# 目录

[目录 1](#_Toc483312463)

[Video4Linux2简介 2](#_Toc483312464)

[SOC\_Camera代码解析 3](#_Toc483312465)

[Soc\_camera: device的注册 3](#_Toc483312466)

[注册platform\_device: 3](#_Toc483312467)

[注册platform\_driver: 4](#_Toc483312468)

[注册soc\_camera\_device 5](#_Toc483312469)

[Soc\_camera: driver的注册 6](#_Toc483312470)

[注册platform\_device 6](#_Toc483312471)

[注册platform\_driver 7](#_Toc483312472)

[注册soc\_camera\_host 8](#_Toc483312473)

[Soc\_camera: V4l2\_dev的创建 9](#_Toc483312474)

[Soc\_camera: V4l2\_subdev的创建, 注册i2c\_client设备 10](#_Toc483312475)

[soc\_camera\_probe() 10](#_Toc483312476)

[注册i2c\_client 10](#_Toc483312477)

[注册i2c\_driver 12](#_Toc483312478)

[注册v4l2\_subdev 13](#_Toc483312479)

[Soc\_camera: video的创建 16](#_Toc483312480)

[soc\_camera\_probe() 16](#_Toc483312481)

[video\_dev\_create() 16](#_Toc483312482)

[struct v4l2\_file\_operations soc\_camera\_fops 17](#_Toc483312483)

[struct v4l2\_ioctl\_ops soc\_camera\_ioctl\_ops 17](#_Toc483312484)

[soc\_camera\_video\_start() 18](#_Toc483312485)

[video\_register\_device() 18](#_Toc483312486)

[Soc\_camera: video的注册，创建字符设备 19](#_Toc483312487)

[static struct video\_device \*video\_device[VIDEO\_NUM\_DEVICES]; 19](#_Toc483312488)

[\_\_video\_register\_device() 19](#_Toc483312489)

[struct file\_operations v4l2\_fops，字符设备入口函数 20](#_Toc483312490)

[struct class video\_class 20](#_Toc483312491)

[Soc\_camera: ioctl的流程 20](#_Toc483312492)

[struct file\_operations v4l2\_fops，字符设备入口函数 21](#_Toc483312493)

[struct v4l2\_file\_operations soc\_camera\_fops 22](#_Toc483312494)

[struct v4l2\_ioctl\_info v4l2\_ioctls 24](#_Toc483312495)

[struct v4l2\_ioctl\_ops soc\_camera\_ioctl\_ops 25](#_Toc483312496)

[Soc\_camera: 内存的管理 25](#_Toc483312497)

[Open：初始化videobuf\_queue设备//icd->vb\_vidq 25](#_Toc483312498)

[VIDIOC\_REQBUFS: 初始化videobuf\_buffer设备 //icd->vb\_vidq.bufs[i] 27](#_Toc483312499)

[mmap: 初始化videobuf\_mapping设备 // icd->vb\_vidq.bufs[i]->map 29](#_Toc483312500)

[VIDIOC\_QBUF: list\_add\_tail(&buf->stream, &q->stream); list\_add\_tail(&vb->queue, &cam\_param->capture); 31](#_Toc483312501)

[VIDIOC\_STREAMON: capture\_start() 33](#_Toc483312502)

[IRQ: list\_del\_init(&cam\_param->cur\_frm->queue); 34](#_Toc483312503)

[VIDIOC\_DQBUF: list\_del(&buf->stream); 35](#_Toc483312504)

[VIDIOC\_STREAMOFF: capture\_stop(),list\_del(&q->bufs[i]->queue); 37](#_Toc483312505)

[munmap: kfree(map) 38](#_Toc483312506)

[Close: kfree((struct camera\_param \*)cam\_param) 39](#_Toc483312507)

[QBUF – IRQ – DQBUF 40](#_Toc483312508)

[V4L2 Control 41](#_Toc483312509)

[创建v4l2 control设备 41](#_Toc483312510)

[struct camera\_module\_priv 41](#_Toc483312511)

[camera\_module\_probe() //配置v4l2\_ctrl\_ops 41](#_Toc483312512)

[camera\_module\_init\_ops() 42](#_Toc483312513)

[非菜单式的控制变量 42](#_Toc483312514)

[struct v4l2\_ctl\_cmd\_info v4l2\_ctl\_array 42](#_Toc483312515)

[v4l2\_ctrl\_new\_std() 43](#_Toc483312516)

[菜单式的控制变量 43](#_Toc483312517)

[struct v4l2\_ctl\_cmd\_info\_menu v4l2\_ctl\_array\_menu 43](#_Toc483312518)

[v4l2\_ctrl\_new\_std\_menu () 43](#_Toc483312519)

[v4l2\_ctrl\_ops的实现 44](#_Toc483312520)

[struct v4l2\_ctrl\_ops camera\_module\_ctrl\_ops 44](#_Toc483312521)

[camera\_module\_s\_ctrl() 44](#_Toc483312522)

[camera\_module\_g\_volatile\_ctrl() 44](#_Toc483312523)

[销毁v4l2 control设备 45](#_Toc483312524)

[camera\_module\_remove() 45](#_Toc483312525)

[增加v4l2 control的控制变量方法 45](#_Toc483312526)

[v4l2\_ctrl\_new\_std() 45](#_Toc483312527)

[v4l2\_ctrl\_new\_std\_menu() 45](#_Toc483312528)

[v4l2\_ctrl\_new\_int\_menu() 45](#_Toc483312529)

[v4l2\_ctrl\_new\_std\_menu\_items() 46](#_Toc483312530)

[Video设备的属性配置 46](#_Toc483312531)

[分辨率和数据格式 46](#_Toc483312532)

[VIDIOC\_G\_FMT 47](#_Toc483312533)

[VIDIOC\_ENUM\_FMT 48](#_Toc483312534)

[VIDIOC\_ENUM\_FRAMESIZES 49](#_Toc483312535)

[VIDIOC\_S\_FMT 50](#_Toc483312536)

[曝光配置 50](#_Toc483312537)

[VIDIOC\_QUERYCTRL 51](#_Toc483312538)

[VIDIOC\_G\_CTRL 52](#_Toc483312539)

[VIDIOC\_S\_CTRL 53](#_Toc483312540)

[帧率配置 53](#_Toc483312541)

[VIDIOC\_S\_PARM 54](#_Toc483312542)

[VIDIOC\_G\_PARM 55](#_Toc483312543)

[End 55](#_Toc483312544)

# Video4Linux2简介

Video4Linux2是Linux内核中关于视频设备的内核驱动框架，为上层的访问底层的视频设备提供了统一的接口。

V4L2支持三类设备：视频输入输出设备、VBI设备和radio设备，分别会在/dev目录下产生videoX、radioX和vbiX设备节点。

**Linux系统中视频输入设备主要包括以下四个部分：**

**字符设备驱动程序核心**：V4L2本身就是一个字符设备，具有字符设备所有的特性，暴露接口给用户空间；

**V4L2驱动核心**：主要是构建一个内核中标准视频设备驱动的框架，为视频操作提供统一的接口函数；

**平台V4L2设备驱动**：在V4L2框架下，根据平台自身的特性实现与平台相关的V4L2驱动部分，包括注册video\_device和v4l2\_dev。

**具体的sensor驱动**：主要上电、提供工作时钟、视频图像裁剪、流IO开启等，实现各种设备控制方法供上层调用并注册v4l2\_subdev。

**V4L2的核心源码位于drivers/media/v4l2-core，源码以实现的功能可以划分为四类：**

**核心模块实现**：由v4l2-dev.c实现，主要作用申请字符主设备号、注册class和提供video device注册注销等相关函数；

**V4L2框架**：由v4l2-device.c、v4l2-subdev.c、v4l2-fh.c、v4l2-ctrls.c等文件实现，构建V4L2框架；

**Videobuf管理**：由videobuf2-core.c、videobuf2-dma-contig.c、videobuf2-dma-sg.c、videobuf2-memops.c、videobuf2-vmalloc.c、v4l2-mem2mem.c等文件实现，完成videobuffer的分配、管理和注销。

**Ioctl框架**：由v4l2-ioctl.c文件实现，构建V4L2ioctl的框架。

# SOC\_Camera代码解析

假设，camera设备驱动为i2c类型，在加载camera设备驱动时会同时创建i2c\_driver驱动和soc\_camera\_device驱动。Camera主控驱动加载时会创建soc\_camera\_host驱动。

基于Soc\_camera框架，camera设备驱动作为soc\_camera\_device驱动，会通过soc\_camera\_device\_register()注册到Soc\_camera的devices链表中。Camera主控驱动(host)通过soc\_camera\_host\_register()注册到Soc\_camera的hosts链表中。Soc\_camera的device设备通过iface(即camera的id)和host主控的nr(即camera的bus\_id)进行匹配，若值一样，则触发Soc\_camera框架的soc\_camera\_probe()函数的执行。

soc\_camera\_probe()函数会根据camera设备驱动的board\_info创建i2c\_client设备，并跟camera设备驱动注册的i2c\_driver进行匹配，触发camera设备驱动的camera\_module\_probe()函数。

camera\_module\_probe()函数会创建v4l2\_subdev设备及其相关的ops。并通过v4l2\_device\_register\_subdev()注册到v4l2\_device的subdevs链表中。

soc\_camera\_probe()函数创建完v4l2\_subdev后，创建video\_device设备。video\_device设备中又包含char字符设备及其ops。字符设备作为linux的标准设备提供给用户使用。用户可使用open(), close(), ioctl()通过字符设备的ops入口函数struct file\_operations v4l2\_fops控制v4l2设备。

## Soc\_camera: device的注册

### 注册platform\_device:

drivers/media/i2c/camera/module\_comm/module\_comm.c

#### struct platform\_device asoc\_camera\_device

struct platform\_device asoc\_camera\_device = {

**.name = "soc-camera-pdrv",**

**.id = 0, //soc\_camera\_host.nr //参见camera\_host\_probe()**

.dev = {

.platform\_data = &camera\_module\_link,

.release = camera\_dummy\_release,

},

};

static struct soc\_camera\_link camera\_module\_link = {

**.bus\_id = 0, //soc\_camera\_device.iface //参见soc\_camera\_pdrv\_probe()**

.power = camera\_module\_power,

.reset = camera\_module\_reset,

.board\_info = &asoc\_i2c\_camera,

**.i2c\_adapter\_id = 1, //id编号从0开始 //创建i2c类型的v4l2\_subdev**

.module\_name = CAMERA\_MODULE\_NAME,

.priv = &camera\_module\_info,

};

struct module\_info camera\_module\_info = {

.flags = 0 | SENSOR\_FLAG\_10BIT | SENSOR\_FLAG\_RAW | \

SENSOR\_FLAG\_MIPI | SENSOR\_FLAG\_CHANNEL1 ,

.mipi\_cfg = &mipi\_csi\_setting,

.isp\_cfg = &isp\_setting,

};

#### camera\_module\_init()

static int \_\_init **camera\_module\_init**(void)

{

platform\_device\_register(&asoc\_camera\_device);

i2c\_add\_driver(&camera\_i2c\_driver);

}

module\_init(camera\_module\_init);

### 注册platform\_driver:

drivers/media/platform/soc\_camera/soc\_camera.c

#### struct platform\_driver soc\_camera\_pdrv

static **struct platform\_driver \_\_refdata soc\_camera\_pdrv** = {

.probe = soc\_camera\_pdrv\_probe,

.remove = soc\_camera\_pdrv\_remove,

.driver = {

**.name = "soc-camera-pdrv",**

.owner = THIS\_MODULE,

},

};

module\_platform\_driver(soc\_camera\_pdrv);

#### soc\_camera\_pdrv\_probe()

static int **soc\_camera\_pdrv\_probe**(struct platform\_device \*pdev)

