NXP Semiconductors
User's Guide

Document identifier: IMX8MPCSPUG
Rev. 1, 03/2021

i.MX 8M Plus Camera Sensor Porting User Guide

Contents

Chapter 1
Overview

This document describes the architecture of the i.MX 8M PLUS Image Signal Processing (ISP sensor driver, API functions, calling process, methods to add new APIs, and how to implement the methods for mounting different sensors.

This document is applicable to BSP release 5.4.70_2.3.0.

Acronyms and Conventions

3A: Auto Exposure, Auto Focus, Auto White Balance

AE: Auto Exposure

AF: Auto Focus

API: Application Programming Interface

AWB: Automatic White Balance

BLC: Black Level Correction

fps: Frames Per Second

I2C: Inter-Integrated Circuit

IOCTL: Input Output Control

ISI: Independent Sensor Interface

ISP: Image Signal Processing

ISS: Image Sensor Specific

VVCAM: Vivante’s kernel driver integration layer

WB: White Balance

Hexadecimal numbers are indicated by the prefix “0x” or suffix “H” —for example, 0x32CF or 32CFH.

Binary numbers are indicated by the prefix “0b” —for example, 0b0011.0010.1100.1111

Code snippets are given in Consolas typeset.

Chapter 2
ISP Software Architecture

In the ISP framework, the application layer and 3A (Auto Exposure, Auto Focus, Auto White Balance) layer calls the sensor API using function pointers in the ISS through the ISI layer code. The data stream which is output by the sensor is sent directly to ISP for processing. In the following figure, the gray arrows represent the function calls and the white arrows represent the direction of the output image data of the sensor.

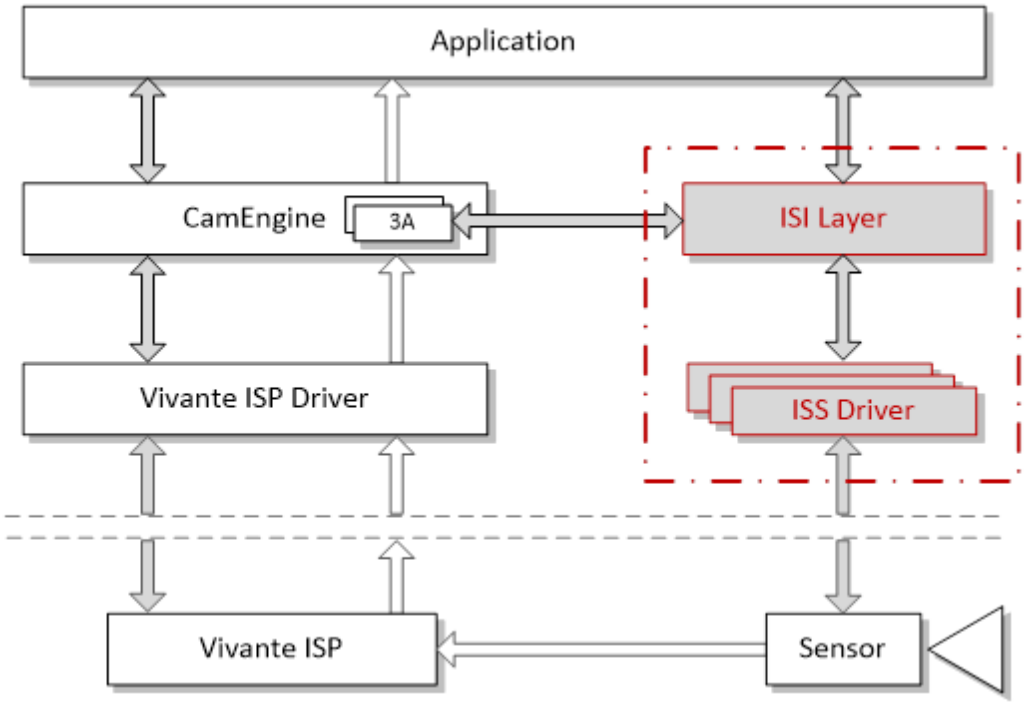


Figure 1. ISP Software Architecture

2.1 ISS (Image Sensor Specific) Driver

- Sensor specific implementation
- Sensor specific attributes and behavior from:
  - Sensor datasheet
  - Calibration data

2.2 ISP Sensor Module Block Diagrams

The i.MX 8M Plus ISP sensor module is organized as shown in the following figures.

1. **Sensor Module in Linux Kernel**: I2C is called in the kernel to read and write the sensor register as shown in Figure 2 below.

- **ISI Layer**: includes the interface to call the corresponding sensor functions, function pointers to mount different sensors and the structure composed of these function pointers.
  - **ISS**: uses function pointers so that the ISP driver code can use different sensors independently without modifying the code of other modules.

- **Sensor API**: includes sensor power on, initialization, reading and writing sensor registers, configuring sensor resolution, exposure parameters, obtaining current sensor configuration parameters and other functions.

- **VVCAM**: i.MX 8M Plus ISP kernel driver integration layer which includes ISP, MIPI, camera sensor and I^2C kernel driver.
  - **Sensor Driver**: performs sensor API operations on sensor hardware.
  - **I^2C:** Read-Write Sensor Register. When writing a register, its value must be a 32-bit value. There is no restriction on reading a register.
  - **Kernel Working Mode:** VVCAM has two types of working modes in the kernel:

1. **V4L2 Mode**: kernel driver that acts as a part of V4L2 kernel driver, register device name and operations as a V4L2 sub-device style. This mode is compatible with the V4L2 sensor device format.

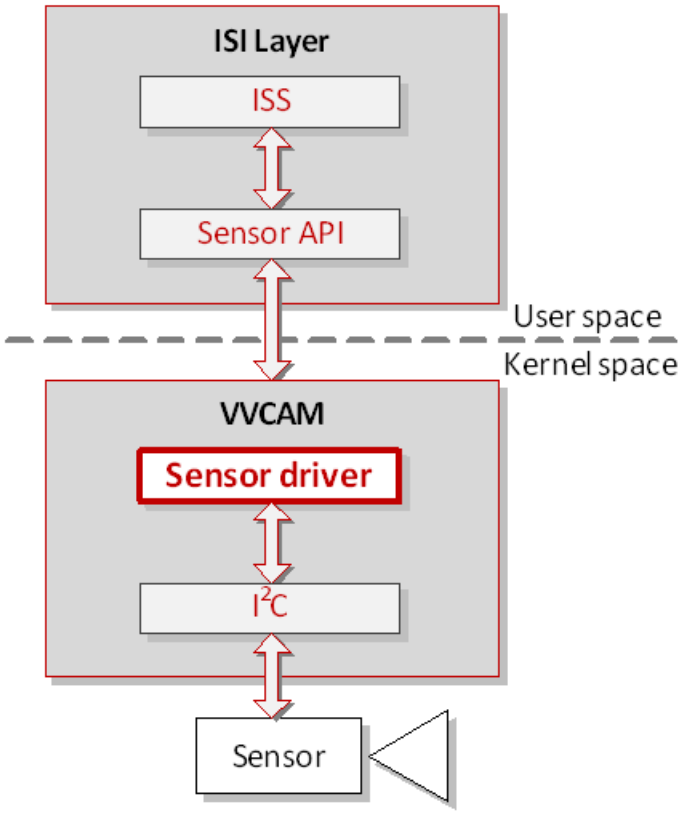


Figure 2. Sensor Module in Linux Kernel

Chapter 3
ISP Independent Sensor Interface (ISI) API reference

Structures and functions are provided here for convenience.

3.1 ISI Structures

3.1.1 IsiCamDrvConfig_s

This structure defines camera sensor driver specific data. Refer to section 2.4.1 of the i.MX 8M Plus ISP ISI API document for details.

3.1.2 IsiSensorInstanceConfig_s

This structure defines the configuration structure used to create a new sensor instance. Refer to section 2.4.8 of the i.MX 8M Plus ISP ISI API document for details.

3.1.3 IsiSensor_s

This structure defines attributes for the sensor. Refer to section 2.4.5 of the i.MX 8M Plus ISP ISI API document for details.

3.2 ISI Functions

The following ISI API will use the function pointers defined in the IsiSensor_s data structure to call the corresponding sensor functions defined in the Sensor API Reference section.

