗曝光时间的裕量）。

3.3.2.6.2 输出寄存器地址

传感器输出（x_output 和 y_output）、frame_length_lines 和 line_length_pclk 的寄存器地址必须根据摄像头传感器数据表中的寄存器规范进行配置。

```
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 接收器配置

传感器使用 MIPI CSI2 协议传输成像数据。QTI 接收器通过 MIPI CSI PHY 和 CSID 接收这些数据。

3.3.2.7.1 CSI-DPHY 配置

该结构显示 CSI 通道参数。

```
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,
    .combo_mode = 1, //Present on recent builds
  },
};
```

- csi_lane_assign – 有时客户硬件的管脚映射设计可能不同于 MSM™ 芯片组摄像头数据通道的参考管脚映射，例如传感器数据通道 0 可能与 MSM 数据通道 4 相连。可通过配置 csi_lane_assign 参数配置处理这类情况。该参数是一个 16 位值，每个位域的含义显示如下：

位的位置	含义
15:12	MSM 侧 PHY 通道编号，连接传感器的数据通道 3
11:8	MSM 侧 PHY 通道编号，连接传感器的数据通道 2
7:4	MSM 侧 PHY 通道编号，连接传感器的数据通道 1
3:0	MSM 侧 PHY 通道编号，连接传感器的数据通道 0

注： 通道 1 是留给时钟的。客户不应使用该通道映射任何数据通道。

- csi_lane_mask – 这个 8 位字段用于指示有效和要启用的 MIPI 通道。

在组合 PHY 上连接单个摄像头时，该值代表的含义如下：

位的位置	含义
7:5	保留
4	数据通道 3 是否有效？ ▪ 0 – 否 ▪ 1 – 是
3	数据通道 2 是否有效？ ▪ 0 – 否 ▪ 1 – 是
2	数据通道 1 是否有效？ ▪ 0 – 否 ▪ 1 – 是
1	时钟通道是否有效？ ▪ 0 – 否 ▪ 1 – 是 **注：**该位应始终设置为 1。
0	数据通道 0 是否有效？ ▪ 0 – 否 ▪ 1 – 是

例如，值 0x1F 表示摄像头有 4 个有效数据通道和一个时钟通道。

若是连接两个摄像头 （Cam0：2 通道摄像头连接至通道 2:0；Cam1：1 通道摄像头连接到通道 4:3），则该值代表的含义如下：

位的位置	含义
7:5	保留
4	Cam1 的数据通道 1 是否有效？ ▪ 0 – 否 ▪ 1 – 是
3	Cam1 的时钟通道是否有效？ ▪ 0 – 否 ▪ 1 – 是 **注：**只要摄像头存在，该位应始终设置为 1。
2	Cam0 的数据通道 1 是否有效？ ▪ 0 – 否 ▪ 1 – 是
1	Cam0 的时钟通道是否有效？ ▪ 0 – 否 ▪ 1 – 是 **注：**只要摄像头存在，该位应始终设置为 1。
0	Cam0 的数据通道 0 是否有效？ ▪ 0 – 否 ▪ 1 – 是

- csi_if – 未使用
- csid_core – 传感器所用 CSID 硬件的索引；若同时使用两个传感器，则该设置不能使用同一个值。
- csi_phy_sel – 传感器所用 CSI-PHY 核心的索引，对于每个传感器都应不同，除非外部 MIPI 桥将两个传感器连接到 MSM 的同一个 PHY 接口。该值应基于客户的硬件原理图进行配置。
- lane_cnt – 传感器在指定工作模式下输出数据所用数据通道的数量；根据传感器的最大数据通道能力（在数据表中提供）和驱动程序中配置的传感器寄存器设置来确定该值。
- combo_mode – 值 0 表示有一个摄像头连接至指定的 PHY 接口。值 1 表示有两个摄像头共享该 PHY 接口，在这种情况下：
  - 第一个摄像头 – 该摄像头连接至 PHY 通道 2:0。时钟通道连接至 PHY 通道 1。数据通道（最多 2 个）可连接至 PHY 通道 0 或通道 2
  - 第二个摄像头 – 该摄像头连接至 PHY 通道 4:3。时钟通道连接至 PHY 通道 4。数据通道可连接至 PHY 通道 3。
- settle_cnt（即稳定计数）– 必须根据传感器输出特性配置该值，以确保传感器的 PHY 发送器与 MSM 的 PHY 接收器无障碍同步。
  - 对于 28 nm 以及更小的 MSM 芯片，使用以下公式计算稳定计数：

    settle_cnt = T(HS_SETTLE)_avg /T(TIMER_CLK)，

    其中 T(HS_SETTLE)_avg = (T(HS_SETTLE)_min + T(HS_SETTLE)_max) / 2，如传感器数据表所指示。

– TIMER_CLK 指摄像头传感器所连接的 PHY 接口的工作频率（例如 PHY0 的 CAMSS_PHY0_CSI0PHYTIMER_CLK）。该值在 kernel/arch/arm/boot/dts/msm/msmXXXX-camera.dtsi 文件中设置，其中 XXXX 指正在使用 MSM 芯片组。另外，也可在摄像头数据流传输期间确认，方法是通过 adb shell 检查相应的时钟信息。例如，可通过命令提示窗口发出以下命令以确认 PHY0 定时器时钟值：

