学校代码：10286
分 类 号：TP311
密　　级：公开
U　D　C：004.4
学　　号：194352

工程硕士学位论文

基于 GStreamer 框架的视频处理子系统设计与实现

（学位论文形式：应用研究 ）

研 究 生 姓 名：**于向前**

导 师 姓 名：**孔佑勇 副教授**

周清　高工

申请学位类别 **工程硕士**　学位授予单位 **东 南 大 学**

工程领域名称 **计算机技术**　论文答辩日期 **2022 年 5 月 29 日**

研 究 方 向 **软件工程**　学位授予日期 **20　年　月　日**

答辩委员会主席 **翟玉庆**　评　阅　人 **盲审**

2022 年 6 月 3 日

東南大學

工程硕士学位论文

基于 GStreamer 框架的视频处理子系统设计与实现

专 业 名 称： 计算机技术

研究生姓名： 于向前

导 师 姓 名： 孔佑勇 副教授

周清 高工

DESIGN AND IMPLEMENTATION OF VIDEO PROCESS SUBSYSTEM BASED ON GSTREAMER FRAMEWORK

A Thesis Submitted to

Southeast University

For the Professional Degree of Master of Engineering

BY

YU Xiang-qian

Supervised by

Associate Prof. KONG You-yong

and

Senior Engineer ZHOU Qing

Southeast University-Monash University Joint Graduate School

Southeast University

June 2022

东南大学学位论文独创性声明

本人声明所呈交的学位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的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____日期：_____

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆、《中国学术期刊（光盘版）》电子杂志社有限公司、万方数据电子出版社、北京万方数据股份有限公司有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名：_____导师签名：_____日期：_____

摘要

新冠疫情以来，视频会议作为多媒体通讯技术的一个重要领域，在人们的生活当中发挥重要的作用。学校、企业、政府通过视频会议来远程交流，完成学习工作任务。随着视频会议的应用越来越广泛，对视频会议的性能、可维护性等方面提出了更高的要求。视频会议可维护性差，导致维护人员难以对视频流状态进行把控，当出现问题时，无法及时定位解决。视频会议性能低则会直接导致其所消耗的硬件资源更多，在有限的硬件资源下，难以召开规格更大的视频会议。视频处理子系统作为视频会议中进行视频流处理控制的关键模块，对提高视频会议可维护性、性能具有重要的作用。

针对上述问题，结合具体应用场景，本文基于 GStreamer 框架设计并实现了视频处理子系统。通过设计视频流节点与功能元件相结合的方案，实现了视频流处理控制过程。主要工作如下：

（1）设计实现一对一单画面视频流处理控制方案。基于 GStreamer 框架，通过设计实时传输控制协议 RTP（Real-time Transport Protocol）码流解码、视频解码、图像缩放、视频编码、RTP 码流编码等功能元件，并利用视频流节点控制功能元件，实现了高效可控的单画面视频流处理流程，提高了系统的可维护性。

（2）在单画面视频流传输的基础上，设计实现多画面模式，满足多方会议视频流传输的需求。通过添加多画面处理元件，实现了视频会议多画面拼接功能，并通过设计帧率控制策略来实现不同子画面合成后帧率的统一，以便于后续的编码。

（3）针对视频处理子系统中视频流处理的性能问题，基于 x86 架构硬件平台，设计了一种 YUV 颜色空间图像缩放优化策略。并通过单指令多数据流 SIMD（Single Instruction Multiple Data）技术对所设计优化策略进行实现，缩短了视频流处理过程中图像缩放的处理时间。

实际测试结果表明本文所设计的视频处理子系统功能完善，实现了单画面、多画面下的视频流处理控制功能。同时在保证图像质量的前提下，图像缩放优化效果显著，满足了实际的应用需要，达到了预期的目标，具有较高的工程应用价值。

关键字：视频会议，视频处理子系统，GStreamer 框架，单指令多数据流

Abstract

Since the spread of the COVID-19, video conference as an important field of multimedia communication technology, has played an important role in people's lives. Schools, enterprises, and governments use video conference to communicate, complete learning and work tasks remotely. As the application of video conferencing becomes more and more extensive, new challenges are brought forward to the performance and maintainability of video conferencing. The maintainability of video conferences is poor, which makes it difficult for maintenance personnel to control the status of video streams. When problems occur, they cannot locate and solve them in time. Low video conferencing performance will directly result in more hardware resources being consumed. With limited hardware resources, it is difficult to hold a large-capacity video conference. The video process subsystem, as the key module of video stream transmission control in the video conference, plays an important role in improving the maintainability and performance of the video conference.

Aiming at the above problems, this thesis implements a video process subsystem based on the GStreamer framework. The efficient and controllable video stream processing and control process is realized by designing a scheme combining video stream nodes and functional elements. The specific work is as follows:

(1) Design and implement a one-to-one single-screen video stream processing control scheme. Based on the GStreamer framework, by designing functional elements such as RTP stream decoding, video decoding, image scaling, video encoding, and RTP stream encoding, and using video stream nodes to control the functional elements, this thesis implements an efficient and controllable single-screen video stream process and improve the maintainability of the system.

(2) Based on single-screen video streaming, this thesis designs a multi-screen mode to meet the needs of multi-party conferences. By adding multi-picture processing components, this thesis implements the video conference multi-picture splicing function, designs a frame rate control strategy to achieve the unification of frame rates after compositing different sub-pictures for subsequent coding.

(3) This thesis designs a YUV color space image scaling optimization strategy based on the x86 hardware platform to solve the performance problem of video stream transmission in the video process subsystem. The optimization strategy is implemented through the single instruction multiple data stream SIMD (Single Instruction Multiple Data) technology and reduces the process time of image scaling.

The actual test results show that the video process subsystem designed in this thesis has perfect functions, realizes the video stream transmission control function under single-screen and multi-screen. At the same time, under the premise of ensuring image quality, the image scaling optimization effect is obvious, which meets the actual application needs and achieves the expected goals.

Keywords: Video conference, video process subsystem, GStreamer framework, SIMD

目录

图目录

表目录

缩略词表

缩写	英文名称	中文名称
MCU	Multi Control Unit	多点控制单元
SMC	Service Management Center	业务管理平台
RSE	Recording & Streaming Engine	录播服务器
IVR	Interactive Voice Response	互动式语音应答
RTP	Real-time Transport Protocol	实时传输协议
RTCP	Real-time Transport Control Protocol	实时传输控制协议
GK	Gate Keeper	网守
SIMD	Single Instruction Multiple Data	单指令多数据流
QoS	Quality of Service	网络服务质量
NALU	Network Abstraction Layer Unit	网络提取层单元
GPPs	General Purpose Process	通用处理器
PSNR	Peak Signal to Noise Ratio	峰值信噪比
SSIM	Structural Similarity	结构相似性
MTU	Maximum Transmission Unit	最大传输单元
MSWT	Media Switch System	媒体交换系统
SPS	Sequence Parameter Set	序列参数集
PPS	Picture Parameter Set	图像参数集

第一章 绪论

1.1 研究的背景和意义

多媒体通信技术作为多媒体技术与通信技术的有机结合经过多年的发展已被广泛地应用到社会的各行各业中。视频会议作为多媒体通信的重要应用，极大地丰富了现代人们的生活。视频会议系统是一种把音频、视频等多媒体从一个地域传送到另一个地域的通信系统[1]。当前视频会议系统主要有两种：基于硬件的视频会议系统和基于软件的视频会议系统[2]。基于硬件的视频会议系统通过使用特定的硬件设备来进行通信。如华为公司的 TE30 一体化高清视频会议终端，集成摄像头与麦克风，使用自身硬件能力进行视频编解码，能够满足高清视频会议的需求，这一类系统的优点是视频会议可靠性高，通常面向企业、政府等用户。基于软件的视频会议系统，使用计算机内置性能来进行视频编解码功能。优点是使用简单方便，造价低。常见的有腾讯会议、钉钉、ZOOM 等平台，常用于个人之间的视频通信。

基于硬件的视频会议系统厂商常见有华为、宝利通、中兴等。宝利通是一家提供语音、视频和数据等媒体协作应用的跨国企业，宝利通公司的 Polycom HDX8000 视频会议系列产品支持 IPV4 及 IPV6 双栈，以实现复杂网络场景下的视频会议通信。支持 H.323 协议和 SIP 协议，满足 1080P、720P、4CIF、CIF 等多种视频格式，并支持 H.235 加密协议[3]。中兴目前最新的视讯平台 M930，是一款面向云化推出的电信级视讯服务器，支持丰富的音频、视频编码格式，支持软硬件视频会议终端接入，满足各类企事业单位和运营商用户的视频会议需求。华为技术有限公司面向政府、交通、安全、金融、大企业、中小企业等领域，提供高临场感的远程会议、桌面及移动视频接入、企业流媒体应用等场景下的视频会议整体解决方案。先后推出了 96 系列 MCU、98 系列 MCU 和 CloudMCU（Cloud Multi Control Unit，云化多点控制单元），实现媒体处理与转发功能[4]。同时推出多种终端，如 TE 系列、IdeaHub 系列、Box 系列产品，形成完整的视讯解决方案。

随着新冠疫情的到来，人们逐渐开启了线上办公的新模式，在线上活动越来越频繁的同时，视频会议作为人们线上交流的一种方式逐渐流行起来[5]。同时，视频会议在越来越多的领域得到了应用，包括在线教学、在线办公、在线视频聊天等领域，视频会议在人们日常生活中发挥着重要的作用。

本文的主要工作背景来自于实习期间参与的 MCU 项目。该项目基于云化的需求，设计实现功能强大的企业云通信融合媒体平台，可以传输多种媒体内容，包括视频、音频和数据等内容，帮助企业快速构建自己的云通信业务。同时，该

平台支持多种终端设备进行使用，包括个人计算机、手机等设备，来实现统一无缝的沟通协作。

本文的主要研究内容为基于 GStreamer 框架的视频处理子系统设计与实现。视频处理子系统作为视频会议中视频流处理控制的关键子系统，主要进行视频流的处理操作，并能够根据会控下发的消息及时对视频流进行控制。因此在实际的应用中，若能优化视频流处理控制流程，将有助于提高 MCU 的可维护性，方便维护人员及时定位解决问题，同时也将提升系统的性能。

1.2 研究现状

1.2.1 视频会议系统的发展现状

视频会议系统的发展主要分为五个阶段：模拟视频会议系统、数字视频会议系统、基于 IP 网络的视频会议系统、高清视频会议系统和基于人工智能的视频会议系统。

（1）模拟视频会议系统

在模拟视频会议阶段，视频信息主要由模拟信号进行发送，并且只能传输黑白图像，且会议参与方数量、占用带宽等有较大限制。同时，各视频会议系统没有统一的标准，都是基于自身的技术进行研发，因此这一阶段视频会议系统发展速度较慢。这一时期代表的视频会议系统主要有美国贝尔实验室 PicturePhone MOD-1 视频电话[6]、英国 BT 公司的 1MHZ 带宽的黑白会议电视会议。其中贝尔实验室的可视电话是世界上第一台可视电话。该阶段系统只能通过模拟信号线路提供点对点的视频会议业务，并且由卫星信号分时传送语音与图像，导致造价高昂，没有得到广泛的应用[7]。

（2）数字视频会议系统

在上世纪 80 年代初，随着集成电路、数字信号处理、数字图像压缩技术的发展，视频会议的理论研究及实际应用得到迅速发展，视频会议系统进入到了数字视频会议阶段。数字视频会议频带占用窄、图像质量好，并且逐步形成了统一规范的通信标准，这一阶段的标志是国际电报电话委员会出台了 H.100 等一些标准。这一阶段下，在局部地区形成了视频会议网，特别是国际电视会议的统一标准为视频会议的普及创造了有利条件。在 1990 年，国际电信联盟远程通信标准化组织（International Telecommunications Union，ITU-T）发布了第一代会议电话系统标准化协议 H.320[8]，该协议对视频会议领域的标准化做出了重大贡献。同时由于 H.261 视频编解码协议的出现[9]，为不同厂商确定了一个统一的视频编解码标准，促进了视频会议的发展。但由于这一阶段视频会议主要用于专网，应用范围小，难以得到大规模的普及。

（3）基于 IP 网络的视频会议系统

90 年代中期，随着计算机网络的发展，视频会议向基于 IP 网络的方向发展。这一阶段的标志是 ITU-T 推出适用于分组交换网络中的 H.323 协议[10]，该协议成为了全球范围内数据 IP 电话及企业内部 IP 电话发展的基础，标志着视频会议系统进入到互联网时代。H.323 协议提供了丰富的多媒体通信元素，如终端、MCU、网关等。互联网工程任务组（Internet Engineering Task Force，IETF）同时制定了实时传输协议（Real-time Transport Protocol， RTP）、实时传输控制协议（Real-time Transport Control Protocol，RTCP）[11]和视频编解码标准 H.263[12]。RTP 与 RTCP 协议是一种专门应用于实时视频流传输控制的协议，促进了互联网时代视频会议的发展。H.263 协议的出现，相比于 H.261 格式，输入有多种格式，并提高了视频的压缩比。同时使用半像素预测的方式，进一步促进了更高码率视频的传输。但该阶段基于 IP 网络的视频会议由于受网络因素的影响较大，难以保证视频会议的服务质量。

（4）高清视频会议系统

进入 21 世纪，随着网络带宽的增长、视频编解码协议的成熟，基于互联网的视频会议系统能够传输更高清晰度的视频。2003 年，ITU-T 和国际标准化组织（International Organization for Standardization，ISO）共同发布了视频编解码标准 H.264[13]。该协议是一种面向块、基于运动补偿的视频编码标准，并成为高精度视频录制、压缩和发布的常用格式之一[14]。能够在网络带宽有限的条件下依旧提供良好的视频质量，并成为了现代视频编解码标准的一个基石。2008 年 KEDACOM 发布首款 1080p 高清视频会议系统，标志着视频会议系统进入了高清时代。

（5）基于人工智能的视频会议系统

随着人工智能时代的到来，如今视频会议系统的发展又进入了一个新的阶段。借助人工智能技术，逐渐将人脸识别、画质增强、语音增强、语音去噪、图像去噪等应用到视频会议当中，以提升视频会议的质量[15]。同时，利用人工智能优化视频编解码算法也在逐渐发展，这也是未来视频会议系统发展的一个主要方向。

随着视频会议系统的发展，对视频流处理过程的可维护性、性能等方面提出了更高的要求。针对视频流处理过程中的可维护性，使用合理的多媒体框架具有很高的必要性。同时针对视频流处理的性能问题，使用合理有效的优化方法具有很高的应用价值。

1.2.2 多媒体框架的发展现状

多媒体框架是一种在电脑上处理多媒体并经网络传播的软件框架。多媒体框架提供了直观的 API 和模块化的架构，并且具有易于添加的视频、音频和容器格

式，对传输协议具有较好的支持。常见的多媒体框架有 Phonon[16]、DirectShow[17]、Media Foundation[18]和 GStreamer[19]等。

Phonon 原本是 KDE 4 的开放原始码多媒体 API，后与 Qt 合并，作为 Qt 开发环境下音视频播放的接口[20]。但由于 Phonon 只提供一套接口，并不实现具体后端功能，需要 Phonon 后端插件实现具体功能，对于 Qt 不提供默认插件的后端来说则存在较大问题。

微软公司开发的多媒体应用框架 DirectShow 最初是一套应用在 windows 平台上的开发包，并逐步扩展为 windows sdk 下的一部分[21]，但由于其对第三方播放器功能的限制及应用平台的局限性，导致其应用范围有限，发展缓慢。

Media Foundation 是微软公司针对 Directshow 为主的旧式多媒体系统的一种替代方式，能够提供更高质量的音视频播放与数字版权管理。但是其仅支持 windows vista 及以后更新的系统，使其支持平台受限，目前仅作为 windows 下音视频开发的多媒体开发库。

GStreamer 是一种跨平台的、基于管道（pipeline）的多媒体框架。使用插件化的思想，以元件为核心，将视频流处理过程功能分解为一个一个的元件，提高了系统的可维护性。GStreamer 最初由 Erik Walthinsen 在 1999 年创建，2001 年发布第一个版本 0.1，后由 Fluendo 公司使用并编写出一个流媒体服务器 Flumotion，并逐步得到各大公司如诺基亚、摩托罗拉、英特尔的采用，成为目前主流的多媒体框架。

目前针对 GStreamer 的研究一方面体现在框架的平台泛化研究。最初 GStreamer 利用 C 语言进行实现，文献[22]利用 Python 语言实现了 GStreamer 的基本框架，使其能够在深度学习领域发挥更大的作用。另一方面应用 GStreamer 框架进行具体的视频会议的设计实现也是 GStreamer 框架目前的主流应用。文献[23]在 GStreamer 框架的基础上，使用树莓派进行一个小型视频会议系统的搭建。文献[24]在 GStreamer 的基础上，利用 FFmpeg 进行流媒体的传输设计与实现，两者相互配合，进行高效视频会议系统的实现。文献[25]在 GStreamer 框架的基础上进行目标检测嵌入式系统的应用。根据对 GStreamer 框架的研究可知，由于 GStreamer 框架的可维护性好，跨平台等优点，因此 GStreamer 框架也成为视频会议开发领域常用框架。

1.2.3 视频处理子系统优化方法的研究现状

视频处理子系统的优化通常体现在两个方面：视频编解码算法优化和视频流处理实现过程优化。

（1）视频编解码算法优化

当视频会议发展到数字视频会议阶段后，各种音频和视频压缩的技术也逐渐发展了起来。由于网络带宽有限，将整个数字视频放在网络上传输显然不现实，因此需要对其进行数据压缩。但压缩必须满足一定的平衡，需要将视频的质量、码率、编解码算法复杂度、数据丢失、错误的鲁棒性、延时等综合进行考虑。如H.264 协议可以说具备了一定的平衡性，才能得到普及。H.264 协议不仅具有良好的压缩性能，也对网络具有较好的适应性，具有传输的可靠性，因此得到广泛应用。

对随着人工智能时代的到来，利用深度学习对视频编解码算法中的帧内预测、帧间预测、去宏块滤波等算法进行优化成为一种趋势。如文献[26]利用深度神经网络，对视频编解码中的环路滤波过程进行优化，提升了性能。文献[27]利用神经网络作帧内预测算法的优化。但利用深度学习对视频编解码算法优化效果对比现有视频编解码算法提升有限，在实际应用中存在应用复杂度高的问题。

（2）视频流处理实现过程优化

另一方面，对视频流处理实现过程进行优化，往往能取得较为明显的优化效果，且在实际应用中能够快速落地，实现性能的提升。

视频流处理的过程中，涉及到图像的缩放、视频的编解码等过程。同时这些步骤也是视频流处理过程中最为耗时的步骤。因此，可以利用 SIMD 技术，根据不同的硬件平台如 ARM、x86 来进行特定的优化。

SIMD 是利用相关硬件能力，使用并行化计算的指令进行多个值的同时计算，从而来提高计算的速度[28]。SIMD 在 1966 年首次出现在大规模并行计算机 ILLIAC IV[29]上，后在 intel x86 架构上的 MMX 指令集上得到进一步应用。因为 SIMD 能够同时对长度在 2-16 字长的向量进行操作运算，因此逐步应用到图像、视频处理领域。在 1995 年，SIMD 技术首次应用在 CIF 格式下 MPEG-1 视频实时解码过程中。此后，SIMD 逐步应用在后来出现的 MPEG-2、H.264、H.265 等视频编解码协议当中。在 H.264 协议中，SIMD 可以应用在常见的编解码算法中，如帧内预测、帧间预测、去宏块滤波等算法中。文献[30]在 SIMD 的基础上实现了去宏块滤波算法算法性能的提升。文献[31]利用 SIMD 进行精简指令集 RISC-V 的扩展。文献[32]利用 SIMD 进行多核系统低功耗 H.264 解码的实现。文献[33]利用 SIMD 进行视频解码过程 YUV 图像的性能优化。随着不同硬件平台能力的提升，利用 SIMD 进行视频流处理的优化也具有更好的效果。

目前主流的基于 H.264 协议的视频编解码库有 FFmpeg、OpenH264、X264 等。X264 是一款由 VideoLAN 组织开发的一款编码器，并针对不同架构的硬件平台使用 SIMD 进行了优化，因此显著提升了视频编解码器的性能[34]。FFmpeg 也是一款进行视频处理的软件，实现了音视频的编解码库，具有广泛的应用。

OpenH264 库是由思科公司开发的开源的用于 H.264 编码和解码库，该库同样也使用了 SIMD 进行了 H.264 视频编解码过程的性能优化，并具备良好的性能。

通过以上分析可知，在视频处理子系统中视频流处理的过程中，SIMD 通过并行化计算，提高了数据处理的运算速度。因此，本文将选用 SIMD 技术，进行视频流处理过程中 YUV 图像缩放的性能优化。

1.3 论文研究目标与内容

本文的主要研究目标是设计实现视频处理子系统，以完成视频会议中视频流的处理控制。本文结合 MCU 项目背景，在进行视频处理子系统的上下文分析后，确定视频处理子系统的需求。然后根据系统需求，通过使用多媒体框架 GStreamer，来设计实现视频处理子系统。具体的内容主要有以下三个方面：

（1）对视频处理子系统中的最基本的单画面视频流处理流程进行设计实现。利用 GStreamer 元件化的思想，设计 RTP 解码、视频解码、图像缩放、视频编码、RTP 编码等功能元件，并通过视频控制流节点进行元件的控制操作，快速构建视频流处理 pipeline，实现单画面 RTP 码流的端到端处理。

（2）在单画面视频流传输的基础上，进行多画面模式的设计实现。设计实现多画面视频流的传输控制流程，对其中的多画面合成元件进行设计与实现，实现多画面图像拼接功能。为保证图像合成后帧率的统一，设计实现多画面合成帧率控制策略。

（3）针对视频处理子系统中视频流传输的性能问题，基于 x86 架构的硬件平台设计 YUV 图像缩放优化策略，并通过 SIMD 技术使用汇编指令集对缩放优化策略进行实现。

