Linux system CSI equipment drive development-2012.1.30

Index Record

	Drafter	Date	Remark
V1.0	Raymonxiu	2012-1-17	Initial index

CONTENT

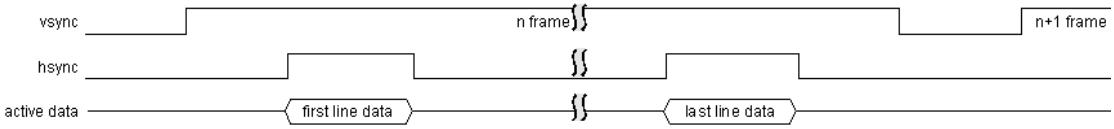
1. CSI introduction

CSI is the port that collects the symbols from Cmos Sensor, Video Encoder and other video output equipments. The port supports Hsync, Vsync synchronous control way or embedded synchronous BT656 way.

The general camera module uses Hsync, Vsync synchronous control way.

1.1. CSI hardware working principles

CSI timing



Vref= positive; Href= positive

In the valid area of Vsync and Hsync, making the data sample through pclk and buffering the picture data to dram

1.2. CSI hardware debugging cautions

1. Before sensor initialization, make sure the voltage of all sensors' charge is correct
2. Before sensor initialization, make sure to reset, standby according to the control process in sensor; otherwise, which will cause series insoluble problems
3. Make sure the MCLK has the signal output before writing I2C order to sensor, the MCLK should be at 24MHz.
4. If the I2C order is fail when initializing, the operator should check all sensors' chargers, power on process and MCLK signal situation. The operator can also reduce the speed of I2C or delay a period of time after writing I2C order.
5. Generally speaking, the initialization is successful if PCLK and VSYNC, HSYNC have the signals output; if the polarization configuration of PCLK, VSYNC and HSYNC are correct, the picture receiving is correct too.
6. If the received pictures appear irregular green lines, which might means the PCLK drive power is short. The operator can solve it by turn up the drive power; which is outputted by sensor through I2C.
7. If the received image appears irregular fine color lines, it might be that the added file of camera is interfered. The operator can add small size capacity filtering.
8. If two sensors share one CSI, the blank screen, video corruption, color lines and other problems happen when transferring data, the reasons caused in transferring process, the operator doesn't shift sensor IO PAD to high resistance status, then, it could be held on when the other sensor IO outputs.

2. Linux system CSI drive procedure

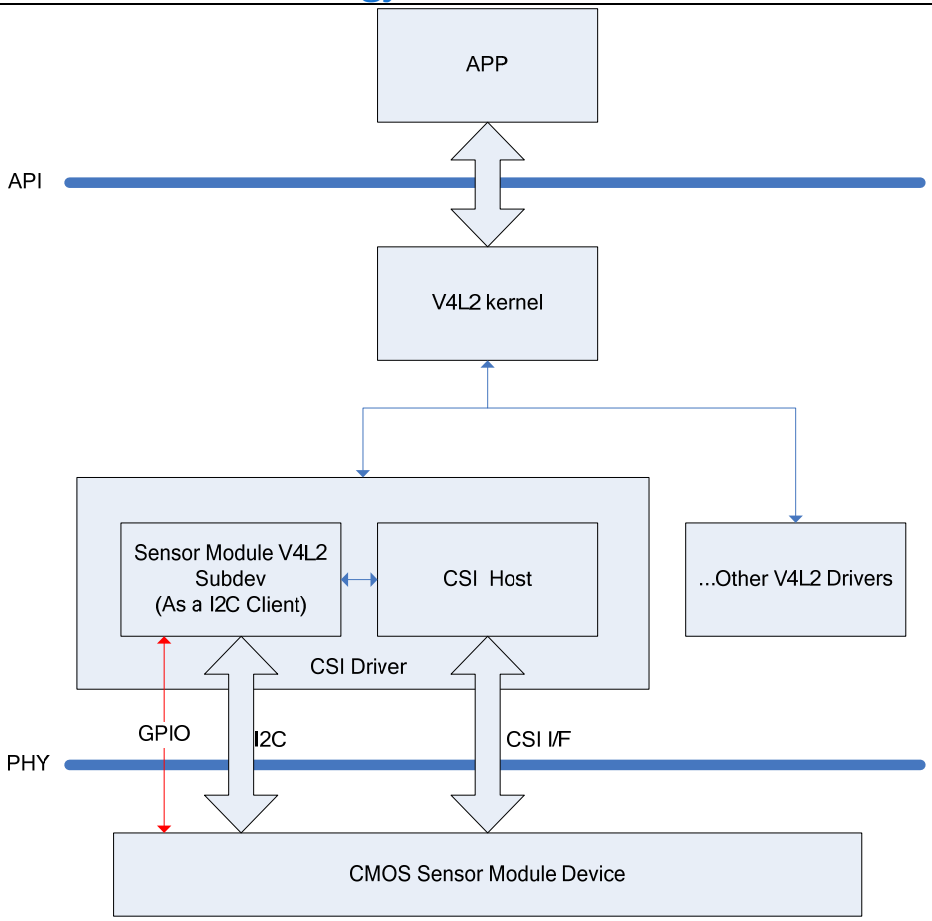
2.1. CSI drive file content framework

CSI drive file site is: linux-3.0/drivers/media/video/sun4i_csi/

The framework is as below:

```
|-- sun4i_csi
    |--csi0
       |--sun4i_csi_reg.c
       |--sun4i_drv_csi.c
       |--sun4i_csi_reg.h
       |--Makefile
    |--csi1
       |--sun4i_csi_reg.c
       |--sun4i_drv_csi.c
       |--sun4i_csi_reg.h
       |--Makefile
    |--device
       |--gc0307.c
       |--gc0308.c
       |--gt2005.c
       |--hi253.c
       |--hi704.c
       |--mt9d112.c
       |--mt9m112.c
       |--mt9m113.c
       |--ov2655.c
       |--ov5640.c
       |--ov7670.c
        |--sp0838.c
        |--Makefile
     |--include
       |--sun4i_csi_core.h
       |--sun4i_dev_csi.h
     |--Kconfig
```

