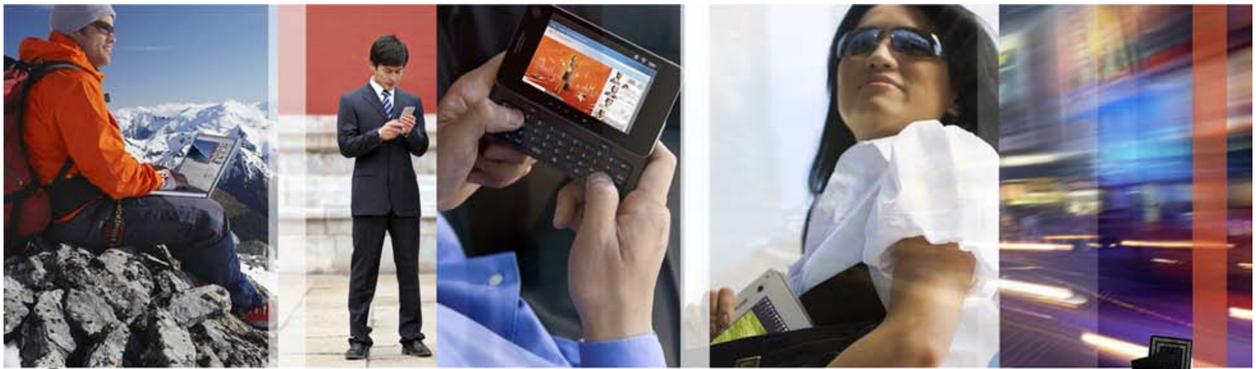

QUALCOMM CDMA Technologies

REDEFINING MOBILITY

# Android™ Camera Porting Guide

***80-VN756-1 C***

***April 5, 2011***

**Submit technical questions at:**
**https://support.cdmatech.com/**

## Qualcomm Confidential and Proprietary



**QUALCOMM Incorporated**
**5775 Morehouse Drive**
**San Diego, CA 92121-1714**
**U.S.A.**

# Contents


**1 Introduction........................................................................................ 5**

1.1 Purpose........................................................................................................ 5
1.2 Scope........................................................................................................... 5
1.3 Conventions ................................................................................................ 5
1.4 References................................................................................................... 5
1.5 Technical assistance.................................................................................... 6
1.6 Acronyms.................................................................................................... 6

**2 Camera Driver Porting ........................................................................ 7**

2.1 Kernel porting.............................................................................................. 7
2.1.1 Porting with provided camera sensors................................................ 7
2.1.2 Porting your own sensor drivers ....................................................... 10
2.2 User space porting ..................................................................................... 14

**3 MIPI CSI Configuration...................................................................... 18**

3.1 Meaning of parameters .............................................................................. 18
3.2 MIPI CSI configuration sequence.............................................................. 19

# Tables

Table 1-1 Reference documents and standards.............................................................. 5

# Revision history

| Revision | Date | Description |
|---|---|---|
| A | Jan 2009 | Initial release |
| B | Jan 2010 | Changed example build to the latest Donut build |
| C | Apr 2011 | Changed example build to the latest Gingerbread build on MSM8x60 |

# 1 Introduction

NOTE: Numerous changes were made to this document. It should be read in its entirety.

## 1.1 Purpose

This document provides guidelines for porting camera drivers to an Android™ platform.

## 1.2 Scope

This document is intended for software engineers who are planning to use Qcamera™ features on MSM7201A™, MSM7x25, MSM7x27, MSM8x50, MSM7x30/8x55, or MSM8x60 ASICs on the Android platform.

## 1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

Code variables appear in angle brackets, e.g., `<number>`.

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

Shading indicates content that has been added or changed in this revision of the document.

