MPEG-1 Layer III Audio CODEC

Project Design

Brad Erwin
Mary Lou Kesse

March 25, 1999

1 Project Summary

For our senior project, we propose to develop an MPEG-1 Layer III (MP3) audio decoder with a Texas Instruments TMS320C6x digital signal processor (DSP). Achieving this goal requires a thorough investigation of the MPEG coding algorithm, the structure of MP3 data frames, and the general theories behind signal processing and audio compression. Because of the complexity of the project, there are several deliverables:

A) simulation of a simplified compression algorithm with MATLAB

B) development and simulation of C code to decode standard MPEG-1 Layer III data frames with the TMS320C6x DSP.

C) implementation of the decoding software on a TMS320C6x evaluation board.

2 Project Description

Before going into a detailed description of the different phases of the project, background information concerning the MPEG-1 Layer III algorithm and the project framework will be discussed. Afterwards, the project phases will be described and related to the major subsystems of the Layer III algorithm.

2.1 Background & Project Framework

To fully understand the framework of the project, it is necessary to be familiar with the fundamentals of the MPEG-1 Layer III algorithm. A simplified functional description of the algorithm is illustrated below as Figure 1.

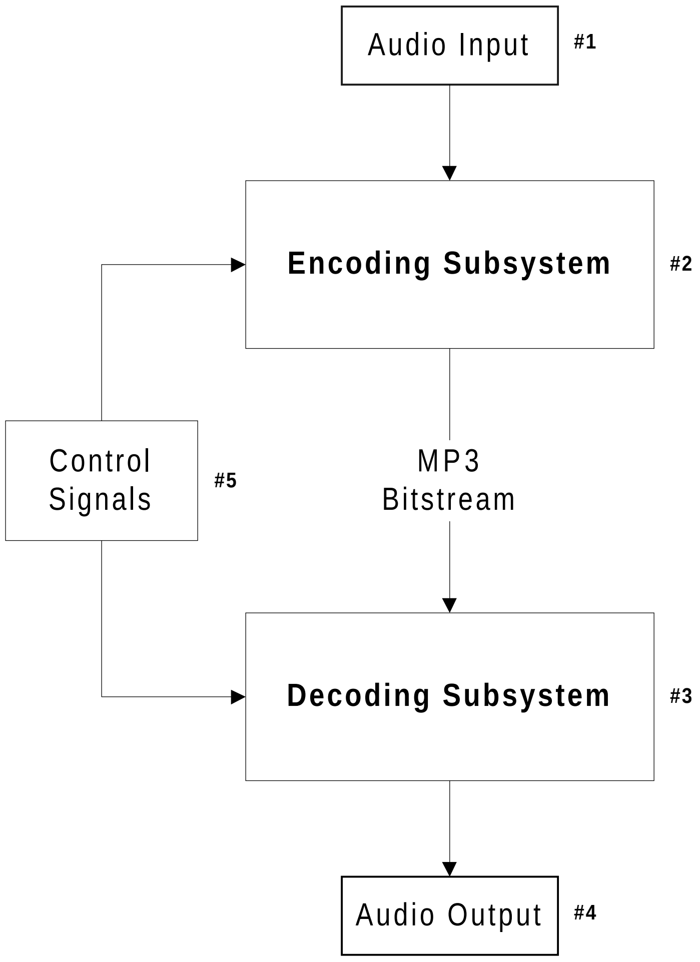


Figure 1 -- Functional Description of MP3 Algorithm

From the illustration, it is obvious that the MP3 algorithm has two major subsystems, the encoder and decoder. The encoding subsystem (Figure 1, Block #2) produces an MP3 bitstream by compressing the audio input (Figure 1, Block #1) according to the MPEG-1 Layer III specification. Under ideal circumstances, the compression algorithm is capable of compressing CD-quality stereo audio by a factor of 12:1 over PCM-coded digital audio with little loss in signal quality. The decoding subsystem (Figure 1, Block #3) interprets the MP3 bitstream and produces an audio signal (Figure 1, Block #4). Ideally, the signal produced by the decoding subsystem is perceptually identical to the original signal. The behavior of both subsystems is governed by a set of control signals (Figure 1, Block #5). These control signals provide information about the audio signal such as the sampling frequency, bitrate, and signal type (monophonic or stereophonic).

