Linux 系统 CSI 设备驱动开发

Copyright © All Winner Technology. All Rights Reserved

版本历史

	修改人	时间	备注
V1.0	Raymonxiu	2012-1-17	建立初始版本

CopyRight©2011 All Winner Technology, Right Reserved

目 录

CopyRight©2011 All Winner Technology, Right Reserved

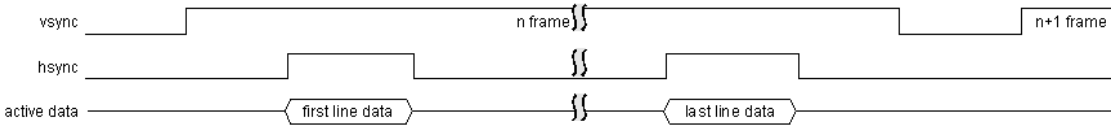
CopyRight©2011 All Winner Technology, Right Reserved

1. CSI 简介

CSI 是采集 Cmos Sensor，Video Encoder 等视频输出设备信号的接口。该接口支持 Hsync，Vsync 同步控制方式或内嵌同步 BT656 方式。一般 camera 模组采用 Hsync，Vsync 同步方式。

1.1. CSI 硬件工作原理

CSI timing



Vref= positive; Href= positive

在 Vsync 和 Hsync 有效的区域内，通过 pclk 对 data 的采样，将图像数据 buffer 到 dram

1.2. CSI 硬件调试注意

1. 在初始化 sensor 前，请保证 sensor 各个电源的电压正确
2. 在初始化 sensor 前，请确保 reset，standby 按照 sensor 规定的上电时序控制，否则可能带来很多难以解释的问题
3. 往 sensor 写 I2C 命令前，请保证 MCLK 已经有信号输出，一般来说 MCLK 应该在 24MHz。
4. 如果初始化时，发现 I2C 写命令 fail，则应该检查 sensor 各个电源，power on 的时序，以及 MCLK 是否有信号。也可以尝试将 I2C 降速，或每写一个 I2C 命令后，延时一定时间。
5. 一般来说，初始化后，如果 PCLK 和 VSYNC，HSYNC 有信号输出，则初始化应该成功，如果 PCLK，VSYNC 和 HSYNC 的极性配置正确，则图像接受一般都正确。
6. 若发现接收到的图像有不规则出现的绿横线，一般来说，可能是 PCLK 的驱动能力不足。可以通过 I2C 调大 sensor 输出 PCLK 的驱动能力，一般可以解决问题。
7. 若发现接收到的图像有横向的细彩线出现，可能是 camera 的 avdd 受到干扰。可尝试加小电容滤波。
8. 若两个 sensor 共用到一个 CSI 上，出现切换时黑屏，花屏，有彩线等问题，原因往往出现在切换时，没有将对应的 sensor 的 IO PAD 切换到高阻状态，此时另外一个 sensor 的 IO 输出时，就会被拉住。

