嵌入式系统原理实验报告七

通信 1503 班 201503090323 叶启彬

实验七 视频采集发送接收实验

一、实验目的

1. 了解视频传输及回放的原理；
2. 了解共享内存在视频存储中的作用；
3. 了解视频解码模块的实现方式。

二、实验要求

1. 掌握视频存储模块的实现方式；
2. 掌握视频回放显示的实现方式；
3. 编写视频解码回放的启动脚本。

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，应用层 encode 源码，DMAI 库。

四、实验原理

1. 概述

1.1 视频采集

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。

为了允许在 DSP 处理视频的同时把已编码的视频帧写入 linux 文件系统，writer 线程被设置成一个单独的线程，来执行 linux 文件系统的 I/O 操作。视频存储分两路进行存储，一路存储在 SD 卡中，一路存储在共享内存中。

SD 卡（Secure Digital Memory Card）即安全数码卡，是一种基于半导体快闪记忆器的新一代记忆设备，被广泛应用于便携式装置，例如，数码相机、个人数码助理（PDA）和多媒体播放器等。SD 卡由日本松下、东芝及美国 SanDisk 公司于 1999 年 8 月共同开发研制。大小犹如一张邮票的 SD 记忆卡，重量只有 2 克，但却拥有高记忆容量、快速数据传输率、极大的移动灵活性以及很好的安全性。

SD 卡在 24mm×32mm×2.1mm 的体积内结合了 SanDisk 快闪记忆卡控制与 MLC（Multilevel Cell）技术和 Toshiba（东芝）0.16u 及 0.13u 的 NAND 技术，通过 9 针的接口界面与专门的驱动器相连接，不需要额外的电源来保持其上记忆的信息。而且是一体化固体介质，没有任何移动部分，所以不用担心机械运动的损坏。SD 卡的技术是基于 MultiMedia 卡（MMC）格式上发展而来，大小和 MMC 相近，尺寸为 32mm×24mm×2.1mm。长宽和 MMC 一样，只是比 MMC 厚了 0.7mm，以容纳更大容量的存储单元。SD 卡与 MMC 卡保持着向上兼容，也就是说，MMC 可以被新的 SD 设备存取，兼容性则取决于应用软件，但 SD 卡却不可以被 MMC 设备存取（SD 卡外型采用了与 MMC 厚度一样的导轨式设计，以使 SD 设备可以适合 MMC)。

ENCODE 和 DEV 间，DECODE 和 DEV 间有视频数据交互，均采用了共享内存的方式，共享内存是被多个进程共享的一块物理内存区域，进程间的数据可以通过共享内存交互，是进程间数据共享速度最快的一种方式。

图 1. ENCODE 进程与 DEV 进程间共享内存实现方式

共享内存两个进程间的同步都通过信号量来实现。信号量（Semaphore）是一种在多线程（进程）环境下保护共享资源的机制，使得共享资源在被多个线程（进程）同时访问时，只被一个线程（进程）使用。信号量的值只能通过 PV 操作改变。

通常情况，当信号量 S>=0 时，S 的数值就表示当前可用资源的数量。如果某进程想要申请一个单元的共享资源，则可以调用 P 操作将 S 减 1，此时就表示这个单元的共享资源已经被该进程占用，其他进程无法使用；若 S<=0，则表示所有共享资源都已经被占用，如果再有进程想要使用共享资源就必须等待正在使用共享资源的进程释放这部分资源，才能够调用 P 操作申请。调用进程在使用完共享资源后，需要调用 V 操作释放一个单元的共享资源并将信号量 S 加 1，如果此时 S>0，则可以唤醒一个处在等待状态的进程去获取这部分共享资源。

1.2 视频回放

H.264 是在 MPEG-4 的基础上建立起来的，因此它也继承了 MPEG-4 视频编解码标准的特性，编码效率高、视频质量佳、网络适应能力强、差错控制可靠，但与以往的视频编解码标准相比，它具有如下几个显著新特性：

全新的帧内预测编码模式

以往的视频编解码标准采用的都是帧间预测的编解码方式，但是在 H.264 视频编解码标准中，有主帧 I 帧与辅帧 P 帧之分，往往编码后的几十个帧中只有一个为 I 帧，更多的是 P 帧，当且仅当编码 I 帧时会使用帧内预测，I 帧与 P 帧间使用帧间预测编码，P 帧只记录与其前一帧图像的差值，因此图像的冗余度很低，编码效率大大提高。

全新的算法结构

根据具体实现功能的不同，H.264 视频编解码标准被分为视频编码层（Video Coding Layer，VCL）和网络抽象层（Network Abstraction Layer，NAL），由 VCL 层完成视频数据的高效编码，并由 NAL 层完成编码后数据的网络传输。前文中提到了 H.264 具有较强的网络适应能力，因此 NAL 可以利用 RTP、TCP、UDP 等网络传输协议，在不同的网络条件下（如 CDMA，GPRS，WCDMA 等）完成视频数据的上传与接收。

