MASTER THESIS XIDIAN UNIVERSITY

西安电子科技大学

硕士学位论文

音视频信号采集压缩及传输系统的设计与实现

作者姓名 高兴鹏

指导教师姓名、职称 那彦 教授

申请学位类别 工学硕士

学校代码 10701 学 号 1602120692
分 类 号 TP31 密 级 公开

西安电子科技大学

硕士学位论文

音视频信号采集压缩及传输系统的设计与实现

作者姓名：高兴鹏

一级学科：电子科学与技术

二级学科：电路与系统

学位类别：工学硕士

指导教师姓名、职称：那彦 教授

学 院：电子工程学院

提交日期：2019 年 6 月

Design and Implementation of Audio and Video Signals Acquisition Compression and Transmission System

A thesis submitted to

XIDIAN UNIVERSITY

in partial fulfillment of the requirements

for the degree of Master

in Circuits and Systems

By

Gao Xingpeng

Supervisor: Na Yan Title: Professor

June 2019

西安电子科技大学
学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安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。与我一同工作的同事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了谢意。

学位论文若有不实之处，本人承担一切法律责任。

本人签名：高兴鹏　　日　　期：2015年6月14日

西安电子科技大学
关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权属于西安电子科技大学。学校有权保留送交论文的复印件，允许查阅、借阅论文；学校可以公布论文的全部或部分内容，允许采用影印、缩印或其它复制手段保存论文。同时本人保证，结合学位论文研究成果完成的论文、发明专利等成果，署名单位为西安电子科技大学。

保密的学位论文在___年解密后适用本授权书。

本人签名：高兴鹏　　导师签名：

日　　期：2015年6月14日　　日　　期：2019.6.14

摘要

随着数字技术的发展，音视频技术正在进入人们的工作和生活，提高了人们的工作效率和生活水平。音视频系统包括对声音和图像信号的采集、处理、存储或者传输，广泛应用于通信、娱乐和安防等领域。目前，音视频系统已经应用在了视频监控、ATM、医疗设备、和许多手持设备上。随着音视频技术在嵌入式平台上的广泛应用，嵌入式音视频采集传输系统已经成为研究热点。

本文设计并实现了音视频信号采集压缩及传输系统，其主要内容如下：

该系统采用 TI 的 TMS320DM365 处理器，加上相应的外设，可以采集音视频信号，进行模数转换，对转换后的音视频数据进行压缩封装以及传输。并在此基础上扩展了许多其它功能。

通过研究嵌入式 Linux 软件编程技术和 DaVinci 技术，实现了对音视频信号的采集和压缩。首先在嵌入式硬件平台上搭建起嵌入式 Linux 系统，然后详细分析 DaVinci 技术中的 DVSDK（音视频软件开发套件），设计了包括视频捕获线程、视频编码线程、视频写线程和音频线程的多线程程序，实现了音视频信号的采集和压缩，并在此基础上作进一步的开发，实现了参数配置、视频添加时间字幕的功能。

基于 FFmpeg 音视频框架设计并实现了音视频压缩数据的封装和传输。分析了 TS 流格式和 FFmpeg 音视频框架的结构，通过调用 FFmpeg 中的函数设计程序，将音频压缩码流和视频压缩码流封装成了 TS 流，并实现了音视频同步，然后通过 FFmpeg 的内存数据操作方法，结合 EMIF 驱动把封装后的 TS 流通过 EMIF 接口从 DSP 传输给 FPGA。

最后对整个系统进行了测试，正确并高效的完成了音视频信号的采集、压缩、封装并从 DM365 传输给 FPGA，然后 FPGA 对音视频数据进行缓存，传输给上位机播放。并且实现了上位机对音视频参数的配置和视频添加时间字幕的功能。

本系统采用了 DaVinci 技术的硬件以及软件和 FFmpeg 音视频框架，使开发者可以更便捷和快速的实现二次开发，并且拥有很好的移植性。

关 键 词：嵌入式， 音视频， DaVinci， Linux， FFmpeg

ABSTRACT

With the development of digital technology, audio and video technology is entering into people's work and life, improving people's work efficiency and living standards. Audio and video system include the processing, storage and output or display of sounds and images that has widely been used in communications, entertainment, and security. Currently, audio and video system has been used in video surveillance, ATM, medical devices, and many handheld devices. With the widely use of audio and video technology on embedded platforms, embedded audio and video acquisition and transmission systems have become a research hotspot.

The audio and video signal acquisition compression and transmission system is designed and implemented in this paper. The main contents are as follows:

Adopting TI TMS320DM365 processor, equipped with corresponding peripherals, the system can acquire audio and video signals, and convert analog signal into digital signal, then compress and package the digital audio and video data and transmit it. And on this basis, many other functions have been extended.

By studying embedded Linux software programming technology and DaVinci technology, the acquisition and compression of audio and video signals are realized in this paper. Firstly, the embedded Linux system is built on the embedded hardware platform, Then the DVSDK (audio and video software development kit) of DaVinci technology is analyzed in detail, and the multi-threading program including video capture thread, video coding thread, video write thread and audio thread is designed, realizing the acquisition and compression of audio and video signals. And further developed on the basis of this, realizes the functions of parameter configuration and video added time subtitles.

Based on the FFmpeg audio and video framework, the packaging and transmission of audio and video compressed data is designed and implemented. The TS stream format and the structure of the FFmpeg are analyzed. By invoking the function in FFmpeg, the program can pakage the audio compressed code stream and the video compressed

code stream into TS streams, and realize audio and video synchronization. Then, by the method operating data in memory, combined with the EMIF driver, packaged TS stream is transmitted from the DSP to the FPGA through the EMIF.

Finally, the whole system was tested, and the audio and video signals were acquired, compressed and packaged correctly and efficiently, and transmitted from the DM365 to the FPGA. Then the FPGA buffered the audio and video data and transmitted it to the host computer for play. And realize the configuration of the audio and video parameters of the upper computer and the function of adding time subtitles to the video.

The system uses DaVinci technology hardware and software and FFmpeg audio and video framework, enabling developers to achieve secondary development more easily and quickly, and has a good portability.

Keywords: embedded, audio and video, DaVinci, Linux, FFmpeg

插图索引

表格索引

符号对照表

符号	符号名称
fps	帧每秒
bit	比特
KB	千字节
MB	兆字节
bps	比特每秒
Hz	赫兹

缩略语对照表

缩略语	英文全称	中文对照
DVR	Digital video recorder	数字视频录像机
SOC	System On Chip	片上系统
MPEG	Moving Picture Experts Group	动态图像专家组
DSP	Digital Signal Processing	数字信号处理器
FPGA	Field Programmable Gate Array	现场可编程门阵列
AAC	Advanced Audio Coding	高级音频编码
DVSDK	Digital Video Software Development Kit	数字视频开发套件
RAM	Random-Access Memory	随机存取存储器
TS	Transport Stream	传输流
ARM	Advanced RISC Machines	高级精简指令集计算机
API	Application Programming Interface	应用程序编程接口
IP	Internet Protocol	网络协议
EMIF	External Memory Interface	外部存储器接口
DCT	Discrete Cosine Transform	离散余弦变换
CPU	Central Processing Unit	中央处理器
TCP	Transmission Control Protocol	传输控制协议
UDP	User Datagram Protocol	用户数据报协议
NFS	Network File System	网络文件系统
VFS	Virtual File System	虚拟文件系统
IPC	Inter-Process Communication	进程间通信
VPSS	Video Process Sub-System	视频处理子系统
V4L2	Video For Linux 2	Linux 视频设备驱动
ALSA	Advancec Linux Sound Architecture	高级 Linux 声音架构
GOP	Group Of Pictrues	图片组
DTS	Decoding Time Stamp	解码时间戳
PTS	Presentation Time Stamp	显示时间戳

目录

第一章 绪论

1.1 课题的研究背景和意义

自 20 世纪四五十年代以来，计算机和互联网技术应运而生，如今人们已进入信息化时代。一开始，由于信号只能由模拟量进行传输，无法传输视频，人们通过语音或文字传递信息。随着视频编解码技术的发展，视频技术不仅在工业上得到广泛应用，而且还进入了千家万户。而且随着嵌入式技术的发展，音视频技术可以在嵌入式平台上实现，让音视频技术更广泛地普及开来。目前，嵌入式音视频技术已进入了人们的工作和生活，提高了人们的工作效率和生活水平。嵌入式音视频系统包括对声音和图像信号进行采集，然后对采集到的音视频数据进行压缩或封装等处理，最后进行输出或者显示[1]。嵌入式音视频系统可以应用于通信、娱乐和安防等领域，目前音视频系统已经应用在了视频监控、ATM、医疗设备、和许多手持设备上。同时，随着嵌入式技术和多媒体技术的发展，DSP 和 FPGA 这类数据运算芯片可以对大量的数据进行快速、高效的运算，视频压缩技术不断的推陈出新，音视频系统正朝着嵌入式、数字化的方向迅猛发展[2]。嵌入式音视频技术将得到越来越广泛的应用，因此对嵌入式音视频技术的研究正在全面展开。并且随着人工智能技术的突破和物联网的兴起，嵌入式音视频技术必定会走向智能化的道路[3]，另外可以通过视频技术来传递信息打造万物互联的世界，所以嵌入式音视频领域具有很大的发展前景。

1.2 国内外研究现状

基于计算机技术的嵌入式系统可以为专门的应用进行设计，是一种特殊的计算机系统。我们可以裁剪嵌入式系统的硬件和软件，来让它适用于某一特定的应用。嵌入式系统具有更高的可靠性，允许系统在恶劣环境中工作并可以应对突然断电的情况。它具有低成本、小尺寸、低功耗的特点，让系统能更好的嵌入到其它设备中。对于音视频系统来说，它必须还要具有很高的实时性[3]。嵌入式处理器经历了从简单到复杂，从单核到多核的演变，其数据处理能力不断提高。其中数字信号处理器 DSP 可以高效的进行大量的数据运算，让音视频算法的运行成为可能。而 ARM 处理器作为高性能的处理器，广泛应用于嵌入式系统设计中[4]。

在模拟信号时代，信号波形在传输时容易受到干扰，其存储也需要巨大的容量。随着数字技术的发展，音视频技术也获得了大幅的进展。音视频信号先经过采样量化得到音视频数字信号，再对音视频数字信号进行压缩编码减少其数据量。其中，对音视频数字信号的压缩技术不断在发展，压缩比率不断提高。压缩标准从一开始的

MPEG-1 发展到了现在的 MPEG-4，在保证音视频质量的同时，不断提升压缩比率，让高清或超高清的视频的传输和存储成为可能。而且鉴于音视频压缩算法的实现复杂度高，各个厂商都研发出了音视频开发方案，把音视频压缩算法做成一个模块供开发人员使用。例如 TI 开发的 DaVinci 音视频开发方案就是一个封装了大量的算法，能满足各种数字视频开发需求的综合解决方案[5]。

数字视频技术的开发比较复杂，包含有众多的采样或压缩标准，而且随着技术的发展，这些标准也在不断的推陈出新，给开发人员的使用带来了困难。已有的数字视频的实现又只能在特定的硬件条件和操作系统上运行，开发人员必须手工编程音视频的底层代码，这使得开发过程很复杂耗时[6]。为了应对市场上的需求，TI 提出了针对数字视频的 DaVinci（达芬奇）技术。DaVinci 技术为各种数字视频开发需求提供全面的解决方案，包括嵌入式平台上的处理器，软件和多媒体应用程序开发工具。

DaVinci 处理器采用 ARM+DSP 的双核架构，ARM 作为中央处理器负责运行应用程序，而 DSP 专门运行视频压缩算法。DaVinci 处理器芯片有丰富的外设接口，给音视频的采集和数据的传输提供了便捷。DaVinci 软件集成了嵌入式开发和数字视频开发所需的基本软件，包括嵌入式内核、音视频采集和编解码算法，并在这些音视频功能模块的基础上提供了编程接口（API），让开发人员不用了解底层代码就可以开发音视频程序[7]。现在，DaVinci 技术在各种音视频场景下有广泛的应用，例如视频监控系统、视频会议系统、视频对讲机等[8]。

DaVinci 技术提供了采集压缩的 API，但一整套的音视频开发会包含采集压缩、封装传输、或者添加字幕等功能，DaVinci 技术只能作为开发音视频系统的平台，然后在其基础上做进一步开发。为了方便音视频软件的开发，目前有很多开源的音视频框架可供开发人员使用，例如 FFmpeg、gstreamer 等。FFmpeg 作为领先的音视频框架，提供了丰富的音视频数据处理接口，是目前应用最为广泛的音视频开发解决方案。许多视频播放器和直播软件都是基于 FFmpeg 开发。

视频监控系统作为嵌入式音视频系统最具代表性的应用，下面对视频监控系统的发展和现状进行介绍。

视频监控是安防领域最重要的一部分，它不仅应用于工业流程监控和交通管制，也应用于金融机构、无人值守和报警系统。视频监控技术包括了很多技术领域，例如数字信号处理，视频编解码，视频传输，视频存储和网络通信等。它涵盖了嵌入式音视频技术所包括的全部内容，并对这些功能的实现有很高的要求，例如压缩比率，传输效率等。视频监控系统经过了长时间的发展，一开始是传输模拟信号的闭路电视监控系统，然后是数字监控系统，可以编码音频和视频数字信号，并且对其存储或者传输。现在已发展成为了智能网络视频监控系统，它是一种基于 Internet 技术的大型监

控系统。传统的模拟监控系统存在传输距离短，联网能力不足，模拟视频信号数据存储量大等诸多局限性。随着视频压缩技术的不断完善，基于个人计算机的视频监控系统在 20 世纪 90 年代迅速发展。然而它有许多缺点，例如缺乏稳定性，高功耗。与前面提到的系统相比，基于嵌入式硬件平台的视频监控系统不仅运行更可靠，而且更易于安装和维护，所以，对嵌入式视频监控系统研发的投入不断加大。

1.3 论文的主要内容及章节安排

本文在嵌入式 Linux 系统上做音视频开发，实现了一个嵌入式音视频采集传输系统。芯片采用的 TI 的 DaVinci 视频处理器 TMS320DM365，嵌入式 Linux 开发采用主机和开发板交叉开发模式，先在开发主机上安装 Linux 开发环境和 TI 的 DVSDK，然后通过仿真器和网线往板子上烧写 U-boot 和 Linux 内核，启动嵌入式系统，再通过 NFS 测试应用程序的功能。

本设计实现的整体功能是通过摄像头采集视频信号，通过麦克风或音频线采集音频信号，视频信号和音频信号分别经过 TVP5146 和 AIC3101 进行模数转换，以 DVSDK 为平台作开发，捕获并压缩音频和视频数字信号，视频编码成 H.264 格式压缩数据，音频编码成 AAC 格式压缩数据。然后调用 FFmpeg 的库函数将音频和视频流封装到 TS 流中，通过 EMIF 接口将 TS 流传输到 FPGA，经 FPGA 缓存后传到上位机进行播放。本文设计中要完成的工作包括以下部分：

（1）分析 DVSDK 的结构和提供的接口，设计音视频信号采集编码程序。通过 EMIF 接口接收上位机传给 DM365 的音视频参数信息，并配置音视频的采集和编码。通过 SPI 接口接收上位机传给 DM365 的时间信息，按照这个时间校正嵌入式系统时间，并把时间字幕添加到视频上。

（2）把 FFmpeg 移植到嵌入式平台上，并调用 FFmpeg 的库函数把音视频编码数据封装成 TS 流，按照 FFmpeg 的内存操作把 TS 流通过 EMIF 接口传输给上位机。

本文的创新点有以下几个方面：

（1）基于 FFmpeg 音视频框架完成了音视频压缩码流的封装，并通过 FFmpeg 的内存操作方法，和 EMIF 驱动相结合，把封装成的 TS 流数据正确的从 DM365 传输给 FPGA 的 FIFO。FFmpeg 是一套成熟的音视频开发开源框架，提供了非常丰富的音视频开发接口，在此基础上做开发，大大提高了程序的可扩展性，有利于程序的二次开发。

（2）通过 EMIF 接口接收上位机传过来的音视频参数配置信息，并设计采集编码程序对音视频参数进行动态赋值，实现了上位机对音视频参数，例如帧率、码率、

分辨率等的控制。

（3）设计程序通过 SPI 接口接收上位机发送的时间信息，并校正 DM365 上的系统时间。并且提取校正后的系统时间，在视频采集程序里设计叠加时间字幕的程序，让 YUV 视频叠加上时间信息，然后进行压缩并封装成 TS 流，传输到上位机播放带有时间字幕。

本文用六章的内容详细介绍了这个音视频系统的设计与实现，其章节安排如下：

第一章为绪论，详细介绍了嵌入式音视频技术的研究背景和发展状况，以及本文的主要工作及结构安排。

第二章介绍嵌入式音视频系统开发的相关技术，详细介绍了 DaVinci 技术，和本系统涉及到的基于 DaVinci 技术的视频处理器和 DVSDK（音视频开发套件）。并介绍嵌入式 Linux 系统和 Linux 编程的知识。

第三章介绍音视频采集编码的原理和标准，设计音视频采集编码程序，并介绍音视频参数配置和时间字幕添加的设计与实现。

第四章介绍 TS 流的格式和 FFmpeg 的结构，并详细介绍如何调用 FFmpeg 的库函数实现音视频数据的封装以及通过 EMIF 传输。

第五章是介绍对本文设计的音视频程序的功能的测试与结果分析。

第六章为总结与展望。

第二章 嵌入式音视频采集传输系统开发的相关技术

本章将介绍嵌入式音视频系统开发涉及到的相关技术，首先介绍系统所用到的DaVinci 技术，包括 DaVinci 视频处理器和 DaVinci 软件。然后介绍我们采用的操作系统即 Linux 操作系统，包括 Linux 内核的功能以及如何编译和移植，因为本文设计的程序是多个进程，多个线程，所以要介绍 Linux 上的多进程和多线程编程。最后给出该音视频系统的总体方案设计，包括硬件系统的方案设计和软件系统的方案设计。

2.1 DaVinci 技术

为了便于嵌入式音视频系统的开发，TI 公司推出了一套音视频开发解决方案即DaVinci（达芬奇）技术，它封装了大量的算法，能满足各种数字视频开发需求。

2.1.1 DaVinci 技术概述

随着生活水平的提高和科学技术的发展，各种智能终端和视频图像电子产品进入了普通百姓的生活，人们对图像/视频处理的要求越来越高。然而早期没有一个整体的包括硬件和软件的音视频开发方案，而音视频技术的实现又较为复杂，所以造成了嵌入式音视频开发复杂，开发周期长的问题。针对存在的问题和人们的需求，各个音视频领域的公司都在投入大量的研发人员加紧嵌入式音视频技术的开发。其中 TI 公司提出了 DaVinci 技术，它提供了一个完整的音视频开发方案，让开发者不需掌握音视频数据压缩等音视频技术的底层实现，就可以开发音视频系统，大大简化了开发流程。

