# Camera驱动

## Xc6130 Device Driver

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

module\_init(camera\_module\_init); //device driver入口函数

module\_exit(camera\_module\_exit);

kernel\drivers\media\i2c\camera\ xc6130\ xc6130.c //包含跟xc6130相关的接口函数

#include "module\_diff.h"

#include "../module\_comm/module\_comm.c"

#include "../module\_comm/module\_detect.c" //包含了camera驱动的公共部分

## SI + CSI Host Driver

kernel\drivers\media\i2c\camera\owl\_camera.c

late\_initcall(camera\_init); //host driver入口函数

module\_exit(camera\_exit);

kernel\drivers\media\i2c\camera\owl\_si.c

#include "module\_diff.h"

#include "../host\_comm/owl\_camera.c" //包含了host驱动的公共部分

# Struct

## struct soc\_camera\_link

|  |
| --- |
| struct soc\_camera\_link {  //下面这一部分等价与soc\_camera\_subdev\_desc  unsigned long flags;  void \*priv;  struct regulator\_bulk\_data \*regulators;  int num\_regulators;  int (\*power)(struct device \*, int);  int (\*reset)(struct device \*);  int (\*set\_bus\_param)(struct soc\_camera\_link \*, unsigned long flags);  unsigned long (\*query\_bus\_param)(struct soc\_camera\_link \*);  void (\*free\_bus)(struct soc\_camera\_link \*);  //下面这一部分等价与struct soc\_camera\_host\_desc  **int bus\_id;**  int i2c\_adapter\_id;  struct i2c\_board\_info \*board\_info;  const char \*module\_name;  int (\*add\_device)(struct soc\_camera\_device \*);  void (\*del\_device)(struct soc\_camera\_device \*);  }; |

## struct device

|  |
| --- |
| **struct device** {  **struct device \*parent;**  struct device\_private \*p;  struct kobject kobj;  **const char \*init\_name;** /\* initial name of the device \*/  const struct device\_type \*type;  struct mutex mutex;  struct bus\_type \*bus; /\* type of bus device is on \*/  **struct device\_driver \*driver**; /\* which driver has allocated this device \*/  **void \*platform\_data; //指向struct soc\_camera\_link**  struct dev\_pm\_info power;  struct dev\_pm\_domain \*pm\_domain;  u64 \*dma\_mask; /\* dma mask (if dma'able device) \*/  u64 coherent\_dma\_mask;  struct device\_dma\_parameters \*dma\_parms;  struct list\_head dma\_pools; /\* dma pools (if dma'ble) \*/  struct dma\_coherent\_mem \*dma\_mem; /\* internal for coherent mem override \*/  #ifdef CONFIG\_DMA\_CMA  struct cma \*cma\_area; /\* contiguous memory area for dma allocations \*/  #endif  struct dev\_archdata archdata;  struct device\_node \*of\_node; /\* associated device tree node \*/  struct acpi\_dev\_node acpi\_node; /\* associated ACPI device node \*/  dev\_t devt; /\* dev\_t, creates the sysfs "dev" \*/  u32 id; /\* device instance \*/  spinlock\_t devres\_lock;  struct list\_head devres\_head;  struct klist\_node knode\_class;  struct class \*class;  const struct attribute\_group \*\*groups; /\* optional groups \*/  void (\*release)(struct device \*dev);  struct iommu\_group \*iommu\_group;  } |

## struct platform\_device

**struct platform\_device** {

const char \*name; // "soc-camera-pdrv"

int id; // 0

bool id\_auto;

**struct device dev;**

u32 num\_resources;

struct resource \*resource;

const struct platform\_device\_id \*id\_entry;

struct mfd\_cell \*mfd\_cell;

struct pdev\_archdata archdata;

};

## struct device\_driver

|  |
| --- |
| struct device\_driver {  const char \*name;  struct bus\_type \*bus;  struct module \*owner;  const char \*mod\_name; /\* used for built-in modules \*/  bool suppress\_bind\_attrs; /\* disables bind/unbind via sysfs \*/  **const struct of\_device\_id \*of\_match\_table;**  const struct acpi\_device\_id \*acpi\_match\_table;  int (\*probe) (struct device \*dev);  int (\*remove) (struct device \*dev);  void (\*shutdown) (struct device \*dev);  int (\*suspend) (struct device \*dev, pm\_message\_t state);  int (\*resume) (struct device \*dev);  const struct attribute\_group \*\*groups;  const struct dev\_pm\_ops \*pm;  struct driver\_private \*p;  }; |

## struct platform\_driver

|  |
| --- |
| struct platform\_driver {  int (\*probe)(struct platform\_device \*);  int (\*remove)(struct platform\_device \*);  void (\*shutdown)(struct platform\_device \*);  int (\*suspend)(struct platform\_device \*, pm\_message\_t state);  int (\*resume)(struct platform\_device \*);  **struct device\_driver driver;**  const struct platform\_device\_id \*id\_table;  }; |

|  |
| --- |
|  |

|  |
| --- |
|  |

## struct soc\_camera\_device : icd

|  |
| --- |
| struct soc\_camera\_device {  struct list\_head list; /\* list of all registered devices \*/  struct soc\_camera\_desc \*sdesc;  **struct device \*pdev;** //pdev->dev: "soc-camera-pdrv"  **struct device \*parent;** // ici->v4l2\_dev.dev: " actions,owl-isp "  **struct device \*control;** /\* E.g., the i2c client \*/  s32 user\_width;  s32 user\_height;  u32 bytesperline; /\* for padding, zero if unused \*/  u32 sizeimage;  enum v4l2\_colorspace colorspace;  **unsigned char iface;** // platform\_device.dev. platform\_data.bus\_id  unsigned char devnum; /\* Device number per host \*/  struct soc\_camera\_sense \*sense; /\* See comment in struct definition \*/  **struct video\_device \*vdev;**  struct v4l2\_ctrl\_handler ctrl\_handler;  const struct soc\_camera\_format\_xlate \*current\_fmt;  struct soc\_camera\_format\_xlate \*user\_formats;  int num\_user\_formats;  enum v4l2\_field field; /\* Preserve field over close() \*/  void \*host\_priv; // struct camera\_param  /\* soc\_camera.c private count. Only accessed with .host\_lock held \*/  int use\_count;  struct file \*streamer; /\* stream owner \*/  union {  struct videobuf\_queue vb\_vidq;  struct vb2\_queue vb2\_vidq;  };  } |