1.4 论文组织结构

本文首先对课题的研究背景和现状进行了分析，确定视频处理子系统中需要用到的技术，并对视频处理子系统中涉及到的技术背景进行介绍。然后进行视频处理子系统的需求分析，根据需求对视频处理子系统进行设计，并给出具体实现。最后对整个系统进行测试，测试系统的功能需求与非功能需求。本文具体的组织结构如下：

第一章：绪论。主要介绍本文的研究背景和意义、项目的来源、研究目标、研究内容以及论文的主要工作和组织结构。

第二章：视频处理子系统的相关技术介绍。主要介绍 GStreamer 多媒体框架、流媒体相关技术以及在硬件平台上进行性能优化的 SIMD 技术。

第三章：视频处理子系统的需求分析。分析视频处理子系统的上下文，并确定视频处理子系统的功能需求与非功能需求。

第四章：视频处理子系统的详细设计。介绍系统的整体框架以及各功能需求设计方案，并设计缩放优化策略。

第五章：视频处理子系统的实现。根据功能需求设计方案，给出对应的实现方式。同时，使用 SIMD 对设计的缩放优化策略进行实现。

第六章：系统测试。对整个系统进行测试，利用灰盒测试用例与白盒测试用例对整个系统的功能与非功能需求进行测试，并对测试结果进行分析。

第七章：总结与展望。总结本论文的主要工作内容以及未来研究与改善的地方。

第二章 相关技术

本章主要对视频处理子系统中所涉及到的相关技术进行介绍。首先是流媒体技术及其相关协议，包括实时传输协议 RTP 和实时传输控制协议 RTCP。其次是 GStreamer 框架的相关技术，包括 GStreamer 的基本概念和框架基础。最后是 SIMD 的相关技术，包括 SIMD 概念和主要原理。

2.1 流媒体技术

2.1.1 流媒体技术概念

流媒体技术是指将一连串的多媒体数据进行压缩后，经过网络分段发送数据，实现在网上实时传输多媒体数据以及在线进行观看的一种技术与过程[35]。在流媒体技术出现之前，观众如果想观看网上的多媒体数据，就需要将其完整地下载到本地电脑上，受网络带宽和本地存储空间的限制较大。而流媒体技术可以使多媒体数据包进行流式传输，通过服务器向客户端实时传输音视频数据包，客户端在收到数据包后，只需经过短时间的缓冲延时即可进行媒体数据地播放，实现以较小的缓存来进行多媒体数据流地实时播放。

2.1.2 流式传输原理

流式传输需要在网上进行，但由于受到网络带宽等因素的影响，多媒体文件需要首先进行压缩，减小文件大小。对于一路高清视频信号，如果不进行压缩，其需要的网络带宽达到 1Gbit/s。因此基于网络带宽的限制，有必要对多媒体视频进行压缩。在压缩过程中，需要满足两个基本的要求，一是视频编码具有足够的压缩比，使得网络带宽可以承受。二是视频压缩后必须保持一定的压缩质量，防止失去了视频会议的本质需求。对于视频质量的评估，一种方式是基于主观质量，即人主观上从自己的视觉角度进行判断，另一种是客观质量，即通常用信噪比（S/N）来表示[36]，常见的评估图像的指标有峰值信噪比（Peak Signal to Noise Ratio，PSNR）和结构相似性（Structural Similarity，SSIM）[37]，也有利用神经网络进行评估的方式，如文献[38]通过 BP 神经网络来构建模型对视频质量进行评估。

视频压缩一方面是对基本的图像格式进行压缩，另一方面是针对视频序列进行压缩。常见的图片格式为 RGB 格式，为一种对颜色进行编码的颜色空间，即用红绿蓝三色来进行叠加，来表示其他颜色，这种方式由于其通用性与方便性，许多图像处理任务都是以 RGB 形式进行的。但是这种颜色空间也有其缺点，那就是占用空间大，对于一张 8bit 深的 4x4 的 RGB 图像，其大小为 384bit。因此在视频会议领域，对于带宽要求较高。因此需要在尽量不降低视频质量的前提下，

尽可能使用占用空间小的颜色编码方式，YUV 的颜色空间因此发挥了重要的作用。YUV 是利用人眼对亮度信息较色度信息更为敏感，因此在储存色彩的时候，对亮度信号全部采样，对色度信号部分采样，因此实现图片大小的压缩。Y 代表亮度信号，UV 代表色度信号，常见的 YUV 格式有 YUV444、YUV422、YUV420 等[39]，其中 YUV420 代表 4：2：0 的采样方式，4 个 Y 分量公用一组 UV 分量。具体采样方式如图 2-1 所示。

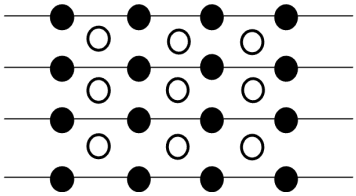

图 2-1 YUV420 格式采样图

因此，对于一张 8bit 深的 4x4 的 RGB 图像，其转换为 YUV420 格式下的大小为 16*8+4*8 = 160bit，相比于 RGB 格式，大小有明显的下降，因此，在视频会议领域，图像格式一般采用 YUV 颜色空间。

对于视频序列进行压缩，常见的视频编码格式有 H.264、HEVC、VP8 等。其中，在视频会议领域，H.264 协议为常见的视频编解码协议。H.264 协议定义了视频比特流的句法格式和该比特流的解码方法，使得只要满足该句法下的不同厂商的编解码器能够互通。H.264 协议主要利用视频流序列的特点，即当前帧的图像内容与其相邻帧的内容关联很大，存在大量冗余信息，因此可以提取部分关键帧，利用关键帧和其与附近帧的残差信息还进行还原，从而起到视频压缩的目的。因此在 H.264 协议中，划分了 I 帧，P 帧和 B 帧，I 帧为关键帧，只采用帧内预测的方法进行压缩，而 P、B 帧是利用关键帧与帧间预测算法进行编码，只保留了较少的信息，使得视频序列编码后大小得到大幅缩减。

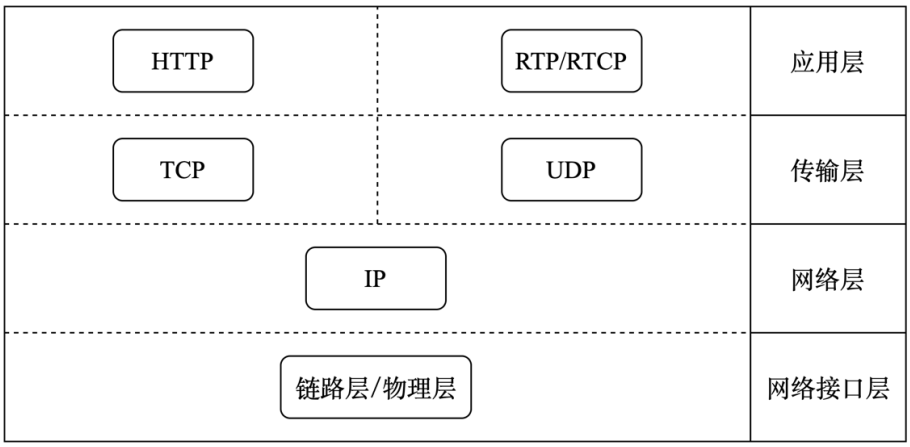

图 2-2 流媒体协议网络层次示意图

在多媒体进行编码后，通过网络进行传输，在 TCP/IP 族的 IP 网络结构下，多媒体数据一般在应用层和传输层需要使用合适的协议，应用层常见的协议有 HTTP、RTP、RTCP 等，传输层的主要协议有 TCP 和 UDP 协议。目前网上多媒体流基于不同网络协议的传输方式有两种，一种是基于 HTTP/TCP 的传输，一种是基于 RTP/UDP 的传输[40]，其对应的网络层次架构如图 2-2 所示。根据 RTP 报文的格式以及其在网络层次中的位置，可以知道 RTP 协议是面向非连接的协议。

第一种方式通常对应顺序流式传输，其最主要的特点就是用户只能观看以及下载好的部分，而不能调到还未下载到的部分。视频为无损压缩，主要适合一些高质量的短片段如广告等使用，由于 HTTP 的无状态特性与 TCP 的非实时特性，使得这种传输方式在视频会议等应用领域具有较大的缺陷，因此目前主流的视频会议系统中视频流的传输都是基于 RTP/UDP 协议的，即实时流式传输方式。发送端的多媒体数据经过数据压缩后，通过 RTP 打包，通过 UDP 协议进行数据包的传输，在网络传输中，根据 RTCP 协议接受者报文 RR 来得到网络状态的反馈，并结合上层应用进行服务质量（Quality of Service，QoS）反馈控制[41]。

根据多媒体数据类型的不同，对于视频压缩数据，根据 H.264 视频编解码协议的特点，接收端在收到 RTP 包后，根据包头信息判断其携带的视频帧信息，如果为参考帧 I 帧，则将其保存在缓冲区中，后续的 P 帧、B 帧即可根据缓冲的 I 帧信息进行解码，由此来实现使用较小的缓冲区来实现实时的视频会议[42]。其基本的传输流程如图 2-3 所示。

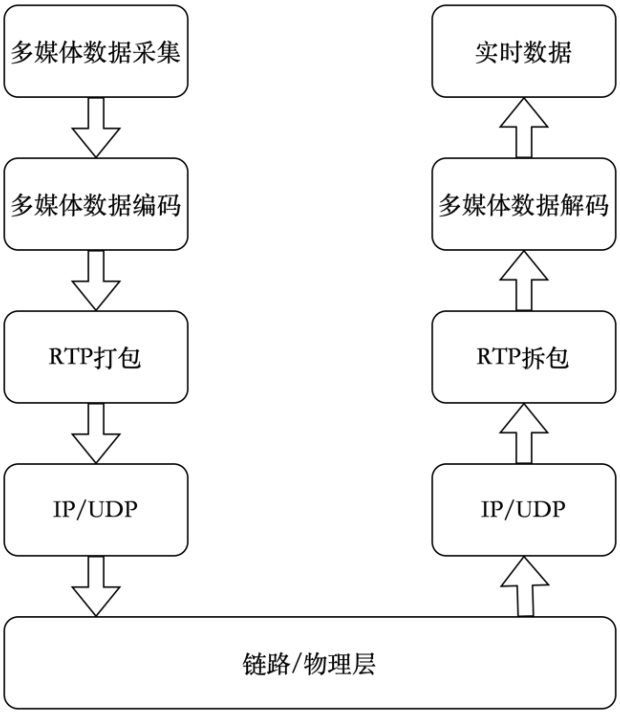

图 2-3 多媒体数据流传输过程示意图

2.2 实时传输协议

在实时传输协议 RTP 协议推出之前，多媒体数据流的传输都是基于 HTTP 与 TCP 协议，但这种方式存在实时性差的问题，于是 IETF 音视频传输工作组在 1996 年首次提出 RFC1889 文档描述了 RTP 协议，并在 2003 年在 RFC3550 文档中进行完善[43]。通过 UDP/RTP 协议，来实现实时视频流的传输，在实际应用中，RTCP 协议与 RTP 协议密切相关，并相互配合。

2.2.1 RTP 数据协议

实时传输协议 RTP 是一种在 IP 网络之上主要进行音视频传播的协议，它专门为端到端、实时的流媒体传输而设计。RTP 协议主要负责传输实时流媒体数据。RTP 协议直接依赖 UDP 协议来实现传输的实时性和吞吐量[44]。

RTP 数据包主要有包头（Header）和有效载荷（Payload）两部分构成，在实际应用中，有效载荷中主要内容为经过压缩后的多媒体数据。在基于 H.264 协议的视频传输中，有效载荷内容为网络提取层单元（Network Abstraction Layer Unit，NALU）[45]，包含了经过压缩后的视频帧的信息。

在 RTP 的包头中，前 12 字节是固定的，后面的参与元标识符（Contributing SouRCe，CSRC）是可选的，仅当前面的 CC（CSRC count）字段不为 0 时出现，表示产生 RTP 数据包的所有源，RTP 报文头格式如图 2-4 所示。

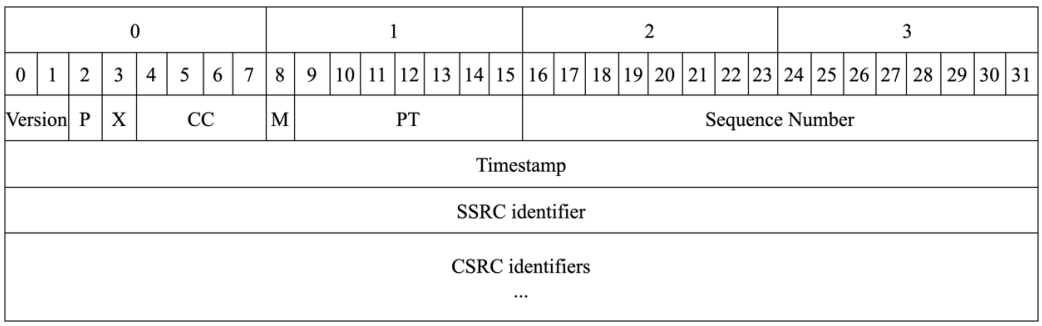

图 2-4 RTP 报文头格式示意图

上述字段的具体含义如下表 2-1 所示。

表 2-1 RTP 报文字段表

字段名	长度（bit）	字段含义
版本 Version(V)	2	标识 RTP 版本号，当前版本号为 2
填充标志 Padding(P)	1	如果该字段设置为 1，报文中就携带 1 位或多位的 0 填充字节，来使报文达到规定的长度，但这些填充位不属于有效载荷
扩展位 eXtention(X)	1	如果该字段被设置为 1，表示报文头部还有一个扩展头部
CSRC 计数 CSRC Count(CC)	4	表明 CSRC 标识的数目，CSRC 紧跟在固定头部后，表示 RTP 数据包的来源，RTP 允许在一个会话中存在多个数据源。
标志位 Maker(M)	1	含义根据具体协议而定，在传输视频流时表示每一帧的开始
载荷类型 Payload Type(PT)	7	表示 RTP 有效载荷的类型，接受端据此判断媒体类型
序列号 Sequence Number(SN)	16	表示包的序列号，发送端在发送完一个 RTP 包后就将该字段值加一，接收端在收到后可以判断包是否存在丢失以及是否按顺序发包
时间戳 Timestamp(T)	32	该字段记录了 RTP 报文中第一个字节的采样时刻，接受端可以利用时间戳去除由网络引起的抖动，并为播放提供同步功能
同步源标识符 Synchronization source identifier(SSRC)	32	用于识别同步源，在新数据包开始时，发送端随机分配的号码
参与源标识符 Contributing source identifier(CSRC)	(0-15)*32	根据 CSRC Count 得到具体数量，最多为 15 项，每一项大小为 32bit

2.2.2 RTCP 控制协议

实时传输控制协议 RTCP 是与 RTP 协议相互配合的，RTP 协议只负责数据流的封装传输，而 RTCP 协议则负责对传输提供可靠的保证，对其传输状态进行监测，如对服务质量进行反馈，如丢包的数量、抖动等。并且对会话进行控制，使用 RTCP 的终止标识（BYE）分组告知参与方会话的结束[46]。另外，RTCP 也可用于媒体间同步，如对不同的视频流与音频流进行时间上的同步。

值得注意的是，每次启动一个 RTP 会话时，将同时占用两个相邻的端口，分别供 RTP 与 RTCP 使用，就如同 HTTP 协议默认占用 80 端口一样，RTP 协议默认为 5004 端口，而 RTCP 协议则相应地使用与其相邻的 5005 端口。

RTCP 协议的主要功能主要依赖其报文类型来实现的，其主要报文有发送端报文（Sender Report，SR）、接收端报文（Receiver Report，RR）、源描述项（Source

DEScription Items，SDES）、参与结束标识（Indicates End of Participation，BYE）和特定应用功能（APPlication Specific Functions，APP）。

其中 SR 是在发送端向本次会话参与方以组播的形式周期地发送，一次会话包含若干 RTP 流，一个 RTP 流又包含若干 RTP 分组，每发送一个 RTP 流，就会发送一个 SR 报文，用于提示 RTP 流内的分组情况。因此，SR 报文中包含 RTP 流的 SSRC、RTP 流包含的分组数和 RTP 字节数等，类型号为 200。

接收端报文 RR 是指接受端向本次会话的所有参与方周期性地以组播的形式发送报文，其内容包括所收到 RTP 流的 SSRC、丢包率、最后一个 RTP 分组的序号等。RR 可以使当前会话中所有的参与方了解网络的状况以便自适应地进行调整，类型号为 201。SDES 主要作为有关标识信息的描述，如包括用户名、电话号码、邮箱等信息，其类型号为 202。BYE 作为参与结束标识，指示关闭数据流。APP 则作为应用程序自己定义的功能，并作为扩展以提高灵活性。虽然报文具体内容不同，但是其拥有相同的 RTCP 报文头，RTCP 的报文头格式如图 2-5 所示。

<table>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="2">Version</td><td>P</td><td colspan="5">RC</td><td colspan="8">PT</td><td colspan="16">Length</td></tr>
<tr><td colspan="32">SSRC identifier</td></tr>
</table>

图 2-5 RTCP 报文头格式

其中 V 代表版本号，为 2，P 为填充位，RC 表示报文接受报告的数量，PT 表示载荷类型，如载荷类型为 200,表示为发送端报文 SR。length 表示本报文的长度，SSRC 表示报文的同步源。根据具体的载荷类型，在报文头部后继续添加其他内容使之成为有具体功能的报文，从而实现 RTCP 的作用。

2.3 GStreamer 框架

GStreamer 是一个用于构建媒体处理组件的开源库，利用该框架可以快速实现多媒体的应用。GStreamer 核心的设计原则是利用元件（elements）进行功能模块的封装，再将这些功能模块安装到管道（pileline）上，从而使多媒体数据流可以按照流式的方式通过管道并进行一步步处理[47]。元件可以由框架自身提供也可由外部的第三方提供。尽管 GStreamer 现在被认为是处理多媒体数据流的框架，其实它还可以处理任意类型的数据流，下面本文将详细介绍 GStreamer 框架的详细内容。

2.3.1 元件

元件是 GStreamer 框架中最基础也是最重要的内容，一个元件是一个独立的功能模块，可由开发者自己定义相关的内容，并使用特定的接口调用。将多个元件进行连接，就形成了一个管道，从而可以完成特定的功能。如图 2-6 所示。

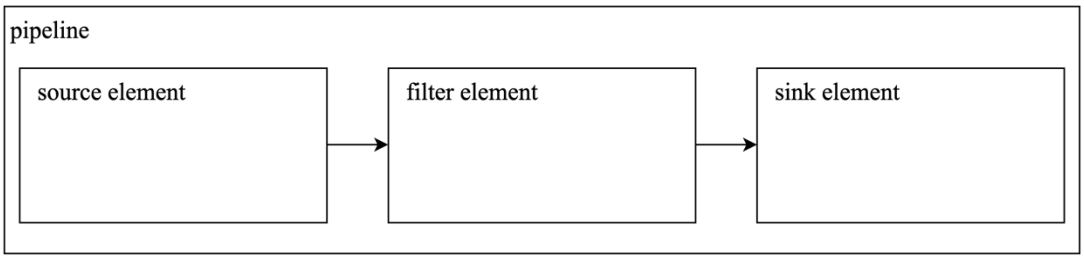

图 2-6 GStreamer 管道图

如在视频处理系统中，经常用到视频编码的元件，其主要功能就是将图像数据进行编码，压缩大小，如基于 H.264 协议的编码元件输入为图像，输出则是 H.264 码流。按照元件的功能不同，常用的元件主要分为三类：数据源元件（Source Element）、过滤器元件（Filter Element）和接收器元件（Sink Element）。

（1）数据源元件

作为数据的发送端，向下游发送数据，如在视频会议系统中，视频采集系统通常作为数据源，向下游发送采集好的视频数据。

（2）过滤器元件

过滤器元件通常位于整个管道的中间，主要进行数据流的处理操作，过滤器具体的功能可以根据实际需求进行自定义，具有较高灵活性。

在视频处理系统中，一般 H.264 编码作为一个独立的过滤器元件，接收视频图像数据，进过编码处理，然后通过 src pad 发送给下游。

（3）接收器元件

作为数据的接收端，接收器元件通常位于一个管道的最末端，通常只有一接收衬垫，最常见的应用是在磁盘写入和音视频播放等。

在介绍完元件的主要类型后，还需关注元件的状态，在一个元件进行创建后，不会进行任何操作，需要改变元件状态。元件共有四种状态：默认状态、准备状态、暂停状态和运行状态[48]。

（1）默认状态（GST_STATE_NULL）:该状态会回收该元件占用的所用资源。

（2）准备状态（GST_STATE_READY）:元件得到所需的全局资源，这些资源将被通过该元件的数据流使用。

（3）暂停状态（GST_STATE_PAUSE）:在这种状态下，元件以及对流进行了处理，但此刻暂停了处理，因此该状态下可以改变流的位置信息，读取或处理数据，但时钟是禁止运行的。

（4）运行状态（GST_STATE_PLAYING）:时钟在该状态下可以运行，其他与暂停状态一致。

元件的状态可以通过函数 gst_element_set_state()来更改，但是元件之间状态的迁移需要遵循一定的顺序，如图 2-7 所示。即当把元件从 NULL 状态切换带 PLAYING 状态时，也需要经过 READY、PAUSE 状态，系统会自行进行中间状态的切换[49]。

图 2-7 元件状态切换图

2.3.2 通信机制

GStreamer 提供部分通信机制来实现元件之间、管道和应用之间的数据交换，主要有四种机制：事件（events）、查询（queries）、消息（messages）和缓冲（buffers）。一个 GStreamer 应用直白来说，就是将元件组成管道，然后启动管道[50]，应用可以通过向元件发送事件和询问来影响管道的操作。具体机制如图 2-8 所示。