```
adb root
adb remount
adb shell
cd /sys/kernel/debug/clk/camss_phy0_csi0phytimer_clk
cat measure
```

– T(TIMER_CLK) 为工作频率等于 TIMER_CLK 时的时钟周期持续时间，以纳秒为单位表示。例如，TIMER_CLK 200 MHz 的 T(TIMER_CLK) 为 (1 * (10^9)) / (200 * (10^6)) = 5 ns。

注： 如果使用以上公式计算出的值未达到预期效果，可尝试使用 T(HS_SETTLE)_avg/T(TIMER_CLK) 和 T(HS_SETTLE)_min/T(TIMER_CLK) 中较小的值代替。

□ 对于 45 nm MSM 芯片，使用与 28 nm MSM 芯片相似的公式，其中的 T(TIMER_CLK) 替换为 T(DDR_CLK)。

– DDR_CLK 指摄像头传感器的 MIPI CLK 通道的工作频率，该值由通过传感器摄像头驱动程序设置的摄像头传感器 PLL 配置确定。

– T(DDR_CLK) 为工作频率等于 DDR_CLK 时的时钟周期持续时间，以纳秒为单位表示。例如，DDR_CLK 200 MHz 的 T(DDR_CLK) 为 (1 * (10^9)) / (200 * (10^6)) = 5 ns。

有关 T(HS_SETTLE) 的定义，可参见针对 D-PHY（版本 1.1）的 MIPI(R) 联盟规范。

为了防止上述系数在传感器工作时所处的不同数据流传输模式间发生变化，必须为摄像头传感器驱动程序中每个唯一的数据流传输模式单独配置 settle_cnt。

3.3.2.7.2 CSI-CPHY 配置

注： 本节为修订新增内容。

对于 CPHY 配置，将 is_csi_3phase 设置为 1 并按下文列出的表设置 lane_cnt、lane_mask 和 lane_assign。

```
.csi_params =
{
  .lane_cnt = 3,
  .settle_cnt = 0xA,
  .is_csi_3phase = 1,
},
```

	三路	双路	单路
lane_cnt	3	2	1
lane_mask	0x7	0x3	0x1
lane_assign	0x210	0x10	0x0

3.3.2.7.3 VFE 时钟频率计算

摄像头传感器时钟通道（即 MIPI DDR 时钟）的工作频率与激活的数据通道数决定摄像头传感器在指定操作模式下的总数据传输速率（吞吐量）。每个通道的数据传输速率是 MIPI DDR 时钟速度的两倍。例如，工作在 200 MHz MIPI DDR 时钟频率和 4 个激活通道下的摄像头传感器的总数据传输速率为 1600 Mbps（每个通道的数据传输速率为 200 * 2 = 400 Mbps）。

每个帧的分辨率、额外/虚拟像素/线数、水平消隐、垂直消隐、MIPI 包开销、每像素位数、数据格式、内部是否存在多个交错数据流以及每个流的数据传输速率/开销等，都会影响数据传输速率。指定工作模式下初步摄像头调通，计算：

*X = 帧宽 * (帧高*垂直消隐) * 每像素位数 * 每秒帧数 * (MIPI 协议和其他数据流的开销)*

在 VFE 时钟优化中为给定的 MSM 找到大于 X 的最接近的值作为 VFE 时钟的初始值。

3.3.2.7.4 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}
};
```

上面每行的第一个数值称为通道 ID (CID)。每个虚拟通道 (VC) 和数据类型 (DT) 的唯一组合应映射到一个唯一的 CID 值。对应指定 VC 的可能 CID 值有：

- 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 示例中，数据类型分别为 CSI_RAW10、0x35 和 CSI_EMBED_DATA 的三个信号流全部通过摄像头传感器在 VC 0 中发送，因此 CID 值的范围是 0 到 3。

3.3.2.8 寄存器设置

可配置摄像头传感器寄存器以通过 I2C 传输数据流。也可以按照本节所述为执行特定操作配置传感器。

3.3.2.8.1 初始化设置

在摄像头启动时执行一次性寄存器配置。

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

3.3.2.8.2 Grouphold 开启设置

将曝光设置的运行时更新分配给许多寄存器（粗略的曝光时间、每帧行数和增益），更新必须在一个图像换帧周期完成。这些寄存器都是双重缓存类型，并具有可一次更新完成的“分组保存参数”功能。若“分组保存参数”寄存器置 1，则转换数据保存在缓冲寄存器中。

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

3.3.2.8.3 Grouphold 关闭设置

若“分组保存参数”寄存器设置为 0，曝光寄存器的值就象同时在传输那样被更新，从而实现帧边界处的参数更改能够平稳更新。

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

3.3.2.8.4 分辨率设置

传感器可以有多种工作模式，例如以四分之一分辨率预览和以全分辨率拍照。不同的分辨率可按如下方式配置。

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

3.3.2.8.5 曝光设置

AEC 算法使用实际增益。由于必须将传感器硬件的实际增益转换为寄存器增益才能配置传感器，因此，客户必须基于传感器数据表实现以下函数。参考传感器数据表中的模拟和数字增益设置部分。

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

下列函数用于计算下一次曝光配置的曝光时间和增益。

```
xxxx_calculate_exposure()
```

以下函数用于准备下一次曝光配置的数组。

```
xxxx_fill_exposure_array()
```

注： 参见参考驱动程序。

3.3.2.8.6 启动数据流设置

MIPI 数据包通过传输开始 (SoT) 和传输结束 (EoT) 来定界。

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

在传感器驱动程序中，为满足该准则，应配置以下设置。

找出从 LP11 到 HS Tx 状态的时钟和数据通道：

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

3.3.2.8.7 停止数据流设置

应通过停止数据流设置将传感器的时钟和数据通道置于 LP11 状态。设置错误可导致 MIPI 摄像头与 MSM 的同步不一致。

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

3.3.3 使用 QUP/SPI

对于 BLSP 配置，参见 *BAM Low-Speed Peripherals for Linux Kernel Configuration and Debugging Guide* (80-NE436-1)。尽管本文档并非专门针对 MSM8916，但相同的概念对于 MSM8916 也适用。对于没有 CCI 硬件的芯片组，可使用 QUP 执行与摄像头的 I2C 数据传输。

