



RTOS CSI 开发指南

版本号: 1.0
发布日期: 2020-10-22

版本历史

版本号	日期	制/修订人	内容描述
0.1	2019.09.06	SWC	1. 初版
1.0	2020.08.27	AWA0916	1. 适配 RTOS 2. 支持 sysconfig



目 录

1 前言	1
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
2 模块介绍	2
2.1 模块功能介绍	2
2.2 相关术语介绍	2
2.3 模块配置介绍	2
2.3.1 menuconfig 选项配置	2
2.3.2 VIN 模块配置	3
2.3.2.1 模块寄存器、中断号、个数以及 GPIO 配置	3
2.3.2.2 csic top clk 与 isp clk	6
2.3.2.3 csic master clk 与 pin	7
2.3.2.4 sunxi_vinc 配置	7
2.3.2.5 sensor 配置	10
2.4 源码结构介绍	13
3 Sensor 驱动开发	14
3.1 SENSOR_NAME	14
3.2 Register list 填充	14
3.3 sensor_win_sizes 填充	15
3.4 sensor_formats 填充	15
3.5 sensor 接口实现	16
3.6 sensor 测试	16
4 接口描述	18
4.1 VIDIOC_QUERYCAP	19
4.2 VIDIOC_ENUM_INPUT	19
4.3 VIDIOC_S_INPUT	20
4.4 VIDIOC_G_INPUT	20
4.5 VIDIOC_S_PARM	20
4.6 VIDIOC_G_PARM	21
4.7 VIDIOC_ENUM_FMT	22
4.8 VIDIOC_TRY_FMT	22
4.9 VIDIOC_S_FMT	23
4.10 VIDIOC_G_FMT	24
4.11 VIDIOC_OVERLAY	24
4.12 VIDIOC_REQBUFS	24
4.13 VIDIOC_QUERYBUF	25
4.14 VIDIOC_DQBUF	26
4.15 VIDIOC_QBUF	26

4.16 VIDIOC_STREAMON	27
4.17 VIDIOC_STREAMOFF	27
4.18 VIDIOC_QUERYCTRL	27
4.19 VIDIOC_S_CTRL	28
4.20 VIDIOC_G_CTRL	28
4.21 VIDIOC_ENUM_FRAMESIZES	29
4.22 VIDIOC_ENUM_FRAMEINTERVALS	30



1 前言

1.1 文档简介

CSI 模块在 RTOS 平台上的驱动为 SUNXI-VIN，本文将介绍 VIN (Video Input) 驱动的设计结构、流程、API 接口。主要包含接口部分 (CSI/MIPI) 和算法处理部分 (ISP/VIPP) 部分。

1.2 目标读者

硬件底层设计人员，驱动编写、维护人员，应用开发人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
V833	Melis	ekernel/subsys/.../sunxi-vin/*

2 模块介绍

2.1 模块功能介绍

Video Input 主要由接口部分 (CSI/MIPI) 和图像处理单元 (ISP/VIPP) 组成。

CSI/MIPI 部分主要实现视频数据的捕捉。

ISP 实现 sensor raw data 数据的处理, 包括 lens 补偿、去坏点、gain、gamma、de-mosaic、de-noise、color matrix 等以及一些 3A 的统计。

VIPP 能对将图进行缩小、和打水印处理。VIPP 支持 bayer raw data 经过 ISP 处理后再缩小, 也支持对一般的 YUV 格式的 sensor 图像直接缩小。

说明

对于 V833 来说, 其 CSIC IP 包括 2 个 Input Parser, 1 个 ISP, 4 个 VIPP, 4 个 DMA。

2.2 相关术语介绍

- ISP: Image Signal Processor
- VIPP: Video Input Post Processor
- MIPI: Mobile Industry Processor Interface
- CCI: camera control interface
- MCLK: Master clock (From AP to camera)
- PCLK: Pixel clock (From camera to AP, Sampling clock for data-bus)
- YUV: Color presentation (Y for luminance, U&V for chrominance)
- CSIC: CMOS Sensor Interface Controller(sunxi image/video input control module)

2.3 模块配置介绍

2.3.1 menuconfig 选项配置

VIN 驱动会使用到了 ccmu、gpio、twi、regulator 驱动, 以及 media controller 框架等, 其 menuconfig 的配置如下:

```
Kernel Setup --->
  Subsystem support --->
    avframework --->
      [*] Multimedia support --->
        [*] Cameras/video grabbers support
        [*] Media Controller API
        [*] Video4Linux support
        [*] V4L2 sub-device userspace API
        [*] Enable the V4L2 core and API
        [*] V4L platform devices --->
          [*] sunxi video input (camera csi/mipi isp vipp)driver
          [*] v4l2 new driver for SUNXI
              select sensor (use sensor imx386) --->
          [*] ISP WDR module
```

其中 select sensor 选项请配置为对应的 sensor。

2.3.2 VIN 模块配置

VIN 中各个子模块以及 sensor 在 sun8iw19p1_vin_cfg.h 以及 vin_config_sun8iw19p1_real.h 文件中
进行配置。

⚠ 警告

如果开启了 CONFIG_FEXCONFIG，需要在 sys_config.fex 文件对 VIN 模块进行配置。sys_config
路径如下：

```
source/projects/defconfig/{PLATFORM}/aw_{PLATFORM}/sys_config.fex
```

下面以 V833 为例对主要配置进行说明。

2.3.2.1 模块寄存器、中断号、个数以及 GPIO 配置

V833 CSIC 模块寄存器、中断号与个数如下，需要根据 datasheet 来进行配置。

```
#define CSI_CCU_REGS_BASE          0x06600000
#define CSI_TOP_REGS_BASE         0x06600800

#define CSI0_REGS_BASE            0x06601000
#define CSI1_REGS_BASE            0x06602000

#define MIPI_REGS_BASE            0x0660C000

#define ISP_REGS_BASE              0x02100000

#define VIPP0_REGS_BASE           0x02104000
```

```

#define VIPP1_REGS_BASE                0x02104400
#define VIPP2_REGS_BASE                0x02104800
#define VIPP3_REGS_BASE                0x02104c00

#define CSI_DMA0_REG_BASE              0x06609000
#define CSI_DMA1_REG_BASE              0x06609200
#define CSI_DMA2_REG_BASE              0x06609400
#define CSI_DMA3_REG_BASE              0x06609600

#define GPIO_REGS_VBASE                0x0300b000

#define SUNXI_GIC_START 32
#define SUNXI_IRQ_CSIC_DMA0           (SUNXI_GIC_START + 74)
#define SUNXI_IRQ_CSIC_DMA1           (SUNXI_GIC_START + 75)
#define SUNXI_IRQ_CSIC_DMA2           (SUNXI_GIC_START + 76)
#define SUNXI_IRQ_CSIC_DMA3           (SUNXI_GIC_START + 77)
#define SUNXI_IRQ_ISP0                 (SUNXI_GIC_START + 33)
#define SUNXI_IRQ_CSI_TOP_PKT         (SUNXI_GIC_START + 92)

#define VIN_MAX_DEV                    4
#define VIN_MAX_CSI                    2
#define VIN_MAX_CCI                    2
#define VIN_MAX_TDM                    0
#define VIN_MAX_MIPI                   1
#define VIN_MAX_ISP                     1
#define VIN_MAX_SCALER                 4

#define MAX_CH_NUM                     4