## struct camera\_dev

|  |
| --- |
| struct camera\_dev {  struct soc\_camera\_device \*icds[2];  unsigned int dvp\_mbus\_flags;  **struct soc\_camera\_host soc\_host;**  int skip\_frames;  spinlock\_t lock; /\* Protects video buffer lists \*/  int irq;  struct pinctrl \*mfp;  struct sensor\_pwd\_info spinfo;  struct module\_regulators ir;  struct clk \*module\_clk\_parent;  struct clk \*module\_clk;  struct clk \*sensor\_clk\_parent[2];  struct clk \*sensor\_clk;  /\*csi clock src parent correspond to HOST\_MODULE\_CHANNEL\_0/1 (should be 0 or 1) \*/  struct clk \*csi\_clk\_parent;  struct clk \*csi\_clk;  /\*csi clock correspond to HOST\_MODULE\_CHANNEL\_0/1 (should be 0 or 1) \*/  struct clk \*ch\_clk[2];  }; |

## struct soc\_camera\_host : ici

|  |
| --- |
| struct soc\_camera\_host {  **struct v4l2\_device v4l2\_dev;**  struct list\_head list;  struct mutex host\_lock; /\* Protect pipeline modifications \*/  unsigned char nr; /\* Host number \*/  u32 capabilities;  void \*priv;  const char \*drv\_name;  struct soc\_camera\_host\_ops \*ops;  } |

## struct v4l2\_device

|  |
| --- |
| struct v4l2\_device {  **struct device \*dev;** //pdev->dev: " actions,owl-isp "  #if defined(CONFIG\_MEDIA\_CONTROLLER)  struct media\_device \*mdev;  #endif  /\* used to keep track of the registered subdevs \*/  struct list\_head subdevs;  /\* lock this struct; can be used by the driver as well if this  struct is embedded into a larger struct. \*/  spinlock\_t lock;  /\* unique device name, by default the driver name + bus ID \*/  char name[V4L2\_DEVICE\_NAME\_SIZE];  /\* notify callback called by some sub-devices. \*/  void (\*notify)(struct v4l2\_subdev \*sd,  unsigned int notification, void \*arg);  /\* The control handler. May be NULL. \*/  struct v4l2\_ctrl\_handler \*ctrl\_handler;  /\* Device's priority state \*/  struct v4l2\_prio\_state prio;  /\* BKL replacement mutex. Temporary solution only. \*/  struct mutex ioctl\_lock;  /\* Keep track of the references to this struct. \*/  struct kref ref;  /\* Release function that is called when the ref count goes to 0. \*/  void (\*release)(struct v4l2\_device \*v4l2\_dev);  } |
|  |

## struct camera\_param

|  |
| --- |
| struct camera\_param {  struct soc\_camera\_device \*icd;  struct list\_head capture;  struct videobuf\_buffer \*cur\_frm;  struct videobuf\_buffer \*prev\_frm;  struct completion wait\_stop;  int started;  /\* SI data offsets within croped by the S700 camera output \*/  unsigned int left;  unsigned int top;  /\* Client output, as seen by the S700 \*/  unsigned int width;  unsigned int height;  /\*  \* User window from S\_CROP / G\_CROP, produced by client cropping,  \* S700 cropping, mapped back onto the client  \* input window  \*/  struct v4l2\_rect subrect;  /\* Camera cropping rectangle \*/  struct v4l2\_rect rect;  const struct soc\_mbus\_pixelfmt \*extra\_fmt;  enum v4l2\_mbus\_pixelcode code;  unsigned long flags;  unsigned int skip\_frames;  int channel;  enum v4l2\_mbus\_type bus\_type;  int lane\_num;  int raw\_width;  int data\_type; /\*0 for yuv sensor,1 for raw-bayer senso\*/  int (\*ext\_cmd) (struct v4l2\_subdev \*sd, int cmd, void \*args);  int b\_splited\_capture;  int rb\_w;  int rb\_lsub\_col\_size;  int rb\_lright\_cut;  int rb\_lcol\_stride;  int rb\_lleft\_cut;  int rb\_rsub\_col\_size;  int rb\_rright\_cut;  int rb\_rcol\_stride;  int rb\_rleft\_cut;  int rb\_rows;  int rb\_cnt;  phys\_addr\_t p\_raw\_addr;  int n\_crop\_x;  int n\_crop\_y;  int n\_r\_skip\_num;  int n\_r\_skip\_size;  int real\_w;  int real\_h;  /\*  \* 0 disable raw store  \* 1 enable raw store in preview mode  \* 2 enable raw store in capture mode  \* 3 raw store finished oneFrame  \* 4 raw store finished twoFrame  \* 5 ....  \*/  unsigned long b\_raw\_store\_status;  } |