2. Linux 系统 CSI 驱动程序

2.1. CSI 驱动文件目录结构

CSI 驱动的位置在 linux-3.0/drivers/media/video/sun4i_csi/
目录结构如下

CopyRight©2011 All Winner Technology, Right Reserved

```
|-- 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 驱动层次结构

CopyRight©2011 All Winner Technology, Right Reserved

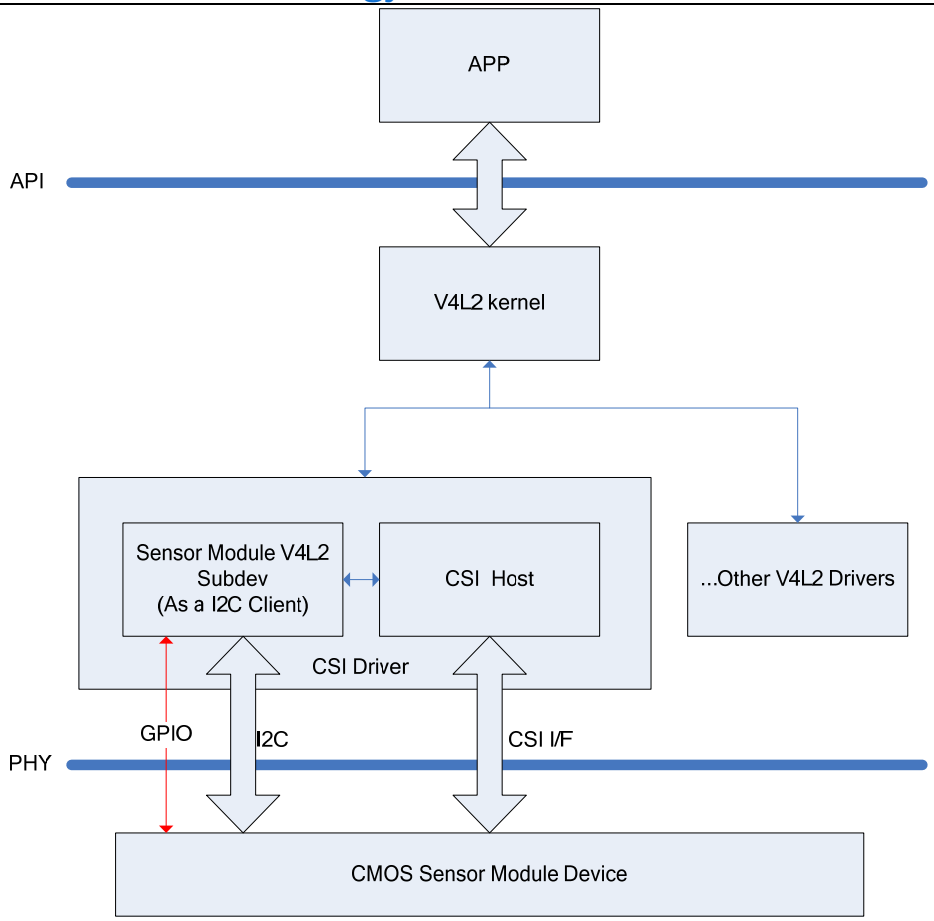


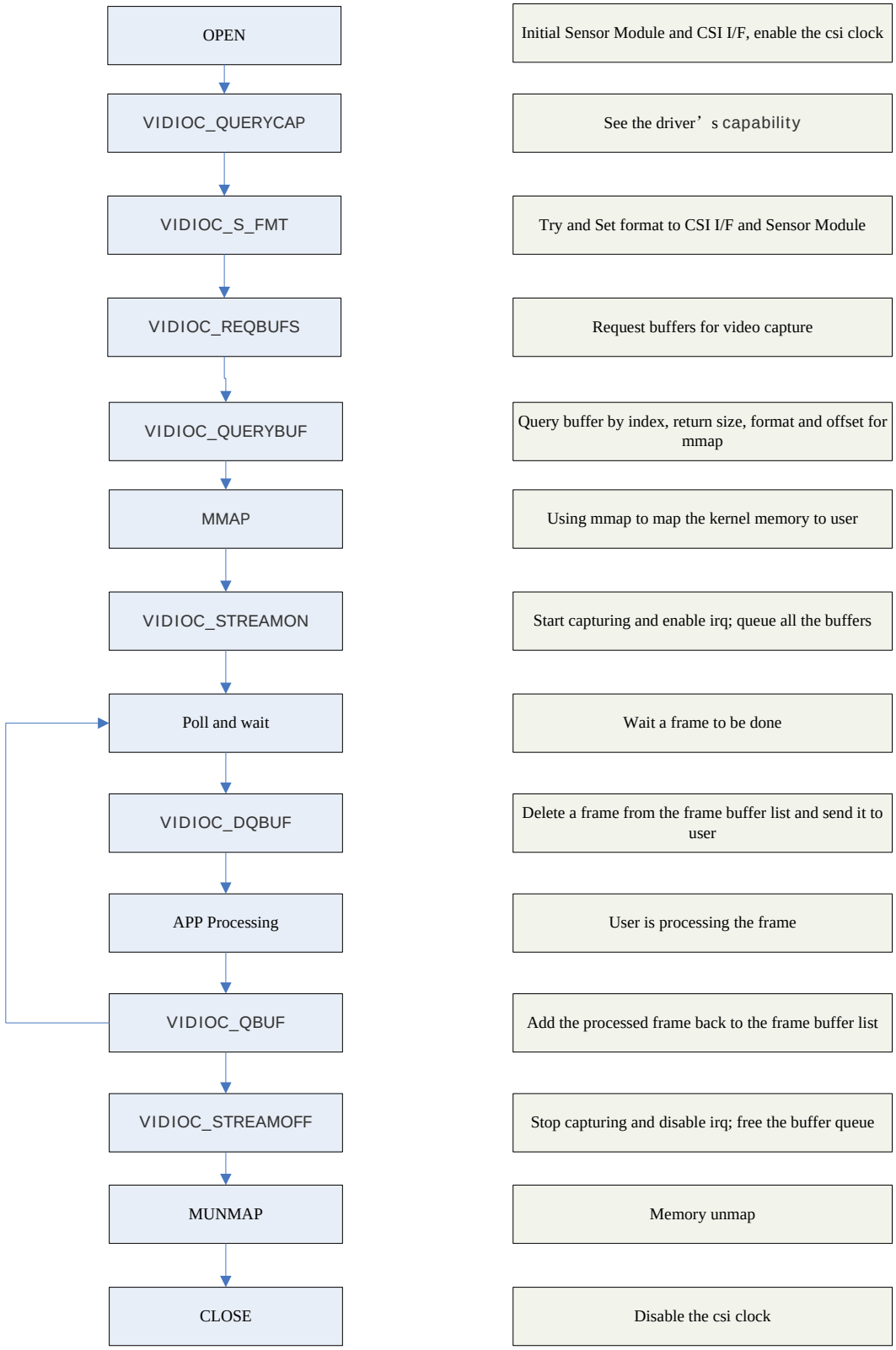
图 CSI 驱动层次结构图

CSI 驱动基于 Linux 的 V4L2 架构设计，满足标准的 V4L2 API 调用方式。由于 CSI 硬件采集的视频缓冲数据要求物理上连续的内存，故采用 video-dma-contig 这种连续的内存申请管理方式。

如图，除 V4L2 Kernel 相关的源代码外，CSI 驱动主体包括 CSI Host 和 Sensor Module V4L2 Subdev。对应的源码为 sun4i_csi_reg.c, sun4i_drv_csi.c 以及 camera 模组源代码(如 gc0308.c,gt2005.c 等)。其中 sun4i_csi_reg.c 是 CSI 硬件的 HAL 层, sun4i_drv_csi.c 是 CSI 驱动的主体，实现 CSI 驱动的初始化，V4L2 API 对接以及 video buffer 的管理等。Camera 模组源代码（如 gc0308.c,gt2005.c 等），实现相应 camera 的初始化，分辨率，图像格式，白平衡，特效，曝光等效果设置，以及电源管理。

2.3. CSI 驱动调用流程

CopyRight©2011 All Winner Technology, Right Reserved



2.4. CSI V4L2 Subdev 接口函数

CopyRight©2011 All Winner Technology, Right Reserved

2.4.1. V4L2 Subdev 函数集

Camera 驱动开发者重点关注的应该为 camera 模组与 CSI 主体之间的接口函数。其分为两大类别，分别为 sensor_core_ops 和 sensor_video_ops。sensor_core_ops 定义 power，standby，效果设置以及 ioctl 扩展；sensor_video_ops 定义设置图像数据格式以及帧率的接口。其具体定义如下：