注： 需要在具有 CCI 硬件的芯片组上配置 QUP/SPI 的客户必须联系 QTI。

3.3.4 更新 CCI 运行速度

根据 I2C 技术规范，CCI（适用于配有 CCI 的芯片组）可工作在 100 和 400 kHz 之间的频率下，并且由结构 msm_camera_sensor_slave_info 中名为 i2c_freq_mode 的参数进行控制。相关示例可参见 IMX135 传感器驱动程序文件 $(MM_CAMERA_DIR)/ mm-camera2/media-controller/modules/sensors/sensor_libs/imx135/imx135_lib.c。

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {

  . . .

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

. . .

};
```

可用的 I2C 频率模式在 kernel/include/media/msm_cam_sensor.h 中定义，它们分别是标准 (100 kHz)、高速 (400 kHz) 和定制模式。

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

对于定制模式，MSM8916 的 CCI 调试参数设置信息位于 kernel/arch/arm/boot/dts/qcom/msm8916-camera.dtsi 中。

```
&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";
};
```

如果使用定制 CCI 配置，则可以通过以下公式计算 I2C 频率速度信息：

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

由于 CCI 时钟频率通常为 19.2 MHz，因此标准 CCI 频率速度为：

CCI 时钟 = 19.2 MHz / (78 + 114) = 100 kHz

如果无 CCI 硬件，或将 QUP 用于 I2C，可通过 .dtsi 文件将 QUP 速度设置为 100 或 400 kHz。有关详细信息，可参见章节 8.1 的 kernel/arch/arm/boot/dts/qcom/msm8909.dtsi 文件中节点 i2c_3 的条目 qcom,clk-freq-out。

4 AF 致动器驱动程序

本章提供相关操作指南，以方便客户查看基本版本中提供的参考 AF 致动器驱动程序自己编写 AF 致动器驱动程序。

4.1 AF 致动器驱动程序目录结构

客户若要编写新 AF 致动器驱动程序，可借鉴以下参考 AF 致动器驱动程序。本示例涉及基本版本中 imx135 传感器驱动程序使用的致动器 dw9714。

- $(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 位于 kernel/arch/arm/boot/dts/qcom/ 中，例如 msm8916-camera-sensor-mtp.dtsi

4.2 需要更新/添加的文件

为添加新的 AF 致动器驱动程序，必须更新或修改以下指定的文件。

4.2.1 更新设备树文件

在目标的摄像头 .dtsi 文件（例如 msm8916-camera-sensor-mtp.dtsi）中，为致动器节点添加一个条目并将 qcom,actuator-src 分配为该致动器节点。

- <target>_camera*.dtsi 位于 kernel/arch/arm/boot/dts/qcom/ 中，例如 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 设置 AF 致动器电源

可参考每个参考芯片组平台 DTSI 文件指定 AF 致动器电源，因为正确的设置可能略有差异。在 MSM8916 中，使用摄像头传感器节点指定 cam_vaf-supply，但在 MSM8992、MSM8994、MSM8996、MSM8952 中，需要在 MSM8992、MSM8994、MSM8996、MSM8952 致动器节点中指定：

以下是针对 MSM8916 的示例：

注意，AF 的电源随摄像头传感器一同指定，它是每个 vreg 名称、类型、最小电压、最大电压和工作模式列表的第四个条目。

```
&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>;
```

以下是针对 MSM8992/MSM8994 的示例：

调压器 (PMIC) 在“qcom,actuator”设备树节点下指定。

```
&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 为致动器添加可选的 GPIO 控制管脚

在某些情况下，可将 GPIO 管脚而非 PMIC 的默认调压器用作致动器电源。首先应删除上节中介绍的调压器节点信息，然后必须添加以下 DTSI 设备树属性：

以下是针对 MSM8916 的示例：

```
&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>;        };
}
```

MSM8916 的 GPIO 调用序列（即 gpio_set_value_cansleep）位于 msm_actuator_power_up() / msm_actuator_power_down() 中。

以下是针对 MSM8992/MSM8994 的示例：

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

若是 MSM8992/MSM8994，致动器的 GPIO (“qcom,gpio-vaf”)被摄像头传感器节点的 GPIO 列表中的索引引用。在摄像头传感器上电/掉电时序（例如 msm_camera_power_up/down）中，可启用或禁用致动器电源。

4.2.4 更新传感器驱动程序文件

以 imx135_lib.c 驱动程序文件为例，为使致动器驱动程序与传感器相关联，则必须给 sensor_lib_t 结构中的 actuator_name 字段赋值。

- $(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 添加 AF 致动器文件

必须为新致动器驱动程序添加以下 <actuator>_actuator.c 和 <actuator>_actuator.h 文件。

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

<actuator>_actuator.c 文件仅为返回 actuator_lib_ptr 的层。

```
#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 文件应包含为调试致动器而定义的以下参数。该文件的有些参数来自致动器数据表，有些参数是执行 AF 调试后的结果。

```
{
    /* 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 添加 AF 算法调试文件

必须为照相机和摄像机模式添加以下 AF 算法调试文件。有关 AF 算法的调试信息，参见 *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 闪光灯驱动程序

