NXP Semiconductors
User's Guide

Document identifier: IMX8MPCDUG
Rev. LF5.10.35_2.0.0, 30 June 2021

i.MX 8M Plus Camera and Display Guide

Contents

Chapter 1
ISP Independent Sensor ISP Independent Sensor Interface API

1.1 Overview

This document describes the Application Programming Interface (API) of the i.MX 8M Plus ISP Independent Sensor Interface (ISI) module.

Details of the i.MX 8M Plus ISP Independent Sensor Interface API are described in this document.

- First, components such as data types, return codes, enumerations, and relevant structures are described
- Then function syntax and description are presented.

The API explained in this document is applicable to BSP release 5.10.35_2.0.0.

The code is written in C and parameter types follow standard C conventions. This document assumes that the reader understands the fundamentals of C language.

Currently, there are no deprecated functions in this API.

Acronyms and conventions

AE - Auto Exposure

AEC - Auto Exposure Control

AF - Auto Focus

AFM - Auto Focus Measurement

AHB – Advance High-Performance Bus

AWB - Auto White Balance

AXI – Advanced eXtensible Interface

BPT - Bad Pixel Table

CAC - Chromatic Aberration Correction

CPROC – Color Processing Module

CTRL – Control Logic Module

DPCC - Defect Pixel Cluster Correction

DPF - De-noising Pre-Filter

FMF - Focus Measure Function

HVS - Human Visual System

IE – Image Effects Module

ISP – Image Signal Processor

ISR – Interrupt Set/Enable Register

LSC - Lens Shade Correction

MI – Memory Interface

MIPI – Mobile Industry Processor Interface (MIPI) Alliance Standard for camera serial interface 2 (CSI-2)

MRZE – Main Resize Module

SIMP – Super Impose Module

SMIA – Standard Mobile Imaging Architecture

SoC – System on Chip

SRZE – Self Resize

VSM - Video Stabilization Measurement

WDR - Wide Dynamic Range

YCbCr - Color space with one luma and two chroma components used for digital encoding

- Hexadecimal numbers are indicated by the prefix “0x”. For example, 0x32CF.
- Binary numbers are indicated by the prefix “0b”. For example, 0b0011.0010.1100.1111
- Code snippets are given in Consolas typeset.

1.2 Independent Sensor Interface API Components

This section describes the API declared in the **isi/include** directory. Enumerations and structures are listed alphabetically in this document.

1.2.1 Numeric Data Types

The following common numeric data types are used.

Name	Data type
uint8_t	Unsigned 8-bit integer
int8_t	Signed 8-bit integer
uint16_t	Unsigned 16-bit integer
int16_t	Signed 16-bit integer
uint32_t	Unsigned 32-bit integer
int32_t	Signed 32-bit integer
float	Float

1.2.2 RESULT Return Codes

This table specifies the return values for the API functions.

RESULT String Values	Description
RET_FAILURE	General failure
RET_INVALID_PARM	Invalid parameter
RET_NOTSUPP	Feature not supported

Table continues on the next page...

Table continued from the previous page...

RESULT String Values	Description
RET_NULL_POINTER	Callback is a NULL pointer
RET_OUTOFMEM	Not enough memory available
RET_OUTOFRANGE	A configuration parameter is out of range
RET_PENDING	Function successful
RET_SUCCESS	Function successful
RET_WRONG_CONFIG	Given configuration is invalid
RET_WRONG_HANDLE	Invalid instance/HAL handle
RET_WRONG_STATE	Instance is in the wrong state to shutdown

1.2.3 Enumerations

This section describes the enumeration definitions.

1.2.3.1 IsiSensorAwbMode_e

This table specifies the enumeration values for the sensor AWB mode.

Enumeration Values	Value	Description
ISI_SENSOR_AWB_MODE_NORMAL	0	ISP AWB mode
ISI_SENSOR_AWB_MODE_SENSOR	1	Sensor AWB mode

1.2.3.2 IsiColorComponent_e

This table specifies the enumeration values for the color components.

Enumeration Values	Value	Description
ISI_COLOR_COMPONENT_RED	0	
ISI_COLOR_COMPONENT_GREENR	1	
ISI_COLOR_COMPONENT_GREENB	2	
ISI_COLOR_COMPONENT_BLUE	3	
ISI_COLOR_COMPONENT_MAX	4	

1.2.4 Structures

This section describes the structure definitions.

1.2.4.1 IsiCamDrvConfig_s

This structure defines the camera sensor driver specific data.

Structure Members	Type	Description
CameraDriverID	uint32_t	Camera sensor driver ID
*pIsiQuerySensorSupportIss	IsiQuerySensorSupportIss_t	Query sensor support mode
*pfIsiGetSensorIss	IsiGetSensorIss_t	Get the sensor specific information
IsiSensor	IsiSensor_t	Sensor name and function pointers to control the sensor in the ISI layer.

1.2.4.2 IsiRegisterFlags_s

This structure defines the register and flags specific data.

Structure Members	Type	Description
Addr	uint32_t	Register address
DefaultValue	uint32_t	Register value
pName	const char*	Register name
Flags	uint32_t	Read/write flags

1.2.4.3 IsiResolution_s

This structure defines the resolution of sensor.

Structure Members	Type	Description
width	uint32_t	Width of output image in pixels
height	uint32_t	Height of output image in pixels

1.2.4.4 IsiSccbInfo_s

Structure Members	Type	Description
slave_addr	uint8_t	I2C slave address
addr_byte	uint8_t	Address width in Bytes
data_byte	uint8_t	Data width in Bytes

1.2.4.5 IsiSensor_s

This structure defines attributes for the sensor.

Structure Members	Type	Description
*pszName	const char	Name of the camera sensor
*pRegisterTable	const IsiRegDescription_t	Pointer to register table. IsiRegDescription_t is typedef of IsiRegisterFlags_s.
*pIsiCreateSensorIss	IsiCreateSensorIss_t	Create a sensor handle
*pIsiInitSensorIss	IsiInitSensorIss_t	Initialize a sensor handle
*pIsiGetSensorModeIss	IsiGetSensorModeIss_t	Get sensor mode
*pIsiReleaseSensorIss	IsiReleaseSensorIss_t	Release a sensor handle
*pIsiGetCapsIss	IsiGetCapsIss_t	Get sensor capabilities
*pIsiSetupSensorIss	IsiSetupSensorIss_t	Setup sensor capabilities
*pIsiChangeSensorResolutionIss	IsiChangeSensorResolutionIss_t	Change sensor resolution
*pIsiSensorSetStreamingIss	IsiSensorSetStreamingIss_t	Enable/disable streaming of data once sensor is configured
*pIsiSensorSetPowerIss	IsiSensorSetPowerIss_t	Turn sensor power on/off
*pIsiCheckSensorConnectionIss	IsiCheckSensorConnectionIss_t	Check sensor connection with I2C
*pIsiGetSensorRevisionIss	IsiGetSensorRevisionIss_t	Read sensor revision register (if available)
*pIsiRegisterReadIss	IsiRegisterReadIss_t	Read sensor register
*pIsiRegisterWriteIss	IsiRegisterWriteIss_t	Write sensor register
*pIsiExposureControlIss	IsiExposureControlIss_t	[AEC function]
*pIsiGetGainLimitsIss	IsiGetGainLimitsIss_t	[AEC function]
*pIsiGetIntegrationTimeLimitsIss	IsiGetIntegrationTimeLimitsIss_t	[AEC function]
*pIsiGetCurrentExposureIss	IsiGetCurrentExposureIss_t	[AEC function] Get the currently adjusted AE values (gain and integration time)
*pIsiGetGainIss	IsiGetGainIss_t	[AEC function]
*pIsiGetVSGainIss	IsiGetVSGainIss_t	[AEC function]

Table continues on the next page...

Table continued from the previous page...

Structure Members	Type	Description
*pIsiGetLongGainIss	IsiGetLongGainIss_t	[AEC function]
*pIsiGetGainIncrementIss	IsiGetGainIncrementIss_t	[AEC function]
*pIsiSetGainIss	IsiSetGainIss_t	[AEC function]
*pIsiGetIntegrationTimeIss	IsiGetIntegrationTimeIss_t	[AEC function]
*pIsiGetVSIntegrationTimeIss	IsiGetVSIntegrationTimeIss_t	[AEC function]
*pIsiGetLongIntegrationTimeIss	IsiGetLongIntegrationTimeIss_t	[AEC function]
*pIsiGetIntegrationTimeIncrementIss	IsiGetIntegrationTimeIncrementIss_t	[AEC function]
*pIsiSetIntegrationTimeIss	IsiSetIntegrationTimeIss_t	[AEC function]
*pIsiQuerySensorIss	IsiQuerySensorIss_t	[AEC function] Query sensor support info
*pIsiGetResolutionIss	IsiGetResolutionIss_t	[AEC function]
*pIsiGetSensorFpsIss	IsiGetSensorFpsIss_t	[AEC function] Get the current frame rate of sensor
*pIsiSetSensorFpsIss	IsiSetSensorFpsIss_t	[AEC function] Set the frame rate
*pIsiSensorGetExpandCurveIss	IsiSensorGetExpandCurveIss_t	Get expand curve
*pIsiMdiInitMotoDriveMds	IsiMdiInitMotoDriveMds_t	[AF function]
*pIsiMdiSetupMotoDrive	IsiMdiSetupMotoDrive_t	[AF function]
*pIsiMdiFocusSet	IsiMdiFocusSet_t	[AF function]
*pIsiMdiFocusGet	IsiMdiFocusGet_t	[AF function]
*pIsiMdiFocusCalibrate	IsiMdiFocusCalibrate_t	[AF function]
*pIsiGetSensorMipiInfoIss	IsiGetSensorMipiInfoIss_t	[MIDI function]
*pIsiResetSensorIss	IsiResetSensorIss_t	Reset sensor
*pIsiActivateTestPattern	IsiActivateTestPattern_t	Enable/disable test pattern
*pIsiEnableHdr	IsiEnableHdr_t	Enable/disable HDR
*pIsiSetBayerPattern	IsiSetBayerPattern_t	Set Bayer pattern
*pIsiSensorSetBlcIss	IsiSensorSetBlcIss_t	Set sensor BLC
*pIsiSensorSetWBIss	IsiSensorSetWBIss_t	Set sensor WB gain

Table continues on the next page...

Table continued from the previous page...

Structure Members	Type	Description
*pIsiGetSensorAWBModeIss	IsiGetSensorAWBModeIss_t	Get AWB mode

1.2.4.6 IsiSensorCaps_s

This structure defines the sensor capabilities.

Structure Members	Type	Description
BusWidth	uint32_t	Sensor bus width
Mode	uint32_t	Operating mode of the image sensor in terms of output data format and timing data transmission
FieldSelection	uint32_t	Sample fields selection: 0x1: sample all fields 0x2: sample only even fields 0x4: sample only odd fields
YCSequence	uint32_t	If the sensor input is YCbCr data, YCSequence defines the YCbCr type
Conv422	uint32_t	Color sub-sampling mode
BPat	uint32_t	Bayer pattern
HPol	uint32_t	Horizontal polarity
VPol	uint32_t	Vertical polarity
Edge	uint32_t	Sample edge
Resolution	IsiResolution_t	Sensor resolution
SmiaMode	uint32_t	Unused
MipiMode	uint32_t	MIPI transfer data format
MipiLanes	uint32_t	Number of MIPI lanes
enableHdr	uint32_t	Enable HDR

1.2.4.7 IsiSensorContext_s

Structure Members	Type	Description
fd	int	/dev/v4l-subdev file description
HalHandle	HalHandle_t	Handle of HAL session to use. HalHandle_t is typedef void *.
*pSensor	IsiSensor_t	Pointer to the sensor device. IsiSensor_t is typedef of IsiSensor_s.

1.2.4.8 IsiSensorInstanceConfig_s

This structure defines the configuration structure used to create a new sensor instance.

Structure Members	Type	Description
HalHandle	HalHandle_t	Handle of HAL session to use
HalDevID	uint32_t	HAL device ID of this sensor
I2cBusNum	uint8_t	Sensor connector I2C bus
SlaveAddr	uint16_t	Sensor I2C slave address
I2cAfBusNum	uint8_t	I2C bus for the Auto Focus module
SlaveAfAddr	uint16_t	I2C slave address of the Auto Focus module
SensorModeIndex	uint32_t	Sensor mode index
*pSensor	IsiSensor_t	The pointer to the sensor driver interface
hSensor	IsiSensorHandle_t	Sensor handle returned by IsiCreateSensorIss. IsiSensorHandle_t is typedef void *.
szSensorNodeName[32]	char	Sensor node name

1.2.4.9 IsiSensorMipiInfo

This structure defines the sensor specific information for MIPI.

Structure Members	Type	Description
ucMipiLanes	uint8_t	Number of used MIPI lanes by sensor

1.2.4.10 sensor_blc_s

This structure defines the configuration structure used to set the sensor black level.

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

1.2.4.11 sensor_data_compress_s

This structure defines the configuration structure used to set the sensor expand curve.

Structure Members	Type	Description
enable	uint32_t	Enable the sensor expand curve
x_bit	uint32_t	Expand curve input data bit width
y_bit	uint32_t	Expand curve output data bit width

1.2.4.12 sensor_expand_curve_s

This structure defines the configuration structure used to set the sensor data compress.

