Sensor Debugging Guide

Version: 1.1.3

Release date: 2021-12-28

Contents

Revision History

Revision	Date	Description
1.0	2019/10/12	First draft.
1.1.0	2021/10/1	Supplemented practical operation details.
1.1.2	2021/12/28	Added sensor_test.
1.1.3	2023/04/13	Revise details and update the latest content
1.1.4	2023/04/21	Add alios-related adaptations

1 Disclaimer

Contact Us

Address Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Building T10, UpperCoast Park, Huizhanwan, Zhancheng Community, Fuhai Street, Baoan District, Shenzhen, 518100, China

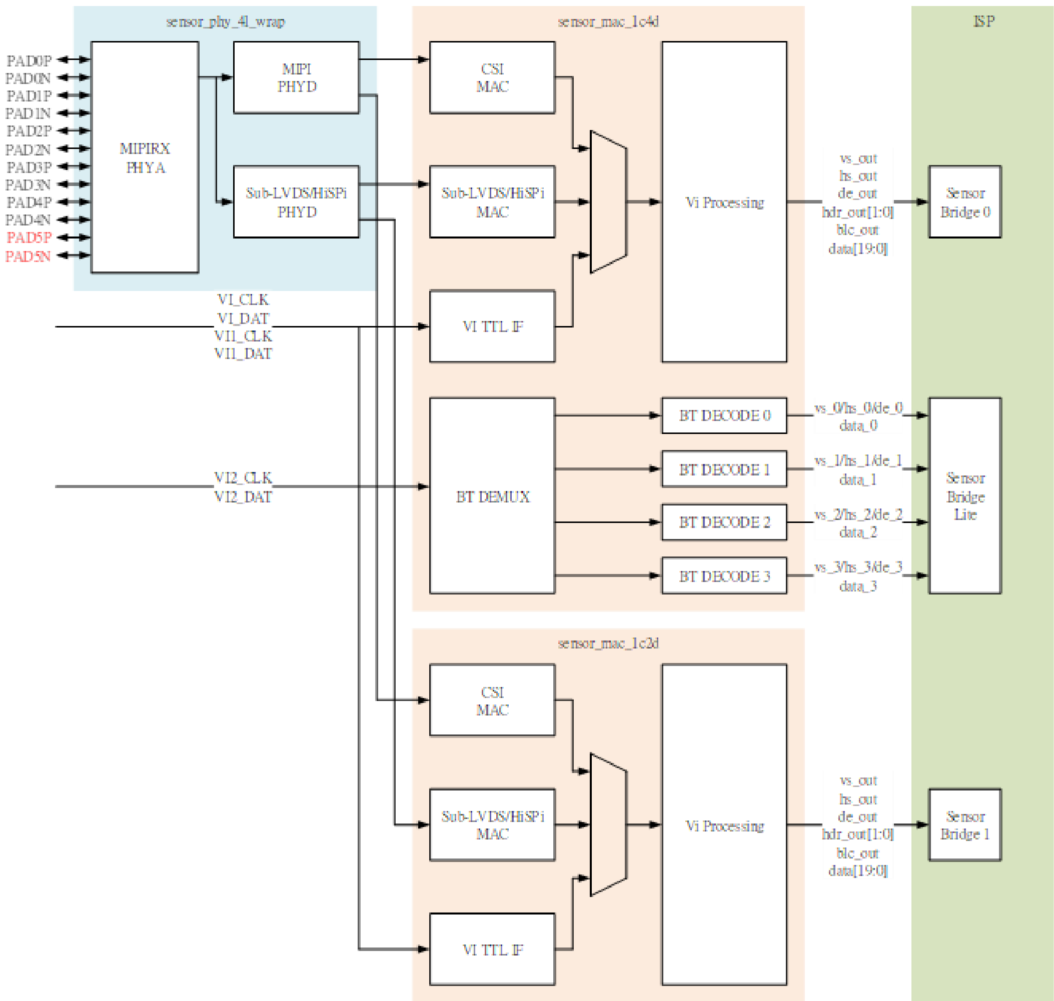
Phone +86-10-57590723 +86-10-57590724

Website https://www.sophgo.com/

Forum https://developer.sophgo.com/forum/index.html

2 Introduction to Sensor Drivers

2.1 Hardware Architecture



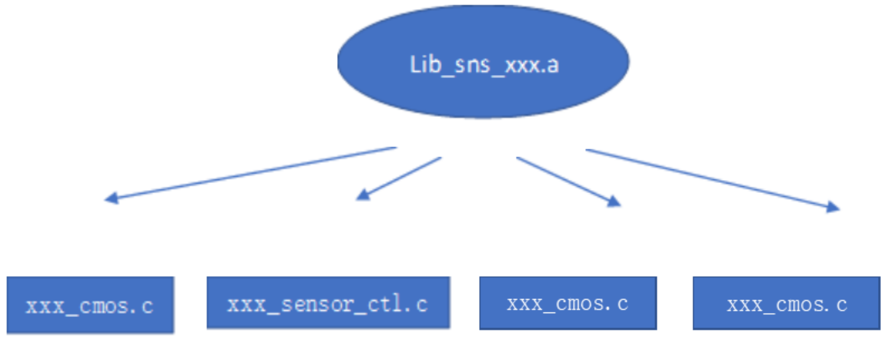
The data flow is roughly as follows: Sensor -> PHYA -> PHYD -> MAC (CSI/sub-LVDS/TTL) -> ISP’s CSI BDG.

The Sensor outputs differential signals on the lane bus, which is received and assembled by PHYA. The signal is then converted into pixel digital signals by PHYD, and the frame data is combined with the MAC clk sync, processed by VI, and then sent to the ISP for further processing.

2.2 Sensor Library Structure

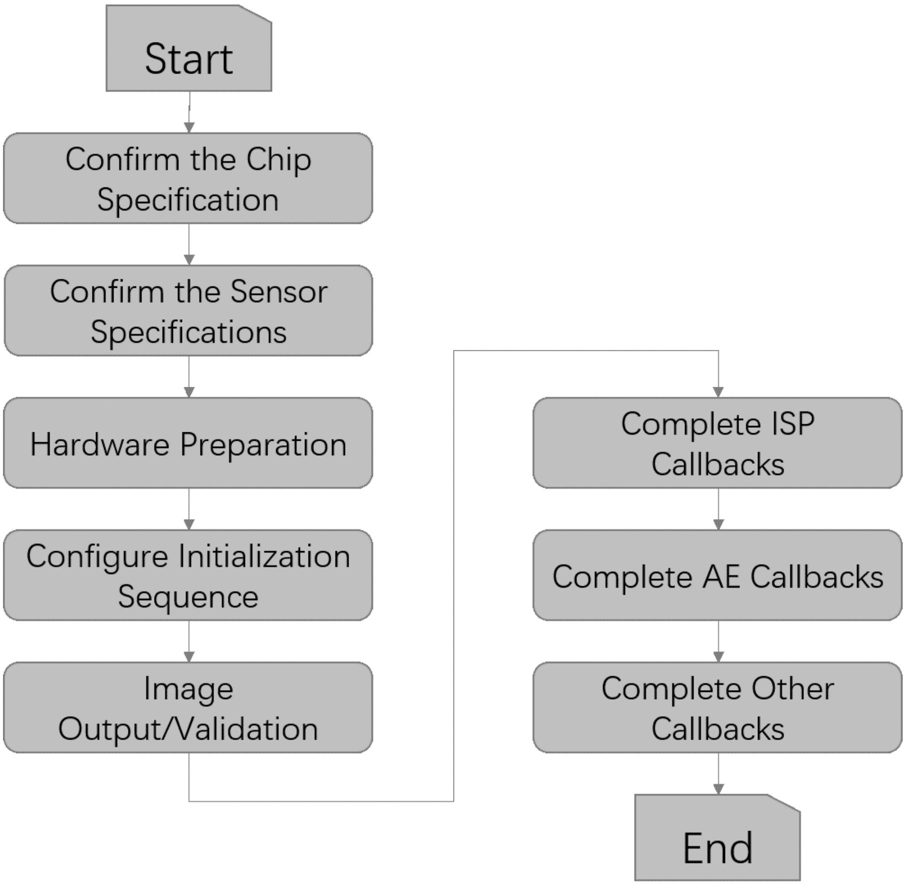
The structure of the Sensor library is shown in the following diagram, which generally includes 4 files: xxx_cmos.c, xxx_sensor_ctl.c, xxx_cmos_param.h, and xxx_cmos_ex.h.

In alios, the sensor library is located in mars_alios/components/cvi_mmf_sdk/cvi_sensor/



- xxx_cmos.c contains the main functional functions of the Sensor driver, which implements the AE control related functions, ISP default configuration, Sensor startup mode selection function, Sensor registration and deregistration functions to AE, AWB, ISP, and SnsxxxObj.
- xxx_cmos.c contains the main functional functions of the Sensor driver, which implements the AE control related functions, ISP default configuration, Sensor startup mode selection function, Sensor registration and deregistration functions to AE, AWB, ISP, and SnsxxxObj.
- xxx_sensor_ctl.c mainly includes the initialization sequence of the Sensor, communication interface initialization, and implementation of read and write functions.
- xxx_cmos_ex.h is a header file that declares the definitions of some structures, resolutions, mode types, and so on.
- xxx_cmos_param.h mainly includes the configuration of sensor property parameters, mipi property parameters, and isp noise profiles.

2.3 Debugging Process



3 Confirm Specifications

3.1 Confirm Main Processor Specifications

- Supported upper limit of Combo PHY input frequency.
- Supported Combo PHY lane configuration.
- Supported linear/WDR interface modes.
- Supported I2C bus number.
- Supported output reference clock.

For example, cv181x supports the following:

- 1C4D（1clk lane，4data lane）
- 2.5Gbps/lane
- RAW(8/10/12)+YUV422(8/10)
- 2-frame HDR (180X no support WDR)
- Support lane/pn swap
- I20-I2C3
- 200 – 600M MAC clock:

```
enum rx_mac_clk_e {
	RX_MAC_CLK_400M = 0,
	RX_MAC_CLK_600M,
	RX_MAC_CLK_200M,
	RX_MAC_CLK_BUTT,
};
```

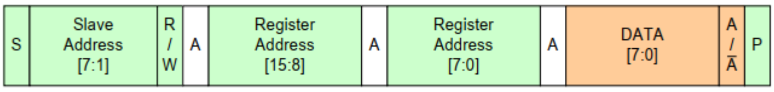
Mclk reference clock:

```
enum cam_pll_freq_e {
	CAMPLL_FREQ_NONE = 0,
	CAMPLL_FREQ_37P125M,
	CAMPLL_FREQ_25M,
	CAMPLL_FREQ_27M,
	CAMPLL_FREQ_NUM
};
```

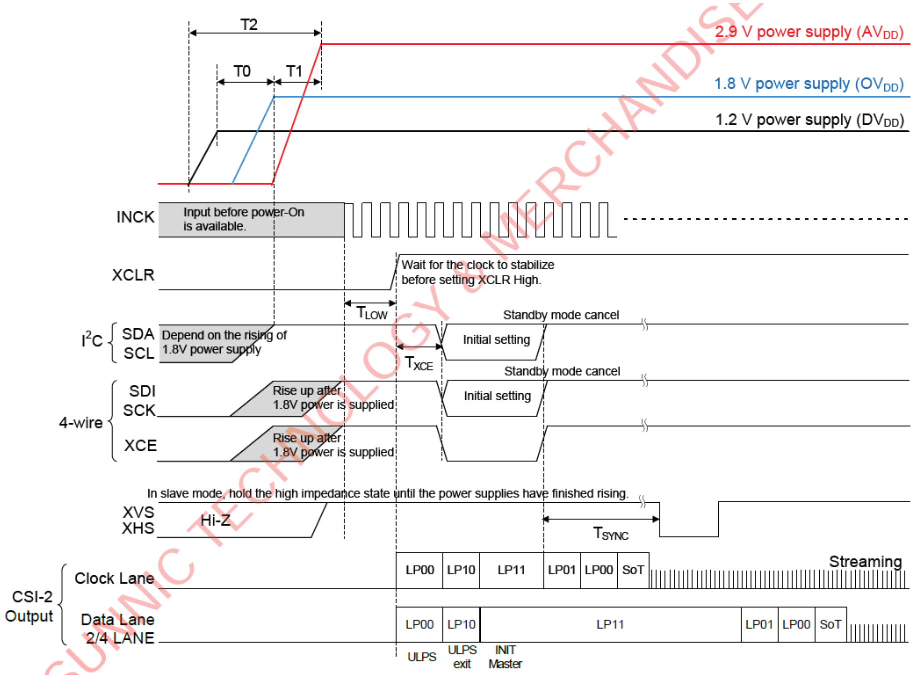
3.2 Confirm Sensor Specifications

- Confirm Sensor Control Interface (I2C/SPI).

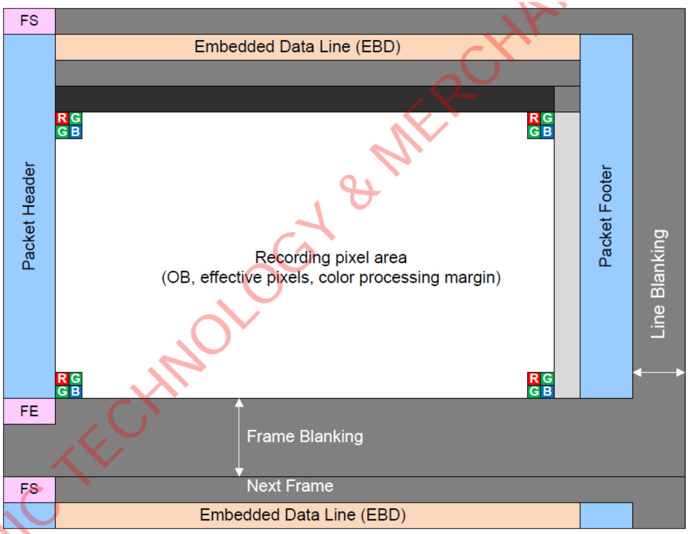
I^2C serial communication supports a 16-bit register address and 8-bit data message type.



- Confirm Sensor Power-on Sequence.



- Confirm sensor input reference clock.
- Confirm Bayer pattern and pixel code width.



- Confirm the image transfer interface mode and output frequency for linear/WDR mode.

Image Data Output Format

All-pixel scan mode

List of Setting Register

Address	bit	Register Name	Initial Value	CSI-2 serial								Remarks
				4 lane 30 / 25 [frame /s]	8 lane 30 / 25 [frame /s]	4lane 60 / 50 [frame /s]	8lane 60 / 50 [frame /s]	4 lane 30 / 25 [frame /s]	8 lane 30 / 25 [frame /s]	4lane 60 / 50 [frame /s]	8lane 60 / 50 [frame /s]	
AD Conversion [bit]				10	10	10	10	12	12	12	12	
Output bit width [bit]				10	10	10	10	12	12	12	12	
Data rate [Mbps/lane]				891/1188	891/1188	1782	891/1188	891/1188	891/1188	1782	891/1188	
3018h	[3:0]	WINMODE	0h	0h								
3030h	[7:0]	VMAX	08CAh	08CAh								25 / 30 / 50 / 60 [frame/s]
3031h	[7:0]											
3032h	[3:0]											
3034h	[7:0]	HMAX	0226h	044Ch / 0528h	044Ch / 0528h	0226h / 0294h	0226h / 0294h	044Ch / 0528h	044Ch / 0528h	0226h / 0294h	0226h / 0294h	30 / 25 [frame / s] / 60 / 50 [frame / s]
3035h	[7:0]											

- Confirm how to set exposure time and gain for linear/WDR mode.
- Confirm how to modify frame rate for linear/WDR mode.
- Confirm the sync code when the interface is subLVDS/HiSPi.
- Request Sensor Initialize Settings from the sensor manufacturer.

4 Image Output Debugging(Linux is not a quick starter)

4.1 Hardware Preparation

- Confirm that the power supply to the sensor is correct.
- Confirm that the Sensor Reset GPIO is correct.
- Confirm the source of the sensor’s input reference clock (main processor or external crystal oscillator).
- Confirm that the I2C-writable sensor registers can be erased.

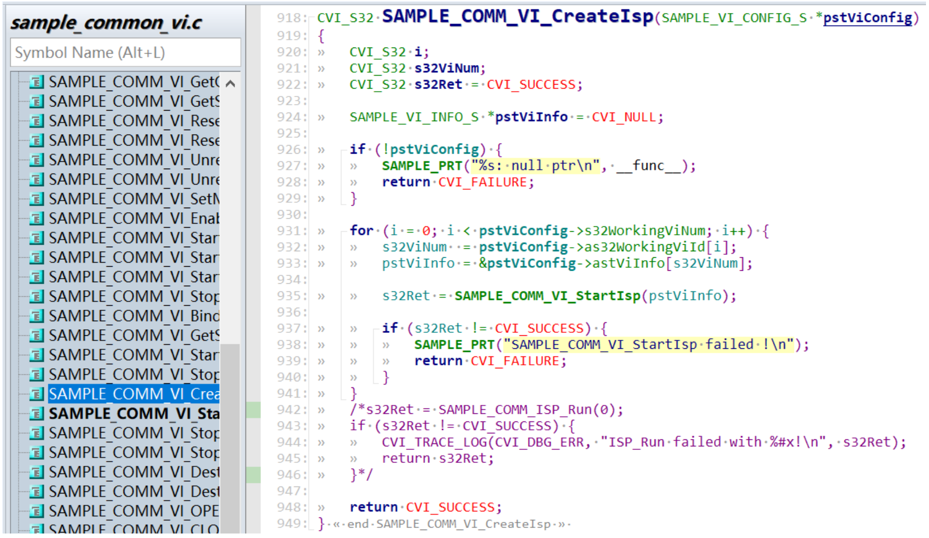
  Use the default i2c_read/i2c_write commands in the file system to verify.

4.2 Configure the Initialization Sequence

Refer to the driver for the sensor of the same manufacturer in the version release package to configure the initialization sequence.

