分类号_____ 密级_____

UDC[注1]_____

学 位 论 文

基于 TCP 传输的嵌入式流媒体播放系统

（题名和副题名）

周 司

（作者姓名）

指导教师姓名 孙建红 副教授

申请学位级别 硕士 专业名称 电子与通信工程

论文提交日期 2014.02 论文答辩日期 2014.03

学位授予单位和日期 南 京 理 工 大 学

答辩委员会主席_____

评阅人_____

2014 年 3 月 21 日

注 1：注明《国际十进分类法 UDC》的类号

声　明

本学位论文是我在导师的指导下取得的研究成果，尽我所知，在本学位论文中，除了加以标注和致谢的部分外，不包含其他人已经发表或公布过的研究成果，也不包含我为获得任何教育机构的学位或学历而使用过的材料。与我一同工作的同事对本学位论文做出的贡献均已在论文中作了明确的说明。

研究生签名：周司　　2014年3月25日

学位论文使用授权声明

南京理工大学有权保存本学位论文的电子和纸质文档，可以借阅或上网公布本学位论文的部分或全部内容，可以向有关部门或机构送交并授权其保存、借阅或上网公布本学位论文的部分或全部内容。对于保密论文，按保密的有关规定和程序处理。

研究生签名：周司　　2014年3月25日

摘 要

流媒体技术是一种新型的网络媒体传播技术，区别于传统的多媒体，其主要特点是以“音视频流”的方式进行媒体的实时传送，使人们可以在线观赏到不间断的高质量画面。流媒体传输作为流媒体的关键技术之一，传统上一般采用基于 UDP 的 RTP 协议，而 TCP 则由于本身需要较多的开销被认为不适合作为实时音频、视频传输的协议。然而，媒体流的传送并不是一定不能采用 TCP，它只是不适合对话方式的实时媒体流传送，但是在流媒体服务器的单向传送方式下，如果接收端缓冲区很大且播放时延也可以足够大,那么是可以使用 TCP 协议传送媒体流的。针对这种情况，论文设计了基于 TCP 传输的嵌入式流媒体播放系统。

论文首先介绍了课题的研究背景及意义，阐述了流媒体技术的国内外发展现状。同时介绍了嵌入式流媒体系统的关键技术，详细分析了实现这些技术的不同方法的优缺点。然后介绍了本系统的整体设计框架，并分别描述了各模块的设计架构。

其次，论文重点介绍了系统的软件硬件设计与实现。硬件部分主要是嵌入式终端播放器的硬件选型，选用 FL6410 开发板，并对其内核 ARM11、处理器 S3C6410 及主要硬件模块进行阐述。软件部分包括操作系统的选择、Linux 开发环境的搭建、嵌入式 ARM-Linux 系统构建、FFmpeg 库的分析、裁剪与移植、各模块具体功能实现及系统测试等，测试结果表明系统播放视频稳定流畅，实现了预期的目标。

最后，论文详细阐述了系统在数据传输模块实现速度控制的方法。

关键词：流媒体，TCP，嵌入式，ARM11，Linux

Abstract

Streaming media technology is different from traditional media, it is a new network media communication technologies. Its main feature is that it transmits the real-time media at "audio and video streaming" approach, so people can watch the high-quality screen uninterruptedly. As one of the key technologies of streaming media, the transmission of streaming media adopts RTP protocol based on UDP in traditional, while TCP is not considered suitable for real-time audio/video transmission protocol for its more overhead. However, it is not true that TCP can't be used to transfer streaming media always. It's just not suitable for transmitting real-time streaming media at dialogue. In the one-way transmission of streaming media, if the receiver has a large buffer and the player delay can be large enough, then media streams can be transmitted by TCP protocol. For this case, the paper designs an embedded streaming media playing system based on TCP transmission.

At first, the thesis introduces the research background and significance, describes the present developent situation of streaming media technology. It also introduces the key technologies of embedded streaming media system, and analysis both the advantages and disadvantages of different methods to achieve these technologies. Then the thesis introduces the overall design of the system's framework, and elaborates the design architecture of each module.

Secondly, the thesis focuses on the software and hardware design and implementation of the system. The hardware is mainly the hardware selection of the embedded playing terminal. It selectes FL6410 development board, and elaborates its core ARM11, S3C6410 processor and the main hardware modules. The software includes the choice of operating system, building Linux development environment, constructing embedded ARM-Linux system, analyzation, cutting and transplantation of FFmpeg library, achieving specific function of each module and system testing. The test result showes that the system can play video stablely and smoothly, achieves the desired goal.

Finally, the thesis elaborates the approach to achieve the speed control of the system's data transmission module.

Key words: Streaming media, TCP, Embedded, ARM11, Linux

目 录

1 绪论

1.1 课题研究背景

近年来，现代网络通信和多媒体技术的快速发展极大地推动 Internet。现在，Internet 提供给人们的服务不再局限于简单的图文，诸如远程教育、电子商务和视频点播的服务及应用如雨后春笋般出现。同时，在网络应用不断丰富的过程中，流媒体技术的出现改变了传统的媒体传播方式，也改变了传统的视频播放方式。

流媒体技术是当前最流行的网络传播技术之一，它是综合了多媒体领域和网络通信领域的交叉学科[1]，包含数据采集、音视频解码、传输、播放等技术[2][3]。利用流媒体技术首先要把编码处理后的音视频数据存放在流媒体服务器上，当客户端有播放请求时，首先会在本地创建一个缓冲区存储多媒体文件开始一部分的数据，然后客户端即可进行播放，与此同时，文件的剩余部分继续在后台下载[4][5]。为了保证播放质量，避免出现播放中断现象，当播放媒体数据的速率大于网络传输速率时，客户端就会播放从缓冲区中读取的数据[6]。流媒体由于其所具有的一些技术特点而应用于互联网信息服务的很多方面，诸如电子商务、视频点播/直播、远程医疗、远程教育等[7][8][9]。在互联网迅猛发展的时代，人们的日常生活和工作将因为流媒体的发展和应用而发生深刻的变化。

流媒体业务中，通常要求在带宽非常有限的网络上传输大量的音频、视频信息，并且视频服务必须能够满足一定的质量需求[10]。然而通常情况下，网络数据流量很大，如果不对媒体数据的传输速度加以控制，就比较容易造成网络拥塞，出现丢包等现象而影响服务质量[11][12]。因此，既要充分地利用网络带宽资源，同时又要达到一定的 QoS，实际上就是一个流量控制问题。网络流量控制主要分为流量整形、流量分类识别、队列调度及管理等几个模块，目前使用的流量控制策略主要有两种方式：基于信源的策略和基于接收机的策略[11][13][14]。对于传输流量控制中所涉及的传输协议，由于 Internet 传统通信协议不支持服务质量，为了满足实时性要求，网络视频通信一般采用基于 UDP 的 RTP 协议，而在相当长的时间里，TCP 由于其重传机制和拥塞控制机制都被认为不适用于实时流媒体传输。然而，TCP 并不是一定不能用于媒体流的传送，只要解决了用 TCP 传送媒体流时的时延波动问题，它也是可以用于实时视频传输的。现在国内很多主流的流媒体服务网站如爱奇艺、优酷等采用的 HTTP 渐行下载技术，其实际上就是基于 TCP 来传送媒体流的。

此外，作为流媒体系统必不可少的一部分—流媒体终端播放器，紧随着嵌入式技术的发展，也经历着一代又一代的变化。传统的播放器（如 Windows Media Player）都必须依赖于 PC，而不能独立于 PC 之外，这种局限性严重限制了流媒体播放器在其它独立于 PC 的设备、便携式或移动设备上的推广[15][16]。

1.2 流媒体技术的发展

随着计算机网络技术以及现代通信技术的不断发展，以 Internet 为核心的互联网络逐渐地成为继平面媒体、电视媒体和广播媒体之后的第四媒体。人们通过网络媒体来获取相关的服务和信息已经成为一种普遍的现象，在这个演变的过程中，人们对网络信息内容的要求也变得越来越高[17][18][19]。以前简单的图文就能够满足广大受众的需要，而现在人们更多的是要求信息具有高时效、多媒体、现场感、交互性等特点，这些不断提高的要求使得在网络上可以实时地传输多媒体内容的流媒体技术应运而生[20][21]。

在窄带互联网时代，为了促进经济的快速发展，人们希望能有一种网络技术使他们之间的信息沟通可以远程进行，因此，流媒体技术诞生了[22]。在某种意义上来说，美国的 Progressive Networks 公司，也就是现在的 RealNetworks，是流媒体领域的鼻祖。该公司成立于 1994 年，其推出的基于 C/S 架构的音频接收系统 Real Audio 后的曾经掀起了一场网络流式技术的热潮，并且始终处在这股潮流的领先地位。在 RealNetworks 公司的鼎盛时期，其在流媒体领域的的市场份额曾一度超过 85%，它在 1997 发布的 Realplayer 系列流媒体播放器至今仍被广泛地应用于多种场合[23]。

然而，RealNetworks 公司也无时无刻地感受到危机的存在，流媒体的大好发展前景被 Apple 和微软等看好，它们的加入带来了流媒体领域激烈的竞争攻势，同时也促进了流媒体的发展速度。

YunEr 是中国第一家面向全国进行服务的流媒体信息网，于 1999 年 6 月引入海外的流广播。它具有强烈的实效性和文化性，富有广泛影响的公众性和社会性以及深远的意义和作用。其定位于多维视听娱乐平台，依托其系列网站通过互联网、有线网、无线通讯网、卫星等多渠道的形式向全国用户提供一个集合多媒体信息内容的咨询平台，其中包括海内外影视和音乐等版权内容，网友 DIY 作品、传统媒体的节目直播、传统性质的大中小型活动的现场直播及 YunEr Live!的流广播节目等，同时为相关企事业单位提供流技术和节目设计制作的服务，让广大人们能无限领略到网络技术的深层次应用魅力[22]。

此后许多电台、电视台都利用流媒体技术软件提供节目的网上直播或点播服务。普通用户也可以借助一定的软硬件设备进行网上音视频信息的发布。1999 年，上海通力公司采用 Realsystem G2 帮助中央电视台完成了 CCTV 春节联欢晚会在 Internet 上的实时直播。我国最早发布的中文信息网站之一中央电视台的央视网际网站 CCTV.com，目前已经发展成以新闻及电视相关内容为主、以视频为特色的流式媒体专业站点[22]。

目前，我国在流媒体业务方面存在建设重复、规模偏小、安全性较差的特点，这主要是由于全国各主要媒体的直播和点播系统平台采用的核心技术几乎都是国外的，并没有一套属于自己的完善的流媒体技术理论体系，造成在我国难以大规模地开展流媒体业

务的局面。第四媒体 Internet 成为主流的互联网信息服务平台，超过平面媒体、电视媒体和广播媒体，是宽带网络发展带来的必然趋势。由此可见，一个国家能否自己开发具有自主知识产权且安全性高的流媒体业务平台是非常关键的，同时应当站在国家安全的高度看待流媒体的安全性问题。

流媒体业务之所以能够如此快速的在多媒体领域占据市场，要归功于因特网的发展和普及。作为一种日益时兴的新媒体技术，流媒体的强大魅力在于它改变了传统互联网在人们心中机械的形象，能够满足人们在生活（如娱乐活动）、工作（如视频会议和新闻出版）、理财（如证券交易和电子商务）、学习（如远程培训和远程教育）等方面的服务需求，其应用掀起了互联网信息服务的一次革命。

1.3 基于 PC 的流媒体播放器发展状况

流媒体技术的最终目的是将多媒体数据通过终端播放器高质量地呈现给用户。当前主流的流媒体播放器有 Windows Media 的技术代表 Windows Media Player，Real Audio/Video 的技术代表 RealPlayer，及 Apple 公司的 QT 技术代表 QuickTime[24]。另外，还有其他一些具有代表性的媒体播放器，如 DivX Player 是 DivX 影片的专用播放器；暴风影音是一个媒体播放的通用解决方案；超级解霸不仅是一款多功能的视、音频播放器，同时也是一个针对 IPTV 领域的互动娱乐平台，等等[7]。

Windows Media Player：Windows 系统最早的一个媒体播放组件，自 7.0 版本后，已发展成为一个全功能的网络多媒体播放软件。Windows Media Player 支持的视频流媒体格式主要是 Microsoft 自己开发的 ASF、WMV 及 WMA 格式，WMV 及 WMA 格式在本质上就是 ASF 格式。Windows Media Player 外观华丽，内核性能出色，但是由于集成了较多的附加功能，如广播电台、播放复制 CD、寻找 Internet 视频等，因此整体速度较慢。最新的 Windows Media Player10 适应于 Windows XP 系统，可提供广泛、流畅的网络媒体播放方案，支持目前绝大多数媒体格式；能够智能监测网络的速度并调整播放窗口大小和播放速度；可以播放和组织计算机及 Internet 上的数字媒体文件，收听全世界的电台广播；能够播放和复制 CD，创建自己的 CD，播放 DVD 以及将音乐或视频复制到便携设备（如便携式数字音频播放器和 Pocket PC）中；还可以通过访问播放器的在线商店区域，查找并购买 Internet 上的数字媒体内容。

RealPlayer：作为流媒体技术的先行者，Real Networks 公司至今仍拥有一大批相对固定的忠实用户。与 Windows Media Player 相比，RealPlayer 显得更简洁实用、更具娱乐性、更像一个纯粹的流媒体播放器。最新的 RealPlayer 11 是一个高集成度的专业流媒体影音播放平台，能够很好的兼容 Windows Vista 和 XP。新版本在许多方面都有所提升，如音/视频的整体播放性能、音/视频编辑功能、媒体文件网络传输与刻录功能、网络游戏与网络歌曲等娱乐扩展功能，等等。

QuickTime：Apple 公司出品的 QuickTime 是一个集数字媒体下载、播放以及编辑功能于一体的软件包。1999 年发布的 QuickTime 4.0 版本开始支持流媒体技术，视频格式为 MOV。MOV 的视频压缩部分采用 Sorenson Video 技术，该技术支持 VBR（Variable Bit Rate,可变比特率），可以动态地分配带宽，以尽可能小的文件获得最好的播放效果，并能在解压缩时获得平滑流畅的画面；音频部分，QuickTime 采用一种名为 QDesiglMusic 的技术，是一种比 MP3 更好的音视频流技术。在 MPEG-4 问世之前，MOV 格式被用于提供质量优异的网上影片，当时很多电影发行网站都使用 MOV 格式作为标准的试映片播放格式。2005 年 6 月，Apple 公司发布了 QuickTime 7 Player 和 QuickTime 7 Pro for Windows，被认为是业界第一款在 Windows 平台上回放流式高清晰视频的主流 H.264 解决方案。QuickTime 7 Pro 为所有喜欢观看、录制、制作和共享高品质视/音频的人们提供了完美的解决方案。

尽管上述三款主流的流媒体播放器播放功能强大且用户界面友好，但是它们都必须依赖于 PC，而不能独立于 PC 之外。这种局限性严重限制了流媒体播放器在其它独立于 PC 的设备、便携式或移动设备上的推广。

1.4 论文主要工作

本文主要对嵌入式系统和流媒体技术进行研究，特别是流媒体相关的网络传输协议及数据传输速度控制机制，以 FFmpeg 项目作为中间件，以高性价比的 ARM 架构处理器为硬件平台，以 Linux 作为嵌入式操作系统，在此基础上结合流媒体技术和嵌入式技术，并对 TCP 的拥塞控制机制进行改善，实现媒体流传送速度控制，设计并完成了基于 TCP 传输的嵌入式流媒体播放系统的应用程序开发。

自从确定课题方向以来，论文主要的研究工作包括以下几个阶段：

1. 学习流媒体方面的相关技术理论，掌握流媒体的基本思想及关键技术特点，了解流媒体技术的当前发展动态，奠定开展论文工作的基础。

2. 对多种用作音/视频流方案的库和中间件进行调研，选用开源的 FFmpeg 项目作为系统开发的中间件。其较全面的音视频功能及支持多种媒体格式的特点，极大地降低了开发的难度。

3. 分析嵌入式流媒体播放系统的技术特点及功能需求，设计系统总体实现方案。媒体源模块的设计选择直接流化已经录制好的视频文件，流媒体服务器接收媒体源模块传送的流并传递给终端播放器，这两者的程序运行在 Linux 操作系统的普通 PC 上。对于流媒体终端播放器，其嵌入式处理器选择 ARM11 处理器 S3C6410，其嵌入式操作系统选择嵌入式 Linux 操作系统。在系统的传输协议上，区别于传统的 RTP/RTCP/RTSP,选择基于 TCP 的 HTTP 协议，其中，HTTP 负责会话，TCP 负责传输。

4. 学习 ARM 体系架构，以 ARM11 处理器 S3C6410 为核心，设计嵌入式流媒体终

端播放器硬件平台。

5. 研究相关嵌入式领域的技术知识，学习嵌入式 Linux 操作系统构建方法，搭建 ARM-Linux 系统平台。

6. 研究 FFmpeg 库源代码，学习 FFmpeg 的架构，掌握 FFmpeg 库的裁剪、安装及移植方法，并将裁剪后的满足设计需求的 FFmpeg 库分别安装在基于 x86 架构的宿主机系统上、基于 ARM 架构的嵌入式开发板上。

7. 研究流媒体传输速度控制原理及网络传输协议 TCP 协议的超时重传和拥塞控制机制，在 TCP 对拥塞反应的强烈的基础上设计了一种基于终端播放器反馈、流媒体服务器控制的改进的适合流媒体传输速度控制的方法。

8. 在完成系统应用程序开发的基础上，将终端播放器程序移植到 ARM 架构开发板上，测试并验证系统功能。

1.5 论文组织结构

全文总共分为七章，具体内容安排如下：

第一章是绪论。通过大量地查阅相关文献，主要对课题研究背景及意义、流媒体技术进行概述，并介绍了论文的主要工作内容和结构安排。

第二章是系统设计和框架。首先从流媒体传输方式、流媒体传输协议、流媒体播放技术、流媒体服务器及嵌入式系统等几个方面对嵌入式流媒体系统涉及的关键技术进行阐述，并确定了系统及核心模块的总体设计方案。

第三章是系统硬件设计与实现。对嵌入式处理器进行了介绍并比较，选择飞凌公司的 FL6410 开发板，并对硬件选型进行解释。

第四章是嵌入式流媒体播放系统平台的构建。这部分内容包括操作系统的介绍与选择，系统开发环境的搭建、嵌入式 ARM-Linux 操作系统的构建及移植。

第五章是系统软件设计与实现。首先分析了 FFmpeg 库的架构，并对其进行裁剪、安装，然后详细讲述了实现流媒体播放系统及其核心模块的软件工作流程、相关重要数据结构，并展示播放结果。

第六章是流媒体传输速度控制的实现。详细地给出了嵌入式流媒体播放系统在媒体流传输过程中实现传输速度控制的策略。

第七章是总结与展望。对全文工作的一个总结，并对嵌入式流媒体播放系统未来需要完善的功能和研究方向进行展望。

2 系统设计和框架

2.1 嵌入式流媒体播放系统相关技术

流媒体技术不是一种单一的技术，而是音频/视频技术和网络技术的有机结合。要在网络上实现流媒体技术，需要解决流媒体的制作、发布、传输以及播放等方面的问题，而这些问题就需要利用网络技术和音/视频技术来解决，而嵌入式流媒体播放系统又是嵌入式技术与流媒体技术的结合。

2.1.1 流媒体传输方式

如果把文件传输过程看作是一次接水的过程，文件相当于桶中的水，那么在过去的传输方式中用户必须要等到一桶水接满时才可以使用桶中的水，就像是对用户的一个规定，显然桶的大小和水流量大小会影响等待时间。然而，对于流式传输，用户随时都可以用上水，只要打开水龙头并稍微等待一小会儿，水就会源源不断地流出来，这种随接随用的方式既不会受到桶的大小的影响，也不用担心水流量的大小[7]。

流式传输技术分为两种，实时流式传输和顺序流式传输[22]。下面对这两种传输方法进行简要的介绍。

实时流式传输：所谓实时流式传输就是指实时传送，用户可以实时地观赏到网络上传输过来的媒体内容，且用户在观看的过程中可以对多媒体进行控制，根据自己的需求选择观看前面或者后面的内容。这种实时性使得实时流式传输特别适合现场事件，但是它要求在传输过程中媒体信号带宽要与网络连接带宽匹配，而且由于是实时播放的，所以在这种传输方式中网络的状况直接影响播放质量。尽管在理论上实时流一旦开始播放就可以不间断，但在实际情况中很可能由于网络状况不理想而出现周期性暂停现象，而且当网络出现问题或者拥挤时，分组的丢失也将导致视频质量变差[25][26]。为了能够在传输时对多媒体数据进行更多级别的控制，实时流式传输需要如 QuickTime Streaming Server，RealServer 与 Windows Media Server 等这种设置及管理相对复杂的专用服务器。此外，它与 HTTP 流式传输不同，它需要专用的传输协议，如 RTSP 协议或 MMS 协议等，但这些协议在有防火墙时可能被防火墙阻拦，导致用户不能看到实时内容。

顺序流式传输：所谓顺序流式传输就是顺序下载，用户可以边下载边观看，但是比较受限的是用户不能随机访问而只可以观看服务器在若干时间以前发送的那部分信息，所以它不适合有随机访问要求的视频，如演说与演示、讲座等。另外，与实时流式传输不同，顺序流式传输不能在传输的过程中根据网络连接速度调整传输速度，因而它也不适合于片段长的视频。由于顺序流式传输的这些特点与标准 HTTP 服务器发送文件的形式相同，而且这种服务器易于管理与防火墙无关，所以一般把顺序流式文件存放在标准

HTTP 服务器或者 FTP 服务器上，因而顺序流式传输又被称为 HTTP 流式传输。由于它可以比较好地保证节目的最终播放质量，所以它比较适合于高质量的短片段[25]。另外，它不支持现场广播，严格地来说，它是一种点播技术，适合于在网站上发布的以供用户点播的音频/视频节目。

表 2.1 实时流与顺序流的对比表

区 别	实时流	顺序流
音视频数据源	实时从录制设备上采集，或者使用专用协议传输的文件	可播放的音视频文件
服务器类型	专用流媒体服务器，如： Windows Media Server QuickTime Streaming Server Real Server Flash Media Server	FTP 服务器，或普通的 HTTP 服务器
传输协议	专用协议 HLS 或 RTMP 等	一般的 HTTP 协议，与传输网页的协议相同
跳播	可随机访问任意片段	不能随机观看，只能够访问已经下载的内容

因而，采用顺序流式传输方式可以更好地保证视频的传输质量。

2.1.2 流媒体传输协议

因特网最初是为了传输文本数据设计的，它不支持服务质量。但是对于数据量庞大的多媒体内容来说，它有着较高的实时性要求，特别是对于网络音视频应用，用户需要能达到实时传输、连续播放的效果。因此，这种强大的服务质量需求要求系统能够在网络吞吐量、网络延时、传输抖动、分组出错率等方面做出一定的保证[27]。

显然，传统的因特网通信协议是不适合传输有实时要求的连续时基媒体的。能够满足实时性要求的传输协议有 TCP(Transfer Control Protocol，传输控制协议)和 UDP（User Datagram Protocol,用户数据报协议），但是由于面向连接的、可靠的 TCP 协议具有超时重传和拥塞控制机制需要较大的开销，因此它被认为不适合用于实时多媒体传输。所以，因特网采用 UDP 进行实时多媒体通信，然而由于 UDP 的无连接、不可靠特性，在传输实时内容时容易出现分组丢失现象，从而导致了通信质量下降。所以为了解决这个问题，IP 协议提出对所有数据通信进行分类，为不同的应用指定优先权和安排预订机制的根本思想。其中，IETF 专为因特网传输多媒体数据而开发了一个因特网增强服务模型，它包括全力传送(best-effort)，实时传送(real-time)和相应的多媒体数据流实时传输协议。与此同时，其他的一些多媒体传输标准也相继被其他的国际标准化组织和研究机构制定出

来，在一定程度上这些协议满足了在 Internet 上实时传输多媒体数据的要求。网络视频协议在 TCP/IP 协议栈中的位置如表2.2所示。

表 2.2 网络视频协议在 TCP/IP 协议栈中的位置

应用层	HTTP RMTP TFTP RTSP MMS
传输层	RTP RTCP RSVP
	UDP TCP ST-II
网络层	IPv6 BGP RIP OSFP ST-II
网络接口层	Ethernet ATM MPLS ISDN FR
物理层	UTP SDH SONET DWDM

传统的流媒体传输一般采用 RTP/RTCP/RTSP 协议。

RTP 协议是 Internet 上的传输层协议，通常为一个具体的应用来提供服务，提供端到端的网络传输功能，一般基于 UDP 协议来传输多媒体数据流，但由于 RTP 不对下层协议做任何指定，所以它也可以建立在 TCP 或者其他传输协议之上。RTP 传送的数据包带有实时信息，为了能够通过其中的时间信息实现媒体流的同步，它工作在一对一或一对多的传输情况下。RTP 不涉及质量保证和资源预留等实时服务，适合以组播和点播方式传输音视频数据、仿真数据等实时数据[23]（组播和点播将在下一节中详细介绍）。