Structure Members	Type	Description
x_bit	uint32_t	Expand curve input data bit width
y_bit	uint32_t	Expand curve output data bit width
expand_px[64]	uint8_t	X axis interval (1 << px[i]) = expand_x_data[i+1] - expand_x_data[i]
expand_x_data[65]	uinst32_t	Expand curve X axis - 65 points
expand_y_data[65]	uinst32_t	Expand curve Y axis - 65 points

1.2.4.13 sensor_white_balance_s

This structure defines the configuration structure used to set the sensor white balance.

Structure Members	Type	Description
r_gain	uint32_t	Red Gain
gr_gain	uint32_t	Green-Red Gain

Table continues on the next page...

Table continued from the previous page...

Structure Members	Type	Description
gb_gain	uint32_t	Green-Blue Gain
b_gain	uint32_t	Blue Gain

1.2.4.14 vvcam_ae_info_t

This structure defines the parameter configuration structure of the sensor Auto Exposure information.

Structure Members	Type	Description
DefaultFrameLengthLines	uint32_t	Sensor mode initial frame length lines
CurFrameLengthLines	uint32_t	Sensor mode current frame length lines
one_line_exp_time_ns	uint32_t	One line exposure time in ns
max_interrgation_time	uint32_t	Maximum exposure line
min_interrgation_time	uint32_t	Minimum exposure line
interrgation_accuracy	uint32_t	Exposure accuracy (always 1)
max_gain	uint32_t	Maximum gain
min_gain	uint32_t	Minimum gain
gain_accuracy	uint32_t	Gain accuracy (always 1024)
cur_fps	uint32_t	Current FPS
hdr_ratio	uint32_t	HD ratio

1.2.4.15 vvcam_mode_info_t

This structure defines the parameter configuration structure of the sensor mode information.

Structure Members	Type	Description
index	uint32_t	Sensor mode index
width	uint32_t	Sensor mode width
height	uint32_t	Sensor mode height
fps	uint32_t	Sensor mode FPS

Table continues on the next page...

Table continued from the previous page...

Structure Members	Type	Description
hdr_mode	uint32_t	Sensor HDR mode
stitching_mode	uint32_t	Sensor stitching mode
bit_width	uint32_t	Sensor data bit width
data_compress	sensor_data_compress_t	Sensor data compress information
bayer_pattern	uint32_t	Sensor data Bayer pattern
ae_info	vvcam_ae_info_t	Sensor AE information
preg_data	void *	Sensor register init array pointer
reg_data_count	uint32_t	Sensor register data count

1.2.4.16 vvcam_mode_info_array_t

This structure defines the number and information of the sensor mode.

Structure Members	Type	Description
count	uint32_t	Sensor mode count
modes[VVCAM_SUPPORT_MAX_MODE_COUNT]	vvcam_mode_info	Sensor mode information

1.2.4.17 IsiResolution_t

This structure provides the sensor resolution.

Structure Members	Type	Description
width	uint16_t	Width of sensor image
height	uint16_t	Height of sensor image

1.3 Independent Sensor Interface Functions

This section provides an overview of the functions for independent sensor interface.

1.3.1 General API Functions

IsiInitSensorIss_t

Description:

This function initializes a sensor.

Syntax:

```
RESULT IsiInitSensorIss_t (
IsiSensorHandle_t  handle
);
```

Parameters:

handle	Sensor instance handle.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER, RET_OUTOFMEM, RET_WRONG_HANDLE, RET_NOTSUPP,
RET_FAILURE
```

IsiCreateSensorIss_t

Description:

This function creates a new sensor instance.

Syntax:

```
RESULT IsiCreateSensorIss_t (
IsiSensorInstanceConfig_t  *pConfig
);
```

Parameters:

***pConfig**	Pointer to the configuration of the new sensor instance.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER, RET_OUTOFMEM
```

IsiGetSensorModeIss_t

Description:

This function is used to get the sensor mode info by sensor mode index.

Syntax:

```
RESULT IsiGetSensorModeIss_t (
IsiSensorHandle_t  *handle,
void    *pmode
);
```

Parameters:

* handle	Sensor instance handle.
*pmode	Pointer to the vvcam_mode_info data structure.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER, RET_OUTOFMEM
```

IsiQuerySensorIss_t

Description:

This function is used to query the sensor support modes info.

Syntax:

```
RESULT IsiQuerySensorIss_t (
IsiSensorHandle_t  *handle,
vvcam_mode_info_array_t *pSensorInfo
);
```

Parameters:

* handle	Sensor instance handle.
* pSensorInfo	Pointer to the vvcam_mode_info_array_s data structure.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER, RET_OUTOFMEM
```

IsiReleaseSensorIss_t

Description:

This function destroys/releases a sensor instance.

Syntax:

```
RESULT IsiReleaseSensorIss_t (
IsiSensorHandle_t   handle
);
```

Parameters:

Handle	Sensor instance handle.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NOTSUPP
```

IsiGetCapsIss_t

Description:

This function fills in the correct pointers for the sensor description structure.

Syntax:

```
RESULT IsiGetCapsIss_t (
IsiSensorHandle_t   handle,
```

```
IsiSensorCaps_t   *pIsiSensorCaps
);
```

Parameters:

handle	Sensor instance handle.
***pIsiSensorCaps**	Pointer to the IsiSensorCaps_t data structure.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiSetupSensorIss_t

Description:

This function sets up the image sensor with the specified configuration.

Syntax:

```
RESULT IsiSetupSensorIss_t (
IsiSensorHandle_t   handle,
IsiSensorConfig_t   *pConfig
);
```

Parameters:

handle	Sensor instance handle.
***pConfig**	Pointer to the IsiSensorCaps_t data structure. (typedef IsiSensorCaps_t IsiSensorConfig_t)

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiChangeSensorResolutionIss_t

Description:

This function changes the image sensor resolution while keeping all other static settings. Dynamic settings, such as current gain and integration time are kept as close as possible.

Note: re-read current and minimum/maximum values as they may have changed.

Syntax:

```
RESULT IsiChangeSensorResolutionIss_t (
IsiSensorHandle_t   handle,
uint16_t      width,
uint16_t      height
);
```

Parameters:

handle	Sensor instance handle.
width	Resolution width.
height	Resolution height.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_WRONG_STATE, RET_OUTOFRANGE
```

IsiSensorSetStreamingIss_t

Description:

This function enables/disables streaming of sensor data, if possible.

Syntax:

```
RESULT IsiSensorSetStreamingIss_t (
IsiSensorHandle_t   handle,
bool_t     on
);
```

Parameters:

handle	Sensor instance handle.
on	New streaming state. BOOL_TRUE = on; BOOL_FALSE = off

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER, RET_WRONG_STATE
```

IsiSensorSetPowerIss_t

Description:

This function performs the power-up/power-down sequence of the camera, if possible.

Syntax:

```
RESULT IsiSensorSetPowerIss_t (
IsiSensorHandle_t   handle,
bool_t     on
);
```

Parameters:

handle	Sensor instance handle.
on	New power state. BOOL_TRUE = on; BOOL_FALSE = off

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiCheckSensorConnectionIss_t

Description:

This function checks the connection to the camera sensor, if possible.

Syntax:

```
RESULT IsiCheckSensorConnectionIss_t (
IsiSensorHandle_t   handle
);
```

Parameters:

handle	Sensor instance handle.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiGetSensorRevisionIss_t

Description:

This function reads the sensor revision register and returns it.

Syntax:

```
RESULT IsiGetSensorRevisionIss_t (
IsiSensorHandle_t   handle,
uint32_t    *p_value
);
```

Parameters:

handle	Sensor instance handle.
***p_value**	Pointer to the sensor revision register value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER, RET_NOTSUPP
```

IsiGetResolutionIss_t

Description:

This function reads the resolution value from the image sensor module.

Syntax:

```
RESULT IsiGetResolutionIss_t (
IsiSensorHandle_t   handle,
uint16_t     *pwidth,
```

```
uint16_t     *pheight
);
```

Parameters:

handle	Sensor instance handle.
***pwidth**	Pointer to resolution width.
***pheight**	Pointer to resolution height.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiRegisterWriteIss_t

Description:

This function writes a given number of bytes to the image sensor device by calling the corresponding sensor function.

Syntax:

```
RESULT IsiRegisterWriteIss_t (
IsiSensorHandle_t   handle,
const uint32_t   address,
const uint32_t   *p_value
);
```

Parameters:

handle	Sensor instance handle.
address	Register address.
***p_value**	Register value to write.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NOTSUPP
```

IsiRegisterReadIss_t

Description:

This function reads the value from the specified register from the image sensor device.

Syntax:

```
RESULT IsiRegisterReadIss_t (
IsiSensorHandle_t   handle,
const uint32_t   address,
uint32_t    value
);
```

Parameters:

handle	Sensor instance handle.
address	Register address.
value	Register value read from the register.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER, RET_NOTSUPP
```

IsiGetSensorMipiInfoIss_t

Description:

This function is used to get the MIPI information.

Syntax:

```
RESULT IsiGetSensorMipiInfoIss_t (
                IsiSensorHandle_t handle,
                IsiSensorMipiInfo *ptIsiSensorMipiInfo
                );
```

Parameters:

handle	Sensor instance handle.
***ptIsiSensorMipiInfo**	MIPI sensor information.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiResetSensorIss_t

Description:

This function is used to reset the sensor.

Syntax:

```
RESULT IsiResetSensorIss_t (
                IsiSensorHandle_t handle
                );
```

Parameters:

handle	Sensor instance handle.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSetCsiConfig_t

Description:

This function sets the CSI configuration.

Syntax:

```
RESULT IsiSetCsiConfig_t (
IsiSensorHandle_t handle,
uint32_t clk
);
```

Parameters:

handle	Sensor instance handle.
clk	Set clock.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER, RET_NOTSUPP
```

IsiEnableHdr_t

Description:

This function enables/disables the HDR.

Syntax:

```
RESULT IsiEnableHdr_t (
                IsiSensorHandle_t handle,
                const bool_t enable
                );
```

Parameters:

handle	Sensor instance handle.
enable	Enable/disable flag.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER, RET_NOTSUPP
```

1.3.2 AEC API Functions

IsiGetIntegrationTimeLimitsIss_t

Description:

This function returns the integration time minimum and maximum values of a sensor instance.

Syntax:

```
RESULT IsiGetIntegrationTimeLimitsIss_t (
IsiSensorHandle_t   handle,
float     *pMinIntegrationTime,
float     *pMaxIntegrationTime
);
```

Parameters:

handle	Sensor instance handle.
***pMinIntegrationTime**	Pointer to the minimum integration time value.
***pMaxIntegrationTime**	Pointer to the maximum integration time value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiGetGainLimitsIss_t

Description:

This function returns the gain minimum and maximum values of a sensor instance.

Syntax:

```
RESULT IsiGetGainLimitsIss_t (
IsiSensorHandle_t   handle,
float     *pMinGain,
float     *pMaxGain
);
```

Parameters:

handle	Sensor instance handle.
***pMinGain**	Pointer to the minimum exposure value.
***pMaxGain**	Pointer to the maximum exposure value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiGetGainIncrementIss_t

Description:

This function returns the smallest possible gain increment.

Syntax:

```
RESULT IsiGetGainIncrementIss_t (
IsiSensorHandle_t   handle,
float     *pIncr
);
```

Parameters:

handle	OV14825 sensor instance handle.
***pIncr**	Pointer to the smallest possible gain increment.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiExposureControlIss_t

Description:

This function sets the exposure values (gain and integration time) of a sensor instance.

Syntax:

```
RESULT IsiExposureControlIss_t (
IsiSensorHandle_t   handle,
float     NewGain,
float     NewIntegrationTime,
uint8_t    *pNumberOfFramesToSkip,
float     *pSetGain,
float     *pSetIntegrationTime,
float     *hdr_ratio
);
```

Parameters:

handle	Sensor instance handle.
NewGain	Newly calculated gain to be set.
NewIntegrationTime	Newly calculated integration time to be set.
***pNumberOfFramesToSkip**	Pointer to the number of frames to skip until AE is executed again.
***pSetGain**	Pointer to the exposure gain that is set.
***pSetIntegrationTime**	Pointer to the integration time that is set.
***hdr_ratio**	Pointer to the HDR ratio that is set.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_NULL_POINTER
```

IsiGetCurrentExposureIss_t

Description:

This function returns the currently adjusted AE values.

Syntax:

```
RESULT IsiGetCurrentExposureIss_t (
IsiSensorHandle_t   handle,
float     *pCurGain,
float     *pCurIntegrationTime
);
```

Parameters:

handle	Sensor instance handle.
***pCurGain**	Pointer to the current exposure gain that is set.
***pCurIntegrationTime**	Pointer to the current integration time that is set.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetGainIss_t

Description:

This function reads gain values from the image sensor module.

Syntax:

```
RESULT IsiGetGainIss_t (
IsiSensorHandle_t   handle,
float     *pGain
);
```

Parameters:

handle	Sensor instance handle.
***pGain**	Pointer to the gain values.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSetGainIss_t

Description:

This function writes gain values to the image sensor module.

Syntax:

```
RESULT IsiSetGainIss_t (
IsiSensorHandle_t   handle,
float     NewGain,
float     *pSetGain,
float     *hdr_ratio
);
```

Parameters:

handle	Sensor instance handle.
NewGain	Gain to be set.

Table continues on the next page...

Table continued from the previous page...

***pSetGain**	Pointer to the gain values.
&hdr_ratio	Pointer to the HDR ratio.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetIntegrationTimeIss_t

Description:

This function reads integration time values from the image sensor module.

Syntax:

```
RESULT IsiGetIntegrationTimeIss_t (
IsiSensorHandle_t   handle,
float     *pIntegrationTime
);
```

Parameters:

handle	Sensor instance handle.
***pIntegrationTime**	Pointer to the integration time values.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSetIntegrationTimeIss_t

Description:

This function writes integration time values to the image sensor module.

Syntax:

```
RESULT IsiSetIntegrationTimeIss_t (
IsiSensorHandle_t   handle,
float     NewSetIntegrationTime,
float     *pSetIntegrationTime,
float     *hdr_ratio
);
```

Parameters:

handle	Sensor instance handle.
NewSetIntegrationTime	New integration time value to write.

Table continues on the next page...

Table continued from the previous page...

***pSetIntegrationTime**	Pointer to the set integration time value.
***hdr_ratio**	Pointer to the set HDR ratio.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetSensorFpsIss_t

Description:

This function is used to get the sensor current frame rate.

Syntax:

```
RESULT IsiGetSensorFpsIss_t (
IsiSensorHandle_t   handle,
uint32_t     *pFps,
);
```

Parameters:

handle	Sensor instance handle.
*** pFps**	Pointer to frame rate.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSetSensorFpsIss_t

Description:

This function is used to set the sensor frame rate.

Syntax:

```
RESULT IsiSetSensorFpsIss_t(
IsiSensorHandle_t   handle,
uint32_t     Fps,
);
```

Parameters:

handle	Sensor instance handle.
Fps	frame rate.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetIntegrationTimeIncrementIss_t

Description:

This function returns the smallest possible integration time increment.

Syntax:

```
RESULT IsiGetIntegrationTimeIncrementIss_t(
IsiSensorHandle_t   handle,
float     *pIncr
);
```

Parameters:

handle	Sensor instance handle.
***pIncr**	Pointer to the smallest possible integration time increment.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

1.3.3 AWB API Functions

IsiGetSensorAWBModeIss_t

Description:

This function is used to get the sensor AWB mode.

Syntax:

```
RESULT IsiGetSensorAWBModeIss_t(
IsiSensorHandle_t  handle,
IsiSensorAwbMode_t  *pawb_mode
);
```

Parameters:

handle	Sensor instance handle.
pawb_mode	Pointer to the IsiSensorAwbMode_e enumeration.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSensorSetBlcIss_t

Description:

This function is used to set the sensor black level.

Syntax:

```
RESULT IsiSensorSetBlcIss_t(
IsiSensorHandle_t  handle,
```

```
sensor_blc_t   *pblc
);
```

Parameters:

handle	Sensor instance handle.
pblc	Pointer to the sensor_blc_t structure.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSensorSetWBIss_t

Description:

This function is used to set the sensor white balance.

Syntax:

```
RESULT IsiSensorSetWBIss_t(
IsiSensorHandle_t   handle,
sensor_white_balance_t   *pwb
);
```

Parameters:

handle	Sensor instance handle.
pwb	Pointer to the sensor_white_balance_t structure.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

1.3.4 Expand API Functions

IsiSensorGetExpandCurveIss_t

Description:

This function used to get the sensor expand curve.

Syntax:

```
RESULT IsiSensorGetExpandCurveIss_t(
IsiSensorHandle_t   handle,
sensor_expand_curve_t  *pexpand_curve
);
```

Parameters:

handle	Sensor instance handle.

Table continues on the next page...

Table continued from the previous page...

pexpand_curve	Pointer to the sensor_expand_curve_t structure.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

1.3.5 AF API Functions

IsiMdiInitMotoDriveMds_t

Description:

This function performs the general initialization tasks, such as I/O initialization.

Syntax:

```
RESULT IsiMdiInitMotoDriveMds_t(
IsiSensorHandle_t   handle
);
```

Parameters:

handle	Sensor instance handle.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiMdiSetupMotoDrive_t

Description:

This function setups the MotoDrive and returns the maximum possible focus step.

Syntax:

```
RESULT IsiMdiSetupMotoDrive_t(
IsiSensorHandle_t   handle,
uint32_t    *pMaxStep
);
```

Parameters:

handle	Sensor instance handle.
***pMaxStep**	Pointer to the maximum possible focus step.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiMdiFocusSet_t

Description:

This function sets the absolute focus point for the lens system.

Syntax:

```
RESULT IsiMdiFocusSet_t(
IsiSensorHandle_t   handle,
const uint32_t   AbsStep
);
```

Parameters:

handle	Sensor instance handle.
AbsStep	Absolute focus point to apply.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiMdiFocusGet_t

Description:

This function gets the current absolute focus point for the lens system.

Syntax:

```
RESULT IsiMdiFocusGet_t(
IsiSensorHandle_t   handle,
const uint32_t   *pAbsStep
);
```

Parameters:

handle	Sensor instance handle.
***pAbsStep**	Pointer to the current absolute focus point.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiMdiFocusCalibrate_t

Description:

This function triggers a forced calibration of the focus hardware.

Syntax:

```
RESULT IsiMdiFocusCalibrate_t(
IsiSensorHandle_t   handle
);
```

Parameters:

handle	Sensor instance handle.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

1.3.6 Test Pattern API Functions

IsiActivateTestPattern_t

Description:

This function activates or deactivates the sensor test pattern. The default pattern is color bar.

Syntax:

```
RESULT IsiActivateTestPattern_t(
IsiSensorHandle_t   handle,
const bool_t    enable
);
```

Parameters:

handle	Sensor instance handle.
enable	0: deactivate the sensor test pattern; 1: activate the sensor test pattern.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiSetBayerPattern_t

Description:

This function sets the Bayer pattern.

Syntax:

```
RESULT IsiSetBayerPattern_t(
IsiSensorHandle_t   handle,
uint8_t    pattern
);
```

Parameters:

handle	Sensor instance handle.
pattern	Sets the Bayer pattern.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

1.3.7 Miscellaneous API Functions

IsiGetLongIntegrationTimeIss_t

Description:

This function sets long exposure integration time.

Syntax:

```
RESULT IsiGetLongIntegrationTimeIss_t(
    IsiSensorHandle_t   handle,
    float *pIntegrationTime
    );
```

Parameters:

handle	Sensor instance handle.
***pIntegrationTime**	Long exposure integration time value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetLongGainIss_t

Description:

This function sets long exposure gain value.

Syntax:

```
RESULT IsiGetLongGainIss_t(
    IsiSensorHandle_t   handle,
    float *pSetGain
    );
```

Parameters:

handle	Sensor instance handle.
***pSetGain**	Long Gain value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetVSGainIss_t

Description:

This function sets very short exposure gain value.

Syntax:

```
RESULT IsiGetVSGainIss_t(
    IsiSensorHandle_t   handle,
```

```
    float *pSetGain
    );
```

Parameters:

handle	Sensor instance handle.
***pSetGain**	Short Gain value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiGetVSIntegrationTimeIss_t

Description:

This function sets very short exposure integration time.

Syntax:

```
RESULT IsiGetVSIntegrationTimeIss_t(
    IsiSensorHandle_t   handle,
    float *pSetIntegrationTime
    );
```

Parameters:

handle	Sensor instance handle.
***pSetIntegrationTime**	Very short integration time value.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiDumpAllRegisters_t

Description:

This function dumps all registers to the specified file.

Syntax:

```
RESULT IsiDumpAllRegisters_t(
IsiSensorHandle_t   handle,
const uint8_t    *filename
);
```

Parameters:

handle	Sensor instance handle.
***filename**	File name to dump all registers.

Returns:

```
RESULT Return Code: RET_SUCCESS, RET_WRONG_HANDLE, RET_NULL_POINTER
```

IsiTryToSetConfigFromPreferredCaps_t

Description:

This function tries to set the referenced sensor configuration parameter to the first of the given preferred capabilities that is included in the given capability mask. If none of the preferred capabilities is supported, the configuration parameter value remains unchanged.

Note: Use this function, for example, to modify the retrieved default sensor configuration, parameter for parameter, according to some external preferences while taking the retrieved sensor capabilities for that configuration parameter into account.

Syntax:

```
boot_t IsiTryToSetConfigFromPreferredCaps_t(
uint32_t    *pConfigParam,
uint32_t    *prefList,
uint32_t    capsmask
);
```

Parameters:

***pConfigParam**	Pointer to parameter of the sensor configuration structure.
***prefList**	Reference to 0 (zero) terminated array of preferred capability values in descending order.
capsmask	Bit mask of supported capabilities for that parameter.

Returns:

```
BOOL_TRUE: preferred capability set in the reference configuration parameter
BOOL_FALSE: preferred capability is not supported
```

IsiTryToSetConfigFromPreferredCap_t

Description:

This function tries to set the referenced sensor configuration parameter to the given preferred capability while checking that capability against the given capability mask. If that capability is not supported, the config parameter value remains unchanged.

Note: Use this function, for example, to modify the retrieved default sensor configuration, parameter for parameter, according to some external preferences while taking the retrieved sensor capabilities for that configuration parameter into account.

Syntax:

```
boot_t IsiTryToSetConfigFromPreferredCap_t(
uint32_t    *pConfigParam,
uint32_t    *prefcap,
uint32_t    capsmask
);
```

Parameters:

***pConfigParam**	Pointer to parameter of the sensor configuration structure.
***prefcap**	Preferred capability value.
capsmask	Bit mask of supported capabilities for that parameter.

Returns:

```
BOOL_TRUE: preferred capability set in the reference configuration parameter
BOOL_FALSE: preferred capability is not supported
```

Chapter 2
Camera Sensor Porting Guide

2.1 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.

2.2 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.

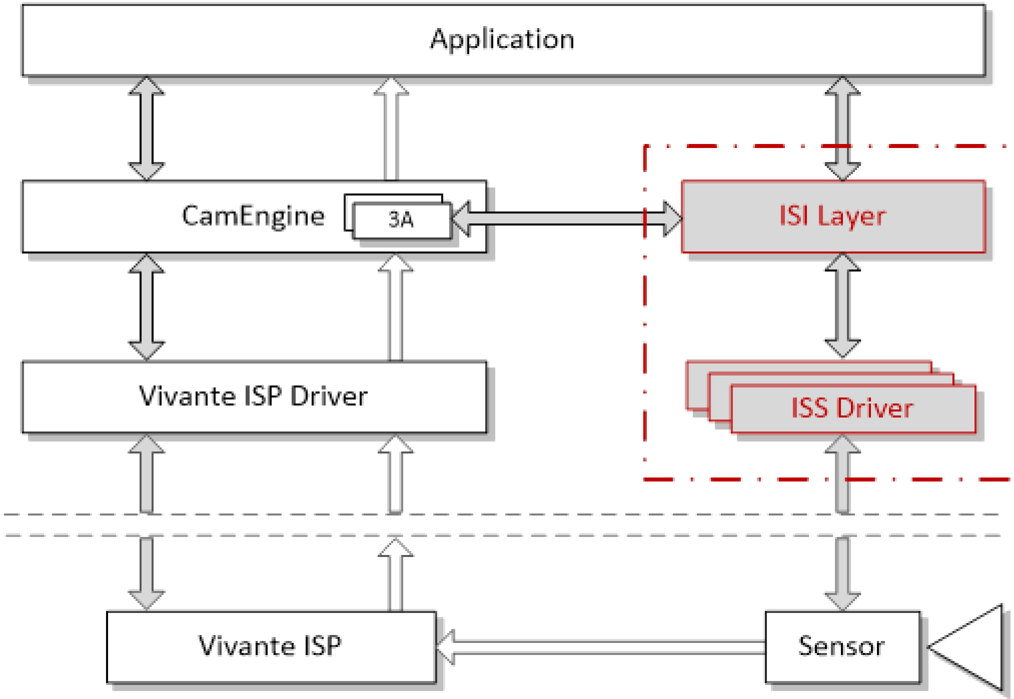


Figure 1. ISP Software Architecture

2.2.1 ISS (Image Sensor Specific) Driver

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

2.2.2 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^{2}C kernel driver.
  - **Sensor Driver**: performs sensor API operations on sensor hardware.
  - **I^{2}C:** 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.

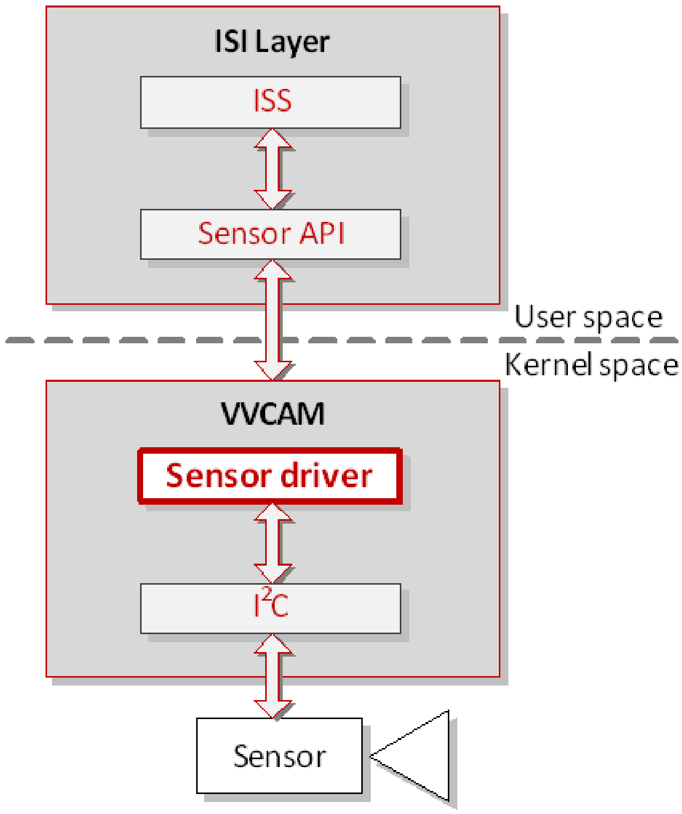


Figure 2. Sensor Module in Linux Kernel

2.3 ISP Independent Sensor Interface (ISI) API reference

Structures and functions are provided here for convenience.

2.3.1 ISI Structures

2.3.1.1 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.

2.3.1.2 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.

2.3.1.3 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.

2.3.2 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
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

Table continues on the next page...

Table continued from the previous page...

ISI API	Function Description
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
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

2.3.3 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
<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

Table continues on the next page...

Table 1. Sensor API Reference (continued)

Sensor API	Description
<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
<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

Table continues on the next page...

Table 1. Sensor API Reference (continued)

Sensor API	Description
<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

2.3.4 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.

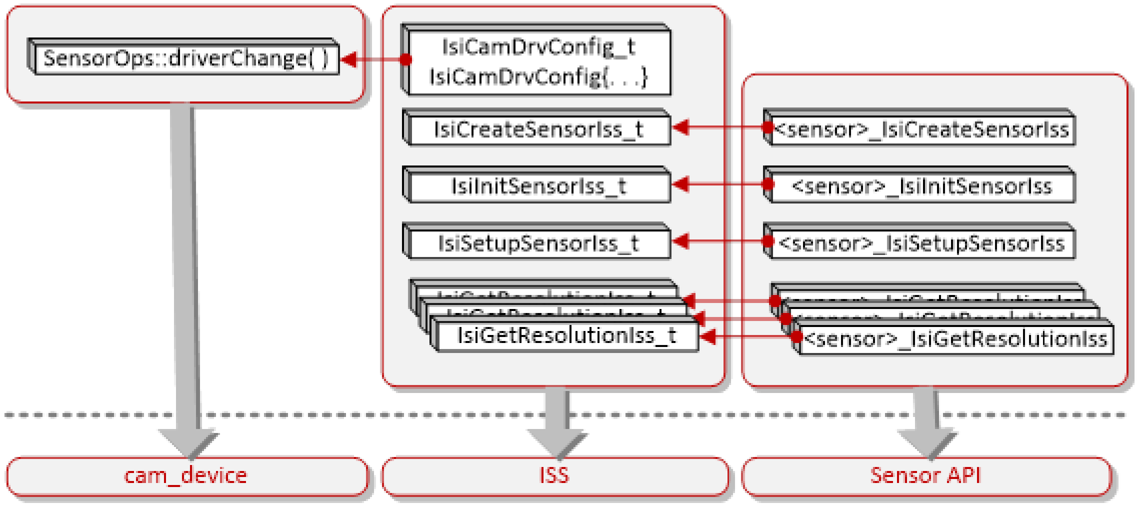


Figure 3. User Space Flow

2.4 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.

2.4.1 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

Table continues on the next page...

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

IOCTL	IOCTL Operation
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
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

2.4.2 IOCTL Call Flow

The IOCTL supports V4L2 Mode as described below.

2.4.2.1 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.

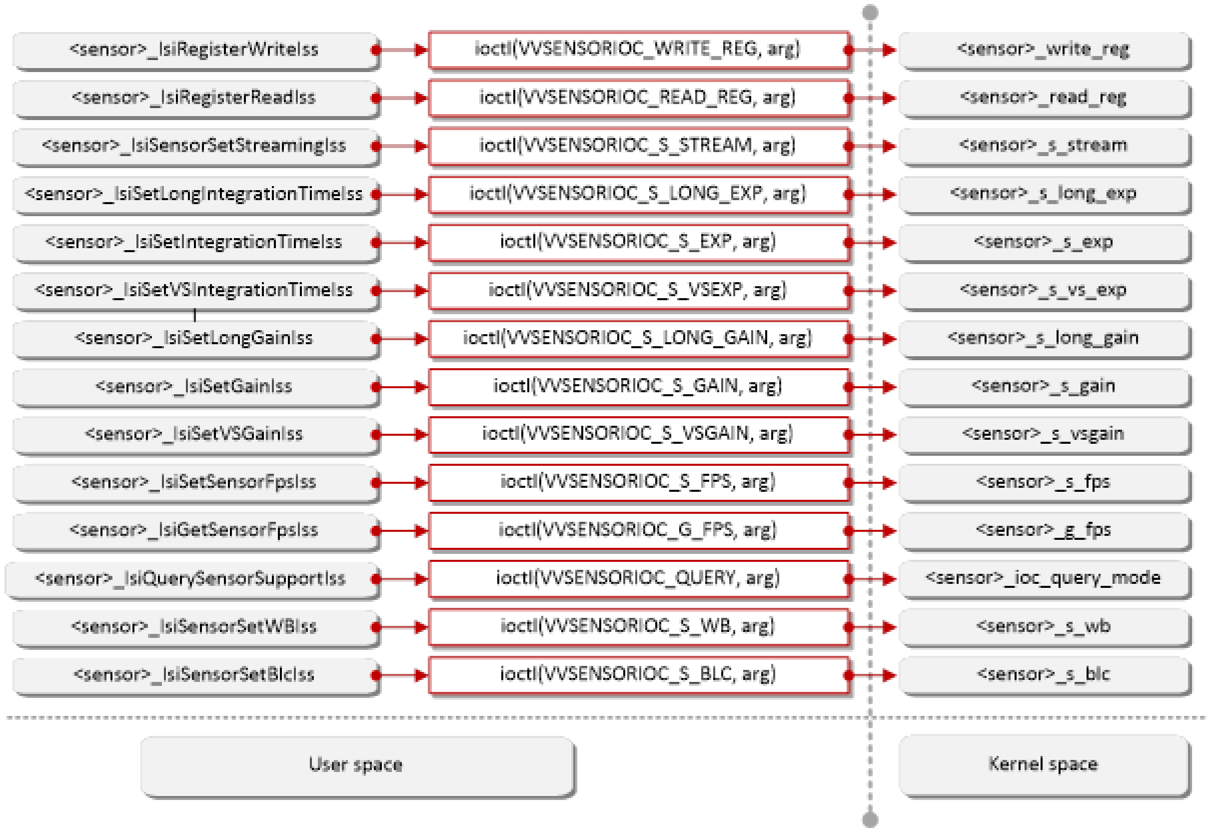


Figure 4. IOCTL Call Flow in V4L2 Mode

2.5 VVCam API Reference

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

2.5.1 Sensor Driver Enumerations

2.5.1.1 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

2.5.1.2 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.

2.5.1.3 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

2.5.2 Sensor Driver Structures

2.5.2.1 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

2.5.2.2 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.

2.5.2.3 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

2.5.2.4 sensor_mipi_info

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

2.5.2.5 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

2.5.2.6 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.)
cur_fps	uint32_t	Current frame rate
hdr_radio	uint32_t	HDR radio

2.5.2.7 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

2.5.2.8 vvcam_mode_info_t

Structure Members	Type	Description
index	uint32_t	Mode index

Table continues on the next page...

Table continued from the previous page...

Structure Members	Type	Description
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

2.5.3 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

Table continues on the next page...

Table 3. Sensor V4l2 Driver API (continued)

API Name	Description
<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

2.6 Camera Sensor Driver in V4L2 Mode

2.6.1 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.

2.6.1.1 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.

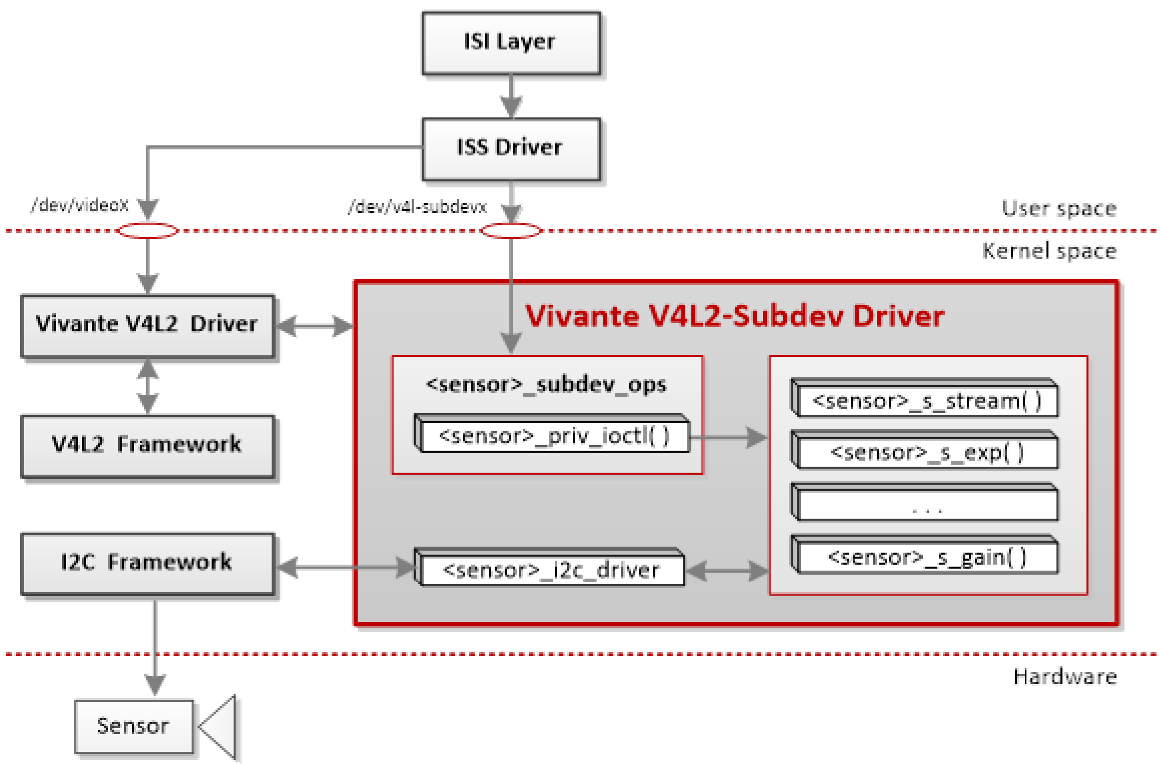


Figure 5. VVCAM Software Architecture in V4L2 Mode

2.6.2 Camera Sensor Porting Setup in V4L2 Mode

2.6.2.1 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);
…
}
```

2.6.2.2 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)
```

2.6.2.3 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

Table continues on the next page...

Table continued from the previous page...

ov2775 Structure Members	Type	Description
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
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);
…
}
```

2.6.2.4 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).

2.6.2.5 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;
```

2.6.2.6 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>;
};
```

2.6.3 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.

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

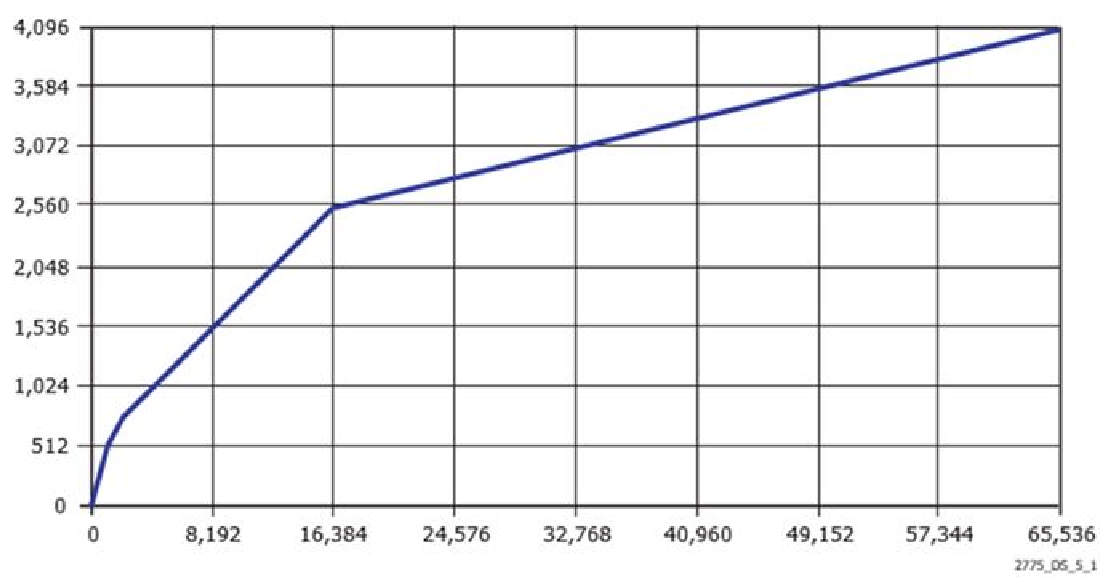
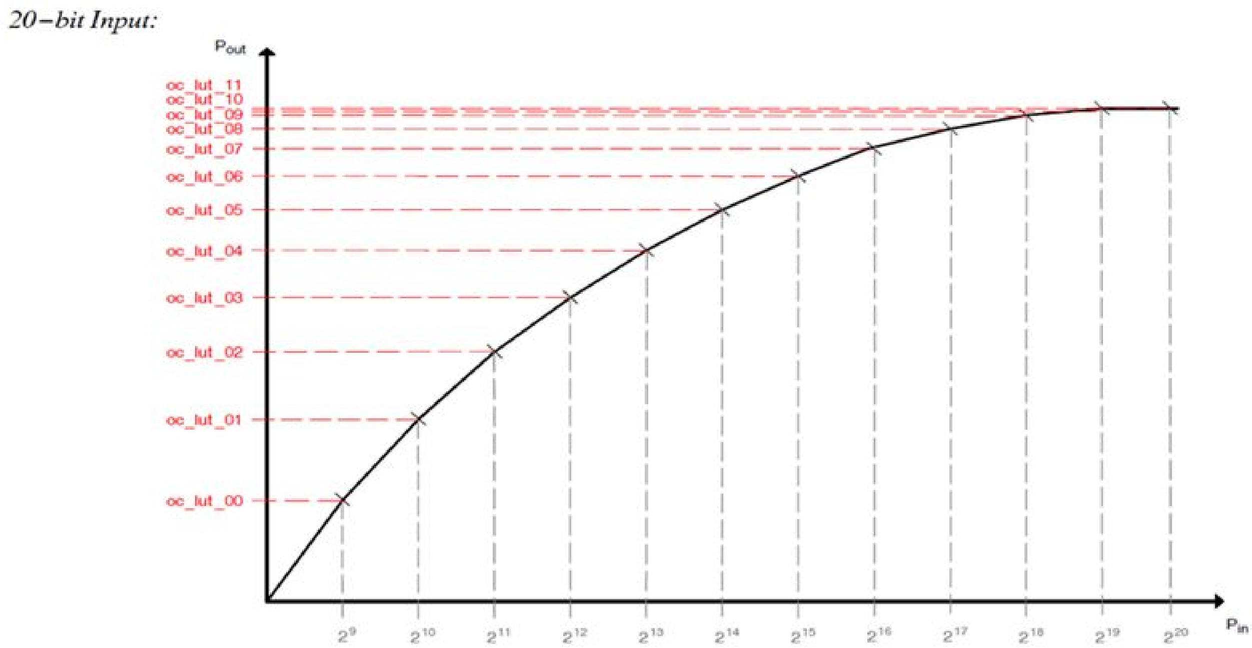


