Of\_display\_timing.c (drivers\video) 8354 2016-04-01

static int **of\_parse\_display\_timing**(const struct device\_node \*np,

struct display\_timing \*dt)

{

#if defined(CONFIG\_FB\_ROCKCHIP)

**if (!of\_property\_read\_u32(np, "lvds-format", &val))**

**dt->lvds\_format = val;**

}

struct display\_timings \***of\_get\_display\_timings**(struct device\_node \*np)

{

r = of\_parse\_display\_timing(entry, dt);

}

Rockchip\_screen.c (drivers\video\rockchip) 12948 2016-04-01

static int **rockchip\_prase\_timing\_dt**(struct device\_node \*np,

struct rockchip\_screen \*screen)

{

disp\_timing = **of\_get\_display\_timings**(display\_dev\_n);

**rockchip\_fb\_videomode\_from\_timing**(dt, screen);

}

static int **rockchip\_screen\_probe**(struct platform\_device \*pdev)

{

ret = **rockchip\_prase\_timing\_dt**(np, sfa\_screen);

**rockchip\_fb\_videomode\_from\_timing**(dt, screen);

}

static int **rockchip\_fb\_videomode\_from\_timing**(const struct display\_timing \*dt,

struct rockchip\_screen \*screen)

{

screen->mode.pixclock = dt->pixelclock.typ;

screen->mode.left\_margin = dt->hback\_porch.typ;

screen->mode.right\_margin = dt->hfront\_porch.typ;

screen->mode.xres = dt->hactive.typ;

screen->mode.hsync\_len = dt->hsync\_len.typ;

screen->mode.upper\_margin = dt->vback\_porch.typ;

screen->mode.lower\_margin = dt->vfront\_porch.typ;

screen->mode.yres = dt->vactive.typ;

screen->mode.vsync\_len = dt->vsync\_len.typ;

screen->type = dt->screen\_type;

screen->lvds\_format = dt->lvds\_format;

screen->face = dt->face;

screen->color\_mode = dt->color\_mode;

screen->dsp\_lut = dt->dsp\_lut;

screen->width = dt->width;

}

static int **parse\_timing\_property**(const struct device\_node \*np, const char \*name,

struct timing\_entry \*result)

{

#if defined(CONFIG\_FB\_ROCKCHIP)

if (!of\_property\_read\_u32(np, "swap-rg", &val))

dt->flags |= val ? DISPLAY\_FLAGS\_SWAP\_RG : 0;

if (!of\_property\_read\_u32(np, "swap-gb", &val))

dt->flags |= val ? DISPLAY\_FLAGS\_SWAP\_GB : 0;

if (!of\_property\_read\_u32(np, "swap-rb", &val))

dt->flags |= val ? DISPLAY\_FLAGS\_SWAP\_RB : 0;

if (!of\_property\_read\_u32(np, "screen-type", &val))

dt->screen\_type = val;

if (!of\_property\_read\_u32(np, "lvds-format", &val))

dt->lvds\_format = val;

if (!of\_property\_read\_u32(np, "out-face", &val))

dt->face = val;

if (!of\_property\_read\_u32(np, "color-mode", &val))

dt->color\_mode = val;

if (!of\_property\_read\_u32(np, "width", &val))

dt->width = val;

if (!of\_property\_read\_u32(np, "height", &val))

dt->height = val;

prop = of\_find\_property(np, "dsp-lut", &length);

if (prop) {

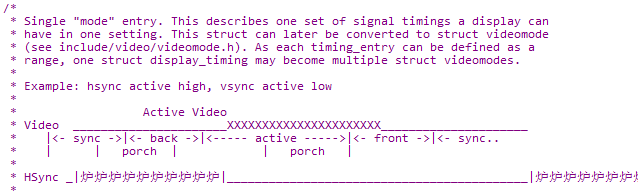
dt->dsp\_lut = kzalloc(length, GFP\_KERNEL);

if (dt->dsp\_lut)

ret = of\_property\_read\_u32\_array(np,

"dsp-lut", dt->dsp\_lut, length >> 2);

}



Dma\_alloc\_writecombine

Alloc\_dis\_buffer

Rockchip\_fb.c (drivers\video\rockchip) 76015 2016-04-01

int rockchip\_fb\_register(struct rockchip\_vop\_driver \*dev\_drv,

struct rockchip\_vop\_win \*vop\_win, int id)

{

init\_vop\_device\_driver(dev\_drv, vop\_win, id);

}

static int init\_vop\_device\_driver(struct rockchip\_vop\_driver \*dev\_drv,

struct rockchip\_vop\_win \*def\_win, int id)

{

dev\_drv->trsm\_ops = rockchip\_fb\_trsm\_ops\_get(screen->type);

}

struct rockchip\_fb\_trsm\_ops \*rockchip\_fb\_trsm\_ops\_get(u16 type)

{

switch (type) {

case **SCREEN\_RGB**:

case **SCREEN\_LVDS**:

case **SCREEN\_DUAL\_LVDS**:

ops = **trsm\_lvds\_ops**;

case SCREEN\_MIPI:

case SCREEN\_DUAL\_MIPI:

ops = trsm\_mipi\_ops;

}

Nanosilicon\_lvds.c (drivers\video\nanosilicon) 8366 2016-04-01

static struct rockchip\_fb\_trsm\_ops **trsm\_lvds\_ops** = {

.enable = **nanosilicon\_lvds\_enable**,

.disable = **nanosilicon\_lvds\_disable**,

};

Rockchip\_fb.c (drivers\video\rockchip) 76015 2016-04-01

int **rockchip\_fb\_register**(struct rockchip\_vop\_driver \*dev\_drv,

struct rockchip\_vop\_win \*vop\_win, int id)

{

**fb\_videomode\_to\_var**(&fbi->var, &dev\_drv->cur\_screen->mode);//固定参数

rockchip\_fb\_create\_sysfs(fbi);

**rockchip\_fb\_show\_copy\_from\_loader**(main\_fbi);

dev\_drv->vsync\_info.thread =

kthread\_run(**rockchip\_fb\_wait\_for\_vsync\_thread**,//帧同步信号

dev\_drv, "fb-vsync");

init\_kthread\_worker(&dev\_drv->update\_regs\_worker);

init\_kthread\_work(&dev\_drv->update\_regs\_work,

**rockchip\_fb\_update\_regs\_handler**);//刷新寄存器

main\_fbi->fbops->fb\_open(main\_fbi, 1);

显示linux的图片：

#if defined(CONFIG\_LOGO\_LINUX\_BMP)

if (**fb\_prepare\_bmp\_logo**(main\_fbi, FB\_ROTATE\_UR)) {

fb\_set\_cmap(&main\_fbi->cmap, main\_fbi);

fb\_show\_bmp\_logo(main\_fbi, FB\_ROTATE\_UR);

}

if (fb\_prepare\_logo(main\_fbi, FB\_ROTATE\_UR)) {

fb\_set\_cmap(&main\_fbi->cmap, main\_fbi);

fb\_show\_logo(main\_fbi, FB\_ROTATE\_UR);

}

main\_fbi->fbops->fb\_pan\_display(&main\_fbi->var, main\_fbi);

dev\_drv->ops->cfg\_done(dev\_drv);

}

static void **rockchip\_fb\_update\_regs\_handler**(struct kthread\_work \*work)

{

**rockchip\_fb\_update\_reg**(dev\_drv, data);

}

static void rockchip\_fb\_update\_reg(struct rockchip\_vop\_driver \*dev\_drv,

struct rockchip\_fb\_reg\_data \*regs)

{

ret = **rockchip\_fb\_get\_win\_from\_regs**(regs, win);

dev\_drv->ops->set\_par(dev\_drv, i);

dev\_drv->ops->pan\_display(dev\_drv, i);

mutex\_unlock(&dev\_drv->win\_cfg\_lock);

rockchip\_fb\_free\_reg\_data(dev\_drv, dev\_drv->last\_regs);

}

static int **rockchip\_fb\_open**(struct fb\_info \*info, int user)

{

win\_id = dev\_drv->ops->fb\_get\_win\_id(dev\_drv, info->fix.id);

/\* if this layer aready opened ,no need to reopen \*/

if (!dev\_drv->win[win\_id]->state)

**dev\_drv->ops->open**(dev\_drv, win\_id, 1);

}

Rockchip\_vop.c (drivers\video\rockchip\vop) 64986 2016-04-01

static struct rockchip\_vop\_drv\_ops vop\_drv\_ops = {

.open = **rockchip\_vop\_open**,

}

static int **rockchip\_vop\_open**(struct rockchip\_vop\_driver \*dev\_drv, int win\_id,

bool open)

{

struct vop\_device \*vop\_dev =

container\_of(dev\_drv, struct vop\_device, driver);

**rockchip\_vop\_pre\_init**(dev\_drv);

**rockchip\_vop\_clk\_enable**(vop\_dev);

**rockchip\_vop\_reg\_restore**(vop\_dev);

if (**support\_loader\_display**()) {

**rockchip\_vop\_set\_dclk**(dev\_drv);

**rockchip\_vop\_enable\_irq**(dev\_drv);

} else {

**rockchip\_vop\_mmu\_en**(dev\_drv, open);

**rockchip\_vop\_load\_screen**(dev\_drv, 1);

}

/\* set screen lut \*/

if (dev\_drv->cur\_screen->dsp\_lut)

**rockchip\_vop\_set\_lut**(dev\_drv,

dev\_drv->cur\_screen->dsp\_lut, 0);

}

if (win\_id < ARRAY\_SIZE(vop\_win))

**vop\_win\_enable**(vop\_dev, win\_id, open);

/\* when all layer closed,disable clk \*/

if ((!open) && (!vop\_dev->atv\_layer\_cnt))

**rockchip\_vop\_deinit**(vop\_dev);

}

第一帧开始的中断：

Rockchip\_vop.c (drivers\video\rockchip\vop) 64986 2016-04-01



第一帧开始的中断

static int rockchip\_vop\_pre\_init(struct rockchip\_vop\_driver \*dev\_drv)

{

/\* backup reg config at uboot \*/

vop\_read\_reg\_default\_cfg(vop\_dev);

}

loader-logo-on:1

[ 9.559709] create rockchip ion client

[ 9.559763] rockchip-fb rockchip-fb: rockchip framebuffer driver probe

[ 9.560614] stephen== screen->index = 1

[ 9.561353] dt->flags = 149

[ 9.561373] dt->screen\_type = 2

[ 9.561385] dt->lvds\_format = 0

[ 9.561397] dt->face = 0

[ 9.561408] dt->color\_mode = 0

[ 9.561420] dt->width = 0

[ 9.561432] dt->height = 0

[ 9.561443] dt->dsp\_lut = 0

[ 9.561460] stephen===dt = display\_timings\_get(disp\_timing, disp\_timing->native\_mode);stephen screen->mode.pixclock = 52000000

[ 9.561480] stephen screen->mode.left\_margin = 140

[ 9.561492] stephen screen->mode.right\_margin = 140

[ 9.561505] stephen screen->mode.xres = 1024

[ 9.561517] stephen screen->mode.hsync\_len = 40

[ 9.561529] stephen screen->mode.upper\_margin = 10

[ 9.561541] stephen screen->mode.lower\_margin= 20

[ 9.561553] stephen screen->mode.yres = 768

[ 9.561565] stephen screen->mode.vsync\_len = 5

[ 9.561577] stephen screen->type = 2

[ 9.561589] stephen screen->lvds\_format = 0

[ 9.561601] stephen screen->face= 0

[ 9.561787] stephen screen->color\_mode= 0

[ 9.561806] stephen screen->dsp\_lut = 0

[ 9.561819] stephen screen->width = 0

[ 9.561831] stephen screen->height = 0

[ 9.561843] stephen screen->pin\_dclk = 0

[ 9.561855] stephen screen->pin\_hsync = 0

[ 9.561867] stephen screen->pin\_den = 0

[ 9.561972] stephen#####pcl\_write offset:0x290,%value:0x1700, gpio+base+offset:0xc0868290

[ 9.562175] rockchip-screen screen.198: rockchip screen probe success

[ 9.563103] nanosilicon-lvds nanosilicon-lvds: nanosilicon lvds driver probe success

[ 9.563128] nanosilicon-lvds nanosilicon-lvds: set gpio 56 output high

graphics fb0: rockchip framebuffer registered:fb0

[ 9.469844] graphics fb1: rockchip framebuffer registered:fb1

[ 9.470157] graphics fb2: rockchip framebuffer registered:fb2

[ 9.470967] rockchip-vop vop0: vop0: **dclk**:0>>**fps**:384

[ 9.470992] rockchip-vop vop0: wakeup from standby!