由于嵌入式音视频系统中不仅有复杂的控制任务，例如音视频信号的捕获、传输、用户界面等，也有计算密集型的视频编码任务。考虑到这一点，DaVinci 视频处理器采用了双核架构，让 ARM 和 DSP 分别负责相应的任务。首先 ARM 端的应用程序控制视频子系统通过驱动程序捕获图像数据，然后把数据放到 ARM 和 DSP 的共享内存，向 DSP 发信号，DSP 端通过压缩算法对图像数据进行硬编码，编码结束后，再将数据放到共享内存，并通知 ARM 端取走数据进行传输等操作。DM365 中的 ARM 端实现音视频应用程序的运行，而 DSP 端只实现视频编码。音视频信号采集过程和音视频压缩算法都集成为了易于调用的 API 函数。因此，对开发者来说，利用 DaVinci 技术只需完成音视频应用程序的设计，不用浪费大量的时间对图像处理的各种复杂算法进行实现[9]。

2.1.2 DaVinci 视频处理器 TMS320DM365

TMS320DM365 是 TI 发布的一种视频处理器，是一个可以满足大多数音视频开发需求的高度集成的处理器芯片。它集成了 DSP 和 RISC 的优点，其 ARM+DSP 的双核架构能处理大量的数据，具有优良的音视频数据处理的能力。ARM 基于 RISC 指令集，适合进行控制任务，例如操控外部设备。DSP 专为实时数字信号处理而设计，可用来对音视频数据进行处理。基于 DM365 开发的音视频系统可以完成多种格式的音视频数据压缩，并以高速率传输出去。并且 DM365 还提供了许多接口和模块，让其可以与众多外设相连。在 DM365 中，DSP 来运行音视频算法例如编解码等，ARM 来处理与外设之间的数据交换，调用 DSP 中的音视频算法运行音视频应用程序。图 2.1 给出了 TMS320DM365 的功能框图，从图中可以看出，DaVinci 系列 TMS320DM365 芯片主要包括 ARM 处理器、DSP 处理器、DDR 内存和众多的外设接口[10]。

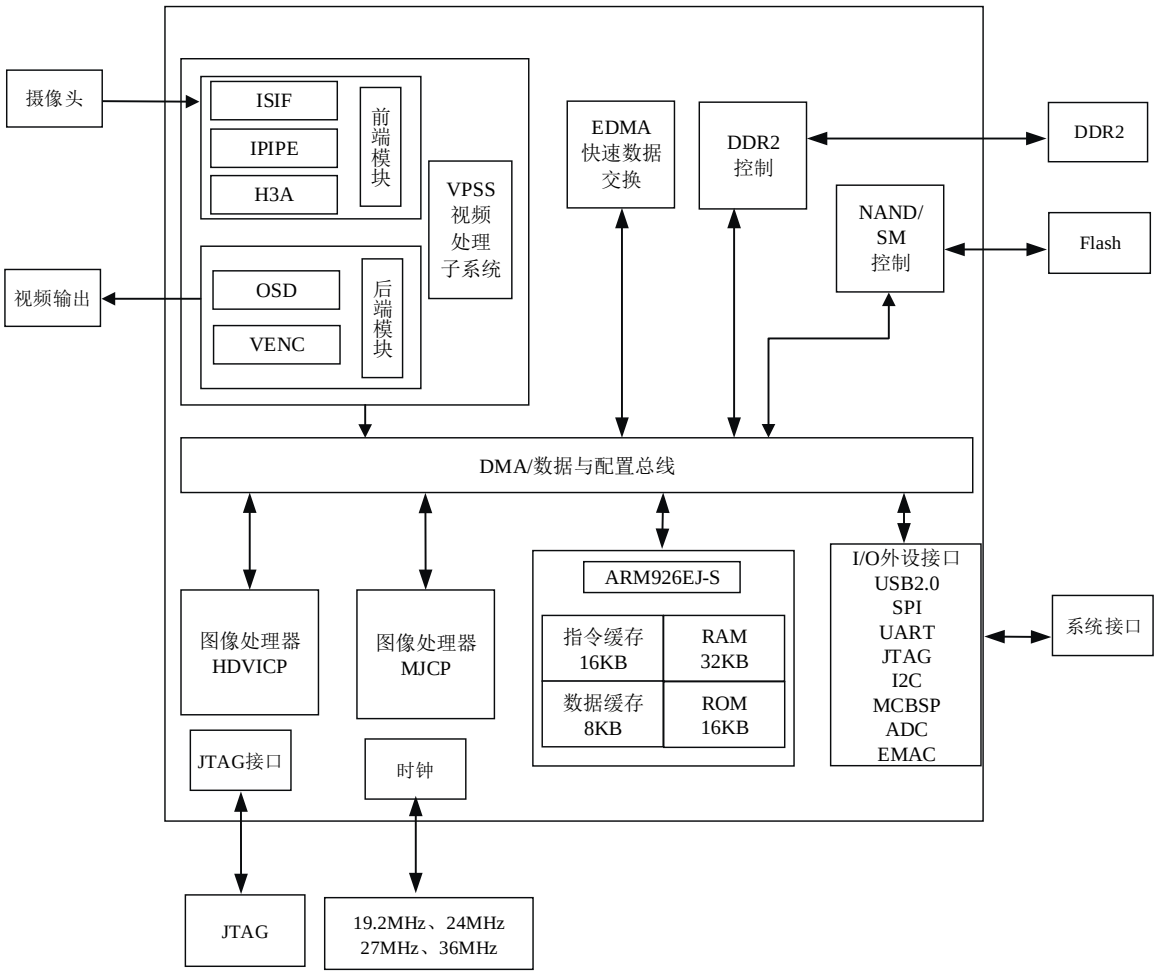


图 2.1 TMS320DM365 功能框图

相比之前的 DaVinci 芯片，TMS320DM365 具有以下的特点和优势：

ARM926EJ-S 内核达到了 300MHz 的高速率，其视频编码和解码任务由为加快数字视频运算速度而专门设计的 DSP 协处理器来执行，从而大大提高了系统性能。

DM365 支持 H.264，MPEG4 和其它多种编/解码器，可以灵活地应用于各种场景，并且可以与旧的视频编/解码器兼容。

DSP 提供两个乘法器（MAC）和一个算术运算逻辑单元（ALU），实现了数字信号处理的高性能和低功耗。

DM365 包含了大容量的内存，可以满足音视频程序运行需求，并且提供了各种设备接口，如 JTAG、SPI、EMIF 等。

2.1.3 数字视频软件开发套件（DVSDK）

数字视频开发套件（DVSDK）是 TI 基于 DaVinci 硬件开发的软件包，集成了音视频开发相关的软件，包括嵌入式开发用到的引导程序、内核源码、和音视频开发要用到的算法和接口。它提供了优化的、高效率的、随时可调用的音视频编码器，开发人员还可以根据自己的需要添加编码器。并且通过 DMAI 模块为音视频开发提供了丰富的 API，有利于音视频应用的开发[11]。它提供的模块主要包括 Linux 内核源码、音视频程序开发算法和 API 等，其主要功能模块如下表。

表 2.1 DVSDK 功能模块

DVSDK 模块内容	类型	主要功能
Linux kernel 2.6.32.17	Platform Support Package	Linux 内核，嵌入式操作系统
Boot loaders (u-boot,UBL) and their flashing utilities	Platform Support Package	引导、加载并启动 Linux 内核
Codec Engine Framework	Multimedia Package	编码器引擎
CMEM	Multimedia Package	ARM 和 DSP 通信
DMAI	Multimedia Package	多媒体应用程序接口
accelerated codecs	Multimedia Package	音频和视频编码器
demos	Multimedia Package	音视频例程
Gstreamer	Multimedia Package	多媒体开发框架
Qt	Graphics Package	应用程序界面开发框架

DVSDK 提供了 UBL 和 u-boot，它们可以初始化嵌入式硬件，并引导加载嵌入式 Linux 内核并启动。DVSDK 也提供了 Linux 内核源码，开发人员可以采用 TI 预编译

的 Linux 内核镜像，也可以重新配置内核并编译。

音视频技术的开发人员可以通过 Codec Engine 来和 DSP 编码器建立联系，Codec Engine 可以实例化 DSP 中的音视频算法，还提供接口用于与 DSP 算法进行交互。其中编解码器算法分为了四类：视频、图像、语音和音频。每个编码器提供一组 API。因此，开发人员可以在 Codec Engine 中替换或添加编解码器，而无需更改调用编码器的代码，只需更改某些配置，建立调用编码器 API 和编码器的联系即可[12]。ARM 端的应用程序需要通过 Codec Engine 的 API 接口来调用 DSP 里的音视频算法程序，在这期间，ARM 和 DSP 必须能够通信，才能让程序正确运行。

DSP 端与 ARM 端如果有大量的数据交互，例如音视频数据的交互，则需要通过 CMEM 来实现。在操作音视频数据时，需要用 malloc 函数来分配内存。malloc 函数分配的是虚拟内存，用页表映射为物理内存时不连续，然而嵌入式系统中的算法接口又要求连续的地址。CMEM 模块可以在内存映射时用一种机制来实现分配连续的物理内存空间。在启动内核后，CMEM 模块会通过动态加载的方式加入到内核中。通过调用 CMEM 里的库函数，可以分配 DSP 和 ARM 的共享内存，来用于 DSP 和 ARM 之间交换数据。

为了给开发者提供简便易用的接口，DVSDK 提供了 DMAI（Digital Media Application Interface）模块，DMAI 在 Codec Engine 基础上封装了大量的音视频应用接口，包括音视频信号捕捉、回放、帧拷贝、修改尺寸等多个多媒体开发接口，开发者不必了解音视频算法如何实现，就可以根据自己的需求调用相应的模块，提高了开发效率。

2.2 嵌入式 Linux 操作系统

2.2.1 Linux 操作系统概述

Linux 和 MacOS 或者 Windows 一样，是一种操作系统。它是许多软件模块的组合，包括了一系列帮助用户管理计算机硬件的程序，另外，它也可以存放并运行应用程序，让用户可以完成一些任务，例如写文档、浏览网页或者发电子邮件等。由于 Linux 系统和其它操作系统相比具有良好的特性，功能强大，开发能力强，所以很受大学里的老师学生和公司的开发人员欢迎。由于 Linux 强大的功能和开放性，许多大公司都投入到 Linux 系统的开发研究中来，让 Linux 系统更加完善，拓展了 Linux 系统的市场，极大地促进了 Linux 的商业化。Linux 系统兼容了 UNIX 的大部分功能，但是 Linux 系统和 UNIX 系统相比还有很多优点。例如 UNIX 过于庞大，不适用于 PC，而 Linux 对 UNIX 进行了精简，而且继承了 UNIX 支持多用户、多任务、多平台的优势。

Linux 系统开源，让世界各地的开发人员可以完善其功能，而且，在 Linux 基础上进行裁剪和修改，可以适用于多种场合，例如手机操作系统安卓，和嵌入式 Linux 系统都是基于 Linux 系统为核心[13]。这种开放性和灵活性使 Linux 系统在全世界范围内广受欢迎。到现在为止，Linux 系统还有很大的潜力。Linux 作为 Linux/GNU 操作系统的核心，大约 26%的电脑服务器以 Linux 作为操作系统，还有少量的个人电脑的操作系统采用了 Linux。

操作系统相当于用户程序和计算机硬件之间的媒介，它的作用是高效的利用计算机硬件来完成任务。操作系统会给应用程序分配资源，相当于资源分配器[14]。由于多个应用程序可以在计算机上同时运行，因此经常发生资源请求冲突。操作系统在分配资源时的一个目标是高效的使用计算机系统，因此设计操作系统以使其以最佳状态运行非常重要。

Linux 操作系统主要包括如下：

（1）内存管理

Linux 将内存分页管理，一页为 4KB，有时页面的大小也会随着处理芯片的不同而不同，分页管理的方式可以有效的减少内存碎片。在 Linux 中，分配给进程的是虚拟内存，大小为 4G。每个进程会保存一张页表，来把虚拟内存映射到物理内存。Linux 系统到程序运行需要的时候才会分配物理空间，对于暂时不需要的内容会保存到磁盘中，在程序运行到这一部分时会通过页面置换从磁盘置换到内存中来，这样极大的提高了系统内存的利用率。

（2）进程管理

在计算机系统上运行的程序叫做进程，一个应用程序会通过 fork()函数创建子进程。进程管理是操作系统处理同一时间有多个进程时的应对方法，它通过操作系统的多任务处理能力实现。在只有一个中央处理单元（CPU）的计算机系统中，通过在多个进程之间切换来实现多任务。

（3）文件系统

Linux 系统的一个重要特点是 Linux 系统支持多达 32 种文件系统。在 Linux 中，一切都是文件，一切资源包括硬件设备都以文件的形式出现在 Linux 系统中。Linux 中的文件系统是以树的结构呈现，根目录是最大的目录，然后一层一层地往下分为小目录。

（4）进程间通信

在多进程程序中，不同进程之间常常会有数据的分享，Linux 提供了多种进程间通信机制，例如管道、共享内存等来实现不同进程之间的通信。

Linux 还提供了 shell 程序，shell 是 Linux 系统中的用户者界面，我们需要通过 shell 将我们输入的指令传给内核，让内核控制硬件进行正确的工作。shell 给用户提

供了一个接口，在 shell 里可以运行其他程序。Linux 提供了很多 shell 命令，这些命令都是独立的应用程序，我们可以通过 shell 来运行这些应用程序。在 Linux 编程中，经常会用到 shell 脚本，shell 脚本是用特定的语法规则写的批处理程序，在 shell 里边执行。用户可以通过写 shell 脚本来实现程序运行逻辑，以达到我们所想要的处理目的。

2.2.2 嵌入式 Linux 内核

嵌入式 Linux 内核是 Linux 操作系统的核心，在内核的基础上又加上了桌面、终端等交互界面才成为了我们看到的操作系统。它为分配系统资源，管理硬件设备提供必要的服务[15]。内核完成了操作系统要执行的所有工作。

嵌入式 Linux 内核是 Linux 操作系统根据实际应用场景裁剪而来，例如在本文中的应用于音视频系统的嵌入式内核增加了音视频驱动模块的支持，而去掉了鼠标键盘、触摸屏驱动等无关的部分。嵌入式 Linux 内核的主要特点之一是它具有良好的实时性，适用于对实时性要求较高的音视频系统，并且核心规模较小。这些特点与嵌入式系统本身的特点密不可分。

Linux 内核版本经过了长时间的变迁，现在最新的版本为 Linux4.x，本文中用到的版本为 Linux2.6 版本。Linux 内核源代码量非常的大，有将近 2 万个文件，源代码的总数有几百万行。对于如此庞大的结构，如果不能分门别类的对 Linux 内核进行编写，那么 Linux 内核的组织结构将会无比复杂，对于想学习 Linux 源码的人增加了难度，也不利于 Linux 内核应对不同场景下的裁剪和修改。幸运的是，Linux 内核开发者从一开始就规划好了 Linux 内核源码结构，对不同用途的文件进行了分类放到了相应的文件夹，所以现在的 Linux 内核虽然无比庞大，但组织结构却并不复杂。Linux 内核源码中，实现不同功能的文件放到各自的文件夹中，例如文件系统、网络、安全、进程间通信就各自组成了一个文件夹存放其源码文件。源码的各级目录如表 2.2 所示。

表 2.2 内核源码目录

目录名称	作用描述
arch	不同 CPU 架构的源码
block	块设备的通用函数
documentation	说明文档
drivers	设备驱动程序
fs	VFS 和 Linux 支持的各种文件系统
include	内核源码和其它各组成部分的头文件
init	内核引导和初始化源码
ipc	进程间通信代码
kernel	内核管理的核心代码
lib	库函数
mm	内存管理子系统
net	网络系统
scripts	脚本文件，用来配置、编译内核
security	Linux 安全模块
sound	音频设备的驱动程序
usr	用户空间代码

2.2.3 内核的编译及移植

Linux 移植就是针对特定目标平台和特定应用场景修改 Linux 源码，然后编译并下载到对应的硬件平台以使其正确运行起来并能够执行项目所需的功能[17]。

在每个内核源码子目录中，都会有一个 Makefile 文件和 Kconfig 文件。Makefile 主要是按照内核的配置把需要编译的部分进行编译并链接成可执行文件；而 Kconfig 用于配置内核，针对项目要实现的功能对内核进行配置和裁剪，它是用来配置内核的源文件。

配置内核用到 make menuconfig 命令，输入该命令后，就会看到一个菜单，这就是内核的配置界面，其配置选项如表 2.3，其中主要对设备驱动进行配置。通过配置界面，可以选择视频 AD 芯片，例如本文中的嵌入式音视频系统采用的是 TVP5146，我们就可以选中此芯片，并去掉其它芯片。除此之外，还可以对驱动模块进行选择，对不需要的模块进行裁剪使内核更轻便。

表 2.3 内核配置目录

选项	作用
General setup	系统整体的配置
System Type	选择硬件系统类型
Bus support	总线配置
Boot options	选择启动选项
CPU Power Management	CPU 电源管理配置
Networking support	网络配置
Device Drivers	设备驱动配置
File systems	文件系统配置

配置界面中，[*]选项表示相应功能的文件被编译进内核中、<M>选项表示相应功能的文件被编译成一个模块，在内核启动时，可以按自己的需求动态的加载该模块，[]开头的选项表示没有编译进内核中。在该界面中，可以对编译选项进行选择，Y 表示编译进内核中，M 表示编译成模块，N 表示去除该部分。按照项目的需要进行配置之后，一级一级地退出界面，最后保存并退出。

内核的源码在 DVSDK 开发包中，其 Makefile 文件已经由 TI 公司写好，直接在 DVSDK 文件夹中编译并生成内核镜像。原始的 Makefile 是按默认的配置进行编译，它为了功能完善所以不够精简，有一些用不到的驱动会编译进去，如果想要根据自己项目的实际需要进行编译的话，就要进入配置页面重新配置。

使用如下命令编译内核：

```
cd ${DVSDK}
make linux
```

运行该命令后，会按照相应的内核配置编译内核，生成内核镜像文件，文件在 dvsdk_dm365_4_02_00_06/psp/linux-2.6.32.17/arch/arm/boot 文件夹下。

2.3 Linux 多进程和多线程编程

2.3.1 进程和进程调度

程序执行之后，系统中就会有一个进程。进程的生命周期不等，例如守护进程在开机时就一直运行，在程序终止时释放系统资源。进程的执行需要占用 CPU 时间，需要分配系统资源，在多个进程并行执行时需要操作系统对进程进行有效的调度[18]。

在进程执行的同时，如果有另外一个进程需要处理，操作系统就要按照其优先级来决定它的状态，例如一些 I/O 请求到达等重要事件需要处理时，系统就会让其进入执行状态。Linux 内核会判断进程的状态，例如是正在执行，还是处在被阻塞状态，并把这些信息存储到 task_struct 结构体中。此外，该结构体还包含进程标识符、调度信息、内存信息等。