{

struct soc\_camera\_desc \*sdesc = pdev->dev.platform\_data; // platform\_data指向soc\_camera\_link结构体

struct soc\_camera\_subdev\_desc \*ssdd = &sdesc->subdev\_desc;

struct **soc\_camera\_device** \*icd = **devm\_kzalloc**(&pdev->dev, sizeof(\*icd), GFP\_KERNEL);

**icd->iface = sdesc->host\_desc.bus\_id;**

icd->sdesc = sdesc;

icd->pdev = &pdev->dev;

platform\_set\_drvdata(pdev, icd);

icd->user\_width = DEFAULT\_WIDTH; //640

icd->user\_height = DEFAULT\_HEIGHT; //480

return **soc\_camera\_device\_register(icd);**

}

### 注册soc\_camera\_device

drivers/media/platform/soc\_camera/soc\_camera.c

#### static LIST\_HEAD(devices);

#### soc\_camera\_device\_register()

static int **soc\_camera\_device\_register**(struct soc\_camera\_device \*icd)

{

struct soc\_camera\_device \*ix;

for (i = 0; i < 256 && num < 0; i++) {

num = i;

list\_for\_each\_entry(ix, &devices, list) {

if (ix->iface == icd->iface && ix->devnum == i) { num = -1; break; }

}

}

if (num < 0) return -ENOMEM; //已经注册了256个soc\_camera\_device

icd->devnum = num;

icd->use\_count = 0;

icd->host\_priv = NULL;

**list\_add\_tail(&icd->list, &devices);** //将soc\_camera\_device添加到device链表中

}

## Soc\_camera: driver的注册

### 注册platform\_device

#### DTS: isp0: isp@e0268000

**isp0: isp@e0268000** {

**compatible = "actions,owl-isp";**

reg = <0 0xe0268000 0 0x90>,

<0 0xe0240000 0 0x134>;

interrupts = <GIC\_SPI 12 IRQ\_TYPE\_LEVEL\_HIGH>;

power-domains = <&powergate POWER\_DOMAIN\_SENSOR>;

clocks = <&clock CLK\_SI>, <&clock CLK\_CSI>,

<&clock CLK\_SENSOR0>;

clock-names = "si\_clk", "csi", "sensor0";

resets = <&reset RESET\_SI>, <&reset RESET\_CSI>;

reset-names = "si", "csi";

};

### 注册platform\_driver

drivers/media/i2c/camera/host\_comm/owl\_camera.c

#### struct platform\_driver camera\_host\_driver

static struct **platform\_driver camera\_host\_driver** = {

.driver = {

**.name = CAM\_HOST\_NAME,** //"atm9009a-camera-host"

.owner = THIS\_MODULE,

.pm = &camera\_dev\_pm\_ops,

.of\_match\_table = camera\_of\_match,

},

.probe = **camera\_host\_probe**,

.remove = camera\_remove,

};

static const struct of\_device\_id camera\_of\_match[] = { **{.compatible = FDT\_COMPATIBLE,},**  // "actions,owl-isp"};

#### \_\_init camera\_init()

late\_initcall(camera\_init);

static int **\_\_init camera\_init**(void)

{

ret = **platform\_driver\_register**(&camera\_host\_driver);

}

#### camera\_host\_probe() //构建soc\_camera\_host

static int **camera\_host\_probe**(struct platform\_device \*pdev)

{

struct **camera\_dev** \*cam\_dev = **cam\_dev\_alloc**(&pdev->dev, dn); //分配camera\_dev结构体

//配置soc\_camera\_host结构体

struct soc\_camera\_host \*soc\_host = &cam\_dev->soc\_host;

soc\_host->ops = &soc\_camera\_host\_ops;

soc\_host->priv = cam\_dev;

soc\_host->v4l2\_dev.dev = &pdev->dev; //v4l2\_dev的创建

**soc\_host->nr = pdev->id;**

soc\_host->drv\_name = CAM\_HOST\_NAME; // "atm9009a-camera-host"

**soc\_camera\_host\_register(soc\_host)**;

}

#### struct soc\_camera\_host\_ops soc\_camera\_host\_ops

static **struct soc\_camera\_host\_ops soc\_camera\_host\_ops** = {

.owner = THIS\_MODULE,

.add = camera\_add\_device,

.remove = camera\_remove\_device,

.get\_formats = camera\_get\_formats,

.put\_formats = camera\_put\_formats,

.cropcap = camera\_cropcap,

.get\_crop = camera\_get\_crop,

.set\_crop = camera\_set\_crop,

.set\_livecrop = camera\_set\_crop,

.set\_fmt = camera\_set\_fmt,

.try\_fmt = camera\_try\_fmt,

.set\_parm = camera\_set\_parm,

.get\_parm = camera\_get\_parm,

.reqbufs = camera\_reqbufs,

.poll = camera\_poll,

.querycap = camera\_querycap,

.set\_bus\_param = camera\_set\_bus\_param,

.init\_videobuf = camera\_init\_videobuf,

.enum\_framesizes = camera\_enum\_fsizes,

};

### 注册soc\_camera\_host

drivers/media/platform/soc\_camera/soc\_camera.c

**struct soc\_camera\_host** {

**struct v4l2\_device v4l2\_dev;**

struct list\_head list;

unsigned char nr; /\* Host number \*/

void \*priv;

struct soc\_camera\_host\_ops \*ops;

};

#### static LIST\_HEAD(hosts);

#### soc\_camera\_host\_register()

int **soc\_camera\_host\_register**(struct soc\_camera\_host \*ici)

{

**v4l2\_device\_register**(ici->v4l2\_dev.dev, &ici->v4l2\_dev);

**list\_add\_tail(&ici->list, &hosts)**; //将soc\_camera\_host添加到host链表中

**scan\_add\_host**(ici);

}

#### scan\_add\_host()

static void **scan\_add\_host**(struct soc\_camera\_host \*ici)

{

struct soc\_camera\_device \*icd;

list\_for\_each\_entry(icd, &devices, list) {

**if (icd->iface == ici->nr)** { //寻找soc\_camera\_device和soc\_camera\_host之间的匹配设备

icd->parent = ici->v4l2\_dev.dev;

**soc\_camera\_probe(icd);**

}}}

## Soc\_camera: V4l2\_dev的创建

drivers/media/platform/soc\_camera/soc\_camera.c

#### struct v4l2\_device v4l2\_dev： video\_device设备的创建

camera主控驱动会创建camera\_dev结构体。camera\_dev结构体中包含了soc\_camera\_host结构体。而soc\_camera\_host中包含v4l2\_dev结构体。所以在创建camera\_dev结构体时就创建了video\_device设备。

struct soc\_camera\_host {

struct v4l2\_device v4l2\_dev;

}

#### soc\_camera\_host\_register()

int **soc\_camera\_host\_register**(struct soc\_camera\_host \*ici)

{

**v4l2\_device\_register**(ici->v4l2\_dev.dev, &ici->v4l2\_dev);

list\_add\_tail(&ici->list, &hosts); //将soc\_camera\_host添加到host链表中

scan\_add\_host(ici);

}

## Soc\_camera: V4l2\_subdev的创建, 注册i2c\_client设备

先创建i2c\_client设备，创建后会触发xc6130中的i2c\_driver对应的probe()。Probe()中会创建v4l2\_subdev，及相关ops函数。最后调用**v4l2\_device\_register\_subdev()注册。**

drivers/media/platform/soc\_camera/soc\_camera.c

### soc\_camera\_probe()

static int **soc\_camera\_probe**(struct soc\_camera\_device \*icd)

{

v4l2\_ctrl\_handler\_init(&icd->ctrl\_handler, 16);

if (shd->board\_info) { //指向 asoc\_camera\_device.dev.platform\_data.board\_info

**ret = soc\_camera\_init\_i2c(icd, sdesc);**

}

sd = soc\_camera\_to\_subdev(icd);

sd->grp\_id = soc\_camera\_grp\_id(icd);

v4l2\_set\_subdev\_hostdata(sd, icd);

v4l2\_ctrl\_add\_handler(&icd->ctrl\_handler, sd->ctrl\_handler, NULL);

}

### 注册i2c\_client

drivers/media/i2c/camera/module\_comm/module\_comm.c

#### struct i2c\_board\_info asoc\_i2c\_camera

static **struct i2c\_board\_info asoc\_i2c\_camera** = {

//driver的name和device的name匹配时，将触发probe()的执行

I2C\_BOARD\_INFO(**CAMERA\_MODULE\_NAME**, MODULE\_I2C\_REG\_ADDRESS), //XC6130

};

#define I2C\_BOARD\_INFO(dev\_type, dev\_addr) \

.type = dev\_type, .addr = (dev\_addr)

#### soc\_camera\_init\_i2c()

drivers/media/platform/soc\_camera/soc\_camera.c

static int **soc\_camera\_init\_i2c**(struct soc\_camera\_device \*icd, struct soc\_camera\_desc \*sdesc)

{

struct soc\_camera\_host\_desc \*shd = &sdesc->host\_desc;

struct i2c\_adapter \*adap = i2c\_get\_adapter(shd->i2c\_adapter\_id); //i2c\_adapter\_id=1, 表示使用i2c1

**shd->board\_info->platform\_data = &sdesc->subdev\_desc**; //指向asoc\_camera\_device.dev.platform\_data

struct v4l2\_subdev \*subdev = **v4l2\_i2c\_new\_subdev\_board**(&ici->v4l2\_dev, adap, shd->board\_info, NULL);

struct i2c\_client \*client = v4l2\_get\_subdevdata(subdev);

**icd->control = &client->dev;**

}

#### v4l2\_i2c\_new\_subdev\_board()

struct v4l2\_subdev \***v4l2\_i2c\_new\_subdev\_board**(struct v4l2\_device \*v4l2\_dev,

struct i2c\_adapter \*adapter, struct i2c\_board\_info \*info, const unsigned short \*probe\_addrs)

{

struct i2c\_client \*client = **i2c\_new\_device**(adapter, info);

struct v4l2\_subdev \*sd = i2c\_get\_clientdata(client);

**v4l2\_device\_register\_subdev**(v4l2\_dev, sd);

}

#### i2c\_new\_device()

struct i2c\_client \***i2c\_new\_device**(struct i2c\_adapter \*adap, struct i2c\_board\_info const \*info)

{

struct **i2c\_client** \*client = **kzalloc**(sizeof \*client, GFP\_KERNEL);

client->adapter = adap;

client->dev.platform\_data = info->platform\_data;

client->flags = info->flags;

client->addr = info->addr;

client->irq = info->irq;

strlcpy(client->name, info->type, sizeof(client->name));

i2c\_check\_client\_addr\_validity(client); //检查i2c地址是否可用

i2c\_check\_addr\_busy(adap, client->addr); //检查地址是否已经被占用

client->dev.parent = &client->adapter->dev;

client->dev.bus = &i2c\_bus\_type;

client->dev.type = &i2c\_client\_type;

client->dev.of\_node = info->of\_node;

ACPI\_HANDLE\_SET(&client->dev, info->acpi\_node.handle);

dev\_set\_name(&client->dev, "%d-%04x", i2c\_adapter\_id(adap),

client->addr | ((client->flags & I2C\_CLIENT\_TEN) ? 0xa000 : 0));

//注册i2c client，会触发对应的i2c driver的probe函数。

**device\_register**(&client->dev);

}

### 注册i2c\_driver

#### struct i2c\_driver camera\_i2c\_driver

drivers/media/i2c/camera/module\_comm/module\_comm.c

static **struct i2c\_driver camera\_i2c\_driver** = {

.driver = {

**.name = CAMERA\_MODULE\_NAME,** //XC6130

},

**.probe = camera\_module\_probe,**

.suspend = camera\_module\_suspend,

.resume = camera\_module\_resume,

.remove = camera\_module\_remove,

.id\_table = camera\_module\_id,

};

