



CV180X & CV181X Sensor 调试使用手册

Version: 1.1.4

Release date: 2023-04-21

©2022 北京晶视智能科技有限公司
本文件所含信息归北京晶视智能科技有限公司所有。
未经授权，严禁全部或部分复制或披露该等信息。

目录

1	声明	2
2	Sensor 驱动介绍	3
2.1	硬件架构	3
2.2	sensor 库结构	4
2.3	调试流程	5
3	确认规格	6
3.1	确认主处理器规格	6
3.2	确认 Sensor 规格	7
4	图像输出调试 (Linux 非快启)	9
4.1	硬件准备	9
4.2	配置初始化序列	9
4.2.1	准备 Sensor 驱动	10
4.2.2	Sensor 初始化序列	12
4.3	适配 sample common 和 alios config	12
4.4	添加 sensor ini cfg 配置	20
4.5	编译运行 sensor_test	22
5	图像输出调试 (Alios 快启)	23
5.1	硬件准备	23
5.2	配置初始化序列	23
5.2.1	准备 Sensor 驱动	24
5.2.2	Sensor 初始化序列	26
5.2.3	修改 package.yaml 添加编译文件	27
5.3	适配 solution	27
5.3.1	增加 sensor type	27
5.3.2	修改 sensor mipi 相关配置	30
5.3.3	修改 VB 配置	32
5.3.4	修改添加 pinmux	32
5.3.5	修改编译配置	33
5.4	运行 sensor	34
6	图像输出验证	35
6.1	Dump RAW	35
6.2	Dump YUV	36
7	ISP 基本功能	38
7.1	开发流程	38
7.2	注意事项	38

8	完成 AE 配置功能	42
8.1	开发流程	42
8.2	注意事项	42
9	完善其它功能	46
9.1	Sensor 初始化流程	46
9.2	Sensor 关闭流程	48
9.3	Sensor AE 同步流程	48
10	AE 相关验证	50
10.1	BLC 确认和验证	50
10.2	曝光线性度验证	51
10.3	增益线性度验证	53
10.4	进阶验证	54
10.5	Response Frame 验证	55
10.6	曝光增益同步生效验证	56
10.7	FPS 可控性验证	58
11	常见问题	60
11.1	Proc 讯息解读	60
11.2	Sensor 相关 log 开启	61
11.3	如何配置 lane 线序	61
11.4	如何选择 MAC 频率	63
11.5	错误检查流程	64
11.5.1	I2C Write Fail	64
11.5.2	Decode err	65
11.5.3	ECC err	66
11.5.4	CRC err/Word count err	67
11.5.5	vi_select timeout	67
12	颜色、去噪等矫正	68
13	调试工具	69
13.1	基础功能	69
13.2	Dump RAW	70
13.3	Dump YUV	70
13.4	Set flip/mirror	70
13.5	WDR 和 Linear 切换	70
13.6	AE 相关验证	71

修订记录

Revision	Date	Description
1.0	2019/10/12	初稿
1.1.0	2021/10/1	补充实操细节
1.1.2	2021/12/28	添加 sensor_test
1.1.3	2023/04/13	修改细节以及更新最新内容
1.1.4	2023/04/21	添加 Alios 相关适配

1 声明



法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

联系我们

地址 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

电话 +86-10-57590723 +86-10-57590724

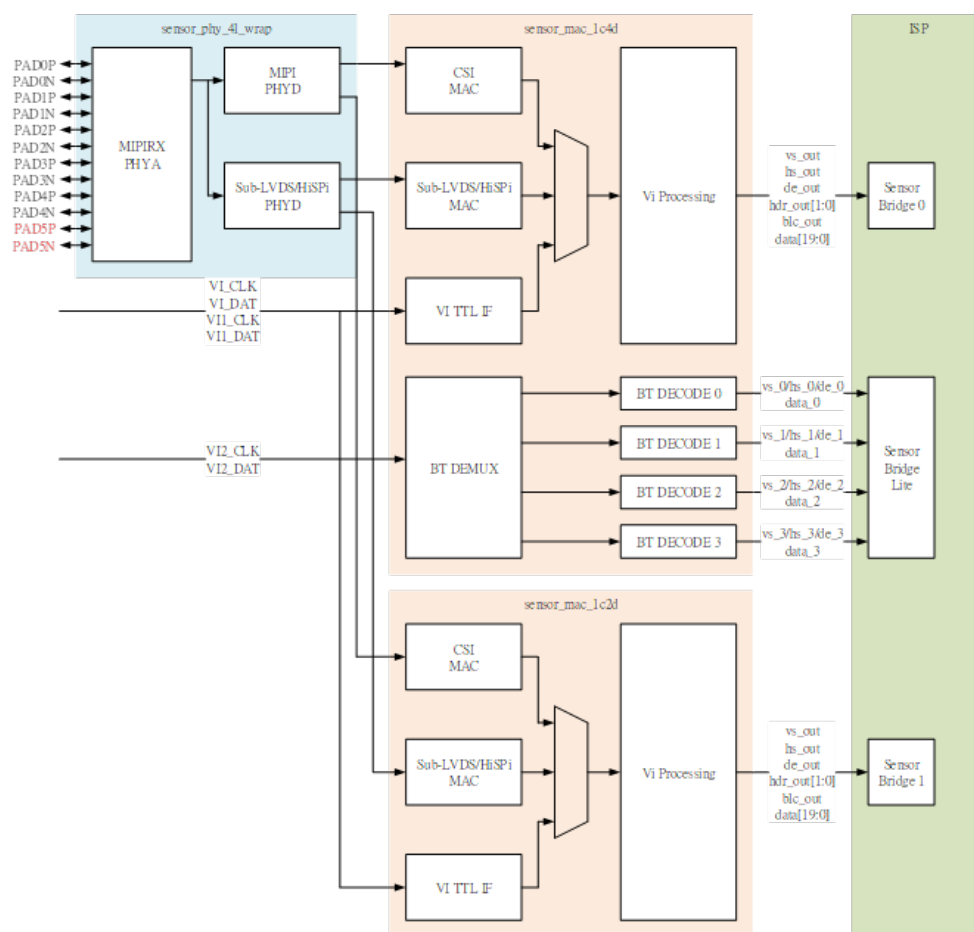
邮编 100094（北京）518100（深圳）

官方网站 <https://www.sophgo.com/>

技术论坛 <https://developer.sophgo.com/forum/index.html>

2 Sensor 驱动介绍

2.1 硬件架构



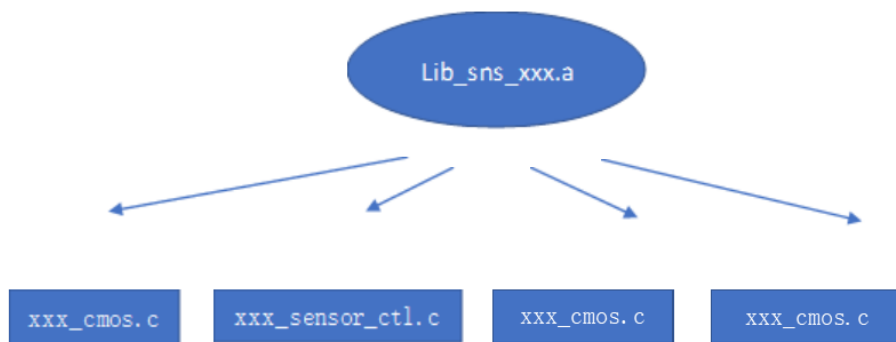
数据流向大致为: Sensor->PHYA -> PHYD -> MAC(CSI/sub-LVDS/TTL)->ISP 的 CSI BDG。

Sensor 吐出 lane 总线上的差分信号, PHYA 用来接收组装差分信号, 然后经过 PHYD 转换成 pixel 数字信号, 最后结合 MAC clk sync 出 frame 数据经过 VI 处理送往 ISP 做进一步处理。

2.2 sensor 库结构

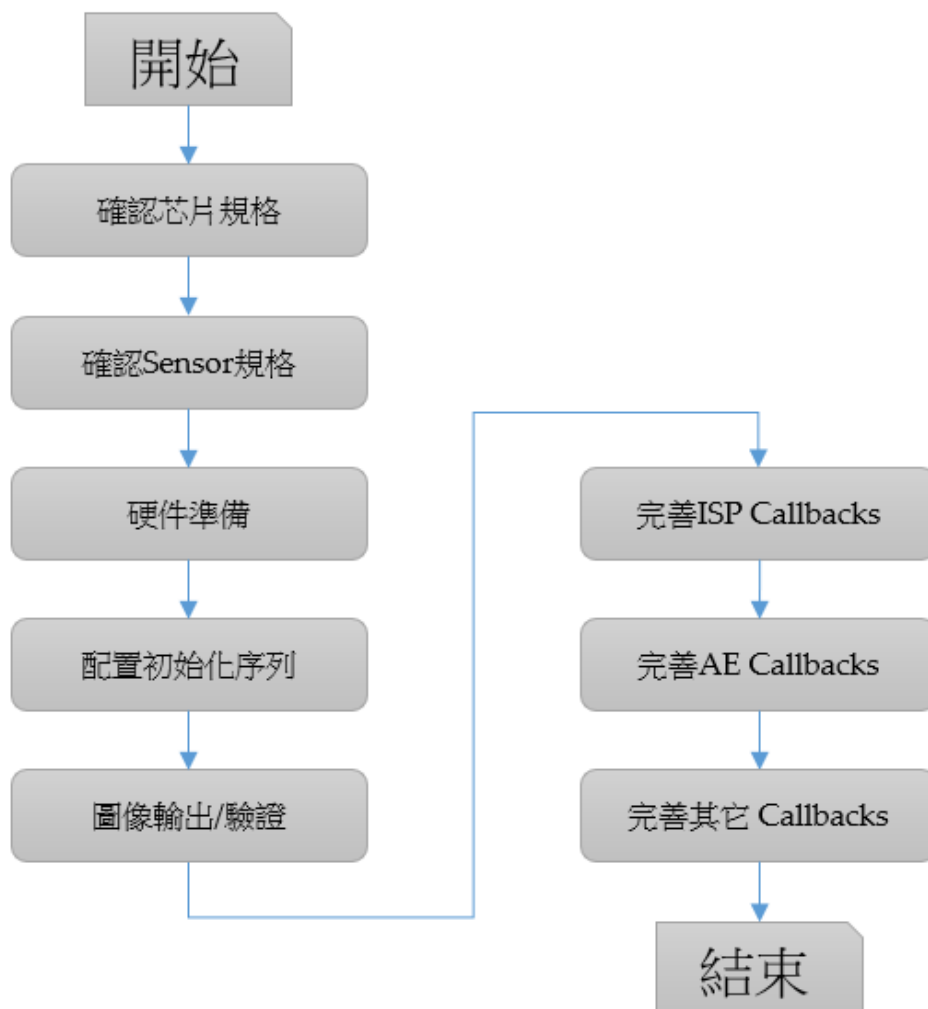
Sensor 库结构如下图，一般包括 4 个文件：xxx_cmos.c、xxx_sensor_ctl.c、xxx_cmos_param.h、xxx_cmos_ex.h。

在 alios 中，sensor 库位于 mars_alios/components/cvi_mmf_sdk/cvi_sensor/ 之中



- xxx_cmos.c 包含 Sensor 驱动的主要功能函数，里面实现了 Sensor 相关的 AE 控制相关函数、ISP 默认配置、Sensor 启动模式选择函数、Sensor 向 AE、AWB、ISP 注册和反注册函数、SnsxxxObj；
- xxx_sensor_ctl.c 主要包含 Sensor 的初始化序列、通讯接口初始化、读写函数实现；
- xxx_cmos_ex.h 是一些结构体、分辨率、模式类型等定义声明的头文件。
- xxx_cmos_param.h 主要是 sensor 属性参数，mipi 属性参数，isp noise profile 的配置。

2.3 调试流程



3 确认规格

3.1 确认主处理器规格

- 支持的 Combo PHY 输入频率上限.
- 支持的 Combo PHY Lane 配置.
- 支持的线性/WDR 接口模式.

注意: 180X 不支持 WDR mode

- 支持的 I2C 组数.
- 支持的输出参考时钟.

