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	Add 181X/180X details

CHAPTER

ONE

DISCLAIMER

Terms and Conditions

The document and all information contained herein remain the CVITEK Co., Ltd' s ("CVITEK") confidential information, and should not disclose to any third party or use it in any way without CVITEK' s prior written consent.

User shall be liable for any damage and loss caused by unauthority use and disclosure.

CVITEK reserves the right to make changes to information contained in this document at any time and without notice.

All information contained herein is provided in "AS IS" basis, without warranties of any kind, expressed or implied, including without limitation mercantability, non-infringement and fitness for a particular purpose.

In no event shall CVITEK be liable for any third party' s software provided herein, User shall only seek remedy against such third party.

CVITEK especially claims that CVITEK shall have no liable for CVITEK' s work result based on Customer' s specification or published shandard.

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/

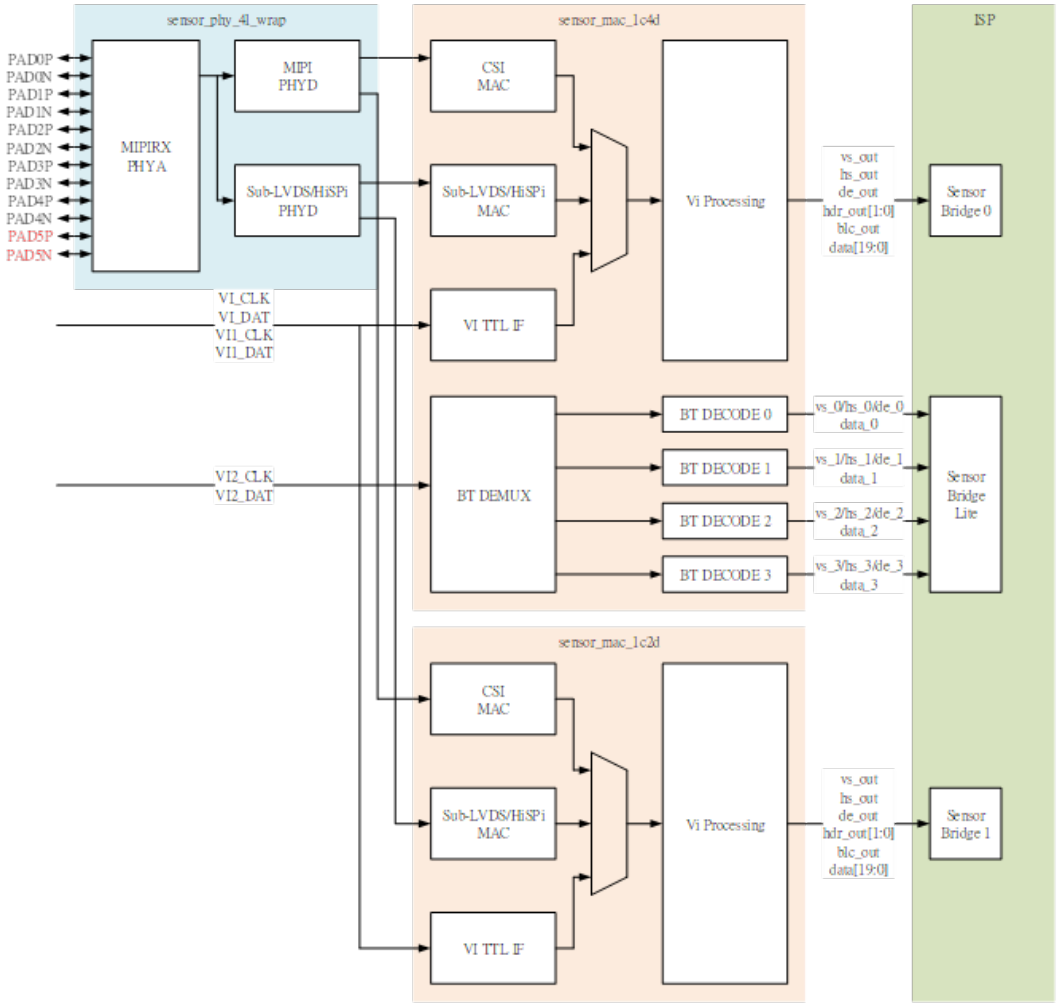
Forum https://developer.sophgo.com/forum/index.html

CHAPTER

TWO

INTRODUCTION TO SENSOR DRIVERS

2.1 Hardware Architecture

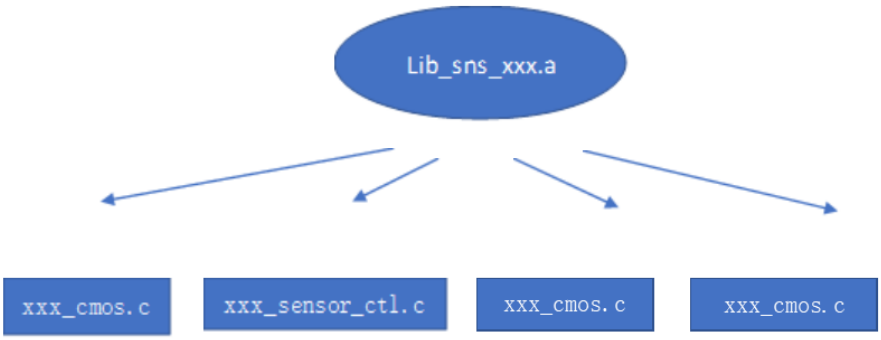


The data flow is roughly as follows: Sensor -> PHYA -> PHYD -> MAC (CSI/sub-LVDS/TTL) -> ISP’ s CSI BDG.

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.

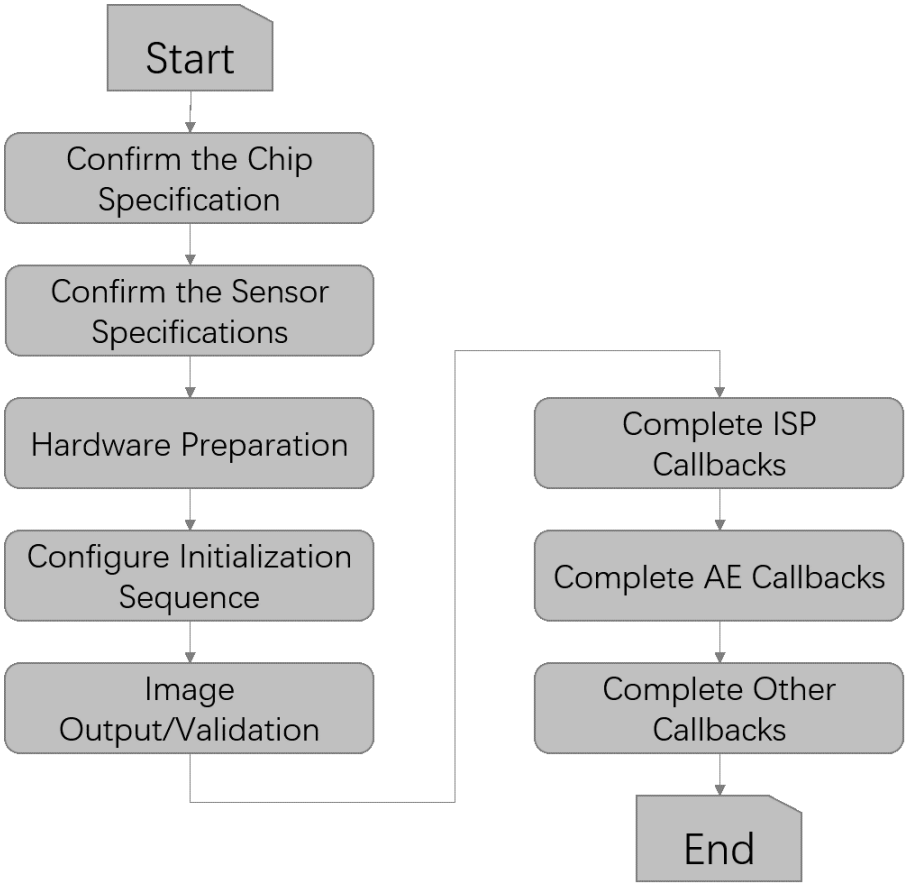
2.2 Sensor Library Structure

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.



