咕唧咕唧shubo.lk的专栏

不在乎我拥有多少，而是我奉献多少!

基于Linux 3.0.8 Samsung FIMC（S5PV210）的摄像头驱动框架解读（一）

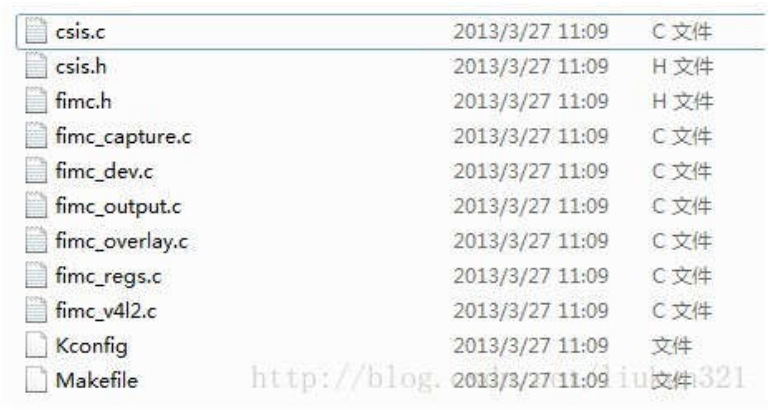
2014-08-04 22:32 6665人阅读 评论(4) 收藏 举报

分类： linux 设备驱动（24） linux 移植（31） linux kernel（6）

作者：咕唧咕唧liukun321

来自：http://blog.csdn.NET/liukun321

FIMC这个名字应该是从S5PC1x0开始出现的，在s5pv210里面的定义是摄像头接口，但是它同样具有图像数据颜色空间转换的作用。而exynos4412对它的定义看起来更清晰些，摄像头接口被定义为FIMC-LITE 。颜色空间转换的硬件结构被定义为FIMC-IS。不多说了，我们先来看看Linux3.0.8 三星的BSP包中与fimc驱动相关的文件。



上面的源码文件组成了整个fimc的驱动框架。通过.c文件的命名也大致可以猜测到FIMC的几个用途：

1、**Capture** ，Camera Interface 用于控制Camera，及m2m操作

2、**Output**，这个用途可以简单看成：只使用了FIMC的m2m功能，这里fimc实际上就成了一个带有颜色空间转换功能的高速DMA。

3、**Overlay**，比如**Android** 的Overlay就依赖了FIMC的这个功能，可以简单把它看作是个m2fb，当然实质上还是m2m。

清楚FIMC的大致用途了。再来说说，每个C文件在FIMC驱动框架中扮演了何种角色：

csis.c文件，用于MIPI 接口的摄像头设备，这里不多说什么了。

fimc_dev.c 是驱动中对FIMC硬件设备最高层的抽象，这在后面会详细介绍。

fimc_v4l2.c Linux驱动中 ，将fimc 设备的功能操作接口（**Capture，output，Overlay**），用v4l2框架封装。在应用层用过摄像头设备，或在应用层使用FMIC设备完成过m2m操作的朋友应该都清楚，fimc经层层封装后最终暴露给用户空间的是v4l2 标准接口的设备文件 videoX。 这里面也引出了一个我们应该关注的问题：Fimc设备在软件层上是如何同摄像头设备关联的。

fimc_capture.c 实现对camera Interface 的控制操作，它实现的基础依赖硬件相关的摄像头驱动（eg.ov965X.c / ov5642.c 等）。 并且提供以下函数接口，由**fimc_v4l2.c**文件进一步封装

int fimc_g_parm(struct file *file, void*fh, struct v4l2_streamparm *a)

int fimc_s_parm(struct file *file, void*fh, struct v4l2_streamparm *a)

文章：10篇
阅读：90935

文章分类

liukun321原创s5pv210+XC3S400A工程板 (1)
视频编解码（210硬件编解码） (3)
Android 编译 (4)
Android Develope (20)
IT新动态 (4)
MultiMedia (16)
程序人生 (8)
linux kernel (7)
linux 应用程序编程 (8)
linux 设备驱动 (25)
linux 移植 (32)
linux 系统使用及维护 (2)
硬件 (6)
错误杂项 (1)
S5pv210-u-boot (12)
uboot移植 (18)
OS (35)
s3c2440 (4)
文件系统 (4)
ARM语言基础 (22)
各模式下的UART (5)
关于中断和代码搬运 (4)
44B0底层分析+bootloader (10)
QT (1)
Opencv (4)
数据库 (1)
hadoop (2)
网络与安全 (5)

