S5PV210 camera驱动分析（android）

一、Camera工作原理介绍

1． 结构

一般来说，camera主要是由lens和sensor IC两部分组成，其中有的sensor IC集成了DSP，有的没有集成，但也需要外部DSP处理。细分的来讲，camera设备由下边几部分构成：

1） lens（镜头）

一般camera的镜头结构是有几片透镜组成，分有塑胶透镜（Plastic)和玻璃透镜(Glass) ，通常镜头结构有：1P,2P,1G1P,1G3P,2G2P,4G等。 2） sensor（图像传感器）

Senor是一种半导体芯片，有两种类型：CCD和CMOS。Sensor将从lens上传导过来的光线转换为电信号，再通过内部的AD转换为数字信号。由于Sensor的每个pixel只能感光R光或者B光或者G光，因此每个像素此时存贮的是单色的，我们称之为RAW DATA数据。要想将每个像素的RAW DATA数据还原成三基色，就需要ISP来处理。

3）ISP（图像信号处理）

主要完成数字图像的处理工作，把sensor采集到的原始数据转换为显示支持的格式。

4）CAMIF（camera控制器）

芯片上的camera接口电路，对设备进行控制，接收sensor采集的数据交给CPU，并送入LCD进行显示。

2． 工作原理

外部光线穿过lens后，经过color filter滤波后照射到Sensor面上， Sensor将从lens上传导过来的光线转换为电信号，再通过内部的AD转换为数字信号。如果Sensor没有集成DSP，则通过DVP的方式传输到baseband，此时的数据格式是RAW DATA。如果集成了DSP，则RAW DATA 数据经过AWB、color matrix、lens shading、gamma、sharpness、AE和de-noise处理，后输出YUV或者RGB格式的数据。

最后会由CPU送到framebuffer中进行显示，这样我们就看到camera

拍摄到的景象了。 3． YUV

与YCbCr

YUV和RGB一样，是色彩空间中常用的色彩模型之一，两者可以相互转换。YUV中得Y表示亮度，U和V表示色度。与RGB相比，它的优点在于占用更少的空间。

YCbCr 则是在世界数字组织视频标准研制过程中作为ITU - R BT601 建议的一部分, 其实是YUV经过缩放和偏移的翻版。其中Y与YUV 中的Y含义一致, Cb , Cr 同样都指色彩, 只是在表示方法上不同而已。在YUV 家族中, YCbCr 是在计算机系统中应用最多的成员, 其应用领域很广泛,JPEG、MPEG均采用此格式。一般人们所讲的YUV大多是指YCbCr。YCbCr 有许多取样格式, 如4∶4∶4 , 4∶2∶2 , 4∶1∶1 和4∶2∶0。

二、Camera硬件

1． CAMIF

如下是S5PV210的camera系统的结构图：

S5PV210的camera接口控制器叫FIMC4.3，它支持ITU R BT-601/656、AXI和MIPI(CSI)三种接口方式，最大输入像素是8192\*8192。S5PV210有三组camera接口。

主要特性：

支持多种输入接口类型：

 ITU-R BT 601/656模式  DMA(AXI 64位)模式  MIPI(CSI)模式  Direct FIFO模式

支持多种输出接口：DMA模式/Direct FIFO模式 支持数码调焦（DZI） 最大输入像素8192\*8192 支持图像翻转、旋转等处理效果 生成多种图片格式 支持采集帧控制

2． 接口信号

FIMC信号定义如下所示(YCbCr模式)

Signal I/O

帧同步信号 VSYNC I 行同步信号 HREF I 像素时钟 PCLK I 像素数据 DATA[7:0] I FIELD信号 FIELD O

Description

Type

Muxed

系统时钟信号 CAM\_MCLK O

通过CAM\_MCLK给摄像头提供时钟，RST是复位线，PWDN在摄像头工作时应该始终为低。HREF是行参考信号，PCLK是像素时钟，VSYNC是场同步信号。一旦给摄像头提供了时钟，并且复位摄像头，摄像头就开始工作了，通过HREF，PCLK和VSYNC同步传输数字图像信号。数据是通过DATA0~DATA7这八根数据线并行送出的。