Figure 6. 16-bit to 12-bit PWL compression

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

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

```
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,
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,
},
```

2.6.4 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 3
ISP Using V4L2 Interface

3.1 Overview

This document describes the ISP software Application Programming Interface (API) using Video For Linux 2. The ISP software V4L2 API controls the ISP hardware, sensor hardware, and its calibration data from the Linux standard API. The kernel V4L2 driver handles the API commands and requests from the V4L2 user application, communicates to the ISP software stack and delivers image buffers to the V4L2 user application.

Currently, there are no deprecated functions in this API.

3.1.1 Requirements/dependencies

- Linux environment is compatible with V4L2.

3.1.2 Supported features

ISP features which are listed in Table 4 are currently supported in the ISP V4L2 API.

Table 4. ISP features

Feature	Abbreviation
Auto Focus	AF
Auto Exposure	AE
Auto White Balance	AWB
Auto Video Stabilization	AVS
Black Level Subtraction	BLS
Chromatic Aberration Correction	CAC
Color Noise Reduction	CNR
Color Processing	CPROC
Demosaic	--
Defect Pixel Cluster Correction	DPCC
De-noising Pre-filter	DPF
High Dynamic Range	HDR
Image Effect	IE
Lens Shade Correction	LSC
Noise Reduce 2D	2DNR
Noise Reduce 3D	3DNR
Wide Dynamic Range	WDR

Sensor features: Additional functionality provided in future releases.

3.2 V4L2 API components

The ISP software V4L2 API is written in ANSI C++ code and is defined in the *v4l2/hal-vivante-camera/ sub folder*. All commands are performed in the user space using an IOCTL interface which calls kernel space actions directly. The IOCTL control words are described in the IOCTL Interface and Commands.

The ISP software V4L2 API components are defined in the following sections:

- Buffer API
- Event API
- Feature control API

3.2.1 IOCTL interface and commands

V4L2 provides Input and Output Control (IOCTL) interfaces to communicate directly with device drivers. Table 5 lists key IOCTLs relevant to the ISP V4L2 software. Each IOCTL command corresponds to an operation function.

Table 5. Key video IOTCLs

IOCTL	Type	Description
VIDIOC_QUERYCAP	.vidioc_querycap	Query the capabilities of the driver, such as V4L2_CAP_STREAMING
VIDIOC_S_FMT	.vidioc_s_fmt_*	Set format information
VIDIOC_REQBUFS	.vidioc_reqbufs	Request buffers. Buffer types: DMA, MMAP, USER_PTR
VIDIOC_QBUF	.vidioc_qbuf	Enqueue buffer to kernel, then the driver fills this buffer
VIDIOC_QUERYBUF	.vidioc_querybuf	Get buffer information from the kernel and mmap
VIDIOC_DQBUF	.vidioc_dqbuf	De-queue the buffer from the kernel. User gets frame data
VIDIOC_STREAMON	.vidioc_streamon	Start stream
VIDIOC_STREAMOFF	.vidioc_streamoff	Close stream
VIDIOC_G_EXT_CTRLS	.vidioc_g_ext_ctrls	Get feature control commands
VIDIOC_S_EXT_CTRLS	.vidioc_s_ext_ctrls	Set feature control commands

3.2.2 IOCTL call flow

IOTCL call flow is described in Figure 7 and the ISP reference code is based on this implementation.

This flow will be expanded in the future.

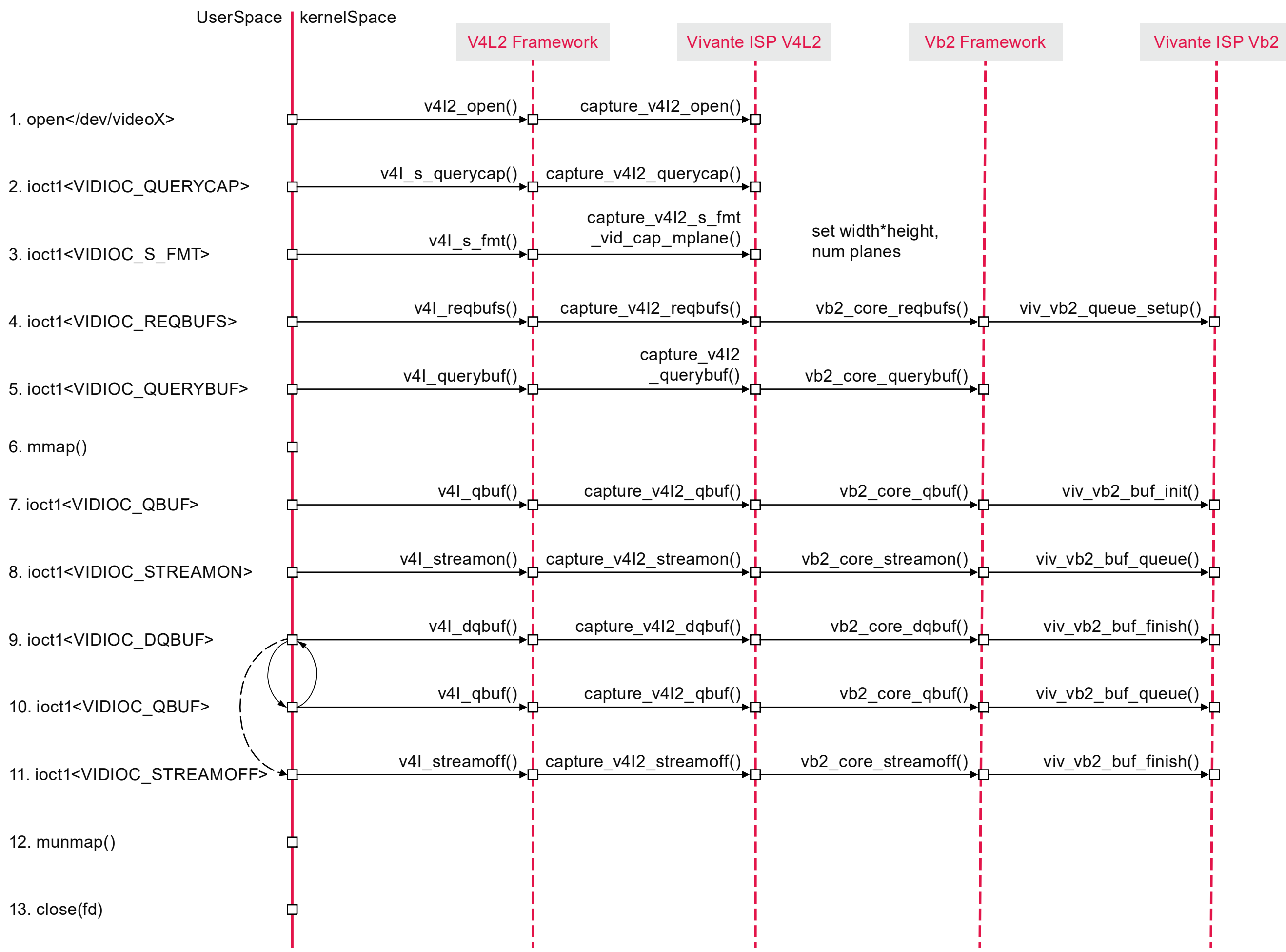


Figure 7. IOCTL call flow

3.2.3 Buffer API

A buffer contains data exchanged by the application and driver using memory mapping I/O. Only pointers to buffers are exchanged; the data itself is not copied. Memory mapping is primarily intended to map buffers in device memory into the application's address space.

The V4L2 driver supports the following buffer IOCTLs:

- VIDIOC_REQBUFS
- VIDIOC_QUERYBUF
- VIDIOC_QBUF
- VIDIOC_DQBUF
- VIDIOC_STREAMON
- VIDIOC_STREAMOFF

In addition, the following functions are supported.

- mmap()
- munmap()

- select()
- poll()

3.2.3.1 Buffer IOCTL control words

- VIDIOC_REQBUFS

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-reqbufs.html
- VIDIOC_QUERYBUF

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-querybuf.html
- VIDIOC_QBUF

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-qbuf.html
- VIDIOC_DQBUF

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-qbuf.html
- VIDIOC_STREAMON

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-streamon.html
- VIDIOC_STREAMOFF

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-streamon.html

3.2.3.2 Buffer functions

- mmap

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/func-mmap.html
- munmap

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/func-munmap.html
- poll

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/func-poll.html

3.2.4 Event API

The V4L2 event interface provides a means for a user to get notified immediately on certain conditions taking place on a device.

To receive events, first the user must subscribe to an event using the VIDIOC_SUBSCRIBE_EVENT and the VIDIOC_UNSUBSCRIBE_EVENT IOCTLs. Once an event is subscribed, the events of subscribed types are de-queueable using the VIDIOC_DQEVENT IOCTL. Events may be unsubscribed using the VIDIOC_UNSUBSCRIBE_EVENT IOCTL. The information on de-queueable events is obtained by using poll() system calls on video devices. The V4L2 events use POLLPRI events on poll system calls.

The V4L2 driver supports the following event IOCTLs:

- VIDIOC_SUBSCRIBE_EVENT
- VIDIOC_UNSUBSCRIBE_EVENT
- VIDIOC_DQEVENT

In addition, the following function is supported.

- poll()

3.2.4.1 Event IOCTL control words

- VIDIOC_SUBSCRIBE_EVENT

Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-subscribe-event.html

- VIDIOC_UNSUBSCRIBE_EVENT

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-subscribe-event.html

- VIDIOC_DQEVENT

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-dqevent.html

3.2.4.2 Event functions

- poll

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/func-poll.html

3.2.4.3 Private event

The private event is an extension based on V4L2_EVENT_PRIVATE_START and defines ID of the private event source, defines event data struct knl_v4l2_event_data based on struct v4l2_event.u.data[64].

Private event type:

- KNL_VIVCAM_V4L2_EVENT_TYPE

ID:

- KNL_VIVCAM_NOTIFY

Struct definition:

- Struct knl_v4l2_event_data, 64 bytes.

Table 6. Private event

Structure member	Type	Description
command	unsigned int	Extension based on V4L2_CID_PRIVATE_BASE
status	unsigned int	
session_id	unsigned int	
stream_id	unsigned int	
nop1	unsigned int	Reserved for future extensions
…	…	…
nop12	unsigned int	

3.2.5 Feature control API

The feature control API, uses JavaScript Object Notation (JSON) objects in user application threads and shares the objects directly with the daemon using share memory methods.

The ISP daemon sets ISPCore feature control words directly with the JSON parameters. In the user space and kernel space transfer, the Json::Value object is translated to a char string and transferred between the user and kernel space as shown in Figure 8.

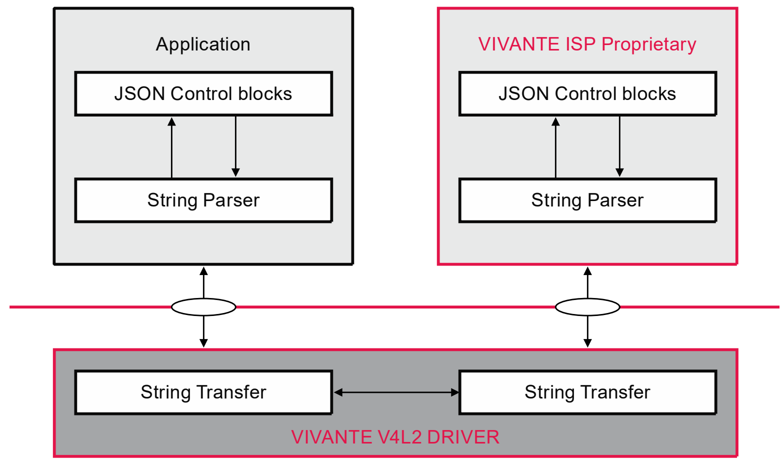


Figure 8. Feature control block diagram

3.2.5.1 String parser

The JSON format used for the APIs and the string transfer can be handled using open source code.