传输可靠性高

H.264 视频编解码标准提供了解决网络传输丢包的差错消除技术，通过采用帧内图像刷新来实现视频流的时间同步，通过动态改变量化步长以及数据分割方法来适应信道当前的码率，适用于在高误码率和丢包多发环境中传输视频数据，提升了传输的可靠性。

2. 基本原理

2.1 视频采集

CIF 和 D1 分辨率的视频数据经 Video 线程编码后通过管道发送至 Writer 线程的缓存中。考虑到视频数据的大小和传输信道的带宽限制等条件，因此把 D1 分辨率的数据存储到设备的 SD 卡中，将 CIF 分辨率的数据用于网络上传。

对于编码后 D1 格式的视频数据，首先调用标准的 fopen 函数以可写方式打开一个文件，以当前时间为文件名，将 D1 格式的数据以帧为单位调用标准 fwrite()函数顺序写入文件，存放在设备的/mnt/mmc/video 路径下，该路径即 SD 卡在文件系统下的挂载路径。存放在 SD 卡下的视频文件既可以通过视频解码模块解码播放，也可以导出到电脑上播放。

对于 CIF 格式的数据，由于其占用内存较小，所以本文将其用作网络上传的源数据。视频传输主要用到了 DEV 进程和 ENCODE 进程的 Writer 线程，在两个进程间开辟了一块共享内存供进程间数据的交互，Writer 线程在接收到 Video 线程编码后的 CIF 格式的数据之后，将其放入共享内存，供 DEV 进程读取。

2.2 视频回放

视频回放功能是由 DECODE 进程实现的，DECODE 进程是参照 ENCODE 进程设计的用于解码 H.264 码流的应用程序。

DECODE 进程由三个线程组成，主线程在完成一系列参数配置和同步工作后分别创建 display 线程、video 线程和 loader 线程]。Loader 线程用于读取视频数据，并通过管道传递给 video 线程，video 线程是基于 DM365 的 Codec Engine 的，用于解码 H.264 码流，解码后的数据通过管道传给 display 线程，由 display 线程调用显示相关驱动将数据在 LCD 液晶显示屏上播放出来。

五、实验内容（代码注释及步骤）

1. 实验内容

   搭建点对点视频传输模式。

2. 实验步骤：

步骤 0：将实验箱主板和底板上的部件按下图 4 方式连接；接上 CCD 摄像头（箱子里没有的话问徐老师），问徐老师拿 SD 卡，将其放入卡槽：

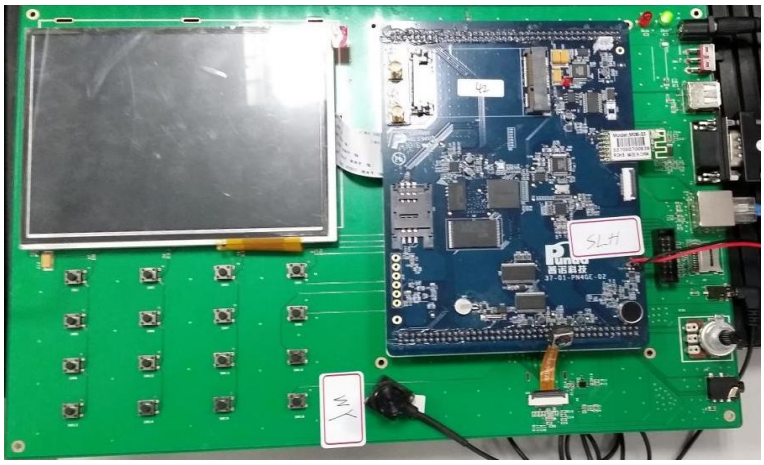

图 2. 设备连接图

步骤 1：打开 putty 软件，进行挂载：（此次挂载参数中 mem=90M）

步骤 2：在 com 口输入 cd /mnt/mmc/video/，按 ls 查看里面视频：

```
[root@zjut ~]# cd /mnt/mmc/video/
[root@zjut video]# ls
2017-01-12-12-17-51.264  2017-07-23-08-36-39.264  2017-07-23-15-45-00.264
2017-07-09-09-30-48.264  2017-07-23-14-49-39.264  2017-07-24-10-46-44.264
2017-07-09-09-30-54.264  2017-07-23-14-50-16.264
```