RTCP 协议是应用层上一个控制协议，它与 RTP 配合使用来提供拥塞控制和流量控制服务，其关键作用就是让接收方能同步多个 RTP 流。在进行后者的会话期间，各会话参与者周期性地给服务器发送包含有已发送数据包数量、丢失数据包数量等统计资料的 RTCP 控制包[28][29]，服务器利用该包中的信息启动流量控制策略以提高网络传输效率。RTP 与 RTCP 一起通过这种反馈机制实现对流媒体传输服务质量 QoS 控制，保证数据包正确、可靠的传送，保证流媒体的实时性特征，非常适合网络上实时数据的传送[19]。

RTSP 协议用于控制实时数据的发送，是在服务器和客户端之间建立的控制音视频流的应用层协议。它在功能上与 HTTP 有重叠，在体系架构上，其位于 RTP 协议和 RTCP 协议之上[30]，使用 TCP 或者 RTP 完成数据传输。RTSP 协议定义了如何通过 IP 网络来有效地传送实时数据，提供了一个使音视频等实时数据的点播、受控成为可能的可扩展框架。该协议提供用于音频和视频流的“VCR 模式”远程控制功能，用于控制流媒体的播放、暂停等操作，在服务器和客户端之间扮演着远程遥控器的角色[31][32][33]。RTSP 协议控制的流，其操作不依赖于携带连续媒体的传输机制。

总而言之，在流媒体数据传输过程中，一般采用 RTP 封装实时多媒体数据，提供解码时所需的编码类型和时间戳等相关信息；采用 RTCP 反馈网络传输状况，提供拥塞处理机制和流量控制机制；采用 RTSP 对具有实时特性的流媒体数据的传输进行控制。三者之间协议栈的设计如图 2.1 所示。

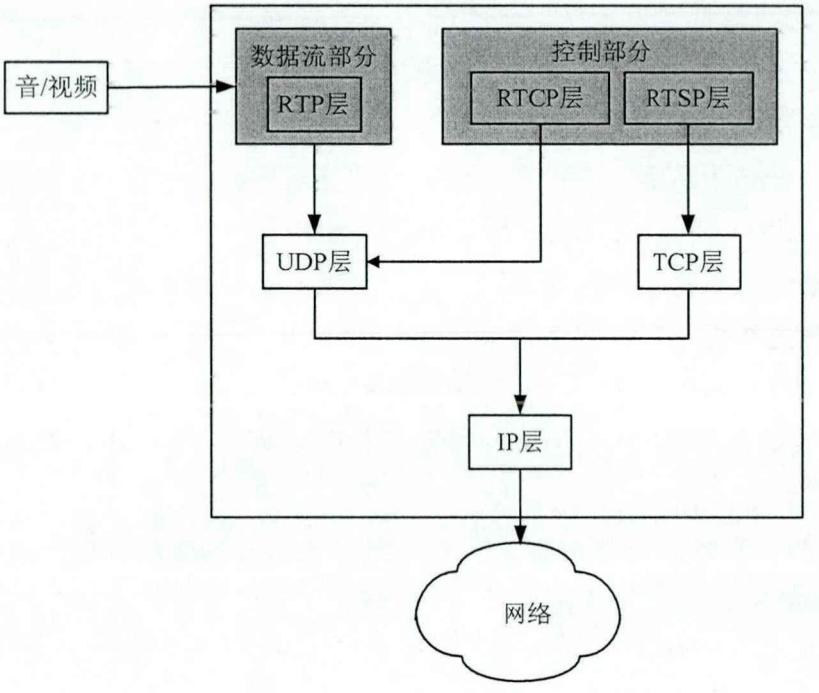

图 2.1 流媒体传输 RTP/RTCP/RTSP 通信协议栈

在相当长的一段时间里，由于 TCP 协议本身具有拥塞控制及重传时延的特性，需要较多的开销，都被认为不适合作为实时音/视频传输的协议[34]。

2.1.3 流媒体播放技术

对于用户来说，要真正能够看到和听到多媒体内容，必须要有相应的媒体播放技术。

(1) 单播（Unicast）

单播是指客户端与服务器之间的点到点的连接，一台服务器发送的每个数据包通过二者之间的独立数据通道传送给客户端，但是仅可以传送给一个客户端。在单播方式下，只有一个发送方和一个接收方。因此，如果有多个接收者需要相同的数据，那么这个发送者需要复制多份相同的数据包进行发送。所以，如果当有大量的主机希望获得数据包的同一份拷贝时，上述方法将导致发送者负担沉重、传输延迟长，这样比较容易造成网络拥塞。

单播的优点：

a) 对于客户端提出的请求，服务器可以及时响应。

b) 针对每个客户提出的不同的请求，服务器相应地发送不同的数据，可以实现个性化服务。

单播的缺点：

a) 由于服务器针对每个客户机发送不同的数据流，所以，服务器流量=客户机数量×客户机流量，那么在客户机数量多、每个客户机流量大的流媒体应用中服务器将不堪重负。

b) 现有的网络带宽是金字塔结构的，在城际省际网络，若全部使用单播协议，则会导致网络主干瘫痪，因为其主干带宽只相当于其所有用户带宽之和的 5%。

(2) 组播（Multicast）

组播是一种基于网络硬件设备实现的一种分组广播的数据传输方式，组播数据传输时，服务器可以将一个数据包通过网络硬件设备复制的方法同时分送给多个需要接收的客户端，网络中的所有客户端共享同一流。这种方式很适合通过网络传输多媒体内容，因为其最大好处是可以节省网络带宽。

组播的优点：

a) 节省服务器负载，因为当客户端需要相同的数据流时，会以共享的方法加入相同的组。

b) 服务器的服务总带宽不会受到客户端接入带宽的限制，因为组播协议复制和转发数据流是按照接收者的需求的。

组播的缺点：

a) 组播协议不像单播协议那样有纠错机制，在发生丢包或错包后需要通过一定的容错机制及 QoS 来进行弥补。

b) 尽管现行网络都支持组播的传输，然而其在 QoS、客户认证等方面仍然需要完善，理论上这些缺点都有较为成熟的解决方案，只是需要逐步地推广并应用到现存网络中。

(3)点播（On demand）

点播是客户端与服务器之间的主动连接，在这种播放方式中，客户端拥有流的控制权，可以像控制本地文件一样操控媒体流，如开始、暂停、停止、快进或者快退等操作。在点播连接中，每个客户端通过选择内容项目各自连接到服务器，同时独占这个连接并利用这个连接接收媒体流。因此，点播的播放方式对服务器资源和网络带宽的要求都相对较高。

(4)广播(Broadcast)

广播是客户端被动接收媒体流的播放方式，它不允许客户端对媒体流进行停止、快进快退等操作。在广播过程中，采用的数据发送手段既可以是广播也可以是单播，若使用前者则服务器不管客户端是否需要都会发送数据包的一个拷贝给网络上的所有用户，若使用后者则服务器需要以多个点对点的方式分别将数据包的多个拷贝发送给那些需要的客户端。

广播的优点：

a) 布网成本低廉，网络设备及维护简单。

b) 服务器流量负载极低，因为服务器不需要向每个客户机单独地发送数据。

广播的缺点：

a) 不能像单播那样针对每个用户的不同的需求和时间来及时地提供个性化的服务。

b) 网络允许服务器提供的数据带宽比较有限，客户端的最大带宽=服务总带宽。

c) 广播禁止在 Internet 宽带网上传输。

2.1.4 流媒体服务器

流媒体服务器的主要功能是以流式协议（RTP/RTSP、MMS、RTMP、HTTP 等）将视频文件传输到客户端，供用户在线观看；也可从视频采集、压缩软件接收实时视频流，再以流式协议直播给客户端。

现在主流的流媒体服务器采用的协议有：RTSP，RTMP 和 HLS。

RTSP（Real-Time Streaming Protocol）实时流协议，是由 Netscape 公司和 RealNetworks 及哥伦比亚大学共同研制的。它是从 Netscape 的"LiveMedia"和 RealNetwoks 的"RealAudio"的实践和经验发展而来的。第一份 RTSP 协议是由 IETF 在 1996 年 8 月 9 日正式提交，1998 年 4 月成为 Internet 的正式标准，此后该协议又经过了很多明显的变化，现在已被广泛应用，APPLE、IBM、Netscape、Silicon Graphics、Vxtreme、Sun 等公司都宣布它们的在线播放器支持 RTSP 协议，不过 Microsoft 公司一直都不支持此协议。它是一个非常类似于 HTTP 的应用层协议，每个发布的媒体文件都被定义为 RTSP URL。媒体文件的发布信息被写为媒体发布文件，在这个文件中包括编码器、语言、RTSP URL、地址、端口号以及其他相关参数。RTSP 主要用在安防领域，实现比较成熟，苹果的 DSS 流媒体服务器就是基于 RTSP 协议开发。然而，RTSP 对码流进行传输时对网络的要求也比较高，很多局域网或者广域网出于安全考虑，都禁用 RTSP 协议，所以造成 RTSP 现在的应用领域主要局限在安防领域。另外，不是所有客户端播放器都支持 RTSP 的码流，比如最常见的 flash 播放器就不支持 RTSP 协议，客户端要想使用 RTSP 协议提供的码流，还必须安装特定的播放器，这些都限制了 RTSP 的使用范围。以上这些原因造成 RTSP 在实际中的应用范围越来越小。RTSP 一般用在广电安防，使用 TS 流对数据进行分发。

RTMP（Real Time Messaging Protocol）实时消息传送协议，是专门用来传输音视频数据的流媒体协议，最初由 Macromedia 公司创建，后来归 Adobe 公司所有，是一种私有协议，主要用来联系 Flash Player 和 RtmpServer，如 FMS，Red5，crtmpserver 等。RTMP 协议可用于实现直播、点播应用，通过 FMLE（Flash Media Live Encoder）推送音视频数据至 RtmpServer，可实现摄像头实时直播。不过，毕竟 FMLE 应用范围有限，想要把它嵌入到自己的程序中，还是要自己来实现 RTMP 协议的推送。RTMP 协议是 Adobe 开发的用来传输对象、音频、视频的协议，这个协议建立在 TCP 协议或者轮询 HTTP 协议之上[35]。RTMP 协议就像一个用来装数据包的容器,这些数据既可以是 AMF 格式的数据，也可以是 FLV 中的视/音频数据[35][36]。支持 RTMP 协议的流媒体服务器有 FMS

和 RED5，开源的 Web 服务器 Nginx 经过扩展也可以很好的支持 RTMP 协议。RTMP 协议目前的使用越来越广泛，基于 TCP 协议使它不受防火墙的限制，而且 flash 播放器可以完美的支持 RTMP 协议的码流，不需要用户安装特定客户端。RTMP 一般运用在视频会议，通过 web 方式、使用 mp4 或 flv 等容器来分发数据。

HLS 协议（Http Live Stream）是由 Apple 公司定义的用于实时流传输的协议，HLS 协议基于 HTTP 协议实现，是 Apple 公司的动态码率自适应技术,主要用于 PC 和 Apple 终端的音视频服务[37]，包括一个 m3u(8)的索引文件，TS 媒体分片文件和 key 加密串文件。现在主流的流媒体服务网站，如优酷、爱奇艺等，都是用这种 HTTP 渐行下载技术。相对于常见的流媒体直播协议，例如 RTMP 协议、RTSP 协议、MMS 协议等，HLS 直播最大的不同在于，直播客户端获取到的，并不是一个完整的数据流。HLS 协议在服务器端将直播数据流存储为连续的、很短时长的媒体文件（MPEG-TS 格式），而客户端则不断的下载并播放这些小文件，因为服务器端总是会将最新的直播数据生成新的小文件，这样客户端只要不停的按顺序播放从服务器获取到的文件，就实现了直播。由此可见，基本上可以认为，HLS 是以点播的技术方式来实现直播。由于数据通过 HTTP 协议传输，所以完全不用考虑防火墙或者代理的问题，而且分段文件的时长很短，客户端可以很快的选择和切换码率，以适应不同带宽条件下的播放。不过 HLS 的这种技术特点，决定了它的延迟一般总是会高于普通的流媒体直播协议。这种协议的特点就是实现简单，不受防火墙的制约，对客户端无特别要求[38]。用户通过 HTTP 协议请求视频文件的 m3u(8)索引文件（索引文件是视频分段的索引信息，该索引文件既可以用于直播，也可以用于点播），然后播放器解析 m3u(8)索引文件，并通过 HTTP 协议向流媒体服务器请求码流，服务器通过 TCP 协议将码流传递给客户端。使用 m3u(8)播放列表的时候，应该尽量减少 TS 的分片大小，可以减小缓冲的时间，如果终端的下载速度跟得上，那么只要缓冲一个文件就可以实现连续播放了[39]。相比于 RTMP、RTSP 协议，HLS 不适合做直播。

三者之间的对比关系如表 2.3 所示。

表 2.3 主流流媒体服务器协议对比表

区别	RTMP	HLS	RTSP
全称	Real Time Message Protocol	Http Live Stream	Real Time Streaming Protocol
上层协议	TCP 或 HTTP	HTTP	RTP，RTCP
软件模型	C\S	B\S	C\S
研发主要来自	Adobe	Apple	Microsoft

针对客户端	支持 Flash 类产品的浏览器、支持 HTML5 的浏览器	支持 HTML5 的浏览器	播放器
视频格式要求	FLV， F4V	MP4	无
服务器要求	专用 Flash 服务器 Flash Media Server Red5	普通 HTTP 服务器	专用 RTSP 流媒体服务器
实况直播要求	专用编码器上传 Flash Media Encoder	专用编码器上传 Apple 开发工具	与服务器相关，自定义上传
文件播放要求	Flv ，F4V 文件即可，服务器会自动分解为 F4f 数据文件、f4x 索引文件	Ts 数据文件，M3u8 索引文件	与服务器相关，与播放器相关

2.1.5 嵌入式系统

嵌入式系统（Embedded System）是针对具体应用而定制的专用计算机系统，是信息技术（Information Technology）在后 PC 时代最重要的发展方向[40]。它一般指非 PC 系统，包括硬件和软件两部分。硬件包括处理器（或微处理器）、存储器、I/O 器件和图形控制器等，软件包括操作系统和应用程序。操作系统控制应用程序与硬件的交互，完成多任务操作，应用程序控制系统的运作和行为，完成各种设计功能。从技术角度上看，嵌入式系统是根据用户在系统某些方面的特殊要求（如功能、成本、可靠性、功耗及应用环境等）而定制的专用计算机系统，同时它的软硬件可裁剪，其中心是应用，基础是计算机技术。广而言之，可以认为凡带有微处理器的专用软硬件系统都可以称为嵌入式系统，它采用“量体裁衣”的方式把所需的功能嵌入到各种应用系统中，融合了计算机软硬件、通信和半导体微电子技术，是信息技术的最终产品。其体系结构如图 2.2 所示：

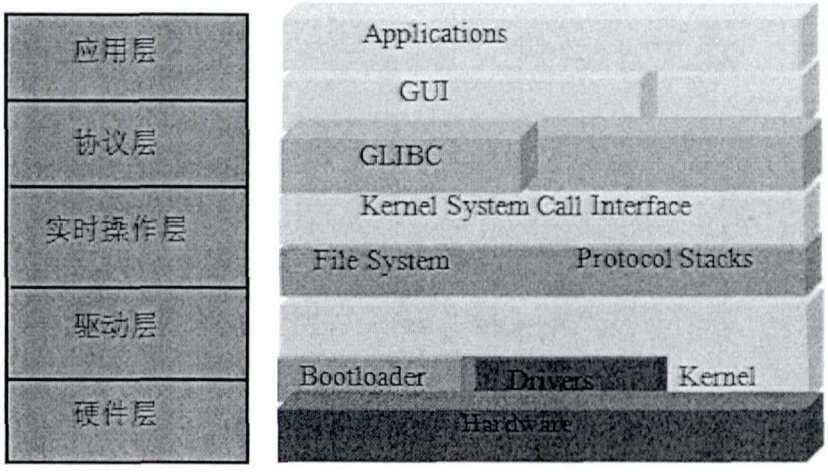

图 2.2 嵌入式系统体系架构

由于嵌入式系统是面向不同行业、不同应用的特定计算机系统，所以，它除了具有计算机系统的一些共性之外，还具有自身独有的特点[41]。

1. 专用系统，应用广泛

任何一个嵌入式系统都和特定应用相关，用途固定，这一点与普通 PC 不同。不同领域的嵌入式系统之间在软、硬件结构上可能会差别很大。嵌入式系统的应用领域广泛，在制造工业、通信、仪表、仪器、汽车、船舶、航空、航天、军事装备、消费类产品等方面均能看到嵌入式系统的身影。全球在用的嵌入式系统数量远远大于通用计算机，目前嵌入式系统每年的芯片使用量要占到芯片市场的 80%以上。

2. 软、硬件可裁剪

嵌入式系统在开发之初就要根据其所要应用的领域，对其体积、功耗、配置、可靠性等方面进行周密的设计，这一设计过程不仅牵涉到硬件也会牵涉到软件。在设计过程中，可以裁剪不需要的接口、存储器件或是软件模块、通信协议，只保留和应用相关的部分。

3. 实时性要求高

因为嵌入式系统被广泛地应用于军工等工业领域，所以有不少嵌入式系统要求对任务的响应具有实时性，其使用的操作系统一般是实时操作系统（RTOS）。根据响应速度的不同，实时操作系统还可以分为硬实时操作系统和软实时操作系统。硬实时操作系统的任务响应速度较快，可以达到 ms 以上级别，如汽车的防抱死刹车系统；软实时操作系统的任务响应速度可以慢一些，一般在数秒以内，如智能手机的应用程序、电视机机顶盒等。

4. 程序固化、具有较长的生命周期

嵌入式系统的目标代码通常是固化在非易失性存储器件（如 ROM、Flash 等）中。嵌入式系统开机后，引导程序会检测系统的硬件情况，并进行运行环境的配置，之后再调入操作系统，所以一般引导程序都会被固化在 ROM 或 Flash 中。为了提高执行速度

和系统可靠性，大多数嵌入式系统常把所有代码固化并存放在存储器芯片或处理器的内部存储器中，而不使用外部的磁盘存储介质。由于嵌入式系统是软、硬件结合的产品，因此，它的升级换代往往是伴随着硬件产品的升级一起进行的。所以，一旦某个型号产品进入市场就具有比较长的生命周期。

5. 具有特定的开发和调试方法

嵌入式系统的程序开发和普通 PC 上的程序开发有很大不同。由于嵌入式系统不具备自主开发能力，开发者难以直接在嵌入式平台上编写、调试程序，必须使用一套开发环境来完成此类工作。一般借助于 PC 平台，先在 PC 上编写程序并模拟运行，再下载到目标板上运行测试。现在的嵌入式目标板一般都具有专用的调试接口，用于和 PC 进行交叉调试。

2.2 系统核心模块框架

2.2.1 流媒体播放系统整体框架

一个完整的最基本的流媒体播放系统，其核心部分就是完成从服务器传输实时的音/视频数据到客户端。

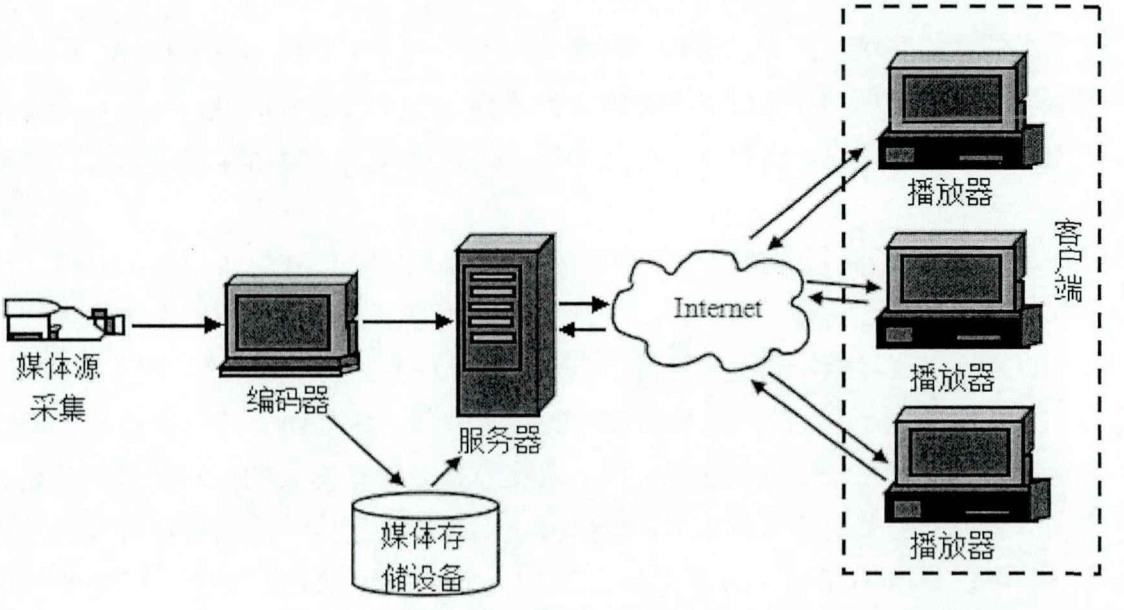

图 2.3 流媒体系统组成

因此，如图 2.3 所示，必须包括媒体源模块（它又包括媒体源采集和编码器(Encoder)）、流媒体服务器(Server)、传输通道(Transport Channel)和客户端播放器(Player)四个模块。模块之间通过特定的协议互相通信，并按照特定格式互相交换文件数据。其中编码器用来将原始的音/视频转换成适合于在网络中传输的流格式文件；媒体存储设备用于存储编码压缩后的数据流，这适合于点播的情况，若是直播，则不需要存储设备而

是将原始数据由直播服务器实时编码并通过网络发送给客户端；网络作为服务器向客户端发送编码后的媒体流的传输信道；客户端播放器则负责解码和播放接收到的媒体数据。

在 2.1.1 节关于流媒体播放方式的介绍中，文章提到相对于实时流式传输，顺序流式传输以点播的方式可以更好地保证视频的传输质量，因此本文提出的嵌入式流媒体播放系统采用顺序流式传输，点播的播放方式，其中媒体源模块、服务器及客户端均运行在 LINUX 操作系统上，其总体设计方案包括以下几个部分：

(1) 流媒体媒体源模块的构建，负责流化音频、视频文件并将流化后的媒体数据流推送给流媒体服务器；

(2) 流媒体服务器的搭建，负责接收媒体源推送的数据流并将数据流传送给客户端，即流媒体终端播放器；

(3) 嵌入式流媒体终端播放器的硬件平台选型及在嵌入式操作系统上的软件设计，负责接收服务器发送的视频数据流并解码播放；

(4) 流媒体网络传输速度控制的实现，包括控制方法的分析及其在流媒体播放系统中的软件实现。

2.2.2 流媒体服务器框架

流媒体播放系统在应用的过程中需要完成会话建立、管理和媒体传输，因此流媒体服务器可分为会话层和传输层。流媒体服务器作为系统的核心部分，它需要具备高性能的处理能力，才能完成流式媒体的连接管理、优先级调度、会话管理以及进行音视频数据的传送等工作。

在 2.1.2 节关于流媒体传输协议的介绍中，文章提到 TCP 需要较多的开销，很长时间其都被认为不适合传输实时数据，因此一般采用基于 UDP 的 RTP 来传输数据，并依靠 RTCP 来提供流量控制和拥塞控制。但是，媒体流的传送并不是就一定不能采用 TCP。用 TCP 传送媒体流最大的问题是媒体流的时延波动，这是由于其拥塞控制、反馈重发和滑动窗口的流量控制机制引起的。TCP 确实不适合对话方式的实时媒体流传送，但是在流媒体服务器的单向传送方式下，如果接收端缓冲区很大，播放时延也可以足够大的情况下是可以使用 TCP 协议传送媒体流的[42]。虽然 TCP 流的波动大，但是若网络的平均带宽大，大到接收端缓冲区的媒体数据包积累得非常多，大的接收缓冲区就能够平滑掉 TCP 流的波动。由于 TCP 是按顺序到达的，并且有反馈重发的差错控制，TCP 流的数据质量要比 UDP 高，因此，有些流媒体服务器也可以采用 TCP。

在 2.1.4 节关于流媒体服务器的介绍中，文章提到现在主流的流媒体服务器采用的协议 RTSP、RTMP 和 HLS。由于 RTSP 点播性能不好、容易受防火墙制约、需要专门的播放器支持、且对服务器压力大，同时 RTMP 嵌入到自己的程序中时又比较麻烦，所

以，本文中的流媒体服务器主推新的技术：HLS。现在主流的视频服务网站（优酷、土豆、爱奇艺）和新闻门户网站（新浪、网易、凤凰）提供视频服务都是基于 HLS 的，而 HLS 是基于 TCP 传输的。

本文中的流媒体服务器是连接媒体源模块和嵌入式流媒体终端播放器的桥梁，负责从媒体源模块接收其推送的多媒体流式数据，并将该数据传送给终端播放器供其解码播放，其中传输数据的速度控制也主要由其来实现。系统的流媒体服务器使用的协议可以看成是 HLS 协议的简化版本，本质上也是使用 HTTP 协议和 TCP 协议，但是没有使用 m3u(8)的索引文件，TS 媒体分片文件和 key 加密串文件。流媒体服务器的整体设计框架如图 2.4 所示。