```

V833 CSIC 支持一路 serial interface (即 MIPI, csi0) 与一路 parallel interface (csi1)。MIPI 接口 pin 一般是独享不需要配置，parallel interface 的 pin 通常是 gpio，需要进行配置才能使用，V833 CSIC 模块 parallel interface 的 pin 配置如下。

```

/*
"csil_pck", "csil_hsync", "csil_vsync",
"csil_d0", "csil_d1", "csil_d2", "csil_d3",
"csil_d4", "csil_d5", "csil_d6", "csil_d7",
"csil_d8", "csil_d9", "csil_d10", "csil_d11",
"csil_d12", "csil_d13", "csil_d14", "csil_d15";
*/
int vind_csi_parallel_pins[VIN_MAX_CSI][19] = {
#ifdef CONFIG_FEXCONFIG
    GPIO_INDEX_INVALID
#else
    {
        GPIO_INDEX_INVALID
    },
    {
        GPIOE(0),
        GPIOE(2),
        GPIOE(3),
        GPIOE(4),
        GPIOE(5),
        GPIOE(6),
        GPIOE(7),
        GPIOE(8),

```



```

GPIOE(9),
GPIOE(10),
GPIOE(11),
GPIOE(12),
GPIOE(13),
GPIOE(14),
GPIOE(15),
GPIOE(18),
GPIOE(19),
GPIOE(20),
GPIOE(21)
}
#endif
};

```

⚠ 警告

如果使用 `sys_config.fex`，需要加入如下配置。

在定义 `CONFIG_FEXCONFIG` 的情况下，需要在 `sys_config.fex` 中配置 `parallel interface` 相关的 `gpio`。

```

[vind0/csil]
csil_used          = 0
csil_pck           = port:PE00<2><default><default><default>
csil_hsync         = port:PE02<2><default><default><default>
csil_vsync         = port:PE03<2><default><default><default>
csil_d0            = port:PE04<2><default><default><default>
csil_d1            = port:PE05<2><default><default><default>
csil_d2            = port:PE06<2><default><default><default>
csil_d3            = port:PE07<2><default><default><default>
csil_d4            = port:PE08<2><default><default><default>
csil_d5            = port:PE09<2><default><default><default>
csil_d6            = port:PE10<2><default><default><default>
csil_d7            = port:PE11<2><default><default><default>
csil_d8            = port:PE12<2><default><default><default>
csil_d9            = port:PE13<2><default><default><default>
csil_d10           = port:PE14<2><default><default><default>
csil_d11           = port:PE15<2><default><default><default>
csil_d12           = port:PE18<2><default><default><default>
csil_d13           = port:PE19<2><default><default><default>
csil_d14           = port:PE20<2><default><default><default>
csil_d15           = port:PE21<2><default><default><default>

```

`sys_config` 配置说明：

- `csi(x)_used`: 使能开关，0-disable，1-enable。该字段暂未生效。
- 其他 `gpio` 需要基于 `datasheet` 来进行配置。

2.3.2.2 csic top clk 与 isp clk

这部分涉及 vin top clk 与 isp clk 的相关配置。需要改动的主要是这两个时钟的频率，其他配置通常不需要修改。

```
struct vin_clk_info vind_default_clk[VIN_MAX_CLK] = {
    [VIN_TOP_CLK] = {
        .clock = HAL_CLK_PERIPH_CSI_TOP,
#ifdef CONFIG_FEXCONFIG
#ifdef CONFIG_VIN_SENSOR_imx386
        .frequency = 336000000,
#elif defined CONFIG_VIN_SENSOR_C2398
        .frequency = 300000000,
#endif
#endif
    },
    [VIN_TOP_CLK_SRC] = {
        .clock = HAL_CLK_PLL_CSI,
    },
};

struct vin_clk_info vind_default_isp_clk[VIN_ISP_MAX_CLK] = {
    [VIN_ISP_CLK] = {
        .clock = HAL_CLK_PERIPH_ISP,
#ifdef CONFIG_FEXCONFIG
        .frequency = 300000000,
#endif
    },
    [VIN_ISP_CLK_SRC] = {
        .clock = HAL_CLK_PLL_PERI1,
    },
};
```

结构体成员说明：

- csic top clk frequency: vin 模块时钟, 实际使用可根据 sensor 的帧率和分辨率设置。
- isp clk frequency: isp 时钟频率。

警告

如果使用 sys_config.fex, 需要加入如下配置。

在定义 CONFIG_FEXCONFIG 的情况下, 需要在 sys_config.fex 中配置 top clk 与 isp clk 的频率。

```
[vind0]
vind0_used      = 1
vind0_clk       = 336000000
vind0_isp       = 300000000
```

2.3.2.3 csic master clk 与 pin

这部分涉及到 master clk 与 pin 的配置，通常不需要改动。

```
struct vin_mclk_info vind_default_mclk[VIN_MAX_CCI] = {
    {
        .mclk = HAL_CLK_PERIPH_CSI_MASTER0,
#ifdef CONFIG_CSI_PLL_CLK_SPREAD_SPECTRUM
        .clk_24m = HAL_CLK_PLL_CSI,
#else
        .clk_24m = HAL_CLK_SRC_HOSC24M,
#endif
        .clk_pll = HAL_CLK_PLL_CSI,
        .pin = {
            [0] = {0, GPIOI(0), 2, 0, 2},
            [1] = {0, GPIOI(0), 7, 0, 2},
        },
    },
    {
        .mclk = HAL_CLK_PERIPH_CSI_MASTER1,
#ifdef CONFIG_CSI_PLL_CLK_SPREAD_SPECTRUM
        .clk_24m = HAL_CLK_PLL_CSI,
#else
        .clk_24m = HAL_CLK_SRC_HOSC24M,
#endif
        .clk_pll = HAL_CLK_PLL_CSI,
        .pin = {
            [0] = {0, GPIOE(1), 2, 0, 2},
            [1] = {0, GPIOE(1), 7, 0, 2},
        },
    },
};
```

