
电 子 科 技 大 学
UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文
MASTER THESIS FOR PROFESSIONAL DEGREE



论文题目 基于网络的高清数字视频矩阵卡

专业学位类别 工程硕士

学 号 200950101015

作 者 姓 名 司国鹏

指 导 教 师 李晓峰教授

分类号 _____ 密级 _____

UDC ^{注 1} _____

学 位 论 文

基于网络的高清数字视频矩阵卡

(题名和副题名)

司国鹏

(作者姓名)

指导教师姓名	李晓峰	教授
	电子科技大学	成都
	苏大同	高工
	北京英夫美迪数字技术公司	

(姓名、职称、单位名称)

申请学位级别 硕士 专业学位类别 工程硕士

工程领域名称 电子与通信工程

提交论文日期 2013 年 4 月 论文答辩日期 2013 年 6 月

学位授予单位和日期 电子科技大学 2013 年 6 月 23 日

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

HIGH-DEFINITION DIGITAL VIDEO MATRIX CARD WORKED ON NETWROK

A Doctor Dissertation (Master Thesis or Master Research Report)Submitted to
University of Electronic Science and Technology of China

Major: **Master of Engineering**

Author: **Si Guopeng**

Advisor: **Professsor Li Xiaofeng**

School: **School of Communication & Information**
Engineering

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：_____ 日期：____年__月__日

论 文 使 用 授 权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：_____ 导师签名：_____

日期：____年__月__日

摘 要

随着科技的发展，在生活和工作中遇到的需要视频采集并播看的场合越来越多，但往往是视频采集设备很多，大屏或者播放设备很少。为了解决视频采集设备多，而观看设备少的问题，通常使用视频矩阵对视频信号源进行选择播放。以模拟开关的方式实现信号切换的模拟视频矩阵已经发展了相当长的一段时间，但由于模拟视频矩阵众多的缺点，人们更希望能使用基于网络的数字视频矩阵来替代模拟矩阵。采用基于网络的数字视频矩阵可以充分利用网络的优势，使监看视频传输更远的距离，还可以单输入多输出监看。高清晰度的矩阵是视频矩阵技术发展的趋势。因此基于网络的高清数字视频矩阵越来越受到人们的关注。

基于网络的高清数字视频矩阵，功能需求多样且计算过程复杂，对矩阵卡实现平台的处理能力和灵活性提出了较高的要求，传统的方案难以同时满足高清视频矩阵灵活性和复杂度的要求。随着高速数字信号处理器技术的不断发展，集成了 ARM 处理器内核和可编程数字信号处理器（DSP）内核的达芬奇处理器同时满足了视频矩阵较高的计算复杂度要求和功能需求的灵活性要求，德州仪器公司推出的专门针对高清数字多媒体应用的达芬奇处理器 TMS320DM6467T 具有明显的优势。

针对视频监看人员对基于网络的高清数字视频矩阵的需求，本论文主要设计并实现了一种基于 TMS320DM6467T 平台的网络高清数字视频矩阵卡。论文首先介绍了 TMS320DM6467T 平台的特点和数字视频矩阵卡的硬件方案，设计并实现了网络高清数字视频矩阵卡的嵌入式系统软件。论文的研究内容包括：

- 1, 研究了 TMS320DM6467T 嵌入式平台的开发方法，并搭建了开发和调试环境。
- 2, 研究并实现了基于 TMS320DM6467T 平台的 Linux 下设备驱动程序，主要包括 GS1671 字符设备驱动和 BT. 1120 输入时的 V4L2 视频设备驱动。
- 3, 分析了矩阵卡系统的功能需求。按照功能需求对嵌入式系统软件进行了总体设计，模块划分和详细设计。主要包括音视频采集处理，视频处理，音频处理，网络流媒体传输，硬件报警和输入信号监测。

设计出满足需求并稳定运行的高清数字视频矩阵卡。该产品实现后可用于广播电视，视频监控领域，实现视频点播，录像，存储，信号切换等功能。

关键词：高清视频矩阵，TMS320DM6467T，数字信号处理器（DSP），设备驱动

ABSTRACT

With the development of science and technology, the need to capture and play video is more and more common in lifetime and worktime, but there is often one fact: a lot of video capture devices, rarely play equipment. In order to solve the video capture device more than play equipment's problem. People usually used video matrix to choose one video source to play. Analog matrix switch video in the form of analog signal switch, it has been worked a long time, and so many shortcomings, people hope to be able to use digital video matrix based on the network to replace analog matrix. Network digital video matrix can make full use of the advantage of network, Transmitter video to farther distance, Monitor video in multiple output. High definition video matrix is the development trend of video matrix technology. Therefore, high-definition digital video matrix which based on network got more and more attention of people.

High-definition digital video matrix based on network, which is function demand diverse and complex calculation, need the implementation platform have faster handling ability and flexibility, the traditional solution is difficult to simultaneously meet the requirements of the hd video matrix flexibility and complexity. With the continuous development of high speed digital signal processor technology, integrated with the ARM processor core and programmable digital signal processor (DSP) which is the kernel of DaVinci processor meet the high computational complexity and flexibility requirements, Texas instruments inc. launched specifically for hd multimedia application of DaVinci processor TMS320DM6467T has obvious advantages.

For video surveillance personnel demand for high-definition digital video matrix based on network, this thesis mainly designs and implements a high-definition digital video matrix based on TMS320DM6467T platform. This paper first introduced the characteristics of the platform and digital video matrix card hardware scheme, designed and implemented network high-definition digital video matrix card embedded system software. Thesis research content includes:

- 1, Study the TMS320DM6467T embedded platform development approach, and builds the development and debugging environment.

2, Study and implements Linux device driver based on TMS320DM6467T platform, mainly including GS1671 character device driver and V4L2 video device driver which is BT.1120 inputed.

3, Analyze the functional requirements of the matrix card system. According to the functional requirements of embedded system software has carried on the overall design, do module partition, and detailed design. Including audio and video caputre, video processing, audio processing, network streaming media transmission, the hardware monitoring alarm and input signal detecting.

Designed a card which meets the requirements of the high definition digital video matrix card. Designed for high definition video matrix based on the network provides a reference scheme. The product can be used to broadcast television after the implementation, video monitoring field, the realization of video on demand, video recording, storage, signal switching, etc.

Keywords: High-Definition Video Matrix, TMS320DM6467T,Digital Signal Processor, Device Driver.

目 录

第一章 绪论	1
1.1 课题的背景及意义	1
1.2 课题的研究目标	3
1.3 论文的章节安排	3
第二章 TMS320DM6467T 平台设计及开发环境搭建	4
2.1 TMS320DM6467T 平台介绍	4
2.2 矩阵卡系统硬件功能设计	6
2.3 矩阵卡嵌入式软件系统分层	8
2.4 TMS320DM6467T 软件开发及调试环境搭建	9
2.4.1 安装工具软件及开发平台软件	9
2.4.2 安装配置网络文件系统	9
2.4.3 设置交叉编译工具	11
2.4.4 编译 MontaVista Linux 内核	11
2.4.5 编译 Dv sdk 软件开发包	12
2.5 本章小结	14
第三章 矩阵卡设备驱动程序设计	15
3.1 嵌入式设备驱动程序简介	15
3.1.1 嵌入式设备驱动程序的作用	15
3.1.2 嵌入式设备驱动的分类	15
3.1.3 嵌入式设备驱动程序与其它部分的关系	16
3.1.4 嵌入式设备驱动程序使用	16
3.1.5 嵌入式设备驱动程序的调试	18
3.2 TMS320DM6467T 平台 SPI 适配器驱动和 GS1671 驱动	19
3.2.1 SPI 适配器简介	19
3.2.2 SPI 适配器配置流程	19
3.2.3 SPI 适配器驱动的实现	20
3.3 GS1671 设备驱动的设计	22
3.3.1 GS1671 功能简介	22

3.3.2 GS1671 设备驱动的实现.....	24
3.4 基于 V4L2 的视频采集设备驱动设计.....	25
3.4.1 VPIF 接口简介.....	25
3.4.2 Linux 视频设备驱动 V4L2 简介.....	26
3.4.3 视频采集设备驱动的实现.....	26
3.4.4 基于 V4L2 接口视频采集应用程序.....	30
3.5 本章小结.....	33
第四章 矩阵卡软件设计与实现.....	34
4.1 视频矩阵卡的应用场景.....	34
4.2 视频矩阵卡的软件功能需求分析.....	34
4.3 软件模块划分.....	35
4.4 命令接收管理模块.....	36
4.5 视频采集编码发送模块.....	38
4.5.1 音视频采集.....	38
4.5.2 基于 DSP 核的音视频处理.....	42
4.5.3 音视频流传输.....	44
4.5.4 音视频组管理.....	51
4.5.5 辅助设备的管理.....	51
4.5.6 音视频报警功能.....	53
4.5.7 命令执行模块.....	54
4.6 系统监视模块.....	55
4.7 配置模块.....	55
4.8 软件测试.....	55
4.9 本章小结.....	59
第五章 总结.....	60
5.1 全文总结.....	60
致 谢.....	61
参考文献.....	62

第一章 绪论

1.1 课题的背景及意义

随着科技的发展和人们生活水平的提高，在生活和工作中遇到的需要视频捕获并播看的场合越来越多，但往往是视频捕获设备很多，大屏显示或者播放设备很少。为了观看多种视频捕获的设备的输出，必须对显示设备的输入视频信号进行重新选择，但人们通常不愿意到机房在密密麻麻的电缆中重新插拔视频信号线，而是借用视频矩阵对输入的视频信号进行切换选择。视频矩阵产生就是为了解决了视频多输入单输出的切换问题。

视频矩阵指的是能够实现将任意端口输入的视频信号，经过设备内部的处理，然后输出到任意指定的端口来进行传输，从而达到在不进行线缆插拔动作的前提下就可以进行视频信号的任意输入输出端口切换的仪器设备。

在视频矩阵产品发展的早期，使用的是模拟视频矩阵。模拟视频矩阵主要采用单片机或更复杂的芯片以控制模拟开关的方式来实现视频矩阵的信号切换。模拟视频矩阵存在运行稳定，操作简单，成本较低等优点，但是也存在很多不足：

- 1，模拟视频矩阵仅仅适合于小范围的视频信号传输。即使采用视频光端机，也经常局限于同一座城市内；
- 2，模拟视频矩阵系统的扩展能力差。对已经建好的视频传输系统，如果要增加新的视频输入，操作过程十分麻烦，往往是牵一发而动全身，新的设备因为各种原因也很难添加到原有的系统之中；
- 3，在大型系统中，往往线缆分布复杂，造成施工困难，安装调试复杂；
- 4，模拟矩阵系统升级十分困难，各个厂家产品之间往往兼容性不好；
- 5，不能实现视频矩阵的网络化管理和远程监控。

因为模拟视频矩阵系统存在以上的问题，视频矩阵系统迫切需要使用数字化，网络化的视频矩阵系统代替现有的模拟视频矩阵系统。

另一方面，传统的低分辨率视频信号存在图像质量差，内容分辨不清的问题，人们迫切需要图像质量更高的高清视频来替代现有的 QCIF, CIF, D1 视频。广电和专业的高清视频捕获设备输出端通常是高清数字分量串行接口（HD-SDI）。

高清数字分量串行接口 (HD-SDI) 是相对于数字分量串行接口 (SDI) 而言的。国际电信联盟无线电通信部 (ITU-R) 在 1994 年发布了 BT. 656-2 建议书, 该建议书吸纳了欧洲广播联盟 (EBU) 的 Tech. 3267 标准与美国电影电视电视工程师协会 (SMPTE) 的 259M 标准中定义的新型串行数字接口, 该接口采用 10 比特传输与非归零反向 (NRZI) 编码。在传送 ITU-R BT. 601 (A 部分) 4: 2: 2 级别信号时, 其时钟还率为 270 Mb/s, 这就是数字分量串行接口 (SDI)。之后, 美国电影电视电视工程师协会 (SMPTE) 在 292M 标准中定义了时钟频率高达 1. 5 Gb/s 级别的高清串行数字接口, 对应国际标准为 ITU-R BT . 1120。

速率高达 1. 5Gb/s 高清视频信号, 对其进行采集压缩处理具有很高的运算量。因此实现数字视频矩阵传输来传输高清视频信号, 对数字视频矩阵中央处理器单元的运算能力有很高的要求, 如果使用多片中央处理器则存在开发难度大, 成本高的问题。

网络化是指通过以太网或者其它网络协议进行传输, 而直接传输高清视频信号需要巨大的网络带宽, 消耗巨大的网络流量。因此必须对待网络传输的高清数字视频信号进行编码压缩, 再进行网络传输。

H. 264 是一种先进的视频压缩算法。虽然 H. 264 具有较多的优点, 但是采用 H. 264 压缩仍然需要较大的运算量。

高速 DSP 芯片技术的发展使采用高清数字分量串行接口的数字视频矩阵实现难度大为降低。德州仪器公司推出的达芬奇技术系列 DSP 芯片, 是专门针对数字多媒体应用而定制的系统解决方案。达芬奇处理器的优势:

1, 显著降低成本。达芬奇芯片系列多媒体片上系统 (soc) 集成了可编程数字信号处理器 (DSP) 内核, ARM 处理器内核和高清视频协处理器。专门针对数字视频进行了精心优化, 提供业界领先的性能。

2, 完善的开发工具。达芬奇平台提供了完整的开发系统, 参考设计以及全面的 ARM/DSP 系统级集成开发环境。

在综合比较后, 选择使用德州仪器公司的 TMS320DM6467T 多媒体片上系统 (soc) 作为开发数字视频矩阵卡的硬件平台。TMS3206467T 平台特点:

- 1, 集成了 500MHz 的 ARM926EJS 内核与 1GHz 的 C64x+DSP 内核;
- 2, 采用高清视频/影像协处理器 (HD—VICP) 和视频数据转换引擎 (VDCE);
- 3, 集成了千兆以太网控制器;

以上资源保障了 TMS320DM6467T 硬件平台可以作为实现高清数字视频矩阵卡的合适硬件平台。

进行本论文的意义：

高清数字视频矩阵卡可以应用在包括广播电视、电视会议、安防监控等领域，实现高清数字视频信号的信号分析，视频信号内容的存储前处理和视频输出的切换等功能。

1.2 课题的研究目标

设计实现一种高清串行数字输入接口的视频矩阵卡，输入一路高清-串行数字视频信号（HD-SDI），系统对音视频进行压缩，输出网络传输流。

1.3 论文的章节安排

第一章绪论，简要介绍了课题产生的相关的背景。

第二章简单介绍 TMS320DM6467T 平台的特性，基于 TMS320DM6467T 的矩阵卡硬件系统设计方案和基于 Linux 嵌入式操作系统的 TMS320DM6467T 平台软件系统的分层结构。搭建了 TMS320DM6467T 平台的软件的开发环境和调试环境。

第三章介绍 TMS320DM6467T 平台下系统底层驱动程序的设计和实现。主要包括 GS1671 控制功能驱动和视频输入驱动程序。

第四章对基于 TMS320DM6467T 编码传输系统进行了需求分析，总体设计，模块划分，并对各个模块进行了详细设计，最后进行了软件整体测试。

第五章对本文的系统设计工作进行了总结并对下一步工作提出了展望。

第二章 TMS320DM6467T 平台设计及开发环境搭建

2.1 TMS320DM6467T 平台介绍

TMS320DM6467T 是基于达芬奇技术的高性能的多媒体片上处理系统 (soc)，能够满足下一代嵌入式网络多媒体编码，解码和应用处理的需求。TMS320DM6467T 包含健壮的操作系统支持，丰富的用户接口，高性能的处理能力，并且通过多种解决方案提供了长时间的电池工作寿命。

TMS320DM6467T 芯片内部集成了高性能的 ARM 核心 (ARM926EJ-S)，数字信号处理器核心 (TMS320C64x+) 和两个高清视频/图像协处理器，通过资源交换控制单元与外部设备接口控制器进行数据的交换，如下图 2-1 所示。

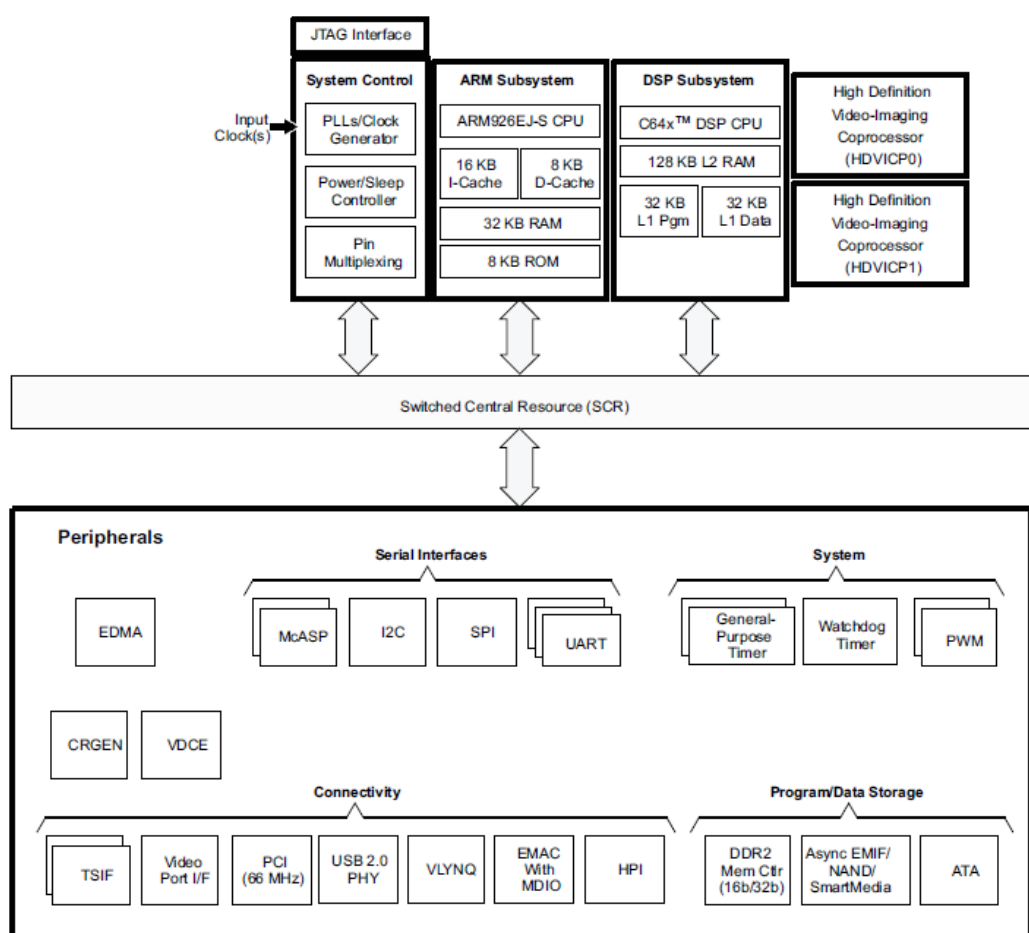


图 2-1 TMS320DM6467T 的功能块图

作为双核架构，TMS320DM6467T 兼有数字信号处理器和精简指令集处理器的优点。其中精简指令集处理器（ARM926EJ-S），最高主频 500MHz；数字信号处理器（TMS320DM64x+），最高主频为 1GHz。