#### camera\_module\_init ()

static int \_\_init camera\_module\_init(void)

{

platform\_device\_register(&asoc\_camera\_device);

**i2c\_add\_driver(&camera\_i2c\_driver);**

}

module\_init(camera\_module\_init);

#### camera\_module\_probe()

static int **camera\_module\_probe**(struct i2c\_client \*client, const struct i2c\_device\_id \*did)

{

struct **camera\_module\_priv** \*priv = **devm\_kzalloc**(&client->dev, sizeof(\*priv), GFP\_KERNEL);

camera\_module\_priv\_init(priv);

priv->info = desc->drv\_priv;

**v4l2\_i2c\_subdev\_init(&priv->subdev, client, &module\_subdev\_ops);**

**camera\_module\_init\_ops**(&priv->hdl, &camera\_module\_ctrl\_ops);

priv->subdev.ctrl\_handler = &priv->hdl;

priv->hdl.error = 0;

subdev = i2c\_get\_clientdata(client);

camera\_client = client;

priv->pcv\_mode = ACTS\_ISP\_PREVIEW\_MODE;

**v4l2\_ctrl**\_handler\_setup(&priv->hdl); //这个函数会调用camera\_module\_ctrl\_ops.s\_ctrl

}

### 注册v4l2\_subdev

drivers/media/i2c/camera/module\_comm/module\_comm.c

#### v4l2\_i2c\_subdev\_init()

void **v4l2\_i2c\_subdev\_init**(struct v4l2\_subdev \*sd, struct i2c\_client \*client, const struct v4l2\_subdev\_ops \*ops)

{

v4l2\_subdev\_init(sd, ops); //sd->ops = ops;

sd->flags |= V4L2\_SUBDEV\_FL\_IS\_I2C;

sd->owner = client->driver->driver.owner;

v4l2\_set\_subdevdata(sd, client);

i2c\_set\_clientdata(client, sd);

snprintf(sd->name, sizeof(sd->name), "%s %d-%04x", client->driver->driver.name,

i2c\_adapter\_id(client->adapter), client->addr);

}

#### struct v4l2\_subdev\_ops module\_subdev\_ops

static **struct v4l2\_subdev\_ops module\_subdev\_ops** = {

.core = &camera\_module\_subdev\_core\_ops,

.video = &camera\_module\_subdev\_video\_ops,

.sensor = &module\_subdev\_sensor\_ops,

};

#### struct v4l2\_subdev\_core\_ops camera\_module\_subdev\_core\_ops

static **struct v4l2\_subdev\_core\_ops camera\_module\_subdev\_core\_ops** = {

.g\_chip\_ident = camera\_module\_g\_chip\_ident,

.ioctl = camera\_module\_ioctrl,

.s\_power = camera\_module\_s\_power,

};

#### struct v4l2\_subdev\_video\_ops camera\_module\_subdev\_video\_ops

static **struct v4l2\_subdev\_video\_ops camera\_module\_subdev\_video\_ops** = {

.s\_stream = camera\_module\_s\_stream,

.cropcap = camera\_module\_cropcap,

.g\_crop = camera\_module\_g\_crop,

.g\_parm = camera\_module\_g\_parm,

.s\_parm = camera\_module\_s\_parm,

.enum\_framesizes = camera\_module\_enum\_framesizes,

.enum\_frameintervals = camera\_module\_enum\_frameintervals,

.enum\_mbus\_fmt = camera\_module\_enum\_fmt,

.g\_mbus\_fmt = camera\_module\_g\_fmt,

.try\_mbus\_fmt = camera\_module\_try\_fmt,

.s\_mbus\_fmt = camera\_module\_s\_fmt,

.g\_mbus\_config = camera\_module\_g\_mbus\_config,

.s\_mbus\_config = camera\_module\_s\_mbus\_config,

};

#### struct v4l2\_subdev\_sensor\_ops module\_subdev\_sensor\_ops

static **struct v4l2\_subdev\_sensor\_ops module\_subdev\_sensor\_ops** = {

.g\_skip\_frames = camera\_module\_g\_skip\_frames,

};

#### camera\_module\_init\_ops()

static void **camera\_module\_init\_ops**(struct v4l2\_ctrl\_handler \*hdl, const struct v4l2\_ctrl\_ops \*ops)

{

v4l2\_ctrl\_handler\_init(hdl, cmd\_array\_size + cmd\_menu\_array\_size);

for (i = 0; i < cmd\_array\_size; i++) {

const struct v4l2\_ctl\_cmd\_info \*pctl = v4l2\_ctl\_array + i;

v4l2\_ctrl\_new\_std(hdl, ops, pctl->id, pctl->min, pctl->max, pctl->step, pctl->def);

}

for (i = 0; i < cmd\_menu\_array\_size; i++) {

const struct v4l2\_ctl\_cmd\_info\_menu \*pmenu = v4l2\_ctl\_array\_menu + i;

v4l2\_ctrl\_new\_std\_menu(hdl, ops, pmenu->id, pmenu->max, pmenu->mask, pmenu->def);

}

}

#### struct v4l2\_ctrl\_ops camera\_module\_ctrl\_ops

static const **struct v4l2\_ctrl\_ops camera\_module\_ctrl\_ops** = {

.g\_volatile\_ctrl = camera\_module\_g\_volatile\_ctrl,

.s\_ctrl = camera\_module\_s\_ctrl,

};

#### v4l2\_i2c\_new\_subdev\_board()

struct v4l2\_subdev \***v4l2\_i2c\_new\_subdev\_board**(struct v4l2\_device \*v4l2\_dev,

struct i2c\_adapter \*adapter, struct i2c\_board\_info \*info, const unsigned short \*probe\_addrs)

{

struct i2c\_client \*client = **i2c\_new\_device**(adapter, info);

struct v4l2\_subdev \*sd = i2c\_get\_clientdata(client);

**v4l2\_device\_register\_subdev**(v4l2\_dev, sd);

}

#### v4l2\_device\_register\_subdev()

int **v4l2\_device\_register\_subdev**(struct v4l2\_device \*v4l2\_dev, struct v4l2\_subdev \*sd)

{

**list\_add\_tail(&sd->list, &v4l2\_dev->subdevs);**

}

## Soc\_camera: video的创建

drivers/media/platform/soc\_camera/soc\_camera.c

### soc\_camera\_probe()

static int **soc\_camera\_probe**(struct soc\_camera\_device \*icd)

{

ici->ops->add(icd);

**video\_dev\_create(icd);**

...//v4l2\_subdev代码参见soc\_camera: 创建v4l2\_subdev...

soc\_camera\_init\_user\_formats(icd);

icd->field = V4L2\_FIELD\_ANY;

**soc\_camera\_video\_start(icd);**

if (!v4l2\_subdev\_call(sd, video, g\_mbus\_fmt, &mf)) {

icd->user\_width = mf.width;

icd->user\_height = mf.height;

icd->colorspace = mf.colorspace;

icd->field = mf.field;

}

ici->ops->remove(icd);

}

### video\_dev\_create()

static int **video\_dev\_create**(struct soc\_camera\_device \*icd)

{

struct video\_device \*vdev = video\_device\_alloc();

strlcpy(vdev->name, ici->drv\_name, sizeof(vdev->name));

vdev->parent = icd->pdev;

vdev->current\_norm = V4L2\_STD\_UNKNOWN;

**vdev->fops = &soc\_camera\_fops; //ioctl**

vdev->ioctl\_ops = &soc\_camera\_ioctl\_ops;

vdev->release = video\_device\_release;

vdev->tvnorms = V4L2\_STD\_UNKNOWN;

vdev->ctrl\_handler = &icd->ctrl\_handler;

vdev->lock = &ici->host\_lock;

icd->vdev = vdev;

}

### struct v4l2\_file\_operations soc\_camera\_fops

static **struct v4l2\_file\_operations soc\_camera\_fops** = {

.owner = THIS\_MODULE,

.open = soc\_camera\_open,

.release = soc\_camera\_close,

.unlocked\_ioctl = video\_ioctl2,

.read = soc\_camera\_read,

.mmap = soc\_camera\_mmap,

.poll = soc\_camera\_poll,

};

### struct v4l2\_ioctl\_ops soc\_camera\_ioctl\_ops

static const **struct v4l2\_ioctl\_ops soc\_camera\_ioctl\_ops** = {

.vidioc\_querycap = soc\_camera\_querycap,

.vidioc\_try\_fmt\_vid\_cap = soc\_camera\_try\_fmt\_vid\_cap,

.vidioc\_g\_fmt\_vid\_cap = soc\_camera\_g\_fmt\_vid\_cap,

.vidioc\_s\_fmt\_vid\_cap = soc\_camera\_s\_fmt\_vid\_cap,

.vidioc\_enum\_fmt\_vid\_cap = soc\_camera\_enum\_fmt\_vid\_cap,

.vidioc\_enum\_input = soc\_camera\_enum\_input,

.vidioc\_g\_input = soc\_camera\_g\_input,

.vidioc\_s\_input = soc\_camera\_s\_input,

.vidioc\_s\_std = soc\_camera\_s\_std,

.vidioc\_g\_std = soc\_camera\_g\_std,

.vidioc\_enum\_framesizes = soc\_camera\_enum\_framesizes,

.vidioc\_reqbufs = soc\_camera\_reqbufs,

.vidioc\_querybuf = soc\_camera\_querybuf,

.vidioc\_qbuf = soc\_camera\_qbuf,

.vidioc\_dqbuf = soc\_camera\_dqbuf,

.vidioc\_create\_bufs = soc\_camera\_create\_bufs,

.vidioc\_prepare\_buf = soc\_camera\_prepare\_buf,

.vidioc\_streamon = soc\_camera\_streamon,

.vidioc\_streamoff = soc\_camera\_streamoff,

.vidioc\_cropcap = soc\_camera\_cropcap,

.vidioc\_g\_crop = soc\_camera\_g\_crop,

.vidioc\_s\_crop = soc\_camera\_s\_crop,

.vidioc\_g\_selection = soc\_camera\_g\_selection,

.vidioc\_s\_selection = soc\_camera\_s\_selection,

.vidioc\_g\_parm = soc\_camera\_g\_parm,

.vidioc\_s\_parm = soc\_camera\_s\_parm,

.vidioc\_g\_chip\_ident = soc\_camera\_g\_chip\_ident,

};

### soc\_camera\_video\_start()

static int s**oc\_camera\_video\_start**(struct soc\_camera\_device \*icd)

{

const struct device\_type \*type = icd->vdev->dev.type;

**video\_register\_device**(icd->vdev, VFL\_TYPE\_GRABBER, -1);

icd->vdev->dev.type = type;

}

### video\_register\_device()

static inline int \_\_must\_check **video\_register\_device**(struct video\_device \*vdev, int type, int nr)

{

return \_\_video\_register\_device(vdev, type, nr, 1, vdev->fops->owner);

}

## Soc\_camera: video的注册，创建字符设备

### static struct video\_device \*video\_device[VIDEO\_NUM\_DEVICES];

### \_\_video\_register\_device()

int **\_\_video\_register\_device**(struct video\_device \*vdev, int type, int nr, int warn\_if\_nr\_in\_use, struct module \*owner)

{

vdev->vfl\_type = type; //type = VFL\_TYPE\_GRABBER

vdev->parent = vdev->v4l2\_dev->dev;

vdev->ctrl\_handler = vdev->v4l2\_dev->ctrl\_handler;

vdev->prio = &vdev->v4l2\_dev->prio;

//寻找可用节点，节点范围[0, 256]

nr = devnode\_find(vdev, nr == -1 ? 0 : nr, minor\_cnt); //minor\_cnt=VIDEO\_NUM\_DEVICES

if (nr == minor\_cnt) nr = devnode\_find(vdev, 0, minor\_cnt);

vdev->minor = i + minor\_offset; //minor\_offset=0

vdev->num = nr;

vdev->index = get\_index(vdev);