## struct i2c\_client

|  |
| --- |
| struct i2c\_client {  unsigned short flags; /\* div., see below \*/  unsigned short addr; /\* chip address - NOTE: 7bit \*/  /\* addresses are stored in the \*/  /\* \_LOWER\_ 7 bits \*/  char name[I2C\_NAME\_SIZE];  **// i2c\_board\_info->type: XC6130**  struct i2c\_adapter \*adapter; /\* the adapter we sit on \*/  struct i2c\_driver \*driver; /\* and our access routines \*/  **struct device dev; /\* the device structure \*/**  int irq; /\* irq issued by device \*/  struct list\_head detected;  }; |

## struct i2c\_driver

|  |
| --- |
| struct i2c\_driver {  unsigned int class;  int (\*attach\_adapter)(struct i2c\_adapter \*) \_\_deprecated;  int (\*probe)(struct i2c\_client \*, const struct i2c\_device\_id \*);  int (\*remove)(struct i2c\_client \*);  void (\*shutdown)(struct i2c\_client \*);  int (\*suspend)(struct i2c\_client \*, pm\_message\_t mesg);  int (\*resume)(struct i2c\_client \*);  void (\*alert)(struct i2c\_client \*, unsigned int data);  int (\*command)(struct i2c\_client \*client, unsigned int cmd, void \*arg);  **struct device\_driver driver;**  const struct i2c\_device\_id \*id\_table;  int (\*detect)(struct i2c\_client \*, struct i2c\_board\_info \*);  const unsigned short \*address\_list;  struct list\_head clients;  } |

## struct i2c\_device\_id

|  |
| --- |
| struct i2c\_device\_id {  char name[I2C\_NAME\_SIZE];  kernel\_ulong\_t driver\_data; /\* Data private to the driver \*/  }; |

## struct camera\_module\_priv

|  |
| --- |
| struct camera\_module\_priv {  **struct v4l2\_subdev subdev;**  struct v4l2\_ctrl\_handler hdl;  struct module\_info \*info;  const struct module\_color\_format \*cfmt;  const struct camera\_module\_win\_size \*win;  int model;  int pcv\_mode;  int flip\_flag;  unsigned short auto\_white\_balance;  unsigned short exposure;  unsigned short power\_line\_frequency;  unsigned short white\_balance\_temperature;  unsigned short colorfx;  unsigned short exposure\_auto;  unsigned short scene\_exposure;  exposure\_param\_t preview\_exposure\_param;  exposure\_param\_t capture\_exposure\_param;  struct v4l2\_afregion af\_region;  enum v4l2\_flash\_led\_mode flash\_led\_mode;  enum af\_status af\_status;  enum af\_mode af\_mode;  }; |

## struct v4l2\_subdev

|  |
| --- |
| struct v4l2\_subdev {  struct list\_head list;  struct module \*owner;  u32 flags;  **struct v4l2\_device \*v4l2\_dev;**  **const struct v4l2\_subdev\_ops \*ops;**  /\* Never call these internal ops from within a driver! \*/  const struct v4l2\_subdev\_internal\_ops \*internal\_ops;  /\* The control handler of this subdev. May be NULL. \*/  struct v4l2\_ctrl\_handler \*ctrl\_handler;  /\* name must be unique \*/  char name[V4L2\_SUBDEV\_NAME\_SIZE];  /\* can be used to group similar subdevs, value is driver-specific \*/  u32 grp\_id;  /\* pointer to private data \*/  void \*dev\_priv;  void \*host\_priv;  /\* subdev device node \*/  struct video\_device \*devnode;  }; |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

# Ops

## struct 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,  }; |

## struct v4l2\_file\_operations

|  |
| --- |
| 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

|  |
| --- |
| 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,  #ifdef CONFIG\_VIDEO\_ADV\_DEBUG  .vidioc\_g\_register = soc\_camera\_g\_register,  .vidioc\_s\_register = soc\_camera\_s\_register,  #endif  }; |

## struct v4l2\_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,  }; |
| 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,  }; |
| 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,  }; |
| static struct v4l2\_subdev\_sensor\_ops module\_subdev\_sensor\_ops = {  .g\_skip\_frames = camera\_module\_g\_skip\_frames,  }; |

## struct v4l2\_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,  }; |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

# soc\_camera\_pdrv\_probe -- **构建**soc\_camera\_device**，并**添加到device链表中

**device name: "soc-camera-pdrv"**

**构建platform \_device(soc-camera-pdrv)，注册到platform\_device\_register()中**

**构建platform \_driver(soc-camera-pdrv)，注册到module\_platform\_driver ()中**

***执行soc\_camera\_pdrv\_probe()***

**soc\_camera\_pdrv\_probe()会构建soc\_camera\_device，并将其设备添加到device链表中**

module\_comm.c

|  |  |
| --- | --- |
| struct platform\_device asoc\_camera\_device  .name = "soc-camera-pdrv",  .id = 0/1, //rear or front  .dev   |  | | --- | | struct device dev;  ->platform\_data = &camera\_module\_link,  .release = camera\_dummy\_release,  ->of\_node = dsc->dn; //"sensor\_common"  ->p->driver\_data = data; // struct soc\_camera\_device \*icd | |

Soc\_camera.c

|  |  |
| --- | --- |
| struct platform\_driver soc\_camera\_pdrv =  .probe = soc\_camera\_pdrv\_probe,  .remove = soc\_camera\_pdrv\_remove,  .driver   |  | | --- | | struct device\_driver driver;  ->name = "soc-camera-pdrv",  .owner = THIS\_MODULE, | |

Soc\_camera.c

|  |
| --- |
| struct soc\_camera\_device \*icd  .iface = sdesc->host\_desc.bus\_id; //rear or front  ->sdesc = sdesc; //sdesc = pdev->dev.platform\_data;  ->pdev = &pdev->dev; //"soc-camera-pdrv"  .user\_width = DEFAULT\_WIDTH;  .user\_height = DEFAULT\_HEIGHT;  .devnum = num;  .use\_count = 0;  ->host\_priv = NULL; |