在操作系统中，进程的状态大致有三种，操作系统会根据进程调度规则让进程进入不同的状态，如图 2.2 所示。

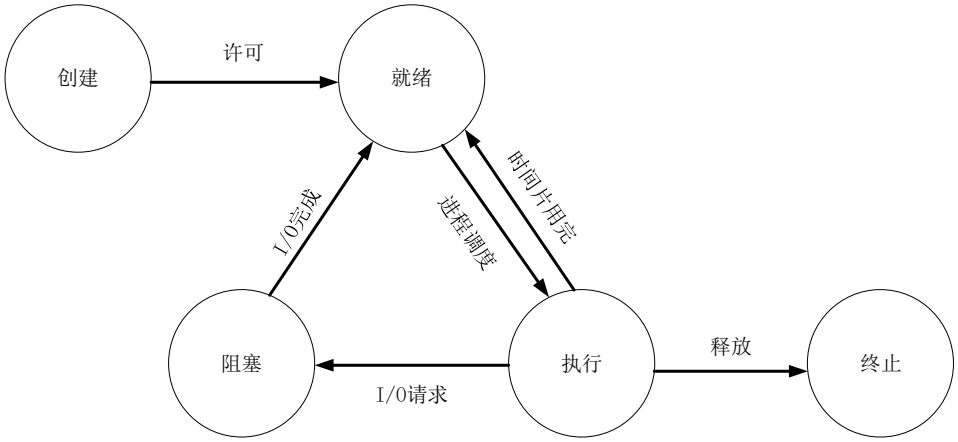


图 2.2 进程状态转换示意图

进程的转换如上图所示，创建进程并运行后，若 CPU 正在运行其它进程，则该进程会处于就绪状态，随时等待 CPU 调度。被调度后，CPU 会给进程分配时间片，进入执行状态，若时间片用完，则进入就绪状态。在进程运行期间，若遇到特殊事件例如 I/O 请求，进程会停止执行，进入阻塞状态，当事件完成之后，会回到就绪状态。进程运行完后，会停止占用 CPU 并释放内存资源，进程终止。

进程调度的方式有很多，分为抢占式和非抢占式，非抢占式表示若一个进程没有运行完就会一直执行，如果一个系统程序没有分配到时间，就会会导致系统崩溃。Linux 内核采用的是抢占式进程调度，由进程调度算法来安排进程的优先级，并分配给进程时间片，让其按优先级先后执行。Linux 内核会把 CPU 时间分成许多片，让进程按时间片长度先后运行，即时间多路复用，这样，从用户看来就像是在并行执行。同时，调度策略会根据任务的不同给进程分配优先级，并按优先级进行分类，优先级高的分到的 CPU 时间多。而且进程的优先级是变化的，调度程序会查询进程的状态，根据实际情况调整进程的优先级。通常情况下，阻塞等待 I/O 的进程被赋予更高的优先级，会耗用大量 CPU 时间的计算密集型进程被赋予较低的优先级。

2.3.2 Linux 多线程编程

一个进程包含一个或多个线程。如果一个进程只包含一个线程，那么进程中只有一个执行单元，我们将此类进程称为单线程。它们是经典的 Unix 进程。如果一个进程里不止一个线程，那么同时会进行多个任务，我们将此类进程称为多线程。其中多线程有以下特点。

进程有独立的内存空间，而一个进程之间的多个线程共享进程的内存空间。线程可以被认为是 CPU 调度的可执行单元，而进程是资源分配单元。多个进程也可以并行运行，但多线程程序有更高的效率[19]。并且，多线程上下文切换的开销比多进程低。在单核的处理器中，多个线程轮流使用 CPU 来实现并行，这种并行是“假并行”。在多核的处理器中，多个线程可以运行在不同的处理器内核中，而且还可以通过编程来控制某个线程在特定的处理器内核运行。在单处理器机器上，多线程也可以提高进程的响应能力。在单线程进程中，响应用户输入和其它任务无法同时进行。而对于多线程程序，可以让其中一个线程响应输入，复杂的计算或控制任务交给其它线程。

由于进程之间具有相互独立的数据空间，进程和进程之间必须要相互通信才能传递数据，这些通信方式由操作系统提供，例如 Linux 系统的管道通信、共享内存等。这会使程序变得复杂而不利于开发，而多线程共享同一进程的内存空间，所以多线程可以访问程序中的全局变量而不用特殊的通信机制，然而，多线程在操作同一个变量时如果不进行同步，可能会造成数据混乱。

如果某个操作不是原子操作，当多个线程操作共享数据时，可能会得到错误的结果。如果是原子操作，即一个线程在操作某个变量时，其它线程就无法对其进行操作，这时数据就不需要进行同步[20]。如果共享数据是只读的，也不需要同步。但是，当一个线程在操作某个变量的同时其它线程也可以操作时，就会导致数据结果和预期的不一致。例如，对于++i 操作，CPU 会从内存中取出数据放到寄存器，变量 i 在寄存器中加 1,再放回内存。如果两个线程同时进行++i 操作，其中 A 线程把 i 从内存中取出放到寄存器进行加 1 操作，这时 B 线程从内存中取出数据进行加 1 操作，由于 A 线程还未把加 1 后的 i 放回到内存，导致 B 线程取出的是未加 1 的 i，所以两次++i 实际上只加了一个 1，而导致结果错误。

实现线程同步有很多种方式，包括互斥量、条件变量、信号量等。互斥量方法是当某个线程操作变量时，会为这个互斥量加锁而禁止其它线程访问。条件变量方式是在满足特定条件时为变量加锁，当任务完成时为变量解锁。信号量方法是当线程访问变量时，信号量会加 1，而当信号量大于 0 时，其它线程无法访问该变量，以此来实现在某个时刻只有一个线程操作该变量。

2.3.3 进程间通信

在多进程程序运行时，很多情况下需要进程之间的通信，例如在本文中采集编码程序生成的音视频压缩码流需要传输给封装程序进行封装，这时就需要采集编码进程和封装进程之间通信。Linux 系统提供了很多进程间通信机制（IPC），包括管道、共享内存、socket 网络通信等。开发者可以根据实际要求选择进程间通信方法。其中 socket 网络通信实现的是不同机器上两个进程之间的通信。下面介绍另外两种方法：管道和共享内存。

由于进程有独立的内存空间，所以不同进程之间不能直接进行通信。如图 2.3，管道机制是在内核空间打通一条管道，这个管道为内核空间里的一个缓存区，大小为 4KB，让两个进程之间通过这条管道来传递数据。管道分为两种类型，无名管道（pipe）和有名管道（fifo）。无名管道用于一个进程和通过 fork()函数创建出的子进程之间通信。对于有名管道（fifo），被创建后会出现一个文件，这个文件并不是把里面的内容存到磁盘，而是指向内核空间的缓存区即管道。所以可以通过这个文件名来识别管道，从而完成不相关的进程之间的通信。

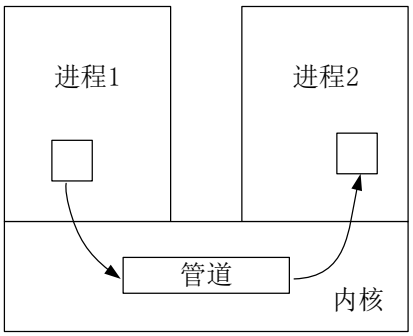


图 2.3 进程间通信示意图

共享内存映射机制可以将一段物理内存映射到两个不同的进程的地址空间，所以两个进程都可以去访问并操作这块内存里的数据，由于它不需要像管道一样把数据从用户空间拷贝到内核空间，所以效率更高。然而两个进程在同时操作同一块物理内存的数据时，会出现前面所说线程同步一样的问题，需要对数据加锁，否则会造成数据混乱。

2.4 音视频采集传输系统总体方案设计

本设计以 TI 公司 DaVinci 架构的 TMS320DM365 为处理器，在其外围加上必要的辅助芯片和电路组成一个音视频采集传输的硬件系统。其硬件总体方案设计为：由摄像头接收的图像信号，通过 TVP5146 转换为数字图像数据，通过视频前端送给

DM365 进行视频压缩。由麦克风接收的声音信号，通过 AIC3101 转换为数字音频数据，并送给 DM365 进行压缩。其中，DM365 通过 EMIF 接口接收上位机发送的音视频配置信息，并把在 DM365 上处理后的音视频数据通过 EMIF 接口传输给 FPGA 进行缓存，然后 FPGA 通过 PCI 接口将音视频数据传输给上位机。DM365 通过 SPI 接口接收上位机发送的时间信息，来校正 DM365 的系统时间。系统的总体设计如图 2.4。

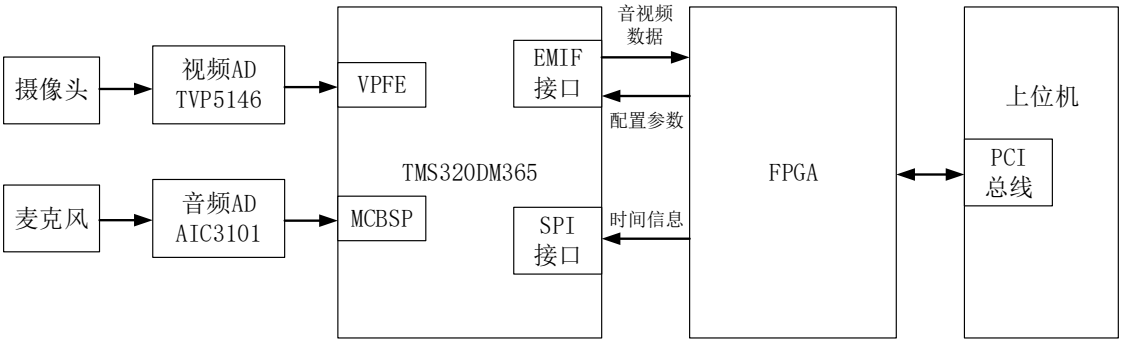


图 2.4 系统总体设计图

本文主要介绍系统的应用程序设计，其应用程序主要以 TI 的音视频软件开发套件（DVSDK）为开发平台，对采集的音视频信号进行压缩编码。其间，根据 EMIF 接口接收到的音视频参数信息对音视频的采集压缩进行配置，根据 SPI 接口接收到的时间信息校正 DM365 的系统时间，并把时间字幕添加到视频上。然后把音视频压缩数据封装成 TS 流，通过 EMIF 接口传输给 FPGA 的 FIFO。其应用程序总体设计如图 2.5。

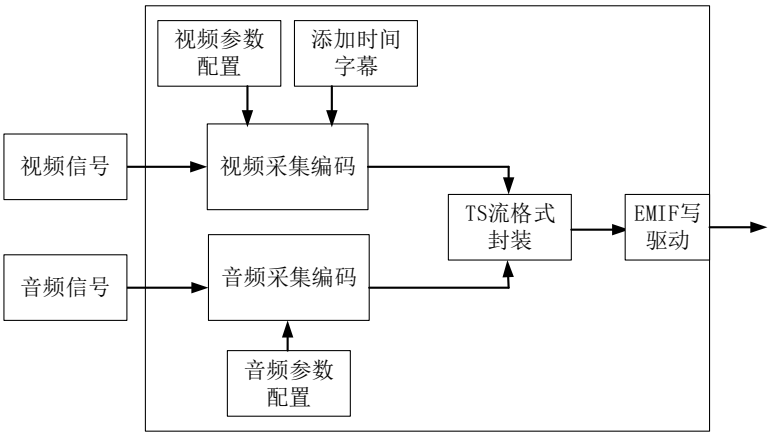


图 2.5 应用程序总体设计图

2.5 本章小结

本章对嵌入式音视频系统开发所涉及的相关技术进行了介绍。包括系统使用到的TI 的 DaVinci 架构的处理器和音视频开发套件，对嵌入式 Linux 系统以及在 Linux 系统上编程作了介绍，最后介绍了本系统的总体方案。为系统的实现提供了理论基础和具体思路。

第三章 基于嵌入式的音视频信号采集和编码的原理及实现

本章设计了基于嵌入式的音视频信号采集和编码的程序。首先介绍音视频采集的过程和音视频压缩编码的原理以及音视频压缩格式。然后详细介绍音视频采集编码的程序设计，最后对音视频参数配置和视频字幕添加的实现进行介绍。

3.1 音视频采集过程介绍

音视频采集的过程是：摄像头采集到的模拟视频信号传输给 TVP5146 芯片进行 A/D 转换，经 A/D 转换好的视频格式为 16 位的 YUV4:2:0 标准的数字视频信号。麦克风采集到的模拟音频信号传输给 AIC3101 芯片进行 A/D 转换，转换为 PCM 格式的音频采样数据。音视频采集底层程序封装在了 DSP 核内部，ARM 端的应用程序可以调用其接口。视频设备的驱动基于 V4L2 视频框架进行开发，来实现常用的数字视频接口[21]。所以，在设计应用程序时不需要考虑底层设备和底层代码的编写，只需要调用其接口即可。而音频设备的驱动程序是基于 Linux 音频框架 ALSA 编写的，可以实现音频信号的捕获和处理，它可以作为底层音频硬件设备和上层应用的桥梁，为应用程序提供相应的接口。

3.2 音视频压缩编码的原理及标准

音视频信号采集得到的数据的数据量非常大，如果直接存储或传输会非常困难，因此必须采用压缩技术减少其数据量。

3.2.1 数字视频压缩 MEPG 系列标准简介

MPEG 是音视频领域通用的一系列音视频标准。MPEG-1 应用于以 1.5Mbps 的码率对逐行扫描的视频进行编码。MPEG-2 是 MPEG-1 视频编码标准的扩展，该标准适用的码率为 5Mbit/s~20Mbit/s，它为隔行扫描视频的编码提供了额外的算法，支持高码率，并提供多声道环绕声编码。可以应用于高清数字电视、远程视频通话、多媒体计算机等高速率和高分辨率的场景中，弥补了 MPEG-1 压缩比低的缺点。MPEG-4 是 1999 年公布的标准，它适用于低码率的场景，例如普通画质的视频监控、视频电话、视频会议等。它除了提供低带宽的传输外，还提供了丰富的多媒体交互功能。它的实现方法也与前面的标准不同，它把图像的前景和背景分开，对于前景保留主体和边缘的细节，提供较清晰的画面，对于背景则大比例的压缩[22]。

3.2.2 视频压缩编码原理

视频压缩的目的是删除视频数据中冗余和不重要的信息，冗余类型如下：

（1）空间冗余：在同一帧内，邻近像素值具有相关性，一个像素点的值在一定程度上可以通过邻近像素点的值预测出来。

（2）时间冗余：一个像素的值与邻近帧上的像素的值相关，一个像素的值可以通过前一帧或后一帧上的邻近像素点的值在一定程度上预测出来。

空间冗余可以通过对像素块进行 DCT 即离散余弦变换来有效的减少。离散余弦变换通常用于信号和图像处理，特别是用于有损压缩数据。它具有强大的压缩特性，大多数信号信息集中在离散余弦变换的低频分量中。

运动补偿技术可以有效地减少帧间时间冗余，广泛应用于运动预测技术和编码技术。运动补偿的概念基于视频帧之间的运动估计，即，如果视频场景中的物体空间移动比较小，则物体的运动可以通过一个参数即运动矢量来描述，用于表示该物体所有像素点的平移。

在视频编码过程中，为使算法单元高效处理，视频数据分为以下几层：

运动序列：由表头+图片组+结束标志构成

图片组：由一系列图片构成

图像层：图片组中的一张图片

图像切片：图片中分割出的一部分，由宏块组成

宏块：运动补偿的基本单元

块：DCT 的基本单元，一个块由 8×8 的亮度信息或色度信息组成。

为了在去除空间冗余的基础上去除时间冗余，获得高压缩比、高质量的图像，MPEG 的一个图片组里有 I 帧、P 帧及 B 帧三种帧类型，分别简称为帧内图、预测图、及双向图，用于表示一个图像组帧序列画面，图像组中的图片还被分成了许多宏块和块，作为运动补偿和余弦变换的基本单元[23]。因为 I 帧包括了一张图像的全部信息，所以只有 I 帧也可以进行存取和传输，视频可以正常播放。但是为了满足高压缩比的要求，去除时间冗余，还要进行帧间编码以及块匹配运动补偿预测来减少数据量，所以除了 I 帧还要加入 P 帧和 B 帧。MPEG 的 GOP 结构如图 3.1 所示。

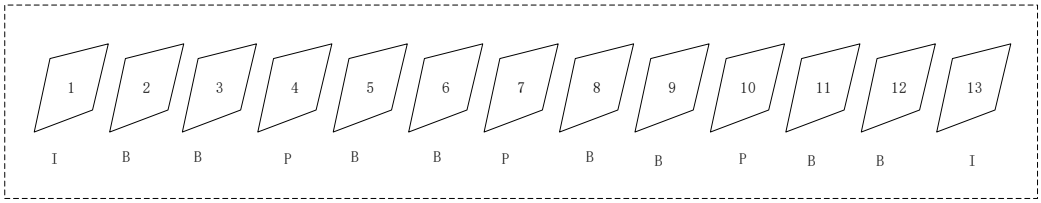


图 3.1 MPEG 的 GOP 结构图

I 帧的编码方式参考 JPEG 的编码方式，它不以任何其他帧做参考，用减少空间冗余的方法来压缩一帧图像。因此，通过单独解码 I 帧，就可以获得一幅图像的全部信息。I 帧的编码过程简单，如果图像为 RGB，则首先要进行色彩空间变换，将 RGB 转为 YUV。然后将图像分为小块，再对每个图块进行 DCT 离散余弦变换以去除空间冗余，再对其进行压缩编码，和原始数据相比大大降低了数据量。I 帧图像没有参考其它帧的图像，只对本帧图像进行编码，所以数据量最大，一个图像组的第一帧就是 I 帧。

P 帧参考了前面的 I 帧和 P 帧通过运动补偿技术保存与相邻帧的差值。运动补偿算法会遍历当前帧的宏块，并在相邻帧上搜索与此宏块相似或完全相同的宏块，计算出它们之间的位移并保存，并把这两个宏块不同的信息保存下来。由于 P 帧只保存与前一帧图像不同的部分，对于帧率为 25fps 的视频来说，相邻两帧图像的差异不大，所以这大大减少了数据量。然而，由于 P 帧参考的上一个 I 帧或 P 帧，如果上一个帧有错误，则这个误差会累积下去。

B 帧是进行双向预测得到的结果，其压缩比与 P 帧相比更高。它参考了前面的 I 帧和 P 帧以及后面的 I 帧和 P 帧通过运动补偿技术保存运动矢量和宏块之间的差值，也就是保存了与前后图像的差值，所以其数据量更小。视频压缩正是通过 I 帧、P 帧、B 帧的方式大大减少了时间冗余，让视频在保质的情况下可以占用更少的带宽和更小的存储空间。

3.2.3 视频压缩编码标准

因为不同的应用环境会使用不同的视频编码标准，所以制定了很多视频编码标准。MPEG-1 是最初统一制定的音视频编码标准，其压缩率不高，能满足当时的存储和传输需求。MPEG-2 标准提高了压缩比率，可适用于目前大多数场景，例如高清数字电视和多媒体视频等。由于 MPEG-2 的出色表现，MPEG-3 没有推广开来，而 MPEG-4 是针对低速率传播的场景[24]。H.264 由于在视频压缩上有良好的表现，是目前应用最为广泛的视频压缩标准。HEVC 制定的主要目标是进一步提高压缩率，相对于现有标准，在传输友好特性、容错性能及便于并行处理方面会有所提升。

