实验 20：视频采集播放程序编写实验@

一、实验目的

1. 理解视频采集的原理；
2. 了解摄像头采集的方式。

二、实验内容

1. 掌握视频采集模块的实现方式；
2. 掌握视频采集的方法。

三、实验设备

1. 硬件：PC 机，基于 ARM9 系统教学实验系统，网线，串口线，SD 卡；
2. 软件：putty 软件；
3. 环境：ubuntu 系统版本 12.04，内核版本 kernel-for-mceb ，文件系统 filesys_test，内核文件 uImage_zh.zh ,应用层 encode 源码，DMAI 库。

四、预备知识

1. C 语言的基础知识。
2. 软件调试的基础知识和方法。
3. Linux 基本操作。
4. Linux 应用程序的编写。
5.了解 DM365 设备 dvsdk 的使用

五、实验说明

1.概述

TI 公司的基于 DaVinci 技术的 TMS320DM365 芯片，集成了一颗 ARM926EJ-S 内核，一个图像处理子系统（VPSS），一个 H.264 高清编码器协处理器 HDVICP 和一个 MPEG-4/JPEG 高清编码器协处理器 MJCP， 支持多格式编解码，特别适合用于图像处理。

TMS320DM365 芯片上提供了一个视频处理子系统(VPSS)，用于视频数据的实时采集、播放等功能。VPSS 内部集成了一个视频处理前端模块(VPFE)和一个视频处理后端模块(VPBE)，VPFE 用来控制接入的外部图像采集设备，如图像传感器，视频解码器等。VPBE 则用来控制接入的显示设备，如标清的模拟电视显示器，数字的 LCD 液晶显示屏等。

整个视频采集系统终端以 TMS320DM365 芯片作为处理器芯片，基本模块有 DDR2 SDRAM、NANDFLASH、网口、串口，以及负责进行信令和视频数据传输的 3G 模块等。视频采集部分主要由 DM365 的 VPFE（视频处理前端），一个多路转换器和两个视频数据采集芯片：分别是豪威科技的 OV5640 和 TI 公司的 TVP5151 组成。摄像头硬件切换由 DM365 的 GPIO 口对多路转换器进行控制，选择输入到 VPFE 的数据源，应用程序通过 V4L2 接口和 DMAI 接口获取 VPFE 驱动中采集的视频数据。

视频数据的采集是系统的数据源获取部分，其过程是通过 CCD/CMOS 摄像头采集模拟视频信号并转化为数字信号。视频采集是一个从底层到上层的分层实现的过程。

实现的功能

将摄像头捕捉到的模拟画面进行处理转化成数字信号并通过 DM365 上的视频处理子系统 VPSS 完成视频信号的采集，流程如图 1 所示。

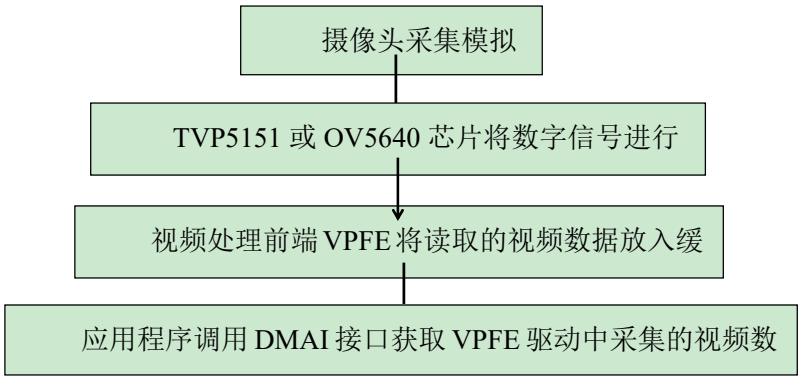


图 1　功能实现流程图

基本原理

V4L2（Video4Linux2）是 LINUX 内核中关于视频数据处理的驱动框架，为上层应用程序访问底层的视频设备提供了统一的接口。V4L2 支持各种视频输入输出设备，当视频设备连结到 LINUX 主机上时，视频处理驱动程序会在内核上注册一个主设备号为 81 的字符设备用于标识该硬件，然后内核会利用主设备号将该视频设备的驱动程序与视频处理驱动程序建立关联，同时加载设备驱动程序的各功能函数，并为其分配次设备号，使该设备可以正常工作。V4L2 架构如图 2 所示，用户空间中的应用程序通过调用 V4L2 接口来访问内核空间中的设备驱动，用户空间与内核空间内存间的映射工作由 V4L2 接口来完成，设备的硬件在最下层，通过各种数据总线与设备驱动建立联系。

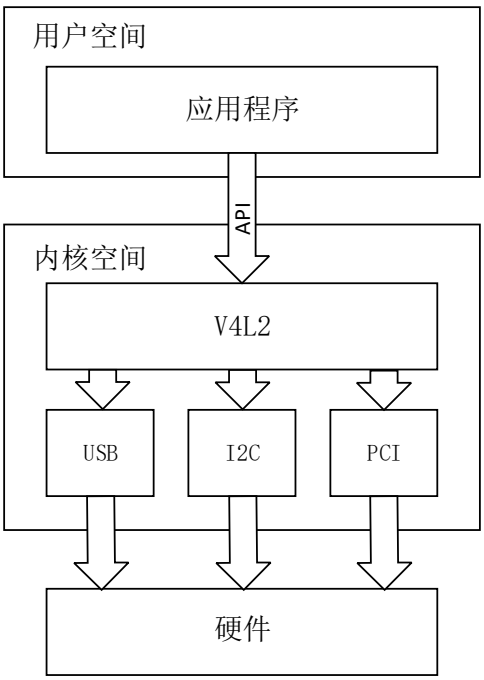


图 2　V4L2 框架

V4L2 框架中视频数据的具体采集流程如下所示：

1. 打开内核设备驱动节点 dev/video0。int fd=open(“dev/video0”,O_RDWR)；
2. 获取内核设备的 capability，查询内核驱动能实现的功能。VIDIOC_QUERYCAP，struct v4l2_capability
3. 设置视频输入，内核驱动支持多个设备同时采集视频数据。VIDIOC_S_INPUT，struct v4l2_input
4. 设置视频数据的分辨率和帧格式，如 D1、CIF，帧格式包括图像的长宽和像素排列等。VIDIOC_S_STD，VIDIOC_S_FMT，struct v4l2_std_id，struct v4l2_format
5. 向内核驱动申请缓存，用于存放采集上来的视频数据。struct v4l2_requestbuffers
6. 利用 mmap()函数将内核空间申请的缓存映射到用户空间，这样应用层程序就可以对视频数据直接进行编解码工作。
7. 将申请到的缓存添加到缓存队列中。VIDIOC_QBUF，struct v4l2_buffer
8. 开始视频的采集。VIDIOC_STREAMON
9. 从缓存队列中取得视频原始数据。VIDIOC_DQBUF
10. 将缓存重新放回缓存队列，进行循环采集。VIDIOC_QBUF
11. 停止视频的采集。VIDIOC_STREAMOFF
12. 关闭视频设备。close(fd)
13. 解除内存映射。munmap( )

打开视频设备后，可以设置该视频设备的属性，例如裁剪、缩放等。这一步是可选的。在 Linux 编程中，一般使用 ioctl 函数来对设备的 I/O 通道进行管理：

```
int ioctl(int fd, int cmd,…)
```

fd：设备的 ID，例如刚才用 open 函数打开视频通道后返回的设备 ID；

cmd：具体的命令标志符。至于后面的省略号，那是一些补充参数，一般最多一个，这个参数的有无和 cmd 的意义相关。

在进行 V4L2 开发中，一般会用到以下的命令标志符：

VIDIOC_REQBUFS：分配内存
VIDIOC_QUERYBUF：把 VIDIOC_REQBUFS 中分配的数据缓存转换成物理地址
VIDIOC_QUERYCAP：查询驱动功能
VIDIOC_ENUM_FMT：获取当前驱动支持的视频格式
VIDIOC_S_FMT：设置当前驱动的帧捕获格式
VIDIOC_G_FMT：读取当前驱动的帧捕获格式
VIDIOC_TRY_FMT：验证当前驱动的显示格式
VIDIOC_CROPCAP：查询驱动的修剪能力
VIDIOC_S_CROP：设置视频信号的边框
VIDIOC_G_CROP：读取视频信号的边框
VIDIOC_QBUF：把数据从缓存中读取出来
VIDIOC_DQBUF：把数据放回缓存队列
VIDIOC_STREAMON：开始视频显示函数
VIDIOC_STREAMOFF：结束视频显示函数
VIDIOC_QUERYSTD：检查当前视频设备支持的标准，例如 PAL 或 NTSC。