[ 9.495474] graphics fb0: fb0:**phy**:10000000>>**vir**:c1581000>>**len**:0x900000

[ 9.495512] graphics fb0: copy fb data **1024 x 600**  from addr:**0x3f119200** to addr:**0x10000000**

[ 9.555334] rockchip-vop vop0: vop0 probe ok, iommu enabled

Rockchip\_fb.c (drivers\video\rockchip) 76015 2016-04-01

static int **rockchip\_fb\_show\_copy\_from\_loader**(struct fb\_info \*info)

{

dev\_drv->ops->get\_dsp\_addr(dev\_drv, dsp\_addr);

src = dsp\_addr[0];

dev\_info(info->dev, "copy fb data %d x %d from addr:0x%08x to addr:0x%08x\n",

**win->area[0].xact**, **win->area[0].yact**, **src**,

(**u32)info->fix.smem\_start**);

}

static int **rockchip\_fb\_alloc\_buffer**(struct fb\_info \*fbi, int fb\_id)

{

fb\_mem\_size = get\_fb\_size();

fb\_mem\_size = get\_rotate\_fb\_size();

if (rockchip\_fb\_alloc\_buffer\_by\_ion(fbi, fb\_mem\_size) < 0)

if (**rockchip\_fb\_alloc\_dma\_buffer**(fbi, fb\_mem\_size) < 0)

}

参数设置：

**hactive, vactive**: display resolution

- **hfront-porch, hback-porch, hsync-len**: horizontal display timing parameters in pixels

**vfront-porch, vback-porch, vsync-len:** vertical display timing parameters in lines

- clock-frequency: display clock in Hz

optional properties:

- **hsync-active:** hsync pulse is active low/high/ignored

- **vsync-active:** vsync pulse is active low/high/ignored

- **de-active:** data-enable pulse is active low/high/ignored

- **pixelclk-active**: with

- active high = drive pixel data on rising edge/ sample data on falling edge

- active low = drive pixel data on falling edge/ sample data on rising edge

- ignored = ignored

- **interlaced (bool)**: boolean to enable interlaced mode

- **doublescan (bool):** boolean to enable doublescan mode

- **doubleclk (bool):** boolean to enable doubleclock mode

All the optional properties that are not bool follow the following logic: <1>: high active

<0>: low active

omitted: not used on hardware

**//等待帧同步信号**

**Rockchip\_fb.c** (drivers\video\rockchip) 76015 2016-04-01

static int **rockchip\_fb\_wait\_for\_vsync\_thread**(void \*data)

{

while (!**kthread\_should\_stop**()) {

ktime\_t timestamp = dev\_drv->vsync\_info.timestamp;

int ret = **wait\_event\_interruptible**(dev\_drv->vsync\_info.wait,

(!ktime\_equal(timestamp,

dev\_drv->vsync\_info.timestamp) &&

dev\_drv->vsync\_info.active) ||

**dev\_drv->vsync\_info.irq\_stop**);

if (!ret)

**sysfs\_notify**(&fbi->dev->kobj, NULL, "vsync");

}

调试BUG：

graphics fb0: rockchip fb use 3 buffers

[ 28.691184] stephen rockchip\_fb\_set\_par var->pixclock = **2403**

[ 28.691210] lcdc0>>rockchip\_vop\_set\_par

[ 28.691210] >>**format:ARGB888**>>>**xact:1024**>>**yact:768**>>**xsize:1024**>>**ysize:768**

[ 28.691210] >>**xvir:1024**>>**yvir:2304**>>xpos:0>>ypos:0>>

[ 28.691214] stephen **win->area[0].yact= 768**

**lcdc0>>rockchip\_vop\_set\_par**

**[ 30.153603] >>format:XBGR888>>>xact:1024>>yact:768>>xsize:1024>>ysize:768**

**[ 30.153603] >>xvir:1024>>yvir:768>>xpos:0>>ypos:0>>**

**[ 30.223770] stephen rockchip\_fb\_check\_config\_var area\_par->yact= 768**

stephen rockchip\_fb\_set\_win\_par vop\_win->area[i].yact = 768

**[ 9.588868]**

**stephen===dt = display\_timings\_get(disp\_timing, disp\_timing->native\_mode);**

**stephen screen->mode.pixclock = 416000000**

**[ 9.593478] stephen rockchip\_fb\_register fbi->var.pixclock = 0**

**[ 9.593867] graphics fb0: rockchip framebuffer registered:fb0**

**[ 9.594835] stephen rockchip\_fb\_register fbi->var.pixclock = 0**

**[ 9.595175] graphics fb1: rockchip framebuffer registered:fb1**

**[ 9.595198] stephen rockchip\_fb\_register fbi->var.pixclock = 0**

**[ 9.595493] graphics fb2: rockchip framebuffer registered:fb2**

**[ 9.596203] stephen vop\_read\_reg\_default\_cfg win0->area[0].yact = 600,win1->area[0].yact = 1**

**[ 9.596203] stephen screen->mode.xres + screen->mode.left\_margin + screen->mode.right\_margin + screen->mode.hsync\_len;rockchip\_fb\_calc\_fps x = 1344,y= 806**

**[ 9.596321] stephen rockchip\_fb\_calc\_fps**

**[ 9.596347] rockchip-vop vop0: vop0: dclk:0>>fps:384**

**[ 9.596365] stephen rockchip\_vop\_set\_dclk screen->mode.pixclock = 416000000**

**[ 9.596379] stephen rockchip\_vop\_set\_dclk vop\_dev->pixclock = 2403**

**[ 28.691184] stephen rockchip\_fb\_set\_par var->pixclock = 2403**

**[ 28.691210] lcdc0>>rockchip\_vop\_set\_par**

**[ 28.691210] >>format:ARGB888>>>xact:1024>>yact:768>>xsize:1024>>ysize:768**

**[ 28.691210] >>xvir:1024>>yvir:2304>>xpos:0>>ypos:0>>**

**[ 28.691214] stephen win->area[0].yact= 768**

static int **init\_vop\_device\_driver**(struct rockchip\_vop\_driver \*dev\_drv,

struct rockchip\_vop\_win \*def\_win, int id)

{

//获取操作方法

dev\_drv->trsm\_ops = **rockchip\_fb\_trsm\_ops\_get**(screen->type);

}

unsigned int lcd\_face; /\* lcd rgb tye (i.e. RGB888) \*/

unsigned int screen\_type; /\* LVDS/MIPI/RGB/HDMI \*/

unsigned int lvds\_format;

struct fb\_videomode timing; /\* screen display timing\*/

unsigned int width; /\* physical size of lcd width \*/

unsigned int height; /\* physical size of lcd height \*/

unsigned char pin\_dclk; /\* Clock polarity \*/

unsigned char pin\_den; /\* Output Enable polarity \*/

unsigned char pin\_hsync; /\* Horizontal Sync polarity \*/

unsigned char pin\_vsync; /\* Vertical Sync polarity \*/

unsigned char swap\_rb;

unsigned char swap\_rg;

unsigned char swap\_gb;

**Rockchip\_vop.c** (drivers\video\rockchip\vop) 64986 2016-04-01

static int **rockchip\_vop\_early\_suspend**(struct rockchip\_vop\_driver \*dev\_drv)

{

if (**dev\_drv->suspend\_flag)**

return 0;

**dev\_drv->suspend\_flag** = **true;**

}

static int **rockchip\_vop\_early\_resume**(struct rockchip\_vop\_driver \*dev\_drv)

{

**dev\_drv->suspend\_flag** = **false**;

}

static void **rockchip\_vop\_shutdown**(struct platform\_device \*pdev)

{

**dev\_drv->suspend\_flag** = **true**;

}

Rockchip\_fb.c (drivers\video\rockchip) 76015 2016-04-01

static int rockchip\_fb\_pan\_display(struct fb\_var\_screeninfo \*var,

struct fb\_info \*info)

{

if (**dev\_drv->suspend\_flag**)

return 0;

}

static int rockchip\_fb\_set\_par(struct fb\_info \*info)

{

if (dev\_drv->suspend\_flag)

return 0;

}

static int rockchip\_fb\_update\_win\_config(struct fb\_info \*info,

struct rockchip\_fb\_win\_cfg\_data \*win\_data)

{

f (!(dev\_drv->suspend\_flag == 0)) {

pr\_err("%s: error update frame when suspend!!!\n", \_\_func\_\_);

}

static void rockchip\_fb\_free\_update\_reg(struct rockchip\_vop\_driver \*dev\_drv,

struct rockchip\_fb\_reg\_data \*regs,

bool timeline\_inc)

{

}

**Fbmem.c** (drivers\video) 46998 2016-04-01

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

{

return **do\_fb\_ioctl**(info, cmd, arg);

}

static long **do\_fb\_ioctl**(struct fb\_info \*info, unsigned int cmd,

unsigned long arg)

{

case FBIOBLANK:

ret = **fb\_blank**(info, arg);

}

Int **fb\_blank**(struct fb\_info \*info, int blank)

{

if (info->fbops->fb\_blank)

ret = info->fbops->fb\_blank(blank, info);

}

Rockchip\_fb.c (drivers\video\rockchip) 76015 2016-04-01

static struct fb\_ops fb\_ops = {

.**fb\_blank** = **rockchip\_fb\_blank**,

}

static int **rockchip\_fb\_blank**(int blank\_mode, struct fb\_info \*info)

{

win\_id = dev\_drv->ops->fb\_get\_win\_id(dev\_drv, fix->id);

dev\_drv->ops->blank(dev\_drv, win\_id, blank\_mode);

}

**Rockchip\_vop.c** (drivers\video\rockchip\vop) 64986 2016-04-01

static struct rockchip\_vop\_drv\_ops vop\_drv\_ops = {

.blank = rockchip\_vop\_blank,

}

static int **rockchip\_vop\_blank**(struct rockchip\_vop\_driver \*dev\_drv,

int win\_id, int blank\_mode)

{

switch (blank\_mode) {

case FB\_BLANK\_UNBLANK://0

rockchip\_vop\_early\_resume(dev\_drv);

break;

case FB\_BLANK\_NORMAL://1

**rockchip\_vop\_early\_suspend**(dev\_drv);

break;

default:

**rockchip\_vop\_early\_suspend**(dev\_drv);

break;

}

}

static int **rockchip\_vop\_early\_suspend**(struct rockchip\_vop\_driver \*dev\_drv)

{

rockchip\_vop\_standby(dev\_drv, true);

rockchip\_vop\_mmu\_en(dev\_drv, false);

rockchip\_vop\_clk\_disable(vop\_dev);

**rockchip\_disp\_pwr\_disable**(screen);

}

Rockchip\_screen.c (drivers\video\rockchip) 12948 2016-04-01

int **rockchip\_disp\_pwr\_disable**(struct rockchip\_screen \*screen)

{

if (screen->type != SCREEN\_MIPI) {

if (screen->power\_off)

screen->power\_off(screen);

}

}

static int **rockchip\_screen\_probe**(struct platform\_device \*pdev)

{

sfa\_screen->power\_on = rockchip\_screen\_power\_on;

sfa\_screen->power\_off = **rockchip\_screen\_power\_off**;

}

ret = mv\_svc\_reg\_write\_only(led->physio + offset, val,0xFFFFFFFF);

iowrite32(val, (char \*)led->mmio + offset);

Xgold-cpufreq.c (drivers\cpufreq) 15261 2016-04-01

static int xgold\_cpufreq\_probe(struct platform\_device \*pdev)

{

if (**cpufreq\_register\_driver**(&xgold\_cpufreq\_driver))

}

Imc\_idi\_btif.c (drivers\idi\peripherals) 74432 2016-04-01

struct imc\_idi\_btif\_platdata {

struct clk \*kernel;

struct clk \*bt\_78M;

struct clk \*bt\_104M;

struct clk \*bt\_rtc;