图 3.原来在 video 中有的视频文件

步骤 3：开启实验箱的录像功能（点击试验箱的左下角按键第二排最后一个（2，4）按键），进入摄像界面，点击试验箱的左下角按键第四排最后一个（4，4）按键，出现画面后点击屏幕上的开始录象，录下来的视频数据会保存到实验箱的本地 SD 卡内：

```
[root@zjut video]# ls
2017-01-12-12-17-51.264  2017-07-09-09-30-54.264  2017-07-23-15-45-00.264
2017-01-25-15-31-05.264  2017-07-23-08-36-39.264  2017-07-24-10-46-44.264
2017-01-25-15-31-11.264  2017-07-23-14-49-39.264
2017-07-09-09-30-48.264  2017-07-23-14-50-16.264
```

图 4.录制视频后产生了两个视频文件

视频文件 2017-01-25-15-31-05.264 以及 2017-01-25-15-31-11.264 为录制视频。视频文件存储成功。

步骤 4：编写视频解码回放脚本：在实验箱文件系统 filesys_test 的/opt/dm365 目录下新建一个脚本文件，输入命令： vim dec_zh.sh 然后输入以下内容并保存：

```
#!/bin/sh
echo -n 1 > /sys/module/dm365_imp/parameters/oper_mode
rmmod cmemk 2>/dev/null
rmmod irqk 2>/dev/null
rmmod edmak 2>/dev/null
rmmod dm365mmap 2>/dev/null
insmod cmemk.ko phys_start=0x86000000 phys_end=0x88000000 \
pools=1x384,29x56,6x1024,1x2688,1x3328,1x8704,1x10240,1x13184,1x26224,1x48396,4x460800,1x158368,2x282624,2x60825,20x718848,2x2697152,1x6045696 allowOverlap=1
phys_start_1=0x00001000 phys_end_1=0x00008000 pools_1=1x28672
insmod irqk.ko
insmod edmak.ko
insmod dm365mmap.ko
sleep 3
DMAI_DEBUG=2 decode_zhfinal -y 2 -v /mnt/mmc/video/2017-01-25-15-31-11.264 -O lcd > decode_D1.log &
```

脚本中 2017-01-25-15-31-11.264 是存储在 SD 卡中的 H.264 格式的视频的名称，根据自身情况可做修改。

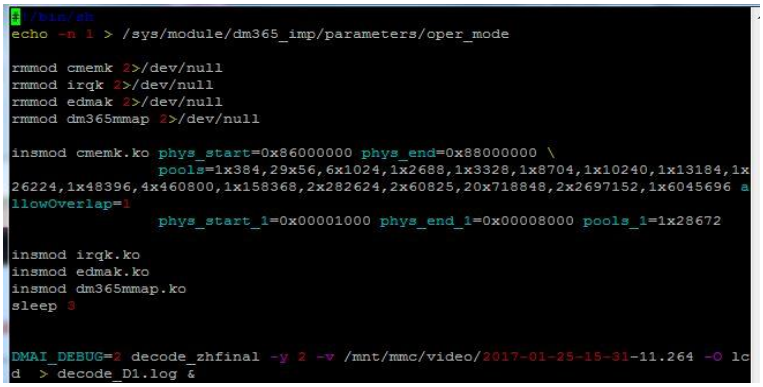

图 5. 更改脚本

步骤 5：9.以 root 身份进入板子文件系统的/opt/dm365 目录，输入命令 cd /opt/dm365，然后输入命令：ps 查看正在运行的进程，如果显示有 encode 进程，则需要 kill 掉：

图 6.kill encode 进程

主要目的是为了视频解码播放，dm365 上 encode 进程和 decode 进程由于模式不同无法同时进行。

步骤 6：在板子/opt/dm365 目录下执行解码回放脚本，输入 dec_zh.sh，然后回车：

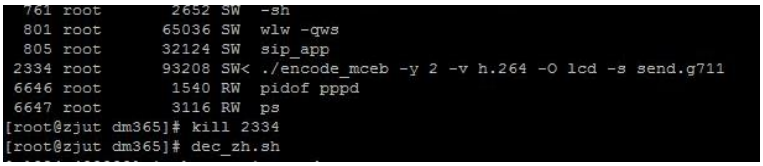

图 7.实现视频回放

当脚本执行的时候，即开始了解码进程，当按下板子上第二排最后一个按键时候，就可以看见解码的视频。视频左上角显示的 2017.01.25 15:31:18 是解码回放的视频的时间,与之前解码脚本 dec_zh.sh 中写入的/mnt/mmc/video/路径下的视频文件一一对应，表示视频解码回放成功。

图 8.采集到视频

三、实验总结及心得体会

在本次实验开始前，视频传输及回放的原理进行了理解，并了解到共享内存在视频存储中的作用以及视频解码模块的实现方式。在实验过程中，第一步的挂载花费了较多的时间，可能是在同一时间，多台设备同时进行挂载，对实验器材负荷较大导致的。此外在实验过程中，另外碰到的一个问题是，dec_zh.sh 文件在挂载后实验机子上并未更新，需要在更改完参数后，在挂载才可以更新，否则需要在 root 下进行对 dec_zh.sh 的更改。