```
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 函数

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

Main Function：实现三个与 reset 相关的指令

```
switch(val)
    {
        case CSI_SUBDEV_RST_OFF:
            //控制摄像头 IO 或发送 I2C 命令使其 release reset
            break;
        case CSI_SUBDEV_RST_ON:
            //控制摄像头 IO 或发送 I2C 命令使其 hold reset
            break;
        case CSI_SUBDEV_RST_PUL:
            //控制摄像头 IO 或发送 I2C 命令使其经过 reset release→hold→release 的时序
            break;
```

CopyRight©2011 All Winner Technology, Right Reserved

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

2.4.3. sensor_power 函数

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

Main Function: 实现 4 个与 power/standby 相关的指令

```
switch(on)
    {
        case CSI_SUBDEV_STBY_ON:
            //控制 sensor 进入 standby 的时序
            break;
        case CSI_SUBDEV_STBY_OFF:
            //控制 sensor 退出 standby 的时序
            break;
        case CSI_SUBDEV_PWR_ON:
            //控制 sensor 上电的时序
            break;
        case CSI_SUBDEV_PWR_OFF:
            //控制 sensor 关电的时序
            break;
        default:
            return -EINVAL;
    }
```

2.4.4. sensor_init 函数

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

Main Function：实现检测 sensor id，以及对 sensor 初始化。其中，sensor_detect 函数实现读取 sensor 的 id 值，若与目标相同则返回 ok；sensor_default_regs 则是保存 sensor 初始化 I2C 命令的数组，不同 sensor 需要填不同的 I2C 初始化参数。

```
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 函数

Prototype：static int sensor_queryctrl(struct v4l2_subdev *sd, struct v4l2_queryctrl *qc)

Main Function：返回 sensor 支持的各种效果设置，以及对应的设置最大值/最小值/步长。需要注意的是，若实际操作中，sensor 不支持的效果特性，最好在这里注释掉，以免上层进行 query 时，误认为 sensor 支持。一般而言，除了 GAIN 参数外，其他各个设置的最大值/最小值/步长都不能再做修改。基本需要实现的参数设置是 VFLIP，HFLIP（这两个参数涉及到摄像头的成像方向，对应 sys_config1 文件配置中的 csi_hflip 与 csi_vflip），EXPOSURE

CopyRight©2011 All Winner Technology, Right Reserved

（曝光目标值），DO_WHITE_BALANCE（各种白平衡场景），COLORFX（各种特效）。

```
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 函数

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

Main Function：各种 sensor 效果特性的设置。基本实现 VFLIP，HFLIP，EXPOSURE，DO_WHITE_BALANCE，AUTO_WHITE_BALANCE，COLORFX，其他可不实现。

```
switch (ctrl->id) {
    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:
```

CopyRight©2011 All Winner Technology, Right Reserved

```
            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;
```