Soc\_camera.c

|  |
| --- |
| LIST\_HEAD(devices);  list\_add\_tail(&icd->list, &devices); |

## platform device & driver name : "soc-camera-pdrv"

## device //module\_comm.c

### struct platform\_device -- 构建platform\_device

构建platform\_device和device设备

|  |
| --- |
| static struct platform\_device asoc\_camera\_device = {  .name = "soc-camera-pdrv",  .id = 0,  .dev = {  .platform\_data = &camera\_module\_link,  .release = camera\_dummy\_release,  },  };  static struct soc\_camera\_link camera\_module\_link = {  .bus\_id = 0,  .power = camera\_module\_power,  .reset = camera\_module\_reset,  .board\_info = &asoc\_i2c\_camera,  .i2c\_adapter\_id = 1, //id编号从0开始  .module\_name = "XC6130",  .priv = &camera\_module\_info,  };  struct module\_info camera\_module\_info = {  .flags = 0 | SENSOR\_FLAG\_8BIT | SENSOR\_FLAG\_YUV | SENSOR\_FLAG\_MIPI | SENSOR\_FLAG\_CHANNEL0 ,  .mipi\_cfg = &mipi\_sensor\_setting,  .isp\_cfg = &isp\_setting,  };  static mipi\_setting mipi\_sensor\_setting = {  .lan\_num = 1, //0~3  .contex0\_en = 1,  .contex0\_virtual\_num = 0,  .contex0\_data\_type = MIPI\_YUV422, //MIPI\_YUV422 MIPI\_RAW8 MIPI\_RAW10 MIPI\_RAW12  .clk\_settle\_time = 13, //13;//10, //8  .clk\_term\_time = 0, //8, //6  .data\_settle\_time = 13, //10, //8  .data\_term\_time = 0, //8, //6  .crc\_en = 1,  .ecc\_en = 1,  .hclk\_om\_ent\_en = 1, //1, non-continue  .lp11\_not\_chek = 0,  .hsclk\_edge = 0, //0: rising edge; 1: falling edge  .lane0\_map = 0,  .lane1\_map = 1,  .lane2\_map = 2,  .lane3\_map = 3,  .mipi\_en = 1,  .csi\_clk = 150000000,  }; |

### camera\_module\_init() -- 注册platform\_device

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

{

ret = sensor\_mod\_init(&camera\_module\_link, **&asoc\_camera\_device**, **&camera\_i2c\_driver**);

return ret;

}

static int **sensor\_mod\_init**(struct soc\_camera\_link \*link, struct platform\_device \***pdev,** struct i2c\_driver \***idrv**)

{

pdev->dev.of\_node = dsc->dn;

pdev->id = !(!dsc->rear);

ret = **platform\_device\_register(pdev);**

ret = **i2c\_add\_driver(idrv);**

}

## Driver //soc\_camera.c

### struct platform\_driver -- 构建platform\_driver

|  |
| --- |
| 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() -- 注册platform\_driver

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

## Probe //soc\_camera.c

### soc\_camera\_pdrv\_probe() -- 构建soc\_camera\_device

|  |
| --- |
| 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);**  } |

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

|  |
| --- |
| static LIST\_HEAD(devices); |
| static int soc\_camera\_device\_register(struct soc\_camera\_device \***icd**)  {  struct soc\_camera\_device \*ix;  int num = -1, i;  for (i = 0; i < 256 && num < 0; i++) {  num = i;  /\* Check if this index is available on this interface \*/  list\_for\_each\_entry(ix, **&devices**, list) {  if (ix->iface == **icd->iface** && ix->devnum == i) {  num = -1;  break;  }  }  }  if (num < 0)  /\*  \* ok, we have 256 cameras on this host...  \* man, stay reasonable...  \*/  return -ENOMEM;  icd->devnum = num;  icd->use\_count = 0;  icd->host\_priv = NULL;  **list\_add\_tail(&icd->list, &devices);**  return 0;  } |

# camera\_host\_probe() -- **构建**soc\_camera\_host**，并**添加到host链表中

**device name: " actions,owl-isp "**

**构建platform \_driver(atm9009a-camera-host)，注册到platform\_driver\_register中。**

***platform \_driver的of\_match\_table中添加device: actions,owl-isp.***

**构建platform \_device(actions,owl-isp)，通过dts注册到platform中。**

***执行camera\_host\_probe ()***

***camera\_host\_probe()构建struct camera\_dev。***

***struct camera\_dev中包含struct soc\_camera\_host，最终将其添加到host链表中***

**struct soc\_camera\_host中包含struct v4l2\_device，最终将其注册到v4l2\_device\_register()**

owl\_camera.c

|  |  |  |
| --- | --- | --- |
| struct platform\_driver soc\_camera\_pdrv =  .driver   |  |  | | --- | --- | | struct device\_driver driver;  .name = "atm9009a-camera-host",  .owner = THIS\_MODULE,  .pm = &camera\_dev\_pm\_ops,  .of\_match\_table   |  | | --- | | struct of\_device\_id \*of\_match\_table  .compatible = "actions,owl-isp" | |   .probe = camera\_host\_probe,  .remove = camera\_remove, |