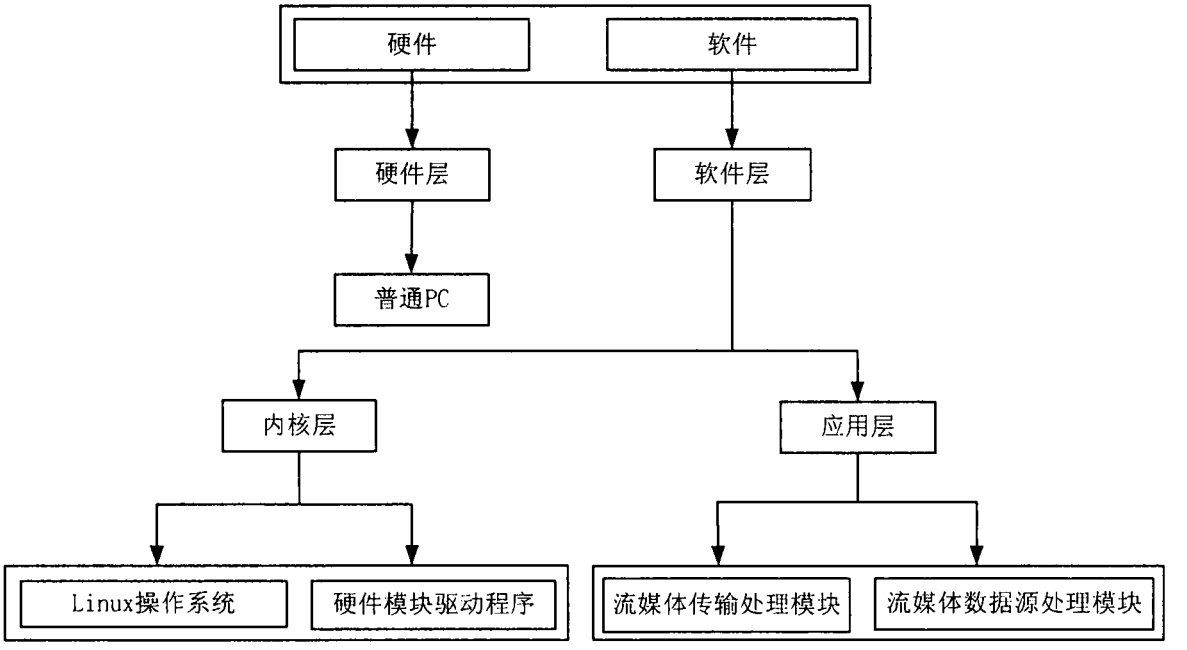

图 2.4 流媒体服务器整体设计框架

其中，流媒体传输处理模块的协议栈如图 2.5 所示，主要分为网络层、传输层及应用层。而流媒体数据源处理模块的功能主要由媒体源模块完成，这里它只负责接收媒体源模块推送的媒体流。

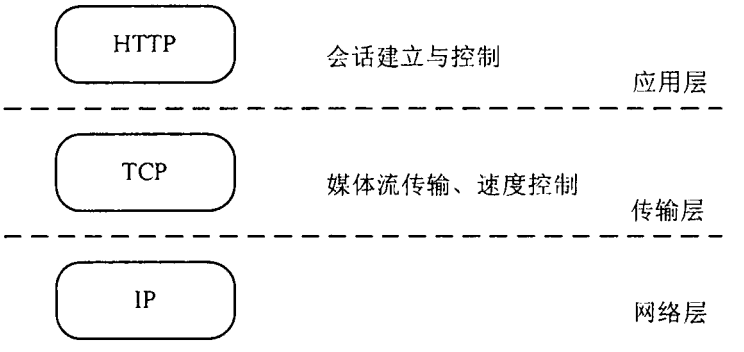

图 2.5 基于 HLS 协议的流媒体传输协议栈

2.2.3 嵌入式流媒体终端播放器框架

本课题实现的嵌入式流媒体终端播放器应具有以下基本功能：

(1) 能够接收服务器发送过来的流媒体格式文件；

(2) 能够反馈服务质量，以便服务器根据网络状况作出发送速率的调整；

(3) 能够对音/视频文件实时解码和同步播放，等等。

嵌入式流媒体终端播放器是嵌入式技术与流媒体技术相结合的产物，它涉及到嵌入式操作系统、嵌入式微处理器、音视频解码及网络传输等多项技术。鉴于这些分析，本文实现的嵌入式流媒体终端播放器的体系架构分为两个部分：硬件部分和软件部分，其中软件部分包括应用层和内核层，如图 2.6 所示。

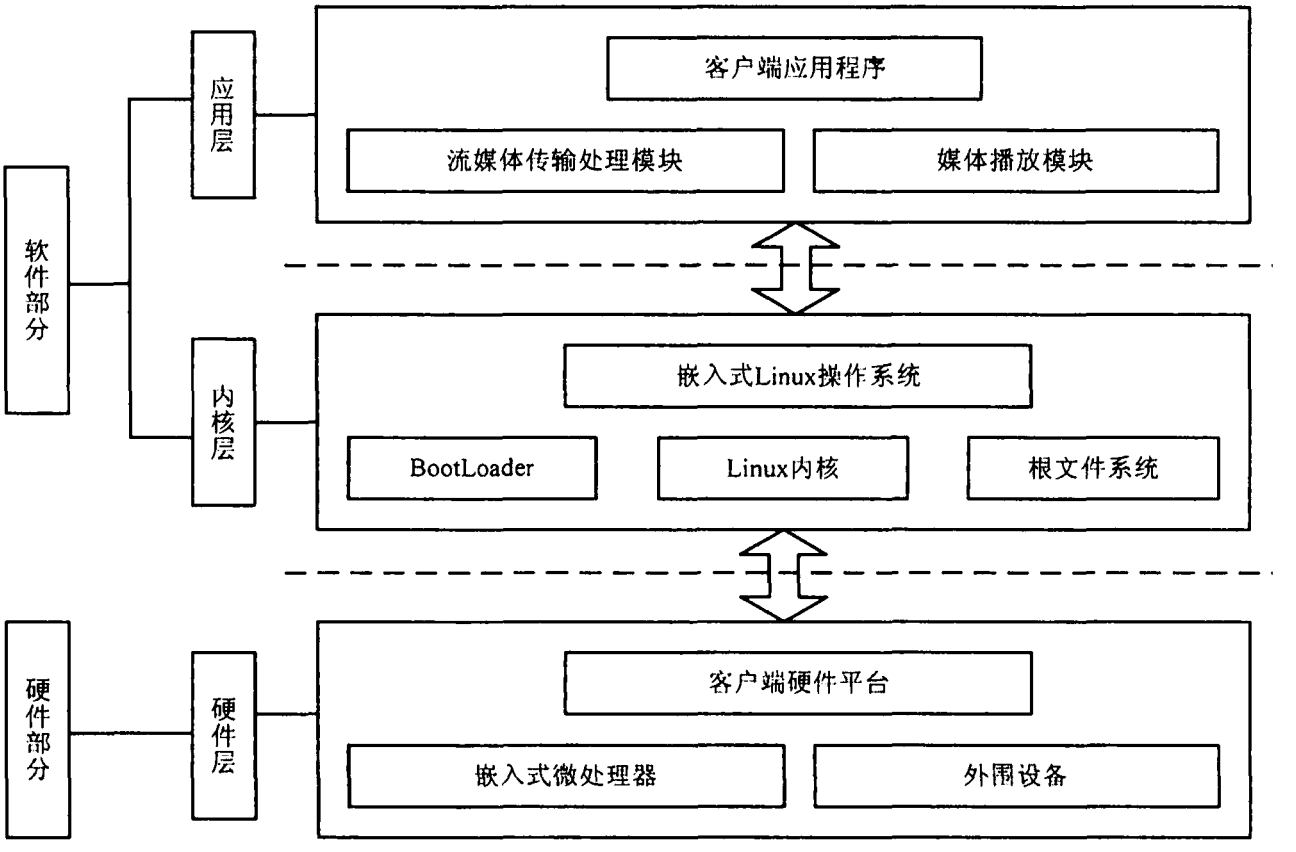

图 2.6 嵌入式流媒体终端播放器体系架构

3 系统硬件设计与实现

3.1 嵌入式终端播放器硬件选型

3.1.1 嵌入式处理器简介

现在几乎所有的嵌入式系统都是基于嵌入式处理器设计的，它是嵌入式系统的硬件核心组成部件。在早期的嵌入式系统中，通常采用个人计算机中的处理器作为嵌入式处理器，这种处理器也称为通用目的处理器。它的通用体现在它作为计算机的处理器提供全部的特性、广泛的功能和大量的应用编程资源，可用于各种场合中。正是由于处理器通用的特性，它具有能源消耗大、产生热量多、体积大、成本高等不足。由于嵌入式系统是针对一定应用对象而设计的，所以通用目的处理器不利于嵌入式系统的发展[43]。

为了满足不同应用对象的要求，嵌入式处理器向不同侧重点发展。例如，有的嵌入式处理器注重尺寸、消耗和价格，这类嵌入式处理器通常功能强、集成度高、能够满足计算密集型、速度快的应用要求，但价格比较昂贵。还有的嵌入式处理器对性能、价格、能耗和尺寸都比较关注。近年来，嵌入式系统的核心器件—处理器已经开始向片上系统发展。

嵌入式处理器按技术特点，分为嵌入式微处理器、嵌入式微控制器、嵌入式数字信号处理器、嵌入式片上系统、嵌入式双核处理器和嵌入式多核处理器6类[44][45]。

(1) 嵌入式微处理器

嵌入式微处理器(Embedded MicroProcessor Unit,EMPU)是嵌入式系统的核心，由通用计算机中的CPU演变而来。由于其ROM、RAM、总线接口、各种外设等都必须安装在一块电路板上，因此又称为单板计算机，如STD-BUS、PC104等。它的特征是具有32位以上的处理器，具有较高的性能，当然其价格也相应较高，具有体积小、重量轻、成本低且可靠性高的特点，但是单板的设计同时又降低了系统的可靠性和技术保密性。与通用计算机处理器不同的是，在实际嵌入式应用中，只保留和嵌入式应用紧密相关的功能硬件，去除冗余功能的部分，这样以最低的功耗和资源实现嵌入式应用的特殊要求。

目前，主要的嵌入式微处理器有AML86/88、PowerPC、MIPS、ARM、Motorola68K等。

(2) 嵌入式微控制器

嵌入式微控制器(Embedded MicroController Unit,EMCU)又称单片微型计算机（Single Chip MicroComputer,SCMC,简称单片机），是指随着大规模集成电路技术的发展，将计算机的CPU、RAM、ROM、定时/计数器和多种I/O接口集成在一片芯片上形

成的芯片级计算机，针对不同场合可做不同组合控制。近几年，单片机的集成度更高，将通用的 USB、CAN 以及以太网等现场总线接口集成于芯片内部。采用单片机的嵌入式系统性能更好，其硬件成本低、系统可靠性高、功耗低、速度快、体积小且抗电磁辐射，因而被广泛地应用在通信、航天和家电等领域。目前，嵌入式微控制器是嵌入式系统工业的主流，其片上外设资源丰富，功耗、成本低、可靠性高，适合于控制。通常情况下，同属一个系列的这种单片化、体积小的单片机具有多种衍生产品，它们的存储器和外设的配置及封装方式各不相同，但是其处理器的内核都是一样的。

目前，嵌入式微控制器的品种和数量最多，比较有代表性的有 MCS-51 系列，P51XA,MCS-51、MCS-96/196/296、MC68HC05/11/12/16 等。

(3) 嵌入式数字信号处理器

嵌入式数字信号处理器(Embedded Digital Signal Processor,EDSP)是一种具有特殊功能的微处理器，其内部采用程序和数据分开的哈佛结构，具有专门的硬件乘法器，广泛采用流水线操作，提供特殊的数字信号处理命令，可以用来快速地实现各种数字信号处理算法。其工作原理是接收模拟信号，将其转换为 0 或 1 的数字信号，然后再对数字信号进行修改、删除、强化，并在其他芯片中把数字数据解译回模拟数据或实际环境格式。它不仅具有可编程性，而且实际运行速度可达每秒数以千万条复杂指令程序，远远超过同档次的通用微处理器。它的强大数据处理能力和高运行速度，是最值得称道的两大特色。

嵌入式数字信号处理器中比较有代表性的产品有 TI 公司的 TMS320 系列和 Motorola 公司的 DSP56000 系列。

(4) 嵌入式片上系统

嵌入式片上系统(Embedded System on Chip，ESoC)是一种追求产品系统最大包容的集成器件，是目前嵌入式应用领域的热门话题之一。嵌入式片上系统实质上就是在一个硅片上实现一个系统。具体来说，各种通用处理器的内核，具有知识产权的标准部件、标准外设作为嵌入式片上系统设计公司的标准器件，这些标准器件通常存在于器件库中且以标准的 VHDL 等硬件语言描述。用户在将设计图交给半导体工厂制作样品之前，只需定义出其整个应用系统并仿真通过即可。嵌入式片上系统的最大特点是成功地实现了软件和硬件的无缝结合，直接在处理器的片内嵌入了操作系统。

由于嵌入式片上系统绝大部分系统构件都是在系统内部，因此整个系统特别简洁，不仅减小了系统的体积和功耗，而且提高了系统的可靠性和设计生产效率。

(5) 嵌入式双核处理器和嵌入式多核处理器

双核(Dual Core)处理器就是基于单个半导体的一个处理芯片上拥有两个一样功能的处理器核心，即将两个物理处理器整合到一个内核中，通过协同运算来提升性能。这样做的优势在于，它克服了传统处理器通过提升工作效率来提升处理器性能而导致耗电

量和发热量越来越大的缺点。另外，采用双核架构可以全面增加处理器的功能，两个处理器核心在共享芯片组存储界面的同时，可以独立地完成各自的工作，从而能在平衡功耗的基础上大幅度提高 CPU 性能。

3.1.2 硬件选型依据

嵌入式流媒体终端播放器的选型依据如下：

(1) 终端播放器最重要的任务是提供音/视频文件的解码并播放，因此解码的运算量大，对处理器的要求也较高，一般情况下处理器的主频要达到 200MHz 以上才可以随音/视频数据进行流畅的处理；

(2) 处理器内存要可以运行起来嵌入式 LINUX 操作系统，能够满足软件设计的需求；

(3) 丰富的外围设备，以方便扩展终端功能。

在 3.1.1 节中对嵌入式处理器做了简单介绍，结合选型依据，那么本文实现的嵌入式流媒体终端播放器在硬件层上包括嵌入式处理器和外围设备，其需要实现以下功能：运行 LINUX 操作系统、存储嵌入式软件系统、提供系统内存、提供以太网的功能、实现流媒体数据的网络传输和显示。因此除了嵌入式处理器外，外围设备应包括存储设备、以太网、串口、LCD 显示屏等模块。本文实现的嵌入式流媒体终端播放器的硬件采用飞凌公司的 FL6410 开发板，如图 3.1 所示。

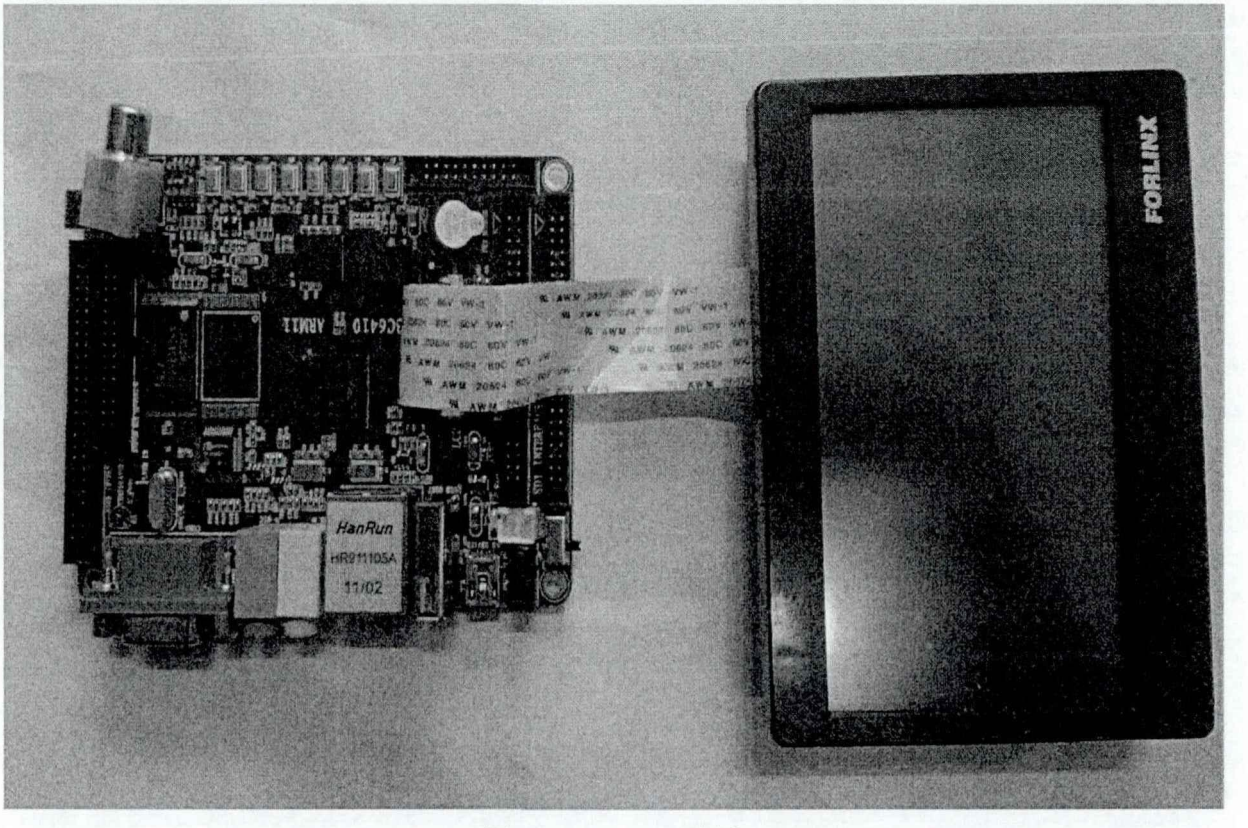

图 3.1 FL6410 开发板

3.2 FL6410 开发板核心模块简介

FL6410 开发板基于三星公司最新的 ARM11 处理器 S3C6410，严格按照 CE、CCC 等国内外电子产品的认证标准设计，充分考虑电磁环境严酷的工作条件。它采用单板结构，尺寸规格为‘8.1CM×10.5CM’，最大限度地保留并集成了多种高端接口给用户（如复合视频信号、摄像头、USB、SD 卡、液晶屏、以太网等），并配备温度传感器和红外接收头，可以方便地进行各种形式的扩展应用或者二次开发。FL6410 支持 Mobile DDR 和多种 NAND Flash，可在 667MHz 主频以上稳定运行，而且其视频处理能力和内部资源也非常强大。

FL6410 开发板结构复杂，具有很多特性，由于文章篇幅的限制，在这里主要介绍在此平台上开发嵌入式流媒体终端播放器所涉及到的主要硬件资源的一些基本特性。

3.2.1 ARM11 及其内核

ARM11 处理器与 ARM9 处理器相比，在结构上有了很大的提高。首先，二者的流水线不同，ARM11 采用 8 级流水线，与 ARM9 的 5 级不同，前者是独立的 load-store 和算术流水，在相同工艺下，采用前者时处理器的性能要比后者提高差不多 40%。其次，ARM11 增加了向量浮点单元，可以快速地进行浮点运算。再者，ARM11 在视频处理方面的性能得到了改善，它执行通过特殊设计的 ARMv6 指令，该指令含有单指令流和多数据流（SIMD）扩展，专门针对媒体处理。另外，ARM11 微架构确保系统性能可以从基本的 350-500MHz 扩展到最终的 1GHz 以上，这种高效率的表现使得无线及便携式应用获得了史无前例的高超性能，同时它还在功耗和性能之间找到了最佳平衡点，并将成本最大化缩减。例如，有一个使用 0.13um 工艺来实现的基于 ARM11 微架构的处理器，如果工作在 1.2V 电压下的话，其功耗将小于 0.4mW/MHz。

ARM11 可以说是一款“准 64 位”微处理器，这是因为它只在高速缓存、协处理器和处理器整数单位之间实现了 64 位数据总线。之所以采用这种方案是因为对于绝大部分的嵌入式应用来说，以增大功耗和面积为代价而在系统中选用一个真正的 64 位处理器是没有必要的，而这种明智的折中方案代价既小，又能满足应用需求。ARM11 处理器利用这些 64 位数据通道可以在一个时钟周期内执行多条数据读写指令或是同时读取两条指令，这种特点特别有利于那些允许数据处理和搬移并行处理的代码序列的执行，可以达到很高的性能[46]。

3.2.2 S3C6410 处理器

S3C6410 处理器由 Samsung 公司推出，是一款基于 ARM11 内核的 16/32 位 RSIC 处理器，采用由 AXI、AHB、APB 总线组成的 64/32 位内部总线架构，且它的存储器系统具有双重外部存储器端口，DRAM 和 FLASH/ROM/DRAM 端口。其内部集成了一个

MFC(Multi-Format video Codec)，内置了强大的硬件加速器和最先进的 3D 加速器，同时还有一个连线到外部存储器的优化的接口，这些优化硬件配置使得 S3C6410 支持 2.5G/3G 通信服务、支持 H.263/H.264/MPEG4 编解码、支持 D3DM API 和 OpenGL ES 1.1/2.0。另外，S3C6410 还有丰富的硬件外设来达到减少系统总成本和提高整体性能的要求，这些外设有系统管理器（电源管理等）、相机接口、TFT 24 位真彩色液晶显示控制器、32 通道 DMA、4 通道 UART 及定时器、通用 I/O 端口、IIS 及 IIC 总线接口、SD 主设备和 USB 主设备、高速多媒体卡接口、用于产生时钟的 PLL。在高端通信网络服务中，S3C6410 能满足数据带宽的要求[46]，另外它还能提供实时视频会议和 NSRC、PAL 制式的 TV 输出，也可以实现 4M triangles/s 的 3D 加速。可见，S3C6410 处理器的解决方案性能表现非常突出，再加上它具有低成本、低功耗和高性能的优点，正被广泛应用于移动电话和通信处理等领域，众所周知的 Apple 公司手机 iphone 就是基于 S3C6410 处理器的[47]。

S3C6410 的结构框图如图 3.2 所示。

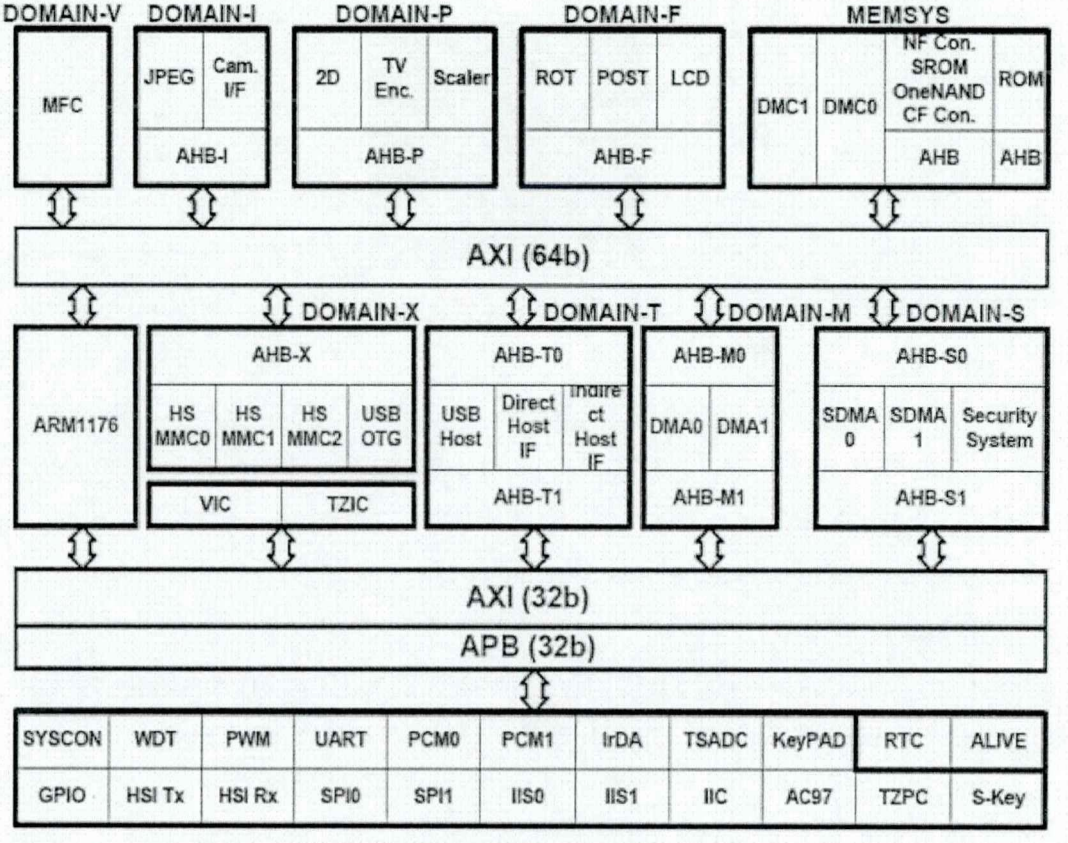

图 3.2 S3C6410 结构框图

3.2.3 FL6410 主要硬件设计说明

1. NAND FLASH

NAND FLASH 存放的主要是内核代码、文件系统、应用程序和数据资料，FL6410

开发板配置的 NAND FLASH 型号为 K9F2G08U0B，大小为 256M Bytes（另有 MLC 结构 2G Bytes NAND FLASH 的 K9GAG08U0D 供用户选择），其中片选信号使用 CSn2[48]。。如图 3.3 所示，为 K9F2G08U0B 芯片。