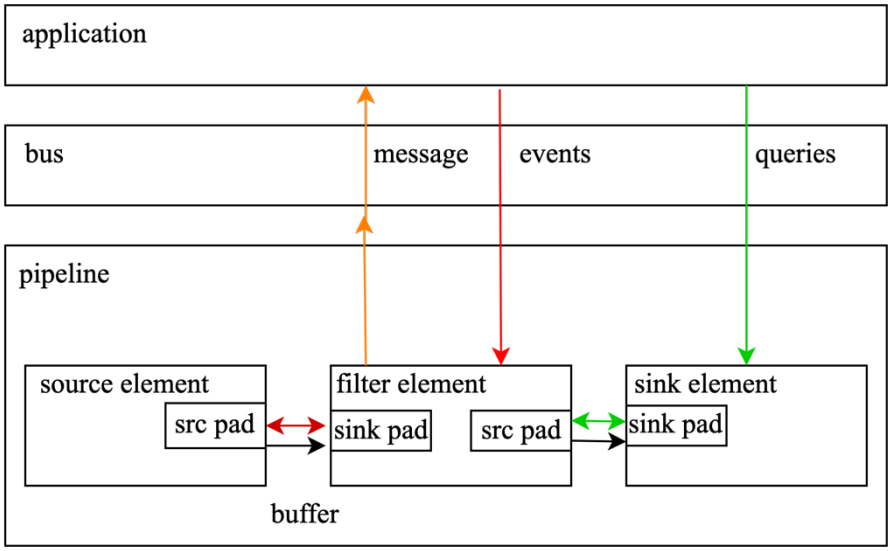

图 2-8 GStreamer 通信机制示意图

（1）缓冲

缓冲主要用于管道中的元件之间进行数据流的传递，如下游元件在接受数据流时需要在内存中开辟一块缓冲用于缓冲数据，缓冲只能用于上游向下游数据流的传递。

（2）事件

事件是元件之间、应用到元件传递的对象，事件一般可以从上游传到下游，下游事件也可以同步到数据流中。

（3）消息

消息是元件通过消息总线将收集到的消息传递给应用，常见的消息类型有错误、状态转换、缓冲状态和数据流结束等，消息监听有两种方式，一种是使用 Glib 的主循环，使用监听器 gst_bus_add_watch()/gst_bus_add_signal_watch()进行监听，还有一种方式是自己主动获取总线消息，这种方式并不能及时获得消息，需要使用 gst_bus_peek()/gst_bus_poll()来获得。

（4）查询

与事件类似，查询通常用语上下游的元件之间交流，与事件是提供信息不同，询问是针对某一类特定消息对元件进行询问，如元件的状态、数据流的长度、元件能接受的 caps 等，查询具有时间敏感性，即查询的信息可能有实效性，因此当一个元件无法处理查询时，会将该查询向上下游传递。

2.3.3 箱柜

箱柜（bin）在 GStreamer 中是一种容器类的元件，它内部可以装载元件，因为箱柜本身就是元件，因此普通元件的操作也适合箱柜，当箱柜的状态改变时，其内部的所有元件的状态也一起自动改变。使用箱柜可以简化系统的逻辑结构图，方便管理，统一进行元件的消息收集和传递。

2.4 SIMD 相关技术

2.4.1 SIMD 简介

单指令多数据流（SIMD），就是利用一条指令进行多条数据流的同时操作，从而极大地提升运算的效率。在图像处理领域，因为涉及到大量像素值的操作，如果使用传统的 C 语言对像素值进行操作将大大降低效率，因此可以利用 SIMD 的并行化操作来提升性能[51]。目前，在通用处理器（general purpose process，GPPs）领域，针对基本的指令集架构（instruction set architectures，ISAs），结合 SIMD 的技术，形成了 SIMD 扩展 ISAs，具体在 GPPs 上 SIMD 的拓展如下表 2-2 所示[52]。

表 2-2 通用处理器上 SIMD 指令集拓展

SIMD ISA	Base ISA	Vendor	SIMD Registers
MAX	PA-RISC	HP	31x32b
MAX-2	PA-RISC	HP	32x64b
MMX	x86	Intel	8x64b
MDMX	MIPS-V	MIPS	32x64b
MIPS-3D	MIPS-64	MIPS	32x64b
SSE	x86/x86-64	Intel	8/16x128b
SSE2	x86/x86-64	Intel	8/16x128b
SSE3	x86/x86-64	Intel	8/16x128b
NEON	ARMv7	ARM	16x128b
SSSE3	x86/x86-64	Intel	8/16x128b
SSE4	x86/x86-64	Intel	8/16x128b
AVX	x86/x86-64	Intel	16x256b
AVX2	x86/x86-64	Intel	16x256b
AVX-512	X86-64	Intel	32x512b

x86 架构下，有 MMX、SSE、SSE2、SSE3、SSSE3、SSE4、AVX、AVX2 和 AVX512 等指令集[53]，ARM 架构下主要为 NEON 指令集[54]。MMX 指令集进行 64 位寄存器的运算操作，SSE 系列的指令集进行 128 位寄存器的存算操作，AVX、AVX2 进行 256 位寄存器的运算操作，而 AVX-512 则可以进行 512 位的运算操作，这意味着可以一次进行 64 个 8bit 的像素值的并行化操作，从而极大提升视频编解码和缩放的效率。

2.4.2 SIMD 原理

SIMD 的主要思想就是充分利用寄存器空间，进行并行化运算操作。随着硬件技术的发展，寄存器的大小逐渐从最初的 32bit 发展到 512bit 甚至更大，如何更有效地利用寄存器空间，成为大家心中的难题，因此出现了 SIMD，使用规定的指令集，可以对数据进行并行化操作[55]。SIMD 的操作不仅包括基本的运算如加减乘除，同时也支持其他计算任务如数据比较、转换、按位布尔运算、元素重新排列和移位等。处理器通过重新解释寄存器或内存位置中操作数的位来完成 SIMD 操作。

如对一个 128bit 的寄存器，可以同时存放两个独立的 64bit 的整数，同时也支持存放 4 个 32bit 的整数、8 个 16bit 的整数或者 16 个 8bit 的整数，就如同下图 2-9 所示[56]。

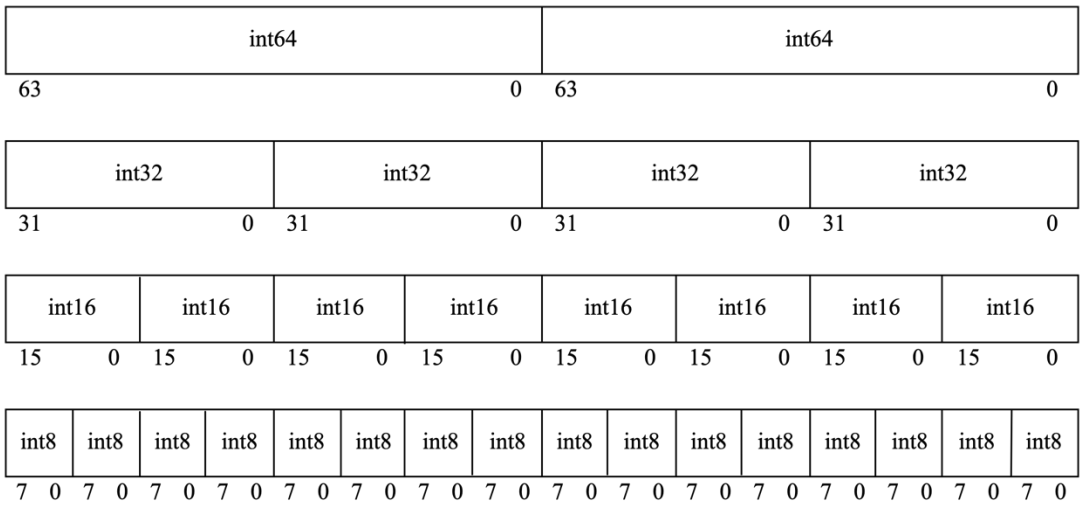

图 2-9 128bit 寄存器上使用 SIMD 操作不同整数类型示意图

根据不同的 SIMD 指令，同一个寄存器可以在逻辑上进行分割成不同大小的区间来存放操作数，目前常见的分割区间大小为 64bit、32bit、16bit 和 8bit。针对不同大小的分割区间，使用对应的操作指令，可以实现统一寄存器内操作数的运算，不同寄存器之间对应位置的操作数运算。同时，还可以将同一寄存器内操作数的占位进行调整，如将 8bit 大小的整数进行扩展到 16bit 大小，从而完成 16bit 整数之间的运算操作。

2.5 本章小结

本章主要对视频处理子系统中需要用到的技术背景进行了介绍。首先介绍了流媒体技术背景和 YUV 颜色空间。然后对 RTP 与 RTCP 协议进行了介绍，并对 GStreamer 框架进行了介绍，对其基本概念与框架主要元素进行介绍。最后对 SIMD 技术进行了介绍，包括常用处理器上的 SIMD 指令集拓展，以及利用 SIMD 进行优化的原理。

第三章 系统需求分析

本章主要介绍视频处理子系统的需求。在整个视频会议系统中，对视频处理子系统的上下文进行分析，从视频处理子系统主要功能出发，确定其功能需求，包括单画面视频流处理的需求和多画面模式的需求。非功能需求主要包括系统的安全需求、可维护性需求和性能需求。

3.1 功能需求

在进行视频处理子系统的需求分析之前，需要对视频处理子系统的上下文进行分析，了解视频处理子系统在整个视频会议系统中的位置，从而确定视频处理子系统的功能需求与非功能需求。整个视频会议系统的架构图如下图 3-1 所示。

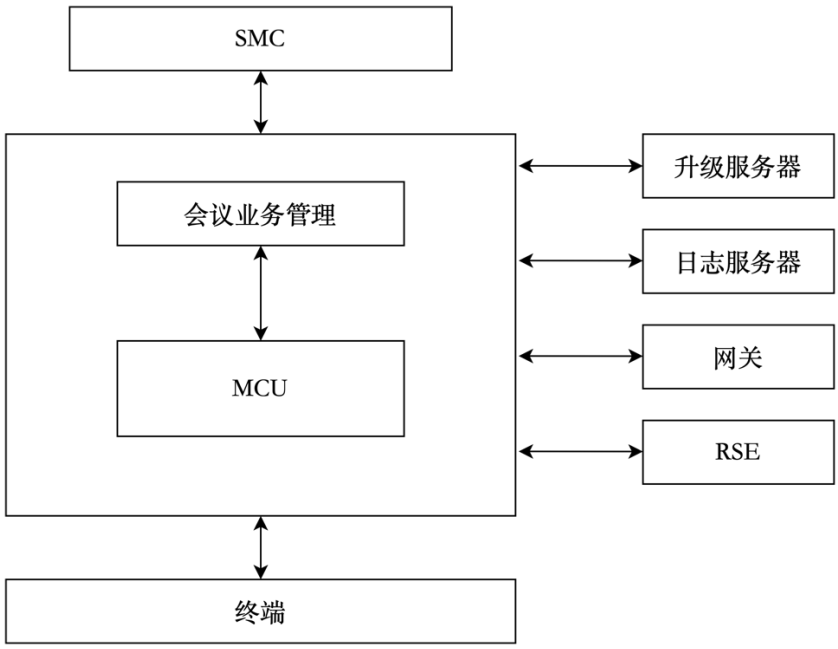

图 3-1 视频会议系统整体架构图

其中，SMC（Service Management Center）为业务管理系统，负责视频会议的召开、MCU 资源的选择、会场的添加、会议画面的设置等一系列在会议设置中的操作，可以通过 SMC 在 web 客户端对整个视频会议进行管理。会议业务管理主要进行 MCU 与 SMC 的对接，包括会议过程中消息的分发，会议资源的调度优化等操作，是处理消息和触发业务的核心部件。升级服务器负责对整个视频会议系统平台进行升级维护。主要包括新版本的发布。日志服务器是对整个视频会议系统运行过程进行独立统一的日志记录收集。通过日志服务器，对整个视频会议系统的出现的问题进行记录并根据出现的黑匣子对问题进行及时解决。网关基于标准的 SNMP 网管协议，实现视频会议系统中各个设备之间的网络通信并对网络运行状况进行实时检测。RSE（Recording & Streaming Engine，录播服务

器）主要对视频会议中视频、音频进行同步录制，并通过独立的页面进行录播画面的回放方便参与人员对能够对视频会议进行后续的回顾。终端主要包括硬终端与软终端。硬终端主要为视频会议录播设备等硬件，可以依据对应的硬件能力进行视频与音频的采集。软终端主要是指运行在个人计算机、移动客户端等设备上的软件应用，通过个人计算机自身的硬件能力来实现视频画面、音频的采集操作。MCU 主要包括视频会议系统中最核心的视频、音频处理子系统、会议控制子系统、媒体交换子系统、维护管理子系统和 IVR 子系统，其主要的架构图如下图 3-2 所示。

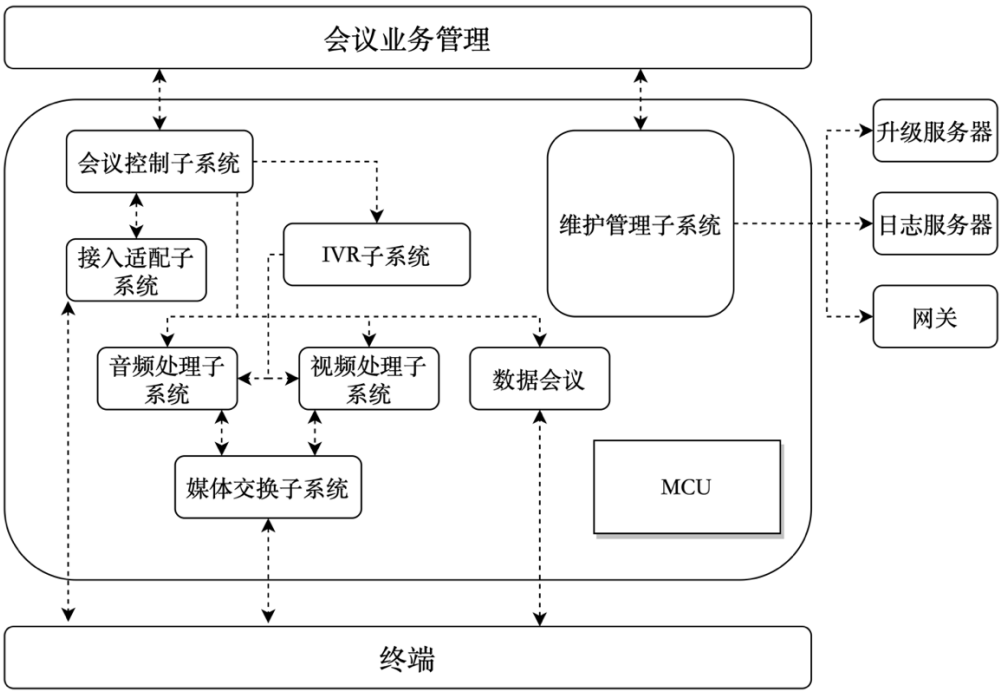

图 3-2 MCU 结构图

会议控制子系统主要进行 MCU 与终端的注册、媒体与终端的呼叫、能力协商、会议召集、网络自适应与会议的基本控制等操作。IVR（Interactive Voice Response，互动式语音应答）子系统主要根据会议控制子系统消息，进行视频字幕、会场名的叠加和音频欢迎入会的语音等操作。音频处理子系统主要进行音频的控制、音频的编解码处理、混音处理等操作。媒体交换子系统主要对网络中传输的 RTP 包进行分发、转发、加密等操作。将 RTP 包通过路由表转发到指定会场，将会场传输来的 RTP 包根据路由表分发到对应的音频与视频处理系统中，以及对 RTP 包进行加密机密。接入适配子系统主要进行不同终端设备接入系统后，根据终端设备的能力，对视频会议召开规格进行限制，同时确保不同终端能够对系统进行适配。维护管理子系统主要进行系统维护与更新，主要包括网元的维护管理、MCU 维护管理、终端维护管理和日志记录的收集维护等操作。

视频子系统是本文主要进行实现的系统，主要进行视频流的传输控制。视频处理子系统与会议控制子系统、媒体交换子系统进行交互，会议控制子系统进行会控信令的下发，以消息结构体的形式发送到视频处理子系统进行各种操作。包括视频流处理管道 pipeline 的构建和各种视频流参数设置的控制命令消息，如视频编解码的质量设置、视频帧率的设置、多画面模式的设置等。媒体交换子系统与视频处理子系统进行 RTP 包的交换，媒体交换子系统将采集的 RTP 包发送给视频处理子系统进行处理，处理之后的 RTP 包再返回给媒体交换子系统进行转发。

3.1.1 单画面视频流处理的需求

在了解视频处理子系统的上下文后，可以看到视频处理子系统主要进行视频流的处理控制操作。其最基本的一个功能需求就是一对一单画面视频流的传输处理，即视频会议中共有两个会场，将其中一个会场发来的 RTP 码流数据，经过处理形成新的 RTP 码流，发送到另一个会场进行播放。

在单画面视频流的处理过程中，需要依次进行 RTP 包的解码、H.264 视频流的解码、YUV 图像的缩放、H.264 视频流的编码、RTP 包的编码等一系列操作。同时依据会议控制子系统的信令，进行对应的控制操作。

会议控制子系统将会控消息发送到视频处理子系统中的视频控制模块后，视频控制模块首先进行消息的校验工作，然后根据消息类型将消息发送到视频处理模块进行处理。若为创建单画面视频流处理 pipeline 消息，则会构建 RTP 解码元件、视频解码元件、图像缩放元件、视频编码元件、RTP 编码元件。然后将其连接为完整的视频流处理 pipeline，并向会控子系统返回创建消息。若为控制命令，则首先寻找对应功能元件，然后通过元件上的控制命令处理函数进行处理。

在进行各个功能元件的创建时，首先需要进行向系统注册元件类型，表明现在系统中已存在该元件类型，可以进行元件的实例化。

在进行元件实例初始化时，需要对该元件实例涉及到的各种参数、数据处理函数与控制命令处理函数进行初始化。根据各个元件功能的不同，设置其对应的数据处理函数与控制命令处理函数。在单画面视频流处理过程中，涉及到的元件类型共有五个：RTP_DEC、VIDEO_DEC、RESIZE、VIDEO_ENC 和 RTP_ENC。下面对每一个元件的功能作分析，对于 RTP_DEC 类型元件，主要功能是完成 RTP 的拆包以及组帧，并将处理后的 H.264 视频流送到下一个元件进行处理。对于 VIDEO_DEC 类型元件，其主要功能为将 H.264 视频流解码为 YUV 图像。对于 RESIZE 类型元件，主要功能是 YUV 图像的缩放。对于 VIDEO_ENC 类型元件，主要功能是将 YUV 图像进行编码压缩。对于 RTP_ENC 类型元件，主要功能是完成 RTP 的打包封装。

完成功能元件的创建后，要将各个功能元件进行连接，从而构成视频流处理 pipeline，才能进行完整的视频流处理操作。

根据单画面视频流处理的需求，得到其需求用例图，如图 3-3 所示。

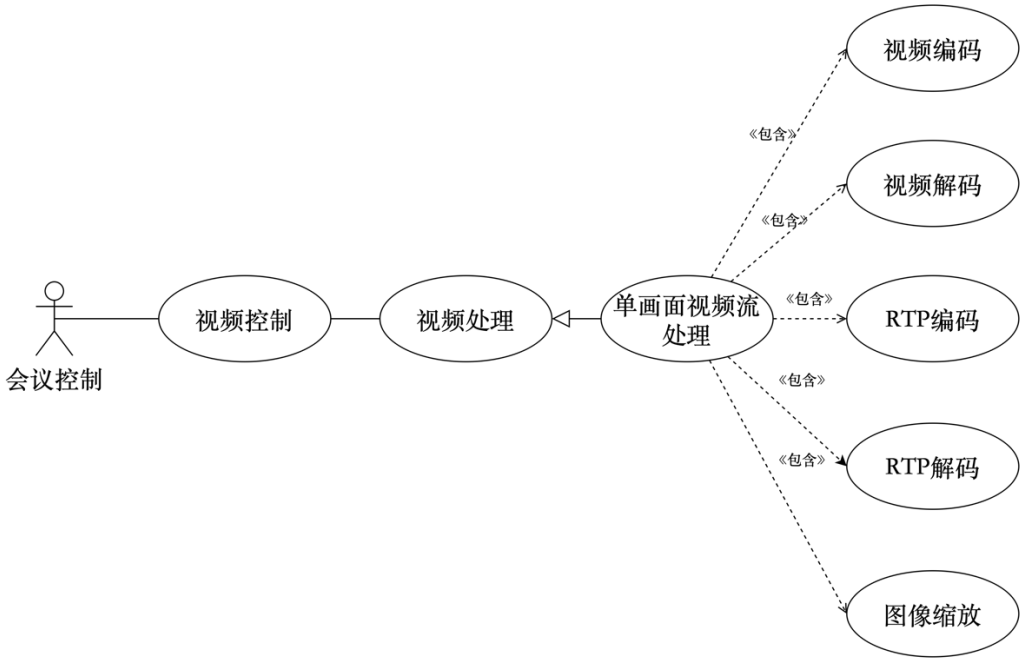

图 3-3 单画面视频流处理用例图

用例名称：单画面视频流处理。

说明：会议控制下发消息到视频控制模块，视频控制模块控制视频处理模块进行功能元件创建并连接为单画面视频流处理 pipeline。单画面视频流处理包含视频编码、视频解码、RTP 编码、RTP 解码和图像缩放功能。

参与者：会议控制子系统。

前置条件：会议控制子系统向视频控制模块下发创建单画面视频流 pipeline 消息，消息校验成功。

后置条件：单画面视频流处理 pipeline 构建成功。

3.1.2 多画面模式的需求

多画面模式主要针对多方会议的需求，即在一个视频会议中，包含多个会场，本地会场需要同时观看远端 1、2、3 会场时，需要将远端会场的画面进行拼接，整合成一个画面供本地会场观看。因此在多画面模式下，需要对不同的子画面进行缩放拼接，同时又因为子画面的帧率各不相同，因此需要对合成后的帧率进行统一。