比如 cv181x 支持以下:

1C4D (1clk lane, 4data lane)

2.5Gbps/lane

RAW(8/10/12)+YUV422(8/10)

2-frame HDR

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 参考时钟:

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

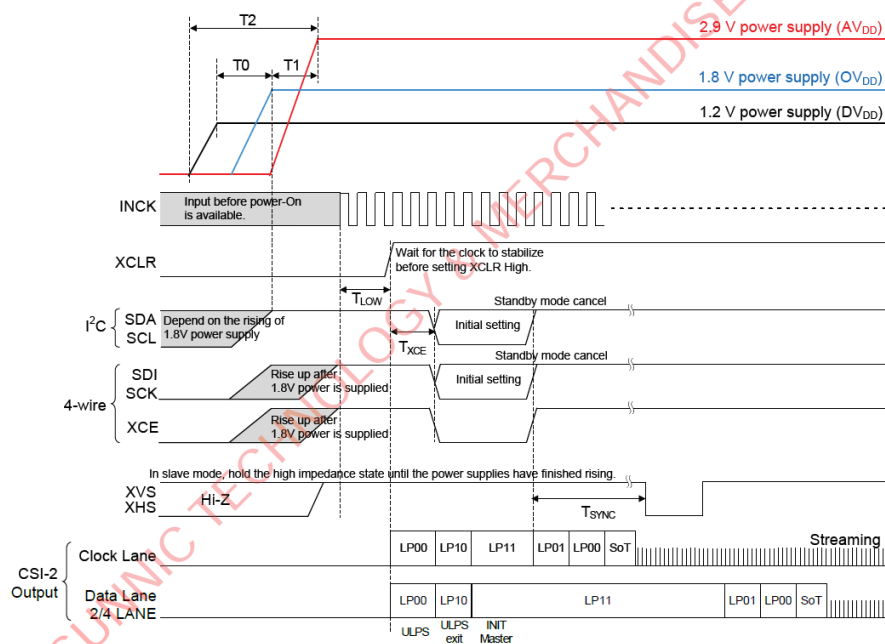
3.2 确认 Sensor 规格

- 确认 Sensor 控制接口 (I2C/SPI).

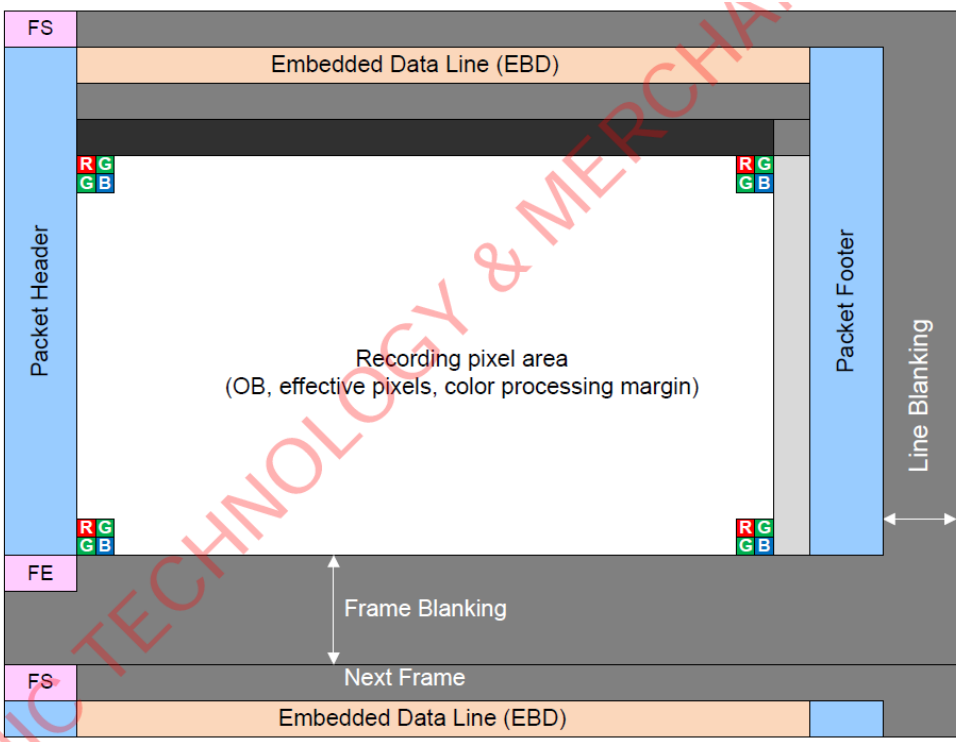
I²C serial communication supports a 16-bit register address and 8-bit data message type.

S	Slave Address [7:1]	R / W	A	Register Address [15:8]	A	Register Address [7:0]	A	DATA [7:0]	A / P
---	---------------------	-------	---	-------------------------	---	------------------------	---	------------	-------

- 确认 Sensor 的开机序列.



- 确认 Sensor 的输入 Reference Clock.
- 确认 Bayer Pattern、像素码宽.



· 确认线性/WDR 模式的图像传输接口模式、输出频率.

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
3031h	[7:0]											
3032h	[3:0]											[frame/s]
3034h	[7:0]	HMAX	0226h	044Ch / 0528h	044Ch / 0528h	0226h / 0294h	0226h / 0294h	044Ch / 0528h	044Ch / 0528h	0226h / 0294h	0226h / 0294h	30 / 25
3035h	[7:0]											[frame / s] / 60 / 50 [frame / s]

- 确认线性/WDR 模式的曝光时间与增益如何设置.
- 确认线性/WDR 模式的帧率如何修改.
- 当接口为 subLVDS/HiSPi 时, 需确认同步码.
- 和 Sensor 厂商索取 Sensor Initialize Settings.

4 图像输出调试 (Linux 非快启)

4.1 硬件准备

- 确认 Sensor 电源供应正确.
- 确认 Sensor Reset GPIO 正确.
- 确认 Sensor 输入参考时钟来源 (主处理器或外部晶振)
- 确认 I2C 可擦写 Sensor 寄存器.

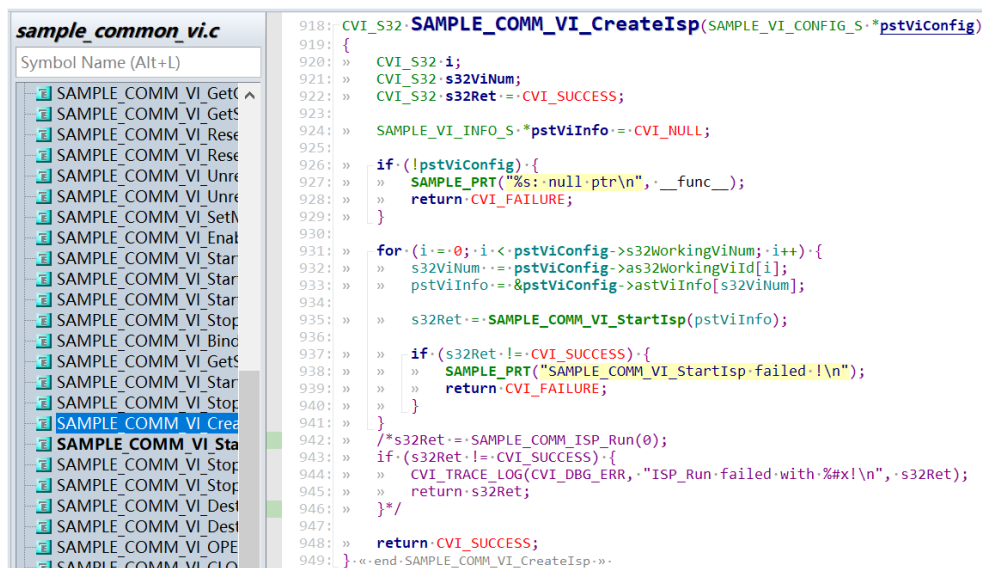
可直接调用文件系统内默认的 i2c_read/i2c_write 命令验证.

4.2 配置初始化序列

配置初始化序列建议参考版本发布包里的同厂商 sensor 驱动。

新 sensor bringup 建议先注释掉 AE 算法相关的 callbacks 以排除算法影响。

- 修改 sample_common_vi.c, 先去掉 SAMPLE_COMM_ISP_Run 调用。



- 修改 xxx_cmos_ctrl.c 中的 init 函数, 先注解 xxx_default_reg_init 调用。

```

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);

```

当 sensor 适配完能显示出图像后，记得重新打开这些注解。

4.2.1 准备 Sensor 驱动

- 依 Sensor 厂商、最大分辨率及 WDR 模式选择发布包内规格最接近的 sensor 驱动来作修改并编出 sensor 库。

具体可见 component/isp/user/sensor/cv18xx/xxxx 内的 xxxx_cmos.c、xxxx_cmos_ex.h、xxxx_cmos_param.h 与 xxxx_sensor_ctl.c

- 修改 xxxx_sensor_ctl.c 内的 I2C 配置如 i2c_addr, addr_byte 与 data_byte

```

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

```

- 依照 sensor 接口规格，修改 xxxx_cmos_param.h 中的 xxxx_rx_attr 与 pfnGetRxAttr，设置 mipi-rx 的属性。

```

176: struct combo_dev_attr_s gc2053_rx_attr = {
177: »    .input_mode = INPUT_MODE_MIPI,
178: »    .mac_clk = RX_MAC_CLK_600M,
179: »    .mipi_attr = {
180: » »    .raw_data_type = RAW_DATA_10BIT,
181: » »    .lane_id = {1, 3, 2, -1, -1},
182: » »    .wdr_mode = CVI_MIPI_WDR_MODE_NONE,
183: »    },
184: »    .mclk = {
185: » »    .cam = 0,
186: » »    .freq = CAMPLL_FREQ_27M,
187: »    },
188: »    .devno = 0,
189: };

```

.Input_mode: 设置输入模式是 mipi 还是 lvds 等等。

.Mac_clk: mac 时钟频率

.raw_date_type: data 的位宽

.lane id: mipi 数据 lane、时钟 lane 的 ID 配置

.cam: mclk ID

.freq: SOC 给 sensor 提供的参考输入时钟

.devno: mipirx 的编号，sensor ID

- 依照 sensor 输出模式，修改 xxxx_cmos_param.h 中的 g_astxxx_mode。

```
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,
        .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,
    },
},
}
```

- 修改 pfn_cmos_set_image_mode, 依照指定的宽高与帧率决定相符的 sensor 模式。

我们一般拿到的 init sequence 对应的输出模式都是最大分辨率, 也就是 all pixel scan 模式。

但有些情况客户需要对 sensor 吐出来 data 进行裁剪, 就需要适配成 window crop mode, 要找 sensor 厂商

提供 crop mode 下与之对应的 init sequence, 或者自行根据 sensor spec 进行修改。

4.2.2 Sensor 初始化序列

- 实作 xxxx_sensor_ctrl.c 内 sensor 模式的初始序列 pfn_cmos_sensor_init.
- 暂时注释 xxxx_sensor_ctrl.c 内的 xxxx_default_reg_init 的呼叫.
- 新增 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 适配 sample common 和 alios config

- 将 sensor object extern 到

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

的 getSnsObj(SNS_TYPE_E enSnsType) 函数中。

```
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_WDR2T01:
    >>         return &stSnsGc2093_Obj;
    #endif
    }
```

· 添加新的 `_SNS_TYPE_E` 到

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

的 `_SNS_TYPE_E` 枚举列表中, linear 在上半部, WDR 在下半部。


```

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 2T01 BEGIN -----*/
»   SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2T01 = SNS_TYPE_LINEAR_BUTT,
»   SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2T01,
»   SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2T01,

```

- sample_common_vi.c 在 SAMPLE_COMM_VI_GetDevAttrBySns、
SAMPLE_COMM_VI_GetChnAttrBySns、
SAMPLE_COMM_VI_GetSizeBySensor 添加对应的 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_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;

```

- 在 sample_common_vi.c 的 snsr_type_name 数组中加入新增的 sensor name, 注意 sensor name 要和新添加在 sensor_cfg.h 中的 _SNS_TYPE_E 的枚举名字和顺序一致。