图 3.3 K9F2G08U0B 芯片

2. DDR 存储器

FL6410 开发板配置了 128M Bytes 的 Mobile DDR 存储器，DDR 数据传输总线频率可达到 266MHz，存储器使用了两片 Samsung K4X51163PC 芯片。为了确保 FL6410 开发板系统运行的可靠性，该芯片在 PCB 板布线时充分考虑串扰、反射及信号等长等电磁兼容设计规则，并采用小尺寸 BGA 封装。如图 3.4 所示，为 K4X51163PC 芯片。

图 3.4 K4X51163PC 芯片

3. UART 接口

FL6410 开发板设计了包含 3 个三线 TTL 电平串口（20pin 2.0mm 间距插头座）和 1 个五线 RS-232 电平串口（DB9 母座）在内的 4 路串口，另外，该产品还开发了配套串口接线板，专门用来满足某些用户的特殊要求[48]。该串口接线板默认 UART0 为调试串口，需要查看系统调试信息时直接将其与 PC 机相连即可。

4. USB HOST 接口

FL6410 开发板的 USB HOST 接口采用卧插式的 USB 母口（A 型插座），支持 USB1.1 协议，可以连接 USB 移动硬盘、USB 键盘、U 盘、USB 鼠标等设备。

5. USB OTG 接口

FL6410 开发板的 USB OTG 接口采用 Mini USB A/B 型号接口（U9），支持 USB2.0 协议，最高运行速度可达到 480Mbps，可以使用它在系统开发过程中下载程序。

6. SD CARD0 卡座

FL6410 开发板集成了一个使用四线 SD 卡接口的 SD 卡座（CON2），它支持 SDIO 规格 1.0 协议和 SD Memory 规格 2.0 协议。在进行软件升级和批量生产时，可以把该接

口作为系统启动设备使用。另外，当该 SD 卡接口被作为 SD Memory 使用时，它能够支持 64G 的 SD 存储卡。如图 3.5 所示，为 SD 卡座。

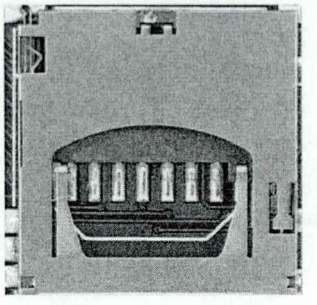

图 3.5 SD 卡座

7. LCD 液晶屏及触摸屏接口

FL6410 开发板能够支持 3.5/4.3/5.6/7/8/10.4 寸等的 TFT 液晶屏。

8. 音频接口

FL6410 开发板有集成音频输出、Mic 输入以及 Line in 输入功能，通过使用 S3C6410 的 AC97 总线将其音频接口外接到 WM9714 音频芯片上来实现。其中，Mic 输入和 Line in 输入以及集成音频输出均采用标准音频插座。

9. 100M 网口

FL6410 开发板上通过 DM9000AE 芯片扩展集成了一个 100M 的以太网接口，该网口在使用时有两种方法，一种是用直连网线连接到路由器或者交换机上，一种是用交叉网线直接连接到 PC 上。它可应用于 Linux 和 WinCE 的系统开发过程中，分别用其来挂载 NFS 网络文件系统和下载 WinCE 镜像。如图 3.6 所示，为 DM9000AE 芯片。

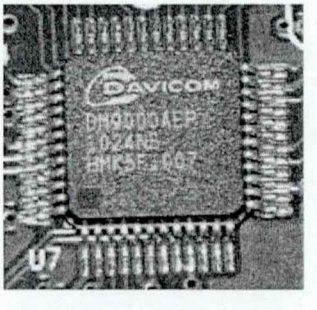

图 3.6 DM9000AE 芯片

10. 复位按键

FL6410 开发板选择的复位芯片是专业的 MAX811t，采用轻触开关的设计方法，能够确保系统在运行时稳定可靠。如图 3.7 所示，为 RESET 复位键。

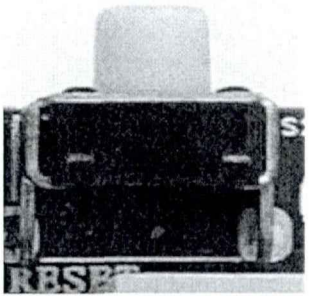

图 3.7 RESET 复位键

4 嵌入式流媒体播放系统平台的构建

4.1 操作系统的选型

4.1.1 嵌入式操作系统简介

嵌入式操作系统（Embedded Operating System,EOS）是嵌入式系统软件的核心部分，嵌入式系统的全部软件和硬件资源的分配调度、操控协作等并发活动都由嵌入式操作系统来负责[43]。对于功能简单、仅包括应用程序的嵌入式系统来说，通常只包含设备驱动程序和应用程序，并不使用操作系统。但是，由于高性能的嵌入式系统的广泛应用，在设计较复杂的程序时，就需要一个操作系统来管理和控制内存、多任务、周边资源等。随着互联网技术的发展、信息家电的广泛应用及 EOS 的微型化和专业化，EOS 开始从单一的弱功能向高专业化的强功能方向发展。EOS 不仅具备了一般操作系统最基本的功能，如任务调度、同步机制、中断处理、文件功能等，还有以下特点。

(1) 可装卸性。

(2) 强实时性。

(3) 统一的接口。

(4) 操作方便、简单、友好的图形 GUI。

(5) 提供强大的网络功能。

(6) 强稳定性，弱交互性。

(7) 操作系统和应用固化在 ROM 中。

(8) 更好的硬件适应性。

在较大规模的系统中，经常需要用到多任务的并行处理能力，所以这些大规模的嵌入式系统通常配有多任务操作系统。这类操作系统与一般常见的操作系统不一样，它必须对事件提前做出实时处理，在处理它所管理工作下的各个事件时，必须在规定的时间内做出响应。

目前，常用的嵌入式操作系统有 Linux、VxWorks、Windows CE、Palm OS、QNX、Symbian OS 等[43][49]。

1. Linux

Linux 是符合 POSIX 标准的、免费的、开源的类 UNIX 操作系统。现代操作系统所具备的内容，Linux 也都拥有。它支持绝大多数的 32/64 位处理器、对称多处理机、多用户和 TCP/IP 通信协议，也具有虚拟内存和内存保护机制，同时还有真正的抢先式多任务处理。

2. VxWorks

VxWorks 是由 Wind River System 公司开发的、基于微内核结构的嵌入式实时操作系统，它支持主流的 32 位处理器，基本构成模块包括：高效的实时微内核 Wind、兼容 POSIX 实时系统标准、I/O 处理系统、本机文件系统、网络处理模块、虚拟内存模块、共享内存模块及板级支持包。除了这些基本模块外，VxWorks 还有许多其它的目标模板，这些模板相对独立、短小精练，由于 VxWorks 允许用户按照自己的需要并通过裁剪模块的方法配置自己的系统，由于所以这些模块可以根据应用需求选择性链接到 VxWorks 的链接器上。

3. Windows CE

Microsoft 公司的嵌入式操作系统 Windows CE 支持多线程、Win32API 子集、多种串行和网络通信技术和 COM/OLE 等进程通信方法，同时 Microsoft 公司同时提供了方便的 Embedded Visual Studio 开发工具。作为一个非硬实时操作系统，Windows CE 可以按优先级执行多任务，其基本核心需要不小于 200KB 的 ROM。另外，Windows CE 主要用于 PDA、SmartPhone 等个人手持终端上，它的主要模块有：内核模块、内核系统调用接口模块、文件系统模块、图形窗口和事件子系统模块及通信模块。

4. Symbian OS

Symbian OS 是最早由 Psion 公司开发的一个专门应用于手机等移动设备的操作系统，目前由诺基亚、爱立信、松下、三星、索尼爱立信和西门子等手机巨头共同拥护。由于 Symbian OS 使用 Symbian 风格的编程、清楚堆栈理念（这种编程基于事件驱动机制，CPU 在应用程式无处理事件时会关闭），而且节省内存，所以当其与其他技术配合使用时，内存的占有量会降低且内存泄漏量也非常少。这种特点使得 Symbian 可以在便携式装置这种有限的资源条件下，连续运行数据甚至是数年，这是 Symbian 相对其它的操作系统的最大优势。Symbian OS 以 EPOC 为基础，包括多执行序列和内存保护，具有真正的抢先式多任务处理，其架构与很多桌上型的操作系统类似。

5.QNX

QNX 是一种商用的遵从 POSIX 规范的类 UNIX 实时操作系统，目标市场主要是面向嵌入式系统。它可能是最成功的微内核操作系统之一。在 QNX 环境下，所有驱动程序、应用程序、协议栈、和文件系统都在内核外部运行，以确保内存受保护的用户空间的安全，同时 QNX 内建了容错功能。因此，几乎所有组件在出现故障时都能自动重启而不会影响其他组件或内核。

4.1.2 操作系统的选择及 Linux 的特点

从 4.1.1 节中对各嵌入式操作系统的介绍和比较中可以看到，嵌入式 Linux 操作系统源代码开放、网络功能成熟、安全性能高且移植性强，适合应用于嵌入式开发，且 Linux 在实时性上取得了突破性的进展。因此，本文实现的嵌入式流媒体播放系统选择

嵌入式 Linux 操作系统作为软件开发平台。

Linux 是一个功能强大的操作系统，同时它是一个自由软件，是免费的、源代码开放的、可以自由使用的 UNIX 兼容产品，它第一次正式向外公布的时间是 1991 年 10 月 5 日。作为一种 UNIX 类操作系统，Linux 的使用人数是最多的一种并且还在迅速增长，这要归功于 Internet 和全世界各地计算机爱好者的共同努力。

Linux 系统除具有所有 UNIX 系统的功能和特点之外，还具有以下特点[50][51]。

1. 自由软件

Linux 是一款自由软件，受公用许可证 GPL 保护。来自全世界的无数程序员可以免费获得其源码（如通过网络等其他途径），并根据自己的兴趣和灵感对其进行修改、编写工作，这也使得 Linux 系统不断壮大。

2. 良好的兼容性

Linux 完全符合 POSIX 标准，可兼容现在主流的 UNIX 系统。在 UNIX 系统下可以执行的程序，几乎完全可以在 Linux 上运行，为应用系统从 UNIX 系统向 Linux 系统转移提供了可能。

3. 良好的界面

在 Linux 系统中，用户的操作是多样的，既可以通过键盘输入指令的方式操作，也可以通过鼠标操作，这是因为 Linux 既提供了字符界面又提供了图像界面（类似于 Windows 图像界面）。

4. 丰富的网络功能

在 Linux 中，用户可以轻松地实现多种网络工作，如网页浏览、远程登录、文件传输等，同时运行 Linux 系统的计算机还以发挥网络服务器的功能，提供电子邮件、FTP、WWW 等众多服务。Linux 之所以有如此强大的网络功能是因为互联网是就是在 UNIX 的基础上繁荣起来的。

5. 支持多种平台

Linux 可以运行在如具有 x86、PPC、Alpha 和 SPARC 等处理器的多种硬件平台上，同时它还支持多处理器技术，使系统的性能大大提高。除了可以运行在计算机上外，它还可以运行在手机、掌上电脑、机顶盒等嵌入式产品上。

4.2 嵌入式开发环境搭建

4.2.1 嵌入式系统开发流程

在嵌入式系统的开发过程中，首先应该根据设计需求选择合适的嵌入式处理器和外围设备来搭建硬件平台，然后则需要选择相应的嵌入式操作系统，并在此基础上进行应用程序的开发，最后对整个系统进行调试运行。系统开发流程如图 4.1 所示。

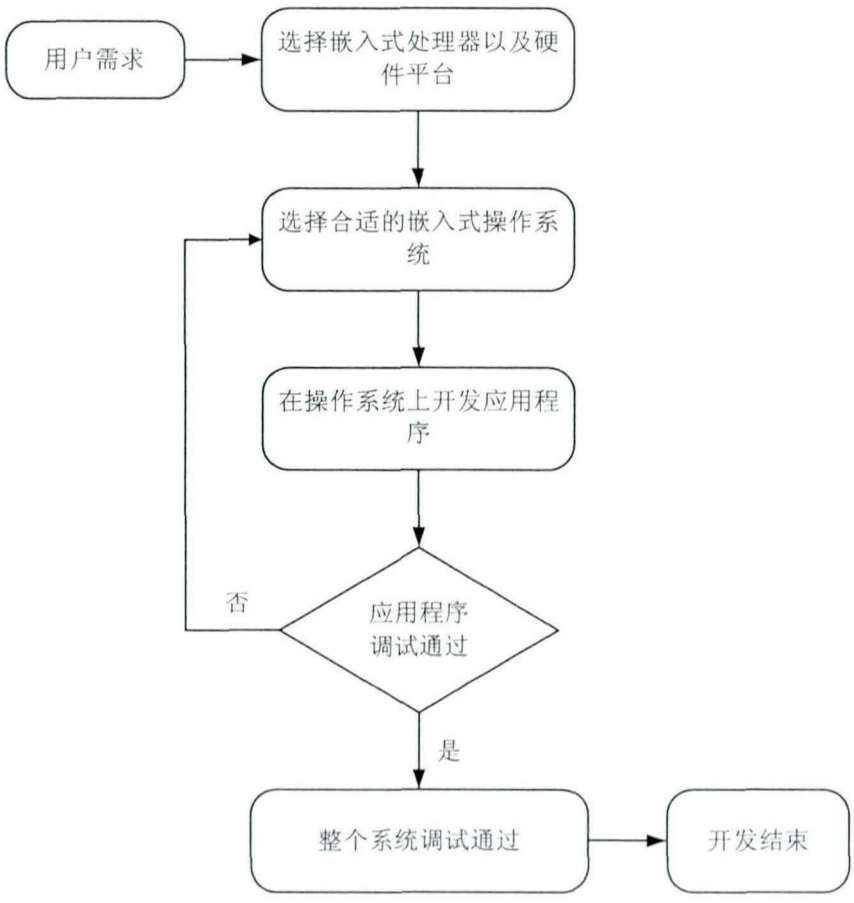

图 4.1 嵌入式系统开发流程图

4.2.2 交叉编译环境

简单地说，交叉编译就是在一个平台上生成在另一个平台上可执行的代码。体系结构(Architecture)和操作系统(Operating System)是平台的两个概念，同一个体系结构上可以运行不同的操作系统，同样，同一个操作系统也可以运行在不同的体系结构上。在嵌入式系统的开发过程中，处处体现着交叉和远程的设计思想，主要是因为嵌入式系统是一个资源有限的系统，如果直接在其硬件平台上编写软件非常困难甚至是不可能的，因此目前一般采用宿主机/开发板开发模式，以嵌入式 LINUX 系统开发为例，如图 4.2 所示是其开发模式。

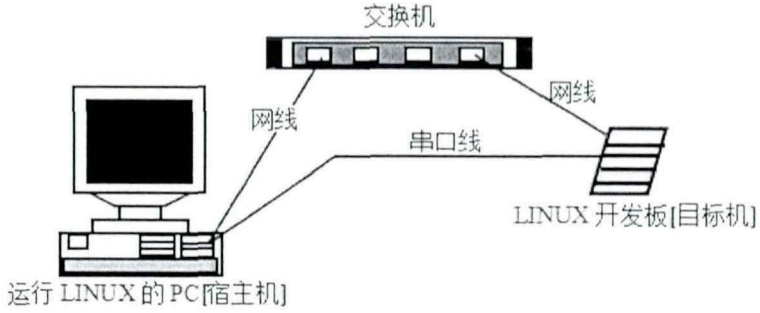

图 4.2 宿主机/开发板开发模式

这种开发模式即是先在通用 PC（宿主机）上编写和编译软件，这时的程序对应于 x86 体系架构，然后用跨平台的交叉编译工具编译生成可运行于目标板（如 ARM 体系架构开发板）的代码格式，最后将这种代码格式下载到目标板运行。通过这种方法，可以充分地利用宿主机上丰富的软硬件资源、良好的开发环境和调试工具来开发目标板上软件。交叉开发环境一般由运行在宿主机上的交叉开发软件和宿主机到目标开发板的调试通道组成，其中，调试通道有两种方法，一种是通过串口将交叉编译后的代码烧写到目标板上，一种是通过网络文件系统挂载的方法，本文中采用第二种方法，具体设置方法在下一节中讲述。

4.2.3 宿主机开发环境构建

1. 宿主机操作系统及安装

本文中宿主机是通过在 Windows7 操作系统的 PC 上运行一个虚拟 Linux 机实现的，这样做的好处是可以在 Windows 和 Linux 之间快速切换并可以在两者之间共享文件，文件共享的方法有 VMware-Tools 和 Xftp，本文中采用第一种方法。虚拟 Linux 机的创建过程如下：

(1) 在 Windows7 中安装虚拟机 VMware Workstation，本文用的版本是 7.1.3；

(2) 在虚拟机中安装 Linux 系统软件，本文安装的是 Ubuntu-9.10；

(3) 在虚拟 Linux 机 Ubuntu 系统中安装 VMware-Tools。

整个安装过程比较简单，受文章篇幅的限制，具体安装步骤不在这里赘述。

2. 交叉编译器的安装

交叉编译工具的选择有很多种的方法，但是由于各个软件版本的不同，不同的版本之间会存在一个是否能相互兼容的问题。可以用 crosstool 自己来编译定制交叉编译工具，但这种方法比较费时间与精力，也可以在网上直接下载使用很多已经编译好的交叉工具链。本文中采用第二种方法，交叉编译工具的版本为 4.2.2。

(1) 将飞凌公司提供的交叉编译工具包 cross-4.2.2-eabi.tar.bz2 拷贝到 Ubuntu 下（本文中使用的目录是/usr/local/arm）并解压；

```
#cd /usr/local/arm
#tar jxvf cross-4.2.2-eabi.tar.bz2
```

解压完成后会在/usr/local/arm 目录下生成 4.2.2-eabi 的目录。

(2) 设置环境变量，将 arm-linux-工具添加到 Ubuntu 中去。（需要注意的是以下操作均在 root 权限执行）

打开/etc/bash.bashrc 文件：

```
#gedit /etc/bash.bashrc
```

在打开的文件中添加以下这一句话：

export PATH=$PATH:/usr/local/arm/4.2.2-eabi/usr/bin

保存文件后执行命令：

#source /etc/profile

#exit

#echo $PATH

得到下面这样的显示信息就说明已经成功安装了交叉编译工具链：

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/arm/4.2.2-eabi/usr/bin
```

还可以通过执行下面的命令来检查交叉编译工具是否安装成功并查看编译器的版本：

#arm-linux-gcc -v

得到下面的显示信息即可：

```
Using built-in specs.
Target: arm-unknown-linux-gnueabi
Configured with: /home/scsuh/workplace/coffee/buildroot-20071011/toolchain_build
_arm/gcc-4.2.2/configure --prefix=/usr --build=i386-pc-linux-gnu --host=i386-pc-
linux-gnu --target=arm-unknown-linux-gnueabi --enable-languages=c,c++ --with-sys
root=/usr/local/arm/4.2.2-eabi/ --with-build-time-tools=/usr/local/arm/4.2.2-eab
i//usr/arm-unknown-linux-gnueabi/bin --disable-__cxa_atexit --enable-target-opts
pace --with-gnu-ld --enable-shared --with-gmp=/usr/local/arm/4.2.2-eabi//gmp --w
ith-mpfr=/usr/local/arm/4.2.2-eabi//mpfr --disable-nls --enable-threads --disabl
e-multilib --disable-largefile --with-arch=armv4t --with-float=soft --enable-cxx
-flags=-msoft-float
Thread model: posix
gcc version 4.2.2
```

4.3 开发板 ARM-Linux 系统构建

在 ARM 处理器上运行的 Linux 系统称为 ARM-Linux 系统。在 ARM 处理器上运行的 Linux 与在 PC 上运行的 Linux 使用的是基本相同的内核（包括内存管理、虚拟文件系统、进程调度、进程间通信、网络几个部分）。在 Linux 操作系统中，无论基于 ARM 架构的 Linux 还是基于 x86 架构的 Linux，绝大多数使用 C 语言编写的操作系统的内核是相同的，只有部分与体系架构相关的代码是使用相应的汇编语言编写的。因此，很多桌面 Linux 程序都能很容易地移植到嵌入式的 ARM 系统中。基于嵌入式 Linux 的系统一般来说由三个部分组成：BootLoader、Linux 内核、根文件系统。

4.3.1 编译 U-Boot

BootLoader（引导加载器）是系统引导程序，它是整个系统运行的第一部分。一般情况下，BootLoader 会作为二进制的形式被烧写入系统的启动地址处。当系统启动后，首先会运行 BootLoader，在 BootLoader 的前面含有系统的启动代码，由其来完成系统硬件的初始化，之后就进入 BootLoader 的环境。在运行 BootLoader 的情况下，用户可

以依据它的功能进行相应的操作，通常 BootLoader 都会提供下载、烧写、简单用户界面等功能。BootLoader 最最基本的功能就是加载 Linux 内核并运行。

本文采用 U-Boot(Universal Boot Loader)通用引导加载程序，它是一个功能强大的 BootLoader 开源代码，支持多种操作系统、多种 CPU 体系架构、多种具体的开发板及众多的设备驱动程序，并且具有良好的可配置型和可扩展性。

(1) 将飞凌公司提供的 U-Boot 源码包 uboot1.1.6_FORLINX_6410.tgz 拷贝到虚拟 Linux 机相应的目录下（本文中用的目录是/forlinux），并在该目录下解压；

#tar zxf uboot1.1.6_FORLINX_6410.tgz

解压后在/forlinux 目录下得到 uboot1.1.6 目录。

(2) 进入 uboot1.1.6 目录，对 U-Boot 进行配置，编译使生成开发板可以运行的程序。

#cd /forlinux/uboot1.1.6

#make smdk6410_config（配置 config）

#make clean（删除以前编译的文件）

#make（编译）

编译完成后在/forlinux 目录下生成 uboot.bin 文件，这个文件即是我们需要的 ARM-Linux 系统的引导程序。

注意：在编译 U-Boot 之前非常重要的一步是，进入其 Makefile 文件查看其中的交叉编译环境路径是否与系统安装的一致：

在/forlinux/uboot1.1.6 目录下执行命令：

#gedit Makefile

在打开的文件中找到下面这句：

```
CROSS_COMPILE = /usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-
export   CROSS_COMPILE
```

如果黑框处与系统的交叉编译工具链的路径不一致则要修改一致。

4.3.2 编译内核

嵌入式 Linux 内核是整个嵌入式 Linux 系统的核心，在 BootLoader 完成引导以后，系统将把控制权交给内核接管。Linux 内核执行系统级别的工作，负责内存分配和管理、运行和调度进程、进程间通信以及文件系统和硬件设备的管理。

本文使用的内核版本是 Linux-3.0.1。

(1) 将飞凌公司提供的内核源码包 FORLINX_linux-3.0.1.tar.gz 拷贝到虚拟 Linux 机相应的目录下（本文使用的目录是/forlinux）并解压；

#tar zxvf FORLINX_linux-3.0.1.tar.gz

解压后在/forlinux 目录下生成目录 linux-3.0.1。

(2) 配置内核；

#make menuconfig

在打开的窗口中选择需要用到的开发模块，其中 System Type 选项要设置为 SMDK6410，如图 4.3 所示。

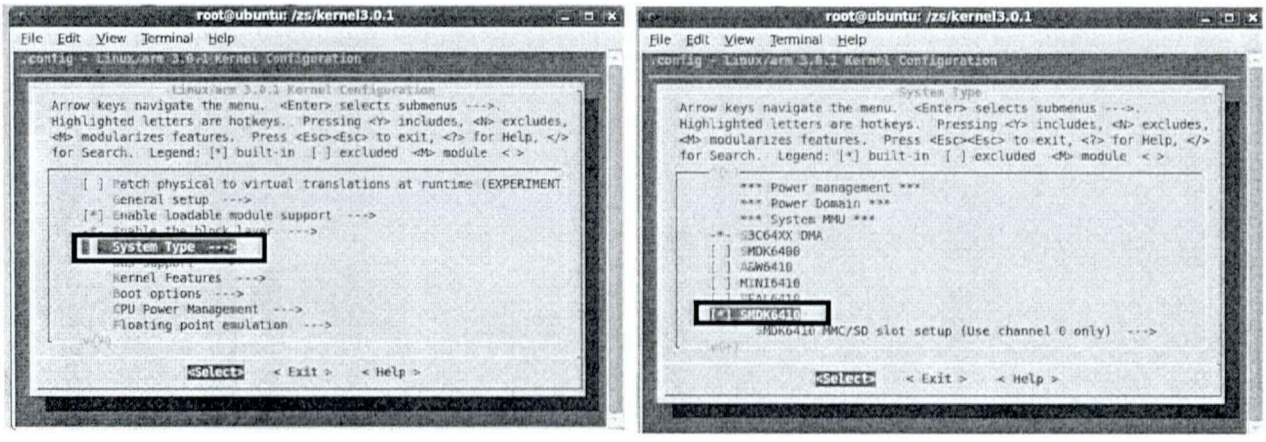

图 4.3 内核配置时 System Type 选项设置

(3) 编译；

首先，进入 linux-3.0.1 目录，修改该目录下 Makefile 文件中，找到 ARCH ：= 和 CROSS_COMPILE ：=两行，修改为如下内容：