多画面模式在单画面视频流处理的基础上进行设计实现。需要将不同远端会场的画面进行 RTP 解码、视频解码等操作，处理为 YUV 图像。然后将不同子画

面的图像进行缩放拼接，合成新的图像后，再进行视频编码、RTP 编码，形成新的 RTP 流发送到本地终端会场进行播放。

会议控制子系统向视频控制模块发送多画面模式的消息后，视频控制模块首先进行消息的校验工作，然后控制视频处理模块进行多画面视频流处理 pipeline 的构建。其中 RTP 解码元件、视频解码元件、图像缩放元件、视频编码元件、RTP 编码元件等元件功能与单画面视频流处理中一致。唯一不同的地方需要新增一个新的元件类型 MP_SPLICE（Multi Picture Splice），完成多画面模式下的图像拼接，并将不同子画面的帧率进行统一。同时可以根据会控子系统下发的控制命令，对多画面模式下不同子画面合成后的帧率进行设置。

完成所需的功能元件创建后，需要将各个功能元件进行连接，构成完整的多画面视频流处理 pipeline，实现多画面模式的功能。

根据多画面模式的需求，得到其对应的用例图如图 3-4 所示。

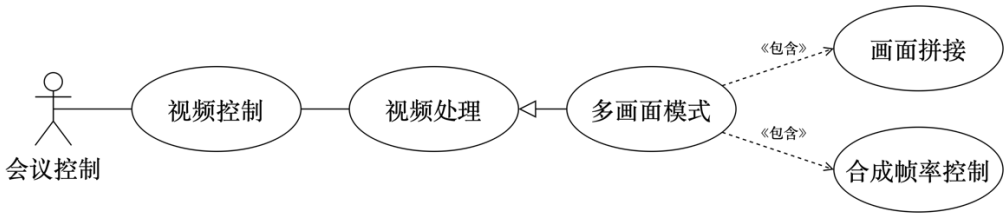

图 3-4 多画面模式用例图

用例名称：多画面模式。

说明：会议控制下发消息到视频控制模块，视频控制模块控制视频处理模块进行功能元件创建并连接为多画面模式下视频流处理 pipeline。多画面模式包含画面拼接和合成帧率控制功能。

参与者：会议控制子系统。

前置条件：会议控制子系统向视频控制模块下发多画面模式消息，消息校验成功。

后置条件：多画面视频流处理 pipeline 构建成功。

3.2 非功能需求

非功能需求主要包括性能需求、安全需求、系统可维护性需求。

安全需求主要指编码过程中满足安全编码要求，如指针的判空、内存拷贝的安全要求。安全需求在视频处理子系统中最基本的需求，如不能满足安全需求，会导致视频挂死、花屏等一系列的严重问题，且难以定位。因此需要在编码过程中对系统的安全需求作强调。安全需求是视频处理子系统的最基本的需求。

系统可维护性需求主要指系统在后续的更新维护中，能够根据业务需求的变更及时进行扩展。同时对于系统中关键日志能够及时记录，方便在出现错误的时

候能够及时根据日志记录进行解决。如要进行新需求的开发时，能够使用模块化的思想，不影响其他模块功能的前提下，新增新的功能模块，实现业务的快速部署。

性能需求主要指提高视频处理子系统中视频流处理性能。因为在视频会议系统中，视频流的处理过程是消耗系统性能主要的地方。如果能在视频处理子系统的性能上作优化，不仅可以召开更大规格的会议，而且可以使系统在一些低性能的机器上部署，提高系统的硬件平台兼容性。按照目前系统的规格，在 RH2288H V3 机器上，需要满足裸金属部署时，可以完成 25 路 1080P30 会场处理的需求，即能够容纳 25 个分辨率为 1080P，帧率为 30fps 的会场。

3.3 本章小结

本章主要首先介绍了视频处理子系统在整个视频会议系统的位置，确定其主要功能。然后对其主要的两个功能单画面视频流处理与多画面模式进行需求分析。其中，单画面视频流处理包括 RTP 解码、视频解码、图像缩放、视频编码、RTP 编码等过程。多画面模式需要在单画面视频流传输的基础上，实现画面拼接与合成帧率的控制的功能。最后对系统的非功能需求进行了分析，主要为性能、安全性和可维护性需求。

第四章 视频处理子系统详细设计

本章主要介绍视频处理子系统的详细设计。首先对视频处理子系统进行整体框架的设计。然后对功能需求中的单画面视频流处理进行详细设计，对多画面模式的需求进行详细设计。最后对非功能需求中性能需求进行设计，利用 SIMD 设计缩放元件中 YUV 图像缩放优化策略。

4.1 系统框架

根据视频会议系统的整体框架以及 MCU 的架构，已知视频处理子系统对外与会议控制子系统与媒体交换子系统进行交互。会议控制子系统对视频处理子系统进行各种信令控制，媒体交换子系统与视频处理子系统之间进行 RTP 码流的传输。在视频处理子系统内部，需要设置视频控制模块与会议控制模块进行消息的传递。在接受到会议控制子系统发送来的消息后，视频控制模块进行消息的校验，将视频处理控制消息进行转发，发送给视频处理模块。视频处理模块进行视频流处理 pipeline 的搭建，主要实现 RTP 编码、RTP 解码、视频编码、视频解码、多画面处理、图像缩放等功能。在利用 GStreamer 框架进行搭建完成后，发送消息给视频控制模块，视频控制模块再发送消息给媒体处理子系统，此时媒体处理子系统开始向视频处理子系统发送 RTP 码流。RTP 码流经过已建好的视频流处理 pipeline，完成码流的处理，并返回 RTP 码流到媒体交换子系统。视频处理子系统整体的框架图如下图 4-1 所示。在完成整体框架设计后，对系统功能需求进行详细设计。

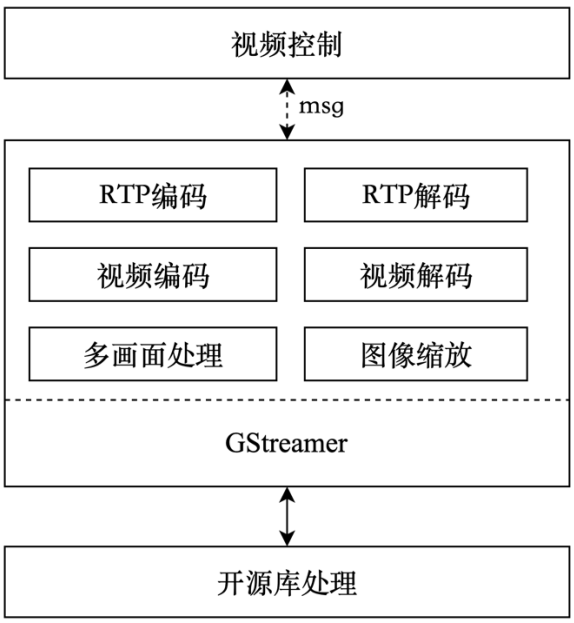

图 4-1 视频处理子系统框架图

4.2 单画面视频流处理设计

4.2.1 元件设计

在单画面视频流处理过程中，首先对该需求中所需要的元件类型进行设计，共包含RTP_DEC类型元件、VIDEO_DEC类型元件、RESIZE类型元件、RTP_ENC类型元件和VIDEO_ENC类型元件。

在元件类型名称确定后，进行元件类型的注册，完成元件类型的注册后，可以进行元件实例的创建。元件实例创建时，因为创建过程是互斥的，所以要采用信号量锁来对元件创建进行资源保护。首先对元件的初始化参数进行校验，若校验失败则直接返回，否则进行加锁，传入元件的类型、元件实例名称、元件初始化参数。根据元件类型获取元件构造函数，判断元件构造函数是否为空，不为空则判断元件实例索引是否超过系统支持的最大元件数量，若不超过则为元件分配全局唯一索引。然后创建元件，根据元件初始化参数初始化元件，并将元件注册到系统中，传入该元件的索引和句柄，表示该索引对应的元件已被创建，后续该元件索引将不会被其他元件所申请到。最后互斥锁解锁，完成整个元件实例的创建，具体的流程如图4-2所示。

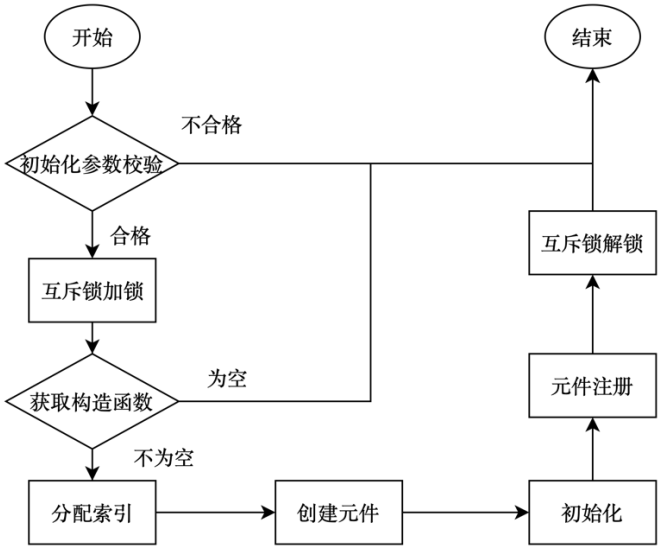

图4-2 元件实例创建流程图

在元件实例初始化函数中，定义元件可接受数据类型。在GStreamer的框架下，两个元件是可以相互连接，但是传递的数据结构可能不同，因此需要对元件之间进行数据传递的结构进行定义。一个元件只能接收与其定义的数据类型匹配的数据。在视频流处理过程中，定义四种数据类型：非法数据、RTP数据、NALU数据、YUV数据。对于RTP解码元件，只能接收RTP数据进行处理。对于Video解码元件，只能接收NALU数据。对于Resize元件和Video编码元件，只能接

收 YUV 数据。对于 RTP 编码元件，只能接收 NALU 数据。元件在收到对应的数据后，通过数据处理接口进行处理，实现元件的主要功能。

同时，定义元件实例可接受的控制命令消息的类型。对于 RTP 解码元件，其只能设置 H.264 视频流大小消息。对于 Video 解码元件，能接受的控制消息有设置旋转角度。对于缩放元件，可以接收的控制消息为设置缩放质量的等级。对于 Video 编码元件，可以接收的控制消息为设置编码质量、设置编码带宽和设置编码帧率。对于 RTP 编码元件，可以接收的控制消息为设置帧率。元件在收到控制消息后，使用对应的控制命令处理函数进行处理。控制命令消息一般对元件的参数进行更新，在系统后续更新维护中可添加元件的控制命令消息。

4.2.2 视频流端到端处理流程设计

对于一个会议，定义一个会议号。对于会场之间的视频流传输，使用视频流来描述，并通过流号来进行唯一的标识。视频流中设有编码节点、解码节点和适配节点，流号与对应的流节点进行绑定，并注册到系统当中，后续可以通过查询流号获取到对应的流节点。整个处理流程可以分为从上到下节点、元件、开源处理三层。

第一层为节点层，当会议控制子系统向视频处理子系统发送创建单画面视频流 pipeline 消息时，视频控制模块完成消息的校验后根据消息中的流号进行最上层节点层的创建。判断编码、适配、解码节点是否存在，流节点中包含其所控制元件的句柄，用于元件的创建销毁。编码节点中主要含有编码流号、节点 ID、会议号、Video 编码元件实例句柄、RTP 编码元件实例句柄等信息。解码节点中含有解码流号、节点 ID、会议号、Video 解码元件实例句柄、RTP 解码元件实例句柄等信息。适配节点中含有编码流号、解码流号、节点 ID，会议号、缩放 Resize 元件实例句柄等信息。

第二层为元件层，主要是各个功能元件的实现，由节点层进行控制创建与销毁。根据之前元件的设计流程，单画面视频流处理共涉及五种类型元件，由节点层中的节点进行对应类型元件实例的创建。根据编码节点 encode_node 创建 RTP 编码、Video 编码元件。RTP 编码元件主要功能是将 NALU 编码为 RTP 码流，Video 编码元件主要将 YUV 图像编码为 NALU。通过解码节点 decode_node 进行 RTP 解码元件、Video 解码元件的创建。RTP 解码元件进行 RTP 包的拆包与 H.264 的组帧，形成 NALU 送到 Video 解码元件。Video 解码元件进行 NALU 的解码，解为 YUV 图像。根据适配节点 adapt_node 进行缩放元件的创建，缩放元件主要实现 YUV 图像的缩放。

第三层为开源处理层，主要是元件在实现具体功能时需要调用的一些开源库，在元件接收到输入的数据后，需要根据业务的需求对输入数据进行参数的配置，然后传入到开源库的接口中进行数据的处理操作。

通过节点层、元件层和开源处理层的设计，得到单画面视频流处理的一个过程。输入为 RTP 码流，经过 RTP 解码、Video 解码、图像缩放、Video 编码、RTP 编码元件处理，最终形成新的 RTP 包发送出去，实现 RTP 码流的端到端处理。通过在元件层上方添加节点层，可以更好的实现元件的控制操作，方便进行元件的扩展，增加了系统的可维护性。

4.3 多画面模式设计

多画面模式下，需要对不同会场的解码流画面根据多画面模式进行拼接。对于一个多画面会议，如果以三画面为例，本地会场需要观看远端三个会场 1、2、3 的画面，无法直接将会场 1、2、3 的视频流进行拼接。需要将三个会场的视频流分别进行 RTP 解包、H.264 视频流解码操作，将其解为 YUV 图像，然后将 YUV 图像进行拼接。因此，需要在单画面视频流传输的基础上，新增一个多画面处理元件，来完成多画面的拼接，同时进行合成后画面帧率的控制。

4.3.1 多画面处理元件设计

多画面处理元件的功能主要是接收不同子画面的 YUV 图像，然后进行画面拼接，进行合成后画面帧率控制，然后将合成后画面送到 Video 编码元件进行视频编码。在向系统进行 MP_SPLICE 类型元件的注册后，即可进行多画面处理元件实例的创建，元件创建流程与其他元件流程类似。多画面处理元件可以接收的数据类型为 YUV 数据，同时对其可接受的控制命令消息进行定义，主要接收多画面模式设置、合成后帧率设置的控制命令。

其中对元件进行初始化时，要对元件的配置参数、数据处理函数、控制命令处理函数进行设置。在进行元件参数配置时，首先要对多画面画板进行初始化，对传入的图像格式（YUV420、YVU420 等）进行转化，统一转为 YUV420 格式，根据合成后的画面宽高、图像格式进行画板缓存的申请。

其次，要对多画面驱动进行初始化。在这里采用定时驱动的方式，通过设计帧率控制策略，实现不同子画面的帧率进行统一。最后进行子画面参数的初始化，每个子画面用一个队列维护，即每次将 10ms 内收集的消息存放到一个消息队列中。下面就帧率控制策略展开设计。

首先进行帧率的预处理，根据编码器的能力设置最大帧率为 60fps，同时为保证视频质量规定最小帧率为 5fps。将帧率向下取整化为 5 的整数倍，并控制在 5-60fps 之间，使用处理后的帧率进行帧率控制。

然后进行触发多画面合成时刻表的设计。设置一个定时驱动编码标志 flag 来标识合成多画面时刻。通过使用 10ms 计数器，设置局部变量 10msCounter 来进行 10ms 触发数量统计。每隔 10ms 变量 10msCounter +1，判断 flag 是否为 True。若为 True，则进行多画面合成。因最小帧率为 5，故最大时间间隔为 200ms，即每隔 200ms 中有一次将 flag 置为 True 即可。若帧率为 60，则 200ms 中设置 12 次将 flag 设为 True 即可。因此可以通过查表的方式来进行决定是否进行多画面合成。设计的触发合成多画面合成时刻表如表 4-1 所示。

表 4-1 触发合成多画面时刻表

帧率 fps	触发时刻
5	0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 1
10	0 0 0 0 0 | 0 0 0 0 1 | 0 0 0 0 0 | 0 0 0 0 1
15	0 0 0 0 0 | 1 0 0 0 0 | 0 1 0 0 0 | 0 0 1 0 0
20	0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1
25	0 0 0 1 0 | 0 0 1 0 0 | 0 1 0 0 0 | 1 0 0 0 1
30	0 0 1 0 0 | 1 0 0 1 0 | 0 1 0 0 1 | 0 0 1 0 0
35	0 0 1 0 0 | 1 0 0 1 0 | 0 1 0 0 1 | 0 0 1 0 1
40	0 0 1 0 1 | 0 0 1 0 1 | 0 0 1 0 1 | 0 0 1 0 1
45	0 1 0 1 0 | 0 1 0 1 0 | 0 1 0 1 0 | 1 0 1 0 1
50	0 1 0 1 0 | 1 0 1 0 1 | 0 1 0 1 0 | 1 0 1 0 1
55	0 1 0 1 0 | 1 0 1 0 1 | 1 0 1 0 1 | 1 0 1 0 1
60	1 0 1 0 1 | 1 0 1 0 1 | 1 0 1 0 1 | 1 0 1 0 1

最后通过变量 10msCounter 的值与输入帧率进行查表，若定时驱动编码标志返回为真则进行多画面合成。若变量 10msCounter 值大于等于 20 则进行清零重新计数。通过设计触发合成多画面时刻表，可以迅速通过查表进行帧率的控制操作。

4.3.2 多画面视频流传输控制设计

在完成多画面处理元件的设计后，可以设计整个多画面视频流传输的控制流程。多画面视频流传输控制流程分为从上到下节点层、元件层、开源处理层三层。

其中节点层包括编码、解码、适配流节点。编码节点与单画面视频流处理流程功能类似，主要完成 Video 编码、RTP 编码元件的控制操作。对于不同的子画面解码流，定义不同节点 ID 的子画面解码节点，每个子画面解码节点上进行 Video 解码、RTP 解码元件的控制操作。这里较为关键的是适配节点，适配节点与单画面视频流处理过程中的适配节点有所不同，需要新增多画面处理元件句柄，用以增加对多画面处理元件的操作。同时需要在节点参数中记录每一个子画面的解码流号，方便后续通过解码流号将子画面缩放元件与对应流号上的 Video 解码元件相连。

元件层中包含 RTP 解码、Video 解码、Resize、多画面处理、Video 编码、RTP 编码元件。其中新增多画面处理元件，负责将不同子画面缩放后的图像进行拼接，然后将拼接后的图像送到 Video 编码元件中进行编码操作，同时需要在合成图像后统一不同子画面的帧率。其他元件功能与单画面视频流处理过程中功能一致。

开源处理层与单画面视频流处理的过程相比无变化，因为新增的多画面处理元件在元件层进行具体的数据处理操作。同时在图像缩放时仍使用 libyuv 开源库完成缩放元件中 YUV 图像的缩放操作。

4.4 缩放性能优化设计

4.4.1 libyuv 开源库介绍

libyuv 库是应用在 YUV 图像处理领域的常用开源库，可以实现 YUV 图像格式的转换、旋转、缩放等操作。在缩放操作中，实现了基本的图像缩放算法，包括：

（1）点采样：直接进行像素值的拷贝，速度最快。

（2）线性插值：只进行水平方向的线性插值。

（3）双线性插值：完成水平垂直两次线性插值。

（4）盒采样：类似于点采样，只不过只应用在缩小且高尺度缩小 2 倍以上的场景。对于一个 box，不像点采样只选择 box 里的一个点，而是选取整个 box 内像素值的平均值。

libyuv 对这些缩放算法在不同的硬件平台上做了一定的优化，同时，libyuv 针对四种特定的缩放比例进行了优化，如表 4-2 所示。

表 4-2 libyuv 优化缩放比例表

缩放比例	宽缩放比	高缩放比
1	4:3	4:3
2	4:1	4:1
3	8:3	8:3
4	2:1	2:1

在 libyuv 库中，根据图像缩放采用的算法，定义了四种缩放质量等级，如下表 4-3 所示。

表 4-3 libyuv 定义缩放质量等级表

缩放质量等级	采用方式	描述
0	点采样	速度最快，质量最差
1	线性插值	速度较快，质量较差
2	双线性插值	速度较慢，质量较好
3	盒采样	速度最慢，质量最好

每次进行图像的缩放时，根据图像缩放质量等级与图像缩放宽高比采取不同的缩放手段。在进入到对应的缩放函数中后，根据硬件平台的特性，首先进行缩放函数的 C 语言实现，然后通过函数指针的方式，将核心的像素值操作采取对应的 SIMD 指令重新进行实现，从而实现性能的进一步提升。因此可以根据具体的应用场景，自定义缩放质量等级，从而进行缩放元件的性能优化。

4.4.2 缩放优化策略设计

在视频流的传输过程中，YUV 的缩放在缩放元件 Resize 中进行。因此，利用 Resize 元件的控制命令函数，定义缩放的质量等级。在 libyuv 已定义缩放质量等级的基础上，定义第 5 种缩放质量自定义优化 kFliterSelfOpt。当处于该缩放质量下后，在 libyuv 库中，调用自定义的优化方式。下面以 720p 图像到 476x268 图像为例进行设计，平台为 x86，SIMD 指令集为 SSSE3，寄存器大小为 128bit，像素值类型为无符号整型，占 8bit。在设计缩放优化策略时，需要考虑以下问题。

（1）可用的寄存器资源，以及寄存器大小。

（2）汇编指令计算方式，图像处理需满足汇编指令计算方式。将多个像素值求平均时，通过寄存器移位来操作，只能进行 2、4、8、16 等 2^n 个像素值的平均值求解。

（3）可用的 SIMD 指令，不同的 SIMD 指令集上，所支持的 SIMD 指令有所不同。

（4）UV 分量的像素值计算，UV 分量图像宽高为 Y 分量图像宽高一半，需要考虑 UV 分量计算过程是否与 Y 分量计算一致。

在综合考虑以上问题后，定义缩放优化策略，设计的基本思路如下。

（1）判别硬件平台

目前视频处理子系统主要应用在 x86 硬件平台上，根据平台能力确定可以使用的 SIMD 指令集。x86 平台上，一般可以使用 MMX、SSE、SSE2、SSE3、SSSE3 等指令集。

（2）确定并分解宽高比

720p 图像到 476x268 图像的宽高比分别为 320:119、180:67，Y、U、V 三个分量计算过程一致。无法一次将 320 个像素值放到 128bit 寄存器中，因此将缩放比进行分解。定义若干小尺寸缩放，320/119 大于 2，小于 3，因此可以分解为若干 2:1、3:1 的组合。在 3:1 的缩放下，因为 SIMD 的特性，每次只能进行相邻像素值之间的操作，因此需要定义偏移值，将三个像素值偏移到寄存器四个连续的位置上，累加除四，从而实现 3:1 的缩放。320:119 分解为 82 个 3:1，37 个 2:1。同理，得到高缩放比下的分解，分解为 46 个 3:1，21 个 2:1。据此设计宽和高的缩放策略。