值得注意的是，sensor_s_ctrl 里面调用到的各个函数，针对不同的 sensor，需要不同的实现。下面说一下各个函数需要实现的内容。

```
sensor_s_brightness(sd, ctrl->value)
sensor_s_contrast(sd, ctrl->value)
sensor_s_saturation(sd, ctrl->value)
sensor_s_exp(sd, ctrl->value)
//实现亮度/对比度/饱和度/曝光目标的调节（ctrl->value 最小值为-4,最大值为 4, step=1）。将
//抽象出来的-4~4 这 9 个等级的值，对应 sensor 具体的寄存器设置。
//brightness 对应实现 sensor_brightness_zero_regs[]等寄存器数组
//contrast 对应实现 sensor_contrast_zero_regs[]等寄存器数组
//saturation 对应实现 sensor_saturation_zero_regs[]等寄存器数组
//exp 对应实现 sensor_ev_zero_regs[]等寄存器数组

sensor_s_hue(sd, ctrl->value)
//调节 sensor 具体的寄存器，设置色调
```

CopyRight©2011 All Winner Technology, Right Reserved

```
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)
//设置 sensor vflip（upsidedown），hflip（mirror），AWB（自动白平衡），AE（自动曝光）的
//enable 位

sensor_s_wb(sd,(enum v4l2_whiteblance) ctrl->value)
//设置各种白平衡场景
//对应实现
//sensor_wb_auto_regs[],sensor_wb_cloud_regs[],sensor_wb_daylight_regs[],
//sensor_wb_incandescence_regs[],sensor_wb_fluorescent_regs[],sensor_wb_tungsten_regs[]
//等寄存器数组

sensor_s_colorfx(sd,(enum v4l2_colorfx) ctrl->value);
//设置各种特效
//对应实现
// 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[]
//等寄存器数组

sensor_s_flash_mode(sd,(enum v4l2_flash_mode) ctrl->value)
//设置闪光灯模式
```

2.4.7. sensor_g_ctrl 函数

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

Main Function：获取各种 sensor 效果特性的设置。基本实现 VFLIP，HFLIP，EXPOSURE，DO_WHITE_BALANCE，AUTO_WHITE_BALANCE，COLORFX，其他可不实现。

```
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:
```

CopyRight©2011 All Winner Technology, Right Reserved

```
            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 函数

Prototype: static long sensor_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)

Main Function: 与 CSI 主体文件 sun4i_drv_csi.c 的一个扩展接口。通过__csi_subdev_info_t 结构体，传递模组的 clock，vsync,hsync 的极性，mclk 的频率，还有 iocfg 属性（用作两个 sensor 接到同一个 CSI 时，对应的模组是 id 0 还是 id 1）。这个接口不能作任何修改。

```
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    ;
```

CopyRight©2011 All Winner Technology, Right Reserved

```
                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 函数

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

Main Function：通过*code 返回 index 对应的图像格式。其中 sensor_formats[]数组保存了所有支持的图像格式。

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

2.4.10. sensor_try_fmt 函数

Prototype：static int sensor_try_fmt(struct v4l2_subdev *sd, struct v4l2_mbus_framefmt *fmt)

Main Function：通过 v4l2_mbus_framefmt 数据结构，上层设定好图像数据格式，size 大小，通过调用 sensor_try_fmt，分别与 sensor_formats[]数组里面的格式作比较，获取到支持的图像格式；与 sensor_win_sizes[]数组的 size 做比较，获取最接近的支持 size 大小。调用该接口后，sensor 的设置不会作任何改变。该接口内容是通用接口，针对不同的 sensor，sensor_try_fmt 不需要作任何改动，只是实现 sensor_formats[]和 sensor_win_sizes[]里面.regs 对应的寄存器数组便可以。

```
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++)
```

CopyRight©2011 All Winner Technology, Right Reserved

```
        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 函数

Prototype：static int sensor_s_fmt(struct v4l2_subdev *sd, struct v4l2_mbus_framefmt *fmt)

Main Function：通过 v4l2_mbus_framefmt 数据结构，上层设定好图像数据格式，size 大小，通过调用 sensor_s_fmt，首先会调用一次 sensor_try_fmt_internal，分别与 sensor_formats[]数组里面的格式作比较，获取到支持的图像格式；与 sensor_win_sizes[]数组的 size 做比较，获取最接近的支持 size 大小，并将对应的指针返回。然后将 sensor 设置到获取到的图像格式

CopyRight©2011 All Winner Technology, Right Reserved

和 size 大小。该接口内容是通用接口，针对不同的 sensor，sensor_s_fmt 不需要作任何改动，只是实现 sensor_formats[]和 sensor_win_sizes[]里面.regs 对应的寄存器数组便可以。

