基于FPGA的音频加解码处理的设计与实现

高德睿，周刘培，李泽宇，靳丛羽

(电子科技大学 四川成都 610000)(贡献度 1：1：1：1)

【摘要】针对音频信号的压缩和解压缩处理环节，设计并实现一种基于现场可编程门阵列（FPGA）的模块。首先软件部分，利用MATLAB将一段音频文件的数据进行加窗处理，后经过量化，mdct，imdct，霍夫曼编解码等步骤，实现音频文件的压缩与解压缩。利用Verilog硬件描述语言编写硬件代码，并最终利用vivado软件对其进行仿真。实验结果表明，音频文件经过压缩并解压缩后，音频文件数据有些许损耗，但仍可辨认出原音频曲调。

关 键 词 FPGA； 音频加解码； AC-3

Design and implementation of FPGA-based audio coding and decoding processing

GAO Derui, ZHOU Liupei，LI Zeyu，JIN Congyu

(University of Electronic Science and Technology of China，Chengdu 610000，China) (Contribution 1：1：1：1)

Abstract Aiming at the compression and decompression of audio signals, a module based on field programmable gate array (FPGA) and cordic algorithm is designed and implemented. First, the software part uses MATLAB to window the data of an audio file, and then through quantization, mdct, imdct, Hoffman encoding and decoding and other steps, the compression and decompression of the audio file is realized. Audition comparison can be made by importing pcm data into Audacity software. Write hardware code using Verilog hardware description language and eventually simulate it using vivado software. The experimental results show that after the audio file is compressed and decompressed, the audio file data is slightly lost, but the original audio tune can still be identified.

Key words FPGA；audio coding and decoding processing；AC-3

1 引言

音频信号处理是现代通信和娱乐领域中的重要技术，涉及到音频信号的采集、编辑、传输、压缩和解压等多个方面。随着数字信号处理技术的发展和成熟，音频处理逐渐成为了人们日常生活中不可或缺的一部分。

FPGA（Field-Programmable Gate Array）是一种可编程逻辑器件，它具有高度的灵活性和可重构性。与传统的固定功能芯片相比，FPGA可以根据需要重新配置其内部电路，从而实现各种不同的功能。在音频信号处理领域中，FPGA因其高速度、低延迟、低功耗和灵活性等优势而备受关注，并被广泛应用于音频设备和系统中。

本次实验旨在探究FPGA在音频信号处理中的应用，重点关注音频信号的压缩和编解码技术。压缩是将音频信号的数据量减少以达到节省存储空间和传输带宽的目的，而编解码则是将经过压缩的音频信号解码还原为原始的音频数据。

在音频信号的压缩和编解码中，MDCT（Modified Discrete Cosine Transform）和IMDCT（Inverse Modified Discrete Cosine Transform）是常用的变换方法。MDCT将时域上的音频信号转换为频域上的系数，而IMDCT则将频域上的系数恢复为时域上的音频信号。这种变换方法在音频编解码中具有重要的作用，能够有效地减少音频信号的冗余信息，提高信号的压缩比和音质。

作者简介：电子科技大学本科生，集成电路设计与集成系统专业

另外，量化和反量化是音频信号处理中常用的技术，用于将连续的音频信号离散化处理。量化过程将连续的音频信号值映射到一组有限的离散值上，而反量化则将离散的信号值重新映射到连续的音频信号空间上。通过合理选择量化步长和反量化算法，可以在保证一定的音频质量的前提下，降低数据的存储和传输成本。

此外，霍夫曼编码是一种常用的无损数据压缩算法，被广泛应用于音频信号的压缩。它通过统计信号中各个符号出现的概率，然后对各个符号进行编码，使得高频出现的符号用较短的编码表示，低频出现的符号用较长的编码表示。这样可以大大减少数据的存储和传输空间，并保持压缩后的音频信号与原始信号的一致性。