硬件平台架构如下：

以 TI 公司 TMS320DM365 为核心的硬件平台能够实现视频采集功能，整个视频采集系统技术以 TMS320DM365 芯片作为处理器芯片，基本模块有 DDR2 SDRAM、NANDFLASH、网口、串口，以及负责进行信令和视频数据传输的 3G 模块等。视频采集部分主要由 DM365 的 VPFE（视频处理前端），一个多路转换器和两个视频数据采集芯片：OV5640 和 TVP5151 组成。摄像头硬件切换由 DM365 的 GPIO 口对多路转换器进行控制，选择输入到 VPFE 的数据源。摄像头采集上来的视频原始数据经由 VPFE 的图像管道存放到 SDRAM 中，供 DM365 的 VPBE（视频处理后端）使用，它可以将视频原始数据经过处理，直接输出到设备的 LCD 液晶显示屏上进行实时播放，总体框架图如图 3 所示。

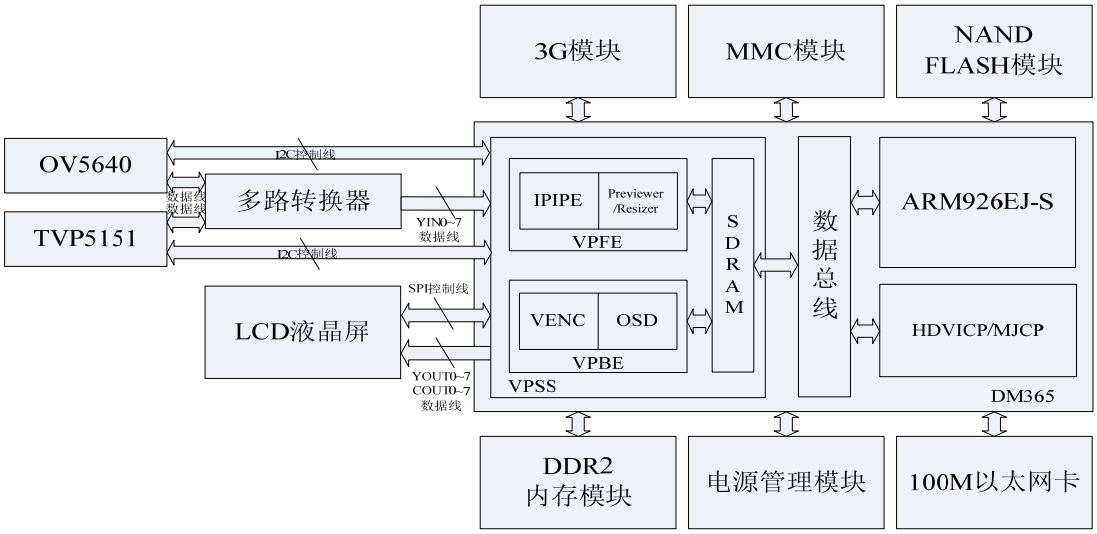


图 3　硬件框架

4.1 摄像头模块

摄像头的核心成像部件有两种：一种是广泛使用的 CCD（电荷藕合）元件，如 TVP5151；另一种是 CMOS（互补金属氧化物导体）器件，如 OV6540。

电荷藕合器件图像传感器 CCD（Charge Coupled Device），它使用一种高感光度的半导体材料制成，能把光线转变成电荷，通过模数转换器芯片转换成数字信号，数字信号经过压缩以后由相机内部的闪速存储器或内置硬盘卡保存，因而可以轻而易举地把数据传输给计算机，并借助于计算机的处理手段，根据需要和想象来修改图像。CCD 由许多感光单位组成，通常以百万像素为单位。当 CCD 表面受到光线照射时，每个感光单位会将电荷反映在组件上，所有的感光单位所产生的信号加在一起，就构成了一幅完整的画面。

互补性氧化金属半导体 CMOS（Complementary Metal-OxideSemiconductor）和 CCD 一样同为在数码相机中可记录光线变化的半导体。CMOS 的制造技术和一般计算机芯片没什么差别，主要是利用硅和锗这两种元素所做成的半导体，使其在 CMOS 上共存着带 N（带–电）和 P（带+电）级的半导体，这两个互补效应所产生的电流即可被处理芯片纪录和解读成影像。然而，CMOS 的缺点就是太容易出现杂点, 这主要是因为早期的设计使 CMOS 在处理快速变化的影像时，由于电流变化过于频繁而会产生过热的现象。

TMS320DM365 核心处理器芯片内置的 DSP 协处理器负责视频数据的 H.264 压缩编码。但是在接入核心处理器端的数据电压要求 3.3v，即高电平的参考电压为 3.3v，但是 OV5640 摄像头端的高电平电压只有 1.8v，若直接接入 DM365 经行处理，将造成数字数据解码错误，故要在输入端经行一个电压转化，将 1.8v 的数据电压转换成 3.3v，OV5640 连接图如图 4 所示。

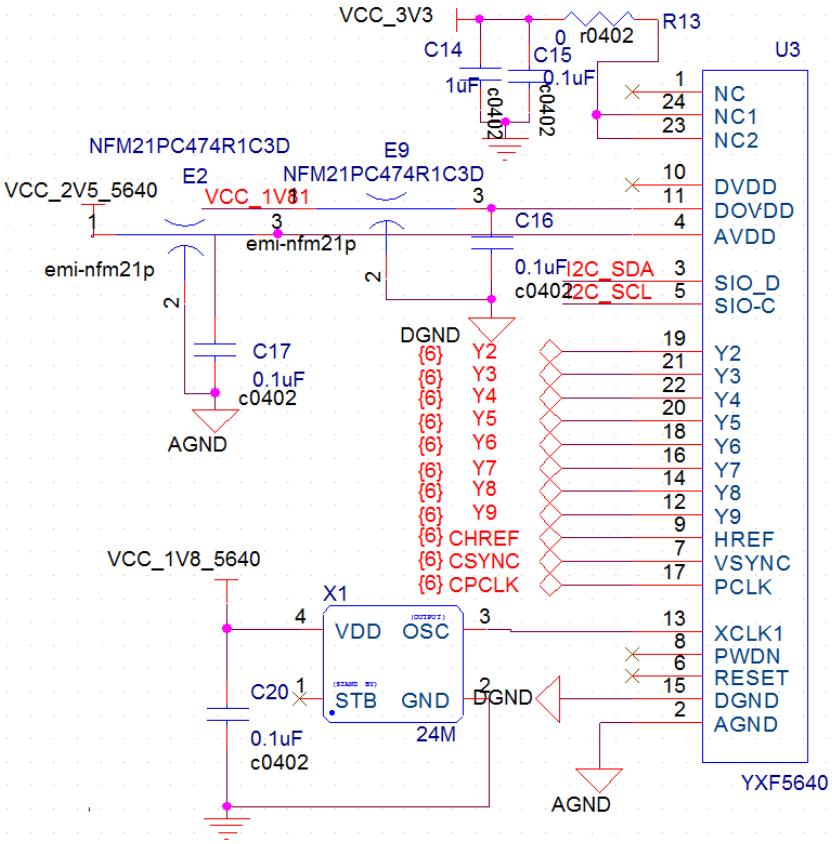


图 4　OV5640 接口图

TVP5150 系列是一颗使用简易，超低功耗，封装极小的数字视频解码器。使用单一 14.31818MHz 时钟就可以实现 PAL/NTSC/SECAM 各种制式的解码，输出 8-bit ITU-R BT.656 数据，也可输出分离同步。MCU 通过标准 I2C 接口控制 TVP5150 的诸多参数，比如色调，对比度，亮度，饱和度和锐度等等。TVP5150 内部的 VBI 处理器可以分离解析出 VBI（Vertical Blanking Interval）里面的 teletext,closed caption 等等信息。

TVP5151 是 TVP5150AM1 的升级版本，其将 TVP5150AM1 的最新补丁固化在内部的 program ROM，并扩大了内部 RAM 的空间。在硬件上唯一的改动就是时钟的输入频率，为单 27MHz。其硬件和寄存器和 TVP5150AM1 完全兼容。所以，即使我们用的芯片是 tvp5151，但在名字上，我们仍然用的是 tvp5150，以便跟内核的名字一致，不需要多次修改，防止名字更改时有漏洞。

当系统将模拟摄像头作为视频采集端接入时，则需要先对模拟摄像头采集的模拟视频数据进行 A/D 转换，转换成数字信号以后再传给 TMS320DM365 经行编解码。本实验箱选择了 TVP5151 这款，TVP5151 芯片支持自动调节对比度，而且功耗低。电路图设计如图 5。模拟摄像头将采集到的一路视频信号接入 TVP5151 芯片中，芯片将根据 I2C 控制总线中的控制信息对视频进行相应的视频编码：视频格式，分辨率等参数。同时上文中提到的视频信号选择芯片 S0S1S2 要配置成 010，才能将数据传给 TMS320DM365，TVP5151 的连接图如图 5 所示。