2.2. CSI drive level framework

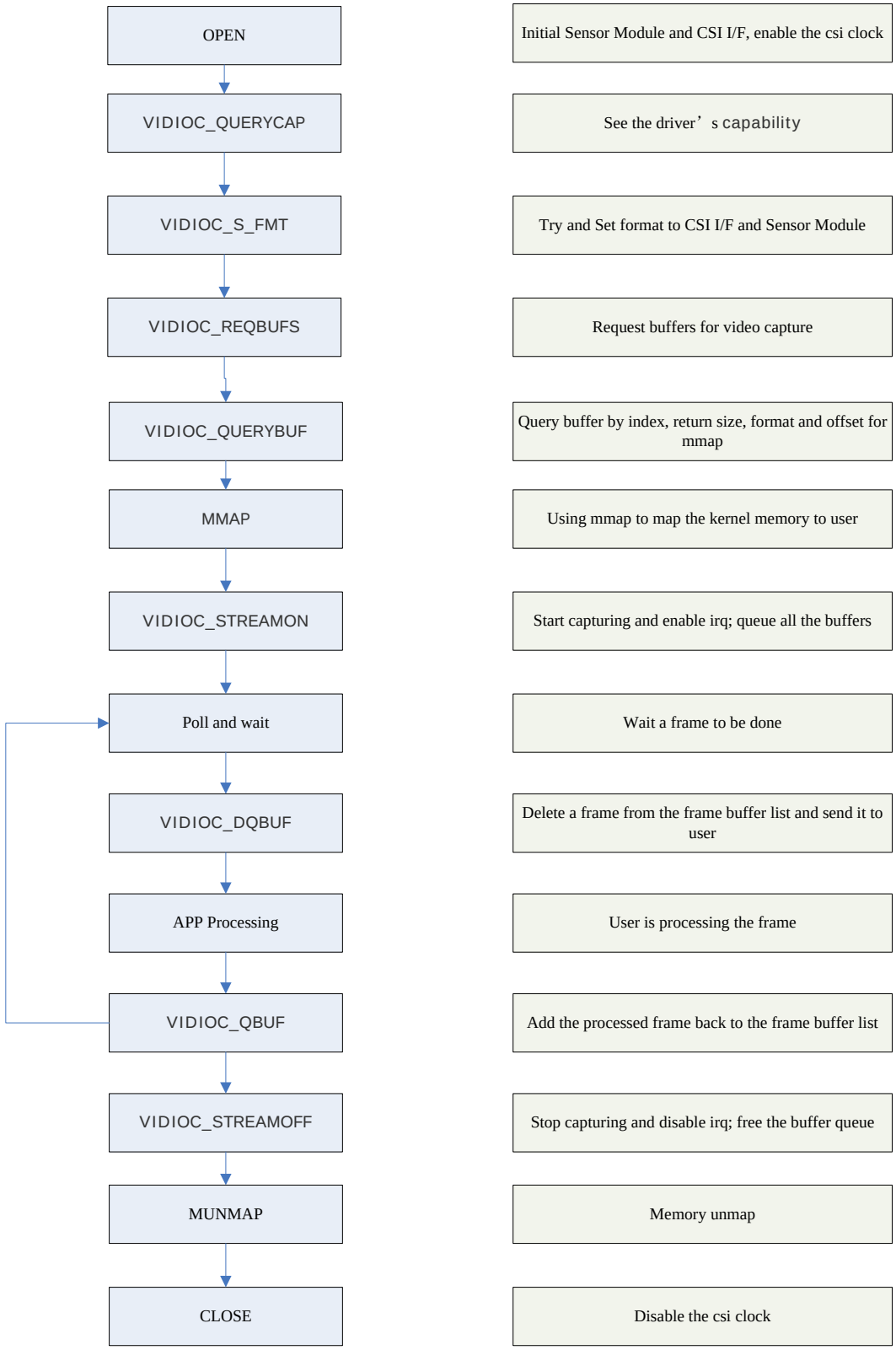


CSI drive level constructure

CSI drive is designed based on Linux V4L2 framework and can meet the standard V4L2 API calling mode. Because the data buffering of CSI hardware collection requires physical continuous internal memory, the video-dma-contig internal memory way is applied for the management.

Like the picture above, the CSI drive main part includes CSI Host and Sensor Module V4L2 Subdev as well as V4L2 Kernel original codes. The corresponding original code is sun4i_csi_reg.c, sun4i_drv_csi.c and camera module original code (such as gc0308.c,gt2005.c and so on). Among those, sun4i_csi_reg.c is the HAL level of CSI hardware, sun4i_drv_csi.c is the main part of CSI drive, which applies to initialize CSI drive, connect V4;2 API and manage video buffer and so on. Camera module original code (like gc0308.c, gt2005.c and so on), which makes the camera initialization, resolution, come true, image format, white balance, special effects, exposure and other effects set as well as charge management come true.

2.3. CSI drive calling procedure



2.4. CSI V4L2 Subdev port function

2.4.1. V4L2 Subdev function set

Camera drive developer should focus on the port function between camera module and CSI main part. There are two types, they are sensor_core_ops and sensor_video_ops. sensor_core_ops is power, standby, effect set and ioctl extension; sensor_video_ops is defined as the port that sets image data format and frame rate. Below is the specific definition:

```
static const struct v4l2_subdev_core_ops sensor_core_ops = {
    .g_chip_ident = sensor_g_chip_ident,
    .g_ctrl = sensor_g_ctrl,
    .s_ctrl = sensor_s_ctrl,
    .queryctrl = sensor_queryctrl,
    .reset = sensor_reset,
    .init = sensor_init,
    .s_power = sensor_power,
    .ioctl = sensor_ioctl,
};

static const struct v4l2_subdev_video_ops sensor_video_ops = {
    .enum_mbus_fmt = sensor_enum_fmt,
    .try_mbus_fmt = sensor_try_fmt,
    .s_mbus_fmt = sensor_s_fmt,
    .s_parm = sensor_s_parm,
    .g_parm = sensor_g_parm,
};

static const struct v4l2_subdev_ops sensor_ops = {
    .core = &sensor_core_ops,
    .video = &sensor_video_ops,
};
```

2.4.2. sensor_reset function

Prototype：static int sensor_reset(struct v4l2_subdev *sd, u32 val)

Main Function：implement three orders related to reset

```
switch(val)
    {
        case CSI_SUBDEV_RST_OFF:

  // control camera IO or send I2C order and make it release reset
  break;
  // control camera IO or send I2C order and make it hold reset
  break;
  // control camera IO or send I2C order and make it as the time order break like: reset->hold->release
  break
```

```
        default:
            return -EINVAL;
    }
```

2.4.3. sensor_power function

Prototype：static int sensor_power(struct v4l2_subdev *sd, int on)

Main Function: implement four orders related power/standby

```
switch(on)
    {
        case CSI_SUBDEV_STBY_ON:

        // control sensor and enter into standby timing sequence
        break;
        // control sensor exit standby timing sequence
        break;
        // control sensor power on timing sequence
        break;
        // control sensor power off timing sequence
        break;



        default:
            return -EINVAL;
    }
```

2.4.4. sensor_init function

Main function:To test sensor id, and initialize sensor.
sensor_detect function is to read sensor id figure, if it is the same with the object, it will get back to OK;
sensor_default_regs is the array that reserves I2C order of sensor initialization.

```
ret = sensor_detect(sd);
if (ret) {
    csi_dev_err("chip found is not an target chip.\n");
    return ret;
}
return sensor_write_array(sd, sensor_default_regs , ARRAY_SIZE(sensor_default_regs));
```

2.4.5. sensor_queryctrl function

Prototype：static int sensor_queryctrl(struct v4l2_subdev *sd, struct v4l2_queryctrl *qc)
Main function: get back to all effect sets that sensor supports and the related max/min figure and step size.
The developer should pay attention that in the practical operation,
it is better to note it if the sensor doesn't support effect feature; otherwise,
the system regards it as the sensor can support in the default way when processing the former stage.
Generally speaking, the mix/min figure and step size of all allocations couldn't be modified except GAIN parameter.
VFLIP , HFLIP (these two parameters involve the camera imaging direction,
and it is corresponding to csi_hflip and csi_vflip in sys_config1 file), EXPOSURE,DO_WHITE_BALANCE,
COLORFX are the parameters should be implemented,

```
switch (qc->id) {
        case V4L2_CID_BRIGHTNESS:
                return v4l2_ctrl_query_fill(qc, -4, 4, 1, 1);
        case V4L2_CID_CONTRAST:
                return v4l2_ctrl_query_fill(qc, -4, 4, 1, 1);
        case V4L2_CID_SATURATION:
                return v4l2_ctrl_query_fill(qc, -4, 4, 1, 1);
        case V4L2_CID_HUE:
                return v4l2_ctrl_query_fill(qc, -180, 180, 5, 0);
        case V4L2_CID_VFLIP:
        case V4L2_CID_HFLIP:
                return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
        case V4L2_CID_GAIN:
                return v4l2_ctrl_query_fill(qc, 0, 255, 1, 128);
        case V4L2_CID_AUTOGAIN:
                return v4l2_ctrl_query_fill(qc, 0, 1, 1, 1);
        case V4L2_CID_EXPOSURE:
                return v4l2_ctrl_query_fill(qc, -4, 4, 1, 0);
        case V4L2_CID_EXPOSURE_AUTO:
                return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
        case V4L2_CID_DO_WHITE_BALANCE:
                return v4l2_ctrl_query_fill(qc, 0, 5, 1, 0);
        case V4L2_CID_AUTO_WHITE_BALANCE:
                return v4l2_ctrl_query_fill(qc, 0, 1, 1, 1);
        case V4L2_CID_COLORFX:
                return v4l2_ctrl_query_fill(qc, 0, 9, 1, 0);
        case V4L2_CID_CAMERA_FLASH_MODE:
           return v4l2_ctrl_query_fill(qc, 0, 4, 1, 0);
}
return -EINVAL;
```

2.4.6. sensor_s_ctrl function

Prototype：static int sensor_s_ctrl(struct v4l2_subdev *sd, struct v4l2_control *ctrl)

