多媒体会议系统音频多点处理器的软件设计

张海峰,白骋宇
(杭州电子科技大学 电子信息学院,浙江 杭州 310018)

摘要:针对多媒体会议系统的实际应用需求,归类分析了涉及到的音频信息处理策略,介绍了音频多点处理技术的原理及其对应的数据传输层和信号处理层策略,设计了一个能有效消除溢出,同时能最大可能的防止信号失真的混音算法,并最终实现了一个性能良好的音频多点处理器。研究结果表明,该处理器可以作为研究 H.323 多媒体会议系统的基础平台,也可作为商用多点电话会议系统的参考模型。

关键词:多媒体会议;音频;多点处理器;混音
中图分类号:TH7;TN915 **文献标识码**:A **文章编号**:1001-4551(2010)06-0104-04

Software design of audio multi-point processor of multimedia conference system

ZHANG Hai-feng, BAI Cheng-yu
(School of Electronics Information, Hangzhou Dianzi University, Hangzhou 310018, China)

Abstract: According to the practical need of multimedia conference system, audio processing strategies were categorized, audio multi-point processing principle and the corresponding strategies in data transport layer and signal processing layer were introduced. A practical audio mixing algorithm was designed, which can effectively eliminate overflow, at the same time it can be able to make the greatest possible to prevent signal distortion. Finally, a well-behaved audio multi-point processor was realized. The results show that the processor can be used in the research of the H.323 multimedia conference system and it can also be employed as a reference model of the commercial multi-point teleconference.

Key words: multimedia conference; audio; multi-point processor; mixer

0 引 言

目前成熟的商用多媒体会议系统基本上都采用了 H.323 标准[1],H.323 视频会议系统由多点控制单元(MCU)、终端(Terminal)、网关(Gateway)、网守(Gatekeeper)共 4 种组件构成。其中 MCU(Multimedia Control Unit)是核心组件,它由 MC(Multimedia Control)和 MP(Multimedia Process)两个逻辑部件构成,音/视频信息的处理都是在 MP 中完成的。对于会议来说,音频能力是最基本的,所以音频多点处理器是 H.323 会议系统的核心部件。

现有的音频多点处理器在多路混音信号的溢出处理上普遍实现效果不理想,本研究设计一个可以有效消除溢出同时又能最大程度防止信号失真的音频多点处理器。

1 数据传输层多点处理技术原理

在 H.323 系统中,音/视频信息的交互都是通过承载在 UDP 之上的 RTP[2](Real-time Transport Protocol,实时传输协议)来实现的。发送端将数据封装成 RTP 包发送到网络,接收端从网络接收到 RTP 包后,将数据从中提取出来。

收稿日期:2009-07-20
作者简介:张海峰(1961-),男,浙江杭州人,副教授,硕士生导师,主要从事电子信息系统集成方面的研究. E-mail: hfzhang0811@126.com

严格的讲,RTP 协议属于应用层而不是传输层,但是在实现方式上它跟一般的高层应用软件有所不同,而更类似于传输层协议,所以本研究引入了数据传输层的概念来描述应用软件中 RTP 协议负责的内容,同时使用信号处理层的概念来描述应用软件中音频信号处理部分的内容。

MCU 音频多点处理的数据传输层实现依靠了 RTP 协议的 SSRC 和 CSRC 两个字段,SSRC 指同步源标识符,即 RTP 包的来源,32 Bytes,在每次 RTP 会话建立时随机产生,同一个 RTP 会话中不能有两个相同的 SSRC 值。CSRC 指贡献源列表,可以有 0 ~ 15 共 16 项,每项 32 Bytes,用来标志对一个 RTP 混合器产生的新包有贡献的所有 RTP 包的源。经过 MCU 处理后的每路音频输出的 SSRC 由 MCU 重新生成,如图 1 所示。将数据最初来源者的 SSRC 置于 CSRC 列表中,是为了确认与会各方的身份。

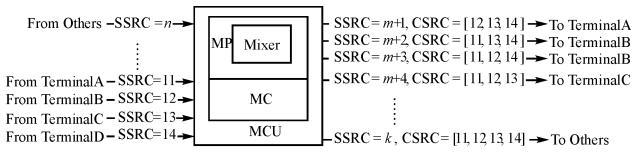

图 1　数据传输层多点处理技术原理

由图 1 也可以看出,多媒体会议系统音频多点处理器(Audio MCU)中的核心部件是 MP 中的混合器(Audio Mixer)。MC 主要负责会议建立与释放、网守注册与注销、呼叫连通与挂断等控制功能。对于 MC 的实现 H.323 标准有严格的规定,目前也有很多厂商(如 Radavision,Polycom 等),已经推出了功能完备的标准协议栈。在这种情况下,视讯产品开发的难点及创新点均在于 MP 的实现。