- 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



CHAPTER

THREE

CONFIRM SPECIFICATIONS

3.1 Confirm Main Chip 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,
};
```

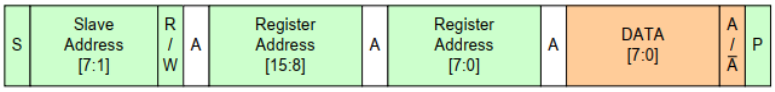
Mclk reference clock:

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

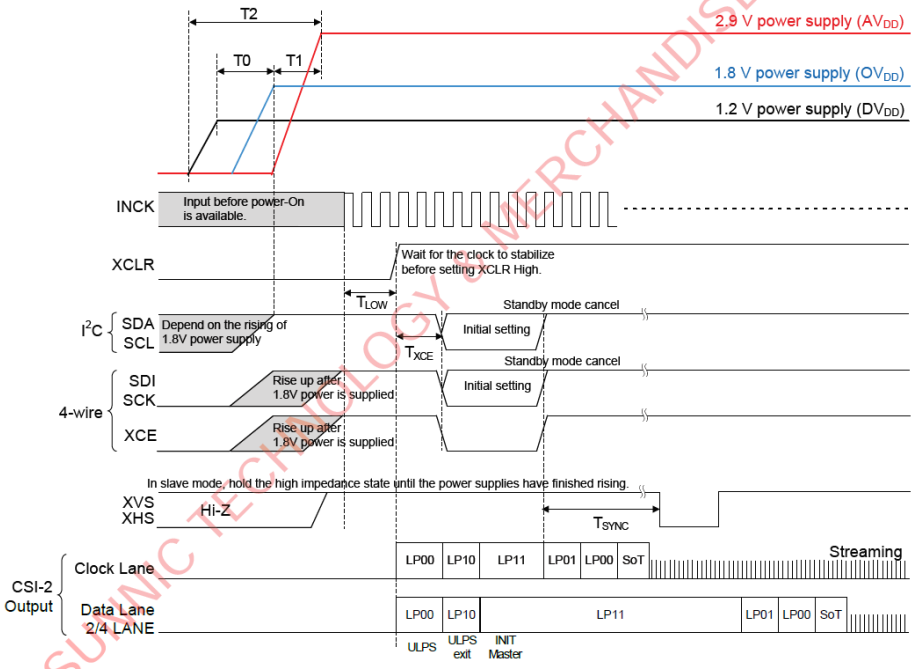
3.2 Confirm Sensor Specifications

- Confirm Sensor Control Interface (I2C/SPI).

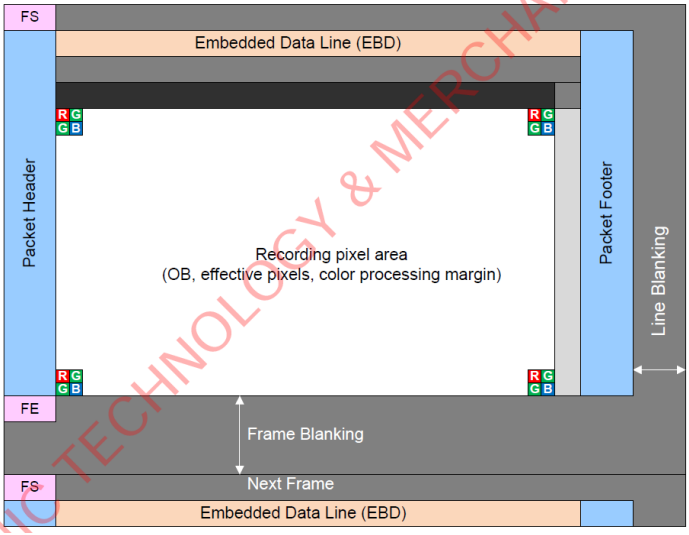
I²C serial communication supports a 16-bit register address and 8-bit data message type.



- 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	8 lane	4lane	8lane	4 lane	8 lane	4lane	8lane	
				30 / 25 [frame /s]	30 / 25 [frame /s]	60 / 50 [frame /s]	60 / 50 [frame /s]	30 / 25 [frame /s]	30 / 25 [frame /s]	60 / 50 [frame /s]	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.

CHAPTER

FOUR

IMAGE OUTPUT DEBUGGING

4.1 Hardware Preparation

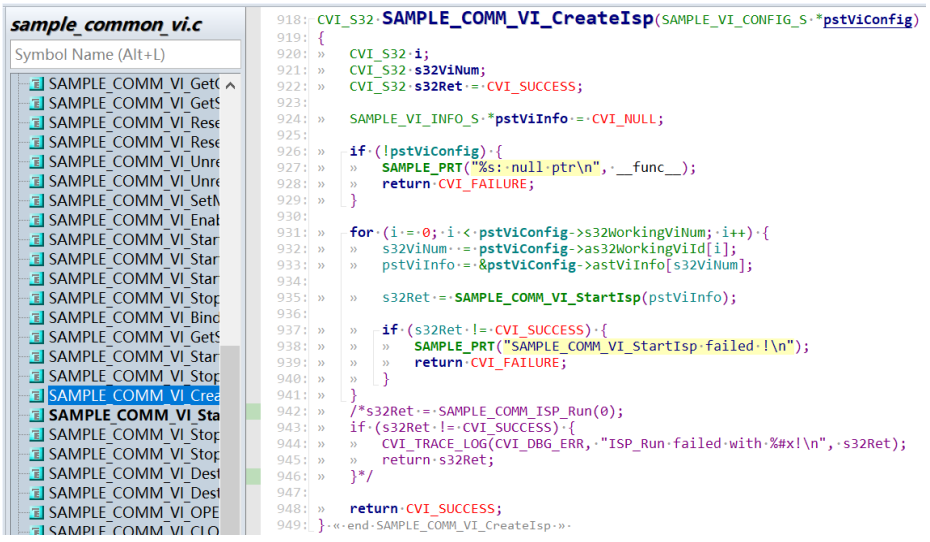
- Confirm that the power supply to the sensor is correct.
- Confirm that the Sensor Reset GPIO is correct.
- Confirm the source of the sensor' s input reference clock (main chip or external crystal oscillator).
- Confirm that the I2C-writable sensor registers can be erased. Use the default i2c_read/i2c_write 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.