在本次实验中，我们将使用Vivado软件设计FPGA音频处理系统。具体步骤包括创建一个音频处理工程，添加FPGA芯片，导入音频信号，对信号进行预处理，实现MDCT和IMDCT变换、量化和反量化等功能模块，最后使用霍夫曼编码对音频信号进行压缩。通过Verilog HDL语言编写硬件描述，我们可以实现各个模块的功能，并通过模拟器和实际硬件平台验证设计的正确性和性能。

通过本次实验，我们将深入了解FPGA在音频信号处理中的原理和应用，为进一步优化音频处理系统性能提供参考。同时，这也为学习数字信号处理和音频编解码算法提供了宝贵的实践机会。

2 系统框架

2.1 系统设计思路

本项目的设计实现可大体分为两个部分，第一个为软件部分，第二个则为硬件部分。软件部分主要运用电脑的MATLAB软件及其他PCM音频播放软件来完成设计验证工作；而硬件部分则主要运用电脑的vivado仿真软件与FPGA来设计实现。之所以要将整体项目划分成软硬件两部分，是因为这样可以利用软件得到的结果来与硬件得出的结果做对比，以此判断实验效果。

根据上面的概念介绍，不难总结出总体的设计思路如下图1所示

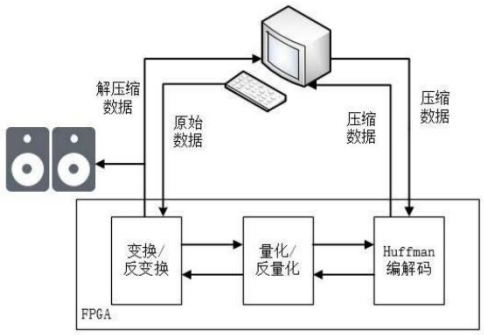

图2-1 系统设计思路

2.2 系统设计流程

原始数据从电脑端传输进入FPGA，在此进行MDCT变换，并经过量化与反量化，再通过接口传输回电脑端完成一次压缩过程；而要将数据进行解压缩，则可以将上述过程逆着实施一遍，即从电脑端传入FPGA，首先进行霍夫曼解码，结果通过索引值进行反量化，最后进行IMDCT变换，传输回电脑端，得到解压缩的数据。理想情况下得到的解压缩的数据将和原数据相同，但实际上根据上述原理得知，压缩解压缩过程中将存在数据的损耗，于是还原回的数据将会和原始数据有一定出入。下图2为基本设计流程，本次项目将在软件部分完整实现图中所展示的流程，硬件部分我们实现了MDCT、IMDCT和量化、反量化的过程。

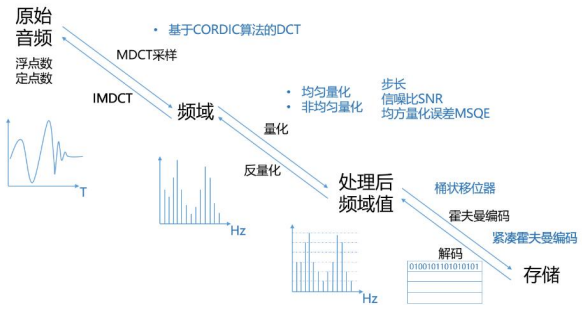

图2-2 系统设计流程

3 音频加解码的基础概念

3.1 MDCT与IMDCT

对于离散信号从时域到频域的变化，常用的处理方法有很多，其中DFT（离散傅立叶变换）用于将时域信号转换到频域上，通过将信号从时域采样变换到频域，而DCT（离散余弦变换/正交变换）则可以看成是一种简化的傅立叶变换，以一组不同频率和幅值的余弦函数和来近似信号，但是存在着一定的局限性，当fmax>S*F/2时会产生混叠现象。

MDCT（改进离散余弦变换）正是通过时域重叠来消除混叠这一现象。其使用的时域混叠抵消技术(TDAC)，包含50%的时域交叠窗，在不降低编码性能的情况下有效地克服加窗离散余弦变换(DCT)块处理运算中的边缘效应，从而有效地去除由边缘效应产生的周期化噪声。MDCT是一种非对称的变换形式，其反变换为IMDCT（改进型离散余弦逆变换），时域信号在经过MDCT编码以及IMDCT解码后，还原出的并不是原始信号。