Main function: The installation of all sensor effect features. Implementing VFLIP, HFLIP,EXPOSURE, DO_WHITE_BALANCE,AUTO_WHITE_BALANCE,COLORFX only.

```
        case V4L2_CID_BRIGHTNESS:
                return sensor_s_brightness(sd, ctrl->value);
        case V4L2_CID_CONTRAST:
                return sensor_s_contrast(sd, ctrl->value);
        case V4L2_CID_SATURATION:
                return sensor_s_saturation(sd, ctrl->value);
        case V4L2_CID_HUE:
```

```
            return sensor_s_hue(sd, ctrl->value);
        case V4L2_CID_VFLIP:
            return sensor_s_vflip(sd, ctrl->value);
        case V4L2_CID_HFLIP:
            return sensor_s_hflip(sd, ctrl->value);
        case V4L2_CID_GAIN:
            return sensor_s_gain(sd, ctrl->value);
        case V4L2_CID_AUTOGAIN:
            return sensor_s_autogain(sd, ctrl->value);
        case V4L2_CID_EXPOSURE:
            return sensor_s_exp(sd, ctrl->value);
        case V4L2_CID_EXPOSURE_AUTO:
            return sensor_s_autoexp(sd,
                    (enum v4l2_exposure_auto_type) ctrl->value);
        case V4L2_CID_DO_WHITE_BALANCE:
            return sensor_s_wb(sd,
                    (enum v4l2_whiteblance) ctrl->value);
        case V4L2_CID_AUTO_WHITE_BALANCE:
            return sensor_s_autowb(sd, ctrl->value);
        case V4L2_CID_COLORFX:
            return sensor_s_colorfx(sd,
                    (enum v4l2_colorfx) ctrl->value);
        case V4L2_CID_CAMERA_FLASH_MODE:
           return sensor_s_flash_mode(sd,
                (enum v4l2_flash_mode) ctrl->value);
        }
return -EINVAL;
```

Attention, the functions called from sensor_s_ctrl are for different sensors, which needs the different implementations. Below contents are what all functions need to implement.

```
sensor_s_brightness(sd, ctrl->value)
sensor_s_contrast(sd, ctrl->value)
sensor_s_saturation(sd, ctrl->value)
sensor_s_exp(sd, ctrl->value)


 // implement the callings of  brightness/contrast ratio/saturation/exposure
    (ctrl=>value, min=-4,max=4,step=1).
 // and make the nine data abstract form -4~4 to the specific register of sensor and install.
 //brightness is to implement the array of sensor_brightness_zero_regs[] and the register
 //contrast is to implement the array of sensor_contrast_zero_regs[] and the register
 //saturation is to implement the array of sensor_saturation_zero_regs[] and the register
 //exp is to implement the array of sensor_ev_zero_regs[] and the register
 sensor_shue(sd,ctrl->value)
 // call the specific register of sensor, install hue
```

```
sensor_s_vflip(sd, ctrl->value)
sensor_s_hflip(sd, ctrl->value)
sensor_s_autowb(sd, ctrl->value)
sensor_s_autoexp(sd,(enum v4l2_exposure_auto_type) ctrl->value)
// install the sensor vflip (upsidedown), hflip (mirror),AWB, AE
//enable position

sensor_s_wb(sd,(enum v4l2_whiteblance) ctrl->value)
// install all white balance scene
// implementation
//sensor_wb_auto_regs[],sensor_wb_cloud_regs[],sensor_wb_daylight_regs[],
//sensor_wb_incandescence_regs[],sensor_wb_fluorescent_regs[],sensor_wb_tungsten_regs[]
// and the array of registers.

sensor_s_colorfx(sd,(enum v4l2_colorfx) ctrl->value);
// install various special effects
// implementation
// sensor_colorfx_none_regs[],sensor_colorfx_bw_regs[],sensor_colorfx_sepia_regs[],
// sensor_colorfx_negative_regs[],sensor_colorfx_emboss_regs[],sensor_colorfx_sketch_regs[]
// sensor_colorfx_sky_blue_regs[],sensor_colorfx_grass_green_regs[],
// sensor_colorfx_skin_whiten_regs[],sensor_colorfx_vivid_regs[]
// and the array of register

sensor_s_flash_mode(sd,(enum v4l2_flash_mode) ctrl->value)
// install flashlight module
```

2.4.7. sensor_g_ctrl function

Prototype： static int sensor_g_ctrl(struct v4l2_subdev *sd, struct v4l2_control *ctrl)
Main function: the equipment that obtains all sensors' effect features.
it is basically to implement VFLIP,HFLIP,EXPOSURE,
DP_WHITE_BALANCE,AUTO_WHITE_BALANCE,COLORFX only.

```
switch (ctrl->id) {
    case V4L2_CID_BRIGHTNESS:
        return sensor_g_brightness(sd, &ctrl->value);
    case V4L2_CID_CONTRAST:
        return sensor_g_contrast(sd, &ctrl->value);
    case V4L2_CID_SATURATION:
        return sensor_g_saturation(sd, &ctrl->value);
    case V4L2_CID_HUE:
        return sensor_g_hue(sd, &ctrl->value);
    case V4L2_CID_VFLIP:
        return sensor_g_vflip(sd, &ctrl->value);
    case V4L2_CID_HFLIP:
        return sensor_g_hflip(sd, &ctrl->value);
    case V4L2_CID_GAIN:
```

```
            return sensor_g_gain(sd, &ctrl->value);
        case V4L2_CID_AUTOGAIN:
            return sensor_g_autogain(sd, &ctrl->value);
        case V4L2_CID_EXPOSURE:
            return sensor_g_exp(sd, &ctrl->value);
        case V4L2_CID_EXPOSURE_AUTO:
            return sensor_g_autoexp(sd, &ctrl->value);
        case V4L2_CID_DO_WHITE_BALANCE:
            return sensor_g_wb(sd, &ctrl->value);
        case V4L2_CID_AUTO_WHITE_BALANCE:
            return sensor_g_autowb(sd, &ctrl->value);
        case V4L2_CID_COLORFX:
            return sensor_g_colorfx(sd,    &ctrl->value);
        case V4L2_CID_CAMERA_FLASH_MODE:
            return sensor_g_flash_mode(sd, &ctrl->value);
    }
    return -EINVAL;
```

2.4.8. sensor_ioctl function

Prototype: static long sensor_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)
Main Function: an extentive port of sun4i_drv_csi.c from CSI main part. Transmitting module clok, polarity of vsync, hsync, mclk frequency and iocfg property (when two sensors connect with one CSI, and the corresponding module is id 0 or id 1) through _csi_subdev_info_t framework.
This port doesn't allow any modifications.

```
static long sensor_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)
{
    int ret=0;

    switch(cmd){
        case CSI_SUBDEV_CMD_GET_INFO:
        {
            struct sensor_info *info = to_state(sd);
            __csi_subdev_info_t *ccm_info = arg;
            ccm_info->mclk    =    info->ccm_info->mclk ;
            ccm_info->vref    =    info->ccm_info->vref ;
            ccm_info->href    =    info->ccm_info->href ;
            ccm_info->clock   =    info->ccm_info->clock;
            ccm_info->iocfg   =    info->ccm_info->iocfg;
            break;
        }
        case CSI_SUBDEV_CMD_SET_INFO:
        {
            struct sensor_info *info = to_state(sd);
            __csi_subdev_info_t *ccm_info = arg;
            info->ccm_info->mclk  =    ccm_info->mclk    ;
```

```
                info->ccm_info->vref    =    ccm_info->vref     ;
                info->ccm_info->href    =    ccm_info->href     ;
                info->ccm_info->clock   =    ccm_info->clock    ;
                info->ccm_info->iocfg   =    ccm_info->iocfg    ;
                break;
            }
            default:
                return -EINVAL;
        }
        return ret;
}
```

2.4.9. sensor_enum_fmt function

Prototype：static int sensor_enum_fmt(struct v4l2_subdev *sd, unsigned index,
enum v4l2_mbus_pixelcode *code)