//注册字符设备

vdev->cdev = **cdev\_alloc**();

**vdev->cdev->ops = &v4l2\_fops;**

vdev->cdev->owner = owner;

ret = **cdev\_add(**vdev->cdev, MKDEV(VIDEO\_MAJOR, vdev->minor), 1);

//配置dev.class

**vdev->dev.class = &video\_class;**

vdev->dev.devt = MKDEV(VIDEO\_MAJOR, vdev->minor);

vdev->dev.parent = vdev->parent;

//注册video设备

switch (type) {

**case VFL\_TYPE\_GRABBER: name\_base = "video"; break;**

case VFL\_TYPE\_VBI: name\_base = "vbi"; break;

case VFL\_TYPE\_RADIO: name\_base = "radio"; break;

case VFL\_TYPE\_SUBDEV: name\_base = "v4l-subdev"; break;

}

dev\_set\_name(&vdev->dev, "%s%d", name\_base, vdev->num);

ret = **device\_register(&vdev->dev);**

vdev->dev.release = v4l2\_device\_release;

**video\_device[vdev->minor] = vdev;**  //全局变量

}

### struct file\_operations v4l2\_fops，字符设备入口函数

static const **struct file\_operations v4l2\_fops** = {

.owner = THIS\_MODULE,

.read = v4l2\_read,

.write = v4l2\_write,

.open = v4l2\_open,

.get\_unmapped\_area = v4l2\_get\_unmapped\_area,

.mmap = v4l2\_mmap,

.unlocked\_ioctl = v4l2\_ioctl,

#ifdef CONFIG\_COMPAT

.compat\_ioctl = v4l2\_compat\_ioctl32,

#endif

.release = v4l2\_release,

.poll = v4l2\_poll,

.llseek = no\_llseek,

};

### struct class video\_class

static struct class video\_class = {

**.name = VIDEO\_NAME, // "video4linux"**

.dev\_attrs = video\_device\_attrs,

};

## Soc\_camera: ioctl的流程

drivers/media/v4l2-core/v4l2-compat-ioctl32.c

### struct file\_operations v4l2\_fops，字符设备入口函数

static const **struct file\_operations v4l2\_fops** = {

.owner = THIS\_MODULE,

.read = v4l2\_read,

.write = v4l2\_write,

.open = v4l2\_open,

.get\_unmapped\_area = v4l2\_get\_unmapped\_area,

.mmap = v4l2\_mmap,

.unlocked\_ioctl = v4l2\_ioctl,

#ifdef CONFIG\_COMPAT // CONFIG\_COMPAT = y

**.compat\_ioctl = v4l2\_compat\_ioctl32,**

#endif

.release = v4l2\_release,

.poll = v4l2\_poll,

.llseek = no\_llseek,

};

#### .compat\_ioctl = v4l2\_compat\_ioctl32

long v4l2\_compat\_ioctl32(struct file \*file, unsigned int cmd, unsigned long arg)

{

switch (cmd) {

case XXX: //兼容bit32操作系统命令

**ret = do\_video\_ioctl(file, cmd, arg);**

break;

default: //bit64位操作系统命令

if (vdev->fops->compat\_ioctl32)

ret = vdev->fops->compat\_ioctl32(file, cmd, arg);

break;

}

}

static long do\_video\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg)

{

int compatible\_arg = 1;

if (compatible\_arg)

**err = native\_ioctl(file, cmd, (unsigned long)up);**

}

static long native\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg)

{

if (file->f\_op->unlocked\_ioctl)

ret = **file->f\_op->unlocked\_ioctl**(file, cmd, arg);

}

#### .unlocked\_ioctl = v4l2\_ioctl

drivers/media/v4l2-core/v4l2-dev.c

**vdev->fops = &soc\_camera\_fops; //ioctl**

static long **v4l2\_ioctl**(struct file \*filp, unsigned int cmd, unsigned long arg)

{

struct video\_device \*vdev = video\_devdata(filp);

if (vdev->fops->unlocked\_ioctl) {

if (video\_is\_registered(vdev))

ret = **vdev->fops->unlocked\_ioctl**(filp, cmd, arg);

} else if (vdev->fops->ioctl) {

if (video\_is\_registered(vdev))

ret = vdev->fops->ioctl(filp, cmd, arg);

}

return ret;

}

### struct v4l2\_file\_operations soc\_camera\_fops

static **struct v4l2\_file\_operations soc\_camera\_fops** = {

.owner = THIS\_MODULE,

.open = soc\_camera\_open,

.release = soc\_camera\_close,

**.unlocked\_ioctl = video\_ioctl2,**

.read = soc\_camera\_read,

.mmap = soc\_camera\_mmap,

.poll = soc\_camera\_poll,

};

#### .unlocked\_ioctl = video\_ioctl2

long video\_ioctl2(struct file \*file, unsigned int cmd, unsigned long arg)

{

return **video\_usercopy**(file, cmd, arg, **\_\_video\_do\_ioct**l);

}

long video\_usercopy(struct file \*file, unsigned int cmd, unsigned long arg, v4l2\_kioctl func)

{

err = **func(file, cmd, parg);**

}

static long \_\_video\_do\_ioctl(struct file \*file, unsigned int cmd, void \*arg)

{

if (**v4l2\_is\_known\_ioctl(cmd**)) {

info = &**v4l2\_ioctls**[\_IOC\_NR(cmd)];

}

if (info->flags & **INFO\_FL\_STD**) {

typedef int (\*vidioc\_op)(struct file \*file, void \*fh, void \*p);

const void \*p = vfd->ioctl\_ops;

const vidioc\_op **\*vidioc = p + info->u.offset;**

ret = **(\*vidioc)**(file, fh, arg);

} else if (info->flags & **INFO\_FL\_FUNC**) {

ret = **info->u.func**(ops, file, fh, arg);

}

}

### struct v4l2\_ioctl\_info v4l2\_ioctls

static struct **v4l2\_ioctl\_info** **v4l2\_ioctls**[] = {

IOCTL\_INFO\_**FNC**(VIDIOC\_QUERYCAP, **v4l\_querycap**, v4l\_print\_querycap, 0),

IOCTL\_INFO\_FNC(VIDIOC\_G\_FMT, **v4l\_g\_fmt,** v4l\_print\_format, INFO\_FL\_CLEAR(v4l2\_format, type)),

IOCTL\_INFO\_FNC(VIDIOC\_S\_FMT, **v4l\_s\_fmt,** v4l\_print\_format, INFO\_FL\_PRIO),

**IOCTL\_INFO\_FNC(VIDIOC\_REQBUFS, v4l\_reqbufs**, v4l\_print\_requestbuffers, INFO\_FL\_PRIO | INFO\_FL\_QUEUE),

IOCTL\_INFO\_FNC(VIDIOC\_QUERYBUF, **v4l\_querybuf,** v4l\_print\_buffer, INFO\_FL\_QUEUE | INFO\_FL\_CLEAR(v4l2\_buffer, length)),

IOCTL\_INFO\_**STD**(VIDIOC\_G\_FBUF, **vidioc\_g\_fbuf,** v4l\_print\_framebuffer, 0),

IOCTL\_INFO\_STD(VIDIOC\_S\_FBUF, **vidioc\_s\_fbuf**, v4l\_print\_framebuffer, INFO\_FL\_PRIO),

...

}

#define **IOCTL\_INFO\_STD**(\_ioctl, \_vidioc, \_debug, \_flags) \

[\_IOC\_NR(\_ioctl)] = { \

.ioctl = \_ioctl, \

.flags = \_flags | **INFO\_FL\_STD**, \

.name = #\_ioctl, \

.u.offset = offsetof(struct v4l2\_ioctl\_ops, \_vidioc), \

.debug = \_debug, \

}

#define **IOCTL\_INFO\_FNC**(\_ioctl, \_func, \_debug, \_flags) \

[\_IOC\_NR(\_ioctl)] = { \

.ioctl = \_ioctl, \

.flags = \_flags | **INFO\_FL\_FUNC**, \

.name = #\_ioctl, \

.u.func = \_func, \

.debug = \_debug, \

}

#### IOCTL\_INFO\_FNC(VIDIOC\_REQBUFS, v4l\_reqbufs

static int **v4l\_reqbufs**(const struct **v4l2\_ioctl\_ops** \*ops, struct file \*file, void \*fh, void \*arg)

{

return **ops->vidioc\_reqbufs**(file, fh, p);

}

### struct v4l2\_ioctl\_ops soc\_camera\_ioctl\_ops

static const **struct v4l2\_ioctl\_ops soc\_camera\_ioctl\_ops** = {

.vidioc\_querycap = soc\_camera\_querycap,

.vidioc\_try\_fmt\_vid\_cap = soc\_camera\_try\_fmt\_vid\_cap,

.vidioc\_g\_fmt\_vid\_cap = soc\_camera\_g\_fmt\_vid\_cap,

.vidioc\_s\_fmt\_vid\_cap = soc\_camera\_s\_fmt\_vid\_cap,

.vidioc\_enum\_fmt\_vid\_cap = soc\_camera\_enum\_fmt\_vid\_cap,

.vidioc\_enum\_input = soc\_camera\_enum\_input,

.vidioc\_g\_input = soc\_camera\_g\_input,

.vidioc\_s\_input = soc\_camera\_s\_input,

.vidioc\_s\_std = soc\_camera\_s\_std,

.vidioc\_g\_std = soc\_camera\_g\_std,

.vidioc\_enum\_framesizes = soc\_camera\_enum\_framesizes,

**.vidioc\_reqbufs = soc\_camera\_reqbufs,**

.vidioc\_querybuf = soc\_camera\_querybuf,

.vidioc\_qbuf = soc\_camera\_qbuf,

.vidioc\_dqbuf = soc\_camera\_dqbuf,

.vidioc\_create\_bufs = soc\_camera\_create\_bufs,

.vidioc\_prepare\_buf = soc\_camera\_prepare\_buf,

.vidioc\_streamon = soc\_camera\_streamon,

.vidioc\_streamoff = soc\_camera\_streamoff,

.vidioc\_cropcap = soc\_camera\_cropcap,

.vidioc\_g\_crop = soc\_camera\_g\_crop,

.vidioc\_s\_crop = soc\_camera\_s\_crop,

.vidioc\_g\_selection = soc\_camera\_g\_selection,

.vidioc\_s\_selection = soc\_camera\_s\_selection,

.vidioc\_g\_parm = soc\_camera\_g\_parm,

.vidioc\_s\_parm = soc\_camera\_s\_parm,

.vidioc\_g\_chip\_ident = soc\_camera\_g\_chip\_ident,

};

## Soc\_camera: 内存的管理

### Open：初始化videobuf\_queue设备//icd->vb\_vidq

#### Sample code

#define CAMERA\_DEVICE **"/dev/video0"**

int fd = **open**(**CAMERA\_DEVICE**, O\_RDWR, 0);

#### soc\_camera\_open():

static int **soc\_camera\_open**(struct file \*file)

{

if (ici->ops->init\_videobuf) { **ici->ops->init\_videobuf**(&icd->vb\_vidq, icd); }

}

static **struct soc\_camera\_host\_ops soc\_camera\_host\_ops** = {

**.init\_videobuf = camera\_init\_videobuf,**

};

#### camera\_init\_videobuf()

static void **camera\_init\_videobuf**(struct videobuf\_queue \*q, struct soc\_camera\_device \*icd)

{

**videobuf\_queue\_dma\_contig\_init**(q, **&camera\_videobuf\_ops**, dev, &cam\_dev->lock,

V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE, V4L2\_FIELD\_NONE, **sizeof(struct camera\_buffer),** **icd,** NULL);

}

void **videobuf\_queue\_dma\_contig\_init**(struct videobuf\_queue \*q, const struct **videobuf\_queue\_ops** \*ops,

struct device \*dev, spinlock\_t \*irqlock, enum v4l2\_buf\_type type, enum v4l2\_field field,

unsigned int **msize**, void \***priv,** struct mutex \*ext\_lock)