（3）组合高缩放比

在得到高缩放比例的缩放分解后，不能简单地进行 46 次 3:1 的循环，再接 21 次 2:1 的循环，这样缩放后的图像会出现上下图像缩放比例不协调的问题。因此，本文目前采用组合缩放比的方式，每两个 3:1 缩放后紧跟一个 2:1 的高缩放。即首先进行 21 次循环，每次循环里进行 3:1、3:1、2:1 的高缩放，最后将剩下的四次 3:1 进行缩放。

（4）组合宽缩放比

宽缩放与高缩放思路类似，不过宽缩放需要考虑寄存器大小，并定义偏移值。将 82 次的 3:1 缩放与 37 次的 2:1 缩放组合成 37 次 3:1、3:1、2:1 的缩放，再进行 8 次 3:1 的缩放。这样可以由一个寄存器完成源图像 8 个像素值到目的图像 3 个像素值的缩放。

在 128bit 寄存器下，最多存储 16 个 8bit 像素值，对于 3:1、3:1、2:1，可以定义偏移值{0,1,1,2,3,4,4,5,6,6,7,7,0,0,0,0}，即将源图像位置为 0、1、2 的像素值按偏移值{0,1,1,2}放入寄存器前 32bit，做两次相邻值累加，后向右移两位除四，从而实现 3:1 的缩放。因为要同时实现 2:1 的缩放因此需要将源图像位置 6、7 的像素值以 6、6、7、7 的形式偏移到寄存器中，从而可以与 3:1 的缩放在同一个寄存器中同时进行计算。不过这样导致有 32bit 的空间未能得到利用，造成了一定的资源浪费。最后剩下 8 次 3:1 的缩放，可以定义偏移值{0,1,1,2,3,4,4,5,6,7,7,8,9,10,10,11}，充分利用寄存器资源。一次将 12 个源图像像素值缩放成目的图像 4 个像素值，循环两次，完成整行的缩放。

（5）特殊处理

在目的图像像素值进行赋值的时候，赋值指令为 mov 类指令，只能进行 movb（1 个字节的赋值）、movw（2 个字节的赋值）、movl（4 个字节的赋值）、movq（8 个字节的赋值）、mov（整体赋值）。而每次目的图像每行只需要三个有效像素值，所以至少使用 movl 或 movq 或 mov 指令，将有效像素值进行覆盖。与此同时，会造成过拷贝的问题，使用 movl 指令，会多将一个字节赋值到目的图像，如图 4-3 所示。

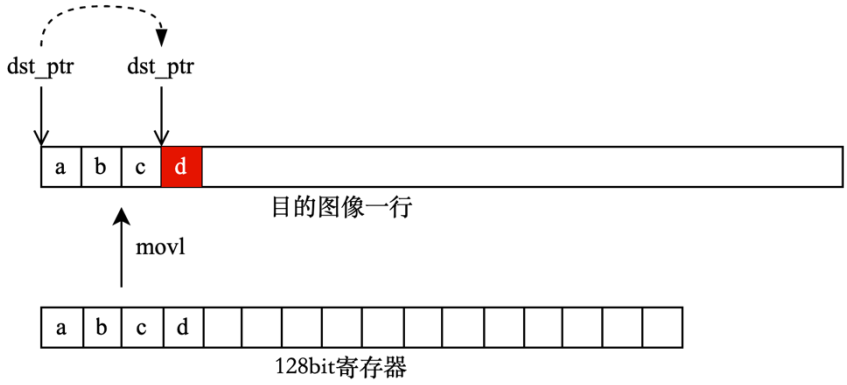

图 4-3 目的图像像素值赋值错误示意图

目的图像最后一行最后三个像素值赋值时会造成内存越界的问题，解决这个问题主要有两种方式。

1、优化行缩放策略。在进行行缩放时，在进行目的图像行最后几个像素点的赋值时，尽量采用 2、4、8、16 个像素值赋值方式，如上文采用的行缩放策略则不会出现内存越界问题，因为每行的最后 8 个像素值是分两次，每次进行 4 个像素值的赋值，直接使用 movl 指令即可。

2、将目的图像最后一行的赋值改为 C 语言实现，不采用 SIMD 进行优化。在上文高缩放策略中，最后一行的缩放是采用 3:1 的方式，此时可以将该实现方式改为 C 语言实现，从而规避汇编优化可能出现的错误。因为只是将最后三行缩放没有采用优化，对于一张 YUV 图像整体而言，影响较小。

（6）汇编函数设计

在设计完成整个优化流程后，可以看到需要利用 SIMD 实现两个关键函数，一个是 ScaleRowDown3To1_Box_SSSE3 与 ScaleRowDown2To1_Box_SSSE3。第一个函数实现高 3:1 的缩放，同时采用已定义行缩放策略。第二个函数实现高 2:1 缩放，同时采用已定义行缩放策略，都采用盒采样的像素值计算方式。对于 ScaleRowDown3To1_Box_SSSE3 函数，首先从源图像三行像素依据偏移值 OffsetRow1{0,1,1,2,3,4,4,5,6,6,7,7,0,0,0,0}分别取 8 个像素点放置到对应的三个寄存器上，先进行行内像素值计算，再进行行间像素值计算，最后将像素值赋值到目的图像上，如此循环 37 次，如图 4-4 所示。然后根据偏移值 OffsetRow2{0,1,1,2,3,4,4,5,6,7,7,8,9,10,10,11}将源图像两行上末尾的 12 个像素值放置到对应的两个寄存器上，先进行行内像素值计算，再进行行间像素值计算，最后将计算好的 4 个像素值赋值到目的图像上，共进行两次，具体示意图如图 4-5 所示，从而完成 ScaleRowDown3To1_Box_SSSE3 函数功能。

在完成 ScaleRowDown3To1_Box_SSSE3 函数的设计后，进行 ScaleRowDown2To1_Box_SSSE3 函数的设计，该函数需要将两行像素值缩放成一行，因此只需使用两个寄存器，两次行缩放的偏移值与之前一致。至此，完成整个缩放优化的一个设计流程。

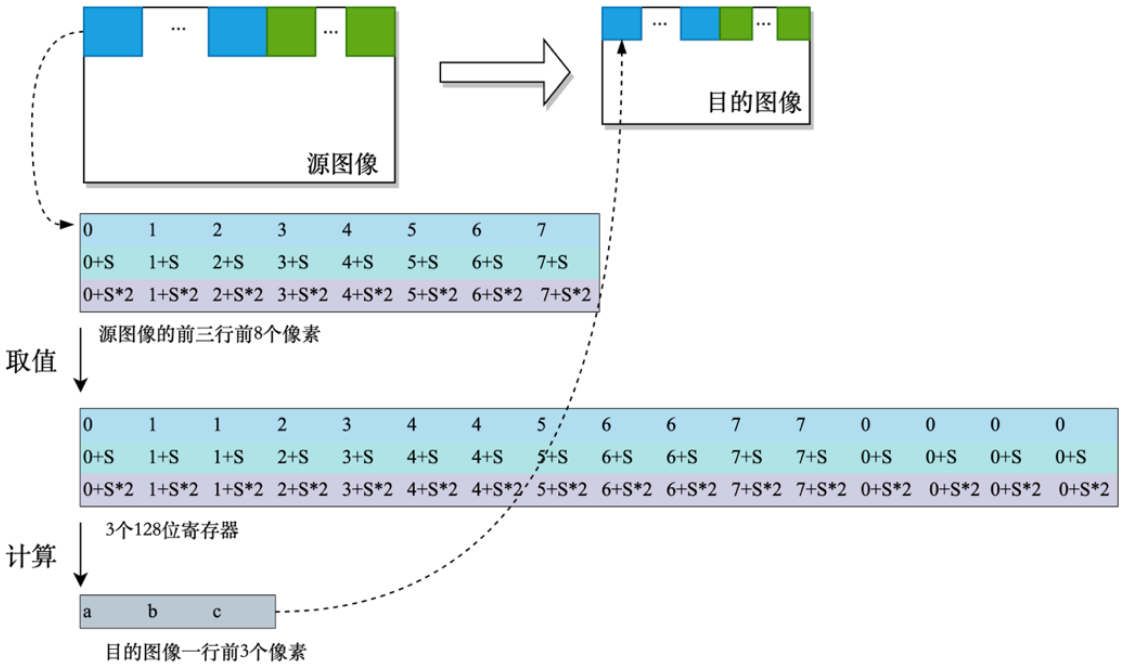

图 4-4 偏移值 OffsetRow1 缩放示意图

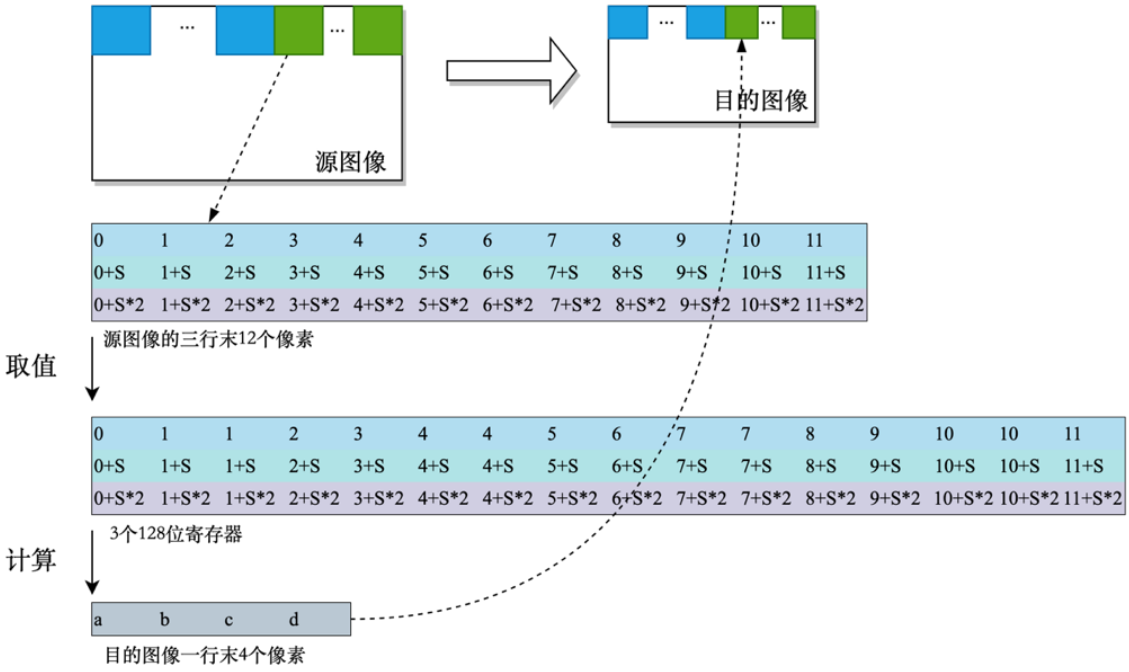

图 4-5 偏移值 OffsetRow2 缩放示意图

4.5 本章小结

本章根据之前的需求分析，对视频处理子系统的功能进行了详细设计。包括系统整体的框架，单画面视频流处理的设计流程，多画面模式的设计流程。其中单画面视频流设计流程包括功能元件的设计，以及 RTP 码流端到端的一个处理流程。多画画模式设计包括多画面处理元件的设计，多画面合成后帧率控制策略的设计，以及整个多画面视频流的一个控制处理流程设计。最后对缩放优化策略进行了详细设计。

第五章 视频处理子系统的实现

本章主要介绍视频处理子系统的实现过程。基于之前对视频处理子系统的功能设计，对单画面视频流处理、多画面模式整体时序进行实现。对单画面视频流处理流程中涉及到的 RTP 解码、Video 解码、Resize、Video 编码、RTP 编码元件功能进行实现，对多画面模式中的多画面处理元件进行功能实现。根据之前设计的缩放优化策略，使用 SIMD 汇编指令进行实现。

5.1 单画面视频流处理的实现

5.1.1 整体时序实现

根据上一章中对单画面视频流处理流程的设计，可以看到，会议控制子系统对视频处理子系统进行一个整体的控制，进行单画面视频流处理 pipeline 的搭建。在本节中，根据视频处理子系统整体的框架与 RTP 码流的端到端处理流程，对单画面视频流处理的一个整体时序过程进行实现。

会议控制系统发送创建单画面视频流 pipeline 消息到视频控制模块，视频控制模块首先进行消息的校验，进行消息非空判断。若为空，直接打印错误日志，包括消息中的会议号、编码流号、解码流号，并返回错误码。若不为空，则根据消息中流号查询编码、解码、适配节点是否存在，若已存在各节点，则对节点参数进行更新，否则依次创建各节点。

首先进行编码节点的创建。创建空编码节点并进行初始化编码节点参数，将编码流号于编码节点 ID 进行绑定，并向系统注册，方便后续的节点查询。然后进行 Video 编码元件参数初始化，包括流号、帧率、码率、编码协议和图像宽高等参数，并创建 Video 编码元件实例。然后进行 RTP 编码元件参数初始化，包含流号、编码协议、RFC 打包模式，帧率等信息，并进行 RTP 编码元件实例的创建。最后将 Video 编码元件与 RTP 编码元件进行连接并激活，更新编码节点信息。

然后进行解码节点的创建。创建空解码节点并初始化解码节点参数，将解码流号与解码节点 ID 进行绑定并注册到系统当中。然后进行 RTP 解码元件参数初始化，主要包含流号、视频协议、RFC 打包模式。参数初始化完成后创建 RTP 解码元件。完成 RTP 解码元件创建后，进行 Video 解码参数的初始化，主要包含流号、视频协议、视频格式等参数，并创建 Video 解码元件。创建成功后，将元件 RTP 解码与 Video 解码连接并激活，更新解码节点，并将解码节点 ID 与解码流号绑定注册到系统当中。

在完成编码、解码节点的创建后，进行适配节点的创建并进行新节点参数的初始化，主要包括编码流号、解码流号、Resize 元件句柄。根据解码流号创建 Resize 元件并初始化，Resize 元件参数主要包括解码流号、源图像的宽高、目的图像的宽高、缩放质量的等级等信息。完成 Resize 元件的创建后，对适配节点的参数进行更新，更新 Resize 元件句柄信息。完成适配节点创建后，将适配节点 ID 与编码流号绑定并注册到系统当中。

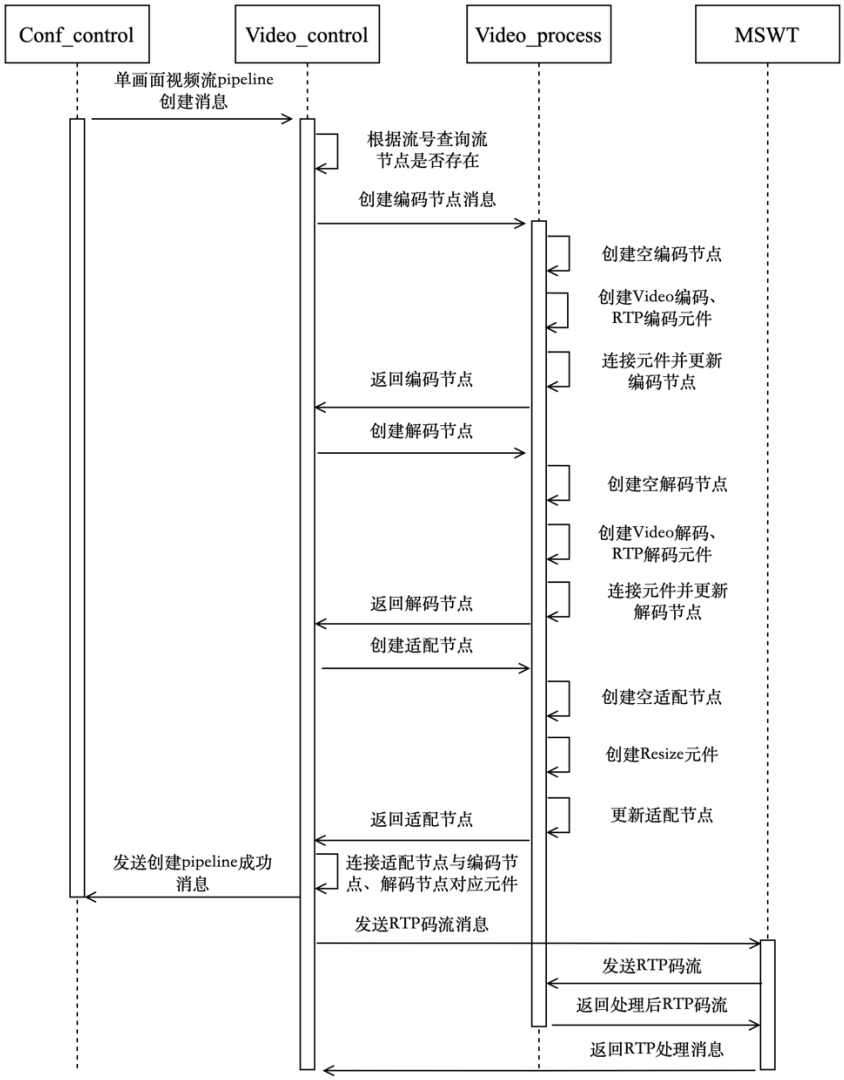

图 5-1 单画面视频流处理整体时序图

完成适配节点创建后，根据编码流号、解码流号查询对应的编码节点、解码节点。将适配节点中的 Resize 元件与编码节点中的 Video 编码元件连接，然后与解码节点中的 Video 解码元件进行连接。完成单画面视频流 pipeline 的创建，打印创建成功消息，然后向会议控制模块返回 pipeline 创建成功消息。

视频控制模块在完成 pipeline 的搭建后，向 MSWT 发送 RTP 转发消息，MSWT 在收到视频控制模块下发的消息后，进行消息的校验，然后开始向视频

处理模块发送 RTP 码流，经过视频处理模块处理后的码流返回到 MSWT 中，由 MSWT 转发到指定会场，同时打印处理日志，然后向视频控制模块发送处理成功消息。具体处理实现如图 5-1 所示。

在创建编码、解码、适配节点的过程中，在完成节点创建后，会将节点创建关键信息打印。在元件创建后，会将元件创建信息打印。后续通过日志判断视频流中的节点是否正常创建，各功能元件是否正常创建。

5.1.2 元件功能实现

单画面视频流处理中主要涉及 Video 编码元件、RTP 编码元件、Resize 元件、Video 解码元件、RTP 解码元件，每个元件都有自己的参数配置、数据处理函数，控制命令处理功能，下面介绍各个元件的实现过程。

（1）Video 编码元件

在 Video 编码元件，在进行元件的创建时，主要包括元件的配置参数设置、元件功能实现函数、元件控制命令函数。其中元件配置参数包括视频流号、视频协议、视频格式、帧率、带宽、编码质量、图像宽高和编码统计信息。

数据处理函数是完成元件功能的重要函数，对于 Video 编码元件的数据处理函数，完成 YUV 图像到 H.264 视频流的压缩过程。首先判断接收到的数据是否为可接受数据类型，然后申请空图像缓冲池，对于传入的数据进行提取，对负载数据进行图像有效性检验，判断其是否符合视频会议系统的要求，然后将负载数据存入图像缓冲池。对编码输入参数进行设置，对 YUV 图像的 Y、U、V 像素起始指针进行设置，对图像的步幅进行设置，并判断是否编码 I 帧，若编 I 帧，则设置 I 帧量化参数 QP 值，否则设置 P 帧 QP 值。然后将编码参数与图像数据一起传入编码器接口函数，然后判断返回是否成功，若成功，则编码元件参数编码成功数加一，更新编码时间信息。若编码失败，编码统计参数编码失败数加一，最后将申请的图像缓冲池释放。具体过程如图 5-2 所示。

控制命令处理功能也在元件创建时进行配置，元件可以接收的控制命令有限且需要提前定义，这样可以防止无效的控制命令。在 Video 编码元件中，定义三种控制命令：设置编码器质量、设置编码带宽、设置编码帧率。控制命令接收函数在收到控制命令后，找到对应的控制命令处理函数，进行参数有效性校验，然后将编码器质量等级、编码带宽、编码帧率等参数传入编码器接口函数中。最后判断是否设置成功，若设置成功，则对 Video 编码元件对应的参数进行更新。

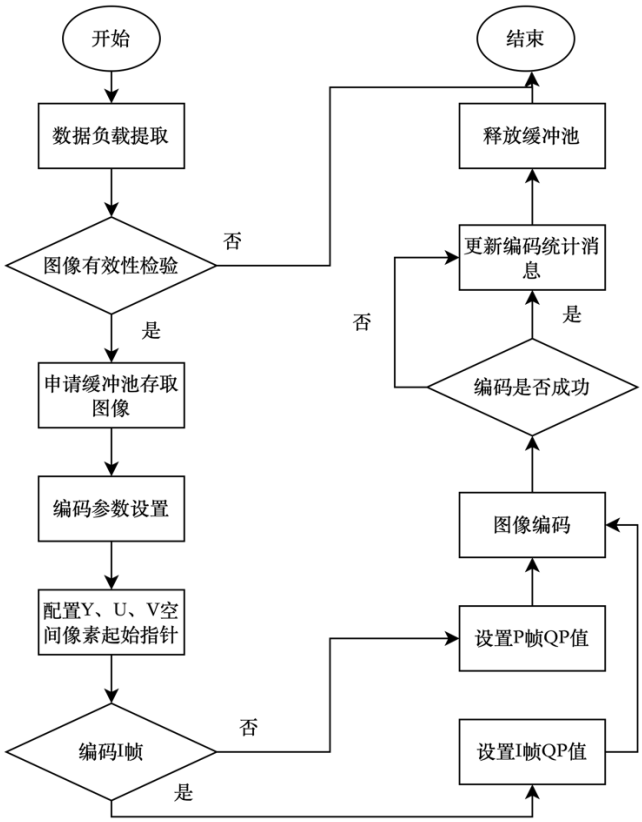

图 5-2 Video 编码元件功能实现

（2）RTP 编码元件

在 RTP 编码元件中，主要完成 RTP 包的封装过程，首先从 RTP 编码元件的创建开始，进行参数配置、数据处理函数设置、控制命令函数设置。其中配置参数主要有视频流号、视频编解码协议、RFC 打包模式、负载类型和帧率。