2.3.2.4 sunxi_vinc 配置

用于 video 节点的注册和配置各个子模块的使用情况。需要填充如下结构体。

```
struct vin_core sunxi_vinc[VIN_MAX_DEV] = {
#ifdef CONFIG_FEXCONFIG
    0
#else
    [0] = {
        .id = 0,
        .rear_sensor = 0,
        .front_sensor = 0,
        .csi_sel = 0,
        .mipi_sel = 0,
        .isp_sel = 0,
    },
    [1] = {
        .id = 1,
    },
};
```

```
        .rear_sensor = 0,
        .front_sensor = 0,
        .csi_sel = 0,
        .mipi_sel = 0,
        .isp_sel = 0,
    },
    [2] = {
        .id = 2,
        .rear_sensor = 0,
        .front_sensor = 0,
        .csi_sel = 0,
        .mipi_sel = 0,
        .isp_sel = 0,
    },
    [3] = {
        .id = 3,
        .rear_sensor = 0,
        .front_sensor = 0,
        .csi_sel = 0,
        .mipi_sel = 0,
        .isp_sel = 0,
    }
}
#endif /*CONFIG_FEXCONFIG*/
};
```

相关结构体成员说明：

- id: index 号。
- rear_sensor: 表示该 pipeline 上使用的后置 sensor 的 id。
- front_sensor: 表示该 pipeline 上使用的前置 sensor 的 id。
- csi_sel: 表示该 pipeline 上 parser 的 id，必须配置，且为有效 id。
- mipi_sel: 表示该 pipeline 上 mipi(sublvds/hispi) 的 id，不使用时配置为 0xff。
- isp_sel: 表示该 pipeline 上 isp 的 id，必须配置，当 isp 为空时，这个 isp 只是表示路由不做 isp 的效果处理。

⚠ 警告

如果使用 `sys_config.fex`，需要加入如下配置。

在定义 `CONFIG_FEXCONFIG` 的情况下，上述结构体的内容为空，全部需要在 `sys_config.fex` 中进行配置。

```
[vind0/vinc0]
vinc0_used      = 1
vinc0_csi_sel   = 0
vinc0_mipi_sel  = 0
vinc0_isp_sel   = 0
vinc0_isp_tx_ch = 0
vinc0_rear_sensor_sel = 0
vinc0_front_sensor_sel = 0
vinc0_sensor_list = 0
```

```
[vind0/vinc1]
vinc1_used      = 0
vinc1_csi_sel   = 0
vinc1_mipi_sel  = 0xff
vinc1_isp_sel   = 0
vinc1_isp_tx_ch = 0
vinc1_rear_sensor_sel = 0
vinc1_front_sensor_sel = 0
vinc1_sensor_list = 0

[vind0/vinc2]
vinc2_used      = 0
vinc2_csi_sel   = 0
vinc2_mipi_sel  = 0
vinc2_isp_sel   = 0
vinc2_isp_tx_ch = 0
vinc2_rear_sensor_sel = 0
vinc2_front_sensor_sel = 0
vinc2_sensor_list = 0

[vind0/vinc3]
vinc3_used      = 0
vinc3_csi_sel   = 0
vinc3_mipi_sel  = 0
vinc3_isp_sel   = 0
vinc3_isp_tx_ch = 0
vinc3_rear_sensor_sel = 0
vinc3_front_sensor_sel = 0
vinc3_sensor_list = 0
```

sys_config 配置说明如下：

- `vinc(x)_used`: `vipp` 的使能开关, 0-disable, 1-enable。该字段暂未生效。
- `vinc(x)_csi_sel`: 表示该 pipeline 上 parser 的 id, 必须配置, 且为有效 id。同上述 `vin_core` 结构体中的 `csi_sel` 成员。
- `vinc(x)_mipi_sel`: 表示该 pipeline 上 mipi(sublvds/hispi) 的 id, 不使用时配置为 0xff。同上述 `vin_core` 结构体中的 `mipi_sel` 成员。
- `vinc(x)_isp_sel`: 表示该 pipeline 上 isp 的 id, 必须配置, 当 isp 为空时, 这个 isp 只是表示路由不做 isp 的效果处理。同上述 `vin_core` 结构体中的 `isp_sel` 成员。
- `vinc(x)_isp_tx_ch`: 表示该 pipeline 上 isp 的 ch, 必须配置, 默认为 0。当 sensor 是 bt656 多通道或者 WDR 出 RAW 时, 该 ch 可以配置 0~3 的值。同上述 `vin_core` 结构体中的 `isp_tx_ch` 成员。
- `vinc(x)_rear_sensor_sel`: 表示该 pipeline 上使用的后置 sensor 的 id。同上述 `vin_core` 结构体中的 `rear_sensor` 成员。
- `vinc(x)_front_sensor_sel`: 表示该 pipeline 上使用的前置 sensor 的 id。同上述 `vin_core` 结构体中的 `front_sensor` 成员。
- `vinc(x)_sensor_list`: 表示是否使用 `sensor_list` 来时适配不同的模组, 1 表示使用, 0 表示不使用。同 `sensor_list` 结构体中的 `use_sensor_list` 成员。