{

**videobuf\_queue\_core\_init(**q, **ops**, dev, irqlock, type, field, msize, **priv**, **&qops**, ext\_lock);

}

void **videobuf\_queue\_core\_init**(struct videobuf\_queue \*q,const struct **videobuf\_queue\_ops** \*ops,

struct device \*dev,spinlock\_t \*irqlock,enum v4l2\_buf\_type type,enum v4l2\_field field,

unsigned int **msize**,void \***priv,**struct **videobuf\_qtype**\_ops \*int\_ops,struct mutex \*ext\_lock)

{

q->type = type; //V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE

q->msize = msize; // sizeof(struct camera\_buffer)

**q->ops = ops;** //camera\_videobuf\_ops

**q->priv\_data = priv;** //soc\_camera\_device

**q->int\_ops = int\_ops;** //qops

INIT\_LIST\_HEAD(&q->stream);

}

#### struct videobuf\_queue\_ops camera\_videobuf\_ops

static struct **videobuf\_queue\_ops** **camera\_videobuf\_ops** = {

.buf\_setup = camera\_videobuf\_setup,

.buf\_prepare = camera\_videobuf\_prepare,

.buf\_queue = camera\_videobuf\_queue,

.buf\_release = camera\_videobuf\_release,

};

#### struct videobuf\_qtype\_ops qops

static struct **videobuf\_qtype\_ops** **qops** = {

.magic = MAGIC\_QTYPE\_OPS,

.alloc\_vb = \_\_videobuf\_alloc,

.iolock = \_\_videobuf\_iolock,

**.mmap\_mapper = \_\_videobuf\_mmap\_mapper,**

.vaddr = \_\_videobuf\_to\_vaddr,

};

### VIDIOC\_REQBUFS: 初始化videobuf\_buffer设备 //icd->vb\_vidq.bufs[i]

#### Sample code

struct v4l2\_requestbuffers reqbuf;

**reqbuf.count = BUFFER\_COUNT; //最终count由kernel可用dma空间决定**

reqbuf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

reqbuf.memory = **V4L2\_MEMORY\_MMAP;**

ioctl(fd , **VIDIOC\_REQBUFS**, &reqbuf);

#### struct v4l2\_requestbuffers

**struct v4l2\_requestbuffers** {

\_\_u32 count;

\_\_u32 type; /\* enum v4l2\_buf\_type \*/

\_\_u32 memory; /\* enum v4l2\_memory \*/

\_\_u32 reserved[2];

};

#### soc\_camera\_reqbufs()

static int **soc\_camera\_reqbufs**(struct file \*file, void \*priv, struct v4l2\_requestbuffers \*p)

{

if (ici->ops->init\_videobuf) {

ret = **videobuf\_reqbufs**(&icd->vb\_vidq, p);

ret = ici->ops->reqbufs(icd, p);

}

}

int **videobuf\_reqbufs**(struct videobuf\_queue \*q, struct v4l2\_requestbuffers \*req)

{

count = req->count;

if (count > VIDEO\_MAX\_FRAME) count = VIDEO\_MAX\_FRAME; //count = [2,32]

q->ops->buf\_setup(q, &count, &size); //size = byte\_pre\_line x height

req->count = **\_\_videobuf\_mmap\_setup**(q, count, size, req->memory);

}

#### \_\_videobuf\_mmap\_setup()

int **\_\_videobuf\_mmap\_setup**(struct videobuf\_queue \*q, unsigned int bcount, unsigned int bsize, enum v4l2\_memory memory)

{

err = \_\_videobuf\_free(q);

for (i = 0; i < bcount; i++) {

**q->bufs[i] = videobuf\_alloc\_vb(q);**

q->bufs[i]->i = i;

q->bufs[i]->memory = memory;

**q->bufs[i]->bsize = bsize;** //size = byte\_pre\_line x height

switch (memory) { case V4L2\_MEMORY\_MMAP: q->bufs[i]->boff = PAGE\_ALIGN(bsize) \* i; break; }

}

return i;

}

struct videobuf\_buffer \***videobuf\_alloc\_vb**(struct videobuf\_queue \*q)

{

struct videobuf\_buffer \*vb = **q->int\_ops->alloc\_vb**(q->msize); //msize = sizeof(struct camera\_buffer)

vb->magic = MAGIC\_BUFFER;

return vb;

}

#### \_\_videobuf\_alloc()

static struct videobuf\_buffer \***\_\_videobuf\_alloc**(size\_t size)

{

struct videobuf\_dma\_contig\_memory \*mem;

struct **videobuf\_buffer** \*vb = **kzalloc(size + sizeof(\*mem),** GFP\_KERNEL);

if (vb) {

vb->priv = ((char \*)vb) + size;

mem = vb->priv;

mem->magic = MAGIC\_DC\_MEM;

}

return vb;

}

### mmap: 初始化videobuf\_mapping设备 // icd->vb\_vidq.bufs[i]->map

#### Sample code

buf.index = i;

buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

buf.memory = V4L2\_MEMORY\_MMAP;

ret = ioctl(fd , VIDIOC\_QUERYBUF, &buf);

framebuf[i].length = buf.length;

framebuf[i].start = (char \*) **mmap**(0, **buf.length**, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, **buf.m.offset**);

ret = ioctl(fd , VIDIOC\_QBUF, &buf);

#### soc\_camera\_mmap()

static int **soc\_camera\_mmap**(struct file \*file, struct vm\_area\_struct \*vma)

{

if (ici->ops->init\_videobuf)

err = **videobuf\_mmap\_mapper**(&icd->vb\_vidq, vma);

}

int **videobuf\_mmap\_mapper**(struct videobuf\_queue \*q, struct vm\_area\_struct \*vma)

{

for (i = 0; i < VIDEO\_MAX\_FRAME; i++) {

struct videobuf\_buffer \*buf = q->bufs[i];

if (buf && buf->memory == V4L2\_MEMORY\_MMAP && buf->boff == (vma->vm\_pgoff << PAGE\_SHIFT)) {

rc = **CALL(q, mmap\_mapper, q, buf, vma);** break;

}}}

**#define CALL(q, f, arg...)** \

((q->int\_ops->f) ? q->**int\_ops->f**(arg) : 0)

#### \_\_videobuf\_mmap\_mapper()

static int **\_\_videobuf\_mmap\_mapper**(struct videobuf\_queue \*q,struct videobuf\_buffer \*buf,struct vm\_area\_struct \*vma)

{

struct **videobuf\_mapping** \*map = **kzalloc**(**sizeof(struct videobuf\_mapping)**, GFP\_KERNEL);

map->q = q;

buf->map = map;

buf->baddr = vma->vm\_start;

struct videobuf\_dma\_contig\_memory \*mem = buf->priv;

**\_\_videobuf\_dc\_alloc**(q->dev, mem, PAGE\_ALIGN(buf->bsize),GFP\_KERNEL | \_\_GFP\_COMP);

size = vma->vm\_end - vma->vm\_start;

size = (size < mem->size) ? size : mem->size;

vma->vm\_page\_prot = pgprot\_noncached(vma->vm\_page\_prot);

remap\_pfn\_range(vma, vma->vm\_start,mem->dma\_handle >> PAGE\_SHIFT,size, vma->vm\_page\_prot);

vma->vm\_ops = **&videobuf\_vm\_ops;**

vma->vm\_flags |= VM\_DONTEXPAND;

vma->vm\_private\_data = map;

**videobuf\_vm\_open(vma);**

}

static int **\_\_videobuf\_dc\_alloc**(struct device \*dev,struct videobuf\_dma\_contig\_memory \*mem,unsigned long size, gfp\_t flags)

{

mem->size = size;

mem->vaddr = **dma\_alloc\_coherent**(dev, mem->size, &mem->dma\_handle, flags);

}

#### struct vm\_operations\_struct videobuf\_vm\_ops

static const struct **vm\_operations\_struct videobuf\_vm\_ops** = {

.open = videobuf\_vm\_open,

.close = videobuf\_vm\_close,

};

### VIDIOC\_QBUF: list\_add\_tail(&buf->stream, &q->stream); list\_add\_tail(&vb->queue, &cam\_param->capture);

#### Sample code

buf.index = i;

buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

buf.memory = V4L2\_MEMORY\_MMAP;

ret = ioctl(fd , VIDIOC\_QUERYBUF, &buf);

framebuf[i].length = buf.length;

framebuf[i].start = (char \*) mmap(0, buf.length, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, buf.m.offset);

ret = ioctl(fd , **VIDIOC\_QBUF,** &buf);

#### struct v4l2\_buffer

struct v4l2\_buffer {

\_\_u32 index;

\_\_u32 type;

\_\_u32 bytesused;

\_\_u32 flags;

\_\_u32 field;

struct timeval timestamp;

struct v4l2\_timecode timecode;

\_\_u32 sequence;

\_\_u32 memory;

union {

\_\_u32 offset;

unsigned long userptr;

struct v4l2\_plane \*planes;

\_\_s32 fd;

} m;

\_\_u32 length;

\_\_u32 reserved2;

\_\_u32 reserved;

};

#### soc\_camera\_qbuf()

static int **soc\_camera\_qbuf**(struct file \*file, void \*priv, struct v4l2\_buffer \*p)

{

if (ici->ops->init\_videobuf)

return **videobuf\_qbuf**(&icd->vb\_vidq, p);

}

int **videobuf\_qbuf**(struct videobuf\_queue \*q, struct v4l2\_buffer \*b)

{

struct videobuf\_buffer \*buf = q->bufs[b->index];

retval = **q->ops->buf\_prepare**(q, buf, field);

**list\_add\_tail(&buf->stream, &q->stream);**

if (q->streaming) q->ops->buf\_queue(q, buf);

}

#### camera\_videobuf\_prepare()

static int **camera\_videobuf\_prepare**(struct videobuf\_queue \*vq, struct videobuf\_buffer \*vb, enum v4l2\_field field)

{

struct camera\_buffer \*buf = container\_of(vb, struct camera\_buffer, vb);

if (buf->code != icd->current\_fmt->code || vb->width != icd->user\_width \

|| vb->height != icd->user\_height || vb->field != field) {

buf->code = icd->current\_fmt->code;

vb->width = icd->user\_width;

vb->height = icd->user\_height;

vb->field = field;

**vb->state = VIDEOBUF\_NEEDS\_INIT;**

bytes\_per\_line = soc\_mbus\_bytes\_per\_line(icd->user\_width, icd->current\_fmt->host\_fmt);

}

**vb->size = vb->height \* bytes\_per\_line;**

if (vb->state == VIDEOBUF\_NEEDS\_INIT) **vb->state = VIDEOBUF\_PREPARED**;

}

#### camera\_videobuf\_queue()

static void **camera\_videobuf\_queue**(struct videobuf\_queue \*vq, struct videobuf\_buffer \*vb)

{

vb->state = VIDEOBUF\_QUEUED;

**list\_add\_tail(&vb->queue, &cam\_param->capture);**

if (!cam\_param->cur\_frm) { … }

}

### VIDIOC\_STREAMON: capture\_start()

#### Sample code

enum v4l2\_buf\_type type = **V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE**;

ret = ioctl(fd, **VIDIOC\_STREAMON**, &type);

#### soc\_camera\_streamon()

static int **soc\_camera\_streamon**(struct file \*file, void \*priv, enum v4l2\_buf\_type i)

{

if (ici->ops->init\_videobuf)

ret = **videobuf\_streamon**(&icd->vb\_vidq);

v4l2\_subdev\_call(sd, video, s\_stream, 1);

}

int **videobuf\_streamon**(struct videobuf\_queue \*q)

{

struct videobuf\_buffer \*buf;