Main function: get back to the corresponding image format of index through *code. Sensor_format[] array reserves all image formats that could be supported.

```
        if (index >= N_FMTS)
            return -EINVAL;
        *code = sensor_formats[index].mbus_code;
        return 0;
```

2.4.10 Sensor_try_fmt function
To make the comparison with the array in sensor_win_sizes[] and then get the supported image format;
and the size in sensor_win_sizes[] and get the closest size format through v412_mbus_framefmt data framework,
upper setting imaging format, size format, and also through calling sensor_try_fmt.
The sensor won't be changed at all after calling. The port is the general type to various sensor,
there is no need to change sensor_try_fmt, but just install the related register array of .regs
in both sensor_formats[] and sensor_win_sizes[].

```
static int sensor_try_fmt(struct v4l2_subdev *sd,
                    struct v4l2_mbus_framefmt *fmt)
{
        return sensor_try_fmt_internal(sd, fmt, NULL, NULL);
}

static int sensor_try_fmt_internal(struct v4l2_subdev *sd,
            struct v4l2_mbus_framefmt *fmt,
            struct sensor_format_struct **ret_fmt,
            struct sensor_win_size **ret_wsize)
{
        int index;
        struct sensor_win_size *wsize;
        for (index = 0; index < N_FMTS; index++)
```

```
            if (sensor_formats[index].mbus_code == fmt->code)
                break;

    if (index >= N_FMTS) {
            /* default to first format */
            index = 0;
            fmt->code = sensor_formats[0].mbus_code;
    }

    if (ret_fmt != NULL)
            *ret_fmt = sensor_formats + index;

    /*
     * Fields: the sensor devices claim to be progressive.
     */
    fmt->field = V4L2_FIELD_NONE;

    /*
     * Round requested image size down to the nearest
     * we support, but not below the smallest.
     */
    for (wsize = sensor_win_sizes; wsize < sensor_win_sizes + N_WIN_SIZES;
             wsize++)
            if (fmt->width >= wsize->width && fmt->height >= wsize->height)
                break;

    if (wsize >= sensor_win_sizes + N_WIN_SIZES)
            wsize--;      /* Take the smallest one */
    if (ret_wsize != NULL)
            *ret_wsize = wsize;
    /*
     * Note the size we'll actually handle.
     */
    fmt->width = wsize->width;
    fmt->height = wsize->height;
    return 0;
}
```

2.4.11 Sensor_s_fmt function
To make the comparison with the array in sensor_win_sizes[]
and then get the supported image format; and the size in sensor_win_sizes[]
and get the closest size format through v412_mbus_framefmt data framework,
upper setting imaging format, size format as well as returning back the indicator
and also through calling sensor_s_fmt, but first to call sensor_try_fmt_internal.
The port is the general type to various sensor, there is no need to change sensor_s_fmt,
but just install the related register array of .regs in both sensor_formats[] and sensor_win_sizes[].

```
static int sensor_s_fmt(struct v4l2_subdev *sd,
                        struct v4l2_mbus_framefmt *fmt)
{
    int ret;
    struct sensor_format_struct *sensor_fmt;
    struct sensor_win_size *wsize;
    struct sensor_info *info = to_state(sd);
    ret = sensor_try_fmt_internal(sd, fmt, &sensor_fmt, &wsize);
    if (ret)
        return ret;

    sensor_write_array(sd, sensor_fmt->regs , sensor_fmt->regs_size);

    ret = 0;
    if (wsize->regs)
    {
        ret = sensor_write_array(sd, wsize->regs , wsize->regs_size);
        if (ret < 0)
            return ret;
    }

    if (wsize->set_size)
    {
        ret = wsize->set_size(sd);
        if (ret < 0)
            return ret;
    }

    info->fmt = sensor_fmt;
    info->width = wsize->width;
    info->height = wsize->height;

    return 0;
}
```

2.5. I2C Access Way in CSI drive

2.5.1. sensor_write

Prototype： static int sensor_write(struct v4l2_subdev *sd, unsigned char *reg,
unsigned char *value)

Take gt2005 for example, there are 16 digital register addresses,
how does the I2C register with 8 digital data write Oxaa to Ox0505?

```
struct regval_list regs;
regs.reg_num[0] = 0x05;
regs.reg_num[1] = 0x05;
regs. value [0] = 0xaa;
ret = sensor_write(sd, regs.reg_num, regs.value);
if(ret < 0)
   csi_dev_err("sensor_write err!\n");
```

2.5.2. sensor_read

Prototype：static int sensor_read(struct v4l2_subdev *sd, unsigned char *reg,
unsigned char *value)

Take gt2005 for example, there are 16 digital register address,
how does the I2C register with 8 digital data read Ox0505 register figure and print it out?

```
struct regval_list regs;
regs.reg_num[0] = 0x05;
regs.reg_num[1] = 0x05;
ret = sensor_read(sd, regs.reg_num, regs.value);
if(ret < 0)
   csi_dev_err("sensor_read err!\n");
else
   csi_dev_print("sensor read from 0x0505 = %x\n",regs.value[0]);
```

2.5.3. sensor_write_array

Prototype：static int sensor_write_array(struct v4l2_subdev *sd, struct regval_list *vals , uint size)

The function is to install to I2C writing process to a definite struct regval_list array.
Take gt2005 for example, in sensor_init, the operator should follow I2C writing process by calling sensor_default_regs \
and initialize the sensor.

```
static struct regval_list sensor_default_regs[] = {
//……
}

sensor_write_array(sd, sensor_default_regs , ARRAY_SIZE(sensor_default_regs));
```

3. Camera module transmitting based on SUN4I platform

3.1 The key factors of camera module transmitting

Generally speaking, if the operator wants to add a camera module support,
the key factors are: add the xxx.c file under sun4i_csi/device content, implement poweron/off,
standby on/off, reset ports, install mclk, vsync, hsync polarities and clock frequency,
finally fill the initialized code and default VGA resolution set in sensor_default_regs[] and
sensor_vga_regs[], then, it is possible to receive the images.

3.2 Camera module transmitting procedure

Take the gt2005.c for example to explain how the new camera module should be transmitted.

3.2.1 Camera ID modification

The camera ID should be modified to a specific and easily to identify ID from gt2005 and
unify the setting in sys_config1.fex in the future. It is recommended that the camera ID should be revised
to the same name of c file to identify.

```
static const struct i2c_device_id sensor_id[] = {
        { "gt2005", 0 },
        { }
};

static struct i2c_driver sensor_driver = {
        .driver = {
                .owner = THIS_MODULE,
        .name = "gt2005",
        },
        .probe = sensor_probe,
        .remove = sensor_remove,
        .id_table = sensor_id,
};
```

3.2.2 Camera signal output

The polarity of the different MCLK, VSYNC, HSYNC outputted from different module patterns,
when calling, the operator can get the information according to the sensor original initialized code
and the related datasheet or through the camera module pattern agent or sensor original plant.
The ccm_info_com in gt3005.c file is just used for storing the datasheet,
it communicates with sun4i_drv_csi.c through sensor_ioctl extended port.

```
#define MCLK (24*1000*1000)
#define VREF_POL        CSI_HIGH
#define HREF_POL        CSI_HIGH
#define CLK_POL         CSI_RISING
#define IO_CFG          0

__csi_subdev_info_t ccm_info_con =
{
        .mclk       = MCLK,
        .vref       = VREF_POL,
        .href       = HREF_POL,
```

```
    .clock      = CLK_POL,
    .iocfg      = IO_CFG,
};
```

Remark:
Generally speaking, it can realize through these macro definitions like MCLK, VREF_POL,HREF_POL, CLK_POL. The unit of MCLK is Hz, and it is set to 24MHz or 12MHz, and the other frequencies are got from the PLL in the system, the original frequency is 270MHz or 297HMz, the frequency demultiplication is 1~16. VREF_POL, HREF_POL can be set to CSI_HIGH or CSI_LOW, it is the valid area for image transmitting, and the related AVSYNC and HSYNC signal isn't stable (sometimes is high, sometimes is low). CLK_POL can be set to CSI_RISING or CSI_FALLING, it decides the sensor PCLK to collect the data in the ascended way or descended way. The IO_CFG is used to mark the different camera when two cameras share one CIS. The users don't need to care about it because it can modify the data positively when calling sensor_ioclt port through sun4i_drv_csi.c. The default data is 0.