精简指令集处理器（ARM926EJ-S）作为 TMS320DM6467T 芯片的功能控制单元，用于对设备的配置和控制操作，包括对数字信号处理器（TMS320DM64x+）核心，外部存储器和其它丰富的外部接口设备的控制。精简指令集处理器（ARM926EJ-S）具有以下特点：

- 1，支持流水操作，32 位或 16 位 ARM 指令集，能够处理 32 位，16 位，或者 8 位位宽的数据；
- 2，支持保护模式操作，内部包含一个协处理器单元（CP15）；
- 3，支持查找表缓存的数据和程序内存管理单元（MMU）；
- 4，独立的 16K 字节指令片内缓存和 8K 字节的数据片内缓存。这两者均支持与虚拟索引虚拟标签（VIVT）的四路关联。

TMS320C64x+数字信号处理器是 TMS320C6000 系列数字信号处理器中性能最高的，建立在德州仪器公司开发的第二代高性能，超长指令集架构的增强版本之上，这些使得 TMS320C64x+数字信号处理器成为数字多媒体视频应用的理想选择。C6000 系统 DSP 主要特点有：

- 1，支持超长指令集，拥有八个高度独立的功能单元。其中包括六个算术运算单元（ALUS）和两个乘法运算单元。在一个指令周期内最高可执行八条指令，性能是其它典型数字信号处理器的十倍。使得设计者可以在较短的开发周期内开发出高效精简指令集类似的代码。
- 2，指令封包技术。该技术使得执行一条代码的结果等效于八条串行或者并行执行的命令。该技术可以减小代码文件的大小，使用较少的取指令动作，降低了系统的功耗。
- 3，绝大多数的指令都可以条件执行。减小了程序中分支的数量，增加了程序执行的并行度，从而得到持续较高的性能。
- 4，两级缓存结构：一级程序（L1P）缓存大小为32K字节。一级数据（L1D）缓存大小为32K字节。二级缓存大小为128K字节。
- 5，支持第二代双数据速率静态随机存储器（Double Data Rate 2-Synchronous Dynamic Random Access Memory, DDR2-SDRAM）。比传统平台所搭载的普通SDRAM数据吞吐量提高一倍以上。配合芯片内部集成的增强型数据直接访问（EDMA）功能，实现更方便快速的存储器读写。

相比之前的数字信号处理器核心，TMS320DMC64x+核心又引入了一些新的特性：

1，支持软件流水循环（SPL00P）技术。软件流水循环（SPL00P）技术是通过借助中央处理单元（CPU）内部一个小的指令存储区来帮助创建多个并行执行迭代操作的软件流水。软件流水循环（SPL00P）存储区减少了软件流水的代码尺寸关联，而且软件流水循环（SPL00P）存储区中的循环数是完全可中断的。

2，更加紧凑的指令。C6000设备的原始指令大小是32位，许多常见的指令，例如MPY, AND, OR, ADD, SUB指令，在C64x+编译器限制使用某些寄存器文件中的寄存器时，可以被表示为16位指令。这种指令宽度的压缩是由代码生成工具执行的。

3，指令集增强。如前面叙述的，有很多新的指令，例如32位乘法，复数乘法，封包，分类，位操作和32位高斯域乘法。

4，异常处理。目的是为了帮助程序员快速定位并解决程序的漏洞。C64x+CPU可以检测异常，响应异常，这两个功能的触发源都可以是来自内部检测源代码（例如非法操作码）或者系统时间（例如一个开门狗系统的时间溢出）。

5，特权模式。定义了用户模式和特权模式，允许操作系统给一个敏感资源的操作模式保护。本地内存被划分为多个页面，每个页面都有读、写和执行权限。

6，时间戳计数器。主要针对实时操作系统（RTOS）的健壮性，在CPU中有一个不同步的时间戳计数器实现，该时间戳计数器对系统阻塞并不敏感。

2.2 矩阵卡系统硬件功能设计

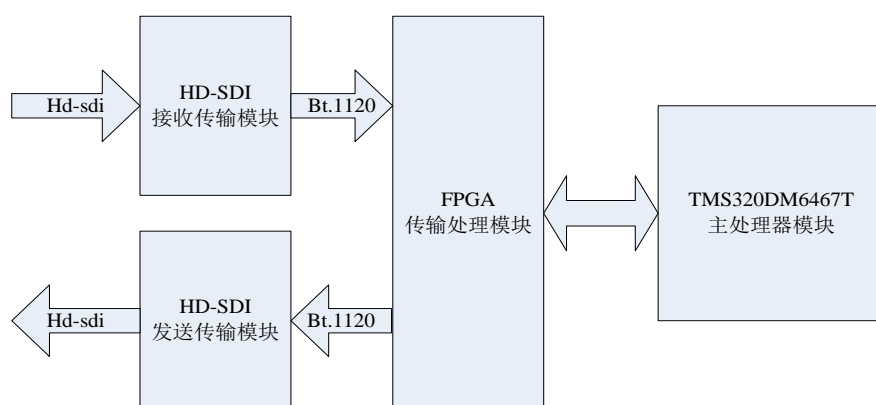


图 2-2 硬件各个模块关系图

硬件电路板功能有 HD-SDI 接收/发送处理模块，FPGA 传输处理模块，TMS320DM6467T 主处理器模块三个模块组成，模块关系如图 2-2 所示。

矩阵卡内部数据流向和控制关系描述为：

- 1, 输入 HD-SDI 视频信号到 GS1671 芯片, GS1671 对音视频信号解嵌后, 输出并行 BT. 1120 视频信号和四路 IIS 音频信号。
- 2, 输入并行 BT. 1120 视频信号到现场可编程门阵列 (FPGA), FPGA 旁通输出 BT. 1120 信号给其它板卡或者直接输出到 TMS320DM6467T 芯片的视频端接口 (VPIF)。输入四路音频 IIS 信号给 FPGA, FPGA 旁通输出或者直接输出到 TMS320DM6467T 的多通道音频接口 (MCASP)。
- 3, 并行视频信号 BT. 1120 通过视频端接口 (VPIF) 接口输入到 TMS320DM6467T, VPIF 模块对 BT. 1120 信号进行解码, 输出视频原始数据, 通过芯片内部的 EDMA 搬运至 DDR2-SDRAM 中。
- 4, 四路音频 IIS 信号通过 MCASP 接口输入到 DM6467T, mcasp 模块拆分四路音频, 保存到 DDR2 中, 供后续编码处理。
- 5, TMS320DM6467T 芯片通过串行外设接口 (SPI), 通过两个片选信号 CS0 和 CS1, 分别对 GS1671 和 FPGA 进行控制, 实现对 GS1671 芯片和 FPGA 芯片的配置或状态读取和写入。
- 6, EMFI 接口外部连接 NAND 存储芯片, 作为设备的文件系统存储设备。
- 7, IIC 接口连接 RTC 时钟芯片, 获取当前时间。
- 8, IIC 接口连接温度传感器芯片, 监控系统的工作温度, 当温度过高时, 产生报警信号。
- 9, EMAC/MDIO 接口连接千兆网络物理设备芯片, 实现网络流传输。

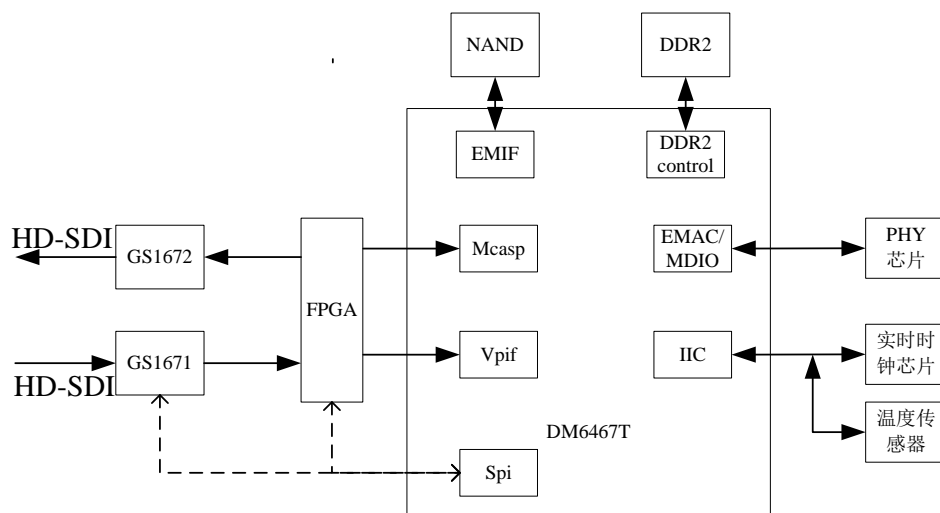


图 2-3 硬件电路板功能块图



图 2-4 设计好的矩阵卡正面

2.3 矩阵卡嵌入式软件系统分层

基于 TMS320DM6467T 平台设计的矩阵卡嵌入式软件系统主要由四部分组成：

1，引导加载程序（BootLoader），包括 RBL, UBL, U-BOOT。引导加载程序用于对硬件的初始化，引导并加载操作系统。

2，操作系统(Operating System)，本项目使用的是 montavistalinux 操作系统。用于提供板卡及外部设备的驱动程序，实现线程机制，进程间通信机制，网络协议栈，提供系统调用接口等。

3，文件系统（root filesystem）。文件系统通常又分为根文件系统和用户文件系统。根文件系统包含系统引导和使其他文件系统得以 mount 所必要的文件。主要由网络文件系统（NFS），YAFFS2（Yet Another Flash File System）文件系统等。用户文件系统包含各种应用程序，通常存放在可插拔存储介质中，如各种闪存盘或硬盘，使用 YAFFS2 或者 FAT 文件系统。

4，应用软件（Application Software）。应用软件是用户可以使用的各种应用程序的集合。是计算机为解决某类问题而设计的程序的集合。

本项目研究的对象包括操作系统，板卡和外部设备驱动。使用了线程机制，进程间通信机制，网络协议栈和系统调用，完成的软件功能作为一个应用软件供用户使用。

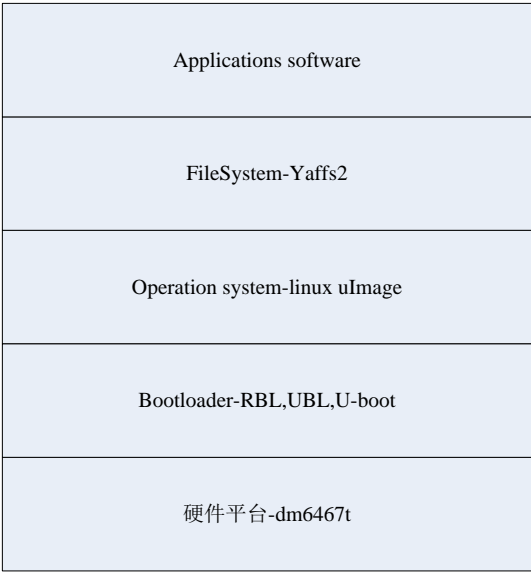


图 2-5 嵌入式系统软件组成

2. 4 TMS320DM6467T 软件开发及调试环境搭建

2. 4. 1 安装工具软件及开发平台软件

在ubuntu主机端，挂载DVEVM开发包光盘，拷贝安装包到目录/tmp。

```
mv1_5_0_#_demo_sys_setuplinux.bin
mv1_5_0_0_demo_lsp_setuplinux_#_#_#_#.bin
dvsdk_setuplinux_#_#_#_#.bin
xdctools_setuplinux_#_#_#_#.bin
bios_setuplinux_#_#_#_#.bin
TI-C6x-CGT-v#.#.#.#.bin
linux_performance_testbench_setup_#.#.#.#.bin
```

按顺序安装linux目标软件，dvsdk软件，codec servers，片上系统分析软件，linux性能测试平台。

2. 4. 2 安装配置网络文件系统

尽管板卡支持flash文件系统，但是挂载网络文件系统到主机端能够更方便的进行调试时的文件更新。调试文件完成测试后可以保存应用到板子的flash文件系统中。挂载网络文件系统之前，在主机端必须建立一个网络文件系统导出。通过

使用网络文件系统服务器，使导出的文件系统包含目标文件系统和可执行程序。
按照下面的操作，就可以使用网络文件系统服务器导出一个文件系统服务：

1, 在主机端登陆一个用户账号。

2, 执行下面的命令准备存放文件系统的位置，例如：

```
Host $ cd /home/username
```

```
Host $ mkdir -p dm6467t/nfs
```

```
Host $ cd dm6467t/nfs
```

3, 切换到主机的超级用户，

```
Host $ su root
```

4, 执行下面命令来创建目标文件系统的一个拷贝，将拷贝的权限设置为username, 这样才允许操作共享区域。如果目标文件系统没有被安装在/opt/mv_pro_5.0目录，那么使用真正安装的目录。

```
Host$ cp -a /opt/mv_pro_5.0/.../target/* .
```

```
Host $ chown -R username opt
```

5, 编辑主机端etc目录下的exports文件，导出目标文件系统，

```
/home/username/dm6467t/nfs*(rw,no_root_squash,no_all_squash,sync)
```

6, 切换到超级用户执行下面的命令来更新配置文件，然后重启nfs。

```
Host $ sudo /usr/sbin/exportfs -av
```

```
Host $ sudo /sbin/service nfs restart
```

使用exportfs -rav，重新导出所有的目录，使用/etc/init.d/nfs status来确认网络文件系统服务器当前正在运行。

7, 验证服务器网络防火墙已经关闭，

```
Host $ /etc/init.d/iptables status
```

如果防火墙正在运行，那么关掉防火墙，

```
Host $ /etc/init.d/iptables stop
```

测试共享文件系统：

1, 获取linux host的IP地址，查看eth0以太网的IP地址。

2, 打开终端窗口连接到RS232串行口，给电路板加电。取消自动启动过程。

3, 在终端设置下面的环境变量：

```
# setenv nfshost 192.168.0.21
```

```
# setenv rootpath /home/username/workdir/filesys
```

```
# setenv bootargs mem=120M root=/dev/nfs ninitrd rw ip=dhcp
```

```
nfsroot=$(nfsroot):$(rootpath),nolock    console=ttyS0,115200n8    mem=120M
davincihd_capture.channel10_numbuffers=4
```

4, 保存环境变量。

```
#saveenv
```

5, 重启电路板。 # boot

之后就可以使用root账号登陆网络文件系统进行操作。

2.4.3 设置交叉编译工具

为了在主机端编译出目标机可以运行的程序和库, 必须安装目标机器交叉编译工具链, 主机端默认不能使用交叉编译工具, 必须对交叉编译工具进行设置:

1, 使用普通用户账号登陆主机端。

2, 设置环境变量PATH为MontaVista工具链所在位置。例如, 如果安装MontaVista LSP选择的是默认安装, 那么在shell资源文件中添加一个新的定义:

```
PATH="/opt/mv_pro_5.0/montavista/pro/devkit/arm/v5t_le/bin:
/opt/mv_pro_5.0/montavista/pro/bin:
/opt/mv_pro_5.0/montavista/common/bin:$PATH"
```

如果没有安装在默认位置/opt/mv_pro_5.0, 那么使用安装的位置替代。

3, 修改资源文件后, 使用命令来执行新添加的命令:

```
Host $ source ~/.bashrc
```

2.4.4 编译MontaVista Linux内核

MontaVista Linux 是MontaVista公司为德州仪器TMS320DM6467T芯片定制的嵌入式设备操作系统, 具有较高的稳定性和实时性能。编译步骤:

1, 以普通用户登录主机端。

2, 设置dvsdk目录Rules.make文件中的PLATFORM变量为dm6467。

3, 建立MontaVista Linux支持包在本地工作目录的拷贝。拷贝内容包括montavista linux.2.6.18 内核和各种常用外部设备驱动。

```
Host $ cd /home/username
```

```
Host $ mkdir -p wrokdir/lsp
```

```
Host $ cd work/lsp
```

```
Host $ cp -R /opt/mv_pro_5.0/montavista/pro/devkit/lsp/ti-davinci.
```

4, 使用下面的命令来使用默认配置, 其中CROSS_COMPILE指定了交叉编译器。

```
Host $ cd ti-davinci/linux-2.6.18_pro500
```

```
Host $ cd make ARCH=arm CROSS_COMPILE=arm_v5t_le-  
davinci_dm6467_defconfig
```

5, 如果需要修改内核选项, 使用配置命令” make menuconfig” 或者 “make xconfig”。使用下面的命令可以验证默认内核选项:

```
Host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- checksetconfig
```

6, 如果需要使能对于片上系统分析器的跟踪功能, 执行下面步骤, 否则, 直接执行第7步。

步骤1 使用命令跳转到ARM的配置菜单:

```
Host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- menuconfig
```

步骤2 跳转到 Device Driver->Filesystems->Pseudo Filesystems,

步骤3 设置Relayfs文件系统支持为编译到内核中,

步骤4 返回到主菜单的通用设置, 跳转到General Setup,

步骤5 设置Linux Trace Toolkit Support选项为编译进内核,

步骤6 选择Exit来保存以上修改的配置, 并推出配置选项。

7, 使用下面的命令来编译内核:

```
Host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

8, 如果内核使用了任何可以加载模块 (对应到内核配置选项中的使用<M>选中的模块), 使用下面的命令来重新编译并安装这些模块:

```
Host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
```

```
Host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-  
INSTALL_MOD_PATH=/home/username/dm6467t/nfs modules_install
```

9, 使用下面的命令来拷贝内核镜像文件uImage到tftp目录, uboot可以使用tftp方式来下载内核镜像。