## 1.4 References

Reference documents, which may include QUALCOMM®, standards, and resource documents, are listed in Table 1-1. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

**Table 1-1 Reference documents and standards**

| Ref. | Document | |
|---|---|---|
| ***Qualcomm*** | | |
| Q1 | *Application Note: Software Glossary for Customers* | CL93-V3077-1 |
| Q2 | *Presentation: MSM7x27 Linux Camera Overview* | 80-VP828-1 |
| ***Resources*** | | |
| R1 | *MIPI Alliance Standard for CSI-2* | http://www.mipi.org/ |

## 1.5 Technical assistance

For assistance or clarification on information in this guide, submit a case to Qualcomm CDMA Technologies at https://support.cdmatech.com/.

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

## 1.6 Acronyms

For definitions of terms and abbreviations, see [Q1].

# 2 Camera Driver Porting

Before starting, verify that you are using the latest Android device tree and Qualcomm's proprietary source.

The information described in this document is based on the Sony 13MP Bayer sensor (IMX074) for primary camera and the Omni Vision VGA YUV sensor (OV7692) for secondary camera on build M8660AAABQNLYA1075.

## 2.1 Kernel porting

### 2.1.1 Porting with provided camera sensors

Sensor-specific information is described in one of the following board files that matches the MSM™ chipset IDs. The sensor information depends on the camera sensor used on your project:

- (Root)\Linux\android\kernel\arch\arm\mach-msm\board-halibut.c (for MSM7201A)
- (Root)\Linux\android\kernel\arch\arm\mach-msm\board-msm7x27.c
- (Root)\Linux\android\kernel\arch\arm\mach-msm\board-qsd8x50.c
- (Root)\Linux\android\kernel\arch\arm\mach-msm\board-msm7x30.c
- (Root)\Linux\android\kernel\arch\arm\mach-msm\board-msm8x60.c

To port the kernel, change the following in the board file:

1. Add the I2C driver for the sensor to be controlled to the msm8x60_i2c_devices[] array. It will register when the msm8x60_init() function is called. Add I2C device information in the msm_camera_boardinfo[] array. Also, change the slave address information of your sensor device.

   **Example**

```
static struct i2c_board_info msm_camera_boardinfo[] __initdata = {
   #ifdef CONFIG_IMX074
   {
         I2C_BOARD_INFO("imx074", 0x1A), // 0x1A is slave address
   },
   #endif
   #ifdef CONFIG_WEBCAM_OV7692
   {
         I2C_BOARD_INFO("ov7692", 0x78), // 0x78 is slave address
   },
   #endif
```

```
};
static struct i2c_registry msm8x60_i2c_devices[] __initdata = {
...
#ifdef CONFIG_MSM_CAMERA
    {
         I2C_SURF | I2C_FFA | I2C_FLUID,
         MSM_GSBI4_QUP_I2C_BUS_ID,
         msm_camera_boardinfo,
         ARRAY_SIZE(msm_camera_boardinfo),
   },
#endif
...
}
```

2. The msm_camera_sensor_info includes camera device information to bring up the device. To add a new sensor driver, add data that is the msm_camera_sensor_info type structure shown in the following example. Also, add to the array GPIO information that is related to the camera bringup, such as sensor reset, sensor power on/off, and AF actuator power on/off pins. The sensor_name in the array is used as a driver ID.