文章存档

2017年03月 (1)
2017年01月 (3)
2016年03月 (2)
2016年01月 (1)
2015年09月 (1)

展开

阅读排行

关于connect: network is unr... (209752)
学习Opencv 2.4.9（一）---O.. (56574)
linux-3.1.4下的驱动模块 "Unk... (21840)
ldconfig提示is not a symbol... (20877)
错误代码：0x800704cf 不能访.. (19600)
Uncompressing Linux..... do... (18555)
u-boot for tiny210 ver3.1 (b... (18041)
教你如何用路由器连接网页登... (17210)
学习Opencv2.4.9(四)---SVM... (16775)
warning: control reaches en... (15727)

评论排行

u-boot for tiny210 ver3.1 (b... (74)
u-boot for tiny210 ver4.0 (b... (44)
u-boot for tiny210 ver1.0(b... (31)
Gps driver for Tiny4412+An... (25)
基于Android2.3的车载导航--... (20)
基于S5pv210流媒体服务器的... (18)
结构体在内存中的对齐规则 (10)
u-boot for tiny210 ver3.0 (b... (10)
S5pv210 HDMI 接口在 Linux... (10)
关于connect: network is unr... (9)

推荐文章

intfimc_queryctrl(struct file *file, void *fh, struct v4l2_queryctrl *qc)

intfimc_querymenu(struct file *file, void *fh, struct v4l2_querymenu *qm)

intfimc_enum_input(struct file *file, void *fh, struct v4l2_input *inp)

intfimc_g_input(struct file *file, void *fh, unsigned int *i)

intfimc_release_subdev(struct fimc_control *ctrl)

intfimc_s_input(struct file *file, void *fh, unsigned int i)

intfimc_enum_fmt_vid_capture(struct file *file, void *fh,struct v4l2_fmtdesc *f)

intfimc_g_fmt_vid_capture(struct file *file, void *fh, struct v4l2_format *f)

intfimc_s_fmt_vid_capture(struct file *file, void *fh, struct v4l2_format *f)

intfimc_try_fmt_vid_capture(struct file *file, void *fh, struct v4l2_format *f)

intfimc_reqbufs_capture(void *fh, struct v4l2_requestbuffers *b)

intfimc_querybuf_capture(void *fh, struct v4l2_buffer *b)

intfimc_g_ctrl_capture(void *fh, struct v4l2_control *c)

intfimc_s_ctrl_capture(void *fh, struct v4l2_control *c)

intfimc_s_ext_ctrls_capture(void *fh, struct v4l2_ext_controls *c)

intfimc_cropcap_capture(void *fh, struct v4l2_cropcap *a)

intfimc_g_crop_capture(void *fh, struct v4l2_crop *a)

intfimc_s_crop_capture(void *fh, struct v4l2_crop *a)

intfimc_start_capture(struct fimc_control *ctrl)

intfimc_stop_capture(struct fimc_control *ctrl)

intfimc_streamon_capture(void *fh)

intfimc_streamoff_capture(void *fh)

intfimc_qbuf_capture(void *fh, struct v4l2_buffer *b)

intfimc_dqbuf_capture(void *fh, struct v4l2_buffer *b)

fimc_output.c 实现fimc m2m操作，需要用FIMC实现硬件颜色空间转换的时候，这个文件里的函数就派上作用了，另外在fimc 用于Capture 和 overlay 过程本质上也包含m2m操作。因此除了提供功能函数接口，由**fimc_v4l2.c**文件进一步封装。另外还提供了一些功能函数供**fimc_dev.c**调用，比如用于设置一个m2m过程的srcAddr（源地址）和 dstAddr（目的地址）。这部分接口太多就不贴出来了。

fimc_overlay.c 实现fimc overlay操作。同样提供函数接口，由**fimc_v4l2.c**文件进一步封装。

fimc_regs.c Fimc硬件相关操作，基本寄存器配置等。这个文件提供函数接口供**fimc_capture.c**、**fimc_output.c**、**fimc_overlay.c**调用。

通过刚才的分析，可以总结出下面的源码结构图：

好了，框架有了，再来看源码就轻松多了

接下来，先来看看FIMC设备的注册过程。以FIMC-0为例，说说/dev/video0 这个设备文件是怎么出来的。

* 5月书讯：流畅的Python，终于等到你！
* 【新收录】CSDN日报 —— Kotlin 专场
* Android中带你开发一款自动爆破签名校验工具kstools
* Android图片加载框架最全解析——深入探究Glide的缓存机制
* Android 热修复 Tinker Gradle Plugin解析
* Unity Shader-死亡溶解效果

最新评论

结构体在内存中的对齐规则
Genius_pig ：解释的吼啊啊啊！！！

悲催的程序员，以及程序员的悲催
咕唧咕唧shuboLK ：@fc4soda:文章第一句是本文中心是^_^: 偶尔怀疑自己怀疑生活，但不后悔当初的选择。做个程序...