```
ARCH            := arm
CROSS_COMPILE   := /usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-
```

其中，ARCH ：=arm 指定目标机为 ARM 架构，

CROSS_COMPILE 指定交叉编译工具的位置。

接着，执行编译操作：

#make zImage

编译结束后会在存放内核源码包目录下的 arch/arm/boot 目录（本文中即 /forlinux/linux-3.0.1/ar-ch/arm/boot）生成 Linux 内核镜像文件 zImage，这就是所需要的 ARM-LINUX 系统的内核文件。

4.3.3 制作文件系统

文件系统（File System）是 UNIX 系统最基本的资源，是文件存放在磁盘等存储设备上的组织方法，主要体现在对文件及目录的组织上。目录提供了管理文件的一个方便且有效的途径，而 Linux 正是使用了这种方式。在 Linux 的嵌入式应用中，对于嵌入式文件系统需要考虑可写入、可更新、掉电可靠性、可压缩和 RAM 启动等特性，目前常用的有 EXT2、CRAMFS、JFFS2、NFTL、NFS 和 RAM 等文件系统。

本文中使用的文件系统是 CRAMFS。CRAMFS 是一个压缩式的文件系统，解压缩以及解压缩之后内存中的数据存放位置都是由 CRAMFS 文件系统本身进行维护的，用

户并不需要了解具体的实现过程，因此这种方式增强了透明度，对开发人员来说，既方便，又节省了存储空间。

根文件系统的制作整体上分为三大步骤：

(1) 准备 mkcramfs 压缩工具；

下载 cramfs-1.1.tar.gz 压缩包拷贝到 Linux 系统相应的目录下，并解压、编译，编译之后会生成 mkcramfs 和 cramfsck 两个工具，其中 cramfsck 工具用来创建 CRAMFS 文件系统的，而 mkcramfs 工具则用来进行 CRAMFS 文件系统的释放以及检查。

(2) 下载 Busybox 压缩包并解压、配置、编译、安装；

配置时要特别注意 Build Options—>选项及 Init Utilities—>选项的设置，编译前要注意制定 Makefile 文件中 ARCH ?=和 CROSS_COMPILE ?= 这两行的修改。

(3) 生成 CRAMFS 根文件系统。

包括创建根目录、准备启动文件、创建节点及生成文件系统镜像，本文中最后生成的 CRAMFS 文件系统镜像的名称为 FORLINUX_6410.cramfs。

整个 CRAMFS 文件系统的创建比较复杂，具体的过程不在本文中详细罗列。

4.4 一键烧写 ARM-Linux 系统

烧写 Linux 操作系统到开发板有两种方法，即一键烧写和 USB 烧写。这里所谓一键烧写，就是借助 SD 卡及飞凌公司提供的程序和系统映像，通过一系列的操作，非常快速地烧写 Linux 到开发板的 NandFlash 中。它们之间的异同点如下：

相同点：

(a) 目的都是将 Linux 烧写在 NandFlash 中。所以，烧写方法任选其一，都可完成 Linux 的烧写。

(b) 基本方法都是借助 SD 卡启动。

(c) 都需要通过串口查看烧写状态。

不同点：

(a) 一键烧写速度快，一次烧写必须烧写所有文件；USB 烧写速度慢，可以更新单个文件。

(b) 一键烧写只需借助 SD 卡，USB 烧写除了需要 SD 卡外还需要借助 USB 线。

本文中采用一键烧写的方法，烧写需要准备的工具及文件如表 4.1 所示。

表 4.1 一键烧写需要的准备的工具及文件

烧写工具及文件	工具及文件来源
SD_Writer.exe(PC 烧写工具)	飞凌公司提供
mmc.bin（sdboot）	
DNW(串口、USB 口工具)	
u-boot.bin(uboot 映像)	上述步骤 1、2、3 中制作
zImage	
FORLINUX_6410.cramfs	

(1) Eboot 擦除 NandFlash

由于飞凌 6410 开发板在出厂时默认烧写的是 WinCE 系统，而 WinCE 系统在微软设计时，把前四个块都标记为坏块，也就是说把 Bootloader 分区都标记为坏块，以防止 Bootloader 被 WinCE 的应用程序擦除。因此，在开发板更换为 Linux 的时候，需要首先将这几个“假坏块”恢复过来。

a) 用串口线连接好开发板的 COM0 与 PC 机的串口，打开并设置 DNW 软件；

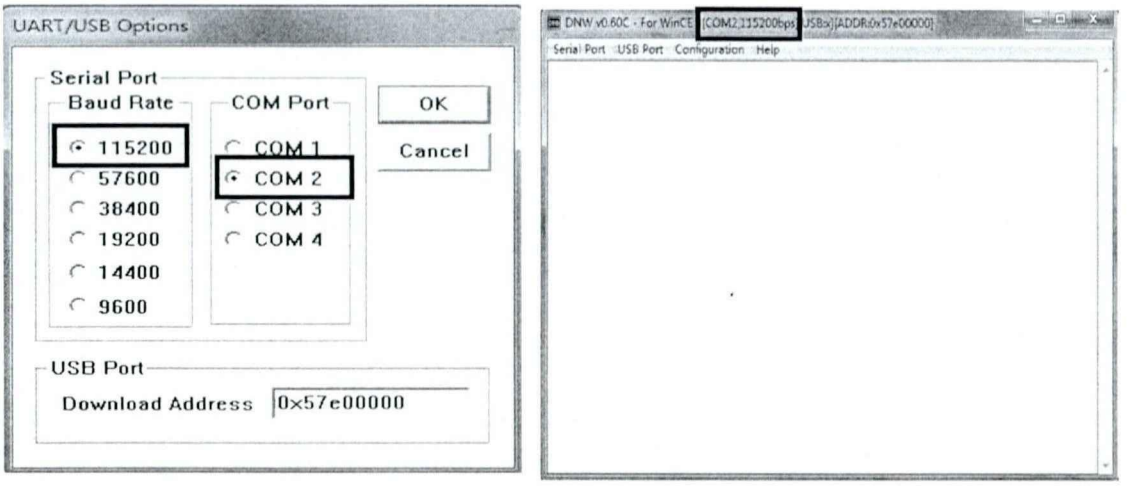

图 4.4 DNW 软件设置

b) 开发板上电，待出现延时 5 秒启动系统时，在 DNW 软件中按下 PC 的空格键使开发板停留在 Eboot 状态；

c) 按下 PC 的‘A’键擦除 NandFlash。

(2) 在宿主机上制作用于一键烧写 Linux 的 SD 卡

a) 将 SD 卡格式化为 FAT32 格式；

b) 通过 SD_Writer.exe 把 mmc.bin 烧写到 SD 卡中，烧写步骤在图 4.5 中按序标出；

c) 将 u-boot.bin 、 zImage 、 FORLINUX_6410.cramfs 拷贝到 SD 卡中，FORLINUX_6410.cramfs 拷贝之后在 SD 卡中将其重命名为 cramfs，正确的文件名如图 4.5 所示。

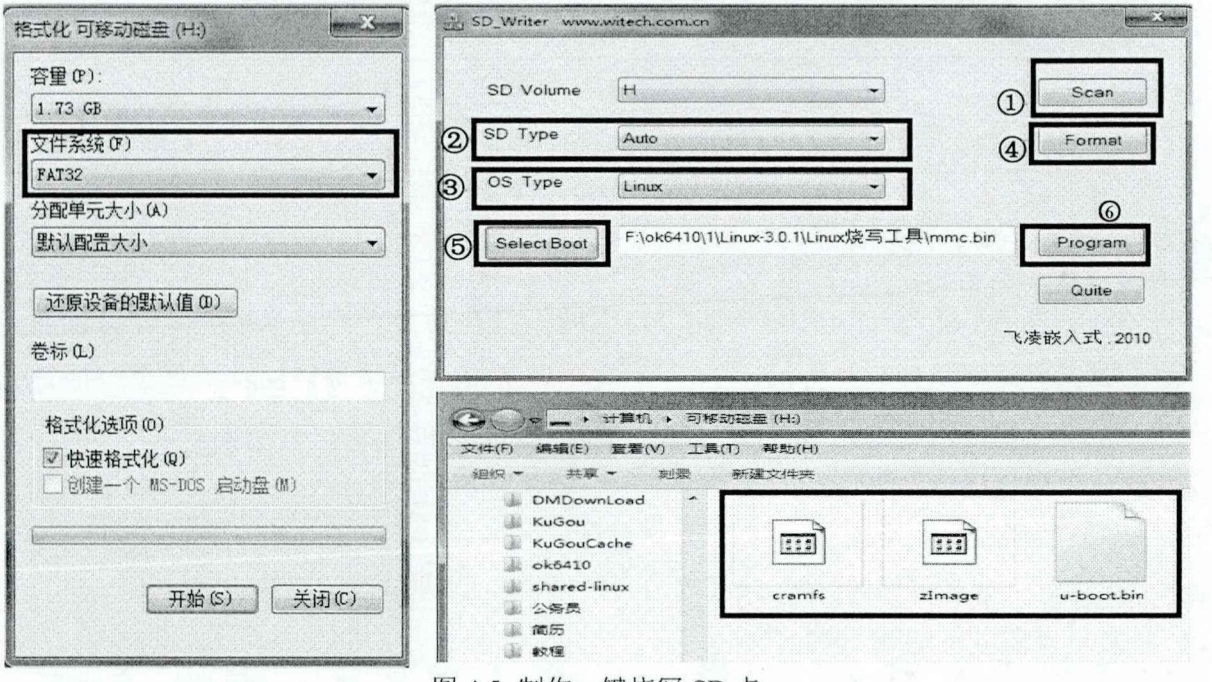

图 4.5 制作一键烧写 SD 卡

(3) 烧写 Linux 到开发板的 NandFlash

a) 将制作好的 SD 卡插在开发板的 SD 卡槽中，开发板设置为 SD 卡启动模式；

b) 打开 DNW 软件并设置好 DNW 的串口；

c) 开发板上电，会在 DNW 窗口显示串口信息，如图 4.6 所示为开始和结束时的串口信息；

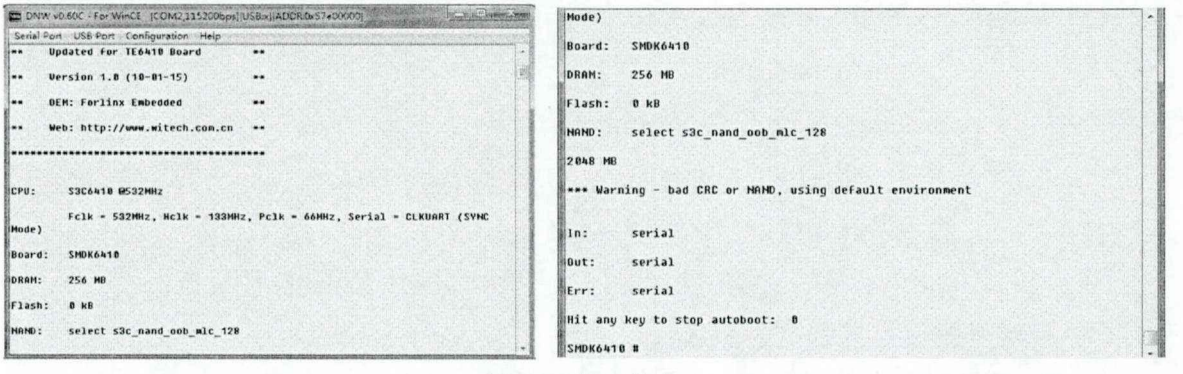

图 4.6 串口信息

d) 开发板断电，将开发板重新设置为 Nand Flash 启动模式，重启电源，Linux 系统就可以正常启动了。

注：开发板启动模式的设置通过拨码开关来实现，对于本文中使用的开发板，其启动模式的设置方法如表 4.2 所示。

表 4.2 开发板启动模式设置

BOOT SELECT 引脚号	Pin 4	Pin 3	Pin 2	Pin 1
SD 卡启动	1	1	1	0
Nand Flash 启动	0	0	1	0

4.5 NFS 调试环境搭建

NFS(Network File System, 网络文件系统)在 Linux 系统之间实现共享，可以通过其把远程主机的目录挂载到本机，使得访问远程主机的目录就像访问本地目录一样方便快捷。通过这种方法，可以直接在开发板上挂载开发主机上的文件系统，减少程序的烧写次数，提高程序开发速度。

(1) 在宿主机上准备 NFS 文件系统目录：

#mkdir /nfs (根据自己的情况自行设定要挂载的目录)

(2) 为宿主机配置 NFS 服务：

#sudo apt-get install portmap

#sudo apt-get install nfs-kernel-server

#sudo gedit /etc/exports

在弹出的文本编辑器中编辑 exports 文件，在最后一行添加：

/nfs *(rw,sync,no_root_squash)

(3) 启动宿主机 NFS 服务并设置宿主机 IP：

#sudo exports -r

#sudo /etc/init.d/portmap restart

#sudo /etc/init.d/nfs-kernel-server restart

#ifconfig eth0 192.168.0.111 up

注：192.168.0.111 是设置的主机的 IP，应根据自己的情况自行设定。

(4) 启动开发板，挂载文件系统：

用网线连接开发板和宿主机，打开 DNW 软件并设置好 DNW 的串口，启动开发板 Linux 系统，在 DNW 窗口执行命令：

#ifconfig eth0 192.168.0.222

#ping 192.168.0.111

注：192.168.0.222 是设置的开发板的 IP，其设置必须要和主机 IP 在同一网段内。

如果网络畅通，则可以挂载 nfs,执行如下命令完成 nfs 挂载：

#mount -t nfs 192.168.0.111:/nfs /mnt -o nonlock

5 系统软件设计与实现

5.1 FFmpeg 库的研究

在嵌入式操作系统中用到很多种类的库和中间件，其中，常见的用作音/视频流方案的库主要有两种：FFmpeg 库和 mencoder 库。FFmpeg 在 Linux 平台下开发，但是其跨平台性很好，同样也可以在其它操作系统环境中编译运行，包括 Windows、Mac OS X、嵌入式 Linux、WinCE 等；而 mencoder 一般情况下只被作为 Mplayer 自带的编码工具使用（Mplayer 是 Linux 下开源的播放器），虽然 mencoder 包含比较全面的编解码器，可以支持几乎所有格式的视频转换，但是其跨平台性较差，只可以用于 Windows 和 Mac 中。而且，从编码速度的角度来看，FFmpeg 比 mencoders 要快。因此，本文设计的嵌入式流媒体播放系统的软件部分基于 FFmpeg 库来实现。

5.1.1 FFmpeg 库的架构

FFmpeg 项目集音视频的采集、编解码、格式转换及播放等功能为一体，支持 MPEG、DivX、MPEG4、AC3、DV、FLV 等 40 多种编码，AVI、MPEG、OGG、Matroska、ASF 等 90 多种解码。FFmpeg 项目的组件如表 5.1 所示。

表 5.1 FFmpeg 项目的组件

组件	该组件说明
ffmpeg	以参数形式进行命令行文件格式转换，例如，将.avi 格式的视频转成.mpg 格式，则执行命令：ffmpeg -i video_origine.avi video_finale.mpg
ffplay	通过 SDL 来回显，利用 FFmpeg 库的简易播放器。
ffprobe	基于 FFmpeg 库的简易流媒体分析器
ffserver	通过 HTTP /RTSP 方式进行多媒体数据流式播放的即时广播串流服务器

在 FFmpeg 库文件的源代码主目录下主要有 libavcodec、libavformat、libavutil 和 libswcale 等子目录，各子目录的功能如表 5.2 所示。

表 5.2 FFmpeg 主目录下子目录功能

子目录	子目录功能
libavcodec	存放各个 encode/decode 模块，用于各种类型声音/图像编解码
libavformat	存放 muxer/demuxer 模块，用于各种音视频封装格式的生成和解析
libavutil	存放内存操作等辅助性模块，包含一些公共的工具函数
libswscale	视频场景比例缩放、色彩映射转换

另外，FFmpeg 库采用的是主程序+核心库的编程模式，由于其对外提供统一的调用方法，而各种具体格式的实现隐藏在核心库中，因此主程序不需要关心如何调用具体格

式相关的函数。下面简要分析 FFmpeg 库的几个重要模块。

1. I\O 模块

FFmpeg 项目的数据 I\O 部分主要是在 libavformat 库中实现，某些对于内存的操作部分在 libavutil 库中。数据 I\O 是基于文件格式（Format）以及文件传输协议 (Protocol) 的，与具体的编解码标准无关。

FFmpeg 工程转码时数据 I\O 层次关系如图 5.1 所示。

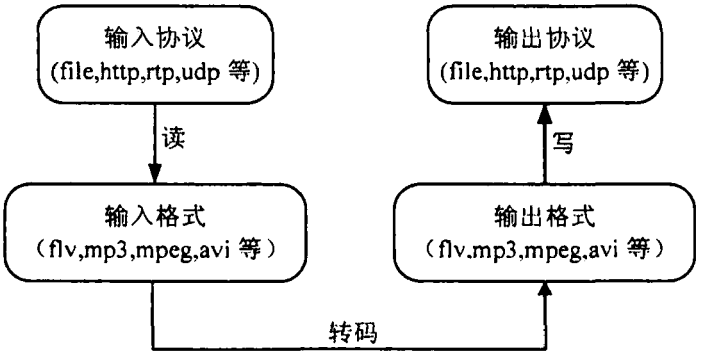

图 5.1 FFmpeg 转码数据 I\O 流程

2. demuxer 和 muxer 模块

Ffmpeg 的 demuxer 和 muxer 接口分别在 AVInputFormat 和 AVOutputFormat 两个结构体中实现，在 av_register_all( )函数中将两个结构分别静态初始化为两个链表，保存在全局变量：first_iformat 和 first_oformat 两个变量中。在 FFmpeg 的文件转换或者打开过程中，首先要做的就是根据传入文件和传出文件的后缀名匹配合适的 demuxer 和 muxer，得到合适的信息后保存在 AVFormatContext 中。

3. decoder/encoder 模块

编解码模块主要包含的数据结构为：AVCodec、AVCodecContext。每一个解码类型都会有自己的 Codec 静态对像，Codec 的 int priv_data_size 记录该解码器上下文的结构大小，如 MsrleContext。这些都是编译时确定的，程序运行时通过 avcodec_register_all() 将所有的解码器注册成一个链表。在 av_open_input_stream( )函数中调用 AVInputFormat 中的 read_header( )函数读文件头信息时，会读出数据流的 CodecID，即确定了他的解码器 Codec。在 main( )函数中除了解析传入参数并初始化 demuxer 与 muxer 的 parse_options( )函数以外，其他的功能都是在 av_encode( )函数里完成的。在 libavcodec\utils.c 中有如下二个函数：AVCodec *avcodec_find_encoder(enum CodecID id) 和 AVCodec*avcodec_fin d_decode r(enum CodecID id)，他们的功能就是根据传入的 CodecID，找到匹配的 encoder 和 decoder。在 av_encode( )函数的开头，首先初始化各个 AVInputStream 和 AVOutputStream，然后分别调用上述两个函数，并将匹配上的 encoder 与 decoder 分别保存在：AVInputStream->AVStream*st->AVCodecContext*codec->struct AVCodec*codec 与 AVOutputStream->AVStream*st->AVCodecContext*codec->struct AVCo

dec*codec 变量中。

5.1.2 FFmpeg 库的裁剪、安装与代码移植

FFmpeg 库在不作任何操作的情况下本身占有 20M 左右的大小，在熟悉了其庞大且结构复杂的源代码的基础上，找出实现流媒体播放系统所需要的代码，对源码库进行裁剪是很有必要的。对 FFmpeg 库进行裁剪的方法主要有两种，一种是对源代码进行修改，即对不需要的目录或者目录下的某些文件进行删减操作，另一种方法则是配置(configure)所需的接口，不需要的不配置。另外，本文设计的嵌入式流媒体播放系统的媒体源模块和服务器模块运行在基于 x86 架构的普通 PC 上，而播放模块运行在基于 ARM 架构的 FL6410 开发板上，因此在实现软件编程时，需要分别生成可以运行于上述两种架构的 FFmpeg 库。由于 FFmpeg 属于自由软件，采用 LGPL 或 GPL 许可证，因而它的移植同样遵循 LGPL 或 GPL 移植方法：configure、make、make insatll。所以，本文中对 FFmpeg 进行裁剪采用配置所需接口的方法，并将裁剪后的 FFmpeg 库移植到基于 ARM-Linux 系统的 FL6410 开发板上。

1. 获取 FFmpeg 源码软件包

本文使用的 FFmpeg 版本为 0.8.7（其他低于这个版本的 FFmpeg 编译生成的 ffplay 视频音频不同步，其下载地址为 http://ffmpeg.org/release/ffmpeg-0.8.7.tar.gz）。将下载后的源码包拷贝到宿主机 Ubuntu 下并解压（本文使用的目录是/forlinux)。在 Ubuntu 下进入终端，执行如下命令：

```
#cd /forlinux
#tar zxvf ffmpeg-0.8.7.tar.gz
```

进入解压后的文件目录，输入如下命令：

```
#cd ffmpeg-0.8.7
#./configure -help
```

可以得到 configure 的基本选项参数，如图 5.2 所示是其中一部分参数截图。

```
Standard options:
  --help                   print this message
  --logfile=FILE           log tests and output to FILE [config.log]
  --disable-logging        do not log configure debug information
  --prefix=PREFIX          install in PREFIX []
  --bindir=DIR             install binaries in DIR [PREFIX/bin]
  --datadir=DIR            install data files in DIR [PREFIX/share/ffmpeg]
  --libdir=DIR             install libs in DIR [PREFIX/lib]
  --shlibdir=DIR           install shared libs in DIR [PREFIX/lib]
  --incdir=DIR             install includes in DIR [PREFIX/include]
  --mandir=DIR             install man page in DIR [PREFIX/share/man]