以下是典型视频编码标准的简要介绍：

H.261 针对视频会议开发，H.261 为了满足低延迟的需求，只支持 I 帧和 P 帧。H.263 是对 H.261 的改进，应用于低码率（<64kbps）的视频通信。

MPEG-2 旨在提供高清电视质量的视频，应用于广播电视，视频通信，各种格式的数字视频存储等。MPEG-4 旨在满足下一代交互式多媒体应用以及传统多媒体应用的需求。提出了视频对象（VO）的概念，它是能够访问并操纵的码流中的实体。可以基于视频内容对视频数据进行压缩。

H.264 是目前较先进，应用最广泛的压缩格式，它大大提高了压缩比，和 MPEG-4 相比，压缩比提高了一倍多。H.264 的压缩技术让视频数据的存储和传输成本大大降低[25]。H.264 有非常高的压缩比率，同 H.263 等格式相比，能够节省大约 50%的码率。在视频监控领域，正在往高清方向发展，而且经常是多路传输，对网络带宽要求高，目前的高清视频监控也正在使用 H.264 格式，占用了较少的网络带宽。

由于视频分辨率和帧速率的增加（例如 8K 视频即 7680×4320 像素，120fps）,最近，开发出了 HEVC 标准，它是一种更有效的视频编码标准。在 H.264 的基础上，进一步提高了压缩比率。对于相同的视频质量，通过 HEVC 编码的视频的码率低于 H.264[26]。HEVC 的出现为大范围的视频监控、更大的视频数据传输和存储提供了支持。虽然现在 H.264 是主流，但在未来几年内，HEVC 的应用会越来越广泛。

3.2.4 音频压缩编码原理

数字音频压缩编码在保证解压后听觉方面不产生失真的前提下，减小数字音频的数据量，降低传输所需要的信道带宽，同时保持输入语音的高质量。大自然中的声音信号对人耳来说包含了很多无用信息，音频压缩通过去除这些冗余信号降低数据量，而不会影响听觉。

由于人耳的构造和大脑听觉中枢的原因，人类只能听到频率为 20Hz~20KHz 的声音，因此，可以去除声音信号中的低频和高频分量。此外，根据人耳听觉的生理和心理声学现象，当两个声音强度相差比较大时，音量大的声音会把音量小的声音遮掩而导致人们听不到音量小的声音，这被称为人耳听觉的掩蔽效应[27]。这样，弱音信号就不必传送，作为冗余信号去除，不会影响听觉。

目前的数字编码虽然具体方案和实现方法不同，但编码思路差别不大，如下图所示。

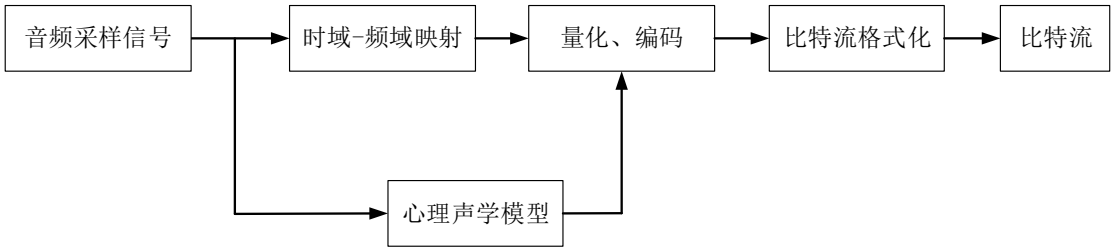


图 3.2 音频编码模型示意图

首先采样音频模拟信号得到原始的音频离散信号，然后进行频域转换，把人耳听觉范围之外的音频信号去掉。计算出人耳听觉的掩蔽效应的门限值，把低于这个门限值的弱信号去除，以上已将把音频信号的冗余部分去除，而不会影响音频质量。然后

把处理过的音频采样信号量化成为数字信号，然后进行压缩编码压缩其数据量，例如霍夫曼编码，然后以比特流的形式输出。

3.2.5 音频压缩编码标准

声音经过采样量化得到 PCM 格式的原始数据，再通过不同的压缩算法对数据进行压缩，得到不同格式的压缩数据。目前的压缩格式如下表所示：

表 3.1 音频压缩编码标准

名称	压缩比	应用情况
G.711	1:2	用于语音压缩
MP3	1:10	逐渐被淘汰
WMA	1:18	微软平台
AAC	1:18	目前应用最广泛

音频编码技术经过长时间的发展，压缩比率越来越大，并且可以提供多种音质的选择，以便应用于不同场景。例如音质高的音乐人耳听起来效果好，但相对应的就是数据量大。声音的采样率不同，也会造成编码后数据量不同。由于 AAC 格式音质高，且应用广泛，所以在本设计中，我们采用了 AAC 格式。

3.3 音视频采集编码程序设计

音视频采集编码部分有一个控制线程来创建其它线程，要创建的包括视频部分和音频部分，其中视频部分包括视频捕获线程、视频编码线程和写线程，音频部分只有音频线程。由于音视频程序要求实时性比较高，所以除了控制线程外，视频线程和音频线程要设置为抢占式，对其安排优先级。视频捕获线程和视频编码线程共享最高优先级，接下来是视频写线程和语音线程，优先级最低的是控制线程。由于 DVSDK 中提供了音视频编码算法、音频设备驱动和视频驱动，并提供了大量的 API 提供给开发人员使用。所以对于音视频开发只需设计程序运行的流程，其底层实现可以调用 DVSDK 提供的接口。

控制线程主要做一些初始化的工作，并创建线程。首先用 Dmai_clear()函数清除线程环境，然后解析命令行后面所跟的参数，根据命令行后面跟的参数的值来设置音视频参数变量，例如视频制式、音频码率和视频码率、分辨率等。创建从小到大的优先级，初始化 Codec Engine 模块，用 Dmai_init()函数初始化 DaVinci 多媒体应用接口 DMAI。创建用户界面，用户可以从显示屏上的用户界面中输入音视频参数。确定需要同步的线程数量，编码一个视频文件线程数加 3，编码一个音频文件线程数加 1。

初始化线程属性，并依此创建视频捕获线程、视频编码线程、写线程和音频线程。具体流程如图 3.3 所示。

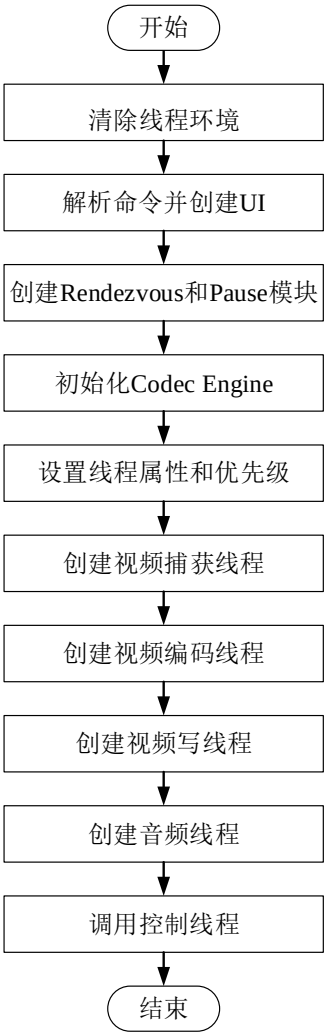


图 3.3 主线程流程图

3.3.1 音频线程程序设计

音频部分只有一个线程，对于视频部分由于是 DSP 硬件编码，需要通过 CMEM 模块进行 ARM 和 DSP 之间的数据传递，所以需要采集线程、编码线程和写线程的共同运行。而音频编码为软编码，只需从采集设备上获取数据，然后编码，写入到文件中即可。

音频线程初始化流程如图 3.4 所示：

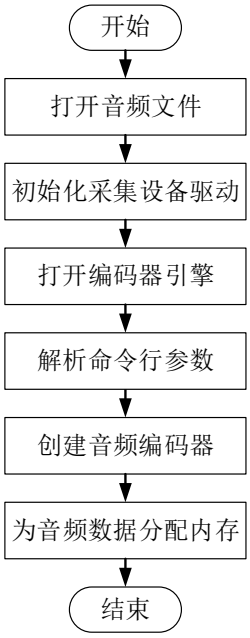


图 3.4 音频线程初始化流程图

打开待存储的目标音频文件，以便写入 Linux 文件系统。

初始化音频采集设备驱动，我们采用的是 AIC3101 芯片，设置 AIC3101 的寄存器以便进入采集工作。调用 Engine_open()函数创建并打开编解码器引擎实例，这将返回一个句柄。解析命令行中的音视频参数，调用 Aenc1_create()函数根据这些参数来创建音频编码器。为音频采样数据的输入分配缓存区并为压缩数据的输出分配缓存区。

当音频线程已完成初始化，它将通过集成功能模块与其他线程同步。音频线程只有在其他线程完成初始化后才执行，流程如图 3.5 所示。

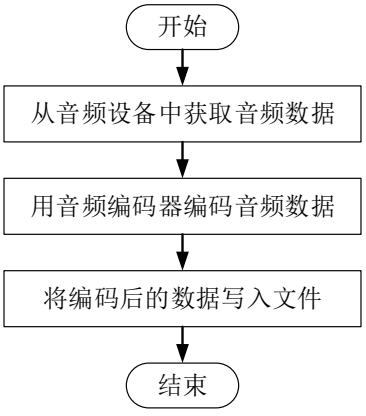


图 3.5 音频线程主流程图

主循环流程介绍如下：

调用 Sound_read()函数从音频设备里读取采样数据。

调用 Aenc1_process()函数编码音频数据。编解码器会调用音频编码算法对音频采

样数据进行编码，把采集缓存区里的数据进行压缩编码后，将压缩数据放到输出缓存区里。

最后调用 fwrite()函数把输出缓存区里的压缩数据写到文件中，这个循环一直到进程停止执行时结束。

3.3.2 视频线程程序设计

视频部分由于要和 DSP 进行交互，所以包括了三个线程。其中视频捕获线程从视频采集驱动器中一帧一帧地获取图像数据并放到采集缓存区中，并把采集到的数据放到显示缓存区，供显示程序播放；视频编码线程对视频源数据进行压缩编码；写线程将压缩后的视频数据写到磁盘文件中[28]。

视频捕获线程的流程如下：

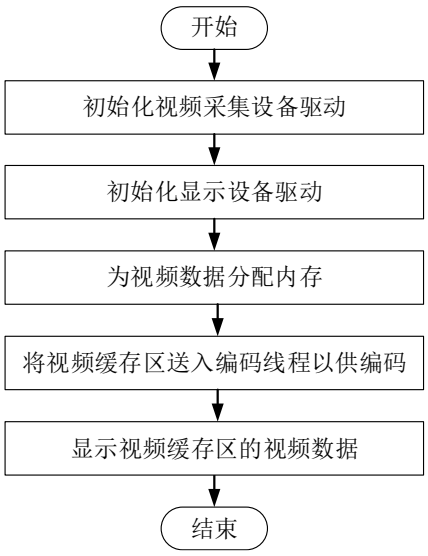


图 3.6 视频捕获线程流程图

创建并初始化采集设备驱动，它由 V4L2 视频框架提供，其接口封装到了 DMAI 中，我们只需调用 DMAI 中的函数即可。视频采集只支持 D1、720P、1080P 分辨率的输入。输入的视频信号是 RGB，需要将 RGB 转换为 YUV。将采集缓存区的大小和视频编码缓存区的大小统一，以便两个缓存区之间交换数据。创建多个缓存区并放入到 FIFO 队列中，以便存放采集到的原始数据，以及方便和视频线程中的编码缓存区交换数据。配置视频制式和视频分辨率，并将信息通知给主线程。下面创建显示设备驱动，并创建多个缓存区放入显示缓存区队列中，采集到的视频数据会从采集缓存区中输入到显示缓存区中，采集到的视频会从显示器上显示以供人们预览。

完成视频捕获线程的初始化之后，进入视频捕获线程的主循环。

视频捕获线程的主循环主要完成了采集缓存区、编码缓存区和显示缓存区的调度，正确的完成采集编码并显示视频的动作。一个采集缓存区放一帧图像的数据，每采集一帧就会把数据放到一个采集缓存区中。然后调用 Fifo_put()函数把采集缓存区放到视频编码缓存区队列中，视频编码线程中会把编码缓存区中的数据进行压缩编码。调用 Fifo_get()函数从采集缓存区队列中取出一个采集缓存区做显示用途，如果这个缓存区里有视频采集数据，则处理里面的数据进行显示，如果缓存区里面没有数据，则放回到采集缓存区队列继续接收采集数据。主循环会不断的进行这样的操作。

视频编码线程的流程如下：

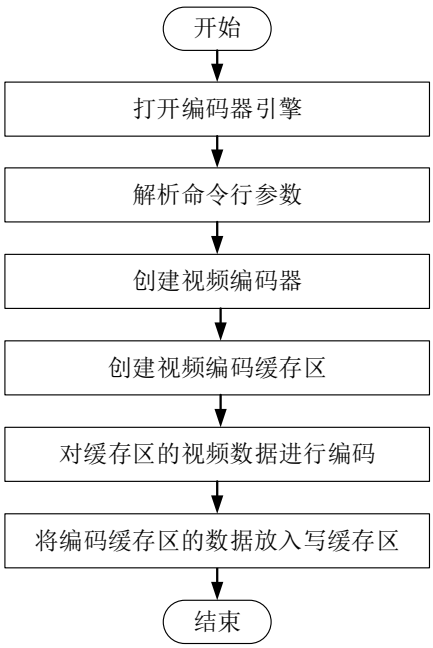


图 3.7 视频编码线程流程图

视频编码线程首先打开 Codec Engine，根据用户设置的音视频参数来配置编码器参数。然后用 Venc1_creat()函数创建视频编码器，Venc1_creat()函数是 DMAI 提供的一个函数接口，它可以根据默认的音视频参数或动态的参数创建一个编码器实例。创建多个视频编码缓存区，并使它的大小与采集缓存区大小相同，以便与采集缓存区交换数据[29]。

视频编码线程的主循环主要完成了采集缓存区、编码缓存区和写缓存区的调度，正确的完成采集编码并写文件的动作。主循环中调用 Fifo_get()函数从采集缓存区队列中取出一个采集缓存区，从写文件缓存区队列中取出一个写缓存区。然后调用 Venc1_process()函数来编码压缩采集缓存区中的数据，然后将压缩后的数据存到写缓存区中，然后调用 Fifo_put()函数把写缓存区放回到写缓存区队列中，写线程中会把写缓存区中的数据写入到文件中。把采集缓存区放回到采集缓存区队列中，继续接收

采集的数据。最后令帧数加一，主循环将连续地操作每帧数据。图 3.7 是视频线程中所涉及的线程内部的交互关系。

视频部分需要一个独立的写线程在采集编码视频信号的同时并行地将压缩数据写入到文件中。写线程中先创建一批用于输出的缓存区，并用 Fifo_put()函数将输出缓存区放到队列中，再用 Fifo_get()函数从视频线程中取出压缩数据放入到输出缓存区中，最后用 fwrite()函数将输出缓存区中的压缩数据写到磁盘文件中。

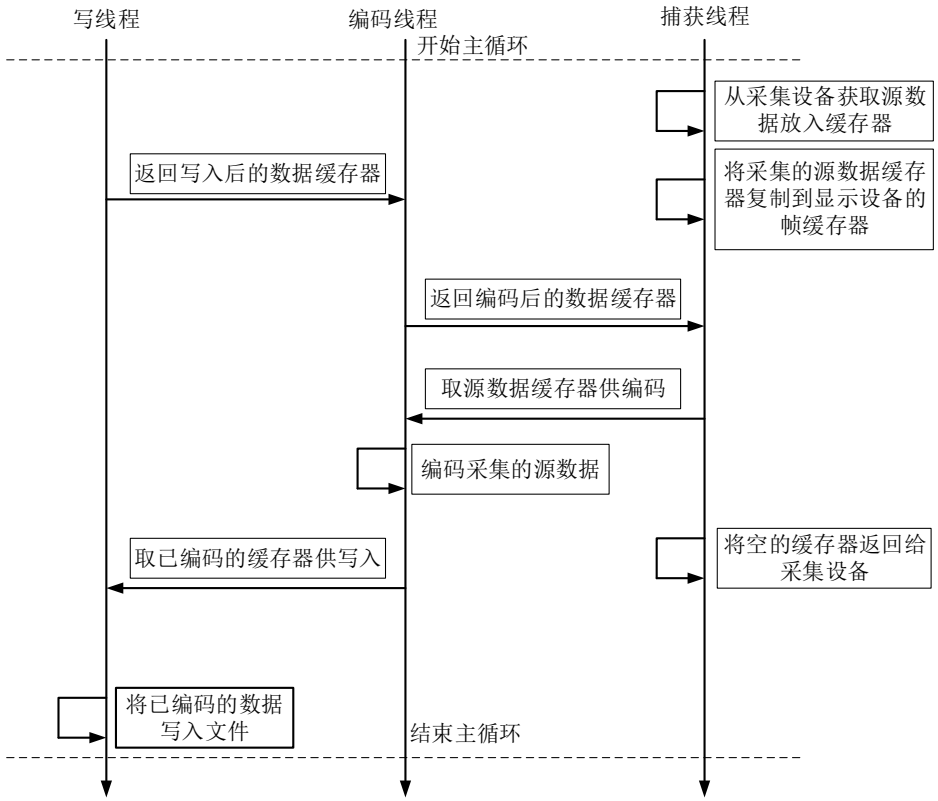


图 3.8 视频线程内部交互关系图

3.4 音视频采集编码参数配置

3.4.1 音视频参数介绍

音视频在采集编码时可以按不同的参数来编码成不同质量的声音和视频，这些参数可以按照实际需求和硬件条件来设置，下面对常用的几个参数，例如帧率、码率、分辨率做出介绍。

人眼在看高速移动的物体时，其前一时刻的画面会在大脑中保持一段时间，这称为视觉保持，正是这种特性，成为视频行业的基石。一帧帧静止的画面在高速切换时

就成为一个流畅的视频。帧率就是显示器一秒钟显示图片的个数，用 FPS（Frames Per Second）表示，当帧率低时，例如 5FPS，人们会看到明显的卡顿，而且可能会丢失一些关键信息。帧率越高，视频会越流畅，但是码率也会变高。一般帧率达到 10FPS，就和视频没什么差异了，当前，最常用的是 25FPS[30]。

码率是指单位时间内音频或视频数据的数据量的大小，单位为 kbps。帧率低，分辨率低或者压缩比高，其数据量就越小，视频数据的码流越小。码率越大对视频数据传输的压力就越大。

分辨率是指一张图像所包含像素数的多少，它决定了视频的清晰程度。本文中的音视频系统中的分辨率有三种选择：CIF（352×288）、2CIF（704×288）和 D1（704×576）。在分辨率的两个数中，前者为图片长方向上的像素数，后者为在宽方向上的像素数，两者相乘得出的是图片的总像素数，长宽比一般为 4:3 格式，在高清视频监控中主要为 16:9 格式。

3.4.2 音视频参数配置

在本项目中，上位机会给 DM365 发送音视频参数配置信息，DM365 通过 EMIF 接口来读取配置信息，DM365 上的音视频程序按发送过来的参数进行采集和编码[31]。上位机发送过来的音视频参数如表 3.2 所示。