数据处理函数首先进行数据类型的校验，然后进行数据内容的非空判断，同时将数据转化为 NALU 类型。因为视频 NALU 大小不一，对于小于最大传输单元 MTU（Maximum Transmission Unit）的 NALU，采用 single-nalu 的封包方式，对于大于 MTU 的 NALU，使用 FU-A 进行分片，然后再进行封包。Single-NALU 的方式下，不用处理 NALU 头。在 FU-A 方式下，根据 NALU 头构建 FU-A indicator（1 字节）与 FU header（1 字节）。具体构建方式如图 5-3 所示。

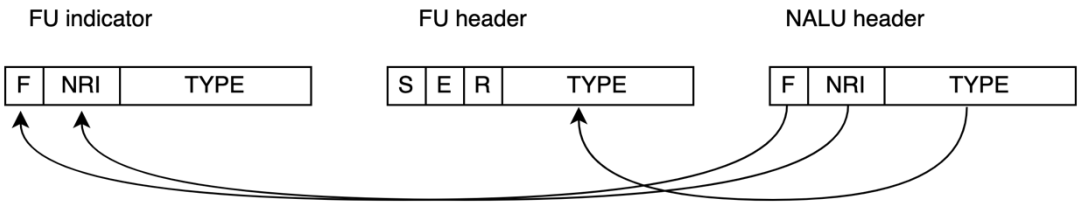

图 5-3 FU-A 分片方式设置示意图

FU indicator 中 F 字段拷贝自 NALU header 中的 F 字段在 H.264 协议中默认为 0，用于适应不同种类的网络环境。NRI 字段拷贝自 NALU header 的 NRI，用于标记 NALU 的重要性。TYPE 字段值设置为 28，代表这是 FU-A 分片形式的 NALU。FU header 中的 TYPE 使用原 NALU header 中 TYPE，标识本单元内的原始序列载荷 RBSP（Raw Byte Sequence Payload）类型。S 位标识 FU 片开始，E 标识结束，R 保留位值默认为 0。

在完成分片操作后，添加 RTP 包头，其中负载类型参数由元件初始化参数 payload 定义。在本系统中，设置为 H.264 协议。由此得到 RTP 编码整体功能如图 5-4 所示。

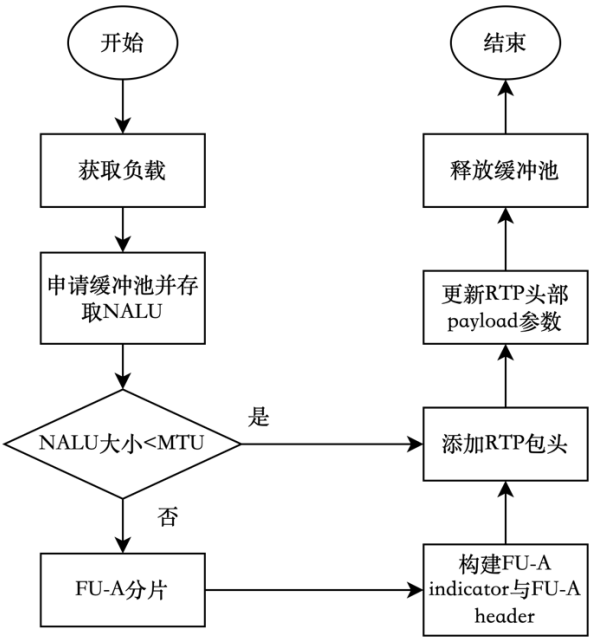

图 5-4 RTP 编码元件功能实现

控制命令处理功能进行帧率的设置，从而完成 RTP 包时间戳增量值的设置。在视频领域，时钟频率为 90000Hz，两帧之间 RTP timestamp 的增量 = 时钟频率 / 帧率。因此可以通过设置帧率完成时间戳值增量的设置，通过控制命令处理函数对元件参数进行更新。

（3）RTP 解码元件

RTP 解码元件功能与 RTP 编码元件功能相反，主要完成 RTP 的拆包。同时在送到解码器解码之前，需要进行组帧操作，合成完整的 NALU。在完成 RTP 解码元件的创建时，要对元件的初始化参数进行设置。

数据处理函数在收到数据后，首先进行数据类型的校验，对于 RFC3984 的组帧进行处理，申请缓冲池存入数据负载。然后解析 RTP 头部，RTP 包头大小为 12B，然后去掉 RTP 包头，获取 RTP 负载，获取 RTP 负载的第一个字节，判

断 TYPE 字段是否为 28。若为 28，则进行 FU-A 分片方式进行 NALU 的组帧。获取第二个字节 FU header 中开始与结束标志位，若开始标志位为真，设置 NALU header，然后负载数据指针向后移动，直到找到结束标志位，将分片内容整合。若不为 28，则进行 single NALU 的组帧。最后判断 NALU 是否为起始 NALU,若是，则在 NALU 头前加上起始标识符 0x00000001 四个字节，标识起始 NALU。否则在 NALU 之间用 0x000001 三个字节进行间隔。具体过程如图 5-5 所示。

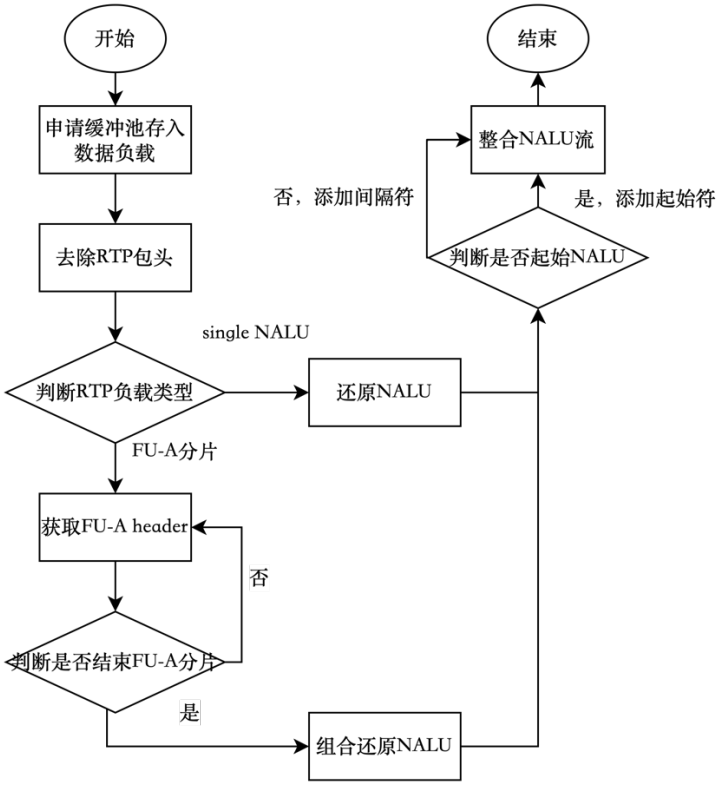

图 5-5 RTP 解码元件功能实现

控制命令功能主要设置申请最大 H.264 缓冲池的大小，用于确定整合后 NALU 所能存放的最大缓冲。在控制命令校验合格后，对元件中最大缓冲池大小参数进行更新。

（4）Video 解码元件

Video 解码元件主要进行 NALU 的解码操作，将 H.264 视频流解为 YUV 图像，与 Video 解码元件的功能对应。在进行元件的创建时，主要进行元件参数的配置、数据处理函数的设置、控制命令函数的设置。其中元件的配置参数列表主要包含流号、视频协议、视频格式、解码后图像宽高、RFC 打包模式、旋转角度和解码统计参数等信息。

数据处理函数在收到数据后，首先进行数据类型的校验，获取数据负载。然后创建空缓冲池，将负载内容存入缓冲池中。对输入到解码器的视频流参数进行设置，包括流的地址、流的大小、帧结束标志，是否为完整 NALU 等。然后初始

化输出参数，获取 NALU 头，并判断 NALU 类型，判断是否是 SPS（7）或者 PPS（8）类型。如果是，则进行 SPS 或 PPS 的解析，否则，进行普通类型帧解码。在解码完成后，根据输出参数判断解码过程是否成功，并更新解码统计信息。若成功，则清空 NALU 缓冲池并退出，否则，打印解码错误日志。具体过程如图 5-6 所示。

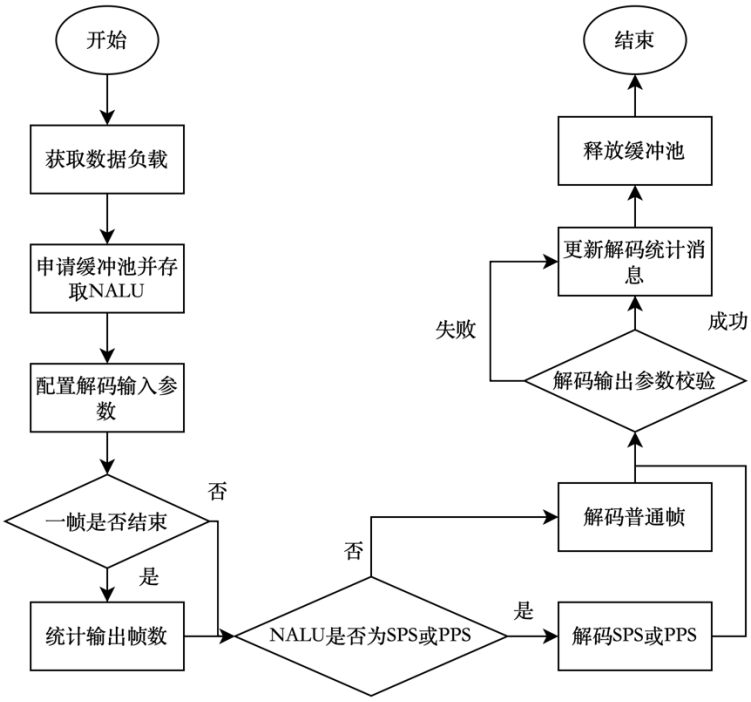

图 5-6 Video 解码元件功能实现

控制命令处理功能可接受控制命令为设置图像旋转角度，在完成控制命令的校验后，对元件旋转角度参数进行更新。

（5）Resize 元件

Resize 元件主要进行 YUV 图像的缩放，其元件的配置参数主要有流号、源图像的宽高、目的图像的宽高、缩放质量的等级和缩放统计消息等。

数据处理函数中首先进行数据类型的校验，判断是否为 YUV 类型数据，获取数据负载并对输入图像的格式进行检验，判断是否为 YUV420 格式，如果不是，则进行图像格式转换。然后获取输入输出图像参数，进行输入输出图像大小的比较，若大小相同，则不用缩放，直接结束。否则申请 YUV 图像缓冲池并将数据负载填充到缓冲池中，将图像缓冲池、图像缩放参数作为输入参数传入到 libyuv 库中进行图像缩放并判断缩放是否成功，更新缩放统计消息。最后向下一个元件发送缓冲后，将缓冲池释放。具体流程如图 5-7 所示。

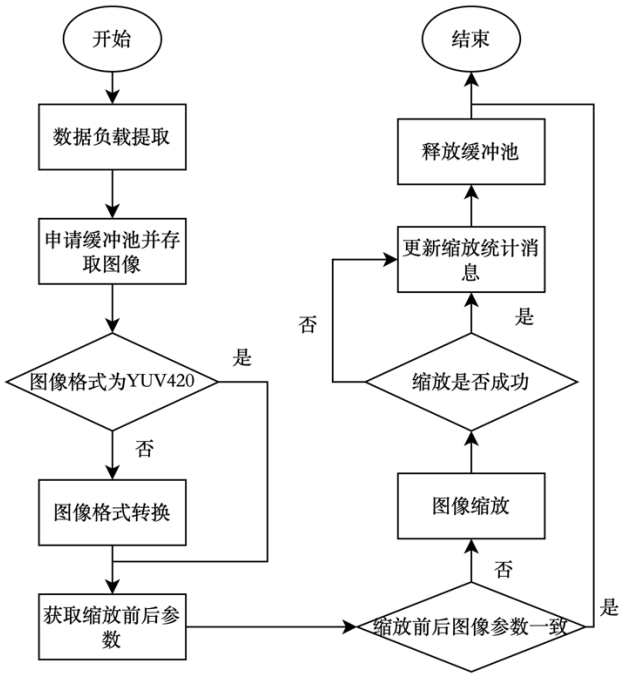

图 5-7 Resize 元件功能实现

控制命令功能主要进行图像缩放质量等级的设定。在这里使用的是 libyuv 的开源库，在 libyuv 中定义了四种缩放质量。本文针对特定场景的缩放，定义第五种缩放质量 kFliterSelfOpt，当处于这种缩放质量下时，启用自定义缩放优化。

5.2 多画面模式的实现

5.2.1 多画面模式整体时序实现

在多画面模式中，根据之前对多画面模式视频流传输控制的设计，首先对多画面模式的时序进行实现，构建多画面视频流处理 pipeline。

会议控制系统下发多画面模式的消息后，首先完成消息的校验，然后根据编码、解码流号判断编码、解码、适配节点是否存在。若存在，则根据消息内容对节点参数进行更新。若不存在则进行节点的创建，首先进行编码节点创建，然后进行根据子画面流号创建子画面对应的解码节点。编码节点的创建包含 Video 编码、RTP 编码元件的创建，解码节点的创建包含 RTP 解码、Video 解码元件的创建，与单画面视频流编、解码节点创建过程一致。

在完成编码、解码节点的创建后，进行适配节点的创建。首先创建多画面处理元件，根据多画面模式消息对多画面处理元件进行参数初始化并进行元件的创建。在元件创建后，进行子画面 Resize 元件的创建。根据多画面模式消息，获取子画面解码流号，根据子画面解码流号获取 Video 解码元件解码后图像参数，同时获取多画面合成后子画面参数，进行子画面 Resize 元件的参数初始化，并创建

Resize 元件。最后分别将子画面 Resize 元件与多画面处理元件进行连接激活，并更新适配节点的参数。

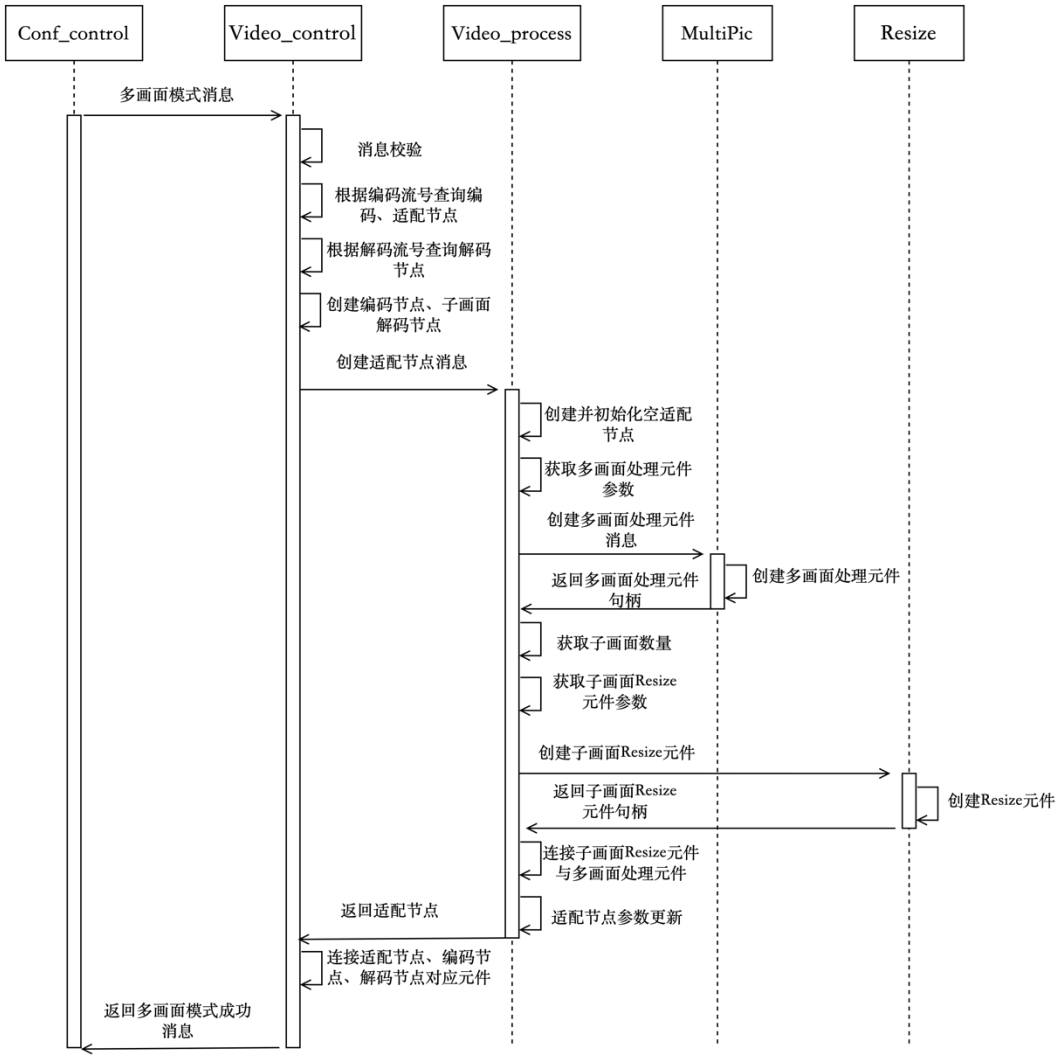

图 5-8 多画面模式整体时序图

在完成适配节点的创建后，根据编码流号，将适配节点中多画面处理元件与 Video 编码元件连接。根据子画面解码流号，分别将适配节点 Resize 元件与对应流号上的 Video 解码元件进行连接，从而构建好多画面视频流处理 pipeline。具体实现时序图如图 5-8 所示。

5.2.2 多画面处理元件功能的实现

在多画面元件中，主要完成多画面的拼接工作，完成合成多画面后帧率的统一设定。元件的创建工作与其他元件的创建过程相同，不同的地方在于元件的配置参数，数据处理和控制命令处理功能。其中元件的参数配置包括多画面对应的流号、合成后的多画面样式、合成后的多画面宽和高、合成多画面的帧率、多画

面模式、视频协议、子画面队列长度和子画面数据。其中子画面数据包括子画面流号、子画面图像宽高、子画面在整个大画面中的偏移值、子画面帧率等信息。具体的参数配置见下表 5-1。

表 5-1 多画面处理元件初始化参数

参数	参数注释
uint32 stmEncNo	视频编码流号
uint32 mode	多画面格式
uint32 width	多画面宽
uint32 height	多画面高
uint32 protocol	视频协议
uint32 subQueLength	子画面队列长度
uint32 framerate	合成后帧率
SUBPIC_PARMS subPicParams[SUBPIC_NUM]	子画面数据

数据处理函数完成子画面的拼接及多画面合成后帧率的控制，在收到数据后，因为采取的是 10ms 定时驱动的方式，因此需要将 10ms 内收到的子画面消息统一存储到子画面队列中，判断子画面是否已经都接收到消息，然后等待子画面都到齐。根据合成后帧率设置，首先将帧率处理为常用帧率，预处理为到 5-60 之间 5 的整数倍帧率。然后，获取图像合成标志，若为 True，则进行合成。每过 10ms，10ms 计数器自增加 1，并判断定时驱动编码标志是否为 True。具体过程如图 5-9 所示。

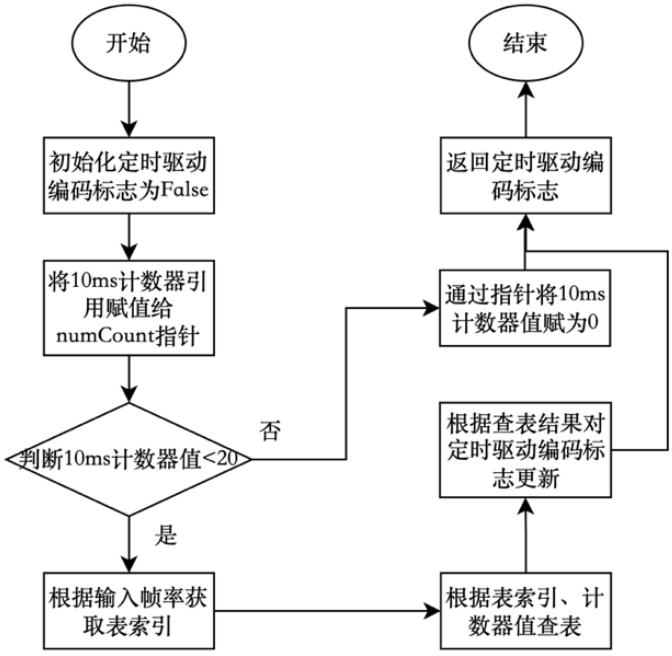

图 5-9 获取定时驱动编码标志

若定时驱动编码标志为真，则开始进行多画面的合成。在合成多画面中，首先遍历子画面，获取到每一个子画面的最新帧，并保存起来，然后根据合成后多

画面属性，进行多画面画板的申请，多画面画板参数包括 YUV 格式、图像宽、图像高、图像 stride、图像缓存大小、图像缓存基地址、YUV 分量指针。

然后进行子画面图像的拷贝，遍历每个子画面，对于存在的子画面，根据子画面对应的偏移值进行图像的拷贝。合成图像后，获取合成图像参数，与输入参数进行比对，判断图像是否合成成功。若成功，则判断图片宽高是是否符合编码器输入要求，将合成后图像进行扩边处理，以满足编码器的要求。最后将合成后多画面发送到下一个元件。

控制命令主要进行多画面模式的设置、多画面合成帧率的设置。在视频会议子系统中中定义不同的多画面模式，最大 25 画面，对每一种多画面模式，设计不同的子画面参数。在收到控制命令消息后，对控制命令进行校验，合格后对元件多画面模式、合成帧率参数进行更新。