```
308:    gc2053_slave_write_register(ViPipe,0x13,0x07);
309:    gc2053_slave_write_register(ViPipe,0x15,0x12);
310:    gc2053_slave_write_register(ViPipe,0xfe,0x00);
311:    gc2053_slave_write_register(ViPipe,0x3e,0x91);
312:    gc2053_slave_write_register(ViPipe,0x17,0x83);
313:
314:    //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.

Step 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.

The detailed content can refer to the data types in the document " MIPI User Guide_v1.1.1.docx" .

- 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,
                    },
```

(continues on next page)

(continued from previous page)

```
                .stMaxSize = {
                        .u32Width = 1948,
                        .u32Height = 1097,
                },
        },
        .f32MaxFps = 30,
        .f32MinFps = 0.119,
        .u32HtsDef = 0x1130,
        .u32VtsDef = 1125,
        .stExp[0] = {
                .u16Min = 1,
                .u16Max = 1123,
                .u16Def = 400,
                .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 init sequence we usually receive corresponds to the output mode of the maximum resolution, which is the all-pixel scan mode. However, in some cases, customers may need to crop the data output by the sensor, so it is necessary to adapt it to the window crop mode. To do so, you would need to ask the sensor manufacturer to provide an init sequence that corresponds to the crop mode, or modify the init sequence according to the sensor spec on your own.

Step 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 (New)

When adapting to the sensor, if the specific VI_DEV_ATTR_S structure variable such as DEV_ATTR_NEXTCHIP_N5_2M_BASE is removed from sample_common_vi.c and replaced with a universal variable DEV_ATTR_SENSOR_BASE, as shown in the figure below, it indicates that the new sample common architecture is used.

```
VI_DEV_ATTR_S DEV_ATTR_NEXTCHIP_N5_2M_BASE = {
    VI_MODE_BT656,
    VI_WORK_MODE_1Multiplex,
    VI_SCAN_PROGRESSIVE,//VI_SCAN_INTERLACED,
    {-1, -1, -1, -1},
    VI_DATA_SEQ_UYVY,

    {
    /*port_vsync   port_vsync_neg     port_hsync        port_hsync_neg        */
    VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SINGNAL, VI_HSYNC_NEG_HIGH,
    VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,

    /*hsync_hfb    hsync_act    hsync_hhb*/
    {0,            1920,        0,
    /*vsync0_vhb vsync0_act vsync0_hhb*/
     0,            1080,        0,
    /*vsync1_vhb vsync1_act vsync1_hhb*/
     0,            0,           0}
    },
    VI_DATA_TYPE_YUV,
    {1920, 1080},
    {
        WDR_MODE_NONE,
        1080
    },
    .enBayerFormat = BAYER_FORMAT_BG,
};

VI_DEV_ATTR_S DEV_ATTR_PIXELPLUS_PR2020_1M_BASE = {
    VI_MODE_BT656,
    VI_WORK_MODE_1Multiplex,
    VI_SCAN_PROGRESSIVE,//VI_SCAN_INTERLACED,
    {-1, -1, -1, -1},
    VI_DATA_SEQ_UYVY,

    {
    /*port_vsync   port_vsync_neg     port_hsync        port_hsync_neg        */
```

```
// default is MIPI-CSI Bayer format sensor
VI_DEV_ATTR_S DEV_ATTR_SENSOR_BASE = {
    VI_MODE_MIPI,
    VI_WORK_MODE_1Multiplex,
    VI_SCAN_PROGRESSIVE,
    {-1, -1, -1, -1},
    VI_DATA_SEQ_YUYV,

    {
    /*port_vsync   port_vsync_neg     port_hsync        port_hsync_neg        */
    VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SINGNAL, VI_HSYNC_NEG_HIGH,
    VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,

    /*hsync_hfb    hsync_act    hsync_hhb*/
    {0,            1920,        0,
    /*vsync0_vhb vsync0_act vsync0_hhb*/
     0,            1080,        0,
    /*vsync1_vhb vsync1_act vsync1_hhb*/
     0,            0,           0}
    },
    VI_DATA_TYPE_RGB,
    {1920, 1080},
    {
        WDR_MODE_NONE,
        1080
    },
    .enBayerFormat = BAYER_FORMAT_BG,
};
```

At this point, adaptation can be carried out according to the following steps:

- Extern the sensor object to include/cvi_sns_ctrl.h.
- Add a new SAMPLE_SNS_TYPE_E to sample_comm.h.

```
typedef enum _SAMPLE_SNS_TYPE_E {
    /*-------LINEAR BEGIN-------*/
    SONY_IMX290_MIPI_1M_30FPS_12BIT,
    SONY_IMX290_MIPI_2M_60FPS_12BIT,
    SONY_IMX327_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_MIPI_2M_30FPS_12BIT,
    SONY_IMX327_2L_MIPI_2M_30FPS_12BIT,
    SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_2L_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT,
    OV_OS08A20_MIPI_8M_30FPS_10BIT,
    OV_OS08A20_MIPI_5M_30FPS_10BIT,
    SOI_F35_MIPI_2M_30FPS_10BIT,
    SOI_F35_SLAVE_MIPI_2M_30FPS_10BIT,
    SOI_H65_MIPI_1M_30FPS_10BIT,
    PICO640_THERMAL_479P,
    PICO384_THERMAL_384X288,
    SONY_IMX327_SUBLVDS_2M_30FPS_12BIT,
    SONY_IMX307_SUBLVDS_2M_30FPS_12BIT,
    VIVO_MCS369Q_4M_30FPS_12BIT,
    VIVO_MM308M2_2M_25FPS_8BIT,
    NEXTCHIP_N5_2M_25FPS_8BIT,
    SMS_SC3335_MIPI_3M_30FPS_10BIT,
    SONY_IMX335_MIPI_5M_30FPS_12BIT,
    SONY_IMX335_MIPI_4M_30FPS_12BIT,
    PIXELPLUS_PR2020_2M_25FPS_8BIT,
```

- Add a sensor Obj to SAMPLE_COMM_GetSnsObj in sample_common_isp.c.

```
742 #endif
743 #if defined(SENSOR_SMS_SC035GS)
744     case SMS_SC035GS_MIPI_480P_120FPS_12BIT:
745         pSnsObj = &stSnsSC035GS_Obj;
746         break;
747 #endif
748 #if defined(SENSOR_TECHPOINT_TP2850)
749     case TECHPOINT_TP2850_MIPI_2M_30FPS_8BIT:
750     case TECHPOINT_TP2850_MIPI_4M_30FPS_8BIT:
751         pSnsObj = &stSnsTP2850_Obj;
752         break;
753 #endif
754
755 #if defined(SENSOR_BRIGATES_BG0808)
756             case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT:
757             case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT_WDR2T01:
758                 pSnsObj = &stSnsBG0808_Obj;
759                 break;
760 #endif
761     default:
762         pSnsObj = CVI_NULL;
763         break;
764     }
765
766     return pSnsObj;
767 }
```