During the initial bringup of a new sensor, it is recommended to comment out AE algorithm-related callbacks to exclude the influence of the algorithm.

- Modify sample_common_vi.c and remove the call to SAMPLE_COMM_ISP_Run.



- Modify the init function in xxx_cmos_ctrl.c and comment out the call to xxx_default_reg_init.

```
	gc2053_slave_write_register(ViPipe,0x13,0x07);
	gc2053_slave_write_register(ViPipe,0x15,0x12);
	gc2053_slave_write_register(ViPipe,0xfe,0x00);
	gc2053_slave_write_register(ViPipe,0x3e,0x91);
	gc2053_slave_write_register(ViPipe,0x17,0x83);

	//gc2053_slave_default_reg_init(ViPipe);
```

Once the sensor adaptation is complete and the image can be displayed, remember to uncomment these lines of code.

4.2.1 Prepare the sensor driver

- Select the sensor driver closest to the specifications in the release package based on the sensor vendor, maximum resolution, and WDR mode, make the necessary modifications, and compile the sensor library. Details can be found in the xxxx_cmos.c, xxxx_cmos_ex.h, xxxx_cmos_param.h, and xxxx_sensor_ctl.c files in component/isp/user/sensor/cv18xx/xxxx.
- Modify the I2C configuration in xxxx_sensor_ctl.c, such as i2c_addr, addr_byte, and data_byte.

```
const CVI_U8 imx327_i2c_addr = 0x1A;
const CVI_U32 imx327_addr_byte = 2;
const CVI_U32 imx327_data_byte = 1;
```

- According to the sensor interface specification, modify the xxxx_rx_attr and pfnGetRxAttr in xxxx_cmos_param.h to set the attributes of the MIPI-RX.

```
struct combo_dev_attr_s gc2053_rx_attr = {
	.input_mode = INPUT_MODE_MIPI,
	.mac_clk = RX_MAC_CLK_600M,
	.mipi_attr = {
		.raw_data_type = RAW_DATA_10BIT,
		.lane_id = {1, 3, 2, -1, -1},
		.wdr_mode = CVI_MIPI_WDR_MODE_NONE,
	},
	.mclk = {
		.cam = 0,
		.freq = CAMPLL_FREQ_27M,
	},
	.devno = 0,
};
```

.Input_mode: Sets the input mode to MIPI, LVDS, or other interface types.

.Mac_clk: mac clock frequency.

.raw_data_type: bit width of data.

.lane id: Configuration of the MIPI data lane and clock lane IDs.

.cam: mclk ID.

.freq: Reference input clock provided by SOC to the sensor.

.devno: mipirx number, sensor ID.

- According to the sensor output mode, modify g_astxxx_mode in xxxx_cmos_param.h.

```
static const IMX327_MODE_S g_astImx327_mode[IMX327_MODE_NUM] = {
	[IMX327_MODE_1080P30] = {
		.name = "1080p30",
		.astImg[0] = {
			.stSnsSize = {
				.u32Width = 1948,
				.u32Height = 1097,
			},
			.stWndRect = {
				.s32X = 12,
				.s32Y = 8,
				.u32Width = 1920,
				.u32Height = 1080,
			},
			.stMaxSize = {
				.u32Width = 1948,
				.u32Height = 1097,
			},
		},
		.f32MaxFps = 30,
		.f32MinFps = 0.119,
		.u32HtsDef = 0x1130,
		.u32VtsDef = 1125,
		.stExp[0] = {
			.u16Min = 1,
			.u16Max = 1123,
			.u16Def = 400,
```

(continues on next page)

(continued from previous page)

```
                .u16Step = 1,
        },
        .stAgain[0] = {
                .u16Min = 1024,
                .u16Max = 62416,
                .u16Def = 1024,
                .u16Step = 1,
        },
        .stDgain[0] = {
                .u16Min = 1024,
                .u16Max = 38485,
                .u16Def = 1024,
                .u16Step = 1,
        },
        .u16RHS1 = 11,
        .u16BRL = 1109,
        .u16OpbSize = 10,
        .u16MarginVtop = 8,
        .u16MarginVbot = 9,
    },
}
```

- Modify pfn_cmos_set_image_mode to determine the corresponding sensor mode based on the specified width, height, and frame rate.

  The output mode corresponding to the init sequence we generally get is the maximum resolution, that is, the all pixel scan mode.

  However, in some cases, customers need to cut the data spit out of the sensor, and they need to adapt to the window crop mode, and they need to find the sensor manufacturer

  Provide the corresponding init sequence in crop mode, or modify it according to the sensor spec.

4.2.2 Sensor initialization sequence

- Implement pfn_cmos_sensor_init, the initial sequence for the sensor mode, in xxxx_sensor_ctrl.c.
- Temporarily comment out the call to xxxx_default_reg_init in xxxx_sensor_ctrl.c.
- Add new sensor object.

```
ISP_SNS_OBJ_S stSnsGc2053_Obj = {
    .pfnRegisterCallback    = sensor_register_callback, //注册ISP、AE相关callback
    .pfnUnRegisterCallback  = sensor_unregister_callback,
    .pfnStandby             = gc2053_standby,           //sensor休眠
    .pfnRestart             = gc2053_restart,           //sensor唤醒
    .pfnMirrorFlip          = CVI_NULL,
    .pfnWriteReg            = gc2053_write_register,    //写寄存器函数
    .pfnReadReg             = gc2053_read_register,     //读寄存器函数
    .pfnSetBusInfo          = gc2053_set_bus_info,      //i2c端口设置
    .pfnSetInit             = sensor_set_init,          //设置快速启动时，AE、AWB相关参数
    .pfnPatchRxAttr     = sensor_patch_rx_attr,         //MIPI-rx属性相关设定
    .pfnGetRxAttr       = sensor_rx_attr,               //获取MIPI-rx属性
    .pfnExpSensorCb     = cmos_init_sensor_exp_function,//ISP相关callback
    .pfnExpAeCb     = cmos_init_ae_exp_function,        //AE相关callback
};
```

4.3 Adapting to Sample Common and alios config

- extern the sensor object to

  mars_alios/components/cvi_mmf_sdk/cvi_sensor/sensor_cfg/sensor_cfg.c

  getSnsObj(SNS_TYPE_E enSnsType) function.

```
ISP_SNS_OBJ_S *getSnsObj(SNS_TYPE_E enSnsType)
{
        switch (enSnsType) {
#if CONFIG_SENSOR_GCORE_GC02M1
        case GCORE_GC02M1_MIPI_2M_30FPS_10BIT:
                return &stSnsGc02m1_Obj;
#endif
#if CONFIG_SENSOR_GCORE_GC02M1_SLAVE
        case GCORE_GC02M1_SLAVE_MIPI_2M_30FPS_10BIT:
                return &stSnsGc02m1_Slave_Obj;
#endif
#if CONFIG_SENSOR_GCORE_GC1054
        case GCORE_GC1054_MIPI_1M_30FPS_10BIT:
                return &stSnsGc1054_Obj;
#endif
#if CONFIG_SENSOR_GCORE_GC2053
        case GCORE_GC2053_MIPI_2M_30FPS_10BIT:
                return &stSnsGc2053_Obj;
#endif
#if CONFIG_SENSOR_GCORE_GC2053_1L
        case GCORE_GC2053_1L_MIPI_2M_30FPS_10BIT:
                return &stSnsGc2053_1l_Obj;
#endif
#if CONFIG_SENSOR_GCORE_GC2093
        case GCORE_GC2093_MIPI_2M_30FPS_10BIT:
        case GCORE_GC2093_MIPI_2M_30FPS_10BIT_WDR2TO1:
                return &stSnsGc2093_Obj;
#endif
```

- Add a new _SNS_TYPE_E to

  mars_alios/components/cvi_mmf_sdk/cvi_sensor/sensor_cfg/sensor_cfg.h

  In the _SNS_TYPE_E enumeration list of, linear is in the top half and WDR is in the bottom half.

```
typedef enum _SNS_TYPE_E {
        SNS_TYPE_NONE = 0,
        /* ------ LINEAR BEGIN ------*/
        SONY_IMX327_MIPI_2M_30FPS_12BIT,
        SONY_IMX307_MIPI_2M_30FPS_12BIT,
        SONY_IMX307_2L_MIPI_2M_30FPS_12BIT,
        SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT,
        GCORE_GC1054_MIPI_1M_30FPS_10BIT,
        GCORE_GC2053_MIPI_2M_30FPS_10BIT,
        GCORE_GC2053_1L_MIPI_2M_30FPS_10BIT,
        GCORE_GC2093_MIPI_2M_30FPS_10BIT,

        GCORE_GC02M1_MIPI_2M_30FPS_10BIT,
        GCORE_GC02M1_SLAVE_MIPI_2M_30FPS_10BIT,
        GCORE_GC4653_MIPI_4M_30FPS_10BIT,
        PIXELPLUS_PR2020_1M_25FPS_8BIT,
        PIXELPLUS_PR2020_1M_30FPS_8BIT,
        PIXELPLUS_PR2020_2M_25FPS_8BIT,
        PIXELPLUS_PR2020_2M_30FPS_8BIT,
        TECHPOINT_TP9950_1M_30FPS_8BIT,
        TECHPOINT_TP9950_2M_30FPS_8BIT,
        TECHPOINT_TP9950_1M_25FPS_8BIT,
        TECHPOINT_TP9950_2M_25FPS_8BIT,
        SMS_SC1336_2L_MIPI_1M_30FPS_10BIT,
        SMS_SC1336_2L_MIPI_1M_60FPS_10BIT,
        SMS_SC1336_2L_SLAVE_MIPI_1M_30FPS_10BIT,
        CISTA_C4390_MIPI_4M_30FPS_10BIT,
        /* ------ LINEAR END ------*/
        SNS_TYPE_LINEAR_BUTT,
        /* ------ WDR 2TO1 BEGIN ------*/
        SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1 = SNS_TYPE_LINEAR_BUTT,
        SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1,
        SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1,
```

- sample_common_vi.c in SAMPLE_COMM_VI_GetDevAttrBySns, SAMPLE_COMM_VI_GetChnAttrBySns, SAMPLE_COMM_VI_GetSizeBySensor adds the corresponding case.

```
case GCORE_GC2053_SLAVE_MIPI_2M_30FPS_10BIT:
case GCORE_GC2053_1L_MIPI_2M_30FPS_10BIT:
case GCORE_GC2093_MIPI_2M_30FPS_10BIT:
case GCORE_GC2093_SLAVE_MIPI_2M_30FPS_10BIT:
case GCORE_GC2093_MIPI_2M_30FPS_10BIT_WDR2TO1:
case GCORE_GC2093_SLAVE_MIPI_2M_30FPS_10BIT_WDR2TO1:
case GCORE_GC1054_MIPI_1M_30FPS_10BIT:
        pstViDevAttr->enBayerFormat = BAYER_FORMAT_RG;
        break;
case GCORE_GC4653_MIPI_4M_30FPS_10BIT:
case GCORE_GC4653_SLAVE_MIPI_4M_30FPS_10BIT:
case TECHPOINT_TP2850_MIPI_2M_30FPS_8BIT:
case TECHPOINT_TP2850_MIPI_4M_30FPS_8BIT:
// brigates
case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT:
case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT_WDR2TO1:
        pstViDevAttr->enBayerFormat = BAYER_FORMAT_GR;
        break;
default:
        pstViDevAttr->enBayerFormat = BAYER_FORMAT_BG;
        break;
};
```

```
    case SONY_IMX307_SUBLVDS_2M_60FPS_12BIT:
    case SONY_IMX307_MIPI_2M_60FPS_12BIT:
    case GCORE_GC2053_1L_MIPI_2M_30FPS_10BIT:
    case SONY_IMX335_MIPI_2M_60FPS_10BIT:
    case TECHPOINT_TP2850_MIPI_2M_30FPS_8BIT:
    case SONY_IMX335_MIPI_2M_30FPS_10BIT_WDR2TO1:
    case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT:
    case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT_WDR2TO1:
        *penSize = PIC_1080P;
        break;
    case OV_OS08A20_MIPI_8M_30FPS_10BIT:
    case OV_OS08A20_MIPI_8M_30FPS_10BIT_WDR2TO1:
    case OV_OS08A20_SLAVE_MIPI_8M_30FPS_10BIT:
    case OV_OS08A20_SLAVE_MIPI_8M_30FPS_10BIT_WDR2TO1:
    case SONY_IMX334_MIPI_8M_30FPS_12BIT:
    case SONY_IMX334_MIPI_8M_30FPS_12BIT_WDR2TO1:
    case SMS_SC8238_MIPI_8M_30FPS_10BIT:
    case SMS_SC8238_MIPI_8M_15FPS_10BIT_WDR2TO1:
    case SMS_SC850SL_MIPI_8M_30FPS_12BIT:
    case SMS_SC850SL_MIPI_8M_30FPS_10BIT_WDR2TO1:
        *penSize = PIC_3840x2160;
        break;
```

- Add the new sensor name to the snsr_type_name array in sample_common_vi.c,

  Notice that the sensor name is added with the new one in sensor_cfg.h

  The enum of _SNS_TYPE_E has the same name and order.

```
    "SMS_SC4210_MIPI_4M_30FPS_12BIT",
    /* ------ LINEAR END ------*/

    /* ------ WDR 2TO1 BEGIN ------*/
    "SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX327_2L_MIPI_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1",
    "OV_OS08A20_MIPI_8M_30FPS_10BIT_WDR2TO1",
    "OV_OS08A20_MIPI_5M_30FPS_10BIT_WDR2TO1",
    "SOI_F35_MIPI_2M_30FPS_10BIT_WDR2TO1",
    "SOI_F35_SLAVE_MIPI_2M_30FPS_10BIT_WDR2TO1",
    "SONY_IMX327_SUBLVDS_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX307_SUBLVDS_2M_30FPS_12BIT_WDR2TO1",
    "SONY_IMX335_MIPI_5M_30FPS_10BIT_WDR2TO1",
    "SONY_IMX335_MIPI_4M_30FPS_10BIT_WDR2TO1",
    "SONY_IMX334_MIPI_8M_30FPS_12BIT_WDR2TO1",
    "SMS_SC8238_MIPI_8M_15FPS_10BIT_WDR2TO1",
    "SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1",
```

- Add the sensor driver directory name and source information to mars_alios/components/cvi_mmf_sdk/cvi_sensor/package.yaml

```
build_config:
  include:
    - sensor_cfg
    - sensor_i2c
    - gcore_gc02m1
    - gcore_gc02m1_slave
    - gcore_gc2053
    - gcore_gc2053_1L
    - gcore_gc2093

    - gcore_gc4653
    - pixelplus_pr2020
    - sony_imx307
    - sony_imx327
    - techpoint_tp9950
    - sms_sc1336_2L
    - sms_sc1336_slave
    - cista_c4390
```

```
source_file:
## middleware
  - sensor_cfg/*.c
  - sensor_i2c/*.c
  - gcore_gc02m1/*.c ? <CONFIG_SENSOR_GCORE_GC02M1>
  - gcore_gc02m1_slave/*.c ? <CONFIG_SENSOR_GCORE_GC02M1_SLAVE>
  - gcore_gc2053/*.c ? <CONFIG_SENSOR_GCORE_GC2053>
  - gcore_gc2053_1L/*.c ? <CONFIG_SENSOR_GCORE_GC2053_1L>
  - gcore_gc2093/*.c ? <CONFIG_SENSOR_GCORE_GC2093>
  - gcore_gc4653/*.c ? <CONFIG_SENSOR_GCORE_GC4653>

  - pixelplus_pr2020/*.c ? <CONFIG_SENSOR_PIXELPLUS_PR2020>
  - sony_imx307/*.c ? <CONFIG_SENSOR_SONY_IMX307>
  - sony_imx327/*.c ? <CONFIG_SENSOR_SONY_IMX327>
  - techpoint_tp9950/*.c ? <CONFIG_SENSOR_TECHPOINT_TP9950>
  - sms_sc1336_2L/*.c ? <CONFIG_SENSOR_SMS_SC1336_2L>
  - sms_sc1336_slave/*.c ? <CONFIG_SENSOR_SMS_SC1336_SALVE>
  - cista_c4390/*.c ? <CONFIG_SENSOR_CISTA_C4390>
```

- In Linux, the sensor configuration uses the following interface, see sample for usage:
- *CVI_S32 CVI_SENSOR_GPIO_Init(VI_PIPE ViPipe, SNS_I2C_GPIO_INFO_S *pstGpioCfg);*

  To configure the reset GPIO information of each sensor, SNS_I2C_GPIO_INFO_S structure is as follows:

  typedef struct _SNS_I2C_GPIO_INFO_S {

  CVI_S8 s8I2cDev;

  CVI_S32 s32I2cAddr;

  CVI_U32 u32Rst_port_idx;

  CVI_U32 u32Rst_pin;

  CVI_U32 u32Rst_pol;

  } SNS_I2C_GPIO_INFO_S;

- *CVI_S32 CVI_SENSOR_GetAhdStatus(VI_PIPE ViPipe, SNS_AHD_MODE_S *pstStatus);*

  Get the status of AHD Sensor, restricted to AHD sensor, SNS_AHD_MODE_S structure is as follows:

  typedef enum _SNS_AHD_MODE_E {

  AHD_MODE_NONE,

AHD_MODE_1280X720H_NTSC,

AHD_MODE_1280X720H_PAL,

AHD_MODE_1280X720P25,

AHD_MODE_1280X720P30,

AHD_MODE_1280X720P50,

AHD_MODE_1280X720P60,

AHD_MODE_1920X1080P25,

AHD_MODE_1920X1080P30,

AHD_MODE_2304X1296P25,

AHD_MODE_2304X1296P30,

AHD_MODE_BUIsensor_cfg.h } SNS_AHD_MODE_S;

- *CVI_S32 CVI_SENSOR_SetSnsType(VI_PIPE ViPipe, CVI_U32 SnsType);*

  Set the sensor ID of the corresponding PIPE. This method needs to be called before calling other methods. SnsType can be seen in sensor_cfg.h