```
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. CSI 驱动中 I2C 访问方式

2.5.1. sensor_write

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

CopyRight©2011 All Winner Technology, Right Reserved

以 gt2005.c 为例子，举例 16 位寄存器地址，8 位寄存器数据的 I2C 寄存器如何访问往 0x0505 寄存器写 0xaa。

```
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)

以 gt2005.c 为例子，举例 16 位寄存器地址，8 位寄存器数据的 I2C 寄存器如何访问读取 0x0505 寄存器的值，并打印出来。

```
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)

该函数实现对一个已定义的 struct regval_list 数组进行 I2C 写操作。

以 gt2005.c 为例子，在 sensor_init 里面，需要调用 sensor_default_regs 进行 I2C 写，从而对 sensor 初始化。

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

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

3. 基于 SUN4I 平台的 Camera 模组移植

3.1. Camera 模组移植关键

CopyRight©2011 All Winner Technology, Right Reserved

一般来说，增加对一个 camera 模组的支持，关键在于在 sun4i_csi/device/目录下，增加 xxx.c 文件，实现 poweron/off, standby on/off, reset 接口，设置好 mclk，vsync，hsync 的极性，设置好 clock 的频率，最后在 sensor_default_regs[]以及 sensor_vga_regs[]填上初始化代码和默认 VGA 分辨率的设置，应该就可以接收到基本的图像。

3.2. Camera 模组移植步骤

以现成的已经调试好的 gt2005.c 为例子，讲述一个新的 camera 模组应该如何移植。

3.2.1. Camera ID 修改

Camera ID 应该从原来的 gt2005 修改为一个特定容易辨认的 ID，而且需要在后续的 sys_config1.fex 里面的配置一致。推荐将 Camera ID 修改为与 c 文件一样的名字，以便辨认。

```
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 输出信号

各个模组输出的 MCLK，VSYNC，HSYNC 的极性有所不同，调试的时候可以根据 sensor 的初始化代码以及相应的 datasheet，或者通过 camera 模组代理商或 sensor 原厂，获取这些信息。gt2005.c 中 ccm_info_con 这个结构体变量就是用来存储这些信息，通过 sensor_ioctl 扩展接口，与 sun4i_drv_csi.c 通信。

```
#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,
```

CopyRight©2011 All Winner Technology, Right Reserved

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

说明：

一般来说，通过设置 MCLK，VREF_POL，HREF_POL，CLK_POL 这几个宏定义就可以实现。MCLK 的单位为 Hz，一般设置为 24MHz 或 12MHz 是准确的频率点，设置为其余的频率是从系统的 PLL 获取，源头频率为 270MHz 或 297MHz，1~16 分频。VREF_POL，HREF_POL 可以设置为 CSI_HIGH 或 CSI_LOW，意义为图像传输有效的区域，对应的 VSYNC 和 HSYNC 信号时高还是低。CLK_POL 可以设置为 CSI_RISING 或 CSI_FALLING，意义为 sensor 出来的 PCLK 是用上升沿还是下降沿采样数据。IO_CFG 是当两个 camera 共用一个 CSI 时，标识对应 camera 的。一般是通过 sun4i_drv_csi.c 调用 sensor_ioctl 这个接口来主动修改其值，用户不用关心。一般填写默认值 0 就可以。

3.2.3. Camera 控制 IO 极性

各个 camera 模组控制 standby，reset 的极性不尽相同，这里用了一些宏定义去表示。

```
#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
```

说明：

CSI_STBY_ON 表示 standby 有效时，对应 camera standby 引脚的状态，大部分 sensor 这个值都是 1。

CSI_STBY_OFF 表示 standby 无效，即 camera 正常工作时，对应 camera standby 引脚的状态。大部分 sensor 这个值都是 0。

CSI_RST_ON 表示 reset 有效时，对应 camera reset 引脚的状态，大部分 sensor 这个值都是 0。

CSI_RST_OFF 表示 reset 无效，即 camera 正常工作时，对应 camera reset 引脚的状态，大部分 sensor 这个值都是 0。

P.S.上述各个引脚状态的定义不确定时，请参考 camera 对应的 datasheet。正确配置这些信息对于调试新的摄像头模组是至关重要的。

3.2.4. Camera I2C 命令长度

各个 camera 模组寄存器和数据的长度不一样，通过以下宏定义修改

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

REG_ADDR_STEP 表示通过 I2C 访问 sensor 寄存器地址的长度，按 byte 的倍数计算

REG_DATA_SETP 表示通过 I2C 访问 sensor 寄存器数据的长度，按 byte 的倍数计算

这个例子里面，表示 gt2005 的寄存器地址是 2 个 byte 的长度，即 16 位的地址。寄存器数据时 1 个 byte 的长度，即 8 为的数据。

定义寄存器数组时，需要按照下列格式书写

CopyRight©2011 All Winner Technology, Right Reserved

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

3.2.5. Camera 初始化寄存器数组