}

#define TX\_BUFFER\_SIZE 2048

#define **AG610\_BTIF** 0x03015201

#define AG620\_BTIF 0x03075202

**Prh\_cfg.c** (e:\sofia\soclib\devices\pm\power\_management\prh\src\board\_sf\_3gr\_garnet) 52502 2016-04-01

static const sPRH\_PER\_CFG\_INFO\_T prh\_cfg\_info[PRH\_PER\_NOF\_PER\_ID] =

{

static const sPRH\_PER\_CFG\_INFO\_T prh\_cfg\_info[PRH\_PER\_NOF\_PER\_ID] =

}

休眠流程：

worker\_thread

process\_one\_work

try\_to\_suspend

pm\_suspend

suspend\_devices\_and\_enter

dpm\_suspend\_start

dpm\_suspend

\_\_device\_suspend

platform\_drv\_shutdown

dpm\_run\_callback

platform\_pm\_suspend

intel\_otg\_pm\_suspend

intel\_otg\_suspend

EventHub

InputReader

UsbHostManager

AudioPolicyManager

AudioFlinger

AUD\_HAL

IAudioFlinger.cpp (frameworks\av\media\libmedia) 53711 2016-04-01

class BpAudioFlinger : public BpInterface<IAudioFlinger>

{

public:

BpAudioFlinger(const sp<IBinder>& impl)

: BpInterface<IAudioFlinger>(impl)

{

}

}

virtual audio\_module\_handle\_t loadHwModule(const char \*name)

{

Parcel data, reply;

data.writeInterfaceToken(IAudioFlinger::getInterfaceDescriptor());

data.writeCString(name);

remote()->transact(LOAD\_HW\_MODULE, data, &reply);

return (audio\_module\_handle\_t) reply.readInt32();

}

status\_t BnAudioFlinger::onTransact(uint32\_t code, const Parcel& data, Parcel\* reply, uint32\_t flags)

{

case LOAD\_HW\_MODULE: {

CHECK\_INTERFACE(IAudioFlinger, data, reply);

reply->writeInt32(loadHwModule(data.readCString()));

return NO\_ERROR;

} break;

}

AudioPolicyClientImpl.cpp (frameworks\av\services\audiopolicy) 7885 2016-04-01

audio\_module\_handle\_t AudioPolicyService::AudioPolicyClient::loadHwModule(const char \*name)

{

sp<IAudioFlinger> af = AudioSystem::get\_audio\_flinger();

if (af == 0) {

ALOGW("%s: could not get AudioFlinger", \_\_func\_\_);

return 0;

}

return af->loadHwModule(name);

}

AudioSystem.cpp (frameworks\av\media\libmedia) 34471 2016-04-01

// establish binder interface to AudioFlinger service

const sp<IAudioFlinger> AudioSystem::get\_audio\_flinger()

{

binder = sm->getService(String16("media.audio\_flinger"));

gAudioFlingerClient = new AudioFlingerClient();

gAudioFlinger = interface\_cast<IAudioFlinger>(binder);

afc = gAudioFlingerClient;

af = gAudioFlinger;

af->registerClient(afc);

}

loadHwModule位于：

AudioFlinger.cpp (frameworks\av\services\audioflinger)

AudioPolicyClientImpl.cpp (frameworks\av\services\audiopolicy) 7885 2016-04-01

AudioPolicyCompatClient.cpp (hardware\libhardware\_legacy\audio) 5787 2016-04-01

AudioPolicyCompatClient.h (hardware\libhardware\_legacy\audio) 4039 2016-04-01

AudioPolicyInterface.h (hardware\libhardware\_legacy\include\hardware\_legacy) 13834

AudioPolicyManager.cpp (frameworks\av\services\audiopolicy) 354604 2016-04-01

IAudioFlinger.cpp (frameworks\av\media\libmedia) 53711 2016-04-01

AudioFlinger.cpp (frameworks\av\services\audioflinger) 105431 2016-04-01

audio\_module\_handle\_t AudioFlinger::loadHwModule\_l(const char \*name)

{

int rc = load\_audio\_interface(name, &dev);

}

audio\_module\_handle\_t AudioFlinger::loadHwModule(const char \*name)

{

return loadHwModule\_l(name);

}

static int load\_audio\_interface(const char \*if\_name, audio\_hw\_device\_t \*\*dev)

{

rc = hw\_get\_module\_by\_class(AUDIO\_HARDWARE\_MODULE\_ID, if\_name, &mod);

rc = audio\_hw\_device\_open(mod, dev);

}

AudioSystem.java (frameworks\base\media\java\android\media) 27836 2016-04-01

public static String getOutputDeviceName(int device)

{

switch(device) {

case DEVICE\_OUT\_USB\_ACCESSORY:

return DEVICE\_OUT\_USB\_ACCESSORY\_NAME;

case DEVICE\_OUT\_USB\_DEVICE:

return DEVICE\_OUT\_USB\_DEVICE\_NAME;

}

}

AudioPolicyManager.cpp (frameworks\av\services\audiopolicy) 354604 2016-04-01

audio\_devices\_t AudioPolicyManager::getNewOutputDevice(audio\_io\_handle\_t output, bool fromCache)

{

if (outputDesc->isStrategyActive(STRATEGY\_REROUTING)) {

device = getDeviceForStrategy(STRATEGY\_REROUTING, fromCache);

ALOGV("getNewOutputDevice() selected device %x", device);

}

audio\_devices\_t AudioPolicyManager::getDeviceForStrategy(routing\_strategy strategy,

bool fromCache)

{

case STRATEGY\_FM:

device = getDeviceForStrategy(STRATEGY\_MEDIA, false /\*fromCache\*/);

}

enum {

AUDIO\_DEVICE\_NONE = 0x0,

AUDIO\_DEVICE\_OUT\_EARPIECE = 0x1,

AUDIO\_DEVICE\_OUT\_SPEAKER = 0x2,

AUDIO\_DEVICE\_OUT\_WIRED\_HEADSET = 0x4,

AUDIO\_DEVICE\_OUT\_WIRED\_HEADPHONE = 0x8,

/\* USB accessory mode: your Android device is a USB device and the dock is a USB host \*/

AUDIO\_DEVICE\_OUT\_USB\_ACCESSORY = 0x2000,

/\* USB host mode: your Android device is a USB host and the dock is a USB device \*/

AUDIO\_DEVICE\_OUT\_USB\_DEVICE = 0x4000,

}

AudioPolicyManager.cpp (frameworks\av\services\audiopolicy) 354604 2016-04-01

sp<AudioPolicyManager::DeviceDescriptor> AudioPolicyManager::DeviceVector::getDevice(

audio\_devices\_t type, String8 address) const

{

sp<DeviceDescriptor> device;

device = itemAt(i);

return device;

}

E/AudioPolicyManager(  188): STRATEGY\_DTMF: STRATEGY\_FM, device[4000]

E/AudioPolicyManager( 173): STRATEGY\_DTMF: STRATEGY\_FM, device[4]

EventHub.cpp (frameworks\native\services\inputflinger) 62623 2016-04-01

status\_t EventHub::openDeviceLocked(const char \*devicePath) {

}

音频输出通道的更新：

audio\_hw\_out\_dev\_update(out->dev);

Audio\_hw.c (hardware\intel\libaudio\_hal) 99327 2016-04-01

static int audio\_hw\_close\_out\_stream(

struct xgold\_stream\_out \*out)

{

ret = audio\_hw\_out\_dev\_update(out->dev);

}

int handle\_output\_stream\_routing\_update( struct xgold\_audio\_device\* p\_adev)

{

ret = audio\_hw\_out\_dev\_update(p\_adev);

}

static ssize\_t out\_write(

struct audio\_stream\_out \*stream,

const void\* buffer,

size\_t bytes)

{

ret = audio\_hw\_out\_dev\_update(adev);

}

static int adev\_set\_mode(

struct audio\_hw\_device \*dev,

audio\_mode\_t mode)

{

ret = audio\_hw\_out\_dev\_update(adev);

}

static int adev\_open(

const hw\_module\_t\* module,

const char\* name,

hw\_device\_t\*\* device)

{

audio\_hw\_out\_dev\_update(adev);

}

getOutputForAttr的代码分析：

IAudioPolicyService.h (frameworks\av\include\media) 8728 2016-04-01

virtual status\_t getOutputForAttr(const audio\_attributes\_t \*attr,

audio\_io\_handle\_t \*output,

audio\_session\_t session,

audio\_stream\_type\_t \*stream,

uint32\_t samplingRate = 0,

audio\_format\_t format = AUDIO\_FORMAT\_DEFAULT,

audio\_channel\_mask\_t channelMask = 0,

audio\_output\_flags\_t flags = AUDIO\_OUTPUT\_FLAG\_NONE,

const audio\_offload\_info\_t \*offloadInfo = NULL) = 0;

IAudioPolicyService.cpp (frameworks\av\media\libmedia) 50190 2016-04-01

virtual status\_t getOutputForAttr(const audio\_attributes\_t \*attr,

audio\_io\_handle\_t \*output,

audio\_session\_t session,

audio\_stream\_type\_t \*stream,

uint32\_t samplingRate,

audio\_format\_t format,

audio\_channel\_mask\_t channelMask,

audio\_output\_flags\_t flags,

const audio\_offload\_info\_t \*offloadInfo)

{

status\_t status = remote()->transact(GET\_OUTPUT\_FOR\_ATTR, data, &reply);

}

status\_t BnAudioPolicyService::onTransact(

uint32\_t code, const Parcel& data, Parcel\* reply, uint32\_t flags)

{

case GET\_OUTPUT\_FOR\_ATTR: {

status\_t status = getOutputForAttr(hasAttributes ? &attr : NULL,

&output, session, &stream,

samplingRate, format, channelMask,

flags, hasOffloadInfo ? &offloadInfo : NULL);

}

}

AudioTrack.cpp (frameworks\av\media\libmedia) 76381 2016-04-01

status\_t AudioTrack::createTrack\_l()

{

const sp<IAudioFlinger>& audioFlinger = AudioSystem::get\_audio\_flinger();

status\_t status = AudioSystem::getOutputForAttr(attr, &output,

(audio\_session\_t)mSessionId, &streamType,

mSampleRate, mFormat, mChannelMask,

mFlags, mOffloadInfo);

}

AudioSystem.h (frameworks\av\include\media) 18530 2016-04-01

static status\_t getOutputForAttr();

AudioSystem.cpp (frameworks\av\media\libmedia) 34471 2016-04-01

status\_t AudioSystem::getOutputForAttr(){

const sp<IAudioPolicyService>& aps = AudioSystem::get\_audio\_policy\_service();

return aps->getOutputForAttr(attr, output, session, stream,

samplingRate, format, channelMask,

flags, offloadInfo);

}

AudioPolicyInterfaceImplLegacy.cpp (frameworks\av\services\audiopolicy) 20206 2016-04-01

status\_t AudioPolicyService::getOutputForAttr(){

\*stream = audio\_attributes\_to\_stream\_type(attr);

\*output = getOutput(\*stream, samplingRate, format, channelMask,

flags, offloadInfo);

}

AudioPolicyInterfaceImpl.cpp (frameworks\av\services\audiopolicy) 22203 2016-04-01

status\_t AudioPolicyService::getOutputForAttr(){

return mAudioPolicyManager->getOutputForAttr(attr, output, session, stream, samplingRate,

format, channelMask, flags, offloadInfo);

}

IAudioPolicyService.cpp (frameworks\av\media\libmedia) 50190 2016-04-01

status\_t BnAudioPolicyService::onTransact(

uint32\_t code, const Parcel& data, Parcel\* reply, uint32\_t flags)

{

}

class BpAudioPolicyService : public BpInterface<IAudioPolicyService>

{

}

AudioManager.java (frameworks\base\media\java\android\media) 149824 2016-04-01

public static final String ACTION\_USB\_AUDIO\_DEVICE\_PLUG =

"android.media.action.USB\_AUDIO\_DEVICE\_PLUG";

UsbAudioManager.java (frameworks\base\services\usb\java\com\android\server\usb) 7376 2016-04-01

private void sendDeviceNotification(AudioDevice audioDevice, boolean enabled) {

// send a sticky broadcast containing current USB state

Intent intent = new Intent(AudioManager.ACTION\_USB\_AUDIO\_DEVICE\_PLUG);

AudioService.java (frameworks\base\media\java\android\media) 263236 2016-04-01

private class AudioServiceBroadcastReceiver extends BroadcastReceiver {

else if (action.equals(AudioManager.ACTION\_USB\_AUDIO\_DEVICE\_PLUG)) {

if (hasPlayback) {

outDevice = AudioSystem.DEVICE\_OUT\_USB\_DEVICE;

setWiredDeviceConnectionState(outDevice, state, params);

}

}

if (hasCapture) {

inDevice = AudioSystem.DEVICE\_IN\_USB\_DEVICE;

setWiredDeviceConnectionState(inDevice, state, params);