其变换基本流程如下图3所示，输入的离散数字信号长度为4M，取相互交叠的2M区间长度的信号进行MDCT变换得到M长度的信号，再经过IMDCT变换，得到相互交叠的2M区间长度的信号，经过加窗、叠加后得到最终有效信号。

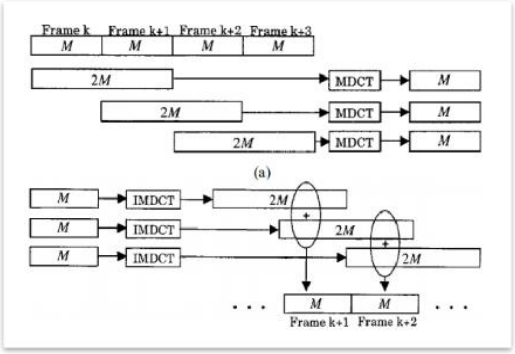

图2-3 MDCT及IMDCT变换流程

MDCT和IMDCT的计算公式如下，

$$X(k)=\sum_{n=0}^{N-1}x(n)\cos[(n+\frac{M+1}{2})(k+\frac{1}{2})\frac{\pi}{M}],k=0,1,...,M-1$$

$$\hat{x}(n)=\sum_{k=0}^{M-1}X(k)\cos[(n+\frac{M+1}{2})(k+\frac{1}{2})\frac{\pi}{M}],n=0,1,...,N-1$$

3.2 量化与反量化

现代化音频信号通常都以数字信号的形式进行储存，对于实际信号而言，数据精度通常较高，但对于人耳有限的分辨率而言，不需要精确的连续采样值，所以将处于不同区间上（即决策区间）的采样值拟合成一系列精度较低值，并将其以二进制序列（量化索引）进行编码，这个过程便称为量化。反量化则是通过索引得到其对应的区间量化值。

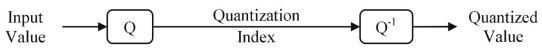

图4 量化-反量化过程图

均匀量化和非均匀量化是两种在信号处理中常用的量化方法，它们之间的区别主要在于量化间隔是否相同以及量化水平是否相等。

均匀量化（Uniform Quantization）：在均匀量化中，信号的幅度范围被均匀地划分成若干个间隔，每个间隔具有相同的宽度。因此，在均匀量化中，量化间隔是固定的，而且各个量化水平之间的距离也是相等的。均匀量化简单直观，易于实现，但对于信号的动态范围变化较大时会导致浪费比特数，影响量化效率。[1]

非均匀量化（Non-uniform Quantization）：在非均匀量化中，信号的幅度范围被划分成不同宽度的间隔，并且不同间隔内的量化水平可以不等距离地分布。这样做可以使得对信号的量化更加灵活，对于动态范围变化大的信号可以更好地适应信号的特性，提高了量化的效率和质量。

我们在该系统中采用了Lloyd-Max算法来实现对音频信号的非均匀量化。

Lloyd-Max算法通过迭代优化分段边界和量化水平，使得量化后的信号失真最小化。在每一次迭代中，算法根据当前的量化器状态计算分段边界和量化水平，并通过计算失真来更新它们。通过反复迭代，算法逐渐收敛到一个局部最优解。

4 音频编解码的设计

4.1 MDCT与IMDCT的软件实现

```
load music1
load Fs1

x0=music1;
N=512;
p=mod(length(x0),N/2);
x1=x0(1:length(x0)-p);
m=length(x1)/(N/2);
x=zeros(m-1,N);
for i=1:(length(x1)/(N/2)-1)
x(i,:)=x1((N/2*(i-1)+1):N/2*(i+1));
end
X=zeros(m-1,N/2);
X_temp=zeros(m-1,N/2);
for i=1:(m-1)
X_temp(i,:)=MDCT_new(x(i,:));
end
X_t2=fi(X_temp,1,16,-6);
```

```
X_t2_qt=mdct_qt(X_t2);

X=double(X_t2_qt);
x_IMDCT=zeros(m-1,N);

%imdct
for i=1:(m-1)
x_IMDCT(i,:)=IMDCT_new(X(i,:));
end
x_fi=zeros(length(x1),1);
x_fi(1:N/2)=x_IMDCT(1,1:N/2);
for i=2:(m-1)
x_fi(N/2*(i-1)+1:N/2*i)=x_IMDCT(i-1,N/2+1:
N)+x_IMDCT(i,1:N/2);
end
x_fi(N/2*(m-1)+1:N/2*(m))=x_IMDCT((m-1),N/
2+1:N);
```