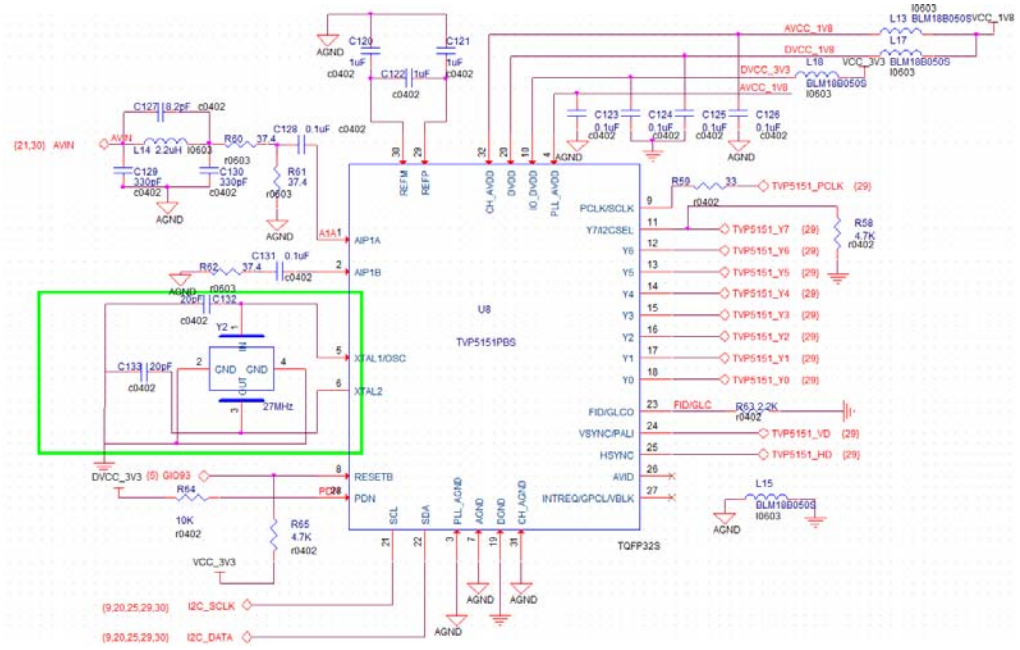


图 5　TVP5151 连接图

5 软件系统架构

5.1 视频采集驱动

视频采集驱动程序主要由 DM365 视频处理前端（VPFE）驱动和视频解码芯片驱动（TVP5151 和 OV5640）两部分组成。其中 VPFE 支持各种图像传感器和视频解码器，它既可以接收 RGB 数据，也可以支持 YUV 格式的视频数据；OV5640 和 TVP5151 的驱动主要实现两个功能，一是将设备注册到 I2C 总线上，还有就是将摄像头驱动注册到内核 VPFE 驱动上，供采集设备调用。TVP5151 和 OV5640 的驱动基本一致，以 OV5640 为例，其驱动是通过 I2C 总线进行寄存器配置的，所以驱动首先要做的就是将设备注册到 I2C 总线上。

除了实现 I2C 注册的功能外，OV5640 驱动还有一个很重要的任务就是在内核的 VPFE 中注册一个设备对应的 decoder，供具体的视频采集程序调用。通过这个 decoder，采集程序可以访问到驱动程序。

VPFE 采集驱动通过标准的 V4L2 接口将底层硬件的功能暴露给了上层应用程序，当 VPFE 驱动注册到内核中时，会产生一个设备节点/dev/Video0，应用程序可以通过对/dev/Video0 进行一系列操作来调用 VPFE 驱动来实现相关的采集功能，VPFE 硬件模块支持以下两种不同的数据管道，如图 6 所示：

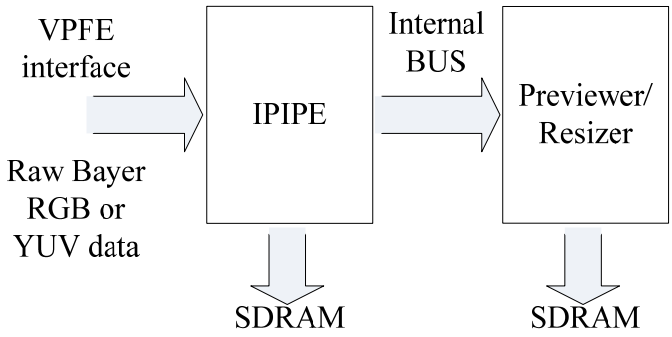


图 6　VPFE 硬件模块数据管道图

1) 输入接口通过 IPIPE 直接接入到 SDRAM，所有型号的 SoC 都支持这个通道。

2) 输入接口通过 IPIPE 接入到 Resizer 图像处理单元，并最终从 Resizer 输出两种不同分辨率大小的视频数据到 SDRAM，从 RSZ-A 输出的是 D1 格式分辨率为 720*576 的图像，后续编码后会存放在 SD 卡中，从 RSZ-B 输出的是 CIF 格式分辨率为 352*288 的图像，后续用于编码网络上传。

为了实现上传和本地存储功能，我们通过 VPFE 驱动将 VPFE 硬件模块配置成第二种数据流通道的采集方式。

5.1.1 OV5640 驱动主要函数及结构体

函数：static int ov5640_i2c_init(void);
功能：初始化 ov5640
参数：无

函数：static int ov5640_i2c_probe_adapter(struct i2c_adapter *adap);
功能：ov5640 设备探测函数
参数：i2c 设备适配结构体指针

函数：static int ov5640_i2c_detach_client(struct i2c_client *client);
功能：主要功能卸载驱动，释放资源
参数：i2c 客户结构体指针

函数：static int ov5640_initialize(void *dec, int flag);
功能：ov5640 初始化
参数：第一个设备属性指针，第二个为标识

函数：static int ov5640_write_regs(struct i2c_client *client, const struct ov5640_reg

reglist[]);

功能：ov5640 写寄存器

参数：第一个 i2c 上客户结构体指针，第二个 ov5640 结构体数组

结构体 static struct decoder_device ov5640_dev 设备参数结构体

```
static struct decoder_device ov5640_dev{
  .name = "OV5640",
        .if_type = INTERFACE_TYPE_YCBCR_SYNC_8,
        .channel_id = 0,
        .capabilities =V4L2_CAP_SLICED_VBI_CAPTURE | V4L2_CAP_VBI_CAPTURE,
        .initialize = ov5640_initialize,
        .std_ops = &standards_ops,
        .ctrl_ops = &controls_ops,
        .input_ops = &chan0_inputs_ops,
        .fmt_ops = &formats_ops,
        .params_ops = &params_ops,
        .deinitialize = ov5640_deinitialize,
        .get_sliced_vbi_cap = NULL,
        .read_vbi_data = NULL
};
```

结构体 static struct ov5640_config ov5640_configuration 主要是 ov5640 的配置

```
static struct ov5640_config ov5640_configuration[OV5640_MAX_CHANNELS] = {

        {
              .no_of_inputs = 1,
              .input[0] = {
                    .input_type = 0x05,
                    .lock_mask = 0x0E,
                    .input_info = {
                         .name = "COMPONENT",//ov5640_cb_720p_vga_30
                          /* .name = "COMPOSITE", */
                          .index = 0,
                          .type = V4L2_INPUT_TYPE_CAMERA,
                          .std = V4L2_STD_OV5640_ALL
                    },
              .no_of_standard = OV5640_MAX_NO_STANDARDS,
              .standard = (struct v4l2_standard *)ov5640_standards,
              .def_std = VPFE_STD_AUTO,
```

```
            .mode = (enum ov5640_mode(*)[])&ov5640_modes,
            .no_of_controls = OV5640_MAX_NO_CONTROLS,
            .controls = (struct ov5640_control_info *)&ov5640_control_information
            },
            .sliced_cap = {
            .service_set = (V4L2_SLICED_CAPTION_525 ，
                            V4L2_SLICED_WSS_625 ，
                            V4L2_SLICED_CGMS_525),
            },
            .num_services = 0
        }
};
```

5.1.2 TVP5151 驱动主要结构体

```
struct tvp5150 {
        struct v4l2_subdev sd;
      v4l2_std_id norm;      /* Current set standard */
        u32 input;
        u32 output;
        int enable;
        int bright;
        int contrast;
        int hue;
        int sat;
};

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;
};
```

5.1.3 VPFE 主要函数及结构体

函数：vpfe_init(void);

功能：vpfe_init（位于内核 davinci_vpfe.c)驱动模块中主要完成的内容就是完成 video_device 的注册

参数：无

函数：imp_get_hw_if(void)

功能：初始化 imp_hw_if 函数接口，将 vpfe 绑定 imp 接口函数组

参数：无

函数：driver_register(&vpfe_driver)

功能：注册 vpfe 驱动

参数：vpfe_driver

函数：vpfe_probe(struct device *device)

功能：设备和驱动匹配后调用该函数注册驱动

参数：device 设备结构体

函数：vpfe_isr(int irq, void *dev_id)

功能：vpfe 对视频信号的中断处理函数

参数：第一个参数 irq 为中断号，第二个为设备号

函数：vpif_register_decoder(&ov5640_dev[i])