表 3.2 音视频配置参数描述

Bit	Name	Description
4bit	帧抽取率(GOP)	1，2，3，5，10，15，20，25，30
4bit	P/I 帧比率	1，5，15，30，45，90，180，360，900，1800
2bit	视频输入通道	00=CVBS_0，01=CVBS_1，10=Y/C
1bit	视频输入格式	0=PAL，1=NTSC
2bit	视频分辨率	00=CIF，01=2CIF，10=D1
3bit	音频通道	0=Stereo，1=Left，2=Right，3=None
16bit	视频输出码率	64-10000Kbps
16bit	音频输出码率	0-65535bps

本项目中的采集编码程序(encode)调用编码算法把来自驱动设备的音视频数据按照给定的参数编码，并写成文件。encode 命令后面跟很多参数。参数-v 后面跟视频文件名，指定视频数据的编码格式，程序会检测是否支持这种文件格式，程序支持的视频格式有 mpeg4, H.264 等。参数-a 后面跟音频文件名，指定音频数据的编码格式，支持的音频格式只有 AAC。参数-y 设置显示的分辨率即视频制式，支持的有 NTSC

和 PAL。参数-r 指定视频编码时的分辨率。参数-b 设置视频码率，程序采用的是动态码率，编码器根据图像内容不同压缩后的数据量不同，所以码率也不同。参数-p 指定音频码率值。其中 GOP 的值会随着视频帧率和码率的变化而变化。我们需要把从 EMIF 接口取出的音视频参数值赋给 encode 命令后所跟的对应的参数。

从 EMIF 接口读取上位机传下来的音视频参数的步骤如下：

调用 Linux 系统函数中的 open()函数打开内存设备，获得内存设备的文件描述符，通过 mmap()函数建立内存映射，将 EMIF 寄存器物理地址映射到用户空间，返回映射首地址的指针，获得寄存器的虚拟内存地址，对寄存器的虚拟内存地址偏移量为 14h 处的内存地址重新赋值，来配置 EMIF 数据传输的位数和 EMIF 读数据传输速率；通过 mmap()函数建立内存映射，将 EMIF 数据传输物理地址映射到的用户空间，返回映射首地址的指针,获得 EMIF 数据传输的虚拟内存地址，然后上位机传来的音视频参数从该内存地址里读出。

由于我们项目所有的程序都写在一个 shell 脚本中，shell 中有很多对文件操作的命令供我们使用。所以我们先把从 EMIF 接口中收到的音视频参数值通过 fprintf()函数写到文件中，并以冒号隔开。cut 命令可以以冒号为分隔取出每段的数据，在 shell 脚本中用 cut 命令把每个参数值取出赋给我们定义的 shell 变量，encode 命令后面的参数值用变量值来代替。例如：

./encode -y 2 composite -b ${videobitrate} -p ${soundbitrate} -r ${resolution} -v test.mpeg4 -a test.aac

在 encode 提供的参数选项里没有帧率，需要我们自己添加，encode 命令后的参数值是由 main 函数传入，即 main(int argc,char* argv[])，其中 argc 为整数，表示命令后面跟的参数数量，当没有参数时，argc=1，表示只有一个程序名，当有一个参数时，argc=2，以此类推。argv[]是指针数组，指向对应的参数字符串，argv[0]指向该程序名称，argv[n]指向第 n 个参数字符串。我们添加视频帧率参数，就令 argc+1,argv[]数组添加一个指向视频帧率的指针。参数传递进来之后赋给控制视频帧率的变量即可。

3.5 视频字幕叠加的设计与实现

3.5.1 校正 Linux 系统时间

根据项目要求，视频需要带有时间字幕。时间信息会每隔一段时间由上位机发送给 DM365，DM365 通过 SPI 接口接收上位机传下来的时间信息，再用这个上位机发送过来的时间信息校正系统时间。从 SPI 接口读取时间信息的过程和上一节从 EMIF 接口读取音视频信息的过程一样，先调用 mmap()函数将 SPI 物理地址映射到用户空

间，对 SPI 进行初始化，从 SPI 的虚拟内存地址里读取时间信息。然后将时分秒的信息存放到 C/C++用来存放时间的 tm 结构体中，用 mktime()函数将该 tm 结构体转换成从 1970 年 1 月 1 日 0 时 0 分 0 秒距离此刻所经过的秒数，把该秒数赋给 timeval 结构体中存放秒数的成员变量，最后以这个 timeval 结构体为参数通过 settimeofday()函数来设置系统时间。时间校正之后，程序就可以提取系统时间，并把时间添加到视频图像上。

3.5.2 图像的颜色编码

在表示彩色图像时会用到色彩编码，当今的色彩编码方式主要为 RGB 和 YUV[32]。RGB 为最早采用的色彩编码方式，它利用了红绿蓝三种颜色用不同比例混合会生成其它颜色的原理，而 YUV 把彩色图像用亮度和色调信息来表示。

1）RGB 色彩模型

光的三原色是红色、绿色和蓝色，任何其它颜色都可以用这三种颜色按不同比例叠加得到[33]。RGB 色彩模型就是利用这种原理来显示颜色，它是目前最常用的面向硬件的色彩模型，广泛应用于彩色监视器和彩色摄像机。通过把三原色对应到三维笛卡尔坐标系上得到 RGB 色彩空间，通过 RGB 色彩空间，彩色图像上的每个元素都可以由 R、G、B 三个数值来表示。对于 24 位颜色，（0，0，0）表示黑色，（255，255，255）表示白色。

2）YUV 色彩模型

YUV 也是一种面向硬件的常用的色彩空间模型，广泛应用于彩色电视广播和视频系统。在 YUV 色彩空间中，Y 对应于亮度，U 和 V 对应于色调的饱和度和色度分量。YUV 空间可以通过对 RGB 空间进行线性变换来得到[34]。YUV 利用了人类视觉的特点，将较少的数据分配给在感知上不太重要的高频色度信息。另外，Y 分量可以独立于色度分量（U 和 V）进行处理。YUV 码流的采样格式有很多种，常见的有 YUV4:4:4, YUV4:2:2, YUV4:2:0 三种[35]。它们的亮度分量没有变，只是色度分量上有所不同。对于 YUV 的三种采样格式，在 Y 分量即亮度分量上没有损失，在保证图像质量的情况下而在色度分量上减少了数据量，这是由于人眼对色度分量不是很敏感。用 YUV4:2:0 采样格式就可以还原视频，还降低了数据量。

下面我们需要知道 YUV 码流的存储方式，因为只有正确的知道每个像素点的 YUV 的位置，我们才能对 YUV 值进行例如叠加字幕的操作。YUV 存储有两种格式，一种是先连续存储所有像素点的 Y，然后存储所有像素点的 U，然后是所有像素点的 V。另一种是每个像素点的 Y、U、V 交叉存储。

3.5.3 视频字幕叠加方案设计

本项目要求在传输的视频上添加时间字幕，对于添加字幕，我们无法直接对压缩后的视频数据进行修改，要对采集的视频源数据即 YUV 数据进行操作。采集编码程序中对视频数据的处理是把采集的 YUV 源数据放到缓存里，视频编码线程从这个缓存里取数据并进行编码，所以我们只需要把这个缓存里的数据进行添加字幕的操作即可。对 YUV 数据进行字幕叠加的操作实际上是用字符图像的 YUV 值替换指定位置处的图像的 YUV 值。

我们使用的字库分为点阵字库和矢量字库两种。点阵字库是把一个汉字在 16×16 或 24×24 的点阵里表示，修改点阵里的一部分像素值来表示汉字的轮廓，可以应用于低要求场景下的显示。由于用来表示汉字的像素点是固定的，在放大时表示汉字的像素点的排列不变，在汉字的边缘会出现锯齿。矢量字库弥补了点阵字库的缺点，它包含了一个汉字在形状上的所有信息，例如每个笔划的长度比例，笔划的弧度等等。在显示时，计算机会通过复杂的数学计算来对汉字进行还原，并显示或打印出来。这一类字库在放大时仍然保持汉字的形状，所以在 Linux 系统和 Windows 系统都提供有矢量字库，来显示或打印字符，不会用到点阵字库[36]。在项目中，我们添加字幕只为了显示时间信息，不会进行放大操作，所以采用点阵字库。

而在本文的设计中，我们需要在视频上叠加时间字幕，所以只需要点阵字库中的数字字模就可以了，字幕叠加的流程图如下图所示。

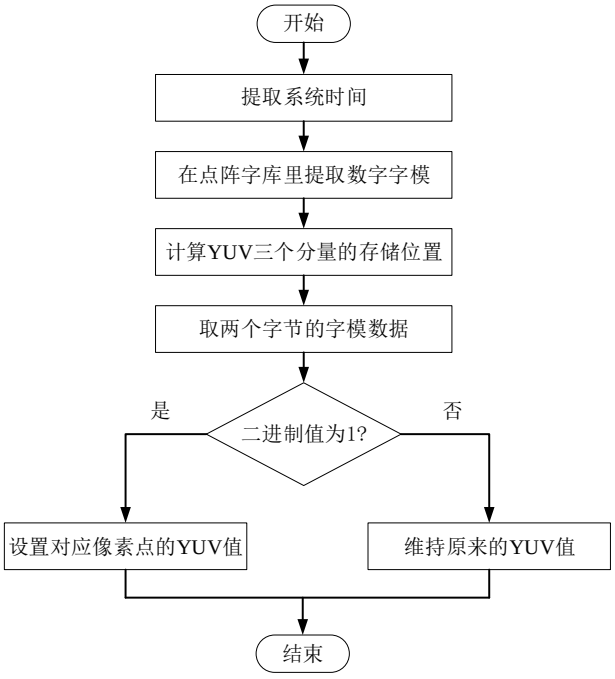


图 3.9 视频字幕叠加流程图

下面以“高”字分析点阵字模的排列方式，如图 3.10。采用的是 16×16 点阵。点阵中的每一个点用一位来表示，表示字符的点用二进制 1 表示，没有字符的点用二进制 0 表示，那些二进制为 1 的点连起来就是一个字符。16×16 点阵可以用 16×16/8=32 个字节来表示。一个字每一行用两个十六进制的数来表示，16×16 点阵有 16 行，则用 32 个十六进制的数来表示这个字符的点阵字模。

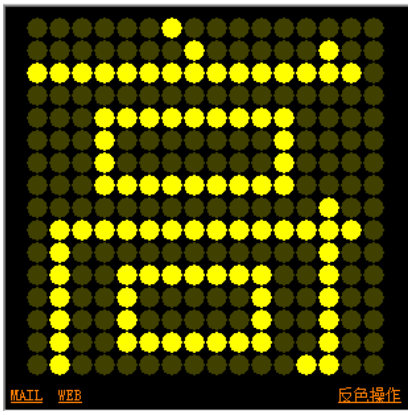


图 3.10 点阵字模示意图

有了点阵字库，我们就可以从字库里提取字模，往视频上添加字幕了。在本项目中，先用 localtime()函数把系统时间取出，这个函数返回一个结构体，结构体里面包含时分秒变量。然后把时间的每个字符的字模从字库中取出，把视频的每一帧进行处理，把字幕加到视频上。程序中以 YUV4:2:0 的格式采集视频，存储方式为先存储所有的 Y 分量，后面存储 U 分量，最后是 V 分量。U 分量和 V 分量分别是 Y 分量的 1/4。则 Y 分量的起始位置是 YUV 数据的起始位置，U 分量位置=Y 分量位置+FRAME_WIDTH×FRAME_HEIGHT，V 分量位置=U 分量位置+FRAME_WIDTH×FRAME_HEIGHT/4（FRAME_WIDTH 为一帧图像宽方向上的像素点数量，FRAME_HEIGHT 为一帧图像长方向上的像素点数量）。知道了 Y、U、V 分量的位置，就可以通过点阵字模操作对应像素点的 Y、U、V 分量的值了。具体操作为，依次取出时分秒的字模数据，其中一个字模有 32 个十六进制的数，每两个十六进制的数就代表这个字符其中一行的数据。循环对这 32 个十六进制数进行操作，如果二进制为 1，则设置这个像素点的 YUV 值，如果为 0，则不做处理。此处，当设置 Y=128，U=128，V=128 时，字幕为白色；当 Y=226，U=0，V=149 时，字幕为黄色；当 Y=76，U=85，V=225 时，字幕为红色；当 Y=150，U=44，V=21 时，字幕为绿色。

3.6 本章小结

本章主要介绍了音视频信号的采集和编码部分。首先介绍了音视频压缩编码的基础理论，然后分析了音视频采集编码的程序设计，最后介绍了音视频参数配置和视频添加字幕的实现。

第四章 嵌入式平台上的音视频流封装及传输的原理及实现

本章主要介绍在 DaVinci 平台上音视频码流封装成 TS 流以及通过 EMIF 接口传输的实现。首先会介绍 MPEG-2 TS 格式以及 FFmpeg 音视频框架。然后介绍在 FFmpeg 音视频框架的基础上音视频流封装和传输的实现方法和程序设计。

4.1 MPEG-2 传输流介绍

压缩后的音频码流和视频码流是分开的，我们看的视频都是声音和图像一同播放，所以要把音频压缩码流和视频压缩码流复用到一起，并实现声音和图像的同步，这称为音视频的封装。其封装格式有 AVI, MP4, TS, FLV, MKV 等。我们播放的电影大都是 MP4 或者 FLV 格式，这种格式适合存储到计算机上，可以随时打开文件进行播放。而对于视频监控、视频会议这种实时传输的情况，TS 流应用比较广泛[37]。因为 TS 流在任意位置截取或者拼接都可以播放，并且 TS 流在传输时抗干扰能力强。

4.1.1 TS 流格式介绍

TS 流的生成过程如图 4.1，将原始音视频数据压缩之后，压缩数据组成一个基本码流（ES）。然后把 ES 流重新分组，并加入 PTS 和 DTS 信息，添加 PES 包头打包成 PES 包，再将一个或多个 PES 包复合成具有恒定比特率的 MPEG2-TS 流。

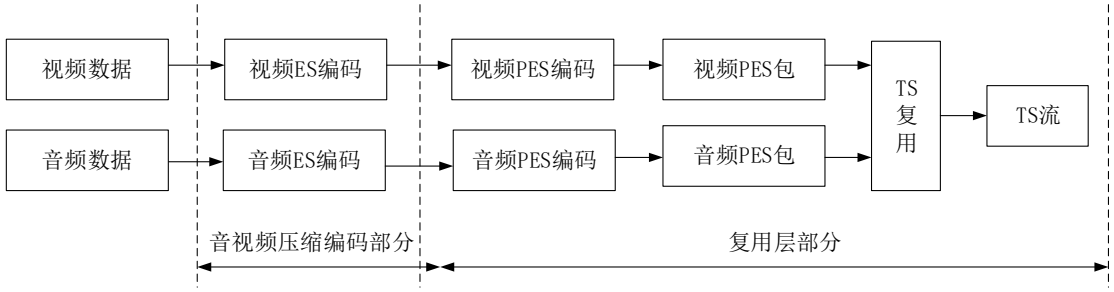


图 4.1 TS 流的形成过程图

TS 流里面有音视频数据，还有 PAT 表和 PMT 表，一个 PAT 表或 PMT 表组成一个 TS 包，这两个表来表示节目信息，用于电视广播。在本文中的系统中，要封装的只有一路音频和一路视频，所以不需要 PAT 表和 PMT 表。TS 包固定为 188 字节，其中前 4 个字节为包头，存放这个 TS 包的信息，最后 184 个字节是音频数据和视频数据。若一个包中的音视频数据不满 184 个字节，则用字节 FF 来填充[38]。TS 流就是

由若干个 TS 包首位相接组成，如下图：

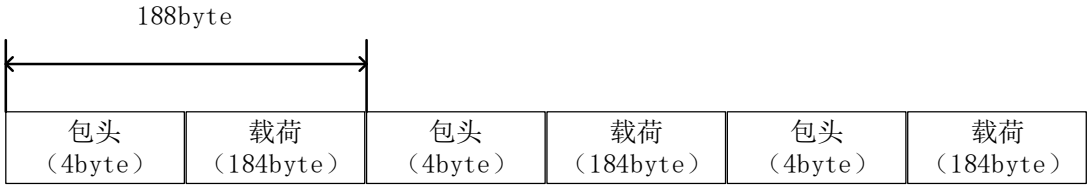


图 4.2 TS 流的结构图

4.1.2 音视频同步原理分析

音视频同步通过 PTS（显示时间戳）和 DTS（解码时间戳）来实现，它们在编码的时候生成。TS 是基于数据包的封装格式，并且音频和视频经过复用再分成特定大小的数据包，其中视频数据和音频数据中包含 PTS 和 DTS。由于 MPEG-2 编码视频中有三种类型的视频帧：I 帧、B 帧和 P 帧，并且这三种类型的帧解码顺序和显示顺序不同，所以需要两个不同的时间戳：PTS 和 DTS[39]。I 帧为帧内压缩，不需要其它帧进行解码。解码 P 帧需要参考前面的 I 帧和 P 帧。解码 B 帧需要参考上一个和下一个 I 帧或 P 帧。因此，解码器需要在解码 B 帧之前获得后面的 I 帧或 P 帧。PTS 可以被认为是显示帧号，它是一个 33 位的数字，它是解压缩的数据显示给用户的时间，PTS 值必须大于或等于每帧的 DTS 值，因为显示必须是在解码之后。DTS 同样由 33 位的数来表示，它表示被解压的时间。作为音视频数据包的时钟参考字段，PTS 和 DTS 被用来音频和视频之间的同步，通过公共时基（系统时钟）来统一计算解码时间和显示时间，通过这个时间来确保正确的音视频同步。

4.2 FFmpeg 音视频封装解决方案

4.2.1 FFmpeg 音视频编解码框架介绍

FFmpeg 是一个用 C 语言编写的用于录制、流化和播放多媒体文件的开源框架，它是一个完备的，跨平台的音视频解决方案。FFmpeg 提供了很多音视频相关的功能，例如编解码、封装和解封装、转换格式、设置帧率码率等参数、加滤镜等。另外，FFmpeg 可移植到 Windows、Linux、和嵌入式系统上编译及运行[40]。FFmpeg 处理音视频数据的性能比较强，且功能强大，很多播放器都是在 FFmpeg 的基础上做开发。有很多领域例如娱乐、医疗等中的视频技术开发都是基于 FFmpeg。VLC 播放器、Mplayer 等多媒体播放器都是基于 FFmpeg 开发，来实现视频播放、视频编辑、流媒体等功能[41]。目前的医学影像数据分析仪也使用 FFmpeg 来实现音视频的编解码并对其进行分析。