}

}

AudioManager.java (frameworks\base\media\java\android\media) 149824 2016-04-01

public boolean isWiredHeadsetOn() {

if (AudioSystem.getDeviceConnectionState(DEVICE\_OUT\_WIRED\_HEADSET,"")

== AudioSystem.DEVICE\_STATE\_UNAVAILABLE &&

AudioSystem.getDeviceConnectionState(DEVICE\_OUT\_WIRED\_HEADPHONE,"")

== AudioSystem.DEVICE\_STATE\_UNAVAILABLE) {

return false;

} else {

return true;

}

}

上报耳机事件

WiredAccessoryManager.java (frameworks\base\services\core\java\com\android\server) 19551 2016-04-01

private void setDeviceStateLocked(int headset,

int headsetState, int prevHeadsetState, String headsetName) {

mAudioManager.setWiredDeviceConnectionState(outDevice, state, headsetName);

}

MSG\_SET\_WIRED\_DEVICE\_CONNECTION\_STATE事件

AudioService.java (frameworks\base\media\java\android\media) 263236 2016-04-01

public void handleMessage(Message msg) {

case MSG\_SET\_WIRED\_DEVICE\_CONNECTION\_STATE:

onSetWiredDeviceConnectionState(msg.arg1, msg.arg2, (String)msg.obj);

mAudioEventWakeLock.release();

break;

}

private int checkSendBecomingNoisyIntent(int device, int state) {

mAudioHandler.hasMessages(MSG\_SET\_A2DP\_SINK\_CONNECTION\_STATE)

}

音频的通道设置流程：

AudioService.java (frameworks\base\media\java\android\media) 263236 2016-04-01

private boolean handleDeviceConnection(boolean connected, int device, String params) {

AudioSystem.setDeviceConnectionState(device,

AudioSystem.DEVICE\_STATE\_UNAVAILABLE,

mConnectedDevices.get(device));

}

AudioSystem.java (frameworks\base\media\java\android\media) 27836 2016-04-01

public static native int setDeviceConnectionState(int device, int state, String device\_address);

android\_media\_MediaProfiles.cpp (frameworks\base\media\jni) 16972 2016-04-01

static JNINativeMethod gMethods[] = {

{"setDeviceConnectionState", "(IILjava/lang/String;)I", (void \*)android\_media\_AudioSystem\_setDeviceConnectionState},

}

static jint

android\_media\_AudioSystem\_setDeviceConnectionState(JNIEnv \*env, jobject thiz, jint device, jint state, jstring device\_address)

{

int status = check\_AudioSystem\_Command(AudioSystem::setDeviceConnectionState(static\_cast <audio\_devices\_t>(device),

static\_cast <audio\_policy\_dev\_state\_t>(state),

c\_address));

}

AudioSystem.cpp (frameworks\av\media\libmedia) 34471 2016-04-01

status\_t AudioSystem::setDeviceConnectionState(audio\_devices\_t device,

audio\_policy\_dev\_state\_t state,

const char \*device\_address)

{

const sp<IAudioPolicyService>& aps = AudioSystem::get\_audio\_policy\_service();

return aps->setDeviceConnectionState(device, state, address);

}

IAudioPolicyService.cpp (frameworks\av\media\libmedia) 50190 2016-04-01

virtual status\_t setDeviceConnectionState(

audio\_devices\_t device,

audio\_policy\_dev\_state\_t state,

const char \*device\_address)

{

remote()->transact(SET\_DEVICE\_CONNECTION\_STATE, data, &reply);

}

AudioPolicyService.cpp (frameworks\av\services\audiopolicy) 43126 2016-04-01

status\_t AudioPolicyService::onTransact(

uint32\_t code, const Parcel& data, Parcel\* reply, uint32\_t flags)

{

return BnAudioPolicyService::onTransact(code, data, reply, flags);

}

AudioPolicyInterfaceImpl.cpp (frameworks\av\services\audiopolicy) 22203 2016-04-01

status\_t AudioPolicyService::setDeviceConnectionState(audio\_devices\_t device,

audio\_policy\_dev\_state\_t state,

const char \*device\_address)

{

return mAudioPolicyManager->setDeviceConnectionState(device,

state, device\_address);

}

AudioPolicyManager.h (frameworks\av\services\audiopolicy) 50621 2016-04-01

virtual status\_t setDeviceConnectionState(audio\_devices\_t device,

audio\_policy\_dev\_state\_t state,

const char \*device\_address);

AudioPolicyManager.cpp (frameworks\av\services\audiopolicy) 354604 2016-04-01

status\_t AudioPolicyManager::setDeviceConnectionState(audio\_devices\_t device,

audio\_policy\_dev\_state\_t state,

{

return setDeviceConnectionStateInt(device, state, device\_address);

}

status\_t AudioPolicyManager::setDeviceConnectionStateInt(audio\_devices\_t device,){

sp<DeviceDescriptor> devDesc = getDeviceDescriptor(device, device\_address);

for (size\_t i = 0; i < mOutputs.size(); i++) {

audio\_devices\_t newDevice = getNewOutputDevice(mOutputs.keyAt(i),

true /\*fromCache\*/);

}

}

AudioPolicyManager.cpp (frameworks\av\services\audiopolicy) 354604 2016-04-01

uint32\_t AudioPolicyManager::setOutputDevice(audio\_io\_handle\_t output,){

}

改通道函数：

createAudioPatch

releaseAudioPatch

AudioPolicyManager

checkOutputsForDevice

setOutputDevice

setDeviceConnectionStateInt

updateCallRouting

setPhoneState

setForceUse

startOutput

耳机事件：

AudioService.java (frameworks\base\media\java\android\media) 263236 2016-04-01

public void setWiredDeviceConnectionState(int device, int state, String name) {

queueMsgUnderWakeLock(mAudioHandler,

MSG\_SET\_WIRED\_DEVICE\_CONNECTION\_STATE,

device,

state,

name,

delay);

}

private void sendDeviceConnectionIntent(int device, int state, String name)

{

if (device == AudioSystem.DEVICE\_OUT\_WIRED\_HEADSET) {

connType = AudioRoutesInfo.MAIN\_HEADSET;

intent.setAction(Intent.ACTION\_HEADSET\_PLUG);

intent.putExtra("microphone", 1);

ActivityManagerNative.broadcastStickyIntent(intent, null, UserHandle.USER\_ALL);

}

private void onSetWiredDeviceConnectionState(int device, int state, String name)

{

sendDeviceConnectionIntent(device, state, name);

}

public void handleMessage(Message msg) {

case MSG\_SET\_WIRED\_DEVICE\_CONNECTION\_STATE:

onSetWiredDeviceConnectionState(msg.arg1, msg.arg2, (String)msg.obj);

mAudioEventWakeLock.release();

break;

}

摄像头死机：

[ 68.850397] V4L2 failed to set format! err -16

[ 68.855659] failed to request v4l2 video buffer!

[ 68.867055] CIF ISP2.0 cif\_isp20\_v4l2\_streamoff(574) ERR: failed with error -16

[ 70.219007] CaptureUnit[1742]: segfault at 0 ip 75eff4cd sp 71fff6f0 error 4

[ 70.226222] CIF ISP2.0 cif\_isp20\_check\_poll\_timeout\_wa WARN: poll timeout workaround

[ 70.226282] in camera.sofia3g.so[75ea6000+d6000]

[ 70.255368] CIF ISP2.0 cif\_isp20\_v4l2\_release WARN: streamoff failed

[ 70.261806] CIF ISP2.0 cif\_isp20\_release WARN: CIF SP in streaming state, should be stopped before release, trying to stop it

[ 70.273478] ===3692==========================

[ 70.278332] CIF ISP2.0 cif\_isp20\_v4l2\_release(1073) ERR: videobuf\_mmap\_free failed with error -16

[ 70.287445] CIF ISP2.0 cif\_isp20\_v4l2\_release(1098) ERR: failed with error -14

[ 70.595865] CIF ISP2.0 v4l2\_enum\_input(1311) ERR: index 2 out of bounds

[ 70.864144] XGOLD\_LPMP3 e0100000.lpmp3: lpmp3\_fs\_ioctl: enable lpmp3

[ 70.877351] XGOLD\_LPMP3 e0100000.lpmp3: lpmp3\_fs\_ioctl: wakelock, timeout = 5 sec

[ 70.880952] XGOLD\_LPMP3 e0100000.lpmp3: lpmp3\_fs\_ioctl: disable lpmp3

[ 70.892154] XGOLD\_LPMP3 e0100000.lpmp3: lpmp3\_fs\_ioctl: wakelock, timeout = 5 sec

[ 102.500599] rtc: xgold\_rtc\_alarm\_irq\_enable: alarm irq disable set

[ 114.160325] V4L2 failed to set format! err -16

I/KERNEL ( 63): [ 114.160325] V4L2 failed to set format! err -16

[ 114.165287] failed to request v4l2 video buffer!

[ 114.176001] CIF ISP2.0 cif\_isp20\_v4l2\_streamoff(574) ERR: failed with error -16

I/KERNEL ( 63): [ 114.165287] failed to request v4l2 video buffer!

I/KERNEL ( 63): [ 114.176001] CIF ISP2.0 cif\_isp20\_v4l2\_streamoff(574) ERR: failed with error -16

F/libc ( 174): Fatal signal 11 (SIGSEGV), code 1, fault addr 0x0 in tid 1741 (CaptureUnit)

I/DEBUG ( 172): \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*

I/DEBUG ( 172): Build fingerprint: 'Intel/Sofia3gr/Sf3gr\_mrd6\_p2:5.1.1/LMY47V/shizl06231638:eng/release-keys'

W/NativeCrashListener( 301): Couldn't find ProcessRecord for pid 174

I/DEBUG ( 172): Revision: '0'

E/DEBUG ( 172): AM write failure (32 / Broken pipe)

I/DEBUG ( 172): ABI: 'x86'

I/DEBUG ( 172): pid: 174, tid: 1741, name: CaptureUnit >>> /system/bin/mediaserver <<<

I/DEBUG ( 172): signal 11 (SIGSEGV), code 1 (SEGV\_MAPERR), fault addr 0x0

I/DEBUG ( 172): eax 00000000 ebx 75f81300 ecx 000000b4 edx 721ffb7b

I/DEBUG ( 172): esi 00000000 edi 721ff8ac

I/DEBUG ( 172): xcs 00000073 xds 0000007b xes 0000007b xfs 00000000 xss 0000007b

I/DEBUG ( 172): eip 75eff4cd ebp 721ffb98 esp 721ff6f0 flags 00010246

I/DEBUG ( 172):

I/DEBUG ( 172): backtrace:

I/DEBUG ( 172): #00 pc 000594cd /system/lib/hw/camera.sofia3g.so (android::camera2::CIFCaptureUnit::handleOverlayPoll()+189)

I/DEBUG ( 172): #01 pc 0005f037 /system/lib/hw/camera.sofia3g.so (android::camera2::CIFCaptureUnit::handleMessageNotify(android::camera2::CIFCaptureUnit::Message&)+423)

I/DEBUG ( 172): #02 pc 0005f382 /system/lib/hw/camera.sofia3g.so (android::camera2::CIFCaptureUnit::messageThreadLoop()+514)

I/DEBUG ( 172): #03 pc 00040a54 /system/lib/hw/camera.sofia3g.so (android::camera2::MessageThread::threadLoop()+20)

I/DEBUG ( 172): #04 pc 00017048 /system/lib/libutils.so (android::Thread::\_threadLoop(void\*)+392)

I/DEBUG ( 172): #05 pc 00016730 /system/lib/libutils.so (thread\_data\_t::trampoline(thread\_data\_t const\*)+128)

I/DEBUG ( 172): #06 pc 000303db /system/lib/libc.so (\_\_pthread\_start(void\*)+59)

I/DEBUG ( 172): #07 pc 0002b56b /system/lib/libc.so (\_\_start\_thread+27)

I/DEBUG ( 172): #08 pc 00013366 /system/lib/libc.so (\_\_bionic\_clone+70)

I/DEBUG ( 172):

[ 115.289008] CaptureUnit[1741]: segfault at 0 ip 75eff4cd sp 721ff6f0 error 4I/DEBUG ( 172): Tombstone written to: /data/tombstones/tombstone\_03

I/BootReceiver( 301): Copying /data/tombstones/tombstone\_03 to DropBox (SYSTEM\_TOMBSTONE) in camera.sofia3g.so[75ea6000+d6000]

I/KERNEL ( 174): [ 115.289008] CaptureUnit[1741]: segfault at 0 ip 75eff4cd sp 721ff6f0 error 4

I/KERNEL ( 174): in camera.sofia3g.so[75ea6000+d6000]

I/KERNEL ( 174):

[ 115.324891] CIF ISP2.0 cif\_isp20\_v4l2\_release WARN: streamoff failed

W/AudioSystem( 301): AudioFlinger server died![ 115.336823] CIF ISP2.0 cif\_isp20\_release WARN: CIF SP in streaming state, should be stopped before release, trying to stop it

[ 115.352088] ===3692==========================

[ 115.356826] CIF ISP2.0 cif\_isp20\_v4l2\_release(1073) ERR: videobuf\_mmap\_free failed with error -16

W/IMediaDeathNotifier( 301): media server died

W/AudioSystem( 301): AudioPolicyService server died!