For example:

1. Json::Value to char string:

```
String Json::Value::toStyledString(Json::Value)
```

2. char string to Json::Value:

```
Json::CharReaderBuilder::parse(const char* beginDoc,
                                        const char* endDoc,
                                        Value& root, bool collectComments = true);
```

3.2.5.2 String transfer

All feature-related JSON-String entities are transferred using the following IOCTLs:

- VIDIOC_G_EXT_CTRLS

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-g-ext-ctrls.html

- VIDIOC_S_EXT_CTRLS

  Link: http://www.kernel.org/doc/html/v5.4/media/uapi/v4l/vidioc-g-ext-ctrls.html

For a detailed example, refer to the code *appshell/vvext/vvext.cpp*.

The char string memory block exchange using the v4l2_ext_control struct, as shown in Table 7.

Table 7. v4l2_ext_control Structure

v4l2_ext_control structure member	Type	Description
id	__u32	V4L2 ISP SW feature control words
size	__u32	String length

Table continues on the next page...

Table 7. v4l2_ext_control Structure (continued)

v4l2_ext_control structure member	Type	Description
reserved2[1]	__u32	
value	union of __s32	
value64	union of __s64	
string	union of char *	String transfer pointer
p_u8	union of __u8 *	
p_u16	union of __u16 *	
p_u32	union of __u32 *	
ptr	union of void*	

3.2.5.3 Feature control words

Interface header file: *mediacontrol/include_api/ioctl_cmds.h.*

- IF_AE_G_CFG

  This macro definition is identical to the string “ae.g.cfg”.

  Description: Gets the configuration values for the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 8. Control words for IF_AE_G_CFG

Control word	Description
mode	Configuration mode
damp.over	Damping upper limit
damp.under	Damping lower limit
set.point	Set point
clm.tolerance	Calculation accuracy

- IF_AE_S_CFG

  This macro definition is identical to the string "ae.s.cfg".

  Description: Sets the configuration values for the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 9. Control words for IF_AE_S_CFG

Control word	Description
mode	Configuration mode
damp.over	Damping upper limit
damp.under	Damping lower limit
set.point	Set point
clm.tolerance	Calculation accuracy

- IF_EC_G_CFG

  This macro definition is identical to the string "ec.g.cfg".

  Description: Gets the ECM (Exposure Control Module) values for the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 10. Control words for IF_EC_G_CFG

Control word	Description
flicker.period	The flag of Auto Exposure flicker period
Afps	Auto FPS control value

- IF_EC_S_CFG

  This macro definition is identical to the string "ec.s.cfg".

  Description: Sets the ECM (Exposure Control Module) values for the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 11. Control words for IF_EC_S_CFG

Control word	Description
flicker.period	The flag of Auto Exposure flicker period
afps	Auto FPS control value

- IF_AE_G_EN

  This macro definition is identical to the string "ae.g.en".

  Description: Gets the enabled/disabled state of the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 12. Control words for IF_AE_G_EN

Control word	Description
enable	The state of Auto Exposure

- IF_AE_S_EN

  This macro definition is identical to the string "ae.s.en".

  Description: Sets the enabled/disabled state of the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 13. Control words for IF_AE_S_EN

Control word	Description
enable	Auto Exposure is enabled

- IF_AE_RESET

  This macro definition is identical to the string "ae.reset".

  Description: Reset the Auto Exposure control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 14. Control words for IF_AE_RESET

Control word	Description
N/A	

- IF_AF_G_CFG

  This macro definition is identical to the string "af.g.cfg".

  Description: Gets the configuration of the Auto Focus control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 15. Control words for IF_AF_G_CFG

Control word	Description
algorithm	Algorithm type
oneshot	Trigger mode is one shot

- IF_AF_S_CFG

  This macro definition is identical to the string "af.s.cfg".

Description: Sets the configuration of the Auto Focus control.

Parameters:

— Json::Value &jQuery

— Json::Value &jResponse

Table 16. Control words for IF_AF_S_CFG

Control word	Description
algorithm	Algorithm type
oneshot	Trigger mode is one shot

- IF_AF_G_EN

  This macro definition is identical to the string "af.g.en".

  Description: Gets the enabled/disabled state of the Auto Focus control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 17. Control words for IF_AF_G_EN

Control word	Description
enable	The state of the Auto Focus

- IF_AF_S_EN

  This macro definition is identical to the string "af.s.en".

  Description: Sets the enabled/disabled state of the Auto Focus control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 18. Control words for IF_AF_S_EN

Control word	Description
enable	Auto Focus is enabled

- IF_AWB_G_CFG

  This macro definition is identical to the string "awb.g.cfg".

  Description: Gets the configuration of the Auto White Balance control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 19. Control Words for IF_AWB_G_CFG

Control word	Description
mode	AWB mode
index	The index of calibration data in the database
damping	Have damped data

- IF_AWB_S_CFG

  This macro definition is identical to the string "awb.s.cfg".

  Description: Sets the mode and index of the Auto White Balance control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 20. Control Words for IF_AWB_S_CFG

Control word	Description
mode	AWB mode
index	The index of calibration data in the database
damping	Damping data

- IF_AWB_G_EN

  This macro definition is identical to the string "awb.g.en".

  Description: Gets the enabled/disabled state of the Auto White Balance control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 21. Control words for IF_AWB_G_EN

Control word	Description
enable	The state of the AWB control

- IF_AWB_S_EN

  This macro definition is identical to the string "awb.s.en".

  Description: Sets the enabled/disabled state of the Auto White Balance control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 22. Control words for IF_AWB_S_EN

Control word	Description
enable	Auto White Balance is enabled

- IF_AWB_RESET

  This macro definition is identical to the string "awb.reset".

  Description: Resets the Auto White Balance control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 23. Control words for IF_AWB_RESET

Control word	Description
N/A	-

- IF_AVS_G_CFG

  This macro definition is identical to the string "avs.g.cfg".

  Description: Gets the configuration values for the Auto Video Stabilization control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 24. Control words for IF_AVS_G_CFG

Control word	Description
use.params	AVS use params
acceleration	AVS has acceleration
base.gain	AVS's base gain
fall.off	AVS has fall off
num.itp.points	The number of ITP points
theta	Theta
x	The size of width
y	The size of height

- IF_AVS_S_CFG

  This macro definition is identical to the string "avs.s.cfg".

  Description: Sets the configuration values for the Auto Video Stabilization control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 25. Control words IF_AVS_S_CFG

Control word	Description
use.params	AVS use params
acceleration	AVS has acceleration
base.gain	AVS’s base gain
fall.off	AVS has fall off
num.itp.points	The number of ITP points
theta	Theta

- IF_AVS_G_EN

  This macro definition is identical to the string "avs.g.en".

  Description: Gets the enabled/disabled state of the Auto Video Stabilization control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 26. Control words for IF_AVS_G_EN

Control word	Description
enable	The state of the AVS

- IF_AVS_S_EN

  This macro definition is identical to the string "avs.s.en".

  Description: Sets the enabled/disabled state of the Auto Video Stabilization control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 27. Control words for IF_AVS_S_EN

Control word	Description
enable	AVS is enabled

- IF_BLS_G_CFG

  This macro definition is identical to the string "bls.g.cfg".

  Description: Gets the configuration values for the Black Level Subtraction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 28. Control words for IF_BLS_G_CFG

Control word	Description
red	The red data information
green.r	The Gr data information
green.b	The Gb data information
blue	The blue data information

- IF_BLS_S_CFG

  This macro definition is identical to the string "bls.s.cfg".

  Description: Sets the configuration values for the Black Level Subtraction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 29. Control words for IF_BLS_S_CFG

Control word	Description
red	The red data information
green.r	The Gr data information
green.b	The Gb data information
blue	The blue data information

- IF_CAC_G_EN

  This macro definition is identical to the string "cac.g.en".

  Description: Gets the enabled/disabled state of the Chromatic Aberration Correction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 30. Control words for IF_CAC_G_EN

Control word	Description
enable	The state of the Chromatic Aberration Correction

- IF_CAC_S_EN

  This macro definition is identical to the string "cac.s.en".

  Description: Sets the enabled/disabled state of the Chromatic Aberration Correction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 31. Control words for IF_CAC_S_EN

Control word	Description
enable	Chromatic Aberration Correction is enabled

- IF_CNR_G_CFG

  This macro definition is identical to the string "cnr.g.cfg".

  Description: Gets the configuration values for the Chroma Noise Reduction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 32. Control words for IF_CNR_G_CFG

Control word	Description
tc1	tc1
tc2	tc2

- IF_CNR_S_CFG

  This macro definition is identical to the string "cnr.s.cfg".

  Description: Sets the configuration values for the Chroma Noise Reduction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 33. Control words for IF_CNR_S_CFG

Control word	Description
tc1	tc1
tc2	tc2

- IF_CPROC_G_CFG

  This macro definition is identical to the string "cproc.g.cfg".

  Description: Gets the configuration values for the Color Processing control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 34. Control words for IF_CPROC_G_CFG

Control word	Description
brightness	Brightness value
chroma.out	CPROC chrominance pixel clipping range at output

Table continues on the next page...

Table 34. Control words for IF_CPROC_G_CFG (continued)

Control word	Description
contrast	Contrast value
hue	Hue value
luma.in	CPROC luminance input range (offset processing)
luma.out	CPROC luminance output clipping range
saturation	Saturation value

- IF_CPROC_S_CFG

  This macro definition is identical to the string "cproc.s.cfg".

  Description: Sets the configuration values for the Color Processing control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 35. Control words for IF_CPROC_S_CFG

Control word	Description
brightness	Brightness value
chroma.out	CPROC chrominance pixel clipping range at output
contrast	Contrast value
hue	Hue value
luma.in	CPROC luminance input range(offset processing)
luma.out	CPROC luminance output clipping range
saturation	Saturation value

- IF_CPROC_G_EN

  This macro definition is identical to the string "cproc.g.en".

  Description: Gets the enabled/disabled state of the Color Processing control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 36. Control words for IF_CPROC_G_EN

Control word	Description
enable	The state of the CPROC

- IF_CPROC_S_EN

  This macro definition is identical to the string "cproc.s.en".

  Description: Sets the enabled/disabled state of the Color Processing control.

Parameters:

— Json::Value &jQuery

— Json::Value &jResponse

Table 37. Control words for IF_CPROC_S_EN

Control word	Description
enable	CPROC is enabled

- IF_DEMOSAIC_G_CFG

  This macro definition is identical to the string "dmsc.g.cfg".

  Description: Gets the configuration values for the Demosaic control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 38. Control words for IF_DEMOSAIC_G_CFG

Control word	Description
mode	Demosaic mode
threshold	Demosaic threshold

- IF_DEMOSAIC_S_CFG

  This macro definition is identical to the string "dmsc.s.cfg".

  Description: Sets the configuration values for the Demosaic control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 39. Control words for IF_DEMOSAIC_S_CFG

Control word	Description
mode	Demosaic mode
threshold	Demosaic threshold

- IF_DEMOSAIC_G_EN

  This macro definition is identical to the string "demosaic.g.en".

  Description: Gets the enabled/disabled state of the Demosaic control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 40. Control words for IF_DEMOSAIC_G_EN

Control word	Description
enable	The state of the Demosaic control

- IF_DEMOSAIC_S_EN

  This macro definition is identical to the string "demosaic.s.en".

  Description: Sets the enabled/disabled state of the Demosaic control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 41. Control words for IF_DEMOSAIC_S_EN

Control word	Description
enable	Demosaic is enabled

- IF_DPCC_G_EN

  This macro definition is identical to the string "dpcc.g.en".

  Description: Gets the enabled/disabled state of the Defect Pixel Cluster Correction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 42. Control words for IF_DPCC_G_EN

Control word	Description
enable	The state of the Defect Pixel Cluster Correction

- IF_DPCC_S_EN

  This macro definition is identical to the string "dpcc.s.en".

  Description: Sets the enabled/disabled state of the Defect Pixel Cluster Correction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 43. Control words for IF_DPCC_S_EN

Control word	Description
enable	Defect Pixel Cluster Correction is enabled

- IF_DPF_G_CFG

  This macro definition is identical to the string "dpf.g.cfg".

  Description: Gets the configuration values for the De-noising Pre-Filter control.

Parameters:

— Json::Value &jQuery

— Json::Value &jResponse

Table 44. Control words for IF_DPF_G_CFG

Control word	Description
gradient	Gradient value for dynamic strength calculation
offset	Offset value for dynamic strength calculation
min	Upper bound for dynamic strength calculation
div	Division factor for dynamic strength calculation
sigma.green	Sigma value for green pixel
sigma.red.blue	Sigma value for red/blue pixel

- IF_DPF_S_CFG

  This macro definition is identical to the string "dpf.s.cfg".

  Description: Sets the configuration values for the De-noising Pre-Filter control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 45. Control words for IF_DPF_S_CFG

Control word	Description
gradient	Gradient value for dynamic strength calculation
offset	Offset value for dynamic strength calculation
min	Upper bound for dynamic strength calculation
div	Division factor for dynamic strength calculation
sigma.green	Sigma value for green pixel
sigma.red.blue	Sigma value for red/blue pixel

- IF_DPF_G_EN

  This macro definition is identical to the string "dpf.g.en".