FFmpeg 框架的基本组成包含 AVFormat、AVCodec、AVFilter、AVDevice、AVUtil 等模块库[42]，结构如图 4.3 所示。

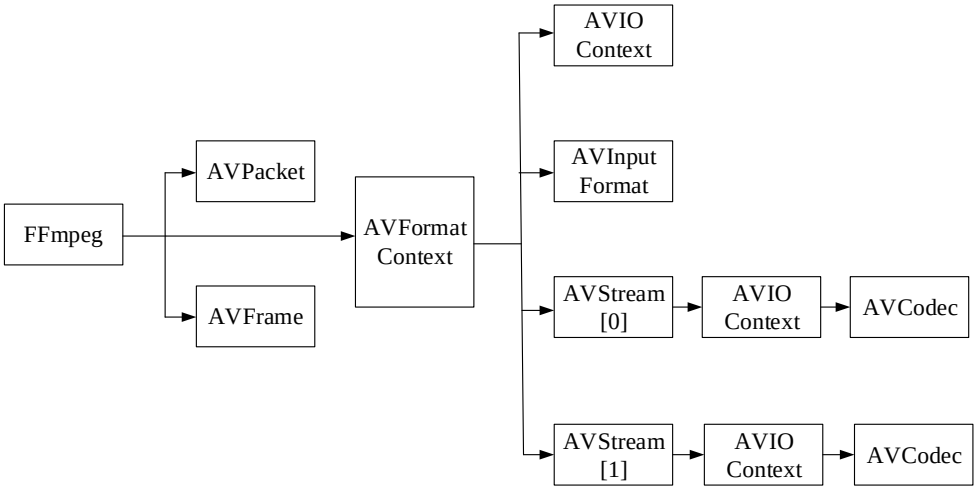


图 4.3 FFmpeg 基本组成模块示意图

下面针对这些模块做一个大概的介绍。

（1）FFmpeg 的封装模块 AVFormatContext

AVFormatContext 结构体里面包含了 AVIOContext 结构体、AVInputFormat 结构体和 AVStream 结构体[43]。AVFormatContext 结构体是 FFmpeg 中包含信息最大的结构体，其中包括了音视频数据和信息，以及各种对音视频数据操作的函数。在使用 FFmpeg 中的库函数操作音视频数据前，首先要把音视频数据读入到 AVFormatContext 结构体中，然后才能对音视频数据进行操作。

（2）FFmpeg 的编解码模块 AVCodec

AVCodec 存储了编解码器的信息，包括编解码器的名称和 ID，音视频类型（视频、音频、字幕），支持的帧率、采样率、像素格式、声道数等信息。AVCodec 支持常用的大部分编解码器，其中，在使用 H.264 编码时，需要加入 x264 编码器；使用 H.265（HEVC）编码时，需要加入 x265 编码器；使用 MP3（mp3lame）编码，需要加入 libmp3lame 编码器。加入这些编码器只需在配置 FFmpeg 编译选项时选中这些编码器即可。

（3）FFmpeg 中的数据容器 AVStream

AVStream 流主要作为存放音频、视频、字幕数据流使用。

（4）FFmpeg 中的 AVFrame

AVFrame 结构体中存储视频的 YUV 或 RGB 数据，音频的 PCM 数据，并且包括了和它们相关的信息。

（5）FFmpeg 存储压缩数据的结构体 AVPacket

AVPacket 结构体存放压缩编码后的数据，因此在使用 FFmpeg 处理音视频时，可以处理 AVPacket 中的数据，例如存成文件或解码等操作。

（6）FFmpeg 的视频图像转换计算模块 swscale

swscale 模块主要用于图像缩放和图像格式转换，例如经常用到的将 RGB 转换为 YUV。

基于 FFmpeg 开发音视频程序，就是调用 FFmpeg 中的函数对以上结构体进行操作。FFmpeg 提供了大量的音视频操作接口函数，可用于采集、编解码、封装、加滤镜等操作。而且每个操作都有一个调用流程，例如对于解码来说，需要先把音视频数据读到 AVFormatContext 结构体中，再通过音/视频流信息找到对应的解码器，从音/视频流中读取 AVPacket 数据包，然后调用 FFmpeg 中的解码函数对音/视频进行解码。正是 FFmpeg 中提供了大量的函数和结构体方便人们使用，让基于 FFmpeg 开发音视频应用变得高效而又能实现丰富的功能。

4.2.2 嵌入式 Linux 平台下 FFmpeg 的移植

由于本系统中的封装程序需要调用 FFmpeg 的 API，所以要把 FFmpeg 移植到嵌入式 Linux 平台上[44]。在 FFmpeg 官网下载安装文件，我们采用的版本是 ffmpeg-2.6.3。

在官网上下载的 FFmpeg 安装文件是压缩文件，需要先进行解压。在终端里执行“tar xvfj ffmpeg-3.2.4.tar.bz2”命令进行解压缩，然后再进行安装。

安装流程是先执行./configure 对编译选项进行配置，然后执行 make 命令进行编译。当执行 make 时，make 会在当时的目录下搜寻 makefile 文件，makefile 文件里记录了如何编译的详细信息。make 会自动判别代码是否经过了变动，而去重新编译改动的那一部分。而 configure 程序为侦测程序来侦测运行环境，并且建立 makefile 文件。

由于 FFmpeg 默认为 PC 上安装，所以移植到嵌入式系统上时需要指出所使用的处理器架构，编译为交叉编译，并且指出所使用的交叉编译器。其中--prefix 是指定安装目录，--enable-shared 是生成动态库。然后执行 make 命令进行编译，并执行 make install 安装，就可以编译并生成适用于指定系统的库文件和执行程序[45]。安装完成之后在安装目录下会看到三个目录，bin 目录下是生成的可执行文件，lib 目录下是静态、动态链接库目录，include 目录下是编译用到的头文件。然后把库文件和执行文件拷贝到开发板就完成了 FFmpeg 的移植。

4.3 音视频流封装及实时传输程序设计

4.3.1 FFmpeg 实现多媒体码流的 TS 格式封装

使用 FFmpeg 的 API 进行封装（Muxing）操作的主要步骤比较简单，流程如下图所示。

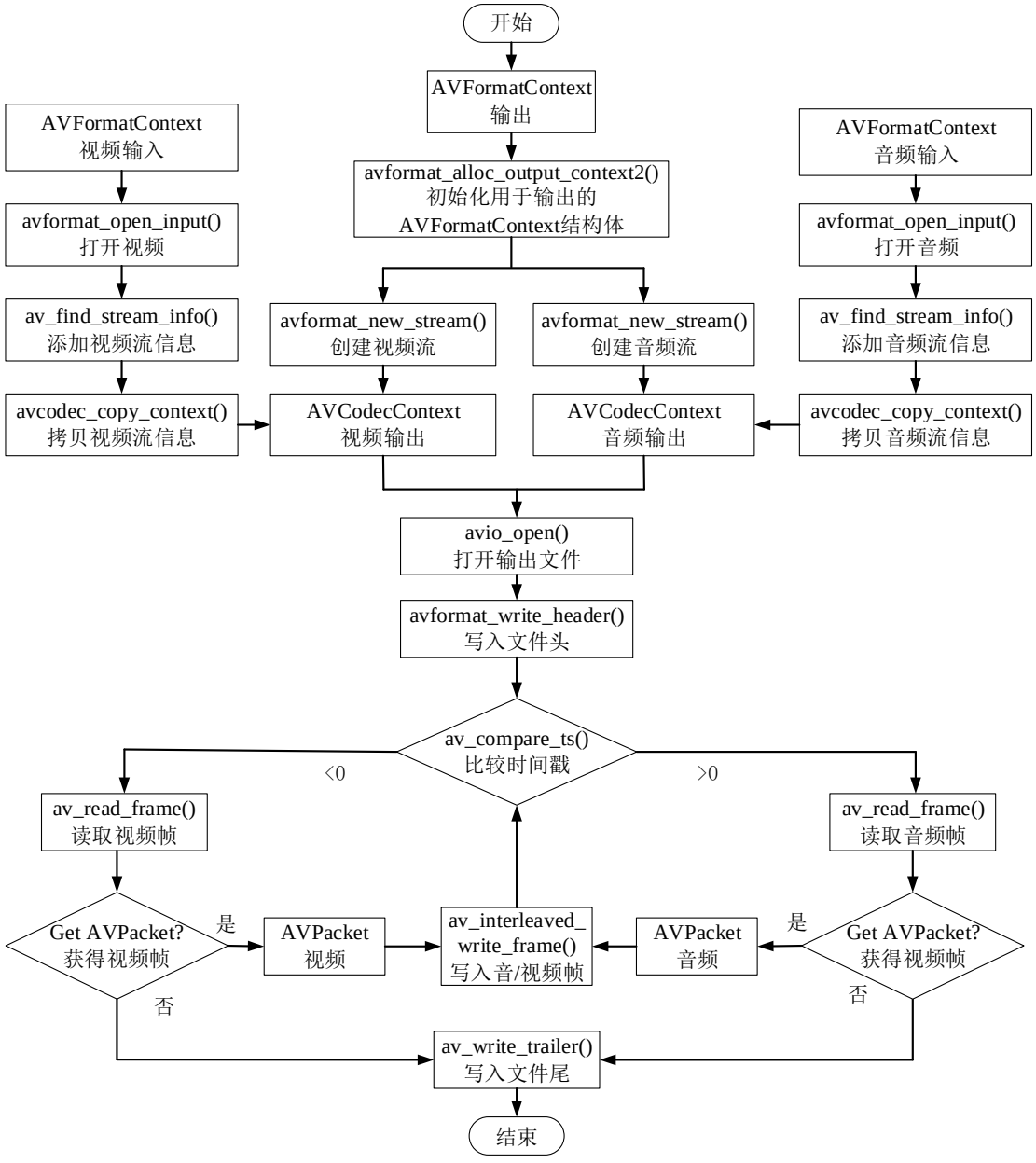


图 4.4 基于 FFmpeg 的 TS 格式封装流程图

在使用 FFmpeg 的 API 之前，需要加入 FFmpeg 中的 avcodec.h 和 avformat.h 等头文件。在全局变量中定义三个 AVFormatContext 结构体指针，分别用于存放音频输入数据和相关信息、视频输入数据和相关信息和音视频输出数据和相关信息。调用

avcodec_register_all()函数初始化 FFmpeg 中的编解码器。用 avformat_open_input()函数打开音频文件和视频文件，这个函数会把音频文件或视频文件的文件头信息加入到 AVFormatContext 结构体中，后面就可以操作 AVFormatContext 结构体，而无需对音频和视频文件进行操作。然后用 avformat_find_stream_info()函数读取音视频流信息。用 av_dump_format()函数可以将音视频信息如视频帧率、音频采样率、音视频码率等打印出来显示到屏幕上。用 avformat_alloc_output_context2()函数初始化用于输出数据的 AVFormatContext 结构体，并指定音视频数据的输出格式，指定音视频输出文件名，或者将音视频数据输出到内存中。下面需要把输入的音频和视频的数据及信息从用于输入的 AVFormatContext 结构体中拷贝到用于输出的 AVFormatContext 结构体中。首先用 avformat_new_stream()函数在用于输出数据的 AVFormatContext 结构体中申请两个 AVStream 结构体指针，分别存放音频流和视频流。并用 avcodec_copy_context()函数将输入流中的编码器信息写入到输出流中。这样，用于输出的 AVFormatContext 结构体里包含了音频和视频的数据和信息，下面就可以操作用于输出的 AVFormatContext 结构体来封装音视频数据了。在封装音频和视频数据时，有些封装格式需要写入头部信息，所以在 FFmpeg 写入封装数据时，需要用 avformat_write_header()写入封装格式的头部。在 FFmpeg 操作数据包时，均采用写帧操作进行音视频数据包的写入，而每一帧在常规情况下均使用 AVPacket 结构体进行音视频数据的存储，AVPacket 结构中包含了 PTS、DTS、音/视频数据等信息。在 FFmpeg 中用 av_read_frame()函数从音视频数据中读取 AVPacket,在将取出的 AVPacket 写入到输出文件之前，要用 av_compare_ts()函数来比较时间戳决定该写入视频还是音频，然后用 av_interleaved_write_frame()函数将 packet 写入到输出的封装格式中。在写入数据结束后，要用 av_write_trailer()函数写入容器尾信息[46]。

4.3.2 FFmpeg 内存数据操作

在上一章中设计了采集编码程序，这个程序可以采集音频和视频并按照项目的要求设置参数，生成音频编码文件和视频编码文件，在这个项目中，音频格式是 AAC，视频格式是 H.264。封装程序是基于 FFmpeg 编写的，它可以把音频编码文件和视频编码文件封装为 TS 格式的文件。由于这两个进程在运行时需要交换数据，所以要用到进程间通信。进程间通信方式有很多种，无名管道用于有血缘关系的进程间传输数据，这里的采集编码程序和音视频流封装程序是两个独立的程序，所以不能采用。这里我们采用的是有名管道(FIFO)。

首先创建有名管道，我们需要传递音频编码数据和视频编码数据，所以需要创建两个管道，一个用于传递音频编码数据，一个用于传递视频编码数据。管道文件名分

别用宏定义定义为#define MYFIFO_v "/tmp/myfifo_v"（视频管道）和#define MYFIFO_a "/tmp/myfifo_a"（音频管道）。创建有名管道调用 mkfifo()函数，获得管道文件名，并设置文件的用户权限，这里设置为 666，即可读可写可执行。然后调用 open()函数打开音频管道和视频管道，采用一端写另一端读的模式，采集编码程序里以只写阻塞方式打开音频管道和视频管道，即调用 open(MYINFO_a,O_WRONLY)函数返回音频管道文件描述符 fd_a，和调用 open(MYINFO_v,O_WRONLY)函数返回视频管道文件描述符 fd_v。封装程序里以只读阻塞方式打开管道，调用 open(MYINFO_a,O_RDONLY)函数和调用 open(MYINFO_v,O_RDONLY)函数打开两个管道。然后在采集编码程序中的音频线程里调用 write(fd_a,Buffer_getUserPtr(hOutBuf),Buffer_getNumBytesUsed(hOutBuf))函数把音频编码数据写到音频管道中，其中 Buffer_getUserPtr(hOutBuf)是存放音频数据的内存地址，Buffer_getNumBytesUsed(hOutBuf)是音频数据的数据量。在采集编码程序中的视频写线程里调用 write(fd_v,Buffer_getUserPtr(hOutBuf)，Buffer_getNumBytes Used(hOutBuf))函数把视频编码数据写到视频管道中。封装程序中调用 read()函数读取管道中的音频数据和视频数据，然后把取出的音视频数据封装成 TS 格式。这里不同于文件操作，需要把从管道中的音视频数据读到申请的内存里，然后对数据进行操作，要用到 FFmpeg 内存数据的操作。

项目中开始设计的封装程序是操作音频编码文件和视频编码文件封装成 TS 格式文件。在一些场景中需要从内存中读取音视频数据，然后进行其他操作，例如，我们要实现的就是把音视频数据从管道里读到内存中，然后封装成 TS 格式。使用 FFmpeg 的 libavformat 中的 avio 方法可以达到该目的。首先用 avformat_alloc_context()函数申请一个 AVFormatContext，因为在 FFmpeg 框架中，针对 AVFormatContext 进行操作会非常方便，所以可以将数据挂在 AVFormatContext 中，然后使用 FFmpeg 中的函数进行操作[47][48]，如下图所示：

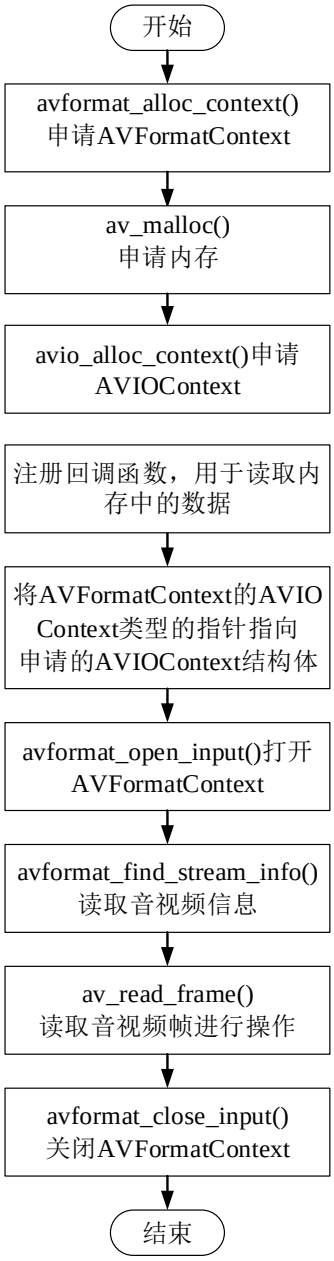


图 4.5 FFmpeg 内存数据操作流程图

下面要申请一个 AVIOContext 结构体，同时将内存数据读取的接口注册给 AVIOContext。通过 FFmpeg 音视频框架中的 av_malloc()函数，在 RAM 中申请一段内存，并将所申请的内存的指针和大小作为 FFmpeg 中 avio_alloc_context()函数的参数，然后在 avio_alloc_context()函数中注册用于读取内存数据的回调函数 fill_buffer()，同时通过 avio_alloc_context()函数申请 FFmpeg 中的 AVIOContext 结构体；将 AVFormatContext 结构体的 AVIOContext 类型的指针 pb 指向 AVIOContext 结构体，得到指定音视频数据读入位置为内存的 AVFormatContext 结构体。当程序需要数据时，

会自动调用回调函数来获取数据。回调函数功能是把管道中的音视频数据读到内存中，调用 read(fd,buf,bufsize)。这样封装程序就可以在管道中读取音视频数据并进行封装操作了。

4.3.3 音视频数据的传输

在该系统的硬件设计中，在将音频和视频数据封装到 TS 流之后，TS 流通过 EMIF 接口传输到 FPGA，然后通过 PCI 接口将 TS 流从 FPGA 传给上位机，上位机再进行解码播放的操作。首先调用 mmap()函数将 EMIF 寄存器物理地址映射到用户空间，返回寄存器的虚拟内存地址，修改寄存器的值来配置 EMIF 数据传输的位数和 EMIF 写数据传输速率；然后调用 mmap()函数将 EMIF 数据传输物理地址映射到的用户空间，返回 EMIF 数据传输的虚拟内存地址。我们的封装程序先把 TS 流数据写到内存里，再把内存里的数据复制到 EMIF 数据传输的虚拟内存地址，EMIF 接口就会把该地址中的数据发送给 FPGA[49][50]。这里我们同样用到了 FFmpeg 内存数据的操作，先用 av_malloc()函数开辟一块内存，封装的 TS 流数据会放到这块内存里，然后调用 avio_alloc_context(outbuffer,sizeof(outbuffer),1, NULL,NULL,write_buffer,NULL)，程序封装成的 TS 数据会输出到 outbuffer 指向的内存中，write_buffer 是回调函数，在回调函数里把内存里的数据复制到 EMIF 数据传输的虚拟内存地址。由于 EMIF 接口数据线有 16 位，所以每次从内存里取两个字节的数据复制到 EMIF 数据传输的虚拟内存地址里，EMIF 接口就会按寄存器配置的数据传输速率从这个地址里取数据传输给 FPGA 的 FIFO。由于 TS 流数据需要通过网络传输，网络字节序采用的是大端字节序，我们的发送主机是小端字节序，所以在传输之前要进行字节序转换，这样上位机收到的数据才是正确的。

4.4 本章小结

本章主要介绍把音视频流封装成 TS 流格式并进行传输的部分。首先对 TS 流格式和 FFmpeg 进行了介绍，然后通过调用 FFmpeg 的函数进行程序设计，将音频压缩码流和视频压缩码流封装成了 TS 流，最后将 TS 流通过 EMIF 接口传输给 FPGA 的 FIFO。