```
Host $ cp /opt/linux-2.6.18_pro500/arch/arm/boot/uImage /tftpboot
```

```
Host $ chmod a+r /tftpboot/uImage #追加更改内核映像文件的使用权限为所有用户  
都可以读。
```

2.4.5 编译Dv sdk软件开发包

Dv sdk软件开发包是TMS320DM6467T平台算法开发及接口调用的软件开发包, 必

须进行正确的配置使用才能支持数字信号处理器。编译dvSDK软件的步骤如下：

- 1, linux内核必须被首先编译。
- 2, 切换目录到/opt/dvSDK_2_00_00_22。
- 3, 按照平台及实际的安装目录编辑dvSDK目录下的Rule.make文件。如果使用

默认的安装路径，只需要修改下面的内容：

设置PLATFORM变量到真实的平台，可以设置为PLATFORM=dm6467。

设置DVSDK_INSTALL_DIR到dvSDK安装的路径：

```
DVSDK_INSTALL_DIR=/home/username/2_00_00_22
```

分别修改下列的变量到实际的路径：

```
XDC_INSTALL_DIR=/opt/dvSDK_2_00_00_22/xdctools_3_10_05_61
```

```
FC_INSTALL_DIR=/opt/dvSDK_2_00_00_22/framework_components_2_23_01
```

```
BIOS_INSTALL_DIR=/opt/dvSDK_2_00_00_22/bios_5_33_03
```

设置LINUXKERNEL_INSTALL_DIR按如下方式被定义：

```
LINUXKERNEL_INSTALL_DIR=/home/username/workdir/lsp/ti-davinci/linux-2.
```

6.18_pro500

设置MVT00L_DIR指向MontaVistaLinux工具的路径：

```
MVTOOL_DIR=/opt/mv_pro_5.0/montavista/pro/devkit/arm/v5t_le
```

确认EXEC_DIR指向网络文件系统导出的路径：

```
EXEC_DIR=/home/username/workdir/filesys/opt/dvSDK/$(PLATFORM)
```

4, 如果已经在Rules.make所在目录，执行下面的命令来编译dvSDK例子应用并安装到EXEC_DIR。

```
Host $ make clean;make;make install
```

- 5, 重启系统后进行测试。

在编译dvSDK后，为了使用新编译的固件，可以使用tftp下载固件到板子上。过程如下：

- 1: 给电路板加电，在RS232串口终端按任意键停止使用自动启动过程。
- 2: 设置下面的环境变量。

```
# setenv bootcmd 'dhcp;bootm'
```

```
# setenv serverip '192.168.0.120'
```

```
# setenv bootfile 'uImage'
```

```
# setenv bootargs 'mem=150M root=/dev/hda1 rw noinitrd console=ttyS0,115200n8  
ip=dhcp davincihd_capture,channel0_numbuffers=4'
```

重新启动板卡，即可下载编译好的固件到电路板上进行测试。

2.5 本章小结

本章首先对达芬奇处理器 TMS320DM6467T 做了简单的介绍，然后介绍了矩阵卡硬件系统的设计和矩阵卡嵌入式软件的构成，最后搭建了 TMS320DM6467T 平台的嵌入式软件开发开发和调试环境。

第三章 矩阵卡设备驱动程序设计

3.1 嵌入式设备驱动程序简介

3.1.1 嵌入式设备驱动程序的作用

嵌入式设备驱动是嵌入式设备软件和硬件的交叉点，在嵌入式系统的软件开发中具有重要的作用。嵌入式设备驱动程序直接关系着设备功能的实现和系统运行的稳定性。编写嵌入式设备驱动程序往往是困难且耗时的，程序员首先需要认真阅读芯片的硬件设备手册，掌握硬件芯片的操作控制过程。由于设备驱动程序在操作系统内核态运行，对设备驱动程序的稳定性和性能的要求比较高，所以要求设备驱动开发者必须具有良好的程序设计经验和技巧，否则可能会导致操作系统意外当机。

3.1.2 嵌入式设备驱动的分类

随着嵌入式系统外部功能设备的种类和数量日益增多，嵌入式设备驱动程序的种类和数量也越来越多。按照外设的特性和操作系统的功能划分，Linux 操作系统将设备分为三个基础大类：字符设备，块设备和网络设备，对应的驱动也分为字符设备驱动程序，块设备驱动程序，网络设备驱动程序。

字符设备是指那些必须以串行顺序依次对设备进行访问的功能模块和外部设备，如触摸屏，鼠标，看门狗，IIC 设备，SPI 设备。

块设备是指可以用任意顺序进行访问，以块为单位进行访问和操作的设备，如硬盘，软盘。

网络设备面向数据包的接受和发送而设计，如网卡，CAN 总线。

在 linux 设备驱动程序的设计中，这三者之间存在异同点和交叉点。字符设备和块设备的使用对于用户而言，都使用了文件系统操作接口 `open()`, `read()`, `write()`, `close()`。网络设备不对应文件系统接口，没有使用文件系统的操作接口。另外字符设备和块设备并没有明显的界限，如 FLASH 设备符合块设备的特点，但是为了操作方便，仍然可以把它作为一个字符设备来进行访问。另一方面字符设备没有经过系统的快速缓冲，块设备经过系统的缓冲，对块设备的访问不必向字

符设备那样每次都操作 I/O 接口。而 usb 接口的 wifi 设备, spi 接口的 can 总线设备, 是字符设备和网络设备两种设备类型的组合。因此这三种设备类型又是紧密联系的。

3.1.3 嵌入式设备驱动程序与其它部分的关系

下面简单介绍应用程序, 操作系统, 驱动程序和硬件这几部分之间的关系。应用程序负责实现嵌入式设备用户需要的业务逻辑, 可以理解为具体的策略。操作系统负责管理软硬件资源的管理, 例如进程调度和资源分配等。驱动程序按照操作系统的接口要求, 封装对硬件设备的操作, 提供硬件设备所有的功能集合, 可理解为机制。硬件负责执行具体的电气特性变换操作。应用程序, 操作系统, 驱动程序, 硬件的关系如下图:

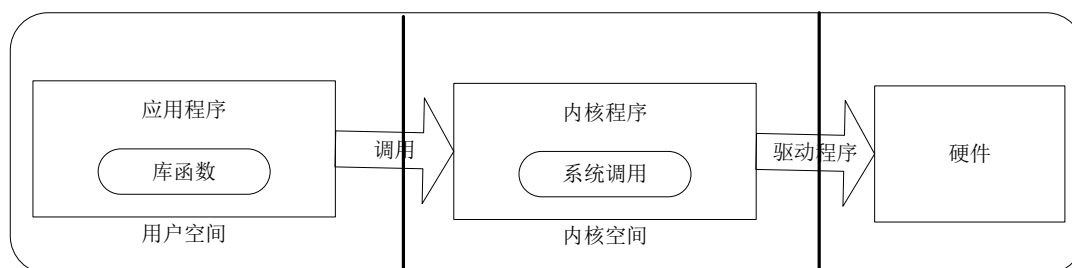


图 3-1 应用程序, 操作系统, 驱动程序, 硬件关系图

3.1.4 嵌入式设备驱动程序使用

本论文讨论的嵌入式设备驱动程序均基于 Linux 操作系统。Linux 设备驱动程序使用简单易用的 C 语言进行编写, 可以被编译进内核作为内核的一部分被操作系统加载, 也可以编译为单独的模块, 在应用程序脚本中进行加载和卸载, 驱动程序模块一旦被加载, 它就和集成进内核中其它的部分完全一样。为了方便调试, 通常将设备驱动程序编译为模块的方式来使用。使用模块方式来加载和卸载驱动程序除了有改变驱动程序不需要重启操作系统的优点外, 还减少了操作系统的文件大小。

设备驱动程序的接口包括:

模块加载函数: `module_init(initialization_function)`。当在 linux 终端通过 `insmod` 命令或者 `modprobe` 命令加载设备驱动模块时, 模块的加载函数会自动被内核执行, 完成本设备驱动模块的相关初始化工作。

模块卸载函数: `module_exit(clean_function)`。当通过 `rmmod` 命令来卸载模块时, 设备驱动模块中的模块卸载函数会自动被内核执行, 实现与加载函数相反的功能。

许可证声明: `module_license("Dual BSD/GPL")`。描述了该设备驱动模块的许可权限, 如果没有声明为 `LICENSE`, 那么模块被加载时, 将受到内核被污染的警告。可选则 `LICENSE` 包括 "GPL", "GPL v2", "Dual BSD/GPL", "Proprietary" 等, 大部分的 内核模块应遵循 GPL 兼容许可权。使用中最常见的是以 `MODULE_LICENSE("Dual BSD/GPL")` 语句声明模块采用 BSD/GPL 双 `LICENSE`。

模块参数: `module_param(param_name, param_type, param_wr)`。当设备驱动模块被加载的时候可以被传递给它的值, 它本身对应模块内部的全局变量。

模块导出符号: `export_symbol(export function)`。内核模块可以到处函数或者变量 (统称为 `symbol`), 这样其它模块可以使用本模块中的变量和函数。

模块作者等信息: `module_author(author)`, `module_description(descript)`, `module_version(version_string)`。声明该设备驱动模块的作者, 功能描述, 版本等信息。

使用命令 `lsmod` 或者 `cat /proc/modules` 可以查看当前操作系统已经加载的设备驱动模块, 如下图, 分别描述了模块的名称, 大小, 正被那个模块使用。

```
Host $ lsmod
Module                Size  Used by
snd_intel8x0           19595  1
snd_ac97_codec         79136  1  snd_intel8x0
ac97_bus               710    1  snd_ac97_codec
snd_pcm                47226  2  snd_intel8x0,snd_ac97_codec
snd_seq                35463  0
....
```

如果使用 `cat /proc/devices`, 则多出在 RAM 中的地址一项。

```
Host $ cat /proc/modules
modules  mounts
Host $ cat /proc/modules
snd_intel8x0 19595 1 - Live 0xe1dd8000
snd_ac97_codec 79136 1 snd_intel8x0, Live 0xe1dad000
ac97_bus 710 1 snd_ac97_codec, Live 0xe1d87000
```

```
snd_pcm 47226 2 snd_intel8x0,snd_ac97_codec, Live 0xe1d6e000
snd_seq 35463 0 - Live 0xe1d46000
```

....

使用命令 `cat /proc/devices` 可以查看当前系统有那些设备已经被配置。

```
Host $ cat /proc/devices
```

Character devices:

```
1 mem
```

```
116 alsa
```

```
254 rtc
```

Block devices:

```
259 blkext
```

```
7 loop
```

```
8 sd
```

```
11 sr
```

```
65 sd
```

3.1.5 嵌入式设备驱动程序的调试

开发设备驱动模块通常需要对编写的驱动程序进行调试。调试嵌入式设备驱动常用的方法有：

- 1, 使用 kgdb 工具进行调试。首先给内核打上 kgdb 补丁, 重新编译内核及驱动模块后, 主机上的 gdb 可与目标机的 kgdb 通过串口或者网络进行通信。

- 2, 使用仿真器, 仿真器可以直接连接到 TMS320DM6467T 芯片的 arm 端 jtag 接口, 这样 gdb 就可以通过与仿真器的通信来控制 TMS320DM6467T 芯片的运行, 调试时可以执行断点, 单步执行等操作。

- 3, 在目标板 TMS320DM6467T 平台上通过 `printk()`, `oops`, `strace` 等软件方法进行输出信息进行调试, 这些方法不具备修改数据结构, 断点操作, 单步执行等功能。内核模块中调试信息输出函数是内核函数 `printk()` 而非用户空间函数 `printf()`, 除了 `printk()` 函数可以定义输出级别外, 两个函数用法完全一致。

本项目包含多个设备驱动程序, 这里选取视频采集相关的设备驱动程序进行简单介绍。TMS320DM6467T 使用 SPI 适配器通过 SPI 总线对 GS1671 进行功能配置,

GS1671 输出 BT. 1120 经过 FPGA 送入 TMS320DM6467T 的 VPIF 接口。涉及的驱动程序分为 spi 适配器驱动程序, gs1671 驱动程序和 v4l2 视频设备驱动程序。

3.2 TMS320DM6467T 平台 SPI 适配器驱动和 GS1671 驱动

3.2.1 SPI 适配器简介

SPI(Serial Peripheral Interface, 串行外设接口)总线是一种同步串行外设接口, 在MCU与各种外部设备之间以串行的方式进行通信来双向传输信息。典型的外部设备如EEPROM, 数模转换器, 显示驱动设备, 移位寄存器。

TMS320DM6467T中的SPI总线可以选择使用3线, 4线或者5线的接口模式与外部设备进行数据传输。SPI总线以主从模块进行通信。当工作在主模式时, 支持多个片选操作, 可以同时支持两个SPI从设备。也可以工作在从模式。SPI总线的传输速率和传输位宽可以通过寄存器进行设置。

3.2.2 SPI 适配器配置流程

TMS320DM6467T芯片内部集成有一个SPI适配器, SPI适配器的初始化过程为:

- 1, 通过清除SPI全局控制寄存器0(SPIGCR0)的复位信号位(RESET), 使得SPI模块位于复位状态。
- 2, 设置复位信号位(RESET)为1, 使用SPI模块离开复位状态。
- 3, 设置SPI全局控制寄存器1(SPIGCR1)中的时钟模式位(CCLKMODE)和主模式位(MASTER), 使SPI工作在需要的主从和时钟模式。
- 4, 通过设置SPI管脚控制寄存器0(SPIPC0)中的相应位, 使能SPI_SIM0, SPI_SOMI, SPI_CLK和必要的片选管脚(SPI_CS0, SPI_CS1, SPI_EN)。
- 5, 通过SPI数据格式寄存器(SPIFMTn)配置符合要求的传输数据格式:
 - a, 通过PRESCALEn位配置时钟预缩放值。
 - b, 通过CHARLENN位配置传输字大小。
 - c, 通过PHASEn位和POLARITYn位设置SPI时钟信号。
 - d, 通过SHIFTDIRn位设置移位方向。
- 6, 通过SPI移位寄存器1(SPIDAT1)的DFSEL位选择合适的预配置数据格式。
- 7, 如果SPI_CS工作在四管脚工作模式, 配置下面这些属性:
 - a, 通过SPI延时寄存器(SPIDELAY)中的T2CDELAY位或者C2TDELAY位配置

片选信号的建立时间或者保持时间。

b, 选择合适的片选数字, SPIDAT1寄存器中的CSNR域设置决定了数据传输时那个片选信号被使能。

c, 当片选线无效时, 通过SPI默认片选寄存器 (SPIDEF) 中的CSDEFn位来设置默认的片选信号管脚电平。

8, 如果使用带SPI使用的4根管脚通信模式, 配置下面这些内容:

a, 通过设置SPIPC0寄存器的ENAFUN位到1, 是的SPI_EN管脚有效。

b, 通过配置SPI中断寄存器 (SPIINT) 中的ENABLE_HIGHZ位来设置SPI控制器未使用时的SPI_EN管脚状态。

9, 如果SPI工作在5根管脚通信模式, 按6和7操作。

10, 通过SPI中断寄存器 (SPIINT) 使用需要的中断。

11, 通过SPI中断级寄存器中的RXINTLVL位来选择是否映射中断事件到SPINT0或者SPINT1标志。

12, 使用SPIGCR1寄存器中的SPIEN位。

13, 如果使用EDMA来进行数据传输, 安装并使能EDMA通道的传输或者接受, 之后设置SPI中断寄存器中的DMAREQEN位。

14, 使用CPU或者EDMA写数据到SPIDAT1寄存器来启动数据进行传输。

3.2.3 SPI 适配器驱动的实现

SPI适配器驱动是SPI控制器模块被linux操作系统识别并加以控制的支持程序。加载SPI适配器驱动后, 软件就可以控制SPI控制器, 通过执行执行, 可以控制在SPI总线上产生需要的信号, 对挂载在SPI总线上的设备进行读, 写操作。SPI适配器的驱动程序结构可以用下图来描述。

1, 平台总线是操作系统中自带的一种总线设备。按照名字进行设备与驱动的匹配, 匹配后执行对应的处理函数。

该结构体是内核定义的平台总线结构体:

```
struct bus_type platform_bus_type = {
    .name      = "platform",           //总线的名称
    .dev_attrs = platform_dev_attrs,   //总线的属性
    .match     = platform_match,       //总线的匹配方法
    .uevent    = platform_uevent,     //总线卸载事件
```



```
.suspend = platform_suspend,    //总线暂停方法
.resume   = platform_resume,    //总线恢复方法
};
```

平台总线的match方法规定了平台总线上设备与驱动的匹配方法。

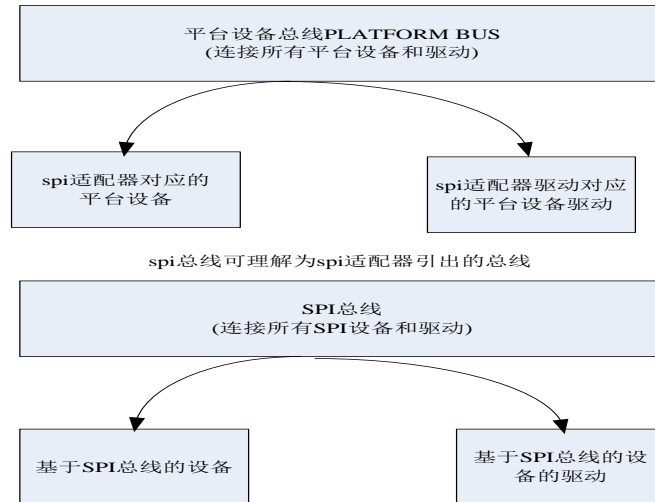


图3-2 SPI适配器结构

2, 注册平台设备。spi适配器作为一个硬件设备随着系统的启动,注册在平台总线。spi适配器的结构体,包含了设备的名字和拥有的资源。

```
static struct platform_device davinci_spi_device = {
    .name = "dm_spi",    //平台设备的名字叫dm_spi
    .id = 0,
    .num_resources = ARRAY_SIZE(dm646x_spi_resources),
    .resource = dm646x_spi_resources, //平台设备使用的资源
};
```

3, 注册平台驱动。注册spi适配器驱动程序,指定了该设备驱动是一个平台设备驱动,该驱动包含了探测到设备时的处理函数和卸载时的处理函数。

```
static struct device_driver davinci_spi_driver = {
    .name = "dm_spi",    //驱动程序名字
    .bus = &platform_bus_type, //所属的总线
    .probe = davinci_spi_probe, //探测到设备时处理函数
    .remove = __devexit_p(davinci_spi_remove), //卸载时处理函数
};
```

4, 当SPI设备和SPI驱动匹配成功后, SPI驱动使用SPI设备的硬件资源建立起

一套操作SPI硬件的接口。