- In the sample_common_vi.c file, add corresponding cases for SAMPLE_COMM_VI_GetDevAttrBySns, SAMPLE_COMM_VI_GetChnAttrBySns, and SAMPLE_COMM_VI_GetSizeBySensor.

```
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_WDR2T01:
case GCORE_GC2093_SLAVE_MIPI_2M_30FPS_10BIT_WDR2T01:
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_WDR2T01:
»       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_WDR2T01:
»       case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT:
»       case BRIGATES_BG0808_MIPI_2M_30FPS_10BIT_WDR2T01:
»       »       *penSize = PIC_1080P;
»       »       break;
»       case OV_OS08A20_MIPI_8M_30FPS_10BIT:
»       case OV_OS08A20_MIPI_8M_30FPS_10BIT_WDR2T01:
»       case OV_OS08A20_SLAVE_MIPI_8M_30FPS_10BIT:
»       case OV_OS08A20_SLAVE_MIPI_8M_30FPS_10BIT_WDR2T01:
»       case SONY_IMX334_MIPI_8M_30FPS_12BIT:
»       case SONY_IMX334_MIPI_8M_30FPS_12BIT_WDR2T01:
»       case SMS_SC8238_MIPI_8M_30FPS_10BIT:
»       case SMS_SC8238_MIPI_8M_15FPS_10BIT_WDR2T01:
»       case SMS_SC850SL_MIPI_8M_30FPS_12BIT:
»       case SMS_SC850SL_MIPI_8M_30FPS_10BIT_WDR2T01:
»       »       *penSize = PIC_3840x2160;
»       »       break;
```

- Add the new sensor name to the snsr_type_name array, and make sure the sensor name is consistent with the enum name and order of SAMPLE_SNS_TYPE_E added in sample_comm.h.

```
1807 »       "SMS_SC4210_MIPI_4M_30FPS_12BIT",
1808 »       /* ------ LINEAR END ------*/
1809
1810 »       /* ------ WDR 2TO1 BEGIN ------*/
1811 »       "SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1",
1812 »       "SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1",
1813 »       "SONY_IMX327_2L_MIPI_2M_30FPS_12BIT_WDR2TO1",
1814 »       "SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1",
1815 »       "SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1",
1816 »       "SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1",
1817 »       "OV_OS08A20_MIPI_8M_30FPS_10BIT_WDR2TO1",
1818 »       "OV_OS08A20_MIPI_5M_30FPS_10BIT_WDR2TO1",
1819 »       "SOI_F35_MIPI_2M_30FPS_10BIT_WDR2TO1",
1820 »       "SOI_F35_SLAVE_MIPI_2M_30FPS_10BIT_WDR2TO1",
1821 »       "SONY_IMX327_SUBLVDS_2M_30FPS_12BIT_WDR2TO1",
1822 »       "SONY_IMX307_SUBLVDS_2M_30FPS_12BIT_WDR2TO1",
1823 »       "SONY_IMX335_MIPI_5M_30FPS_10BIT_WDR2TO1",
1824 »       "SONY_IMX335_MIPI_4M_30FPS_10BIT_WDR2TO1",
1825 »       "SONY_IMX334_MIPI_8M_30FPS_12BIT_WDR2TO1",
1826 »       "SMS_SC8238_MIPI_8M_15FPS_10BIT_WDR2TO1",
1827 »       "SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1",
```

- sample/common/Kbuild.

```
                                                    ... skipped
09      KBUILD_DEFINES += -DSENSOR_SMS_SC035HGS
10  endif
11
12  ifeq ($(CONFIG_SENSOR_SMS_SC035GS), y)
13      KBUILD_DEFINES += -DSENSOR_SMS_SC035GS
14  endif
15
16  ifeq ($(CONFIG_SENSOR_TECHPOINT_TP2850), y)
17      KBUILD_DEFINES += -DSENSOR_TECHPOINT_TP2850
18  endif
19
20  ifeq ($(CONFIG_SENSOR_BRIGATES_BG0808), y)
21      KBUILD_DEFINES += -DSENSOR_BRIGATES_BG0808
22  endif
23
```

4.4 Adapting to Sample Common (Old)

- Extern the sensor object to include/cvi_sns_ctrl.h.
- Add a new SAMPLE_SNS_TYPE_E to sample_comm.h.

```
typedef enum _SAMPLE_SNS_TYPE_E {
    /* ------ LINEAR BEGIN ------ */
    SONY_IMX290_MIPI_1M_30FPS_12BIT,
    SONY_IMX290_MIPI_2M_60FPS_12BIT,
    SONY_IMX327_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_MIPI_2M_30FPS_12BIT,
    SONY_IMX327_2L_MIPI_2M_30FPS_12BIT,
    SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_2L_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT,
    OV_OS08A20_MIPI_8M_30FPS_10BIT,
    OV_OS08A20_MIPI_5M_30FPS_10BIT,
    SOI_F35_MIPI_2M_30FPS_10BIT,
    SOI_F35_SLAVE_MIPI_2M_30FPS_10BIT,
    SOI_H65_MIPI_1M_30FPS_10BIT,
    PICO640_THERMAL_479P,
    PICO384_THERMAL_384X288,
    SONY_IMX327_SUBLVDS_2M_30FPS_12BIT,
    SONY_IMX307_SUBLVDS_2M_30FPS_12BIT,
    VIVO_MCS369Q_4M_30FPS_12BIT,
    VIVO_MM308M2_2M_25FPS_8BIT,
    NEXTCHIP_N5_2M_25FPS_8BIT,
    SMS_SC3335_MIPI_3M_30FPS_10BIT,
    SONY_IMX335_MIPI_5M_30FPS_12BIT,
    SONY_IMX335_MIPI_4M_30FPS_12BIT,
    PIXELPLUS_PR2020_2M_25FPS_8BIT,
```

- In sample_common_isp.c, add the corresponding isp_pub_attr and add the corresponding case statements in SAMPLE_COMM_ISP_GetIspAttrBySns and SAMPLE_COMM_ISP_GetSnsObj.

```
69 ISP_PUB_ATTR_S ISP_PUB_ATTR_SC4210_4M_30FPS = { { 0, 0, 2560, 1440 }, { 2560, 1440 },
70 »       »       »       »       »       »       30, BAYER_BGGR, WDR_MODE_NONE, 0};
```

```
233 »       case SMS_SC4210_MIPI_4M_30FPS_12BIT:
234 »       »       memcpy(pstPubAttr, &ISP_PUB_ATTR_SC4210_4M_30FPS, sizeof(ISP_PUB_ATTR_S));
235 »       »       break;
236 »       case SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1:
237 »       »       memcpy(pstPubAttr, &ISP_PUB_ATTR_SC4210_4M_30FPS, sizeof(ISP_PUB_ATTR_S));
238 »       »       pstPubAttr->enWDRMode = WDR_MODE_2To1_LINE;
239 »       »       break;
240 »       default:
```

```
415 »       »       return &stSnsSC8238_Obj;
416 »       case SMS_SC4210_MIPI_4M_30FPS_12BIT:
417 »       case SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1:
418 »       »       return &stSnsSC4210_Obj;
419 »       default:
420 »       »       return CVI_NULL;
421 »       }
```