**Example**

```
struct msm_camera_device_platform_data msm_camera_device_data = {
   .camera_gpio_on  = config_camera_on_gpios,
   .camera_gpio_off = config_camera_off_gpios,
   .ioext.csiphy = 0x04800000,
   .ioext.csisz  = 0x00000400,
   .ioext.csiirq = CSI_0_IRQ,
   .ioclk.mclk_clk_rate = 24000000,
   .ioclk.vfe_clk_rate  = 228570000,
};
struct msm_camera_device_platform_data msm_camera_device_data_web_cam = {
   .camera_gpio_on  = config_camera_on_gpios_web_cam,
   .camera_gpio_off = config_camera_off_gpios_web_cam,
   .ioext.csiphy = 0x04900000,
   .ioext.csisz  = 0x00000400,
   .ioext.csiirq = CSI_1_IRQ,
   .ioclk.mclk_clk_rate = 24000000,
   .ioclk.vfe_clk_rate  = 228570000,
};
...
#ifdef CONFIG_IMX074 // For the primary camera sensor
static struct msm_camera_sensor_flash_data flash_imx074 = {
   .flash_type     = MSM_CAMERA_FLASH_LED, // type of flash
   .flash_src      = &msm_flash_src // source of flash
};
static struct msm_camera_sensor_info msm_camera_sensor_imx074_data = {
```

```
    .sensor_name    = "imx074",
    .sensor_reset   = 106, // GPIO for reset
    .sensor_pwd     = 85,  // GPIO for power on/off
    .vcm_pwd        = 1,   // GPIO for actuator on/off
    .vcm_enable     = 0,
    .pdata          = &msm_camera_device_data,
    .resource       = msm_camera_resources,
    .num_resources  = ARRAY_SIZE(msm_camera_resources),
    .flash_data     = &flash_imx074,
    .strobe_flash_data   = &strobe_flash_xenon,
    .csi_if         = 1 // 0: Parallel interface , 1: MIPI interface
};
struct platform_device msm_camera_sensor_imx074 = {
    .name  = "msm_camera_imx074",
    .dev   = {
          .platform_data = &msm_camera_sensor_imx074_data,
    },
};
#endif // CONFIG_IMX074
...
#ifdef CONFIG_WEBCAM_OV7692 // For the secondary camera sensor
static struct msm_camera_sensor_flash_data flash_ov7692 = {
    .flash_type = MSM_CAMERA_FLASH_LED,
    .flash_src  = &msm_flash_src
};
static struct msm_camera_sensor_info msm_camera_sensor_ov7692_data = {
    .sensor_name    = "ov7692",
    .sensor_reset   = 106,    // GPIO for reset
    .sensor_pwd = 85,     // GPIO for power on/off
    .vcm_pwd        = 1,      // GPIO for actuator on/off
    .vcm_enable = 0,
    .pdata      = &msm_camera_device_data_web_cam,
    .resource       = msm_camera_resources,
    .num_resources  = ARRAY_SIZE(msm_camera_resources),
    .flash_data = &flash_ov7692,
    .csi_if         = 1
};

static struct platform_device msm_camera_sensor_webcam = {
    .name  = "msm_camera_ov7692",
    .dev   = {
          .platform_data = &msm_camera_sensor_ov7692_data,
    },
};
```

```
#endif

static struct platform_device *surf_devices[] __initdata = {
...
#ifdef CONFIG_MSM_CAMERA
#ifdef CONFIG_IMX074
&msm_camera_sensor_imx074, // The primary camera sensor driver
#endif
...
#ifdef CONFIG_WEBCAM_OV7692
&msm_camera_sensor_webcam, // The secondary camera sensor driver
#endif
#endif
};
```

3. Change the driver name from msm_camera_ov7692 to your driver name in msm_clocks_8x60[] in Devices-8x60.c if you are using the secondary camera sensor.

**Example**

```
struct clk_lookup msm_clocks_8x60[] = {
...
CLK_8X60("csi_clk",  CSI1_CLK,  "msm_camera_ov7692.0", OFF),
...
CLK_8X60("csi_vfe_clk", CSI1_VFE_CLK,  "msm_camera_ov7692.0", OFF),
...
CLK_8X60("csi_pclk", CSI1_P_CLK,  "msm_camera_ov7692.0", OFF),
}
```

### 2.1.2 Porting your own sensor drivers

You can create your own sensor driver source and header files in the path by referring to the reference drivers and adding them to one of the following directories:

- For primary sensor driver
  - (Root)\Linux\android\kernel\drivers\media\video\msm\imx074.c
  - (Root)\Linux\android\kernel\drivers\media\video\msm\imx074.h
  - (Root)\Linux\android\kernel\drivers\media\video\msm\imx074_reg.c
- For secondary sensor driver
  - (Root)\Linux\android\kernel\drivers\media\video\msm\ov7692.c
  - (Root)\Linux\android\kernel\drivers\media\video\msm\ov7692.h