功能：提供给解码芯片驱动的注册函数

参数：ov5640 设备

结构体 static struct video_device vpfe_video_template 模板结构体

```
static struct video_device vpfe_video_template = {
        .name = "vpfe",
        .type = VID_TYPE_CAPTURE,
        .hardware = 0,
        .fops = &vpfe_fops,
        .minor = -1,
};
```

结构体 static struct file_operations vpfe_fops 操作结构体

```
static struct file_operations vpfe_fops = {
        .owner = THIS_MODULE,
```

```
    .open = vpfe_open,
    .release = vpfe_release,
    .ioctl = vpfe_ioctl,
    .mmap = vpfe_mmap,
    .poll = vpfe_poll
};
```

5.2 应用程序设计

视频的采集工作主要是由 ENCODE 进程的 Capture 线程进行的。Capture 线程通过调用 DMAI 接口函数与底层驱动交互，获取摄像头驱动采集上来的原始数据，利用内核模块进行裁剪、缩放，最终配置成两路数据，通过管道将两路数据传给视频编码线程进行编码,一路 CIF:352x288 大小的数据，一路 D1:720x576 大小的数据。同时将其中一路 DI（720*576）格式的数据裁剪成 VGA（640*480）的数据，放入视频播放设备缓存进行实时播放，设计的 capture 线程如图 7 所示。

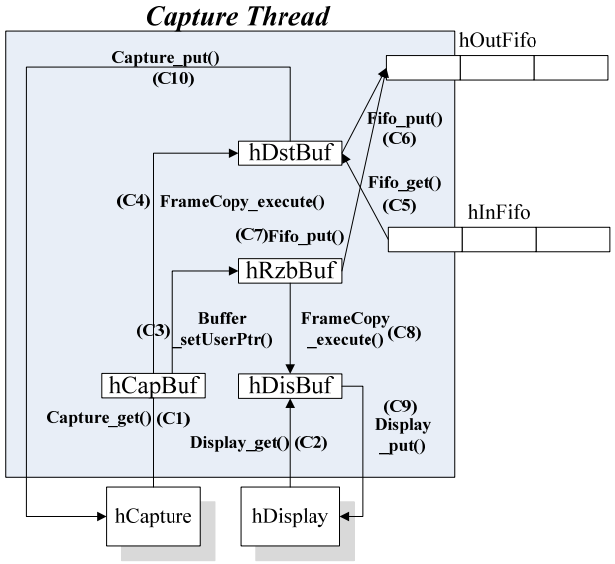


图 7　capture 线程设计图

5.3 应用程序的实现

5.3.1 capture 线程具体实现的流程

a) 调用 Capture_detectVideoStd（）函数，检测采集的视频标准；
b) 调用 VideoStd_getResolution（）函数，获取视频格式的分辨率；
c) 调用 BufTab_create（）函数，创建缓存队列；
d) 调用 Capture_create（）函数，创建视频采集实例；
e) 调用 Capture_get（）函数，从采集设备读取一帧视频数据到 hCapBuf 中；
f) 调用 Display_get（）函数，从显示设备获取一块空 Buf 用于存放经过裁剪的视频数据；
g) 调用 Framecopy_execute（）函数，将 736*576 的数据裁剪成 720*576 的数据用于本地存储，还有一次是把 20*576 的数据裁剪成 640*480 的数据用于实时回放；

h) 调用 Capture_put（）函数，发送一块 Buf 给采集设备用于获取视频数据；

i) 调用 Display_put（）函数，将经过裁剪的视频数据发送给显示设备用于实时显示；

j) 调用 Fifo_get（）函数，通过 hOutFifo 管道获取一块空 Buf，用于存放 D1 格式的视频数据；

k) 调用 Fifo_put（）函数，将 D1 和 CIF 格式的视频数据发送给 video 线程。

capture 线程的流程图如图 8 所示：

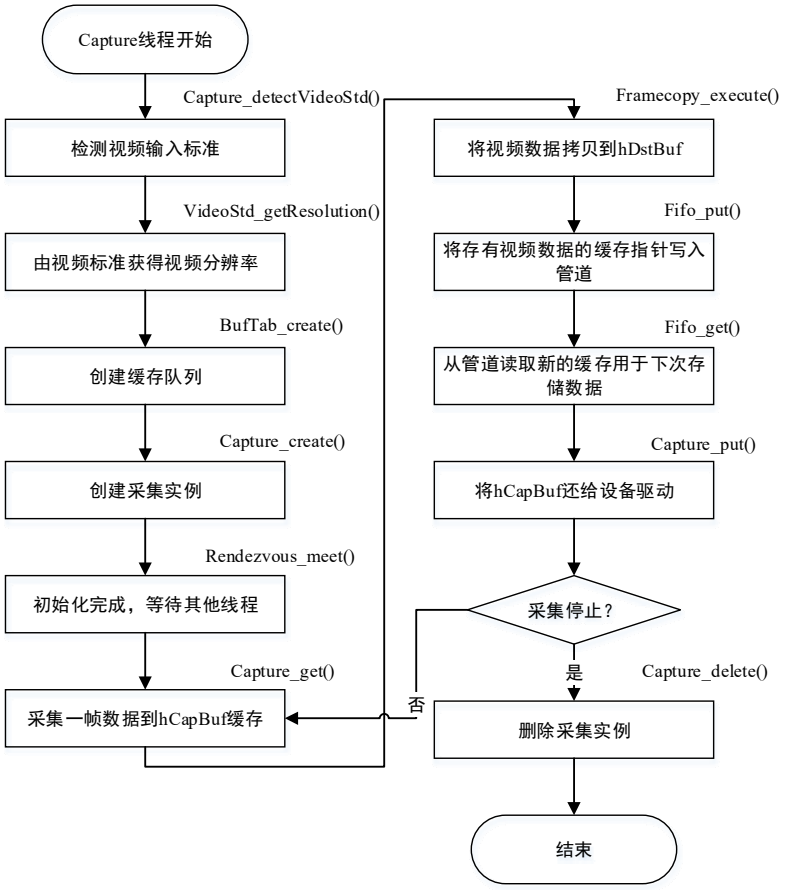


图 8　capture 线程流程图

5.3.2 变量及函数说明

➢ 变量说明

envp：线程创建时，由主线程传入的结构体环境变量；

cAttrs：capture 属性，初始化为 dm365 下 capture 属性的默认值；

dAttrs：display 属性，初始化为 dm365 下 display 属性的默认值；

hCapture：capture 设备描述符；

hDisplay：display 设备描述符；

➢ 函数说明

Int Capture_detectVideoStd(Capture_Handle hCapture, VideoStd_Type *videoStdPtr,Capture_Attrs *attrs)

参数：hCapture [in]：capture 采集设备驱动实例；

videoStdPtr [out]：函数检测到的视频标准由该指针返回；

Attrs [in]：用于创建 capture 设备驱动实例的属性；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：检测与摄像头相连的视频流输入的视频标准。

Int VideoStd_getResolution(VideoStd_Type videoStd,Int32 *widthPtr, Int32 *heightPtr)

参数：videoStd [in]：需要进行计算的视频标准；

widthPtr [out]：该指针返回此视频标准分辨率的宽度；

heightPtr：[out]；该指针返回此视频标准分辨率的高度；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：传入一个视频标准，输出此视频标准的分辨率。

BufTab_Handle BufTab_create(Int numBufs, Int32 size, Buffer_Attrs *attrs)

参数：numBufs [in]：需要分配的内存数量；

size [in]：需要分配的每块内存的大小，单位为 bytes；

attrs [in]：创建内存所用的属性；

返回：成功返回 BufTab_Handle 类型的临时变量，返回 NULL 表示失败；

功能：根据所需的内存大小和内存数量来创建一块内存表。

Capture_Handle Capture_create(BufTab_Handle hBufTab, Capture_Attrs *attrs)

参数：hBufTab [in]：上一个函数创建的内存表，用于 capture 设备；

attrs [in]：创建 capture 设备实例所用的属性，可根据需求自行修改；

返回：成功返回 Capture_Handle 类型的临时变量，NULL 表示失败；

功能：创建一个采集驱动实例，打开设备节点，通过 attrs 属性对内核驱动做一些初始化配置，配置两路视频数据，一路为 D1（720*576）格式，用于编码本地存储，还有一路为 CIF（352*288）格式，用于编码上传。

Display_Handle Display_create(BufTab_Handle hBufTab, Display_Attrs *attrs)

参数：hBufTab [in]：内存表，用于 display 设备；

attrs [in]：创建 display 设备实例所用的属性，可根据需求自行修改；

返回：成功返回 Display_Handle 类型的临时变量，NULL 表示失败；

功能：创建播放设备驱动实例，利用 attrs 参数对内核中播放设备驱动进行一些初始化配置。

Int Framecopy_execute(Framecopy_Handle hFc,Buffer_Handle hSrcBuf, Buffer_Handle hDstBuf)