- Add corresponding vi_dev_attr in sample_common_vi.c, and add corresponding cases in SAMPLE_COMM_VI_GetDevAttrBySns, SAMPLE_COMM_VI_GetChnAttrBySns, and SAMPLE_COMM_VI_GetSizeBySensor.

```
VI_DEV_ATTR_S DEV_ATTR_SC4210_4M_BASE = {
»       VI_MODE_MIPI,
»       VI_WORK_MODE_1Multiplex,
»       VI_SCAN_PROGRESSIVE,
»       {-1, -1, -1, -1},
»       VI_DATA_SEQ_YUYV,

»       {
»       /*port_vsync   port_vsync_neg     port_hsync        port_hsync_neg        */
»       VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SINGNAL, VI_HSYNC_NEG_HIGH,
»       VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,

»       /*hsync_hfb    hsync_act    hsync_hhb*/
»       {0,            2560,        0,
»       /*vsync0_vhb vsync0_act vsync0_hhb*/
»        0,            1440,        0,
»       /*vsync1_vhb vsync1_act vsync1_hhb*/
»        0,            0,           0}
»       },
»       VI_DATA_TYPE_RGB,
»       {2560, 1440},
»       {
»       »       WDR_MODE_NONE,
»       »       1440
»       },
»       .enBayerFormat = BAYER_FORMAT_BG,
};
```

```
»       case SMS_SC4210_MIPI_4M_30FPS_12BIT:
»       »       memcpy(pstViDevAttr, &DEV_ATTR_SC4210_4M_BASE, sizeof(VI_DEV_ATTR_S));
»       »       break;
»       case SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1:
»       »       memcpy(pstViDevAttr, &DEV_ATTR_SC4210_4M_BASE, sizeof(VI_DEV_ATTR_S));
»       »       pstViDevAttr->stWDRAttr.enWDRMode = WDR_MODE_2To1_LINE;
»       »       break;
```

```
»       case SMS_SC4210_MIPI_4M_30FPS_12BIT:
»       case SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1:
»       »       memcpy(pstChnAttr, &CHN_ATTR_2560x1440_420_SDR8_LINEAR, sizeof(VI_CHN_ATTR_S));
»       »       break;
```

```
»       case SMS_SC4210_MIPI_4M_30FPS_12BIT:
»       case SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1:
»       »       *penSize = PIC_1440P;
»       »       break;
```

- Add the new sensor name to the snsr_type_name array, and make sure that the sensor name matches the enumeration name and order of the new SAMPLE_SNS_TYPE_E added to sample_comm.h.

```
1807 »       "SMS_SC4210_MIPI_4M_30FPS_12BIT",
1808 »       /* ------ LINEAR END ------*/
1809
1810 »       /* ------ WDR 2TO1 BEGIN ------*/
1811 »       "SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1",
1812 »       "SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1",
1813 »       "SONY_IMX327_2L_MIPI_2M_30FPS_12BIT_WDR2TO1",
1814 »       "SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1",
1815 »       "SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1",
1816 »       "SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1",
1817 »       "OV_OS08A20_MIPI_8M_30FPS_10BIT_WDR2TO1",
1818 »       "OV_OS08A20_MIPI_5M_30FPS_10BIT_WDR2TO1",
1819 »       "SOI_F35_MIPI_2M_30FPS_10BIT_WDR2TO1",
1820 »       "SOI_F35_SLAVE_MIPI_2M_30FPS_10BIT_WDR2TO1",
1821 »       "SONY_IMX327_SUBLVDS_2M_30FPS_12BIT_WDR2TO1",
1822 »       "SONY_IMX307_SUBLVDS_2M_30FPS_12BIT_WDR2TO1",
1823 »       "SONY_IMX335_MIPI_5M_30FPS_10BIT_WDR2TO1",
1824 »       "SONY_IMX335_MIPI_4M_30FPS_10BIT_WDR2TO1",
1825 »       "SONY_IMX334_MIPI_8M_30FPS_12BIT_WDR2TO1",
1826 »       "SMS_SC8238_MIPI_8M_15FPS_10BIT_WDR2TO1",
1827 »       "SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2TO1",
```

- sample/common/Kbuild

```
                                                    ... skipped
09     KBUILD_DEFINES += -DSENSOR_SMS_SC035HGS
10 endif
11
12 ifeq ($(CONFIG_SENSOR_SMS_SC035GS), y)
13     KBUILD_DEFINES += -DSENSOR_SMS_SC035GS
14 endif
15
16 ifeq ($(CONFIG_SENSOR_TECHPOINT_TP2850), y)
17     KBUILD_DEFINES += -DSENSOR_TECHPOINT_TP2850
18 endif
19
20 ifeq ($(CONFIG_SENSOR_BRIGATES_BG0808), y)
21     KBUILD_DEFINES += -DSENSOR_BRIGATES_BG0808
22 endif
23
```

4.5 Adding Sensor INI Configuration

Some properties of the sensor can be modified by changing the configuration in the ini file, such as the lane sequence, I2C port, and sensor output mode.

By default, the middleware will first read the sensor configuration file from /mnt/data/sensor_ini.cfg. If the file is not found in this directory, it will continue to read from /mnt/system/usr/bin/sensor_cfg.ini. If the file is not found in this directory either, the middleware will use the initial values defined in the code.

Here is an example of the sensor_cfg.ini file for the gc2053 sensor:

```
[source]
;type = SOURCE_USER_FE
dev_num = 2
; section for sensor
[sensor]
; sensor name
name = GCORE_GC2053_MIPI_2M_30FPS_10BIT
bus_id = 0
```

(continues on next page)

(continued from previous page)

```
mipi_dev = 0
lane_id = 1, 3, 2, -1, -1
; section for sensor2
[sensor2]
name = GCORE_GC2053_SLAVE_MIPI_2M_30FPS_10BIT
bus_id = 3
mipi_dev = 1
lane_id = 0, 4, 2, -1, -1
```

name: Represents the output mode of the sensor, and it should be consistent with the enumeration name added in SAMPLE_SNS_TYPE_E in sample_comm.h.

Bus_id: Represents the I2C bus number.

Mipi_dev: Indicates which MIPI-RX group to use.

Lane_id: Indicates the configuration of MIPI lanes.

Pn_swap: Indicates whether P/N swap is required or not. Set to 0 if it' s not required, and set to 1 if it' s required.

Mclk: Indicates which group of mclk to use as reference clock.

Mclk_en: Represents which group of mclk is enabled to output.

hw_sync: Dual sensor frame synchronization, hw_sync=1 means the slave sensor is synchronized with the master sensor.

sns_i2c_addr: The I2C device address of the sensor.

CHAPTER

FIVE

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.

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.