其中MDCT和IMDCT所使用的函数由我们自己编写实现，具体函数代码见附录1。

4.2 量化过程的软件实现

在软件部分，我们采用了Lloyd-Max算法得到了音频信号非均匀量化的阈值，然后使用该阈值对进行了MDCT的音频信号进行4bit量化处理。Lloyd-Max算法的计算步骤如下：

初始化：首先，根据待量化的信号的范围确定初始的分段边界和量化水平。可以选择等间隔的初始分段边界和量化水平，或者通过其他方法进行初始化。

更新分段边界：根据当前的量化水平，计算每个分段的重心（centroid），即该分段内所有样本值的平均值。然后，将每个分段的边界设置为两个相邻重心的中点。

更新量化水平：根据新的分段边界，计算每个分段内所有样本值的平均失真（distortion）。失真是指原始信号与量化后信号之间的差异。根据平均失真，调整每个分段的量化水平，使得失真最小化。

重复迭代：重复执行步骤2和步骤3，直到分段边界和量化水平稳定不再变化或达到预设的迭代次数。

量化：根据最终确定的分段边界和量化水平，对输入信号进行量化。将输入信号映射到最接近的量化水平上，得到量化后的信号。

Lloyd-Max的实现算法代码详见附录2，下面是我们进行量化的代码：

```
function X_t3=mdct_qt(X_t2)

X_t2_d=double(X_t2)/64;
[w l]=size(X_t2_d);
y=X_t2_d(:)';%将MDCT输出的多维向量转换为一维
向量，方便后续量化的处理
m1=max(y);
y=y/m1;
bit=4;

……（中间省去了Lloyd-Max的具体计算过程）

%% 一维
for k=1:length(y)
if y_o(k)>boundaries(end)
y_qt(k)=boundaries(end)+(boundaries(end)-b
oundaries(end-1) )/2;
elseif y_o(k)<=boundaries(1)
y_qt(k)=boundaries(1)-(boundaries(1)-bound
aries(2) )/2;
else

for j=1:length(boundaries)-1
if y_o(k)>=boundaries(j) &&
y_o(k)<boundaries(j+1)
y_qt(k)=(boundaries(j)+boundaries(j+1) )/2
;
end
end
end
end

y_qt=y_qt';
X_t3=reshape(y_qt,[w,l]);

X_t3=m1*X_t3;
X_t3=fi(X_t3,1,16,-6);
```

4.3 MDCT与IMDCT的硬件实现

尽管采用动态加窗修正离散余弦变换(MDCT)实现了更高的频率分辨率，但由于直接计算MPEG编码中的MDCT和MPEG解码中的逆MDCT(IMDCT)是一项计算密集型的工作，因此，高效的