To port the kernel using your own sensor drivers:

1. Add to the driver source file all necessary functions as shown in the following example. Typically, the sensor registers file is stored in a separate file (e.g., imx074_reg.c) for modularity. The following example shows the ov7692_sensor_probe function, where the master clock rate (MCLK) can be set. The two functions in this example explain how a camera driver is registered first as platform_driver with the platform_driver_register function, which in turn registers the driver probe function.

### Example

```
static int ov7692_sensor_probe(const struct msm_camera_sensor_info
*info, struct msm_sensor_ctrl *s)
{
...
          /* Input MCLK = 24MHz */
          msm_camio_clk_rate_set(24000000);
   rc = ov7692_probe_init_sensor(info);
   if (rc < 0)
       goto probe_fail;
   s->s_init = ov7692_sensor_open_init;
   s->s_release = ov7692_sensor_release;
   s->s_config  = ov7692_sensor_config;
   s->s_camera_type = FRONT_CAMERA_2D; // Camera type
   s->s_mount_angle = 0; // mounted angle of the sensor
   ov7692_probe_init_done(info);

   return rc;
...
}

static struct platform_driver msm_camera_driver = {
   .probe = __ov7692_probe,
   .driver = {
         .name = "msm_camera_ov7692",
         .owner = THIS_MODULE,
   },
};

static int __init ov7692_init(void)
{
return platform_driver_register(&msm_camera_driver);
}
```

2. When the basic structure of the camera driver functions is correctly mapped, configure each mode of the camera sensor. To do this, use the xxx_set_sensor_mode function that operates in the following modes. Write the correct dimension values at the register address with the I2C write operation:
   - SENSOR_PREVIEW_MODE
   - SENSOR_SNAPSHOT_MODE
   - SENSOR_RAW_SNAPSHOT_MODE

   **Example for ov7692.c**

```
static int32_t ov7692_set_sensor_mode(int mode, int res)
{
   int32_t rc = 0;
   switch (mode) {
   case SENSOR_PREVIEW_MODE:
         rc = ov7692_video_config(mode);
         break;
   case SENSOR_SNAPSHOT_MODE:
   case SENSOR_RAW_SNAPSHOT_MODE:
         break;
   default:
         rc = -EINVAL;
         break;
   }
   return rc;
}
```

3. For MIPI camera sensor, you should configure the Qualcomm MIPI CSI controller. For more specific information of each parameter, see Chapter 3.

   **Example**

```
static int32_t ov7692_video_config(int mode)
{
...
   rt = RES_PREVIEW;
   if (ov7692_sensor_setting(UPDATE_PERIODIC, rt) < 0)
      return rc;
...
}
static int32_t ov7692_sensor_setting(int update_type, int rt)
{
   struct msm_camera_csi_params ov7692_csi_params;
   switch (update_type) {
   ...
   ov7692_csi_params.lane_cnt = 1;
```

```
    ov7692_csi_params.data_format = CSI_8BIT;
    ov7692_csi_params.lane_assign = 0xe4;
    ov7692_csi_params.dpcm_scheme = 0;
    ov7692_csi_params.settle_cnt = 0x14;

    rc = msm_camio_csi_config(&ov7692_csi_params);
    msleep(10);
    ...
    };
}
```

4. Add your sensor driver inforation in Kconfig and makefile files in the following MSM folders:
   - (Root)\Linux\android\kernel\drivers\media\video\msm\Kconfig
   - (Root)\Linux\android\kernel\drivers\media\video\msm\Makefile

   In the Kconfig file, add information similar to the following.