本章提供相关操作指南，以方便客户查看基本版本中提供的参考 LED 闪光灯驱动程序自己编写 LED 闪光灯驱动程序。

5.1 LED 闪光灯驱动程序目录结构

参考 LED 闪光灯程序可基于 PMIC、QUP/I2C 或 CCI（适用于配有 CCI 硬件块的芯片组）。以下示例使用 LED 闪光灯驱动程序 adp1660.c 文件。

- $(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 – 基于 QUP/I2C 的 LED 闪光灯驱动程序
  - msm_led_flash.c – 可处理用户空间 IOCTL 的通用 LED 闪光灯驱动程序
  - msm_led_flash.h – 通用 LED 闪光灯驱动程序头文件
  - msm_led_i2c_trigger.c – 基于 QUP/I2C 或 CCI 的 LED 闪光灯接口驱动程序
  - msm_led_torch.c – 基于 PMIC 的 LED 聚光灯驱动程序
  - msm_led_trigger.c – 基于 PMIC 的 LED 闪光灯驱动程序
- <target>_camera*.dtsi 位于 kernel/arch/arm/boot/dts/qcom/ 中，例如 msm8916-camera-sensor-mtp.dtsi

5.2 需要修改的文件

为添加新的 LED 闪光灯驱动程序，必须更新或修改以下指定的文件。

5.2.1 更新设备树文件

在目标摄像头 .dtsi 文件（例如 msm8916-camera-sensor-mtp.dtsi）中，为 led_flash 节点添加一个条目并将 qcom,led-flash-src 分配为 led_flash 节点。

- <target>_camera*.dtsi 位于 kernel/arch/arm/boot/dts/qcom/ 中，例如 msm8916-camera-sensor-mtp.dtsi

OEM 可根据 LED 闪光灯硬件来确定配置哪种类型的接口驱动程序。有些 LED 闪光灯硬件需要在输入上的电源来实现开启/关闭。对于这样的 LED 闪光灯硬件，OEM 可使用基于 PMIC 的 LED 闪光灯驱动程序从 PMIC IC 提供电流/电源。该驱动程序非常简单，只需调用 PMICAPI 来控制各种闪光灯状态对应的电流/功率电平。其他 LED 闪光灯硬件必须通过编程寄存器设置来实现开启/关闭。对于这类硬件，OEM 可使用基于 QUP 或 I2C 的 LED 闪光灯驱动程序。

设备树文件中的节点条目须根据 LED 闪光灯驱动程序的类型（基于 -PMIC、基于 I2C 或基于 CCI）进行更改。

有关设备树文件中各个字段的更多信息和解释，可参见 kernel/Documentation/devicetree/bindings/media/video$ vi msm-camera-flash.txt 中的内核。

基于 PMIC 的 LED 闪光灯驱动程序

```
&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 LED 闪光灯驱动程序

```
&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 LED 闪光灯驱动程序

```
    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>;
    };
```

注： 对于以上三种 LED 闪光灯驱动程序的任意一种，都必须添加以下条目才能使其与传感器相关联：

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

要将基于 QUP/I2C 的条目添加到摄像头 .dtsi 文件中，尽早在目标 .dtsi 文件（例如 msm8916-mtp.dtsi）中定义 &i2c 节点。以下示例代码片段显示了连接到 &i2c 的 QUP 节点。要创建新 QUP 设备节点，参见 *BAM Low-Speed Peripherals for Linux Kernel Configuration and Debugging Guide* (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>;
};
```

对基于 PMIC 的 LED 闪光灯驱动程序，在 kernel/arch/arm/boot/dts/<target>-leds.dtsi（例如 msm8916-leds.dtsi）中定义摄像头 .dtsi 文件中由 PMIC IC 提供的闪光源和聚光源参数及其句柄。

```
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 针对基于 CCI 的情形更改 GPIO 管脚编号

如果必须更改 GPIO 管脚编号，则更改以下两个位置：

- kernel/arch/arm/boot/dts/<target>-camera-sensor-mtp.dtsi（例如 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>;
```

可根据硬件原理图设计（GPIO 36、31、32）更改以下配置：

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

- kernel/arch/arm/boot/dts/<target>-camera-sensor-mtp.dtsi（例如 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 */
    };
};
```

可根据硬件原理图设计（GPIO 36、31、32）更改以下配置：

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

5.2.3 添加 LED 闪光灯驱动程序文件

对基于 QUP/I2C 和基于 CCI 的 LED 闪光灯驱动程序，需要在以下目录中添加 <led>.c 文件，例如 adp1660.c：

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

在 <led>.c 驱动程序文件中必须实现以下数据结构和函数。

对基于 QUP/I2C 的 LED 闪光灯驱动程序，必须按如下定义 i2c_driver 结构和探针函数：

```
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);
}
```

对基于 CCI 的 LED 闪光灯驱动程序，必须按如下定义 platform_driver 结构和探针函数：

```
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);
}
```

探测成功后，将完整填充 msm_led_flash_ctrl_t 结构并为 LED 闪光灯驱动程序创建 v4l2 节点。

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

应使用 msm_led_i2c_trigger.c LED 闪光灯接口驱动程序的适当函数初始化以下函数表。对基于 QUP-/I2C- 和基于 CCI 的 LED 闪光灯驱动程序而言，该函数表是相同的。

```
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() – 此函数返回客户端的 subdev_id。
- .flash_led_config () – 此函数是用于处理用户空间命令的句柄函数，用于将 LED 配置为各种状态，例如 LOW、HIGH、OFF、RELEASE 等状态。
- .flash_led_init() – 在此函数中可将 LED 闪光灯的 init_setting 写入 LED 硬件。LED 闪光灯数据表可提供需要写入 LED 硬件的寄存器设置序列。

  以下是 init_setting 结构示例：可参考 LED 闪光灯数据表获取 reg_init 设置、init_array 的大小、addr_type、data_type 和延迟信息。