W/SoundTrigger( 301): Sound trigger service died!

I/ServiceManager( 156): service 'media.audio\_flinger' died

I/ServiceManager( 156): service 'media.camera' died

I/ServiceManager( 156): service 'media.player' died

I/ServiceManager( 156): service 'media.audio\_policy' died

I/ServiceManager( 156): service 'media.sound\_trigger\_hw' died

W/CameraBase( 1691): Camera service died!

W/IMediaDeathNotifier( 1691): media server died

W/IMediaDeathNotifier( 635): media server died

W/AudioSystem( 635): AudioPolicyService server died!

W/AudioSystem( 1467): AudioFlinger server died![ 115.367855] CIF ISP2.0 cif\_isp20\_v4l2\_release(1098) ERR: failed with error -14

I/KERNEL ( 174): [ 115.324891] CIF ISP2.0 cif\_isp20\_v4l2\_release WARN: streamoff failed

I/KERNEL ( 174): [ 115.336823] CIF ISP2.0 cif\_isp20\_release WARN: CIF SP in streaming state, should be stopped before release, trying to stop it

I/KERNEL ( 174): [ 115.352088] ===3692==========================

E/AudioService( 301): Media server died.

I/KERNEL ( 174): [ 115.356826] CIF ISP2.0 cif\_isp20\_v4l2\_release(1073) ERR: videobuf\_mmap\_free failed with error -16

I/KERNEL ( 174): [ 115.367855] CIF ISP2.0 cif\_isp20\_v4l2\_release(1098) ERR: failed with error -14

I/mediaserver( 1771): ServiceManager: 0x761336a0

I/AudioFlinger( 1771): Using default 3000 mSec as standby time.

I/CameraService( 1771): CameraService started (pid=1771)

D/PlatformData( 1771): Camera HAL static init

E/ ( 1771): @getDevName, wront device offset name[ 115.672657] CIF ISP2.0 v4l2\_enum\_input(1311) ERR: index 2 out of bounds

I/KERNEL ( 1771): [ 115.672657] CIF ISP2.0 v4l2\_enum\_input(1311) ERR: index 2 out of bounds

E/Camera\_V4L2VideoNode( 1771): VIDIOC\_ENUMINPUT failed returned: -1 (Invalid argument)

D/PlatformData( 1771): set property media.settings.xml: /etc/media\_profiles\_tw9985.xml|/etc/media\_profiles\_tw9992.xml

D/CIFConfParser( 1771): addCamera: for camera 0

D/CIFConfParser( 1771): addCamera: for camera 1

E/Camera\_PSLConfParser( 1771): getPixelFormatAsValue, Unknown Pixel Format (V4L2\_PIX\_FMT\_UYVY)

E/Camera\_PSLConfParser( 1771): getPixelFormatAsValue, Unknown Pixel Format (V4L2\_PIX\_FMT\_UYVY)

E/Camera\_PSLConfParser( 1771): getPixelFormatAsValue, Unknown Pixel Format (V4L2\_PIX\_FMT\_UYVY)

E/Camera\_PSLConfParser( 1771): getPixelFormatAsValue, Unknown Pixel Format (V4L2\_PIX\_FMT\_UYVY)

E/Camera\_PSLConfParser( 1771): getPixelFormatAsValue, Unknown Pixel Format (V4L2\_PIX\_FMT\_UYVY)

E/Camera\_PSLConfParser( 1771): getPixelFormatAsValue, Unknown Pixel Format (V4L2\_PIX\_FMT\_UYVY)

E/Camera\_Conf( 1771): Error Initializing CMC

E/Camera\_Conf( 1771): Error Initializing CMC

V2g\_bridge.c (battle\_camera\xgold) 26369 2016-06-16

static int v2g\_v4l2\_init(v2g\_context \*ctx)

{

err = v2g\_v4l2\_ioctl(&ctx->v4l2\_file, VIDIOC\_S\_FMT,(unsigned long)&fmt);

if (err) {

pr\_err("V4L2 failed to set format! err %d\n", err);

return err;

}

}

static int v2g\_start\_streaming(v2g\_context \*ctx)

{

err = v2g\_v4l2\_ioctl(v4l2\_file, VIDIOC\_REQBUFS, (unsigned long)&rqbufs);

if (err) {

pr\_err("failed to request v4l2 video buffer!\n");

goto clean\_up;

}

}

Cif\_isp20\_v4l2.c (drivers\media\platform\xgold) 54888 2016-04-01

static int cif\_isp20\_v4l2\_release(struct file \*file)

{

err = videobuf\_mmap\_free(queue);

if (IS\_ERR\_VALUE(err)) {

cif\_isp20\_pltfrm\_pr\_err(dev->dev,

"videobuf\_mmap\_free failed with error %d\n", err);

ret = -EFAULT;

}

}

Sofia3gr的boot通行：usb方式

Usb\_drv\_api.c (drivers\comm\hwdep\usb\src\imc) 17991 2016-04-01

S32 usb\_hw\_setup(comm\_t \*\*fd, T\_COMM\_CONFIG \*config, comm\_int\_cb\_t cb)

{

p\_dev->read = usb\_drv\_read;

p\_dev->write = usb\_drv\_write;

p\_dev->ioctrl = usb\_drv\_ioctrl;

}

static S32 usb\_drv\_read(U32 \*fd, U8 \*buf, U32 size\_to\_read, U32 \*size\_read, comm\_int\_rw\_cb\_t cb, void \*arg)

{

return (usb\_drv\_data\_transfer(p\_ep, (U8 \*)buf, size\_to\_read, size\_read, 0));

}

static S32 usb\_drv\_data\_transfer(T\_USB\_DRV\_EP \*p\_ep, U8 \*buf, U32 size\_to\_transfer, U32 \*size\_transferred, U32 timeout\_in\_msec)

{

status = usb\_drv\_bam\_ep\_data\_transfer(p\_ep->hd, (U8 \*)buf, size\_to\_transfer, usb\_drv\_data\_transfer\_cb, (void \*)p\_ep, timeout\_in\_msec);

}

S32 usb\_drv\_bam\_ep\_data\_transfer(U32 hd, U8 \* buf, U32 size, T\_BC\_USB\_DRV\_BAM\_DATA\_TRANSFER\_CB cb, void \*arg, U32 timeout\_in\_ms)

{

p\_ep->transfer.v0 = p\_usb\_drv\_br->fx.read.v0;

p\_ep->transfer.v1 = p\_usb\_drv\_br->fx.read.v1;

(void)p\_ep->transfer.v1((T\_BC\_USB\_DRV\_BR\_API\_INTERFACE)((T\_BC\_USB\_DRV\_BAM\_DEV \*)p\_ep->dev)->id, buf, &p\_ep->t\_size, p\_ep->cb\_br);

p\_ep->transfer.v2 = p\_usb\_drv\_br->fx.read.v2;

(void)p\_ep->transfer.v2((T\_BC\_USB\_DRV\_BR\_API\_INTERFACE)((T\_BC\_USB\_DRV\_BAM\_DEV \*)p\_ep->dev)->id, p\_ep->descriptor.bEndpAddress, buf, &p\_ep->t\_size, usb\_drv\_bam\_data\_transfer\_cb, (void \*)p\_ep, timeout\_in\_ms);

}

Usb\_drv\_br\_api\_cfg.c (drivers\comm\hwdep\usb\src\imc\sf\_3g\_soc) 8633 2016-04-01

T\_BC\_USB\_DRV\_BR \*Usb\_drv\_br\_fn\_init(BOOL local, U32 local\_addr)

{

usb\_drv\_br.fx.read.v0 = ( T\_BC\_USB\_DRV\_BR\_API\_DATA\_TRANSFER )(mem\_start\_addr + PR\_BR\_USB\_API\_READ\_FN\_OFFSET\_ADDR);

}

串口通信：

Usif\_drv.c (drivers\comm\hwdep\usif\src) 19659 2016-04-01

S32 usif\_drv\_setup(comm\_t \*\*fd, T\_COMM\_CONFIG \*config, comm\_int\_cb\_t cb)

{

usif\_dev.read = usif\_drv\_read;

usif\_dev.write = usif\_drv\_write;

usif\_dev.ioctrl = usif\_drv\_ioctrl;

usif\_drv.dev.event\_handler = usif\_drv\_event\_handler;

}

设置波特率：

static S32 usif\_drv\_baudrate\_set(U32 baud);

static void usif\_drv\_isr\_read(void)

{

usif\_hal\_enable\_rx(&usif\_drv.dev, 0);

usif\_drv.rx.cb(usif\_drv.rx.arg, \*usif\_drv.rx.size\_read);

}

Usif\_drv.c (drivers\comm\hwdep\usif\src) 19659 2016-04-01

static S32 usif\_drv\_read(U32 \*fd, U8HP buf, U32 size\_to\_read, U32 \*size\_read, comm\_int\_rw\_cb\_t cb, void \*arg)

{

usif\_hal\_enable\_rx(&usif\_drv.dev, 1);

usif\_hal\_req\_read(&usif\_drv.dev, (U32\*)buf, size\_to\_read);

}

Usif\_hal.c (drivers\comm\hwdep\usif\src) 47509 2016-04-01

usif\_hal\_result\_t usif\_hal\_req\_read(usif\_hal\_device\_t\* p\_device, U32 \*buf, U32 max\_size)

{

setUsif\_USIF\_FIFO\_CTRL\_rx\_start(p\_device->p\_reg, 1);

}

高通启动代码分析：

void kmain(void)

{

thread\_init\_early();//

}

LK部分串口通信函数：

Uart.c (bootable\bootloader\lk\platform\msm\_shared) 6273 2016-06-29

发送函数：

static int \_uart\_putc(int port, char c)

{

if (!uart\_ready)

return -1;

while (!(urd(UART\_SR) & UART\_SR\_TX\_READY)) ;

uwr(c, UART\_TF);

return 0;

}

int uart\_putc(int port, char c)

{

if (c == '\n') {

\_uart\_putc(0, '\r');

}

\_uart\_putc(0, c);

}

接受：

int uart\_getc(int port, bool wait)

{

if (!uart\_ready)

return -1;

while (!(urd(UART\_SR) & UART\_SR\_RX\_READY))

if (!wait)

return -1;

return urd(UART\_RF);

}

全志平台的fastboot：实现：

Usb\_fastboot.c (usb\_sunxi) 34441 2016-07-01

sunxi\_usb\_module\_init(SUNXI\_USB\_DEVICE\_FASTBOOT,

sunxi\_fastboot\_state\_loop,

static int sunxi\_fastboot\_state\_loop(void \*buffer)

{

tick\_printf("fastboot command = %s\n", sunxi\_ubuf->rx\_req\_buffer);

\_\_try\_to\_download((char \*)(sunxi\_ubuf->rx\_req\_buffer + 9), response);//下载

\_\_get\_var((char \*)(sunxi\_ubuf->rx\_req\_buffer + 7));//获取变量

}

static void \_\_get\_var(char \*ver\_name)

{

char response[68];

memset(response, 0, 68);

strcpy(response,"OKAY");

strcpy(response + 4, SUNXI\_FASTBOOT\_DEVICE\_PRODUCT);

sprintf(response + 4, "0x%08x", SUNXI\_USB\_FASTBOOT\_BUFFER\_MAX);

\_\_sunxi\_fastboot\_send\_status(response, strlen(response));//返回

}

高通lk分析：

Aboot.c (bootable\bootloader\lk\app\aboot) 78336 2016-06-29

void aboot\_init(const struct app\_descriptor \*app)

{

target\_display\_init(device.display\_panel);