MDCT和IMDCT算法在音频编码和解码过程中至关重要。文章[2]提出了一般长度MDCT和IMDCT的回归结构，所提出的规则结构特别适合于并行VLSI的实现。

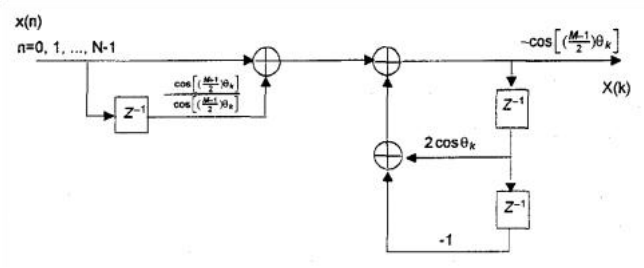

图4-1 MDCT回归结构

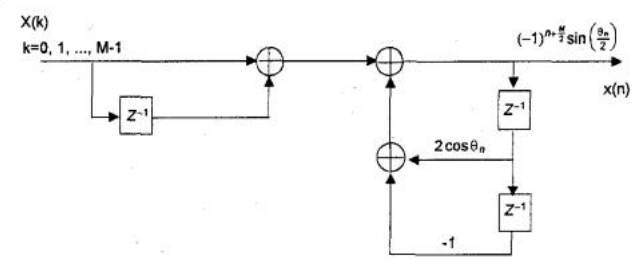

图4-2 IMDCT回归结构

数据	定点参数
2*cos(θ)	1,16,14
(-1)^(n+M/2)*sin(θ/2)	1,16,15
cos((M+1)*θ/2)/(COS((M-1)*θ/2))	1,16,6
2*cos(θ)	1,16,14
-cos(((M-1)/2)*θ)	1,16,15

表1 定点化数据(基于Matlab Simulink定点化分析)

4.4 量化与反量化的硬件实现

量化的本质是降低数据处理的精度从而提高能效，而本次待量化数据为数字信号，即就是降低数据的位数，我们采取的是16比特量化到4比特，量化策略是使用Lloyd-Max算法对已有的数据进行初步处理得出量化的边界，将输入数据分为16个区间，每个区间对应一个4比特数即为输出。

基于硬件资源成本和算法复杂度的角度考量，我们先将已有经验数据输入至matlab中通过软件实现的方式通过Lloyd-Max算法得出所需要的边界值，输入到量化硬件模块进行量化，因此硬件量化模块需要完成的任务就是将输入数据依次与边界进行比较，从而得出其所在区间，并输出4比特量化值。

具体硬件实现：用17个16比特寄存器组存下边界数据，通过16个比较器，比较出输入数据所在区间，并通过16个数据选择器将所需4比特数据输出。

反量化则是将4比特数还原到16比特数，我们采取的是用上边界的值来代替这一区间的所有值，同样，反量化硬件模块使用16个16比特的寄存器存储每个区间的上边界值，通过多路选择器输出16比特数。

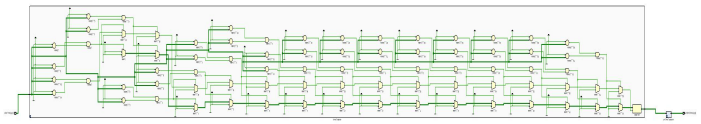

图4-3 量化硬件原理图

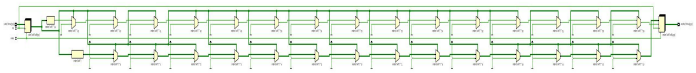

图4-4 反量化硬件原理图

5 系统仿真及硬件实现成果

5.1 软件仿真结果

在软件仿真中，原始音频信号波形图如图5-1所示，经过了MDCT-4bit量化-反量化-IMDCT的完整过程后，输出的结果波形图如图5-2所示。

由于量化的位数较低，因此可以看出经过压缩解压缩处理的音频信号有较为明显的底噪，不过将输出的信号播放出来，人耳并没有辨别出明显的噪声和失真。说明我们的软件系统设计较为成功。