```
1807 >> "SMS_SC4210_MIPI_4M_30FPS_12BIT",
1808 >> /* ----- LINEAR END -----*/
1809
1810 >> /* ----- WDR 2T01 BEGIN -----*/
1811 >> "SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2T01",
1812 >> "SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2T01",
1813 >> "SONY_IMX327_2L_MIPI_2M_30FPS_12BIT_WDR2T01",
1814 >> "SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT_WDR2T01",
1815 >> "SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2T01",
1816 >> "SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2T01",
1817 >> "OV_OS08A20_MIPI_8M_30FPS_10BIT_WDR2T01",
1818 >> "OV_OS08A20_MIPI_5M_30FPS_10BIT_WDR2T01",
1819 >> "SOI_F35_MIPI_2M_30FPS_10BIT_WDR2T01",
1820 >> "SOI_F35_SLAVE_MIPI_2M_30FPS_10BIT_WDR2T01",
1821 >> "SONY_IMX327_SUBLVDS_2M_30FPS_12BIT_WDR2T01",
1822 >> "SONY_IMX307_SUBLVDS_2M_30FPS_12BIT_WDR2T01",
1823 >> "SONY_IMX335_MIPI_5M_30FPS_10BIT_WDR2T01",
1824 >> "SONY_IMX335_MIPI_4M_30FPS_10BIT_WDR2T01",
1825 >> "SONY_IMX334_MIPI_8M_30FPS_12BIT_WDR2T01",
1826 >> "SMS_SC8238_MIPI_8M_15FPS_10BIT_WDR2T01",
1827 >> "SMS_SC4210_MIPI_4M_30FPS_10BIT_WDR2T01",
```

- 在 mars_alios/components/cvi_mmf_sdk/cvi_sensor/package.yaml 中添加对应的 sensor driver 目录名字和源文件信息

```

1 build_config:
2     include:
3         - sensor_cfg
4         - sensor_i2c
5         - gcore_gc02m1
6         - gcore_gc02m1_slave
7         - gcore_gc2053
8         - gcore_gc2053_1L
9         - gcore_gc2093
10
11         - gcore_gc4653
12         - pixelplus_pr2020
13         - sony_imx307
14         - sony_imx327
15         - techpoint_tp9950
16         - sms_sc1336_2L
17         - sms_sc1336_slave
18         - cista_c4390
19
20 5 source_file:
21 3 ## middleware
22 7     - sensor_cfg/*.c
23 3     - sensor_i2c/*.c
24 9     - gcore_gc02m1/*.c ? <CONFIG_SENSOR_GCORE_GC02M1>
25 0     - gcore_gc02m1_slave/*.c ? <CONFIG_SENSOR_GCORE_GC02M1_SLAVE>
26 1     - gcore_gc2053/*.c ? <CONFIG_SENSOR_GCORE_GC2053>
27 2     - gcore_gc2053_1L/*.c ? <CONFIG_SENSOR_GCORE_GC2053_1L>
28 3     - gcore_gc2093/*.c ? <CONFIG_SENSOR_GCORE_GC2093>
29 4     - gcore_gc4653/*.c ? <CONFIG_SENSOR_GCORE_GC4653>
30
31 5     - pixelplus_pr2020/*.c ? <CONFIG_SENSOR_PIXELPLUS_PR2020>
32 3     - sony_imx307/*.c ? <CONFIG_SENSOR_SONY_IMX307>
33 7     - sony_imx327/*.c ? <CONFIG_SENSOR_SONY_IMX327>
34 3     - techpoint_tp9950/*.c ? <CONFIG_SENSOR_TECHPOINT_TP9950>
35 9     - sms_sc1336_2L/*.c ? <CONFIG_SENSOR_SMS_SC1336_2L>
36 0     - sms_sc1336_slave/*.c ? <CONFIG_SENSOR_SMS_SC1336_SLAVE>
37 1     - cista_c4390/*.c ? <CONFIG_SENSOR_CISTA_C4390>
38 2

```

- 在 Linux 中，sensor 配置使用到的接口有如下，用法请参照 sample:
- CVI_S32 CVI_SENSOR_GPIO_Init(VI_PIPE ViPipe, SNS_I2C_GPIO_INFO_S *pstGpioCfg);

配置各个 sensor 的 reset GPIO 信息，SNS_I2C_GPIO_INFO_S 结构体如下：

```
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 *pst-Status);

获取 AHD Sensor 的状态，仅限于 AHD sensor，SNS_AHD_MODE_S 结构体如下：

```
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);
设置对应 PIPE 的 sensor ID，此方法需要在调用其他方法之前调用，SnsType 可以见 sensor_cfg.h
- CVI_S32 CVI_SENSOR_SetSnsRxAttr(VI_PIPE ViPipe, RX_INIT_ATTR_S *pstRx-Attr);

设置对应 sensor 的 RX 配置，RX_INIT_ATTR_S 结构体见 cvi_sns_ctrl.h

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

设置对应 sensor 的 I2C 总线和地址

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

设置 sensor 给到 ISP 的配置，ISP_INIT_ATTR_S 结构体见 cvi_sns_ctrl.h

- CVI_S32 CVI_SENSOR_RegCallback(VI_PIPE ViPipe, ISP_DEV IspDev);
设置 sensor 和 ISP 的 callback
- CVI_S32 CVI_SENSOR_UnRegCallback(VI_PIPE ViPipe, ISP_DEV IspDev);
解除 sensor 和 ISP 的 callback
- CVI_S32 CVI_SENSOR_SetSnsImgMode(VI_PIPE ViPipe, ISP_CMOS_SENSOR_IMAGE_MODE_S *stSnsrMode);
设置 sensor 要跑的 mode, 包括 fps, size 等, ISP_CMOS_SENSOR_IMAGE_MODE_S 结构体见 cvi_comm_sns.h
- CVI_S32 CVI_SENSOR_SetSnsWdrMode(VI_PIPE ViPipe, WDR_MODE_E wdr-Mode);
设置 sensor 的 WDR mode, WDR_MODE_E 结构体见 cvi_comm_cif.h
- CVI_S32 CVI_SENSOR_GetSnsRxAttr(VI_PIPE ViPipe, SNS_COMBO_DEV_ATTR_S *stDevAttr);
获取 sensor 的 RX 配置, SNS_COMBO_DEV_ATTR_S 结构体见 cvi_comm_cif.h
- CVI_S32 CVI_SENSOR_SetSnsProbe(VI_PIPE ViPipe);
设置对应 PIPE 的 sensor 的 probe
- CVI_S32 CVI_SENSOR_SetSnsGpioInit(CVI_U32 devNo, CVI_U32 u32Rst_port_idx, CVI_U32 u32Rst_pin, CVI_U32 u32Rst_pol);
配置各个 sensor 的 reset GPIO 信息, u32Rst_port_idx, u32Rst_pin, u32Rst_pol 见下节的 ini 配置内容
- CVI_S32 CVI_SENSOR_RstSnsGpio(CVI_U32 devNo, CVI_U32 rstEnable);
将 sensor 的 rst 脚拉置有效位
- CVI_S32 CVI_SENSOR_RstMipi(CVI_U32 devNo, CVI_U32 rstEnable);
reset 对应的 sensor 使用的 MIPI
- CVI_S32 CVI_SENSOR_SetMipiAttr(VI_PIPE ViPipe, CVI_U32 SnsType);
将 sensor 的 RX 配置给 CIF
- CVI_S32 CVI_SENSOR_EnableSnsClk(CVI_U32 devNo, CVI_U32 clkEnable);
enable sensor 的 mclk
- CVI_S32 CVI_SENSOR_SetSnsStandby(VI_PIPE ViPipe);
设置 sensor 的 standby 状态
- CVI_S32 CVI_SENSOR_SetSnsInit(VI_PIPE ViPipe);
设置 sensor 开始 init
- CVI_S32 CVI_SENSOR_SetVIFlipMirrorCB(VI_PIPE ViPipe, VI_DEV ViDev);
将 sensor 的 mirror 和 flip 注册到 VI 之中
- 下面的方法提供给 ISP 使用, 使用时请配合 ISP 相关文档查看

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

获取对应 sensor 的 AE default 状态

- `CVI_S32 CVI_SENSOR_GetIsrBlkLev(VI_PIPE ViPipe, ISP_CMOS_BLACK_LEVEL_S *stBlc);`

获取对应 sensor 的 BLK 值, ISP_CMOS_BLACK_LEVEL_S 结构体见 `cvi_comm_sns.h`

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

设置 sensor 的输出 FPS

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

获取 sensor 的曝光范围

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

设置 sensor 的数字增益值

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

设置 sensor 的模拟增益值

4.4 添加 sensor ini cfg 配置

Sensor 的一些属性可以通过改 ini 来修改配置, 比如 lane 线序、I2C 端口 sensor 输出模式等。

默认情况下 middleware 的流程会优先从 `/mnt/data/sensor_ini.cfg` 下读取 sensor 的配置文件, 如果该目录下没有配置文件, 会使用代码内部的初始值。

下面以 SC1336 为例显示的 `sensor_cfg.ini` 内容:

```
[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
```

(下页继续)

(续上页)

```
lane_id = 2, 3, 1, -1, -1  
pn_swap = 1, 1, 1, 0, 0  
mclk_en = 1  
mclk = 0  
port = 0  
pin = 2  
pol = 1  
fps = 60
```

- name: 表示 sensor 的输出模式，注意要和新添加在 sample_comm.h 中的 SAMPLE_SNS_TYPE_E 的枚举名字一致。
- Bus_id: 表示 I2C 端口号
- Mipi_dev: 表示用哪一组 mipi-rx
- Lane_id: 表示 mipi 的线序配置
- pn_swap: 表示这组 mipi 线序 P/N 是否需要反转
- Pn_swap: 表示 P/N 反转，不需要反转贴成 0，需要反转配置成 1
- Mclk: 表示选用哪一组 mclk 作为参考时钟
- Mclk_en: 表示使能哪一组 mclk 输出
- hw_sync: dual sensor 帧同步，hw_sync=1 表示 slave sensor sync with master sensor
- sns_i2c_addr: sensor 的 i2c 设备地址
- port: sensor RST 引脚对应到处理器的 GPIO 使用的 port 口 A/B/C - 0/1/2
- pin: sensor RST 引脚对应到处理器的 GPIO 使用的 port 口的编号
- pol: sensor RST 引脚的有效电平

对应的参数配置如下： 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: sensor 的输出 fps，默认为 25，其他 fps 需要设置对应的 fps 值

4.5 编译运行 sensor_test

前面小节配置完后，在 SDK 顶层目录执行 `make peripherals_test` 进行编译，将编译后的固件烧录到板端；

烧录启动后，Linux 串口终端执行 `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
```

在 alios 串口输入 `proc/vi_dbg` 查看 `vi_dbg` 信息，帧率显示正常则表示 sensor 有正常出图

```
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]
VIIsTopStatus       :0x386
[VI DMA_Dbg_Info]
VIWdma0ErrStatus    :0x3000000     VIWdma0IdleStatus    :0xfffffeaf
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
VIInImaHeight       :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 图像输出调试 (Alios 快启)

5.1 硬件准备

- 确认 Sensor 电源供应正确.
 - 确认 Sensor Reset GPIO 正确.
 - 确认 Sensor 输入参考时钟来源 (主处理器或外部晶振)
 - 确认 I2C 可擦写 Sensor 寄存器.
- 可直接调用文件系统内默认的 iic read /iic write 命令验证.

5.2 配置初始化序列

配置初始化序列建议参考版本发布包里的同厂商 sensor 驱动。

新 sensor bringup 建议先注释掉 AE 算法相关的 callbacks 以排除算法影响。

- 修改 components/cvi_platform/media/src/media_video.c, 先去掉 CVI_ISP_Run 调用。

```
218  #if(CONFIG_APP_ISP_BYPASS == 0)
219      CVI_ISP_SetBypassFrm(0, 0);
220  #endif
221
222      s32Ret = CVI_ISP_Init(ViPipe);
223      if (s32Ret != CVI_SUCCESS) {
224          MEDIABUG_PRINTF("ISP Init failed with %#x!\n", s32Ret);
225          return s32Ret;
226      }
227
228      //Run ISP
229      // s32Ret = CVI_ISP_Run(ViPipe);
230      // if (s32Ret != CVI_SUCCESS) {
231      //     MEDIABUG_PRINTF("ISP Run failed with %#x!\n", s32Ret);
232      //     return s32Ret;
233      // }
```

- 修改 xxx_cmos_ctrl.c 中的 init 函数, 先注解 xxx_default_reg_init 调用。

```

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);

```

当 sensor 适配完能显示出图像后，记得重新打开这些注解。

5.2.1 准备 Sensor 驱动

- 依 Sensor 厂商、最大分辨率及 WDR 模式选择发布包内规格最接近的 sensor 驱动来作修改并编出 sensor 库。

具体可见 mars_alios/components/cvi_mmf_sdk/cvi_sensor/xxxx 内的 xxxx_cmos.c、xxxx_cmos_ex.h、xxxx_cmos_param.h 与 xxxx_sensor_ctl.c

- 修改 xxxx_sensor_ctl.c 内的 I2C 配置如 i2c_addr, addr_byte 与 data_byte

```

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

```

- 依照 sensor 接口规格，修改 xxxx_cmos_param.h 中的 xxxx_rx_attr 与 pfnGetRxAttr，设置 mipi-rx 的属性。

```

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,
};

```

.Input_mode:设置输入模式是mipi还是lvds等等。

.Mac_clk: mac时钟频率

.raw_data_type:data的位宽

.lane id:mipi数据lane、时钟lane的ID配置

.cam: mclk ID

.freq: SOC给sensor提供的参考输入时钟

.devno:mipirx的编号, sensor ID

- 依照 sensor 输出模式, 修改 xxxx_cmos_param.h 中的 g_astxxx_mode.