### Example

```
config IMX074
   bool "Sensor IMX074 (BAYER 13.5M)"
   depends on MSM_CAMERA && ARCH_MSM8X60 && !MSM_CAMERA_V4L2
   default y
   ---help---
   SONY 13.5 MP Bayer Sensor

config WEBCAM_OV7692
   bool "Sensor OV7692 (VGA YUV)"
   depends on MSM_CAMERA && ARCH_MSM8X60 && !MSM_CAMERA_V4L2
   default y
   ---help---
     Omni Vision VGA YUV Sensor.
#    This Senosr is used as a webcam.
#    This uses the CSI interface.
#    Need to enable CSI1 clks for this sensor.
```

In the makefile file, add information similar to the following example.

### Example

```
obj-$(CONFIG_IMX074) += imx074.o imx074_reg.o // For primary camera
obj-$(CONFIG_WEBCAM_OV7692) += ov7692.o // For secondary camera
```

## 2.2 User space porting

For user space porting:

1. Locate the following file to configure camera sensor parameters:
   - (Root)\LINUX\android\vendor\qcom\proprietary\mm-camera\targets\tgtcommon \sensor\sensor.c
2. Register the start function of your sensor in the sensors[] array in sensor.c to start your camera sensor.

   **Example**

   ```
   static sensor_proc_start_t sensors[] = {
   ...
     SENSORS_PROCCESS_START(imx074),
     SENSORS_PROCCESS_START(ov7692),
   ...
   };
   ```

3. Change the parameters related to the camera sensor information, i.e., sensor type, sensor output format, data format of the sensor output, output size, etc., in the following directory. If the parameters are not changed before starting the camera, the camera will not work properly.
   - (Root)\LINUX\android\vendor\qcom\proprietary\mm-camera\targets\tgtcommon\ sensor\ov7692\ov7692_u.c
4. In the ov7692_process_start function, the sensor is configured as a CMOS imager YUV sensor with a YUV422 format. The sensor output size information is also configured for VFE in the same function. The full-size width and height (full_size_width/full_size_height) are for Snapshot mode, and the quarter-size width and height (qtr_size_width/qtr_size_height) are for Preview (or Video) mode. In the case of a small-resolution sensor like VGA, there is no need to use qtr_size for Preview mode.

   **Example**

   ```
   // In case of VGA sensor, we use full size for preview and snapshot
   uint32_t OV7692_FULL_SIZE_DUMMY_PIXELS   = 0;
   uint32_t OV7692_FULL_SIZE_DUMMY_LINES    = 0;
   uint32_t OV7692_FULL_SIZE_WIDTH          = 640;
   uint32_t OV7692_FULL_SIZE_HEIGHT         = 480;

   uint32_t OV7692_QTR_SIZE_DUMMY_PIXELS    = 0;
   uint32_t OV7692_QTR_SIZE_DUMMY_LINES     = 0;
   uint32_t  OV7692_QTR_SIZE_WIDTH          = 640;
   uint32_t OV7692_QTR_SIZE_HEIGHT          = 480;

   int8_t ov7692_process_start(void *ctrl)
   {
      sensor->sensor.prev_res = SENSOR_FULL_SIZE;
      sensor->sensor.pict_res = SENSOR_FULL_SIZE;
   ```

```
    ...
    // Configuration of the sensor
    /* ------------------  Sensor-specific Config -------------- */
    /* CCD or CMOS */
    sensor->sensor.sensor_type = SENSOR_CMOS;

    /* BAYER or YCbCr */
    sensor->sensor.output_format = SENSOR_YCBCR;

    /* Sensor output data format */
      sensor->sensor.format = CAMIF_YCBCR_CB_Y_CR_Y;
    ...
    /* VFE's perception of Sensor output capability */
    sensor->sensor.full_size_width = OV7692_FULL_SIZE_WIDTH;
    sensor->sensor.full_size_height = OV7692_FULL_SIZE_HEIGHT;

    sensor->sensor.qtr_size_width = OV7692_QTR_SIZE_WIDTH;
    sensor->sensor.qtr_size_height = OV7692_QTR_SIZE_HEIGHT;
    ...
    /* ****************************************************
       for primary MIPI camera -  SENSOR_MIPI_CSI
       for secondary MIPI camera - SENSOR_MIPI_CSI_1
       for parallel camera - SENSOR_PARALLEL
    *****************************************************/
    sensor->sensor.connection_mode = SENSOR_MIPI_CSI_1;

    // for YUV422 : SENSOR_8_BIT_DIRECT
    // for Bayer  : SENSOR_8_BIT_DIRECT/SENSOR_10_BIT_DIRECT
    sensor->sensor.raw_output_option = SENSOR_8_BIT_DIRECT;
    ...
    /* Register function table: */
    ov7692_register(&(sensor->fn_table));

    /* Setup camctrl_tbl */
    ov7692_setup_camctrl_tbl(&(sensor->camctrl_tbl));

    sensor_post_init(sensor);
    return TRUE;
}
```