Configuration options:
  --disable-static         do not build static libraries [no]
  --enable-shared          build shared libraries [no]
  --enable-gpl             allow use of GPL code, the resulting libs
                           and binaries will be under GPL [no]
  --enable-version3        upgrade (L)GPL to version 3 [no]
  --enable-nonfree         allow use of nonfree code, the resulting libs
                           and binaries will be unredistributable [no]
  --disable-doc            do not build documentation
  --disable-ffmpeg         disable ffmpeg build
  --enable-avconv          enable  avconv build
```

图 5.2 configure 的部分基本选项参数

2. FFmpeg 支持库的安装

(1) 安装 x264 库

由于 FFmpeg 不支持 H.264 视频编码器，而这两种编码方式又比较常见，有很大的需求，因此若需要一个以 H.264 方式编码的 AVI 文件视频，则需要在 FFmpeg 编译的过程中加入 x264 库来支持 H.264 编码。

首先，下载 x264 源码：

#git clone git://git.videolan.org/x264.git

然后，配置、编译、安装：

#cd /home/x264

1) 生成运行于 x86 架构的 x264 库

#./configure --enable-shared

#make

#make install

注意：/home/x264 是 x264 库源码的位置。

2) 生成运行于 ARM 架构的 x264 库

配置命令：#CC=arm-linux-gcc./configure --enable-pthread --enable-shared --host=arm-linux- --prefix=/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-

编译与安装：

#make

#make install

(2) 安装 SDL 库（如果不安装，则编译后没有 ffplay）

1) 生成运行于 x86 架构的 SDL 库

在宿主机 Ubuntu 系统中执行如下命令自动安装 SDL 库：

#sudo apt-get install build-essential subversion git-core checkinstall yasm texi2html libfaac-dev libfaad-dev libMP3lame-dev libsdll.2 libtheora-dev libx11-dev libxvidcore4-devzliblg-dev

这里，subversion 和 git-core 是软件管理工具，安装了可以方便下载源码，它们就是对应的 svn 和 git。如果不采用上述源码安装方法，而是单独下载 SDL 源码然后 make install 安装，那么编译 FFmpeg 源码后虽然可以产生 ffplay，但是产生的 ffplay 播放视频时没有声音。

2)生成运行于 ARM 架构的 SDL 库

分别对 1)中安装的各个库重新进行配置并交叉编译，编译生成的库即是可移植到开发板的 SDL 库，方法同上述 x264 库的方法。

3. 裁剪、编译、安装 FFmpeg

对 FFmpeg 的裁剪优化主要是对 ffplay 的裁剪优化，本文设计的需求是能播放测试文件（视频为 mpeg4 编码，音频为 mp2 编码，且为 AVI 复用），根据需求，找到相应的选项，或禁用或启用。

#cd /forlinux

#tar zxvf ffmpeg-0.8.7.tar.gz

#cd ffmpeg-0.8.7

1) 生成运行于 x86 架构的 FFmpeg 库

执行命令: #./configure --enable-gpl --enable-shared --enable-pthreads --enable-x264--enable-memalign-hack --enable-static --disable-yasm --disable-parsers --disable-decoders --disable-encoders --enable-decoder=mpeg4 --disable-muxers --disable-demuxers --enable-demuxers=avi --enable-decoder=mp2 --disable-filters --disable-bsfs

其中，--disable-parsers 为禁用所有解析器；

--disable-decoders 为禁用所有解码器；

--disable-encoders 为禁用所有编码器；

--enable-decoder=mpeg4 为启用 mpeg4 的编码器；

-- disable-muxers 和--disable-demuxers 分别为禁用所有复用及解复用；

--enable-demuxer=avi 和--enable-decoder=mp2 为启用 AV I 复用和 mp2 编码；

--disable-filters 和--disable-bsfs 分别为禁用所有过滤器和码流过滤器。

通过以上配置之后，编译、安装，就生成了我们需要的 ffplay。在以后的配置中如果需要其他的编码器或解码器之类的，按照选项要求进行配置即可。

编译与安装：

#make

#sudo make install

2) 生成运行于 ARM 架构的 FFmpeg 库

首先将上面生成的对应于 ARM 架构 x264 库及 SDL 库中的 include、lib 文件分别拷贝至第三方库库路径/usr/local/ffextra/include 和/usr/local/ffextra/lib，然后在上面的配置选项中加入参数--cross-compile --cc=arm-linux-gcc --arch=arm --cross-prefix=/usr/local/arm//4.2.2-eabi/usr/bin/arm-linux-及--extra-cflags=-I/usr/local/ffextra/include --extra-ldflags=-L/usr/local/ffextra/lib 即可。

编译与安装：

#make

#sudo make install

注意：--extra-cflags：FFmpeg 第三方库头文件声明；--extra-ldflags：FFmpeg 第三方库库文件路径。

4. 宿主机测试

#./ffplay gdg.avi

如果看到视频播放画面，如图 5.3 所示，且听到声音则说明 FFmpeg 库安装成功(gdg.avi 是与 ffplay 组件在相同目录下的本地测试文件)。

图 5.3 FFmpeg 库安装成功时的视频播放画面

5. 开发板测试

首先将生成的第三方库的 include 和 lib 文件夹，以及运行于 ARM 架构的 FFmpeg 库拷贝到宿主机 NFS 文件系统目录下，按照 4.5 节中所述搭建 NFS 调试环境。成功挂载文件系统 nfs 后，在开发板上执行如下命令：

```
#cp -r /mnt/include /include
#cp -r /mnt/lib /lib
#cd /mnt/FFmpeg
#./ffplay hello.mpg
```

如果在开发板同样看到如图 5.3 所示的视频播放画面，并听到声音则说明 FFmpeg 库成功移植到开发板上。

5.2 系统核心模块的软件实现

5.2.1 系统软件整体工作流程

在 5.1 节中已经提到本文中的软件基于 FFmpeg 库来实现，基于其 ffmpeg、ffserver 和 ffplay 三个项目来分别实现系统中媒体源模块、流媒体服务器和流媒体播放器的功能，ffmpeg、ffserver 和 ffplay 三者之间紧密配合来完成流媒体数据的推送、分发、解码，这三者之间的合作流程如图 5.4 所示。

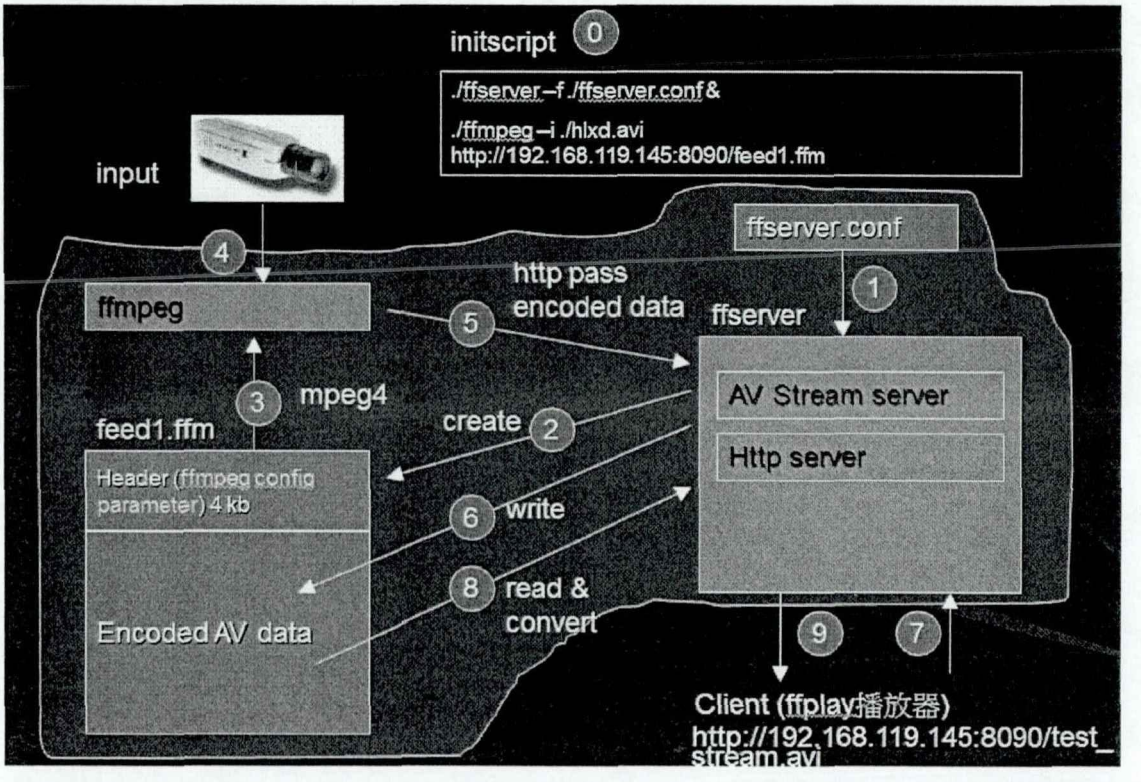

图 5.4 ffmpeg、ffserver、ffplay 合作流程图

图 5.4 中带圆圈的数字表示步骤的编号，编号越小，执行时间越早；箭头表示流程的变迁；箭头旁边的文字表示流程变迁的说明。下面按照步骤的编号说明 ffserver，ffmpeg 和 ffplay 之间的合作流程：

(0) 输入初始化脚本(InitScript)：用管理员身份分别启动流媒体服务器 ffsrever 和推流进程 ffmpeg；

(1) ffsrever 会读取 ffserver 的配置文件 ffserver.conf，在该配置文件中指定了 ffsrever 的初始化参数；

(2) ffserver 启动后，会根据配置文件 ffserver.conf 在 Ubuntu 文件系统目录中创建 feed1.ffm(feed1.ffm 文件采用 FFmpeg 开源项目的自定义文件格式 FFM1，该文件格式的前 4KB 内容为文件头，后面的内容为 ffmpeg 推送到 ffserver 的编码好的数据，ffserver.conf 指定了 feed1.ffm 的最大大小，比如 2M，ffmpeg 通过 HTTP 协议向这个文件循环写入数据，当文件写满后，ffmpeg 会跳过 feed1.ffm 的文件头位置而覆盖旧的数据，所以，feed1.ffm 可以认为是 ffserver 的缓存文件)；

(3) ffmpeg 启动之后会和 ffserver 建立一个短暂的连接，通过这个第一次的连接，ffmpeg 从 ffserver 那里获取它向客户端输出的流的配置，同时把这些配置作为自己的编码输出配置，然后 ffmpeg 主动断开这次连接；

(4) ffmpeg 从音视频输入设备或者录制好的音视频文件中读取音频、视频等信息，并将这些裸流复用(muxer)成一路码流，即将文件转换为流式传输文件。本程序即是流化录制好的文件 hlxd.avi；

(5) ffmpeg 进程使用 HTTP 协议，向 ffserver 发出请求，欲将(4)中的码流写入 ffserver 的 feed1.ffm 缓存文件中；

调用的命令是：

#./ffmpeg –re –i ./hlxd.avi http://192.168.119.145:8090/feed1.ffm

(6) ffserver 接受 ffmpeg 的请求，将 ffmpeg 推送的直播流写入 feed1.ffm；

(7) 客户端 ffplay 通过 http 协议请求和服务器 ffserver 建立连接；

请求的命令是：

#./ffplay http://192.168.119.145:8090/test_stream.avi

test_stream.avi 是在 ffserver.conf 中创建的流节点的名字，它与 feed1.ffm 文件中缓存的数据存在映射关系。

(8) ffserver 响应 ffplay 的请求，并从 feed1.ffm 中读取编码好的数据到 ffserver 的内存缓存中；

(9) ffserver 通过 TCP 协议将(8)中读取的内存缓存中的数据发送个 ffplay，ffplay 对接受到的码流进行解码和渲染，从而完成播放。

注意：

1) 192.168.119.145 是 ffserver 运行的主机 IP，即流媒体服务器的运行 IP 地址，应根据自己的情况设定；

2) 8090 是用于通信的端口号，可在 ffserver.conf 文件中设置；

3) test_stream.avi 是使用视频文件 hlxd.avi 的元数据信息（视频的格式、帧速率、编码信息等）在 ffserver.conf 中创建的流节点名称，用来完成和 feed1.ffm 的映射。

4) 若需要播放其他的视频文件，则可以采用 ffprobe 来查看视频的元数据信息。

a) 执行命令#./ffprobe“视频文件”命令行，则命令行返回视频的元数据信息，包括视频的格式、帧速率、编码信息等。以 gdg.avi 为例，命令行返回的信息如图 5.5 所示。

```
Input #0, avi, from 'gdg.avi':
  Metadata:
    encoder         : Lavf52.20.0
  Duration: 00:30:38.13, start: 0.000000, bitrate: 275 kb/s
    Stream #0:0: Video: mpeg4 (Simple Profile) (FMP4 / 0x34504D46), yuv420p, 320
x240 [SAR 1:1 DAR 4:3], 15 tbr, 15 tbn, 15 tbc
    Stream #0:1: Audio: mp2 (P[0][0][0] / 0x0050), 44100 Hz, stereo, s16, 64 kb/
s
```

图 5.5 ffprobe 查看视频的元数据信息

b) 修改 ffserver.conf, 使用上面的元数据新建一个流节点, 如 test_stream_ibm.avi，并来完成和 feed1.ffm 的映射，如图 5.6 所示，然后保存 ffserver.conf。注意观察图 5.5 与图 5.6 之间的对应关系。

*ffserver.conf

```
<Stream test_stream_ibm.avi>
#Feed feed1.ffm
File "/forlinux/ffmpeg-0.8.7/FFmpeg1/bin/gdg.avi"
Format avi
VideoFrameRate 15
VideoSize 320x240
VideoBitRate 275
VideoBufferSize 40
VideoGopSize 30

AudioBitRate 64

PreRoll 15
#StartSendOnKey
#MaxTime 100
MaxTime 1000
StartSendOnKey
</Stream>
```

图 5.6 利用元数据新建一个流节点 test_stream_ibm.avi

5.2.2 媒体源模块

本文中媒体源模块 ffmpeg 主要完成媒体流的转换的功能和流推送的功能。

1. 媒体流的转换

所谓媒体流的转换分以下三种情况：媒体格式转换、媒体编码类型转换及媒体编码类型和格式同时转换。本文中属于第一种情况，即把媒体流换一种组织样式以适应流式

传输的需要，也就是所谓的流化过程。本文中采用的基于 HTTP 的流化，其主要思想是将较大的流媒体文件切割为较小的文件片段进行 HTTP 的请求、发送及传输，被切割的每个流媒体文件切片可以独立解码。流媒体文件的切片需对流媒体文件原来的容器进行解复用，所以切片和容器格式的转换可以同时进行，那么对流媒体容器格式的转换也就转化成对当前切片进行所需的容器格式的封装。在对流媒体切片进行容器的格式转换过程中，需要提取原有容器中封装的时间戳、文件大小和其它一些音视频的描述信息，连同音/视频数据的裸数据一起按新的容器格式封装起来。如图 5.7 所示，为媒体流格式转换的流程图。

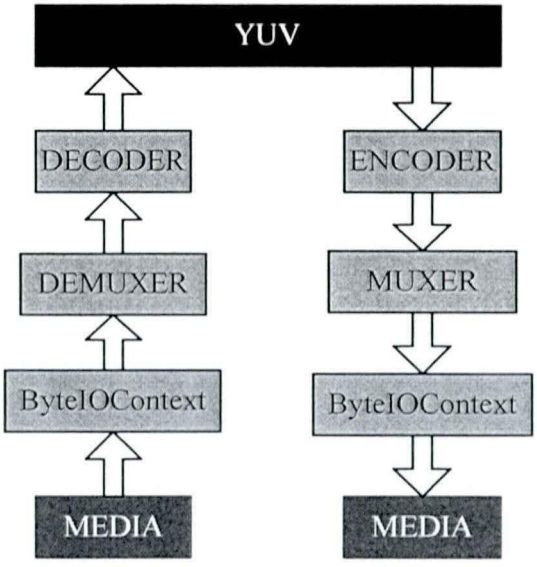

图 5.7 媒体流格式转化流程图

在此转换过程中，如 5.1.1 节所阐述，主要涉及的数据结构有：

(1) AVOutputFormat 与 AVInputFormat 结构，分别对应实现 muxer 和 demuxer 功能；

(2) AVCodec 结构用于 decoder 和 encoder，当被 decoder 所用时函数指针 encode 为 NULL，而当用于 encoder 时其 decode 为 NULL；

(3) AVFormatContext 结构，FFmpeg 格式转换过程中实现输入和输出功能、保存相关数据的主要结构。

2. 媒体流的推送

如图 5.4 所示，实时流媒体的传输是通过 ffmpeg 和 ffserver 配合工作来实现的。ffmpeg 在 ffserver 启动之后就紧接着启动了，它在启动时加的一个关键参数是“http://IP:8090/feed1.ffm”，其中，IP 是运行 ffserver 的主机 IP 地址。如果这两个应用程序是在同一台主机中运行的话，IP 用 localhost 替代也行，在本文中就是这么使用的。在 5.2.1 节中讲到，ffmpeg 启动之后会与 ffserver 建立首个短暂的连接，ffmpeg 通过这个连接从 ffserver 那里获取向客户端输出的流的配置，同时把这些配置作为自己的编码输出配置，然后 ffmpeg 断开了这次连接。接下来，ffmpeg 会再一次与 ffserver 建立一个

长久的连接，ffmpeg 利用这个连接从录制好的音视频文件中获取数据，并按照输出流的编码输出配置方式编码，然后把这些编码后的数据发送给 ffserver，开始进行流的推送。这就是两次连接的过程，如果观察 ffserver 端的输出，可以看到这段时间出现了两次 HTTP 的 200。如图 5.8 所示为 ffmpeg 与 ffserver 配合工作的过程。

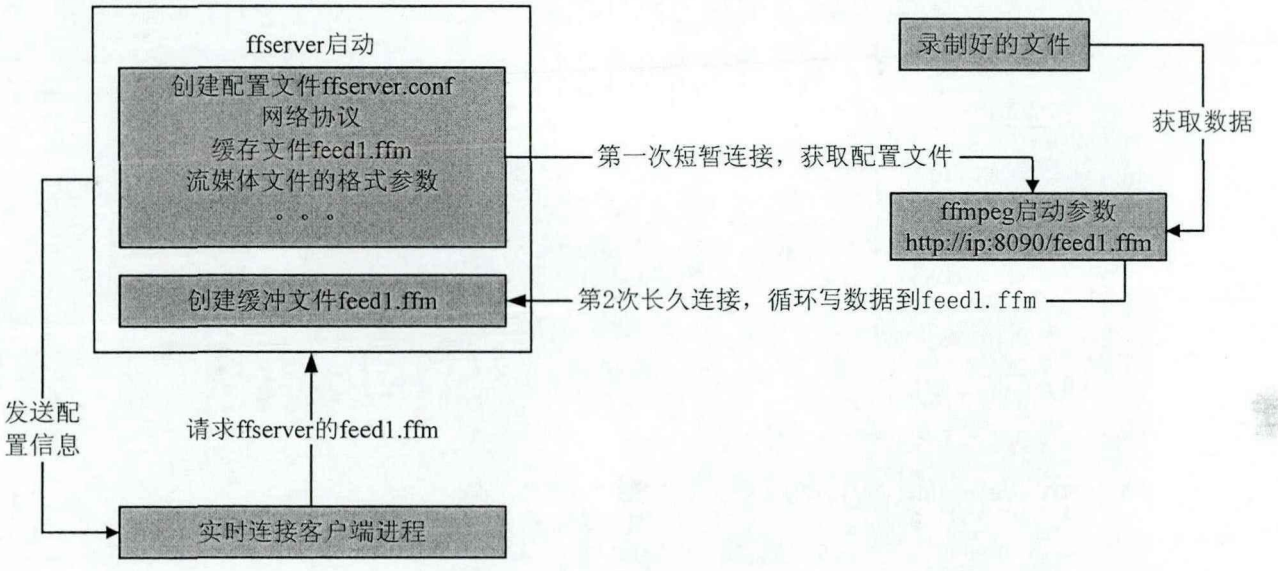

图 5.8 ffmpeg 与 ffserver 合作过程

5.2.3 服务器模块

ffserver 是 FFmpeg 项目自带的一款服务器，支持 RTSP 协议和 HTTP 协议，负责响应客户端的流媒体请求，把流媒体数据发送给客户端，它可以将本地媒体文件作为流输出到网络上，ffserver 启动前后需要做两项工作。首先，写配置文件 ffserver.conf，在这个文件中主要是对网络协议、缓存文件 feed1.ffm 和要发送的流媒体文件的格式参数(如编码方式，帧率，采样率……)进行具体的设定。写完配置文件后，输入 ffserver -f ffserver.conf &（参数-f 指定其配置文件），ffserver 就作为后台程序运行。然后，创建文件 feed1.ffm。feed1.ffm 在这里可以理解为一个缓冲文件，feed1.ffm 在 ffserver 启动后就会立即被创建。在 feed1.ffm 文件中可以查找到向客户端传送流的配置信息和关键字 ffm，这些信息可以理解为 feed1.ffm 的文件头，在 feed1.ffm 被用作缓冲的时候是不会被覆盖掉的[52]。在 ffserver 向 feed1.fm 写入时把数据，需要加上这些头信息并分块，如果 feed1.ffm 文件的大小达到了 ffserver.conf 中规定的大小，ffserver 就会重新从文件开始处（跳过头）写入，从而覆盖旧的数据。在 ffserver 收到 ffmpeg 推送的数据流后，倘若网络上没有播放请求，ffserver 就把数据写入 feed1.ffm 文件中进行缓存。ffserver 会一直等到网络上有播放请求时，才从 feed1.ffm 中读取数据，并将数据发送给客户端。

1. 主要数据结构

ffserver 涉及到两个主要的数据结构：FFStream 和 HTTPContext。FFStream 主要用来存储 ffserver.conf 配置文件中的信息，可以看成是对 AVStream 的进一步封装，本质上还是 AVStream，但又添加了一些跟实时流和文件流相关的一些属性。HTTPContext 主要用来描述与连接网络的上下文结构，是 ffserver 中最重要的一个数据结构，只要连接还没有断开，该数据结构就在发挥作用。HTTPContext 可以看成是对 AVFormatContext 的进一步封装，封装的结果就是使数据更容易被 HTTP 协议或者 RTSP 协议使用，来完成网络连接、数据传递、状态转换等功能。

以下是 FFStream 的定义：

```
typedef struct FFStream {
    enum StreamType stream_type;
    char filename[1024];   //stream filename
    struct FFStream *feed; // feed we are using (can be null if coming from file)
    AVInputFormat *ifmt; // if non NULL, force input format
    AVOutputFormat *fmt;
    AVStream *streams[MAX_STREAMS];
    } FFStream;
```

以下是 HTTPContext 的定义：

```
typedef struct HTTPContext {
    enum HTTPState state;
    int fd; /* socket file descriptor */
    struct sockaddr_in from_addr; /* origin */
    struct pollfd *poll_entry; /* used when polling */
    AVFormatContext *fmt_in;/* input format handling */
    /* output format handling */
    struct FFStream *stream;
    /* -1 is invalid stream */
    int feed_streams[MAX_STREAMS]; /* index of streams in the feed */
    int switch_feed_streams[MAX_STREAMS]; /* index of streams in the feed */
    int switch_pending;
    AVFormatContext fmt_ctx; /* instance of FFStream for one user */
    AVIOContext *pb;
    int seq; /* RTSP sequence number */
    AVFormatContext *rtp_ctx[MAX_STREAMS];
    /* RTP/UDP specific */
```