- *CVI_S32 CVI_SENSOR_SetSnsRxAttr(VI_PIPE ViPipe, RX_INIT_ATTR_S *pstRxAttr);*

  To set the RX configuration of the corresponding sensor, see cvi_sns_ctrl.h for RX_INIT_ATTR_S structure

- *CVI_S32 CVI_SENSOR_SetSnsI2c(VI_PIPE ViPipe, CVI_S32 astI2cDev, CVI_S32 s32I2cAddr);*

  Set the I2C bus and address of the corresponding sensor

- *CVI_S32 CVI_SENSOR_SetSnsIspAttr(VI_PIPE ViPipe, ISP_INIT_ATTR_S *pstInitAttr);*

  To set the configuration of sensor to ISP, the ISP_INIT_ATTR_S structure is shown in cvi_sns_ctrl.h

- *CVI_S32 CVI_SENSOR_RegCallback(VI_PIPE ViPipe, ISP_DEV IspDev);*

  Set the sensor and ISP callbacks

- *CVI_S32 CVI_SENSOR_UnRegCallback(VI_PIPE ViPipe, ISP_DEV IspDev);*

  Remove the callback from the sensor and ISP

- *CVI_S32 CVI_SENSOR_SetSnsImgMode(VI_PIPE ViPipe, ISP_CMOS_SENSOR_IMAGE_MODE_S *stSnsrMode);*

  Set the mode of the sensor to run, including fps, size, etc., ISP_CMOS_SENSOR_IMAGE_MODE_S structure see cvi_comm_sns.h

- *CVI_S32 CVI_SENSOR_SetSnsWdrMode(VI_PIPE ViPipe, WDR_MODE_E wdrMode);*

  Set the sensor WDR mode; see cvi_comm_cif.h for the WDR_MODE_E structure

- *CVI_S32 CVI_SENSOR_GetSnsRxAttr(VI_PIPE ViPipe, SNS_COMBO_DEV_ATTR_S *stDevAttr);*

  Get the RX configuration of the sensor, SNS_COMBO_DEV_ATTR_S structure in cvi_comm_cif.h

- *CVI_S32 CVI_SENSOR_SetSnsProbe(VI_PIPE ViPipe);*

  Set the probe of the sensor corresponding to the PIPE

- *CVI_S32 CVI_SENSOR_SetSnsGpioInit(CVI_U32 devNo, CVI_U32 u32Rst_port_idx, CVI_U32 u32Rst_pin, CVI_U32 u32Rst_pol);*

  Configure the reset GPIO information of each sensor, u32Rst_port_idx, u32Rst_pin, u32Rst_pol, see the ini configuration content in the next section

- *CVI_S32 CVI_SENSOR_RstSnsGpio(CVI_U32 devNo, CVI_U32 rstEnable);*

  Pull the rst foot of the sensor to the valid position

- *CVI_S32 CVI_SENSOR_RstMipi(CVI_U32 devNo, CVI_U32 rstEnable);*

  reset the MIPI used by the corresponding sensor

- *CVI_S32 CVI_SENSOR_SetMipiAttr(VI_PIPE ViPipe, CVI_U32 SnsType);*

  The RX of the sensor is configured to the CIF

- *CVI_S32 CVI_SENSOR_EnableSnsClk(CVI_U32 devNo, CVI_U32 clkEnable);*

  enable sensor mclk

- *CVI_S32 CVI_SENSOR_SetSnsStandby(VI_PIPE ViPipe);*

  Set the standby state of the sensor

- *CVI_S32 CVI_SENSOR_SetSnsInit(VI_PIPE ViPipe);*

  Set the sensor start init

- *CVI_S32 CVI_SENSOR_SetVIFlipMirrorCB(VI_PIPE ViPipe, VI_DEV ViDev);*

  Register the mirror and flip of the sensor into VI

- The following methods are provided to ISPs for use. Please check with the relevant documentation of your ISP

- *CVI_S32 CVI_SENSOR_GetAeDefault(VI_PIPE ViPipe, AE_SENSOR_DEFAULT_S *stAeDefault);*

  The AE default status of the corresponding sensor is obtained

- *CVI_S32 CVI_SENSOR_GetIspBlkLev(VI_PIPE ViPipe, ISP_CMOS_BLACK_LEVEL_S *stBlc);*

  To obtain the BLK value of the corresponding sensor, the ISP_CMOS_BLACK_LEVEL_S structure is given in cvi_comm_sns.h

- *CVI_S32 CVI_SENSOR_SetSnsFps(VI_PIPE ViPipe, CVI_U8 fps, AE_SENSOR_DEFAULT_S *stSnsDft);*

  Set the output FPS of the sensor

- *CVI_S32 CVI_SENSOR_GetExpRatio(VI_PIPE ViPipe, SNS_EXP_MAX_S *stExpMax);*

  Get the exposure range of the sensor

- *CVI_S32 CVI_SENSOR_SetDgainCalc(VI_PIPE ViPipe, SNS_GAIN_S *stDgain);*

  Set the digital gain value of the sensor

- *CVI_S32 CVI_SENSOR_SetAgainCalc(VI_PIPE ViPipe, SNS_GAIN_S *stAgain);*

  Set the simulated gain value of the sensor

4.4 Adding Sensor INI Configuration

Some properties of Sensor can be modified by changing ini configuration, such as lane line order, I2C port sensor output mode, etc.

By default, the middleware process will first read the sensor configuration file from /mnt/data/sensor_ini.cfg. If there is no configuration file in that directory, it will use the initial value from the code.

The following shows the contents of sensor_cfg.ini using SC1336 as an example:

```
[source]

;type = SOURCE_USER_FE

dev_num = 1

; section for sensor

[sensor]

; sensor name

name = SMS_SC1336_2L_MIPI_1M_60FPS_10BIT

bus_id = 3

mipi_dev = 0

lane_id = 2, 3, 1, -1, -1

pn_swap = 1, 1, 1, 0, 0

mclk_en = 1

mclk = 0

port = 0
```

(continues on next page)

(continued from previous page)

```
pin = 2

pol = 1

fps = 60
```

- name: This indicates the output mode of the sensor; be sure to match the name of the SAMPLE_SNS_TYPE_E enum added to sample_comm.h.
- Bus_id: This indicates the I2C port number
- Mipi_dev: This indicates which set of mipi-rx is used
- Lane_id: This indicates the linear order configuration of mipi
- pn_swap: Indicates whether this set of mipi linear order P/N needs to be reversed
- Pn_swap: denotes P/N inversion, does not need to be reversed to 0 configuration, needs to be reversed to 1 configuration
- Mclk: This specifies which set of MCLKS is selected as the reference clock
- Mclk_en: This indicates which set of mclk outputs is enabled
- hw_sync: dual sensor frame synchronization, hw_sync=1 means slave sensor sync with master sensor
- sns_i2c_addr: The i2c device address of the sensor
- port: The sensor RST pin corresponds to port A/B/ C-0/1/2 used by the GPIO of the processor
- pin: The sensor RST pin corresponds to the number of the port used by the GPIO of the processor
- pol: The effective level of the sensor RST pin

  The corresponding parameters are configured as follows： enum of_gpio_flags
  {

  OF_GPIO_ACTIVE_LOW = 0x1,

  OF_GPIO_SINGLE_ENDED = 0x2,

  OF_GPIO_OPEN_DRAIN = 0x4,

  OF_GPIO_TRANSITORY = 0x8,

  OF_GPIO_PULL_UP = 0x10,

  OF_GPIO_PULL_DOWN = 0x20,

  };

- fps：The output fps of sensor is set to 25 by default, and other fps need to set the corresponding fps value

4.5 Build and run sensor test

After the configuration in the previous section, run make peripherals_test in the top-level SDK directory to compile, and burn the compiled firmware to the board side;

After the burn boot, the Linux serial terminal executes sensor_test

```
(cli-uart)# sensor_test
bus info:2
bf314a i2c init
ViPipe:0,===BF314A 720P 30fps 10bit LINEAR Init OK!===
******start isp******
---Basic-----------------------------------------------
1: dump vi raw frame
2: dump vi yuv frame
3: set chn flip/mirror
4: linear wdr switch
5: AE debug
6: sensor dump
7: sensor proc
8: sensor i2c read/write
255: exit
```

Input proc/vi_dbg in alios serial port to check vi_dbg information. If the frame rate shows normal, it means that the sensor has normal output

```
7
input op num is [7]
---debug_info-------------------------------------------
1: proc_vi_dbg
2: proc_vi
3: proc_mipi_rx
1

[VI BE_Dbg_Info]
VIPreBEDoneSts            :0x10          VIPreBEDmaIdleStatus     :0x4
[VI Post_Dbg_Info]
VIIspTopStatus            :0x386
[VI DMA_Dbg_Info]
VIWdma0ErrStatus          :0x3000000     VIWdma0IdleStatus        :0xffffeaf
VIWdma1ErrStatus          :0x3000000     VIWdma1IdleStatus        :0xffffe054
VIRdmaErrStatus           :0x3000000     VIRdmaIdleStatus         :0xfffff8a5
[VI ISP_PIPE_A FE_Dbg_Info]
VIPreFERawDbgSts          :0x3f0         VIPreFEDbgInfo           :0x2f
[VI ISP_PIPE_A]
VIOutImgWidth             :1280
VIOutImgHeight            : 720
VIInImgWidth              :1288
VIInImgHeight             : 728
VIDevFPS                  :  30
VIFPS                     :  30
VISofCh0Cnt               :19893
VIPreFECh0Cnt             :19892
VIPreBECh0Cnt             :19892
VIPostCnt                 :19892
VIDropCnt                 :   0
[VI ISP_PIPE_A Csi_Dbg_Info]
VICsiIntStatus0           :0x0
VICsiIntStatus1           :0x0
VICsiCh0Dbg               :0x0
VICsiCh1Dbg               :0x0
```

5 Image output debugging (Alios Quickstart)

5.1 Hardware Preparation

- Confirm that the power supply to the sensor is correct.
- Confirm that the Sensor Reset GPIO is correct.
- Confirm the source of the sensor’s input reference clock (main processor or external crystal oscillator).
- Confirm that the I2C-writable sensor registers can be erased.

  Use the default i2c_read/i2c_write commands in the file system to verify.

5.2 Configure the Initialization Sequence

Refer to the driver for the sensor of the same manufacturer in the version release package to configure the initialization sequence.

During the initial bringup of a new sensor, it is recommended to comment out AE algorithm-related callbacks to exclude the influence of the algorithm.

- Modify components/cvi_platform/media/src/media_video.c by first removing the CVI_ISP_Run call.

```
#if(CONFIG_APP_ISP_BYPASS == 0)
    CVI_ISP_SetBypassFrm(0, 0);
#endif

    s32Ret = CVI_ISP_Init(ViPipe);
    if (s32Ret != CVI_SUCCESS) {
        MEDIABUG_PRINTF("ISP Init failed with %#x!\n", s32Ret);
        return s32Ret;
    }

    //Run ISP
    // s32Ret = CVI_ISP_Run(ViPipe);
    // if (s32Ret != CVI_SUCCESS) {
    //  MEDIABUG_PRINTF("ISP Run failed with %#x!\n", s32Ret);
    //  return s32Ret;
    // }
```

- Modify the init function in xxx_cmos_ctrl.c by first annotating the call to xxx_default_reg_init.

```
    bf314a_write_register(ViPipe, 0x5c, 0x10);   //black target Gr
    bf314a_write_register(ViPipe, 0x6a, 0x1f);   //模拟增益
    bf314a_write_register(ViPipe, 0x6b, 0x01);   //积分时间 :2step
    bf314a_write_register(ViPipe, 0x6c, 0xc2);   //积分时间
    bf314a_write_register(ViPipe, 0x6d, 0x14);

    //out 1288*728
    bf314a_write_register(ViPipe, 0xcd, 0x08);
    bf314a_write_register(ViPipe, 0xcf, 0xd8);

    // bf314a_default_reg_init(ViPipe);
```

Remember to turn these annotations back on when the sensor is ready to display the image.

5.2.1 Prepare Sensor actuation

- According to the Sensor manufacturer, maximum resolution and WDR mode, select the sensor driver with the closest specifications in the release package to modify and compile the sensor library.

  Specifically, see xxxx_cmos_ex.h, xxxx_cmos_param.h and xxxx_sensor_ctl.c in mars_alios/components/cvi_mmf_sdk/cvi_sensor/xxxx

- Modify the I2C configuration in xxxx_sensor_ctl.c as i2c_addr, addr_byte and data_byte

```
const CVI_U8 bf314a_i2c_addr = 0x6e;
const CVI_U32 bf314a_addr_byte = 1;
const CVI_U32 bf314a_data_byte = 1;
```

- According to the sensor interface specification, modify xxxx_rx_attr and pfnGetRxAttr in xxxx_cmos_param.h to set the attribute of mipi-rx.

```
struct combo_dev_attr_s bf314a_rx_attr = {
    .input_mode = INPUT_MODE_MIPI,
    .mac_clk = RX_MAC_CLK_200M,
    .mipi_attr = {
        .raw_data_type = RAW_DATA_10BIT,
        .lane_id = {3, 4, -1, -1, -1},
        .wdr_mode = CVI_MIPI_WDR_MODE_NONE,
        .dphy = {
            .enable = 1,
            .hs_settle = 8,
        }
    },
    .mclk = {
        .cam = 1,
        .freq = CAMPLL_FREQ_24M,
    },
    .devno = 0,
};
```

```
.Set whether the input mode is mipi or lvds, etc.

.Mac clk: mac clock frequency

.raw_date_type:The bit width of data

.lane id:ID configuration of mipi data lane and clock lane

.cam: mclk ID

.freq: The reference input clock provided by the SOC to the sensor

.devno:mipirx number, sensor ID
```

- According to the sensor output mode, modify g_astxxx_mode in xxxx_cmos_param.h.

```
static const BF314A_MODE_S g_astBf314a_mode[BF314A_MODE_NUM] = {
    [BF314A_MODE_1280X720P30] = {
        .name = "1280X720P30",
        .astImg[0] = {
            .stSnsSize = {
                .u32Width = 1288,
                .u32Height = 728,
            },
            .stWndRect = {
                .s32X = 4,
                .s32Y = 4,
                .u32Width = 1280,
                .u32Height = 720,
```

(continues on next page)

(continued from previous page)

```
            },
            .stMaxSize = {
                .u32Width = 1288,
                .u32Height = 728,
            },
        },
        .f32MaxFps = 30,
        .f32MinFps = 0.34, /* vts * 30 / 0xFFFF */
        .u32HtsDef = 1600,
        .u32VtsDef = 750,
        .stExp[0] = {
            .u16Min = 1,
            .u16Max = 750,
            .u16Def = 450,
            .u16Step = 1,
        },
        .stAgain[0] = {
            .u32Min = 1024,
            .u32Max = 16384,
            .u32Def = 1024,
            .u32Step = 1,
        },
        .stDgain[0] = {
            .u32Min = 1024,
            .u32Max = 16384,
            .u32Def = 1024,
            .u32Step = 1,
        },
    },
};
```

- Modify pfn_cmos_set_image_mode to determine the corresponding sensor mode based on the specified width, height, and frame rate.

  The output mode corresponding to the init sequence we generally get is the maximum resolution, that is, the all pixel scan mode.

  However, in some cases, customers need to cut the data spit out of the sensor, and they need to adapt to the window crop mode, and they need to find the sensor manufacturer

  Provide the corresponding init sequence in crop mode, or modify it according to the sensor spec.