  Description: Gets the enabled/disabled state of the De-noising Pre-Filter control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 46. Control words for IF_DPF_G_EN

Control word	Description
enable	The state of the De-noising Pre-Filter

- IF_DPF_S_EN

  This macro definition is identical to the string "dpf.s.en".

  Description: Sets the enabled/disabled state of the De-noising Pre-Filter control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 47. Control words for IF_DPF_S_EN

Control word	Description
enable	De-noising Pre-Filter is enabled

- IF_EC_G_CFG

  This macro definition is identical to the string "ec.g.cfg".

  Description: Gets the configuration values for the Exposure Control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 48. Control words for IF_EC_G_CFG

Control word	Description
gain	Exposure gain
time	Exposure time

- IF_EC_S_CFG

  This macro definition is identical to the string "ec.s.cfg".

  Description: Sets the configuration values for the Exposure Control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 49. Control words for IF_EC_S_CFG

Control word	Description
gain	Exposure gain
time	Exposure time

- IF_EE_G_CFG

  This macro definition is identical to the string "ee.g.cfg".

  Description: Gets the configuration values for the Edge Enhancement control.

  Parameters:

  — Json::Value &jQuery

— Json::Value &jResponse

Table 50. Control words for IF_EE_G_CFG

Control word	Description
strength	Strength
sharpen	Sharpen
depurple	Depurple

- IF_EE_S_CFG

  This macro definition is identical to the string "ee.s.cfg".

  Description: Sets the configuration values for the Edge Enhancement control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 51. Control words for IF_EE_S_CFG

Control word	Description
strength	Strength
sharpen	Sharpen
depurple	Depurple

- IF_EE_G_EN

  This macro definition is identical to the string "ee.g.en".

  Description: Gets the enabled/disabled state of the Edge Enhancement control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 52. Control words for IF_EE_G_EN

Control word	Description
enable	The state of the Edge Enhancement control

- IF_EE_S_EN

  This macro definition is identical to the string "ee.s.en".

  Description: Sets the enabled/disabled state of the Edge Enhancement control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 53. Control words for IF_EE_S_EN

Control word	Description
enable	Edge Enhancement is enabled

- IF_EE_RESET

  This macro definition is identical to the string "ee.reset".

  Description: Resets the Edge Enhancement control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 54. Control words for IF_EE_RESET

Control word	Description
N/A	-

- IF_GC_G_CURVE

  This macro definition is identical to the string "gc.g.curve".

  Description: Gets the configuration values for the Gamma control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 55. Control words for IF_GC_G_CURVE

Control word	Description
curve	Gamma curve

- IF_GC_S_CURVE

  This macro definition is identical to the string "gc.s.curve".

  Description: Sets the configuration values for the Gamma Control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 56. Control words for IF_GC_S_CURVE

Control word	Description
curve	Gamma curve

- IF_GC_G_EN

  This macro definition is identical to the string "gc.g.en".

  Description: Gets the enabled/disabled state of the Gamma Control.

Parameters:

- — Json::Value &jQuery
- — Json::Value &jResponse

Table 57. Control words for IF_GC_G_EN

Control word	Description
enable	The state of the Gamma Control

- IF_GC_S_EN

  This macro definition is identical to the string "gc.s.en".

  Description: Sets the enabled/disabled state of the Gamma Control.

  Parameters:

  - — Json::Value &jQuery
  - — Json::Value &jResponse

Table 58. Control words for IF_GC_S_EN

Control word	Description
enable	Gamma Control is enabled

- IF_HDR_G_CFG

  This macro definition is identical to the string "hdr.g.cfg".

  Description: Gets the configuration of the High Dynamic Range control.

  Parameters:

  - — Json::Value &jQuery
  - — Json::Value &jResponse

Table 59. Control words for IF_HDR_G_CFG

Control word	Description
extension.bit	Extension bit
range.start.value	Range start value
very.short.weight	Very short weight

- IF_HDR_S_CFG

  This macro definition is identical to the string "hdr.s.cfg".

  Description: Sets the configuration of the High Dynamic Range control.

  Parameters:

  - — Json::Value &jQuery
  - — Json::Value &jResponse

Table 60. Control words for IF_HDR_S_CFG

Control word	Description
extension.bit	Extension bit
range.start.value	Range start value
very.short.weight	Very short weight

- IF_HDR_G_EN

  This macro definition is identical to the string "hdr.g.en".

  Description: Gets the enabled/disabled state of the High Dynamic Range control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 61. Control words for IF_HDR_G_EN

Control word	Description
enable	The state of High Dynamic Range

- IF_HDR_S_EN

  This macro definition is identical to the string "hdr.s.en".

  Description: Sets the enabled/disabled state of the High Dynamic Range control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 62. Control words for IF_HDR_S_EN

Control word	Description
enable	High Dynamic Range is enabled

- IF_HDR_RESET

  This macro definition is identical to the string "hdr.reset".

  Description: Resets the High Dynamic Range control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 63. Control words for IF_HDR_RESET

Control word	Description
N/A	-

- IF_IE_G_CFG

This macro definition is identical to the string "ie.g.cfg".

Description: Gets the configuration values for the Image Effects control.

Parameters:

— Json::Value &jQuery

— Json::Value &jResponse

Table 64. Control words for IF_IE_G_CFG

Control word	Description
Mode	The image can use seven available effect modes
Range	Image Effects configuration range
Config	Image Effects configuration
Tint.cb	Sepia Tint Cb of sepia mode
Tint.cr	Sepia Tint Cr of sepia mode
selection	Color selection of color mode
threshold	Color threshold of color mode
emboss:coeff	Coefficient of emboss mode
sketch:coeff	Coefficient of sketch mode
sharpen:factor	Factor of sharpen mode
sharpen:threshold	Threshold of sharpen mode

- IF_IE_S_CFG

  This macro definition is identical to the string "ie.s.cfg".

  Description: Sets the configuration values for the Image Effects control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 65. Control words for IF_IE_S_CFG

Control word	Description
Mode	The image can use seven available effect modes
Range	Image Effects configuration range
Config	Image Effects configuration
Tint.cb	Sepia Tint Cb of sepia mode
Tint.cr	Sepia Tint Cr of sepia mode
selection	Color selection of color mode
threshold	Color threshold of color mode
emboss:coeff	Coefficient of emboss mode

Table continues on the next page...

Table 65. Control words for IF_IE_S_CFG (continued)

Control word	Description
sketch:coeff	Coefficient of sketch mode
sharpen:factor	Factor of sharpen mode
sharpen:threshold	Threshold of sharpen mode
sharpen:coeff	Coefficient of sharpen mode

- IF_IE_G_EN

  This macro definition is identical to the string "ie.g.en".

  Description: Gets the enabled/disabled state of the Image Effects control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 66. Control words for IF_IE_G_EN

Control word	Description
Enable	The state of the Image Effects control

- IF_IE_S_EN

  This macro definition is identical to the string "ie.s.en".

  Description: Sets the enabled/disabled state of the Image Effects control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 67. Control words for IF_IE_S_EN

Control word	Description
Enable	Image Effects is enabled

- IF_LSC_G_EN

  This macro definition is identical to the string "lsc.g.en".

  Description: Gets the enabled/disabled state of the Lens Shade Correction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 68. Control words for IF_LSC_G_EN

Control word	Description
Enable	The state of Lens Shade Correction

- IF_LSC_S_EN

  This macro definition is identical to the string "lsc.s.en".

  Description: Sets the enabled/disabled state of the Lens Shade Correction control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 69. Control words for IF_LSC_S_EN

Control word	Description
Enable	Lens Shade Correction is enabled

- IF_2DNR_G_CFG

  This macro definition is identical to the string "2dnr.g.cfg".

  Description: Gets the configuration values for the 2DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 70. Control words for IF_2DNR_G_CFG

Control word	Description
Generation	NR2D generation
Strength	Configuration strength
sigma	Sigma strength

- IF_2DNR_S_CFG

  This macro definition is identical to the string "2dnr.s.cfg".

  Description: Sets the configuration values for the 2DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 71. Control words for IF_2DNR_S_CFG

Control word	Description
Generation	NR2D generation
Strength	Configuration strength
sigma	Sigma strength

- IF_2DNR_G_EN

  This macro definition is identical to the string "2dnr.g.en".

  Description: Gets the enabled/disabled state of the 2DNR control.

Parameters:

— Json::Value &jQuery

— Json::Value &jResponse

Table 72. Control words for IF_2DNR_G_EN

Control word	Description
Enable	The state of NR2D

- IF_2DNR_S_EN

  This macro definition is identical to the string "2dnr.s.en".

  Description: Sets the enabled/disabled state of the 2DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 73. Control words for IF_2DNR_S_EN

Control word	Description
Enable	NR2D is enabled

- IF_2DNR_RESET

  This macro definition is identical to the string "2dnr.reset".

  Description: Resets the 2DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 74. Control words for IF_2DNR_RESET

Control word	Description
Generation	NR2D generation

- IF_3DNR_G_CFG

  This macro definition is identical to the string "3dnr.g.cfg".

  Description: Gets the configuration values for the 3DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 75. Control words for IF_3DNR_G_CFG

Control word	Description
Generation	NR3D generation

Table continues on the next page...

Table 75. Control words for IF_3DNR_G_CFG (continued)

Control word	Description
Strength	NR3D strength
spatial.denoise	Spatial denoise
temporal.denoise	Temporal denoise

- IF_3DNR_S_CFG

  This macro definition is identical to the string "3dnr.s.cfg".

  Description: Sets the configuration values for the 3DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 76. Control words for IF_3DNR_S_CFG

Control word	Description
Generation	NR3D generation
Strength	NR3D strength
spatial.denoise	Spatial denoise
temporal.denoise	Temporal denoise

- IF_3DNR_G_EN

  This macro definition is identical to the string "3dnr.g.en".

  Description: Gets the enabled/disabled state of the 3DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 77. Control words for IF_3DNR_G_EN

Control word	Description
enable	The state of NR3D

- IF_3DNR_S_EN

  This macro definition is identical to the string "3dnr.s.en".

  Description: Sets the enabled/disabled state of the 3DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 78. Control words for IF_3DNR_S_EN

Control word	Description
Enable	NR3D is enabled

- IF_3DNR_RESET

  This macro definition is identical to the string "3dnr.reset".

  Description: Resets the 3DNR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 79. Control words for IF_3DNR_RESET

Control word	Description
generation	NR3D generation

- IF_WDR_G_CFG

  This macro definition is identical to the string "wdr.g.cfg".

  Description: Gets the configuration values for the WDR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 80. Control words for IF_WDR_G_CFG

Control word	Description
Generation	WDR generation
Curve:d.y	WDR1 curve Dy
Curve:y.m	WDR1 curve Ym
strength	WDR strength
gain.max	WDR3 gain max
strength.global	WDR3 strength global

- IF_WDR_S_CFG

  This macro definition is identical to the string "wdr.s.cfg".

  Description: Sets the configuration values for the WDR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 81. Control words for IF_WDR_S_CFG

Control word	Description
Generation	WDR generation
Curve:d.y	WDR1 curve Dy
Curve:y.m	WDR1 curve Ym
strength	WDR strength
gain.max	WDR3 gain max
strength.global	WDR3 strength global

- IF_WDR_G_EN

  This macro definition is identical to the string "wdr.g.en".

  Description: Gets the enabled/disabled state of the WDR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 82. Control words for IF_WDR_G_EN

Control word	Description
Enable	The state of WDR

- IF_WDR_S_EN

  This macro definition is identical to the string "wdr.s.en".

  Description: Sets the enabled/disabled state of the WDR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 83. Control words for IF_WDR_S_EN

Control word	Description
Enable	WDR is enabled

- IF_WDR_RESET

  This macro definition is identical to the string "wdr.reset".

  Description: Resets the WDR control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 84. Control words for IF_WDR_RESET

Control word	Description
generation	WDR generation

- IF_WB_G_CFG

  Description: Gets the configuration values for the WB control.

  This macro definition is identical to the string "wb.g.cfg".

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 85. Control words for IF_WB_G_CFG

Control word	Description
Matrix	Matrix
offset	Offset
red	Cc offset red
green	Cc offset Green
blue	Cc offset blue
green.r	WB gains green.R
green.b	WB gains green.B
wb.gains	WB gains

- IF_WB_S_CFG

  This macro definition is identical to the string "wb.s.cfg".

  Description: Sets the configuration values for the WB control.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 86. Control words for IF_WB_S_CFG

Control word	Description
Matrix	Matrix
offset	Offset
red	Cc offset Red
green	Cc offset Green
blue	Cc offset Blue
green.r	WB gains Green.R

Table continues on the next page...

Table 86. Control words for IF_WB_S_CFG (continued)

Control word	Description
green.b	WB gains Green.B
wb.gains	WB gains

3.2.5.4 Dewarp control words

NOTE
Requires hardware with dewarp capability.

- IF_DWE_S_PARAMS

  This macro definition is identical to the string "dwe.s.params".

  Description: Sets the dewarp parameters, such as input format, output format, ROI, scale, split, dewarp type, etc.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 87. Control words for IF_DWE_S_PARAMS

Control word	Description
node	Dewarp parameters, including input format, output format, ROI, scale, split, dewarp type.

- IF_DWE_S_HFLIP

  This macro definition is identical to the string "dwe.s.hflip".

  Description: Sets the image horizontal flip parameters.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 88. Control words for IF_DWE_S_HFLIP

Control word	Description
port	Select flip port
hflip	Set vertical flip

- IF_DWE_S_VFLIP

  This macro definition is identical to the string "dwe.s.vflip".