```
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,
            },
            .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,
        },
    },
};
```

(下页继续)

(续上页)

```
.u32Step = 1,  
    },  
},  
};
```

- 修改 pfn_cmos_set_image_mode, 依照指定的宽高与帧率决定相符的 sensor 模式.

我们一般拿到的 init sequence 对应的输出模式都是最大分辨率, 也就是 all pixel scan 模式.

但有些情况客户需要对 sensor 吐出来 data 进行裁剪, 就需要适配成 window crop mode,

要找 sensor 厂商提供 crop mode 下与之对应的 init sequence, 或者自行根据 sensor spec 进行修改.

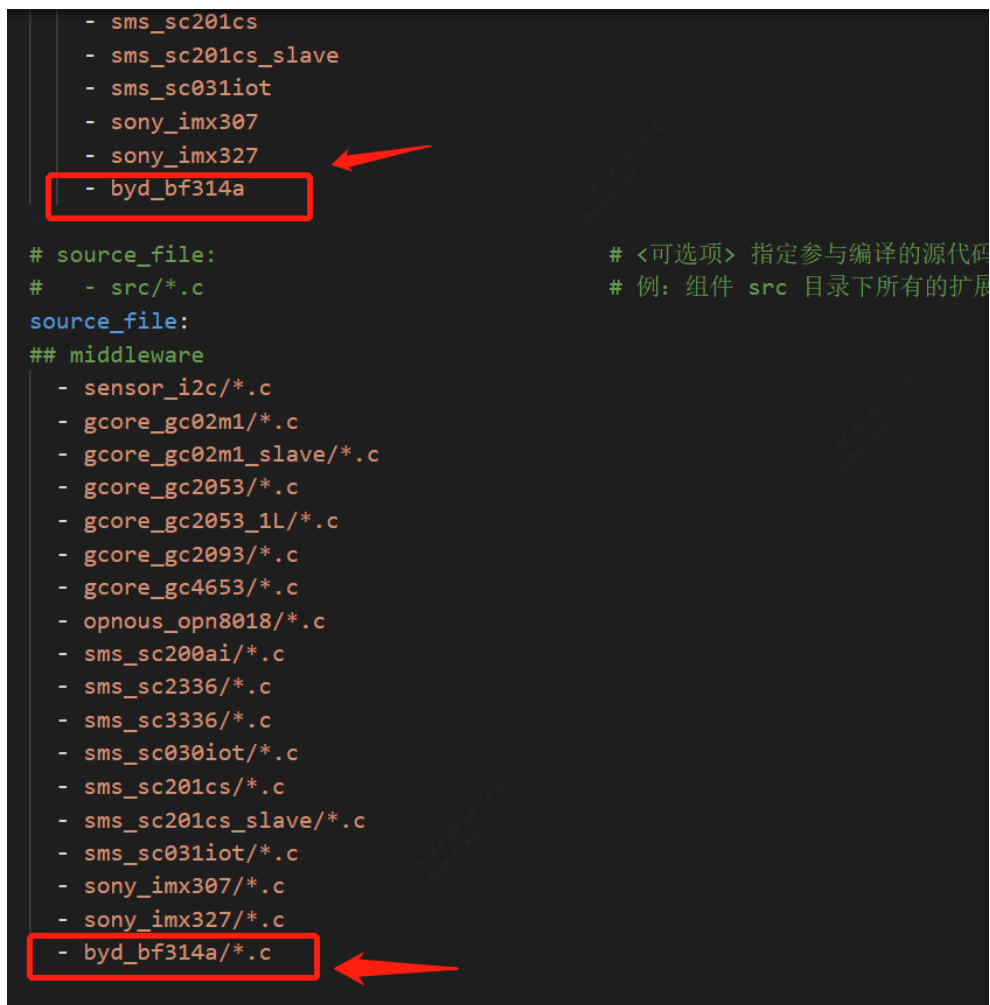
5.2.2 Sensor 初始化序列

- 实作 xxxx_sensor_ctrl.c 内 sensor 模式的初始序列 pfn_cmos_sensor_init.
- 暂时注释 xxxx_sensor_ctrl.c 内的 xxxx_default_reg_init 的呼叫.
- 新增 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 修改 package.yaml 添加编译文件

- 修改 components/cvi_mmf_sdk/cvi_sensor/package.yaml, 添加头文件和源文件



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

# source_file:
#   - src/*.c
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
```

<可选项> 指定参与编译的源代码
例: 组件 **src** 目录下所有的扩展

5.3 适配 solution

5.3.1 增加 sensor type

- mars_alios/components/cvi_mmf_sdk/cvi_sensor/sensor_cfg/sensor_cfg.h 的 `_SNS_TYPE_E` 增加一个 type

```

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
增加 sensor object, 在 getPicSize, getDevAttr 函数中添加对应的 case

```

#ifdef CONFIG_SENSOR_SONY_IMX307_2L
extern ISP_SNS_OBJ_S stSnsImx307_2l_Obj;
#endif
#ifdef CONFIG_SENSOR_SONY_IMX307_SLAVE
extern ISP_SNS_OBJ_S stSnsImx307_Slave_Obj;
#endif
#ifdef CONFIG_SENSOR_SONY_IMX327
extern ISP_SNS_OBJ_S stSnsImx327_Obj;
#endif
#ifdef 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 修改 sensor mipi 相关配置

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

这里面配置的 reset 管脚, mipi lane 等配置会替代 sensor driver 默认设定的信息

配置说明:

s32I2cAddr: sensor 的 i2c 设备地址

s8I2cDev: 表示 I2C 端口号

u32Rst_port_idx: reset 管脚的 GPIO group

u32Rst_pin: reset 管脚的 GPIO num

as16LaneId: 表示 mipi 的线序配置

as8PNSwap: 表示 P/N 反转, 不需要反转贴成 0, 需要反转贴成 1

u8MclkCam: 表示选用哪一组 mclk 作为参考时钟

s16MacClk: mac clk

u8MclkFreq: MCLK 频率

bHwSync: dual sensor 帧同步, bHwSync=1 表示 slave sensor sync with master sensor

s32Framerate: 帧率

```
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,
},
},
```

如果想默认使用驱动的参数, 则可以只配置下图这些即可

```
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 修改 VB 配置

- solutions/peripherals_test/customization/peripherals_qfn/param/custom_sysparam.c 根据 sensor 输出 size 修改 VB 的 u16width 和 u16height

```
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 修改添加 pinmux

- solutions/peripherals_test/customization/peripherals_qfn/src/custom_platform.c 根据板子实际硬件配置, 修改 mipi i2c reset 等管脚复用

```
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 修改编译配置

- 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 运行 sensor

编译运行后，快启下会直接启动 sensor，在 alios 串口输入 `proc/vi_dbg` 查看 `vi_dbg` 信息，帧率显示正常则表示 sensor 有正常出图

```
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    :0xfffffeaf
VIWdma1ErrStatus    :0x3000000     VIWdma1IdleStatus    :0xfffffe054
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
VISOFCnt             :19893
VIPreFECh0Cnt       :19892
VIPreBECh0Cnt       :19892
VIPostCnt            :19892
VIDropCnt            : 0
[VI ISP_PIPE_A Csi_Dbg_Info]
VICsiIntStatus0     :0x0
VICsiIntStatus1     :0x0
VICsiCh0Dbg         :0x0
VICsiCh1Dbg         :0x0
```

6 图像输出验证

配置好 init setting 后如果时序满足 sensor 的工作要求，并且没有 select timeout 的打印，就可以确认 sensor 图像有正常输出了。

若有异常请参考[错误检查流程](#)。

下面以 sensor_test 为例说明如何确认 sensor 的图像输出。请注意，只有非快启模式下才可以使用 sensor_test 来进行测试，在 Linux 端串口操作，打印会在 alios 端输出

图像查看需用到 PC 工具 CvitekRawViewer，找对应 FAE 获取

注意：若前面已经注释掉 AE 相关函数，此时使用的是厂商默认的初始设定，有可能出现图像偏暗或全黑的情况，要手动调整 sensor 曝光增益寄存器。

6.1 Dump RAW

运行 sensor_test，输入 1 选择 “dump vi raw data “，然后根据提示” To get raw dump from dev(0~1): ”，输入 dev (0 表示 vi pipe0，从第 0 路 sensor dump 图像，1 表示 vi pipe1，从第 1 路 sensor dump 图像)。

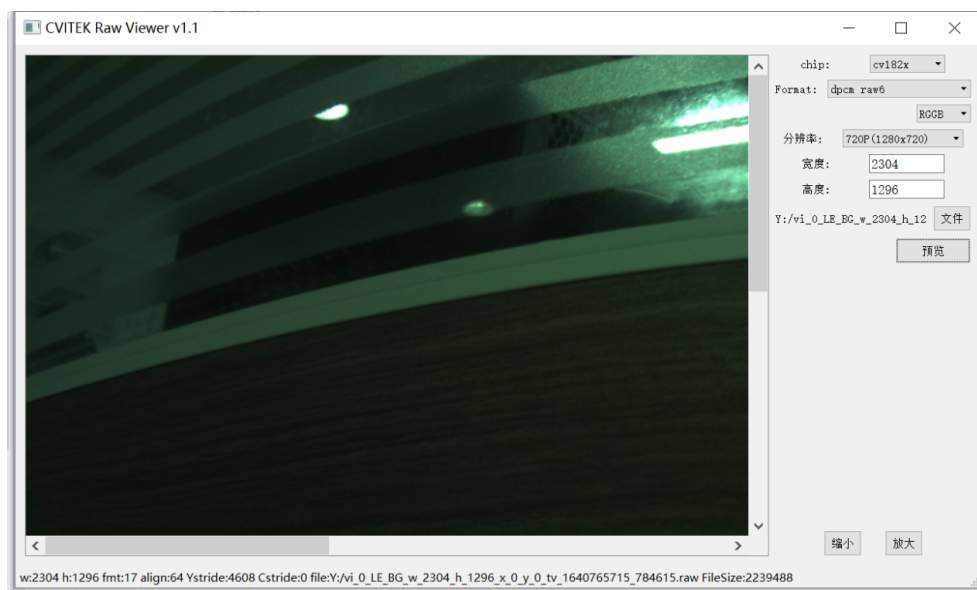
然后根据提示” how many loops to do (1~60)”，输入 loops(表示要 dump 多少 frame)。

注解： Alios 需要插入 SD 卡，dump 文件默认存到 SD 中

RAW 图查看方式：

将 dump 出的 raw 图像在电脑上使用 CvitekRawViewer 工具，配置对应的 processor、format、width、height

工具使用如下图：

**注解:**

- raw 图像显示应该偏绿，如果偏紫或有斜线需检查 Bayer format, flip/mirror 等相关配置是否正确。
- 宽度、高度、颜色格式一般可以由 dump 出的文件名称获取。
- sensor_test 默认使用 raw 图压缩模式 COMPRESS_MODE_TILE，所以在工具中应选择 “dpcm raw6”，若不开启压缩模式，请选择 “raw12”。

6.2 Dump YUV

运行 sensor_test，输入 2 选择 “dump vi yuv”，按照提示 dump yuv 图像：

电脑上使用 CvitekRawViewer 工具，配置对应的 processor、format、width、height



7 ISP 基本功能

Sensor 驱动功能皆由 operation callbacks 实现。本章节以使用者以详悉 Sensor datasheet 为前提描述 ISP Callbacks 应实现的基本功能。调试 ISP 相关 callback 时，请重新打开之前注解的 SAMPLE_COMM_ISP_Run , xxx_default_reg_init 的调用。

7.1 开发流程

请依序实现以下 ISP 基本功能 callbacks

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 注意事项

- pfn_cmos_sensor_init: 使用 Sensor 沟通接口 (I2C/SPI) 实现厂商提供的初始序列。
应注意沟通接口结构的正确性。

因为 AE 相关的 Callbacks 也会在 sensor init 之前呼叫, 需在 sensor 开始输出数据之前设定 Sensor AE 缓存器。

可参考 xxxx_sensor_ctrl.c 内的 xxxx_default_reg_init。

- pfn_cmos_sensor_exit: 关闭使用的沟通接口。
- pfn_cmos_sensor_global_init: 初始化 sensor 驱动参数。
- pfn_cmos_set_image_mode: 设定 sensor 输出的格式。Sensor 驱动应选择最接近的分辨率作输出格式。
- pfn_cmos_set_wdr_mode: 设定 sensor 输出是否为 WDR 模式。

- `pfn_cmos_get_isp_default`: 提供与 sensor 相关的 ISP 参数。
- `pfn_cmos_get_sns_reg_info`: 提供储存在 Sensor 驱动里的 AE 同步讯息。为了同步 AE 设定与 Sensor 输出的影像,

当 AE callbacks 被呼叫时, Sensor 驱动不会立刻写入 sensor 的缓存器, 而是储存修改的设定;

Firmware 会在固定周期呼叫 `pfn_cmos_get_sns_reg_info` 获得同步信息并且传递给 kernel space 的 ISP 驱动。

ISP 驱动负责同步写入 sensor 缓存器。

此外 Sensor 可能有不同的 WDR 输出格式, 因此影像的大小、Crop 位置与 MIPI-RX 设定也可能随着不同的曝光值重新计算与设定。

Sensor 驱动须询问厂商计算公式, ISP 驱动会依此更新对应的模块。

- `pfn_cmos_get_sns_reg_info` 回传的结构分三大类:

```
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` 表示需要同步的 sensor 缓存器, `ispCfg` 表示需要同步的 Crop 信息, `cifCfg` 表示需要同步的 mipi-rx 设定。

当 `need_update` 为 True 时, 表示这类的同步数据需要 ISP 在指定的 `u8DelayFrmNum` 更新。

`snsCfg` 内每一个缓存器也有 `bUpdate` 指示缓存器是否需要更新。

第一次调用 `pfn_cmos_get_sns_reg_info` 会进行配置 i2c 相关讯息, 建立 register 地址映像、获取 `sns`、`crop`、`wdr size` 等信息, 如下图。

```

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;
        }
    }
    if (pstSnsState->bSyncIn... else {

```

后面反复调用 pfn_cmos_get_sns_reg_info 则是进行对修改的 AE register 讯息做出暂存，如下图。

```

} 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);
    }
}

```

暂存后的 AE 寄存器讯息最终会 call isp_snsSync_info_set 将 AE 资料 updata 到 ISP driver, ISP driver 会在 delayFrmNum 后下 i2c, 设定 sensor 寄存器。

- pfn_cmos_get_isp_black_level:

从 sensor spec 中获取 black level offset.

将 offset 转成 12 bit 并代入公式得到 $gain = 4095 / (4095 - offset) * 1024$

```
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},
        .tAuto = {
            {257, 257, 257, 259, 259, 260, 267, 278, 298, 366, 383, 366, 373, 372, 372},
            {257, 257, 257, 258, 259, 261, 266, 274, 297, 379, 377, 372, 365, 373, 374},
            {257, 257, 257, 258, 259, 261, 266, 275, 296, 376, 388, 366, 374, 376, 372},
            {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 完成 AE 配置功能

Sensor 驱动功能皆由 operation callbacks 实现。

本章节以使用者已详悉 Sensor datasheet 为前提描述 AE Callbacks 应实现的基本功能。

8.1 开发流程

请依序实现以下 AE 基本功能 callbacks

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 注意事项

- pfn_cmos_get_ae_default: 回传 AE 算法相关的 sensor 数据。

需要提供 AE 算法在 linear 模式下的最大及最小曝光条数, linear/ WDR 模式仿真/数字增益的最大及最小值以及增益的类型, 数字增益若是只有如: 0db, 6db, 12db 等少数几种选择, 则为 DB 型, 其余为线性, 曝光生效周期的 frame 数, 开机第几个 frame 后稳定的 frame 数。

- u32FullLinesStd: 在初使化序列内, 一帧内时间的 line 数.
- u32MaxAgain: 最大的 AGain 值.
- u32MinAgain: 最小的 AGain 值.
- u32MaxDgain: 最大的 DGain 值.
- u32MinDgain: 最小的 DGain 值.
- u32MaxIntTime: 线性模式中最大的曝光值.

- u32MinIntTimeTarget: 线性模式中最小的曝光值。
- u32AEResponseFrame: AE 最大的反应时间 (unit: frame)

主要是要按照 sensor spec 填写相关 AE 的属性, 包括 FullLinesStd, FullLinesMax, IntTime 的 max、min 以及 step, gain 的 max 和 min 以及 step 等。注意要确认好 intTime 和 gain 的 AccuType:

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

一般情况下 intTime 设定和对应 register 是成线性关系的, AccuType 配置成 AE_ACCURACY_LINEAR;

gain 的设定一般设为 AE_ACCURACY_TABLE, 表示从 gain table 去映射, 后面会介绍 pfn_cmos_again_calc_table/pfn_cmos_dgain_calc_table。

不过有些 sensor 可能 gain 设定比较特殊, 比如 soi_F35 这款, 它的 dgain 就不能做到很细的调节, 只有 1x, 2x, 3x, 4x 这 4 个挡位。

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: 设定 sensor 帧率。

默认为 Sensor 输出模式的最大帧率。Sensor 驱动增加输出的垂直 Blanking 条数达到降低帧率的效果。

注意由于输出总条数改变, 有些 Sensor 的曝光范围也会改变, Sensor 驱动须重新计算。

例如, 若初始序列的 FPS=30, 新的 FPS 不能大于 30。通常调整 frame rate 的方法是依照比例增加 sensor 输出的 full lines。

例如, 若在 FPS=30 时, full lines = 1125。FPS=25 时的 full lines = $1125 \times 30 / 25 = 1350$ 。

- pfn_cmos_inttime_update: 设定 Sensor 的曝光时间并回传实际生效的曝光条数给 AE。

输入参数为一数列, 在 WDR 模式依序代表短曝帧与长曝帧的曝光值, 单位为水平输出条数。

例如, 当 u32IntTime[0]=8, u32IntTime[1]=1000, 代表短曝帧曝光为 8 条线的时间, 长曝帧为 1000 条线的时间。

若在线性模式, 序列 [0] 即代表曝光值, 序列 [1] 无意义。注意在 WDR 模式, 调整 Sensor 短帧曝光可能需要重新计算 Crop 信息与 mipi-rx 设定。

- pfn_cmos_gains_update: 设定 Sensor 的增益值。

输入参数为 pu32Again 与 pu32Dgain 两个序列。

在 WDR 模式, pu32Again[0] 代表短曝帧的模拟增益值, pu32Again[[1] 代表长曝帧的模拟增益值;

pu32Dgain[0] 代表短曝帧的数字增益值, pu32Dgain[[1] 代表长曝帧的数字增益值。

数值为 Sensor 缓存器的设定值, 可由 pfn_cmos_again_calc_table 与 pfn_cmos_dgain_calc_table 转换真实增益值。

在线性模式, 仅 pu32Again[0] 与 pu32Dgain[0] 有意义。

可由 pfn_cmos_again_calc_table 与 pfn_cmos_dgain_calc_table 转换真实增益值。

在线性模式, 只有 pu32Again[0] 和 pu32Dgain[0] 有意义。

在 WDR mode 下:

- pu32Again[0]: 短帧的 Again 配置.
- pu32Again[1]: 长帧的 Again 配置.
- pu32Dgain[0]: 短帧的 Dgain 配置.
- pu32Dgain[1]: 长帧的 Dgain 配置.
- Gains update 有 3 种模式: SHARE, WDR_2F, ONLY_LEF, 设置在 pfnSetInit.
- SHARE: 长短帧共享 Gain 配置 (Sony, OV).
- WDR_2F: 长短帧分开 Gain 配置 (Sony, OV).
- ONLY_LEF: 仅长曝帧 Gain 可配置 (SOI).
- pfn_cmos_again_calc_table: 输入为 1024 为基准的模拟增益值, Sensor 驱动经查表或计算出出不大于输入值且最相近的模拟增益值, 输出对应的 Sensor 缓存器设定。
 - pu32AgainLin: AE 传入 1024-based 的 Again.
Sensor 驱动依照规格书的 gain table 或公式, 回传最接近的 1024-based Again.
Again 的范围定义于 pfn_cmos_get_ae_default
 - pu32AgainDb: 回传对应的 Sensor Again register 配置.
- pfn_cmos_dgain_calc_table: 输入为 1024 为基准的数字增益值, Sensor 驱动经查表或计算出出不大于输入值且最相近的数字增益值, 输出对应的 Sensor 缓存器设定。
 - pu32DgainLin: AE 传入 1024-based 的 Dgain.
Sensor 驱动依照规格书的 gain table 或公式, 回传最接近的 1024-based Dgain.
Dgain 的范围定义于 pfn_cmos_get_ae_default
 - pu32DgainDb: 回传对应的 Sensor Dgain register 配置.
若 sensor Dgain 调整为阶型 (1X, 2X, 4X...),
pfn_cmos_get_ae_default 内的 stDgainAccu.enAccuType 须设定成 AE_ACCURACY_DB.
- pfn_cmos_get_inttime_max: 用于 WDR 模式, 计算在当前的曝光比下长短帧可容许曝光条数的范围。

SONY DOL, F35 HDR-wo VC, OV HDR-DT, Smartsens SC200AI 都是利用 blanking 区间实现短帧曝光。

有些 sensor (OS08A20, F35) 可设定成 Fix L2S distance, 即设定一个最大的短帧曝光值, 当调整短帧曝光时, L2S distance 不会变化, ISP crop size 不需动态配置.

- u16ManRatioEnable: Manual Ratio Enable, 设定为 1
- au32Ratio[0] : 2 frame HDR 下, 长帧曝光 *64/短帧曝光比
- au32IntTimeMax[0] : 短帧最大曝光值 (单位: 一条 H 时间)
- au32IntTimeMax[1] : 长帧最大曝光值 (单位: 一条 H 时间)
- au32IntTimeMin[0] : 短帧最小曝光值 (单位: 一条 H 时间)
- au32IntTimeMin[1] : 长帧最小曝光值 (单位: 一条 H 时间)
- pu32LFMaxIntTime[0] : NA

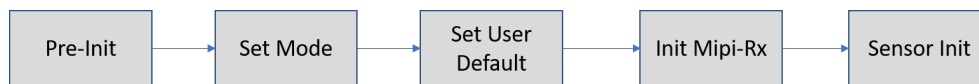
9 完善其它功能

9.1 Sensor 初始化流程

除了 AE/ISP 之外，Sensor 驱动也使用其他 callbacks 完善初始化流程。

Sensor callbacks 中一些参数设定可能会互相影响，须注意呼叫的顺序。

建议的呼叫流程如下：



Pre-Init 准备 Sensor 驱动的环境，呼叫的 callback 如下：

- pfnSetInit：初始化 sensor 通用的参数。enGainMode 决定 sensor 在 WDR 模式下 Gain 的行为。
- pfnSetBusInfo：设定 I2C 信息。
- pfnRegisterCallback：注册 sensor ISP/AE callbacks。
- pfn_cmos_sensor_global_init：初始化 sensor 驱动内部参数。

Set Mode 决定 Sensor 输出的主要格式，呼叫 callback 如下：

- pfn_cmos_set_image_mode：设定输出的影像格式。
- pfn_cmos_set_wdr_mode：设定线性或 WDR 模式。

Set User Default 设定初始化序列的 AE 参数，呼叫的 callback 如下：

- pfn_cmos_fps_set：设定每秒的帧数。从 callback pfn_cmos_get_ae_default 得到预设的帧率 f32Fps，注意新的帧率不得大于默认值。
- pfn_cmos_inttime_update：设定曝光时间的条数并回传给 AE，

线性模式下的曝光条数范围可以由 pfn_cmos_get_ae_default 的 u32MaxIntTime 与 u32MinIntTime。

WDR 模式下可呼叫 pfn_cmos_get_inttime_max 根据曝光比率得到长短曝的曝光条数范围。

- pfn_cmos_gains_update: 设定 Sensor 的 AGAIN 与 DGAIN。

从 pfn_cmos_get_ae_default 的 u32MaxAgain/u32MaxDgain 与 u32MinAgain/u32MinDgain 得到 Gain 范围，

可呼叫 pfn_cmos_again_calc_table/pfn_cmos_dgain_calc_table 得到最接近的 Gain 与对应的 Sensor 缓存器的设定值。

Init Mipi-Rx 初始化 Mipi-Rx 参数, Sensor 开机序列 (Power On Sequence)。

Linux 由 ioctl 呼叫 kernel 的 Mipi-Rx 驱动进行操作。

主要的程序如下:

- 打开/dev/video0。此步骤会打开 VIP 相关电源与时钟。
- 呼叫 Sensor 驱动的 callback pfnGetRxAttr 得到对应 Sensor 的 Mipi-Rx 设定。
- CVI_MIPI_RESET_SENSOR: Mipi-Rx 的 ioctl。呼叫可打开定义在 device tree 的 Sensor Reset pin。

