

# 基于Linux 3.0.8 Samsung FIMC (S5PV210) 的摄像头驱动框架解读 (一) - 咕唧咕唧shubo.1k的专栏 - 博客频道

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)



版权声明: 本文为博主原创文章, 未经博主允许不得转载。

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。

	csis.c	2013/3/27 11:09	C 文件
	csis.h	2013/3/27 11:09	H 文件
	fimc.h	2013/3/27 11:09	H 文件
	fimc_capture.c	2013/3/27 11:09	C 文件
	fimc_dev.c	2013/3/27 11:09	C 文件
	fimc_output.c	2013/3/27 11:09	C 文件
	fimc_overlay.c	2013/3/27 11:09	C 文件
	fimc_regs.c	2013/3/27 11:09	C 文件
	fimc_v4l2.c	2013/3/27 11:09	C 文件
	Kconfig	2013/3/27 11:09	文件
	Makefile	2013/3/27 11:09	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)

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)

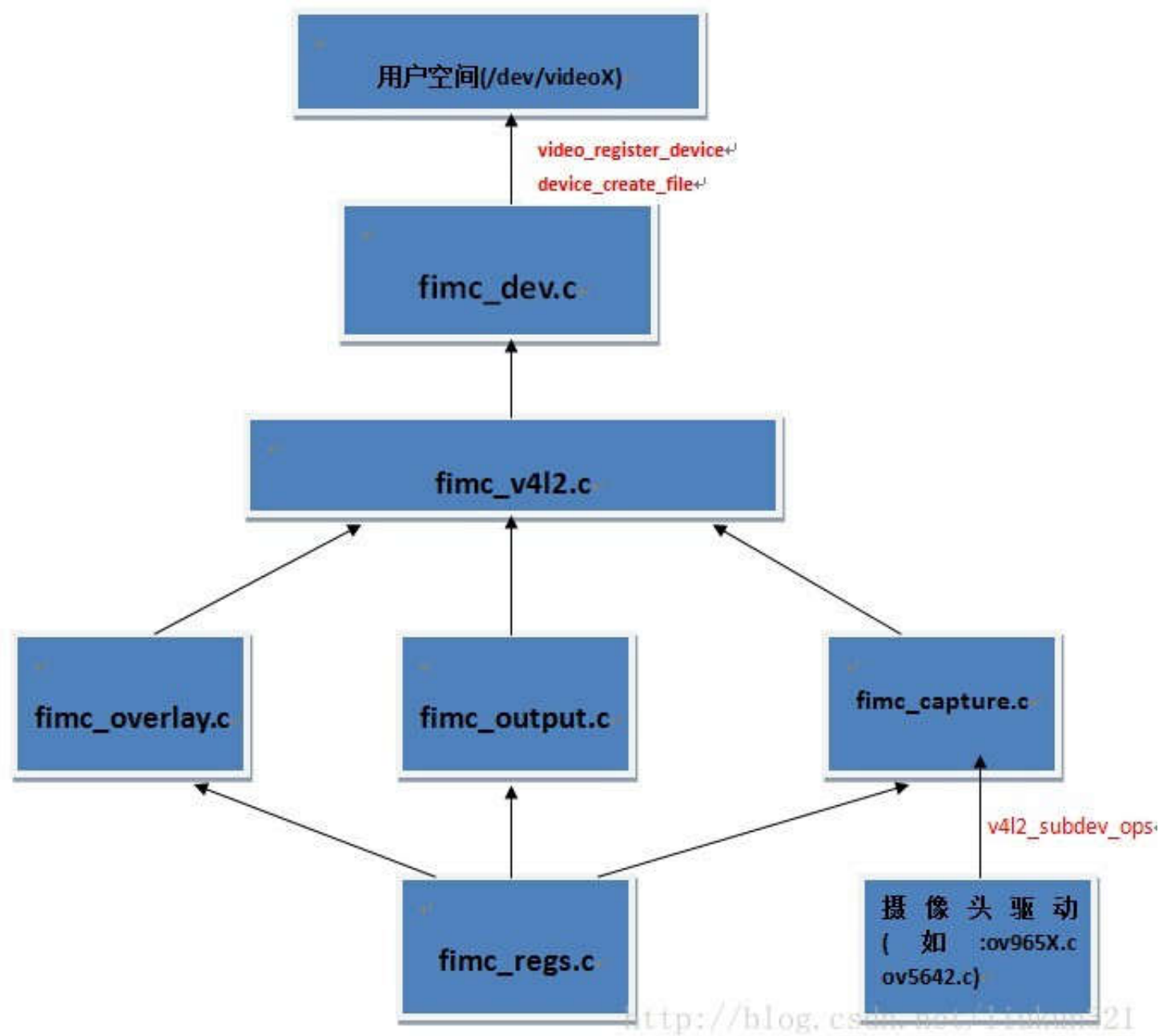
```

`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。

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

```

1.      .srclk_name = "mout_mpll",
2.      /* .srclk_name = "xusbxti", */
3.      .clk_name   = "sclk_cam1",
4.      .clk_rate   = 40000000,
5.      .line_length = 1920,
6.      .width      = 1280,
7.      .height     = 1024,
8.      .window     = {
9.          .left    = 0,
10.         .top     = 0,
11.         .width   = 1280,
12.         .height  = 1024,
13.     },
14.     /* Polarity */
15.     .inv_pclk    = 1,
16.     .inv_vsync   = 1,
17.     .inv_href    = 0,
18.     .inv_hsync   = 0,
19.     .initialized  = 0,
20.     .cam_power   = ov9650_power_en,
21. };
  
```

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。