**q->streaming = 1;**

list\_for\_each\_entry(buf, &q->stream, stream)

if (buf->state == VIDEOBUF\_PREPARED) **q->ops->buf\_queue**(q, buf);

}

#### camera\_videobuf\_queue()

static void **camera\_videobuf\_queue**(struct videobuf\_queue \*vq, struct videobuf\_buffer \*vb)

{

**vb->state = VIDEOBUF\_QUEUED**;

**list\_add\_tail(&vb->queue, &cam\_param->capture);**

if (!cam\_param->cur\_frm) {

cam\_param->cur\_frm = list\_entry(cam\_param->capture.next, struct videobuf\_buffer, queue);

cam\_param->cur\_frm->**state = VIDEOBUF\_ACTIVE;**

list\_del\_init(&cam\_param->cur\_frm->queue);

cam\_param->prev\_frm = NULL;

set\_rect(icd); //设置SI模块寄存器: wxh

**set\_frame(icd, cam\_param->cur\_frm);** //设置SI模块寄存器: YUV addr

**capture\_start**(cam\_dev, icd); //设置SI/CSI模块寄存器: enable

}

}

#### set\_frame()

static void **set\_frame**(struct soc\_camera\_device \*icd, struct videobuf\_buffer \*vb)

{

phys\_addr\_t module\_addr;

switch (vb->memory) {

case V4L2\_MEMORY\_MMAP: **module\_addr = videobuf\_to\_dma\_contig(vb);** break;

case V4L2\_MEMORY\_USERPTR: **module\_addr = vb->baddr;** break;

}

switch (fourcc) {

case V4L2\_PIX\_FMT\_YUV420: module\_set\_frame\_yuv420(module\_addr, icd); break;

case V4L2\_PIX\_FMT\_YVU420: module\_set\_frame\_yvu420(module\_addr, icd); break;

case V4L2\_PIX\_FMT\_YUV422P:module\_set\_frame\_yvu422(module\_addr, icd); break;

case V4L2\_PIX\_FMT\_NV12:

case V4L2\_PIX\_FMT\_NV21: module\_set\_frame\_nv12\_nv21(module\_addr, icd);break;

case V4L2\_PIX\_FMT\_YUYV: module\_set\_frame\_yuyv(module\_addr, icd); break;

}

}

### IRQ: list\_del\_init(&cam\_param->cur\_frm->queue);

#### camera\_host\_probe()

static int **camera\_host\_probe**(struct platform\_device \*pdev)

{

unsigned int **irq = platform\_get\_irq**(pdev, 0);

cam\_dev->irq = irq;

devm\_request\_irq(&pdev->dev, cam\_dev->irq, **camera\_host\_isr**, IRQF\_DISABLED, CAM\_HOST\_NAME, cam\_dev);

}

static irqreturn\_t **camera\_host\_isr**(int irq, void \*data)

{

struct camera\_dev \*cam\_dev = data;

for (i = 0; i < 2; i++) {

if (cam\_dev->icds[i]) {

icd = cam\_dev->icds[i];

cam\_param = icd->host\_priv;

struct v4l2\_subdev \*sd = soc\_camera\_to\_subdev(icd);

module\_int\_stat = reg\_read(GMODULEMAPADDR, MODULE\_STAT, MODULE\_BASE);

**module\_isr**(icd, cam\_param, sd, module\_int\_stat, i);

}

return IRQ\_HANDLED;

}

static inline int **module\_isr**(struct soc\_camera\_device \*icd, struct camera\_param \*cam\_param,

struct v4l2\_subdev \*sd, unsigned int module\_int\_stat, int i)

{

if (module\_int\_stat & preline\_int\_pd) { //si有效中断

//prev\_frm: 指向上一次已填满数据的videobuf\_buffer

if (cam\_param->prev\_frm != NULL) {

struct videobuf\_buffer \*vb = cam\_param->prev\_frm;

**vb->state = VIDEOBUF\_DONE;**

**do\_gettimeofday(&vb->ts);**

vb->field\_count++;

**wake\_up(&vb->done);**

}

//cur\_frm：SI模块已将camera的数据填到cur\_frm指向的videobuf\_buffer地址中

//将prev\_frm指向已填满数据的videobuf\_buffer

//cur\_frm从cam\_param->capture链表中获取空的videobuf\_buffer,

//并将此videobuf\_buffer从cam\_param->capture链表中移除

if (!list\_empty(&cam\_param->capture)) { //QBUF: list\_add\_tail(&vb->queue, &cam\_param->capture)

cam\_param->prev\_frm = cam\_param->cur\_frm;

cam\_param->cur\_frm = **list\_entry(cam\_param->capture.next, struct videobuf\_buffer, queue);**

**list\_del\_init(&cam\_param->cur\_frm->queue);**

set\_frame(icd, cam\_param->cur\_frm);

**cam\_param->cur\_frm->state = VIDEOBUF\_ACTIVE;**

} else {

//cam\_param->capture链表中没有可用的videobuf\_buffer,

// cur\_frm继续指向当前的videobuf\_buffer被SI模块使用

cam\_param->prev\_frm = NULL;

set\_frame(icd, cam\_param->cur\_frm);

}

}

}

### VIDIOC\_DQBUF: list\_del(&buf->stream);

#### Sample code

ioctl(fd, **VIDIOC\_DQBUF,** &buf);

#### soc\_camera\_dqbuf()

static int **soc\_camera\_dqbuf**(struct file \*file, void \*priv, struct v4l2\_buffer \*p)

{

if (ici->ops->init\_videobuf)

return **videobuf\_dqbuf**(&icd->vb\_vidq, p, file->f\_flags & O\_NONBLOCK);

}

int **videobuf\_dqbuf**(struct videobuf\_queue \*q, struct v4l2\_buffer \*b, int nonblocking)

{

struct videobuf\_buffer \*buf = NULL;

memset(b, 0, sizeof(\*b));

retval = **stream\_next\_buffer**(q, &buf, nonblocking);

CALL(q, sync, q, buf);

videobuf\_status(q, b, buf, q->type);

**list\_del(&buf->stream);**

buf->**state = VIDEOBUF\_IDLE;**

b->flags &= ~V4L2\_BUF\_FLAG\_DONE;

}

#### stream\_next\_buffer()

static int **stream\_next\_buffer**(struct videobuf\_queue \*q, struct videobuf\_buffer \*\*vb, int nonblocking)

{

**stream\_next\_buffer\_check\_queue**(q, nonblocking); //判断q->stream是否有buf

struct videobuf\_buffer \*buf = list\_entry(q->stream.next, struct videobuf\_buffer, stream);

**videobuf\_waiton**(q, buf, nonblocking, 1);

\*vb = buf;

}

static int **stream\_next\_buffer\_check\_queue**(struct videobuf\_queue \*q, int noblock)

{

if (list\_empty(&q->stream)) {

if (noblock) { goto done; }

else { **wait\_event\_interruptible**(**q->wait**, !list\_empty(&q->stream) || !q->streaming); }

}

}

int **videobuf\_waiton**(struct videobuf\_queue \*q, struct videobuf\_buffer \*vb, int non\_blocking, int intr)

{

if (intr) **wait\_event\_interruptible**(**vb->done**, is\_state\_active\_or\_queued(q, vb));

else wait\_event(vb->done, is\_state\_active\_or\_queued(q, vb));

}

### VIDIOC\_STREAMOFF: capture\_stop(),list\_del(&q->bufs[i]->queue);

#### Sample code

enum v4l2\_buf\_type type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ret = ioctl(fd, **VIDIOC\_STREAMOFF**, &type);

#### soc\_camera\_streamoff()

static int **soc\_camera\_streamoff**(struct file \*file, void \*priv, enum v4l2\_buf\_type i)

{

if (ici->ops->init\_videobuf)

**videobuf\_streamoff**(&icd->vb\_vidq);

}

int **videobuf\_streamoff**(struct videobuf\_queue \*q)

{

**\_\_videobuf\_streamoff**(q);

}

static int **\_\_videobuf\_streamoff**(struct videobuf\_queue \*q)

{

**videobuf\_queue\_cancel**(q);

}

void **videobuf\_queue\_cancel**(struct videobuf\_queue \*q)

{

**q->streaming = 0;**

q->reading = 0;

wake\_up\_interruptible\_sync(&q->wait);

for (i = 0; i < VIDEO\_MAX\_FRAME; i++) {

if (NULL == q->bufs[i]) continue;

**if (q->bufs[i]->state == VIDEOBUF\_QUEUED)** {

**list\_del(&q->bufs[i]->queue);**

q->bufs[i]->state = VIDEOBUF\_ERROR;

wake\_up\_all(&q->bufs[i]->done);

}

}

for (i = 0; i < VIDEO\_MAX\_FRAME; i++) {

if (NULL == q->bufs[i]) continue;

**q->ops->buf\_release**(q, q->bufs[i]);

}

INIT\_LIST\_HEAD(&q->stream);

}

#### camera\_videobuf\_release()

static void **camera\_videobuf\_release**(struct videobuf\_queue \*vq, struct videobuf\_buffer \*vb)

{

**capture\_stop(cam\_dev, icd);**

**if ((vb->state == VIDEOBUF\_ACTIVE || vb->state == VIDEOBUF\_QUEUED)) {**

vb->state = VIDEOBUF\_ERROR;

if (!list\_empty(&cam\_param->capture)) **list\_del\_init(&vb->queue);**

wake\_up\_all(&vb->done);

}

**free\_buffer**(vq, container\_of(vb, struct camera\_buffer, vb));

**if (buf->vb.memory != V4L2\_MEMORY\_MMAP)**

for (i = 0; i < VIDEO\_MAX\_FRAME; i++) {

if (NULL == vq->bufs[i]) continue;

if (vb == vq->bufs[i]) {  **kfree(vq->bufs[i])**; vq->bufs[i] = NULL; }

}

}

### munmap: kfree(map)

#### Sample code

for (i=0; i< BUFFER\_COUNT; i++)

{

**munmap**(framebuf[i].start, framebuf[i].length);

}

#### videobuf\_vm\_close()

static void **videobuf\_vm\_close**(struct vm\_area\_struct \*vma)

{

struct videobuf\_mapping \*map = vma->vm\_private\_data;

struct videobuf\_queue \*q = map->q;

map->count--;

if (0 == map->count) {

for (i = 0; i < VIDEO\_MAX\_FRAME; i++) {

if (NULL == q->bufs[i]) continue;

if (q->bufs[i]->map != map) continue;

struct videobuf\_dma\_contig\_memory \*mem = q->bufs[i]->priv;

if (mem) {

**\_\_videobuf\_dc\_free**(q->dev, mem);

mem->vaddr = NULL;

}

q->bufs[i]->map = NULL;

q->bufs[i]->baddr = 0;

}

**kfree(map);**

}

}

static void \_\_videobuf\_dc\_free(struct device \*dev, struct videobuf\_dma\_contig\_memory \*mem)

{

**dma\_free\_coherent**(dev, mem->size, mem->vaddr, mem->dma\_handle);

mem->vaddr = NULL;

}

### Close: kfree((struct camera\_param \*)cam\_param)

#### Sample code

**close**(fd);

#### soc\_camera\_close()

static int **soc\_camera\_close**(struct file \*file)

{

icd->use\_count--;

if (!icd->use\_count) {

\_\_soc\_camera\_power\_off(icd);

**ici->ops->remove**(icd);

}

}

#### camera\_remove\_device()

static void camera\_remove\_device(struct soc\_camera\_device \*icd)

{

camera\_clock\_disable(cam\_dev, icd);