  Description: Sets the image vertical flip parameters.

  Parameters:

  — Json::Value &jQuery

  — Json::Value &jResponse

Table 89. Control words for IF_DWE_S_VFLIP

Control word	Description
port	Select flip port
vflip	Set horizontal flip

3.3 ISP software V4L2 programming overview

3.3.1 General concept

The high-level diagram of the ISP V4L2 software stack is shown in Figure 9.

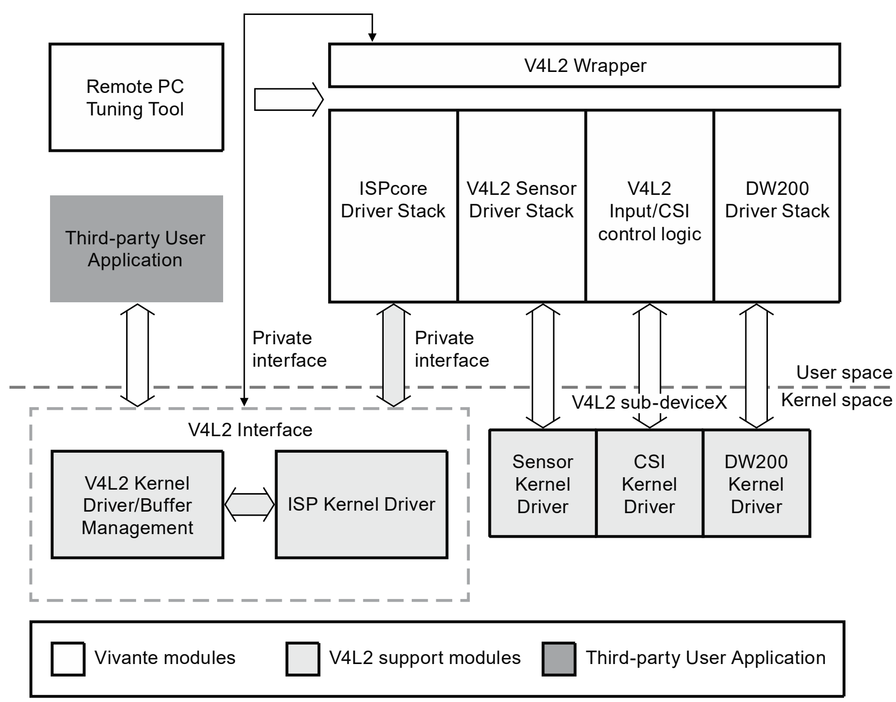


Figure 9. ISP software V4L2 stack

3.3.2 V4L2 kernel driver block diagram

ISP provides some device nodes in its file structure. Customers can operate the corresponding device through the appropriate device node(s).

Table 90. ISP device nodes

Device node/driver	Description
/dev/mediax	Enumerate video devices and subdevs
/dev/videox	Manage stream related operations and events, such as enqueue/dequeue buffers and enqueue/dequeue events
/dev/v4l2-subdevx	Manage buffers, and Control camera relevant hardware, such as MIPI/Sensor

Table continues on the next page...

Table 90. ISP device nodes (continued)

Device node/driver	Description
/dev/xx	Private interface control and dispatch the commands, events, and so on.
V4L2 kernel driver	Register the V4L2_device and video_device and implement the operational functions in the video_device and vb2_queue
ISP kernel driver	ISP kernel driver, implements read/write registers, and so on.

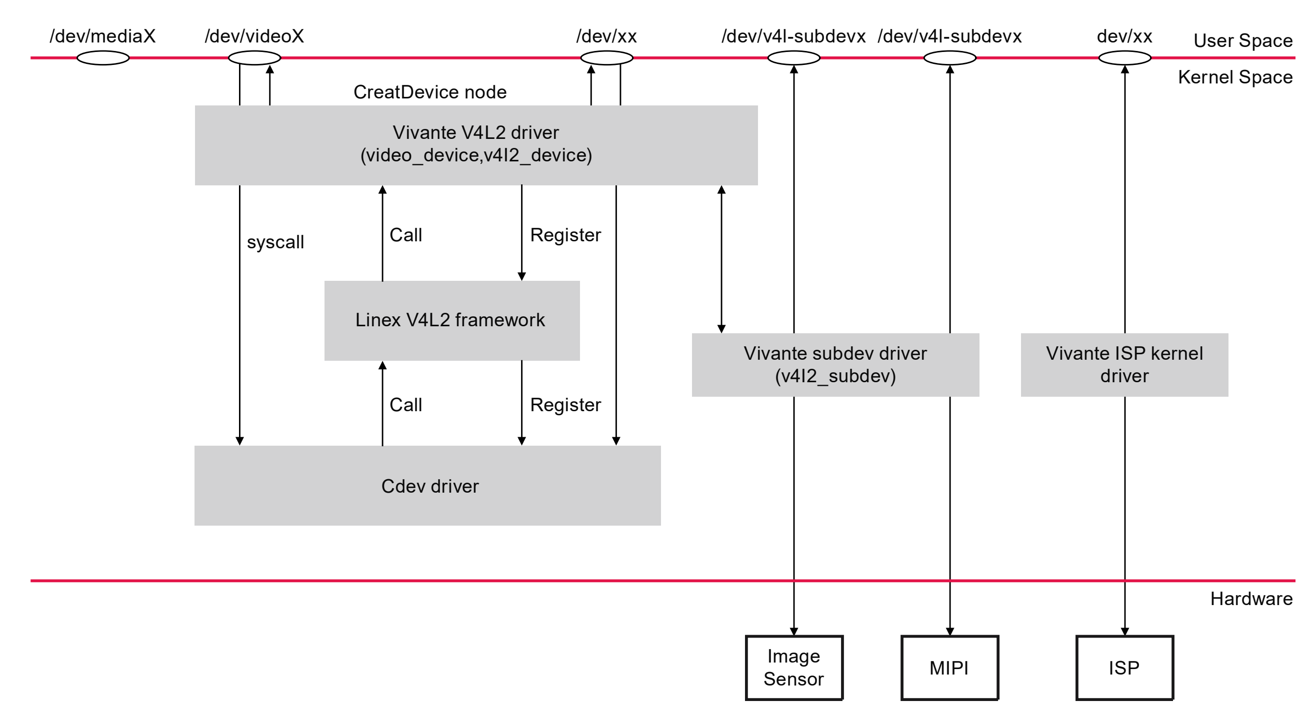


Figure 10. V4L2 kernel driver block diagram

3.3.3 V4L2 third-party user application and ISP stack communication

The V4L2 third-party user application communicates directly with the kernel with V4L2 standard control words and V4L2 extension control commands. All the user application controls pass to the kernel space to the V4L2 kernel driver.

The V4L2 kernel driver handles the API commands and requests from the V4L2 user application, communicates to the ISP software stack and delivers image buffers to the V4L2 user application.

Sub-modules that handle the event and buffer:

- Event Queue: send/get events to/from ISP proprietary software.
- Buffer Queue: manages the vb2 buffer.

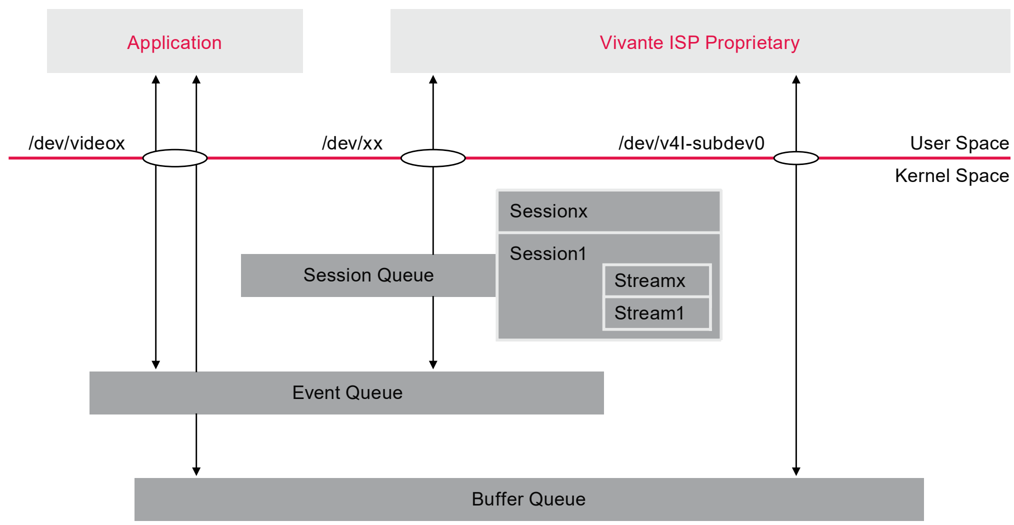


Figure 11. User application communication to ISP proprietary software stack

3.3.4 ISP V4L2 buffer management

There are three memory types as described in Table 91 and Figure 12.

Table 91. Memory types and buffer allocation

Memory type	Buffer allocation	Behavior
USERPTR	user space	User space and kernel space share the memory by buffer pointer
MMAP	kernel space	User space calls mmap to get pointer from kernel space
DMABUF	kernel space	User space gets the buffers using a file descriptor

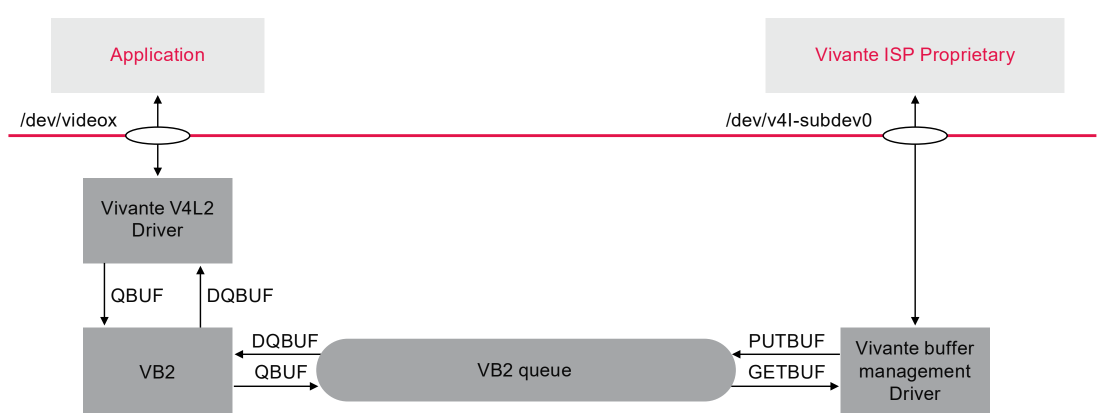


Figure 12. ISP V4L2 buffer management

NOTE

USERPTR mode is not supported.

3.3.5 ISP proprietary software stack

The camera manager receives messages for the kernel and dispatches these events to the corresponding sub-module for processing.

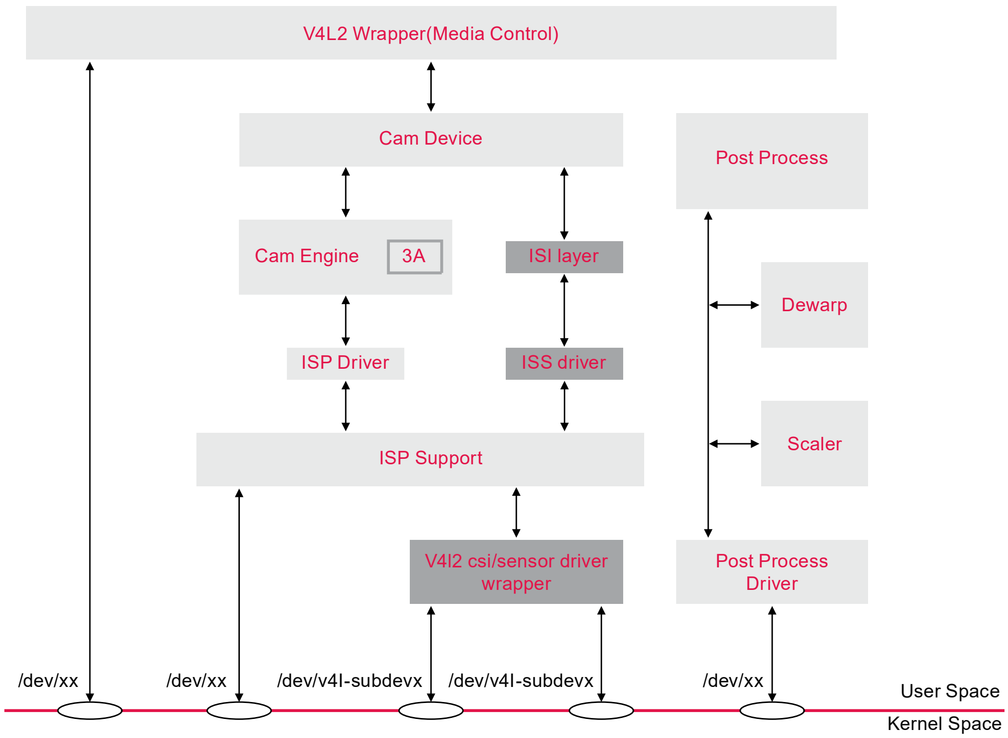


Figure 13. ISP proprietary software stack

Chapter 4
Revision History

This table provides the revision history.

Table 92. Revision history

Revision number	Date	Substantive changes
L5.4.70_2.3.2	05/2021	Initial release
LF5.10.35_2.0.0	06/2021	Released for LF5.10.35_2.0.0

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 30 June 2021

Document identifier: IMX8MPCDUG