5.2.2 Sensor initialization sequence

- Implement the initial sequence pfn_cmos_sensor_init of the sensor mode in xxxx_sensor_ctrl.c.
- Note for a moment the call to xxxx_default_reg_init inside xxxx_sensor_ctrl.c.
- Added sensor object

```
ISP_SNS_OBJ_S stSnsBf314a_Obj = {
    .pfnRegisterCallback    = sensor_register_callback,     //注册ISP、AE相关callback
    .pfnUnRegisterCallback  = sensor_unregister_callback,
    .pfnStandby             = bf314a_standby,               //sensor 休眠
    .pfnRestart             = bf314a_restart,               //sensor 唤醒
    .pfnWriteReg            = bf314a_write_register,        //写寄存器函数
    .pfnReadReg             = bf314a_read_register,         //读寄存器函数
    .pfnSetBusInfo          = bf314a_set_bus_info,          //i2c端口设置
    .pfnSetInit             = sensor_set_init,              //设置快速启动时，AE、AWB相关参数
    .pfnMirrorFlip          = bf314a_mirror_flip,           //镜像、翻转 设定函数
    .pfnPatchRxAttr         = sensor_patch_rx_attr,         //MIPI-rx属性相关设定
    .pfnPatchI2cAddr        = sensor_patch_i2c_addr,        //设定i2c 地址
    .pfnGetRxAttr           = sensor_rx_attr,               //获取MIPI-rx属性
    .pfnExpSensorCb         = cmos_init_sensor_exp_function,//ISP相关callback
    .pfnExpAeCb             = cmos_init_ae_exp_function,    //AE相关callback
    .pfnSnsProbe            = sensor_probe,                 //sensor 探测
};
```

5.2.3 Modify package.yaml to add a build file

- Modify components/cvi_mmf_sdk/cvi_sensor/package.yaml to add headers and source files

```
    - sms_sc201cs
    - sms_sc201cs_slave
    - sms_sc031iot
    - sony_imx307
    - sony_imx327
    - byd_bf314a

# source_file:                            # <可选项> 指定参与编译的源代码
#   - src/*.c                             # 例：组件 src 目录下所有的扩展
source_file:
## middleware
  - sensor_i2c/*.c
  - gcore_gc02m1/*.c
  - gcore_gc02m1_slave/*.c
  - gcore_gc2053/*.c
  - gcore_gc2053_1L/*.c
  - gcore_gc2093/*.c
  - gcore_gc4653/*.c
  - opnous_opn8018/*.c
  - sms_sc200ai/*.c
  - sms_sc2336/*.c
  - sms_sc3336/*.c
  - sms_sc030iot/*.c
  - sms_sc201cs/*.c
  - sms_sc201cs_slave/*.c
  - sms_sc031iot/*.c
  - sony_imx307/*.c
  - sony_imx327/*.c
  - byd_bf314a/*.c
```

5.3 Adaptation solution

5.3.1 Add sensor type

- mars_alios/components/cvi_mmf_sdk/cvi_sensor/sensor_cfg/sensor_cfg.h add a type to _SNS_TYPE_E

```
    SMS_SC3336_MIPI_3M_30FPS_10BIT,
    SMS_SC030IOT_MIPI_480P_30FPS_8BIT,
    SMS_SC201CS_MIPI_2M_30FPS_10BIT,
    SMS_SC201CS_SLAVE_MIPI_2M_30FPS_10BIT,
    SMS_SC031IOT_MIPI_480P_30FPS_8BIT,
    BYD_BF314A_MIPI_720P_30FPS_10BIT,
    /* ------ LINEAR END ------*/
    SNS_TYPE_LINEAR_BUTT,
    /* ------ WDR 2TO1 BEGIN ------*/
    SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1 = SNS_TYPE_LINEAR_BUTT,
    SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1,
    SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1,
    SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1,
    GCORE_GC2093_MIPI_2M_30FPS_10BIT_WDR2TO1,
    SMS_SC200AI_MIPI_2M_30FPS_10BIT_WDR2TO1,
    /* ------ WDR 2TO1 END ------*/
    SNS_TYPE_WDR_BUTT,
} SNS_TYPE_E;
```

- mars_alios/components/cvi_mmf_sdk/cvi_sensor/sensor_cfg/sensor_cfg.c

  Add the sensor object, and add the corresponding case to the getPicSize and getDevAttr functions

```
#if CONFIG_SENSOR_SONY_IMX307_2L
extern ISP_SNS_OBJ_S stSnsImx307_2l_Obj;
#endif
#if CONFIG_SENSOR_SONY_IMX307_SLAVE
extern ISP_SNS_OBJ_S stSnsImx307_Slave_Obj;
#endif
#if CONFIG_SENSOR_SONY_IMX327
extern ISP_SNS_OBJ_S stSnsImx327_Obj;
#endif
#if CONFIG_SENSOR_BYD_BF314A
extern ISP_SNS_OBJ_S stSnsBf314a_Obj;
#endif
```

```
#if CONFIG_SENSOR_SONY_IMX307
    case SONY_IMX307_MIPI_2M_30FPS_12BIT:
    case SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1:
        return &stSnsImx307_Obj;
#endif
#if CONFIG_SENSOR_SONY_IMX307_2L
    case SONY_IMX307_2L_MIPI_2M_30FPS_12BIT:
    case SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1:
        return &stSnsImx307_2l_Obj;
#endif
#if CONFIG_SENSOR_SONY_IMX307_SLAVE
    case SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT:
    case SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1:
        return &stSnsImx307_Slave_Obj;
#endif
#if CONFIG_SENSOR_SONY_IMX327
    case SONY_IMX327_MIPI_2M_30FPS_12BIT:
    case SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1:
        return &stSnsImx327_Obj;
#endif
#if CONFIG_SENSOR_BYD_BF314A
    case BYD_BF314A_MIPI_720P_30FPS_10BIT:
        return &stSnsBf314a_Obj;
#endif
    default:
        return CVI_NULL;
    }
```

```
switch (sensor_type) {
case GCORE_GC02M1_MIPI_2M_30FPS_10BIT:
case GCORE_GC02M1_SLAVE_MIPI_2M_30FPS_10BIT:
    pstSize->u32Width  = 1600;
    pstSize->u32Height = 1200;
    break;
case GCORE_GC02M1_MIPI_600P_30FPS_10BIT:
case GCORE_GC02M1_SLAVE_MIPI_600P_30FPS_10BIT:
    pstSize->u32Width  = 800;
    pstSize->u32Height = 600;
    break;
case GCORE_GC1054_MIPI_1M_30FPS_10BIT:
case BYD_BF314A_MIPI_720P_30FPS_10BIT:
    pstSize->u32Width  = 1280;
    pstSize->u32Height = 720;
    break;
```

```
switch (sensor_type) {
case SONY_IMX327_MIPI_2M_30FPS_12BIT:
case SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1:
case SONY_IMX307_MIPI_2M_30FPS_12BIT:
case SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1:
case SONY_IMX307_2L_MIPI_2M_30FPS_12BIT:
case SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1:
case SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT:
case SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1:
// GalaxyCore
case GCORE_GC02M1_MIPI_2M_30FPS_10BIT:
case GCORE_GC02M1_SLAVE_MIPI_2M_30FPS_10BIT:
case GCORE_GC02M1_MIPI_600P_30FPS_10BIT:
case GCORE_GC02M1_SLAVE_MIPI_600P_30FPS_10BIT:
case GCORE_GC2053_MIPI_2M_30FPS_10BIT:
case GCORE_GC2053_1L_MIPI_2M_30FPS_10BIT:
case GCORE_GC1054_MIPI_1M_30FPS_10BIT:
case GCORE_GC2093_MIPI_2M_30FPS_10BIT:
case GCORE_GC2093_MIPI_2M_30FPS_10BIT_WDR2TO1:
case BYD_BF314A_MIPI_720P_30FPS_10BIT:
    pstViDevAttr->enBayerFormat = BAYER_FORMAT_RG;
    break;
case GCORE_GC4653_MIPI_4M_30FPS_10BIT:
    pstViDevAttr->enBayerFormat = BAYER_FORMAT_GR;
    break;
default:
    pstViDevAttr->enBayerFormat = BAYER_FORMAT_BG;
    break;
};
```

5.3.2 Modify the sensor mipi related configuration

- solutions/peripherals_test/customization/peripherals_qfn/param/custom_viparam.c

  In this configuration, reset pin, mipi lane and other configurations will replace the information set by sensor driver by default

  configuration instruction:

  s32I2cAddr: sensor i2c device address

  s8I2cDev:Represents the I2C port number

  u32Rst_port_idx: reset the GPIO group of the pin

  u32Rst_pin: reset the GPIO num of the pin

  as16LaneId:Represents the linear order configuration of mipi as8PNSwap:Denote P/N inversion, does not need to invert the configuration to 0, needs to invert the configuration to 1

u8MclkCam:Indicates which set of mclk is selected as the reference clock

s16MacClk: mac clk

u8MclkFreq: MCLK frequency

bHwSync: dual sensor frame synchronization, bHwSync =1 means slave sensor sync with master sensor

s32Framerate: frame rate

```
PARAM_CLASSDEFINE(PARAM_SNS_CFG_S,SENSORCFG,CTX,Sensor)[] = {
    {
        .enSnsType = CONFIG_SNS0_TYPE,
        .s32I2cAddr = 0x10,
        .s8I2cDev = 2,
        .u32Rst_port_idx = 2,//GPIOC_16
        .u32Rst_pin = 16,
        .u32Rst_pol = OF_GPIO_ACTIVE_LOW,
        /* config mipi attr */
        .bSetDevAttrMipi = 1,
        .as16LaneId[0] = 1,
        .as16LaneId[1] = 0,
        .as16LaneId[2] = -1,
        .as16LaneId[3] = -1,
        .as16LaneId[4] = -1,
        .as8PNSwap[0] = 0,
        .as8PNSwap[1] = 0,
        .as8PNSwap[2] = 0,
        .as8PNSwap[3] = 0,
        .as8PNSwap[4] = 0,
        /* config dev attr(reference sensor driver) */
        .bSetDevAttr = 1,
        .u8MclkCam = 0,
        .s16MacClk = RX_MAC_CLK_200M,
        .u8MclkFreq = CAMPLL_FREQ_27M,
        .bHwSync = CVI_FALSE,
        .s32Framerate = 30,
    },
```

If you want to use the driver parameters by default, you can just configure the following image

```
PARAM_CLASSDEFINE(PARAM_SNS_CFG_S,SENSORCFG,CTX,Sensor)[] = {
    {
        .enSnsType = CONFIG_SNS0_TYPE,
        .s32I2cAddr = -1,
        .s8I2cDev = 2,
        .u32Rst_port_idx = 2,//GPIOC_13
        .u32Rst_pin = 13,
        .u32Rst_pol = OF_GPIO_ACTIVE_LOW,
    }
};
```

5.3.3 Modifying VB Configuration

- solutions/peripherals_test/customization/peripherals_qfn/param/custom_sysparam.c Modify the u16width and u16height of VB according to the sensor output size

```
PARAM_CLASSDEFINE(PARAM_VB_CFG_S,VBPOOL,CTX,VB)[] = {
    {
        .u16width = 1280,
        .u16height = 720,
        .u8VbBlkCnt = 6,
        .fmt = PIXEL_FORMAT_NV21,
        .enBitWidth = DATA_BITWIDTH_8,
        .enCmpMode = COMPRESS_MODE_NONE,
    },
};
```

5.3.4 Modify to add pinmux

- solutions/peripherals_test/customization/peripherals_qfn/src/custom_platform.c According to the actual hardware configuration of the board, modify the pin multiplexing such as mipi i2c reset

```
static void _SensorPinmux()
{
    //Sensor Pinmux
    PINMUX_CONFIG(PAD_MIPI_TXP1, IIC2_SCL);
    PINMUX_CONFIG(PAD_MIPI_TXM1, IIC2_SDA);
    PINMUX_CONFIG(PAD_MIPI_TXP0, XGPIOC_13);
    PINMUX_CONFIG(PAD_MIPI_TXM0, CAM_MCLK1);
}

static void _MipiRxPinmux(void)
{
//mipi rx pinmux
    PINMUX_CONFIG(PAD_MIPIRX4P, XGPIOC_3);
    PINMUX_CONFIG(PAD_MIPIRX4N, XGPIOC_2);
}
```

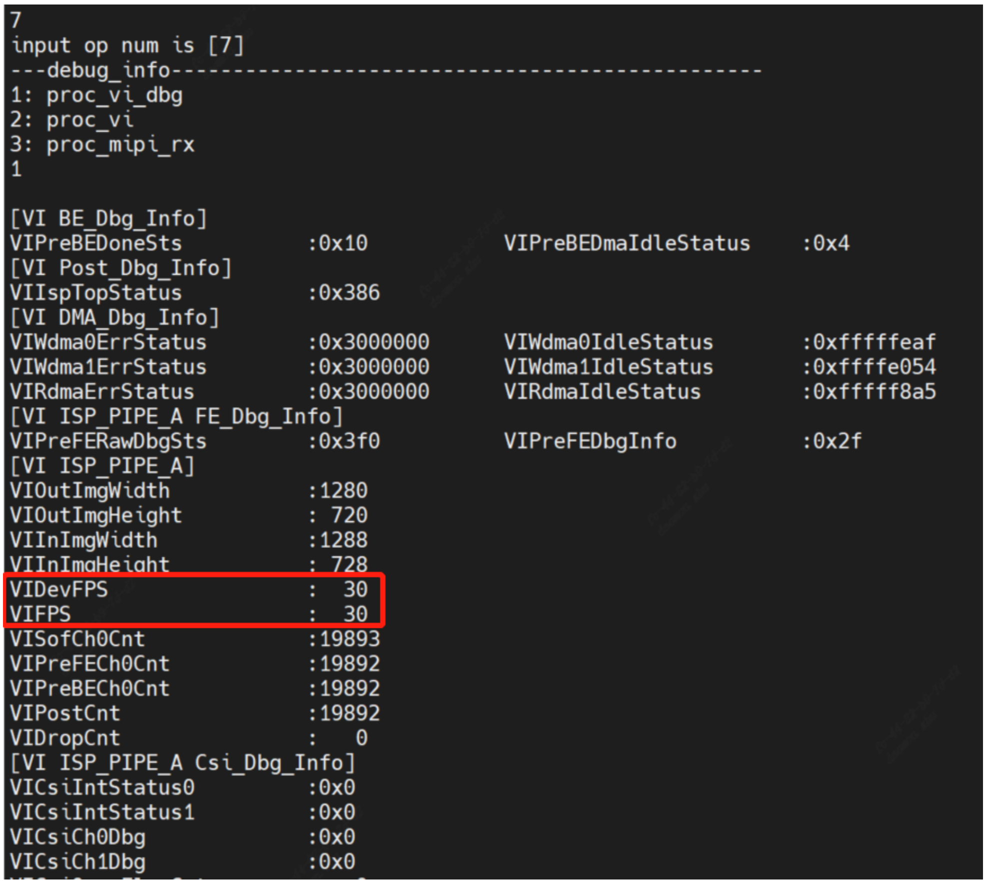
5.3.5 Modifying the build configuration

- solutions/peripherals_test/package.yaml.peripherals_qfn

```
CONFIG_SENSOR_GCORE_GC4653: 1
CONFIG_SENSOR_SMS_SC200AI: 1
CONFIG_SENSOR_SMS_SC2336: 1
CONFIG_SENSOR_SONY_IMX307: 1
CONFIG_SENSOR_SONY_IMX327: 1
CONFIG_SENSOR_BYD_BF314A: 1
CONFIG_SUPPORT_NORFLASH: 1

CONFIG_SNS0_TYPE: 22
CONFIG_SNS1_TYPE: 0
```

5.4 Running sensor

After compiling and running, the sensor will be started directly under the quick start, and the vi_dbg information will be checked by inputting proc/vi_dbg in the alios serial port. If the frame rate is normal, the sensor will produce the picture normally



6 Image Output Verification

If the timing meets the working requirements of the sensor and there is no "select timeout" message printed, it can be confirmed that the sensor image is outputting normally after configuring the init settings.

If there is an exception, please refer to *10.5. Error Checking Process* .

Below is an example of using sensor_test to confirm the output of a sensor's image.

** Note that sensor_test is only available in non-fast boot mode. When operating on Linux, the print will be output on alios **

The PC tool CvitekRawViewer is required for image viewing, the link is: CvitekRawViewer.

Note: If you have commented out the AE-related functions earlier, the manufacturer's default initial settings will be used. This may result in dark or completely black images. You may need to manually adjust the sensor's exposure and gain registers.

6.1 Dump RAW

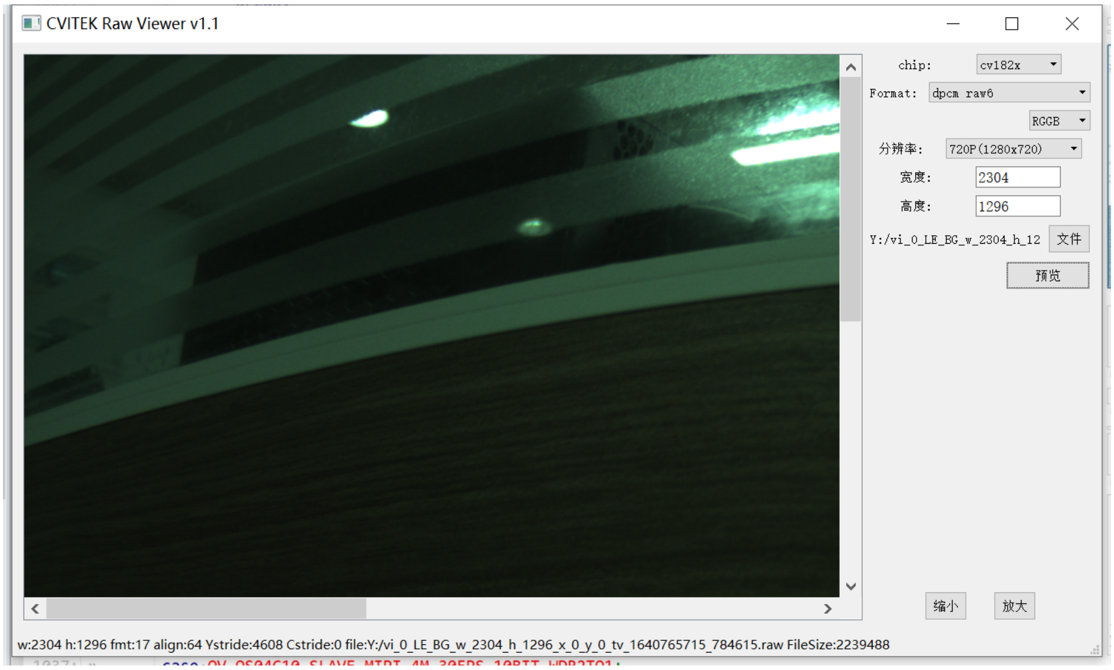
Run the sensor_test program, enter 1 to select "dump vi raw data", then follow the prompt "To get raw dump from dev(0~1):" and enter dev (0 represents vi pipe0, dump images from the first sensor, 1 represents vi pipe1, dump images from the second sensor).

Then according to the prompt "how many loops to do (1~60)", enter loops (indicating how many frames to dump).

RAW image viewing method:

To view the dumped raw image, use the CvitekRawViewer tool on the computer and configure the corresponding processor, format, width, and height.

The tool is used as shown in the figure below:



Note:

a. The displayed raw image should have a greenish bias. If it appears purplish or has diagonal lines, the configuration of the Bayer format, flip/mirror, and related settings should be checked.

b. The width, height, and color format can generally be obtained from the dumped file name.

c. By default, sensor_test uses the raw image compression mode COMPRESS_MODE_TILE, so "dpcm raw6" should be selected in the tool. If compression mode is not enabled, "raw12" should be selected.

6.2 Dump YUV

Run sensor_test, select "dump vi yuv" by inputting 2, and follow the prompts to dump yuv images:

Use CvitekRawViewer tool on your computer to configure the corresponding processor, format, width, and height.


CVITEK Raw Viewer v1.1
chip: cv182x
Format: NV21
分辨率: 720P(1280x720)
宽度: 2304
高度: 1296
align: 64
Y:/sample_0.yuv
文件
预览
缩小
放大
w:2304 h:1296 fmt:11 align:64 Ystride:2304 Cstride:2304 file:Y:/sample_0.yuv FileSize:4478976

7 Basic Functions of ISP

The functionality of the sensor driver is implemented by operation callbacks. This chapter describes the basic functions that should be implemented by the ISP callbacks, assuming that the user is familiar with the Sensor datasheet. When debugging the ISP-related callbacks, please reopen the SAMPLE_COMM_ISP_Run and xxx_default_reg_init calls that were previously commented out.