2.3.2.5 sensor 配置

对 sensor 进行配置。这些节点的配置一般需要参考对应方案的原理图和 sensor 的 data sheet 来完成。需要填充如下结构体。

```
struct sensor_list sensors_default[VIN_MAX_DEV] = {
#ifdef CONFIG_FEXCONFIG
    {
        .power =
        {
            [IOVDD] = {NULL, AXP2101_ID_ALD02, 0, "iovdd"},
            [AVDD] = {NULL, AXP2101_ID_BLD02, 0, "avdd"},
            [DVDD] = {NULL, AXP2101_ID_DLD02, 0, "dvdd"},
            [AFVDD] = {NULL, AXP2101_ID_MAX, 0, ""},
            [FLVDD] = {NULL, AXP2101_ID_MAX, 0, ""},
            [CAMERAVDD] = {NULL, AXP2101_ID_MAX, 0, ""},
        },
    },
#else
    {
        .use_sensor_list = 0,
        .used = 0,
        .csi_sel = 0,
        .device_sel = 0,
        .mclk_id = 0,
        .sensor_bus_sel = 1,
        .sensor_bus_type = 0,
        .act_bus_sel = 0,
        .act_bus_type = 0,
        .act_separate = 0,
        .power_set = 0,
        .detect_num = 1,
        .sensor_pos = "rear",
        .valid_idx = 0,
        .power = {
            [IOVDD] = {NULL, AXP2101_ID_ALD02, 1800000, "iovdd"},
            [AVDD] = {NULL, AXP2101_ID_BLD02, 2800000, "avdd"},
            [DVDD] = {NULL, AXP2101_ID_DLD02, 1200000, "dvdd"},
            [AFVDD] = {NULL, AXP2101_ID_MAX, 0, ""},
            [FLVDD] = {NULL, AXP2101_ID_MAX, 0, ""},
            [CAMERAVDD] = {NULL, AXP2101_ID_MAX, 0, ""},
        },
        .gpio = {
            [POWER_EN] = {0, GPIO_INDEX_INVALID, 0, 0, 0},
            [PWDN] = {0, GPIOI(4), 1, 0, 1},
            [RESET] = {0, GPIOI(3), 1, 0, 1},
            [SM_HS] = {0, GPIO_INDEX_INVALID, 0, 0, 0},
            [SM_VS] = {0, GPIO_INDEX_INVALID, 0, 0, 0},
            [AF_PWDN] = {0, GPIO_INDEX_INVALID, 0, 0, 0},
            [FLASH_EN] = {0, GPIO_INDEX_INVALID, 0, 0, 0},
            [FLASH_MODE] = {0, GPIO_INDEX_INVALID, 0, 0, 0},
        },
        .inst = {
#ifdef CONFIG_VIN_SENSOR_imx386
            [0] = {
                .cam_name = "imx386_mipi",
            },
#endif
        },
    },
#endif
};
```

```
        .cam_addr = 0x20,  
        .cam_type = 1,  
        .is_isp_used = 1,  
        .is_bayer_raw = 1,  
        .vflip = 0,  
        .hflip = 0,  
        .act_addr = 0x0,  
        .act_name = "",  
        .isp_cfg_name = "",  
    },  
},  
#elif defined CONFIG_VIN_SENSOR_C2398  
    [0] = {  
        .cam_name = "C2398_mipi",  
        .cam_addr = 0x6c,  
        .cam_type = 1,  
        .is_isp_used = 1,  
        .is_bayer_raw = 1,  
        .vflip = 0,  
        .hflip = 0,  
        .act_addr = 0x0,  
        .act_name = "",  
        .isp_cfg_name = "",  
    },  
},  
#endif  
},  
#endif /*CONFIG_FEXCONFIG*/  
};
```

相关结构体成员说明:

- use_sensor_list: 表示是否使用 sensor_list 来时适配不同的模组,1 表示使用,0 表示不使用。
- used: 该字段暂未使用。
- csi_sel: 表示该 pipeline 上 parser 的 id, 必须配置, 且为有效 id。
- device_sel: 该字段暂未使用。
- mclk_id: sensor 所使用的 mclk 的 id。
- sensor_bus_sel: sensor 所使用的 bus 的 id。
- sensor_bus_type: 表示 sensor 所使用的 bus 类型 (0 - twi/i2c, 1 - cci, 2 - spi, 3 - gpio)。
- act_bus_sel: actuator 所使用的 bus 的 id。
- act_bus_type: actuator 所使用的 bus 类型 (twi, cci, spi 或 gpio)。
- act_separate: 该字段暂未使用。
- power_set: 该字段暂未使用。
- detect_num: sensor 的个数。
- sensor_pos: sensor 的位置, 前置还是后置。
- valid_idx: 可用的 sensor id, 与 inst 数组配合。
- power: 电源配置。需要参考对应方案的原理图和外设的 data sheet 来完成。
- gpio: gpio 配置。需要参考对应方案的原理图和外设的 data sheet 来完成。
- inst: 具体的 sensor 实例。

- cam_name: sensor 的名字。
- cam_addr: sensor 的 twi 地址。
- cam_type: sensor 的类型。0 - YUV; 1 - RAW。
- is_isp_used: 是否使用 ISP。0 - 不使用; 1 - 使用。
- is_bayer_raw: 是否是 bayer raw。0 - 不是; 1 - 是。
- vflip: flip in vertical direction. 0 - disable; 1 - enable。
- hflip: flip in horizontal direction. 0 - disable; 1 - enable。
- act_addr: actuator 的 twi 地址。
- act_name: actuator 的名字。
- isp_cfg_name: 该字段暂未使用。

⚠ 警告

如果使用 sys_config.fex, 需要加入如下配置。

在定义 CONFIG_FEXCONFIG 的情况下, 上述配置内容全部为空, 需要在 sys_config.fex 中进行配置。

```
[vind0/sensor0]
sensor0_used      = 1
sensor0_mname     = "imx386_mipi"
sensor0_twi_cci_spi = 0
sensor0_twi_cci_id = 1
sensor0_twi_addr  = 0x20
sensor0_cam_type  = 1
sensor0_mclk_id   = 0
sensor0_pos       = "rear"
sensor0_isp_used  = 1
sensor0_fmt       = 1
sensor0_vflip     = 0
sensor0_hflip     = 0
sensor0_iovdd_vol = 1800000
sensor0_avdd_vol  = 2800000
sensor0_dvdd_vol  = 1200000
sensor0_power_en  =
sensor0_pwdn      = port:PI04<1><0><1><0>
sensor0_reset     = port:PI03<1><0><1><0>
```

sys_config 配置说明如下:

- sensor(x)_used: 0 - disable, 1 - enable。该字段暂未生效。
- sensor(x)_mname: 表示 sensor 的名字。同上述 sensor_list 结构体下的 cam_name 成员。
- sensor(x)_twi_cci_spi: 表示 sensor 所使用的 bus 类型 (0 - twi/i2c, 1 - cci, 2 - spi, 3 - gpio)。同上述 sensor_list 结构体中的 sensor_bus_type 成员。
- sensor(x)_twi_cci_id: sensor 所使用的 bus 的 id。同上述 sensor_list 结构体中的 sensor_bus_sel 成员。
- sensor(x)_twi_addr: sensor 的 twi 地址。同上述 sensor_list 结构体下的 cam_addr 成员。

- sensor(x)_cam_type: sensor 的类型。0 - YUV; 1 - RAW。同上述 sensor_list 结构体下的 cam_type 成员。
- sensor(x)_mclk_id: sensor 所使用的 mclk 的 id。同上述 sensor_list 结构体中的 mclk_id 成员。
- sensor(x)_pos: sensor 的位置，前置还是后置。同上述 sensor_list 结构体中的 sensor_pos 成员。
- sensor(x)_isp_used: 是否使用 ISP。0 - 不使用; 1 - 使用。同上述 sensor_list 结构体下的 is_isp_used 成员。
- sensor(x)_fmt: 是否是 bayer raw。0 - 不是; 1 - 是。同上述 sensor_list 结构体下的 is_bayer_raw 成员。
- sensor(x)_vflip: flip in vertical direction. 0 - disable; 1 - enable。同上述 sensor_list 结构体下的 vflip 成员。
- sensor(x)_hflip: flip in horizontal direction. 0 - disable; 1 - enable。同上述 sensor_list 结构体下的 hflip 成员。
- sensor(x)_iovsdd_vol: camera 模组的 io power voltage。
- sensor(x)_avdd_vol: camera 模组的 analog power voltage。
- sensor(x)_dvdd_vol: camera 模组的 core power voltage。
- sensor(x)_power_en: camera 模组的 power enable gpio。
- sensor(x)_pwn: camera 模组的 power pwn gpio。
- sensor(x)_reset: camera 模组的 power reset gpio。