**capture\_stop(cam\_dev, icd);**

**kfree(cam\_param);**

}

### QBUF – IRQ – DQBUF

|  |  |
| --- | --- |
| QBUF： | (struct videobuf\_buffer \*)vb->state = **VIDEOBUF\_QUEUED**;  **list\_add\_tail**((struct videobuf\_buffer \*)&**buf->stream**, (struct videobuf\_queue \*)&q->stream);  **list\_add\_tail**((struct videobuf\_buffer \*)&**vb->queue**, (struct camera\_param \*)&cam\_param->capture); |
| IRQ: | struct videobuf\_buffer \*vb = cam\_param->**prev\_frm**:  vb->state = VIDEOBUF\_DONE;  wake\_up(&vb->done);  cam\_param->**prev\_frm** = cam\_param->**cur\_frm**  cam\_param->c**ur\_frm** = list\_entry(cam\_param->capture.next, struct videobuf\_buffer, queue);  **list\_del\_init**(&cam\_param->**cur\_frm->queue**);  cam\_param->cur\_frm->state = **VIDEOBUF\_ACTIVE;** |
| DQBUF: | **list\_del**((struct videobuf\_buffer \*)&**buf->stream**);  buf->state = **VIDEOBUF\_IDLE**; |

# V4L2 Control

## 创建v4l2 control设备

### struct camera\_module\_priv

**struct camera\_module\_priv** {

struct v4l2\_subdev subdev;

struct v4l2\_ctrl\_handler hdl;

}

### camera\_module\_probe() //配置v4l2\_ctrl\_ops

static int **camera\_module\_probe**(struct i2c\_client \*client, const struct i2c\_device\_id \*did)

{

//1. 创建v4l2\_ctrl\_handler设备

struct **camera\_module\_priv** \*priv;

priv = **devm\_kzalloc**(&client->dev, sizeof(\*priv), GFP\_KERNEL);

//2. 初始化v4l2\_ctrl\_handler和v4l2\_subdev, 配置v4l2\_ctrl\_ops

v4l2\_i2c\_subdev\_init(&**priv->subdev**, client, &module\_subdev\_ops);

**camera\_module\_init\_ops**(&**priv->hdl,** &**camera\_module\_ctrl\_ops**);

//3. 关联v4l2\_ctrl\_handler和v4l2\_device

priv->**subdev**.ctrl\_handler = &priv->**hdl**;

//4. 执行所有的control，使用控制变量的默认值初始化所有硬件

**v4l2\_ctrl\_handler\_setup**(&priv->hdl);

}

### camera\_module\_init\_ops()

static void **camera\_module\_init\_ops**(struct v4l2\_ctrl\_handler \*hdl, const struct v4l2\_ctrl\_ops \*ops)

{

//初始化v4l2\_ctrl\_handler:第二个参数是你要使用的控制变量的个数

unsigned int cmd\_array\_size = ARRAY\_SIZE(**v4l2\_ctl\_array**);

unsigned int cmd\_menu\_array\_size = ARRAY\_SIZE(**v4l2\_ctl\_array\_menu**);

**v4l2\_ctrl\_handler\_init**(hdl, **cmd\_array\_size + cmd\_menu\_array\_size**);

for (i = 0; i < cmd\_array\_size; i++) {

//这个函数用来增加非菜单式的控制变量

const struct **v4l2\_ctl\_cmd\_info** \*pctl = v4l2\_ctl\_array + i;

**v4l2\_ctrl\_new\_std**(hdl, ops, pctl->id, pctl->min, pctl->max, pctl->step, pctl->def);

hdl->error = 0;

}

for (i = 0; i < cmd\_menu\_array\_size; i++) {

//这个函数用来增加菜单式的控制变量

const struct **v4l2\_ctl\_cmd\_info\_menu** \*pmenu = v4l2\_ctl\_array\_menu + i;

**v4l2\_ctrl\_new\_std\_menu**(hdl, ops, pmenu->id, pmenu->max, pmenu->mask, pmenu->def);

hdl->error = 0;

}

}

## 非菜单式的控制变量

### struct v4l2\_ctl\_cmd\_info v4l2\_ctl\_array

static struct **v4l2\_ctl\_cmd\_info** **v4l2\_ctl\_array**[] =

{

{

**.id = V4L2\_CID\_EXPOSURE,**

.min = 0x8,

.max = 0x580,

.step = 1,

.def = 0x478,

},

}

struct **v4l2\_ctl\_cmd\_info** {

unsigned int id;

int min;

int max;

unsigned int step;

int def;

};

### v4l2\_ctrl\_new\_std()

unsigned int cmd\_array\_size = ARRAY\_SIZE(**v4l2\_ctl\_array**);

for (i = 0; i < cmd\_array\_size; i++) {

const struct v4l2\_ctl\_cmd\_info \*pctl = **v4l2\_ctl\_array** + i;

ret = **v4l2\_ctrl\_new\_std**(hdl, ops, pctl->id, pctl->min, pctl->max, pctl->step, pctl->def);

}

## 菜单式的控制变量

### struct v4l2\_ctl\_cmd\_info\_menu v4l2\_ctl\_array\_menu

static struct **v4l2\_ctl\_cmd\_info\_menu** **v4l2\_ctl\_array\_menu**[] =

{

{ **.id = V4L2\_CID\_COLORFX,**

.max = 3,

.mask = 0x0,

.def = 0,

},

}

struct **v4l2\_ctl\_cmd\_info\_menu** {

unsigned int id;

int max;

int mask;

int def;

};

### v4l2\_ctrl\_new\_std\_menu ()

unsigned int cmd\_menu\_array\_size = ARRAY\_SIZE(**v4l2\_ctl\_array\_menu**);

for (i = 0; i < cmd\_menu\_array\_size; i++) {

const struct v4l2\_ctl\_cmd\_info\_menu \*pmenu = **v4l2\_ctl\_array\_menu** + i;

ret = **v4l2\_ctrl\_new\_std\_menu**(hdl, ops, pmenu->id, pmenu->max, pmenu->mask, pmenu->def);

}

## v4l2\_ctrl\_ops的实现

### struct v4l2\_ctrl\_ops camera\_module\_ctrl\_ops

static const **struct v4l2\_ctrl\_ops camera\_module\_ctrl\_ops** = {

**.g\_volatile\_ctrl** = camera\_module\_g\_volatile\_ctrl,

**.s\_ctrl** = camera\_module\_s\_ctrl,

};

### camera\_module\_s\_ctrl()

static int **camera\_module\_s\_ctrl**(**struct v4l2\_ctrl \*ctrl)**

{

struct camera\_module\_priv \*priv = container\_of(ctrl->handler, struct camera\_module\_priv , hdl);

struct v4l2\_subdev \*sd = &priv->subdev;

struct i2c\_client \*client = v4l2\_get\_subdevdata(sd);

switch (**ctrl->id)** {

case **V4L2\_CID\_EXPOSURE:** module\_set\_exposure(client, **ctrl->val**); break;

case **V4L2\_CID\_COLORFX:** module\_set\_colorfx(sd, ctrl); break;

}

### camera\_module\_g\_volatile\_ctrl()

static int **camera\_module\_g\_volatile\_ctrl**(**struct v4l2\_ctrl \*ctrl**)

{

struct camera\_module\_priv \*priv = container\_of(ctrl->handler, struct camera\_module\_priv , hdl);

struct v4l2\_subdev \*sd = &priv->subdev;

struct i2c\_client \*client = v4l2\_get\_subdevdata(sd);

switch (**ctrl->id**) {

case **V4L2\_CID\_EXPOSURE**: ret = module\_get\_exposure(client, &**ctrl->val)**; break;

}

## 销毁v4l2 control设备

### camera\_module\_remove()

static int **camera\_module\_remove**(struct i2c\_client \*client)

{

//在驱动退出的时候清除v4l2\_ctrl\_handler

struct camera\_module\_priv \*priv = to\_camera\_priv(client);

v4l2\_device\_unregister\_subdev(&priv->subdev);

**v4l2\_ctrl\_handler\_free**(&priv->**hdl**);

}

## 增加v4l2 control的控制变量方法

### v4l2\_ctrl\_new\_std()

struct v4l2\_ctrl \***v4l2\_ctrl\_new\_std**(struct v4l2\_ctrl\_handler \*hdl, const struct **v4l2\_ctrl\_ops** \*ops, u32 **id,** s32 **min**, s32 **max**, u32 **step**, s32 **def**);

v4l2\_ctrl\_new\_std(&foo->ctrl\_handler, &foo\_ctrl\_ops,V4L2\_CID\_VOLUM, 0, 255, 1, 128);

这个函数用来增加**非菜单式的控制变量**，如声音等。

min是最小值,max是最大值,step是步进长度，def是默认值。

这个函数适用于在某一范围内均匀变化的控制变量。

### v4l2\_ctrl\_new\_std\_menu()

struct v4l2\_ctrl \*v4l2\_ctrl\_new\_std\_menu(struct v4l2\_ctrl\_handler \*hdl, const struct **v4l2\_ctrl\_ops** \*ops, u32 **id**, s32 **max**, s32 **skip\_mask**, s32 **def**);

enum{ BW\_6M, BW\_7M, BW\_8M }BW;

v4l2\_ctrl\_new\_std\_menu(&foo->ctrl\_handler, & ops, V4L2\_CID\_BW, BW\_8M , 0, BW\_6M);

这个函数用来增加**菜单式的控制变量**，

max是菜单最大值, skip\_mask是屏蔽标志，比如它的值是0x4,就屏蔽菜单值为4的菜单选项，def是默认菜单值。

这个函数适用于从0开始，相邻**变量值**增**加1**的控制变量。

### v4l2\_ctrl\_new\_int\_menu()

struct v4l2\_ctrl \*v4l2\_ctrl\_new\_int\_menu(struct v4l2\_ctrl\_handler \*hdl, const struct **v4l2\_ctrl\_ops** \*ops, u32 **id**, s32 **max**, s32 **def,** const s64 \***qmenu\_int**);

static const s64 audio\_sample\_qmenu[] = {22100,44100,48000,96000};

v4l2\_ctrl\_new\_int\_menu(&foo->ctrl\_handler, & ops, V4L2\_CID\_AUDIOSAMPLERATE, ARRAY\_SIZE(audio\_sample\_qmenu)-1, 1, audio\_sample\_qmenu);

这个函数用来增加**整数型菜单的控制变量**，

max是最大的菜单索引号，def是默认的菜单索引号，qmenu\_int是菜单的数组指针。

这个函数**适用**于**变量值**是**不连续**的**无规则**的**整数**的控制变量。

### v4l2\_ctrl\_new\_std\_menu\_items()

struct v4l2\_ctrl \*v4l2\_ctrl\_new\_std\_menu\_items(struct v4l2\_ctrl\_handler \*hdl, const struct **v4l2\_ctrl\_ops** \*ops, u32 **id**, s32 **max**, s32 **skip\_mask**, s32 **def**, const char \* const \***qmenu**);

static const char \* const test\_pattern[] = {"Disabled", "Vertical Bars", "Solid Black", "Solid White", };

v4l2\_ctrl\_new\_std\_menu\_items(&foo->ctrl\_handler, & ops, V4L2\_CID\_TEST\_PATTERN, ARRAY\_SIZE(test\_pattern) - 1, 0, 0, test\_pattern);

这个函数用来增加一个**驱动指定的菜单数组给控制变量**，

max是菜单最大索引号，skip\_mask是需要跳过的菜单的索引号，def是默认菜单索引号，qmenu是菜单数组。

# Video设备的属性配置

## 分辨率和数据格式

camera\_G\_FMT(); //获取驱动当前的width,height,format的配置

camera\_enumfmt\_all(); //获取驱动的format支持的分辨率有哪些

camera\_enumframesize\_all(); //获取驱动的width,height支持的分辨率有哪些

camera\_S\_FMT\_index(framesize\_index\_i, format\_index\_i); //配置驱动当前的width,height,format的配置

### VIDIOC\_G\_FMT

int **camera\_G\_FMT**(void)