7.1 Development Process

Please implement the following basic ISP callbacks in order:

1. pfn_cmos_sensor_init
2. pfn_cmos_sensor_exit
3. pfn_cmos_sensor_global_init
4. pfn_cmos_set_image_mode
5. pfn_cmos_set_wdr_mode
6. pfn_cmos_get_isp_default
7. pfn_cmos_get_sns_reg_info

7.2 Notes

- pfn_cmos_sensor_init - Implement the vendor-provided initialization sequence using the sensor communication interface (I2C/SPI). The correctness of the communication interface structure should be noted. Because AE-related callbacks will also be called before the sensor initialization, the sensor AE buffer should be set before the sensor starts outputting data. Refer to the xxxx_default_reg_init in the xxxx_sensor_ctrl.c.
- pfn_cmos_sensor_exit - Close the communication interface used.
- pfn_cmos_sensor_global_init - Initialize the sensor driver parameters.
- pfn_cmos_set_image_mode - Set the output format of the sensor. The sensor driver should choose the closest resolution as the output format.
- pfn_cmos_set_wdr_mode - Set whether the sensor output is in WDR mode

- pfn_cmos_get_isp_default - Provide ISP parameters related to the sensor.
- pfn_cmos_get_sns_reg_info - Provide AE synchronization information stored in the sensor driver. To synchronize the AE settings with the sensor output image, when the AE callbacks are called, the sensor driver does not immediately write to the sensor buffer, but stores the modified settings. The firmware will call pfn_cmos_get_sns_reg_info at a fixed period to obtain synchronization information and pass it to the kernel space ISP driver. The ISP driver is responsible for synchronously writing to the sensor buffer. In addition, the sensor may have different WDR output formats, so the size of the image, crop position, and MIPI-RX settings may be recalculated and set with different exposure values. The sensor driver should ask the vendor for the calculation formula, and the ISP driver will update the corresponding module accordingly.
- The structure returned by pfn_cmos_get_sns_reg_info is divided into three categories:

```
typedef struct _ISP_SNS_SYNC_INFO_S {
  ISP_SNS_REGS_INFO_S snsCfg;
  ISP_SNS_ISP_INFO_S ispCfg;
  ISP_SNS_CIF_INFO_S cifCfg;
} ISP_SNS_SYNC_INFO_S;
```

snsCfg represents the sensor buffers that need to be synchronized, ispCfg represents the Crop information that needs to be synchronized, and cifCfg represents the mipi-rx settings that need to be synchronized. When need_update is True, it means that the synchronization data of this type needs to be updated by ISP at the specified u8DelayFrmNum. Each buffer in snsCfg also has bUpdate to indicate whether the buffer needs to be updated.

The first call to pfn_cmos_get_sns_reg_info will configure the I2C-related messages, establish register address mapping, and obtain information such as sns, crop, and WDR size, as shown in the following figure.

```c
static CVI_S32 cmos_get_sns_regs_info(VI_PIPE ViPipe, ISP_SNS_SYNC_INFO_S *pstSnsSyncInfo
{
	CVI_S32 i;
	ISP_SNS_STATE_S *pstSnsState = CVI_NULL;
	ISP_SNS_REGS_INFO_S *pstSnsRegsInfo = CVI_NULL;
	ISP_SNS_SYNC_INFO_S *pstCfg0 = CVI_NULL;
	ISP_SNS_SYNC_INFO_S *pstCfg1 = CVI_NULL;
	ISP_I2C_DATA_S *pstI2c_data = CVI_NULL;

	CMOS_CHECK_POINTER(pstSnsSyncInfo);
	GC2053_SENSOR_GET_CTX(ViPipe, pstSnsState);
	CMOS_CHECK_POINTER(pstSnsState);
	pstSnsRegsInfo = &pstSnsSyncInfo->snsCfg;
	pstCfg0 = &pstSnsState->astSyncInfo[0];
	pstCfg1 = &pstSnsState->astSyncInfo[1];
	pstI2c_data = pstCfg0->snsCfg.astI2cData;

	if ((pstSnsState->bSyncInit == CVI_FALSE) || (pstSnsRegsInfo->bConfig == CVI_FALSE)) {
		pstCfg0->snsCfg.enSnsType = ISP_SNS_I2C_TYPE;
		pstCfg0->snsCfg.unComBus.s8I2cDev = g_aunGc2053_BusInfo[ViPipe].s8I2cDev;
		pstCfg0->snsCfg.u8Cfg2ValidDelayMax = 0;
		pstCfg0->snsCfg.use_snsr_sram = CVI_TRUE;
		pstCfg0->snsCfg.u32RegNum = LINEAR_REGS_NUM;

		for (i = 0; i < pstCfg0->snsCfg.u32RegNum; i++) {
			pstI2c_data[i].bUpdate = CVI_TRUE;
			pstI2c_data[i].u8DevAddr = gc2053_i2c_addr;
			pstI2c_data[i].u32AddrByteNum = gc2053_addr_byte;
			pstI2c_data[i].u32DataByteNum = gc2053_data_byte;
		}

		pstI2c_data[LINEAR_EXP_H].u32RegAddr = GC2053_EXP_H_ADDR;
		pstI2c_data[LINEAR_EXP_L].u32RegAddr = GC2053_EXP_L_ADDR;
		pstI2c_data[LINEAR_AGAIN_H].u32RegAddr = GC2053_AGAIN_H_ADDR;
		pstI2c_data[LINEAR_AGAIN_L].u32RegAddr = GC2053_AGAIN_L_ADDR;
		pstI2c_data[LINEAR_COL_AGAIN_H].u32RegAddr = GC2053_COL_AGAIN_H_ADDR;
		pstI2c_data[LINEAR_COL_AGAIN_L].u32RegAddr = GC2053_COL_AGAIN_L_ADDR;
		pstI2c_data[LINEAR_DGAIN_H].u32RegAddr = GC2053_DGAIN_H_ADDR;
		pstI2c_data[LINEAR_DGAIN_L].u32RegAddr = GC2053_DGAIN_L_ADDR;
		pstI2c_data[LINEAR_VTS_H].u32RegAddr = GC2053_VTS_H_ADDR;
		pstI2c_data[LINEAR_VTS_L].u32RegAddr = GC2053_VTS_L_ADDR;

		pstSnsState->bSyncInit = CVI_TRUE;
		pstCfg0->snsCfg.need_update = CVI_TRUE;
		/* recalcualte WDR size */
		cmos_get_wdr_size(ViPipe, &pstCfg0->ispCfg);
		pstCfg0->ispCfg.need_update = CVI_TRUE;
	} «end if (pstSnsState->bSyncIn...» else {
```

Subsequent calls to pfn_cmos_get_sns_reg_info are used to temporarily store modified AE register information, as shown in the following diagram.

```c
	} «end if (pstSnsState->bSyncIn...» else {

		CVI_U32 gainsUpdate = 0;

		pstCfg0->snsCfg.need_update = CVI_FALSE;
		for (i = 0; i < pstCfg0->snsCfg.u32RegNum; i++) {
			if (pstCfg0->snsCfg.astI2cData[i].u32Data == pstCfg1->snsCfg.astI2cData[i].u32Data) {
				pstCfg0->snsCfg.astI2cData[i].bUpdate = CVI_FALSE;
			} else {

				if ((i >= LINEAR_AGAIN_H) && (i <= LINEAR_DGAIN_L))
					gainsUpdate = 1;

				pstCfg0->snsCfg.astI2cData[i].bUpdate = CVI_TRUE;
				pstCfg0->snsCfg.need_update = CVI_TRUE;
			}
		}

		if (gainsUpdate) {
			pstCfg0->snsCfg.astI2cData[LINEAR_AGAIN_H].bUpdate = CVI_TRUE;
			pstCfg0->snsCfg.astI2cData[LINEAR_AGAIN_L].bUpdate = CVI_TRUE;
			pstCfg0->snsCfg.astI2cData[LINEAR_COL_AGAIN_H].bUpdate = CVI_TRUE;
			pstCfg0->snsCfg.astI2cData[LINEAR_COL_AGAIN_L].bUpdate = CVI_TRUE;
			pstCfg0->snsCfg.astI2cData[LINEAR_DGAIN_H].bUpdate = CVI_TRUE;
			pstCfg0->snsCfg.astI2cData[LINEAR_DGAIN_L].bUpdate = CVI_TRUE;
		}

		pstCfg0->ispCfg.need_update = (sensor_cmp_wdr_size(&pstCfg0->ispCfg, &pstCfg1->ispCfg) ?
				CVI_TRUE : CVI_FALSE);
	} «end else»
```

The temporarily stored AE register information will eventually be updated to the ISP driver by calling isp_snsSync_info_set, and the ISP driver will set the sensor register by sending an I2C command after delayFrmNum.

- pfn_cmos_get_isp_black_level - Retrieve the black level offset from the sensor spec. Convert the offset to a 12-bit value and use it in the formula to obtain the gain: gain = 4095 /

(4095 - offset) * 1024.

```
static ISP_CMOS_BLACK_LEVEL_S g_stIspBlcCalibratio = {
	.bUpdate = CVI_TRUE,
	.blcAttr = {
		.Enable = 1,
		.enOpType = OP_TYPE_AUTO,
		.stManual = {257, 257, 257, 257, 1093, 1093, 1093, 1093},
		.stAuto = {
			{257, 257, 257, 257, 259, 259, 260, 267, 278, 298, 366, 383, 366, 373, 372, 372 },
			{257, 257, 257, 257, 258, 259, 261, 266, 274, 297, 379, 377, 372, 365, 373, 374 },
			{257, 257, 257, 257, 258, 259, 261, 266, 275, 296, 376, 388, 366, 374, 376, 372 },
			{257, 257, 257, 257, 258, 259, 260, 264, 274, 294, 362, 363, 365, 361, 353, 367 },
			{1093, 1093, 1093, 1093, 1093, 1093, 1093, 1095, 1099, 1104, 1125, 1130, 1125, 1127, 1126, 1126},
			{1093, 1093, 1093, 1093, 1093, 1093, 1094, 1095, 1097, 1104, 1128, 1128, 1126, 1124, 1127, 1127},
			{1093, 1093, 1093, 1093, 1093, 1093, 1094, 1095, 1098, 1104, 1128, 1131, 1125, 1127, 1128, 1126},
			{1093, 1093, 1093, 1093, 1093, 1093, 1093, 1095, 1097, 1103, 1123, 1124, 1124, 1123, 1121, 1125},
		},
	},
};
```

8 Complete the AE Configuration Function

The functionality of the sensor driver is implemented through operation callbacks. This section assumes that the user is familiar with the sensor datasheet and describes the basic functions that should be implemented by the AE callbacks.

8.1 Development Process

Please implement the following basic AE functional callbacks in order.

1. pfn_cmos_get_ae_default
2. pfn_cmos_fps_set
3. pfn_cmos_inttime_update
4. pfn_cmos_gains_update
5. pfn_cmos_again_calc_table
6. pfn_cmos_dgain_calc_table
7. pfn_cmos_get_inttime_max

8.2 Notes

- pfn_cmos_get_ae_default - Returns sensor data related to AE algorithm.

It is required to provide the maximum and minimum number of exposure steps in linear mode of AE algorithm, the maximum and minimum values and types of gain in linear/WDR mode simulation/digital gain. If the digital gain only has a few choices such as 0dB, 6dB, 12dB, etc., it is a DB type, otherwise it is linear. Also, the number of frames in the exposure effective period, and the number of frames after the start-up which is stable.

u32FullLinesStd: Number of lines in one frame at initialization.

u32MaxAgain: Maximum AGain value.

u32MinAgain: Minimum AGain value.

u32MaxDgain: Maximum DGain value.

u32MinDgain: Minimum DGain value.

u32MaxIntTime: Maximum exposure value in linear mode.

u32MinIntTimeTarget: Minimum exposure value in linear mode.

u32AEResponseFrame: Maximum AE response time (unit: frame).

The main task is to fill in the relevant AE properties according to the sensor spec, including FullLinesStd, FullLinesMax, max/min/step values for IntTime, as well as max/min/step values for gain. It is important to confirm the AccuType for IntTime and gain:

```
typedef enum _AE_ACCURACY_E {
    AE_ACCURACY_DB = 0,
    AE_ACCURACY_LINEAR,
    AE_ACCURACY_TABLE,
    AE_ACCURACY_BUTT,
} AE_ACCURACY_E;
```

In general, the intTime setting is linearly related to the corresponding register, with AccuType set to AE_ACCURACY_LINEAR. The gain setting is usually set to AE_ACCURACY_TABLE, indicating mapping from the gain table, and we will introduce pfn_cmos_again_calc_table/pfn_cmos_dgain_calc_table later. However, some sensors may have special gain settings, such as the SOI_F35, which can only be adjusted in four steps: 1x, 2x, 3x, and 4x.

0D	DVP2	50	RW	DVP control 2. DVP2[7:4]: P-Pump and N-Pump clock selection. DVP2[3:2]: PAD drive capability. "00": min, "11": max. DVP2[1:0]: Digital gain. "00":1x, "01" and "10": 2x, "11":4x

- pfn_cmos_fps_set - Sets the frame rate of the sensor.

The default is the maximum frame rate of the Sensor output mode. The Sensor driver can reduce the frame rate by increasing the number of vertical blanking lines in the output. Note that changing the total number of output lines may also change the exposure range of some sensors, and the Sensor driver must recalculate it. For example, if the initial sequence has an FPS of 30, the new FPS cannot be greater than 30. The usual method of adjusting the frame rate is to increase the sensor output full lines in proportion. For example, if full lines = 1125 at FPS=30, the full lines at FPS=25 would be 1125*30/25 = 1350.

- pfn_cmos_inttime_update - Sets the exposure time of the sensor and returns the actual number of exposure lines to the AE.

The input parameter is a sequence, which represents the exposure values of short and long exposure frames in order in WDR mode, in units of horizontal output lines. For example, when u32IntTime[0]=8 and u32IntTime[1]=1000, it means that the exposure time for the short exposure frame is 8 lines and for the long exposure frame is 1000 lines. If in linear mode, the value in sequence[0] represents the exposure value, and sequence[1] is meaningless. Note that in WDR mode, adjusting the exposure of the short frame of the sensor may require recalculation of the Crop information and MIPI-RX settings.

- pfn_cmos_gains_update - Set the gain value for the sensor.

The input parameters are two arrays: pu32Again and pu32Dgain. In WDR mode, pu32Again[0] represents the analog gain value of the short exposure frame, pu32Again[1] represents the analog gain value of the long exposure frame; pu32Dgain[0] represents the digital gain value of the short exposure frame, and pu32Dgain[[1] represents the digital gain value of the long exposure frame. The values are the settings in the sensor buffer and can be converted to real gain values by pfn_cmos_again_calc_table and pfn_cmos_dgain_calc_table. In linear mode, only pu32Again[0] and pu32Dgain[0] are meaningful.

They can be converted to real gain values by pfn_cmos_again_calc_table and pfn_cmos_dgain_calc_table. In linear mode, only pu32Again[0] and pu32Dgain[0] are meaningful.

In WDR mode:

pu32Again[0]: Gain configuration for short frame.

pu32Again[1]: Gain configuration for long frame.

pu32Dgain[0]: Dgain configuration for short frame.

pu32Dgain[1]: Dgain configuration for long frame.

There are 3 modes for gains update - SHARE, WDR_2F, ONLY_LEF, which are set in pfnSetInit.

SHARE: Both short and long frames share the same gain configuration (Sony, OV).

WDR_2F: Short and long frames have separate gain configurations (Sony, OV).

ONLY_LEF: Only the gain for long exposure frame can be configured (SOI).

- pfn_cmos_again_calc_table - Input is the analog gain value based on a reference of 1024. The sensor driver searches a lookup table or calculates the analog gain value that is closest and not greater than the input value, and outputs the corresponding sensor buffer setting.

  pu32AgainLin: AE passes in the 1024-based Again value. The Sensor driver calculates the closest 1024-based Again value based on the gain table or formula specified in the datasheet and returns it. The range of Again is defined in pfn_cmos_get_ae_default.

  pu32AgainDb: Returns the corresponding Sensor Again register configuration.

- pfn_cmos_dgain_calc_table - The input is a 1024-based digital gain value. The sensor driver looks up or calculates the closest digital gain value that is not greater than the input value and outputs the corresponding sensor buffer setting.

  pu32DgainLin: AE passes in 1024-based Dgain. The Sensor driver calculates the closest 1024-based Dgain based on the gain table or formula specified in the specification and returns it. The range of Dgain is defined in pfn_cmos_get_ae_default.

  pu32DgainDb: Returns the corresponding Sensor Dgain register configuration. If the sensor Dgain adjustment is step-wise (1X, 2X, 4X, etc.), the stDgainAccu.enAccuType in pfn_cmos_get_ae_default must be set to AE_ACCURACY_DB.

- pfn_cmos_get_inttime_max - Used in WDR mode to calculate the range of permissible exposure lines for the short and long frames at the current exposure ratio.

  SONY DOL, F35 HDR without VC, OV HDR-DT, Smartsens SC200AI all use the blanking interval to achieve short frame exposure.

  For some sensors (OS08A20, F35), they can be set to a fixed L2S distance, which means setting a maximum short frame exposure value. When adjusting the short frame exposure, the L2S distance will not change, and the ISP crop size does not need to be dynamically configured.

u16ManRatioEnable: Manual Ratio Enable, set to 1.

au32Ratio[0]: For 2-frame HDR, long frame exposure * 64 / short frame exposure.

au32IntTimeMax[0]: The maximum exposure value for the short frame (unit: one H time).

au32IntTimeMax[1]: The maximum exposure value for the long frame (unit: one H time).

au32IntTimeMin[0]: The minimum exposure value for the short frame (unit: one H time).