```
static struct regval_list sensor_default_regs[] = {
//填上 sensor 初始化的代码，当驱动被 open 的时候，会设置一遍
}
static struct regval_list sensor_uxga_regs[] = {
//填上 sensor 设置到 uxga 分辨率的代码，每当拍照或视频分辨率更改时，都会设置一遍
}
static struct regval_list sensor_hd720_regs[] = {
//填上 sensor 设置到 720p 分辨率的代码，每当拍照或视频分辨率更改时，都会设置一遍
}
static struct regval_list sensor_svga_regs[] = {
//填上 sensor 设置到 svga 分辨率的代码，每当拍照或视频分辨率更改时，都会设置一遍
}
static struct regval_list sensor_vga_regs[] = {
//填上 sensor 设置到 VGA 分辨率的代码，一般默认预览使用此分辨率，预览时会设置一遍
}

static struct regval_list sensor_fmt_yuv422_yuyv[] = {
//填上 sensor 设置到 yuyv 输出的寄存器代码。上层每次调用 s_fmt，都会设置一遍
};
static struct regval_list sensor_fmt_yuv422_yvyu[] = {
//填上 sensor 设置到 yvyu 输出的寄存器代码。上层每次调用 s_fmt，都会设置一遍
};
static struct regval_list sensor_fmt_yuv422_vyuy[] = {
//填上 sensor 设置到 vyuy 输出的寄存器代码。上层每次调用 s_fmt，都会设置一遍
};
static struct regval_list sensor_fmt_yuv422_uyvy[] = {
//填上 sensor 设置到 uyvy 输出的寄存器代码。上层每次调用 s_fmt，都会设置一遍
};
static struct regval_list sensor_fmt_raw[] = {
//填上 sensor 设置到 raw 输出的寄存器代码。上层每次调用 s_fmt，都会设置一遍
};
```

上述的各种寄存器设置数组，其中 sensor_default_regs[]，sensor_vga_regs[]和 sensor_fmt_yuv422_yuyv[], sensor_fmt_yuv422_yvyu[], sensor_fmt_yuv422_vyuy[], sensor_fmt_yuv422_uyvy[]是必备的数组，否则不能正常接收图像。其他各种分辨率的设置，按照应用的需求添加。另外 sensor_fmt_raw[]可以不需要支持，目前平台并不支持 raw 的格式。

CopyRight©2011 All Winner Technology, Right Reserved

3.2.6. Camera 图像格式映射数组

sensor_formats[]用来保存上层调用格式 mbus_code 与实际对 sensor 设置的对应关系

```
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),
```

CopyRight©2011 All Winner Technology, Right Reserved

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

基本上这个数组都不需要作修改，上层调用的格式和对应的数组名称的映射关系都是固定的

3.2.7. Camera 分辨率映射数组

sensor_win_size []用来保存上层调用分辨率与实际对 sensor 设置的对应关系

```
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 */
```

CopyRight©2011 All Winner Technology, Right Reserved

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

sensor_win_size[]里面的内容根据实际应用的分辨率设置。width 和 height 的设置要与 regs 对应的数据一致。一般来说预览使用 VGA 分辨率，必须要实现。其他不用到的分辨率请注释掉。

3.2.8. Camera Power Standby 控制

sensor_power 接口主要实现 CSI_SUBDEV_STBY_ON，CSI_SUBDEV_STBY_OFF，CSI_SUBDEV_PWR_ON，CSI_SUBDEV_PWR_OFF 等 4 个命令对应的控制时序。

下面提供一个比较标准的控制以作参考。实际的 power/standby 时序，最好参考 sensor 的 datesheet 说明。

```
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
```

CopyRight©2011 All Winner Technology, Right Reserved

```
                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);
```

CopyRight©2011 All Winner Technology, Right Reserved

```
        }
        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
```

CopyRight©2011 All Winner Technology, Right Reserved

```
                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;
}
```

其中，开始对 info->ccm_info->iocfg 的判断，是为了两个 sensor 共用到一个 CSI 接口上，分别使用不同的 IO 控制的判断。以下是造 power on/off,standby on/off 所用到的一些函数。

gpio_set_one_pin_io_status() //设置某个 IO 的 input 或 output 状态
gpio_write_one_pin_value() //设置某个 IO output 高电平或低电平
clk_enable() //enable mclk, mclk 从 csi_mclk pin 放出来
clk_disable() //disable mclk, mclk 变成低电平
regulator_enable() //enable 对应的 pmu ldo
regulator_disable() //disable 对应的 pmu ldo

特别需要注意的是，如果两个 sensor 共用到一个 CSI 接口上，实现 CSI_SUBDEV_STBY_ON 时，需要将对应的 sensor 的 IO 设置为高阻状态。否则，在切换到另外一个 sensor 时，sensor 的输出信号会被拉住，导致拍摄的图像黑屏或花屏。

3.2.9. Camera Reset 控制

sensor_reset 接口主要实现 CSI_SUBDEV_RST_OFF， CSI_SUBDEV_RST_ON，CSI_SUBDEV_RST_PUL 等 3 个命令对应的控制。下面提供一个比较标准的 reset io 的控制