参数：hFc [in]：配置完成的 Framecoyp 句柄；

hSrcBuf [in]：裁剪之前的源数据内存；

hDstBuf [in]：裁剪之后的数据内存；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：执行裁剪工作，将 hSrcBuf 内存中特定大小的视频数据裁剪成 hDstBuf 内存中设定好的视频大小，在我们的应用程序中两次调用了该函数，一次是将 736*576 的数据裁剪成 720*576 的数据，还有一次是把 720*576 的数据裁剪成 640*480 的数据，用于视频实时播放。

Int Capture_get(Capture_Handle hCapture, Buffer_Handle *hBufPtr)

参数：hCapture [in]：之前 capture_create()创建的采集设备句柄，视频数据由此获取；

hBufPtr [out]：该指针指向采集而得的视频数据的首地址；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：从采集设备驱动中获取原始视频数据。

Int Display_get(Display_Handle hDisplay, Buffer_Handle *hBufPtr)

参数：hDisplay [in]：之前 display_create()创建的显示设备句柄；

hBufPtr [out]：该指针指向显示设备提供的内存的首地址；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：从显示设备驱动实例中获取一块空 buffer，用于存放经过处理的，适应屏幕分辨率的视频数据。

Int Capture_put(Capture_Handle hCapture, Buffer_Handle hBuf)

参数：hCapture [in]：之前 capture_create()创建的采集设备句柄；

hBuf [in]：发送给采集设备的 buffer；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：发送一块 buffer 给采集设备驱动来获取视频数据。

Int Display_put(Display_Handle hDisplay, Buffer_Handle hBuf)

参数：hDisplay [in]：之前 display_create()创建的显示设备句柄；

hBuf [in]：发送给显示设备的 buffer；

返回：Dmai_EOK 表示成功，返回负值表示失败；

功能：将 hBuf 中的经过缩放裁剪的视频数据传给显示设备进行实时播放。

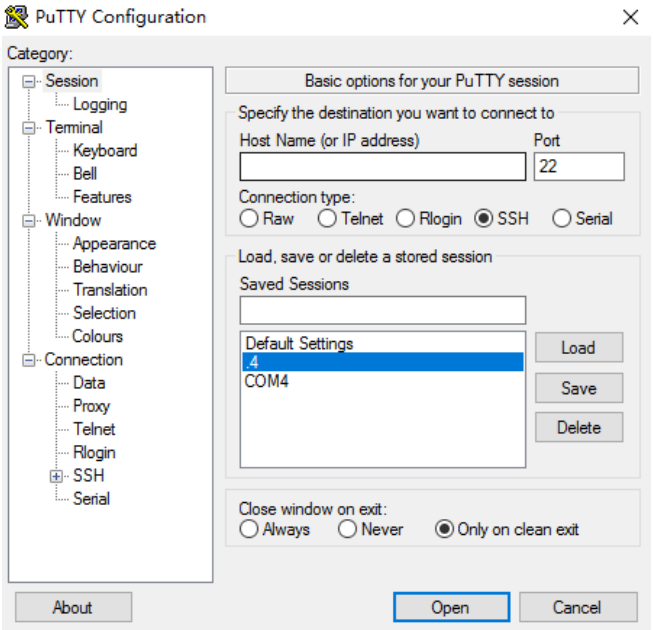
➢ 源码

参见试验箱 encode 下的 capture.c 文件

六. 实验步骤

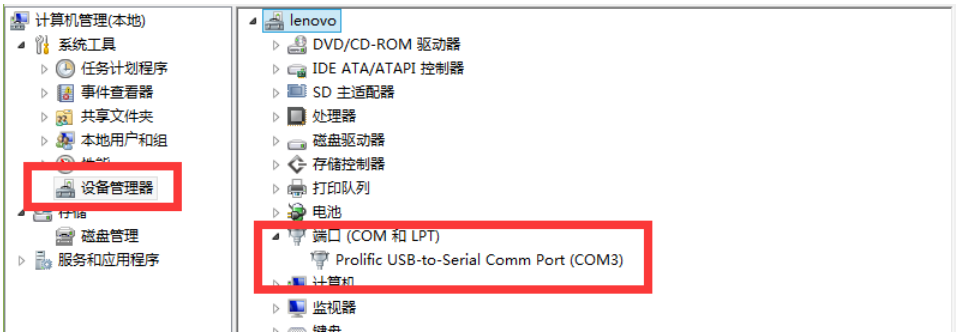
步骤 1：硬件连接

首先通过 putty 软件使用 ssh 通信方式登录到服务器，如下图一所示（在 Hostname 栏输入服务器的 ip 地址）：



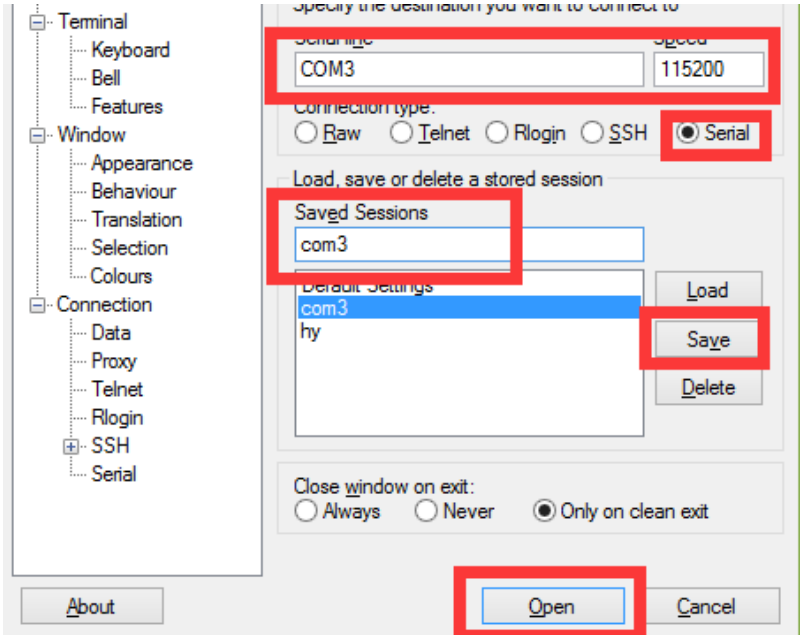
图一 打开 putty 连接

接着查看串口号，通过 putty 软件使用串口通信方式连接实验箱，如下图二所示：



图二 端口号查询

选择 putty 串口连接实验箱如三所：s



图三 putty 串口连接配置

输入启动参数，接着启动内核，如下图四所示：

```
DM365 EVM :>setenv bootargs mem=70M console=ttyS0,115200n8 root=/dev/nfs rw nfsr
oot=192.168.0.135:/home/st1/zh/filesys_test/ ip=192.168.0.109:192.168.0.135:192.
168.0.1:255.255.255.0::eth0:off eth=00:40:01:C1:56:19 video=davincifb:vid0=OFF:v
id1=OFF:osd0=640x480x16,600K:osd1=0x0x0,0K dm365_imp.oper_mode=0 davinci_capture
.device_type=1 davinci_enc_mngr.ch0_output=LCD
DM365 EVM :>boot

Loading from NAND 1GiB 3,3V 8-bit, offset 0x400000
   Image Name:   Linux-2.6.18-plc_pro500-davinci_
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1995748 Bytes =  1.9 MB
   Load Address: 80008000
   Entry Point:  80008000
## Booting kernel from Legacy Image at 80700000 ...
   Image Name:   Linux-2.6.18-plc_pro500-davinci_
   Image Type:   ARM Linux Kernel Image (uncompressed)
```

图四 输入启动参数启动

输入用户名 root 登录实验箱如下图五所示：

```
zjut login: root


Welcome to MontaVista(R) Linux(R) Professional Edition 5.0.0 (0801921).

login[754]: root login on 'console'
/******Set QT environment********/
[root@zjut ~]#
```

图五 登陆实验箱

步骤 2：编译 dvsdk 的代码

首先确保 pc 或服务器上已经安装了 dvsdk，接着进入到 dvsdk 的 encode 源码目录。

```
$ cd /root/wy/dvsdk/dvsdk_demos_2_10_00_17/dm365/encode
```

（注意）/wy/dvsdk 表示 dvsdk 安装目录在/root/wy/dvsdk，请根据实际安装情况进行修改。

先执行编译清除命令 make clean 然后执行编译命令 make。

```
st1@ubuntu:/root/wy/dvsdk/dvsdk_demos_2_10_00_17/dm365/encode$ make clean
Removing generated files..
st1@ubuntu:/root/wy/dvsdk/dvsdk_demos_2_10_00_17/dm365/encode$ make

======== Building encode ========
Configuring application using encode.cfg

making package.mak (because of package.bld) ...
generating interfaces for package encode_config (because package/package.xdc.inc is
older than package.xdc) ...
configuring encode.x470MV from package/cfg/encode_x470MV.cfg ...
Auto register ti.sdo.fc.ires.hdvicp.HDVICP
Auto register ti.sdo.fc.ires.vicp.VICP2
Auto register ti.sdo.fc.ires.addrspace.ADDRSPACE
Auto register ti.sdo.fc.ires.edma3chan.EDMA3CHAN
    will link with ti.sdo.simplewidget:lib/simplewidget_dm365.a470MV
```