5. In the beginning of the xxx_video_config function (for Preview mode), a sensor IO configuration IOCTL system call is made, which invokes the kernel space xxx_set_sensor_mode function. During the IOCTL call, a number of Preview mode register values are written via I2C communication to the camera sensor. The following example executes the SENSOR_PREVIEW_MODE section of the ov7962_set_sensor_mode function. Likewise, the ov7962_snapshot_config function (for Snapshot mode) and the ov7962_raw_snapshot_config function (for Raw Snapshot mode) make the IOCTL system call from the user space.

### Example

```
static int8_t ov7692_video_config(void *sctrl)
{
   ...
   cfg.cfgtype = CFG_SET_MODE;
   cfg.mode = SENSOR_PREVIEW_MODE;
   ...
   if (ioctl(ctrl->sfd, MSM_CAM_IOCTL_SENSOR_IO_CFG, &cfg) < 0) {
     CDBG("ov7692 failed %d\n", __LINE__);
     return FALSE;
   }
   ...
}
```

6. Configure the following CAMIF settings in the xxx_video_config function (for Preview mode) and in the xxx_snapshot_config function (for Snapshot mode):
   - Lines per frame
   - Pixels per line
   - First pixel position for both window width/height
   - Last pixel position for both window width/height

**NOTE:** The values written in the camera registers in the kernel space xxx_set_sensor_mode function *must* match the four VFE_CAMIF configuration values in the xxx_video_config function. If using a YUV sensor, configure the pixels per line value to be twice the width.

### Example

```
static int8_t ov7692_snapshot_config(void *sctrl)
{
  ...
  ctrl->sensor.sensor_width = ctrl->sensor.full_size_width;
  ctrl->sensor.sensor_height = ctrl->sensor.full_size_height;

  /* CAMIF frame */
  ctrl->sensor.camif_frame_config.pixelsPerLine =
    OV7692_FULL_SIZE_WIDTH * 2;
```

```
    ctrl->sensor.camif_frame_config.linesPerFrame =
      OV7692_FULL_SIZE_HEIGHT;

    /* CAMIF window */
    ctrl->sensor.camif_window_width_config.firstPixel =
      OV7692_FULL_SIZE_DUMMY_PIXELS;
    ctrl->sensor.camif_window_width_config.lastPixel =
      ctrl->sensor.camif_window_width_config.firstPixel +
      (ctrl->sensor.sensor_width * 2) - 1;
    ctrl->sensor.camif_window_height_config.firstLine =
      OV7692_FULL_SIZE_DUMMY_LINES;
    ctrl->sensor.camif_window_height_config.lastLine =
    ctrl->sensor.camif_window_height_config.firstLine +
    ctrl->sensor.sensor_height - 1;
    ...
  }
```

# 3 MIPI CSI Configuration

## 3.1 Meaning of parameters

To bring up your MIPI camera sensor, you should know the meaning of each parameter of msm_camera_csi_params before configuring the Qualcomm MIPI CSI controller.

```
struct msm_camera_csi_params {
 enum msm_camera_csi_data_format data_format;
 uint8_t lane_cnt;
 uint8_t lane_assign;
 uint8_t settle_cnt;
 uint8_t dpcm_scheme;
};
```