3.2.3 Camera controls IO polarity

The polarities of standby, reset controlled by different camera module pattern is different, we apply some macro definitions to present.

```
#define CSI_STBY_ON         0
#define CSI_STBY_OFF        1
#define CSI_RST_ON          0
#define CSI_RST_OFF         1
#define CSI_PWR_ON          1
#define CSI_PWR_OFF         0
```

Remark:
CSI_STBY_ON means the pin status of camera standby when standby is valid, and the data of most sensors is 1.
CSI_STBY_OFF means the pin status of camera standby with camera works when standby is invalid, and the data of most sensors is 0.
CSI_RST_ON means the pin status of camera reset when reset is valid, and the data of most sensors is 0.
CSI_RST_OFF means the pin status of camera reset with camera works when rest is invalid, and the data of most sensors is 0.
P.S, when the definitions of all above pin status is uncertain, please refer to the datasheet of camera.
The right setting of the above information can be important of calling the new camera module patterns.

3.2.4 camera I2C order size

The register and data size of different camera module pattern are different, the operator can modify them through below macro definitions.

```
#define REG_ADDR_STEP 2
#define REG_DATA_STEP 1
#define REG_STEP              (REG_ADDR_STEP+REG_DATA_STEP)
```

REG_ADDR_STEP means the length or size of sensor register address would be visited through I2C and calculate it by the mulriple of byte.
REG_DATA_SETP means the size of sensor register data would be visited through I2C and calculate it by the mulriple of byte.
In this case, the register address of gt2005 is the length (size) of two bytes (16 digital addresses). The register data is the length (size) of one byte (8 digital data).

The following format must be followed when defining the register data array.

```
static struct regval_list xxx_regs[] = {
{{0x00,   0x00} , {0xff}},
}
```

3.2.5 camera initialized register data array

```
Static struct regval_list sensor_default_regs[]={
// fill sensor initialized code, it could be set when the drive is opened}
Static struct regval_list sensor_usga_regs[]={
// fill sensor and set to the code of uxga resolution, it could be set when
taking photo or video resolution is changed};
Static struct regval_list sensor_hd720_regs[]={
// fill sensor and set to the code of 720p resolution, it could be set when
taking photo or video resolution is changed};
Static struct regval_list sensor_svga_regs[]={
// fill sensor and set to the code of svga resolution, it could be set when
taking photo or video resolution is changed};
Static struct regval_list sensor_vga_regs[]={
//fill sensor and set to the code of VGA resolution, the resolution could be used
in default preview and it could be set when previewing};
Static struct regval_list sensor_yuv422_yuyv[]={
// fill sensor and set to register code outputted from yuyv. It could be set
when the upper layer calls s_fmt};


 Static struct regval_list sensor_yuv422_yvyu[]={
// fill sensor and set to register code outputted from yvyu.
It could be set when the upper layer calls s_fmt};
Static struct regval_list sensor_yuv422_vyuy[]={
// fill sensor and set to register code outputted from vyuy.
It could be set when the upper layer calls s_fmt};
Static struct regval_list sensor_yuv422_ uyvy []={
// fill sensor and set to register code outputted from uyvy.
It could be set when the upper layer calls s_fmt};
Static struct regval_list sensor_fmt_ raw []={
// fill sensor and set to register code outputted from raw.
It could be set when the upper layer calls s_fmt};
```

Among the above mentioned register setting arrays, sensor_Defualt_regs[], sensor_vga_regs[] and sensor_fmt_yuv422_yuyv[], sensor_fmt_yuv422_yvyu[], sensor_fmt_yuv422_vyuy[], sensor_fmt_yuv422_uyvy[] are the necessary arrays, otherwise, the images couldn't be received normally. The setting of other resolutions could be added in such way. Furthermore, sensor_fmt_raw[] shouldn't be supported, because the current platform doesn't support raw format.

3.2.6. Camera picture format mapping array

sensor_formats[] is used to store upper layer calling format mbus_code and sensor setting

```
static struct sensor_format_struct {
    __u8 *desc;
    //__u32 pixelformat;
    enum v4l2_mbus_pixelcode mbus_code;//linux-3.0
    struct regval_list *regs;
    int    regs_size;
    int bpp;    /* Bytes per pixel */
} sensor_formats[] = {
    {
        .desc        = "YUYV 4:2:2",
        .mbus_code   = V4L2_MBUS_FMT_YUYV8_2X8,
        .regs        = sensor_fmt_yuv422_yuyv,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_yuyv),
        .bpp         = 2,
    },
    {
        .desc        = "YVYU 4:2:2",
        .mbus_code   = V4L2_MBUS_FMT_YVYU8_2X8,
        .regs        = sensor_fmt_yuv422_yvyu,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_yvyu),
        .bpp         = 2,
    },
    {
        .desc        = "UYVY 4:2:2",
        .mbus_code   = V4L2_MBUS_FMT_UYVY8_2X8,
        .regs        = sensor_fmt_yuv422_uyvy,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_uyvy),
        .bpp         = 2,
    },
    {
        .desc        = "VYUY 4:2:2",
        .mbus_code   = V4L2_MBUS_FMT_VYUY8_2X8,
        .regs        = sensor_fmt_yuv422_vyuy,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_vyuy),
        .bpp         = 2,
    },
    {
        .desc        = "Raw RGB Bayer",
        .mbus_code   = V4L2_PIX_FMT_SBGGR8,
        .regs        = sensor_fmt_raw,
        .regs_size = ARRAY_SIZE(sensor_fmt_raw),
```

```
        .bpp        = 1
    },
};
```

Basically speaking, there is no need to revise this array, the format of upper layer calling and the mapping relationship of the related array names is fix.

3.2.7 Camera resolution mapping array

Sensor_win_size[] is to used to store the relationship between upper layer calling resolution and actual sensor setting.

```
static struct sensor_win_size {
    int    width;
    int    height;
    int    hstart;        /* Start/stop values for the camera.    Note */
    int    hstop;         /* that they do not always make complete */
    int    vstart;        /* sense to humans, but evidently the sensor */
    int    vstop;         /* will do the right thing... */
    struct regval_list *regs; /* Regs to tweak */
    int regs_size;
    int (*set_size) (struct v4l2_subdev *sd);
/* h/vref stuff */
} sensor_win_sizes[] = {
    /* UXGA */
    {
        .width          = UXGA_WIDTH,
        .height         = UXGA_HEIGHT,
        .regs           = sensor_uxga_regs,
        .regs_size      = ARRAY_SIZE(sensor_uxga_regs),
        .set_size       = NULL,
    },
    /* 720p */
    {
        .width          = HD720_WIDTH,
        .height         = HD720_HEIGHT,
        .regs           = sensor_hd720_regs,
        .regs_size      = ARRAY_SIZE(sensor_hd720_regs),
        .set_size       = NULL,
    },
    /* SVGA */
    {
        .width          = SVGA_WIDTH,
        .height         = SVGA_HEIGHT,
        .regs           = sensor_svga_regs,
        .regs_size      = ARRAY_SIZE(sensor_svga_regs),
        .set_size       = NULL,
    },
    /* VGA */
```

```
    {
        .width          = VGA_WIDTH,
        .height         = VGA_HEIGHT,
        .regs           = sensor_vga_regs,
        .regs_size      = ARRAY_SIZE(sensor_vga_regs),
        .set_size       = NULL,
    },
};
```

Sensor_win_size[] content is set according to the actual applicable resolution.
Width and height settings must be the same as regs data. Generally speaking,
it must be realized to use VGA resolution when previewing. Other resolutions that aren't used should be annotated.