在这里可以发现用户名变成了 root，是因为使用了 root 用户进行操作，

编译完成后可以查看目录通过命令 ls -ls 。可以看到生成了 encode 代码如下图所示：

```
st1@ubuntu:/root/wy/dvsdk/dvsdk_demos_2_10_00_17/dm365/encode$ ls -ls
total 15272
  20 -rwxrwxrwx 1 st1  st1    19969  5月 15  2017 !
  16 -rwxrwxrwx 1 st1  st1    16173 12月 26  2016 capture.c
   4 -rwxrwxrwx 1 st1  st1     1238  3月 30  2016 capture.h
  20 -rw-rw-r-- 1 st1  st1    18284  5月  8 15:12 capture.o
   4 -rwxrwxrwx 1 st1  st1      435  3月 30  2016 CIFppssps.txt
   4 -rwxrwxrwx 1 st1  st1     3118  3月 30  2016 codecs.c
   8 -rw-rw-r-- 1 st1  st1     4356  5月  8 15:12 codecs.o
  72 -rwxrwxrwx 1 st1  st1    73728  5月 10  2016 cscope.in.out
 128 -rwxrwxrwx 1 st1  st1   127175  5月 10  2016 cscope.out
  92 -rwxrwxrwx 1 st1  st1    90372  5月 10  2016 cscope.po.out
   4 -rwxrwxrwx 1 st1  st1      604  3月 30  2016 D1ppssps.txt
1028 -rwxrwxr-x 1 st1  st1  1052442  5月  8 15:12 encode
```

接着通过命令行将 encode 代码复制到 filesys_test/opt/dm365 目录下

```
$ cp encode /home/st1/filesys_test/opt/dm365/encode_stx        (x 为自定义数字)
```

将编译后生成的 encode 可执行文件拷贝到实验箱文件系统 filesys_test 的/opt/dm365 目

录下并同时重新命名为 encode_stx,如图下所示：

```
st1@ubuntu:/root/wy/dvsdk/dvsdk_demos_2_10_00_17/dm365/encode$ cp encode /home/st1/f
ilesys_test/opt/dm365/encode_stx
st1@ubuntu:/root/wy/dvsdk/dvsdk_demos_2_10_00_17/dm365/encode$
```

接着在 pc 机或服务器上修改/opt/dm365 下的两个文件 task_db_1.sh 和 task_db_2.sh ，task_db_1.sh 文件修改内容如下：

第一处修改。

```
#!/bin/sh
killall encode
echo ======================/opt/dm365/task_db_1================================
```

将上图改为下图所示。

```
#!/bin/sh
killall encode_stx
echo ======================/opt/dm365/task_db_1================================
```

第二处修改。

```

sleep 3
DMAI_DEBUG=2 encode -y 9 -v h.264 -0 lcd > encode.log &
# ./encodedb -y 2 -v h.264 -0 lcd -s 1.g711 &
```

将上图改为下图所示。

```

sleep 3
DMAI_DEBUG=2 encode_stx -y 9 -v h.264 -0 lcd > encode.log &
# ./encodedb -y 2 -v h.264 -0 lcd -s 1.g711 &
```

同理将 task_db_2.sh 文件对应位置的 encode 改为 encode_stx，下图为修改后样式。

```
#!/bin/sh
killall encode_stx
echo ======================/opt/dm365/task_db_2================================
```

```
sleep 3
DMAI_DEBUG=2 encode -y 2 -v h.264 -0 lcd > encode.log &
#DMAI_DEBUG=2 ./encode_mceb -y 2 -v h.264 -0 lcd > encode.log &
```

当设置完毕后记得要保存。接着点机键盘第二排第四个按钮将出现视频画面（如过没有出现画面请点击第四排第四个按键切换摄像头）如下图 1 所示：

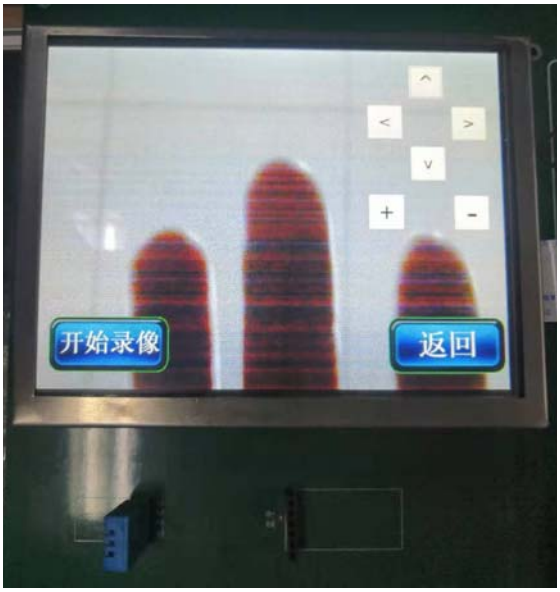


图 1 点击按键 lcd 显示图

接着插入 SD 卡，然后点击开始录像功能，录制一段时间后点击结束录像，那么视屏就会被保存在本地的 SD 卡当中，如下图 2 所示，点击停止录像如图 3 所示：

图 2 点击开始录像按钮

图 3 点击结束录制后

此时查看/mnt/mmc/video 目录发现生成视频文件如下图所示：

```
[root@zjut video]# ls
2017-01-13-08-08-13CIF.264  2017-01-13-08-08-13D1.264
[root@zjut video]#
```