Refer to section 3 of the i.MX 8M Plus ISP ISI API document for details.

ISI API	Function Description
IsiCreateSensorIss(…)	Create a new sensor instance and assign resources to sensor
IsiInitSensor(…)	Initialization of sensor
IsiGetSensorModeIss (…)	Get the sensor mode information
IsiReleaseSensorIss(…)	Release the sensor's resources
IsiGetCapsIss(…)	Get the capabilities of sensor
IsiSetupSensorIss(…)	Launch sensor
IsiChangeSensorResolutionIss(…)	Change image sensor resolution while keeping all other static settings
IsiSensorSetStreamingIss(…)	Enables/disables streaming of sensor data
IsiSensorSetPowerIss(…)	Power-up/power-down the sensor
IsiCheckSensorConnectionIss(…)	Checks the connection to the camera sensor
IsiGetSensorRevisionIss(…)	Read sensor ID

Table continues on the next page...

Table continued from the previous page...

ISI API	Function Description
IsiRegisterReadIss(…)	Read sensor register
IsiRegisterWriteIss(…)	Write a value to the sensor register
IsiGetGainLimitsIss(…)	Get the minimum and maximum value of gain
IsiGetIntegrationTimeLimitsIss(…)	Get the minimum and maximum exposure time
IsiExposureControlIss(…)	Exposure control
IsiGetCurrentExposureIss(…)	Get current gain and exposure time
IsiGetGainIss(…)	Get the gain value of the current sensor
IsiGetVSGainIss(…)	Get gain of the very short exposure frame in HDR mode
IsiGetLongGainIss(…)	Get gain of the long exposure frame in HDR mode
IsiGetGainIncrementIss(…)	Get step size of gain
IsiSetGainIss(…)	Set sensor gain
IsiGetIntegrationTimeIss(…)	Get current exposure time
IsiGetVSIntegrationTimeIss(…)	Get exposure time of the very short exposure frame in HDR mode
IsiGetLongIntegrationTimeIss	Get exposure time of the long exposure frame in HDR mode
IsiGetIntegrationTimeIncrementIss(…)	Get the maximum exposure time of a row
IsiSetIntegrationTimeIss(…)	Set exposure time
IsiQuerySensorIss(…)	Query sensor support mode information
IsiGetSensorFpsIss(…)	Get current framerate
IsiSetSensorFpsIss(…)	Set the new framerate to sensor
IsiGetResolutionIss(…)	Get the resolution of the sensor
IsiMdiInitMotoDrive(…)	Initialization of moto control interface
IsiMdiSetupMotoDrive(…)	Setup the mote control step parameter
IsiMdiFocusSet(…)	Set the moto step value
IsiMdiFocusGet(…)	Get the moto step value
IsiMdiFocusCalibrate(…)	Handle of AF calibration

Table continues on the next page...

Table continued from the previous page...

ISI API	Function Description
IsiGetSensorMipiInfoIss(…)	Get the Mipi information of sensor configuration
IsiActivateTestPattern(…)	Sensor TPG interface
IsiEnableHdr(…)	Enable/disable sensor HDR function
IsiResetSensorIss(…)	Reserved
IsiSetBayerPattern(…)	Bayer Pattern interface
IsiDumpAllRegisters(…)	Dump all the sensor registers to file
IsiTryToSetConfigFromPreferredCap(…)	Reserved
IsiGetSensorAWBMode(…)	Get AWB mode by sensor or ISP
IsiSensorSetBlcIss(…)	Set sensor BLC
IsiSensorSetWBIss(…)	Set sensor WB gain.
IsiSensorGetExpandCurveIss(…)	Get the curve of the sensor extended bit width
IsiQuerySensorSupportIss(…)	Get the current sensor information

3.3 Sensor API Reference

This section describes the API defined in units/isi/drv/***<sensor>***/source/***<sensor>*.c** where *<sensor>* is the name of the sensor (for example, OV2775). You can refer to the APIs in the following table to define your own API for the sensor which you are using. The upper application layer can use the structure of IsiCamDrvConfig_t to call the following functions.

Table 1. Sensor API Reference

Sensor API	Description
SENSOR DEFINES	
<sensor>_SLAVE_ADDR	I2C is used when reading and writing the register of sensor
<sensor>_MIN_GAIN_STEP	When AE decomposes the exposure, it is the smallest unit of gain
<sensor>_MAX_GAIN_AEC	AE will be used when decomposing exposure
<sensor>_VS_MAX_INTEGRATION_TIME	The maximum exposure time for the very short exposure frame in HDR mode. The maximum exposure time of ISP is 48 lines (the exposure time of lines x 48 is: <sensor>_VS_MAX_INTEGRATION_TIME)
<sensor>_VTS_NUM	The VTS of the sensor needs to be modified according to the configuration of sensor, which affects the exposure

Table continues on the next page...

Table 1. Sensor API Reference (continued)

Sensor API	Description
<sensor>_HTS_NUM	The HTS of the sensor needs to be modified according to the configuration of sensor, which affects the exposure time
<sensor>_PIX_CLOCK	The pixel clock of the sensor needs to be modified according to the sensor's configuration, which affects the sensor's exposure
SENSOR STRUCTURES	
IsiCamDrvConfig_t IsiCamDrvConfig{}	Provide a structure for upper layer to access function pointer
SENSOR FUNCTIONS	
<sensor>_IsiGetSensorIss(…)	Mount the sensor API under the ISI function pointer
<sensor>_IsiCreateSensorIss(…)	Assign resources to sensor
<sensor>_IsiInitSensorIss(…)	Initialization of sensor
<sensor>_IsiReleaseSensorIss(…)	Release the sensor's resources
<sensor>_IsiResetSensorIss(…)	Reset sensor
<sensor>_IsiGetCapsIss(…)	Get the capabilities of sensor
<sensor>_IsiSetupSensorIss(…)	Launch sensor
<sensor>_IsiChangeSensorResolutionIss(…)	Change image sensor resolution while keeping all other static settings
<sensor>_IsiSensorSetStreamingIss(…)	Enables/disables streaming of sensor data
<sensor>_IsiSensorSetPowerIss(…)	Power-up/power-down the sensor
<sensor>_IsiCheckSensorConnectionIss(…)	Check the connection to the camera sensor
<sensor>_IsiGetSensorRevisionIss(…)	Read sensor ID
<sensor>_IsiActivateTestPattern(…)	Sensor TPG interface
<sensor>_IsiRegisterReadIss(…)	Read sensor register
<sensor>_IsiRegisterWriteIss(…)	Write sensor register
<sensor>_IsiGetGainLimitsIss(…)	Get the minimum and maximum value of gain
<sensor>_IsiGetIntegrationTimeLimitsIss(…)	Get the minimum and maximum exposure time
<sensor>_IsiExposureControlIss(…)	Exposure control
<sensor>_IsiGetCurrentExposureIss(…)	Get current gain and exposure time
<sensor>_IsiGetGainIss(…)	Get the gain value of the current sensor

Table continues on the next page...

Table 1. Sensor API Reference (continued)

Sensor API	Description
<sensor>_IsiGetVSGainIss(…)	Get gain of the very short exposure frame in HDR mode
<sensor>_IsiGetLongGainIss(…)	Get gain of the long exposure frame in HDR mode
<sensor>_IsiGetGainIncrementIss(…)	Step size of gain
<sensor>_IsiSetGainIss(…)	Set sensor gain
<sensor>_IsiEnableHdr(…)	Enable/disable sensor HDR function
<sensor>_IsiSetBayerPattern(…)	Bayer Pattern interface
<sensor>_IsiGetIntegrationTimeIss(…)	Get current exposure time
<sensor>_IsiGetVSIntegrationTimeIss(…)	Get exposure time of the very short exposure frame in HDR mode
<sensor>_IsiGetLongIntegrationTimeIss(…)	Get exposure time of the long exposure frame in HDR mode
<sensor>_IsiGetIntegrationTimeIncrementIss(…)	Get the maximum exposure time of a row
<sensor>_IsiSetIntegrationTimeIss(…)	Set exposure time
<sensor>_IsiQuerySensorIss(…)	Query sensor supports
<sensor>_IsiGetResolutionIss(…)	Get the resolution of the sensor
<sensor>_IsiGetSensorFpsIss(…)	Get current frame rate
<sensor>_IsiSetSensorFpsIss(…)	Set the new frame rate to sensor
<sensor>_IsiGetSensorModeIss(…)	Get sensor mode information
<sensor>_pIsiGetSensorAWBModeIss(…)	Get sensor AWB mode
<sensor>_pIsiSensorSetBlcIss(…)	Set sensor sub BLC
<sensor>_IsiSensorSetWBIss(…)	Set sensor WB gain
<sensor>_IsiSensorGetExpandCurveIss(…)	Get sensor expand curve

3.4 ISS Sensor Driver User Space Flow

Function Pointers

In the ISS (Image Sensor Specific) driver, we define function pointers of the same type as the sensor API and integrate these function pointers into the IsiSensor_s data structure. The driver then integrates the IsiSensor_s structure, camera driver ID and IsiGetSensorIss_t function pointers into the IsiCamDrvConfig_s data structure. In the function corresponding to the IsiGetSensorIss_t function pointer, the driver mounts the sensor API to the function pointer defined in the ISS layer. The application layer can operate the sensor API by accessing this data structure. Refer to the Define the Camera Driver Configuration Data Structure in ISS driver section for additional information.