```
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: Mipi-Rx 的 ioctl。呼叫可重置 Mipi-Rx 的设定。
- CVI_MIPI_SET_DEV_ATTR: Mipi-Rx 的 ioctl。呼叫可设定 Mipi-Rx 属性。
- CVI_MIPI_ENABLE_SENSOR_CLOCK: Mipi-Rx 的 ioctl。呼叫可打开 Sensor 时钟。频率由 CVI_MIPI_SET_DEV_ATTR 的 mclk 属性决定。
- CVI_MIPI_UNRESET_SENSOR: Mipi-Rx 的 ioctl。呼叫可关闭定义在 device tree 的 Sensor Reset pin。

Alios 则通过配置 custom_viparam.c 文件来设定 reset pin 的属性

```
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,
```

(下页继续)

(续上页)

```
/* config dev attr(reference sensor driver) */
.bSetDevAttr = 1,
.u8MclkCam = 0,
.s16MacClk = RX_MAC_CLK_200M,
.u8MclkFreq = CAMPLL_FREQ_27M,
.bHwSync = CVI_FALSE,
.s32Framerate = 30,
}
```

Sensor Init 呼叫 Sensor 驱动的 callback `pfn_cmos_sensor_init` 启动 Sensor 的初始序列。

9.2 Sensor 关闭流程

关闭 Sensor 时可参考以下流程：



- Disable ISP：关闭近端 ISP 接口。
- Disable Sensor：呼叫 sensor 驱动 callback `pfn_cmos_sensor_exit` 关闭 sensor 码流与 I2C 接口。呼叫 `pfnUnRegisterCallback` 卸除 Sensor 驱动。
- Linux 呼叫 Mipi-Rx ioctl `CVI_MIPI_RESET_SENSOR` 打开 Sensor reset pin。呼叫 `CVI_MIPI_DISABLE_SENSOR_CLOCK` 关闭 Sensor 时钟。呼叫 `CVI_MIPI_RESET_MIPI` 重置 Mipi-Rx 设定。

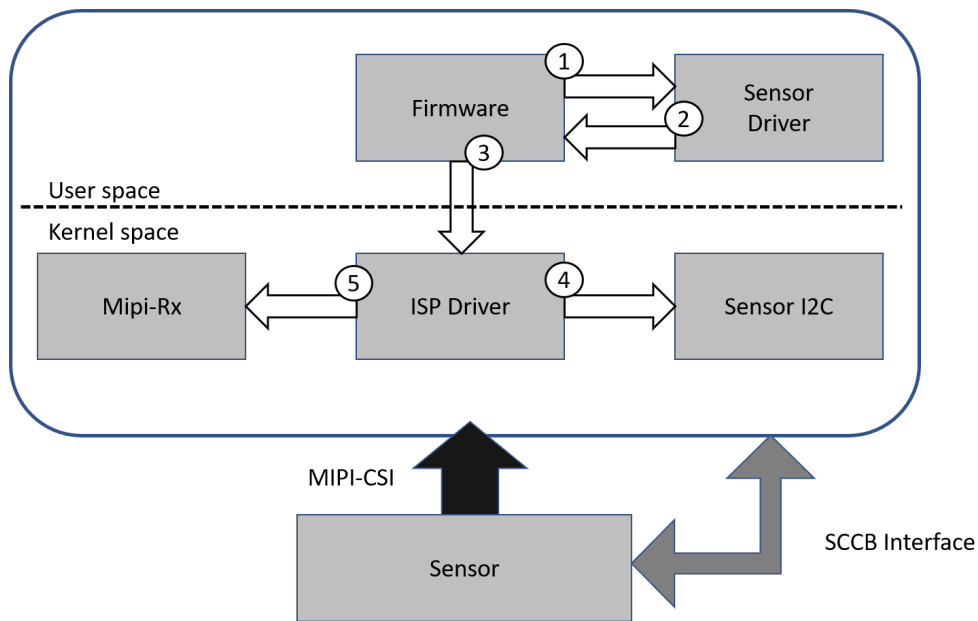
Alios 呼叫 `cif_reset_snsr_gpio` 打开 Sensor reset pin，呼叫 `cif_reset_mipi` 重置 mipi-rx 设定。

9.3 Sensor AE 同步流程

Sensor 曝光与增益设定后可能反应在不同的帧上，因此需要有机制同步 Sensor 与 ISP 的设定。

此外，在 WDR Manual 模式，调整短曝帧的曝光可能需要更新 Mipi-Rx 的设定。

以下是 Sensor AE 同步流程：



1. Firmware 呼叫 sensor callbacks `pfn_cmos_gains_update` 与 `pfn_cmos_inttime_update` 更新 AE 设定。
2. Firmware 在固定周期呼叫 sensor callbacks `pfn_cmos_get_sns_reg_info` 得到 sensor/ISP/Mipi-Rx 的设定。
3. Firmware 将 sensor/ISP/CIF 设定经由 ISP ioctl 传递给 ISP 驱动内的同步处理机制。
4. 当需要更新 sensor 设定时, ISP driver 呼叫在 `cv18xx_vip.ko` 内的 I2C 介面更新 sensor 缓存器。
5. 当需要更新 Mipi-Rx 的设定时, ISP driver 呼叫在 `cvi_mipi_rx.ko` 内的 Mipi-Rx 驱动。

10 AE 相关验证

当完成图像验证后就可以进行 AE 交接工作，AE 交接要确保基本的曝光、增益是线性的，还有反应帧、同步问题等需要验证。

主要是完成 SensorPorting_AE(sensor_test) 这份表格的验证。验证工作需要用到灯箱和 sensor_test 测试程序。

注意：在进行 AE 相关验证时，需要将之前注释的代码给放开。

10.1 BLC 确认和验证

sensor spec 上一般都会写 blc offset 的值，可直接写入 xxx_cmos_param.h。

如果没有可通过以下方式测试得出实际 blc 值：

修改 xxx_cmos_param.h，将下面红色圈出的部分，273 表示 blc offset，全部改为 0，1097 表示 gain，全部改成 1024。

```
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},
        },
    },
};
```

遮住镜头，在纯黑环境下运行 sensor_test，输入 CMD

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

打印出 Luma 值再乘 4 就是对应的 blc offset 值。

比如下面对应的 blc offset 为 $74 \times 4 = 286$ 。

```

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

```

最后将测试出来的 blc offset 代入公式 $gain = 4095 / (4095 - offset) * 1024$ 得到 1106，将确认好的 blc 和 gain 填入 xxx_cmos_param.h。

10.2 曝光线性度验证

将镜头对着灯箱环境下运行 sensor_test，linear 模式下输入 CMD

```

5
2 0 71 0 0

```

Wdr 模式下长曝光输入 CMD

```

5
2 0 75 0 0

```

Wdr 模式下短曝光输入 CMD

```

5
2 0 76 0 0

```

要满足 AE 亮度的统计值在 1/60s 的曝光时间应为 1/30s 的一半，AE 亮度的统计值在 1/120s 的曝光时间应为 1/60s 的一半这样的关系，比如下图所示结果就是符合的。

```
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
Sensor EL:749 AG:0 DG:0
```

10.3 增益线性度验证

将镜头对着灯箱环境下运行 sensor_test, linear 模式下输入 CMD

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

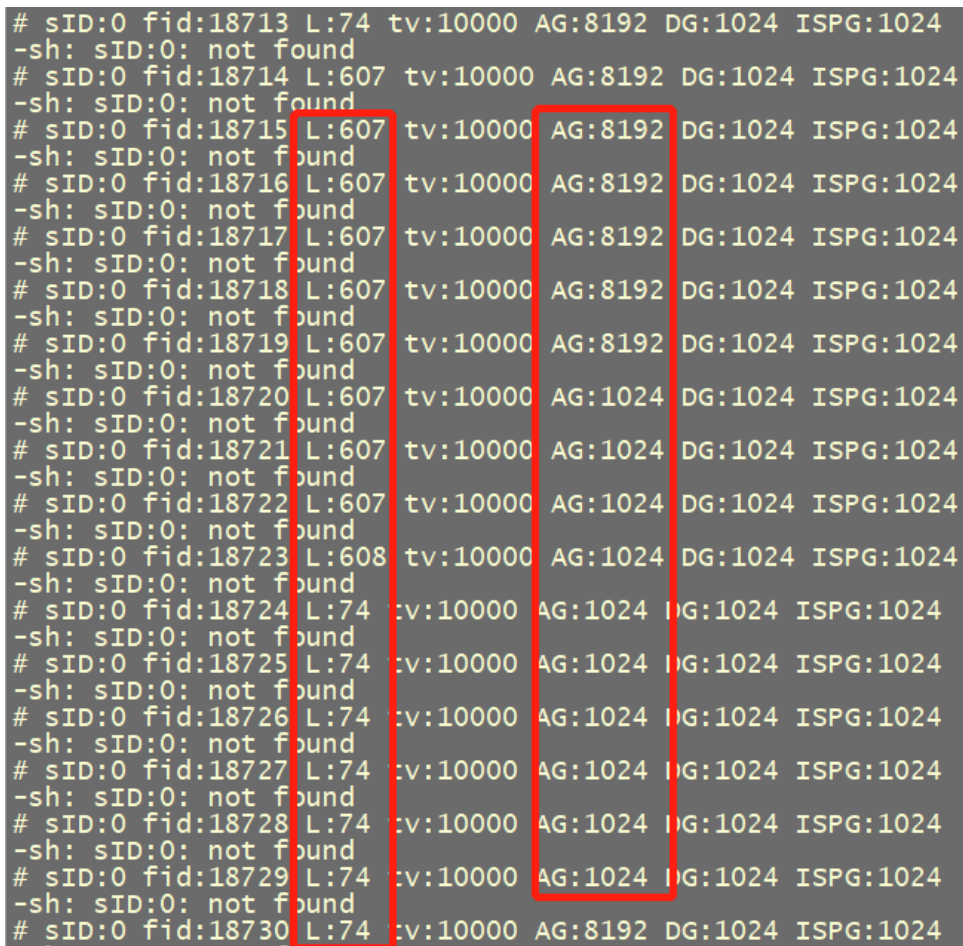
Wdr 模式下输入 CMD

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

下图是测试结果, linear mode 下 again 从 1024 增加到 8192, luma 值从 74 变到 607, 基本符合 8 倍的关系。

Wdr mode 下 again 从 1024 改变成 2048, luma 从 51 变成 100, 基本符合 2 倍的关系。

所以下图的 luma 和 again 能够满足线性关系。



```
# sID:0 fid:18713 L:74 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18714 L:607 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18715 L:607 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18716 L:607 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18717 L:607 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18718 L:607 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18719 L:607 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18720 L:607 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18721 L:607 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18722 L:607 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18723 L:608 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18724 L:74 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18725 L:74 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18726 L:74 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18727 L:74 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18728 L:74 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18729 L:74 tv:10000 AG:1024 DG:1024 ISPG:1024
-sh: sID:0: not found
# sID:0 fid:18730 L:74 tv:10000 AG:8192 DG:1024 ISPG:1024
-sh: sID:0: not found
```



```

sID:0 fid:36645 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:1024 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:0 DG:0 FL:3600

sID:0 fid:36646 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:1024 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:0 DG:0 FL:3600

sID:0 fid:36647 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:1024 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:0 DG:0 FL:3600

sID:0 fid:36648 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:1024 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:0 DG:0 FL:3600

sID:0 fid:36649 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:1024 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:0 DG:0 FL:3600

sID:0 fid:36650 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36651 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36652 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36653 L_L:51 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36654 L_L:100 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36655 L_L:100 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36656 L_L:100 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

sID:0 fid:36657 L_L:100 S_L:0 L_T:33333 S_T:12 L_EL:2999 S_EL:1 AG:2048 DG:1024 IG:1024
Sensor LE_EL:2999 SE_EL:1 AG:64 DG:0 FL:3600