第五章 音视频采集压缩和封装传输程序测试及结果分析

本章主要测试音视频程序，并对结果进行分析。，其中程序测试包括对音视频采集编码和封装的测试，以及对生成的 TS 流是否正确的传输到上位机、参数配置和时间字幕添加等问题的测试，并将文件系统和应用程序烧进板子，实现程序的开机自启动。

5.1 程序功能测试与分析

5.1.1 音视频采集编码程序测试

音视频采集编码程序实现了音视频信号的采集并压缩，程序运行后，开发板外接的摄像头会采集图像信号，音频信号可以通过麦克风或线路输入获取。信号采集后会进行模数转换，然后程序调用 DSP 中的压缩算法对原始数据进行压缩，然后写成文件放到磁盘里。采集编码程序经过交叉编译生成 encode 命令，encode 命令后跟许多参数。例如参数-v 后面跟视频文件名，程序会检测文件的扩展名来生成对应的格式，支持 MPEG4、H.264 等格式。参数-a 后面跟音频文件名，程序同样根据扩展名来生成相应的音频格式。参数-y 来设置视频制式，例如 NTSC 和 PAL。参数-r 设置视频分辨率，参数-b 设置视频码率，参数-p 设置音频码率，这里的码率为动态码率。encode 程序会根据这些命令行参数来初始化编码器进行采样编码的工作。

本文的设计中，视频采用了 H.264 格式，音频采用了 AAC 格式。在超级终端运行 encode –y 2 composite –v test.264 –a test.aac，程序执行后，超级终端上会显示音视频参数信息，如图 5.1。

```
s Time: 00:00:35 Demo: Encode Display: D1 PAL Video Codec: H.264 HP Resolution:
736x576 Sound Codec: AACLC Encoder Sampling Freq: 16 KHz samp rate

ARM Load: 17% Video fps: 25 fps Video bit rate: 308 kbps Sound bit rate: 99 kbps
 Time: 00:00:36 Demo: Encode Display: D1 PAL Video Codec: H.264 HP Resolution: 7
36x576 Sound Codec: AACLC Encoder Sampling Freq: 16 KHz samp rate

ARM Load: 19% Video fps: 25 fps Video bit rate: 294 kbps Sound bit rate: 102 kbp
s Time: 00:00:37 Demo: Encode Display: D1 PAL Video Codec: H.264 HP Resolution:
736x576 Sound Codec: AACLC Encoder Sampling Freq: 16 KHz samp rate

ARM Load: 18% Video fps: 25 fps Video bit rate: 296 kbps Sound bit rate: 81 kbps
 Time: 00:00:38 Demo: Encode Display: D1 PAL Video Codec: H.264 HP Resolution: 7
36x576 Sound Codec: AACLC Encoder Sampling Freq: 16 KHz samp rate

ARM Load: 15% Video fps: 25 fps Video bit rate: 287 kbps Sound bit rate: 102 kbp
s Time: 00:00:39 Demo: Encode Display: D1 PAL Video Codec: H.264 HP Resolution:
736x576 Sound Codec: AACLC Encoder Sampling Freq: 16 KHz samp rate

ARM Load: 16% Video fps: 25 fps Video bit rate: 286 kbps Sound bit rate: 102 kbp
s Time: 00:00:41 Demo: Encode Display: D1 PAL Video Codec: H.264 HP Resolution:
736x576 Sound Codec: AACLC Encoder Sampling Freq: 16 KHz samp rate
```

图 5.1 音视频采集编码参数显示图

如图所示，超级终端上会显示 CPU 利用率、视频帧率、音频码率、视频制式、视频编码格式、分辨率、音频编码格式、采样率等信息，并生成音频文件和视频文件。其中，默认视频帧率为 25fps，音频采样率为 16KHz，分辨率为 736×576，码率会随着音频数据或视频数据的大小动态的变化，例如当图像为静止时，视频码率较小，图像有运动时，视频码率变大。

5.1.2 音视频封装程序测试

在封装程序的编写过程中，首先基于 FFmpeg 设计了一个 PC 版本程序，然后把 FFmpeg 移植到了嵌入式平台上，再对封装程序进行交叉编译，交叉编译器为 arm-none-linux-gnueabi-gcc，得到可以在嵌入式平台上运行的封装程序。

由于平台上要运行封装程序和采集编码程序两个进程，如第四章所述，我们通过管道完成了两个进程的通信，两个进程在 Linux 系统中并行运行，就需要把其中一个进程放到后台中运行。

执行./TS_mutex &，封装程序会放到后台运行，执行 encode 程序，采集编码程序生成的音视频压缩数据会通过管道传给封装程序进行封装。封装程序生成的 TS 流可以写成文件也可以进行传输，生成的 TS 格式文件可以用 VLC 播放器打开。其播放效果如下图：

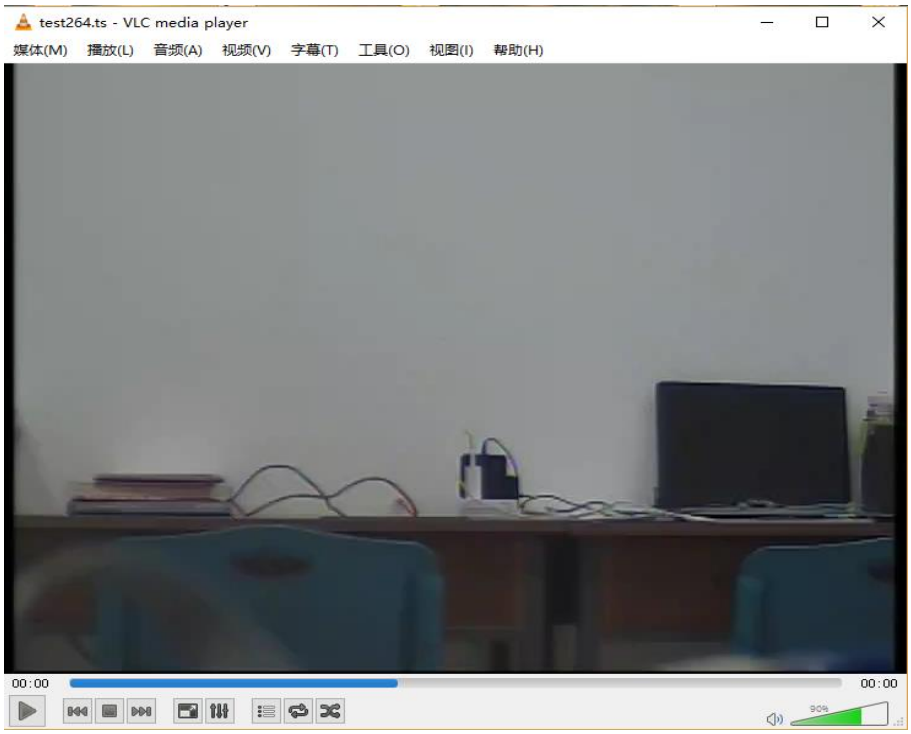


图 5.2 TS 格式视频播放图

在程序运行时嵌入式设备采集音视频信号，由采集编码程序捕获音视频数据并压缩，然后将音视频压缩数据封装成 TS 流并写入文件，播放该视频文件，没有丢帧、模糊现象，表明程序运行正常。

5.1.3 TS 流传输程序测试

下面测试 TS 流传输功能。生成的 TS 流会通过 EMIF 接口传输给 FPGA 的 FIFO，再通过 PCI 接口传输给上位机。

由于 DM365 需要通过 EMIF 接口把数据传输给 FPGA 的 FIFO，所以要测试 EMIF 写数据的功能。我们把 0 到 9 十个数循环通过 EMIF 接口发送，通过 signaltap 检查 FIFO 端的数据，如下图。

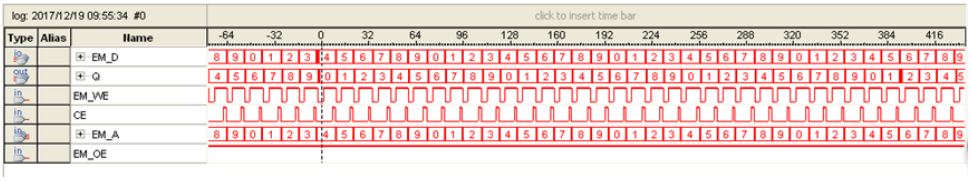


图 5.3 EMIF 接口写数据信号捕获图

图中，EM_D 为 EMIF 发送的数据，为 0~9 循环发送，Q 为 FIFO 接收到的数据，也为 0~9，测试表明，EMIF 通信正常。

为了测试数据从 DM365 传输到上位机的正确性，我们把一个录好的 TS 格式文件从 DM365 传输给上位机，上位机接收到 TS 流数据后可以进行解析，保存成 TS 格式文件。对 DM365 端生成的 TS 文件数据和传输到上位机的 TS 文件数据进行比较，来判断数据传输的正确性。内容比较可以用 UltraCompare 工具，比较 TS 文件数据用 UltraCompare 以二进制模式打开进行比较，如下图所示：

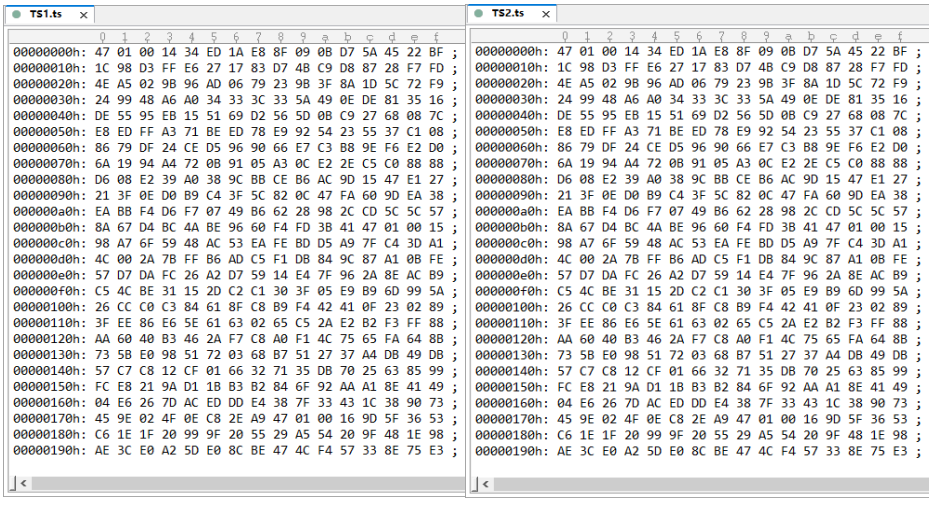


图 5.4 TS 文件数据比较图

TS 格式文件数据以十六进制数显示。一个 TS 包以同步字节 0x47 开头，前 4 个字节为包头，后边 184 个字节为数据，当没有音频和视频数据时，以 FF 字节填充。通过 UltraCompare 比较，DM365 端的 TS 数据和上位机收到的 TS 数据完全相同。

DM365 采集的音视频数据封装成 TS 流实时传输给上位机，上位机进行解析，保存成 TS 格式文件，用 VLC 播放器播放正常。

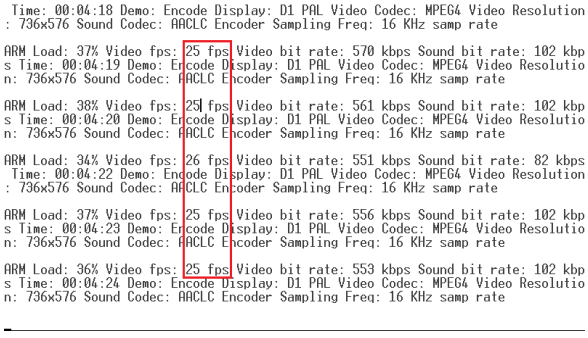
5.1.4 音视频参数配置及测试

下面测试该系统的音视频参数配置部分。音视频参数信息通过 PCI 接口从上位机发送，并通过 EMIF 接口传给 DM365，首先测试 EMIF 读数据功能是否正常。

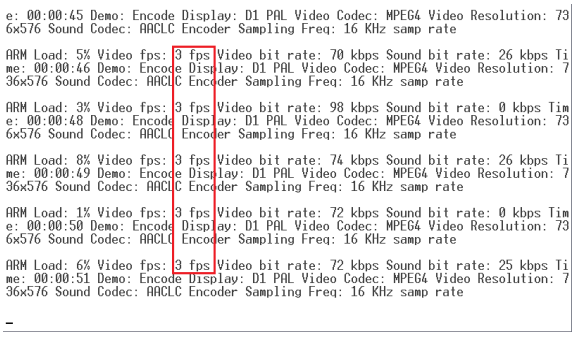
图 5.5 EMIF 接口读取数据信号捕获图

图中，EM_D 为 EMIF 接收的数据，从图中可以看出 EMIF 读出了正确连续的数据。

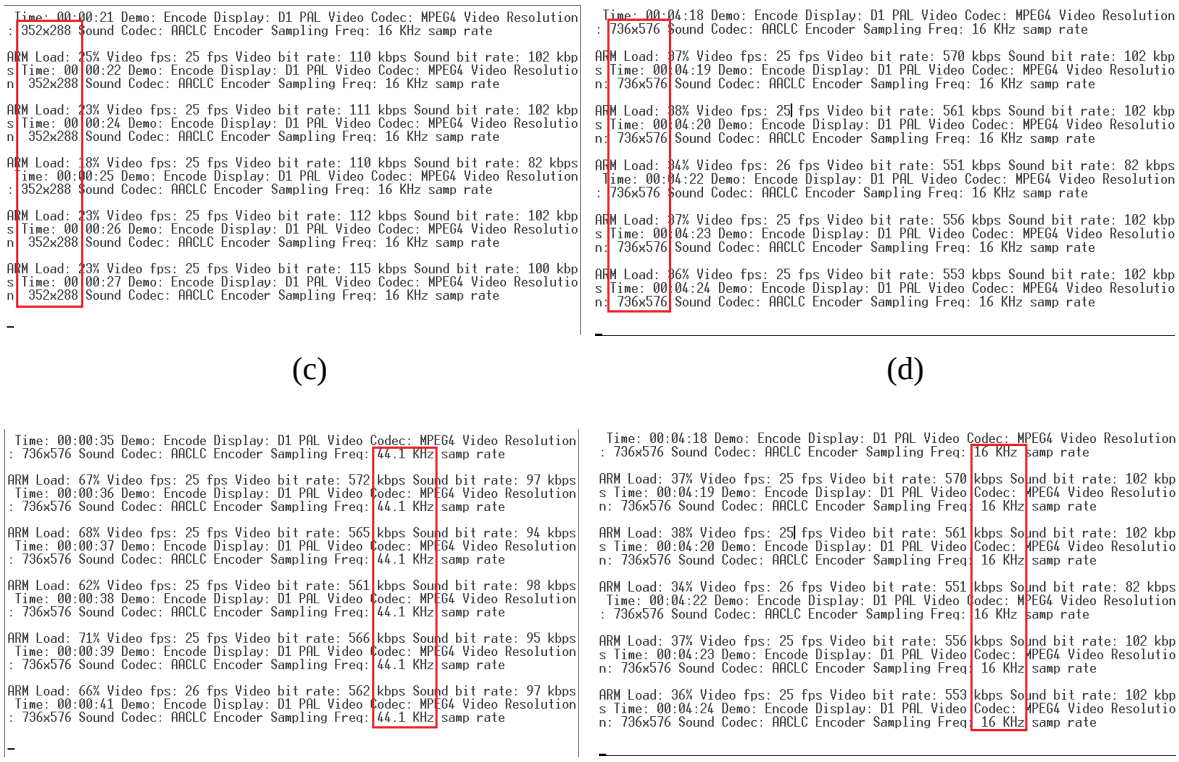
通过 EMIF 接口接收上位机传给 DM365 的音视频参数信息，程序接收到音视频配置信息后将音视频参数值赋给 encode 程序的命令行参数，执行 encode 程序，会根据这些参数值进行采集编码的工作，超级终端上会显示音视频参数信息，例如码率、帧率、分辨率等。



(a)



(b)



(c) (d)

(e) (f)

图 5.6 不同的音视频参数终端显示信息图

图(a)和图 (b)为对视频帧率的配置。其中，视频分辨率为默认的 736×576，码率会随着帧率变化，帧率越低，码率越低。图(c)和图(d)为对分辨率的配置，其中图(c)的分辨率为 352×288，图(d)的分辨率为 736×576，帧率都为 25fps，由于图(c)的分辨率低，所以数据量小，码率低。图(e)和图(f)为对音频采样率的配置，图(e)中的音频采样率为 44.1KHz，图 f 中的音频采样率为 16KHz。

5.1.5 视频字幕叠加程序测试

在本设计中，DM365 通过 SPI 接口读取上位机发送的时间信息，DM365 收到时间信息后，就用这个时间来校准嵌入式系统上的时间，由于上位机每隔一段时间就会向 DM365 发送当前时间，所以校准时间的程序放到后台循环执行。在超级终端用 date 命令就可以查看当前的系统时间。

时间校准程序通过 SPI 接口获得上位机发给 DM365 的时间信息，对系统时间进行校正后，encode 程序会在处理每一帧图像的时候提取一次系统时间，并在采集的 YUV 数据上通过点阵字库的方式叠加上系统时间字幕，然后压缩并封装，生成的 TS 流也带有时间字幕。如下图：

图 5.7 带字幕视频播放图

视频播放期间时间显示准确，声音和图像连续，播放正常。

经过对以上功能模块的测试，本文所设计的音视频程序完成了音视频信号的采集编码、封装传输、音视频参数配置和视频添加时间字幕的功能。在整体运行程序时，需要先将时间校准的程序放到后台运行，来接收上位机发送的时间信息并校准系统时间，然后运行读取音视频参数的程序来接收上位机传给 DM365 的音视频参数信息，然后把封装传输程序放到后台运行，等待接收音视频压缩数据，最后运行采集编码程序，以接收到的音视频参数值来配置采集编码参数，并在采集到的视频 YUV 数据上叠加上时间字幕。所有程序运行后，就可以实现把带有时间字幕的 TS 流传输给上位机并播放，并且上位机可以配置音视频参数。

5.2 程序烧写及系统启动

5.2.1 烧写内核和文件系统

首先需要将重新配置并编译的内核烧写进 Flash，其步骤为用 tftp 工具将内核镜像传到开发板的内存，用 nand erase 擦除 Flash 的一片存储空间，然后用 nand write 将内存中的内核镜像烧写进 Flash，下面输入并保存内核引导参数，就可以启动内核了。但这时 Flash 中没有文件系统，所以内核不能完全启动起来。

Linux 中并没有 Windows 中的 C 盘、D 盘、E 盘的分区，它以树状的结构管理所

有的目录和文件，其它分区挂接在挂接在挂接点上，挂接点就是某个目录，然后可以通过这个目录访问分区中的文件。例如根文件系统就是挂接在/目录下，/目录下有/bin、/etc、/usr、/mnt 等目录或文件，在/mnt 下可以挂接其它文件系统。