取出保存有上图的 h264 格式的视频文件的 SD 卡，然后在 pc 机上使用能够支持 h264 文件格式的播放器播放 SD 卡中的文件，如图 4 所示：

图 4 在 pc 机进行播放

实验结束。

本实验设计的主要代码 capture.c 如下所示：

```
/*
 *capture.c*
 ============================================================================
 * Copyright (c) Texas Instruments Inc 2009
 *
```

```
 * Use of this software is controlled by the terms and conditions found in the
 * license agreement under which this software has been supplied or provided.
 *
============================================================================
 */
#include <xdc/std.h>
#include <string.h>

#include <ti/sdo/dmai/Fifo.h>
#include <ti/sdo/dmai/Pause.h>
#include <ti/sdo/dmai/BufTab.h>
#include <ti/sdo/dmai/Capture.h>
#include <ti/sdo/dmai/Display.h>
#include <ti/sdo/dmai/VideoStd.h>
#include <ti/sdo/dmai/Framecopy.h>
#include <ti/sdo/dmai/BufferGfx.h>
#include <ti/sdo/dmai/Rendezvous.h>

#include "capture.h"
#include "../demo.h"

#define MODULE_NAME         "Capture Thread"

/* Buffering for the display driver */
#define NUM_DISPLAY_BUFS                3

/* Buffering for the capture driver */
#define NUM_CAPTURE_BUFS                3

/* Number of buffers in the pipe to the capture thread */
/* Note: It needs to match capture.c pipe size */
#define VIDEO_PIPE_SIZE                 3

#define NUM_BUFS (NUM_CAPTURE_BUFS + NUM_DISPLAY_BUFS + VIDEO_PIPE_SIZE)

/******************************************************************************
 * captureThrFxn
 ******************************************************************************/
Void *captureThrFxn(Void *arg)
{
    CaptureEnv              *envp       = (CaptureEnv *) arg;
    Void                    *status     = THREAD_SUCCESS;
    Capture_Attrs           cAttrs      = Capture_Attrs_DM365_DEFAULT;
    Display_Attrs           dAttrs      = Display_Attrs_DM365_VID_DEFAULT;
    Framecopy_Attrs         fcAttrs     = Framecopy_Attrs_DEFAULT;
    BufferGfx_Attrs         gfxAttrs = BufferGfx_Attrs_DEFAULT;
    BufferGfx_Attrs         RzbgfxAttrs = BufferGfx_Attrs_DEFAULT;
```

```
    Capture_Handle              hCapture = NULL;
    Display_Handle              hDisplay = NULL;
    Framecopy_Handle            hFcDisp    = NULL;
    Framecopy_Handle            hFcEnc     = NULL;
    BufTab_Handle               hBufTab    = NULL;
    Buffer_Handle               hDstBuf, hCapBuf, hDisBuf, hRzbBuf, hBuf;
    BufferGfx_Dimensions        disDim, capDim;
    VideoStd_Type               videoStd;
    Int32                       width, height, lcdwidth, lcdheight, bufSize, RzbbufSize;
    Int                         fifoRet;
    ColorSpace_Type             colorSpace = ColorSpace_YUV420PSEMI; //ColorSpace_UYVY;
    Int                         bufIdx;
    Bool                        frameCopy = TRUE;

#ifdef PDEBUG
    FILE                        *outFile                = NULL;
    /* Open the output video file */
    outFile = fopen("display736x576_yuv420sp.yuv", "w");

    if (outFile == NULL) {
        ERR("Failed to open %s for writing\n", "display736x576_yuv420sp.yuv");
        cleanup(THREAD_FAILURE);
    }
#endif

    /* Create capture device driver instance */
    cAttrs.numBufs = NUM_CAPTURE_BUFS;
    cAttrs.videoInput = envp->videoInput;
    cAttrs.videoStd = envp->videoStd;

    if (Capture_detectVideoStd(NULL, &videoStd, &cAttrs) < 0) {
        ERR("Failed to detect video standard, video input connected?\n");
        cleanup(THREAD_FAILURE);
//detect
    }

    /* We only support D1 & 720P input */
    if (videoStd != VideoStd_D1_NTSC && videoStd != VideoStd_D1_PAL
        && videoStd != VideoStd_720P_60 && videoStd != VideoStd_VGA && videoStd !=
VideoStd_720P_30 && videoStd != VideoStd_D1_30) {//ov5640_cb_d1_720p

        ERR("Need D1/720P input to this demo\n");
        cleanup(THREAD_FAILURE);
    }

    if (envp->imageWidth > 0 && envp->imageHeight > 0) {
        if (VideoStd_getResolution(videoStd, &width, &height) < 0) {
```

```
                ERR("Failed to calculate resolution of video standard\n");
                cleanup(THREAD_FAILURE);
            }

            if (width < envp->imageWidth && height < envp->imageHeight) {
                ERR("User resolution (%ldx%ld) larger than detected (%ldx%ld)\n",
                    envp->imageWidth, envp->imageHeight, width, height);
                cleanup(THREAD_FAILURE);
            }

            capDim.x                = 0;
            capDim.y                = 0;
            capDim.height           = envp->imageHeight;
            capDim.width            = envp->imageWidth;
            capDim.lineLength = BufferGfx_calcLineLength(width, colorSpace);
        } else {
            /* Calculate the dimensions of a video standard given a color space */
            if (BufferGfx_calcDimensions(videoStd, colorSpace, &capDim) < 0) {
                ERR("Failed to calculate Buffer dimensions\n");
                cleanup(THREAD_FAILURE);
            }

            envp->imageWidth    = capDim.width;
            envp->imageHeight = capDim.height;
        }
        /* If it is not component capture with 720P resolution then use framecopy as
           there is a size mismatch between capture, display and video buffers. */
        if((envp->imageWidth == VideoStd_720P_WIDTH) &&
            (envp->imageHeight == VideoStd_720P_HEIGHT)) {
            frameCopy = FALSE;                  //lcy_720P
        }
        /* Calculate the dimensions of a video standard given a color space */
        envp->imageWidth    = capDim.width;
        envp->imageHeight = capDim.height;

        /* TODO: Add resizer support for frameCopy=FALSE */
        if(frameCopy == FALSE) {
            gfxAttrs.dim.height = capDim.height;
            gfxAttrs.dim.width = capDim.width;
            gfxAttrs.dim.lineLength             =             ((Int32)((BufferGfx_calcLineLength(gfxAttrs.dim.width,
colorSpace)+31)/32))*32;
            gfxAttrs.dim.x = 0;
            gfxAttrs.dim.y = 0;
            if (colorSpace ==    ColorSpace_YUV420PSEMI) {
                bufSize = gfxAttrs.dim.lineLength * gfxAttrs.dim.height * 3 / 2;
            } else {
                bufSize = gfxAttrs.dim.lineLength * gfxAttrs.dim.height * 2;
```

```
        }

        /* Create a table of buffers to use with the device drivers */
        gfxAttrs.colorSpace = colorSpace;
        hBufTab = BufTab_create(NUM_BUFS, bufSize,
                                BufferGfx_getBufferAttrs(&gfxAttrs));
        if (hBufTab == NULL) {
            ERR("Failed to create buftab\n");
            cleanup(THREAD_FAILURE);
        }
    } else {
        gfxAttrs.dim = capDim;
    }
    /* Update global data for user interface */
    gblSetImageWidth(envp->imageWidth);
    gblSetImageHeight(envp->imageHeight);

    /* Report the video standard and image size back to the main thread */
    Rendezvous_meet(envp->hRendezvousCapStd);

    if(envp->videoStd == VideoStd_720P_60) {
        cAttrs.videoStd = VideoStd_720P_30;
    } else {
        cAttrs.videoStd = envp->videoStd;
    }
    cAttrs.numBufs       = NUM_CAPTURE_BUFS;
    cAttrs.colorSpace = colorSpace;
    cAttrs.captureDimension = &gfxAttrs.dim;
    /* Create the capture device driver instance */
    hCapture = Capture_create(hBufTab, &cAttrs);

    if (hCapture == NULL) {
        ERR("Failed to create capture device\n");
        cleanup(THREAD_FAILURE);
    }

    /* Create display device driver instance */
    /* TODO: Set the videooutput to LCD according to the envp */
    dAttrs.videoOutput = Display_Output_LCD;
    dAttrs.numBufs       = NUM_DISPLAY_BUFS;
    dAttrs.colorSpace = colorSpace;
    hDisplay = Display_create(hBufTab, &dAttrs);

    if (hDisplay == NULL) {
        ERR("Failed to create display device\n");
        cleanup(THREAD_FAILURE);
    }
```

```
if (frameCopy == TRUE) {
    /* Create a buffer for the output of resizer b */
    RzbgfxAttrs.colorSpace = ColorSpace_YUV420PSEMI;
    /* TODO: Get the resolution from resizer b */
    width = envp->resizeWidth;
    height = envp->resizeHeight;
    RzbgfxAttrs.dim.width = width;
    RzbgfxAttrs.dim.height = height;
    /* Ensure that lineLength is multiple of 32 */
    RzbgfxAttrs.dim.lineLength =
    ((Int32)((BufferGfx_calcLineLength(RzbgfxAttrs.dim.width,
                                       ColorSpace_YUV420PSEMI)+31)/32))*32;
    RzbgfxAttrs.bAttrs.reference = TRUE;

    printf("width##############======%d\n", width);
    printf("height##############======%d\n", height);

    if (colorSpace ==    ColorSpace_YUV420PSEMI) {
        RzbbufSize = RzbgfxAttrs.dim.lineLength
                     * RzbgfxAttrs.dim.height * 3 / 2;
    } else {
        RzbbufSize = RzbgfxAttrs.dim.lineLength
                     * RzbgfxAttrs.dim.height * 2;
    }
    hRzbBuf = Buffer_create(RzbbufSize, BufferGfx_getBufferAttrs(&RzbgfxAttrs));

    if (hRzbBuf == NULL) {
        ERR("Failed to create DstBuf\n");
        cleanup(THREAD_FAILURE);
    }

    /* Get a buffer from the video thread */
    fifoRet = Fifo_get(envp->hInFifo, &hDstBuf);

    if (fifoRet < 0) {
        ERR("Failed to get buffer from video thread\n");
        cleanup(THREAD_FAILURE);
    }

    /* Did the video thread flush the fifo? */
    if (fifoRet == Dmai_EFLUSH) {
        cleanup(THREAD_SUCCESS);
    }

    /* Create frame copy module for display buffer */
    fcAttrs.accel = TRUE;
    hFcDisp = Framecopy_create(&fcAttrs);
```

```
        if (hFcDisp == NULL) {
            ERR("Failed to create frame copy job\n");
            cleanup(THREAD_FAILURE);
        }

        /* Configure frame copy jobs */
        if (Framecopy_config(hFcDisp,
                                BufTab_getBuf(Capture_getBufTab(hCapture), 0),
                                BufTab_getBuf(Display_getBufTab(hDisplay), 0)) < 0) {
            ERR("Failed to configure frame copy job\n");
            cleanup(THREAD_FAILURE);
        }

        /* Create frame copy module for encode buffer */
        fcAttrs.accel = TRUE;
        hFcEnc = Framecopy_create(&fcAttrs);

        if (hFcEnc == NULL) {
            ERR("Failed to create frame copy job\n");
            cleanup(THREAD_FAILURE);
        }

        if (Framecopy_config(hFcEnc,
                                BufTab_getBuf(Capture_getBufTab(hCapture), 0),
                                hDstBuf) < 0) {
            ERR("Failed to configure frame copy job\n");
            cleanup(THREAD_FAILURE);
        }
    }else {
        for (bufIdx = 0; bufIdx < VIDEO_PIPE_SIZE; bufIdx++) {
            /* Queue the video buffers for main thread processing */
            hBuf = BufTab_getFreeBuf(hBufTab);
            if (hBuf == NULL) {
                ERR("Failed to fill video pipeline\n");
                cleanup(THREAD_FAILURE);
            }
            /* Send buffer to video thread for encoding */
            if (Fifo_put(envp->hOutFifo, hBuf) < 0) {
                ERR("Failed to send buffer to video thread\n");
                cleanup(THREAD_FAILURE);
            }
        }
    }
    /* Signal that initialization is done and wait for other threads */
    Rendezvous_meet(envp->hRendezvousInit);
```

```
while (!gblGetQuit()) {
    /* Pause processing? */
    Pause_test(envp->hPauseProcess);

    /* Capture a frame */
    if (Capture_get(hCapture, &hCapBuf) < 0) {
        ERR("Failed to get capture buffer\n");
        cleanup(THREAD_FAILURE);
    }

    /* Get a buffer from the display device */
    if (Display_get(hDisplay, &hDisBuf) < 0) {
        ERR("Failed to get display buffer\n");
        cleanup(THREAD_FAILURE);
    }

    if (frameCopy == TRUE) {
        /* Set hRzbBuf's UserPtr to the offset of the hCapBuf */
        Buffer_setUserPtr(hRzbBuf,(Int8*)(Buffer_getUserPtr(hCapBuf)
                                          +Buffer_getSize(hCapBuf)));


        /* Get a buffer from the video thread */
        fifoRet = Fifo_get(envp->hInFifo, &hDstBuf);

        if (fifoRet < 0) {
            ERR("Failed to get buffer from video thread\n");
            cleanup(THREAD_FAILURE);
        }

        /* Did the video thread flush the fifo? */
        if (fifoRet == Dmai_EFLUSH) {
            cleanup(THREAD_SUCCESS);
        }

        /* Copy the captured buffer to the encode buffer */
        if (Framecopy_execute(hFcEnc, hCapBuf, hDstBuf) < 0) {
            ERR("Failed to execute frame copy job\n");
            cleanup(THREAD_FAILURE);
        }

        /* Send buffer to video thread for encoding */
        if (Fifo_put(envp->hOutFifo, hDstBuf) < 0) {
            ERR("Failed to send buffer to video thread\n");
            cleanup(THREAD_FAILURE);
        }
```

```
                /* Send resized buffer to video thread for encoding */
                if (Fifo_put(envp->hOutFifo, hRzbBuf) < 0) {
                    ERR("Failed to send buffer to video thread\n");
                    cleanup(THREAD_FAILURE);
                }
            }else {
                /* Send buffer to video thread for encoding */
                if (Fifo_put(envp->hOutFifo, hCapBuf) < 0) {
                    ERR("Failed to send buffer to video thread\n");
                    cleanup(THREAD_FAILURE);
                }
            }
            if (frameCopy == TRUE) {
                /* Framecopy start at the user defined (x,y) */
                BufferGfx_getDimensions(hCapBuf, &capDim);
                /*TODO*/
                capDim.x = (720-640)/2;
                capDim.y = (576-480)/2 & ~0x1;
                BufferGfx_setDimensions(hCapBuf, &capDim);

                /* Copy the captured buffer to the display buffer */
                if (Framecopy_execute(hFcDisp, hCapBuf, hDisBuf) < 0) {
                    ERR("Failed to execute frame copy job\n");
                    cleanup(THREAD_FAILURE);
                }

                /* Framecopy start at the user defined (x,y) */
                BufferGfx_getDimensions(hCapBuf, &capDim);
                /*TODO*/
                capDim.x = 0;
                capDim.y = 0;
                BufferGfx_setDimensions(hCapBuf, &capDim);

#ifdef PDEBUG
                if (fwrite(Buffer_getUserPtr(hCapBuf),
                                635904, 1, outFile) != 1) {
                    ERR("Error writing the data to file\n");
                    cleanup(THREAD_FAILURE);
                }
                cleanup(THREAD_FAILURE);
#endif

                /* Release display buffer to the display device driver */
                if (Display_put(hDisplay, hDisBuf) < 0) {
                    ERR("Failed to put display buffer\n");
                    cleanup(THREAD_FAILURE);
                }
```

```
            } else {
                /* Release display buffer to the display device driver */
                if (Display_put(hDisplay, hCapBuf) < 0) {
                    ERR("Failed to put display buffer\n");
                    cleanup(THREAD_FAILURE);
                }
            }

            if (frameCopy == TRUE) {
                /* Return the buffer to the capture driver */
                if (Capture_put(hCapture, hCapBuf) < 0) {
                    ERR("Failed to put capture buffer\n");
                    cleanup(THREAD_FAILURE);
                }
            } else {
                /* Return the buffer to the capture driver */
                if (Capture_put(hCapture, hDstBuf) < 0) {
                    ERR("Failed to put capture buffer\n");
                    cleanup(THREAD_FAILURE);
                }
            }

            /* Incremement statistics for the user interface */
            gblIncFrames();

        }

cleanup:
        /* Make sure the other threads aren't waiting for us */
        Rendezvous_force(envp->hRendezvousCapStd);
        Rendezvous_force(envp->hRendezvousInit);
        Pause_off(envp->hPauseProcess);
        Fifo_flush(envp->hOutFifo);

        /* Meet up with other threads before cleaning up */
        Rendezvous_meet(envp->hRendezvousCleanup);

        /* Clean up the thread before exiting */
        if (hFcDisp) {
            Framecopy_delete(hFcDisp);
        }

        if (hFcEnc) {
            Framecopy_delete(hFcEnc);
        }

        if (hDisplay) {
```

```
        Display_delete(hDisplay);
    }

    if (hCapture) {
        Capture_delete(hCapture);
    }

    /* Clean up the thread before exiting */
    if (hBufTab) {
        BufTab_delete(hBufTab);
    }

    if (hRzbBuf) {
        Buffer_delete(hRzbBuf);
    }

#ifdef PDEBUG
    if (outFile) {
        fclose(outFile);
    }
#endif

    return status;
}
```