Sensor Defines

There are #defines for the sensor which are unique to each sensor. These #defines need to be set according to the requirements of the application. An example of a custom set of #defines for a sensor is given here in the Define the Camera Driver Configuration Data Structure in ISS driver section.

Sensor Exposure Function

The exposure function in the sensor is also different for each sensor. To modify the exposure function, refer to the sensor's data sheet for specific implementation methods. An example of a customized exposure function is given here in the Modify the Sensor Driver in V4L2 Mode section. The IsiGetSensorIss_t function pointer interface defined in ISI corresponds to the sensor API. Each ISI API calls the corresponding sensor API through the function pointer.

The application layer obtains the address of the function pointer with the IsiCamDrvConfig_t data structure through the SensorOps::driverChange() function.

```
SensorOps::driverChange(std::string driverFileName, std::string calibFileName) {
…..
DCT_ASSERT(!pCamDrvConfig->pfIsiGetSensorIss(&pCamDrvConfig->IsiSensor));
pSensor = &pCamDrvConfig->IsiSensor;
```

At the same time, the application layer will pass this address down to ISS so that ISS can access different sensors.

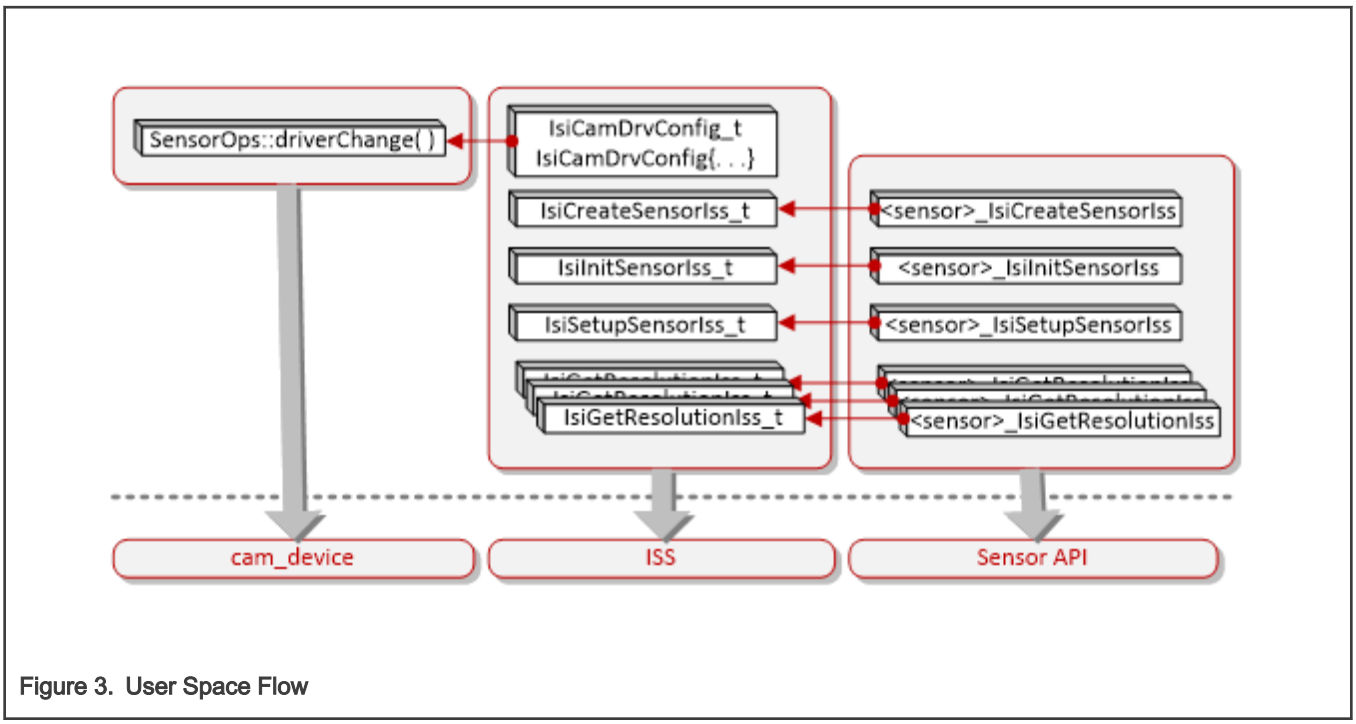


Figure 3. User Space Flow

Chapter 4
IOCTL Introduction

The interface in the user space cannot operate the functions directly in the kernel space. Commands and parameters of the operations are called with the use of IOCTL commands.

4.1 IOCTL Commands

The corresponding operations for IOCTL commands in the kernel space are shown in the following table.

Table 2. IOCTL Commands (V4L2 Mode)

IOCTL	IOCTL Operation
VVSENSORIOC_WRITE_REG	Call <sensor>_write_reg to write the sensor register
VVSENSORIOC_READ_REG	Call <sensor>_read_reg to read the sensor register
VVSENSORIOC_S_STREAM	Call <sensor>_s_stream to set sensor stream start or stop
VVSENSORIOC_S_LONG_EXP	Call <sensor>_s_long_exp to set long exposure frame exposure
VVSENSORIOC_S_EXP	Call <sensor>_s_exp to set exposure frame exposure
VVSENSORIOC_S_VSEXP	Call <sensor>_s_vsexp to set very short exposure frame exposure
VVSENSORIOC_S_LONG_GAIN	Call <sensor>_s_long_gain to set long exposure frame gain
VVSENSORIOC_S_GAIN	Call <sensor>_s_gain to set exposure frame gain
VVSENSORIOC_S_VSGAIN	Call <sensor>_s_vsgain to set very short exposure frame gain
VVSENSORIOC_S_FPS	Call <sensor>_s_fps to set the sensor fps
VVSENSORIOC_G_FPS	Call <sensor>_g_fps to get the sensor fps
VVSENSORIOC_S_CLK	Call <sensor>_s_clk to set the sensor clk
VVSENSORIOC_G_CLK	Call <sensor>_g_clk to get the sensor clk
VIDIOC_QUERYCAP	Call <sensor>_ioc_qcap
VVSENSORIOC_G_CHIP_ID	Call <sensor>_g_chipid to get the sensor chip id
VVSENSORIOC_G_RESERVE_ID	Return the sensor correct ID
VVSENSORIOC_S_HDR_MODE	Call <sensor>_s_hdr to set the sensor HDR mode
VVSENSORIOC_QUERY	Call <sensor>_ioc_query_mode to get all modes of the sensor
VVSENSORIOC_G_SENSOR_MODE	Call <sensor>_g_mode to get the sensor current mode

Table continues on the next page...

Table 2. IOCTL Commands (V4L2 Mode) (continued)

IOCTL	IOCTL Operation
VVSENSORIOC_S_WB	Call <sensor>_s_wb to set the sensor white balance register value
VVSENSORIOC_S_BLC	Call <sensor>_s_blc to set the sensor BLC register value
VVSENSORIOC_G_EXPAND_CURVE	Call <sensor>_get_expand_curve to get the sensor expand curve

4.2 IOCTL Call Flow

The IOCTL supports V4L2 Mode as described below.