```
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,
};
```

  若是基于 CCI 的 LED 闪光灯，CCI 硬件也使用此函数进行初始化。

- .flash_led_release() – 在摄像头关闭序列执行期间调用此函数。所有 GPIO 都在此关闭，如果使用基于 CCI 的 LED 闪光灯驱动程序，它将反初始化 CCI 硬件。
- .flash_led_off() – 调用此函数可关闭 LOW 或 HIGH 状态下的 LED。在此函数中将 LED 闪光灯驱动程序 off_setting 写入 LED 硬件。有关 off_setting 的信息，参见 LED 闪光灯驱动程序数据表。

  以下是 off_setting 结构示例：可参考 LED 闪光灯驱动程序数据表获取 off_settings 设置、off_array 的大小、addr_type、data_type 和延迟信息。

```
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() – 调用此函数可将 LED 设为 LOW 状态。在此函数中将 LED 闪光灯驱动程序 low_setting 写入 LED 硬件。有关 low_setting 的信息，可参见 LED 闪光灯驱动程序数据表。

以下是 low_setting 结构示例：可参考 LED 闪光灯驱动程序数据表获取 low_setting 设置、low_array 的大小、addr_type、data_type 和延迟信息。

```
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() – 调用此函数可将 LED 设为 HIGH 状态。在此函数中将 LED 闪光灯驱动程序 high_setting 写入 LED 硬件。有关 high_settings 的信息，可参见 LED 闪光灯驱动程序数据表。

  以下是 high_settings 结构示例：可参考 LED 闪光灯数据表获取 high_setting、high_array 的大小、addr_type、data_type 和延迟信息。

```
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 添加基于 PWM 的闪光灯驱动程序

要添加基于 PWM 的 LED 闪光灯驱动程序，可参见用例系统解决方案 #00028369 –“如何使用 GPIO 在 MSM8926 上生成 PWM”。对于其他目标器件，以类似方法实现用例系统解决方案 #00028369 中介绍的更改。

6 EEPROM 驱动程序

本章提供相关操作指南，以方便客户查看基本版本中提供的参考 EEPROM 驱动程序自己编写 EEPROM 驱动程序。

6.1 EEPROM 驱动程序目录结构

以下示例使用 EEPROM 驱动程序 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 位于 kernel/arch/arm/boot/dts/qcom/ 中，例如 msm8916-camera-sensor-mtp.dtsi

6.2 需要修改的文件

为添加新的 EEPROM 驱动程序，必须更新或修改以下指定的文件。

6.2.1 更新设备树文件

在目标的摄像头 .dtsi 文件（例如 msm8916-camera-sensor-mtp.dtsi）中，为 EEPROM 节点添加一个条目并将 qcom,eeprom-src 分配为 EEPROM 节点。

- <target>_camera*.dtsi 位于 kernel/arch/arm/boot/dts/qcom/ 中，例如 msm8916-camera-sensor-mtp.dtsi

必须使用以下 .dtsi 字段更新 <target>_camera*.dtsi 文件。有关设备树文件中各个字段的解释，可参见 kernel/Documentation/devicetree/bindings/media/video/msm-eeprom.txt 中的内核。

```
&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 更新传感器驱动程序文件

以 ov5648_lib.c 驱动程序文件为例，为使 EEPROM 驱动程序与传感器相关联，必须填写 sensor_lib_t 结构中的 eeprom_name 字段。

- $(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 添加 EEPROM 驱动程序文件

必须为新的 EEPROM 驱动程序添加以下 <eeprom>.c 文件。

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

  任何新 <eeprom>.c 文件都应映射和定义以下函数指针。所有未在此 EEPROM 驱动程序中定义的函数必须设置为 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() – 此函数应返回 EEPROM 模块所支持的配置。基于 EEPROM 所支持的配置，将指定标记设置为 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 – 如果传感器模块本身支持 EEPROM 配置，则将此标记设置为 TRUE。外部 EEPROM 均不可用。
  - Is_afc – 如果支持 AF 校准，将此标记设置为 TRUE。
  - Is_wbc – 如果支持白平衡校准，将此标记设置为 TRUE。
  - Is_lsc – 如果支持镜头阴影校准，将此标记设置为 TRUE。
  - Is_dpc – 如果支持缺陷像素校正，将此标记设置为 TRUE。

- .format_calibration_data() – 此函数用于格式化可写入 eeprom/ 传感器模块的数据。可参考 eeprom/ 传感器数据表获取 addr_type、data_type 和寄存器设置信息。在此函数结尾，reg_setting 应包含符合 eeprom/ 传感器格式要求的 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() – 此函数用于处理所有与 AF 相关的校准操作，如格式化数据和将其写入 EEPROM 以执行 AF 校准。
- .do_wbc_calibration() – 此函数用于处理所有与白平衡相关的校准操作，如格式化数据和将其写入 EEPROM 以执行白平衡校准。
- .do_lsc_calibration() – 此函数用于处理所有与镜头阴影校正相关的校准操作，如格式化数据和将其写入 EEPROM 以执行镜头阴影校准。
- .do_dpc_calibration() – 此函数用于处理所有与缺陷像素校正相关的校准操作，如格式化数据和将其写入 EEPROM 以执行缺陷像素校正。

7 MSM8952/MSM8992/MSM8994/MSM8996/MSM8998/SDM660/SDM630 更新

本章介绍新 MSM8952、MSM8992、MSM8994、MSM8996、MSM8998、SDM660 和 SDM630 代码中的不同传感器驱动程序结构。

7.1 传感器驱动程序更改

7.1.1 用户空间更改

在 MSM8994.LA.1.1 中重新规划了传感器驱动结构。主结构和驱动程序配置基本保持不变，但对一些结构成员的命名约定进行了更改，特别是带有后缀“_a”的“数组”类型。

同时也对驱动器文件样式进行了更改，从而将多数数据信息移至头文件。例如，文件 ov4688_lib.c 现在是一个很短的文件，其中只包含函数句柄，例如：

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

章节 3.3.2 中提到的多数数据结构都已移至相应的头文件。查看 ov4688_lib.h 文件，除了 I2C 寄存器数组结构外，现在还包含一个单独的合并数据结构 sensor_lib_ptr。实际上，嵌套结构可由多个类似于之前驱动程序（具有等效数据）的赋值构成。有关对命名约定的小变化，可参见文件 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;
```