Gt7\_ebox.dts

|  |
| --- |
| isp0: isp@e0268000 {  compatible = "**actions,owl-isp"**;  } |
| struct platform\_device asoc\_camera\_device  .id = 0, //sip0  .dev   |  | | --- | | struct device dev; | |

owl\_camera.c

|  |  |  |  |
| --- | --- | --- | --- |
| struct camera\_dev \*cam\_dev;  .irq = irq;  .soc\_host   |  |  |  | | --- | --- | --- | | struct soc\_camera\_host soc\_host;  ->ops = &soc\_camera\_host\_ops;  ->priv = cam\_dev;  .nr = pdev->id;  ->drv\_name = "atm9009a-camera-host";  . v4l2\_dev   |  |  | | --- | --- | | struct v4l2\_device v4l2\_dev;  ->dev = &pdev->dev //"actions,owl-isp"   |  | | --- | | struct device \*dev; | |   struct v4l2\_device v4l2\_dev; | |

Soc\_camera.c

|  |
| --- |
| LIST\_HEAD(hosts);  list\_add\_tail(&ici->list, &hosts); |

## platform device & driver name : " actions,owl-isp "

## driver //owl\_camera.c

### struct platform\_driver -- 构建platform\_driver

|  |
| --- |
| static struct platform\_driver camera\_host\_driver = {  .driver = {  .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 = **"actions,owl-isp"**,},  {},  }; |

### camera\_init() -- 注册platform\_driver

|  |
| --- |
| static int \_\_init camera\_init(void)  {  ret = host\_module\_init();  ret = **platform\_driver\_register**(&camera\_host\_driver);  return ret;  } |

## Device //gt7\_ebox.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";  };  isp0: isp@e0268000 {  mirror\_flip = <3>; /\* bit-0:rear camera; bit-1:front camera; 0: no flip,no mirror; 1: no flip,horizontal mirror; 2:vertical flip,no mirror; 3: vertical flip,horizontal mirror \*/  dvdd-gpios = <&gpiob 8 1>; /\* GPIOB8 \*/  dvdd-src = "regulator"; /\* only: "gpio", "regulator" \*/  dvdd-regulator = "ldo7";  dvdd-regulator-scope = <1800000 18100000>; /\* uV \*/  avdd-src = "gpio"; /\* only: "gpio", "regulator" \*/  avdd-gpios = <&gpioa 28 1>; /\* GPIOA28 \*/  dovdd-src = "gpio"; /\* only: "gpio", "regulator" \*/  dovdd-gpios = <&gpiob 0 1>; /\* GPIOB0 \*/  board\_type = "evb";/\* "ces" or "evb"\*/  sensors = "front"; /\* "dual", "front" or "rear" \*/  //pwdn-rear-gpios = <&gpioa 27 1>; /\* GPIOA27 \*/  //pwdn-front-gpios = <&gpiob 3 1>; /\* GPIO3 \*/  reset-gpios = <&gpiob 3 1>; /\*GPIOB3\*/  pinctrl-names = "default";  pinctrl-0 = <&mipi\_csi\_state\_default>;  status = "okay"; /\* create platform-device \*/  }; |

## Probe //owl\_camera.c

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

|  |
| --- |
| static int camera\_host\_probe(struct platform\_device \*pdev)  {  struct device\_node \*dn = pdev->dev.of\_node;  struct device \*dev = &pdev->dev;  **pdev->id = of\_alias\_get\_id(dn, "isp");**  //创建struct camera\_dev  struct camera\_dev cam\_dev = cam\_dev\_alloc(&pdev->dev, dn);  attach\_sensor\_pwd\_info(&pdev->dev, &cam\_dev->spinfo, pdev->id);  //获取dts中的io地址  struct resource \* module\_mem = platform\_get\_resource(pdev, IORESOURCE\_MEM, 0);  GMODULEMAPADDR = devm\_ioremap(dev, module\_mem->start,  resource\_size(module\_mem));  struct resource \*csi0\_mem = platform\_get\_resource(pdev, IORESOURCE\_MEM, 1);  GCSI1MAPADDR = devm\_ioremap(dev, csi0\_mem->start,  resource\_size(csi0\_mem));  GCMUMAPADDR = ioremap(CMU\_BASE, 0xfc);    //获取dts中的irq信息  unsigned int irq = platform\_get\_irq(pdev, 0);  cam\_dev->irq = irq;  err = devm\_request\_irq(&pdev->dev, cam\_dev->irq, camera\_host\_isr,  IRQF\_DISABLED, CAM\_HOST\_NAME, cam\_dev);    //注册soc\_caemra\_host  **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;**  **soc\_host->nr = pdev->id**;  soc\_host->drv\_name = **"atm9009a-camera-host"**;  err = **soc\_camera\_host\_register**(soc\_host);  return 0;  } |

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

|  |
| --- |
| static LIST\_HEAD(hosts); |
| int soc\_camera\_host\_register(struct soc\_camera\_host \***ici**)  {  struct soc\_camera\_host \*ix;  mutex\_lock(&list\_lock);  list\_for\_each\_entry(ix, **&hosts,** list) {  if (ix->nr == **ici->nr**) {  ret = -EBUSY;  goto edevreg;  }  }  ret = **v4l2\_device\_register**(ici->v4l2\_dev.dev, &ici->v4l2\_dev);  if (ret < 0)  goto edevreg;  **list\_add\_tail(&ici->list, &hosts);**  mutex\_unlock(&list\_lock);  mutex\_init(&ici->host\_lock);  **scan\_add\_host(ici);**  return 0;  } |