3

．

工作时序

FIMC43支持如下两种视频数据：

ITU-R BT 601输入时序图

这种方式下行和帧同步信号独立于视频数据，因此需要同步信号。

ITU-R BT 656输入时序图

这种方式下同步信号已经内嵌到视频数据中了，因此不需要额外的行和帧同步信号。

（ITU-R BT 601: 16位数据传输；21芯；Y、U、V信号同时传输。

ITU-R BT 656: 9芯，不需要同步信号；8位数据传输；串行视频传输；传输速率是601的2倍；先传Y，后传UV。）

同步信号的时延参数

t1：表示VSYNC前、后插入周期 t2：表示HREF前插入周期 t3：表示 HREF宽度

t4：表示HREF后插入周期

4． 外部接口

硬件原理图的CAM A部分：

CAM B部分

5．

Camera内部结构图

下图是camera内部结构框图，以OV sensor为例：

三、Camera驱动

1． V4L2

1）简介

在Linux中，摄像头方面的标准化程度比较高，这个标准就是V4L2驱动程序，这也是业界比较公认的方式。

V4L全称是Video for Linux，是Linux内核中标准的关于视频驱动程序，目前使用比较多的版本是Video for Linux 2，简称V4L2。它为Linux下的视频驱动提供了统一的接口，使得应用程序可以使用统一的API操作不同的视频设备。从内核空间到用户空间，主要的数据流和控制类均由V4L2驱动程序的框架来定义。

V4L2驱动程序一般只提供Video数据的获得，而如何实现视频预览，如何向上层发送数据，如何把纯视频流和取景器、视频录制等实际业务组织起来，都是camera的硬件抽象层需要负责的工作。

V4L2驱动核心实现为如下文件：drivers/media/video/v4l2-dev.c。

V4l2-dev.h中定义的video\_device是V4L2驱动程序的核心数据结构，它为具体的摄像头sensor驱动提供了接口调用。

V4l2的采集过程（应用程序）： 1) 打开设备，获得文件描述符； 2) 设置图片格式； 3) 分配缓冲区；

4) 启动采集过程，读取数据； 5) 停止采集，关闭设备。

2）数据结构

要数据结构是是video\_devvice，定义在在v4l2\_dev.hh中： V4L2的主要

struct video\_\_device {

/\* devicce ops \*/

const sttruct v4l2\_file\_\_operations \*ffops; /\*接口函函数指针\*/

/\* sysfs \*/ struct device dev; struct cdev \*cdev;

/\* Set either parent or v4l2\_dev if yyour driver usees v4l2\_devicee \*/

/\* v4l 设设备结构 \*/ /\* 字符设设备结构\*/

struct device \*parent;; /\* 设备父指针 \*// struct v44l2\_device \*v44l2\_dev; /\* vv4l2设备指针\*/ \*

/\* device info \*/

char name[32]; /\*设备名称\*/

int vfl\_type;

/\* 'minor' is set to -1 if the registration failed \*/ int minor; /\*次设备号\*/

u16 num;

/\* use bitops to set/clear/test flags \*/ unsigned long flags;

/\* attribute to differentiate multiple indices on one physical device \*/

int index;

int debug; /\* debug 级别\*/

/\* Video 标准变量 \*/

/\* 回调函数 \*/

/\* ioctl 回调函数 \*/ };

const struct v4l2\_ioctl\_ops \*ioctl\_ops; void (\*release)(struct video\_device \*vdev); v4l2\_std\_id tvnorms;

/\* Supported tv norms \*/

/\* V4L2 file handles \*/ spinlock\_t

fh\_lock; /\* Lock for all v4l2\_fhs \*/

struct list\_head fh\_list; /\* List of struct v4l2\_fh \*/

v4l2\_std\_id current\_norm; /\* Current tvnorm \*/

主要接口函数有：

int video\_register\_device(struct video\_device \*vdev, int type, int nr)；

static int v4l2\_ioctl(struct inode \*inode, struct file \*filp, long arg)；

unsigned int cmd, unsigned

2. FIMC

1）简介

FIMC这个模块不仅仅是一个摄像头的控制接口，它还承担着V4L2的output功能和overlay的功能。

FIMC的驱动在内核中的位置：drivers/media/video/samsung/fimc 它包含下边的文件：

fimc\_regs.c fimc\_capture.c fimc\_dev.c fimc\_output.c fimc\_overlay.c fimc\_v4l2.c

它们的组织关系如下：

可以看到，FIMC的驱动实现了v4l2所有的接口，可以分为v4l2-input

设备接口，v4l2-output设备接口以及v4l2-overlay设备接口。这里我们主要关注v4l2-input设备接口，因为摄像头属于视频输入设备。

fimc\_v4l2.c里面注册了很多的回调函数，都是用于实现v4l2的标准接口的，但是这些回调函数基本上都不是在fimc\_v4l2.c里面实现的，而是有相应的.c分别去实现。比如：

v4l2-input设备的操作实现： fimc\_capture.c v4l2-output设备的操作实现： fimc\_output.c v4l2-overlay设备的操作实现： fimc\_overlay.c

这些代码其实都是和具体硬件操作无关的，这个驱动把所有操作硬件寄存器的代码都写到一个文件里面了，就是fimc40\_regs.c。这样把硬件相关的代码和硬件无关的代码分开来实现是非常好的方式，可以最大限度的实现代码复用。 2） 数据结构

FIMC的主要数据结构fimc\_control，定义在fimc.h中：

struct fimc\_control {

int id; /\* 控制器 id \*/ char name[16]; atomic\_t in\_use;

void \_\_iomem \*regs; /\* 寄存器 i/o \*/

struct clk

\*clk;

/\* interface clock \*/ /\* pd regulator \*/

/\* for reserved mem \*/

struct regulator \*regulator;

struct fimc\_meminfo mem;

/\* kernel helpers \*/ struct mutex

lock;

/\* controller lock \*/

struct mutex alloc\_lock; struct mutex v4l2\_lock; wait\_queue\_head\_t wq; struct device \*dev; int irq;

/\* v4l2 related \*/

struct video\_device \*vd; struct v4l2\_device v4l2\_dev;

/\* fimc specific \*/

/\* H/W limitation \*/

/\* activated camera \*/

struct fimc\_limit \*limit;

struct s3c\_platform\_camera \*cam;

struct fimc\_capinfo struct fimc\_outinfo

\*cap; \*out;

/\* capture dev info \*/ /\* output dev info \*/ /\* fimd info \*/ /\* scaler info \*/

struct fimc\_fbinfo fb; struct fimc\_scaler sc;

struct fimc\_effect fe; /\* fimc effect info \*/

enum fimc\_status status; enum fimc\_log log;

u32 ctx\_busy[FIMC\_MAX\_CTXS]; };

因为FIMC一共有三套一样的控制器（fimc0, fimc1, fimc2），所以驱动里使用了一个数组来描述：： struct video\_device fimc\_video\_device[FIMC\_DEVICES] = {

[0] = {

.fops = fimc\_fops,

.ioctl\_ops = fimc\_v4l2\_ops, .release = fimc\_vdev\_release,

},

[1] = {

.fops = fimc\_fops,

.ioctl\_ops = fimc\_v4l2\_ops, .release = fimc\_vdev\_release,

},

[2] = {

.fops = fimc\_fops,

.ioctl\_ops = fimc\_v4l2\_ops,

.release = fimc\_vdev\_release,

}, };

fb\_ops结构体是针对v4l2设备的基本操作，定义如下：

static const struct v4l2\_file\_operations fimc\_fops = { .owner = THIS\_MODULE, .open = fimc\_open, .release = fimc\_release,

.ioctl

= video\_ioctl2,

.read = fimc\_read, .write = fimc\_write, .mmap = fimc\_mmap, .poll = fimc\_poll, };

3）FIMC初始设置

在S5PV210中，FIMC初始设置代码在 /drivers/ arch/arm/mach-s5pv210/mach-t34h.c中：

static struct s3c\_platform\_fimc fimc\_plat = { .srclk\_name = "mout\_mpll", .clk\_name = "sclk\_fimc", .lclk\_name = "sclk\_fimc\_lclk", .clk\_rate = 166750000,

.default\_cam = CAMERA\_CSI\_C,

.camera

= {

mt9p111,//5M back cam s5k6aafx,///1.3M front cam

},

.hw\_ver = 0x43, };

对于GPIO的配置代码在 /drivers/ arch/arm/mach-s5pv210/setup-fimc0.c中： void s3c\_fimc0\_cfg\_gpio(struct platform\_device \*pdev) {

/\* CAM A port(b0010) : PCLK, VSYNC, HREF, DATA[0-4] \*/ for (i = 0; i < 8; i++) { int i = 0;

s3c\_gpio\_cfgpin(S5PV210\_GPE0(i), S3C\_GPIO\_SFN(2)); s3c\_gpio\_setpull(S5PV210\_GPE0(i), S3C\_GPIO\_PULL\_NONE); }

/\* CAM A port(b0010) : DATA[5-7], CLKOUT(MIPI CAM also), FIELD \*/ for (i = 0; i < 5; i++) {

s3c\_gpio\_cfgpin(S5PV210\_GPE1(i), S3C\_GPIO\_SFN(2)); s3c\_gpio\_setpull(S5PV210\_GPE1(i), S3C\_GPIO\_PULL\_NONE); }

/\* CAM B port(b0011) : DATA[0-7] \*/ for (i = 0; i < 8; i++) {

s3c\_gpio\_cfgpin(S5PV210\_GPJ0(i), S3C\_GPIO\_SFN(3));

s3c\_gpio\_setpull(S5PV210\_GPJ0(i), S3C\_GPIO\_PULL\_NONE); }

/\* CAM B port(b0011) : PCLK, VSYNC, HREF, FIELD, CLCKOUT \*/ for (i = 0; i < 5; i++) {

s3c\_gpio\_cfgpin(S5PV210\_GPJ1(i), S3C\_GPIO\_SFN(3));

s3c\_gpio\_setpull(S5PV210\_GPJ1(i), S3C\_GPIO\_PULL\_NONE); } }

4）接口函数

FIMC的主要回调函数如下，实现在fimc\_v4l2.c中：

const struct v4l2\_ioctl\_ops fimc\_v4l2\_ops = { .vidioc\_querycap = fimc\_querycap, .vidioc\_reqbufs = fimc\_reqbufs, .vidioc\_querybuf = fimc\_querybuf, .vidioc\_g\_ctrl = fimc\_g\_ctrl, .vidioc\_s\_ctrl = fimc\_s\_ctrl, .vidioc\_s\_ext\_ctrls = fimc\_s\_ext\_ctrls, .vidioc\_cropcap = fimc\_cropcap, .vidioc\_g\_crop = fimc\_g\_crop, .vidioc\_s\_crop = fimc\_s\_crop, .vidioc\_streamon = fimc\_streamon, .vidioc\_streamoff = fimc\_streamoff, .vidioc\_qbuf = fimc\_qbuf, .vidioc\_dqbuf = fimc\_dqbuf,

.vidioc\_enum\_fmt\_vid\_cap = fimc\_enum\_fmt\_vid\_capture, .vidioc\_g\_fmt\_vid\_cap = fimc\_g\_fmt\_vid\_capture, .vidioc\_s\_fmt\_vid\_cap = fimc\_s\_fmt\_vid\_capture, .vidioc\_try\_fmt\_vid\_cap = fimc\_try\_fmt\_vid\_capture, .vidioc\_enum\_input = fimc\_enum\_input, .vidioc\_g\_input = fimc\_g\_input, .vidioc\_s\_input = fimc\_s\_input, .vidioc\_g\_parm = fimc\_g\_parm, .vidioc\_s\_parm = fimc\_s\_parm, .vidioc\_queryctrl = fimc\_queryctrl, .vidioc\_querymenu = fimc\_querymenu, .vidioc\_g\_fmt\_vid\_out = fimc\_g\_fmt\_vid\_out, .vidioc\_s\_fmt\_vid\_out = fimc\_s\_fmt\_vid\_out, .vidioc\_try\_fmt\_vid\_out = fimc\_try\_fmt\_vid\_out, .vidioc\_g\_fbuf = fimc\_g\_fbuf, .vidioc\_s\_fbuf = fimc\_s\_fbuf,

.vidioc\_try\_fmt\_vid\_overlay = fimc\_try\_fmt\_overlay, .vidioc\_g\_fmt\_vid\_overlay = fimc\_g\_fmt\_vid\_overlay, .vidioc\_s\_fmt\_vid\_overlay = fimc\_s\_fmt\_vid\_overlay, };

5）寄存器操作（fimc\_regs.c）

对于寄存器的操作，实现都在fimc\_regs.c文件中，如

int fimc\_hwset\_camera\_source(struct fimc\_control \*ctrl)

{

cfg |= S3C\_CISRCFMT\_ITU601\_8BIT; cfg |= cam->order422;

if (cam->type == CAM\_TYPE\_ITU)

struct s3c\_platform\_camera \*cam = ctrl->cam; u32 cfg = 0;

cfg |= cam->fmt;

cfg |= S3C\_CISRCFMT\_SOURCEHSIZE(cam->width); cfg |= S3C\_CISRCFMT\_SOURCEVSIZE(cam->height);

return 0; }

int fimc\_hwset\_enable\_irq(struct fimc\_control \*ctrl, int overflow, int level) {

cfg &= ~(S3C\_CIGCTRL\_IRQ\_OVFEN | S3C\_CIGCTRL\_IRQ\_LEVEL); u32 cfg = readl(ctrl->regs + S3C\_CIGCTRL); writel(cfg, ctrl->regs + S3C\_CISRCFMT);

cfg |= S3C\_CIGCTRL\_IRQ\_ENABLE;

if (overflow)

cfg |= S3C\_CIGCTRL\_IRQ\_OVFEN;

if (level)

cfg |= S3C\_CIGCTRL\_IRQ\_LEVEL;

return 0; }

writel(cfg, ctrl->regs + S3C\_CIGCTRL);

3. Sensor驱动

1）简介,

本方案中使用了两个摄像头模组：MT9P111和S5K6AAFX。其中MT9P111是APTINA公司推出的1/4英寸光学格式5M单芯片传感器，用作后摄像头；S5K6AAFX是三星出的1.3M CMOS高清图像传感器，用作前摄像头。

2）参数设置

MT9P111的参数设置

#ifdef MT9P111\_ENABLED

static struct mt9p111\_platform\_data mt9p111\_plat = {

.default\_width = 1024, .default\_height = 600,

.pixelformat = V4L2\_PIX\_FMT\_UYVY, };

static struct i2c\_board\_info mt9p111\_i2c\_info = { I2C\_BOARD\_INFO("MT9P111", 0x3D),//0x7a,0x7b .platform\_data = &mt9p111\_plat, };

static struct s3c\_platform\_camera mt9p111 = { .id = CAMERA\_PAR\_A, .type = CAM\_TYPE\_ITU,

.fmt = ITU\_601\_YCBCR422\_8BIT, .order422 = CAM\_ORDER422\_8BIT\_CBYCRY, .i2c\_busnum = 7,

.info = mt9p111\_i2c\_info,

.pixelformat = V4L2\_PIX\_FMT\_UYVY, .srclk\_name = "xusbxti", .clk\_name = "sclk\_cam0",

.freq = 24000000, .is\_mipi = 0,

.clk\_rate = 24000000, .line\_length = 1920, .width = 1024, .height = 600, .window = { .left = 0, .top = 0,

.width = 1024, .height = 600, },

/\* Polarity \*/

.inv\_pclk = 0, .inv\_vsync = 0, .inv\_href = 0, .inv\_hsync = 0,

.initialized = 0,

.cam\_power = smdkv210\_cam0\_power, }; #endif

S5K6AAFX的参数设置

#ifdef S5K6AAFX\_ENABLED

static struct s5k6aafx\_platform\_data s5k6aafx\_plat = {

.default\_width = 800, .default\_height = 600,

.pixelformat = V4L2\_PIX\_FMT\_YUYV,

.freq = 24000000,

};

.is\_mipi = 0,

static struct i2c\_board\_info s5k6aafx\_i2c\_info = { I2C\_BOARD\_INFO("s5k6aafx", 0x3c), .platform\_data = &s5k6aafx\_plat, };

static struct s3c\_platform\_camera s5k6aafx = { .id = CAMERA\_PAR\_B, .type = CAM\_TYPE\_ITU,

.fmt = ITU\_601\_YCBCR422\_8BIT, .order422 = CAM\_ORDER422\_8BIT\_YCBYCR, .i2c\_busnum = 4,

.info = s5k6aafx\_i2c\_info,

.pixelformat = V4L2\_PIX\_FMT\_YUYV, .srclk\_name = "xusbxti", .clk\_name = "sclk\_cam1", .clk\_rate = 24000000, .line\_length = 1280,

/\* default resol for preview kind of thing \*/

.width = 800, .height = 600, .window = { .left = 0, .top = 0, .width = 800, .height = 600, },

/\* Polarity \*/

.inv\_pclk = 0, .inv\_vsync = 0,

.inv\_href = 0, .inv\_hsync = 0,

.initialized = 0,

.cam\_power = smdkv210\_cam1\_power, }; #endif

3）数据结构

（未注释）

struct v4l2\_subdev {

struct list\_head list; struct module \*owner;

u32 flags;

struct v4l2\_device \*v4l2\_dev; const struct v4l2\_subdev\_ops \*ops; /\* name must be unique \*/

char name[V4L2\_SUBDEV\_NAME\_SIZE];

/\* can be used to group similar subdevs, value is driver-specific \*/

u32 grp\_id;

/\* pointer to private data \*/

void \*priv; }; #endif

4）接口函数

（未注释）

static const struct v4l2\_subdev\_core\_ops mt9p111\_core\_ops = {

.init = mt9p111\_init, /\* initializing API \*/

};

.s\_config = mt9p111\_s\_config, /\* Fetch platform data \*/ .queryctrl = mt9p111\_queryctrl, .querymenu = mt9p111\_querymenu, .g\_ctrl = mt9p111\_g\_ctrl, .s\_ctrl = mt9p111\_s\_ctrl,

static const struct v4l2\_subdev\_video\_ops mt9p111\_video\_ops = { // .s\_crystal\_freq = mt9p111\_s\_crystal\_freq, //

.g\_fmt = mt9p111\_g\_fmt, .s\_fmt = mt9p111\_s\_fmt,

.enum\_framesizes = mt9p111\_enum\_framesizes, .enum\_frameintervals = mt9p111\_enum\_frameintervals,

// .enum\_fmt = mt9p111\_enum\_fmt, };

static const struct v4l2\_subdev\_ops mt9p111\_ops = { }; #endif

.core = mt9p111\_core\_ops, .video = mt9p111\_video\_ops, .try\_fmt = mt9p111\_try\_fmt, .g\_parm = mt9p111\_g\_parm, .s\_parm = mt9p111\_s\_parm, .s\_stream = mt9p111\_s\_stream,

本文档下载自360文档中心，www.360docs.net更多营销,职业规划,工作简历,入党,工作报告,总结,学习资料,学习总结,PPT模板下载,范文等文档下载；转载请保留出处:http://www.360docs.net/doc/info-fdbc8bdf551810a6f52486ed.html