Due to time and resource constraints, the entire MPEG-1 Layer III algorithm will not be implemented in this project. Instead, a simplified encoding subsystem will be simulated with MATLAB to develop a familiarity with the theories behind signal processing and audio compression. Once the encoder simulation phase is complete, software will be developed for the TMS320C6x DSP to decode the MP3 bitstream and reconstruct the original signal. After debugging, this code will be uploaded to a TMS320C6x evaluation board for demonstration.

2.2 Phase I – Encoder Simulation

Though the entire MPEG-1 Layer III encoding subsystem will not be fully implemented in the project, a moderately sophisticated MATLAB simulation will be developed to demonstrate the processes used in a “real” encoder. The components of the standard MP3 encoding system are illustrated below in Figure 2; the simplified encoding simulation, Figure 3.

2.1.1 Standard MP3 Encoding Subsystem

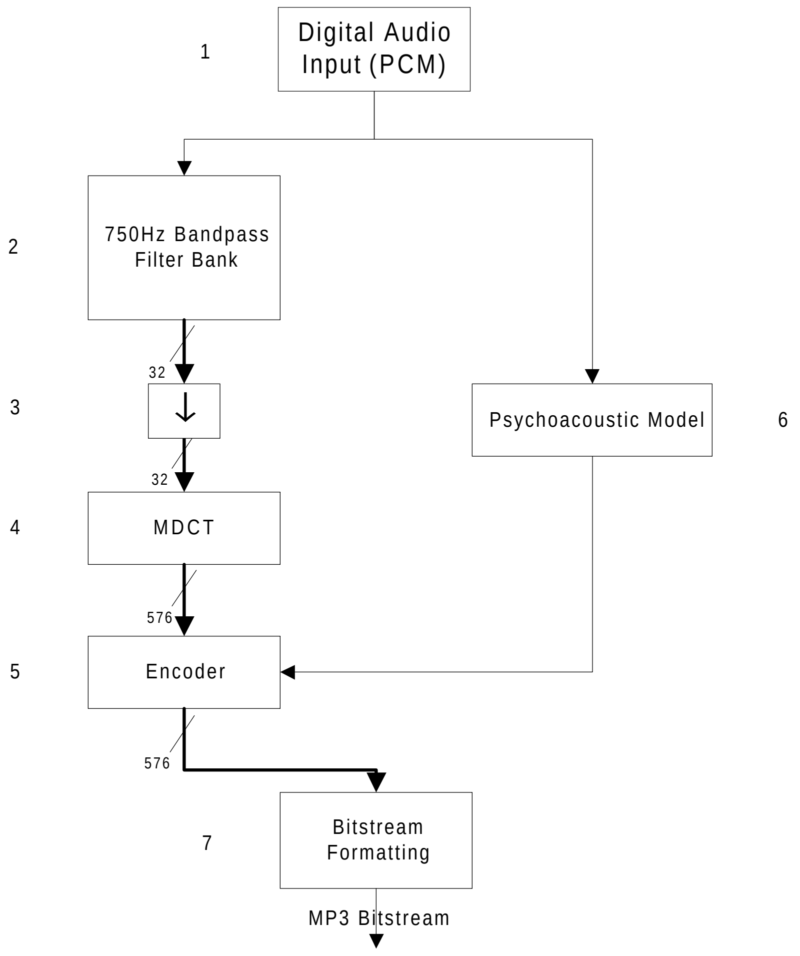


Figure 2 - Standard MP3 Encoding Subsystem

The heart of the Layer III algorithm is a bank of filters (Figure 2, Block #2) that break the input signal (Figure 2, Block #1) into thirty-two equally spaced frequency subbands depending upon the Nyquist frequency of the original signal. For instance, if the Nyquist frequency of the original signal is 24Khz, the filter bank would divide the signal into partitions approximately 750Hz wide. In this case, the output of the lowest magnitude subband would contain signal components from 0 – 750Hz; the next, 750Hz – 1500Hz, etc.