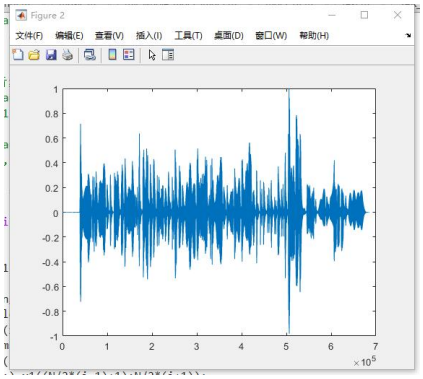

图5-1 原始音频信号波形图

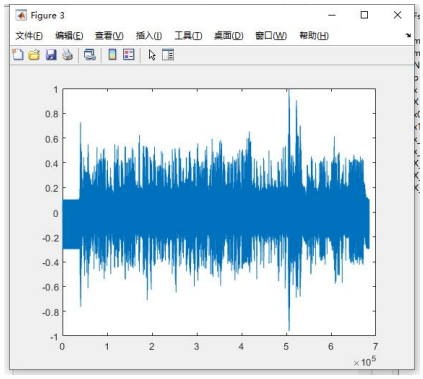

图5-2 软件仿真结果波形图

5.2 硬件实现结果

5.2.1 接口与存储

我们基于ZYNQ7020平台搭建了验证平台，在ROM存储CTIS C0244的一段音频，通过片上ARM CPU控制硬件DMA数据流的移动，方便实现MDCT的加窗，解压结束由硬件触发中断，当软件控制对所有数据进行压缩与解压缩完成后，统一对数据进行加窗恢复，之后将所有数据打印到串口PC上位机，由PC对输出数据进行分析。

表2 AXI从机寄存器与中断设计

BRAM_MDCT_IMDCT_S00_AXI_SLV		
REG0	[31:2]NC [1]INTR_CLR [0]START	
REG1	[31:19]NC[18:0]BASE_ADDR	
INTR	IRQ-F2P	

BRAM_BLOCK		
NAME	WIDTH*DEPTH	BRAM TILE
MUSIC_ROM	16*266240	122.5
GEN_T1	16*512	0.5
GEN_T2	16*512	0.5
GEN_T3	16*256	0.5
GEN_T4	16*256	0.5
GEN_T5	16*256	0.5
UNZIP_RAM	4*512	0.5
RAM_0	16*512	0.5
RAM_1	16*512	0.5
BRAM_0	32*1024	4

表3 存储资源占用

5.2.2 片上资源占用

Resource	Utilization	Available	Utilization %
LUT	6827	53200	12.83
LUTRAM	846	17400	4.86
FF	7363	106400	6.92
BRAM	131.50	140	93.93
DSP	7	220	3.18
MMCM	1	4	25.00

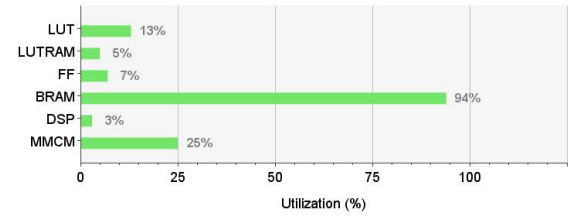

图5-3 系统工程占用资源

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	28.623 ns	Worst Hold Slack (WHS):	0.036 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	23009	Total Number of Endpoints:	22993

All user specified timing constraints are met.

Pulse Width

Worst Pulse Width Slack (WPWS):	7.000 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	9000

图5-4 系统时序摘要

5.2.3 整体性能

压缩率	0.125
计算模块时钟频率	10mhz
压缩时钟数(512data)	512*256(mdct)+256(quantify)
解压缩时钟数(512data)	256(inv-quantify)+512*256(imdct)
压缩速率	623781 bit/s

表4 系统整体性能

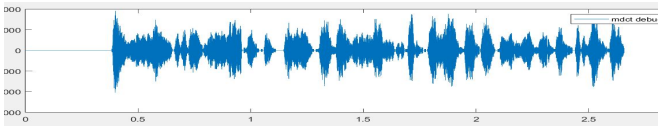

图5-5 硬件压缩软件解压缩数据

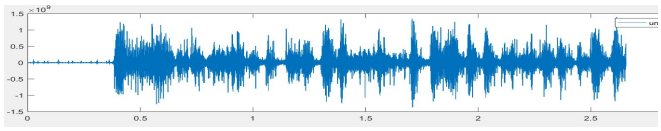

图5-6 硬件压缩与解压缩数据

6 总结与展望

我们在Matlab平台仿真模拟了MDCT/IMDCT以及Lloyd-Max算法，并基于ZYNQ完整开发与验证了回归MDCT-非均匀量化(压缩)&反量化-回归IMDCT音频(解压缩)的整个音频压缩与解压缩系统。我们所使用的MDCT/IMDCT采用回归迭代算法-大量节约组合逻辑资源，于此同时，量化/反量化采用Lloyd-Max算法-高压缩率。我们所搭建的系统，对于基于CTIS C0244数据，经过硬件的MDCT,量化,反量化下，软件的IMDCT解压后，达到了非常高的音频还原效果。