```
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");
```

CopyRight©2011 All Winner Technology, Right Reserved

```
                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;
}
```

同样地，其中，开始对 info->ccm_info->iocfg 的判断，是为了两个 sensor 共用到一个 CSI 接口上，分别使用不同的 IO 控制的判断。

3.2.10. sensor_detect 修改

sensor_detect 主要实现从 camera sensor 读取 id，与目标 id 比较。

以下是 gt2005.c 的例子，从 0x0000 寄存器器读取 id，与 0x51 比较。

最终的实现需要根据实际的 camera 修改。

```
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 修改

sensor_s_hflip 主要实现从 camera sensor 读取 hflip 的 enable 位，根据需要设置 camera

CopyRight©2011 All Winner Technology, Right Reserved

sensor hflip enable 或 disable。

以下为 gt2005.c 的例子，从 0x0101 读取值，根据 hflip 的 enable 或 disable 来设置对应的 bit0。最终的实现需要根据实际 camera 的寄存器去修改。

```
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 修改

sensor_g_hflip 主要实现从 camera sensor 读取 hflip 的 enable 位，返回给上层。

以下为 gt2005.c 的例子，从 0x0101 读取值，根据 bit0 对应的 hflip 返回 enable 或 disable。最终的实现需要根据实际 camera 的寄存器去修改。

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

CopyRight©2011 All Winner Technology, Right Reserved

```
{
	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 修改

sensor_s_vflip 主要实现从 camera sensor 读取 vflip 的 enable 位，根据需要设置 camera sensor vflip enable 或 disable。

以下为 gt2005.c 的例子，从 0x0101 读取值，根据 hflip 的 enable 或 disable 来设置对应的 bit1。最终的实现需要根据实际 camera 的寄存器去修改。

```
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;
```

CopyRight©2011 All Winner Technology, Right Reserved

```
                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 修改

sensor_g_vflip 主要实现从 camera sensor 读取 vflip 的 enable 位，返回给上层。

以下为 gt2005.c 的例子，从 0x0101 读取值，根据 bit1 对应的 vflip 返回 enable 或 disable。

最终的实现需要根据实际 camera 的寄存器去修改。

```
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;
```

CopyRight©2011 All Winner Technology, Right Reserved

```
}
```

3.2.15. sensor_s_autowb 修改

sensor_s_autowb 主要实现读取 camera sensor 的 AWB 自动白平衡的 enable 位，根据上层设置的值，回写 AWB，设置为 enable 或 disable.

以 gt2005.c 为例，从 0x031a 读取值，根据上层设置 bit7 对应的 AWB enable 或 disable。最终的实现需要根据实际 camera 的寄存器去修改。

```
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 修改

CopyRight©2011 All Winner Technology, Right Reserved

sensor_g_autowb 主要实现从 camera sensor 读取 AWB 自动白平衡的 enable 位，返回给上层。

以下为 gt2005.c 的例子，从 0x031a 读取值，根据 bit7 对应的 AWB 返回 enable 或 disable。最终的实现需要根据实际 camera 的寄存器去修改。

```
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. 其他可作的修改

若一些非必须的功能需要实现的话，也可对下列函数进行修改。

sensor_g_autogain //自动增益 enable/disalbe
sensor_s_autogain
sensor_g_autoexp //自动曝光 enable/disable
sensor_s_autoexp
sensor_g_hue //色调调整
sensor_s_hue
sensor_g_gain //增益值
sensor_s_gain
sensor_g_flash_mode //闪光灯模式
sensor_s_flash_mode
sensor_g_parm //目前只做调节帧率用
sensor_s_parm

3.2.18. 一般不进行修改的函数

sensor_read

CopyRight©2011 All Winner Technology, Right Reserved

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 驱动配置说明

3.3.1. linux 层配置

3.3.1.1. 修改 Kconfig

修改 sun4i_csi 目录下的 Kconfig 文件，增加新增 camera 的编译开关
以 gt2005 为例

```
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. 修改 Makefile

修改 sun4i_csi/device 目录下的 Makefile 文件，增加对新增 camera 的编译
以 gt2005 为例，注意编译的开关需要与上面 Kconfig 的一致

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

3.3.1.3. 通过 menuconfig 配置驱动

CopyRight©2011 All Winner Technology, Right Reserved

在 linux-3.0 目录下，敲 make ARCH=arm menuconfig
进去后，按照下列提示选择
Device Drivers --->Multimedia support --->
保证 Video For Linux 选中
然后选择 CSI Driver Config for sun4i--->
进去后，选择相应的摄像头模组，并编译成 module 形式
保存后退出