有关内核头文件的更改信息，可参见头文件 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];
};
```

注： 添加了新的数组类型成员 power_setting_a 和 vc_cfg_a。

7.2 LED 闪光灯驱动程序更改

已添加统一的闪光灯驱动程序以支持不同类型的闪光灯（I2C、PMIC 和 GPIO）

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/flash/flash.c/h

已添加通用的内核驱动程序，以便可从用户空间接收相同的 IOCTL 调用命令来控制不同类型的闪光灯。

- Kernel/drivers/media/platform/msm/camera_v2/sensor/flash/msm_flash.c/h

7.2.1 基于 PMIC

7.2.1.1 用户空间更改

在用户空间中，相关文件位于以下目录下

- $(MM_CAMERA_DIR)/mm-camera2/media-controller/modules/sensors/flash_libs/pmic/

用户空间驱动程序将以下配置类型的配置数据发送到内核驱动程序以对 PMIC 进行相应配置。

- CFG_FLASH_INIT
- CFG_FLASH_RELEASE

- CFG_FLASH_OFF
- CFG_FLASH_LOW
- CFG_FLASH_HIGH

7.2.1.2 内核空间更改

客户可能需要根据电源来更改 dtsi 文件，例如 kernel/arch/arm/boot/dts/qcom/msm8994-camera-sensor-mtp.dtsi&soc {

```
led_flash0: qcom,camera-flash {
   cell-index = <0>;
   compatible = "qcom,camera-flash";
   qcom,flash-type = <1>;
   qcom,flash-source = <&pmi8994_flash0 &pmi8994_flash1>;
   qcom,torch-source = <&pmi8994_torch0 &pmi8994_torch1>;
   };
            };
```

注： 如果客户需要使用不同于软件/硬件设计的基于 PMIC 的解决方案，客户务必通过 Salesforce 用例系统与 QTI 就相关要求进行讨论。

参见 kernel/Documentation/devicetree/bindings/media/video/msm-camera-flash.txt

7.2.2 基于 I2C/GPIO

基于 I2C/GPIO 的闪光灯驱动程序的参考驱动程序不属于参考版本文件的一部分，但可通过统一客户设备的闪光灯架构来启用。客户务必通过 Salesforce 用例系统与 QTI 就相关要求进行讨论。

8 针对 MSM8909 的更新

本章介绍 MSM8909 代码中的不同驱动程序结构。

8.1 无 CCI 硬件

MSM8909 与 MSM8916、MSM8974 等其他芯片组的主要区别在于其不具备 CCI 硬件模块。这意味着 dtsi 文件（例如，kernel/arch/arm/boot/dts/qcom/msm8909-camera-sensor-mtp.dtsi）中的所有摄像头驱动程序特定条目均将列于传统 i2c 节点下。例如：

```
&i2c_3 {

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

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

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

    qcom,camera@0 {
    . . .
    };

    qcom,camera@1 {
    . . .
    };

};
```

对节点 &i2c_3 的引用是表示将 BLSP1 QUP3 用于 MSM8909 的摄像头操作。此节点在 kernel/arch/arm/boot/dts/qcom/msm8909.dtsi 中定义：

```
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 参考驱动程序

MSM8909 MTP 使用后置摄像头 OV8858（模块部件 q8v19w）。因此，有关驱动程序参考代码，可参见 ov8858_q8v19w_lib.c。有关 EEPROM，可参见 sunny_ov8858_q8v19w_eeprom.c。

9 故障排除

本章介绍在创建或配置摄像头调通所需组件遇到错误时应遵守的调试流程。应按本章介绍的顺序执行故障排除操作。

9.1 传感器故障排除

本节介绍在出现传感器模块安装/探测故障时应执行的各种流程。

9.1.1 模块安装

9.1.1.1 Bayer 传感器

1. 打开传感器文件。
2. 查看 sensor_slave_info 数组中 camera_id 的值。

```
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. 查看传感器 dtsi 文件中 cell-index 的值。

```
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. sensor_slave_info 数组中 camera_id 的值应与传感器 dtsi 文件中 cell-index 的值相同。

9.1.1.2 YUV 传感器

1. 将以下传感器 dtsi 设置添加到板载摄像头 dtsi 文件 (arch/arm/boot/dts) 中。

   如果通过彩色显示器查看本文档，或用彩色打印机打印本文档，则**红色粗体**表示要**添加**的代码。

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

   示例中**红色粗体**值是唯一的。它代表 dtsi 树中的摄像头 id。这里使用了 78，因此在调通其他摄像头时不能再使用 78。

2. 将摄像头时钟源添加到 arch/arm/mach-msm/clock-xxxx.c

   - 对于 MSM8974、MSM8x26/28 和 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"),
```

   - 对于 MSM8916/MSM8936/MSM8909/MSM8084/MSM8992/MSM8994 及更高版本的芯片组，则在 .dtsi 文件中指定 clk_mclk1_clk_src 和 clk_gcc_camss_mclk1_clk 设置。相关示例可参见 kernel/arch/arm/boot/dts/qcom/msm8916-camera-sensor-mtp.dtsi。

9.1.2 模块探测

9.1.2.1 上电/掉电时序

1. 如下所示，在文件“msm_camera_dt_util.c”中定义宏来启用日志功能。