2　信号处理层多点处理技术原理

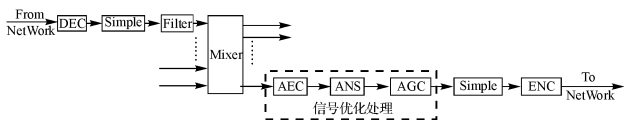

图 2　信号处理层多点处理技术原理

如图 2 所示,每一路从网络收到的 RTP 包在传输层被解析出有效载荷(Payload),还原成媒体流送到 MP,然后必须经过解码(DEC)、升采样(Simple)、滤波(Filter)3 个步骤,才能进入混合器(Mixer)。经过混合处理的信息要经过降采样、然后才能够编码(ENC)输出。在实际应用中,为了补偿前面各处理阶段可能引入的失真,信号编码之前还要经过回声抵消(AEC)、噪声抑制和语音增强(ANS)、自动增益控制(AGC)之类的优化处理[3]。

本研究采用升采样和降采样的原因是:参与混音的各路输入信号可能采用不同的编/解码格式和采样率,而混合器处理的各路信号必须具有相同的采样率,所以信号在进入混合器之前一般都要通过在低采样率信号样点间加入固定的采样点,然后采用对应的系数表进行平滑,来使各路输入信号具有相同的采样率。同理,混合输出后的信号需要送到不同的终端,而各个终端要求的编解码协议和采样率可能不同,所以需要通过降采样来满足各台终端的要求。

滤波是为了消除采样过程中可能引入的噪音。当然,有些采样器中本身已经包含了滤波器,这时就可以省掉外加的滤波了。

3　音频多点处理技术的核心—混音算法

3.1　混音算法的简单模型

混音算法的简单模型如下:

$$m(t) = \sum_{i=0}^{n-1} x_i(t), i = 0,1,\cdots,n-1 \qquad (1)$$

式中　$m(t)$—混音后的输出信号强度;$x_i(t)$—第 i 路的输入信号强度,n—音频信号输入源的数目。

上述简单模型在溢出处理上过于粗糙,容易引入噪音和爆破音。

目前的溢出处理算法大体上分为平均算法、对齐算法和箝位算法三类。平均算法就是将采样数据线性叠加后取平均值,实现简单但混音效果很不理想,而对齐算法实现复杂且有诸多缺点[4],所以本研究采用了一种改进的箝位算法。

3.2　一种改进的箝位算法

该算法的基本思想是通过引入一个可变的的衰减因子 f 来使溢出信号平滑过渡,从而消除爆破音,减小噪音。假设多点处理器为发往每台终端的音频信息建立一个大小为 BufSize 的源数据存贮区 SrcBuf[ ] 和一个混音结果存储区 DestBuf,混音过程如下:

```
void AudioMixer(SAMPLE DestBuf, SAMPLE SrcBuf[ ], int Count, int BufSize)
{
    int f = 1;              //平滑因子,初值取 1 表示无需平滑
    int i = 0,j = 0;        //循环控制参量
    memcpy(DestBuf, SrcBuf[i++], BufSize);
                            //第一路数据拷贝到混音结果存储区
    int temp = 0;           //临时变量
    for (i = 0; i < CNT; i)
    {
```

```
        int * pDest = DestBuf;
                    //定义一个指向混音结果存储区的指针
        int * pSrc  = SrcBuf [ i ];
                    //定义一个指向源数据存储区下一个将
                    //要处理单元的指针
        for( j = 0; j < BufSize; j++ )
        {
            temp = pDest[j] + pSrc[j];
                    //音频数据叠加
            if( temp > MAX ) f = MAX/ temp;
                    //求得满足 output[i] *f≤MAX 的最大f
            if( temp < MIN ) f = MIN/ temp;
                    //求得满足 output[i] *f≥MIN 的最小f
            if(f < 1)          f = f +STEP;
                    //如果f小于1,则取f = f + STEP
            pDest[j] = temp *f;
                    //数据平滑处理后写入混音结果存储区
        }
    }
}
```