2.4 源码结构介绍

驱动位于 source/ekernel/subsys/avframework/v4l2/drivers/media/platform/sunxi-vin。

```

sunxi-vin/
├── modules
│   ├── sensor
│   │   ├── C2398_mipi.c           ;具体的sensor驱动
│   │   ├── camera_cfg.h         ;camera ioctl扩展命令头文件
│   │   ├── camera.h             ;camera公用结构体头文件
│   │   ├── imx386_mipi.c        ;具体的sensor驱动
│   │   ├── Makefile
│   │   ├── sensor_helper.c      ;sensor公用操作接口函数文件
│   │   └── sensor_helper.h      ;sensor公用操作接口函数头文件
│   ├── platform
│   │   ├── platform_cfg.h       ;平台相关的配置接口
│   │   └── sun8iw19p1_vin_cfg.h ;具体的平台配置头文件
│   └── utility
│       ├── vin_config_sun8iw19p1.h ;csi以及sensor配置结构体
│       └── vin_config_sun8iw19p1_real.h ;真实csi以及sensor的配置信息

```

3 Sensor 驱动开发

可以在 Melis/source/ekernel/subsys/avframework/v4l2/drivers/media/platform/sunxi-vin/modules/sensor 目录下拷贝一份 sensor 驱动，通过修改内容来进行新 sensor 驱动开发。

⚠ 警告

当前 Melis V833 支持 MIPI 接口的 sensor、并口 YUV/RAW sensor 以及 BT656/BT1120 接口的 sensor。

由于 V833 CSIC 无 CCI，当前 Melis CSI 驱动不支持 CCI 方式的 bus 来与 sensor 通信。

3.1 SENSOR_NAME

首先，将驱动中的 SENSOR_NAME 宏修改为对应的 sensor 名称，不要与现有驱动重名。如：

```
#define SENSOR_NAME "imx386_mipi"
```

其次，修改 sensor 的地址宽度和数据宽度，如地址宽度为 16bit，数据宽度为 8bit 则：

```
static struct cci_driver cci_drv = {  
    .name = SENSOR_NAME,  
    .addr_width = CCI_BITS_16,  
    .data_width = CCI_BITS_8,  
};
```

3.2 Register list 填充

每一个寄存器表配置 sensor 一种帧率和分辨率的输出。如：

```
static struct regval_list sensor_4k30_regs[] = {  
    {0x0100, 0x00},  
    {0xFFFF, 0x01},  
    {0x0112, 0x0A},  
    .....  
};
```

```
}
```

3.3 sensor_win_sizes 填充

每一个窗口对应一种帧率和分辨率，对应一组 register list。

```
static struct sensor_win_size sensor_win_sizes[] = {
    .....
    {
        .width      = 4032,
        .height     = 2256,
        .hoffset    = 0,
        .voffset    = 0,
        .hts        = 4296,
        .vts        = 2326,
        .pclk       = 300*1000*1000,
        .mipi_bps   = 800*1000*1000,
        .fps_fixed  = 30,
        .bin_factor = 1,
        .intg_min   = 16,
        .intg_max   = (2326-4)<<4,
        .gain_min   = 16,
        .gain_max   = (128<<4),
        .regs       = sensor_4k30_regs,
        .regs_size  = ARRAY_SIZE(sensor_4k30_regs),
        .set_size   = NULL,
        .top_clk    = 336*1000*1000,
        .isp_clk    = 326*1000*1000,
#ifdef CONFIG_SENSOR_CROP
        .vipp_hoff  = VIDEO_OFFSET_H,
        .vipp_voff  = VIDEO_OFFSET_V,
#endif
    },
    .....
}
```

3.4 sensor_formats 填充

主要是配置 mbus_code，如 RG10 应该配置成：

```
static struct sensor_format_struct sensor_formats[] = {
    {
        .desc = "Raw RGB Bayer",
        .mbus_code = MEDIA_BUS_FMT_SRGGB10_1X10,
        .regs = sensor_fmt_raw,
        .regs_size = ARRAY_SIZE(sensor_fmt_raw),
        .bpp = 1
    }
}
```

```
    },  
};
```

3.5 sensor 接口实现

主要需要实现如下接口：

```
static int sensor_s_exp(struct v4l2_subdev *sd, unsigned int exp_val);  
static int sensor_s_gain(struct v4l2_subdev *sd, int gain_val);  
static int sensor_s_exp_gain(struct v4l2_subdev *sd, struct sensor_exp_gain *exp_gain);  
static int sensor_power(struct v4l2_subdev *sd, int on);  
static int sensor_detect(struct v4l2_subdev *sd);  
static int sensor_init(struct v4l2_subdev *sd, u32 val)  
static long sensor_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)  
static int sensor_g_mbus_config(struct v4l2_subdev *sd, struct v4l2_mbus_config *cfg);
```

其中

- sensor_power 要根据 sensor datasheet 中的上电时序来配置。
- sensor_detect 用于检测 IIC 是否正常读写。
- sensor_init 初始化 sensor。
- sensor_ioctl 设置以及获取 sensor 的信息。
- sensor_s_exp、sensor_s_gain、sensor_s_exp_gain 用于 sensor 的曝光和增益控制，isp 的 AE 会调用这些接口。
- sensor_g_mbus_config 用于告知 paser/mipi 该 sensor 的接口属性。如 mipi 4lane 单通道的 mbus_config 如下：

```
static int sensor_g_mbus_config(struct v4l2_subdev *sd,  
                               struct v4l2_mbus_config *cfg)  
{  
    cfg->type = V4L2_MBUS_CSI2;  
    cfg->flags = 0 | V4L2_MBUS_CSI2_4_LANE | V4L2_MBUS_CSI2_CHANNEL_0;  
  
    return 0;  
}
```

3.6 sensor 测试

Melis 上提供一些测试程序，为 camera sensor 的测试提供参考。

测试程序位于 `source/ekernel/subsys/avframework/v4l2/drivers/media/platform/sunxi-vin/vin_test/mplane_image` 目录下，它调用 V4L2 ioctl API，主要实现 camera 视频格式、大小等的设置，以及怎样获取 frame buffer 和释放。



4 接口描述

VIN 驱动基于 V4L2 框架实现，对应用层提供/dev/videoX 与/dev/mediaX 节点。通过/dev/videoX 节点进行相应视频流和控制操作；通过/dev/mediaX 节点应用可以获取媒体设备拓扑结构，并能够通过 API 控制子设备间数据流向。应用层使用标准 V4L2 API 接口即可，主要调用流程如下图所示。