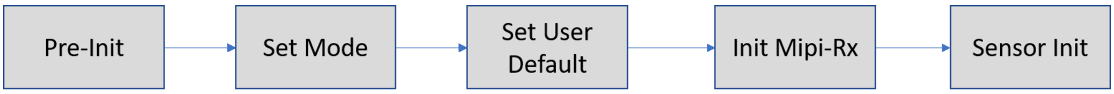
au32IntTimeMin[1]: The minimum exposure value for the long frame (unit: one H time).

pu32LFMaxIntTime[0]: NA.

9 Complete Other Functions

9.1 Sensor Initialization Process

In addition to AE/ISP, the sensor driver also uses other callbacks to complete the initialization process. Some parameter settings in sensor callbacks may affect each other, so the order of calling needs to be carefully considered. The recommended call sequence is as follows:



During the pre-init phase, the environment for the Sensor driver is prepared and the following callbacks are called:

- pfnSetInit - Initializes common parameters for the sensor. The enGainMode determines the behavior of the sensor’ s gain in WDR mode.
- pfnSetBusInfo - Sets I2C information.
- pfnRegisterCallback - Registers the sensor ISP/AE callbacks.
- pfn_cmos_sensor_global_init - Initializes internal parameters of the sensor driver.

Set Mode determines the main output format of the sensor, and the following callbacks are called:

- pfn_cmos_set_image_mode - Sets the output image format.
- pfn_cmos_set_wdr_mode - Sets the linear or WDR mode.

Set User Default is used to set the AE parameters for the initialization sequence, and the following callbacks are called:

- pfn_cmos_fps_set - Sets the frame rate per second. The default frame rate f32Fps is obtained from the callback pfn_cmos_get_ae_default, and the new frame rate must not be greater than the default value.
- pfn_cmos_inttime_update - Sets the number of exposure lines and returns it to AE. In linear mode, the exposure line count range can be obtained from u32MaxIntTime and u32MinIntTime in pfn_cmos_get_ae_default. In WDR mode, pfn_cmos_get_inttime_max can be called to obtain the exposure line count range for long and short exposures based on the exposure ratio.
- pfn_cmos_gains_update - Sets the Sensor’ s AGAIN and DGAIN. The Gain range can be obtained from u32MaxAgain/u32MaxDgain and u32MinAgain/u32MinDgain in

pfn_cmos_get_ae_default, and the closest Gain and corresponding Sensor buffer settings can be obtained from pfn_cmos_again_calc_table/pfn_cmos_dgain_calc_table.

Init Mipi-Rx initializes Mipi-Rx parameters and the Sensor's Power On Sequence by calling the Mipi-Rx driver in the kernel via ioctl. The main program steps are as follows:

- Open /dev/video0, which opens VIP-related power and clock sources.
- Call the callback pfnGetRxAttr in the Sensor driver to obtain the corresponding Mipi-Rx settings.
- CVI_MIPI_RESET_SENSOR - ioctl for Mipi-Rx, calling it opens the Sensor Reset pin defined in the device tree.

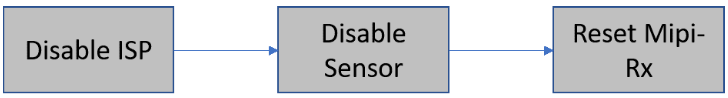
```
 mipi_rx: cif {
        compatible = "cvitek,cif";
        reg = <0x0 0x0a0c2000 0x0 0x2000>, <0x0 0x0300b000 0x0 0x1000>,
                <0x0 0x0a0c4000 0x0 0x2000>, <0x0 0x0300d000 0x0 0x1000>;
        reg-names = "csi_mac0", "csi_wrap0", "csi_mac1", "csi_wrap1";
        interrupts = <GIC_SPI 155 IRQ_TYPE_LEVEL_HIGH>, <GIC_SPI 156 IRQ_TYPE_LEVEL_HIGH>;
        interrupt-names = "csi0", "csi1";
        snsr-reset = <&portd 7 GPIO_ACTIVE_LOW>, <&portd 7 GPIO_ACTIVE_LOW>;
        resets = <&rst RST_CSIPHY0>, <&rst RST_CSIPHY1>,
                <&rst RST_CSIPHY0RST_APB>, <&rst RST_CSIPHY1RST_APB>;
        reset-names = "phy0", "phy1", "phy-apb0", "phy-apb1";
};
```

- CVI_MIPI_RESET_MIPI - ioctl for Mipi-Rx, calling it resets the Mipi-Rx settings.
- CVI_MIPI_SET_DEV_ATTR - ioctl for Mipi-Rx, calling it sets the Mipi-Rx properties.
- CVI_MIPI_ENABLE_SENSOR_CLOCK - ioctl for Mipi-Rx, calling it turns on the Sensor clock. The frequency is determined by the mclk attribute in CVI_MIPI_SET_DEV_ATTR.
- CVI_MIPI_UNRESET_SENSOR - ioctl for Mipi-Rx, calling it closes the Sensor Reset pin defined in the device tree.

To initiate the Sensor's initial sequence, the Sensor Init calls the callback pfn_cmos_sensor_init in the Sensor driver.

9.2 Sensor Shutdown Process

When closing the sensor, the following process can be referred to:

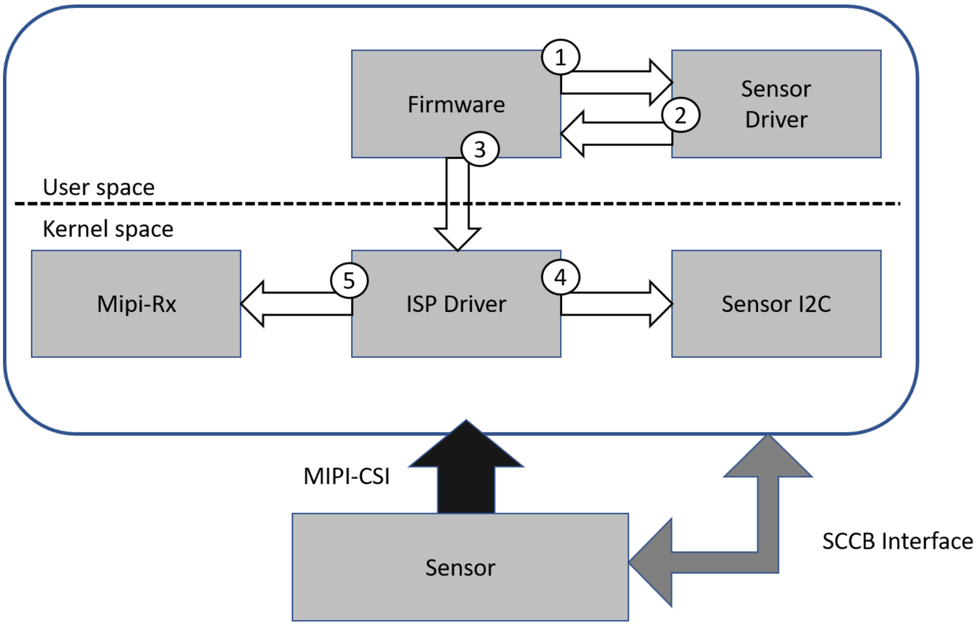


- Disable ISP - Disable the near-end ISP interface.
- Disable Sensor - Call the sensor driver's callback pfn_cmos_sensor_exit to close the sensor stream and I2C interface. Call pfnUnRegisterCallback to remove the sensor driver.

- Call Mipi-Rx ioctl CVI_MIPI_RESET_SENSOR to activate the Sensor reset pin. Call CVI_MIPI_DISABLE_SENSOR_CLOCK to turn off the Sensor clock. Call CVI_MIPI_RESET_MIPI to reset the Mipi-Rx settings.

9.3 Sensor AE Synchronization Process

Exposure and gain settings on the sensor may be reflected in different frames, so there needs to be a mechanism to synchronize the settings between the sensor and the ISP. In addition, in WDR Manual mode, adjusting the exposure of short-exposure frames may require updating the Mipi-Rx settings. The following is the Sensor AE synchronization process:



1. The firmware calls the sensor callbacks pfn_cmos_gains_update and pfn_cmos_inttime_update to update the AE settings.
2. The firmware calls the sensor callback pfn_cmos_get_sns_reg_info at fixed intervals to obtain the sensor/ISP/Mipi-Rx settings.
3. The firmware passes the sensor/ISP/CIF settings to the ISP driver’s synchronization processing mechanism via the ISP ioctl.
4. When it is necessary to update the sensor settings, the ISP driver calls the I2C interface in cv18xx_vip.ko to update the sensor cache.
5. When it is necessary to update the Mipi-Rx settings, the ISP driver calls the Mipi-Rx driver in cvi_mipi_rx.ko.

10 AE Related Verification

After completing the image verification, AE handover work can be performed. AE handover needs to ensure that the basic exposure and gain are linear, and that issues such as response frame and synchronization are verified.

The main task is to perform the verification of the SensorPorting_AE (sensor_test) table. The verification work requires the use of a light box and the sensor_test testing program.

Note: When performing AE related verification, the previously commented-out code needs to be released.

Excel file: 
SensorPorting_AE(sensor_test).xlsx

10.1 BLC Confirmation and Verification

The BLC offset value is usually specified in the sensor specification and can be directly written to xxx_cmos_param.h. If not specified, the actual BLC value can be obtained by the following method:

Modify xxx_cmos_param.h and change the values highlighted in red below: 273 represents the BLC offset and should be changed to 0, 1097 represents the gain and should be changed to 1024 for all instances.

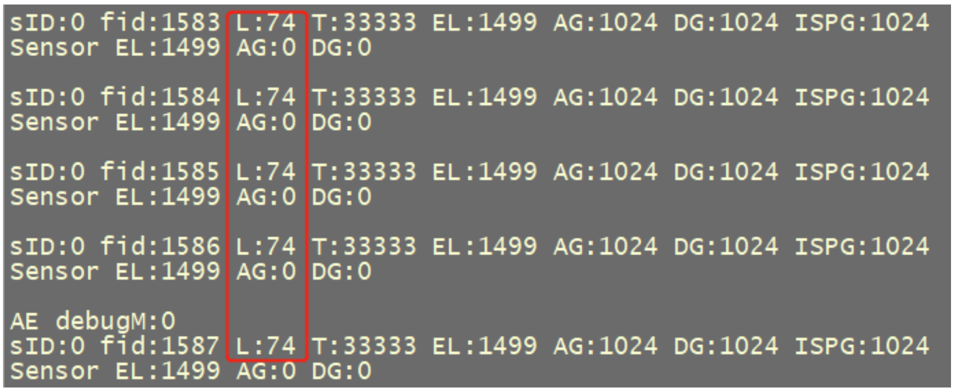
```
static ISP_CMOS_BLACK_LEVEL_S g_stIspBlcCalibratio = {
	.bUpdate = CVI_TRUE,
	.blcAttr = {
		.Enable = 1,
		.enOpType = OP_TYPE_AUTO,
		.stManual = {273, 273, 273, 273, 1097, 1097, 1097, 1097},
		.stAuto = {
			{273, 273, 273, 273, 273, 273, 273, 273, /*8*/273, 273, 273, 273, 273, 273, 273, 273},
			{273, 273, 273, 273, 273, 273, 273, 273, /*8*/273, 273, 273, 273, 273, 273, 273, 273},
			{273, 273, 273, 273, 273, 273, 273, 273, /*8*/273, 273, 273, 273, 273, 273, 273, 273},
			{273, 273, 273, 273, 273, 273, 273, 273, /*8*/273, 273, 273, 273, 273, 273, 273, 273},
			{1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097,
				/*8*/1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097},
			{1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097,
				/*8*/1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097},
			{1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097,
				/*8*/1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097},
			{1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097,
				/*8*/1097, 1097, 1097, 1097, 1097, 1097, 1097, 1097},
		},
	},
};
```

Cover the lens and run sensor_test in a completely dark environment, then enter CMD.

```
5
2 0 70 0 0
```

Printing out the Luma value and multiplying it by 4 will give you the corresponding BLC offset value.

For example, the corresponding blc offset below is 74x4=286.



```
sID:0 fid:1583 L:74 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:1584 L:74 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:1585 L:74 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:1586 L:74 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

AE debugM:0
sID:0 fid:1587 L:74 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0
```

Finally, the tested blc offset was substituted into the formula gain = 4095 / (4095 - offset) * 1024 to obtain 1106, and the confirmed blc and gain were filled in xxx_cmos_param.h.

10.2 Exposure Linearity Verification

Point the camera at the light box and run sensor_test and Type CMD in linear mode.

```
5
2 0 71 0 0
```

Enter CMD for long exposure in Wdr mode.

```
5
2 0 75 0 0
```

Enter CMD for short exposure in Wdr mode.

```
5
2 0 76 0 0
```

In order to satisfy the relationship that the exposure time of AE brightness statistics in 1/60s should be half of that in 1/30s, and the exposure time of AE brightness statistics in 1/120s should be half of that in 1/60s. For example, the results shown in the following figure are consistent.

```
sID:0 fid:59066 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59067 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59068 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59069 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59070 L:77 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0

sID:0 fid:59071 L:77 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0

sID:0 fid:59072 L:77 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0

sID:0 fid:59073 L:77 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0

sID:0 fid:59074 L:38 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0

sID:0 fid:59075 L:38 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0

sID:0 fid:59076 L:38 T:8333 EL:374 AG:1024 DG:1024 ISPG:1024
Sensor EL:374 AG:0 DG:0
```

```
sID:0 fid:59057 L:152 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:59058 L:152 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:59059 L:152 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:59060 L:152 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59061 L:152 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59062 L:152 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59063 L:152 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59064 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59065 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59066 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:59067 L:77 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
```

10.3 Gain Linearity Verification

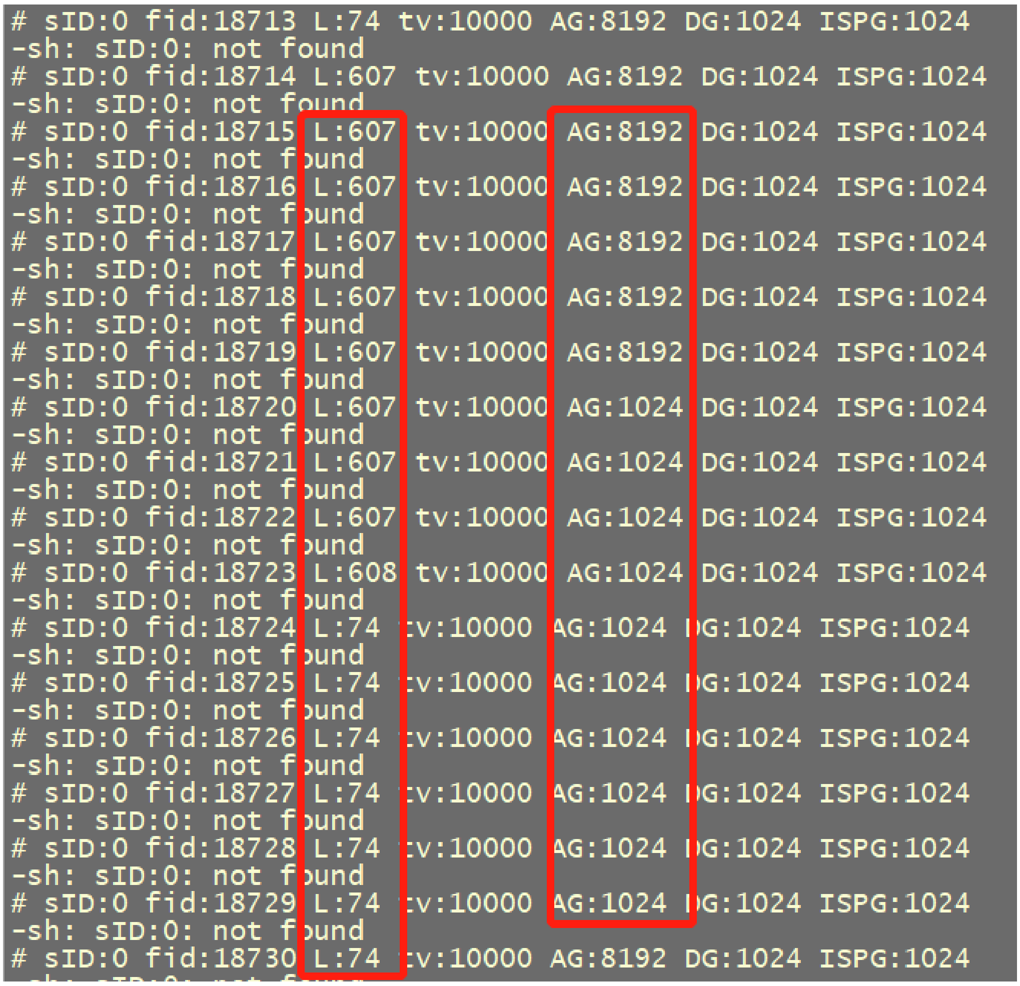
Point the camera at the light box and run sensor_test and type CMD in linear mode.