但我们的系统仍有很多可以优化改进的空间：
1.对于压缩速率计算潦草，之后我们可以通过调用内部计时寄存器获得精准时间。
2.由于ZYNQ输出通过串口打印速度较慢且需要手动处理分析，之后我们可以采用高速接口，写上位机，自动可视化工程效果。
3.我们将音频固化到了rom，这极大的拖慢了调试速度，之后我们可以尝试采用片外存储，比如SD卡等，由PS端软件更方便的控制音频数据的初始写入。

7 致谢

通过这次课程，学习并实践了有关音频加解码相关的知识。完成了软件和硬件部分的实验，锻炼了团队协作能力，提高了动手能力。

虽然小组成员已经抽出较多课余时间完成实验，但是仍有不少不足。例如：软件部分还可以进行相关优化，使得还原后的音频质量更高；硬件部分代码实现还存在问题，需要进一步的学习和试验等。

在实验过程中，也遇到了不少专业知识和软件使用上的问题，要感谢李辉老师的指点和小组成员的努力与坚持。这次课程不仅给每位同学提供一次实际项目锻炼，也有助于选择未来学习研究方向，以及思考和规划未来的职业发展。

参考文献

[1]Yuli You 2010. Audio Coding Theory and Applications. New York Dordrecht Heidelberg London: Science+Business Media, LLC. 349 pp.

[2]Hwang-Cheng Chiang and Jie-Cherng Liu, "Regressive implementations for the forward and inverse MDCT in MPEG audio coding," in IEEE Signal Processing Letters, vol. 3, no. 4, pp. 116-118, April 1996, doi: 10.1109/97.489065.