# soc\_camera\_probe()

**device name: " atm9009a-camera-host "**

**构建soc\_camera\_device，并将其设备添加到device链表中。**

**构建soc\_camera\_host，并将其设备添加到host链表中。**

**调用**scan\_add\_host()，通过**icd->iface == ici->nr寻找配对的soc\_camera\_device和soc\_camera\_host。找到则调用**soc\_camera\_probe()。

soc\_camera\_probe()构建video\_device，并注册video\_register\_device()。

soc\_camera\_probe()构建i2c\_client，并

Soc\_camera.c

|  |  |  |
| --- | --- | --- |
| struct soc\_camera\_device \*icd  .iface = sdesc->host\_desc.bus\_id; //rear or front  ->sdesc = sdesc; //sdesc = pdev->dev.platform\_data;  ->pdev = &pdev->dev; //"soc-camera-pdrv"  ->parent = ici->v4l2\_dev.dev; //"actions,owl-isp"  ->control = &client->dev; //"XC6130"  .user\_width = DEFAULT\_WIDTH;  .user\_height = DEFAULT\_HEIGHT;  .devnum = num;  .use\_count = 0;  ->host\_priv = cam\_param; //struct camera\_param \*cam\_param;;  . ctrl\_handler   |  | | --- | | struct v4l2\_ctrl\_handler ctrl\_handler; |   vdev = video\_device\_alloc();   |  | | --- | | struct video\_device \*vdev;  ->name = ici->drv\_name; //"atm9009a-camera-host"  ->parent = icd->pdev; //"soc-camera-pdrv"  .current\_norm = V4L2\_STD\_UNKNOWN;  ->fops = &soc\_camera\_fops;  ->ioctl\_ops = &soc\_camera\_ioctl\_ops;  ->release = video\_device\_release;  .tvnorms = V4L2\_STD\_UNKNOWN;  ->ctrl\_handler = &icd->ctrl\_handler;  ->lock = &ici->host\_lock; | |

owl\_camera.c

|  |  |  |
| --- | --- | --- |
| struct soc\_camera\_host soc\_host;  ->ops = &soc\_camera\_host\_ops;  ->priv = cam\_dev; //cam\_dev->icds[channel] = icd;  .nr = pdev->id;  ->drv\_name = "atm9009a-camera-host";  . v4l2\_dev   |  |  | | --- | --- | | struct v4l2\_device v4l2\_dev;  ->dev = &pdev->dev //"actions,owl-isp"   |  | | --- | | struct device \*dev; | |   struct v4l2\_device v4l2\_dev; |

|  |  |
| --- | --- |
| struct i2c\_client \*client = kzalloc(sizeof \*client, GFP\_KERNEL);  ->adapter = adap; // adap = i2c\_get\_adapter(shd->i2c\_adapter\_id)  dev.flags = info->flags;  .addr = info->addr;  .irq = info->irq;  .name = info->type  . dev   |  | | --- | | struct device dev;  .platform\_data = info->platform\_data;  .parent = &client->adapter->dev;  .bus = &i2c\_bus\_type;  .type = &i2c\_client\_type;  .of\_node = info->of\_node; | |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

## platform device & driver name : "atm9009a-camera-host"

## device //soc\_camera.c

### soc\_camera\_pdrv\_probe() -- 构建soc\_camera\_device

|  |
| --- |
| 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);**  } |

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

|  |
| --- |
| static LIST\_HEAD(devices); |
| static int soc\_camera\_device\_register(struct soc\_camera\_device \***icd**)  {  struct soc\_camera\_device \*ix;  int num = -1, i;  for (i = 0; i < 256 && num < 0; i++) {  num = i;  /\* Check if this index is available on this interface \*/  list\_for\_each\_entry(ix, **&devices**, list) {  if (ix->iface == **icd->iface** && ix->devnum == i) {  num = -1;  break;  }  }  }  if (num < 0)  /\*  \* ok, we have 256 cameras on this host...  \* man, stay reasonable...  \*/  return -ENOMEM;  icd->devnum = num;  icd->use\_count = 0;  icd->host\_priv = NULL;  **list\_add\_tail(&icd->list, &devices);**  return 0;  } |

## Driver //soc\_camera.c

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

|  |
| --- |
| static int camera\_host\_probe(struct platform\_device \*pdev)  {  struct device\_node \*dn = pdev->dev.of\_node;  struct device \*dev = &pdev->dev;  **pdev->id = of\_alias\_get\_id(dn, "isp");**  //创建struct camera\_dev  struct camera\_dev cam\_dev = cam\_dev\_alloc(&pdev->dev, dn);  attach\_sensor\_pwd\_info(&pdev->dev, &cam\_dev->spinfo, pdev->id);  //获取dts中的io地址  struct resource \* module\_mem = platform\_get\_resource(pdev, IORESOURCE\_MEM, 0);  GMODULEMAPADDR = devm\_ioremap(dev, module\_mem->start,  resource\_size(module\_mem));  struct resource \*csi0\_mem = platform\_get\_resource(pdev, IORESOURCE\_MEM, 1);  GCSI1MAPADDR = devm\_ioremap(dev, csi0\_mem->start,  resource\_size(csi0\_mem));  GCMUMAPADDR = ioremap(CMU\_BASE, 0xfc);    //获取dts中的irq信息  unsigned int irq = platform\_get\_irq(pdev, 0);  cam\_dev->irq = irq;  err = devm\_request\_irq(&pdev->dev, cam\_dev->irq, camera\_host\_isr,  IRQF\_DISABLED, CAM\_HOST\_NAME, cam\_dev);    //注册soc\_caemra\_host  **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;**  **soc\_host->nr = pdev->id**;  soc\_host->drv\_name = **"atm9009a-camera-host"**;  err = **soc\_camera\_host\_register**(soc\_host);  return 0;  } |

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

|  |
| --- |
| static LIST\_HEAD(hosts); |
| int soc\_camera\_host\_register(struct soc\_camera\_host \***ici**)  {  struct soc\_camera\_host \*ix;  mutex\_lock(&list\_lock);  list\_for\_each\_entry(ix, **&hosts,** list) {  if (ix->nr == **ici->nr**) {  ret = -EBUSY;  goto edevreg;  }  }  ret = **v4l2\_device\_register**(ici->v4l2\_dev.dev, &ici->v4l2\_dev);  if (ret < 0)  goto edevreg;  **list\_add\_tail(&ici->list, &hosts);**  mutex\_unlock(&list\_lock);  mutex\_init(&ici->host\_lock);  **scan\_add\_host(ici);**  return 0;  } |