```
struct spi_master {
    struct class_device cdev;
    s16      bus_num;           //适配器的编号
    u16      num_chipselect;    //适配器的片选数量
    /* setup mode and clock, etc (spi driver may call many times) */
    int      (*setup)(struct spi_device *spi);
    int      (*transfer)(struct spi_device *spi,
                        struct spi_message *mesg);
    void      (*cleanup)(const struct spi_device *spi);
};
```

setup函数实现了对SPI控制器按照规定的参数对硬件时序进行设置;

transfer函数实现了SPI数据收发传输的功能;

适配器驱动支持提供了操作SPI总线上信号的方法,但SPI总线的使用必须结合外部连接设备进行,本项目中SPI总线上连接的有FPGA芯片和GS1671芯片。

3.3 GS1671 设备驱动的设计

3.3.1 GS1671 功能简介

GS1671 是一种多速率 SDI 集成接收器,能够实现完整的 SMPTE-292M 和 SMPTE-259C 规格信号的处理,同时也支持旁通功能,将信号传给其它模块进行处理。GS1671 的特性包括:

1, 集成了自适应电缆均衡技术,可以实现前所未有的电缆传输距离和抖动容忍,可以实现对 SMPTE 信号的直流恢复和补偿直流功能。

2, 集成有 VCO (压控振荡器)时钟恢复器,可以容忍较大的输入时钟抖动(达到 0.7UI)。

3, 提供一个串行数字回环输出来连接外部电缆驱动,该输出可以配置为时钟同步或者没有时钟同步的串行数字数据。

芯片可以工作在以下四种基本模式,分别是 smpte 模式, DVB ASI 模式, 数据旁通模式或者待机模式。

1, 在 SMPTE 模式, GS1671 执行 SMPTE 信号的解扰, 不归零编码的解码, 字对齐处理, 基于行的 CRC 错误校验, TRS 错误检测, 辅助数据校验和错误检测。GS1671 执行辅助数据解嵌, 在完整的辅助数据包被解嵌后, 数据被写入主机可以访问的寄存器中。其它处理包括 H:V:F 定时提取, 亮度和色度辅助数据的显示, 视频标准检测和 SMPTE 352M 数据包的检测和解码。以上的处理功能都可以通过主机进行参数配置和选择执行。

2, 在 DVB ASI 模式, 接收到的数据流执行 8 位/10 位解码处理。

3, 在数据旁通模式, 所有形式的 SMPTE 信号处理和 DVB ASI 解码处理都被禁用, 设备仅执行简单的串并转换过程。

4, 在待机工作模式, 设备消耗很小的电流, 不执行任何信号处理过程, 并行输出接口保持在无电流状态。该芯片工作在待机模式时, 内部集成的信号均衡器功能也自动进入关闭状态。

并行数据输出接口支持 10 位和 20 位数据格式, 支持多路信号复用输出, 支持多种标清和高清视频速率。在所有这些情况下, 20 位并行总线可以多路复用到 10 位低管脚接口来与下游设备连接。并行时钟输入信号操作在 148.5 或者 148.5/1.001MHz (对于所有高清 10 位多路复用模式), 74.25 或者 74.25/1.001MHz (高清的 20 位模式), 27MHz (标清 10 位模式) 和 13.5MHz (标清 20 位模式)。

根据 SMPTE 272M 和 SMPTE 299M 标准的规定, 从视频数据流中最多可以解嵌并提取八个通道的串行数字音频, 这八个音频通道的音频被分为两组进行后续处理。GS1671 可以支持的音频信号格式包括 AES/EBU 和其它三个行业标准串行数字音频格式。标清格式信号时支持 48kHz 的采样率, 16, 20, 24 位的音频格式, 可以工作在同步模式。高清格式信号时支持同步和异步两种模式。其它的音频处理功能包括音频分组选择, 通道交换, ECC 错误检测和校正 (只高清模式支持), 音频通道状态提取。芯片提供的音频时钟与控制信号包括采样率时钟 (f_s) 信号, 串行时钟 ($64f_s$) 信号, 可选的音频主时钟 ($128f_s$, $256f_s$, $512f_s$) 信号。

该芯片有一个 GSPI 接口, 即四线接口方式的串行外设接口, 外部主芯片可以通过该接口以读写配置寄存器方式获取输入视频标准信号的状态信息和对 GS1671 的功能执行进行选择控制。GSPI 接口由串行数据输入信号 (SDIN), 串行数据输出信号 (SDOUT), 低有效的片选信号 (CS), 和一个时钟信号 (SCLK) 组成。因为这些管脚与 JTAG 接口端口共享, 所以有另外一个信号来控制当前使用 JTAG 或者 HOST 接口功能, 当信号是高电平时, 表示当前使用 JTAG 接口功能。当工作在 GSPI 接口功能时, 主芯片端必须提供 SCLK, SDIN 和 CS 信号。SDOUT 管脚是一个非 SDIN 方式的定

时循环管脚，该管脚被连接到主芯片的SDIN管脚。

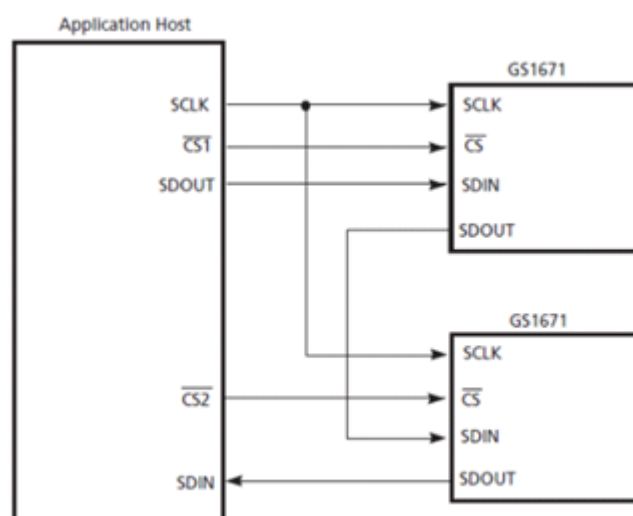


图3-3 GSPI接口连线方式

对GS1671的读或者写访问的启动和终止操作都是由系统主模式端发起的，每次访问总是以一个命令或者地址字开始，后面跟一个将要写的数据或者读到的数据。

3.3.2 GS1671 设备驱动的实现

因为这里只需要对 GS1671 进行随机寄存器位置的读写操作，所有将 GS1671 作为一个字符设备进行操作，下面实现了一个基于 SPI 总线的字符设备驱动程序。该驱动程序的实现步骤包括：

1，注册一个基于 SPI 总线的驱动方法。

```
status = spi_register_driver(&gs1671); //注册基于 SPI 总线的驱动方法
static struct spi_driver gs1671 = {
    .driver = {
        .name = "gs1671",                //gs1671 设备驱动的名字
        .owner = THIS_MODULE,            //该设备驱动的拥有者
    },
    .probe = gs1671_probe,                //驱动加载时执行（匹配后）
    .remove = __devexit_p(gs1671_remove), //驱动卸载时执行
};
```

其中 probe 函数分配驱动数据并且初始化设备驱动相关结构。

2，在 probe 中注册一个名字叫 gs1671 的字符设备驱动。

```
status = register_chrdev(SPIDEV_MAJOR, "gs1671", &gs1671_fops);
static struct file_operations gs1671_fops = {
    .owner = THIS_MODULE,
    .ioctl = gs1671_ioctl,
    .open = gs1671_open,
    .release = gs1671_release,
};
```

文件操作指针中只需要实现对设备的打开，关闭和功能的控制，没有连续的数据读写操作，所以没有实现 read, write 方法。实现字符设备是为了应用层程序可以通过文件接口对 GS1671 设备进行控制，所有的控制功能均定义在 ioctl 控制函数中，这里列举其它的一个：

```
Case    CMD_SET_CHECK_VIDEO_FAULT:    //检测视频信号是否错误
        tmp |= spi->mode & ~SPI_MODE_MASK; //设置读写操作模式
        spi->mode = (u8)tmp;
        spi->buf[0] = VIDEO_STATUS_ADDRESS; //设置寄存器操作地址
        retval = spi_setup(spi); //安装一个 SPI 传输操作。
```

至此，即可以实现通过 ioctl 函数对 GS1671 设备进行读写控制。

注册 SPI 设备驱动，绑定了 SPI 适配器的操作方法。注册字符设备后，将具体的功能实现统一为一个确定的命令。接着应用程序就可以通过调用执行相应的命令执行控制 GS1671 硬件电路完成具体的功能动作。

3.4 基于 V4L2 的视频采集设备驱动设计

3.4.1 VPIF 接口简介

视频端口（VPIF）是 DM6467T 接收和发送视频数据的外设接口，有两个输入通道（通道 0 和通道 1），两个输出通道（通道 2 和通道 3）。输入通道和输出通道分别有相同的架构。

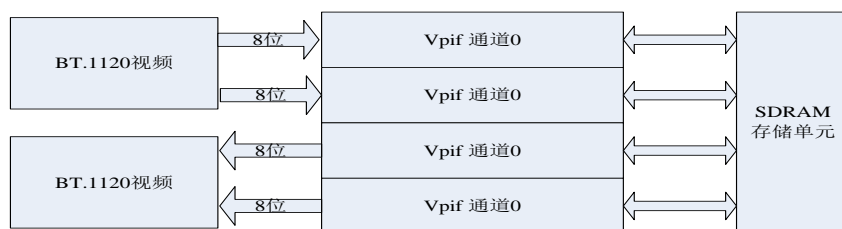


图 3-4 当连接高清视频信号时，vpif 通道的功能

当工作在实现一路高清数字视频输入和输出时的通道功能如图所示。

当 VPIF 工作在输入 BT. 1120 模式时，视频输入时钟 (74.25MHz 或者 148.5MHz) 连接到另一个管理视频输出的 VPIF。为了支持输入和输出高清数字视频，VPIF 必须连接 BT. 1120 信号。所以必须使用两个视频通道来接口数据和输出到显示。VPIF 通道 0 和通道 1 被用来作为高清数字视频的输入，通道 2 和通道 3 作为输出端。BT. 1120 要求一个 27MHz 的系统时钟来同步视频输入时钟，在 TMS320DM6467T 芯片外部必须使用锁相环和分频逻辑电路来产生这个 27MHz 时钟。如果不需要实时传输 BT. 1120 编码的流数据，可以不需要准备锁相环和分频逻辑电路，但仍然需要提供一个 27MHz (或者其它频率) 的时钟给另外一段的锁相环设备来提供一个和视频输出时钟同步的音频时钟。

3.4.2 Linux 视频设备驱动 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 设备节点路径通常是 /dev/video/ 中的 videoX，V4L2 驱动对用户空间来说为字符设备，其主设备号为 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。

基于 V4L2 的视频设备驱动可以支持多种设备：

- 1，视频采集接口；
- 2，视频输出接口；
- 3，直接传输视频接口；
- 4，视频间隔消隐信号接口；
- 5，收音机接口；

3.4.3 视频采集设备驱动的实现

使用 SPI 总线对 GS1671 芯片进行配置后，就可以产生 VPIF 接口需要的 BT. 1120 并行信号，因此 GS1671 相当于视频输入的 CODEC 芯片。由于 GS1671 芯片驱动被单独实现为一个字符设备驱动 (需要读写视频状态信息进行错误检查)，所以在 V4L2 驱动的 CODEC 配置部分没有对 GS1671 进行控制操作。整个视频采集驱动程序

都是用来完成对 TMS320DM6467 芯片 VPIF 模块的初始化配置和完成 V4L2 设备驱动视频框架的接口。

视频采集设备驱动程序从 `vpif_init` 函数, 加载设备驱动时, 调用执行该函数。`vpif_init` 函数完成注册 `vpif` 设备和 `vpif` 驱动到平台总线, 安装视频中断处理函数, 分配内存缓冲区的过程。注册的 `vpif` 驱动和 `vpif` 设备分别为:

```
static struct device_driver vpif_driver = {
    .name = "vpif capture",          //vpif 驱动名
    .bus = &platform_bus_type,      //平台总线类型
    .probe = vpif_probe,            //匹配后执行的函数
    .remove = vpif_remove,          //卸载时执行的函数
};

static struct platform_device _vpif_device = {
    .name = "vpif capture",          //vpif 设备名
    .id = 1,
    .dev = {
        .release = vpif_platform_release, //释放平台设备资源
    }
};
```

`vpif` 设备和 `vpif` 驱动匹配后执行 `vpif_probe` 函数, 该函数注册自己到 v4l2 内核驱动接口, 并初始化视频通道目标。

```
static __init int vpif_probe(struct device *device)
{
    struct video_device *vfd = NULL;
    vfd = video_device_alloc(); //给视频设备分配缓冲区
    *vfd = vpif_video_template; //初始化视频设备
    vfd->dev = device;          //关联 vpif 设备和视频设备
    vfd->release = video_device_release; //注册视频设备卸载函数
    err = video_register_device(channel->video_dev,
        VFL_TYPE_GRABBER, vpif_nr[j]); //注册视频设备
}
```

初始化的视频设备接口为:

```
static struct video_device vpif_video_template = {
```

```
.name = "vpif",           //视频设备名字
.type = VID_TYPE_CAPTURE, //视频设备类型
.hardware = 0,
.fops = &vpif_fops,      //视频设备操作方法
.minor = -1,
};
```

视频设备的操作方法制定了视频设备的文件操作方式：

```
static struct file_operations vpif_fops = {
    .owner = THIS_MODULE,
    .open = vpif_open,           //创建文件指针，关联私有数据
    .release = vpif_release,     //释放设备资源
    .ioctl = vpif_ioctl,
    .mmap = vpif_mmap,          //映射内核空间到用户空间
    .poll = vpif_poll           //视频设备的 poll 调用实现
};
```

视频设备的 ioctl 函数提供了操作 V4L2 设备的命令接口，实现配置驱动或者获取驱动状态的功能。这里只列出重要的命令实现：

```
static int vpif_doioctl(struct inode *inode, struct file *file,
    unsigned int cmd, unsigned long arg)
{
    .....
    case VIDIOC_REQBUFS:        //请求分配视频缓冲区
    {
        struct v4l2_requestbuffers *reqbuf;
        //根据类型参数初始化视频缓冲区
        videobuf_queue_init(&common->buffer_queue,
            &video_ops, NULL,
            &common->irqlock,
            reqbuf->type,
            field,
            sizeof(struct videobuf_buffer), fh);
        INIT_LIST_HEAD(&common->dma_queue); //初始化缓冲区队列
        //分配视频缓冲区
    }
}
```



```

        ret = videobuf_reqbufs(&common->buffer_queue, reqbuf);
        break;
    }
case VIDIOC_QBUF:          //视频缓冲区入队列
    {
        ret = videobuf_qbuf(&common->buffer_queue,
                            (struct v4l2_buffer *)arg); //视频缓冲区入队
        ret = vpif_buffer_prepare(&common->buffer_queue,
                                   buf1, common->buffer_queue.field); //缓冲区入队后执行
        break;
    }
case VIDIOC_DQBUF:        //视频缓冲区出队列
    {
        if (file->f_flags & O_NONBLOCK) //非阻塞模式时的缓冲区出队列
            ret = videobuf_dqbuf(&common->buffer_queue,
                                (struct v4l2_buffer *)arg, 1);
        else //阻塞模式时的缓冲区出队列
            ret = videobuf_dqbuf(&common->buffer_queue,
                                (struct v4l2_buffer *)arg, 0);
        break;
    }

```

其中视频缓冲区队列的操作方法有：

```

static struct videobuf_queue_ops video_qops = {
    .buf_setup = vpif_buffer_setup, //分配视频缓冲区
    .buf_prepare = vpif_buffer_prepare, //将入队用户空间地址转换为物理地址
    .buf_queue = vpif_buffer_queue, //缓冲区入队操作
    .buf_release = vpif_buffer_release, //缓冲区释放
    .buf_config = vpif_buffer_config, //转换虚拟地址到物理地址
};

```

视频设备的 ioctl 函数还提供了视频设备驱动功能查询和兼容性命令：

```

case VIDIOC_QUERYCAP:

```

```

{
    struct v4l2_capability *cap = (struct v4l2_capability *)arg;
    *cap = vpif_videocap;
    cap->capabilities |= dec->capabilities;
    break;
}

static struct v4l2_capability vpif_videocap = {
    .driver = "vpif capture",
    .card = "DM646x EVM",
    .bus_info = "Platform",
    .version = VPIF_CAPTURE_VERSION_CODE,
    .capabilities = V4L2_CAP_VIDEO_CAPTURE | V4L2_CAP_STREAMING
        | V4L2_CAP_VBI_CAPTURE | V4L2_CAP_HBI_CAPTURE
};

```

通过查询命令可以获取 v4l2 视频驱动的驱动名称，板卡名称，总线类型，驱动版本，支持的功能等信息。

3.4.4 基于 V4L2 接口视频采集应用程序

操作 V4L2 视频设备的步骤为：

1，初始化 v4l2 视频接口设备。通常使用 open 函数：

```
videofd = open("/dev/video0", O_RDWR, 0); //阻塞方式打开
```

```
videofd = open("/dev/video0", O_RDWR | O_NONBLOCK, 0); //非阻塞方式
```

阻塞方式和非阻塞方式的却别，阻塞方式时，读操作会等到采集到视频帧数据才返回，非阻塞方式则相反，不管有没有视频帧数据，立即返回结果。

2，设置 v4l2 接口视频设备的采集方式。使用 ioctl 函数对设备的进行设置。