```
    URLContext *rtp_handles[MAX_STREAMS];
    int n_retansmit;
    /*超时重传的次数，重传超过三次，说明网络拥塞或断开，主动断开和客户的连接*/
} HTTPContext;
```

2. 主要函数的剖析

(1) Main( )函数：

a) 在该函数内部，首先调用 parse_ffconfig(config_filename)来解析 ffserver.conf 文件中的配置参数，解析得到的参数 FFStream 结构；

b) 调用 build_file_stream( )建立文件流，该函数会遍历 FFstream*链表，如果当前的 FFStream 结构的 feed 是来自文件，则会调用 avformat_open _input( )来打开指定的文件；

c) 调用 build_feed_streams( )来建立 FFStream 和 FFStream->feed 之间的关系；

d) 调用 http_server( )，该函数是 ffserver 组件的主循环函数，下面进行分析。

(2) http_server()函数：

a) 建立一个 Poll 表：

Struct pollfd=av_mallocz((nb_max_http_connections+2)*sizeof(*oll_table))，之所以+2，是因为有两个监听套接字要考虑,一个用来监听 HTTP 协议，一个用来监听 RTSP 协议。在本文中,则只监听 HTTP 请求。

b) 服务器开始广播：

start_multicast( )，针对 RTP 协议，对于与服务器连接的 Stream，发送广播报文。该函数调用 rtp_new_connection(&dest_addr,session_id,session_id,RTSP_LOWER_TRANSPORT_UDP_MULTICAST)建立连接，并将连接的 HTTPContext 添加到全局 HTTPContext 的头位置。

c) 开始监听是否有新的链接发生。对使用 HTTP 和 RTSP 协议的两个监听套接字，都进行了监听，在本文中只监听到使用 HTTP 协议的套接字。

d) 不断的循环判断 Poll 表上是否有事件发生，这些事件包括：POLLIN（有数据可读），POLLOUT（写数据），POLLUP（挂起事件）等，并调用相应的函数对事件进行处理。这个处理过程函数 handle_connection(HTTPCOntext *c)来统领。

(3) handle_connection(HTTPContext *c)函数：

处理 HTTPContext 状态。HTTPContext 状态是 HTTPState 定义的 11 种状态。HTTPContext 的状态在不断的发生变化，handle_connection( )函数就是来处理对应的状态。

a) 如果当前 HTTPContext 的状态不是 HTTPState 定义的 11 种状态之一，则说明当前连接存在错误，则 handle_connection( )会调用 close_connection(c)关闭当前连接；

b) 如果当前 HTTPContext 的状态是 HTTPSTATE_RECEIVE_DATA 表示需要接收，

这时会调用 static int http_reveive_data(HTTPContext*c)，该函数调用 API 函数 recv(c->fd, c->buffer_ptr,1,0)从客户端接收数据；

c) 如果当前 HTTPContext 的状态是 HTTPSTATE_SEND_DATA、HTTPSTATE_SEND_DATA_HEADER、HTTPSTATE_SEND_DATA_TRAILER 三种状态之一，表示需要发送数据，这时调用 static int http_send_data(HTTPContext *c)，该函数调用 http_prepare_data( )，最后调用 API 函数 send(c->fd,c->buffer_ptr,c->buffer_end-c->buffer_ptr,0)从服务器向客户端发送数据；

这个函数处理了所有的十一种 HTTPContext 状态，逻辑结构比较复杂。在下面函数调用流程图中进行了简单的梳理，如图 5.9 所示。

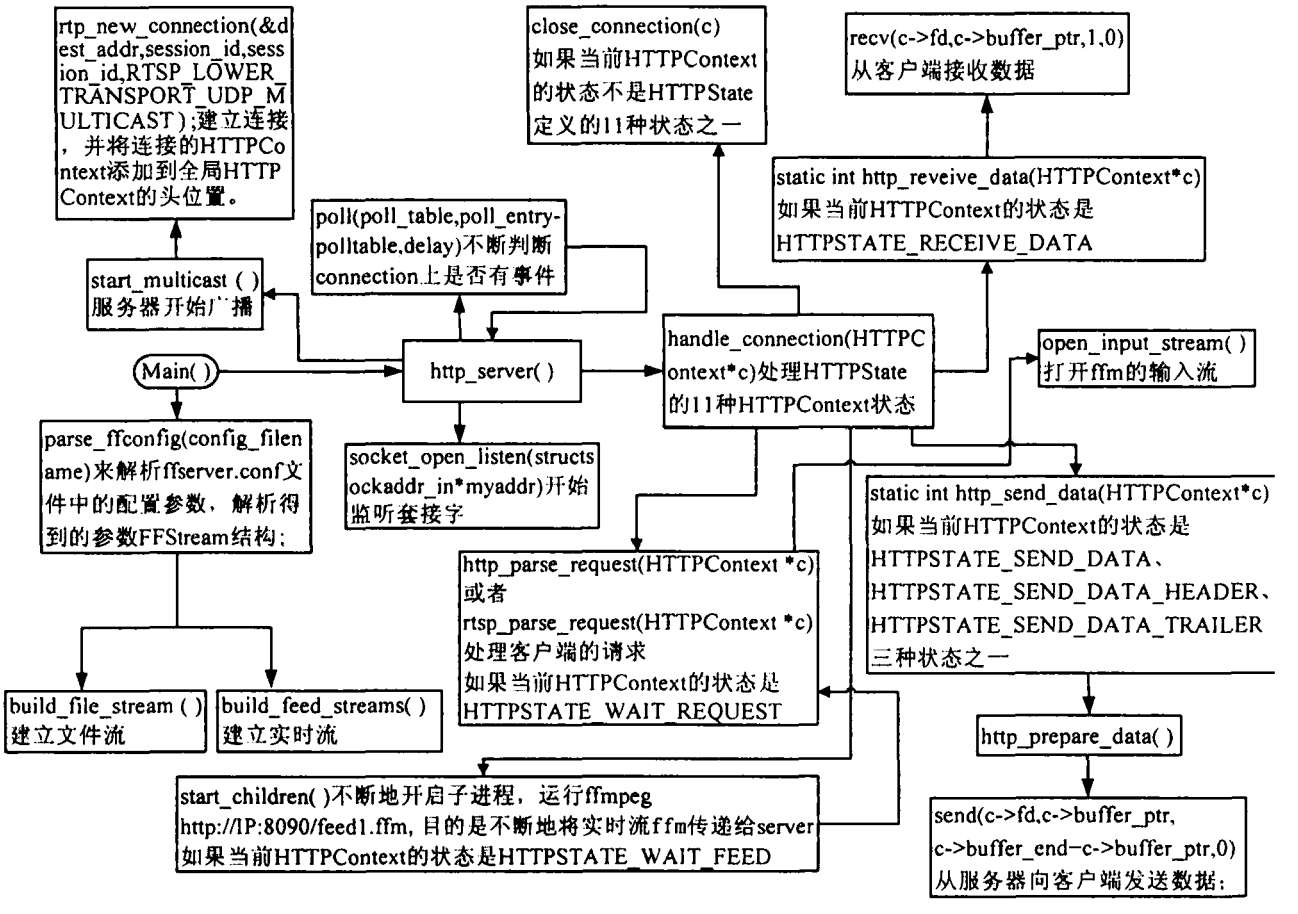

图 5.9 ffserver 函数调用流程图

5.2.4 播放模块

ffplay 是一个基于 FFmpeg 库和 SDL 库的一个非常简单的可移植媒体播放器，它可以作为使用 FFmpeg 的 API 的例子。在本文中，它主要用来接收 ffserver 分发的媒体流，并解码播放，同时向 ffserver 发送反馈信息。ffplay 播放文件的流程可以分为四个模块：文件解析模块、Demux 解复用模块、解码模块及播放模块，这些模块对播放文件的处理过程如图 5.10 所示。

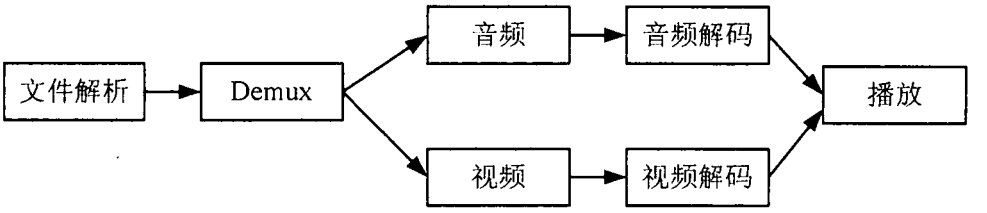

图 5.10 ffplay 播放文件流程图

1. 重要函数及数据结构

(1) 重要数据结构：ffplay 涉及的主要数据结构 AVFormatContext、AVOutputFormat、AVInputFormat、AVCodecContext、AVCodec、AVFrame、AVPacket、AVStream；

(2) 解析函数：parse_options( )，对所输入的命令参数进行解析，其中，在利用 FFmpeg 库中的文件转换特性过程中，还要分别解析输入文件与输出文件的后缀名，以匹配合适的 muxer 和 demuxer。

(3) 初始化函数：avcodec_register_all( )函数和 av_register_all( )函数初始化该库，其中，avcodec_register_all( )用于注册解析器、encoder/decoder、硬件加速器等，而 av_register_all()则主要用于注册 muxer/demuxer 及各种传输协议。另外，还有 avcodec_open( )、avcodec_close( )、av_open_input_file( )、av_find_input_format( )、av_find_stream_info( )、av_close_input_file( )等初始化函数。

(3) 音视频编解码函数：avcodec_find_decoder( )、avcodec_alloc_frame( )、avpicture_get_size( )、avpicture_fill( )、img_convert( )、avcodec_alloc_context( )、avcodec_decode_video( )、av_free_packet( )、av_free( )。

2. 解码流程

ffplay 中的解码可以分为 3 步：

(1) 利用输入端从源数据中取得的信息调用不同的解码库初始化；

(2) 接收输入端传送来的音视频编码数据，分别进行音频解码和视频解码，视频解码出来的数据一般 YUV 或 RGB 数据，称为 picture，音频解码出来的数据是采样数据，是声卡可以播放的数据，称为 sample。

(3) 将解码所得的数据送到输出部分。

如图 5.11 所示是解码流程图。

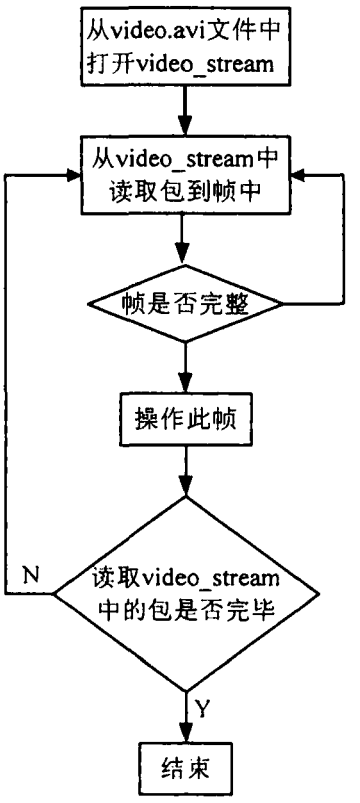

图 5.11 ffplay 解码流程

5.3 系统测试

5.3.1 本机测试

在完成软硬件的设计后，本文先对流媒体播放系统进行了本机测试，即 ffmpeg、ffserver 和 ffplay 均运行在宿主机上。测试时,已经编译好的上面三个可执行程序所在的目录为/home/STREAMING，测试所使用的视频文件名称为 hlxd.avi，测试操作步骤如下：

(1) 在宿主机 Ubuntu 下开启三个命令行终端窗口，分别记为：A、B、C。

(2) 在 A 窗口操作切换到管理员权限登陆根据提示输入密码：

```
#su root
```

进入工作目录：

```
#cd /home/STREAMING
```

首先查看 ffserver 进程是否存在：

```
#ps aux|grep ffserver
```

如果存在，则终止该进程：

```
#kill -9 进程号
```

启动 ffserver 进程：

```
#./ffserver -f ./ffserver.conf &
```

(3) 在 B 窗体切换到管理员权限登陆，根据提示输入密码：

#su root

进入工作目录：

#cd /home/STREAMING

启动 ffplay 播放器进程，并将播放地址传递给播放器：

#./ffmpeg -re -i ./hlxd.avi http://主机 IP:8090/feed1.ffm

(4) 在 C 窗体切换到管理员权限登陆，根据提示输入密码:

#su root

进入工作目录：

#cd /home/STREAMING

启动 ffmpeg 进程，开始向 ffserver 推送 http 直播码流：

#./ffplay http://主机 IP:8090/test_stream.avi

注：主机 IP 地址可通过命令：#ifconfig 查看

测试结果如图 5.12 所示。

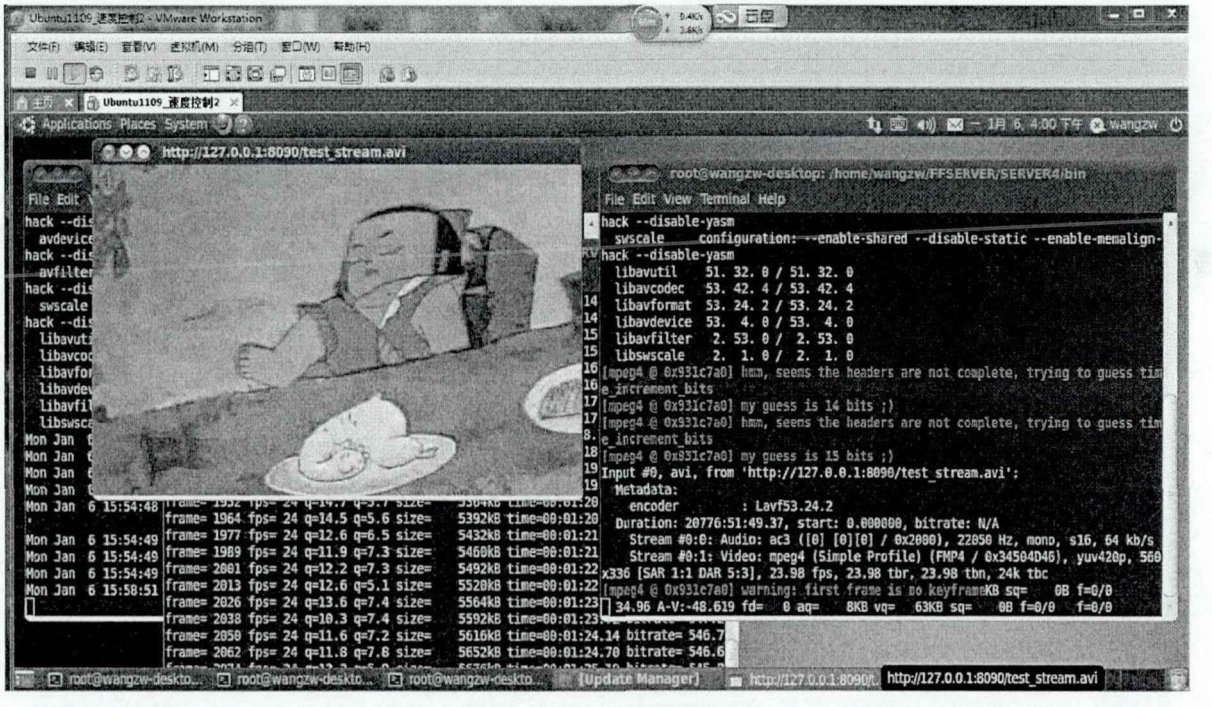

图 5.12 本机测试界面

5.3.2 开发板测试

本机测试及播放通过后，需要将 ffplay 移植到开发板上以实现所谓嵌入式流媒体播放系统。本文中通过 4.2.5 节中介绍的 NFS 挂载的方式让 ffplay 运行于开放板上，而 ffmpeg 和 ffserver 仍运行在宿主机上。因此，测试时 ffmpeg、ffserver 不需要重新编译，而 ffplay 则需要通过交叉编译生成可运行于 ARM 架构的程序。由于 ffplay 使用了 FFmpeg

库中的相关结构和函数，且需要 SDL 库的支持，因此在 ffplay 交叉编译之前，需要先在宿主机上交叉编译 FFmpeg 源码和 SDL 库，见 5.1.2 节所述。所有交叉编译后生成的文件均放在宿主机/nfs 下。测试操作步骤如下：

(1) 启动开发板，正确挂载文件系统（方法参照 4.5 节），本文中是将宿主机/nfs 挂载在/mnt 下；

(2) 在宿主机 ubuntu 上开启一个命令行终端窗口 A，切换到 root 权限，同本机测试操作：

先进入工作目录：

#cd /home/STREAMING

首先查看 ffserver 进程是否存在：

#ps aux|grep ffserver;

如果存在，则终止该进程：

#kill 进程号

启动 ffserver 进程：

#./ffserver -f ./ffserver.conf &

(3) 在开发板 DNW 窗口切换到 root 权限，进入到 nfs 挂载目录，开启 ffpaly：

#cd /mnt

#./ffplay http://主机 IP:8090/test_stream.avi

(4) 在宿主机 ubuntu 上开启另一个命令行终端窗口 B，同样切换到 root 权限，同本机操作：

先进入工作目录：

#cd /home/STREAMING

启动 ffmpeg 进程，开始向 ffserver 推送 http 直播码流：

#./ffmpeg -re -i ./hlxd.avi http://主机 IP:8090/feed1.ffm

测试结果如图 5.13 所示。

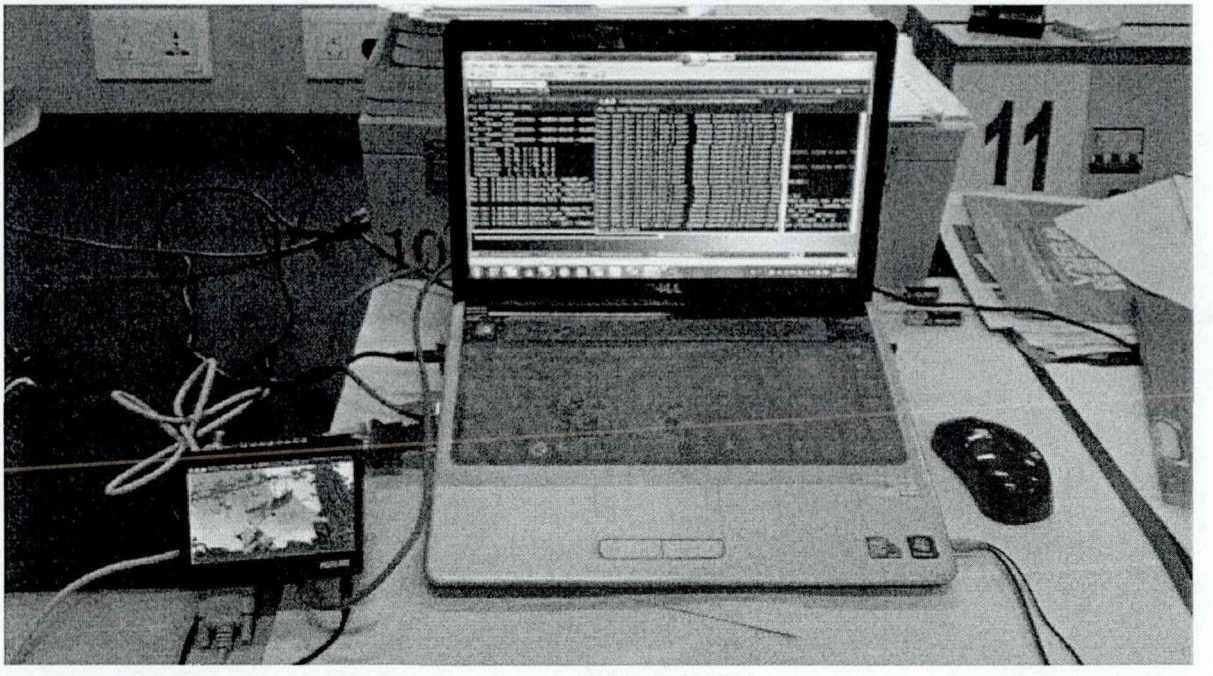

图 5.13 开发板测试界面

测试结果表明，本文设计的基于 TCP 传输的嵌入式流媒体播放系统，其终端播放器软件能够运行于 S3C6410 平台上，可以接收并播放由其媒体源模块和服务器模块配合工作发送过来的多媒体数据，且播放视频时画面稳定流畅，满足对播放实时性的要求，达到了预期的设计目标。

6 媒体流传输速度控制的实现

本文中媒体流传输速度的控制主要由流媒体服务器来实现，其策略可以划分为主动控制、反馈调节和拥塞控制三个部分。

6.1 主动控制

ffserver 接收 ffmpeg 推送的码流并将其缓存到 feed1.ffm 文件中。这个缓存文件是 FFM2 格式，ffserver 采用循环覆盖的方式来缓存到达的码流，即如果 feed1.ffm 已经写满，则跳过 feed1.ffm 的头文件从头开始覆盖旧的内容。feed1.ffm 中的编码数据通过 c->buffer_ptr 和 c->buffer_end 两个指针传递给 HTTPContext 结构体。如果有客户端请求码流，ffserver 需要根据 c->buffer_ptr 和 c->buffer_end 这两个指针来决定要发送哪些数据。

当 ffserver 向客户端 ffplay 发送数据的时候，可以采取两种方式：

(1) 每次客户端 ffplay 需要数据的时候，就发送指定字节长度（比如：2M）的数据给客户端，这样的实现方法优点是简单，ffplay 播放器本身也有自己的播放缓存，可以将收到的这些数据缓存起来，可以减少向 ffserver 请求数据的次数；缺点就是每次发送指定长度的字节流数据，没有考虑到网络的负载和拥塞，在网络状况不佳的情况下，会严重影响客户端的接收质量。

(2) 每次有客户端 ffplay 请求数据的时候，根据缓存在 feed1.ffm 的数据的 PTS（显示时间戳）和当前系统时钟 SCR 计算出字节流的最晚发送时间，即：在这个时间点之前，数据必须发送出去，否则，丢弃。这样可以保证数据的发送是有机制的发送，而不是机械的、无序的发送。

本系统使用第二种方式来实现对媒体流的发送速度的控制。

音频和视频流都有一些关于以多快速度和什么时间来播放它们的信息在里面，如音频流有采样频率，视频流有帧率。然而，如果只是简单地通过数帧和乘以帧率的方式来同步视频，那么就很有可能会失去同步。于是作为一种补充，在媒体流的包中有种叫做 PTS（显示时间戳）和 DTS（解码时间戳）的机制。

DTS（解码时间戳）是在解码阶段使用的相对于 SCR（系统参考）的时间戳（SCR 可以理解为解码器应该开始从磁盘读取数据时的时间），用于视频解码；PTS（显示时间戳）是在显示帧时使用的时间戳，用于视频的同步和输出，同样是相对于 SCR（系统参考）的。一般 DTS/PTS 时间戳指示的是晚于音视频包中的 SCR 的一个时间，它们决定了解码和播放的时间在 SCR 时间等于 DTS/PTS 时间时。例如，如果一个视频数据包的 SCR 是 100ms（意味着此包是播放 100ms 以后从磁盘中读取的），那么 DTS/PTS 值

就差不多是 200/280ms，表明当 SCR 到 200ms 时这个视频数据应该被解码并在 80ms 以后被显示出来（视频数据在一个 buffer 中一直保存到开始解码）。

发送的数据包的 AvPacket 的 PTS 计算方法是：

```
pkt.pts = (INT64)st->time_base.den * (lTimeStamp -
                                    g_nFirstVideoTimeStamp)/10000000i64;
```

其中，st 为指向 AVStream 结构体的指针，默认 st->time_base.den = 90000，lTimeStamp 为当前的时间戳（当前系统时钟的时间戳），g_nFirstVideoTimeStamp 为视频第一帧的时间戳（系统时钟的时间戳 ）。 视频的 AvFrame 的 PTS 基本上是每增加 1 帧加 1（假如视频采集帧率跟编码帧率一致），一开始将 AvFrame 的 PTS 置为 0；音频的 AvFrame 的 PTS 不用处理，音频 AvPacket 的 PTS 每次加上收到音频帧的时长。

通过下面的方法来计算 HTTPContext 的包的 PTS：

```
#define AV_TIME_BASE_Q (AVRational){1, AV_TIME_BASE}
c->cur_pts = av_rescale_q(pkt.dts, ist->time_base, AV_TIME_BASE_Q);
c->cur_pts -= c->first_pts;
c->cur_frame_duration = av_rescale_q(pkt.duration, ist->time_base,
                                     AV_TIME_BASE_Q);
```

av_rescale_q(a,b,c)是用来把时间戳从一个时基调整到另外一个时基时候用的函数。它基本的动作是计算 a*b/c，pkt.duration 是包的播放时长。

计算 HTTPContext 包的时候使用了 AVPacket 的 DTS（解码时间戳），这样做的目的是顺序计算每个包的 PTS，可以简化计算。流媒体服务器计算 HTTPContext 包的 PTS 的目的是为了将包在合适的时间发送出去，而不是为了解码和显示，所以使用了原先包的 DTS 来计算。

ffserver 实现对数据发送速度的控制使用了 int64_t get_packet_send_clock(HTTPC Ontext *c)函数。该函数通过 PTS 来控制数据的最迟发送时间：

```
bytes_left = c->buffer_end - c->buffer_ptr;   //还有多少字节要发送
bytes_sent = frame_bytes - bytes_left;        //目前已经发送多少字节
return c->cur_pts + (c->cur_frame_duration * bytes_sent) / frame_bytes;   //在此 pts 到达之时，
```

这些字节必须全部发送出去。

通过使用 get_packet_send_clock( )，可以保证数据的发送是有机制的，而不是简单的匀速发送。

```
/* 如果包的发送时间还没有到达，则等待（函数返回），反之，发送出去 */
if ((get_packet_send_clock(c) - get_server_clock(c)) > 0) {
        return 0;
    }
```