```
--- 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. 从内核日志搜索关键字“msm_camera_power_up”。日志会指出失败的序列类型。

3. 确保上电/掉电设置数组满足传感器规格要求。

   以下代码片段是配置了上电/掉电时序的代码示例。

```
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,
     },
```

   在以下文件中提供了更多示例：
   Vendor/qcom/proprietary/mm-camera/mm-camera2/
   media-controller/modules/sensors/sensor_libs/xxxx/ xxxx_lib.c

9.1.2.2 CCI

CCI 可为摄像头从器件执行 I2C 操作。有些硬件组件无法运行 I2C 速度模式，如果运行，就可能出现问题。本节介绍如何更改 I2C 速度模式。

1. 在 CCI 版本源代码文件 (kernel/drivers/media/platform/msm/camera_v2/sensor/cci/msm_cci.c) 中启用日志功能来验证寄存器设置。

2. 通过更改 i2c_freq_mode 值来更改 CCI I2C 速度：
   - 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. 启用 CCI 时钟延长：

   要让强制 I2C 主控 (CCI) 时钟延长功能的传感器能正确运行，可尝试在 dtsi 文件中将属性“qcom,hw-scl-stretch-en”设置为 1。

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

   尽管此更改不会造成功能问题，但会降低 I2C 速度，因为主控 (CCI) 此时必须侦听缓慢的 SCL 上升信号，而不是根据内部 CCI 时钟确定所有时序。建议客户在进行此项更改前对 I2C 功能进行全面的压力测试，如有任何问题，可通过 Salesforce 通知 QTI 解决。

9.1.2.3 MCLK

MCLK 在 msm_sensor.c 文件中默认设置为 24 MHz。

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

可以在传感器电源设置数组中指定 MCLK 值，例如指定为 19.2 MHz。

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

对于 MSM8916 和 MSM8939，建议的 MCLK 值为 23.88 MHz。不过，使用默认的 24 MHz 输入 MCLK 配置传感器也可以接受。要为 MSM8916 设置该值，可在文件 kernel/arch/arm/boot/dts/qcom/msm8916-camera-sensor-mtp.dtsi 中设置以下条目

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

9.2 ISP 故障排除

9.2.1 SOF IRQ 超时

1. 打开 mm-camera/mm-camera2/media-controller/mct/bus/mct_bus.c
2. 搜索“mct_bus_sof_thread_run”函数并搜索错误消息“SOF freeze; Sending error message.”。检查日志中是否存在此消息：
   - 如果存在，表示 ISP 未从内核接收到 SOF IRQ 并且必须检查 CSID/CSIPHY/CAMIF。
   - 如果不存在，则表示 ISP 已经接收传感器传输的帧。

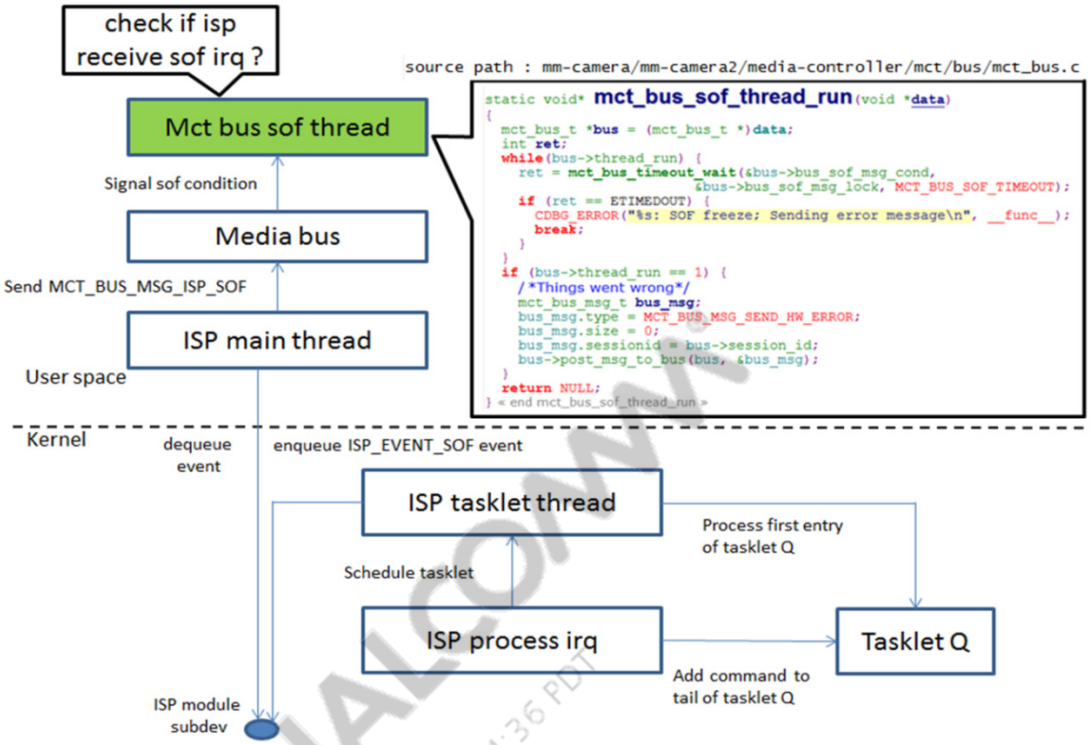


图 9-1 SOF IRQ 超时

9.2.2 VFE 溢出

VFE 时钟设置为小于传感器输出 MIPI 的数据传输速率时，将出现溢出。图 9-2 给出了由于 VFE 时钟小于传感器输出 MIPI 的数据传输速率而导致的溢出。