```
Int ioctl(int videofd, unsigned long int cmdrequest, ...)
```

V4l2 视频设备驱动接口支持的命令请求有：

VIDIOC_QUERYCAP：视频设备功能查询；

VIDIOC_ENUM_FMT：列举支持的视频格式；

VIDIOC_G_FMT： 获取可以选择的捕获格式；

VIDIOC_S_FMT： 设置视频捕获格式；

VIDIOC_G_CROP: 查询捕获视频信号边框尺寸;
 VIDIOC_S_CROP: 设置捕获视频信号边框尺寸;
 VIDIOC_REQBUFS: 请求分配视频缓冲区;
 VIDIOC_QUERYBUF: 把分配的视频缓冲地址转换成物理地址;
 VIDIOC_QBUF: 读取捕获到的视频缓冲区;
 VIDIOC_DQBUF: 放视频缓冲区到捕获队列;
 VIDIOC_STREAMON: 开始视频捕获;
 VIDIOC_STREAMOFF: 结束视频捕获;

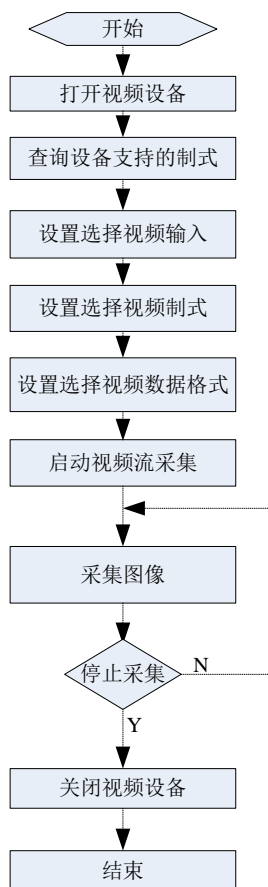


图 3-5 视频采集流程图

3, 设置视频捕获格式。

```

struct v4l2_format    fmt;
memset( &fmt, 0, sizeof(fmt) );

fmt.type= V4L2_BUF_TYPE_VIDEO_CAPTURE;//视频捕获
fmt.fmt.pix.width= 1920;//待捕获视频的宽度
fmt.fmt.pix.height= 1080;//待获取视频的高度
    
```

```

fmt.fmt.pix.pixelformat= V4L2_PIX_FMT_YUYV; //像素存储格式
fmt.fmt.pix.field= V4L2_FIELD_INTERLACED; //隔行存储格式
ioctl(fd, VIDIOC_S_FMT, &fmt); //设置视频采集格式

```

4, 请求分配视频捕获存储缓冲区。

```

struct v4l2_requestbuffers capture_req;
ioctl(fd, VIDIOC_REQBUFS, &capture_req); //请求进行内存分配。

```

5, 查询分配的视频捕获缓冲区物理空间。

```

for( idx = 0; idx < req.count; idx ++ ) {
    buf.type= V4L2_BUF_TYPE_VIDEO_CAPTURE; //当前缓冲区用来捕获
    buf.memory= V4L2_MEMORY_MMAP; //使用内存映射方式
    buf.index= idx; //缓冲区索引号
    ioctl(fd, VIDIOC_QUERYBUF, &buf) //查询指定索引号的缓冲区
    buffers[idx].length= buf.length; //设置每个缓冲区的长度
    buffers[idx].start= mmap(NULL, buf.length,
        PROT_READ | PROT_WRITE,
        MAP_SHARED, fd, buf.m.offset); //映射到用户缓冲区
    ioctl(fd, VIDIOC_QBUF, &buf) //将每个缓冲区加到队列中
}

```

6, 启动视频采集。

7, 获取采集到的视频数据, 并释放占用的缓存。使用 IOCTL 控制函数的 VIDIOC_DQBUF 和 VIDIOC_QBUF 控制命令来获取采集到的视频和释放占用的缓存。

```

struct v4l2_buffer buf;
memset(&buf, 0, sizeof(buf)); //清空视频缓冲区
buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE; //当前缓冲区用来捕获
buf.memory=V4L2_MEMORY_MMAP; //设置为内存映射方式
buf.index=MAXVBUFSIZE-1; //设置从最后一个缓冲区开始
ioctl(videofd, VIDIOC_DQBUF, &buf); //获取采集到的视频缓冲区
ioctl(videofd, VIDIOC_QBUF, &buf); //添加缓冲区到缓冲队列尾部

```

8, 停止视频采集。

9, 关闭视频设备

```

close(videofd);

```

3.5 本章小结

本章对 TMS320DM6467T 平台的嵌入式设备驱动进行了简单介绍，然后介绍了 SPI 适配器设备和适配器驱动程序，在 SPI 适配器的基础上实现了 GS1671 设备驱动程序，设计并实现了基于 GS1671 视频 CODEC 的 V4L2 视频设备驱动，并实现了视频采集的应用程序。

第四章 矩阵卡软件设计与实现

4.1 视频矩阵卡的应用场景

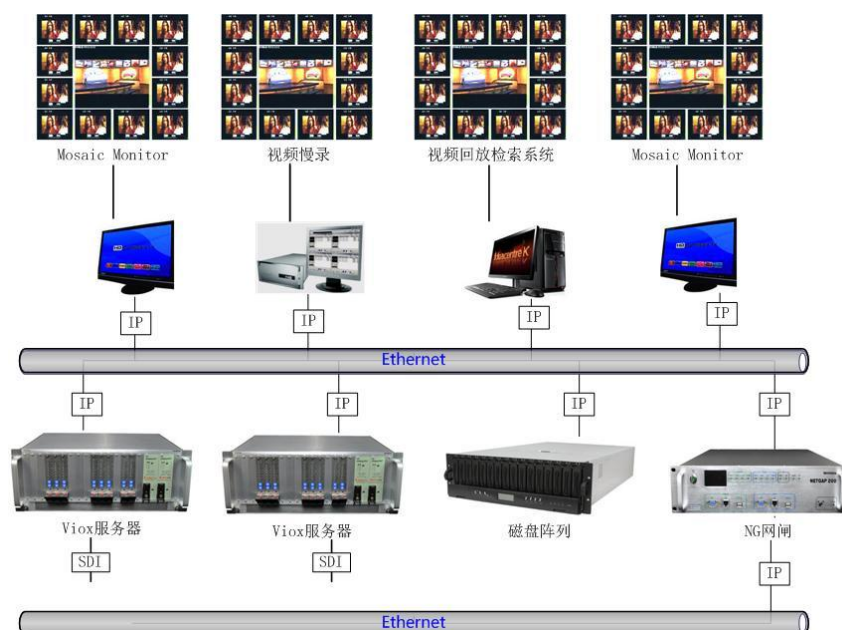


图 4-1 视频矩阵卡的应用场景

在大型的视频监看项目，有很多个视频输入输出接口，而单张视频矩阵卡只实现了一路高清视频信号的输入和 IP 流的输出，因此需要将多张视频矩阵卡安装在一起，作为视频输入输出切换选择单元提供给用户使用，称为视频输入输出切换（Viox）服务器。Viox 服务器的输入端是多个 SDI 或者 HD-SDI 信号，输出 IP 流。多个 Viox 服务器可以并行工作，非常便于扩展系统的规模。针对视频矩阵卡有一整套完整的通信协议，可以提供给客户端软件使用，这里称为视频矩阵卡的客户端驱动。多屏显示程序等客户端软件根据视频矩阵卡的客户端驱动接口可以很方便的实现对矩阵卡的扫描，查询，参数设置和视频流获取等功能。该场景完整的实现了对视频的监看，存储，传输等功能。

4.2 视频矩阵卡的软件功能需求分析

视频矩阵卡的软件功能需求包括：

- 1, 对输入 BT. 1120 视频信号的采集, 输出 IP 码流, 系统延迟不超过两秒;
- 2, 视频使用最先进的 H. 264 压缩, 视频编码码率可调整;
- 3, 音频使用 AAC 或者 mp3 压缩, 编码类型可以进行选择, 编码码率可调整;
- 4, 检测视频信号是否有雪花, 静帧, 黑屏等;
- 5, 视频尺寸可以缩小到原始尺寸的 1/2, 1/4, 1/8;
- 6, 检测音频信号的过低, 过高和反相;
- 7, 输出码流可以发送到 rtmp 服务器;
- 8, 硬件异常时可以输出报警信号;
- 9, 可以通过 web 页面进行系统远程升级。

4.3 软件模块划分

将嵌入式系统应用软件按功能划分为:

- 1, 命令接收管理模块;
- 2, 视频采集编码发送模块;
- 3, 系统状态监视模块;
- 4, Web 模块;

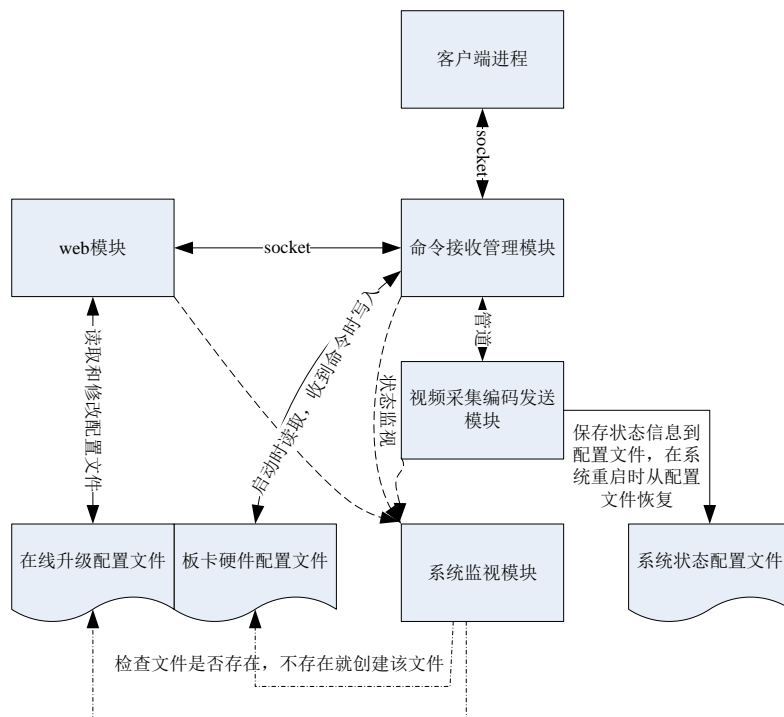


图 4-2 应用软件各个模块之间的关系

命令接收管理模块负责接收和管理从客户端程序或者从 web 页面发出的命令（该命令的格式必须符合客户端和服务端定义的通信协议），产生设备内部命令传递给编码处理模块，编码处理模块执行后返回结果。命令接收管理模块根据编码处理模块执行命令的结果，按照通信协议产生应答命令回复给客户端。

视频采集编码发送模块作为板卡的主要逻辑功能执行单元，负责接收命令接收管理模块传递过来的内部命令，执行该命令，并回复执行结果给命令接收管理模块。编码处理模块还需要发送音视频流给指定的监看端，对音视频数据和板卡硬件状态进行监测，产生报警信息。编码处理模块将板卡硬件配置信息保存到硬件设置配置文件，将编码器处理模块当前的各种运行参数保存到编码模块参数文件。

Web 模块包含 web 服务器程序，web 显示页面和 cgi 程序。Web 页面除了需要实现客户端具有的功能外，还需要支持设置板卡网络参数和从 Web 页面进行板卡固件升级的功能。Web 页面通过查询系统升级配置文件获得当前固件信息。

配置文件有三个。硬件设置配置文件保存板卡的网络地址及通信端口等信息，系统启动时需要进行检查以确认配置文件状态与系统状态一致。编码模块参数文件保存编码处理模块运行时参数，在系统重启后能够恢复到系统重启前的工作状态。在线升级配置文件保存系统的固件信息等。

系统监视模块负责扫描并监视系统当前所有进程的运行状态。当接收到变更网络地址等信号时，重启系统各个功能模块。启动时检查配置文件，如果没有找到配置文件就生成系统默认配置文件。

4.4 命令接收管理模块

命令接收管理模块是矩阵卡嵌入式系统应用程序与矩阵卡客户端软件进行通信的接口模块。命令的收发使用客户端/服务器（C/S）模式，嵌入式板卡作为 TCP 连接的服务器端，负责监听指定端口，开启新的连接线程来处理命令收发，由于嵌入式板卡计算能力限制，设置了最大监听客户端的个数限制。为了保障命令传输的可靠性，在命令收发端对网络传输的数据包均进行了协议检查和校验。使用 TCP 协议传输命令和回复数据，使用 UDP 协议传输广播数据包。

命令接收管理模块内部各个模块作用：

- 1, 监听线程(listen)在客户端和服务端双方约定的监听端口监听接收客户端的 TCP 连接请求，并为每个客户端连接请求创建一个独立的执行线程-客户端命令接收服务线程（client_s[n]）。

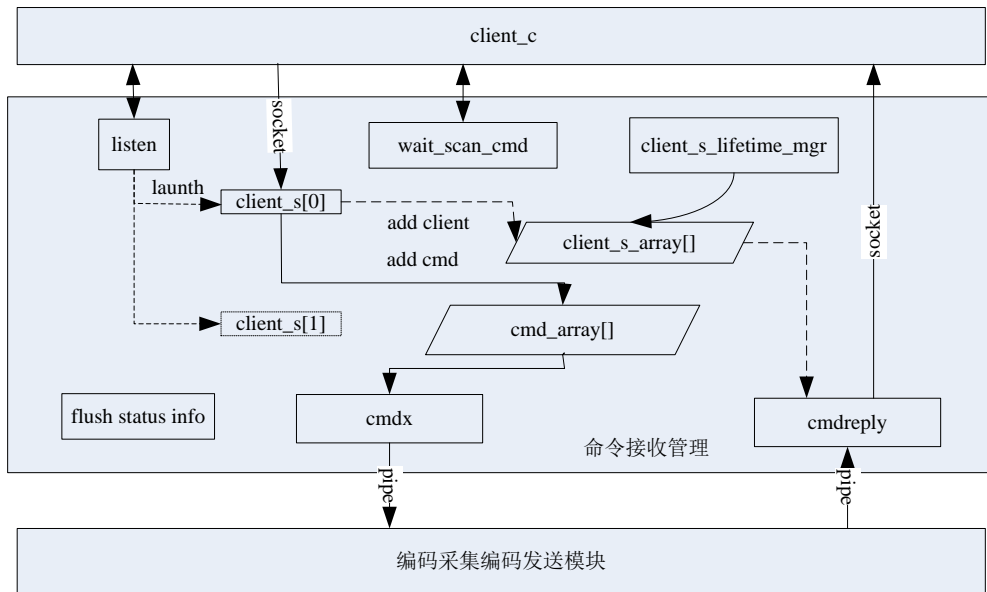


图 4-3: 命令接收管理模块功能框图

2, 客户端命令接收服务线程 (client_s[]) 接收客户端发出的命令。添加客户端的连接信息到客户端服务管理数据 (client_s_array[]), 将接收到的客户端命令添加到命令管理数组 (cmd_array[])。每收到一个命令或者心跳信号, 更新客户端生命期值为满。当接收到客户端生命期管理模块发出的客户端终止信号时, 释放资源并终止客户端命令接收服务线程。

3, 客户端服务器生命期管理线程 (client_s_lifetime_mgr) 对已经连接到矩阵卡的客户端进行生命期管理。如果客户端连接在较长的时间内没有进行任何命令或者心跳活动, 则释放该客户端连接。该线程依次检查各个已连接客户端的生命期, 发现有生命期耗尽的客户端时, 查找该客户端对应客户端命令接收服务线程 (client_s[]), 给该线程发送终止信号。

4, 扫描命令等待线程 (wait_scan_cmd) 启动一个 UDP 服务器, 接收局域网内的板卡扫描 UDP 广播包, 回复板卡扫描数据给客户端。

5, 数组 client_s_array[n] 存储所有的客户端连接信息。以同步方式被访问, 提供添加, 删除和查询操作接口。

6, 命令管理数组 (cmd_array[]), 串行化所有客户端命令接收服务线程 (client_s[]) 接收到的命令, 保证各个命令被串行的执行, 以同步的方式被访问, 提供添加命令和取出命令操作接口。

7, 命令执行线程 (cmdx[]) 被用来查询命令管理数据中是否有新的命令被添加。如果接收到命令就通过管道发送给后续模块进行处理。

8, 命令回复线程(cmdreply[])被用来查询并接受视频采集编码发送模块发出的命令回复数据包, 每收到一个命令回复数据包, 查询客户端服务管理数据(client_s_array[])该回复数据包对应的命令客户端句柄, 回复命令执行结果给对应的客户端。

9, 刷新状态信息(fluash_status_info)线程获取所有客户端的连接记录, 命令接收和执行记录, 所有客户端的连接状态和当前每个客户端的生命期值到/tmp/cmdmgr.status 文件。该模块用于系统监视及程序调试。

4.5 视频采集编码发送模块

视频采集编码发送模块是应用程序的主要功能逻辑执行单元, 执行的功能较多, 各个功能之间的关系较复杂。包含功能执行单元和管理执行单元两部分。

1, 功能执行单元: 这部分实现的功能有: 视频采集, 视频缩放, 视频检测, 视频的编码, 4 路立体声音频的采集, 音频检测, 音频 aac 和 MP3 编码, rtp 协议的封装处理, rtmp 协议的封包, 硬件设备状态的检测。

2, 管理执行单元: 这部分实现的功能有: 命令处理, 报警处理, 音视频组管理, 这部分主要用来对功能执行单元各种功能的管理。

功能执行单元中的每个功能实现都是独立的, 各个功能实现同时或者顺序执行, 后面章节对每个功能实现进行了单独的设计。管理执行单元作为功能实现的管理单元, 将各个功能实现有机的联系起来。

4.5.1 音视频采集

音视频采集功能均依赖于 linux 设备驱动层, 视频采集使用 linux v4l2 设备驱动接口, 而音频采集使用了 linux alsa 音频设备驱动接口。在第三章对 v4l2 驱动和应用程序的编写做了详细介绍, 这里只介绍音频 alsa 驱动程序接口及应用程序接口设计部分的内容。

ALSA (Advanced Linux Sound Architecture, 高级 Linux 声音架构)是在 Linux 操作系统上提供的对音频接口和 MIDI (Musical Instrument Digital Interface, 音乐设备数字化接口)进行支持音频编解码芯片驱动程序架构。在 linux 2.6 系列及以后版本的内核中, ALSA 已经成为默认的声卡驱动子系统。ALSA 支持从入门级声卡到专业级音频设备所有类型的音频设备接口, 完全模块化的设计, 兼容旧的 OSS 声卡驱动架构。

使用 alsa 接口对输入的 IIS 音频信号进行采集的过程：

1, 打开声卡设备。

```
static char *device = "plughw:0,0"; //声卡名
//以捕获的方式打开声卡设备,
rc = snd_pcm_open(&handle, device, SND_PCM_STREAM_CAPTURE, 0);
```

2, 配置声卡设备硬件参数, 包括声道数, 左右声道数据放置方式, 采样率, 量化位数等。