3.3.2. sys_config1.fex 文件配置

配置项中名字所有后缀带_b 的表示两个摄像头同时连接到一个 CSI 时的副摄像头，不带后缀的为主摄像头

配置项	配置项含义
csi_used =xx	是否使用 csi0
csi_mode=xx	配置 csi 接收 buffer 的模式： 0: 一个 CSI 接收对应一个 buffer 1：两个 CSI 接收内容拼接成一个 buffer
csi_dev_qty=xx	配置 csi 目前连接的器件数量,目前只能配置为 1 或 2
csi_stby_mode=xx	配置 standby 时的电源状态 0：standby 时不关电源 1：standby 时关掉电源
csi_mname=”” csi_mname_b=””	csi0 使用的模组名称，需要与驱动匹配，可以查看驱动目录里面的 readme 目前有 ov7670 ， gc0308 ， gt2005 ， hi704,sp0338,mt9m112,gc0307,mt9m113, mt9d112,hi253,ov5640 可选
csi_twi_id =xx csi_twi_id_b =xx	csi0 使用的 IIC
csi_twi_addr=xx csi_twi_addr_b=xx	csi0 使用的模组的 IIC 地址，可以查看驱动目录里面的 readme
csi_if=xx csi_if_b=xx	配置目前使用模组的接口时序： 0:8bit 数据线，带 Hsync,Vsync 1:16bit 数据线，带 Hsync,Vsync 2:24bit 数据线，带 Hsync,Vsync 3:8bit 数据线,BT656 内嵌同步,单通道 4:8bit 数据线,BT656 内嵌同步,双通道 5:8bit 数据线,BT656 内嵌同步,四通道
csi_vflip=xx csi_vflip_b=xx	配置 csi 接收图像默认情况下，上下颠倒情况： 0：正常 1：上下颠倒
csi_hflip=xx csi_hflip_b=xx	配置 csi 接收图像默认情况下，左右颠倒情况：

CopyRight©2011 All Winner Technology, Right Reserved

	0：正常 1：左右颠倒
csi_iovdd = “” csi_iovdd_b=””	摄像头模组的 IO，Analog 和 Core 供电电源如果从 AXP20 的 LDO3 供电的话，则填“axp20_pll”，如果从 AXP20 的 LDO4 供电的话，则填“axp20_hdmi”,若不是从 AXP20 来的电源，则填“ ”
csi_avdd = “” csi_avdd_b = “”	
csi_dvdd = “” csi_dvdd_b = “”	
csi_flash_pol=xx csi_flash_pol_b=xx	若摄像头有控制闪光灯的 IO，这个参数可以控制闪光灯有效的 IO 极性 0: IO 拉低时，闪光灯亮 1: IO 拉高时，闪光灯亮
csi_pck＝xx	模组送给 csi0 的 clock GPIO 配置
csi_ck=xx	csi0 送给模组的 clock GPIO 配置
csi_hsync=xx	模组送给 csi0 的行同步信号 GPIO 配置
csi_vsync=xx	模组送给 csi0 的帧同步信号 GPIO 配置
csi_d0=xx … csi_d15=xx	模组送给 csi0 的 8bit/16bit 数据 GPIO 配置
csi_reset=xx csi_reset_b=xx	控制模组的 reset 的 GPIO 配置，默认值为 reset 有效（高或低有效需要取决于模组）
csi_power_en=xx csi_power_en_b=xx	控制模组的电源的 GPIO 配置，一般默认值有有效（高）
csi_stby=xx csi_stby_b=xx	控制模组的 standby 的 GPIO 配置，默认值为 standby 有效（高或低有效需要取决于模组）
csi_af_en=xx csi_af_en_b=xx	控制模组的 AF 电源

举例：

一个 CSI 接一个摄像头

```
[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
```

CopyRight©2011 All Winner Technology, Right Reserved

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 =

CopyRight©2011 All Winner Technology, Right Reserved

csi_flash_b =
csi_af_en_b =

一个 CSI 接两个摄像头

[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>

CopyRight©2011 All Winner Technology, Right Reserved

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 层配置

以 gt2005 模组连接到 CSI0 接口为例

1. 确保相关的 ko 文件有拷贝到 android 环境
   相关的 ko 文件包括 video-buf-core.ko, video-dma-contig.ko, sun4i_csi0.ko, gt2005.ko
2. 确保相关的 ko 文件在 android 环境下的安装顺序
   insmod video-buf-core.ko
   insmod video-dma-contig.ko
   insmod gt2005.ko
   insmod sun4i_csi0.ko

   若两个摄像头同时接到一个 CSI，则按如下顺序加载，如 gt2005 和 gc0308 同时挂在 CSI0 上
   insmod video-buf-core.ko
   insmod video-dma-contig.ko
   insmod gt2005.ko
   insmod gc0308.ko
   insmod sun4i_csi0.ko

   总之，原则为 camera 模组驱动需要在 sun4i_csi0.ko 前加载
3. camera.cfg 文件需要配置正确，这个配置请参考其他文档

CopyRight©2011 All Winner Technology, Right Reserved