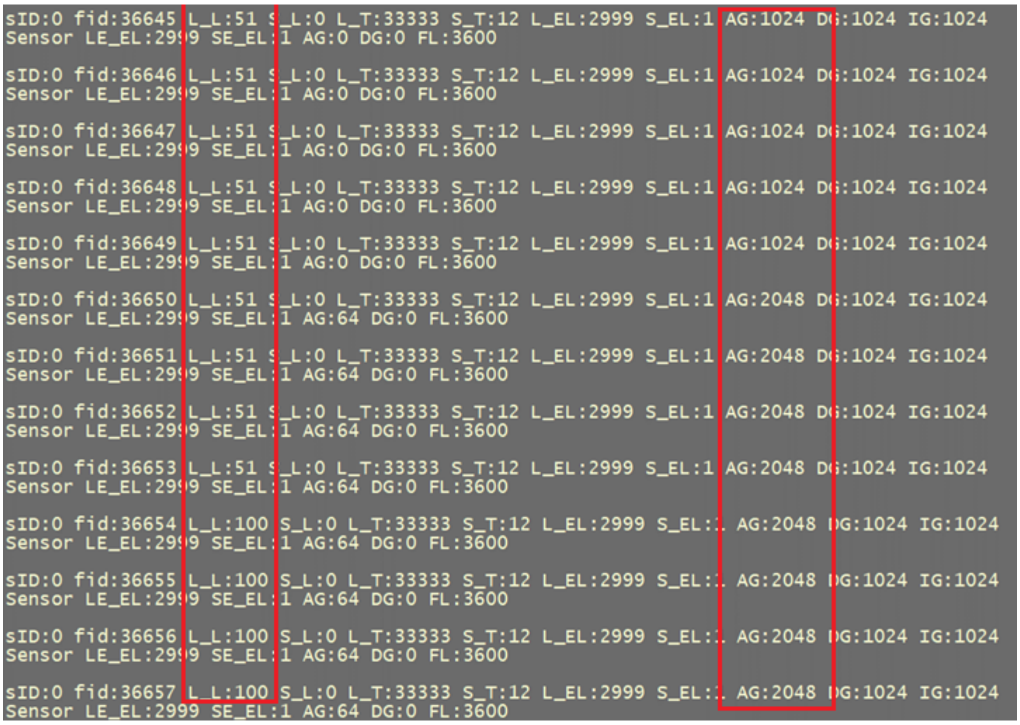
```
5
2 0 72 0 0
```

Enter CMD in Wdr mode.

```
5
2 0 77 0 0
```

The following figure shows the test results. In linear mode, again the value increases from 1024 to 8192, and the luma value changes from 74 to 607, basically conforming to the eight-fold relationship. In Wdr mode, again changed from 1024 to 2048, and luma changed from 51 to 100, basically conforming to the two-fold relationship. So luma and again are linear.

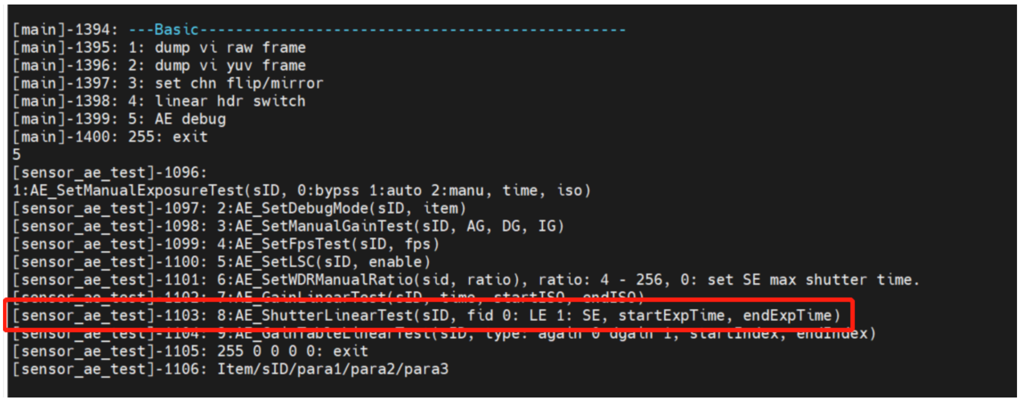




10.4 Advanced Verification

If you want to verify exposure linearity exactly, you need to use CMD.

```
5
8 SID FID startExpTime endExptime
```



Continuous exposure linearity can be tested, representing precision increments of %5 from startExpTime to endExpTime.

SID represents sensorID, FID represents frameID, 0 represents long frame, and 1 represents short frame.

If you want to verify the gain linearity exactly, you need to use CMD.

```
5
7 SID FID time StartISO EndISO
```

You can test continuous gain linearity, representing increments as you traverse the gaintable from StartISO to EndISO. Here, ISO 100 indicates 1x, that is, gain = 1024.

```
[main]-1394: ---Basic-------------------------------------------
[main]-1395: 1: dump vi raw frame
[main]-1396: 2: dump vi yuv frame
[main]-1397: 3: set chn flip/mirror
[main]-1398: 4: linear hdr switch
[main]-1399: 5: AE debug
[main]-1400: 255: exit
5
[sensor_ae_test]-1096:
1:AE_SetManualExposureTest(sID, 0:bypss 1:auto 2:manu, time, iso)
[sensor_ae_test]-1097: 2:AE_SetDebugMode(sID, item)
[sensor_ae_test]-1098: 3:AE_SetManualGainTest(sID, AG, DG, IG)
[sensor_ae_test]-1099: 4:AE_SetFpsTest(sID, fps)
[sensor_ae_test]-1100: 5:AE_SetLSC(sID, enable)
[sensor_ae_test]-1101: 6:AE_SetWDRManualRatio(sid, ratio), ratio: 4 - 256, 0: set SE max shutter time.
[sensor_ae_test]-1102: 7:AE_GainLinearTest(sID, time, startISO, endISO)
[sensor_ae_test]-1103: 8:AE_ShutterLinearTest(sID, fid 0: LE 1: SE, startExpTime, endExpTime)
[sensor_ae_test]-1104: 9:AE_GainTableLinearTest(sID, type: again 0 dgain 1, startIndex, endIndex)
[sensor_ae_test]-1105: 255 0 0 0 0: exit
[sensor_ae_test]-1106: Item/sID/para1/para2/para3
```

10.5 Response Frame Verification

Different sensor parameters take effect at different times. For example, some sensors take effect after 5 frame, while others take effect after 4 frame or 3 frame. Even if the same sensor is set in different registers, the effective time may be different. Therefore, it is necessary to verify the reaction frame of the register related to AE.

Run sensor_test and enter CMD in linear mode.

```
5
2 0 71 0 0
```

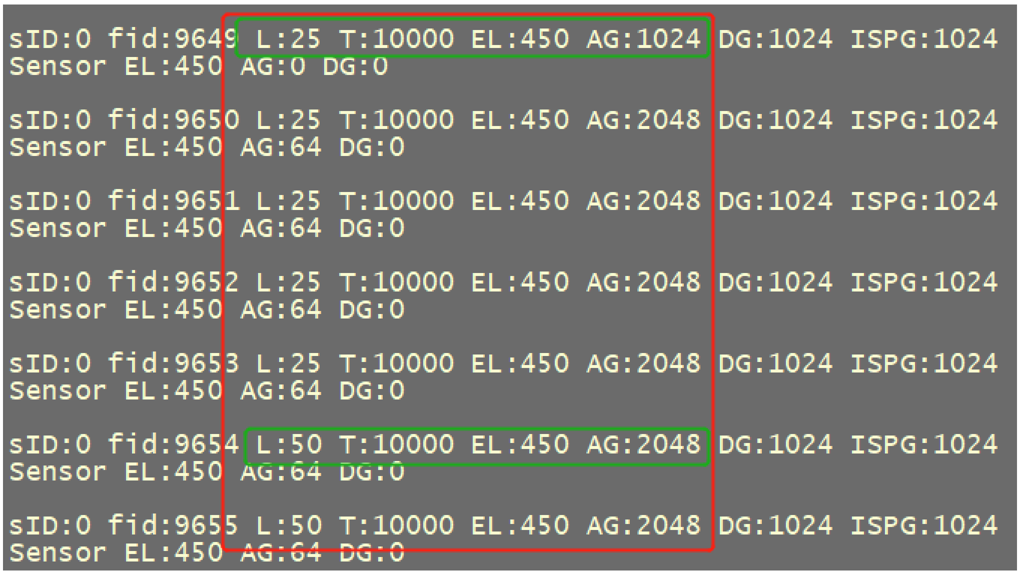
This measures how many frames have to pass before shutter is set to take effect. As you can see below, after shutter changes from 33333 to 16666, it takes 4 frames for the Luma value to change. So the exposed ResponseFrame is 4.

```
sID:0 fid:459 L:133 T:33333 EL:1499 AG:1024 DG:1024 ISPG:1024
Sensor EL:1499 AG:0 DG:0

sID:0 fid:460 L:133 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:461 L:133 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:462 L:133 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:463 L:133 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:464 L:70 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0

sID:0 fid:465 L:70 T:16666 EL:749 AG:1024 DG:1024 ISPG:1024
Sensor EL:749 AG:0 DG:0
```

Enter CMD.

```
5
2 0 72 0 0
```

This measures how many frames elapsed after the gain is set before it takes effect. As you can see from the figure below, after changing again from 1024 to 2048, it takes 4 frames for the Luma value to change. So the ResponseFrame for the gain is 4.



After the ResponseFrame of exposure and gain is tested, ResponseFrame is filled into the cmos_get_ae_default function of xxx_cmos.c, as shown below:

10.6 Validation of Exposure Gain Synchronization

Sometimes, not all sensors take effect with gain and shutter at the same time. For example, it may be possible that the ResponseFrame of gain is 4 and the ResponseFrame of shutter is 3, which requires verification by exposure gain synchronization mechanism.

Run sensor_test and enter CMD in linear mode.

```
5
2 0 73 0 0
```

This CMD indicates how long it takes for AE statistics to change while changing shutter/gain. In the figure below, you can see that changing shutter and again simultaneously takes effect after 4frames.

```
sID:0 fid:50998 L:479 T:33333 EL:1499 AG:8192 DG:1024 ISPG:102
Sensor EL:1499 AG:184 DG:0

sID:0 fid:50999 L:479 T:33333 EL:1499 AG:8192 DG:1024 ISPG:102
Sensor EL:1499 AG:184 DG:0

sID:0 fid:51000 L:479 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51001 L:479 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51002 L:479 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51003 L:479 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51004 L:3 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51005 L:3 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51006 L:3 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0

sID:0 fid:51007 L:3 T:1000 EL:45 AG:1024 DG:1024 ISPG:1024
Sensor EL:45 AG:0 DG:0
```

If the statistical value of AE changes after increasing gain and exposure at the same time, such as the following results, it indicates that gain and exposure do not take effect synchronously.

L:479 T:33333 AG:8192

L:479 T:33333 AG:8192

L:479 T:1000 AG:1024

L:479 T:1000 AG:1024

L:299 T:1000 AG:1024

L:211 T:1000 AG:1024

L:3 T:1000 AG:1024

L:3 T:1000 AG:1024

L:3 T:1000 AG:1024

L:3 T:1000 AG:1024

In this case, delay the gain or shutter to take effect. Modify the cmos_get_sns_regs_info function in xxx_cmos.c to change the delay setting of register.

For example, the following figure shows that the gain is delayed by 2 frames, indicating that it is 2 frames later than other register Settings.

```
pstI2c_data[LINEAR_SHS1_0_DATA].u32RegAddr = F35_SHS1_ADDR;
pstI2c_data[LINEAR_SHS1_1_DATA].u32RegAddr = F35_SHS1_ADDR + 1;
pstI2c_data[LINEAR_AGAIN_DATA].u32RegAddr = F35_GAIN_ADDR;
pstI2c_data[LINEAR_DGAIN_DATA].u32RegAddr = F35_DGAIN_ADDR;
pstI2c_data[LINEAR_DGAIN_DATA].u8DelayFrmNum = 2;
```

10.7 Verify FPS Controllability

Run with sensor_test and type CMD.

```
5
4 SID FPS
```

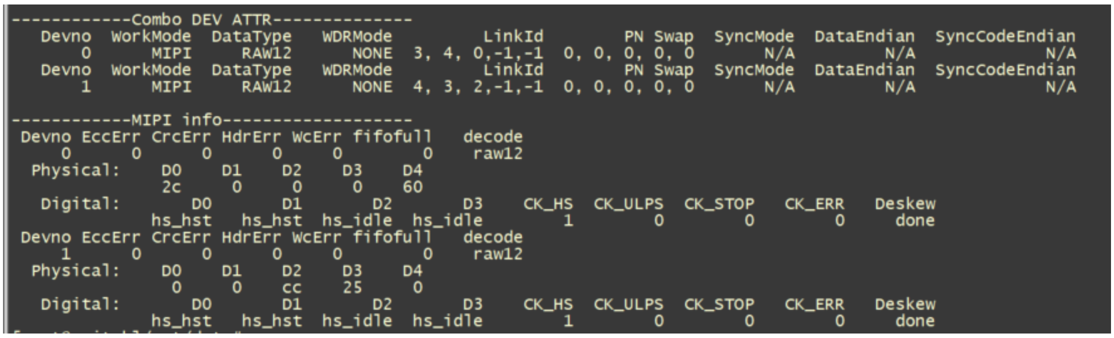
```
[main]-1394: ---Basic-----------------------------------------------
[main]-1395: 1: dump vi raw frame
[main]-1396: 2: dump vi yuv frame
[main]-1397: 3: set chn flip/mirror
[main]-1398: 4: linear hdr switch
[main]-1399: 5: AE debug
[main]-1400: 255: exit
5
[sensor_ae_test]-1096:
1:AE_SetManualExposureTest(sID, 0:bypss 1:auto 2:manu, time, iso)
[sensor_ae_test]-1097: 2:AE_SetDebugMode(sID, item)
[sensor_ae_test]-1098: 3:AE_SetManualGainTest(sID, AG, DG, IG)
[sensor_ae_test]-1099: 4:AE_SetFpsTest(sID, fps)
[sensor_ae_test]-1100: 5:AE_SetLSC(sID, enable)
[sensor_ae_test]-1101: 6:AE_SetWDRManualRatio(sid, ratio), ratio: 4 - 256, 0: set SE max shutter time.
[sensor_ae_test]-1102: 7:AE_GainLinearTest(sID, time, startISO, endISO)
[sensor_ae_test]-1103: 8:AE_ShutterLinearTest(sID, fid 0: LE 1: SE, startExpTime, endExpTime)
[sensor_ae_test]-1104: 9:AE_GainTableLinearTest(sID, type: again 0 dgain 1, startIndex, endIndex)
[sensor_ae_test]-1105: 255 0 0 0 0: exit
[sensor_ae_test]-1106: Item/sID/para1/para2/para3
```

The default fps is 25fps. You can check the output fps of the sensor by cat /proc/cvitek-vi_dbg.

```
# cat /proc/cvitek/vi_dbg
[VI Info]
VIOutImgWidth                :1920
VIOutImgHeight               :1080
VIIspTopStatus               :0x800
[VI ISP_PIPE_A]
VIInImgWidth                 :1948
VIInImgHeight                :1097
VISofCnt                     :42866
VIPreCnt                     :42865
VIPostCnt                    :42864
VIDevFPS                     :  25
VIFPS                        :  25
[VI ISP_PIPE_A CsiBdg_Debug_Info]
VICsiStatus                  :0x0
VICsiDebugStatus             :0x18
VICsiOverFlowCnt             :   0
VICsiWidthGTCnt              :   0
VICsiWidthLSCnt              :   0
VICsiHeightGTCnt             :   0
VICsiHeightLSCnt             :   0
VIPrerawStatus               :0x5ea9
[VI ISP_PIPE_A SW_State_Debug_Info]
VIPreOutBufEmpty             :   0
VIPostInBufEmpty             :   0
VIPostOutBufEmpty            :   0
VIPreSWstatus                :   1
VIPostSWstatus               :   1
VIPostIsRightTile            :   0
```

11 Common Problem

11.1 Proc Message Interpretation



```
------------Combo DEV ATTR--------------
   Devno  WorkMode  DataType   WDRMode          LinkId         PN Swap  SyncMode  DataEndian  SyncCodeEndian
       0      MIPI     RAW12      NONE  3, 4, 0,-1,-1  0, 0, 0, 0, 0       N/A         N/A             N/A
   Devno  WorkMode  DataType   WDRMode          LinkId         PN Swap  SyncMode  DataEndian  SyncCodeEndian
       1      MIPI     RAW12      NONE  4, 3, 2,-1,-1  0, 0, 0, 0, 0       N/A         N/A             N/A

------------MIPI info-------------------
 Devno EccErr CrcErr HdrErr WcErr fifofull   decode
    0      0      0      0     0        0    raw12
  Physical:     D0     D1     D2     D3     D4
                2c      0      0      0     60
   Digital:        D0          D1           D2          D3     CK_HS   CK_ULPS   CK_STOP    CK_ERR    Deskew
             hs_hst    hs_hst  hs_idle  hs_idle         1         0         0         0      done
 Devno EccErr CrcErr HdrErr WcErr fifofull   decode
    1      0      0      0     0        0    raw12
  Physical:     D0     D1     D2     D3     D4
                 0      0     cc     25      0
   Digital:        D0          D1           D2          D3     CK_HS   CK_ULPS   CK_STOP    CK_ERR    Deskew
             hs_hst    hs_hst  hs_idle  hs_idle         1         0         0         0      done
```

cat /proc/mipi-rx

Combo DEV ATTR mainly provides interface configuration information for the sensor:

Devno: indicates the sensor number. 0 indicates sensor0, and 1 indicates sensor1. Currently, only two sensors can be entered at the same time.

WorkMode: indicates the interface type (mipi/sublvds/ HISPI /BT656⋯).

DateType: indicates the sensor data format (raw8/raw10/raw12/ YUV422_8BIT⋯).

WDRMode: wdr mode (none indicates non-wDR, common wdr mode:VC, DT, Manual).

LinkId: lane sequence configuration.

PN swap: indicates PN reversal. If there is PN reversal, set the lane to 1.

SyncMode/DataEndian/SyncCodeEnddian: for mipi interface does not support so no configuration, for sublvds, hispi requires configuration.

MIPI INFO mainly refers to the information parsed by mipi-rx:

EccErr, CrcErr, HdrErr, WcErr: If the value is not 0, it indicates that Ecc,crc, and wc have been used to check err. Check the correctness of lane mapping, mipi timing, and lane hardware circuit.

Fifofull: If the value is not 0, the mac speed is too slow and the mac clk needs to be increased.

Decode: parsing the data type l (Raw12 / raw10 / raw8 / YUV422⋯).

PhySical: D0-D4 Indicates the data on the lane bus. After the hi speed state is entered, data changes in D0-D4.

Digital: D0-D4 Displays the status of each data lane after the hi speed state is entered. CK_HS, CK_ULPS, CK_ERR, and Deskew indicate the status of clk lane. Normally, CK_HS=1 and the rest value is 0, but CK_HS=1 and CK_STOP=1 continue.

11.2 The Open of Sensor-related Log

Enable cif drv log:

echo "module cvi_mipi_rx +p" > /sys/kernel/debug/dynamic_debug/control

dmesg -n 8

Enable syslog to print:

Output to serial port screen,

/sbin/syslogd -l 8 -s 2048 -O /dev/console

or output to file.