```
//设置左右声道的数据存放方式为左右声道数据在内存中交替放置。
snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERL
EAVED);
//设置音频采样时以16位量化采样数据。
snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE);
val = 48000; //设置音频采样率为48khz
snd_pcm_hw_params_set_rate_near(handle, params, &val, 0);
snd_pcm_hw_params_get_buffer_size_max(params, &buffer_time);
snd_pcm_hw_params_get_period_size(params, &chunksize, 0);
period_time = buffer_time/4 ;
snd_pcm_hw_params_set_period_size_near(handle, params, &period_time,
&dir); //设置音频缓冲区的个数
snd_pcm_hw_params_set_buffer_size_near(handle, params, &buffer_time);
snd_pcm_hw_params_get_buffer_size(params, &buffer_time);
snd_pcm_hw_params_get_period_size(params, &chunksize, 0);
//设置音频捕获的通道数为双声道。
snd_pcm_hw_params_set_channels(handle, params, 1);
//将前面设置的内容设置到寄存器。
rc = snd_pcm_hw_params(handle, params);
```

3, 设置声卡驱动程序的配置参数, 这里设置了最小音频捕获数据大小和输出音频采样时间戳。

```
snd_pcm_sw_params_current(handle, swparams); //获取当前参数设置
//设置音频最小捕获数据大小
snd_pcm_sw_params_set_avail_min(handle, swparams, chunksize);
timestamp_en = 1 ; //设置音频驱动为输出时间戳模式。
```

```
snd_pcm_sw_params_set_tstamp_mode(handle, swparams, val);
```

//使能声卡驱动软件参数的设置。

```
snd_pcm_sw_params(handle, swparams);
```

4, 启动音频采集。

```
rc = snd_pcm_start(handle); //开始音频采集
```

5, 读取采集到的音频数据。

```
rc = snd_pcm_readi(handle, buffer, size);
```

6, 读取音频时间戳信息。

```
snd_pcm_status_get_tstamp(status, &p_tstamp);
```

7, 停止音频采集。

```
rc = snd_pcm_drop(handle);
```

8, 关闭声卡设备。

```
snd_pcm_close(handle);
```

以上是 alsa 音频采集的过程, 但通常为了提高代码的可移植性, 对音频采集和视频采集的过程进行了封装。

视频采集功能的接口定义:

HANDLE videodevice_init(); 该函数实现了分配视频设备资源的功能。函数无输入参数, 返回视频设备的操作句柄。

Int32 videodevice_conf_set(HANDLE viddev_hdl, struct VIDDEV_CONF const conf); 该函数用来设置视频设备的初始化参数。输入参数为视频设备句柄和视频配置信息, 配置信息包括当前输入视频信号的制式和对应的宽, 高, 交织模式等信息。函数返回操作结果或者错误码。

Int32 videodevice_conf_get(HANDLE viddev_hdl, struct VIDDEV_CONF *conf); 该函数用来获取视频采集的格式信息, 在用户查询 H264 编码原始图像信息时调用。输入参数为视频设备句柄和存放格式信息的指针。输出操作结果或者错误码。

Int32 videodevice_data_get(HANDLE viddev_hdl, struct VIDEO_DAT *buf); 该函数用来获取一帧采集到的视频数据。输入参数为视频设备句柄。输出采集到的视频数据。函数返回操作结果或错误码。

Int32 videodevice_buf_put(HANDLE viddev_hdl, struct VIDEO_DAT *buf); 该函数用来将视频采集缓冲区重新送入缓冲区队列供视频采集使用。该函数通常在完成对获取的视频数据处理后调用。返回操作结果或者错误码。

Int32 videodevice_uninit(HANDLE * viddev_hdl);释放视频设备资源。

视频采集编码发送模块最重要的功能是实现了音视频的采集编码发送, 音视频采集编码发送的数据流如图 4-4 所示:

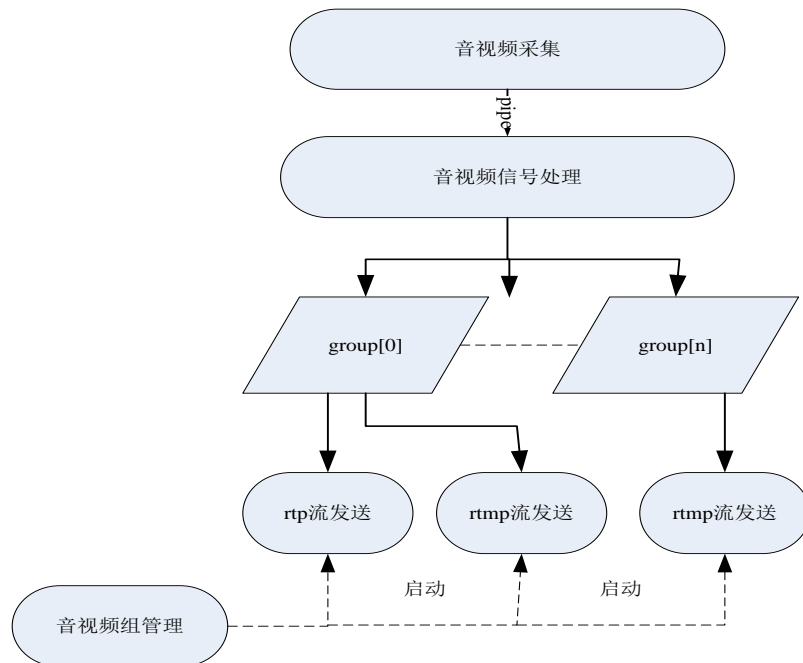


图 4-4 视频采集编码发送模块框图

音频采集功能的接口定义:

HANDLE audiodevice_init();按照指定的方式初始化音频设备, 分配资源, 返回音频设备句柄, 失败则返回空指针 NULL。

Int32 audiodevice_conf_set(HANDLE auddev_hdl, struct AUDDEV_CONF conf);设置音频设备初始化参数。输入音频设备句柄和音频配置参数, 包括音频设备名, 打开方式, 音频的通道数, 采样率, 量化位数, 是否启用时间戳, 输出音频缓冲区大小等。返回操作结果或者错误码。

Int32 audiodevice_conf_get(HANDLE auddev_hdl, struct AUDDEV_CONF * conf);获取音频配置参数。输入音频设备句柄。输出获取到的音频配置参数。返回操作结果或错误码。

Int32 audiodevice_start(HANDLE auddev_hdl, u32 oper);设置启动或停止音频采集。输入音频设备句柄。返回操作结果或者错误码。

Int32 audiodevice_read(HANDLE auddev_hdl, struct AUDIO_DATA * buf);读取采集到的音频数据。输入音频设备句柄。输出采集到的音频数据和音频采样点的时间戳等。返回操作结果或错误码。

Int32 audiodevice_uninit(HANDLE auddev_hdl);释放音频设备资源。

4.5.2 基于 DSP 核的音视频处理

音视频算法处理功能模块包括音频检测，音频编码，视频缩放，视频检测和视频编码，这五个算法功能均运行在DSP核上，ARM核应用层通过Codec Engine提供的VISA（video, image, spech, audio）接口对DSP核上运行的算法功能进行调用执行。

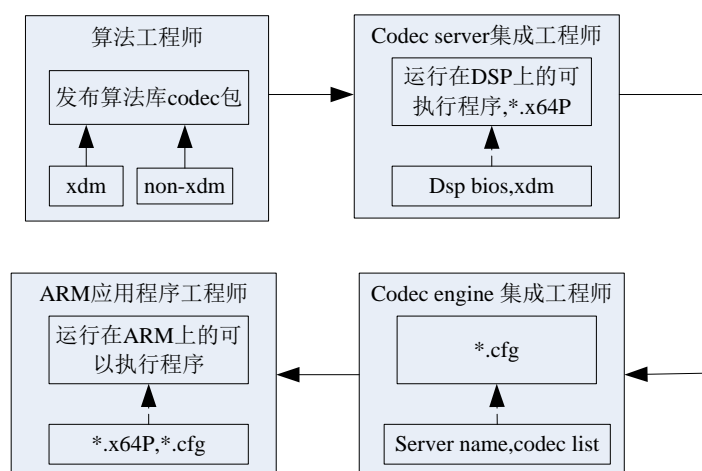


图4-5 DSP算法开发流程

音视频算法开发的步骤：

1, DSP算法工程师需要按照算法的要求在CCS开发环境下开发基于DSP的音视频编解码算法，通常结果为一个编译好的编解码算法库文件，例如encodehd.lib。为了后续过程能够使用Codec Engine接口调用这个算法库，算法库开发的时候总是遵守xDM(xDAIS for Digital Media)和xDAIS(eXpress DSP Algorithm Interface Standard)标准。

2, 生成可以在DSP核上运行的可执行程序库，例如encodehd.x64p, 通常叫做DSP server库。

3, 创建Codec Engine的配置文件encodehd.cfg，在配置文件中设置当前使用的DSP Server算法库名称和包括的算法功能。通常是只使用一个DSP Server端算法库把多个算法功能包含起来。

4, 应用程序工程师按照Codec Engine配置文件(enocodehd.cfg)中的配置对ARM端应用程序进行编译，生成ARM端的可执行程序。

ARM 端应用程序调用执行 DSP 算法库的步骤为：

1, 打开 Codec Engine 设备。

```
venc->hEngine = Engine_open(venc->enginename, NULL, NULL);
```

2, 创建 DSP Server 端编解码算法库实例。

```
venc->hVe1 = Venc1_create(venc->hEngine, venc->encodename,
&venc->extParams, &venc->extDynParams);
```

3, 设置和获取编解码算法库实例的参数。

```
//设置编码器动态参数
```

```
encStatus.size = sizeof(VIDENC1_Status);
```

```
encStatus.data.buf = NULL;
```

```
status = VIDENC1_control(hEncode, XDM_SETPARAMS, dynParams,
&encStatus);
```

```
//获取编码器输入 buf 信息。
```

```
status = VIDENC1_control(hEncode, XDM_GETBUFINFO, dynParams,
&encStatus);
```

4, 执行编解码处理。函数参数为编解码句柄, 输入输出数据和输入输出参数。

```
//输入一帧视频数据进行编码操作。
```

```
status = VIDENC1_process(hVe->hEncode, &inBufDesc, &outBufDesc,
&inArgs, &outArgs);
```

5, 释放编解码算法库实例。

```
VIDENC1_delete(hVe->hEncode); //在应用程序退出时释放编码器算法库实例
```

为了降低系统对 TMS320C6467TDSP 核算法执行过程的关联度, 对算法库的执行过程进行了封装, 定义出简单易用的算法库接口。这里只列出视频检测和视频编码器的接口, 视频缩放, 音频检测和音频编码的定义类似。

视频检测算法接口定义:

HANDLE videodetect_init(); 该函数用来分配视频检测算法需要的系统资源。函数返回视频检测算法的操作句柄。

Int32 videodetect_arithname_set(HANDLE vidscale_hdl, struct DSPARHTLIB_VDETECT_CONF const name); 指定视频检测算法对应的算法库名字等参数。输入参数为视频检测算法的操作句柄和视频检测算法功能名称。

Int32 videodetect_param_set(HANDLE viddetect_hdl, struct VIDDETECT_PARAM const param); 该函数用来设置视频检测算法的检测参数。

Int32 videodetect_param_get(HANDLE viddetect_hdl, struct

VIDDETECT_PARAM *param);该函数用来获取视频检测算法的检测参数。

Int32 videodetect_proc(HANDLE viddetect_hdl, struct VIDDETECT_BUF * const inbuf);该函数用来执行视频检测，视频检测的执行结果存放在视频检测算法的操作句柄。

Int32 videodetect_result_get(HANDLE viddetect_hdl, struct VIDDETECT_RESULT * result);该函数用来获取视频数据算法执行检测的结果。

Int32 videodetect_uninit(HANDLE * viddetect_hdl);释放视频检测算法占用的系统资源。

视频编码算法接口定义：

HANDLE videoenc_init();分配视频编码器需要的资源。返回视频编码器的操作句柄。

Int32 videoencode_arithname_set(HANDLE videnc_hdl, struct DSPARHTLIB_VENCODE_CONF const name);指定视频编码器对应算法功能名称。

Int32 videoencode_param_set(HANDLE videnc_hdl, struct VIDENC_PARAM const param);设置视频编码器的编码参数。

Int32 videoencode_param_get(HANDLE videnc_hdl, struct VIDENC_PARAM * param);获取视频编码器的编码参数。

Int32 videoencode_proc(HANDLE videnc_hdl, struct VIDENC_BUF * inbuf, struct VIDENC_BUF * outbuf);执行一次视频编码器的编码过程，通常为一个视频帧的编码。

Int32 videoencode_uninit(HANDLE videnc_hdl);释放视频编码器占用的系统资源。

4.5.3 音视频流传输

4.5.3.1 实时传输协议（RTP）的实现

实时传输协议（Real-time Transport Protocol, RTP）是在网络上进行传输多媒体数据流的一种网络协议，利用它能够在点对点（单播）或者点对多点（多播）的网络环境中实现流媒体数据的实时传输。

RTP 传输协议通常使用 UDP 进行数据的传输。该协议具有协议简单，易于实现的优点，同时传输控制信息占用通信带宽小，无需路由器支持，缺点是传输不可靠。

实时传输协议（RTP）是由因特网工程任务组(IETF , Internet Engineering Task Force)在 RFC3550（实时应用程序传输协议）标准中定义的。RFC3550 标准中定义的 RTP 协议有两部分组成：RTP 数据协议和 RTP 控制协议。

RTP 数据协议由数据包头部和数据包负载组成。

数据包头部 1 到 12 字节是固定的，固定字节之后是可选的包头扩展部分。

数据包负载可以是自定义格式的消息，音频或者视频数据。

位偏移	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers						
96+32xcc	Profile-specific extension header ID					Extension header length	
128+32xcc	Extension header						

图4-6： RTP包头格式

每个域的含义：

Version (V)，占 2 位，标示 RTP 的版本。当前 RTP 版本定义为数字 2

Padding (P)，占1位，标示是否有填充比特。

Extension (X)，占用1位，扩展位。如果该位被设置，固定头后面必须跟随一个数据包的头扩展。

CSRC count (CC)，占用4位，作用源计数。作用源 (CSRC, Contributing source) 计数表示固定头后面作用源标识符的个数

Marker (M)，占用 1 位，标志位。该位的含义由具体的通信协议来定义，可用于标识有意义的事件例如帧边界等。

PayloadType (PT)，占用 7 位，负载类型。负载类型域标识实时传输协议数据包中负载的格式，由应用层决定它的具体解释。

SequenceNumber，占用 16 位，包序列号。每发送一个 RTP 数据包，包序列号增加一，接收端使用序列号检测是否发生 RTP 数据包丢失，还可以用来按顺序恢复 RTP 数据包。

Timestamp，占用 4 个字节，采样时间戳。

SSRC, 占用 32 比特, 同步源。同步源 (SSRC, Synchronization source) 标示符被随机生成, 来确保同一个 RTP 会话期中同步源识别符的唯一性。

CSRC: 每项占用 32 比特, 最多可以有 15 项。同步源列表识别在此数据包中负载的所有源。

RTP传输协议合理的传输数据安排, 灵活的传输协议支持和网络地址格式无关性使得RTP在实时传输多媒体数据时表现出很大的优越性。

为了实现代码的复用, RTP传输协议中数据包头部定义为:

```
typedef struct _RTP_HEADER {
#ifdef BIG_ENDIAN    //x86架构上使用
    unsigned short sVersion:2;    //版本号
    unsigned short sPadding:1;    //填充位
    unsigned short sExtension:1;  //扩展域
    unsigned short sCC:4;         //作用源计数
    unsigned short sBitMark:1;    //标志位
    unsigned short sPayloadType:7; //负载类型域
#else    //小端模式, ARM平台
    unsigned short sCC:4;
    unsigned short sExtension:1;
    unsigned short sPadding:1;
    unsigned short sVersion:2;
    unsigned short sPayloadType:7;
    unsigned short sBitMark:1;
#endif
    unsigned short sSeqNum;    //包序号
    unsigned int   uTimeStamp; //数据包的时间戳
    unsigned int   uSSrc;      //同步源
    unsigned int   uCSrc[16];  //作用源列表
    RTP_SLICE_FRAME_INFO SliceFrameInfo; //负载数据指针
} RTP_HEADER;

RTP传输协议中负载数据的内容为:
typedef struct _RTP_FRAME_INFO
{
```

```

unsigned int    uFrameType;  //rtp 传输帧类型 0=I 1= 2= 3=Audio
unsigned int    uCurFrameIdx;//当前分片所属的帧号
unsigned int    uCurSliceIdx;//当前分片的序号
unsigned int    uSliceLen;   //当前分片数据包的长度
unsigned int    uTotalOfSlices; //当前数据帧的总分片数
unsigned int    uBackupFrameNum; //备份的帧号数
char           server_extra_data[EXTRA_LEN];
} RTP_SLICE_FRAME_INFO;

```

RTP 传输协议的接口定义:

`HANDLE rtp_init(MONITOR * target);`初始化一个 RTP 传输请求。输入参数为 RTP 传输请求的地址端口等信息。返回 RTP 传输句柄, 失败返回空指针。

`Int32 rtp_frame_write(HANDLE rtp_hdl, RTPTRANS_BUF * buf);`发送一个 RTP 传输包。输入参数为 RTP 传输句柄和 RTP 数据包缓冲区指针。返回 RTP 传输发送的执行结果或者错误码。

`Int32 rtp_reinit(HANDLE rtp_hdl);`重新初始化 RTP 传输请求。输入 RTP 传输请求句柄。返回操作结果或错误码。

`Int32 rtp_uninit(HANDLE rtp_hdl);`关闭一个 RTP 传输。

4.5.3.2 实时消息协议 (RTMP) 的实现

实时消息协议 (RTMP, The Real-time Messaging Protocol) 是 Adobe 公司开发的一种在 flash 播放器和服务器之间传输流化的音视频和数据的网络传输协议。

RTMP 协议建立在 tcp 或者轮询 http 协议之上, 对视频, 音频和数据进行封装, 这些数据可以是 AMF (Action Message Format) 数据, 也可以是 FLV (flash vide) 文件中的音, 视频数据, 封装后的数据可以在 RTMP 客户端和服务端传输, 默认使用 1935 端口。RTMP 流媒体传输协议被设计为具有高度的实时性, 每个传输通道的数据包都是按照固定的大小进行传输。RTMP 传输协议通过传输通道的分包机制来实现并发的音视频包传输, 可以实现多个音视频流并发播放。RTMP 协议由三部分构成: 协议头, 协议体, 协议数据。

RTMP 协议头分为四种, 完整的协议头由包头, 时间戳, 数据类型, 数据大小和流标志组成。