}

Target\_display.c (bootable\bootloader\lk\target\msm8909) 9570 2016-06-29

void target\_display\_init(const char \*panel\_name)

{

target\_force\_cont\_splash\_disable(false);

ret = gcdb\_display\_init(panel\_name, MDP\_REV\_305, MIPI\_FB\_ADDR);

target\_force\_cont\_splash\_disable(true);

msm\_display\_off();

}

Gcdb\_display.c (bootable\bootloader\lk\dev\gcdb\display) 11771 2016-06-29

int gcdb\_display\_init(const char \*panel\_name, uint32\_t rev, void \*base)

{

panel.power\_func = mdss\_dsi\_panel\_power;

ret = msm\_display\_init(&panel);

}

Display.c (bootable\bootloader\lk\platform\msm\_shared) 8937 2016-06-29

int msm\_display\_init(struct msm\_fb\_panel\_data \*pdata)

{

if (pdata->power\_func)

ret = pdata->power\_func(1, &(panel->panel\_info));

ret = msm\_display\_on();

ret = pdata->post\_power\_func(1);

ret = pdata->bl\_func(1);

}

Display.c (bootable\bootloader\lk\platform\msm\_shared) 8937 2016-06-29

int msm\_display\_on()

{

case MIPI\_VIDEO\_PANEL:

ret = mdp\_dsi\_video\_on(pinfo);

ret = mdss\_dsi\_post\_on(panel);

ret = mipi\_dsi\_on();

if (pinfo->on)

ret = pinfo->on();

}

开机涉及pmic：

Gcdb\_display.c (bootable\bootloader\lk\dev\gcdb\display) 11771 2016-06-29

int gcdb\_display\_init(const char \*panel\_name, uint32\_t rev, void \*base)

{

panel.bl\_func = mdss\_dsi\_bl\_enable;

}

Gcdb\_display.c (bootable\bootloader\lk\dev\gcdb\display) 11771 2016-06-29

static int mdss\_dsi\_bl\_enable(uint8\_t enable)

{

ret = panel\_backlight\_ctrl(enable);

}

static uint32\_t panel\_backlight\_ctrl(uint8\_t enable)

{

return target\_backlight\_ctrl(panelstruct.backlightinfo, enable);

}

Target\_display.c (bootable\bootloader\lk\target\msm8909) 9570 2016-06-29

int target\_backlight\_ctrl(struct backlight \*bl, uint8\_t enable)

{

pm\_pwm\_enable(true);

pm8x41\_config\_output\_mpp(&mpp);

pm8x41\_enable\_mpp(&mpp, MPP\_ENABLE);

}

SBL1代码分析：

#define HWIO\_PMIC\_ARBq\_CHNLn\_WDATA0\_ADDR(q,n) (PMIC\_ARB\_CORE\_REGISTERS\_REG\_BASE + 0x00000010 + 0x1000 \* (q) + 0x8000 \* (n))

#define PMIC\_ARB\_CORE\_REGISTERS\_REG\_BASE (PMIC\_ARB\_BASE + 0x00400000)

#define PMIC\_ARB\_BASE pmicArbHwioBase

#define **SPMI\_BARE\_PMIC\_ARB\_ADDRESS** 0x02000000

0x02000000 +

0x00400000　＋

0x00000010　＋

0x1000 \* (q)　＋

0x8000 \* (n)

ＬＫ对比：

Ｖ１

#define PMIC\_ARB\_CHNLn\_CONFIG(x) (SPMI\_BASE + 0xF804 + (x) \* 0x80)

#define SPMI\_BASE  0x02000000

0x02000000　＋

　　　0xF804　＋

(x) \* 0x80

Ｖ２：

#define PMIC\_ARB\_CHNLn\_CONFIG(x) (PMIC\_ARB\_CORE\_REG\_BASE + 0x00000004 + (x) \* 0x8000)

#define PMIC\_ARB\_CORE\_REG\_BASE (SPMI\_BASE + 0x00400000)

#define SPMI\_BASE 0x02000000

0x02000000

0x00400000

0x00000004

(x) \* 0x8000

SLB1的pmic执行命令：

uint32 boot\_usb\_al\_bulk\_receive(byte \*rx\_buf,

uint32 len,

sahara\_rx\_cb\_type rx\_cb,

uint32 \* err\_code)

{

SBL1的启动流程：

Sbl1.s (e:\8909源码\qmss8909\boot\_images\core\boot\secboot3\hw\msm8909\sbl1)

sbl1\_entry

ldr r5, =sbl1\_main\_ctl

Sbl1\_mc.c (e:\8909源码\qmss8909\boot\_images\core\boot\secboot3\hw\msm8952\sbl1) 49056 2016-07-21

void sbl1\_main\_ctl(boot\_pbl\_shared\_data\_type \*pbl\_shared)

{

boot\_config\_process\_bl(&bl\_shared\_data,

SBL1\_IMG,

sbl1\_config\_table);

}

boot\_configuration\_table\_entry sbl1\_config\_table[] =

{

/\* SBL1 -> QSEE \*/

{

load\_qsee\_pre\_procs, /\* pre\_procs \*/

load\_qsee\_post\_procs, /\* post\_procs \*/

}

/\* SBL1 -> APPSBL \*/

{

rpm\_load\_cancel, /\* load\_cancel \*/

rpm\_partition\_name, /\* target\_img\_partition\_id \*/

}

/\* SBL1 -> APPSBL \*/

{

qsee\_jump\_func, /\* jump\_func \*/

}

}

Sbl1的cdt

Boot\_config\_data.c (e:\8909源码\qmss8909\boot\_images\core\boot\secboot3\src) 9193 2016-07-21

//读取cdt

uint8\* boot\_get\_config\_data\_block(const uint8 \*raw\_cdt, uint32 index, uint32 \*length)

{

}

boot\_boolean boot\_copy\_config\_data\_block

(

uint8 \*dst\_cdt,

uint32 cdb\_index,

const uint8 \*src\_cdb\_data,

uint32 src\_cdb\_size

)

{

dst\_cdb\_data = boot\_get\_config\_data\_block(dst\_cdt, cdb\_index, &dst\_cdb\_size);

}

Tcfg\_daatanaza.py (e:\8909源码\qmss8909\boot\_images\build\ms) 2786 2016-07-21

env.AddUsesFlags('USES\_CDT\_ON\_EEPROM', from\_builds\_file = True)

从e2prom读取cdt 如果为false则从emmc读取cdt

Boot\_config\_eeprom.c (e:\8909源码\qmss8909\boot\_images\core\boot\secboot3\src) 19951 2016-07-21

Read the cdt from EEPROM and update the default config\_data\_table array//从EEPROM中更新CDT数组。

void boot\_update\_config\_data\_table(struct cdt\_info \*cdt\_info\_ptr)

{

//1.检查EEPROM是否可读

//2.检查magic number ，确保ID正确

//3.把Platform ID更新到默认的CDT表

/\* Retrieve the platform ID data from the buffer\*/

platform\_id\_cdb = boot\_get\_config\_data\_block(page\_buf, CONFIG\_DATA\_BLOCK\_INDEX\_V1\_PLATFORM\_ID,

&platform\_id\_cdb\_size);

/\* Copy the platform ID data into default CDT\*/

if(TRUE == boot\_copy\_config\_data\_block(cdt\_info\_ptr->cdt\_ptr,

CONFIG\_DATA\_BLOCK\_INDEX\_V1\_PLATFORM\_ID,

platform\_id\_cdb,

platform\_id\_cdb\_size))

}

高通msm8909 sensor hal层分析：

Sensors.cpp (hardware\qcom\sensors) 9403 2016-06-29

struct sensors\_module\_t HAL\_MODULE\_INFO\_SYM = {

common: {

tag: HARDWARE\_MODULE\_TAG,

version\_major: 1,

version\_minor: 0,

id: SENSORS\_HARDWARE\_MODULE\_ID,

name: "Quic Sensor module",

author: "Quic",

methods: &sensors\_module\_methods,

dso: NULL,

reserved: {0},

},

get\_sensors\_list: sensors\_\_get\_sensors\_list,

};

static struct hw\_module\_methods\_t sensors\_module\_methods = {

open: open\_sensors

};

static int open\_sensors(const struct hw\_module\_t\* module, const char\*,

struct hw\_device\_t\*\* device)

{

int status = -EINVAL;

sensors\_poll\_context\_t \*dev = new sensors\_poll\_context\_t();

NativeSensorManager& sm(NativeSensorManager::getInstance());

memset(&dev->device, 0, sizeof(sensors\_poll\_device\_1\_ext\_t));

dev->device.common.version = SENSORS\_DEVICE\_API\_VERSION\_0\_1;

dev->device.common.module = const\_cast<hw\_module\_t\*>(module);

dev->device.common.close = poll\_\_close;

dev->device.activate = poll\_\_activate;

dev->device.setDelay = poll\_\_setDelay;

dev->device.poll = poll\_\_poll;

dev->device.calibrate = poll\_calibrate;

dev->device.batch = poll\_\_batch;

dev->device.flush = poll\_\_flush;

\*device = &dev->device.common;

status = 0;

}

NativeSensorManager.h (hardware\qcom\sensors) 5846 2016-06-29

class NativeSensorManager : public Singleton<NativeSensorManager> {

}

static int poll\_\_poll(struct sensors\_poll\_device\_t \*dev,

sensors\_event\_t\* data, int count) {

sensors\_poll\_context\_t \*ctx = (sensors\_poll\_context\_t \*)dev;

return ctx->pollEvents(data, count);

}

int sensors\_poll\_context\_t::pollEvents(sensors\_event\_t\* data, int count)

{

NativeSensorManager& sm(NativeSensorManager::getInstance());

int nb = sm.readEvents(slist[i].handle, data, count);

}

Sensors.cpp (hardware\qcom\sensors) 9403 2016-06-29

struct sensors\_poll\_context\_t {

struct sensors\_poll\_device\_1\_ext\_t device;// must be first

sensors\_poll\_context\_t();

~sensors\_poll\_context\_t();

int activate(int handle, int enabled);

int setDelay(int handle, int64\_t ns);

int pollEvents(sensors\_event\_t\* data, int count);

int calibrate(int handle, cal\_cmd\_t \*para);

}

打开设备流程分析：

Sensors.cpp (hardware\qcom\sensors) 9403 2016-06-29

static int open\_sensors(const struct hw\_module\_t\* module, const char\*,

struct hw\_device\_t\*\* device)

{

//初始化

sensors\_poll\_context\_t \*dev = new sensors\_poll\_context\_t();

}

Sensors.cpp (hardware\qcom\sensors) 9403 2016-06-29

sensors\_poll\_context\_t::sensors\_poll\_context\_t()

{

NativeSensorManager& sm(NativeSensorManager::getInstance());

//获取sensor列表

number = sm.getSensorList(&slist);

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

context = sm.getInfoByHandle(slist[i].handle);

mPollFds[i].fd = (context == NULL) ? -1 : context->data\_fd;

mPollFds[i].events = POLLIN;

mPollFds[i].revents = 0;

}

int wakeFds[2];

//开辟管道

int result = pipe(wakeFds);

//设置管道

fcntl(wakeFds[0], F\_SETFL, O\_NONBLOCK);

fcntl(wakeFds[1], F\_SETFL, O\_NONBLOCK);

mWritePipeFd = wakeFds[1];

mPollFds[number].fd = wakeFds[0];

mPollFds[number].events = POLLIN;

mPollFds[number].revents = 0;

}

NativeSensorManager.cpp (hardware\qcom\sensors) 33544 2016-06-29

int NativeSensorManager::getSensorList(const sensor\_t \*\*list) {

\*list = mSensorCount ? sensor\_list:NULL;

return mSensorCount;

}

NativeSensorManager::getInstance()

NativeSensorManager.cpp (hardware\qcom\sensors) 33544 2016-06-29

NativeSensorManager::NativeSensorManager():

mSensorCount(0), mScanned(false), mEventCount(0), type\_map(NULL), handle\_map(NULL), fd\_map(NULL)

{

type\_map.setCapacity(MAX\_SENSORS);

handle\_map.setCapacity(MAX\_SENSORS);

fd\_map.setCapacity(MAX\_SENSORS);

getDataInfo()；

}

:#define SYSFS\_CLASS "/sys/class/sensors/"

int NativeSensorManager::getDataInfo() {

struct sensor\_t sensor\_mag;

struct sensor\_t sensor\_acc;

struct sensor\_t sensor\_light;

struct sensor\_t sensor\_proximity;

struct sensor\_t sensor\_gyro;