```
/* 计算当前包在必须发送的最晚时间 */
static int64_t get_packet_send_clock(HTTPContext *c)
{
    int bytes_left, bytes_sent, frame_bytes;
    frame_bytes = c->cur_frame_bytes;
    if (frame_bytes <= 0)
        return c->cur_pts;
    else {
        bytes_left = c->buffer_end - c->buffer_ptr;
        bytes_sent = frame_bytes - bytes_left;
        return c->cur_pts + (c->cur_frame_duration * bytes_sent) / frame_bytes;
    }
}
```

6.2 反馈调节

目前，流媒体传输系统中速度控制策略分为基于信源的速率调整策略和基于接收机的速率调整策略。在基于信源的速率调整策略中，做出速率调整操作的是发送端，接收端向发送端发送包含有延时抖动、时间戳、丢失的数据包数目等信息的接收端报告，发送端根据这些反馈信息估计网络延迟、数据包丢失率等，从而判断出网络状况，并据此调整发送速度以最佳化网络带宽利用率，提高传输服务质量。而在基于接收机的速率调整策略中，则发送方不参与速率控制，而是由接收方通过增加或减少通道的数量进行速率的调整。具体到本文实现的 ffserver，ffserver 只是简单的接收由客户端反馈的接收到的包的序列号，由此来判断哪些数据已经成功发送，接下来可以发送该序号之后的数据了，ffserver 的实现过程如下：

在 ffserver 中，处理 HTTPContext 的状态 HTTPSTATE_WAIT_REQUEST 或 RTSPSTATE_WAIT_REQUEST，调用 recv(c->fd, c->buffer_ptr, 1, 0)接收数据，c->buffer_ptr 中就得到了当前客户端已经接收完成的包的序列号，当 HTTPContext 的状态变为需要发送数据时，则发送数据：send(c->fd, c->buffer_ptr, c->buffer_end - c->buffer_ptr, 0)，只将客户确认收到的数据的序列号后面的数据发送给客户端。这个过程算是 ffserver 的简单的确认机制，如图 6.1 所示。

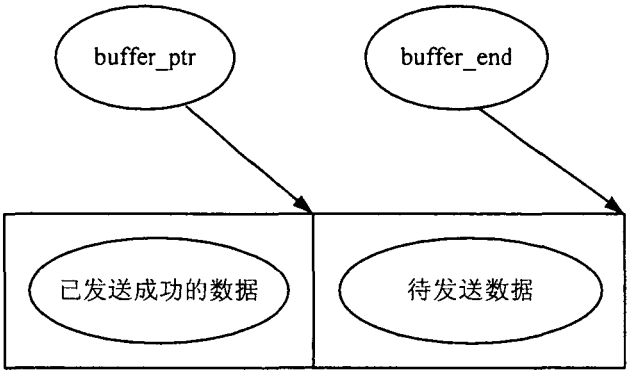

图 6.1 ffserver 的确认机制

6.3 拥塞控制

6.3.1 TCP 的拥塞控制机制

在本文中，ffserver 对媒体流的分发和传输是基于 TCP 协议的，该传输协议提供可靠的、面向连接的字节流服务，它保证传输可靠性的方式如下：

• 采用面向连接的三次握手实现可靠对象传输。

• 应用程序把传输数据分割为 TCP 认为大小最适合发送的数据块，并将该信息单位称作报文段，由 TCP 将其传递给 IP。

• TCP 具有等待确认机制，倘若不能在规定时间内收到一个确认，TCP 就会重发这个报文段，也就是所说的超时重传。等待时间由重传定时器来计算，该定时器在 TCP 发出一个报文段后就立即启动。

• 在收到来自 TCP 连接的另一端数据时，一般延迟几分之一秒的时间 TCP 就会发送一个确认。

• TCP 提供检验数据功能，为了检测在传输过程中的数据发生的任何变化，TCP 保留数据和其首部检验和。该检验和是端到端的，倘若收到的报文段的检验和出现差错，那么 TCP 会丢弃这个报文段并不发送收到该报文段的确认。

• 由于 TCP 位于 IP 之上，故 TCP 报文段的到达有可能乱序，这是因为 IP 数据报的到达可能出现乱序现象。所以，为了将收到的数据按照正确的数据交给应用层，TCP 要对收到的数据重新排列。

• TCP 提供丢弃重复数据的功能，其连接端必须丢弃 IP 数据报产生的重复数据。

• 为连接的每一端建立大小固定的缓冲空间，并且为了防止发送数据较快的主机致使接收数据较慢的主机的缓冲区溢出，它仅允许发送端发送接收端缓冲区可以容纳的数据，这就是 TCP 实现流量控制的方式。

此外，TCP 不解释字节流的内容[53][54]。

因此，当客户端和服务器彼此交换数据前，必须先在双方之间建立一个 TCP 连接，之后才能传输数据，而且加上 TCP 提供的超时重发、丢弃重复数据、检验数据及流量

控制等功能，保证了数据能从一端传到另一端。这跟 RTSP 协议使用的 UDP 传输不同，UDP 在客户端和服务器交换数据之前不建立连接，它仅仅发送应用程序提交给 IP 层的数据报，而且也不像 TCP 那样提供超时重发等机制，所以 UDP 是不可靠的，它不能保证接收端都能接收到发送端发送的数据报。

TCP 拥塞控制的原理是控制一个拥塞窗口（cwnd），该窗口值的大小代表可以发送出去但是还没收到确认的最大数据报文段。可见，窗口值的大小与数据发送速度是成正比关系的，cwnd 的值越大则发送速度越快，但是 cwnd 增大的同时也增大了网络拥塞的可能，这样反过来会降低传输速度。所以，如果要想网络吞吐量最大的同时又不出现网络拥塞的现象，必须在 cwnd 和数据传输速度之间找到一个最佳的平衡点。

慢启动：为了避免产生网络拥塞现象，TCP 在成功建立连接后，会根据网络状况逐步增加每次的数据发送量，而不是一开始就向网络大量发送数据包。具体来说，拥塞窗口 cwnd 的值是随着网络往返时间（RTT）呈指数级增长的。在连接刚建立起来时，TCP 把拥塞窗口的值设置为 1，发送端按照这个初始值发送数据，然后 TCP 每收到一个确认就将拥塞窗口 cwnd 的值加 1。事实上，慢启动只是起点不高而已，其最终传输速度一点也不慢。其简单计算过程如下：

开始时——》 cwnd=1

经 1 个 RTT 后——》cwnd=2*1=2

经 2 个 RTT 后——》cwnd=2*2=4

经 3 个 RTT 后——》cwnd=4*2=8

所以，如果带宽为 W，那么经过 RTT*$\log_2 W$ 时间就可以占满带宽。

拥塞避免：在慢启动阶段，cwnd 的增长速度是非常快的，然而过快的增长很快就会造成网络拥塞。为此，TCP 提供了一个称为慢启动门限（ssthresh）的参量来限制 cwnd 的增长，当 cwnd 的值大于 ssthresh 的值时，TCP 结束慢启动进入拥塞避免阶段。在这个阶段，TCP 只有收到所有发送出去的报文段的确认时，才会把 cwnd 的值加1。这种 cwnd 随着往返延迟时间（RTT）线性增加的慢增长方式既避免了网络拥塞，同时又将网络资源的利用率最大化。

检测及处理拥塞：上述的两种机制都是为了避免发生网络拥塞而采取的办法，但在实际中很可能在采取避免措施之前网络已经出现拥塞，这时 TCP 需要对拥塞进行相应处理。但是，在进行处理之前，TCP 首先要检测到网络拥塞状况，其检测依据是 TCP 是否重传了一个报文段。前面提到，在每次发送一个报文段之后 TCP 会启动一个重传定时器（RTO），当在 RTO 设置的时间内没有收到已经接收到报文段的确认时，TCP 就会重传这个这个报文段，这种情况下也就很可能是出现了网络拥塞，有可能造成某个报文段丢失，进而影响后续报文段的传输。在这种情况下，TCP 的处理方法如下：

(1) 把 ssthresh 的值降低为 cwnd 的值的一半；

(2) 把 cwnd 的值重新设置为 1；

(3) 重新进入慢启动过程；

可以看到，TCP 对拥塞的反映是相当的强烈的，一下子将滑动窗口的窗口长度减小为原来的 1/2[55]。

6.3.2 改进的 TCP 拥塞控制机制

由于本文中 ffserver 对视频的分发是基于 PTS 的，即：ffserver 的分发和 ffplay 的播放有一个时间的同步（基于 PTS 的同步），如果一下子将滑动窗口减小为原来的 1/2，势必会造成网络传输的剧烈抖动，从而使 ffplay 不能及时接收到 TCP 包，造成帧的大量丢失，使画面严重损伤。

另外，TCP 的拥塞控制是一种工作在网络链路层之上的字节流的传输控制，是没有语义的，为了避免这种网络的剧烈抖动和无语义的拥塞控制，ffserver 在 TCP 协议之上对超时重传和拥塞控制进行了控制，主要思想如下：

(a) 在 ffserver 全局关键数据结构 HTTPContext 中增加控制字段：

```
int n_retansmit;
```

这个字段用来标识超时的次数，默认值是 0，当超时发生的时候，则自增 1；

(b) ffserver 是一个基于状态机的 HTTP 服务器，对每一个数据分组的发送，都会启动一个计时器。如果在 ffserver 定义的超时时间到达，接收 ffserver 的客户端(ffplay)还没有返回按序到达的包的最大编号，则状态机会跳转到 HTTP_REQUEST_TIMEOUT 状态，表示网络可能发生了阻塞，这时，需要 n_retansmit 自增 1，并将上次要发送的包的总长度的(3/4)^ n_retansmit 进行重传，实现过程如下：

```
/*每次重传都逐步减少发送的字节数为原来的(3/4)  */
        len = send(c->fd, c->buffer_ptr, (c->buffer_end - c->buffer_ptr)*(3/4)
        ^c->n_retansmit), 0);
```

(c) 重传超过三次，则说明网络阻塞严重，已经不能正常进行 TCP 传输或网络已经中断，则 ffserver 主动断开和客户的连接，实现如下：

```
/*如果重传次数超过三次，则主动断开和客户端的链接*/
        if(c->n_retansmit++ >= 3)
            return -1;
```

6.4 速度控制总结

ffserver 的发送速度的控制概括如下：

ffserver 通过计算音视频包的 pts 来获取当前包的最迟发送时间，在这个时间到达之前，包必须被发送出去；ffserver 接收客户端 ffplay 发送的反馈信息（ffplay 接收到的有

序包的最大序列号），依此序列号来决定下一步要发送的包的范围；如果在规定的时间内没有收到 ffplay 的反馈信息，则 ffserver 认为当前网络不佳，可能有拥塞发生，启动拥塞控制，减小发送窗口为先前发送窗口大小的 3/4，如果超时重传三次依然没有客户端的反馈信息，则主动断开和客户端的连接。如图 6.2 所示是播放速度控制的泳道图。

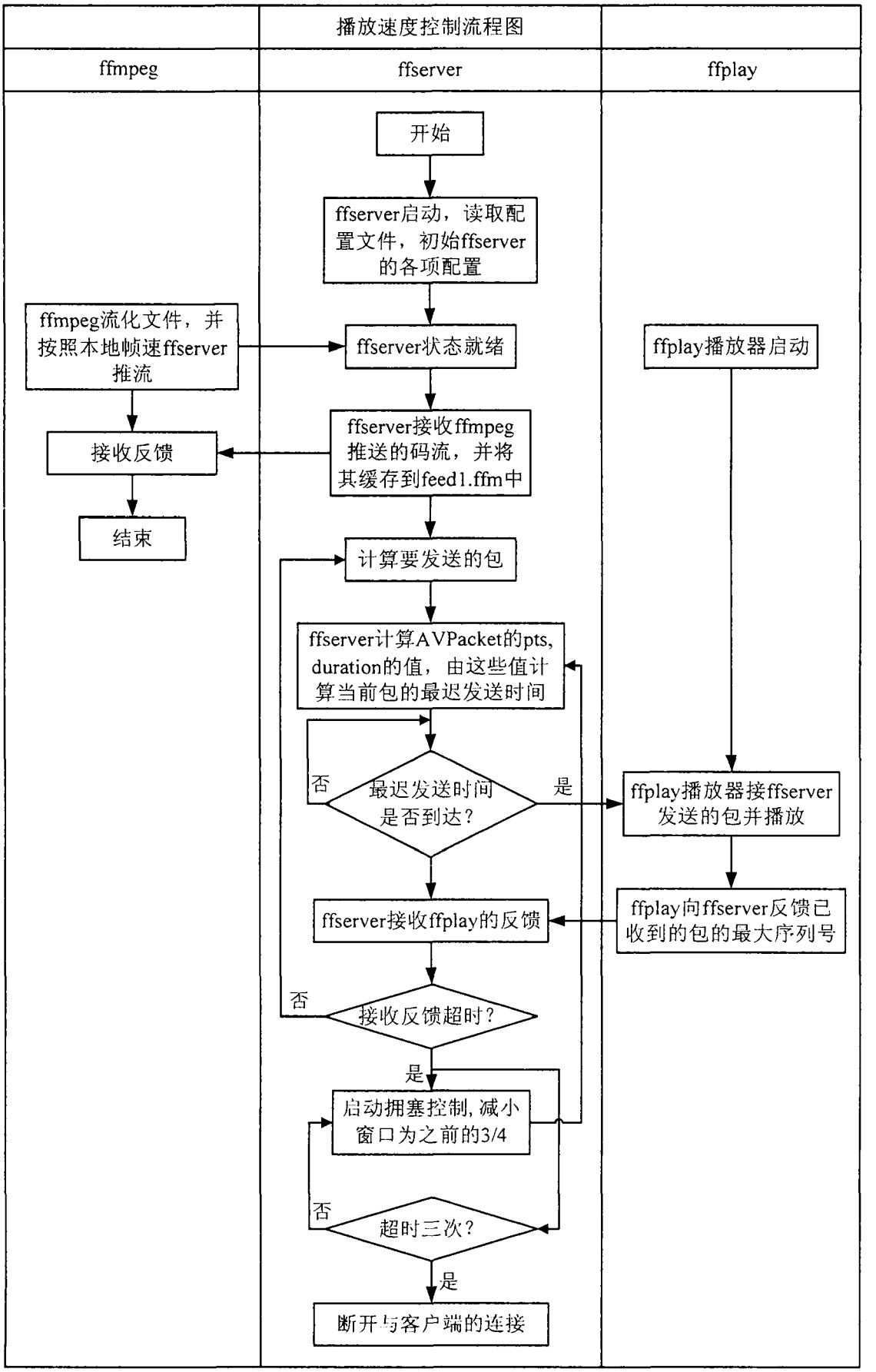

图 6.2 播放速度控制泳道图

7 总结与展望

7.1 总结

嵌入式流媒体播放系统是嵌入式技术和流媒体技术的结合，它解决了传统的流媒体播放器不能独立于 PC，不便于携带的缺点。而流媒体传输技术作为流媒体的重要的部分，长久以来其传输协议的选择都否认 TCP 的适合性。然而现在很多主流的流媒体服务网站，如优酷、爱奇艺等都采用基于 TCP 的 HTTP 渐行下载技术，且几大 IT 公司都有使用基于 TCP 的 HTTP 协议传输流媒体的成功商用的例子，如微软公司的 Smooth Streaming 方案、苹果公司的 HTTP Live Streaming 方案，这使得人们开始考虑 TCP 传输流媒体的诸多优点，采用 TCP 协议作为实时流媒体的传输协议。本文以嵌入式技术和流媒体技术为基础，以 FFmpeg 项目为音视频处理的库和中间件，以简化的基于 TCP 的 HLS 协议为流媒体传输协议，研究并初步实现了嵌入式流媒体播放系统。自从确定课题方向以来，本文围绕嵌入式处理器，嵌入式 LINUX 操作系统及在其下的应用程序的开发及流媒体传输技术等方面进行大量的研究。整个研究过程中完成的主要工作总结如下：

1. 了解嵌入式技术和流媒体技术的相关知识以及研究现状，确定课题方向；

2. 研究嵌入式流媒体播放系统相关的关键技术，确定系统的总体设计方案。系统软件开发基于 FFmpeg 库，在其项目 ffmpeg、ffserver 和 ffplay 基础上进行修改完善，分别完成媒体源处理、流媒体服务器和流媒体终端播放器的功能，三者配合组成完整的流媒体实时播放系统；媒体源模块及流媒体服务器模块的软件平台采用 Linux 操作系统，嵌入式流媒体终端播放器运行平台为 ARM-Linux 系统，硬件设计以 ARM11 处理器 S3C6410 为核心；流媒体传输协议选择 TCP 协议。

3. 学习嵌入式 Linux 的应用程序开发，在基于 ARM11 处理器的平台上构建嵌入式 ARM-Linux 操作系统。这部分工作内容包括了交叉工具链的安装与使用，Bootloader 和内核的编译、文件系统的制作以及这三者的一键烧写，NFS 开发调试环境的搭建。

4. 研究 FFmpeg 项目，借鉴与修改其中的 ffmpeg、ffserver 及 ffplay 的代码，构建完整的流媒体播放系统，并将播放器 ffplay 通过 NFS 挂载方式移植到嵌入式开发平台。

5. 研究网络传输协议 TCP，基于其超时重传和拥塞控制机制进行改进，实现适合流媒体文件的传输速度控制机制，达到播放流媒体文件的预期的流畅效果。

7.2 展望

由于时间、条件的限制及个人的精力有限，虽然已经做了很多工作，但还有很多

的不足。从嵌入式系统设计的层次来看，目前的工作只是完成了操作系统的移植以及基本的上层应用程序的设计，整个嵌入式流媒体播放系统目前还只是一个初步的模型。针对在实际应用场合的可能，未来的研究工作应在以下方面进行改进和细化：

1. 嵌入式用户图形界面的软件

未来工作可以实现 HTTP 浏览器，并增加终端播放器的图形操作界面，使得用户可以更直观进行图形化点播并控制播放器的行为，加强交互性能。

2. 提高系统的软件效率

目前系统在同时进行网络数据流的接收并解码时的处理能力仍然存在缺陷，终端播放画面有时会出现迟钝现象，另外，系统软件功能不是特别稳定，有时需要重启。因此，需要优化网络传输协议的速率控制方法，减少实时媒体流传输过程中的抖动及延迟，以提供更好的用户响应性能以及网络传输效果。

3. 流媒体协议解析的完善

在后续的开发中，可以更深入地完善流媒体服务器 HLS 协议，增加对 m3u(8)的索引文件，TS 媒体分片文件和 key 加密串文件的支持，实现完整的直播或点播功能。同时，在媒体流传输速度控制模块，目前只是播放终端只是简单将接收到的数据包的序列号反馈给服务器端，没有考虑到丢包率、网络延迟等信息，下一步考虑反馈更完整的信息给服务器以提高媒体流的发送速率及最终播放效果。

致　谢

光阴似箭，两年半的时间稍纵即逝，在南理工的研究生生涯即将随着学位论文的完成画上句号。回首过往的岁月，一切都历历在目，一切还是那么的亲切。在导师的悉心教导、在家人的无私关怀、在朋友们的默默关注下，曾经的迷茫跨越了，曾经的困惑清晰了，曾经的疑问解决了。研究生的生活给予我的不仅仅是专业技术知识的提高，更是很多方面的自我认知与成熟，这些都是我人生当中无法取代的重要财富。

首先我要衷心感谢的是我的导师孙建红教授。自从攻读研究生以来，孙老师给予我很多方面的正能量。她工作的认真与严谨，鞭策着我时刻保持向前，不落在队伍后面；她的乐观、充满激情的生活态度，深深的影响我，让我学会了笑对困难；她平易近人的性格，让我们之间没有距离感，能感受到母亲一样的温暖；她更是用自己的亲身经历让我体会做人、做事的道理。所有这些在现在以及未来工作岗位上我都能受用终身，再一次向最可爱的孙老师表示感谢！

感谢我的家人，特别是我的母亲，您对我无私的关爱与付出，对我任性的宽容与理解，对我不管是物质的支持还是精神上的鼓励，是这个世界上没有任何东西可以取代的。您是我最坚实的依靠，我要永远做您最贴心的小棉袄！

感谢我的男朋友，在我求职与做毕设这段时间对我的鼓励。谢谢你理解我的压力，忍受我因此而来的小脾气，你的宽容与支持，是我最幸福的动力！

感谢教研室的兄弟姐们，刘普、宋晓丽、桂祈平、汪涛、李航、宋欠等，谢谢你们对我生活上的照顾和科研上的帮助，与你们的相处是我度过了非常美好的一段时光！

最后再次感谢我的导师、亲人和朋友，感谢南京理工大学！

参考文献

[1] 方开红，邬春学，罗云，卢书田．一种自适应网络带宽控制流媒体视频流传输方法[J]．仪器仪表用户，2007，01：104-105

[2] 王永明，邓中亮，李喆腾．实时流媒体网络编程[OL]．[2007-05-14]．中国科技论文在线，http://www.paper.edu.cn/releasepaper/content/200705-183

[3] 卢宫明．移动流媒体技术[M]．北京：电子工业出版社，2012

[4] 慈文彦，何君，朱明祥．基于ARM处理器的流媒体播放器客户端的构建[J]．信息技术，2012，01：106-108+112

[5] 孙弼阳，李虹，王颖．移动流媒体业务的技术与应用[J]．现代电信科技，2008，06：10-15

[6] 周炜．基于嵌入式Linux无线流媒体播放器的设计与实现[D]．合肥工业大学，2008

[7] 齐俊杰，胡浩，麻信洛．流媒体技术入门与提高[M]．北京：国防工业出版社，2012

[8] Myerson J M．Computer speech： streaming technology[J]．International Journal of Network Managemen，2000，10(1)：51-55

[9] 孟昭林．流媒体系统嵌入式节点的设计[D]．哈尔滨工程大学，2008

[10] Xiong X，Song J，Yue G，et al．Survey：Research on QoS of P2P Reliable Streaming Media[J]．Journal of Networks，2011， 6(8)

[11] 冷波．流媒体自适应传输的研究与实现[D]．武汉理工大学，2007

[12] Liu Guozhu．Research of Self-adaptive Transmission Policy for Stream Media[A]．Proceedings of 2011 IEEE International Conference on Intelligent Computing and Intelligent Systems(ICIS 2011) vol.01[C]．IEEE Beijing Section，China、Guangdong University of Technology，China，2011

[13] 许峰．网络流量控制技术及应用研究[D]．重庆大学，2007

[14] 段雅涛．视频点播系统流量控制研究[D]．中国石油大学，2009

[15] 陈林亮．嵌入式流媒体客户端的研究和实现[D]．浙江大学，2007

[16] 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]．西安电子科技大学，2008

[17] 孙冬柏．流媒体技术及其应用[J]．信息技术，2005，11：138-141

[18] 蔡墩雄，唐依珠．基于嵌入式Linux的网络流媒体播放终端的研究设计[J]．福建电脑，2007，01：124-125+117

[19] 徐晟华．基于MPEG-4的嵌入式流媒体播放器的设计与实现[D]．江苏大学，2007

[20] 李子臣，王文静．流媒体：现状前景与障碍[J]．情报科学，2002，09：1005-1008

[21] 陈爽文. 流媒体技术综述[J]. 北京广播学院学报(自然科学版)，2003，01：58-64
[22] 庄捷. 流媒体原理与应用[M]. 北京：中国广播电视出版社，2013
[23] 董旭. 流媒体服务器中实时传输协议的研究和内核化实现[D]. 电子科技大学，2009
[24] 陶佳，李英祥. 基于 ARM 的嵌入式流媒体客户端软件的设计[A]. 四川省通信学会 2011 年学术年会论文集[C]. 四川省通信学会，2011
[25] 陈小平，王皖陵. Linux 下实时流媒体的编程实现[J]. 安徽工业大学学报(自然科学版)，2005，03：293-297
[26] 李旭，陈霞. 多媒体通信原理[M]. 北京：机械工业出版社，2006
[27] 李祥生. 多媒体信息处理技术[M]. 北京：高等教育出版社，2010
[28] 周鹏，汤银焕，黄秋元，杨纪锋，王福堂. 基于 RTP/RTCP 的音视频 WiFi 传输系统的设计[J]. 武汉理工大学学报(交通科学与工程版)，2011，04：802-804+808
[29] 刘浩，胡栋. 基于 RTP/RTCP 协议的 IP 视频系统设计与实现[J]. 计算机应用研究，2002，10：140-143
[30] 章民融，徐亚锋. 基于 RTSP 的流媒体视频服务器的设计与实现[J]. 计算机应用与软件，2006，07：93-95
[31] 方群，王敏，吉逸. 基于 RTSP/RTP 的媒体点播服务器的设计与实现[J]. 计算机工程与设计，2006，01：4-6+108
[32] 王荣生. RTSP 协议在视频点播系统中的应用[J]. 计算机应用与软件，2004，11：47-48+92
[33] 李校林，刘海波，张杰,刘利权. RTP/RTCP，RTSP 在无线视频监控系统的设计与实现[J]. 电视技术，2011，19：89-92
[34] 王欣. 基于 HTTP 流化的流媒体自适应传输策略研究及实现[D].南京邮电大学,2012
[35] 张冲，杨灿，杨泽铨，黄辉泽. RTMP 协议在 P2P 流媒体系统中的应用[J]. 电视技术，2009，S2：189-191
[36] 姜浩然，徐林. 基于 RTMP 的流媒体服务器的研究[J]. 计算机与数字工程，2011，10：104-108
[37] 金达，叶庆伟，狄红卫. 基于 HLS 的流媒体播放系统的设计与实现[J]. 信息技术，2013，10：49-52
[38] 康亮. HTTP Streaming 技术发展趋势[J]. 电信网技术，2011，06：36-41
[39] 霍龙社，甘霞. 移动流媒体协议综述[J]. 信息通信技术，2010，04：6-13
[40] 朱恺，吉逸，储昊明. 嵌入式系统基础[M]. 北京：机械工业出版社，2012
[41] 何立民. 嵌入式系统的定义与发展历史[J]. 单片机与嵌入式系统应用，2004，01：6-8

[42] 李国辉，涂丹，张军．多媒体通信网络[M]．北京：人民邮电出版社，2010

[43] 晨曦，韩超，沈立，李江峰，陈渝．嵌入式系统教程[M]．北京：清华大学出版社，2013

[44] 张营，李鹏，陈立锋，巩永光．嵌入式系统发展综述[J]．电子技术，2008，06：74-77

[45] 周青云，王建勋．嵌入式系统的应用与发展[J]．工业仪表与自动化装置，2008，03：16-20

[46] 张清小．基于ARM11的无线远程监测和控制系统的设计与实现[D]．太原科技大学，2011

[47] 吴柳.基于S3C6410的嵌入式图像采集及压缩系统的研究[D].安徽理工大学,2012

[48] 飞凌嵌入式OK6410开发板硬件手册[OL]．[2011-11-25]．豆丁网，http://www.docin.com/p-294553343.html#documentinfo

[49] Wang J，Zhao H，Li P，et al． Analysis and Comparison of Five Kinds of Typical Device-Level Embedded Operating Systems[J]．Journal of Software Engineering & Applications，2010，3(1).

[50] 邵国金，郭玉东．Linux操作系统[M]．北京：电子工业出版社，2012

[51] Keir Thomas，Andy Channelle，Jaime Sicam．Beginning Ubuntu Linux From Novice to Professional[M]．Berkeley，CA：Apress，2009

[52] 冯贤全．基于Android和OpenGL的多媒体播放器研究[D]．电子科技大学，2012

[53] TCP协议（一）——TCP提供可靠性的方式和TCP数据包[OL]．[2011-10-10]．新浪博客，http://blog.sina.com.cn/s/blog_83e4ed0e0100z7z3.html

[54] 钱立．人脸识别门禁系统的网络智能化管理[D]．四川大学，2005

[55] tcp拥塞控制[OL].[2012-07-17].ChinaUnix博客,http://blog.chinaunix.net/uid-27122224-id-376910.html