4.2.1 V4L2 Mode

The figure below shows the IOCTL call flow in V4L2 mode. For more details, refer to the VVCAM Flow in V4L2 Mode section.

Figure 4. IOCTL Call Flow in V4L2 Mode

Chapter 5
VVCam API Reference

This section describes the API declared in **vvcam/common/vvsensor.h**.

5.1 Sensor Driver Enumerations

5.1.1 SENSOR_BAYER_PATTERN_E

enum Members	Description
BAYER_RGGB	Bayer RGGB pattern mode
BAYER_GRBG	Bayer GRBG pattern mode
BAYER_GBRG	Bayer GBRB pattern mode
BAYER_BGGR	Bayer BGGR pattern mode

5.1.2 sensor_hdr_mode_e

enum Members	Description
SENSOR_MODE_LINEAR	Linear mode
SENSOR_MODE_HDR_STITCH	ISP HDR mode
SENSOR_MODE_HDR_NATIVE	The different exposure image will be combined in sensor before being processed by ISP.

5.1.3 sensor_stitching_mode_e

enum Members	Description
SENSOR_STITCHING_DUAL_DCG	Dual DCG mode 3x12-bit
SENSOR_STITCHING_3DOL	3 DOL frame 3x12-bit
SENSOR_STITCHING_LINEBYLINE	3x12-bit line by line without waiting
SENSOR_STITCHING_16BIT_COMPRESS	16-bit compressed data + 12-bit RAW
SENSOR_STITCHING_DUAL_DCG_NOWAIT	2x12-bit dual DCG without waiting
SENSOR_STITCHING_2DOL	DOL2 frame or 1 CG+VS sx12-bit RAW
SENSOR_STITCHING_L_AND_S	L+S 2x12-bit RAW

5.2 Sensor Driver Structures

5.2.1 sensor_blc_t

Structure Members	Type	Description
red	uint32_t	Red Black Level Correction (BLC) level
gr	uint32_t	Gr BLC level
gb	uint32_t	Gb BLC level
blue	uint32_t	Blue BLC level

5.2.2 sensor_data_compress_t

Structure Members	Type	Description
enable	uint32_t	0: sensor data is not compressed 1: sensor data is compressed
x_bit	uint32_t	If sensor data is compressed, x_bit represents the data bit width before compression.
y_bit	uint32_t	If sensor data is compressed, y_bit represents the data bit width after compression.

5.2.3 sensor_expand_curve_t

Structure Members	Type	Description
x_bit	uint32_t	Input bit width of data decompression curve
y_bit	uint32_t	Output bit width of data decompression curve
expand_px[64]	uint8_t	Data decompression curve input interval index.exp: 1<<expand_px[i] = expand_x_data[i+1] - expand_x_data[i]
expand_x_data[65]	uint32_t	65 points of data decompression curve input
expand_y_data[65]	uint32_t	65 points of data decompression curve output

5.2.4 sensor_mipi_info

Structure Members	Type	Description
mipi_lane	uint32_t	MIPI lane
sensor_data_bit	uint32_t	Sensor data bit

5.2.5 sensor_white_balance_t

Structure Members	Type	Description
r_gain	uint32_t	White Balance (WB) R gain
gr_gain	uint32_t	WB Gr gain
gb_gain	uint32_t	WB Gb gain
b_gain	uint32_t	WB B gain

5.2.6 vvcam_ae_info_t

Structure Members	Type	Description
DefaultFrameLengthLines	uint32_t	Sensor default Frame length lines (always is sensor default mode vts)
CurFrameLengthLines	uint32_t	Current Frame length lines
one_line_exp_time_ns	uint32_t	One line exposure time (in ns) (always = sensor PCLK * HTS)
max_interrgation_time	uint32_t	Maximum exposure line (Maximum gain multiple *gain_accuracy. Fixed point processing of floating-point numbers.)
min_interrgation_time	uint32_t	Minimum exposure line (Mininum gain multiple *gain_accuracy. Fixed point processing of floating-point numbers.)
interrgation_accuracy	uint32_t	Exposure accuracy, always is one line
max_gain	uint32_t	Maximum sensor gain
min_gain	uint32_t	Minimum sensor gain
gain_accuracy	uint32_t	Gain accuracy (Fixed point precision of floating-point numbers.)

Table continues on the next page...

Table continued from the previous page...

Structure Members	Type	Description
cur_fps	uint32_t	Current frame rate
hdr_radio	uint32_t	HDR radio

5.2.7 vvcam_mode_info_array_t

This structure is an abstraction of vvcam_mode_info.

Structure Members	Type	Description
count	uint32_t	Number of modes supported
modes[VVCAM_SUPPORT_MAX_MODE_COUNT]	struct vvcam_mode_info	Structure of sensor feature

5.2.8 vvcam_mode_info_t

Structure Members	Type	Description
index	uint32_t	Mode index
width	uint32_t	Image width
height	uint32_t	Image height
fps	uint32_t	frame rate
hdr_mode	uint32_t	HDR mode
stitching_mode	uint32_t	HDR stitching mode
bit_width	uint32_t	Sensor bit width
data_compress	sensor_data_compress_t	Sensor data is compressed
bayer_pattern	uint32_t	Bayer mode
ae_info	vvcam_ae_info_t	AE information
preg_data	void *	Sensor register configuration point
reg_data_count	uint32_t	Sensor register configuration size

5.3 Sensor Driver API

V4l2 Sensor Driver API is declared in file <sensor>_mipi_v3.c where <sensor> is the name of the sensor (for example, OV2775).

Table 3. Sensor V4l2 Driver API

API Name	Description
<sensor>_write_reg(…)	Write data to the specified register
<sensor>_read_reg(…)	Read data from the specified register
<sensor>_s_stream(…)	Start or stop the sensor
<sensor>_s_long_exp(…)	Write the exposure time of 3A decomposition exposure parameter for a long exposure frame to the sensor's register
<sensor>_s_exp(…)	Write the exposure time of 3A decomposition exposure parameter to the sensor's register
<sensor>_s_vsexp(…)	Write the exposure time of 3A decomposition exposure parameter for a very short exposure frame to the sensor's register
<sensor>_s_long_gain(…)	Set the gain of the long exposure frame in multiples rather than dB
<sensor>_s_gain(…)	Set the gain in multiples rather than dB
<sensor>_vs_gain(…)	Set the gain of the very short exposure frame in multiples rather than dB
<sensor>_s_fps(…)	Set sensor FPS
<sensor>_g_fps(…)	Get sensor FPS
<sensor>_s_clk(…)	Set sensor clock
<sensor>_g_clk(…)	Get sensor clock
<sensor>_ioc_qcap(…)	V4l2 query driver ability
<sensor>_g_chipid(…)	Get sensor chip ID
<sensor>_s_hdr(…)	Enable or disable sensor HDR combine
<sensor>_ioc_query_mode(…)	Query sensor support mode information
<sensor>_g_mode(…)	Get the sensor mode information according to the index
<sensor>_s_blc(…)	Set sensor sub BLC
<sensor>_s_wb(…)	Set white balance
<sensor>_get_expand_curve(…)	Get sensor expand curve

Chapter 6
Camera Sensor Driver in V4L2 Mode

6.1 VVCAM Flow in V4L2 Mode

Read through this section carefully before porting the new sensor driver in V4L2 Mode. If you have any problems during the sensor porting process, refer to the existing sensor driver of the platform in your source code release.

To add a new function interface, refer to the following sections:

- ISI API Reference
- ISS Sensor Driver User Space Flow
- Sensor API Reference
- VVCAM Flow in V4L2 Mode

Both hub and sensor kernel driver must add corresponding interfaces and calls. While porting the sensor, be aware that different sensors in the sensor data sheet have different conversion methods when converting the exposure parameters which are passed down from the 3A modules to the values written in the registers. The sensor data must be accurately defined.

To port the camera sensor, the following steps must be taken as described in the following sections:

1. Define sensor attributes and create the sensor instance in CamDevice.
2. Define the camera driver configuration data structure in ISS driver.
3. Modify the sensor driver in VVCAM.
4. Setup HDR.
5. Define MIPI lanes.
6. Sensor Driver Configuration in V4L2.

6.1.1 Sensor Driver Software Architecture in V4L2 Mode

The software architecture of the sensor driver in V4L2 Mode is shown in the figure below. The V4L2-subdev driver is defined in file vvcam/v4l2/sensor/<sensor>/<sensor>_xxxx.c where <sensor> is the name of the sensor (for example, OV2775).