```

10.4 进阶验证

如果想要精确验证曝光线性度，需要用到 CMD

```

5
8 SID FID startExpTime endExptime

```

```

[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, gain 0: gain 1, startindex, endindex)
[sensor_ae_test]-1105: 255 0 0 0: exit
[sensor_ae_test]-1106: Item/sID/para1/para2/para3

```

可以测试连续的曝光线性度，表示从 startExpTime 到 endExpTime 每隔%5 的精度进行递增。

SID 表示 sensorID, FID 表示 frameID, 0 表示长帧，1 表示短帧。

如果想要精确验证增益线性度，需要用到 CMD

```

5
7 SID FID time StartISO EndISO

```

可以测试连续的增益线性度，表示从 StartISO 到 EndISO 遍历 gaintable 进行递增。这里 ISO 100 表示 1x，即 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_SetWBManualRatio(sID, ratio), ratio: 1 356, 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: exit
[sensor_ae_test]-1106: Item/sID/para1/para2/para3

```

10.5 Response Frame 验证

不同的 sensor 参数生效时间不一样，比如有些 sensor 要过 5 frame 时间后才能生效，有些只要过 4 frame 或者 3 frame 就能生效。

即使同一颗 sensor，不同的寄存器设定，生效时间也有可能不一样，因此需要验证 AE 相关的寄存器的反应帧。

运行 sensor_test，linear 模式下输入 CMD

```

5
2 0 71 0 0

```

这样测出 shutter 设定后要过多少 frame 后才会生效。

下图看出 shutter 从 33333 改变成 16666 后，要过 4 个 frame 后 Luma 值才发生改变。

所以曝光的 ResponseFrame 为 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

```

输入 CMD

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

这样测出 gain 设定后要过多少 frame 后才会生效。

下图看出 again 从 1024 改变成 2048 后，要过 4 个 frame 后 Luma 值才发生改变。

所以增益的 ResponseFrame 为 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
```

测试完曝光、增益的 ResponseFrame 后，ResponseFrame 填入 xxx_cmos.c 中的 cmos_get_ae_default 函数中，如下图：

```
126 // default:
127 // case MDR_MODE_NONE: /*linear-mode*/
128 //     pstAeSnsDft->f32Fps = g_astSC4210_mode[SC4210_MODE_1440P30].f32MaxFps;
129 //     pstAeSnsDft->au8HistThresh[0] = 0x0;
130 //     pstAeSnsDft->au8HistThresh[1] = 0x28;
131 //     pstAeSnsDft->au8HistThresh[2] = 0x60;
132 //     pstAeSnsDft->au8HistThresh[3] = 0x80;
133 //
134 //     pstAeSnsDft->u32MaxAgain = g_astSC4210_mode[SC4210_MODE_1440P30].stAgain[0].u16Max;
135 //     pstAeSnsDft->u32MinAgain = g_astSC4210_mode[SC4210_MODE_1440P30].stAgain[0].u16Min;
136 //     pstAeSnsDft->u32MaxAgainTarget = pstAeSnsDft->u32MaxAgain;
137 //     pstAeSnsDft->u32MinAgainTarget = pstAeSnsDft->u32MinAgain;
138 //
139 //     pstAeSnsDft->u32MaxDgain = g_astSC4210_mode[SC4210_MODE_1440P30].stDgain[0].u16Max;
140 //     pstAeSnsDft->u32MinDgain = g_astSC4210_mode[SC4210_MODE_1440P30].stDgain[0].u16Min;
141 //     pstAeSnsDft->u32MaxDgainTarget = pstAeSnsDft->u32MaxDgain;
142 //     pstAeSnsDft->u32MinDgainTarget = pstAeSnsDft->u32MinDgain;
143 //
144 //     pstAeSnsDft->u8AeCompensation = 40;
145 //     pstAeSnsDft->u32InitAeSpeed = 64;
146 //     pstAeSnsDft->u32InitAeTolerance = 5;
147 //     pstAeSnsDft->u32AeResponseFrame = 4;
148 //     pstAeSnsDft->u32AeResponseFrame = 4;
```

10.6 曝光增益同步生效验证

有时候不是所有的 sensor 都是 gain、shutter 同时生效，例如可能会出现 gain 的 ResponseFrame 是 4，shutter 的 ResponseFrame 是 3，需要进行曝光增益同步机制验证。

运行 sensor_test，linear 模式下输入 CMD

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

这个 CMD 表示同时改变 shutter/gain，AE 的统计值要过多久才会发生变化。

下面的图可以看到同时改变 shutter 和 again 都是在 4frame 后同步生效。

```

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

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

```

如果同时加大增益、曝光后，AE 的统计值出现跳变，比如像下面这样的结果，则说明增益、曝光没有同步生效。

```

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

```

此时需要对 gain 或者 shutter 进行 delay 让其同步生效，修改 xxx_cmos.c 中的 cmos_get_sns_regs_info 函数，修改 register 的 delay 设定。

比如下图就是对 gain 进行 delay 了 2 frame，表示要相对于其他的 register 设定要晚 2 帧。

```
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 FPS 可控性验证

用 sensor_test 运行后输入 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: exit  
[sensor_ae_test]-1106: Item/sID/para1/para2/para3
```

默认运行起来后 fps 是 25fps，可以查看 vi_dbg 查看 sensor 输出的 fps 是多少。

Linux: cat /proc/cvitek/vi_dbg

Alios: 运行 sensor_test 后输入 7 -> 1

```
# cat /proc/cvitek/vi_dbg
[VI Info]
VIOutImgWidth           :1920
VIOutImgHeight          :1080
VIISP_TopStatus         :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 常见问题

11.1 Proc 讯息解读

```

-----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
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
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

```

Linux: # cat /proc/mipi-rx

Alios: # proc_mipi_rx

- Combo DEV ATTR 主要是对 sensor 的接口配置信息：
- Devno: sensor 编号, 0 表示 sensor0, 1 表示 sensor1, 目前最大只支持 2 路 sensor 同时输入
- WorkMode: 表示哪种接口类型 (mipi/sublvds/ HISPI /BT656…)
- DataType: 表示 sensor 数据格式 (raw8/raw10/raw12/ YUV422_8BIT…)
- WDRMode:wdr 模式 (none 表示非 wdr, 常见 wdr mode:VC, DT, Manual)
- LinkId: lane 线序配置
- PN swap:PN 反转, 如果有反转, 要将该 lane 配置成 1
- SyncMode/DataEndian/SyncCodeEnddian: 对于 mipi 接口不支持因此无需配置, 对于 sublvds、hispi 则需要配置

- MIPI INFO 主要是 mipi-rx 解析到的信息：
- EccErr、CrcErr、HdrErr、WcErr: 如果不为 0 表示有出现 Ecc,crc,wc 等校验相关 err, 需要检查 lane mapping 的正确性、mipi 时序、lane 硬件电路等。
- Fifofull: 如果不为 0 表示 mac 速率太慢, 需要提升 mac clk.
- Decode: 表示解析出的数据 l 类型 (Raw12/raw10/raw8/YUV422…)

- PhySical:D0-D4 表示 lane 总线上的数据, 进入 hi speed state 后, D0-D4 会有数据跳变。
- Digital: D0-D4 显示进入 hi speed state 后的每条 data lane 状态。CK_HS、CK_ULPS、CK_ERR、Deskew 表示 clk lane 状态, 正常情况下 CK_HS=1, 其余的为 0, 但是非 continues clk 的情况下也会跳出 CK_HS=1,CK_STOP=1。

11.2 Sensor 相关 log 开启

Linux 操作

开启 cif drv log:

```
echo "module cvi_mipi_rx +p" > /sys/kernel/debug/dynamic_debug/control  
  
dmesg -n 8
```

开 syslog 打印:

输出到串口屏幕

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

或者输出到文件

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

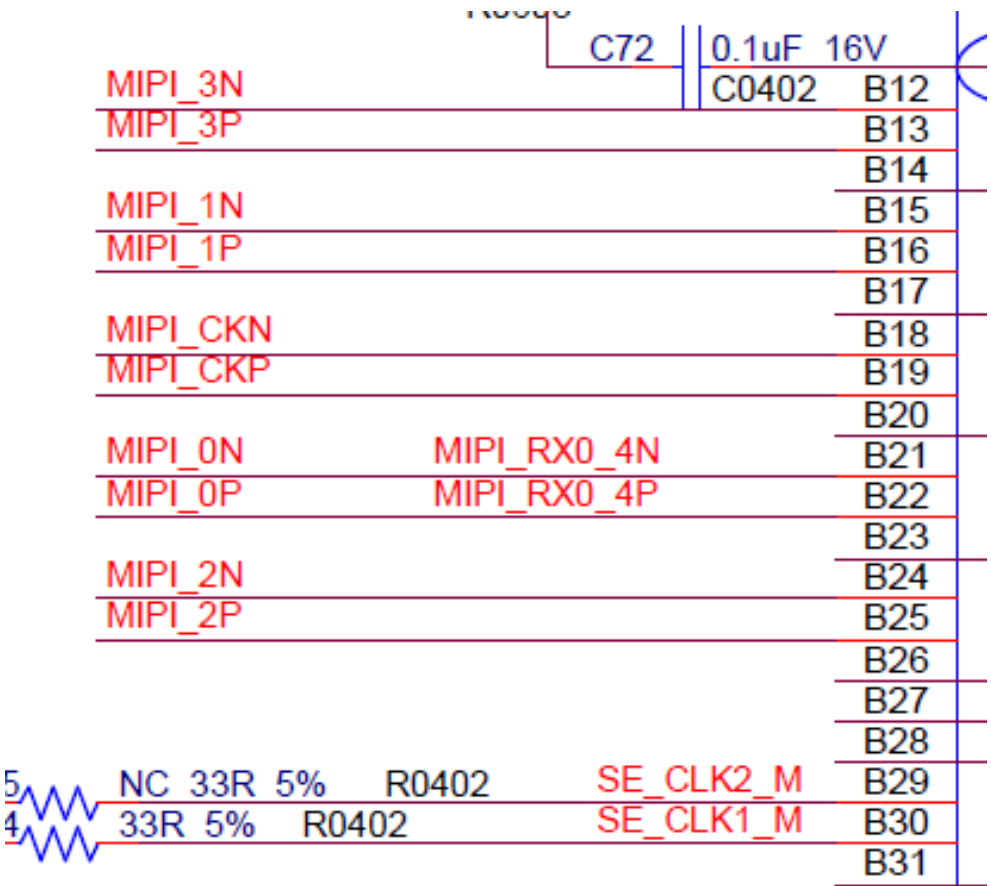
11.3 如何配置 lane 线序

注意我们要配置的 lane id 要以 sensor 为参照物来配置, lane_id 数组的索引号表示的是 Sensor 的 Lane ID, 索引号 0 表示 sensor clock, 索引号 1~4 表示 sensor lane 0~3。