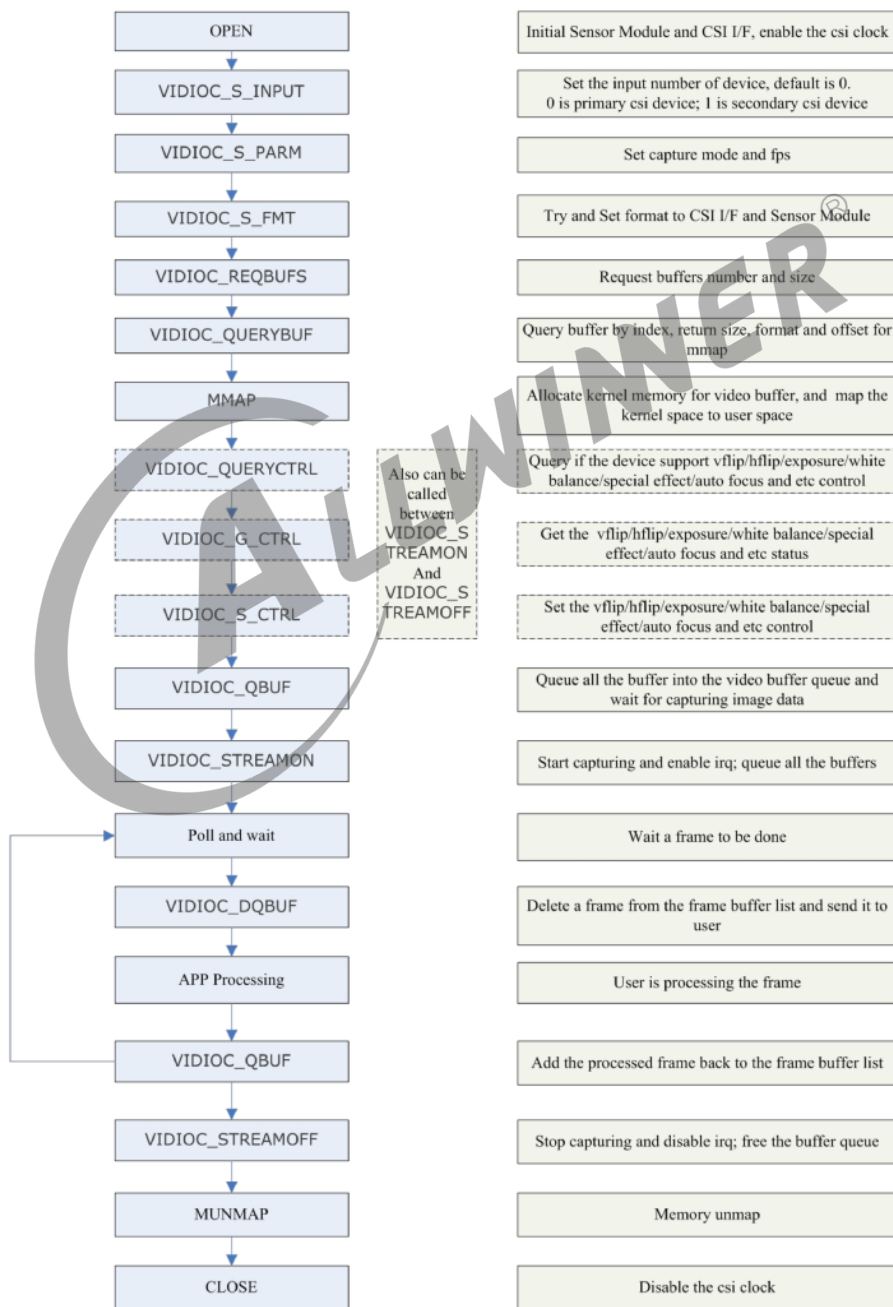


图 4-1: 应用调用流程

4.1 VIDIOC_QUERYCAP

Parameters

Capability of csi driver (struct v4l2_capability *capability)

```
struct v4l2_capability {
    __u8    driver[16]; /* i.e. "bttv" */
    __u8    card[32];  /* i.e. "Hauppauge WinTV" */
    __u8    bus_info[32]; /* "PCI:" + pci_name(pci_dev) */
    __u32   version;    /* should use KERNEL_VERSION() */
    __u32   capabilities; /* Device capabilities */
    __u32   device_caps;
    __u32   reserved[3];
};
```

Returns

Success:0; Fail: Failure Number

Description

获取驱动的名称、版本、支持的 capabilities 等, 如 V4L2_CAP_VIDEO_CAPTURE_MPLANE、V4L2_CAP_STREAMING 等。

4.2 VIDIOC_ENUM_INPUT

Parameters

input (struct v4l2_input *inp)

```
struct v4l2_input {
    __u32   index;      /* Which input */
    __u8    name[32];   /* Label */
    __u32   type;       /* Type of input */
    __u32   audioset;   /* Associated audios (bitfield) */
    __u32   tuner;      /* Associated tuner */
    v4l2_std_id std;
    __u32   status;
    __u32   capabilities;
    __u32   reserved[3];
};
```

Returns

Success:0; Fail: Failure Number

Description

获取驱动支持的 input index。目前驱动只支持 input index = 0 或 index = 1。其中 Index = 0 表示 primary csi device；Index = 1 表示 secondary csi device。

应用输入 index，驱动返回 type。对于 VIN 设备来说，type 为 V4L2_INPUT_TYPE_CAMERA。

4.3 VIDIOC_S_INPUT

Parameters

input (struct v4l2_input *inp)

Returns

Success:0; Fail: Failure Number

Description

通过 inp.index 设置当前要访问的 csi device 为 primary device 还是 secondary device。其中 Index = 0（双摄像头配置中，一般对应后置双摄像头。若只有一个摄像头设备，则 index 固定为 0）；Index = 1（双摄像头配置中，一般对应前置摄像头）。

调用该接口后，实际上会对 csi device 进行初始化工作。

4.4 VIDIOC_G_INPUT

Parameters

input (struct v4l2_input *inp)

Returns

Success:0; Fail: Failure Number

Description

获取 inp.index，判断当前设置的 csi device 为 primary device 还是 secondary device。其中 Index = 0（双摄像头配置中，一般对应后置双摄像头。若只有一个摄像头设备，则 index 固定为 0）；Index = 1（双摄像头配置中，一般对应前置摄像头）。