A device node of the sensor named v4l-subdevx can be created in /dev for direct access. Function <sensor>_priv_ioctl() is used in the kernel space to receive the commands and parameters passed down by the user space through ioctl() and to call the corresponding functions in <sensor>_xxxx.c according to the commands.

NOTE

Developers should replace the Vivante V4L2-Subdev Driver with their own sensor as shown in the figure below.

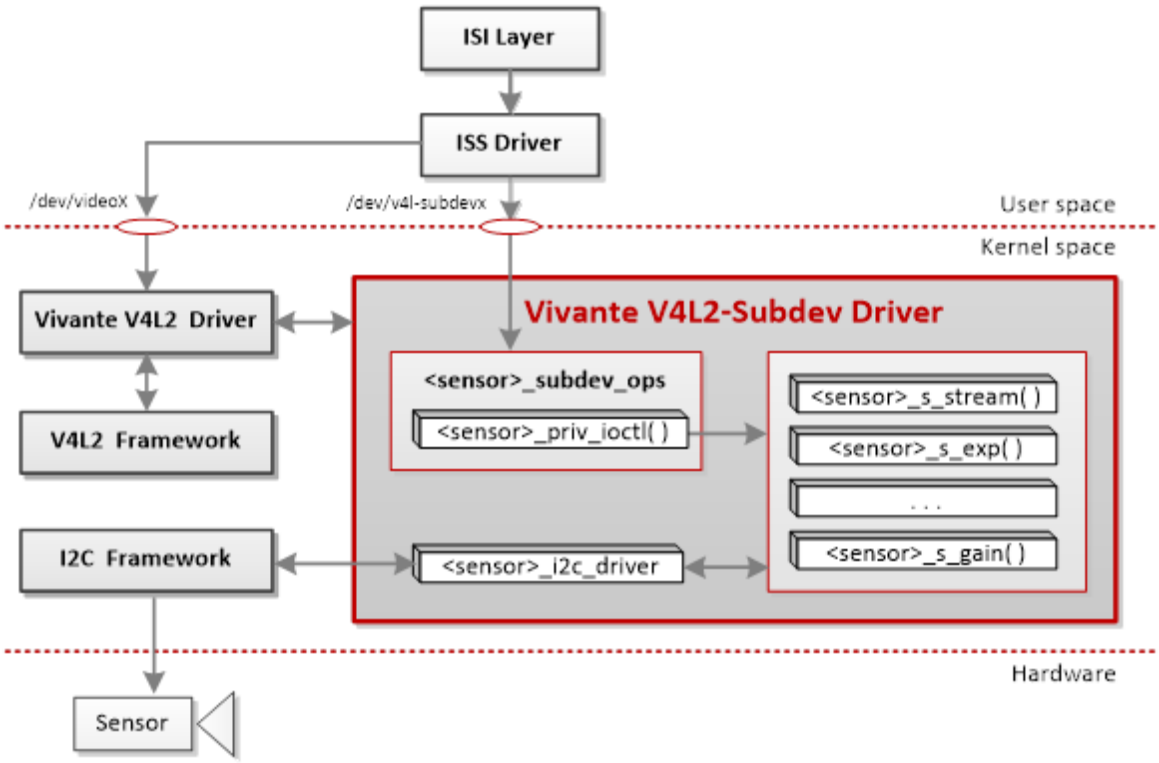


Figure 5. VVCAM Software Architecture in V4L2 Mode

6.2 Camera Sensor Porting Setup in V4L2 Mode

6.2.1 Define Sensor Attributes and Create Sensor Instance in CamDevice

The following three steps are already implemented in CamDevice and are included for reference only. Developers may not modify any code in CamDevice.

step 1) Define the sensor attributes in the IsiSensor_s data structure.

step 2) Define the IsiSensorInstanceConfig_t configuration structure that will be used to create a new sensor instance.

step 3) Call the IsiCreateSensorIss() function to create a new sensor instance.

```
int32_t SensorOps::open() {
…
int32_t ret = RET_SUCCESS;
IsiSensorInstanceConfig_t sensorInstanceConfig;
sensorInstanceConfig.HalHandle = pHalHolder->hHal;
sensorInstanceConfig.pSensor = &pCamDrvConfig->IsiSensor;
ret = IsiCreateSensorIss(&sensorInstanceConfig);
…
}
```

6.2.2 Define the Camera Driver Configuration Data Structure in ISS driver

step 4) Define the IsiCamDrvConfig_s data structure. Data members defined in this data structure include the sensor ID (CameraDriverID) and the function pointer to the IsiSensor data structure. Using the address of the IsiCamDrvConfig_s structure, the driver can then access the sensor API attached to the function pointer.

For example:

```
IsiCamDrvConfig_t IsiCamDrvConfig = {
    0,
    <sensor>_IsiQuerySensorSupportIss,
    <sensor>_IsiGetSensorIss,
    {
     0,            /**< IsiSensor_t.pszName */
     …
    }
};
```

NOTE

- IsiCamDrvConfig is defined in file units/isi/drv/<sensor>/source/<sensor>.c.
- <sensor>_IsiQuerySensorSupportIss() uses the IOCTL command VVSENSORIOC_QUERY to get all the modes supported by <sensor>

<sensor>_IsiGetSensorIss() can initialize the IsiSensor_s data structure. It is called by upper-level application described in the ISS Sensor Driver User Space Flow section. Then the application can get address of all the callback functions. <sensor>_IsiGetSensorIss is defined as follows:

```
RESULT <sensor>_IsiGetSensorIss(IsiSensor_t *pIsiSensor)
{
    …
pIsiSensor->pIsiCreateSensorIss         = <sensor>_IsiCreateSensorIss;
pIsiSensor->pIsiInitSensorIss           = <sensor>_IsiInitSensorIss;
pIsiSensor->pIsiGetSensorModeIss        = <sensor>_IsiGetSensorModeIss;
pIsiSensor->pIsiResetSensorIss          = <sensor>_IsiResetSensorIss;
…
}
```

NOTE

<sensor>_IsiCreateSensorIss, <sensor>_IsiInitSensorIss, <sensor>_IsiGetSensorModeIss, <sensor>_IsiResetSensorIss are described in the Sensor API Reference section.

Sensor macro must be modified to match the sensor attributes in the source file corresponding to the sensor as described below.

An example of a set of sensor defines is given in file units/isi/drv/<sensor>/source/<sensor>.c. See the example below.

```
#define SENSOR_MIN_GAIN_STEP
        (1.0f/16.0f)
```

6.2.3 Modify the Sensor Driver in V4L2 Mode

step 5) The V4L2 architecture of sensor driver is shown in Figure 5. To specify a camera sensor, the sensor driver must be added by developers in file vvcam/v4l2/sensor/<sensor>/<sensor>_xxxx.c where <sensor> is the name of the sensor (for example, OV2775). Developers can refer to the file ov2775_mipi_v3.c to add their own sensors.

In ov2775_mipi_v3.c, there are seven important parts:

1. Define the private data structure of struct ov2775 shown in the following table. This structure includes the key parameters used by ov2775 sensor driver. Developers should modify the structure members according to their own sensor drivers.

ov2775 Structure Members	Type	Description
subdev	struct v4l2_subdev	A V4L2 sub-device struct presents the sensor device
v4l2_dev	struct v4l2_device *	Pointer to struct v4l2_device
i2c_client	struct i2c_client *	Pointer to an i2c slave device. The i2c_client identifies a single device (that is, sensor) connected to an I2C bus
pix	struct v4l2_pix_format	Video image format
fmt	const struct ov2775_datafmt *	struct ov2775_datafmt {u32 code; enum v4l2_colorspace colorspace;};
streamcap	struct v4l2_captureparm	Capture parameters
pads[1]	struct media_pad	A media pad graph object for sensor
on	bool	Sensor streaming on/off
brightness	int	Reserved
hue	int	Reserved
contrast	int	Reserved
saturation	int	Reserved
red	int	Reserved
green	int	Reserved
blue	int	Reserved
ae_mode	int	Reserved
mclk	u32	Reference clock provided to sensor
mclk_source	u8	mclk sources ID
sensor_clk	struct clk *	Pointer to struct clk used to manage the sensor clock
csi	int	ID number of MIPI CSI controller connected to the current sensor
io_init	void (*io_init) (struct ov2775 *)	Function pointer to reset sensor with hardware I/O pin
pwn_gpio	int	GPIO number for powering down sensor

Table continues on the next page...