5.1 Dump RAW

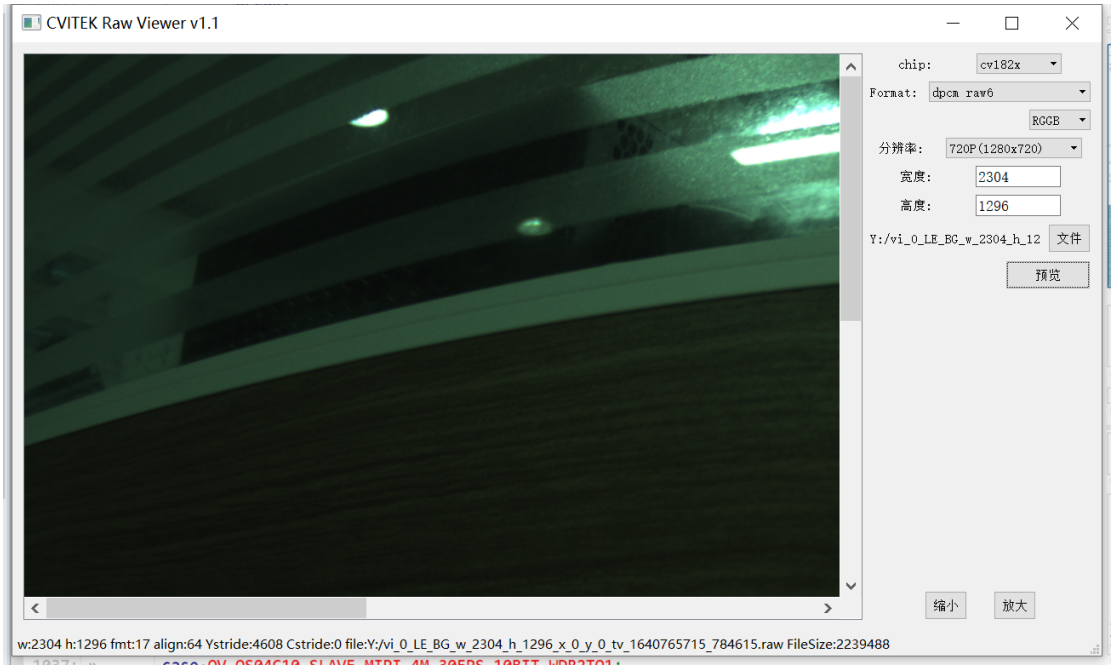
Run the sensor_test program, enter 1 to select "dump vi raw data" , then follow the prompt "To get raw dump from dev(0~1):" and enter dev (0 represents vi pipe0, dump images from the first sensor, 1 represents vi pipe1, dump images from the second sensor).

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 chip, 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.

5.2 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 chip, format, width, and height.

CHAPTER

SIX

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.

6.1 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

6.2 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.

```
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.

```
    }«·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},
        }
    },
};
```

CHAPTER

SEVEN

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.

7.1 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

7.2 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.

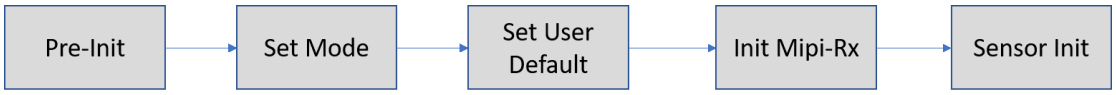
CHAPTER

EIGHT

COMPLETE OTHER FUNCTIONS

8.1 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.

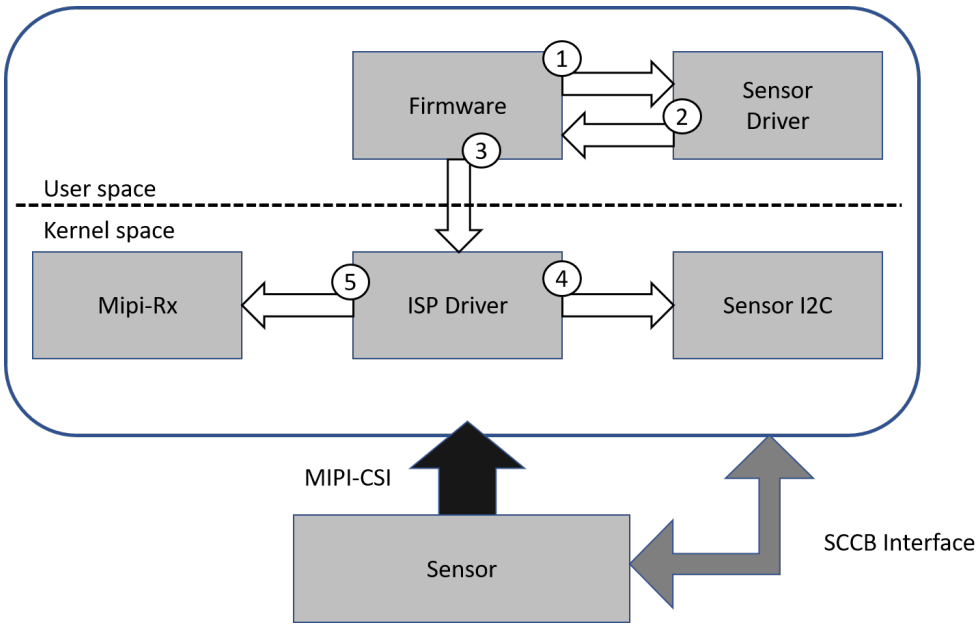
8.2 Sensor Shutdown Process

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.

8.3 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.

CHAPTER

NINE

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

9.1 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.

9.2 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
```

9.3 Gain Linearity Verification

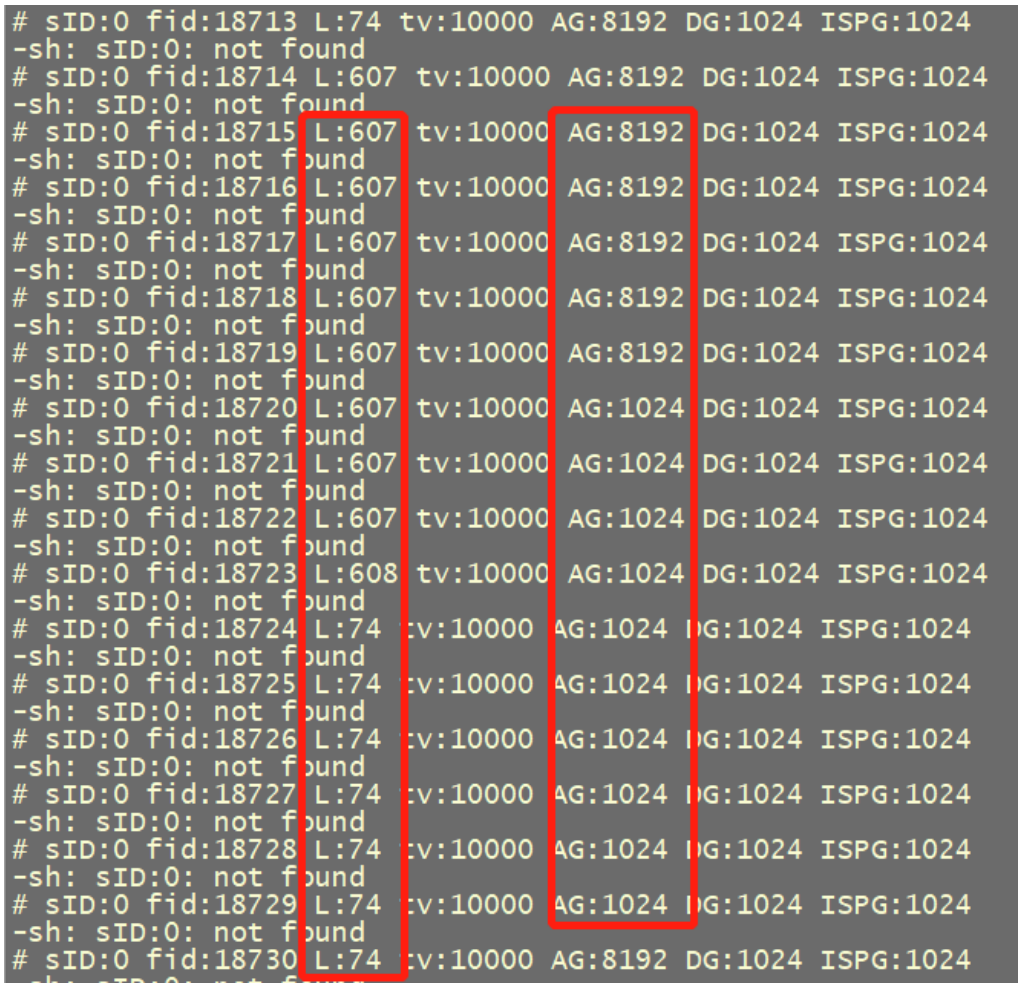
Point the camera at the light box and run sensor_test and type CMD in linear mode.