## Probe //soc\_camera.c

### scan\_add\_host()

|  |
| --- |
| static void scan\_add\_host(struct soc\_camera\_host \*ici)  {  struct soc\_camera\_device \*icd;  mutex\_lock(&list\_lock);  list\_for\_each\_entry(icd, &devices, list) {  if (**icd->iface == ici->nr**) {  icd->parent = ici->v4l2\_dev.dev;  **soc\_camera\_probe**(icd);  }  }  mutex\_unlock(&list\_lock);  } |

### soc\_camera\_probe()

|  |
| --- |
| static int soc\_camera\_probe(struct soc\_camera\_device \*icd)  {  struct soc\_camera\_host \*ici = to\_soc\_camera\_host(icd->parent);  struct soc\_camera\_desc \*sdesc = to\_soc\_camera\_desc(icd);  struct soc\_camera\_host\_desc \*shd = &sdesc->host\_desc;  struct soc\_camera\_subdev\_desc \*ssdd = &sdesc->subdev\_desc;  struct device \*control = NULL;  struct v4l2\_subdev \*sd;  struct v4l2\_mbus\_framefmt mf;  int ret;  ret = v4l2\_ctrl\_handler\_init(&icd->ctrl\_handler, 16);  if (ssdd->reset)  ssdd->reset(icd->pdev);  ret = **ici->ops->add**(icd); // icd->host\_priv = cam\_param: 构建struct camera\_param  ret = video\_dev\_create(icd); // icd->vdev = vdev: 构建struct video\_device  if (shd->board\_info) {  ret = soc\_camera\_init\_i2c(icd, sdesc);  } else if (!shd->add\_device || !shd->del\_device) {  ret = -EINVAL;  } else {  if (shd->module\_name)  ret = request\_module(shd->module\_name);  ret = shd->add\_device(icd);  control = to\_soc\_camera\_control(icd);  if (!control || !control->driver || !dev\_get\_drvdata(control) || \  !try\_module\_get(control->driver->owner)) {  shd->del\_device(icd);  ret = -ENODEV;  }  }  sd = soc\_camera\_to\_subdev(icd);  sd->grp\_id = soc\_camera\_grp\_id(icd);  v4l2\_set\_subdev\_hostdata(sd, icd);  ret = v4l2\_ctrl\_add\_handler(&icd->ctrl\_handler, sd->ctrl\_handler, NULL);  ret = soc\_camera\_init\_user\_formats(icd);  icd->field = V4L2\_FIELD\_ANY;  mutex\_lock(&ici->host\_lock);  ret = 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);  mutex\_unlock(&ici->host\_lock);  return 0; |

#### video\_dev\_create() – 构建video\_device

|  |
| --- |
| static int video\_dev\_create(struct soc\_camera\_device \*icd)  {  struct soc\_camera\_host \*ici = to\_soc\_camera\_host(icd->parent);    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;  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;  return 0;  } |

#### soc\_camera\_video\_start() –注册video\_device

|  |
| --- |
| static int soc\_camera\_video\_start(struct soc\_camera\_device \*icd)  {  const struct device\_type \*type = icd->vdev->dev.type;  ret = **video\_register\_device**(icd->vdev, VFL\_TYPE\_GRABBER, -1);  icd->vdev->dev.type = type;  return 0;  } |

|  |
| --- |
|  |

# camera\_module\_probe

## I2c device & driver name : "XC6130"

## Driver

### struct i2c\_driver -- 构建i2c\_driver

|  |
| --- |
| static struct i2c\_driver camera\_i2c\_driver = {  .driver = {  .name = "XC6130",  },  .probe = camera\_module\_probe,  .suspend = camera\_module\_suspend,  .resume = camera\_module\_resume,  .remove = camera\_module\_remove,  **.id\_table = camera\_module\_id,**  }; |
| static const struct i2c\_device\_id camera\_module\_id[] = {  { "**XC6130**", 0 },  { }  }; |

### camera\_module\_init() -- 注册i2c\_driver

|  |
| --- |
| static int \_\_init **camera\_module\_init**(void)  {  ret = sensor\_mod\_init(&camera\_module\_link, **&asoc\_camera\_device**, **&camera\_i2c\_driver**);  return ret;  }  static int **sensor\_mod\_init**(struct soc\_camera\_link \*link, struct platform\_device \***pdev,** struct i2c\_driver \***idrv**)  {  pdev->dev.of\_node = dsc->dn;  pdev->id = !(!dsc->rear);  ret = **platform\_device\_register(pdev);**  ret = **i2c\_add\_driver(idrv);**  } |
| #define i2c\_add\_driver(driver) \  **i2c\_register\_driver**(THIS\_MODULE, driver) |
| int i2c\_register\_driver(struct module \*owner, struct i2c\_driver \*driver)  {  driver->driver.owner = owner;  driver->driver.**bus = &i2c\_bus\_type;**  res = **driver\_register**(&driver->driver);  i2c\_for\_each\_dev(driver, \_\_process\_new\_driver);  return 0;  } |

## device

### struct i2c\_board\_info

|  |
| --- |
| static struct soc\_camera\_link camera\_module\_link = {  .bus\_id = 0,  .power = camera\_module\_power,  .reset = camera\_module\_reset,  **.board\_info = &asoc\_i2c\_camera,**  .i2c\_adapter\_id = 1, //id编号从0开始  .module\_name = CAMERA\_MODULE\_NAME,  .priv = &camera\_module\_info,  }; |
| static struct i2c\_board\_info asoc\_i2c\_camera = {  I2C\_BOARD\_INFO(CAMERA\_MODULE\_NAME, MODULE\_I2C\_REG\_ADDRESS),  }; |
| #define I2C\_BOARD\_INFO(dev\_type, dev\_addr) \  .type = dev\_type, .addr = (dev\_addr) |