Table continued from the previous page...

ov2775 Structure Members	Type	Description
rst_gpio	int	GPIO number for resetting sensor
hdr	int	HDR mode
fps	int	Frame rate
cur_mode	vvcam_mode_info_t	Current mode index of sensor
blc	sensor_blc_t	sensor_blc_t is used to store the BLC levels of red, GR, GB, blue
wb	sensor_white_balance_t	sensor_white_balance_t is used to store the white balance gains of red, GR, GB, blue in sensor
lock	struct mutex	Mutex lock to access the sensor driver

2. Include the initialization parameter header files for ov2775.

   For example:

```
#include "ov2775_regs_1080p.h"
#include "ov2775_regs_1080p_hdr.h"
#include "ov2775_regs_1080p_native_hdr.h"
#include "ov2775_regs_720p.h"
```

   Each header file includes an array for register settings. The initial array of registers can be obtained from the sensor vendor.

3. Add the vvcam_mode_info data structure array. The array stores all the supported mode information of ov2775. The ISI layer can get all the modes with the VVSENSORIOC_QUERY command.

   For example:

```
static struct vvcam_mode_info pov2775_mode_info[] = {
{
.index      = 0,
.width      = 1920,
.height     = 1080,
.fps        = 30,
.hdr_mode   = SENSOR_MODE_LINEAR,
.bit_width = 12,
.data_compress.enable = 0,
.bayer_pattern = BAYER_BGGR,
.ae_info = {
.DefaultFrameLengthLines = 0x466,
.one_line_exp_time_ns = 29625,
.max_integration_time = 0x466 - 2,
.min_integration_time = 1,
.gain_accuracy = 1024,
.max_gain = 21 * 1024,
.min_gain = 1 * 1024,
},
.preg_data = ov2775_init_setting_1080p,
```

```
.reg_data_count = ARRAY_SIZE(ov2775_init_setting_1080p),
},
…
};
```

NOTE

ov2775_init_setting_1080p is the register setting array which is defined in header file **ov2775_regs_1080p.h** which is described in Step 5, Part 2 (above).

4. Define the v4l2-subdev ioctl function of **ov2775_priv_ioctl**. Function ov2775_priv_ioctl() is used to receive the commands and parameters passed down by the user space through ioctl() and to control the ov2775 sensor.

   For example:

```
long ov2775_priv_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg_user) {
…
switch (cmd) {
…
case VVSENSORIOC_S_LONG_GAIN:{
USER_TO_KERNEL(__u32);
ret = ov2775_s_long_gain(sensor, *(__u32 *)arg);
break;
}
case VVSENSORIOC_S_GAIN: {
USER_TO_KERNEL(__u32);
ret = ov2775_s_gain(sensor, *(__u32 *)arg);
break;
}
case VVSENSORIOC_S_VSGAIN: {
USER_TO_KERNEL(__u32);
ret = ov2775_s_vsgain(sensor, *(__u32 *)arg);
break;
}
…
default:
pr_err("unsupported ov2775 command %d.", cmd);
ret = -1;
break;
} /*end of switch*/
…
}
```

NOTE

cmd is an IOCTL command described in the IOCTL Commands section. Developers should implement each IOCTL function corresponding to their own sensors. These IOCTL functions include ov2775_s_gain(), ov2775_s_vsgain(), ov2775_s_stream(), ov2775_s_fps(), and so on.

5. Since the exposure function in the sensor is unique for each sensor, a customized calculation of exposure parameters must be written. The parameters include gain and integration time. Refer to the data sheet of the sensor for specific implementation values.

   Here is an example of a customized calculation for the gain of sensor OV2775 in linear mode. The function is in the file vvcam/v4l2/sensor/ov2775/ov2775_mipi_v3.c.

```
int ov2775_s_gain(struct ov2775 *sensor, __u32 new_gain)
{
…
sensor_calc_gain(new_gain, &again, &dgain, &hcg);
ret = ov2775_read_reg(sensor, 0x30bb, &reg_val);
```

```
if (hcg == 1) {
reg_val &= ~(1 << 6);
} else {
reg_val |= (1 << 6);
}
reg_val &= ~0x03;
reg_val |= again;
ret = ov2775_write_reg(sensor, 0x3467, 0x00);
ret |= ov2775_write_reg(sensor, 0x3464, 0x04);
ret |= ov2775_write_reg(sensor, 0x315a, (dgain >> 8) & 0xff);
ret |= ov2775_write_reg(sensor, 0x315b, dgain & 0xff);
ret |= ov2775_write_reg(sensor, 0x30bb, reg_val);
ret |= ov2775_write_reg(sensor, 0x3464, 0x14);
ret |= ov2775_write_reg(sensor, 0x3467, 0x01);
…
}
```

NOTE

Developers should add the exposure function in VVCAM corresponding to their own sensor. Refer to the file ov2775_mipi_v3.c for more information about setting or getting gain and integration time.

6. Define struct i2c_driver for the sensor driver. Because the sensor is connected to an I2C bus, the sensor driver also serves as an I2C client driver. It should be registered to the I2C framework in the Linux kernel, then the sensor driver can use the kernel functions of I2C to communicate with the sensor.

   For example:

```
static const struct of_device_id ov2775_dt_ids[] = {
{ .compatible = "ovti,ov2775" },
{ /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, ov2775_dt_ids);
static struct i2c_driver ov2775_i2c_driver = {
.driver = {
.owner = THIS_MODULE,
.name = "ov2775",
.pm = &ov2775_pm_ops,
.of_match_table = ov2775_dt_ids,
},
.probe = ov2775_probe,
.remove = ov2775_remove,
.id_table = ov2775_id,
};
module_i2c_driver(ov2775_i2c_driver);
```

NOTE

ov2775_probe() is the I2C probe function; ov2775_remove() is the I2C detach function.

7. Define struct v4l2_subdev_ops for the sensor driver. Because the sensor also serves as a V4L2 sub-device, the sensor driver should use the struct v4l2_subdev_ops to assign sub-device operations to the V4L2 framework in the Linux kernel.

   For example:

```
static struct v4l2_subdev_video_ops ov2775_subdev_video_ops = {
.g_parm = ov2775_g_parm,
.s_parm = ov2775_s_parm,
.s_stream = ov2775_s_stream,
};
```

```
static const struct v4l2_subdev_pad_ops ov2775_subdev_pad_ops = {
.enum_frame_size = ov2775_enum_framesizes,
.enum_frame_interval = ov2775_enum_frameintervals,
.enum_mbus_code = ov2775_enum_code,
.set_fmt = ov2775_set_fmt,
.get_fmt = ov2775_get_fmt,
};
static struct v4l2_subdev_core_ops ov2775_subdev_core_ops = {
.s_power = ov2775_s_power,
.ioctl = ov2775_priv_ioctl,
};
static struct v4l2_subdev_ops ov2775_subdev_ops = {
.core = &ov2775_subdev_core_ops,
.video = &ov2775_subdev_video_ops,
.pad = &ov2775_subdev_pad_ops,
};
```

After defining the struct v4l2_subdev_ops, the sensor driver uses the v4l2_i2c_subdev_init() function to initialize struct v4l2_subdev and struct i2c_client in function ov2775_probe(). The function ov2775_probe() is shown below. The v4l2_async_register_subdev_sensor_common() function is then used to register the sensor->subdev to V4L2 framework of the Linux kernel.

```
static int ov2775_probe(struct i2c_client *client,
const struct i2c_device_id *id)
{
int retval;
struct ov2775 *sensor;
sensor = devm_kmalloc(dev, sizeof(*sensor), GFP_KERNEL);
…
sd = &sensor->subdev;
v4l2_i2c_subdev_init(sd, client, &ov2775_subdev_ops);
…
retval = v4l2_async_register_subdev_sensor_common(sd);
…
}
```

6.2.4 Setup HDR

step 6) To setup HDR:

- Enable the HDR function of ISP. Define ISP_HDR_STITCH in the ISP configuration file.