4.5 VIDIOC_S_PARM

Parameters

Parameter (struct v4l2_streamparm *parms)

```

struct v4l2_streamparm {
    enum v4l2_buf_type type;
    union {
        struct v4l2_captureparm capture;
        struct v4l2_outputparm output;
        __u8    raw_data[200]; /* user-defined */
    } parm;
};

struct v4l2_captureparm {
    __u32    capability; /* Supported modes */
    __u32    capturemode; /* Current mode */
    struct v4l2_fract timeperframe; /* Time per frame in .1us units */
    __u32    extendedmode; /* Driver-specific extensions */
    __u32    readbuffers; /* # of buffers for read */
    __u32    reserved[4];
};

```

Returns

Success:0; Fail: Failure Number

Description

CSI 作为输入设备，只关注 parms.type 和 parms.capture。

应用使用时，parms.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;

通过设定 parms->capture.capturemode (V4L2_MODE_VIDEO 或 V4L2_MODE_IMAGE) ，实现视频或图片的采集。

通过设定 parms->capture.timeperframe，可以设置帧率。

4.6 VIDIOC_G_PARM

Parameters

Parameter (struct v4l2_streamparm *parms)

Returns

Success:0; Fail: Failure Number

Description

应用使用时，parms.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;

通过 parms->capture.capturemode，返回当前采集模式 V4L2_MODE_VIDEO 或

V4L2_MODE_IMAGE;

通过 `parms->capture.timeperframe`, 返回当前设置的帧率。

4.7 VIDIOC_ENUM_FMT

Parameters

V4L2 format (struct v4l2_fmtdesc *fmtdesc)

```
struct v4l2_fmtdesc {
    __u32      index;           /* Format number      */
    enum v4l2_buf_type type;   /* buffer type       */
    __u32      flags;
    __u8       description[32]; /* Description string */
    __u32      pixelformat;    /* Format fourcc      */
    __u32      reserved[4];
};
```

Returns

Success:0; Fail: Failure Number

Description

获取驱动支持的 V4L2 格式。输入 `type`、`index` 参数, 返回 `pixelformat`。

对于 VIN 设备, `type` 为 `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE`。

4.8 VIDIOC_TRY_FMT

Parameters

Video type, format and size (struct v4l2_format *fmt)

```
struct v4l2_format {
    enum v4l2_buf_type type;
    union {
        struct v4l2_pix_format      pix;
        struct v4l2_pix_format_mplane pix_mp;
        struct v4l2_window          win;
        struct v4l2_vbi_format      vbi;
        struct v4l2_sliced_vbi_format sliced;
        __u8      raw_data[200];
    } fmt;
};
```

```

struct v4l2_pix_format {
    __u32          width;
    __u32          height;
    __u32          pixelformat;
    enum v4l2_field field;
    __u32          bytesperline; /* for padding, zero if unused */
    __u32          sizeimage;
    enum v4l2_colorspace colorspace;
    __u32          priv; /* private data, depends on pixelformat */
    __u32          flags; /* format flags (V4L2_PIX_FMT_FLAG_*) */
    __u32          ycbcr_enc; /* enum v4l2_ycbcr_encoding */
    __u32          quantization; /* enum v4l2_quantization */
    __u32          xfer_func; /* enum v4l2_xfer_func */
};

```

Returns

Success:0; Fail: Failure Number

Description

根据捕捉视频的类型、格式和大小，判断模式、格式等是否被驱动支持。不会改变任何硬件设置。

对于 VIN 设备，type 为 V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE。

使用 struct v4l2_pix_format_mplane 进行参数传递。

应用程序输入 struct v4l2_pix_format_mplane 结构体里面的 width、height、pixelformat、field 等参数，驱动返回最接近的 width、height；若 pixelformat、field 不支持，则默认选择驱动支持的第一种格式。

4.9 VIDIOC_S_FMT

Parameters

Video type, format and size (struct v4l2_format *fmt)

Returns

Success:0; Fail: Failure Number

Description

设置捕捉视频的类型、格式和大小，设置之前会调用 VIDIOC_TRY_FMT。

对于 VIN 设备，type 为 V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE。

使用 struct v4l2_pix_format_mplane 进行参数传递。

应用程序输入 width、height、pixelformat、field 等，驱动返回最接近的 width、height；若 pixelformat、field 不支持，则默认选择驱动支持的第一种格式。

应用程序应该以驱动返回的 width、height、pixelformat、field 等作为后续使用传递的参数。

对于 OSD 设备，type 为 V4L2_BUF_TYPE_VIDEO_OVERLAY。使用 struct v4l2_window 进行参数传递。

应用程序输入水印的个数、窗口位置和大小、bitmap 地址、bitmap 格式以及 global_alpha 等。驱动保存这些参数，并在 VIDIOC_OVERLAY 命令传递使能命令时生效。

4.10 VIDIOC_G_FMT

Parameters

Video type, format and size (struct v4l2_format *fmt)

Returns

Success:0; Fail: Failure Number

Description

获取捕捉视频的 width、height、pixelformat、field、bytesperline、sizeimage 等参数。

4.11 VIDIOC_OVERLAY

Parameters

Overlay on/off (unsigned int i)

Returns

Success:0; Fail: Failure Number

Description

传递 1 表示使能，0 表示关闭。设置使能时会更新 osd 参数，使之生效。

4.12 VIDIOC_REQBUFS

Parameters

Buffer type, count and memory map type (struct v4l2_requestbuffers * req)

```

struct v4l2_requestbuffers {
    __u32          count;
    enum v4l2_buf_type    type;
    enum v4l2_memory     memory;
    __u32          reserved[2];
};

```

Returns

Success:0; Fail: Failure Number

Description

v4l2_requestbuffers 结构中定义了缓存的数量，驱动会据此申请对应数量的视频缓存。多个缓存可以用于建立 FIFO，来提高视频采集的效率。这些 buffer 通过内核申请，申请后需要通过 mmap 方法，映射到 User 空间。

Count: 定义需要申请的video buffer数量
 Type: 对于VIN设备, 为V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE
 Memory: 目前支持V4L2_MEMORY_MMAP、V4L2_MEMORY_USERPTR、V4L2_MEMORY_DMABUF方式

应用程序传递上述三个参数，驱动会根据 VIDIOC_S_FMT 设置的格式计算供需要 buffer 的大小，并返回 count 数量。

4.13 VIDIOC_QUERYBUF

Parameters

Buffer type ,index and memory map type (struct v4l2_buffer *buf)

```

struct v4l2_buffer {
    __u32          index;
    enum v4l2_buf_type    type;
    __u32          bytesused;
    __u32          flags;
    enum v4l2_field      field;
    struct timeval      timestamp;
    struct v4l2_timecode  timecode;
    __u32          sequence;

    /* memory location */
    enum v4l2_memory     memory;
    union {
        __u32          offset;
        unsigned long  userptr;
        struct v4l2_plane *planes;
    };
};