DVSDK 中提供了 jffs2 类型的日志文件系统映像，把内核烧入 flash 后，需要把文件系统烧入 flash 中，其具体步骤为，配置内核的引导参数并启动进入网络文件系统，建立放置文件系统的文件夹，用 flash_eraseall 命令烧写 flash 块设备，然后用 mount 命令将这个块设备挂载到刚建立的文件夹中。然后将文件系统压缩包解压到这个文件夹中，文件系统就被烧写进了 flash 中。然后设置从 flash 启动的地址等参数并保存，再次启动就可以启动日志文件系统了。

init 进程是由内核启动的第一个用户进程，它根据配置文件决定启动哪些程序，比如执行某些脚本、启动 shell、运行用户指定的程序等。init 进程是后续所有进程的发起者，比如 init 进程启动/bin/sh 程序后，才能在控制台上输入各种命令。进入日志文件系统后，会运行 init.d 文件夹中的脚本，我们需要把应用程序放到文件系统中，然后在 init 脚本中添加运行应用程序的命令来实现应用程序的自启动。

5.2.2 嵌入式 Linux 系统启动流程

将以上程序烧写进 Flash，板子加电后便可以启动系统并运行应用程序。

本系统的启动过程如下图

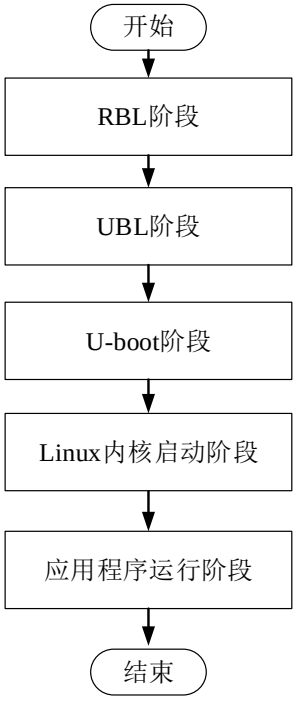


图 5.8 嵌入式 Linux 系统启动流程图

首先，保存在片内 ROM 的 RBL 程序开始运行，RBL 程序判断启动方式，如果是 EMIFA 启动方式，RBL 程序便从外接 Flash 的第一个扇区读取 RBL 的数据到内部 RAM 中，并转到执行 RBL 代码。

UBL 阶段主要完成系统时钟、DDR 频率的初始化，准备好加载 U-boot 镜像的环境。然后加载 U-boot 代码到 DDR 中，并跳转到 U-boot 代码中运行。

U-boot，全称 Universal Boot Loader，它是目前使用最广泛的 bootload（引导加载程序）。U-boot 代码首先设置最基本的系统硬件环境，然后配置系统的内存。将内核镜像和文件系统镜像文件从 Flash 上读到 RAM 空间中。为内核设置启动参数，调用内核。

Linux 内核从 U-boot 中得到参数并初始化相关硬件，压缩的内核进行解压并启动，进入日志文件系统，Linux 的第一个进程 init()运行，该进程根据系统的配置文件初始化系统。在 init 文件中加入应用程序的运行脚本就可以运行 5.1 节中测试的程序，在运行音视频应用程序之前，先要加载需要的内核模块，这样可以在内核保持不变的情况下，可以灵活地支持不同的硬件。在应用程序的运行脚本中，时间校正程序通过 SPI 读取上位机发送的时间信息，并校正板子的系统时间，读取音视频参数程序读取上位机发过来的音视频参数信息，音视频采集压缩程序用读取音视频参数程序收到的音视频参数配置音视频信号的采集编码，并通过管道发送给音视频打包程序，将音视频压缩码流封装成 TS 流，通过 EMIF 发送给上位机。上位机对接收到的数据进行解析，并播放，视频可正常播放。

5.3 本章总结

本章测试并分析了音视频应用程序的运行结果。先后对音视频采集编码程序、音视频封装程序、TS 流传输程序、音视频参数配置和视频字幕叠加程序进行了测试，并分析了运行结果。最后把程序烧写进硬件，完成了项目要求的功能。

第六章 总结与展望

6.1 总结

本文根据项目要求设计音视频采集传输的应用程序，硬件上采用了 TI 公司 DaVinci 系列 ARM+DSP 双核的多媒体处理器 TMS320DM365 作为处理芯片，通过摄像头采集视频，麦克风采集声音，视频经过 TVP5146 模数转换，音频经过 AIC3101 模数转换传输给 DM365。软件在 TI 的 DVSDK 平台上做开发，加入了音视频参数配置，添加时间字幕等功能，并基于 FFmpeg 开发了音视频封装传输程序。在本文中，设计系统方案实现了嵌入式音视频采集、压缩编码、封装、传输的功能，并详细分析了应用程序的设计以及实现。

本文的主要工作包括了以下部分：

1）针对嵌入式音视频采集传输程序设计的需求，分析了嵌入式音视频技术的发展及现状，对视频监控系统的架构做了介绍。并在此基础上，介绍了该系统开发所需要的相关技术，包括系统所需的处理器、TI 的软件 DVSDK、Linux 操作系统以及 Linux 系统上的编程。

2）对音视频信号的采集和编码程序的设计进行了详细的分析。介绍了音视频采集压缩的原理和音视频的压缩格式，分析了音视频采集编码多线程程序的设计，实现了音视频信号的采集和压缩。并分析了音视频参数配置和添加时间字幕的实现。

3）音视频封装传输程序实现。按照程序设计方案，基于 FFmpeg 实现了音视频压缩流封装成 TS 流并传输。介绍了 TS 流格式和 FFmpeg 以及 FFmpeg 的移植，并分析了 FFmpeg 封装音视频流的流程，以及通过 FFmpeg 内存数据操作来实现音视频数据的传输。

4）音视频程序功能测试。经过测试该系统完成了音视频信号的采集、压缩、封装和传输的功能。

6.2 展望

本论文中的嵌入式音视频系统完成了音视频技术中最基本的功能，而目前随着人工智能技术的突破，AI 技术也广泛应用于嵌入式音视频领域。例如目标识别算法已经应用于视频监控系统，可以在大量的监控系统中寻找目标，或者通过机器学习算法来检测火情、偷盗等情况。并且大多数嵌入式音视频系统都是固定的，没有移动性，受传输线的硬件和连接的影响，无法满足行业的特殊需求，如移动监控警察，巡逻车等，应该设计自动跟踪算法，对目标物体进行跟踪。另外，现在高清监控系统中的数

据量越来越大，可以用采用云存储技术上传到云服务器保存到硬盘中。并且目前 5G 技术已经成熟，给物联网的发展增添了动力，而以视频为基础的万物互联必定会蓬勃发展。本论文所阐述的嵌入式音视频系统中还有以下几个方面进行改进和完善：

1）本文中的音视频应用程序的编写用到了 DVSDK 和 FFmpeg，其中，运行 FFmpeg 编写的封装程序，需要将 FFmpeg 库烧写到开发板上，占用了存储空间，不如自己从底层开始写封装程序。另外本文中设计的应用程序为多进程，其在 Linux 上的切换开销比较大，将它们作为线程单元放到到一个程序中会提高运行效率。

2）本设计只是完成了音视频信号的采集和传输工作，没有加入其它算法。随着人工智能技术的发展，很多场景都在实现智能化。现在可以通过人工智能算法对视频进行分析，对视频中的目标进行识别，这个功能会大大的解放人力，在视频监控领域会发挥重要的作用。

参考文献

[1] 潘国辉. 智能网络视频监控技术详解与实践[M]. 北京: 清华大学出版社, 2010.

[2] 黄西. 基于可靠 UDP 的嵌入式视频监控系统的研究和实现[D]. 广州: 华南理工大学, 2014.

[3] Collins R T, Lipton A J, Kanade T, et al. A system for video surveillance and monitoring[J]. VSAM final report, 2000: 605-609.

[4] 杨建全, 梁华, 王成友. 视频监控技术的发展与现状[J]. 现代电子技术, 2006, 29(21): 84-88.

[5] 徐力, 孔岩. 视频监控系统的现状和发展趋势[J]. 信息技术与信息化, 2005(4): 60-62.

[6] 田天. 基于嵌入式音视频采集的视频会议系统的设计与实现[D]. 北京: 北京邮电大学, 2008.

[7] 郭波, 樊丁, 彭凯. 基于 DaVinci 技术的嵌入式视频监控系统设计[J]. 安防科技, 2010, 28(1): 82-84.

[8] Li H , Xue B , Li C , et al. Design and realization of embedded network video surveillance system based on the DM365[C]. International Conference on Computer Science & Education. IEEE, 2012.

[9] Talla D , Gobton J . Using DaVinci Technology for Digital Video Devices[J]. Computer, 2007, 40(10):53-61.

[10] 马洪蕊, 蒋心晓, 刘绍南. 嵌入式视频应用系统设计与实现[M]. 北京: 北京航空航天大学出版社, 2011.

[11] 刘君亮. 基于 TMS320DM365 的音视频传输及智能视频分析系统的设计与实现[D]. 南京: 南京邮电大学, 2012.

[12] 柴俊. MPEG-2 音视频流打包方法设计与实现[D]. 西安: 西安电子科技大学, 2017.

[13] 高玉龙, 白旭, 吴玮. 达芬奇技术开发基础、原理与实例[M]. 北京: 电子工业出版社, 2012.

[14] 许先斌, 熊慧君, 李洲, et al. 基于 ARM9 的嵌入式 Linux 开发流程的研究[J]. 微计算机信息, 2006, 22(11).

[15] 陈莉君, 康华. Linux 操作系统原理与应用[M]. 北京: 清华大学出版社, 2006.

[16] 崔珂, 吴镇炜, 刘明哲. 基于嵌入式实时 Linux 的远程监控系统[J]. 计算机工程与应用, 2005, 41(10):95-97.

[17] 许荣. 基于 AT91RM9200 的嵌入式 Linux 系统移植[J]. 电子技术, 2006, 33(3):62-66.

[18] 唐华. Linux 操作系统高级教程[M]. 北京: 电子工业出版社, 2008.

[19] 申时全. 基于 Linux 多线程技术的网络并发编程及应用研究[J]. 现代计算机, 2016(21):65-70.

[20] 金惠芳, 陶利民, 张基温. Linux 下多线程技术分析及应用[J]. 计算机系统应用, 2003, 12(9):30-32.

[21] 杨峰. 基于 DaVinci 技术的音视频压缩传输系统的设计与实现[D]. 西安: 西安电子科技大学, 2017.

[22] 姜永生, 姜艳芳, 梁绍敏. 多媒体技术与应用[M]. 北京: 高等教育出版社, 2012.

[23] 杨建平. 数字视频技术及应用[M]. 北京: 高等教育出版社, 2009.

[24] 仲颜. 基于 DM365 的视频传输流封装设计与实现[D]. 西安: 西安电子科技大学, 2017.

[25] 秦臻. 基于 DM365 的嵌入式视频监控系统设计[D]. 西安: 西安电子科技大学, 2012.

[26] 车晋. 数字电视音视频压缩编码技术分析与研究[J]. 广播电视信息, 2013(12):96-99.

[27] 彭启琮. 达芬奇技术—数字图像/视频信号处理新平台[M]. 北京: 电子工业出版社, 2008.

[28] 马汉杰, 冯杰, 张桦等. 嵌入式音视频应用—基于 TI DaVinci 技术[M], 北京: 电子工业出版社, 2016.

[29] 王慧州, 彭宏. 嵌入式视频监控系统终端软件设计与实现[J]. 微型机与应用, 2015(6):72-74.

[30] 徐殿武. 实时音视频数据采集和传输系统设计方法的比较研究[J]. 计算机工程与设计, 2008, 29(8):2017-2019.

[31] 韩瑜. 音视频信号采集及传输接口电路设计[D]. 西安: 西安电子科技大学, 2018.

[32] 郝赞. 模拟音视频信号采集系统设计及实现[D]. 西安: 西安电子科技大学, 2017.

[33] 杨知行, 王军, 王昭诚等. 数字电视传输技术[M]. 北京: 电子工业出版社，2011.

[34] 王峰, 王蜂, 夏良正. MPEG-2 系统多路 TS 流软件复用及实现[J]. 南京理工大学学报, 2002(S1):91-96.

[35] 王保雄, 余松煜, 庄建敏. MPEG-2 传输流中的时间信息与音视频同步[J]. 红外与激光工程, 2000, 29(5):76-79.

[36] 刘岐, 赵文杰. FFmpeg 从入门到精通[M]. 北京: 机械工业出版社, 2018.

[37] Wang W, Gao M. Design of embedded media player based on S3C2440 and SDL_FFMPEG[C] International Conference on Electrical & Control Engineering, 2011.

[38] Jianmin L , Bin Y . Video Hardware Codec Based on FFmpeg under Embedded Linux[J]. Microcontrollers & Embedded Systems, 2011.

[39] 李少春. 基于 FFMPEG 的嵌入式视频监控系统[J]. 电子技术, 2007(3):34-37.

[40] Zhao T, Peng H, Sun X. Design and implementation of an embedded video monitoring system based on S3C2440 and FFmpeg[M]. Chicago: Wireless Communication and Sensor Network:Proceedings of the International Conference on Wireless Communication and Sensor Network, 2015.

[41] Yun C, Liu Q, Zhao C. Design and Implementation of Mediaplayer Based on FFmpeg[M]. San Diego: Software Engineering and Knowledge Engineering: Theory and Practice. 2012.

[42] 刘建敏, 杨斌. 嵌入式 Linux 下基于 FFmpeg 的视频硬件编解码[J]. 单片机与嵌入式系统应用, 2011, 11(6):28-31.

[43] Wei Z, Gao Y, Zhang Z. The interface design and realization of EMIF and FPGA for wireless transmission of image data[C]. International Congress on Image & Signal Processing, 2014.

[44] 梁楠. 基于 TMS320DM365 视频采集与传输系统的研究[D]. 安徽: 安徽理工大学, 2014.

[45] 韦东山. 嵌入式 Linux 应用开发[M]. 北京: 人民邮电出版社, 2008.

[46] 文全刚, 张荣高. 嵌入式 Linux 操作系统原理与应用[M], 北京: 北京航空航天大学出版社, 2014.

[47] W. Richard Stevens, Stephen A. Rago. UNIX 环境高级编程[M]. 北京: 人民邮电出版社, 2006.

[48] 韩合民. 基于 ARM 平台的嵌入式流媒体播放技术的研究与应用[D]. 西安: 西安电子科技大学, 2008.

[49] 汪名扬. 嵌入式音视频通信平台研究开发及组网应用[D]. 北京: 中国科学院研究生院, 2008.

[50] 张静, 叶梧, 冯穗力. 基于 ARM920T 的嵌入式 Linux 系统开发[J]. 现代电子技术, 2005, 28(4):22-24.

致谢

研究生生活转眼就要接近尾声了，回首这三年的时光，我不仅学会了很多知识，还成熟了许多。我感谢这三年来老师们对我知识的传授、同学们对我的友爱，我感谢这段时间所有的经历对我的磨练，使我能自信而坚强地面对以后的工作和生活。

首先，在这里，我必须要感谢我的导师那彦教授，感谢他在学习及生活中给与我的指引与帮助。那老师勇于创新和严谨的科研精神影响了我做科研的态度，那老师会认真分析我的科研思路并给予建议和纠正，让我知道了做科研一定要认真仔细，思路清晰。每当我科研以及课题遇到难题时，那老师总是指导我做科研的方法并和我一起找寻思路，让我的研究做的更顺畅。那老师对我做科研的帮助使我受益良多，使我真正的踏入了理工科科研之路的大门。也正是因为那老师，我才有机会接触音视频领域以及嵌入式领域。那老师不仅在科研上帮助了我很多，在日常生活中那老师也对我们关怀备至，每当季节变更那老师都会提醒我们注意饮食以及穿衣，良好的身体才是科研的本钱。在此感谢那老师对我的谆谆教诲，也希望那老师身体健康、万事如意。

感谢陈建春教授对于我科研上的帮助，在做项目过程中总是用他渊博的知识给出指导。每次和陈老师交流，都能够学到很多有用的知识和资源，都能使我的眼界变得更加开阔。在此同样祝陈老师身体健康、万事如意。

其次，我要感谢项目组的其他成员们，正是你们的努力和辛勤工作才让项目得以顺利的进展，也让我学到了很多的知识。感谢柴俊师兄、仲颜师兄、杨峰师兄、郝赞师兄和韩瑜师兄对于我科研上的无私指导，正是由于师兄们的关心才能使得我快速融入到实验室的氛围中，也正是因为你们的帮助使我更加坚定地选择软件的道路。在此，祝师兄师姐们工作顺利，步步高升。

感谢和我在一起学习的的王强军同学、王金鹏同学、李欢欢同学、刘赫师弟和乜亮师弟，三年走来，我们不仅是同窗师兄弟，也是很好的伙伴。我们相互探讨、相互合作，一起攻克科研难题，一起加班加点地干项目，努力完成我们的目标，我会铭记这段美好的实验室生活。

感谢我的室友郭敏智同学和窦睿翰同学，每天虽然只能在晚寝的时间点进行简单的卧谈，但我们还是建立了深厚的友谊。一次次酸甜苦辣的分享，看似简单的聊天，却使得我们放松了心情释放了科研的压力，能以更饱满的状态投入到第二天的工作当中。

感谢我的父母，你们总是默默地支持着我，二十多年来对我关怀备至。大爱无私，润物细无声，有了你们的支持才有了现在的我。我即将踏入工作岗位，必将努力工作，以回报你们对我的养育之恩。

感谢我的好朋友们，分享我的喜怒哀乐，感谢你们的理解与支持。

最后，我还要向在百忙之中评阅论文和参加答辩的各位专家、教授致以真诚的谢意！

作者简介

1. 基本情况

高兴鹏，男，山东禹城人，1994 年 5 月出生，西安电子科技大学电子工程学院电路与系统专业 2016 级硕士研究生。

2. 教育背景

2012.08～2016.07 青岛理工大学，本科，专业：应用物理学
2016.09～2019.06 西安电子科技大学，硕士研究生，专业：电路与系统

3. 攻读硕士学位期间的研究成果

3.1 申请（授权）专利

[1] 高兴鹏. 校园音乐播放器软件 V1.0: 中国, 2018SR198284[P]. 2018-03-23.
[2] 那彦,高兴鹏,王金鹏. 基于 FFmpeg 和 EMIF 驱动的音视频数据传输方法: 中国, 201910273869.6[P]
[3] 那彦,赵丽,高兴鹏. 利用手机前置摄像头自动识别人眼中黑点和血丝的方法: 中国,201610877173.0[P]
[4] 那彦,王强军,刘赫,高兴鹏等. 滤波器预置深度学习神经网络的图像融合方法: 中国,201910205488.4[P]

3.2 参与科研项目及获奖

[1] 音视频采集压缩及传输系统, 2017.04~2019.05, 目前已完成音视频信号的采集压缩和传输，本人负责音视频信号采集压缩和封装传输的程序设计以及音视频参数配置和视频添加字幕的程序设计，经测试，所编写的程序完成了相应的功能。

西安電子科技大學
XIDIAN UNIVERSITY

地址：西安市太白南路2号
邮编：710071
网址：www.xidian.edu.cn