### soc\_camera\_probe()

|  |
| --- |
| static int soc\_camera\_probe(struct soc\_camera\_device \*icd)  {  struct soc\_camera\_desc \*sdesc = to\_soc\_camera\_desc(icd);  struct soc\_camera\_host\_desc \*shd = &sdesc->host\_desc;  if (shd->board\_info) {  ret = **soc\_camera\_init\_i2c**(icd, sdesc);  }  } |
| static int soc\_camera\_init\_i2c(struct soc\_camera\_device \*icd, struct soc\_camera\_desc \*sdesc)  {  **struct i2c\_client** \*client;  struct soc\_camera\_host \*ici = to\_soc\_camera\_host(icd->parent);  struct soc\_camera\_host\_desc \*shd = &sdesc->host\_desc;  struct i2c\_adapter \*adap = i2c\_get\_adapter(shd->i2c\_adapter\_id);  **struct v4l2\_subdev** \*subdev;  shd->board\_info->platform\_data = &sdesc->subdev\_desc;  subdev = **v4l2\_i2c\_new\_subdev\_board(**&ici->v4l2\_dev, adap, shd->board\_info, NULL);  client = v4l2\_get\_subdevdata(subdev);  icd->control = &client->dev;  return 0;  } |
| 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 v4l2\_subdev \*sd = NULL;  struct i2c\_client \*client;  request\_module(I2C\_MODULE\_PREFIX "%s", info->type);  if (info->addr == 0 && probe\_addrs)  client = i2c\_new\_probed\_device(adapter, info, probe\_addrs, NULL);  else  client = **i2c\_new\_device**(adapter, info);  /\* Lock the module so we can safely get the v4l2\_subdev pointer \*/  try\_module\_get(client->driver->driver.owner);  sd = i2c\_get\_clientdata(client);  if (**v4l2\_device\_register\_subdev**(v4l2\_dev, sd))  sd = NULL;  module\_put(client->driver->driver.owner);  return 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;  if (info->archdata)  client->dev.archdata = \*info->archdata;  client->flags = info->flags;  client->addr = info->addr;  client->irq = info->irq;  strlcpy(client->name, info->type, sizeof(client->name));  status = i2c\_check\_client\_addr\_validity(client);  status = i2c\_check\_addr\_busy(adap, client->addr);  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));  status = **device\_register**(&client->dev);  return client;  } |

## Bus

### struct bus\_type -- i2c\_bus\_type

|  |
| --- |
| struct bus\_type i2c\_bus\_type = {  .name = "i2c",  .match = i2c\_device\_match,  .probe = i2c\_device\_probe,  .remove = i2c\_device\_remove,  .shutdown = i2c\_device\_shutdown,  .pm = &i2c\_device\_pm\_ops,  }; |

### i2c\_device\_match()

|  |
| --- |
| static int i2c\_device\_match(struct device \*dev, struct device\_driver \*drv)  {  struct i2c\_client \*client = i2c\_verify\_client(dev);  if (!client)  return 0;  struct i2c\_driver \*driver;  /\* Attempt an OF style match \*/  if (of\_driver\_match\_device(dev, drv))  return 1;  /\* Then ACPI style match \*/  if (acpi\_driver\_match\_device(dev, drv))  return 1;  driver = to\_i2c\_driver(drv);  /\* match on an id table if there is one \*/  **if (driver->id\_table)**  return **i2c\_match\_id(driver->id\_table, client)** != NULL;  return 0;  } |
| static const struct i2c\_device\_id \*i2c\_match\_id(const struct i2c\_device\_id \*id,  const struct i2c\_client \*client)  {  while (id->name[0]) {  if (strcmp(client->name, id->name) == 0) //name: XC6130  return id;  id++;  }  return NULL;  } |

### i2c\_device\_probe()

|  |
| --- |
| static int i2c\_device\_probe(struct device \*dev)  {  struct i2c\_client \*client = i2c\_verify\_client(dev);  struct i2c\_driver \*driver;  driver = to\_i2c\_driver(dev->driver);  client->driver = driver;  status = dev\_pm\_domain\_attach(&client->dev, true);  if (status != -EPROBE\_DEFER) {  status = **driver->probe**(client, i2c\_match\_id(driver->id\_table, client));  }  return status;  } |

## Probe

|  |
| --- |
| static int camera\_module\_probe(struct i2c\_client \*client, const struct i2c\_device\_id \*did)  {  struct soc\_camera\_subdev\_desc \*desc = soc\_camera\_i2c\_to\_desc(client);  struct i2c\_adapter \*adapter = to\_i2c\_adapter(client->dev.parent);  struct v4l2\_subdev \*subdev;    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;  ret = v4l2\_ctrl\_handler\_setup(&priv->hdl);    return ret;  } |

|  |
| --- |
| 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->flags |= V4L2\_SUBDEV\_FL\_IS\_I2C;  /\* the owner is the same as the i2c\_client's driver owner \*/  sd->owner = client->driver->driver.owner;  /\* i2c\_client and v4l2\_subdev point to one another \*/  v4l2\_set\_subdevdata(sd, client);  i2c\_set\_clientdata(client, sd);  /\* initialize name \*/  snprintf(sd->name, sizeof(sd->name), "%s %d-%04x",  client->driver->driver.name, i2c\_adapter\_id(client->adapter), client->addr);  } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |