



【公布获奖】2-3月原创博文评选

cxw3506的ChinaUnix博客

是该记录点东西了！

首页 | 博文目录 | 关于我



cxw3506

博客访问： 11460

博文数量： 22

博客积分： 359

博客等级： 一等列兵

技术积分： 200

用户组： 普通用户

注册时间： 2012-04-08 19:51

加关注

短消息

V4L2驱动程序架构

2012-08-02 10:19:44

分类： 嵌入式

1 V4L2简介

video4linux2 (V4L2) 是Linux内核中关于视频设备的内核驱动，它为Linux中视频设备访问提供了通用接口，在Linux系统中，V4L2驱动的Video设备节点路径通常/dev/video/中的videoX

V4L2驱动对用户空间提供字符设备，主设备号为81，对于视频设备，其次设备号为0-63。除此之外，次设备号为64-127的Radio设备，次设备号为192-223的是Teletext设备，次设备号为224-255的是VBI设备

V4L2驱动的Video设备在用户空间通过各种ioctl调用进行控制，并且可以使用mmap进行内存映射

1.1 V4L2驱动主要使用的ioctl

命令值如下所示：

点击 [此处](#) 折叠或打开

```
1. #define VIDIOC_QUERYCAP _IOR('V', 0, struct v4l2_capability) /*查询能力*/
2. #define VIDIO_G_FMT _IOWR('V', 4, struct v4l2_format) /*获得格式*/
3. #define VIDIOC_S_FMT _IOWR('V', 5, struct v4l2_format) /*设置格式*/
4. #define VIDIOC_REQBUFS _IOWR('V', 8, struct v4l2_requestbuffers) /*申请内存*/
```

论坛

加好友

文章分类

全部博文 (22)

linux (13)

未分配的博文 (9)

文章存档

2012年 (22)

我的朋友



81907343

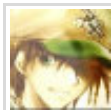
最近访客



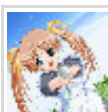
幸福软件



Ubuntu_o



rxlinux



bluefish



xkcp0324



complica

```

5.  #define VIDIOC_G_FBUF _IOW('V', 10, struct v4l2_framebuffer) /*获得Framebuffer*/
6.  #define VIDIOC_S_BUF _IOW('V', 11, struct v4l2_framebuffer) /*设置Framebuffer*/
7.  #define VIDIOC_OVERLAY _IOW('V', 14, int) /*设置Overlay*/
8.  #define VIDIOC_QBUF _IOWR('V', 15, struct v4l2_buffer) /*将内存加入队列*/
9.  #define VIDIOC_DQBUF _IOWR('V', 17, struct v4l2_buffer) /*从队列取出内存*/
10. #define VIDIOC_STREAMON _IOW('V', 18, int) /*开始流*/
11. #define VIDIOC_STREAMOFF _IOW('V', 19, int) /*停止流*/
12. #define VIDIOC_G_CTRL _IOWR('V', 27, struct v4l2_control) /*得到控制*/
13. #define VIDIOC_S_CTRL _IOWR('V', 28, struct v4l2_control) /*设置控制*/

```

1.2 重要结构

头文件 include/linux/videodev2.h

include/media/v4l2-dev.h

V4L2驱动核心实现文件: driver/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

```

1. struct video_device
2. {
3.     const struct v4l2_file_operations *fops;
4.     struct cdev *cdev; //字符设备
5.     struct device *parent; //父设备
6.     struct v4l2_device *v4l2_dev; //父v4l2_device
7.     char name[32]; //名称
8.     int vfl_type; //类型
9.     int minor; //次设备号
10.    /*释放回调*/
11.    void (*release)(struct video_device *vdev);
12.    /*ioctl回调*/
13.    const struct v4l2_ioctl_ops *ioctl_ops;

```



huhuwang



rwlcc



xinyuwux

订阅

推荐博文

- 数据库安全--案例篇
- 一次配置DATAGUARD BROKER 的...
- 简单的命令行词典
- Python 数据结构
- Python 调试

热词专题

- rj45 驱动电路
- android 堆栈信息
- SIGSEGVSegmentation fault
- 哈夫曼编码
- xp and dsdm scrum

友情链接

```
14. }
15. 常用的结构
16. 参见/include/linux/videodev2.h
17. 1)设备能力结构
18. struct v4l2_capability
19. {
20.     __u8 driver[16];//驱动名
21.     __u8 card[32];//例如Hauppauge winTV
22.     __u8 bus_info[32];//PCI总线信息
23.     __u32 version;//内核版本
24.     __u32 capabilities;//设备能力
25.     __u32 reserved[4];
26. };
27. 2)数据格式结构
28. struct v4l2_format
29. {
30.     enum v4l2_buf_type type;//本结构的数据类型
31. };
32. 3)像素格式结构
33. struct v4l2_pix_format
34. {
35.     __u32 width;//宽度
36.     __u32 height;//高度
37. }
38. 4)请求缓冲
39. struct v4l2_requestbuffers
```

```
40. {
41.     __u32    count; //缓存数量
42.     enum v4l2_buf_type type; //数据流类型
43. }
44. 5) 数据流类型包括V4L2_MEMORY_MMAP和V4L2_MEMORY_USERPTR
45. enum v4l2_memory{
46.
47. };
```

点击 [此处](#) 折叠或打开

5) 数据流类型包括V4L2_MEMORY_MMAP和V4L2_MEMORY_USERPTR enum v4l2_memory{ };

2 V4L2驱动注册 2.1 video_register_device

video4linux2驱动程序的注册drivers/media/video

video_register_device函数用来注册一个v4l驱动程序

```
1. int video_register_device(struct video_device *vdev, int type, int nr)
2. {
3.     return __video_register_device(vdev, type, nr, 1);
4. }
```

5. 其中参数type支持的类型如下

```
6. #define VFL_TYPE_GRABBER 0 //视频
7. #define VFL_TYPE_VBI      1 //从视频消隐的时间取得信息的设备
8. #define VFL_TYPE_RADIO    2 //广播
9. #define VFL_TYPE_VTX      3 //视传设备
```

```
10. #define VFL_TYPE_MAX    4//最大值
11. ----->返回调用 __video_register_device()
12. __video_register_device 函数先检查设备类型，接下来
13. 寻找一个可用的子设备号，最后注册相应的字符设备
14. static int __video_register_device(struct video_device *vdev, int type, int nr, int warn_if_nr_in_use)
15. {
16.
17.     switch (type) {
18.         case VFL_TYPE_GRABBER:
19.             minor_offset = 0;
20.             minor_cnt = 64;
21.             break;
22.         case VFL_TYPE_RADIO:
23.             minor_offset = 64;
24.             minor_cnt = 64;
25.             break;
26.         case VFL_TYPE_VTX:
27.             minor_offset = 192;
28.             minor_cnt = 32;
29.             break;
30.         case VFL_TYPE_VBI:
31.             minor_offset = 224;
32.             minor_cnt = 32;
33.             break;
34.         nr = devnode_find(vdev, nr == -1 ? 0 : nr, minor_cnt);
35.     }
```

```
36.     nr = devnode_find(vdev, nr == -1 ? 0 : nr, minor_cnt);
37.     vdev->cdev->ops = &v4l2_fops;
38.     //注册字符设备
39.     ret = cdev_add(vdev->cdev, MKDEV(VIDEO_MAJOR, vdev->minor), 1);
40.     ret = device_register(&vdev->dev);
41.     //注册完毕设备信息存储在video_device数组中
42.     mutex_lock(&videodev_lock);
43.     video_device[vdev->minor] = vdev;
44.     mutex_unlock(&videodev_lock);
45. }

int video_register_device(struct video_device *vdev, int type, int nr)
{
    return __video_register_device(vdev, type, nr, 1);
}
```

其中参数type支持的类型如下

```
#define VFL_TYPE_GRABBER 0//视频
```

```
#define VFL_TYPE_VBI 1//从视频消隐的时间取得信息的设备
```

```
#define VFL_TYPE_RADIO 2 //广播
```

```
#define VFL_TYPE_VTX 3//视传设备
```

```
#define VFL_TYPE_MAX 4//最大值
```

----->返回调用 __video_register_device() __video_register_device 函数先检查设备类型，接下来寻找一个可用的子设备号，

最后注册相应的字符设备

点击 [此处](#) 折叠或打开

2.2 v4l2_fops接口

v4l2_fops为video4linux2设备提供了统一的应用层接口，v4l2_fops定义如下

```
1. static const struct file_operations v4l2_fops = {
2.     .owner = THIS_MODULE,
3.     .read = v4l2_read,
4.     .write = v4l2_write,
5.     .open = v4l2_open,
6.     .get_unmapped_area = v4l2_get_unmapped_area,
7.     .mmap = v4l2_mmap,
8.     .unlocked_ioctl = v4l2_ioctl,
9.     .release = v4l2_release,
10.    .poll = v4l2_poll,
11.    .llseek = no_llseek,
12.
13. };
14. v4l2_fops中的成员函数最终要调用struct video_device->fops中相应的成员
15. struct video_device->fops是具体video4linux2摄像头驱动程序必须实现的接口
16. static ssize_t v4l2_read(struct file *filp, char __user *buf, size_t sz, loff_t *off)
17. {
18.     return vdev->fops->read(filp, buf, sz, off);
19. }
```

2.3 /drivers/media/video/samsung/fimc/s3c_fimc_core.c

驱动探测函数s3c_fimc_probe定义

```
1. static int s3c_fimc_probe(struct platform_device *dev)
```

```
2.  {
3.     ctrl = s3c_fimc_register_controller(pdev);
4.
5.     clk_enable(ctrl->clock); //使能时钟
6.     //注册V4L2驱动
7.     ret = video_register_device(ctrl->vd, VFL_TYPE_GRABBER, ctrl->id);
8. }
9. s3c_fimc_register_controller函数主要用来分配资源与申请中断
10. static struct s3c_fimc_control *s3c_fimc_register_controller(struct platform_device *pdev)
11. {
12.     ctrl->vd = &s3c_fimc_video_device[id];
13.     //申请中断
14.     ctrl->irq = platform_get_irq(pdev, 0);
15.     if(request_irq(ctrl->irq, s3c_fimc_irq, IRQF_DISABLED, ctrl->name, ctrl))
16. };
17. struct video_device s3c_fimc_video_device[S3C_FIMC_MAX_CTRLs] = {
18.     [0] = {
19.         .vfl_type = VID_TYPE_OVERLAY | VID_TYPE_CAPTURE | VID_TYPE_CLIPPING | VID_TYPE_SCALES,
20.         .fops = &s3c_fimc_fops,
21.         .ioctl_ops = &s3c_fimc_v4l2_ops,
22.         .release = s3c_fimc_vdev_release,
23.
24.         .name = "sc3_video0",
25.     },
26. }
```

s3c_fimc_v4l2_ops,是在drivers/media/video/samsung/fimc中实现的v4l2_ioctl_ops, 在用户空间进行ioctl等调用时, 要调用到具体实

现的各个函数指针

3 V4L2 操作

3.1 s3c_fimc_open

```
1. static int s3c_fimc_open(struct file *filp)
2. {
3.     struct s3c_fimc_control *ctrl;
4.     int id, ret;
5.
6.     id =0;
7.     ctrl = &s3c_fimc.ctrl[id];
8.     mutex_lock(&ctrl->lock);
9.     if (atomic_read(&ctrl->in_use)) {
10.         ret = -EBUSY;
11.         goto resource_busy;
12.     } else {
13.         atomic_inc(&ctrl->in_use);
14.         s3c_fimc_reset(ctrl);
15.         filp->private_data = ctrl;
16.     }
17.     mutex_unlock(&ctrl->lock);
18.     return 0;
19. resource_busy:
20.     mutex_unlock(&ctrl->lock);
21.     return ret;
```

```
22. }
23. 用户空间
24. 打开设备文件
25. fd = open(dev_name, O_RDWR | O_NONBLOCK, 0);
    用户空间打开设备文件 fd = open(dev_name, O_RDWR | O_NONBLOCK, 0);
```

3.2 获取设备的capability,查看设备有什么功能

1) 结构体

```
1. struct v4l2_capability cap;
2. ret = ioctl(fd, VIDIOC_QUERYCAP, &cap);
3. /include/linux/videodev2.h
4. struct v4l2_capability {
5.     __u8    driver[16]; /* i.e. "bttv" */
6.     __u8    card[32];   /* i.e. "Hauppauge WinTV" */
7.     __u8    bus_info[32]; /* "PCI:" + pci_name(pci_dev) */
8.     __u32    version;     /* should use KERNEL_VERSION() */
9.     __u32    capabilities; /* Device capabilities */
10.    __u32    reserved[4];
11. };
12. 驱动实现
13.
14. static int s3c_fimc_v4l2_querycap(struct file *filp, void *fh,
15.                                   struct v4l2_capability *cap)
16. {
17.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
18.     strcpy(cap->driver, "Samsung FIMC Driver");
```

```
19.     strcpy(cap->card, ctrl->vd->name, sizeof(cap->card));
20.     sprintf(cap->bus_info, "FIMC AHB-bus");
21.     cap->version = 0;
22.     cap->capabilities = (V4L2_CAP_VIDEO_OVERLAY | \
23.                         V4L2_CAP_VIDEO_CAPTURE | V4L2_CAP_STREAMING);
24.     return 0;
25. }
26. 应用层调用
27. static int video_capability(int fd)
28. {
29.     int ret = 0;
30.     /*****get the device capability*****/
31.     struct v4l2_capability cap;
32.     ret = ioctl(fd, VIDIOC_QUERYCAP, &cap);
33.     if (ret < 0) {
34.         perror("VIDIOC_QUERYCAP failed ");
35.         return ret;
36.     }
37.
38.     printf("\n****Capability informations****\n");
39.     printf("driver:  %s\n", cap.driver);
40.
41.     if (cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)
42.         printf("Capture capability is supported\n");
43.
44.     if (cap.capabilities & V4L2_CAP_STREAMING)
```

```
45.     printf("Streaming capability is supported\n");
46.
47.     if (cap.capabilities & V4L2_CAP_VIDEO_OVERLAY)
48.         printf("Overlay capability is supported\n");
49.
50.     return 0;
51. }
```

3.3 选择视频输入，一个视频设备可以有多个视频输入

```
1. 结构体
2. struct v4l2_input input;
3. int index;
4. 得到INPUT
5. ret = ioctl(fd, VIDIOC_G_INPUT, &index);
6. input.index = index;
7. 列举INPUT
8. ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
9. 设置INPUT
10. ret = ioctl(fd, VIDIOC_S_INPUT, &index);
11.
12. struct v4l2_input {
13.     __u32      index;      /* Which input */
14.     __u8       name[32];    /* Label */
15.     __u32      type;        /* Type of input */
16.     __u32      audioset;     /* Associated audios (bitfield) */
17.     __u32      tuner;       /* Associated tuner */
```

```
18.     v4l2_std_id   std;
19.     __u32         status;
20.     __u32         capabilities;
21.     __u32         reserved[3];
22. };
23.
24. Ioctl: VIDIOC_S_INPUT This IOCTL takes pointer to integer containing index of the input which has to be set. Application will provide the index number as an argument.
25.     0 - Composite input,
26.     1 - S-Video input.
27. 驱动
28. static int s3c_fimc_v4l2_s_input(struct file *filp, void *fh,
29.                                unsigned int i)
30. {
31.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
32.
33.     if (i >= S3C_FIMC_MAX_INPUT_TYPES)
34.         return -EINVAL;
35.
36.     ctrl->v4l2.input = &s3c_fimc_input_types[i];
37.
38.     if (s3c_fimc_input_types[i].type == V4L2_INPUT_TYPE_CAMERA)
39.         ctrl->in_type = PATH_IN_ITU_CAMERA;
40.     else
41.         ctrl->in_type = PATH_IN_DMA;
42.
```

```
43.     return 0;
44. }
45. static struct v4l2_input s3c_fimc_input_types[] = {
46.     {
47.         .index      = 0,
48.         .name        = "External Camera Input",
49.         .type        = V4L2_INPUT_TYPE_CAMERA,
50.         .audioset     = 1,
51.         .tuner        = 0,
52.         .std          = V4L2_STD_PAL_BG | V4L2_STD_NTSC_M,
53.         .status       = 0,
54.     },
55.     {
56.         .index      = 1,
57.         .name        = "Memory Input",
58.         .type        = V4L2_INPUT_TYPE_MEMORY,
59.         .audioset     = 2,
60.         .tuner        = 0,
61.         .std          = V4L2_STD_PAL_BG | V4L2_STD_NTSC_M,
62.         .status       = 0,
63.     }
64. };
65. static int s3c_fimc_v4l2_enum_input(struct file *filp, void *fh,
66.                                     struct v4l2_input *i)
67. {
68.     if (i->index >= S3C_FIMC_MAX_INPUT_TYPES)
```

```
69.         return -EINVAL;
70.
71.     memcpy(i, &s3c_fimc_input_types[i->index], sizeof(struct v4l2_input));
72.
73.     return 0;
74. }
75. 应用
76. static int video_input(int fd)
77. {
78.     /******get and set the VIDIO INPUT*****/
79.     int ret = 0;
80.     struct v4l2_input input;//视频输入信息，对应命令VIDIOC_ENUMINPUT
81.     int index;
82.     index = 0;    //0 - Composite input, 1 - S-Video input.
83.
84.     ret = ioctl (fd, VIDIOC_S_INPUT, &index);
85.     if (ret < 0) {
86.         perror ("VIDIOC_S_INPUT");
87.         return ret;
88.     }
89.
90.     input.index = index;
91.     ret = ioctl (fd, VIDIOC_ENUMINPUT, &input);
92.     if (ret < 0){
93.         perror ("VIDIOC_ENUMINPUT");
94.         return ret;
```

```
95.     }
96.     printf("\n***input informations***\n");
97.     printf("name of the input = %s\n", input.name);
98.     return 0;
99. }
```

3.4 遍历所有视频格式,查询驱动所支持的格式

```
1. 结构
2. struct v4l2_fmtdes fmtdes;
3. ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmtdes);
4. struct v4l2_fmtdesc {
5.     __u32          index;           /* Format number      */
6.     enum v4l2_buf_type  type;       /* buffer type        */
7.     __u32          flags;
8.     __u8          description[32];  /* Description string */
9.     __u32          pixelformat;     /* Format fourcc       */
10.    __u32          reserved[4];
11. };
12. 驱动
13. static int s3c_fimc_v4l2_enum_fmt_vid_cap(struct file *filp, void *fh,
14.                                           struct v4l2_fmtdesc *f)
15. {
16.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
17.     int index = f->index;
18.
19.     if (index >= S3C_FIMC_MAX_CAPTURE_FORMATS)
20.         return -EINVAL;
```



```
21.
22.     memset(f, 0, sizeof(*f));
23.     memcpy(f, ctrl->v4l2.fmtdesc + index, sizeof(*f));
24.
25.     return 0;
26. }
27. #define S3C_FIMC_MAX_CAPTURE_FORMATS    ARRAY_SIZE(s3c_fimc_capture_formats)
28. const static struct v4l2_fmtdesc s3c_fimc_capture_formats[] = {
29.     {
30.         .index      = 0,
31.         .type        = V4L2_BUF_TYPE_VIDEO_CAPTURE,
32.         .flags        = FORMAT_FLAGS_PLANAR,
33.         .description  = "4:2:0, planar, Y-Cb-Cr",
34.         .pixelformat  = V4L2_PIX_FMT_YUV420,
35.     },
36.     {
37.         .index      = 1,
38.         .type        = V4L2_BUF_TYPE_VIDEO_CAPTURE,
39.         .flags        = FORMAT_FLAGS_PLANAR,
40.         .description  = "4:2:2, planar, Y-Cb-Cr",
41.         .pixelformat  = V4L2_PIX_FMT_YUV422P,
42.     },
43.     {
44.
45.         .index      = 2,
46.         .type        = V4L2_BUF_TYPE_VIDEO_CAPTURE,
```

```
47.         .flags      = FORMAT_FLAGS_PACKED,
48.         .description = "4:2:2, packed, YCBYCR",
49.         .pixelformat = V4L2_PIX_FMT_YUYV,
50.     },
51.     {
52.         .index      = 3,
53.         .type       = V4L2_BUF_TYPE_VIDEO_CAPTURE,
54.         .flags      = FORMAT_FLAGS_PACKED,
55.         .description = "4:2:2, packed, CBYCRY",
56.         .pixelformat = V4L2_PIX_FMT_UYVY,
57.     }
58. };
59. const static struct v4l2_fmtdesc s3c_fimc_overlay_formats[] = {
60.     {
61.         .index      = 0,
62.         .type       = V4L2_BUF_TYPE_VIDEO_OVERLAY,
63.         .flags      = FORMAT_FLAGS_PACKED,
64.         .description = "16 bpp RGB, 1e",
65.         .pixelformat = V4L2_PIX_FMT_RGB565,
66.     },
67.     {
68.         .index      = 1,
69.         .type       = V4L2_BUF_TYPE_VIDEO_OVERLAY,
70.         .flags      = FORMAT_FLAGS_PACKED,
71.         .description = "24 bpp RGB, 1e",
72.         .pixelformat = V4L2_PIX_FMT_RGB24,
```

```
73.     },
74. };
75. 应用层
76. static int video_fmtdesc(int fd)
77. {
78.     /*****Format Enumeration*****/
79.     int ret = 0;
80.     struct v4l2_fmtdesc fmtdes;
81.     CLEAR(fmtdes);
82.     fmtdes.index = 0;
83.     fmtdes.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
84.     printf("\n*****vidioc enumeration stream format informations:****\n");
85.     while (1) {
86.
87.         ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmtdes);
88.         if (ret < 0)
89.             break;
90.
91.         printf("{ pixelformat = %c%c%c%c, description = %s }\n",
92.                (fmtdes.pixelformat & 0xFF),
93.                (fmtdes.pixelformat >> 8) & 0xFF,
94.                (fmtdes.pixelformat >> 16) & 0xFF,
95.                (fmtdes.pixelformat >> 24) & 0xFF,
96.                fmtdes.description);
97.
98.         if (fmtdes.type == V4L2_BUF_TYPE_VIDEO_CAPTURE)
```

```
99.         printf("video capture type:\n");
100.     if (fmtdes.pixelformat == V4L2_PIX_FMT_YUYV)
101.         printf("V4L2_PIX_FMT_YUYV\n");
102.     fmtdes.index++;
103. }
104. return 0;
105. }
```

3.5 设置视频捕获格式（重要）

1. 结构体

2. 帧格式包括宽度和高度

```
3. struct v4l2_format fmt;
4. ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
5. struct v4l2_format {
6.     enum v4l2_buf_type type; //数据流类型，必须是V4L2_BUF_TYPE_VIDEO_CAPTURE
7.     union {
8.         struct v4l2_pix_format    pix;    /* V4L2_BUF_TYPE_VIDEO_CAPTURE */
9.         struct v4l2_window        win;    /* V4L2_BUF_TYPE_VIDEO_OVERLAY */
10.        struct v4l2_vbi_format      vbi;    /* V4L2_BUF_TYPE_VBI_CAPTURE */
11.        struct v4l2_sliced_vbi_format sliced; /* V4L2_BUF_TYPE_SLICED_VBI_CAPTURE */
12.        __u8    raw_data[200];             /* user-defined */
13.    } fmt;
14. };
15. struct v4l2_pix_format {
16.     __u32 pixelformat; //视频数据存储类型，例如是YUV4:2:2还是RGB
17. }
18. 驱动
```

```
19. static int s3c_fimc_v4l2_s_fmt_vid_cap(struct file *filp, void *fh,
20.                                     struct v4l2_format *f)
21. {
22.
23.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
24.     ctrl->v4l2.frbuf.fmt = f->fmt.pix;
25.
26.     if (f->fmt.pix.priv == V4L2_FMT_IN)
27.         s3c_fimc_set_input_frame(ctrl, &f->fmt.pix);
28.     else
29.         s3c_fimc_set_output_frame(ctrl, &f->fmt.pix);
30.
31.     return 0;
32. }
33. int s3c_fimc_set_input_frame(struct s3c_fimc_control *ctrl,
34.                             struct v4l2_pix_format *fmt)
35. {
36.     s3c_fimc_set_input_format(ctrl, fmt);
37.
38.     return 0;
39. }
40.
41. static void s3c_fimc_set_input_format(struct s3c_fimc_control *ctrl,
42.                                       struct v4l2_pix_format *fmt)
43. {
44.     struct s3c_fimc_in_frame *frame = &ctrl->in_frame;
```

```
45.  
46.     frame->width = fmt->width;  
47.     frame->height = fmt->height;  
48.  
49.     switch (fmt->pixelformat) {  
50.     case V4L2_PIX_FMT_RGB565:  
51.         frame->format = FORMAT_RGB565;  
52.         frame->planes = 1;  
53.         break;  
54.  
55.     case V4L2_PIX_FMT_RGB24:  
56.         frame->format = FORMAT_RGB888;  
57.         frame->planes = 1;  
58.         break;  
59.  
60.     case V4L2_PIX_FMT_NV12:  
61.         frame->format = FORMAT_YCBCR420;  
62.         frame->planes = 2;  
63.         frame->order_2p = LSB_CBCR;  
64.         break;  
65.  
66.     case V4L2_PIX_FMT_NV21:  
67.         frame->format = FORMAT_YCBCR420;  
68.         frame->planes = 2;  
69.         frame->order_2p = LSB_CRCB;  
70.         break;
```

```
71.  
72.     case V4L2_PIX_FMT_NV12X:  
73.         frame->format = FORMAT_YCBCR420;  
74.         frame->planes = 2;  
75.         frame->order_2p = MSB_CBCR;  
76.         break;  
77.  
78.     case V4L2_PIX_FMT_NV21X:  
79.         frame->format = FORMAT_YCBCR420;  
80.         frame->planes = 2;  
81.         frame->order_2p = MSB_CRCB;  
82.         break;  
83.  
84.     case V4L2_PIX_FMT_YUV420:  
85.         frame->format = FORMAT_YCBCR420;  
86.         frame->planes = 3;  
87.         break;  
88.  
89.     case V4L2_PIX_FMT_YUYV:  
90.         frame->format = FORMAT_YCBCR422;  
91.         frame->planes = 1;  
92.         frame->order_1p = IN_ORDER422_YCBYCR;  
93.         break;  
94.  
95.     case V4L2_PIX_FMT_YVYU:  
96.         frame->format = FORMAT_YCBCR422;
```

```
97.         frame->planes = 1;
98.         frame->order_1p = IN_ORDER422_YCRYCB;
99.         break;
100.
101.     case V4L2_PIX_FMT_UYVY:
102.         frame->format = FORMAT_YCBCR422;
103.         frame->planes = 1;
104.         frame->order_1p = IN_ORDER422_CBYCRY;
105.         break;
106.
107.     case V4L2_PIX_FMT_VYUY:
108.         frame->format = FORMAT_YCBCR422;
109.         frame->planes = 1;
110.         frame->order_1p = IN_ORDER422_CRYCBY;
111.         break;
112.
113.     case V4L2_PIX_FMT_NV16:
114.         frame->format = FORMAT_YCBCR422;
115.         frame->planes = 2;
116.         frame->order_1p = LSB_CBCR;
117.         break;
118.
119.     case V4L2_PIX_FMT_NV61:
120.         frame->format = FORMAT_YCBCR422;
121.         frame->planes = 2;
122.         frame->order_1p = LSB_CRCB;
```



```
123.         break;
124.
125.     case V4L2_PIX_FMT_NV16X:
126.         frame->format = FORMAT_YCBCR422;
127.         frame->planes = 2;
128.         frame->order_1p = MSB_CBCR;
129.         break;
130.
131.     case V4L2_PIX_FMT_NV61X:
132.         frame->format = FORMAT_YCBCR422;
133.         frame->planes = 2;
134.         frame->order_1p = MSB_CRCB;
135.         break;
136.
137.     case V4L2_PIX_FMT_YUV422P:
138.         frame->format = FORMAT_YCBCR422;
139.         frame->planes = 3;
140.         break;
141.     }
142. }
143. 应用层
144. static int video_setfmt(int fd)
145. {
146.     /*****set Stream data format*****/
147.     int ret = 0;
148.     struct v4l2_format fmt;
```

```

149.     CLEAR(fmt);
150.     fmt.type          =   V4L2_BUF_TYPE_VIDEO_CAPTURE;
151.     fmt.fmt.pix.width  =   640;
152.     fmt.fmt.pix.height =   480;
153.     fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;//for PAL
154.     fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
155.
156.     ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
157.     if (ret < 0) {
158.         perror("VIDIOC_S_FMT");
159.         return ret;
160.     }
161.
162.     return 0;
163. }

```

3.6 视频格式查询

在v4l2中，有两种查询视频格式的方法，一个是遍历所有视频格式的

一个是查询出一种格式的

/*查询出一种格式*/

```
ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
```

/*遍历所有视频格式,查询驱动所支持的格式*/

VIDIOC_ENUM_FMT

1. 驱动

```
2. static int s3c_fimc_v4l2_g_fmt_vid_cap(struct file *filp, void *fh,
```

```
3.         struct v4l2_format *f)
```

```
4.  {
5.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
6.     int size = sizeof(struct v4l2_pix_format);
7.
8.     memset(&f->fmt.pix, 0, size);
9.     memcpy(&f->fmt.pix, &(ctrl->v4l2.frbuf.fmt), size);
10.
11.     return 0;
12. }
13. 应用
14. static int video_getfmt(int fd)
15. {
16.     /*****get Stream data format*****/
17.     int ret= 0;
18.     struct v4l2_format fmt;
19.     CLEAR(fmt);
20.     fmt.type    =  V4L2_BUF_TYPE_VIDEO_CAPTURE;
21.     ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
22.     if (ret < 0) {
23.         perror("VIDIOC_G_FMT");
24.         return ret;
25.     }
26.     printf("/n*****vidioc get stream format informations:****\n");
27.     if (fmt.fmt.pix.pixelformat == V4L2_PIX_FMT_YUYV)
28.         printf("8-bit YUYV pixel format\n");
29.         printf("Size of the buffer = %d\n", fmt.fmt.pix.sizeimage);
```

```
30.     printf("Line offset = %d\n", fmt.fmt.pix.bytesperline);
31.     if (fmt.fmt.pix.field == V4L2_FIELD_INTERLACED)
32.         printf("Storate format is interlaced frame format\n");
33.
34.     return 0;
35. }
```

3.7 向驱动申请帧缓冲，内存，一般不超过5个,帧缓冲管理

```
1. 结构体
2. struct v4l2_requestbuffers req;
3. ret = ioctl(fd, VIDIOC_REQBUFS, &req);
4. ret = ioctl(fd, VIDIOC_QUERYBUF, &buf); //读取缓存
5.
6. struct v4l2_requestbuffers {
7.     __u32          count;
8.     enum v4l2_buf_type    type;
9.     enum v4l2_memory     memory;
10.    __u32          reserved[2];
11. };
12.
13. struct v4l2_buffer {
14.     __u32          index;
15.     enum v4l2_buf_type    type;
16.     __u32          bytesused;
17.     __u32          flags;
18.     enum v4l2_field     field;
19.     struct timeval     timestamp;
```

```
20.     struct v4l2_timecode    timecode;
21.     __u32                    sequence;
22.
23.     /* memory location */
24.     enum v4l2_memory          memory;
25.     union {
26.         __u32                offset;
27.         unsigned long        userptr;
28.     } m;
29.     __u32                    length;
30.     __u32                    input;
31.     __u32                    reserved;
32. };
```

33. 使用VIDIOC_REQBUFS 我们获取了req.count个缓存，下一步通过
34. 调用VIDIOC_QUERYBUF 命令来获取这些缓存的地址，然后使用
35. mmap函数转换成应用程序中的绝对地址，最后把这些缓存放入
36. 缓存队列。

37. The main steps that the application must perform for buffer allocation are:

38. Allocating Memory

39. Getting Physical Address

40. Mapping Kernel Space Address to User Space

41. 驱动支持

42.

```
43. static int s3c_fimc_v4l2_reqbufs(struct file *filp, void *fh,
44.                                   struct v4l2_requestbuffers *b)
45. {
```

```
46.     if (b->memory != V4L2_MEMORY_MMAP) {
47.         err("V4L2_MEMORY_MMAP is only supported\n");
48.         return -EINVAL;
49.     }
50.
51.     /* control user input */
52.     if (b->count > 4)
53.         b->count = 4;
54.     else if (b->count < 1)
55.         b->count = 1;
56.
57.     return 0;
58. }
59. static int s3c_fimc_v4l2_querybuf(struct file *filp, void *fh,
60.                                   struct v4l2_buffer *b)
61. {
62.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
63.
64.     if (b->type != V4L2_BUF_TYPE_VIDEO_OVERLAY && \
65.         b->type != V4L2_BUF_TYPE_VIDEO_CAPTURE)
66.         return -EINVAL;
67.
68.     if (b->memory != V4L2_MEMORY_MMAP)
69.         return -EINVAL;
70.
71.     b->length = ctrl->out_frame.buf_size;
```

```
72.
73.     /*
74.      * NOTE: we use the m.offset as an index for multiple frames out.
75.      * Because all frames are not contiguous, we cannot use it as
76.      * original purpose.
77.      * The index value used to find out which frame user wants to mmap.
78.      */
79.     b->m.offset = b->index * PAGE_SIZE;
80.
81.     return 0;
82. }
83. static int s3c_fimc_v4l2_qbuf(struct file *filp, void *fh,
84.                               struct v4l2_buffer *b)
85. {
86.     return 0;
87. }
88. 应用层
89. static int video_mmap(int fd)
90. {
91.     /*****step 1****requestbuffers Allocating Memory *****/
92.     int ret = 0;
93.     struct v4l2_requestbuffers req;
94.     CLEAR(req);
95.     req.count    = 4;
96.     req.type     = V4L2_BUF_TYPE_VIDEO_CAPTURE;
97.     req.memory   = V4L2_MEMORY_MMAP;
```

```
98.
99.     ret = ioctl(fd, VIDIOC_REQBUFS, &req);
100.     if (ret < 0) {
101.         perror("VIDIOC_REQBUFS");
102.         return ret;
103.     }
104.
105.     if (req.count < 2)
106.         printf("insufficient buffer memory\n");
107.         printf("Number of buffers allocated = %d\n", req.count);
108.
109.     /*****step 2****Getting Physical Address *****/
110.     buffers = calloc(req.count, sizeof(*buffers));
111.     for (n_buffers = 0; n_buffers < req.count; ++n_buffers)
112.     {
113.         struct v4l2_buffer buf; //驱动中的一帧
114.         CLEAR(buf);
115.         buf.type    = V4L2_BUF_TYPE_VIDEO_CAPTURE;
116.         buf.memory   = V4L2_MEMORY_MMAP;
117.         buf.index    = n_buffers;
118.
119.         ret = ioctl(fd, VIDIOC_QUERYBUF, &buf);
120.         if (ret < 0) {
121.             perror("VIDIOC_QUERYBUF");
122.             return ret;
123.         }
```



```
124.
125.  /*****step 3****Mapping Kernel Space Address to User Space*****/
126.      buffers[n_buffers].length = buf.length;
127.      buffers[n_buffers].start =
128.      mmap(NULL,
129.          buf.length,
130.          PROT_READ | PROT_WRITE,
131.          MAP_SHARED,
132.          fd,
133.          buf.m.offset);
134.
135.      //if (MAP_FAILED == buffers[n_buffers].start)
136.      //perror("mmap failed \n");
137.  }
138.
139.  /*****requestbuffers in queue*****/
140.  for (i = 0; i < n_buffers; ++i) {
141.      struct v4l2_buffer buf;
142.      CLEAR(buf);
143.
144.      buf.type    = V4L2_BUF_TYPE_VIDEO_CAPTURE;
145.      buf.memory  = V4L2_MEMORY_MMAP;
146.      buf.index = i;
147.
148.      ret = ioctl(fd, VIDIOC_QBUF, &buf); //申请的缓冲进入队列
149.      if (ret < 0) {
```

```
150.         perror("VIDIOC_QBUF");
151.         return ret;
152.     }
153. }
154.
155.     return 0;
156. }
```

3.8 开始捕捉图像数据(重要)

1. <PRE class=csharp name="code">

结构体

<PRE class=csharp name="code">enum v4l2_buf_type type; //开始捕捉图像数据

```
2.     type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
3.     ret = ioctl(fd, VIDIOC_STREAMON, &type);
4.
5.     enum v4l2_buf_type {
6.         V4L2_BUF_TYPE_VIDEO_CAPTURE      = 1,
7.         V4L2_BUF_TYPE_VIDEO_OUTPUT       = 2,
8.         V4L2_BUF_TYPE_VIDEO_OVERLAY      = 3,
9.         V4L2_BUF_TYPE_VBI_CAPTURE        = 4,
10.        V4L2_BUF_TYPE_VBI_OUTPUT           = 5,
11.        V4L2_BUF_TYPE_SLICED_VBI_CAPTURE  = 6,
12.        V4L2_BUF_TYPE_SLICED_VBI_OUTPUT   = 7,
13.    #if 1
14.        /* Experimental */
15.        V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY = 8,
16.    #endif
```

```
17.     V4L2_BUF_TYPE_PRIVATE           = 0x80,
18. };
19.
20. 驱动
21.
22. static int s3c_fimc_v4l2_streamon(struct file *filp, void *fh,
23.                                   enum v4l2_buf_type i)
24. {
25.     struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
26.     if (i != V4L2_BUF_TYPE_VIDEO_CAPTURE)
27.         return -EINVAL;
28.     printk("s3c_fimc_v4l2_streamon is called\n");
29.     if (ctrl->in_type != PATH_IN_DMA)
30.         s3c_fimc_init_camera(ctrl);
31.
32.     ctrl->out_frame.skip_frames = 0;
33.     FSET_CAPTURE(ctrl);
34.     FSET_IRQ_NORMAL(ctrl);
35.     s3c_fimc_start_dma(ctrl);
36.
37.     return 0;
38. }
39. 硬件控制寄存器的配置
40. 应用层
41. static int video_streamon(int fd)
42. {
```

```
43.     int ret = 0;
44.
45.     /*****start stream on*****/
46.
47.     enum v4l2_buf_type types;//开始捕捉图像数据
48.     types = V4L2_BUF_TYPE_VIDEO_CAPTURE;
49.     ret = ioctl(fd, VIDIOC_STREAMON, &types);
50.     if (ret < 0) {
51.         perror("VIDIOC_STREAMON");
52.         return ret;
53.     }
54.
55.     return 0;
56. }
```

阅读(4086) | 评论(0) | 转发(8) |

[上一篇：嵌入式系统启动例程](#)

0

[下一篇：深入分析 Linux 内核链表](#)

相关热门文章

[Android Multimedia Framework...](#)

[linux守护进程的几个关键地方...](#)

[简单字符设备驱动程序...](#)

[Linux设备驱动之I2C架构分析...](#)

[stagefright与opencore对比](#)

[嵌入式Memo 15 SDRAM On May 6...](#)

[自己学着写lcd入门级驱动程序...](#)

[嵌入式Linux之我行——u-boot...](#)

[自己学着写lcd入门级驱动程序...](#)

| | | |
|-------------------|---------------------|------------------------------|
| 网站制作 | 嵌入式Linux之我行——内核、... | Linux2.6.35下s3c64xx/spi子系... |
| 特别是云计算和大数据技术的快... | android的logcat详细用法 | 编译vlc+live555,支持rtsp流播... |

给主人留下些什么吧！~~

评论热议

请登录后评论。
[登录](#) [注册](#)

[关于我们](#) | [关于IT168](#) | [联系方式](#) | [广告合作](#) | [法律声明](#) | [免费注册](#)

Copyright 2001-2010 ChinaUnix.net All Rights Reserved 北京皓辰网域网络信息技术有限公司. 版权所有

感谢所有关心和支持过ChinaUnix的朋友们

京ICP证041476号 京ICP证060528号