  For example, in the ISP configuration file:

  vim units/mkrel/ISP8000xxxx_Vxxxx/product_cfg_ISP8000xxxx_Vxxxx.cmake

  where: ISP8000xxxx_Vxxxx is the version number of the ISP you are using.

  Add the following macro into the cmake file:

  ```
  add_definitions(-DISP_HDR_STITCH)
  ```

- Enable the HDR function of the sensor. Modify the mode to HDR mode in files Sensor0_Entry.cfg and Sensor1_Entry.cfg.
  - Sensor0_Entry.cfg is the configuration file for the sensor connected to ISP0.
  - Sensor1_Entry.cfg is the configuration file for the sensor connected to ISP1.

An example of Sensor0_Entry.cfg for ov2775:

```
name="ov2775"
drv = "ov2775.drv"
```

```
mode= 1
[sensor_mode.0]
xml = "OV2775.xml"
[sensor_mode.1]
xml = "OV2775.xml"
[sensor_mode.2]
xml = "OV2775.xml"
[sensor_mode.3]
xml = "OV2775_8M_02_720p.xml"
```

NOTE

When mode = 1, select the default mode as HDR mode. The assigned number is as the same as the index number of mode information array (vvcam_mode_info).

6.2.5 Define MIPI Lanes

step 7) In the sensor driver, set **SENSOR_MIPI_LANES** for the MIPI Lane used by the sensor.

For example, in the OV2775 sensor driver file **/isi/drv/OV2775/source/OV2775.c**, modify the MipiLanes data member in the OV2775_IsiGetCapsIss() function as shown:

```
pIsiSensorCaps->MipiLanes = ISI_MIPI_4LANES;
```

6.2.6 Sensor Driver Configuration in V4L2

The i.MX 8M PLUS ISP sensor driver supports V4L2, which is developed according to the standard V4L2 architecture on Linux systems. The following the steps are used to configure V4L2 in the sensor driver.

1. Add **-DAPPMODE=V4L2** and **-DSUBDEV_V4L2=1** into the cmake command when building source code in user space.

   ```
   cmake -DCMAKE_BUILD_TYPE=release -DISP_VERSION=ISP8000NANO_V1802 -
   DPLATFORM=ARM64 -DAPPMODE=V4L2 -DQTLESS=1 -DFULL_SRC_COMPILE=1 -
   DWITH_DWE=1 -DWITH_DRM=1 -DSERVER_LESS=1 -DSUBDEV_V4L2=1 -DENABLE_IRQ=1 .. -Wno-dev
   ```

2. Add the following configuration in **vvcam/v4l2/sensor/Makefile**, where <sensor> should be replaced by the name of the new sensor:

   ```
   obj-m += <sensor>/
   ```

3. Update the device tree file in Linux kernel.

   For example:

   ```
   &i2c0 {
   …
   ov2775_0: ov2775_mipi@36 {
   compatible = "ovti,ov2775";
   reg = <0x36>;
   …
   port {
   ov2775_mipi_0_ep: endpoint {
   data-lanes = <1 2 3 4>;
   clock-lanes = <0>;
   remote-endpoint = <&mipi_csi0_ep>;
   };
   ```

6.3 Sensor Compand Curve

In the vvcam_mode_info_t data structure, the sensor_data_compress_t data structure describes whether the sensor data is compressed or not. If the sensor data is compressed, the sensor_data_compress_t data structure describes the data compression type.

NOTE

- *The maximum bit width for the expand module is 20 bits*
- *To remove the expand module, set data_compress.enable = 0*

Example:

For OV2775 native HDR, sensor data is compressed from 16 bits to 12 bits. So,

x_bit =16 and y_bit=12.

This determines the type of decompression curve used by the compand module.

```
{
.index = 2,
.width = 1920,
.height = 1080,
.fps = 30,
.hdr_mode = SENSOR_MODE_HDR_NATIVE,
.bit_width = 12,
.data_compress.enable = 1,
.data_compress.x_bit = 16,
.data_compress.y_bit = 12,
.bayer_pattern = BAYER_BGGR,
.ae_info = {
.DefaultFrameLengthLines = 0x466,
.one_line_exp_time_ns = 59167,
.max_interrgation_time = 0x466 - 2,
.min_interrgation_time = 1,
.gain_accuracy = 1024,
.max_gain = 21 * 1024,
.min_gain = 3 * 1024,
},
.preg_data = ov2775_1080p_native_hdr_regs,
.reg_data_count = ARRAY_SIZE(ov2775_1080p_native_hdr_regs),
}
```

ISP will decompress according to the specified compression method. If the sensor is compressed from 16-bit to 12-bit, the compand module will call the <sensor>_get_expand_curve() function to get the 12-bit to 16-bit expand curve as defined in the sensor_expand_curve_s data structure.

See below the limitations of the expand curve.

```
(1 << pexpand_curve->expand_px[i]) =
pexpand_curve->expand_x_data[i+1] - pexpand_curve->expand_x_data[i]
```

For example, OV2775 expand curve.

The OV2775 has a data compression from 16-bit to 12-bit by a 4-piece piece-wise linear (PWL) curve defined by the following formula and shown in the following figure.

$$y_{out\_12b} = \begin{cases} \frac{y_{in\_16b}}{2}, & y_{in\_16b} < 1024 \\ \frac{y_{in\_16b}}{4} + 256, & 1024 \le y_{in\_16b} < 2048 \\ \frac{y_{in\_16b}}{8} + 512, & 2048 \le y_{in\_16b} < 16384 \\ \frac{y_{in\_16b}}{32} + 2048, & y_{in\_16b} \ge 16384 \end{cases}$$