结构 SIMPLE 用来保存每个音频采样点的有关参数,常量 MAX 和 MIN 分别代表信号强度的最大值和最小值,常量 CNT 代表参与混音的输入源的数目。CNT 的值要根据具体情况来确定,因为 H. 323 多点会议系统的与会者理论上可以通过级联增加到无穷多个,而任何混音算法都不可能实现无限多路输入的混音。一般的做法是,通过能量估计,选出信号最强的几路混音。所以当混合器支持的输入源数目大于与会者的数目 Count 时,CNT 取 Count 减 1,而当 Count 大于混合器支持的最大输入源数目时,发往没有参与混音的各台终端(没有讲话,或者讲话声音太小被忽略)的数据应该是全混音,CNT 取 Count。而发往参与混音的各台终端的数据则是全混音减去该终端自己的声音(每个发言者都不需要再收到自己发出的声音),CNT 取 Count 减 1。本研究限定最大与会者的数目即为混合器所能支持的最大输入源的数目,即 CNT 取 Count 减 1。

本算法相比原有算法[5]最大的创新点是通过引入了衰减因子 f 和其增量 STEP 有效的消除了溢出引起的爆破音,同时又最大程度的防止了信号的失真。其中引入 STEP 的目的是为了避免这种情况:偶尔一次脉冲冲击信号使 temp > MAX,之后信号已经恢复稳定,f 却仍然保持原来的值(小于 1)使信号轻微失真。而引入 STEP 后,f 时刻在动态调整,当信号恢复稳定时,f 也会逐渐恢复到 1,直至下次出现溢出。显然,STEP 越小,f 恢复越快。但是 STEP 过小,算法执行的开销会变大,在实际使用中可根据具体情况通过多次测试来合理选定 STEP 值。

4 多点处理器的软件实现

以开源的 H. 323 协议栈 OPENH323[6-7] 为基础,按照前面各节讲述的方法在 VC ++ 环境下编程实现了一个基于 Windows 的 H. 323 音频多点处理器(Audio Multi-point Processor,简称 AMP),如图 3 所示。用 AMP192. 168. 21. 110 呼叫 3 台模拟终端 192. 168. 21. 166、192. 168. 21. 175 和 192. 168. 21. 164,然后通过 Wireshark 抓包查看 AMP192. 168. 21. 110 收到来自 3 台终端的音频数据包,分别如图 4、图 5、图 6 所示。

图 3 音频多点处理器控制面板

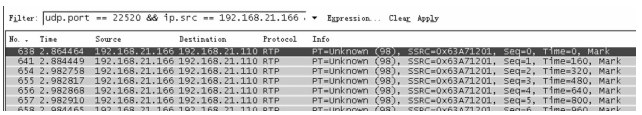

图 4 AMP110 收到的来自终端 192. 168. 21. 166 的音频包

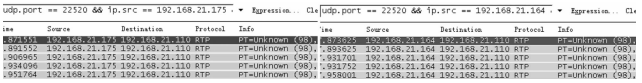

图 5 AMP110 收到的来自终端 192. 168. 21. 175 和终端 192. 168. 21. 164 的音频包

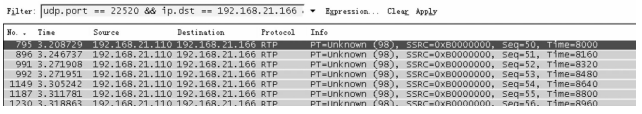

图 6 终端 192. 168. 21. 166 收到的来自 AMP110 的音频包

通过对比可以发现,AMP110 的所有音频数据都是通过同一个端口 22520 收到的,而来自每台台终端的音频数据具有不同的 SSRC。音频数据经过 AMP110 处理后,按照图 1 所描述的策略发往各台终端,其中发往 192. 168. 21. 166 的音频包是来自终端 192. 168. 21. 164 和终端 192. 168. 21. 175 的混音,而 SSRC 则由 AMP 重新生成,对比图 5 和图 7 可以发现,这些都与前面的分析完全一致。

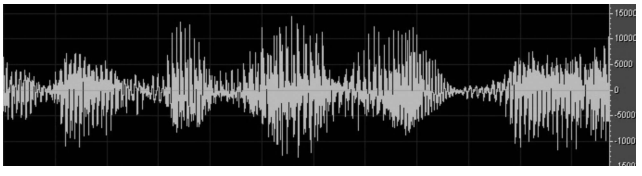

图 7 终端 192. 168. 21. 166 收到音频信号波形

将 Wireshark 抓到的音频数据流保存成 raw/au 格式,然后用 CoolEdit 查看波形,如图 7 所示,对于 3.2 节讲到的混音算法,此处 MAX 取 15 000,MIN 取 -15 000,STEP 取 5%。图 7 中的音频信号波形基本平滑,将音轨充分展开来看,没有任何溢出。将这段音频数据转换成 wav 格式在 RealPlayer 中播放,对比收到的声音和各台终端发出的声音,没有听到任何爆破音。可见,3.2 节设计的算法具有很好的混音效果。