Makefile 文件代码如下所示：

```
# Makefile
#============================================================================
# Copyright (c) Texas Instruments Inc 2009
#
# Use of this software is controlled by the terms and conditions found in the
# license agreement under which this software has been supplied or provided.
#
============================================================================

ROOTDIR = ../../..
TARGET = $(notdir $(CURDIR))

include $(ROOTDIR)/Rules.make

# Comment this out if you want to see full compiler and linker output.
VERBOSE = @

# Package path for the XDC tools
XDC_PATH                                                                    =
$(USER_XDC_PATH);../../packages;$(DMAI_INSTALL_DIR)/packages;$(CE_INSTALL_DIR)/package
s;$(FC_INSTALL_DIR)/packages;$(LINK_INSTALL_DIR)/packages;$(XDAIS_INSTALL_DIR)/packag
es;$(LINUXUTILS_INSTALL_DIR)/packages;$(CODEC_INSTALL_DIR)/packages;$(EDMA3_LLD_IN
```

```
STALL_DIR)/packages

# Where to output configuration files
XDC_CFG         = $(TARGET)_config

# Output compiler options
XDC_CFLAGS  = $(XDC_CFG)/compiler.opt

# Output linker file
XDC_LFILE       = $(XDC_CFG)/linker.cmd

# Input configuration file
XDC_CFGFILE = $(TARGET).cfg

# Platform (board) to build for
XDC_PLATFORM = ti.platforms.evmDM365

# Target tools
XDC_TARGET = gnu.targets.MVArm9

# The XDC configuration tool command line
CONFIGURO = $(XDC_INSTALL_DIR)/xs xdc.tools.configuro

C_FLAGS += -Wall -g

LD_FLAGS += -lpthread -lpng -ljpeg -lfreetype -lasound

COMPILE.c = $(VERBOSE) $(MVTOOL_PREFIX)gcc $(C_FLAGS) $(CPP_FLAGS) -c
LINK.c = $(VERBOSE) $(MVTOOL_PREFIX)gcc $(LD_FLAGS)

SOURCES = $(wildcard *.c) $(wildcard ../*.c)
HEADERS = $(wildcard *.h) $(wildcard ../*.h)

OBJFILES = $(SOURCES:%.c=%.o)

.PHONY: clean install

all:    dm365

dm365:      dm365_al

dm365_al:  $(TARGET)

install:        $(if $(wildcard $(TARGET)), install_$(TARGET))

install_$(TARGET):
        @install -d $(EXEC_DIR)
```

```
	@install $(TARGET) $(EXEC_DIR)
	@install $(TARGET).txt $(EXEC_DIR)
	@echo
	@echo Installed $(TARGET) binaries to $(EXEC_DIR)..

$(TARGET):	$(OBJFILES) $(XDC_LFILE)
	@echo
	@echo Linking $@ from $^..
	$(LINK.c) -o $@ $^

$(OBJFILES):	%.o: %.c $(HEADERS) $(XDC_CFLAGS)
	@echo Compiling $@ from $<..
	$(COMPILE.c) $(shell cat $(XDC_CFLAGS)) -o $@ $<

$(XDC_LFILE) $(XDC_CFLAGS):	$(XDC_CFGFILE)
	@echo
	@echo ======== Building $(TARGET) ========
	@echo Configuring application using $<
	@echo
	$(VERBOSE) XDCPATH="$(XDC_PATH)" $(CONFIGURO) -c $(MVTOOL_DIR) -o
$(XDC_CFG) -t $(XDC_TARGET) -p $(XDC_PLATFORM) $(XDC_CFGFILE)

clean:
	@echo Removing generated files..
	$(VERBOSE) -$(RM) -rf $(XDC_CFG) $(OBJFILES) $(TARGET) *~ *.d .dep
```