5.3 缩放元件性能优化的实现

在缩放元件中，对缩放质量的等级进行了定义，当缩放质量等级为 kFliterSelfOpt 时，进入到自定义的缩放算法中。在上一章中，已经对缩放优化的的策略进行了详细设计，本节主要进行缩放优化的具体实现。

根据缩放比以及缩放质量等级，进入到缩放优化函数 ScalePlaneDownRow720pTo476_268 中，该函数实现步骤如下所示。

算法：缩放优化自定义函数 ScalePlaneDownRow720pTo476_268

输入：源图像宽、高、步幅、起始像素指针，目的图像宽、高、步幅、起始像素指针，缩放优化模式
输出：目的图像指针

1. // 定义行缩放函数 ScaleRowDown3To1_Box、ScaleRowDown2To1_Box 指针，参数包括源图像指针、步幅，目的图像指针、步幅
   ScaleRowDown3To1_Box ← ScaleRowDown3To1_Box_C
   ScaleRowDown2To1_Box ← ScaleRowDown2To1_Box_C
2. // 若定义 ARM 下 NEON 指令集，更新行缩放函数指针
   #if define(HAS_SCALEROWDOWN3TO2_NEON)
   ScaleRowDown3To1_Box ← ScaleRowDown3To1_Box_NEON
   ScaleRowDown2To1_Box ← ScaleRowDown2To1_Box_NEON
3. // 若定义 x86 下 SSSE3 指令集，更新行缩放函数指针
   #if define(HAS_SCALEROWDOWN3TO2_SSSE3)
   ScaleRowDown3To1_Box ← ScaleRowDown3To1_Box_SSSE3
   ScaleRowDown2To1_Box ← ScaleRowDown2To1_Box_SSSE3
4. // 缩放实现

行缩放函数默认是指向 C 语言进行实现的，在判断有对应的 SIMD 指令集时，更新函数指针。通过实现 C 语言行缩放函数，可以提高函数的健壮性，同时在可以更快捷地实现汇编函数。

在实现 C 语言行缩放函数后，进行 x86 下 SSSE3 指令集汇编函数实现。首先对设计好的偏移值进行定义。

uvec8 OffsetRow1 = {0,1,1,2,3,4,4,5,6,6,7,7,0,0,0,0}

uvec8 OffsetRow2 = {0,1,1,2,3,4,4,5,6,7,7,8,9,10,10,11}

将偏移值分别存储到寄存器 xmm3 与 xmm4 中，根据偏移值将源图像像素值存取到寄存器当中。汇编函数 ScaleRowDown3To1_Box_SSSE3 中核心汇编实现过程如下。

（1）使用占位符标记要用到的内存地址。

```
:“+r”(src_ptr1),       // %1
 “+r”(src_ptr2),      // %2
 “+r”(dst_ptr),       // %3
 “+r”(src_stride),    // %4
 “+r”(dst_stride)     // %5
```

（2）从占位符为 0、1、2 的地方分别存取 128bit 到 xmm0、xmm1、xmm2 寄存器中。

```
movdqu MEMACCESS(0) %%xmm0
movdqu MEMACCESS(1) %%xmm1
movdqu MEMACCESS(2) %%xmm2
```

（3）根据偏移值 OffsetRow1 对 xmm0、xmm1、xmm2 中对应像素进行偏移。

```
pshufb %%xmm3, %%xmm0
pshufb %%xmm3, %%xmm1
pshufb %%xmm3, %%xmm2
```

（4）对同一寄存器 xmm0、xmm1、xmm2 内像素值进行运算。

```
pmaddubsw %%xmm0, %%xmm0
pmaddubsw %%xmm1, %%xmm1
pmaddubsw %%xmm2, %%xmm2
```

（5）对不同寄存器之间对应像素值进行运算。

```
paddsw %%xmm1, %%xmm0
paddsw %%xmm1, %%xmm0
paddsw %%xmm2, %%xmm0
```

（6）将寄存器 xmm0 中有符号数压缩为无符号数，同时进行字节对齐。

```
packuswb %%xmm0, %%xmm0
```

（7）将计算后结果存放到目的地址内存中。

```
movl %%xmm0 MEMACCESS(3)
```

（8）将源图像与目的图像像素地址进行偏移。

```
lea MEMLEA(0x3, 3) %3
lea MEMLEA(0x8, 0) %0
lea MEMLEA(0x8, 1) %1
lea MEMLEA(0x8, 2) %2
```

然后利用偏移值 OffsetRow2 进行目的行剩余像素值的计算，计算过程与 OffsetRow1 类似。

ScaleRowDown2To1_Box_SSSE3 函数主要实现两行图像的计算，与 ScaleRowDown3To1_Box_SSSE3 函数不同点在于不需要定义源图像第三行起始像素指针 src_ptr2，同时减少了寄存器 xmm2 的使用。对不同寄存器之间对应像素进行相加操作时，只使用两个寄存器，累计进行 8 个像素值的相加，然后向右移 3 位。

在完成整体的图像优化后，将缩放后的图像与不做优化的图像进行时间性能与图像质量的比较，这部分比较结果在后续的测试篇章中进行叙述。

5.4 本章小结

在本章中，首先对视频处理子系统中单画面视频流的处理过程进行了实现，包括单画面视频流处理的时序过程，并对在这个过程中需要用到的元件进行功能实现。然后对多画面模式进行实现，包括整个多画面模式的时序过程，以及多画面视频流处理过程中关键元件多画面处理元件的功能实现。最后对缩放元件的优化过程进行了实现，主要为汇编函数中关键过程的实现。

第六章 系统测试

本章主要对视频处理子系统的功能与非功能进行测试。功能测试包括单画面视频流处理和多画面视频流整体功能测试，并对其中每一个功能元件进行测试。最后进行非功能测试，包括安全测试、系统可维护性测试和系统性能测试。

6.1 测试环境

测试系统：RH2288H V3 服务器（CPU: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz；内存：64GB）

测试工具：gtest、wireshark、VCAM、TCPdump、codeDEX

测试设备：TE desktop 软终端

测试环境的搭建，首先将含有视频处理子系统的 MCU 安装到 RH2288H V3 服务器上。然后进行 MCU 的配置操作，包括 MCU 的 IP 地址配置、GK 服务器配置等。完成 MCU 的安装后，通过 MobaXterm 进入到对应 IP 地址的 MCU 后台，查看 MCU 是否正常启动，各进程是否正常拉起。

在 MCU 正常启动后，登录 SMC 管理后台，配置整个 SMC 中可以使用的 MCU 资源。通过 MCU 中配置好的账户密码进行连接，在完成 MCU 资源的添加后，即可进入 SMC 前台进行会议的召开。在 SMC 平台上进行会议的召开时，需要选择对应的 MCU，从而根据该 MCU 的能力实现会议的召开。系统测试组网如图 6-1 所示。

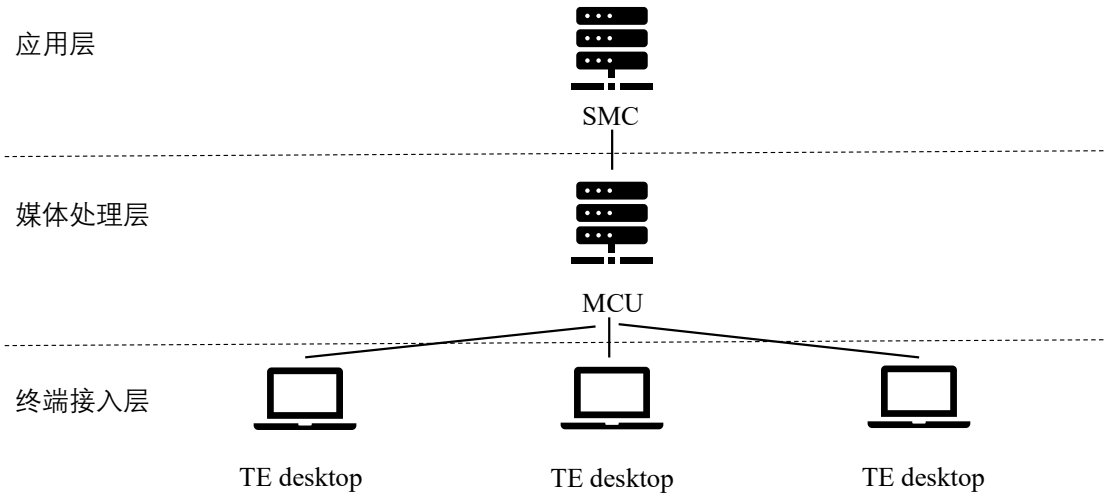

图 6-1 系统测试组网

本文使用灰盒测试与白盒测试相结合的方法来实现系统的功能与性能的检测，通过测试用例来确保系统功能的稳定性，每次上库修改代码后须确保能够通过所有的测试用例，才能合入代码。灰盒测试关注输出对于输入的正确性，既要考虑产品设计要求，又要考虑代码功能实现的效果。因此，利用视频处理子系统

的模块化属性，对各个模块展开测试，同时，又聚焦模块之间的相互协调，利用灰盒测试将多个模块进行组合然后进行测试，从而确保系统整体的质量。白盒测试主要进行函数级别的测试，因此可以通过白盒测试对性能优化进行测试，编写不同缩放比例下的测试用例，根据测试用例的结果来对性能优化结果做定量统计。

6.2 系统测试方案

针对视频处理子系统，在这里本文将分别针对功能测试与非功能测试展开。其中功能测试主要利用灰盒测试对单画面视频流、多画面模式整体功能进行测试，对单画面视频流中涉及的各个元件功能进行测试，对多画面模式中核心元件多画面处理元件进行测试。非功能测试围绕安全测试、系统可维护性测试和系统性能测试展开，安全测试包括代码编写过程中的可能存在的安全问题，如内存泄漏。系统可维护性测试主要是系统的后期维护是否可行，如日志、黑匣子文件记录能否及时记录问题，能否记录关键日志等。性能测试主要是在缩放优化功能开启后，进行缩放优化效果的测试，主要利用白盒测试用例对不同场景缩放进行测试，利用 PSNR 值判断缩放质量，对缩放优化时间作定量分析。

6.2.1 功能测试

功能测试主要验证系统是否能够正常完成需求中所设定的功能，分别对各模块协同的整体功能与各模块特有功能进行测试。

针对灰盒测试，首先搭建灰盒测试框架，利用测试代理进行进程间的通信。新建一个测试代理进程，使用测试代理进程模拟会议控制子系统给视频处理子系统发消息，进行 Rtp 解码、Video 解码、Resize、多画面处理、Video 编码、Rtp 编码等元件的创建，在元件创建成功后，进行元件的连接激活，构成完整的视频流传输链。在构建完成后，利用 socket 进行码流的发送。经视频处理子系统处理后，监听目的端 IP，利用 TCPdump 将处理后的码流进行抓包分析。下面就各个灰盒测试用例进行详细的解释分析。

（1）单画面视频流整体测试用例

该用例主要测试单画面视频流能否正常传输。根据单画面视频流的流程图，输入为一段 RTP 码流，输出为处理后的 RTP 码流，因此可以通过处理后的码流来判断单画面视频流是否正确处理。在进行灰盒测试用例测试时，不需要启动 SMC 及软终端来构建视频会议，只需要启动 MCU，并将灰盒测试代理进程拉起，将对应会议控制进行强制关闭，利用测试代理进程模拟会控进行消息发送，然后执行灰盒测试用例。对于测试用码流，帧率为 30fps，码率为 250kbps，大小为 11MB，利用 wireshark 分析参数如图 6-2、图 6-3 所示。

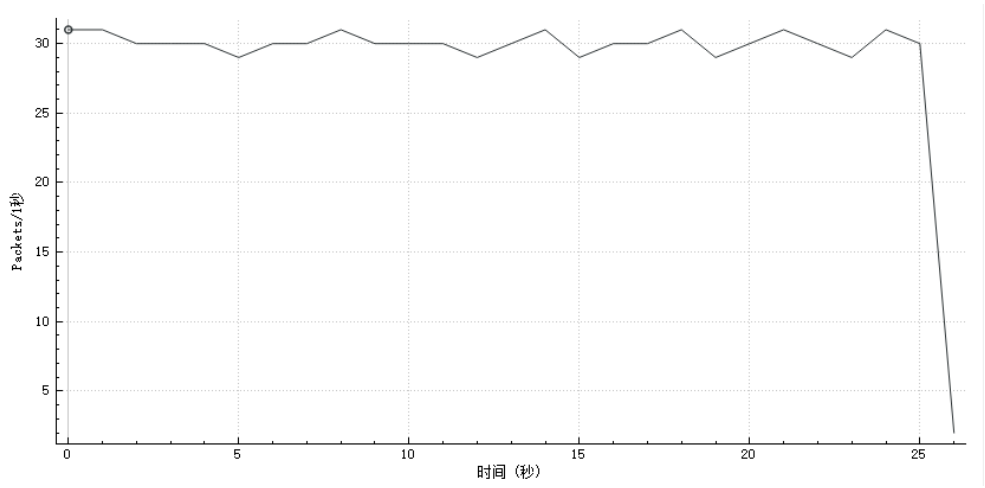

图 6-2 测试输入码流发包速率

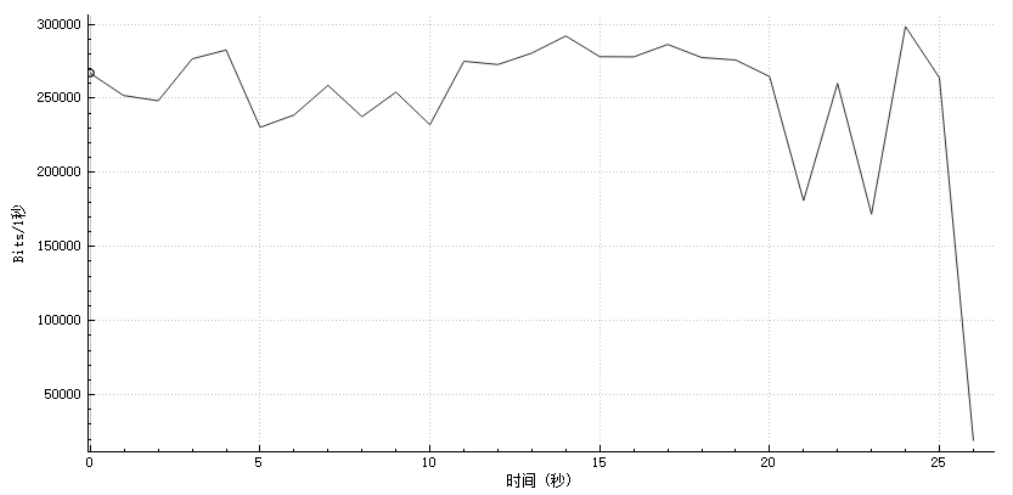

图 6-3 测试输入码流码率

在该灰盒用例中，首先进行发送码流缓冲区的设置，大小为 80MB。然后进行视频处理系统的配置，配置目的端 IP，依次进行编码节点、适配节点、解码节点的配置，构成完整的处理流。最后新建一个媒体交换的进程，给视频处理子系统发数据。处理后的码流参数与已知正确码流参数做比对，若一致，则通过该灰盒测试用例。

（2）多画面模式整体测试用例

该测试用例主要测试多画面模式的功能是否正常，与单画面视频流测试过程类似。在这里输入测试码流，通过设置不同的多画面模式，最大 25 画面，每个子画面都采用同一段输入测试码流，来判断多画面拼接以及合成帧率是否控制正常。

在该灰盒用例中，在完成编码节点、适配节点、解码节点的配置后，构建好多画面视频流传输处理 pipeline，然后利用 socket 由媒体交换进程来进行码流的

发送。利用 TCPdump 来监听目的端口 IP，将抓取后的 pcap 包参数与已知正确码流参数做比对，若一致，则通过该灰盒测试用例。

（3）RTP 解码元件测试用例

该用例主要针对 RTP 解码元件的功能进行测试，主要将 RTP 码流解为 H.264 视频流，同时对其中涉及到的控制命令功能进行测试。具体的测试对象如下表 6-1 所示。

表 6-1 RTP 解码元件测试对象表

测试对象	输入	预期输出	具体描述
输出 H.264 文件句柄	文件名	创建 H.264 文件句柄成功	测试创建输出文件句柄
元件的创建、初始化	元件类型名 RTP_DEC、元件名、元件初始化参数	元件创建、初始化成功，返回 RTP 解码元件句柄	测试元件是否创建、初始化成功
元件数据处理	RTP 解码元件句柄、输入的 pcap 文件、RTP 解码参数	RTP 包组帧成功，返回处理成功消息	测试数据处理是否完成 RTP 码流的处理
元件控制命令处理	RTP 解码元件句柄、控制命令 ID、控制命令内容	元件控制成功，返回处理成功消息	测试元件控制命令处理功能

（4）Video 解码元件测试用例

Video 解码元件测试用例主要对 Video 解码元件的功能进行测试，对控制命令功能进行测试，输入为 H.264 视频流，输出为 YUV 图片。具体的测试对象如下表 6-2 所示。

表 6-2 Video 解码元件测试对象表

测试对象	输入	预期输出	具体描述
输出 YUV 文件句柄	文件名	创建 YUV 文件句柄成功	测试创建输出文件句柄
元件的创建、初始化	元件类型名 VIDEO_DEC、元件名、元件初始化参数	元件创建、初始化成功，返回 Video 解码元件句柄	测试元件创建、初始化是否成功
元件数据处理	Video 解码元件句柄、输入的 H.264 文件、视频解码参数	视频解码成功，返回处理成功消息	测试数据处理是否完成 H.264 流的处理
元件控制命令处理	Video 解码元件句柄、控制命令 ID、控制命令内容	元件控制成功，返回处理成功消息	测试元件控制命令处理功能

（5）Resize 元件测试用例

Resize 元件的主要功能就是对 YUV 图像进行缩放，同时根据不同缩放质量等级，选择不同缩放方法。因此需要对相关功能进行测试，元件输入为 YUV 图像，输出为相同帧数的 YUV 图像。具体测试对象如下表 6-3 所示。

表 6-3 Resize 元件测试对象表

测试对象	输入	预期输出	具体描述
输出 YUV 文件句柄	文件名	创建 YUV 文件句柄成功	测试创建输出文件句柄
元件的创建、初始化	元件类型名 RESIZE、元件名、元件初始化参数	元件创建、初始化成功，返回 Resize 元件句柄	测试元件创建、初始化是否成功
元件数据处理	Resize 元件句柄、输入的 YUV 文件、缩放参数	图像缩放成功，返回处理成功消息	测试数据处理是否完成 YUV 的缩放
元件控制命令处理	Resize 元件句柄、控制命令 ID、控制命令内容	元件控制成功，返回处理成功消息	测试元件控制命令处理功能

（6）Video 编码元件测试用例

Video 编码元件的主要功能是对 YUV 图像进行编码工作，因此测试用例主要对元件的主要功能及控制命令处理等进行测试，元件的输入为 YUV 文件，输出为 H.264 视频流文件。具体的测试对象如下表 6-4 所示。

表 6-4 Video 编码元件测试对象表

测试对象	输入	预期输出	具体描述
输出 H.264 文件句柄	文件名	创建 H.264 文件句柄成功	测试创建输出文件句柄
元件的创建、初始化	元件类型名 VIDEO_ENC、元件名、元件初始化参数	元件创建、初始化成功，返回 Video 编码元件句柄	测试元件创建、初始化是否成功
元件数据处理	Video 编码元件句柄、输入的 YUV 文件、编码参数	图像编码成功，返回处理成功消息	测试数据处理是否完成 YUV 的编码
元件控制命令处理	Video 编码元件句柄、控制命令 ID、控制命令内容	元件控制成功，返回处理成功消息	测试元件控制命令处理功能

（7）RTP 编码元件测试用例

RTP 编码元件的主要功能是完成 H.264 视频流的封包工作，元件输入为 H.264 视频流，输出为包含 RTP 码流的 pcap 文件。测试用例主要对元件的功能及控制命令进行测试。具体的测试对象如下表 6-5 所示。

表 6-5 RTP 编码元件测试对象表

测试对象	输入	预期输出	具体描述
输出 pcap 文件句柄	文件名	创建 pcap 文件句柄成功	测试创建输出文件句柄
元件的创建、初始化	元件类型名 RTP_ENC、元件名、元件初始化参数	元件创建、初始化成功，返回 RTP 编码元件句柄	测试元件创建、初始化是否成功
元件数据处理	RTP 编码元件句柄、输入的 H.264 文件、编码参数	H.264 视频封包成功，返回处理成功消息	测试数据处理是否完成 H.264 数据的处理
元件控制命令处理	RTP 编码元件句柄、控制命令 ID、控制命令内容	元件控制成功，返回处理成功消息	测试元件控制命令处理功能

（8）多画面处理元件测试用例

多画面元件的主要功能是将根据多画面模式的需求将 YUV 图像进行拼接，对合成图像帧率进行控制，因此输入为 YUV 文件，输出也为 YUV 文件。具体的测试对象如下表 6-6 所示。

表 6-6 多画面处理元件测试对象表

测试对象	输入	预期输出	具体描述
输出 YUV 文件句柄	文件名	创建 YUV 文件句柄成功	测试创建输出文件句柄
元件的创建、初始化	元件类型名 MP_SPLICE、元件名、元件初始化参数	元件创建、初始化成功，返回多画面处理元件句柄	测试元件创建、初始化是否成功
元件数据处理	多画面处理元件句柄、输入的 YUV 文件、源图像参数、目的图像画面参数	合成多画面正常，返回处理成功消息	测试数据处理是否完成 YUV 数据的拼接
元件控制命令处理	多画面处理元件句柄、控制命令 ID、控制命令内容	元件控制成功，返回处理成功消息	测试元件控制命令处理功能