```

Struct RTMP_HEAD {
    Int8    cchannelid:6; //首字节后六位, 通道 ID

```

```

Int8   cheadsize:2; //首字节头两位，协议头长度
Int8   ctimer[3];    //时间戳信息
Int8   clength[3];   //AMF 数据大小
Int8   cdatatype;    //AMF 包类型类型，
Int8   sstreamid[4]; //流 ID
};

```

首字节头两位指定了协议头长度：

协议头两位值	协议头长度	含义
0x00	12 字节	新流
0x01	8 字节	
0x10	4 字节	
0x10	1 字节	

表 4-1 RTMP 首字节头两位的意义

RTMP 数据的类型决定了协议上层必须遵守的规则和执行的执行操作。协议中规定的 RTMP 数据的类型如下图：

0x01	Chunk Size	数据包有效数据的长度
0x03	Bytes Read	已读字节数
0x04	Ping	流控制协议的一种
0x05	Server BW	服务器下传流带宽
0x06	Client Bw	客户端上传流带宽
0x08	Audio data	音频数据包
0x09	Video data	视频数据包
0x12	Notify	提醒信息，不需要回复
0x13	Shared object	共享对象
0x14	Invoke	执行操作，类似于远程调用

表 4-2 RTMP 数据类型列表

RTMP 协议的传输数据块使用 AMF(Action Message Format)格式来描述。为了处理复杂的数据类型，必须采用一种特有的方式使 Flash 与应用服务器之间可以来回传送数据，Adobe 公司开发出轻量级，高效能的 AMF 通信协议。AMF 最大的特色在于可以直接将 Flash 中的内置对象，例如 Object, Array, XML 等数据传回服务

器端，并且在服务器端解析成适当的对象，可以大大减轻开发人员的工作，同时也节省了开发时间。由于 AMF 采用二进制编码，数据可以高比率的压缩，因此非常适合用来传递大数据，数据量越大，传输效率就越高。

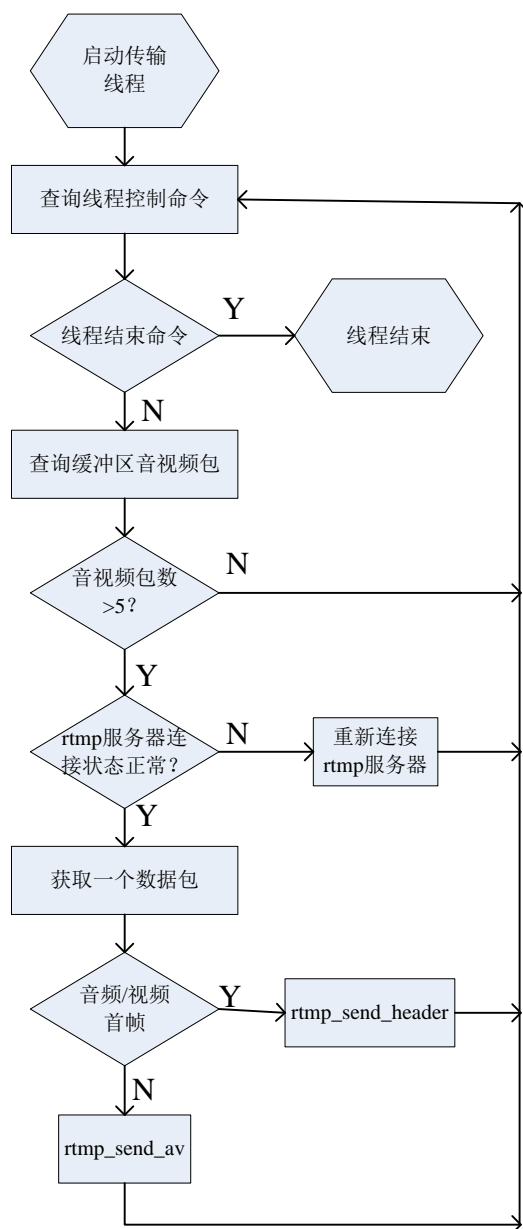


图 4-7 rtmp 传输线程执行流程

客户端和服务器的连接过程分为两步，握手和连接请求。

1，客户端和服务器的握手。客户端以系统时间作为种子，通过某种算法生成数字签名，产生 1537 字节的握手数据包 1，发送给服务器；服务器根据客户端

的数字签名产生一个 3073 字节的验证包（数据包 2），回复给客户端；客户端在接收到服务的回应后会发送一个 1536 字节的回复数据包 3。

2，连接请求的具体过程：客户端发送连接命令；服务器回复服务器带宽信息；服务器回复发送本地带宽消息；服务器返回连接成功消息；客户端发送创建流请求；服务器返回创建流成功消息；客户端发送播放文件消息；服务器返回类型和数据大小信息；服务器返回开始播放消息；服务器返回大小，宽高，速率等视频信息。

RTMP 传输子线程执行的过程为：

1，接收到 RTMP 传输请求时，启动一个 RTMP 传输线程。

2，查询是否接收到线程退出消息。只有当接收到请求删除 RTMP 传输命令时，命令处理程序发送 RTMP 线程退出消息。如果接受到线程退出消息，执行线程退出处理过程，退出线程；否则继续执行后面步骤。

3，查询缓冲区中的音视频数据包数量。如果缓冲区音视频数据包数量少于五个，则跳转到第 2 步继续执行；否则继续执行后面步骤。

4，检查 RTMP 服务器是否连接成功。如果连接失败，则跳转到第 2 步继续执行；如果连接 RTMP 服务器成功，则继续执行。

5，获取一个音视频数据包。

6，判断数据包的类型，如果是音频首帧或者视频的 PPS 帧，使用 `rtmp_send_header` 进行发送；否则，使用普通帧的发送函数 `rtmp_send_av` 函数进行发送。

7，重新跳转到第 2 步进行循环。

RTMP 传输协议的接口定义：

`HANDLE rtmp_init(MONITOR * target)`；初始化一个 RTMP 传输请求。输入参数为 RTMP 传输请求的 URL 地址和流名字等信息。返回 RTMP 传输句柄，失败返回空指针。

`Int32 rtmp_frame_header_write(HANDLE rtmp_hdl, RTMP_FRM * frm)`；封装视频 PPS 帧和音频首帧为 RTMP 协议包并发送到 RTMP 服务器端。

`Int32 rtmp_frame_av_write(HANDLE rtmp_hdl, RTMP_FRM * frm)`；封装普通音视频帧为 RTMP 协议包并发送到 RTMP 服务器端。

`Int32 rtmp_reinit(HANDLE rtmp_hdl)`；重新初始化 RTMP 传输请求。

`Int32 rtmp_uninit(HANDLE rtmp_hdl)`；关闭当前 RTMP 传输。

4.5.4 音视频组管理

音视频组管理功能模块用来对音视频压缩编码后的码流进行管理。当接收到客户端发出的监看命令后，发送音视频流到制定的目的地址，RTP 流对应 IP 地址和端口，RTMP 对应 RTMP 服务器的网络地址和实时流的名称。

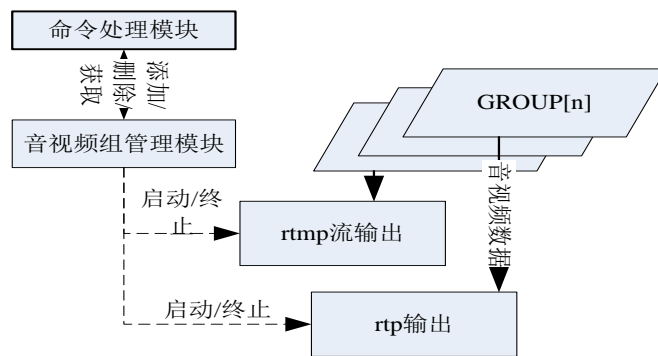


图 4-8 音视频组管理模块和其它模块的关系

1，音视频组管理模块用来执行命令处理模块发送过来的监看命令，提供了添加一个监看目标/删除一个监看目标/获取所有监看目标命令接口。

2，音视频组管理模块可以启动一个 RTP 传输流输出线程或者 RTMP 传输流输出线程。传输流输出线程从音视频流分组的数据链表中取出需要数据包进行流传输数据包的发送。

音视频组管理功能的接口定义：

`HANDLE avgroup_init()`；组管理模块初始化。

`Int32 avgroup_uninit()`；组管理模式的释放。

`int32 avgroup_monitor_add(HANDLE avgrps_hdl , struct AVMONITOR_TARGET const target)`；添加一个音视频流监看地址。

`Int32 avgroup_monitor_del(HANDLE avgrps_hdl , struct AVMONITOR_TARGET const target)`；删除一个已有的流监看目标。

`Int32 avgroup_monitornum_get(HANDLE avgrps_hdl , uint32 * montcnt)`；获取当前组内监看目标的总个数。

`Int32 avgroups_monitors_get(HANDLE avgrps_hdl , struct AVMONITOR_TARGETS * targets)`；获取当前组内所有的监看目标。

4.5.5 辅助设备的管理

本项目中的辅助设备包括温度传感器，实时时钟，千兆网卡，GS1671 芯片，FPGA 芯片，通过对这些设备的读写操作，应用程序可以获得包括温度，系统时间，网络连接状态，音视频输入信号状态在内的硬件监视信息。该模块负责对上述设备的管理。

通过对 GS1671 字符设备驱动程序对 GS1671 芯片操作，可以获取输入 HD-SDI 信号的检测结果。可以作为一个简单的 HD-SDI 信号分析设备。检测到的信息包括：

- 1，SDI 状态信息；
- 2，音频错误信息；
- 3，输入音频状态信息；
- 4，视频错误信息；
- 5，视频交织模式；
- 6，视频帧率；

其中视频错误信息又分为：

- 1，视频信号未锁定；
- 2，视频 EAV 码错误；
- 3，视频 SAV 码错误；
- 4，行号错误；
- 5，亮度 CRC 错误；
- 6，色度 CRC 错误；
- 7，辅助数据亮度校验和错误；
- 8，辅助数据色度校验和错误；
- 9，视频制式错误。

辅助设备管理功能的接口定义为：

`HANDLE auxdevice_init()` ;辅助设备管理模块的初始化，打开设备，申请资源。

`Int32 auxdevice_uninit(HANDLE auxdev_hdl)` ;关闭设备，释放资源。

`Int32 auxdevice_hwsta_get(HANDLE auxdev_hdl , struct AUXDEV_HWSTA * hwsta)` ;获取板卡硬件的输入信号状态信息。

`Int32 auxdevice_ipaddr_set(HANDLE auxdev_hdl , uint32 const ip)` ;设置当前板卡的主 IP 地址。

`Int32 auxdevice_netmask_set(HANDLE auxdev_hdl , struct STNETMASK * mask)` ;设置当前板卡的网络掩码。

Int32 auxdevice_defroute_set(HANDLE auxdev_hdl, uint32 const route);
设置当期板卡的默认路由。

Int32 auxdevice_macaddr_set(HANDLE auxdev_hdl, struct STMAC * mac);
设置当前板卡的 MAC 地址。

Int32 auxdevice_stip_get(HANDLE auxdev_hdl, struct BDINFO_NETSTATE * network);
获取当前板卡网络配置信息。

Int32 auxdevice_tempture_get(HANDLE auxdev_hdl, uint32 * temp);
从辅助设备单独获取当期板卡的温度信息。

Int32 auxdevice_param_set(HANDLE auxdev_hdl, struct AUXDEV_PARAM const param);
设置辅助设备报警参数。

Int32 auxdevice_param_get(HANDLE auxdev_hdl, struct AUXDEV_PARAM * param);
获取辅助设备报警参数。

4.5.6 音视频报警功能

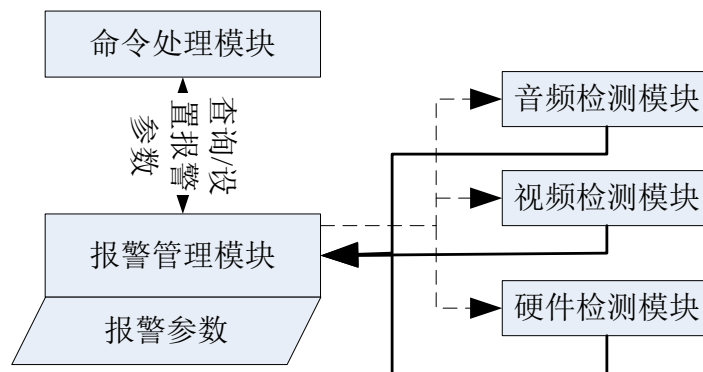


图 4-9 音视频报警模块与其他模块的调用关系

音视频报警管理模块执行报警参数设置命令，更新报警参数到各个检测模块，从各个检测模块获取报警结果，然后发送报警信号。

1，报警管理模块对命令处理模块提供查询和设置报警参数接口。

2，报警管理模块更新报警参数到音频检测模块，视频检测模块和辅助设备管理模块(硬件检测)。

3，报警管理模块获取音频检测，视频检测和硬件检测的结果。

4，告警管理模块对报警内容的协议封装和 UDP 广播发送。

报警管理模块的接口定义为：

HANDLE alarmdeliver_init();初始化报警模块。

Int32 alarmdeliver_flush(HANDLE alarmdev_hdl); 刷新报警消息池。
 Int32 alarmdeliver_param_set(HANDLE alarmdev_hdl); 设置报警参数。
 Int32 alarmdeliver_param_get(HANDLE alarmdev_hdl); 获取报警参数。
 Int32 alarmdeliver_send(HANDLE alarmdev_hdl); 以广播的方式发送板卡报警数据包。

Int32 alarmdeliver_uninit(HANDLE * alarmdev_hdl); 释放资源和重置句柄。

4.5.7 命令执行模块

命令执行模块使用其它功能模块提供的接口，执行接收到的客户端命令，将命令执行结果发送给命令回复模块。

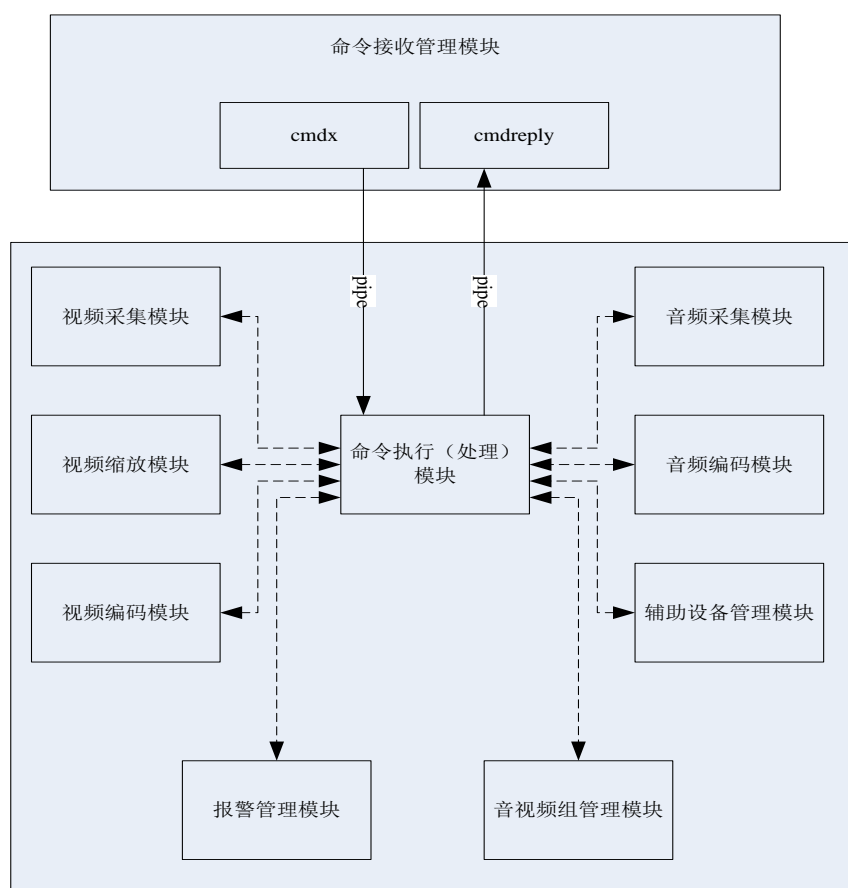


图 4-10：命令执行模块与其他模块的调用关系

1, 命令执行模块通过管道方式接受命令接收管理模块中的 cmdx 线程输出的客户端命令。当前命令没有执行完成时，不会读取后续命令。

- 2, 根据命令的内容调用其它功能模块。
- 3, 获取命令执行结果。
- 4, 回复结果给命令接收管理模块中的 cmdreply 线程。

对应的命令处理模块接口定义为:

HANDLE cmdprocess_init();初始化命令执行模块。

Int32 cmdprocess_proc(HANDLE cmdpro_hdl);命令执行, 包括执行和回复执行结果。

Int32 cmdprocess_uninit(HANDLE * cmdpro_hdl);

4.6 系统监视模块

系统监视模块负责扫描并监视系统当前的运行状态。主要包括:

- 1, 系统启动时扫描检查配置文件是否存在, 如果没有找到配置文件就自动生成系统默认配置文件。
- 2, 当接收到变更网络地址等信号时, 重启各个功能模块。
- 3, 每五秒执行设置定时器的设置, 即喂狗任务。
- 4, 每三秒执行一次系统任务检查, 扫描是否有异常退出的程序, 如果有异常退出, 则重新启动系统。

4.7 配置模块

配置模块包括板卡功能配置文件, 板卡运行参数配置文件及 web 升级配置文件。配置模块保存了当前板卡支持的功能属性, 运行状态, 和升级所需要的信息。

- 1, 板卡功能配置文件包括了当前板卡支持通道数, 最大连接目标数, 支持的功能列表, 板卡的网络地址默认端口等信息。
- 2, 板卡运行参数配置文件制定了当前音视频通道的开关状态, 音视频编码器的参数, 音视频检测的参数, 网络流传输的接收地址等信息。
- 3, web 升级配置文件包括板卡当前使用的软件版本号和目录信息。

4.8 软件测试

将脚本文件, 配置文件和编译生成好的应用程序和编解码算法库拷贝到根文件系统中的/opt/app 目录, 制作出 yaffs2 文件系统镜像, 烧写到 NAND FLASH 的文件系统中。

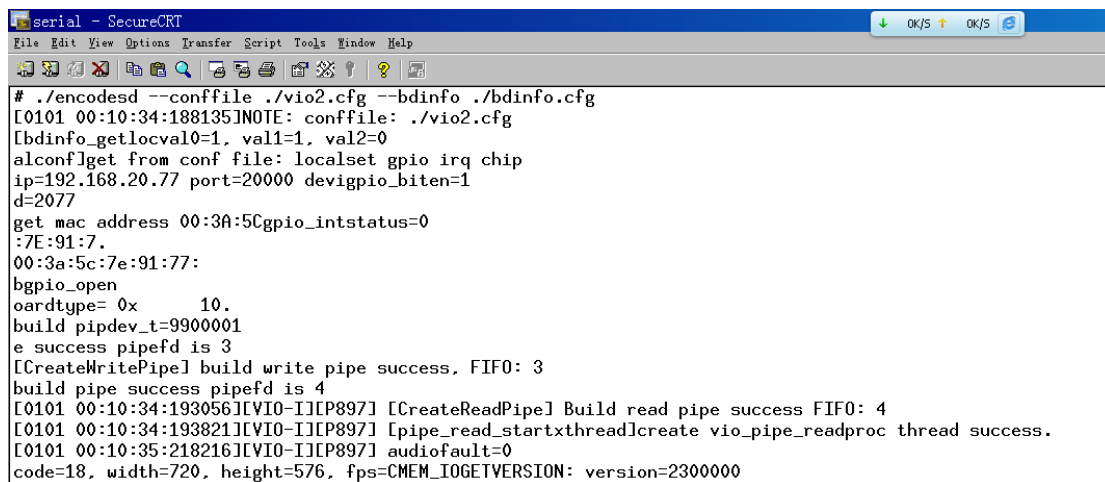
1) 硬件平台加电启动。在 Linux 操作系统启动成功后，登陆系统终端，切换目录到对应的执行脚本目录。

2) 运行脚本文件。

./load_modules.sh //加载驱动程序

./run_encode.sh //加载执行应用程序

应用程序的运行信息：