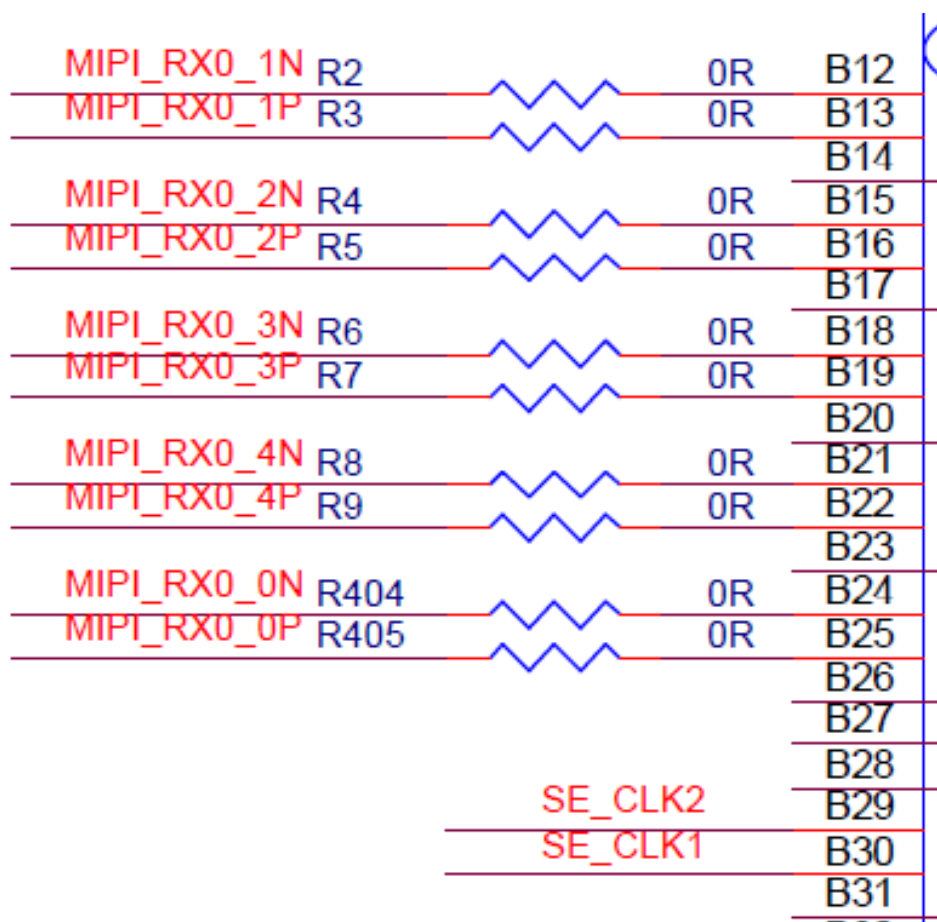
land_id 数组的值表示的是 soc 的 MIPI-Rx 的 Lane ID, 0 表示 MIPIRX1_PAD0, 1 表示 MIPIRX1_PAD1, 未使用的 lane 将 lane_id 配置成-1。

假设 sensor 和 soc 的 lane 接线如下图所示, 对应的 lane id 配置就是 {3, 4, 2, 0, 1}.

sensor



SOC



SENSOR 管脚	IPI Lane 管脚
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)

```
//索引 sensor_clk, sensor_lane0, sensor_lane1, sensor_lane2, sensor_lane3
//
land_id={MIPIRX0_PAD3 MIPIRX0_PAD4 MIPIRX0_PAD2 MIPIRX0_PAD0 MIPIRX0_PAD1}
```

11.4 如何选择 MAC 频率

MAC 表示 isp 接收 sensor 数据的频率,

公式 $MAC_Freq * pix_width = lane_num * MIPI_Freq * 2$ 。

- MAC_Freq: VI MAC 的工作频率
- pixel_width: 像素位宽

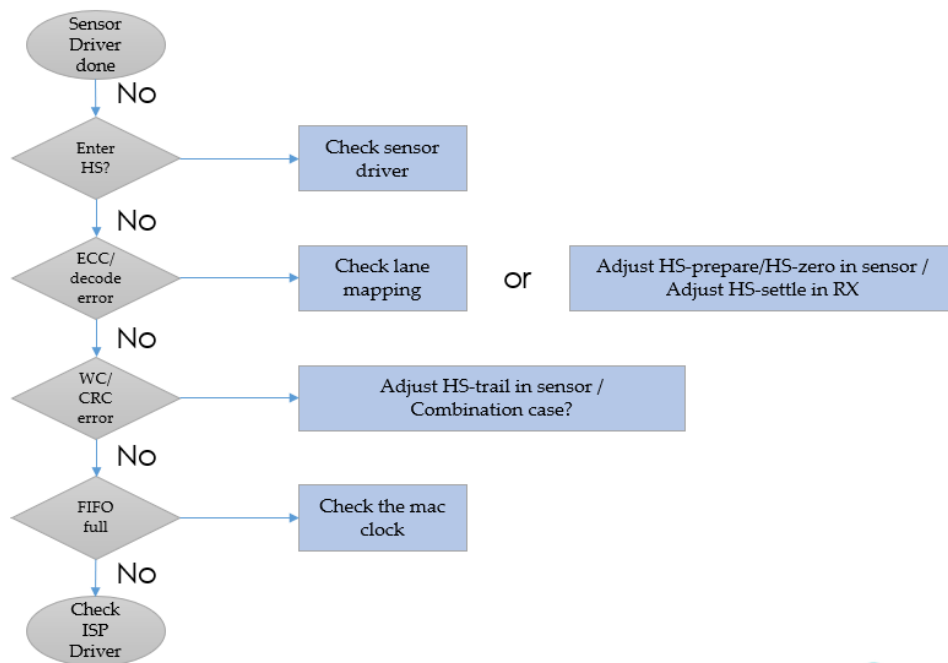
- lane_num: MIPI lane 个数
- MIPI_Freq: 每条 lane 的工作频率

假设 MAC freq 为 400M, pixel_width = 12, lane_num = 4, 可支持最快 MIPI_Freq = $400 * 12 / (4 * 2) = 600\text{MHz}$ 。

其中 MIPI_Freq 是表示 phy_clk, 值是 bps/2。比如 sony imx335 规格是 1188Mbps 每 lane, phy_clk = $1188/2=594\text{Mhz}$ 。

反之, 如果 sensor 给出了 data rate, 我们要能算出合适的 mac freq.

11.5 错误检查流程



11.5.1 I2C Write Fail

- sensor i2c 属性确认
- 检查 I2C bus id
- 检查 I2C slave addr
- 检查 sensor 寄存器的 addr/data 位宽 (8bit or 16bit)

位宽配置不正确一般会报 time out 错误

```

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 00

```

- 检查 hardware 是否正常
- 确认配置中的 rst, pwrn, mclk 引脚配置正确

Linux 操作命令:

```

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

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

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

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

```

Alios 操作命令:

执行 `sensor test` 后输入 255 退出

- 用 `i2cdetect -y -r N` 命令测试 i2c 能否检测到。N 表示 sensor 对应的 i2c 端口 (Alios 使用 `iic detect N` 命令)
- 检查 power on 时序是否满足 spec 要求 (示波器测量 mclk, i2c)

11.5.2 Decode err

cat /proc/mipi-rx (Alios 执行 `proc_mipi_rx`), 查看 proc 讯息, 先看是否有进入 hs-state, 当 sensor 驱动起来后会从 low power state 进入 high speed state。

如下图, 如果 mipi-rx 的 D0-D4 有数据, 并且一直跳变, 则说明有进入 hs-state。

- 确认 i2c 通路 (i2cdetect 能够扫出 sensor 地址)
- 确认 lane 线序正确
 - a. 如果 proc 中的 data lane 没有数据跳变, 并且伴随 CK_HS 为 0, 表示 clk lane 没找对 (请确认 clk lane)

- b. 如果 proc 中的 data lane 有数据跳变, 伴随 CK_HS 为 1, 表示 clk lane 找对了, 已进入 hs 模式, 这时如果出现 ecc、crc 等错误, 这时表示 data lane 没配置对 (请确认 data lane)
- 确认时序
 - a. 若前两点都确认正确, 但还 CK_HS = 0, data lane 没有数据跳变, 可能是时序上未能满足进入 hs 的条件, 此时可以调整加大 hs-zero, hs-trail 数值, 拉长 detect period。
 - b. 若前两点都确认正确, CK_HS = 1, data lane 有数据跳变, 但是还是有 ecc、crc 等 err, 则可能是 Hs-settle 的设定太大或是太小, 压到后面的 data。
- 确认 hw 是否损坏

11.5.3 ECC err

- 检查 lane Id mapping
- 检查 sensor tx hs-zero/hs-prepare

hs-zero、hs-prepare 要从 sensor spec 中确定是多少或者直接问 sensor 厂商, 一般不建议调整。

- 检查 mipi-rx hs-settle

当 hs-settle 的时间太长会压到 data 中的 “sync code”, 就会出现 “sync code” 解析不到, 导致 ecc err。

调整 hs-settle 可以直接修改 xxx_cmos_param.h 如下, 填入正确的 hs_settle。

```

1: struct combo_dev_attr_s sc4210_rx_attr = {
2: » .input_mode = INPUT_MODE_MIPI,
3: » .mac_clk = RX_MAC_CLK_400M,
4: » .mipi_attr = {
5: » » .raw_data_type = RAW_DATA_12BIT,
6: » » .lane_id = {0, 4, 3, 2, 1},
7: » » .wdr_mode = CVT_MIPI_WDR_MODE_VC,
8: » » .dphy = {
9: » » » .enable = 1,
10: » » » .hs_settle = 8,
11: » » },
12: » },

```

Linux 操作: 调整 hs-settle 也可以直接 ctrl+z 跳出, 用 devmem 命令修改寄存器 0x0300b048 的 bit[23:16] 数值, 调整完后输入 fg 跳回程序。

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

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

11.5.4 CRC err/Word count err

调整 sensor tx hs-trail, 如果 hs-trail 拉的太快, 有可能会压到后面的 data, 导致数据丢失, 从而出现 crc err 和 wc err, 需要调整 sensor 的 hs-trail 寄存器设定。

```
-----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 Deskw
          hs_err hs_err hs_hst hs_hst 1 0 0 0 start
```

11.5.5 vi_select timeout

- cat /proc/mipi-rx 显示是否有 i2c、decode、ecc、crc、wc 等 err.

如果前面 4 个步骤都已经确认无误后,

cat /proc/cvitek/vi_dbg 检查是否出现 WidthGTCnt、WidthLSCnt、HeightGTCnt、HeightLSCnt, 如果有这类报错表示 sensor init setting 中的 crop size 和给到 isp 的设定不一致, 请对照 sensor spec 确认修改。

- 检查 MAC clock 是否太低, 如果 mac clock 太低, 会导致 isp 处理速度太慢出现 fifo full, 也会导致 timeout。

12 颜色、去噪等矫正

请参考《图像质量调试工具使用指南 _v1.1.1》

13 调试工具

Linux:

sensor 开发后调试工具使用 `sensor_test`,

sensor 配置文件为 `/mnt/data/sensor_cfg.ini`。

在 `middleware` 目录通过命令 `git apply sensor_test.patch` 添加补丁, 编译后生成 `sensor_test` 来使用。

Alios:

`sensor_test` 的程序默认放在 `peripherals_test` 这个 solution 下面, 编译这个 solution 即可, 源码路径 `solutions/peripherals_test/sensor_test/sensor_test.c`

13.1 基础功能

`sensor_test` 默认有下图所示 5 种功能:

1. dump sensor raw 图
2. dump sensor yuv 图
3. 设置 sensor 输出图像 flip/mirror
4. 若 sensor 驱动支持 linear 和 wdr 模式, 可以使用该选项进行 sensor mode 切换
5. AE 调试功能

```
[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

参考[Dump RAW](#)

13.3 Dump YUV

参考[Dump YUV](#)

13.4 Set flip/mirror

提供 sensor 端镜像/翻转功能。

运行 `sensor_test`，输入 3 选择” set chn flip/mirror”，根据提示 `chn(0~1)`: 输入 `dev` (0 表示 vi pipe0, 控制第 0 路图像, 1 表示 vi pipe1) 后续提醒控制开关 `flip/mirror`。

注意：功能执行后，需要确认 `dump yuv` 图的方向和颜色符合预期。

13.5 WDR 和 Linear 切换

提供 sensor 端宽动态模式与线性模式切换功能。

运行 `sensor_test`，输入 4 选择” linear hdr switch”，根据提醒” Please select sensor input mode (0:linear/1:wdr):” 输入 0 为 linear; 1 为 WDR;

注意： 1、此功能需 sensor 支持 linear 和 wdr 两种模式

2、不同 sensor 配置需在 `sensor_test.c` 中修改对应的 sensor 配置，具体如下图

```
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_WDR2T01;
        g_stIniCfg.enWDRMode = WDR_MODE_2T01_LINE;
        // Reset slave sensor initial config to wdr setting.
        g_stIniCfg.enSns2Type = SONY_IMX327_SLAVE_MIPI_2M_30FPS_12BIT_WDR2T01;
        g_stIniCfg.enSns2WDRMode = WDR_MODE_2T01_LINE;
    }
}
```

13.6 AE 相关验证

参考AE 相关验证