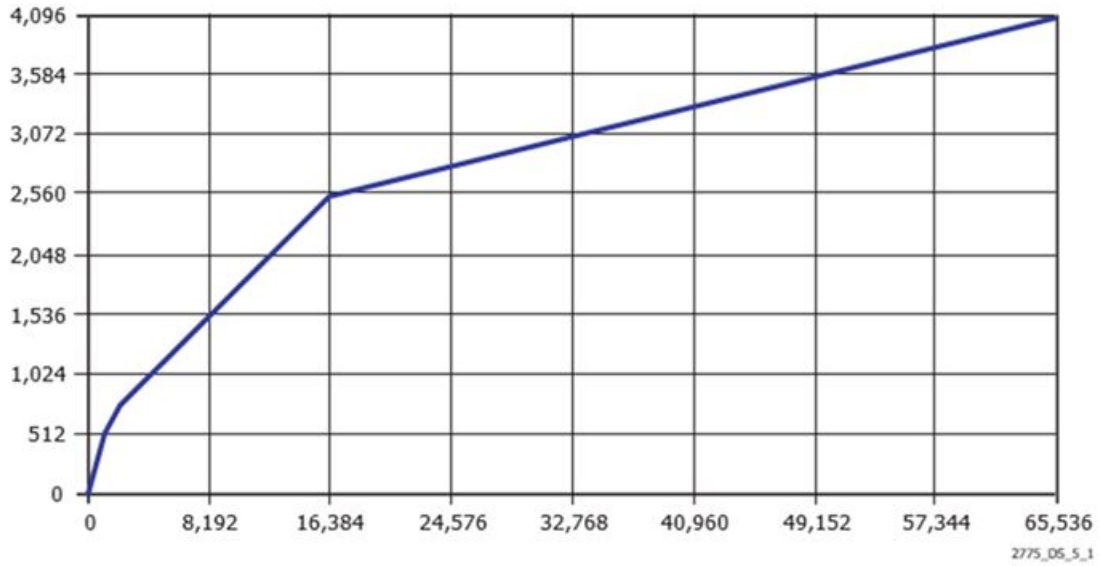


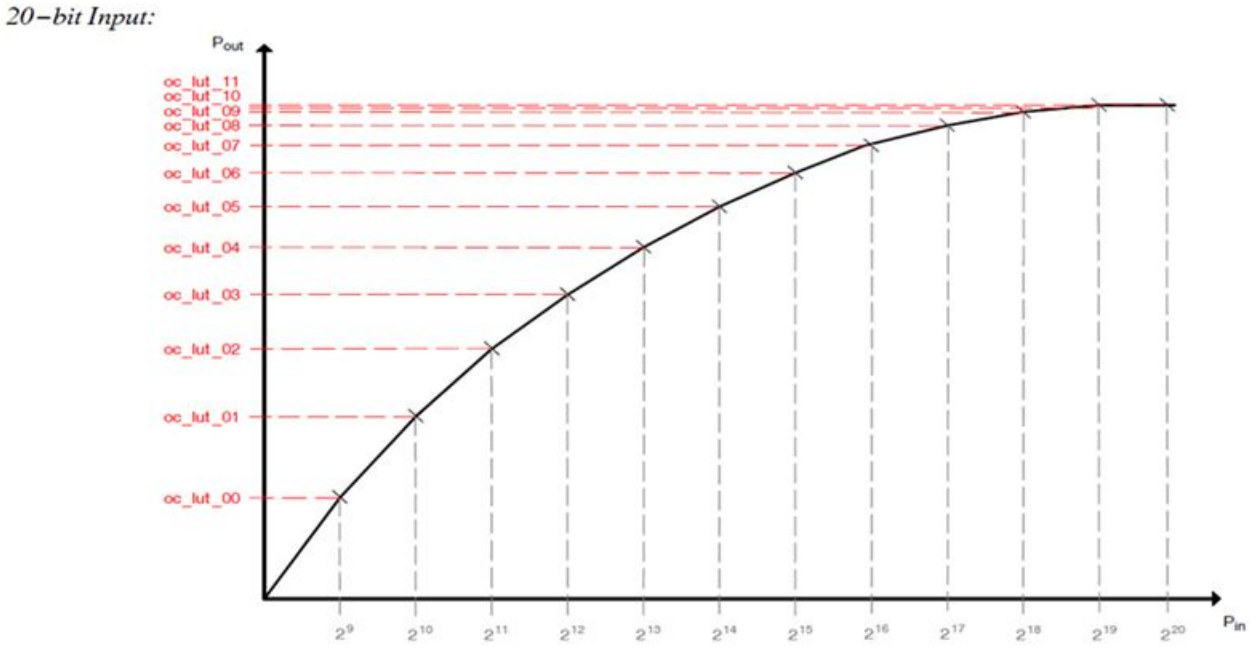
Figure 6. 16-bit to 12-bit PWL compression

The backend processor can decompress 12-bit data to 16-bit data using the following formula.

$$y_{out\_16b} = \begin{cases} 2 \times y_{in\_12b} & y_{in\_12b} < 512 \\ 4 \times (y_{in\_12b} - 256), & 512 \le y_{in\_12b} < 768 \\ 8 \times (y_{in\_12b} - 512), & 768 \le y_{in\_12b} < 2560 \\ 32 \times (y_{in\_12b} - 2048), & y_{in\_12b} \ge 2560 \end{cases}$$

```
int ov2775_get_expand_curve(struct ov2775 *sensor,
sensor_expand_curve_t* pexpand_curve)
{
int i;
if ((pexpand_curve->x_bit) == 12 && (pexpand_curve->y_bit == 16))
{
uint8_t expand_px[64] = {6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6};
memcpy(pexpand_curve->expand_px,expand_px,sizeof(expand_px));
```

```
pexpand_curve->expand_x_data[0] = 0;
pexpand_curve->expand_y_data[0] = 0;
for(i = 1; i < 65; i++)
{
pexpand_curve->expand_x_data[i] =
(1 << pexpand_curve->expand_px[i-1]) +
pexpand_curve->expand_x_data[i-1];
if (pexpand_curve->expand_x_data[i] < 512)
{
pexpand_curve->expand_y_data[i] =
pexpand_curve->expand_x_data[i] << 1;
}
else if (pexpand_curve->expand_x_data[i] < 768)
{
pexpand_curve->expand_y_data[i] =
(pexpand_curve->expand_x_data[i] - 256) << 2;
}
else if (pexpand_curve->expand_x_data[i] < 2560)
{
pexpand_curve->expand_y_data[i] =
(pexpand_curve->expand_x_data[i] - 512) << 3;
}
else
{
pexpand_curve->expand_y_data[i] =
(pexpand_curve->expand_x_data[i] - 2048) << 5;
}
}
return 0;
}
return (-1);
}
ar0820 20-bit to12-bit as 16-bit output:
```



20 → 12−bit	
0x2000	1:1
0x4000	
0x8000	
0x8200	1:64
0x8600	
0x8E00	
0x9E00	
0xBE00	
0xC200	1:1024
0xCA00	
0xDA00	
0xFA00	

The automatic values of the knee-points can be read back from the oc_lut_XX registers but cannot be changed (writes to the oc_lut_XX registers are ignored). All the knee-point registers are MSB-aligned. For example, a programmed value of 0x2000 acts as 0x200 when the output is 12-bit data and acts as 0x2000 when the output is 16-bit data.

The expand curve is defined as follows:

```
expand_px[64] = {13, 13, 14, 9, 10, 11, 12, 13,
10, 11, 12, 13, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,};
expand_x_data[65] ={0,0x2000,0x4000,0x8000,0x8200,0x8600,0x8e00,0x9e00,0xbe00,
0xc200,0xca00,0xda00,0xfa00,0xfa01,0xfa02,0xfa03,0xfa04,
0xfa05,0xfa06,0xfa07,0xfa08,0xfa09,0xfa0a,0xfa0b,0xfa0c,
0xfa0d,0xfa0e,0xfa0f,0xfa10,0xfa11,0xfa12,0xfa13,0xfa14,
0xfa15,0xfa16,0xfa17,0xfa18,0xfa19,0xfa1a,0xfa1b,0xfa1c,
0xfa1d,0xfa1e,0xfa1f,0xfa20,0xfa21,0xfa22,0xfa23,0xfa24,
0xfa25,0xfa26,0xfa27,0xfa28,0xfa29,0xfa2a,0xfa2b,0xfa2c,
0xfa2d,0xfa2e,0xfa2f,0xfa30,0xfa31,0xfa32,0xfa33,0xfa34};
expand_y_data[65] = {0x00,
0x200, 0x400, 0x800, 0x1000, 0x2000, 0x4000, 0x8000, 0x10000,
0x20000, 0x40000, 0x80000, 0x100000, 0x100000, 0x100000, 0x100000,0x100000,
0x100000,0x100000,0x100000,0x100000, 0x100000, 0x100000, 0x100000,0x100000,
0x100000,0x100000,0x100000,0x100000, 0x100000, 0x100000, 0x100000,0x100000,
0x100000,0x100000,0x100000,0x100000, 0x100000, 0x100000, 0x100000,0x100000,
0x100000,0x100000,0x100000,0x100000, 0x100000, 0x100000, 0x100000,0x100000,
0x100000,0x100000,0x100000,0x100000, 0x100000, 0x100000, 0x100000,0x100000};
```

NOTE

Sensor data is 16-bit output, so data_compress must set x_bit = 20 and y_bit = 16.

```
.data_compress = {
.enable = 1,
.x_bit = 20,
.y_bit = 16,
},
```

6.4 Sensor White Balance

ISP AWB is used in normal mode, but in native HDR mode, black level and white balance calibration should be done before the image synthesis at the sensor.

To enable the sensor's WB mode, an interface must be provided to set the AWB mode to ISI_SENSOR_AWB_MODE_SENSOR. In this ISI_SENSOR_AWB_MODE_SENSOR mode, ISP will not perform white balance and black level reduction. Set the sensor for black level and white balance calibration using VVSENSORIOC_S_WB and VVSENSORIOC_S_BLC.

Example：

```
static RESULT OV2775_IsiGetSensorAWBModeIss(IsiSensorHandle_t handle,
IsiSensorAwbMode_t *pawbmode)
{
OV2775_Context_t *pOV2775Ctx = (OV2775_Context_t *) handle;
if (pOV2775Ctx == NULL || pOV2775Ctx->IsiCtx.HalHandle == NULL) {
return RET_NULL_POINTER;
}
if (pOV2775Ctx->SensorMode.hdr_mode == SENSOR_MODE_HDR_NATIVE) {
*pawbmode = ISI_SENSOR_AWB_MODE_SENSOR;
}
else {
*pawbmode = ISI_SENSOR_AWB_MODE_NORMAL;
}
return RET_SUCCESS;
}
```

Chapter 7
Revision history

Table 4. Revision history

Revision number	Date	Substantive changes
1	03/2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2021. **All rights reserved.**

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 03/2021

Document identifier: IMX8MPCSPUG