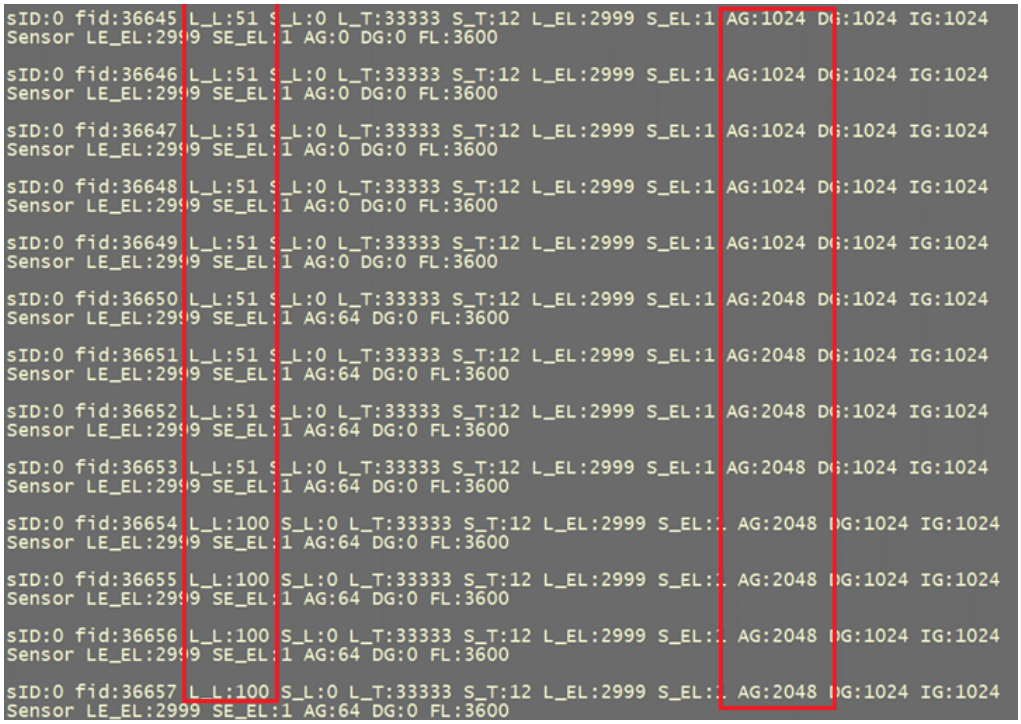
```
5
2 0 72 0 0
```

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.

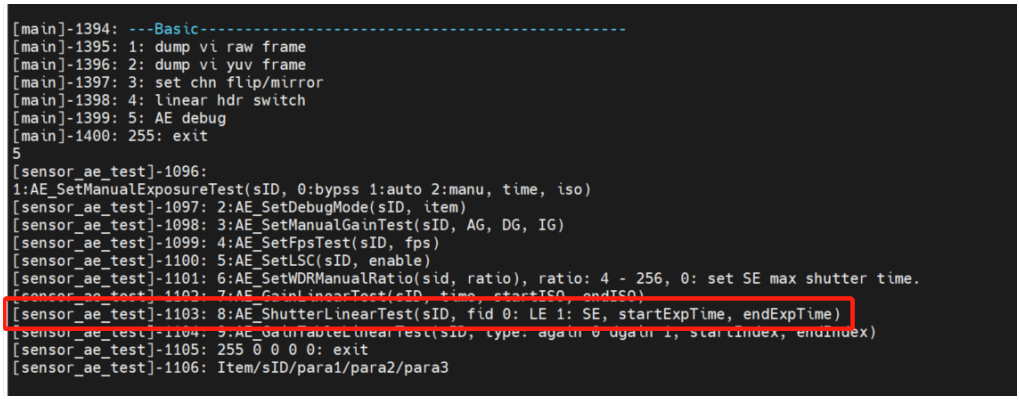




9.4 Advanced Verification

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
```

9.5 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
```

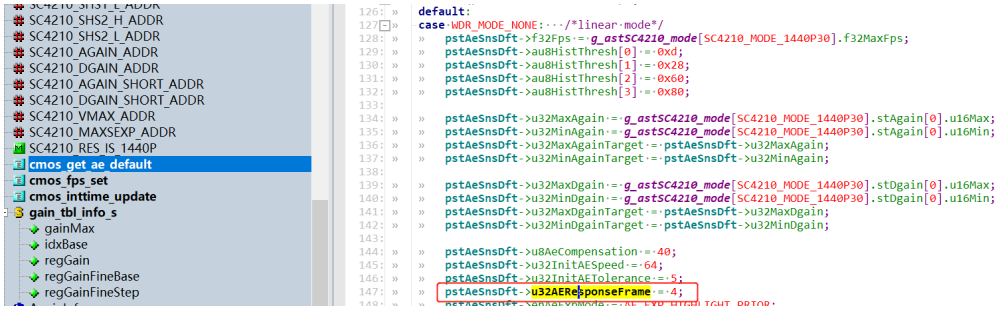
Enter CMD.

```
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.

```
sID:0 fid:9649 L:25 T:10000 EL:450 AG:1024 DG:1024 ISPG:1024
Sensor EL:450 AG:0 DG:0

sID:0 fid:9650 L:25 T:10000 EL:450 AG:2048 DG:1024 ISPG:1024
Sensor EL:450 AG:64 DG:0

sID:0 fid:9651 L:25 T:10000 EL:450 AG:2048 DG:1024 ISPG:1024
Sensor EL:450 AG:64 DG:0

sID:0 fid:9652 L:25 T:10000 EL:450 AG:2048 DG:1024 ISPG:1024
Sensor EL:450 AG:64 DG:0

sID:0 fid:9653 L:25 T:10000 EL:450 AG:2048 DG:1024 ISPG:1024
Sensor EL:450 AG:64 DG:0

sID:0 fid:9654 L:50 T:10000 EL:450 AG:2048 DG:1024 ISPG:1024
Sensor EL:450 AG:64 DG:0

sID:0 fid:9655 L:50 T:10000 EL:450 AG:2048 DG:1024 ISPG:1024
Sensor EL:450 AG:64 DG:0
```

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:



9.6 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;
```

9.7 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.