```
# ./encodesd --conf file ./vio2.cfg --binfo ./binfo.cfg
[0101 00:10:34:188135]NOTE: conf file: ./vio2.cfg
[binfo_getlocval0=1, val1=1, val2=0
alconf]get from conf file: localset gpio irq chip
ip=192.168.20.77 port=20000 devgpio_biten=1
d=2077
get mac address 00:3A:5C:7E:91:77
:7E:91:77.
00:3A:5C:7E:91:77:
bgpio_open
oarttype= 0x 10.
build pipdev_t=9900001
e success pipefd is 3
[CreateWritePipe] build write pipe success, FIFO: 3
build pipe success pipefd is 4
[0101 00:10:34:193056][VIO-I][P897] [CreateReadPipe] Build read pipe success FIFO: 4
[0101 00:10:34:193821][VIO-I][P897] [pipe_read_startxthread]create vio_pipe_readproc thread success.
[0101 00:10:35:218216][VIO-I][P897] audiofault=0
code=18, width=720, height=576, fps=CMEM_I0GETVERSION: version=2300000
```

图 4-11 应用程序启动时的输出信息

3) 运行矩阵卡测试程序。

扫描板卡，可以扫描到一张板卡。

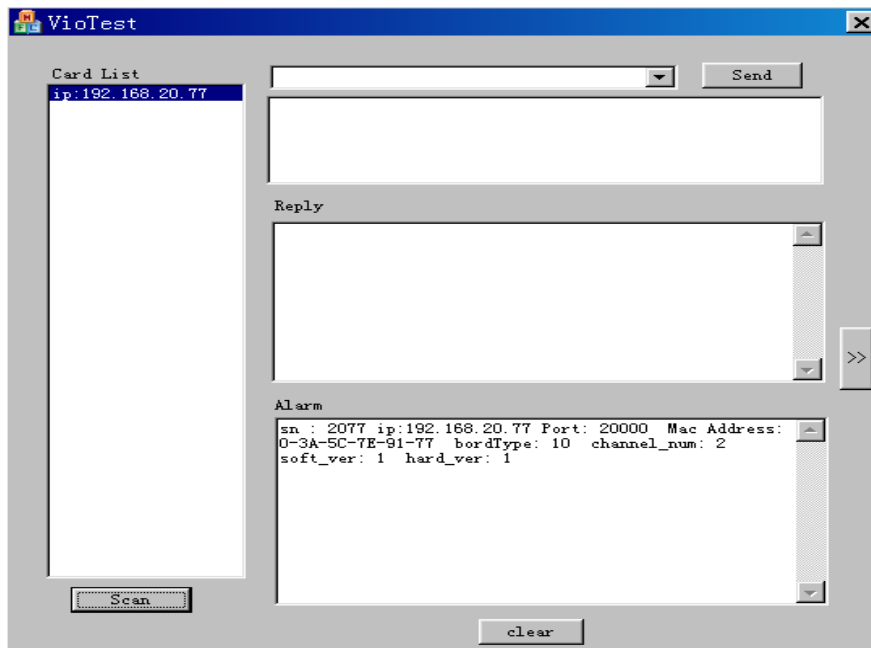


图 4-12 扫描板卡信息

查询板卡信息，可以获得板卡的板卡类型，网络信息，序列号，软件版本和硬件版本等信息。

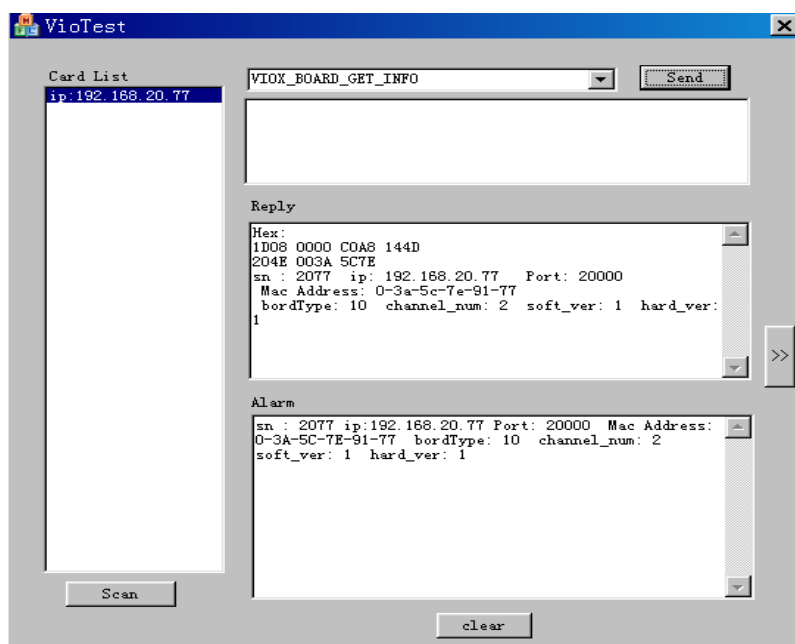


图 4-13 查询板卡信息

查询板卡的状态，可以获得板卡的温度，电压信息。

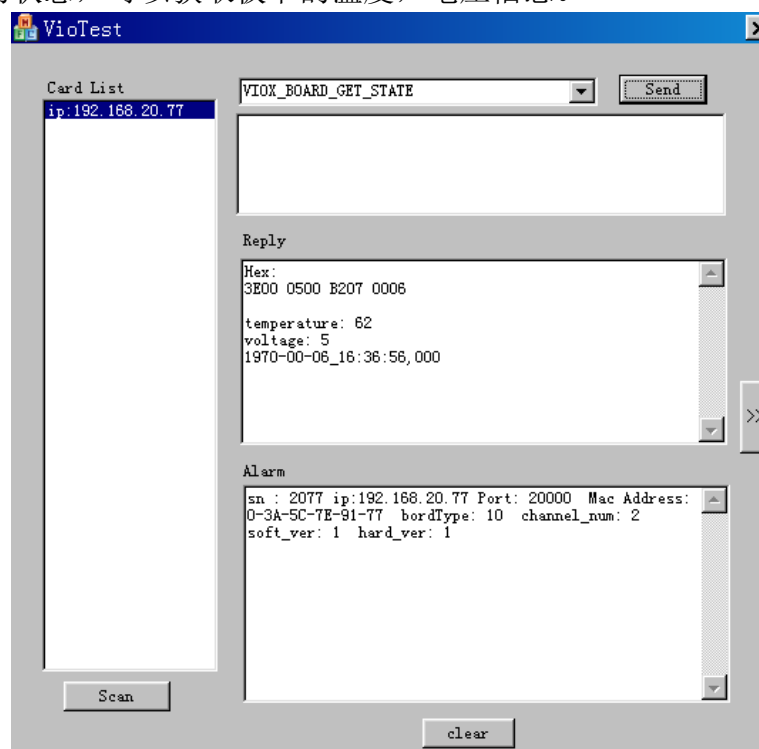


图 4-14 查询板卡状态

查询板卡的硬件信息，获取串行数字接口信号的状态信息，包括音频错误，视频错误等信息。

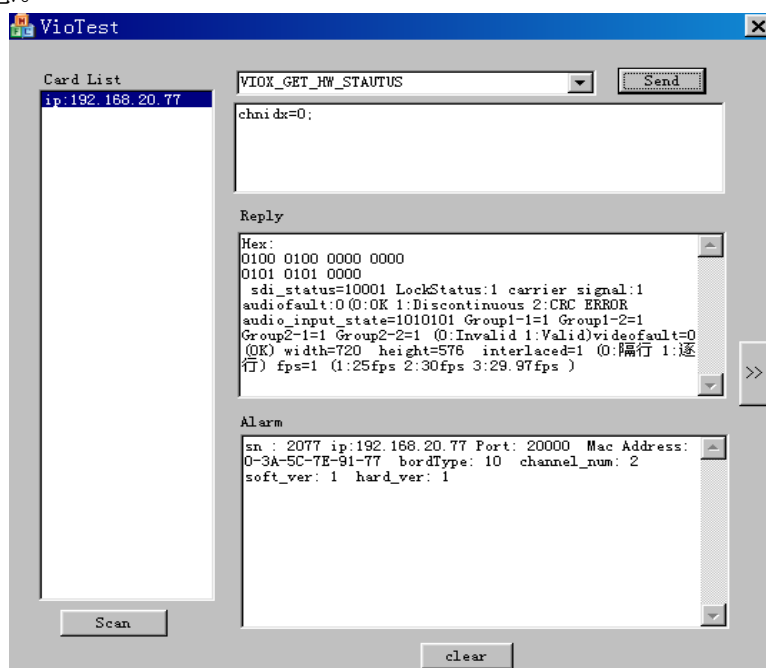


图 4-15 查询板卡硬件信息

使用客户端程序监看一路 RTP 音视频流：

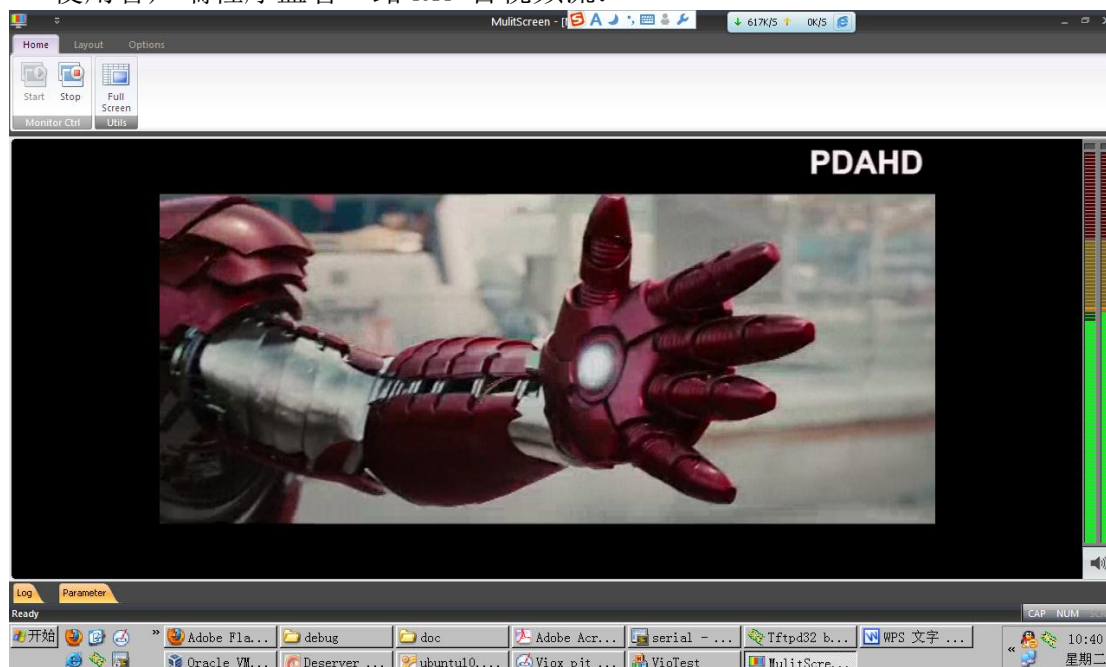


图 4-16 RTP 音视频流

使用 flash-player 监看 RTMP 流：



图 4-17 RTMP 音视频流

4.9 本章小结

本章首先介绍了视频矩阵卡的应用场景，分析了对嵌入式系统软件的功能需求，接着对系统软件进行了模块划分，然后对命令接收模块和视频采集编码发送模块进行了详细设计和接口定义，最后通过测试软件和客户端程序系统软件和驱动程序进行了测试。

第五章 总结

全文总结

本论文的主要工作是使用 TI 公司开发的 TMS320DM6467T 这款基于达芬奇技术的数字信号处理器（DSP）芯片，设计并实现了基于网络的高清数字视频矩阵卡。

首先熟悉了 TMS320DM6467T 开发平台的特点和开发方法，搭建了软件开发环境。针对具体的应用场景设计并实现了基于 Linux 的 GS1671 串行外部接口驱动程序和 FPGA 输出 VPIF 输入的高清视频设备驱动程序。在仔细分析客户对矩阵卡功能需求的基础上，完成功能模块划分并对每个模块进行了详细设计，进行了大量的软件编写，调试和测试工作。最终完成此基于网络的高清数据视频矩阵卡项目。

本论文实现的高清数据视频矩阵卡搭配视频切换控制客户端程序已经可以实现对高清串行数字视频信号的切换输出，取得了一定的社会价值和经济效益。但视频矩阵卡本身还存在着功能和稳定性方面的问题。在后续的工作中有以下值得重点关注并改进的地方：

- 1，系统传输延时较长，希望能进一步研究以缩短该系统的传输延时。
- 2，完善 web 页面的控制功能。现有的 web 页面功能不是很完善，希望能将所有的功能在 web 上实现，不需要客户端即可操作矩阵卡。
- 3，对现有矩阵卡系统进行更进一步的测试。

致 谢

论文和学业的完成，首先要感谢我的父母含辛茹苦把我养育成人，你们的养育之恩是我所有一切的基石。电子科技大学电子与通信学院李晓峰老师，徐进老师对我的论文工作进行了耐心和详细的指导，两位老师深厚的知识积淀和认真负责的工作态度对我产生了深远的影响，还要感谢电子科技大学各位老师三年来对我的培养和帮助。同时特别感谢英夫美迪公司苏大同老师，公司各位领导和同事的支持和帮助，有了你们的帮助和共同努力，论文工作才得以顺利完成。

参考文献

- [1] 孔庆宇.视频矩阵的发展与现状浅析.中小企业管理与科技(下旬刊).2011 年第 02 期.
- [2] 李茜.浅论数字矩阵.中国交通信息产业.2007 年第 11 期.
- [3] 张淑梅、香亚楠.视频矩阵的网络化变革—嵌入式网络视频矩阵.第二十一届中国(天津)2007IT、网络、信息技术、电子、仪器仪表创新学术会议论文集.2007 年.
- [4] 美国 Infinova(英飞拓)有限公司.数字矩阵能取代模拟矩阵吗? 中国交通信息产业.2007 年第 01 期.
- [5] 章俊.一种非压缩数字视频交换矩阵的设计与实现.科技创新导报.2009 年第 30 期.
- [6] 美国 Infinova(英飞拓)有限公司期刊.Infinova(英飞拓)以太网控制联网型视频矩阵系统和数字视频传输系统解决方案.中国交通信息产业.2004 年第 11 期.
- [7] 钟世强.基于 IP 交换的大规模数字视频切换矩阵技术.华东电力.2003 年第 08 期.
- [8] 刘仑.基于网络的数字视频矩阵技术应用研究.西安建筑科技大学.硕士学位论文.2007 年.
- [9] 董天烨.网络数字视频监控系统的设计及实现.上海交通大学.硕士学位论文.2008 年.
- [10] 董海防.基于同轴视控技术的视频矩阵切换/控制系统设计.中国海洋大学.硕士学位论文.2009 年.
- [11] 黄娜.支持模拟矩阵的视频监控系统的设计与实现.北京交通大学.硕士学位论文.2007 年.
- [12] 常宇鹏.IPCAM 实时高清视频流传输的研究与实现.电子科技大学.硕士学位论文.2010 年.
- [13] 王纯纯.基于 DaVinci 平台的网络视频终端的研究与实现.山东大学.硕士学位论文.2010 年.
- [14] TI: TMS320DM646x DMSoC Multichannel Audio Serial Port (McASP) User's Guide
- [15] TI: TMS320DM6467 Digital Media System-on-Chip
- [16] TI: TMS320DM646x DMSoC Ethernet Media Access Controller (EMAC) (MDIO) Module User's Guide
- [17] TI: TMS320DM646x DMSoC Video Data Conversion Engine (VDCE) User's Guide
- [18] TI: TMS320DM646x DMSoC Video Port Interface (VPIF) User's Guide
- [19] Society of Motion Picture and Television Engineers: SMPTE 274M-2005: Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates

- [20] Society of Motion Picture and Television Engineers: SMPTE 292M-1998: Bit-Serial Digital Interface for High Definition Television
- [21] Society of Motion Picture and Television Engineers: SMPTE 291M-1998: Ancillary Data Packet and Space Formatting
- [22] Society of Motion Picture and Television Engineers: SMPTE 372M-2002: Dual Link 292M Interface for 1920 x 1080 Picture Raster
- [23] ITU-R Recommendation BT.656: Interfaces for digital component video signals in 525-line and 625-line television systems operating at the 4:2:2 level of Recommendation ITU-R BT.601 (Part A).
- [24] ITU-R BT.709-5: Parameter values for the HDTV standards for production and international programme exchange. April, 2002.
- [25] RTP: A Transport Protocol for Real-Time Applications .Network Working Group.July 2003.