```

```
__s32      fd;  
} m;  
__u32      length;  
__u32      input;  
__u32      reserved;  
};
```

Returns

Success:0; Fail: Failure Number

Description

通过 struct v4l2_buffer 结构体的 index, 访问对应序号的 buffer, 获取到对应 buffer 的缓存信息。主要利用 length 信息及 m.offset 信息来完成 mmap 操作。

4.14 VIDIOC_DQBUF

Parameters

Buffer type ,index and memory map type (struct v4l2_buffer *buf)

Returns

Success:0; Fail: Failure Number

Description

将 driver 已经填充好数据的 buffer 出列, 供应用使用。

应用程序根据 index 来识别 buffer, 此时 m.offset 表示 buffer 对应的物理地址。

4.15 VIDIOC_QBUF

Parameters

Buffer type ,index and memory map type (struct v4l2_buffer *buf)

Returns

Success:0; Fail: Failure Number

Description

将 User 空间已经处理过的 buffer, 重新入队, 移交给 driver, 等待填充数据。

应用程序根据 index 来识别 buffer。

4.16 VIDIOC_STREAMON

Parameters

Buffer type (enum v4l2_buf_type *type)

Returns

Success:0; Fail: Failure Number

Description

此处的 buffer type 为 V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE。运行此 IOCTL，将 buffer 队列中所有 buffer 入队，并开启 CSIC DMA 硬件中断，每次中断便表示完成一帧 buffer 数据的填入。

4.17 VIDIOC_STREAMOFF

Parameters

Buffer type (enum v4l2_buf_type *type)

Returns

Success:0; Fail: Failure Number

Description

此处的 buffer type 为 V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE。运行此 IOCTL，停止捕捉视频，将 frame buffer 队列清空，以及 video buffer 释放。

4.18 VIDIOC_QUERYCTRL

Parameters

Control id and value (struct v4l2_queryctrl *qc)

```
struct v4l2_queryctrl {
    __u32      id;
    enum v4l2_ctrl_type type;
    __u8       name[32]; /* Whatever */
    __s32      minimum; /* Note signedness */
    __s32      maximum;
    __s32      step;
    __s32      default_value;
```

```
__u32          flags;  
__u32          reserved[2];  
};
```

Returns

Success:0; Fail: Failure Number

Description

应用程序通过 id 参数，驱动返回需要调节参数的 name, minmum, maximum, default_value 以及步进 step。（由 v4l2 conctrols framework 完成）

目前可能支持的 id 请参考 VIDIOC_S_CTRL。

4.19 VIDIOC_S_CTRL

Parameters

Control id and value (struct v4l2_queryctrl *qc)

Returns

Success:0; Fail: Failure Number

Description

应用程序通过 id, value 等参数，对 camera 驱动对应的参数进行设置。

驱动内部会先调用 vidioc_queryctrl, 判断 id 是否支持, value 是否在 minimum 和 maximum 之间。（由 v4l2 conctrols framework 完成）

目前可能支持的 id 和 value 参考附件。

4.20 VIDIOC_G_CTRL

Parameters

Control id and value (struct v4l2_queryctrl *qc)

Returns

Success:0; Fail: Failure Number

Description

应用程序通过 id, 驱动返回对应 id 当前设置的 value。

4.21 VIDIOC_ENUM_FRAMESIZES

Parameters

index, type, format (struct v4l2_frmsizeenum)

```
enum v4l2_frmsizetypes {
    V4L2_FRMSIZE_TYPE_DISCRETE = 1,
    V4L2_FRMSIZE_TYPE_CONTINUOUS = 2,
    V4L2_FRMSIZE_TYPE_STEPWISE = 3,
};

struct v4l2_frmsize_discrete {
    __u32 width; /* Frame width [pixel] */
    __u32 height; /* Frame height [pixel] */
};

struct v4l2_frmsize_stepwise {
    __u32 min_width; /* Minimum frame width [pixel] */
    __u32 max_width; /* Maximum frame width [pixel] */
    __u32 step_width; /* Frame width step size [pixel] */
    __u32 min_height; /* Minimum frame height [pixel] */
    __u32 max_height; /* Maximum frame height [pixel] */
    __u32 step_height; /* Frame height step size [pixel] */
};

struct v4l2_frmsizeenum {
    __u32 index; /* Frame size number */
    __u32 pixel_format; /* Pixel format */
    __u32 type; /* Frame size type the device supports. */

    union { /* Frame size */
        struct v4l2_frmsize_discrete discrete;
        struct v4l2_frmsize_stepwise stepwise;
    };

    __u32 reserved[2]; /* Reserved space for future use */
};
```

Returns

Success:0; Fail: Failure Number

Description

根据应用传进来的 index, pixel_format, 驱动返回 type, 并根据 type 填写 discrete 或 stepwise 的值。Discrete 表示分辨率固定的值; stepwise 表示分辨率有最小值和最大值, 并根据 step 递增。上层根据返回的 type, 做对应不同的操作。

4.22 VIDIOC_ENUM_FRAMEINTERVALS

Parameters

Index, format, size, type (struct v4l2_fmvalenum)

```
enum v4l2_fmvaltypes {
    V4L2_FRMIVAL_TYPE_DISCRETE = 1,
    V4L2_FRMIVAL_TYPE_CONTINUOUS = 2,
    V4L2_FRMIVAL_TYPE_STEPWISE = 3,
};

struct v4l2_fmival_stepwise {
    struct v4l2_fract min; /* Minimum frame interval [s] */
    struct v4l2_fract max; /* Maximum frame interval [s] */
    struct v4l2_fract step; /* Frame interval step size [s] */
};

struct v4l2_fmvalenum {
    __u32 index; /* Frame format index */
    __u32 pixel_format; /* Pixel format */
    __u32 width; /* Frame width */
    __u32 height; /* Frame height */
    __u32 type; /* Frame interval type the device supports. */

    union { /* Frame interval */
        struct v4l2_fract discrete;
        struct v4l2_fmival_stepwise stepwise;
    };

    __u32 reserved[2]; /* Reserved space for future use */
};
```

Returns

Success:0; Fail: Failure Number

Description

应用程序通过 pixel_format、width、height、驱动返回 v4l2_fmvalenum.type，并根据 v4l2_fmvalenum.type 填写 V4L2_FRMIVAL_TYPE_DISCRETE、CONTINUOUS 或 STEPWISE。Discrete 表示支持单一的帧率；stepwise 表示支持步进的帧率。




著作权声明

版权所有 © 2020 珠海全志科技股份有限公司。保留一切权利。

本档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本档内容的部分或全部，且不得以任何形式传播。

商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本档作为使用指导仅供参考。由于产品版本升级或其他原因，本档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本档中提供准确的信息，但并不确保内容完全没有错误，因使用本档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。