3.2.8 Camera Power Standby control

Sensor_power port is to realize the sequential control of the four orders like CSI_SUBDEV_STBY_ON, CSI_SUBDEV_STBY_OFF, CSI_SUBDEV_PWR_ON, CSI_SUBDEV_PWR_OFF. Below is a specific example to illustrate.
It is better to refer to sensor datasheet for the actual power/standby sequential control.

```
static int sensor_power(struct v4l2_subdev *sd, int on)
{
    struct csi_dev *dev=(struct csi_dev *)dev_get_drvdata(sd->v4l2_dev->dev);
    struct sensor_info *info = to_state(sd);
    char csi_stby_str[32],csi_power_str[32],csi_reset_str[32];

    if(info->ccm_info->iocfg == 0) {
        strcpy(csi_stby_str,"csi_stby");
        strcpy(csi_power_str,"csi_power_en");
        strcpy(csi_reset_str,"csi_reset");
    } else if(info->ccm_info->iocfg == 1) {
       strcpy(csi_stby_str,"csi_stby_b");
       strcpy(csi_power_str,"csi_power_en_b");
       strcpy(csi_reset_str,"csi_reset_b");
    }

  switch(on)
    {
        case CSI_SUBDEV_STBY_ON:
            csi_dev_dbg("CSI_SUBDEV_STBY_ON\n");
            //reset off io
            gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
            msleep(10);
            //active mclk before stadby in
            clk_enable(dev->csi_module_clk);
            msleep(100);
            //standby on io
```

```
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_STBY_ON,csi_stby_str);
                msleep(100);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_STBY_OFF,csi_stby_str);
                msleep(100);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_STBY_ON,csi_stby_str);
                msleep(100);
                //inactive mclk after stadby in
                clk_disable(dev->csi_module_clk);
                //reset on
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_ON,csi_reset_str);
                msleep(10);
                break;
        case CSI_SUBDEV_STBY_OFF:
                csi_dev_dbg("CSI_SUBDEV_STBY_OFF\n");
                //active mclk before stadby out
                clk_enable(dev->csi_module_clk);
                msleep(10);
                //reset off io
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
                msleep(10);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_ON,csi_reset_str);
                msleep(100);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_STBY_OFF,csi_stby_str);
                msleep(10);
                break;
        case CSI_SUBDEV_PWR_ON:
                csi_dev_dbg("CSI_SUBDEV_PWR_ON\n");
                //inactive mclk before power on
                clk_disable(dev->csi_module_clk);
                //power on reset
                gpio_set_one_pin_io_status(dev->csi_pin_hd,1,csi_stby_str);//set the gpio to
output
                gpio_set_one_pin_io_status(dev->csi_pin_hd,1,csi_reset_str);//set the gpio to
output
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_STBY_ON,csi_stby_str);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_ON,csi_reset_str);
                msleep(1);
                //power supply
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_PWR_ON,csi_power_str);
                msleep(10);
                if(dev->dvdd) {
                        regulator_enable(dev->dvdd);
                        msleep(10);
```

```
        }
        if(dev->avdd) {
            regulator_enable(dev->avdd);
            msleep(10);
        }
        if(dev->iovdd) {
            regulator_enable(dev->iovdd);
            msleep(10);
        }
        //active mclk before power on
        clk_enable(dev->csi_module_clk);
        //reset after power on
        gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
        msleep(10);
        gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_ON,csi_reset_str);
        msleep(100);
        gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
        msleep(100);
        gpio_write_one_pin_value(dev->csi_pin_hd,CSI_STBY_OFF,csi_stby_str);
        msleep(10);
        break;

    case CSI_SUBDEV_PWR_OFF:
        csi_dev_dbg("CSI_SUBDEV_PWR_OFF\n");
        //power supply off
        if(dev->iovdd) {
            regulator_disable(dev->iovdd);
            msleep(10);
        }
        if(dev->avdd) {
            regulator_disable(dev->avdd);
            msleep(10);
        }
        if(dev->dvdd) {
            regulator_disable(dev->dvdd);
            msleep(10);
        }
        gpio_write_one_pin_value(dev->csi_pin_hd,CSI_PWR_OFF,csi_power_str);
        msleep(10);

        //inactive mclk after power off
        clk_disable(dev->csi_module_clk);

        //set the io to hi-z
```

```
                gpio_set_one_pin_io_status(dev->csi_pin_hd,0,csi_reset_str);//set the gpio to input
                gpio_set_one_pin_io_status(dev->csi_pin_hd,0,csi_stby_str);//set the gpio to input
                break;
        default:
                return -EINVAL;
    }
    return 0;
}
```

Among above introduction, it is firstly to identify info->ccm_info->iocfg in order to let two sensors can connect to one CSI port and control it by the different IO.
Below are several functions to use to create power on/off, standby on/off.

Gpio_set_one_pin_io_status()// set the input or output status of some IO
Gpio_write_one_pin_value()//set the high electrical level or low electrical level of some IO output
Clk_enable() // enable mclk, mclk would be released from csi_mclk pin
Clk_disable() // disable mclk, mclk become low electrical level
Regulartor_enable() // the pmu ldo of the related enable
Regulator_disable() // the pmu ldo of the related disable
Cautions, the related sensor IO should be set t the high resister status
when two sensors connect to one CSI port and realize the CSI_SUBDEV_STBY_ON function.
Otherwise, the sensor output signal would be held when shifting to another sensor,
which could lead to the image black screen or blurred screen.

3.2. 9Camera Reset control
Sensor_Reset port is applied to realize the control function of three major orders
like CSI_SUBDEV_RST_OFF, CSI_SUBDEV_Rst_on, CSI_SUBDEV_RST_PUL. Below is the standard rest io control example.

```
static int sensor_reset(struct v4l2_subdev *sd, u32 val)
{
    struct csi_dev *dev=(struct csi_dev *)dev_get_drvdata(sd->v4l2_dev->dev);
    struct sensor_info *info = to_state(sd);
    char csi_reset_str[32];

    if(info->ccm_info->iocfg == 0) {
        strcpy(csi_reset_str,"csi_reset");
    } else if(info->ccm_info->iocfg == 1) {
       strcpy(csi_reset_str,"csi_reset_b");
    }

    switch(val)
    {
        case CSI_SUBDEV_RST_OFF:
            csi_dev_dbg("CSI_SUBDEV_RST_OFF\n");
            gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
            msleep(10);
            break;
        case CSI_SUBDEV_RST_ON:
            csi_dev_dbg("CSI_SUBDEV_RST_ON\n");
```

```
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_ON,csi_reset_str);
                msleep(10);
                break;
        case CSI_SUBDEV_RST_PUL:
                csi_dev_dbg("CSI_SUBDEV_RST_PUL\n");
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
                msleep(10);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_ON,csi_reset_str);
                msleep(100);
                gpio_write_one_pin_value(dev->csi_pin_hd,CSI_RST_OFF,csi_reset_str);
                msleep(10);
                break;
        default:
                return -EINVAL;
    }
    return 0;
}
```

At the same time, it is firstly to identify info->ccm_info->iocfg in order to use the different IO control to identify when two sensors share one CSI port.

3.2.10. Sensor_detect modification

The main purpose of sensor_Detect is to read id from camera sensor and compare with target id. Below is the example of gt2005.c, sensor detect read id from 0x0000 register and compare with 0x51 The final implementation would be revised according to the actual camera.

```
static int sensor_detect(struct v4l2_subdev *sd)
{
    int ret;
    struct regval_list regs;

    regs.reg_num[0] = 0x00;
    regs.reg_num[1] = 0x00;
    ret = sensor_read(sd, regs.reg_num, regs.value);
    if (ret < 0) {
        csi_dev_err("sensor_read err at sensor_detect!\n");
        return ret;
    }

    if(regs.value[0] != 0x51)
        return -ENODEV;
    return 0;
}
```

3.2.11. sensor_s_hflip modification

Sensor_s_hflip is to read enable position of hflip from camera sensor, and set camera sensor hflip enable or disable according to the actual demand

Below is the example of gt2005.c, read the data form 0x0101,set the related bit0 of enable or disable according to hflip,and modify the camera register according to the actual demand.

```
static int sensor_s_hflip(struct v4l2_subdev *sd, int value)
{
	int ret;
	struct sensor_info *info = to_state(sd);
	struct regval_list regs;

	regs.reg_num[0] = 0x01;
	regs.reg_num[1] = 0x01;
	ret = sensor_read(sd, regs.reg_num, regs.value);
	if (ret < 0) {
		csi_dev_err("sensor_read err at sensor_s_hflip!\n");
		return ret;
	}

	switch (value) {
		case 0:
		   regs.value[0] &= 0xfe;
			break;
		case 1:
			regs.value[0] |= 0x01;
			break;
		default:
			return -EINVAL;
	}
	ret = sensor_write(sd, regs.reg_num, regs.value);
	if (ret < 0) {
		csi_dev_err("sensor_write err at sensor_s_hflip!\n");
		return ret;
	}

	msleep(100);

	info->hflip = value;
	return 0;
}
```