{

struct v4l2\_format fmt;

memset(&fmt, 0, sizeof(fmt));

fmt.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ret = **ioctl(gFd, VIDIOC\_G\_FMT, &fmt);**

char fmtstr[8];

memset(fmtstr, 0, 8);

memcpy(fmtstr, &fmt.fmt.pix.pixelformat, 4);

DBG("=======================[VIDIOC\_G\_FMT: S]=======================\n");

DBG(" type: %d\n", fmt.type);

**DBG(" width\* height: %d \* %d\n", fmt.fmt.pix.width , fmt.fmt.pix.height);**

**DBG(" pixelformat: %s\n", fmtstr);**

DBG(" field: %d\n", fmt.fmt.pix.field);

DBG(" bytesperline: %d\n", fmt.fmt.pix.bytesperline);

DBG(" sizeimage: %d\n", fmt.fmt.pix.sizeimage);

DBG(" colorspace: %d\n", fmt.fmt.pix.colorspace);

DBG(" priv: %d\n", fmt.fmt.pix.priv);

DBG("=======================[VIDIOC\_G\_FMT: E]=======================\n");

}

### VIDIOC\_ENUM\_FMT

int **camera\_enumfmt\_all**(void)

{

DBG("===================[VIDIOC\_ENUM\_FMT: S]===================\n");

Index = 0;

while(!ret){

ret = **camera\_enumfmt(index);**

index++;

}

gFmt\_index = index-1;

DBG("==================[VIDIOC\_ENUM\_FMT: E (count = =============\n", gFmt\_index);

}

int **camera\_enumfmt**(int index)

{

struct v4l2\_fmtdesc fmtdest;

memset(&fmtdest, 0, sizeof(fmtdest));

**fmtdest.index = index;**

fmtdest.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ret = ioctl(gFd, VIDIOC\_ENUM\_FMT, &fmtdest);

**v\_efmt[index].pixfmt = fmtdest.pixelformat;**

**memcpy(v\_efmt[index].desc, fmtdest.description, 32);**

**v\_efmt[index].desc[31] = '\0';**

DBG(" index: %d\n", fmtdest.index);

DBG(" pixelformat: %d\n", fmtdest.pixelformat);

DBG(" description: %s\n", fmtdest.description);

char fmtstr[8];

memset(fmtstr, 0, 8);

memcpy(fmtstr, &fmtdest.pixelformat, 4);

DBG(" pixelformat: %s\n", fmtstr);

}

**struct video\_enumfmt** {

unsigned int pixfmt;

unsigned char desc[32];

};

**struct video\_enumfmt** **v\_efmt**[6];

int **gFmt\_index** = 0;

### VIDIOC\_ENUM\_FRAMESIZES

int **camera\_enumframesize\_all**(void)

{

int index = 0;

DBG("=====================[VIDIOC\_ENUM\_FRAMESIZES: S]=======================\n");

while(!ret){

ret = **camera\_enumframesize(index);**

index++;

}

gFramesize\_index = index-1;

DBG("=====================[VIDIOC\_ENUM\_FRAMESIZES: E: (count = %d)=========\n", gFramesize\_index);

}

int **camera\_enumframesize**(int index)

{

struct v4l2\_frmsizeenum frmsize;

memset(&frmsize, 0, sizeof(frmsize));

frmsize.index = index;

frmsize.pixel\_format = v\_efmt[0].pixfmt;

**ret = ioctl(gFd, VIDIOC\_ENUM\_FRAMESIZES, &frmsize);**

DBG(" index: %d\n", frmsize.index);

DBG(" pixel\_format: %d\n", frmsize.pixel\_format);

DBG(" type: %d\n", frmsize.type);

if(frmsize.type == V4L2\_FRMSIZE\_TYPE\_DISCRETE) {

**v\_eframesize[index].w = frmsize.discrete.width;**

**v\_eframesize[index].h = frmsize.discrete.height;**

DBG(" width: %d, %d\n", frmsize.discrete.width, v\_eframesize[index].w);

DBG(" height: %d, %d\n", frmsize.discrete.height, v\_eframesize[index].h);

}

else {

DBG(" width: %d, %d\n", frmsize.stepwise.min\_width, frmsize.stepwise.max\_width);

DBG(" height: %d, %d\n", frmsize.stepwise.min\_height, frmsize.stepwise.max\_height);

}

}

**struct video\_enumframesizes** {

unsigned int w;

unsigned int h;

};

**struct video\_enumframesizes v\_eframesize**[10];

int **gFramesize\_index** = 0;

### VIDIOC\_S\_FMT

int **camera\_S\_FMT\_index**(int framesize\_index, int fmt\_index)

{

if(fmt\_index > gFmt\_index) fmt\_index = 0;

if(framesize\_index > gFramesize\_index) framesize\_index = 0;

unsigned int w,h,pixfmt;

**w = v\_eframesize[framesize\_index].w;**

**h = v\_eframesize[framesize\_index].h;**

**pixfmt = v\_efmt[fmt\_index].pixfmt;**

**camera\_S\_FMT**(w, h, pixfmt);

}

int **camera\_S\_FMT**(unsigned int w, unsigned int h, unsigned int pixfmt)

{

struct v4l2\_format fmt;

memset(&fmt, 0, sizeof(fmt));

fmt.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ret = ioctl(gFd, VIDIOC\_G\_FMT, &fmt);

fmt.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

**fmt.fmt.pix.width = w;**

**fmt.fmt.pix.height = h;**

**fmt.fmt.pix.pixelformat = pixfmt;**

ret = **ioctl(gFd, VIDIOC\_S\_FMT, &fmt);**

camera\_G\_FMT();

}

## 曝光配置

int **test\_exposure**(int exposure)

{

//get exposure max,min,step

int min=0,max=0,step=0;

**camera\_get\_exposure\_query**(&min, &max, &step);

printf("min=%d, max=%d, step=%d\n", min, max, step);

//set exposure

ret = **camera\_set\_exposure(**exposure);

//get exposure

**camera\_get\_exposure**(&gvalue);

}

### VIDIOC\_QUERYCTRL

int **camera\_get\_exposure\_query**(int \*min, int \*max, int \*step)

{

**camera\_get\_query**(**V4L2\_CID\_EXPOSURE**, min, max, step);

}

int **camera\_get\_query**(int cmd, int \*min, int \*max, int \*step)

{

struct v4l2\_queryctrl queryctrl;

memset(&v4l2\_queryctrl, 0, sizeof(v4l2\_queryctrl))

**queryctrl.id = cmd;**

ret = **ioctl(gFd, VIDIOC\_QUERYCTRL, &queryctrl);**

DBG("min = %d\n", queryctrl.minimum);

DBG("max = %d\n", queryctrl.maximum);

DBG("default = %d\n", queryctrl.default\_value);

DBG("step = %d\n", queryctrl.step);

DBG("flag = %d\n", queryctrl.flags);

\*min = queryctrl.minimum;

\*max = queryctrl.maximum;

\*step = queryctrl.step;

}

### VIDIOC\_G\_CTRL

int **camera\_get\_exposure**(int \*value)

{

**camera\_get\_cmd(V4L2\_CID\_EXPOSURE**, value);

}

int **camera\_get\_cmd**(int cmd, int \*value)

{

struct v4l2\_control ctrl;

memset(&ctrl, 0, sizeof(ctrl));

**ctrl.id = cmd;**

**ioctl(gFd, VIDIOC\_G\_CTRL, &ctrl);**

\*value = ctrl.value;

}

static int **camera\_module\_g\_volatile\_ctrl**(**struct v4l2\_ctrl \*ctrl**)

{

struct camera\_module\_priv \*priv = container\_of(ctrl->handler, struct camera\_module\_priv , hdl);

struct v4l2\_subdev \*sd = &priv->subdev;

struct i2c\_client \*client = v4l2\_get\_subdevdata(sd);

switch (**ctrl->id**) {

case **V4L2\_CID\_EXPOSURE**: ret = module\_get\_exposure(client, &**ctrl->val)**; break;

}

### VIDIOC\_S\_CTRL

int **camera\_set\_exposure**(int value)

{

**camera\_set\_cmd**(**V4L2\_CID\_EXPOSURE**, value);

}

int **camera\_set\_cmd**(int cmd, int value)

{

struct v4l2\_control ctrl;

**ctrl.id = cmd;**

**ctrl.value = value;**

**ioctl(gFd, VIDIOC\_S\_CTRL, &ctrl);**

}

static int **camera\_module\_s\_ctrl**(**struct v4l2\_ctrl \*ctrl)**

{

struct camera\_module\_priv \*priv = container\_of(ctrl->handler, struct camera\_module\_priv , hdl);

struct v4l2\_subdev \*sd = &priv->subdev;

struct i2c\_client \*client = v4l2\_get\_subdevdata(sd);

switch (**ctrl->id)** {

case **V4L2\_CID\_EXPOSURE:** module\_set\_exposure(client, **ctrl->val**); break;

case **V4L2\_CID\_COLORFX:** module\_set\_colorfx(sd, ctrl); break;

}

## 帧率配置

int **test\_framerate**(int fps)

{

//set framerate

framerate = fps;

**camera\_set\_framerate**(framerate);

//get framerate

int framerate;

**camera\_get\_framerate**(&framerate);

}

### VIDIOC\_S\_PARM

int **camera\_set\_framerate**(int framerate)

{

struct **v4l2\_streamparm** streamparm;

memset(&streamparm, 0, sizeof(streamparm));

streamparm**.type** = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

streamparm.**parm.capture.timeperframe.numerator** = 1;

streamparm.**parm.capture.timeperframe.denominator** = **framerate;**

ioctl(gFd, **VIDIOC\_S\_PARM**, &streamparm);

}

static **struct v4l2\_subdev\_video\_ops** camera\_module\_subdev\_video\_ops = {

**.s\_parm**  = **camera\_module\_s\_parm,**

};

static int **camera\_module\_s\_parm**(struct v4l2\_subdev \*sd, struct **v4l2\_streamparm** \*parms)

{

struct v4l2\_captureparm \*cp = &parms->parm.capture;

**struct v4l2\_fract \*tpf = &cp->timeperframe**;

**camera\_module\_set\_framerate**(sd, tpf->**numerator**, tpf->**denominator**);

}

static int **camera\_module\_set\_framerate**(struct v4l2\_subdev \*sd, \_\_u32 **numerator**, \_\_u32 **denominator**)

{

struct i2c\_client \*client = v4l2\_get\_subdevdata(sd);

struct camera\_module\_priv \*priv = to\_camera\_priv(client);

unsigned int **frame\_rate =** denominator / numerator;

**module\_set\_framerate**(client, frame\_rate);

priv->timeperframe**.numerator** = numerator;

priv->timeperframe.**denominator** = denominator;

}

### VIDIOC\_G\_PARM

int **camera\_get\_framerate**(int \*framerate)

{

struct **v4l2\_streamparm** streamparm;

memset(&streamparm, 0, sizeof(streamparm));

streamparm**.type** = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ret = ioctl(gFd, **VIDIOC\_G\_PARM**, &streamparm);

**\*framerate** = streamparm.**parm.capture.timeperframe.denominator;**

}

static **struct v4l2\_subdev\_video\_ops** camera\_module\_subdev\_video\_ops = {

**.g\_parm = camera\_module\_g\_parm,**

};

static int **camera\_module\_g\_parm**(struct v4l2\_subdev \*sd, struct v4l2\_streamparm \*parms)

{

struct i2c\_client \*client = v4l2\_get\_subdevdata(sd);

struct camera\_module\_priv \*priv = to\_camera\_priv(client);

struct v4l2\_captureparm \*cp = &parms->parm.capture;

memset(cp, 0, sizeof(struct v4l2\_captureparm));

cp->capability = V4L2\_CAP\_TIMEPERFRAME;

cp->timeperframe.**numerator** = priv->timeperframe.numerator;

cp->timeperframe.**denominator** = priv->timeperframe.denominator;

}

# End