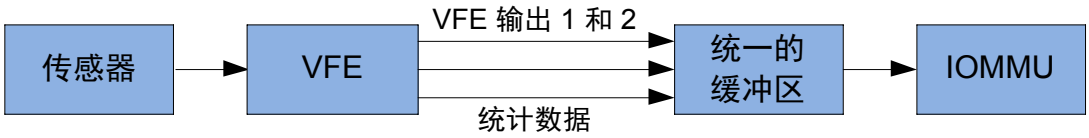


图 9-2 VFE 溢出

要判断是否已出现溢出，可检查以下位置的溢出日志文件：

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

典型溢出日志条目显示如下：

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

增大 VFE 时钟可防止溢出发生。要更改 VFE 时钟，需打开 Vendor/qcom/proprietary/mm-camera/mm-camera2/media-controller/modules/sensors/sensor_libs/xxxx/ xxxx_lib.c 文件并编辑 op_pixel_clk 参数。

9.2.3 CAMIF 错误状态

若声明的传感器输出大小与 VFE 实际接收到的传感器输出大小之间不匹配，将发生 CAMIF 错误。

1. 要验证这些设置，可通过 isp_hw.c 文件中的 isp_hw_camif_dump_cfg 函数对 CAMIF 设置执行转储。

```
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. 将调试消息中指示的帧大小与 ISP 传感器的帧大小进行比较。

   在以下 CAMIF 错误示例中，错误状态 0x9a70a00 表示 ISP 接收帧的大小为 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. 如果存在不匹配，则根本原因可能如下：
   - 检查传感器设置是否正确，是否满足分辨率大小的要求。例如，传感器的输出大小配置为 12 MB，但 ISP 的接收大小却配置为 8 MB。
   - 可能有些传感器无法确保在新分辨率设置发送至传感器后，最初的帧大小能够满足要求。在这种情况下，需要与传感器供应商一起解决此问题。

9.3 CSID 故障排除

为排除 CSID 故障，必须启用全部 CSID IRQ 以检查 CSID 是否收到 mipi 数据或错误位 IRQ。

若您通过彩色显示器查看本文档，或使用彩色打印机打印本文档，**红色粗体**表示要**添加**的代码，~~蓝色带删除线~~表示要被替换或~~删除~~的代码。

```
--- 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,
```

检查 CSID 是否接收 MIPI 数据

要查看 CSID 是否接到 MIPI 数据，需检查 IRQ 的位 0 到位 7。

- CAMSS_A_CSID_X_CSID_IRQ_STATUS 位 0-3（其中 X 是位编号）– 当 CSID 核心在 PHY 数据线 0-3 上接收到 SOT 时，将触发这些 IRQ
- CAMSS_A_CSID_X_CSID_IRQ_STATUS bit 4-7（其中 X 是位编号）– 当 CSID 核心在 PHY 数据线 0-3 上接收到 EOT 时，会触发这些 IRQ

以下日志示例显示一个 4 通道传感器 CSID IRQ 正常状态日志。

```
<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
```

如果触发了任意 mipi 信号错误 IRQ，可参见 [Q5] 查看 CSID IRQ 位定义，咨询传感器供应商或者修改传感器稳定计数器。

9.4 DPHY 故障排除

本节介绍两种 DPHY 故障排除方法。

查看 DPHY 调试日志

为排除 DPHY 故障，必须启用 DPHY 调试日志以查看硬件寄存器 CAMSS_A_CSI_PHY_X_MIPI_CSIPHY_INTERRUPT_STATUSY 是否收到任何 IRQ 错误。

```
--- 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
```

以下日志示例显示 CSIPHY IRQ 正常状态日志。

```
<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
```

如果触发了任意 CSIPHY 信号错误 IRQ，可参见 [Q5] 查看 CSID IRQ 位定义，咨询传感器供应商或者修改传感器稳定计数器。

检查 DPHY 索引与硬件布局之间的关系

如果 DPHY 有问题，则 DPHY 索引与硬件布局之间可能不匹配。将传感器 dtsi 设置中的相应值与传感器 xxxx_lib.c 文件中的相应值进行比较。这些值必须相互匹配才能工作正常。下表中的加粗项表示当前参数及其相应的硬件索引。

传感器的 dtsi 设置	传感器的 xxxx_lib.c 文件

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 参考资料

A.1 相关文档

标题	文档号
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
Chromatix™ 6 Camera Tuning!!	80-NK872-2
Camera Auto Focus Tuning Guide	80-N8459-1
BAM Low-Speed Peripherals for Linux Kernel Configuration and Debugging Guide	80-NE436-1
MSM8960 Voice/Audio Topology and Tools Overview	80-N7634-1
Qualcomm CreatePoint Hardware Component Quick Start Guide（英文版）	80-NC193-10
Qualcomm Createpoint Hardware Component Quick Start Guide（简体中文版）	80-NC193-10SC

A.2 缩略词和术语

缩略词或术语	定义
AF	自动对焦 (Auto Focus)
GCDB	全局组件数据库 (Global Components Database)
IRQ	中断请求 (Interrupt Request)
QRD	Qualcomm 参考设计 (Qualcomm Reference Design)
SOF	帧开始 (Start of Frame)