3.2.12 Sensor_g_hflip modification

Sensor_g_hflip is used to read the enable postion of hflip from camera sensor and get back to upper layer. Below is the example based on gt2005.c, read the data from 0x0101, get back to enable or disable to the related hflip according to bit0 and modify the camera register according to the actual demand.

```
static int sensor_g_hflip(struct v4l2_subdev *sd, __s32 *value)
```

```
{
        int ret;
        struct sensor_info *info = to_state(sd);
        struct regval_list regs;

        regs.reg_num[0] = 0x01;
        regs.reg_num[1] = 0x01;
        ret = sensor_read(sd, regs.reg_num, regs.value);
        if (ret < 0) {
                csi_dev_err("sensor_read err at sensor_g_hflip!\n");
                return ret;
        }

        regs.value[0] &= (1<<0);
        regs.value[0] = regs.value[0]>>0;          //0x0101 bit0 is mirror

        *value = regs.value[0];

        info->hflip = *value;
        return 0;
}
```

3.2.13. Sensor_s_vflip modification

Sensor_S_vflip is to read the enable postion of vflip from camera sensor and set the camera sensor vflip enable or disable.

Below is the example based on gt2005.c, read the data from 0x0101, set the related bit1 of enable or disable according to hflip and modify the camera register according to the actual demand.

```
static int sensor_s_vflip(struct v4l2_subdev *sd, int value)
{
        int ret;
        struct sensor_info *info = to_state(sd);
        struct regval_list regs;

        regs.reg_num[0] = 0x01;
        regs.reg_num[1] = 0x01;
        ret = sensor_read(sd, regs.reg_num, regs.value);
        if (ret < 0) {
                csi_dev_err("sensor_read err at sensor_s_vflip!\n");
                return ret;
        }

        switch (value) {
                case 0:
                  regs.value[0] &= 0xfd;
```

```
                break;
        case 1:
                regs.value[0] |= 0x02;
                break;
        default:
                return -EINVAL;
    }
    ret = sensor_write(sd, regs.reg_num, regs.value);
    if (ret < 0) {
        csi_dev_err("sensor_write err at sensor_s_vflip!\n");
        return ret;
    }

    msleep(100);

    info->vflip = value;
    return 0;
}
```

3.2.14. Sensor_g_vflip modification

Sensor_g_vflip is to read the enable position of vflip from camera sensor and get back to the upper layer. Below is the example based on gt2005.c, read the data from 0x0101, get back to enable or disable of the related vflip according to bit1 and modify the camera register according to the actual demand.

```
static int sensor_g_vflip(struct v4l2_subdev *sd, __s32 *value)
{
    int ret;
    struct sensor_info *info = to_state(sd);
    struct regval_list regs;

    regs.reg_num[0] = 0x01;
    regs.reg_num[1] = 0x01;
    ret = sensor_read(sd, regs.reg_num, regs.value);
    if (ret < 0) {
        csi_dev_err("sensor_read err at sensor_g_vflip!\n");
        return ret;
    }

    regs.value[0] &= (1<<1);
    regs.value[0] = regs.value[0]>>1;          //0x0101 bit1 is upsidedown

    *value = regs.value[0];

    info->vflip = *value;
    return 0;
```

```
}
```

3.2.15. Sensor_S_autowb modification

Sensor_S_autowb is to read AWB automatic white balance enable postion of camera sensor, write back AWB according to the set data from the upper layer and set it as enable or disable. Below is the example based on gt2005.c, read the data from 0x031a, set AWB enable or disable of related bit7 according to the upper layer and modify the camera register according to the actual demand.

```
static int sensor_s_autowb(struct v4l2_subdev *sd, int value)
{
	int ret;
	struct sensor_info *info = to_state(sd);
	struct regval_list regs;

	regs.reg_num[0] = 0x03;
	regs.reg_num[1] = 0x1a;
	ret = sensor_read(sd, regs.reg_num, regs.value);
	if (ret < 0) {
		csi_dev_err("sensor_read err at sensor_s_autowb!\n");
		return ret;
	}

	switch(value) {
	case 0:
		regs.value[0] &= 0x7f;
		break;
	case 1:
		regs.value[0] |= 0x80;
		break;
	default:
		break;
	}
	ret = sensor_write(sd, regs.reg_num, regs.value);
	if (ret < 0) {
		csi_dev_err("sensor_write err at sensor_s_autowb!\n");
		return ret;
	}

	msleep(10);

	info->autowb = value;
	return 0;
}
```

3.2.16. sensor_g_autowb modification

Sensor_g_autowb is to read AWB automatic white balance enable postion from camera sensor and get back to upper layer.
Below is the example based on gt2005.c, read the data from 0x031a, get back to enable or disable of the related AWB according to bit7 and modify the camera register according to the actual demand.

```
static int sensor_g_autowb(struct v4l2_subdev *sd, int *value)
{
	int ret;
	struct sensor_info *info = to_state(sd);
	struct regval_list regs;

	regs.reg_num[0] = 0x03;
	regs.reg_num[1] = 0x1a;
	ret = sensor_read(sd, regs.reg_num, regs.value);
	if (ret < 0) {
		csi_dev_err("sensor_read err at sensor_g_autowb!\n");
		return ret;
	}

	regs.value[0] &= (1<<7);
	regs.value[0] = regs.value[0]>>7;		//0x031a bit7 is awb enable

	*value = regs.value[0];
	info->autowb = *value;

	return 0;
}
```

3.2.17. Other modifications

The below functions are available for the modifications of other unnecessary function implementations

Sensor_g_autogain // automatic gain enable/disable
Sensor_s_autogain
Sensor_g_autoexp// automatic exposure enable/disable
Sensor_s_autoexp
Sensor_s_hue// hue adjustment
Sensor_s_hue
Sensor_g_gain// gain figure
Sensor_s_gain
Sensor_g_flash_mode // flash light mode
Sensor_s_flash_mode
Sensor_g_parm// fram rate adjustment
Sensor_s_parm

sensor_read 3.2.18. The functions without modification

sensor_write
sensor_write_array
sensor_init
sensor_ioctl
sensor_enum_fmt
sensor_try_fmt_internal
sensor_try_fmt
sensor_s_fmt
sensor_g_brightness
sensor_s_brightness
sensor_g_contrast
sensor_s_contrast
sensor_g_saturation
sensor_s_saturation
sensor_g_exp
sensor_s_exp
sensor_g_wb
sensor_s_wb
sensor_g_colorfx
sensor_s_colorfx
sensor_g_ctrl
sensor_s_ctrl

3.3 CSI driving configuration introduction
3.3.1. Linux lay configuration
3.3.1.1. modify Kconfig
Modifying the Kconfig file under sun4i_csi content and add camera compiling swift.
Take gt2005 for example.

```
config CSI_GT2005
        tristate "GalaxyCore GT2005 2M sensor support"
        depends on I2C && VIDEO_V4L2
        select      CSI_DEV_SEL
        ---help---
           This is a Video4Linux2 sensor-level driver for the GalaxyCore
           GT2005 2M camera.
```

3.3.1.2. Modifying Makefile
Modifying the Makefile under sun4i_csi/device content and add the camera compiling function
Take gt2005 for example, the compiling switch should be the same as the Kconfig as above stated.

```
obj-$(CONFIG_CSI_GT2005)                    += gt2005.o
```

3.3.1.3. Configuring drive through memuconfig
Under linux-3.0 content, press make ARCH=arm menuconfig
Enter into and choose per below description
Device Drivers- Multimedia support-
Make sure to choose Video For Linux
Then choose CSI Driver Config for sun4i-
Enter into and choose the related camera module pattern and compile it to module format
Save it and withdraw.