- data_format – This dictates the depth of the pixels being sent by the sensor. In general, sensors send 8-, 10-, or 12-bit pixels. You must check the sensor datasheet to determine its output pixel width.
- lane_cnt – CSI-2 sensors send data over “lanes” that are composed of two physical wires. Every CSI-2 sensor has a clock lane and 1~4 data lanes. You must check the sensor datasheet in order to determine the number of supported data lanes.
- lane_assign (alternatively known as lane swapping) – Sometimes you may inadvertently swap data lanes 0 and 1. MSM ASICs are capable of fixing this swapping for data lanes. If the clock lane is swapped with a data lane, you must rework your board. If the sensor is connected properly, no lane swapping is required and you can keep lane_assign == 0xE4.
- settle_cnt – This acts like a delay before receiving high-speed data from the host. Therefore, more packets can be lost if a longer settle count is set. There is no need to change this value unless you change the DDR clock.
- dpcm_scheme – Differential Pulse Code Modulation (DPCM) is a compression scheme used to increase throughput. DPCM schemes are denoted in the format DPCM X-Y-X, where X is the original depth of each pixel and Y is the compressed size of the pixel. A commonly used DPCM scheme is DPCM 10-8-10. In this scheme, the sensor generates 10-bit pixels but compresses them to 8 bits when transmitting them to the MSM ASIC. When inside, the ASIC then decompresses the pixels back to 10 bits. Note that DPCM compression is lossy and the decompressed pixels will not be exactly the same as the original pixels. DPCM compression and decompression are described in [R1]. As with lane count and data format, you must check the sensor datasheet to determine if DPCM compression is supported.

## 3.2 MIPI CSI configuration sequence

Sometimes the MIPI IRQ does not come after receiving the SW_RST_DONE message, which causes a cam_frame timeout error. In some cases, this issue is caused by wrong sequence between sensor start streaming and CSI configuration. In the normal case of starting transmission, the clock lanes should change the state LP11→LP01→LP00→SoT (Start of Transmission) to inform CSI of the SoT state. Therefore, during the CSI configuration period, the clock lanes should remain at the LP11 state and the sensor shoud stop streaming. After a delay for CSI configuration, the sensor can start to stream and enter the SoT state. The following sample code shows the correct sequence.

```
 /* Stop streaming */
 rc = imx074_i2c_write_b_sensor(REG_MODE_SELECT,
 MODE_SELECT_STANDBY_MODE);
 msleep(imx074_delay_msecs_stdby);
 /* CSI configuration */
 if (config_csi == 0) {
 imx074_csi_params.lane_cnt = 4;
              imx074_csi_params.data_format = CSI_10BIT;
              imx074_csi_params.lane_assign = 0xe4;
              imx074_csi_params.dpcm_scheme = 0;
              imx074_csi_params.settle_cnt = 0x14;
              rc = msm_camio_csi_config(&imx074_csi_params);
              /* Delay_msecs_stdby */
              msleep(imx074_delay_msecs_stream);
              config_csi = 1;
}
/* Sensor start streaming */
rc = imx074_i2c_write_w_table(&mode_tbl[0],
 ARRAY_SIZE(mode_tbl));
```

You can read MIPI status from the MIPI_INTERRUPT_STATUS register and it will be printed whenever the msm_io_csi_irq() function is called. In a normal camera launch case, some bits of the register will be triggered with the sequence of [b21]→[bit4]→[b22]. [b21] is the reset ack for the software reset (SW_RST_DONE) during CSI configuration. Once the camera sensor changes the status of SoT, [b4] will be triggered and that means clock is started(CLK_START). Followed by the clock start IRQ is the SHORT_PACKET_CAPTURE_DONE message which indicates that short packet matching 3 LSB bits are captured.