//获取sensor个数

mSensorCount = getSensorListInner();

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

/\* hardware sensor depend on itself \*/

list\_add\_tail(&list->dep\_list, &item->list);

list->data\_fd = open(list->data\_path, O\_RDONLY | O\_CLOEXEC | O\_NONBLOCK);

fd\_map.add(list->data\_fd, list);

type\_map.add(list->sensor->type, list);

handle\_map.add(list->sensor->handle, list);

switch (list->sensor->type) {

case SENSOR\_TYPE\_PROXIMITY:

has\_proximity = 1;

list->sensor->flags |= SENSOR\_FLAG\_ON\_CHANGE\_MODE;

list->driver = new ProximitySensor(list);

sensor\_proximity = \*(list->sensor);

break;

case SENSOR\_TYPE\_LIGHT:

list->sensor->flags |= SENSOR\_FLAG\_ON\_CHANGE\_MODE;

list->driver = new LightSensor(list);

sensor\_light = \*(list->sensor);

break;

case SENSOR\_TYPE\_GYROSCOPE:

has\_gyro = 1;

list->driver = new GyroSensor(list);

sensor\_gyro = \*(list->sensor);

break;

}

//读xml文件

initCalibrate(list);

}

}

获取sensor个数分析代码：

NativeSensorManager.cpp (hardware\qcom\sensors) 33544 2016-06-29

int NativeSensorManager::getSensorListInner()

{

char devname[PATH\_MAX];

#define SYSFS\_CLASS "/sys/class/sensors/"

const char \*dirname = SYSFS\_CLASS;

//打开目录

dir = opendir(dirname);

strlcpy(devname, dirname, PATH\_MAX);

//devname = /sys/class/sensors/

filename = devname + strlen(devname);

while ((de = readdir(dir))) {

list = &context[number];

strlcpy(filename, de->d\_name, PATH\_MAX - strlen(SYSFS\_CLASS));

nodename = filename + strlen(de->d\_name);

\*nodename++ = '/';

//nodename = /sys/class/sensors/gsenxx/

for (i = 0; i < ARRAY\_SIZE(node\_map); i++) {

//nodename = /sys/class/sensors/gsenxx/xxx

strlcpy(nodename, node\_map[i].node, PATH\_MAX - strlen(SYSFS\_CLASS) - strlen(de->d\_name));

//判断节点的正确性

err = getNode((char\*)(list->sensor), devname, &node\_map[i]);

}

}

list->sensor->handle = SENSORS\_HANDLE(number);

strlcpy(nodename, "", SYSFS\_MAXLEN);

//nodename = “”;

strlcpy(list->enable\_path, devname, PATH\_MAX);

strlcpy(nodename, "device", SYSFS\_MAXLEN);

//nodename = /sys/class/sensors/device

if (getEventPath(devname, list->data\_path) == -ENODEV) {

getEventPathOld(list, list->data\_path);

}

}

#define EVENT\_PATH "/dev/input/"

int NativeSensorManager::getEventPath(const char \*sysfs\_path, char \*event\_path)

{

//sysfs\_path = /sys/class/sensors/device/->/sys/event0/

dir = opendir(sysfs\_path);

//symlink = /sys/event0/

len = readlink(sysfs\_path, symlink, PATH\_MAX);

// needle = /event0

needle = strrchr(symlink, '/');

if (strncmp(needle + 1, "input", strlen("input")) != 0)

//needle = /event0/input

//event\_path = /dev/input/

strlcpy(event\_path, EVENT\_PATH, PATH\_MAX);

while ((de = readdir(dir))) {

if (strncmp(de->d\_name, "event", strlen("event")) == 0) {

strlcat(event\_path, de->d\_name, sizeof(de->d\_name));

break;

}

}

}

initCalibrate(list);

NativeSensorManager.cpp (hardware\qcom\sensors) 33544 2016-06-29

int NativeSensorManager::initCalibrate(const SensorContext \*list)

{

sensors\_XML& sensor\_XML(sensors\_XML :: getInstance());

err = sensor\_XML.read\_sensors\_params(list->sensor, &cal\_result, CAL\_STATIC);

//list->driver = new GyroSensor(list);

err = list->driver->initCalibrate(list->sensor->handle, &cal\_result);

}

sensors\_XML.cpp (hardware\qcom\sensors) 12541 2016-06-29

int sensors\_XML :: write\_sensors\_params(struct sensor\_t \*sensor, struct cal\_result\_t \*cal\_result, int state)

{

}

Xml文件读取的参数：

"name"

"sensor"

"static"

"dynamic"

"offset\_x"

"offset\_y"

"offset\_z"

"threshold\_h"

"threshold\_l"

"bias"

Vold流程分析：

int main() {

vm = VolumeManager::Instance();

nm = NetlinkManager::Instance())

cl = new CommandListener();

vm->setBroadcaster((SocketListener \*) cl);

nm->setBroadcaster((SocketListener \*) cl);

process\_config(vm);

}

static int process\_config(VolumeManager \*vm)

{

dv = new DirectVolume(vm, &(fstab->recs[i]), flags);

vm->addVolume(dv);

}

Volume.cpp 19816 2016-07-12

int Volume::mountVol() {

//得到设备节点,如:/dev/block/vold/8:1

snprintf(nodepath,

sizeof(nodepath), "/dev/block/vold/%d:%d",

new\_major, new\_minor);

//创建设备节点

createDeviceNode(nodepath, new\_major, new\_minor)

//更新设备信息

updateDeviceInfo(nodepath, new\_major, new\_minor);

//由于更新，重新获取设备节点

n = getDeviceNodes((dev\_t \*) &deviceNodes, 4);

//设置状态

setState(Volume::State\_Checking);

//挂载

Fat::doMount(devicePath, getMountpoint(), false, false, false,

AID\_MEDIA\_RW, AID\_MEDIA\_RW, 0007, true)

}

### NetlinkManager模块的分析

NetlinkManager.cpp 2673 2016-07-12

在Vold代码中，使用NM模块的流程是：

·  调用Instance创建一个NM对象。

·  调用setBroadcaster设置CL对象。

·  调用start启动NM。

创建nm

NetlinkManager \*NetlinkManager::Instance() {

if (!sInstance)

sInstance = new NetlinkManager();//采用单列模式

return sInstance;

}

NM模块将使用Netlink和Kernel进行IPC通信

int NetlinkManager::start() {

struct sockaddr\_nl nladdr;

nladdr.nl\_family = AF\_NETLINK;

nladdr.nl\_pid = getpid();//设置自己的进程id

nladdr.nl\_groups = 0xffffffff;

mSock = socket(PF\_NETLINK,

SOCK\_DGRAM,NETLINK\_KOBJECT\_UEVENT)) //NETLINK\_KOBJECT\_UEVENT表示要接受内核的uevent事件

//设置SOcket接受缓冲区的大小

setsockopt(mSock, SOL\_SOCKET, SO\_RCVBUFFORCE, &sz, sizeof(sz)) < 0

setsockopt(mSock, SOL\_SOCKET, SO\_PASSCRED, &on, sizeof(on)) < 0

bind(mSock, (struct sockaddr \*) &nladdr, sizeof(nladdr)) < 0

mHandler = new NetlinkHandler(mSock);

mHandler->start()

}

#### NetlinkHandler的分析

NetlinkHandler.cpp 1440 2016-07-12

class NetlinkHandler: public NetlinkListener {

}

int NetlinkHandler::start() {

return this->startListener();

}

**[NetlinkListener](http://androidxref.com/5.1.1_r6/s?refs=NetlinkListener&project=system)**::**[NetlinkListener](http://androidxref.com/5.1.1_r6/s?refs=NetlinkListener&project=system)**(**int** **[socket](http://androidxref.com/5.1.1_r6/s?refs=socket&project=system)**) : [SocketListener](http://androidxref.com/5.1.1_r6/s?defs=SocketListener&project=system)([socket](http://androidxref.com/5.1.1_r6/s?defs=socket&project=system), **false**) {

[36](http://androidxref.com/5.1.1_r6/xref/system/core/libsysutils/src/NetlinkListener.cpp" \l "36)[mFormat](http://androidxref.com/5.1.1_r6/s?defs=mFormat&project=system) = [NETLINK\_FORMAT\_ASCII](http://androidxref.com/5.1.1_r6/s?defs=NETLINK_FORMAT_ASCII&project=system);

[37](http://androidxref.com/5.1.1_r6/xref/system/core/libsysutils/src/NetlinkListener.cpp" \l "37)}

SocketListener.cpp (e:\8909源码\8909\_android\core\libsysutils\src) 9698 2016-07-12

void SocketListener::init(const char \*socketName, int socketFd, bool listen, bool useCmdNum) {

mListen = listen;

mSocketName = socketName;

mSock = socketFd;

mUseCmdNum = useCmdNum;

pthread\_mutex\_init(&mClientsLock, NULL);//多线程

typedef android::List<SocketClient \*>SocketClientCollection

其中，SocketClient代表和Socket服务端通信的客户端。

mClients = new SocketClientCollection();

}

int SocketListener::startListener(int backlog) {

mSock = android\_get\_control\_socket(mSocketName))；

mListen && listen(mSock, backlog) ；

listen(mSock, backlog) ；

mClients->push\_back(new SocketClient(mSock, false, mUseCmdNum));

pipe(mCtrlPipe)；//开辟管道

pthread\_create(&mThread, NULL, SocketListener::threadStart, this);

}

SocketListener.cpp (e:\8909源码\8909\_android\core\libsysutils\src) 9698 2016-07-12

void \*SocketListener::threadStart(void \*obj) {

SocketListener \*me = reinterpret\_cast<SocketListener \*>(obj);

me->runListener();

pthread\_exit(NULL);

return NULL;

}

void SocketListener::runListener() {

while(1) {

FD\_ZERO(&read\_fds);

if (mListen) {

max = mSock;

FD\_SET(mSock, &read\_fds);

}

FD\_SET(mCtrlPipe[0], &read\_fds);

pthread\_mutex\_lock(&mClientsLock);

select(max + 1, &read\_fds, NULL, NULL, NULL)

FD\_ISSET(mCtrlPipe[0], &read\_fds)；

if (mListen && FD\_ISSET(mSock, &read\_fds)) {

c = accept(mSock, &addr, &alen);

}

onDataAvailable(c)

}

}

NetlinkListener.cpp (e:\8909源码\8909\_android\core\libsysutils\src) 2208 2016-07-12

bool NetlinkListener::onDataAvailable(SocketClient \*cli)

{

//调用recv接受数据

uevent\_kernel\_multicast\_uid\_recv(

socket, mBuffer, sizeof(mBuffer), &uid)

NetlinkEvent \*evt = new NetlinkEvent();//

evt->decode(mBuffer, count, mFormat)；//调用decode解析uevent数据

onEvent(evt);//传递NetlinkEvent事件

}

NetlinkHandler.cpp 1440 2016-07-12

void NetlinkHandler::onEvent(NetlinkEvent \*evt) {

VolumeManager \*vm = VolumeManager::Instance();

const char \*subsys = evt->getSubsystem();

if (!strcmp(subsys, "block")) {

vm->handleBlockEvent(evt);

}

}

NM模块的功能就是从Kernel接收Uevent消息，然后转换成一个NetlinkEvent对象，最后会调用VM的处理函数来处理这个NetlinkEvent对象。

### 

### VolumeManager模块的分析

VolumeManager \*VolumeManager::Instance() {

sInstance = new VolumeManager();

return sInstance;

}

Main.cpp 5229 2016-07-12

static int process\_config(VolumeManager \*vm)

{

fstab = fs\_mgr\_read\_fstab(fstab\_filename);

dv = new DirectVolume(vm, &(fstab->recs[i]), flags);

vm->addVolume(dv);

}

#### DirectVolume的分析

DirectVolume::DirectVolume(VolumeManager \*vm, const fstab\_rec\* rec, int flags) :

Volume(vm, rec, flags) {

setState(Volume::State\_NoMedia);

}

int DirectVolume::addPath(const char \*path) {

mPaths->push\_back(new PathInfo(path));

return 0;

}

#### NM和VM交互

NetlinkHandler.cpp 1440 2016-07-12

void NetlinkHandler::onEvent(NetlinkEvent \*evt) {

VolumeManager \*vm = VolumeManager::Instance();

const char \*subsys = evt->getSubsystem();

if (!strcmp(subsys, "block")) {

vm->handleBlockEvent(evt);