3.3.2. sys_config1.fex file configuration
In the configuration items, if the name with suffix _b means the sub-camera
when two cameras connect to one CSI, the camera without suffix is the main camera.

configuration item	Configuration meaning
csi_used =xx	whether csi0 is used or not
csi_mode=xx	set csi can receive buffer mode: receive one buffer 1: two CSI receive the content and compile to one buffer
csi_dev_qty=xx	set the component quantity of csi connection, the current set is 1 or 2
csi_stby_mode=xx	set the battery status when it is standby status 0: no power off when standby 1: power off when standby
csi_mname="" csi_mname_b=""	module pattern name of csi0 usage, it should be matched with drive and check the readme in the drive content, it has ov7670,gc0308, gt2005,hi704,sp0338,mt9m112,gc0307, mt9m113,mt9d112,hi253,ov5640 for the options at present.
csi_twi_id =xx csi_twi_id_b =xx	IIC of csi0 usage
csi_twi_addr=xx csi_twi_addr_b=xx	IIC address of csi0 module pattern, it can check the readme in the drive content
csi_if=xx csi_if_b=xx	set the port timing sequence of current module pattern: 0:8 bit data line with Hsync, Vsync 1:16 bit data line with Hsync, Vsync Vsync 3:8 bit data line with BT656 embedded synchronous function, single channel; 4:8 bit data line with BT656 embedded synchronous function, dual channel; 5:8 bit data line with BT656 embedded synchronous function, four channel;
csi_vflip=xx csi_vflip_b=xx	set csi receive image in the default way and perversion up and down: 0: normal perversion
csi_hflip=xx csi_hflip_b=xx	set csi receive image in the default way and perversion right and left 0: normal perversion

csi_iovdd = “” csi_iovdd_b=””	the IO of camera module pattern, the operator should fill "axp20_pll" if Analog and Core power is supplied from LDO3 of AXP20, if the power is supplied from LD04 of AXP20, it should fill "axp20_hdmi", it the power isn't supplied from AXP20, it should be filled""
csi_avdd = “” csi_avdd_b = “”	
csi_dvdd = “” csi_dvdd_b = “”	
csi_flash_pol=xx csi_flash_pol_b=xx	if the camera has the IO that controls flash light, and this parameter can control the valid IO polarity of flashlight 0: IO is low, the flashlight is light 1: IO is high, the flashlight is light
csi_pck＝xx	module pattern sends clock GPIO configuraiton of csi0
csi_ck=xx	csi0 sends the clock GPIO configuration of module pattern
csi_hsync=xx	module pattern sends line synchronizing signal GPIO configuration to csi0
csi_vsync=xx	module pattern sends frame synchronizing signal GPIO configuration to csi0
csi_d0=xx … csi_d15=xx	module pattern sends 8bit/16bit data GPIO configuration to csi0
csi_reset=xx csi_reset_b=xx	GPIO configuration of reset that controls module pattern, the default figure is reset valid (low or high depends on module pattern)
csi_power_en=xx csi_power_en_b=xx	GPIO configuration of battery that controls module pattern, the general default figure is valid (high)
csi_stby=xx csi_stby_b=xx	GPIO configuration of standby that controls module pattern, the general default figure is standby valid (low or high depends on module pattern)
csi_af_en=xx csi_af_en_b=xx	AF power supply that controls module pattern

Example:
One CSI connects with one camera

[csi0_para]
csi_used = 1
csi_mode = 0
csi_dev_qty = 1
csi_stby_mode = 1

csi_mname = "gt2005"
csi_twi_id = 1
csi_twi_addr = 0x78
csi_if = 0
csi_vflip = 1
csi_hflip = 1

csi_iovdd = "axp20_pll"
csi_avdd = "axp20_pll"
csi_dvdd = ""
csi_flash_pol = 1

csi_mname_b = ""
csi_twi_id_b = 1
csi_twi_addr_b = 0x42
csi_if_b = 0
csi_vflip_b = 1
csi_hflip_b = 1
csi_iovdd_b = ""
csi_avdd_b = ""
csi_dvdd_b = ""
csi_flash_pol_b = 1

csi_pck = port:PE00<3><default><default><default>
csi_ck = port:PE01<3><default><default><default>
csi_hsync = port:PE02<3><default><default><default>
csi_vsync = port:PE03<3><default><default><default>
csi_d0 = port:PE04<3><default><default><default>
csi_d1 = port:PE05<3><default><default><default>
csi_d2 = port:PE06<3><default><default><default>
csi_d3 = port:PE07<3><default><default><default>
csi_d4 = port:PE08<3><default><default><default>
csi_d5 = port:PE09<3><default><default><default>
csi_d6 = port:PE10<3><default><default><default>
csi_d7 = port:PE11<3><default><default><default>
csi_d8 =
csi_d9 =
csi_d10 =
csi_d11 =
csi_d12 =
csi_d13 =
csi_d14 =
csi_d15 =
csi_reset = port:PH13<1><default><default><0>
csi_power_en = port:PH16<1><default><default><0>
csi_stby = port:PH18<1><default><default><0>
csi_flash =
csi_af_en =
csi_reset_b =
csi_power_en_b =
csi_stby_b =

csi_flash_b =
csi_af_en_b =

One CSI port connects two cameras

[csi0_para]
csi_used = 1
csi_mode = 0
csi_dev_qty = 2
csi_stby_mode = 1

csi_mname = "gt2005"
csi_twi_id = 1
csi_twi_addr = 0x78
csi_if = 0
csi_vflip = 1
csi_hflip = 1
csi_iovdd = "axp20_pll"
csi_avdd = "axp20_pll"
csi_dvdd = ""
csi_flash_pol = 1

csi_mname_b = "gc0308"
csi_twi_id_b = 1
csi_twi_addr_b = 0x42
csi_if_b = 0
csi_vflip_b = 1
csi_hflip_b = 1
csi_iovdd_b = "axp20_pll"
csi_avdd_b = "axp20_pll"
csi_dvdd_b = ""
csi_flash_pol_b = 1

csi_pck = port:PE00<3><default><default><default>
csi_ck = port:PE01<3><default><default><default>
csi_hsync = port:PE02<3><default><default><default>
csi_vsync = port:PE03<3><default><default><default>
csi_d0 = port:PE04<3><default><default><default>
csi_d1 = port:PE05<3><default><default><default>
csi_d2 = port:PE06<3><default><default><default>
csi_d3 = port:PE07<3><default><default><default>
csi_d4 = port:PE08<3><default><default><default>
csi_d5 = port:PE09<3><default><default><default>
csi_d6 = port:PE10<3><default><default><default>
csi_d7 = port:PE11<3><default><default><default>

csi_d8 =
csi_d9 =
csi_d10 =
csi_d11 =
csi_d12 =
csi_d13 =
csi_d14 =
csi_d15 =
csi_reset = port:PH13<1><default><default><0>
csi_power_en = port:PH16<1><default><default><0>
csi_stby = port:PH18<1><default><default><0>
csi_flash =
csi_af_en =
csi_reset_b = port:PH14<1><default><default><0>
csi_power_en_b = port:PH16<1><default><default><0>
csi_stby_b = port:PH19<1><default><default><0>
csi_flash_b =
csi_af_en_b =

3.3.3 android level configuration
Take gt2005 module pattern's connection to CIS0 port for example
1. Make sure the related ko files are copied to android environment
The related ko files include video-buf-core.ko, video-dma-contig.ko, sun4i_csi0.ko, gt2005.ko

2. Make sure the related ki files installation order under android environment

insmod video-buf-core.ko
insmod video-dma-contig.ko
insmod gt2005.ko
insmod sun4i_csi0.ko

If two cameras share one CSI, the following steps would be loaded,
for example, gt2005 andgc0308 are on-hook in CSI0

insmod video-buf-core.ko
insmod video-dma-contig.ko
insmod gt2005.ko
insmod gc0308.ko
insmod sun4i_csi0.ko

Generally speaking, the loading of camera module drive should prior to sun4i_csi0.ko

3. Camera.cfg file should be configured correctly and the developer can refer to other files to configure it.