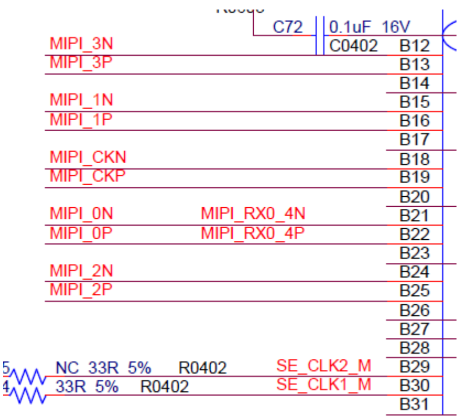
/sbin/syslogd -l 8 -s 2048 -O /mnt/data/mw.txt

11.3 How to Configure Lane Line Sequence

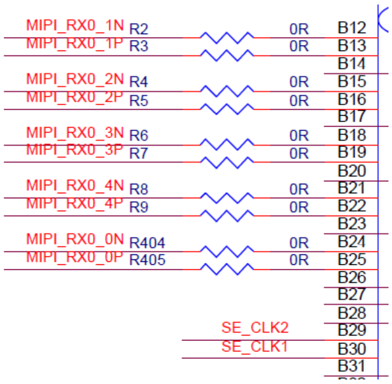
Note that the lane id to be configured should be configured with the sensor as the reference. The index number of lane_id array represents the Lane ID of the Sensor, the index number 0 represents the sensor clock, and the index number 1-4 represents sensor lane 0~3. The value of the land_id array indicates the Lane ID of MIPI-Rx of soc. 0 indicates MIPIRX1_PAD0 and 1 indicates MIPIRX1_PAD1. lane_id is set to -1 for unused lanes.

Assume that the lane connection of sensor and soc is shown in the figure below, and the corresponding lane id configuration is {3,4,2,0,1}.

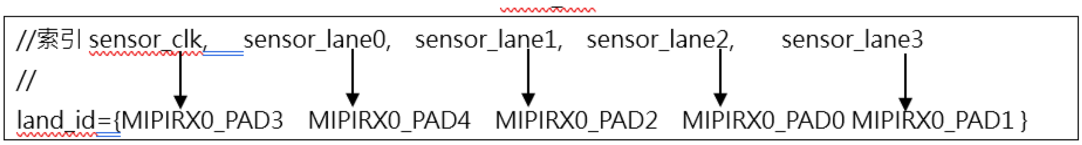
sensor:



soc:



SENSOR Pins	MIPI Lane Pins
MIPI_CK (index = 0)	MIPIRX0_3 (value = 0)
MIPI_0 (index = 1)	MIPIRX0_4 (value = 1)
MIPI_1 (index = 2)	MIPIRX0_2 (value = 2)
MIPI_2 (index = 3)	MIPIRX0_0 (value = 3)
MIPI_3 (index = 4)	MIPIRX0_1 (value = 4)



11.4 How to Select the MAC Frequency

MAC represents how often the isp receives data from the sensor,

Formula MAC_Freq * pix_width = lane_num * MIPI_Freq * 2.

MAC_Freq: VI MAC operating frequency.

pixel_width: pixel bit width.

lane_num: indicates the number of MIPI lanes.

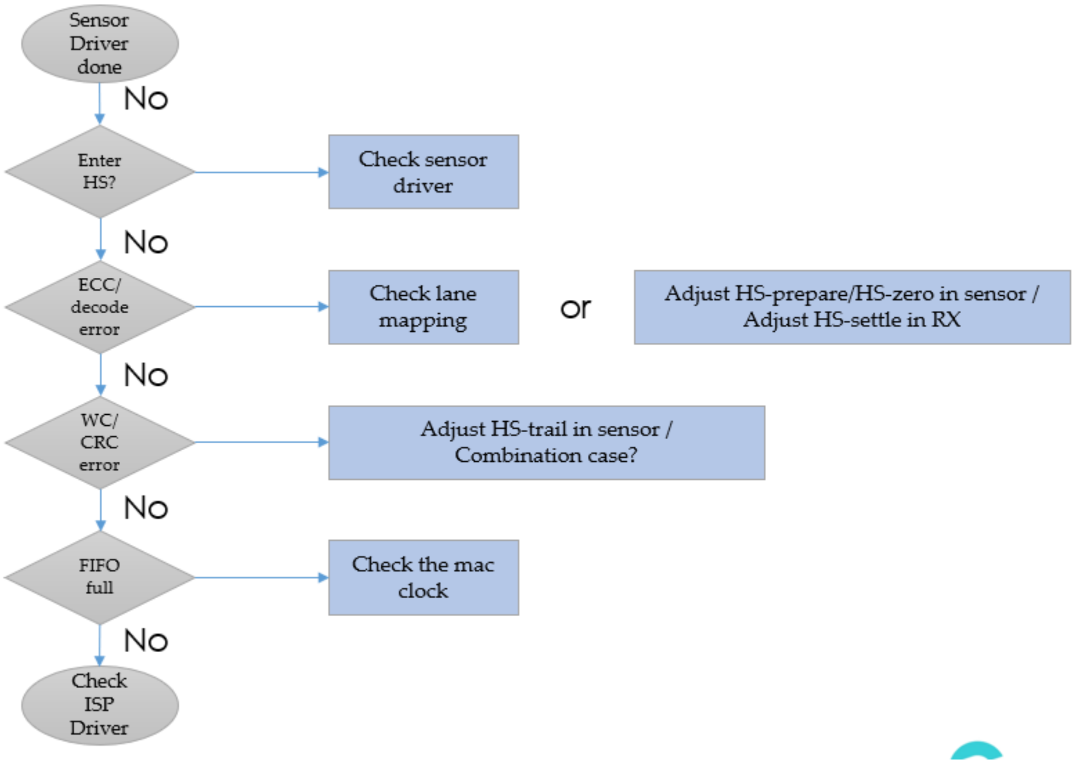
MIPI_Freq: operating frequency of each lane.

Assuming that the MAC freq is 400 M, pixel_width = 12, lane_num = 4, the maximum MIPI_Freq = 400 * 12 / (4 * 2) = 600MHz is supported.

Where MIPI_Freq means phy_Clk, the value is bps/2. For example, the specifications of sony imx335 are 1188Mbps per lane and phy_clk = 1188/2=594Mhz.

Conversely, if the sensor gives us the data rate, we need to be able to figure out the appropriate mac freq.

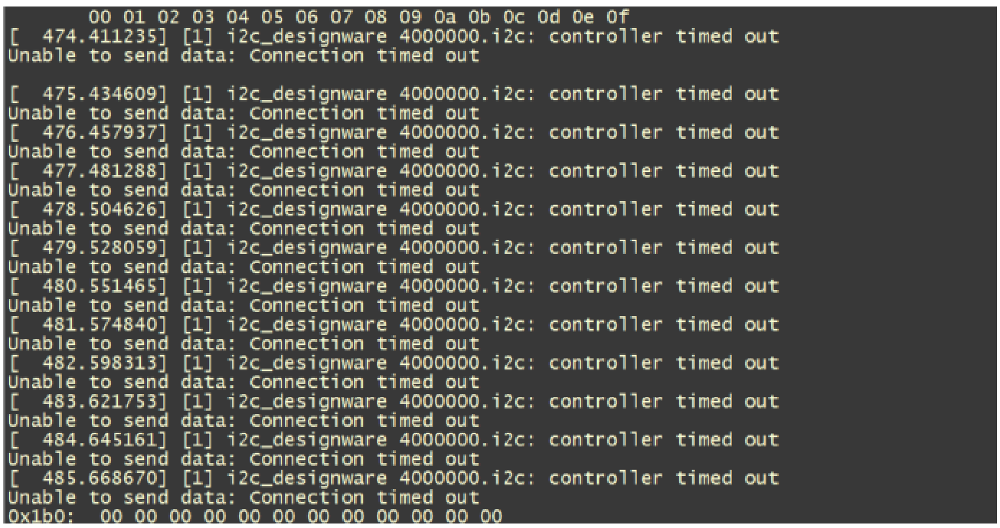
11.5 Error Checking Process



I2C Write Fail

- Sensor i2c attribute confirmation.
  - Check the I2C bus id.
  - Check the I2C slave addr.
  - Check the addr/data bit width of the sensor register (8bit or 16bit).

    If the bit width is incorrectly configured, a time out error is displayed.



```
     00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
[  474.411235] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out

[  475.434609] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  476.457937] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  477.481288] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  478.504626] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  479.528059] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  480.551465] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  481.574840] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  482.598313] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  483.621753] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  484.645161] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
[  485.668670] [1] i2c_designware 4000000.i2c: controller timed out
Unable to send data: Connection timed out
0x1b0:  00 00 00 00 00 00 00 00 00 00 00 00
```

- Check whether the hardware is normal.

- Verify that the rst, pwdn, and mclk pins in the dts are correctly configured.

  echo "snsr_on 0 1 1" > /proc/mipi-rx //1 indicates 37.125M, 2 indicates 25M, and 3 indicates 27M

  echo "snsr_on 1 1 1" > /proc/mipi-rx // 1 indicates 37.125M, 2 indicates 25M, and 3 indicates 27M

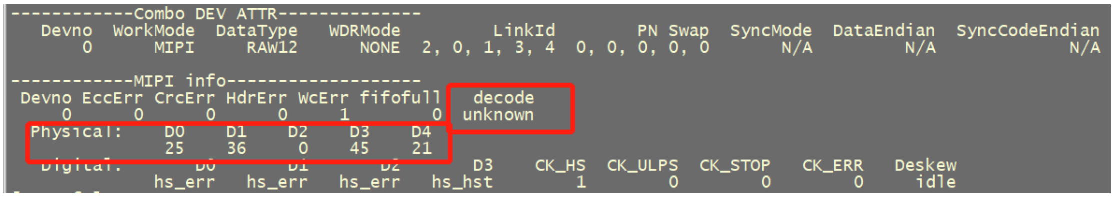
  echo "snsr_r 0 0" > /proc/mipi-rx

  echo "snsr_r 1 0" > /proc/mipi-rx

- Run the i2cdetect -y -r N command to test whether the i2c can detect the detection. N Indicates the i2c port corresponding to the sensor.
- Check if the power on timing meets spec requirements (measure MCLK and I2C with an oscilloscope).

Decode err

cat /proc/mipi-rx, check the proc message and check whether the Proc message is in hs-state. After the sensor is powered, it will enter the high speed state from the low power state. As shown in the following figure, if D0-D4 of mipi-rx has data and keeps changing, it indicates that hs-state is entered.



- Confirm i2c pathways (i2cdetect can sweep out sensor address).
- Confirm order right lane line.

  a. If the data lane in proc has no data jump and the accompanying CK_HS is 0, the clk lane is not found correctly (please confirm the clk lane).

  b. If there is data jump in the data lane in proc and CK_HS is 1, it means that the clk lane is found correctly and has entered hs mode. If ecc, crc and other errors occur, it means that the data lane is not configured correctly (please confirm the data lane).

- Confirm timing.

  c. If the previous two points are confirmed to be correct, but CK_HS =0 and there is no data jump in the data lane, the timing may not meet the conditions for entering hs. In this case, the value of hs-zero and hs-trail can be adjusted and increased to lengthen the detect period.

  d. If the first two points are confirmed to be correct, CK_HS =1, data lane has data jump, but there are still ecc, crc and other err, it may be that the setting of Hs-settle is too large or too small, and the data behind is pressed.

- Confirm whether the hw is damaged.

ECC err

- Check lane Id mapping.

- Check sensor tx hs-zero/hs-prepare.

  hs-zero and hs-parepare need to determine the value from sensor spec or directly ask the sensor manufacturer. It is not recommended to adjust the value.

- Check mipi-rx hs-settle.

  When the hs-settle time is too long, the "sync code" in the data will be pressed, and the "sync code" cannot be resolved, resulting in ecc err.

  Adjust hs-settle you can directly modify xxx_cmos_param.h as follows, fill in the correct hs_settle.

```
struct combo_dev_attr_s sc4210_rx_attr = {
	.input_mode = INPUT_MODE_MIPI,
	.mac_clk = RX_MAC_CLK_400M,
	.mipi_attr = {
		.raw_data_type = RAW_DATA_12BIT,
		.lane_id = {0, 4, 3, 2, 1},
		.wdr_mode = CVI_MIPI_WDR_MODE_VC,
		.dphy = {
			.enable = 1,
			.hs_settle = 8,
		},
	},
```

  You can also directly ctrl+z to adjust hs-settle, and use devmem command to modify the bit[23:16] value of register 0x0300b048. After adjustment, enter fg to jump back to the program.

```
devmem 0x0300b048 32 0xXYZ
```

78	h48	h48	REG_48			rstn	reg_prbs9_test_period	15	0	16	h00ff	rw
79						rstn	reg_t_hs_settle	23	16	8	h10	rw

CRC err/Word count err

Adjust the sensor tx hs-trail. If the hs-trail is pulled too fast, the data behind it may be pressed, resulting in data loss, resulting in crc err and wc err. You need to adjust the hs-trail register setting of the sensor.

```
------------MIPI info-------------------
 Devno EccErr CrcErr HdrErr WcErr fifofull   decode
    0      0      0      0      1        0  unknown
  Physical:     D0     D1     D2     D3     D4
                54      a      0     a8      b
   Digital:        D0         D1         D2         D3     CK_HS   CK_ULPS   CK_STOP    CK_ERR    Deskew
               hs_err     hs_err     hs_hst     hs_hst         1         0         0         0     start
```

vi_select timeout

- cat/proc/mipi - rx show whether there is the i2c, decode, ecc, CRC, wc etc. err. If the preceding 4 steps are correct, cat /proc/cvitek-vi_dbg checks for WidthGTCnt, WidthLSCnt, HeightGTCnt, and HeightLSCnt. If such error occurs, crop size in sensor init setting is inconsistent with the set given to isp. Please confirm the modification against sensor spec.
- Check whether MAC clock is too low, if the MAC clock is too low, can lead to an isp processing speed too slow in fifo full, can also lead to the timeout.

12 Color, Noise Reduction, and Other Corrections

Please refer to the "Image Quality Debugging Tool User Guide_v1.1.1".

13 Image Quality Tuning.

Please refer to the “Image Tuning Guide_V0.2.5” .

14 Debugging Tool

After developing the sensor, use the debugging tool "sensor_test" for testing.

The sensor configuration file is located at "/mnt/data/sensor_cfg.ini".

Apply the patch "sensor_test.patch" in the middleware directory using the "git apply" command, and compile to generate "sensor_test" for use.

Patch file: sensor_test.patch

14.1 Basic Functions.

By default, sensor_test has the following 5 functions, as shown in the figure below:

1. Dump sensor raw image.
2. Dump sensor YUV image.
3. Set flip/mirror for the sensor output image.
4. If the sensor driver supports linear and WDR modes, this option can be used to switch sensor modes.
5. AE debugging function.

```
[main]-1394: ---Basic---------------------------------------------
[main]-1395: 1: dump vi raw frame
[main]-1396: 2: dump vi yuv frame
[main]-1397: 3: set chn flip/mirror
[main]-1398: 4: linear hdr switch
[main]-1399: 5: AE debug
[main]-1400: 255: exit
```

14.2 Dump RAW

Refer to 5.1 Dump RAW.

14.3 Dump YUV

Refer to 5.2 Dump YUV.

14.4 Set flip/mirror

It provides mirror/flip functionality.

Run sensor_test and enter 3 to select “set chn flip/mirror” . Follow the prompt chn(0 to 1): Enter dev (0 indicates vi pipe0 to control channel 0, 1 indicates vi pipe1) to switch on flip/mirror.

Note: After the function is executed, ensure that the direction and color of the dump yuv diagram are as expected.

14.5 Switching between WDR and Linear

It provides the switch function between sensor end width dynamic mode and linear mode.

Run “sensor_test” and select option 4 “linear hdr switch” . Then, follow the prompt “Please select sensor input mode (0:linear/1:wdr) :” to enter 0 for Linear or 1 for WDR.

Note:

1. This function requires the sensor to support both Linear and WDR modes.
2. Different sensor configurations need to be modified in the “sensor_test.c” file, as shown in the figure below.

```c
static CVI_S32 sensor_linear_wdr_switch(void)
{
    int tmp;
    CVI_U8 wdrMode = 0;
    CVI_S32 s32Ret = CVI_SUCCESS;

    SAMPLE_COMM_VI_DestroyIsp(&g_stViConfig);
    // Stop VI.
    SAMPLE_COMM_VI_DestroyVi(&g_stViConfig);
    // Close ISP device.
    s32Ret = SAMPLE_COMM_VI_CLOSE();
    if (s32Ret != CVI_SUCCESS) {
        CVI_TRACE_LOG(CVI_DBG_ERR, "vi close failed. s32Ret: 0x%x !\n", s32Ret);
        return s32Ret;
    }
    // select which mode want to switch.
    printf("Please select sensor input mode (0:linear/1:wdr) :");
    scanf("%d", &tmp);
    wdrMode = tmp;
    if (wdrMode == 0) {
        // Reset main sensor initial config to linear setting.
        g_stIniCfg.enSnsType = SONY_IMX307_MIPI_2M_30FPS_12BIT;
        g_stIniCfg.enWDRMode = WDR_MODE_NONE;
        // Reset slave sensor initial config to linear setting.
        g_stIniCfg.enSns2Type = SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT;
        g_stIniCfg.enSns2WDRMode = WDR_MODE_NONE;
    } else {
        // Reset main sensor initial config to wdr setting.
        g_stIniCfg.enSnsType = SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1;
        g_stIniCfg.enWDRMode = WDR_MODE_2To1_LINE;
        // Reset slave sensor initial config to wdr setting.
        g_stIniCfg.enSns2Type = SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1;
        g_stIniCfg.enSns2WDRMode = WDR_MODE_2To1_LINE;
    }
```

14.6 AE Related Verification

Refer to AE Related Verification.