图 7 得到的是 3 台终端会话时其中一台收到的音频数据波形图,当音频源增加到 5 路时,混音效果逐渐变差,这是由于本研究设计的 AMP 没有采取 AEC 和 ANS 等信号优化处理措施。当输入源数目增加时,各路音频源的背景噪音未能得到有效抑制,导致有用信号逐渐被叠加起来的背景音淹没。而 5 路输入已经达到了人耳能够同时辨别的有用声音的上限了,因此,本研究设计的多点处理器完全能满足实际需要。

5　结束语

本研究最大的亮点在于用最简单的方式和最低的成本设计了一个性能良好,功能完备的音频多点处理器,搭建了一个用于研究 H.323 多媒体会议系统的实验平台,这对于在实验室有限的条件下开展高复杂性,高难度的庞大系统的研究具有很大的借鉴意义。

参考文献(References):

[1] 杭州华三通信技术有限公司. H3C 视讯会议基础 V2.0 [EB/OL]. [2008 - 07 - 01]. http://forum.h3c.com/showtopic-41940.aspx.

[2] SCHULZRINNE H, CASNER S, FREDERICK R, et al. RTP: A Transport Protocol for Real-Time Applications [EB/OL]. [2003-07-01]. http://www.ietf.org/rfc/rfc3550.txt.

[3] 华为技术有限公司. 华为 HUAWEI ViewPoint 8000 视频会议产品介绍[EB/OL]. [2008 - 07 - 16]. http://bbs.vsharing.com/Article.aspx? aid=704944.

[4] 王文林,廖建新,朱晓民,等. 多媒体会议中新型快速实时混音算法[J]. 电子与信息学报,2007,29(3):690 - 695.

[5] 刘新华,李方敏,旷海兰,等. 基于数字语音教室的多路混音算法及应用[J]. 微计算机信息,2005,21(10):34 - 36.

[6] OpenH323 Study Group. openh323[CP/DK]. [2007 - 10 - 22]. http://sourceforge.net/search/? words=openh323&sort=score&sortdir=desc&offset=0&type_of_search=soft&pmode=0&form_cat=18.

[7] 卢　政. 如何通过 Openh323 开发自己的 H.323 协议栈[EB/OL]. [2002 - 12 - 18]. http://www.lsdn.org/82803/.

[编辑:张　翔]

(上接第 86 页)

5　结束语

本研究提出的这种基于 FPGA 的数字化光纤传输的方案,利用 FPGA 控制 ADC 和 DAC,并进行数据处理,系统简单稳定,误码率小,抗干扰能力强,实时性好,能很好地解决全数字控制的电力电子装置中电压电流转速温度等信号的数字化传输问题,实验结果验证了此方法原理和实践的可行性。

参考文献(References):

[1] 陈秀玲,周　欣,陈黎平. 基于 USB 接口的数据采集系统的设计与实现[J]. 自动化仪表,2004(9):19 - 22.

[2] 孟德刚,何国瑜. 基于 FPGA 的数据采集系统[J]. 电子测量技术,2004(5):74 - 75.

[3] SHI Meng-meng, JIANG Hai-he. Optical fiber applying to measuring and controlling systems [J]. **Measurement & Control Technology**,2007,26(9):4 - 6.

[4] 张小平,赵不贿. Altera 新型 FPGA 器件的配置方式[J]. 微处理机,2006(4):93 - 95.

[5] 单　成. 塑料光纤局域网设计[D]. 杭州:浙江大学信息学院,2001:15 - 18.

[6] 王廷尧. 光通信设备基础[M]. 天津:天津科技出版社,1992.

[7] TWYMAN H. Digital clock recovery[Online]. [2005 - 11 - 06]. http://www.twyman.org.uk/clock_recovery.

[8] 苏　红,张俊辉. 利用 FPGA 内部 DLL 实现数字时钟恢复[J]. 科学技术与工程,2007,18(7):4720 - 4722.

[编辑:李　辉]