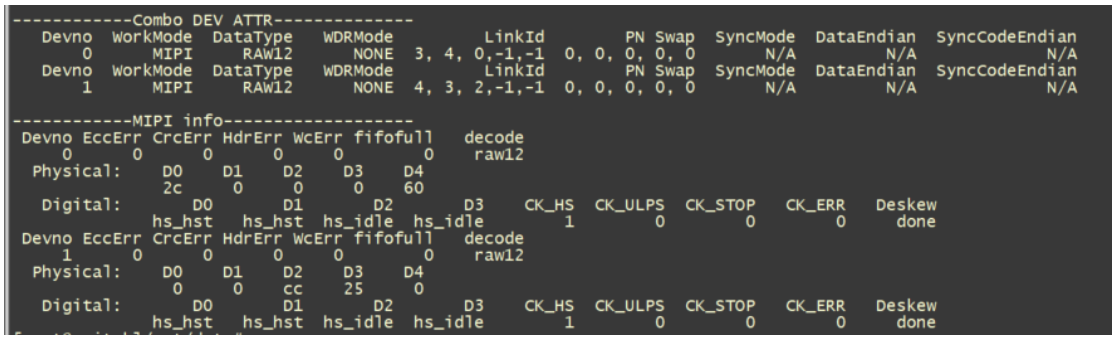
```
# cat /proc/cvitek/vi_dbg
[VI Info]
VIOutImgWidth           :1920
VIOutImgHeight          :1080
VIIspTopStatus          :0x800
[VI ISP_PIPE_A]
VIInImgWidth            :1948
VIInImgHeight           :1097
VISofCnt                :42866
VIPreCnt                :42865
VIPostCnt               :42864
VIDevFPS                :  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
```

CHAPTER

TEN

COMMON PROBLEM

10.1 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.

10.2 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.

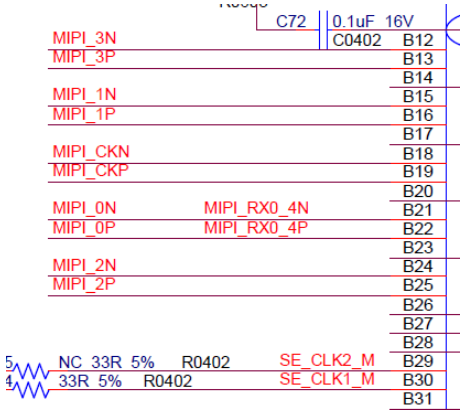
/sbin/syslogd -l 8 -s 2048 -O /mnt/data/mw.txt

10.3 How to Configure Lane Line Sequence

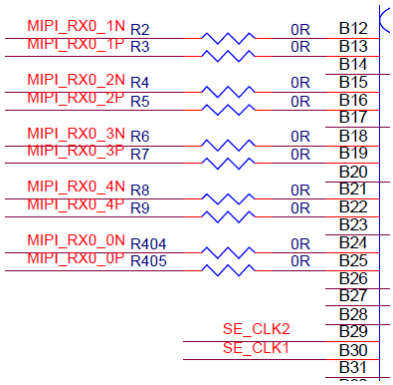
Note that the lane id to be configured should be configured with the sensor as the reference. The index number of lane_id array represents the Lane ID of the Sensor, the index number 0 represents the sensor clock, and the index number 1-4 represents sensor lane 0~3. The value of the land_id array indicates 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}.

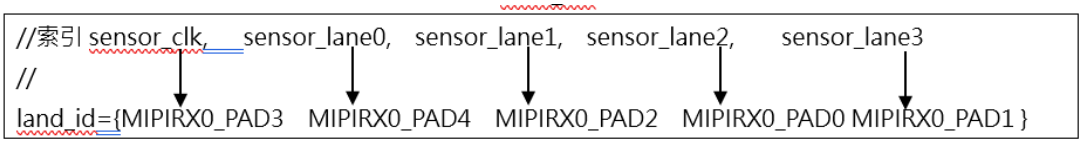
sensor:



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)



10.4 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.

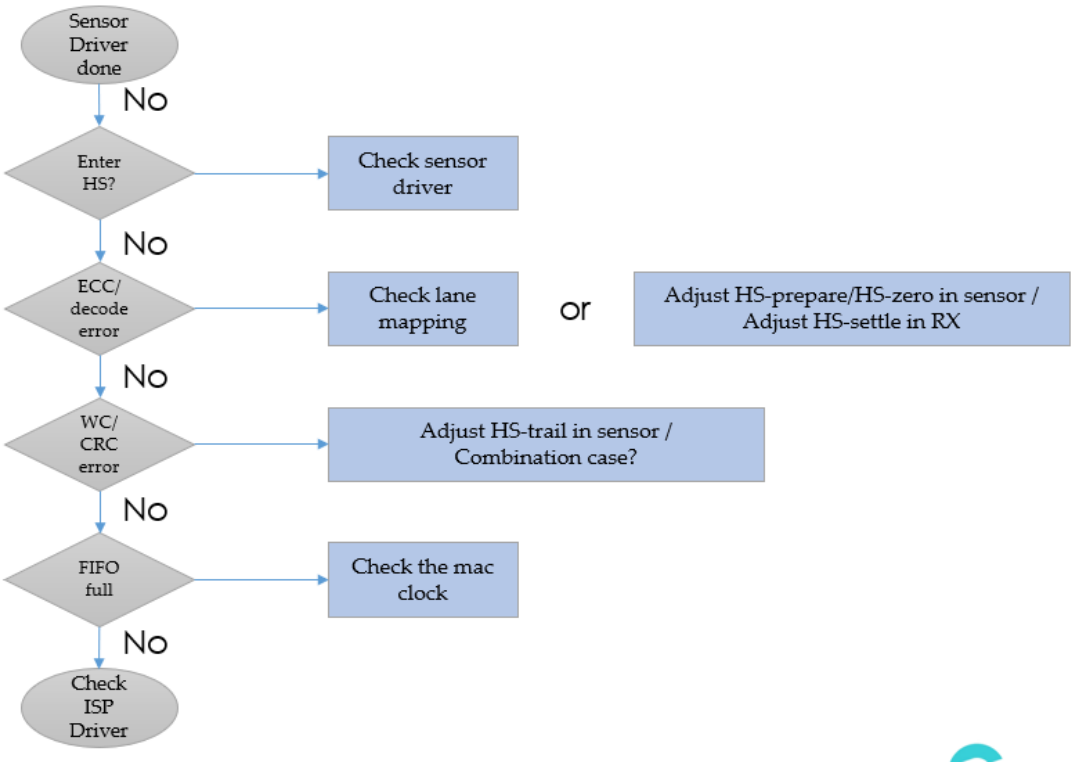
MIPI_Freq: operating frequency of each lane.

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.

10.5 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

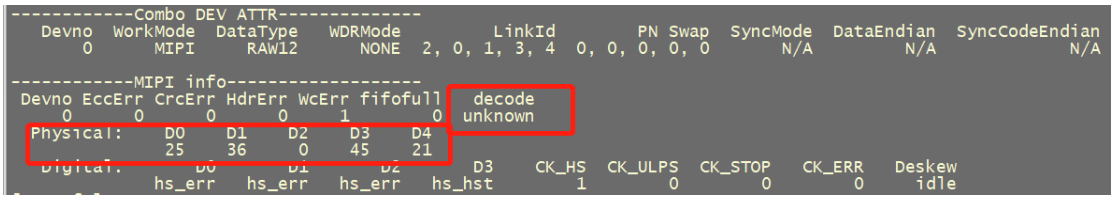
echo "snsr_r 0 0" > /proc/mipi-rx

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.

CHAPTER

ELEVEN

COLOR, NOISE REDUCTION, AND OTHER CORRECTIONS

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

CHAPTER

TWELVE

IMAGE QUALITY TUNING.

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

CHAPTER

THIRTEEN

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: 

13.1 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
```

13.2 Dump RAW

Refer to 5.1 Dump RAW.

13.3 Dump YUV

Refer to 5.2 Dump YUV.

13.4 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.

13.5 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.

```
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;
    }
```

13.6 AE Related Verification

Refer to AE Related Verification.