悲催的程序员，以及程序员的悲催
fc4soda ：有点忧伤.....

基于Android2.3的车载导航---andorid G...
ys166166 ：您好博主，我想给天敏T2电视盒增加GPS模块，在网上也发现新帝豪车载机可以通过加装GPS模块实现GP...

u-boot for tiny210 ver3.0 (by liukun32...
wv112406 ：https://github.com/kangear/tiny210v2-uboot 这个是你发的么...

学习Opencv 2.4.9（一）---Opencv + v...
dripFly ：@u010047380:是因为第9步你添加lib时，添加的不是尾号为249的lib，而是243的li...

学习Opencv 2.4.9（一）---Opencv + v...
心疼每一个微信 ：环境变量bin后面要加个斜杠

关于Linux的线程休眠函数sleep/usleep/n...
laoAyang ：usleep是毫秒， 1/1000 000s

学习Opencv 2.4.9（一）---Opencv + v...
qq_35841125 ：你好 opencv2.4.9你有吗可以发个给我吗

学习Opencv2.4.9(四)---SVM支持向量机

先看几个关键结构：

首先是 s3c_platform_fimcfimc_plat_lsi；也就是抽象fimc模块的**数据结构**，fimc_plat_lsi还包含了一个.camera成员。该结构初始化如下

[cpp]

```cpp
static struct s3c_platform_fimc  fimc_plat_lsi = {
    .srclk_name = "mout_mpll",
    .clk_name   = "sclk_fimc",
    .lclk_name  = "fimc",
    .clk_rate   = 166750000,
#if defined(CONFIG_VIDEO_S5K4EA)
    .default_cam    = CAMERA_CSI_C,
#else
#ifdef CAM_ITU_CH_A
    .default_cam    = CAMERA_PAR_A,
#else
    .default_cam    = CAMERA_PAR_B,
#endif
#endif
    .camera     = {
#ifdef CONFIG_VIDEO_S5K4ECGX
            &s5k4ecgx,
#endif
#ifdef CONFIG_VIDEO_S5KA3DFX
            &s5ka3dfx,
#endif
#ifdef CONFIG_VIDEO_S5K4BA
            &s5k4ba,
#endif
#ifdef CONFIG_VIDEO_S5K4EA
            &s5k4ea,
#endif
#ifdef CONFIG_VIDEO_TVP5150
            &tvp5150,
#endif
#ifdef CONFIG_VIDEO_OV9650
            &ov9650,
#endif
    },
    .hw_ver     = 0x43,
};
```

可以看到在s3c_platform_fimc中有一个camera成员。这里重点看一下ov9650.展开ov9650

[cpp]

```cpp
static struct s3c_platform_camera ov9650 = {
    #ifdef CAM_ITU_CH_A
    .id     = CAMERA_PAR_A,
    #else
    .id     = CAMERA_PAR_B,
    #endif
    .type       = CAM_TYPE_ITU,
    .fmt        = ITU_601_YCBCR422_8BIT,
    .order422   = CAM_ORDER422_8BIT_YCBYCR,
    .i2c_busnum = 0,
    .info       = &ov9650_i2c_info,
    .pixelformat    = V4L2_PIX_FMT_YUYV,
    .srclk_name = "mout_mpll",
    /* .srclk_name  = "xusbxti", */
    .clk_name   = "sclk_cam1",
    .clk_rate   = 40000000,
    .line_length    = 1920,
    .width      = 1280,
    .height     = 1024,
    .window     = {
        .left   = 0,
        .top    = 0,
        .width  = 1280,
        .height = 1024,
    },

    /* Polarity */
    .inv_pclk   = 1,
    .inv_vsync  = 1,
    .inv_href   = 0,
    .inv_hsync  = 0,

    .initialized    = 0,
    .cam_power  = ov9650_power_en,
};
```

这个结构体，实现了对ov9650摄像头硬件结构的抽象。定义了摄像头的关键参数和基本特性。

因为fimc设备在linux3.0.8内核中作为一个平台设备加载，而上面提到的s3c_platform_fimcfimc_plat_lsi仅是fimc的抽象数据而非设备。这就需要将抽象fimc的结构体作为fimc platform_device 的一个私有数据。所以就有了下面的过程。s3c_platform_fimcfimc_plat_lsi 结构在板级设备初始化XXX_machine_init(void) 过程作为s3c_fimc0_set_platdata 的实参传入。之后fimc_plat_lsi就成为了fimc设备的platform_data。

```cpp
s3c_fimc0_set_platdata(&fimc_plat_lsi);
s3c_fimc1_set_platdata(&fimc_plat_lsi);
s3c_fimc2_set_platdata(&fimc_plat_lsi);
```

以s3c_fimc0_set_platdata为例展开

```cpp
void __init s3c_fimc0_set_platdata(struct s3c_platform_fimc *pd)
{
    struct s3c_platform_fimc *npd;

    if (!pd)
        pd = &default_fimc0_data;

    npd = kmemdup(pd, sizeof(struct s3c_platform_fimc), GFP_KERNEL);
    if (!npd)
        printk(KERN_ERR "%s: no memory for platform data\n", __func__);
    else {
        if (!npd->cfg_gpio)
            npd->cfg_gpio = s3c_fimc0_cfg_gpio;

        if (!npd->clk_on)
            npd->clk_on = s3c_fimc_clk_on;

        if (!npd->clk_off)
            npd->clk_off = s3c_fimc_clk_off;

        npd->hw_ver = 0x45;

        /* starting physical address of memory region */
        npd->pmem_start = s5p_get_media_memory_bank(S5P_MDEV_FIMC0, 1);
        /* size of memory region */
        npd->pmem_size = s5p_get_media_memsize_bank(S5P_MDEV_FIMC0, 1);

        s3c_device_fimc0.dev.platform_data = npd;
    }
}
```

最后一句是关键 s3c_device_fimc0.dev.platform_data = npd;

看一下s3c_device_fimc0定义：

```cpp
struct platform_device s3c_device_fimc0 = {
    .name       = "s3c-fimc",
    .id     = 0,
    .num_resources  = ARRAY_SIZE(s3c_fimc0_resource),
    .resource   = s3c_fimc0_resource,
};
```

fimc的抽象数据，则作为它的私有数据被包含进了s3c_device_fimc0这个结构中。到这里才完成了FIMC平台设备的最终定义。这个平台设备的定义s3c_device_fimc0又被添加到了整个硬件平台的 platform_device 列表中，最终在XXX_machine_init(void) 函数中调用 platform_add_devices(mini210_devices, ARRAY_SIZE(mini210_devices)); 完成所有platform_device 的注册：

```cpp
static struct platform_device *mini210_devices[] __initdata = {
    &s3c_device_adc,
    &s3c_device_cfcon,
    &s3c_device_nand,
    。。。
    &s3c_device_fb,
    &mini210_lcd_dev,
#ifdef CONFIG_VIDEO_FIMC
    &s3c_device_fimc0,
    &s3c_device_fimc1,
    &s3c_device_fimc2,
}
```

```html
platform_add_devices(mini210_devices, ARRAY_SIZE(mini210_devices));
```

platform_device 被加载后，等待与之匹配的platform_driver。若此时fimc driver 的驱动模块被加载。这个时候，fimc_dev.c文件里的static int __devinit fimc_probe(structplatform_device *pdev) 函数上场了。

[cpp]

```cpp
static int __devinit fimc_probe(struct platform_device *pdev)
{
    struct s3c_platform_fimc *pdata;
    struct fimc_control *ctrl;
    struct clk *srclk;
    int ret;
    if (!fimc_dev) {
        fimc_dev = kzalloc(sizeof(*fimc_dev), GFP_KERNEL);
        if (!fimc_dev) {
            dev_err(&pdev->dev, "%s: not enough memory\n",
                __func__);
            return -ENOMEM;
        }
    }

    ctrl = fimc_register_controller(pdev);
    if (!ctrl) {
        printk(KERN_ERR "%s: cannot register fimc\n", __func__);
        goto err_alloc;
    }

    pdata = to_fimc_plat(&pdev->dev);
    if (pdata->cfg_gpio)
        pdata->cfg_gpio(pdev);

#ifdef REGULATOR_FIMC
    /* Get fimc power domain regulator */
    ctrl->regulator = regulator_get(&pdev->dev, "pd");
    if (IS_ERR(ctrl->regulator)) {
        fimc_err("%s: failed to get resource %s\n",
                __func__, "s3c-fimc");
        return PTR_ERR(ctrl->regulator);
    }
#endif //REGULATOR_FIMC
    /* fimc source clock */
    srclk = clk_get(&pdev->dev, pdata->srclk_name);
    if (IS_ERR(srclk)) {
        fimc_err("%s: failed to get source clock of fimc\n",
                __func__);
        goto err_v4l2;
    }

    /* fimc clock */
    ctrl->clk = clk_get(&pdev->dev, pdata->clk_name);
    if (IS_ERR(ctrl->clk)) {
        fimc_err("%s: failed to get fimc clock source\n",
            __func__);
        goto err_v4l2;
    }

    /* set parent for mclk */
    clk_set_parent(ctrl->clk, srclk);

    /* set rate for mclk */
    clk_set_rate(ctrl->clk, pdata->clk_rate);

    /* V4L2 device-subdev registration */
    ret = v4l2_device_register(&pdev->dev, &ctrl->v4l2_dev);
    if (ret) {
        fimc_err("%s: v4l2 device register failed\n", __func__);
        goto err_fimc;
    }

    /* things to initialize once */
    if (!fimc_dev->initialized) {
        ret = fimc_init_global(pdev);
        if (ret)
            goto err_v4l2;
    }

    /* video device register */
    ret = video_register_device(ctrl->vd, VFL_TYPE_GRABBER, ctrl->id);
    if (ret) {
        fimc_err("%s: cannot register video driver\n", __func__);
        goto err_v4l2;
    }

    video_set_drvdata(ctrl->vd, ctrl);

    ret = device_create_file(&(pdev->dev), &dev_attr_log_level);
    if (ret < 0) {
        fimc_err("failed to add sysfs entries\n");
        goto err_global;
    }
    printk(KERN_INFO "FIMC%d registered successfully\n", ctrl->id);

    return 0;

err_global:
    video_unregister_device(ctrl->vd);

err_v4l2:
    v4l2_device_unregister(&ctrl->v4l2_dev);

err_fimc:
    fimc_unregister_controller(pdev);
```

```
97.
98.  err_alloc:
99.      kfree(fimc_dev);
100.     return -EINVAL;
101.
102. }
```

在fimc_probe函数中有这么一段

```
[cpp]
01.  if(!fimc_dev->initialized) {
02.                      ret = fimc_init_global(pdev);
03.                      if (ret)
04.                               goto err_v4l2;
05.           }
```

这段代码执行过程：首先判断fimc是否已经被初始化完成（此时FIMC是忙状态的），如果没有被初始化，则执行fimc_init_global(pdev);函数，它的作用是先判断平台数据中是否初始化了摄像头结构（即前面提到的.camera成员），从平台数据中获得摄像头的时钟频率并将平台数据中内嵌的s3c_platform_camera结构数据保存到该驱动模块全局的fimc_dev中，感兴趣的朋友可以展开这个函数看一下，这里就不再贴出来了。

紧接着这段代码还执行了两个非常关键的过程：

```
[cpp]
01.  ret= v4l2_device_register(&pdev->dev, &ctrl->v4l2_dev);
02.           if (ret) {
03.                      fimc_err("%s: v4l2device register failed\n", __func__);
04.                      goto err_fimc;
05.           }
```

这个函数里的核心完成了对v4l2_dev->subdev链表头的初始化，并将ctrl->v4l2_dev关联到pdev->dev结构的私有数据的driver_data成员中（即完成了pdev->dev->p->driver_data= ctrl->v4l2_dev; ），也就是实现了v4l2_dev向内核结构注册的过程。

```
[cpp]
01.  ret= video_register_device(ctrl->vd, VFL_TYPE_GRABBER, ctrl->id);
02.           if (ret) {
03.                      fimc_err("%s: cannotregister video driver\n", __func__);
04.                      goto err_v4l2;
05.           }
06.
07.           video_set_drvdata(ctrl->vd, ctrl);
08.
09.           ret = device_create_file(&(pdev->dev),&dev_attr_log_level);
```

上面的过程完成了对video_device 设备的注册，并且在sys 目录下生成了对应的属性文件。如果系统中移植有mdev，将会生成对应设备节点/dev/videoX。

其实到目前为止，只完成了fimc设备主要数据结构的初始化和注册，几乎没有操作fimc或摄像头的硬件寄存器。也没有完成FIMC驱动和摄像头的驱动模块的软件关联。我们是如何做到仅操作fimc的设备节点/dev/videoX就能控制摄像头设备的效果呢？下回分解吧。。。

- 上一篇 学习Opencv 2.4.9（一）---Opencv + vs2012环境配置
- 下一篇 学习Opencv 2.4.9（二） ---操作像素