}

}

VolumeManager.cpp 55964 2016-07-12

void VolumeManager::handleBlockEvent(NetlinkEvent \*evt) {

//调用每个volume的handleblockevent事件，事件调用directvolume

for (it = mVolumes->begin(); it != mVolumes->end(); ++it) {

(\*it)->handleBlockEvent(evt)；

}

}

DirectVolume.cpp 16842 2016-07-12

int DirectVolume::handleBlockEvent(NetlinkEvent \*evt) {

for (it = mPaths->begin(); it != mPaths->end(); ++it) {

(\*it)->match(dp);

int mNum = getUICCVolumeNum(dp);

setValidSysfs(true);

//创建设备节点

createDeviceNode(nodepath, major, minor);

//添加一个磁盘

handleDiskAdded(dp, evt);

//添加一个分区

handlePartitionAdded(dp, evt);

}

handlePartitionRemoved(dp, evt);

handlePartitionChanged(dp, evt);

}

### 关于 CommandListener模块的分析

CommandListener.cpp 27637 2016-07-12

CommandListener::CommandListener() :

FrameworkListener("vold", true) {

registerCmd(new DumpCmd());

registerCmd(new VolumeCmd());

registerCmd(new AsecCmd());

registerCmd(new ObbCmd());

registerCmd(new StorageCmd());

registerCmd(new DumpCmd());

registerCmd(new VolumeCmd());

registerCmd(new AsecCmd());

registerCmd(new ObbCmd());

registerCmd(new StorageCmd());

}

#### MountService的分析

class MountService extends IMountService.Stub

implements INativeDaemonConnectorCallbacks{//实现该接口

public MountService(Context context) {

mContext = context;

//创建一个HandlerThread

HandlerThread hthread = new HandlerThread(TAG);

hthread.start();

//NativeDaemonConnector用于socket通信，vold表示将于vold通信

mConnector = new NativeDaemonConnector(this, "vold", MAX\_CONTAINERS \* 2, VOLD\_TAG, 25, null);

//启动一个相册和vold通信

Thread thread = new Thread(mConnector, VOLD\_TAG);

thread.start();

}

}

#### 设备插入事件的处理

##### MountService的处理

public boolean onEvent(int code, String raw, String[] cooked) {

else if ((code == VoldResponseCode.VolumeDiskInserted) ||

(code == VoldResponseCode.VolumeDiskRemoved) ||

(code == VoldResponseCode.VolumeBadRemoval)) {

final String label = cooked[2];

final String path = cooked[3];

major = Integer.parseInt(devTok[0]);

minor = Integer.parseInt(devTok[1]);

if (code == VoldResponseCode.VolumeDiskInserted) {

//单独开辟线程处理

public void run() {

rc = doMountVolume(path)

}

}

}

private int doMountVolume(String path) {

//给vold发送mount

mConnector.execute("volume", "mount", path);

}

走了一大圈，最后又回到Vold了。CL模块将收到这个来自MountService的请求，请求的内容为字符串“volume mount/mnt/sdcard”，其中的volume表示命令的名字，CL会根据这个名字找到VolumeCmd对象，并交给它处理这个命令。

##### Vold处理MountService的命令

CommandListener.cpp 27637 2016-07-12

int CommandListener::VolumeCmd::runCommand(SocketClient \*cli,

int argc, char \*\*argv) {

VolumeManager \*vm = VolumeManager::Instance();

cli->sendMsg(ResponseCode::CommandSyntaxError, "Usage: volume mount <path>", false);

cli->sendMsg(ResponseCode::CommandSyntaxError, "Usage: volume unmount <path> [force|force\_and\_revert]", false);

cli->sendMsg(ResponseCode::CommandSyntaxError, "Usage: volume format <path> [wipe]", false);

}

VolumeManager.cpp 55964 2016-07-12

//根据labal找到对应的Volume

nt VolumeManager::mountVolume(const char \*label) {

Volume \*v = lookupVolume(label);

return v->mountVol();

}

Volume.cpp 19816 2016-07-12

int Volume::mountVol() {

//判断这个路径是否已经被mount

if (isMountpointMounted(getMountpoint())) {

setState(Volume::State\_Mounted);

// mCurrentlyMountedKdev = XXX

return 0;

}

n = getDeviceNodes((dev\_t \*) &deviceNodes, 4);

createDeviceNode(nodepath, new\_major, new\_minor)；

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

sprintf(devicePath, "/dev/block/vold/%d:%d", major(deviceNodes[i]),

minor(deviceNodes[i]));

setState(Volume::State\_Checking);

Fat::doMount(devicePath, getMountpoint(), false, false, false,

AID\_MEDIA\_RW, AID\_MEDIA\_RW, 0007, true)；

}

setState(Volume::State\_Mounted);

}

##### MountService处理状态通知

Input 事件上报流程

Input.h (kernel\include\linux) 48470 2015-03-14

static inline void input\_report\_abs(struct input\_dev \*dev, unsigned int code, int value)

{

input\_event(dev, EV\_ABS, code, value);

}

Input.c (kernel\drivers\input) 54718 2015-03-14

void input\_event(struct input\_dev \*dev,

unsigned int type, unsigned int code, int value)

{

input\_handle\_event(dev, NULL, type, code, value);

}

static void input\_handle\_event(struct input\_dev \*dev,

struct input\_handler \*src\_handler,

unsigned int type, unsigned int code, int value)

{

input\_pass\_event(dev, src\_handler, type, code, value);

}

static void input\_pass\_event(struct input\_dev \*dev,

struct input\_handler \*src\_handler,

unsigned int type, unsigned int code, int value)

{

if (handle)

handle->handler->event(handle, type, code, value);

}

Evdev.c (kernel\drivers\input) 22903 2015-03-14

static struct input\_handler evdev\_handler = {

.event = evdev\_event,

.connect = evdev\_connect,

}

Evdev.c (kernel\drivers\input) 22903 2015-03-14

static void evdev\_event(struct input\_handle \*handle,

unsigned int type, unsigned int code, int value)

{

evdev\_pass\_event(client, &event);

}

static void evdev\_pass\_event(struct evdev\_client \*client,

struct input\_event \*event)

{

kill\_fasync(&client->fasync, SIGIO, POLL\_IN);//异步通知函数。

}

Input.c (kernel\drivers\input) 54718 2015-03-14

static const struct file\_operations input\_fops = {

.open = input\_open\_file,

}

static int input\_open\_file(struct inode \*inode, struct file \*file)

{

handler = input\_table[iminor(inode) >> 5];

if (handler)

new\_fops = fops\_get(handler->fops);

//以各自的hander确定对应的fops

err = new\_fops->open(inode, file);

}

Evdev.c (kernel\drivers\input) 22903 2015-03-14

static const struct file\_operations evdev\_fops = {

.owner = THIS\_MODULE,

.read = evdev\_read,

.open = evdev\_open,

}

static int evdev\_open(struct inode \*inode, struct file \*file)

{

int i = iminor(inode) - EVDEV\_MINOR\_BASE;

//根据设备号获取evdev

/\*evdev\_table一共可容纳32个成员，找到次设备号对应的那个\*/  /\*evdev\_table一共可容纳32个成员，找到次设备号对应的那个\*/

evdev = evdev\_table[i];

if (evdev)

get\_device(&evdev->dev);

//创建一个client

client = kzalloc(sizeof(struct evdev\_client) +

bufsize \* sizeof(struct input\_event),

GFP\_KERNEL);

1. //将evdev和client绑定到一起\*/

client->evdev = evdev;

evdev\_attach\_client(evdev, client);

error = evdev\_open\_device(evdev);

}

static int evdev\_open\_device(struct evdev \*evdev)

{

retval = input\_open\_device(&evdev->handle);

}

int input\_open\_device(struct input\_handle \*handle)

{

if (!dev->users++ && dev->open)

retval = dev->open(dev);

}

用户进程读取event的底层实现：

Evdev.c (kernel\drivers\input) 22903 2015-03-14

static const struct file\_operations evdev\_fops = {

.read = evdev\_read,

.write = evdev\_write,

}

static ssize\_t evdev\_read(struct file \*file, char \_\_user \*buffer,

size\_t count, loff\_t \*ppos)

{

while (retval + input\_event\_size() <= count &&

evdev\_fetch\_next\_event(client, &event)) {

input\_event\_to\_user(buffer + retval, &event)

}

}

int input\_event\_to\_user(char \_\_user \*buffer,

const struct input\_event \*event)

{

copy\_to\_user(buffer, &compat\_event,

sizeof(struct input\_event\_compat))

}

static ssize\_t evdev\_write(struct file \*file, const char \_\_user \*buffer,

size\_t count, loff\_t \*ppos)

{

input\_event\_from\_user(buffer + retval, &event)

}

Iio tmd27723分析：

static int \_\_devinit tsl2x7x\_probe(struct i2c\_client \*clientp,

const struct i2c\_device\_id \*id)

{

struct iio\_dev \*indio\_dev;

//分配iio结构体

indio\_dev = iio\_allocate\_device(sizeof(\*chip));

//分配input结构体

tsl2x7x\_prox\_dev = kzalloc(sizeof(struct input\_dev), GFP\_KERNEL);

//注册input：

ret = input\_register\_device(tsl2x7x\_prox\_dev);

//创建sys文件系统

ret = sysfs\_create\_group(&clientp->dev.kobj, &tsl2x7x\_prox\_attr\_group);

tsl2x7x\_als\_dev = kzalloc(sizeof(struct input\_dev), GFP\_KERNEL);

tsl2x7x\_als\_dev = input\_allocate\_device();

ret = input\_register\_device(tsl2x7x\_als\_dev);

ret = sysfs\_create\_group(&clientp->dev.kobj, &tsl2x7x\_als\_attr\_group);

chip = iio\_priv(indio\_dev);

chip->client = clientp;

i2c\_set\_clientdata(clientp, indio\_dev);

ret = tsl2x7x\_i2c\_read(chip->client,

TSL2X7X\_CHIPID, &device\_id);

ret = tsl2x7x\_register\_buffer\_funcs\_and\_init(indio\_dev);

ret = iio\_device\_register(indio\_dev);

}

Intel camera 分析：

Isp驱动分析：

Cif\_isp20\_v4l2.c (drivers\media\platform\xgold) 54888 2016-04-01

static int xgold\_v4l2\_init(void)

{

int ret = platform\_driver\_register(&xgold\_v4l2\_plat\_drv);

}

static struct platform\_driver xgold\_v4l2\_plat\_drv = {

.probe = xgold\_v4l2\_drv\_probe,

}

static int xgold\_v4l2\_drv\_probe(struct platform\_device \*pdev)

{

dev = cif\_isp20\_create(&pdev->dev, cif\_isp20\_v4l2\_event,

cif\_isp20\_v4l2\_requeue\_bufs);

ret = v4l2\_device\_register(dev->dev, &dev->v4l2\_dev);

ret = cif\_isp20\_v4l2\_register\_video\_device(

dev,

&cif\_isp20\_v4l2\_dev.node[SP\_DEV].vdev,

"CIF ISP20 SP",

V4L2\_CAP\_VIDEO\_OVERLAY,

CIIF\_ISP20\_V4L2\_SP\_DEV\_MAJOR,

&cif\_isp20\_v4l2\_fops,

&cif\_isp20\_v4l2\_sp\_ioctlops);

ret = register\_cifisp\_device(&dev->isp\_dev,

&cif\_isp20\_v4l2\_dev.node[ISP\_DEV].vdev,

&dev->v4l2\_dev,

dev->config.base\_addr);

ret = cif\_isp20\_v4l2\_register\_video\_device(

dev,

&cif\_isp20\_v4l2\_dev.node[MP\_DEV].vdev,

"CIF ISP20 MP",

V4L2\_CAP\_VIDEO\_CAPTURE,

CIIF\_ISP20\_V4L2\_MP\_DEV\_MAJOR,

&cif\_isp20\_v4l2\_fops,

&cif\_isp20\_v4l2\_mp\_ioctlops);

ret = cif\_isp20\_v4l2\_register\_video\_device(

dev,

&cif\_isp20\_v4l2\_dev.node[DMA\_DEV].vdev,

"CIF ISP20 DMA",

V4L2\_CAP\_VIDEO\_OUTPUT,

CIIF\_ISP20\_V4L2\_DMA\_DEV\_MAJOR,

&cif\_isp20\_v4l2\_fops,

&cif\_isp20\_v4l2\_dma\_ioctlops);

}