6.2.2 非功能测试

在完成系统的功能测试后，进行非功能测试，主要包括安全测试、系统可维护性测试、系统性能测试等。

（1）安全测试

使用代码扫描工具 codeDEX 对代码规范、代码安全进行扫描，包括指针判空、内存泄漏等问题。

（2）系统可维护性测试

在系统可维护性测试中，本文通过在长时间进行视频会议的情况下，观察会议运行效果，观察是否黑匣子文件，同时对运行中的日志进行分析，判断关键步骤是否正常打印日志。

（3）系统性能测试

主要对优化前后系统性能进行测试，利用白盒测试用例，定义不同缩放比例的测试对象，统计优化前后的时间。因为对一帧图像处理时间过短，在这里作4000 次循环来减少实验误差，具体的测试用例如表 6-7 所示。同时，计算优化后缩放图像与优化前缩放图像的 PSNR 值，要求 PSNR 值大于 20，才能说明优化后缩放的图像图像质量合格。

表 6-7 缩放优化测试用例

编号	测试用例名
1	OptimizeTest.1920_1080TO704_400_scale
2	OptimizeTest.1920_1080TO640_360_scale
3	OptimizeTest.1920_1080TO512_288_scale
4	OptimizeTest.1920_1080TO480_268_scale
5	OptimizeTest.1920_1080TO476_268_scale
6	OptimizeTest.1920_1080TO424_240_scale
7	OptimizeTest.1920_1080TO400_200_scale
8	OptimizeTest.1920_1080TO400_224_scale
9	OptimizeTest.1920_1080TO400_268_scale
10	OptimizeTest.1920_1080TO384_216_scale

6.3 测试结果及分析

首先是功能测试结果，在配置好测试框架后，依次运行各个功能测试用例，测试结果如下表 6-8 所示。

表 6-8 功能测试用例结果表

测试用例	测试结果
（1）单画面视频流整体测试用例	测试用例通过，将目的端口上抓到的包解为 H.264 后播放，画面正常
（2）多画面模式整体测试用例	测试用例通过，将抓到的包解为 H.264 后播放，在不同多画面模式下，画面拼接正常，画面播放流畅。其中三画面模式下输出码流参数如图 6-4、图 6-5 所示。
（3）RTP 解码元件测试用例	测试用例通过，RTP 解码元件创建初始化成功，数据处理与控制命令处理功能正常
（4）Video 解码元件测试用例	测试用例通过，Video 解码元件创建初始化成功，数据处理与控制命令处理功能正常
（5）Resize 元件测试用例	测试用例通过，Resize 元件创建初始化成功，数据处理与控制命令处理功能正常
（6）Video 编码元件测试用例	测试用例通过，Video 编码元件创建初始化成功，数据处理与控制命令处理功能正常
（7）RTP 编码元件测试用例	测试用例通过，RTP 编码元件创建初始化成功，数据处理与控制命令处理功能正常
（8）多画面处理元件测试用例	测试用例通过，多画面处理元件创建初始化成功，数据处理与控制命令处理功能正常

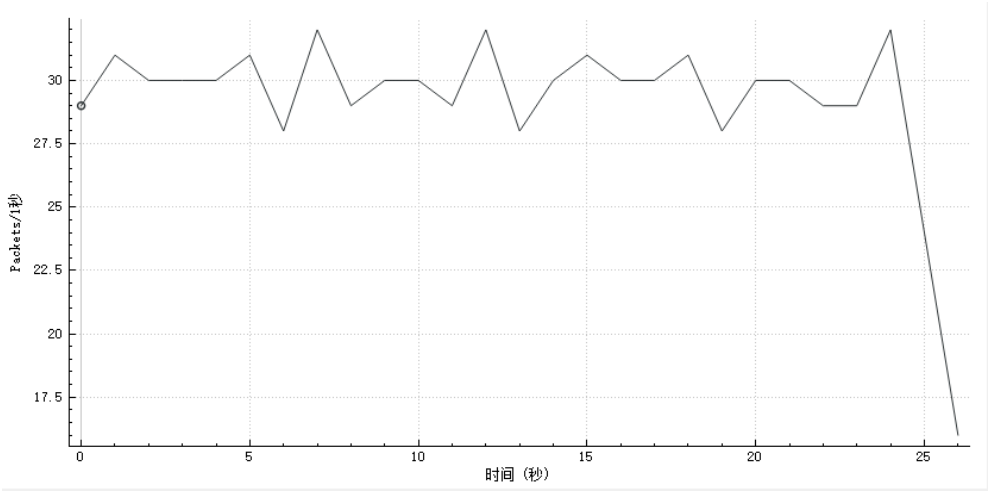

图 6-4 三画面模式下输出码流发包速率

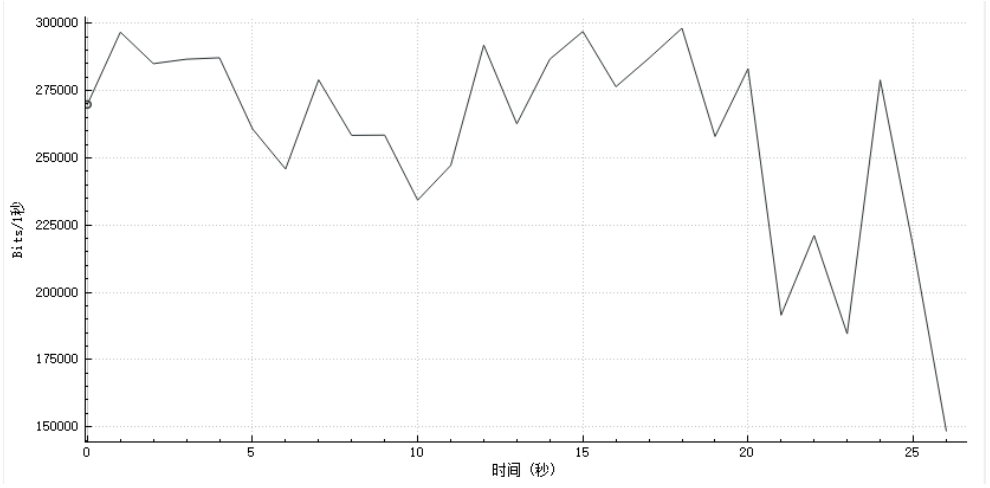

图 6-5 三画面模式下输出码流码率

通过测试用例结果可以看到，所有灰盒测试用例均通过，三画面模式输出码流在考虑网络波动因素下，参数基本与输入码流参数一致，系统的功能得到了实现。然后对非功能测试结果，如下表 6-9 所示。

表 6-9 系统非功能测试结果表

测试对象	测试结果
（1）安全测试	系统所有代码通过 codeDEX 安全扫描，不存在空指针、内存泄漏、使用非安全函数等问题
（2）系统可维护测试	系统长稳运行中，未报黑匣子错误。系统日志完备，关键日志详细，能够打印流节点、元件创建等关键信息
（3）系统性能测试	在使用缩放优化后，10 个缩放测试用例通过，PSNR 值均大于 20，缩放优化前后所需时间如表 6-10 所示，优化效果显著。在接入 25 路 1080P30 的会议后，优化前 CPU 占用率为 92%左右，优化后 25 路 1080P30 会议 CPU 占用率为 85%，满足系统规格需求

表 6-10 缩放性能优化结果表

用例	优化前（us）	优化后（us）	优化比例	PSNR 值
用例 1	26248.25	572.25	0.978	27.924
用例 2	23204.25	865.00	0.963	48.836
用例 3	21693.50	457.00	0.979	38.257
用例 4	20374.25	206.50	0.989	33.292
用例 5	21082.25	724.25	0.966	25.856
用例 6	20242.00	760.75	0.962	35.859
用例 7	19431.50	629.00	0.968	36.273
用例 8	19942.75	660.25	0.967	37.026
用例 9	20907.25	885.50	0.958	31.734
用例 10	19050.00	365.50	0.981	35.351

通过测试结果可以看到，系统安全与可维护性得到保证，同时系统性能优化效果显著。优化前，采用原生的 libyuv 开源库进行图像缩放，通过使用本文设计的优化策略，优化后图像处理时间较优化前提升高达 97%左右。同时，缩放优化后的图像相比于优化前缩放的图像，PSNR 值均大于 20，缩放图像质量合格。

通过功能测试与非功能测试，表明本文所设计实现的视频处理子系统达到了既定目标，实现了单画面与多画面视频流的传输功能。同时系统性能优化效果显著，工程应用价值高。

6.4 本章小结

本章对视频处理子系统进行了系统而全面的测试。首先是利用灰盒测试对系统功能进行测试，对单画面、多画面视频流整体处理流程进行测试，并对其中的功能元件进行测试。然后利用白盒测试对图像缩放的性能进行测试，量化缩放前后图像处理时间。经过测试，本系统功能完善，同时图像缩放优化效果显著。

第七章 总结与展望

7.1 总结

本文主要研究内容为视频会议中视频流处理控制过程。结合MCU应用背景，从系统需求出发，本文设计并实现了基于GStreamer框架的视频处理子系统，提供了高效可控的视频流处理控制方案。同时针对视频流处理过程中的性能问题，设计并实现了YUV图像缩放优化策略，显著地缩短了图像缩放处理时间。本文的主要工作总结如下：

（1）设计实现了单画面RTP码流的端到端处理控制方案。基于GStreamer框架，设计实现了RTP解码、视频解码、图像缩放、视频编码、RTP编码等功能元件，然后设计视频流控制节点控制功能元件，实现了高效可控的单画面视频流处理控制过程。通过视频流控制节点，可以快速高效地构建视频流处理pipeline。通过功能元件的构建，将视频流处理过程模块化，并可通过元件控制命令处理函数，及时对视频流处理参数进行配置。

（2）针对多方视频会议的需求，在单画面视频流传输控制的基础上，设计实现多画面模式，实现了多画面模式下视频流传输与控制。通过设计实现多画面模式下多画面处理元件，实现了多会场画面的拼接。同时针对子画面帧率不统一的问题，设计实现了多画面合成帧率控制策略，使得合成画面帧率可控，方便后续的编码操作。

（3）利用SIMD对视频流处理过程中图像缩放进行了性能优化。目前国际上通用的YUV图像缩放库来自Google的libyuv开源库。本文结合实际场景，对缩放元件进行性能优化。基于SIMD指令集，设计并实现了YUV图像缩放策略，使得图像缩放处理时间显著缩短。

（4）进行详细的系统测试。利用灰盒与白盒测试相结合的方式，设计灰盒测试用例与白盒测试用例对视频处理子系统进行了功能测试与非功能测试。最终测试结果显示系统功能完善，同时在保证图像质量的前提下，图像缩放优化效果明显。

7.2 展望

本文对视频会议系统中核心的视频处理子系统进行了详细的设计与实现，实现了高效可控的视频流处理控制过程，并具备较好的可维护性。同时系统的性能也得到了提升，但是仍存在一些问题，主要包括：

（1）目前系统上使用的视频编解码协议是 H.264 协议，现在 H.265 协议逐步完善，更高效的视频编解码协议也要推出，因此系统需要提高不同协议的兼容性。

（2）在视频流处理过程中，视频编解码和图像缩放消耗了大量的性能。因此，可以利用专业级显卡对视频编解码和图像缩放过程进行加速，提升处理速度，这也是本文下一步主要要进行的工作。

（3）在性能优化方面，当前优化是针对所涉及到的具体场景比例而进行的，因此高度依赖实际业务需求，需要根据视频会议具体场景来设计性能优化方案，未来优化的方向是提高图像缩放优化的自适应性。

致谢

研究生的时光匆匆，三年的点滴时刻，仍历历在目。一路走来，虽然经历了疫情的起起伏伏，但更多仍是老师同学的帮助与支持，回首过去，有许多值得要深深感谢的人。

首先是我的研究生导师孔佑勇老师，饮其流者思其源，学其成时念吾师。孔老师专业知识渊博，待人和蔼可亲，体恤学生，十分有幸能够在求学路上遇到孔老师这样的良师益友。学习上，研一入学，从实习、论文开题、论文修改，都一直尽心尽力地帮助我。生活上，老师全心支持学生的成长发展，做学生成长路上的知心人，引路者。师恩难忘，祝孔老师工作顺利、生活幸福、桃李满天下。

其次感谢我的企业导师周清，天涯海角有尽处，只有师恩无穷期。周老师工程技术能力强，在企业实习的时间中，周老师无私地关心我指导我，使我获得了技术上的提升，对整个的软件开发流程有了一个更加清晰规范的认识，是我技术上的引路人，让我在企业实习的时间快乐而充实。师恩难忘，祝周老师工作顺利，生活幸福，事业进步。

然后要感谢周围的同学、室友，他们一路与我相伴，陪我共同经历成长路上的点滴，是同学的陪伴，让我成长路上不孤单。在此深深谢谢大家。

最后要感谢的就是我的家人，感谢他们一直以来的关心与支持，是他们的支持让我一路走下来。不论走到哪里，家都是最温暖的港湾，在今后及未来的道路上，我会一直努力下去。

参考文献

[1] 郭天锐. 高清视频会议系统中的多点控制单元[J]. 中国科技信息, 2022:91-92.

[2] Silva M A, Bertone R, Schäfer J. Topology distribution for video-conferencing applications[C]. 2019 10th International Conference on Networks of the Future (NoF). IEEE, 2019: 90-97.

[3] 卢月亮. 基于云通信业务的会议控制与维护系统的设计与实现[D]. 东南大学, 2019.

[4] 陈思平,司佳,仇亚俊,石超.基于华为 96 系列 MCU 电视墙的设计[J].集成电路应用,2019,36(01):16-17.

[5] Alimudin A, Muhammad A F. Online video conference system using WebRTC technology for distance learning support[C]. 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC). IEEE, 2018: 384-387.

[6] Jensen J F. Interactive television-a brief media history[C]. European Conference on Interactive Television. Springer, Berlin, Heidelberg, 2008: 1-10.

[7] 廖志文. 企业级视频会议服务质量保证系统资源预留技术研究[D]. 华南理工大学, 2020.

[8] ITU-T Recommendation H.320, Narrow-band visual telephone systems and terminal equipment[S]. International Telecommunication Union, 2004.

[9] ITU-T Recommendation H.261, Video codec for audiovisual services at p x 64 kbit/s[S]. International Telecommunication Union, 1993.

[10] ITU-T Recommendation H.323, Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service[S]. International Telecommunication Union, 1996.

[11] Hercog D. Protocols RTP/RTCP[M]. Communication Protocols. Springer, Cham, 2020: 343-344.

[12] ITU-T Recommendation H.263, video coding for low bit rate communication[S], International Telecommunication Union, 1996.

[13] ITU-T Recommendation H.264, Advanced video coding for generic audiovisual services[S], International Telecommunication Union, 2003.

[14] Janu N, Kumar A, Raja L, et al. Development of an efficient real-time H. 264/AVC advanced video compression encryption scheme[J]. Journal of Discrete Mathematical Sciences and Cryptography, 2021, 24(8): 2245-2255.

[15] Lee R, Venieris S I, Lane N D. Deep neural network–based enhancement for image and video streaming systems: a survey and future directions[J]. ACM Computing Surveys (CSUR), 2021, 54(8): 1-30.

[16] Zhou X C, Yu H W. A Novel Multimedia Player Based on Embedded System[C]. Applied Mechanics and Materials. Trans Tech Publications Ltd, 2014, 556: 1545-1548.

[17] Chatterjee A, Maltz A. Microsoft DirectShow: A new media architecture[J]. SMPTE journal, 1997, 106(12): 865-871.

[18] Media Foundation Multimedia Framework[EB/OL]. https://docs.microsoft.com/zh-cn/windows/win32/medfound/microsoft-media-foundation-sdk.

[19] GStreamer Open Source Multimedia Framework[EB/OL]. http://gstreamer.freedesktop.org/.

[20] Pozueco L, Álvarez A, García X, et al. Subjective video quality evaluation of different content types under different impairments[J]. New Review of Hypermedia and Multimedia, 2017, 23(1): 1-28.

[21] Rahman M. Windows 10 & Universal Windows Platform[M]. Beginning Microsoft Kinect for Windows SDK 2.0. Apress, Berkeley, CA, 2017: 261-294.

[22] Wang L, Zhang L, Ma Y. Gstreamer accomplish video capture and coding with PyGI in Python language[C]. 2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS). IEEE, 2017: 1-4.

[23] 王鑫, 左乐, 施振华, 苏成悦, 罗文俊, 任开众, 陈玉怀. GStreamer 音视频传输系统研究与实现[J]. 单片机与嵌入式系统应用, 2020,20(09):6-10.

[24] 王锋, 陆凯. 基于 Gstreamer 框架的 ffmpeg 流媒体编解码设计[J]. 电子技术与软件工程, 2019(1):3.

[25] Zhao X, Zhang X, Cheng X, et al. Research on Intelligent Target Detection and Coder-decoder Technology Based on Embedded Platform[C]. 2019 IEEE International Conference on Unmanned Systems and Artificial Intelligence (ICUSAI). IEEE, 2019: 210-215.

[26] 邓宣. 基于深度学习的视频编解码技术研究[D]. 电子科技大学, 2020.

[27] Li J, Li B, Xu J, et al. Fully connected network-based intra prediction for image coding[J]. IEEE Transactions on Image Processing, 2018, 27(7): 3236-3247.

[28] Wang H, Wu P, Tanase I G, et al. Simple, portable and fast SIMD intrinsic programming: generic simd library[C]. Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing. 2014: 9-16.

[29] Kumar P. Permutations on Illiac IV-type networks[J]. IEEE transactions on computers, 1986, 100(7): 662-669.

[30] Bogaevskiy D, Minenko M, Ezhov S, et al. Development and Implementation of the H. 264-Codec Deblocking Filter Based on the MIPS SIMD Architecture[C]. 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). IEEE, 2021: 246-251.

[31] Tagliavini G, Mach S, Rossi D, et al. Design and evaluation of SmallFloat SIMD extensions to the RISC-V ISA[C]. 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019: 654-657.

[32] Liu W, Liu W, Li M, et al. Fine-grained task-level parallel and low power H. 264 decoding in multi-Core systems[C]. 2018 IEEE 24th international conference on parallel and distributed systems (ICPADS). IEEE, 2018: 307-314.

[33] Ahovainio S, Mercat A, Vanne J. Live Demonstration: Multi-Laptop HEVC Encoding[C]. 2020 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2020: 1-1.

[34] 王哲诚,葛万成,吴晔. x264 视频编码器中参数设置对编码效率影响的研究[J]. 信息通信, 2018(2):40-42.

[35] 王相海,丛志环,方玲玲. IPTV 体系结构及其流媒体技术研究进展[J]. 通信学报, 2012,33(04):1-8.

[36] 毕厚杰, 王健. 新一代视频压缩编码标准:H.264/AVC.第 2 版[M]. 人民邮电出版社, 2009.

[37] Hore A, Ziou D. Image quality metrics: PSNR vs. SSIM[C].2010 20th international

conference on pattern recognition. IEEE, 2010: 2366-2369.

[38] 姚军财,申静,黄陈蓉. 基于多层 BP 神经网络的无参考视频质量客观评价[J]. 自动化学报, 2022,48(02):594-607.

[39] 刘恺,刘湘,常丽萍,陈滨,吴哲夫.基于 YUV 颜色空间和多特征融合的视频烟雾检测[J].传感技术学报, 2019,32(02):237-243.

[40] 朱秀昌,唐贵进. IP 网络视频传输[M]. 人民邮电出版社, 2017.

[41] Lipovac A, Lipovac V, Grbavac I, et al. Practical cross-layer testing of HARQ-induced delay variation on IP/RTP QoS and VoLTE QoE[J]. EURASIP Journal on Wireless Communications and Networking, 2021(1): 1-18.

[42] 邱海龙. 面向视频会议系统的抗丢包策略的研究与实现[D]. 东南大学, 2017.

[43] An Z, Zhang C, Zhang D, et al. Research on information transmission system of fire fighting UAV[C]. 2021 2nd International Conference on Computing, Networks and Internet of Things. 2021: 1-5.

[44] Altonen A, Räsänen J, Laitinen J, et al. Open-source RTP library for high-speed 4K HEVC video streaming[C]. 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP). IEEE, 2020: 1-6.

[45] Puri A, Chen X, Luthra A. Video coding using the H. 264/MPEG-4 AVC compression standard[J]. Signal processing: Image communication, 2004, 19(9): 793-849.

[46] Li J, Hao Z, Gao Q. Implementation of a multimedia communication system over IP network[C]. 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). IEEE, 2017: 141-145.

[47] Gómez D, Núñez J A, Montagud M, et al. ImmersiaTV: enabling customizable and immersive multi-screen TV experiences[C]. Proceedings of the 9th ACM Multimedia Systems Conference. 2018: 506-508.

[48] Angsuchotmetee C, Chbeir R, Cardinale Y, et al. A pipelining-based framework for processing events in multimedia sensor networks[C]. Proceedings of the 33rd Annual ACM Symposium on Applied Computing. 2018: 247-250.

[49] Ubik S, Travnicek Z. Comparison of Graph-Based Modular Frameworks for Audiovisual Applications[C]. 2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP). IEEE, 2018: 1-4.

[50] Zhao J, Eoff U, Xu G, et al. DevOps for Open Source Multimedia Frameworks[M]. Intelligent Computing. Springer, Cham, 2022: 244-254.

[51] Hajinazar N, Oliveira G F, Gregorio S, et al. SIMDRAM: a framework for bit-serial SIMD processing using DRAM[C]. Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021: 329-345.

[52] Chi C C, Alvarez-Mesa M, Bross B, et al. SIMD acceleration for HEVC decoding[J]. IEEE Transactions on circuits and systems for video technology, 2014, 25(5): 841-855.

[53] Amiri H, Shahbahrami A. SIMD programming using Intel vector extensions[J]. Journal of Parallel and Distributed Computing, 2020, 135: 83-100.

[54] Jang M, Kim K, Kim K. The performance analysis of ARM NEON technology for mobile platforms[C]. Proceedings of the 2011 ACM Symposium on Research in Applied Computation. 2011: 104-106.

[55] Le T M, Snelgrove W M, Panchanathan S. SIMD processor arrays for image and video processing: a review[J]. Multimedia Hardware Architectures 1998, 3311: 30-41.

[56] Kusswurm, D. Modern arm assembly language programming : covers Armv8-A 32-bit, 64-bit, and SIMD [M]. 1st ed, 2020.