附录1

MDCT和IMDCT的实现算法：

```
function X_k=MDCT_new(X_i)
M = 256;
N = M*2;
t3=[];t4=[];t5=[];
X_in=double(X_i);
for k=0:M-1
%%k&n is reverted
theta_k_0=(k+0.5)*pi/M;
t3_reg=-cos((M+1)*theta_k_0/2)/(cos((M-1)*
theta_k_0/2));
t5_reg=-cos(((M-1)/2)*theta_k_0);
for n=0:N-1
j=n+1;
if n<1
P(j)=X_in(j);
elseif n<2
P(j)=X_in(j)+X_in(j-1)*(2*cos(theta_k_0)+t
3_reg);
else
P(j)=X_in(j)+2*cos(theta_k_0)*P(j-1)+t3_re
g*X_in(j-1)-P(j-2);
end

end
t3=[t3,-(cos((M+1)*theta_k_0/2))/(cos((M-1)
*theta_k_0/2))];
t4=[t4,2*cos(theta_k_0)];
t5=[t5,-cos(((M-1)/2)*theta_k_0)];
X_k(k+1)=P(N)*t5_reg;

end
end


function X_k_ideal=MDCT_ideal(x_n_init)
M=256;
N=2*M;
X_k_ideal=[];
for k=0:M-1
theta_k=(k+0.5)*pi/M;
sum=0;
for n=0:N-1
```

```
sum=sum+x_n_init(N-1-n+1)*cos((n-(M-1)/2)*
theta_k);
end
X_k_ideal(k+1)=-sum;

end

end

function X_t=IMDCT_new(X_f)
%X_f = sin((2*pi/M)*(0:M-1));
M = 256;
N = M*2;
t1=[];t2=[];
for n=0:N-1
for k=0:M-1
theta=(n+(M+1)/2)*pi/M;
j=k+1;
if k<1
Q(j)=X_f(j);
elseif k<2
Q(j)=X_f(j)+X_f(j-1)*(1+2*cos(theta));
else
Q(j)=X_f(j)+2*cos(theta)*Q(j-1)+X_f(j-1)-Q
(j-2);
end

end
t1=[t1,2*cos(theta)];
t2=[t2,(-1)^(n+M/2)*sin(theta/2)];
X_t(n+1)=Q(M)*(-1)^(n+M/2)*sin(theta/2);



end

end

function X_k=MDCT_new_fix(X_i)
M = 256;
N = M*2;
t3=[];t4=[];t5=[];
X_in=fi(X_i,1,64,32);
for k=0:M-1
%%k&n is reverted
theta_k_0=(k+0.5)*pi/M;
t3_reg=-cos((M+1)*theta_k_0/2)/(cos((M-1)*
theta_k_0/2));
t5_reg=-cos(((M-1)/2)*theta_k_0);
t3_reg_fix=fi(t3_reg,1,64,32);
t4_reg_fix=fi(2*cos(theta_k_0),1,64,32);
t5_reg_fix=fi(t5_reg,1,64,32);
for n=0:N-1
j=n+1;
if n<1
P(j)=X_in(j);
elseif n<2
P(j)=X_in(j)+X_in(j-1)*(t4_reg_fix+t3_reg_
fix);
else
P(j)=X_in(j)+t4_reg_fix*P(j-1)+t3_reg_fix*
X_in(j-1)-P(j-2);
end

end
t3=[t3,t3_reg_fix];
t4=[t4,t4_reg_fix];
t5=[t5,t5_reg_fix];
X_k(k+1)=P(N)*t5_reg_fix;

end
end

function X_t=IMDCT_new_fix(X_f_i)
%X_f = sin((2*pi/M)*(0:M-1));
M = 256;
N = M*2;
t1=[];t2=[];
X_f=fi(X_f_i,1,64,32);
for n=0:N-1
theta=(n+(M+1)/2)*pi/M;
t1_reg=2*cos(theta);
t2_reg=(-1)^(n+M/2)*sin(theta/2);
t1_reg_fix=fi(t1_reg,1,64,32);
t2_reg_fix=fi(t2_reg,1,64,32);
for k=0:M-1
j=k+1;
if k<1
Q(j)=X_f(j);
```

```
elseif k<2
Q(j)=X_f(j)+X_f(j-1)*(1+t1_reg_fix);
else
Q(j)=X_f(j)+t1_reg_fix*Q(j-1)+X_f(j-1)-Q(j
-2);
end

end
t1=[t1,t1_reg_fix];
t2=[t2,t2_reg_fix];
X_t(n+1)=Q(M)*t2_reg_fix;


end

end
```

附录2

Lloyd-Max的实现算法：

```
%% Lloyd-Max Algorithm
boundaries =
[linspace(-log(2.6),-0.35,0.25*2^bit),
linspace(-0.3,0.3,0.5*2^bit+1),
linspace(0.35,log(2.6),0.25*2^bit)];
reconstruction = zeros(1,2^bit);
mse_prev = 1;
mse_new = 1;

while (1)
[hist_values, edges] = histcounts(y,1000);
%define the reconstruction levels
for x = 1:2^bit
total_data_num = 0;
expectation_t = 0;
i = 1;
while (true)
temp_num = (edges(i)+edges(i+1))/2;
if ( temp_num >= boundaries(x) && temp_num <
boundaries(x+1) )
total_data_num = total_data_num +
hist_values(i);
expectation_t = expectation_t +
hist_values(i)*temp_num;
if i < 1000
i = i + 1;
else
break;
end
else
if total_data_num == 0 && i < 1000
i = i + 1;
continue;
else
break;
end
end
end
reconstruction(x) =
expectation_t/total_data_num;
end
%define new boundaries
for x = 1:2^bit-1
boundaries(x+1) = (1/2)*(reconstruction(x) +
reconstruction(x+1));
end
%check new error
y_nonuni_quantized = quantize_nonuniform(y,
boundaries, reconstruction,bit);
err_nonuni = y - y_nonuni_quantized;
mse_new = calculate_pow(err_nonuni);
%finishing condition (no more than %0.01
improvement on err)
if( abs((mse_new-mse_prev)/mse_new) <=
0.0001)
break;
end
mse_prev = mse_new;
end


y_o = X_t2_d(:)'/m1;
y_qt=zeros(1,length(y_o));
```