From the filterbank, a downsampler (Figure 2, Block #3) and an eighteen point modified discrete cosine transform (MDCT, Figure 2, Block #4) process the thirty-two subbands. The downsampler reduces the amount of data required for each subband by retaining only every thirty-second sample (i.e. a downsampling factor of thirty-two). However, the total amount of data required for the signal remains the same because there are thirty-two subbands in all. For example, if there are 100 original audio samples, the filterbank creates thirty-two subbands, each with 100 samples. This increases the number of samples to 3200 (100 * 32). After downsampling, the number of samples is reduced to the 100. The discrete cosine transform increases the accuracy of the filterbank by splitting each of the thirty-two subbands into eighteen more subbands for a total of 576 (32 * 18).

Simultaneously, a psychoacoustic model is applied to the input signal to determine which frequency bands should be retained. The model (Figure 2, Block #6) takes advantage of the masking properties of the human auditory system to identify unneeded components present in the original signal. (For a more detailed discussion of psychoacoustics, see Appendix A) The psychoacoustic model provides data about these spurious components to the encoder (Figure 5, Block #5).

The encoder uses the data provided by the model to determine how to encode the 576 subband signals. To achieve data compression, the encoder ignores subbands that are completely masked by other signal components. Additionally, subbands that are partially masked are encoded with less accuracy than the dominant subbands. These two methods allow the encoder to compress the data by a factor of 12:1 over PCM coded data with minimal loss of quality.

Finally, a bitstream formatting system (Figure 2, Block #7) assembles the output of the encoding subsystem into a stream of data according to specifications included in the MPEG-1 standard. Depending on the application, this data can be either stored in memory or transmitted to another user.

2.1.2 Simplified MP3 Encoding Subsystem

To save time, the standard MP3 encoding subsystem will be simplified for MATLAB simulation. A block diagram of this simplified system is shown below in Figure 3.

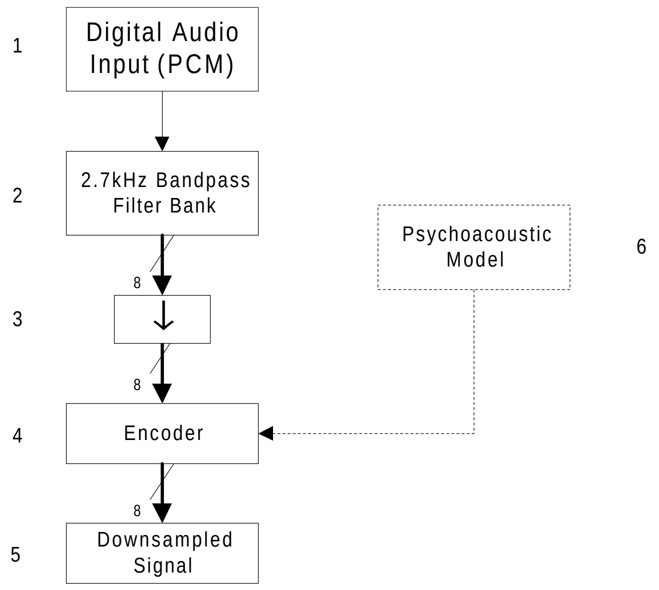


Figure 3 - Simplified MP3 Encoding Subsystem

From a quick comparison of Figures 2 and 3, it is obvious that the simplified subsystem is much less complicated than the standard encoding subsystem. The most significant changes include the elimination of the modified discrete cosine transform

(MDCT) and bitstream formatter. Both of these components were removed from the MATLAB simulation because of time constraints. Though each play an important part in the standard encoding process, they are not required to understand the fundamental theories of signal processing and data compression. Since the bitstream formatter is eliminated, the subsystem will produce a downsampled signal instead of a standard MP3 bitstream.

Less significant changes include a less complicated filterbank, downsampler, encoder, and psychoacoustic model (Figure 3, Blocks #2, 3, 4, and 6, respectively). Unlike the filterbank of the standard encoding subsystem, the simplified filterbank splits the original signal into eight subbands instead of thirty-two. Because the number of subbands and the downsampling factor should be equal, the downsampler is changed to retain every eighth data sample instead of every thirty-second. Eight subbands were chosen instead of the typical thirty-two to simplify the MATLAB code and decrease the computational complexity of the simulation.

Finally, both the encoder and psychoacoustic model lose their ability to function automatically. In the standard encoding subsystem, the psychoacoustic model identifies the dominant subband signals and instructs the encoder about how to prepare these signals for incorporation into the MP3 bitstream without input from the user. In the simulation, the user makes these decisions so their effect upon the compression algorithm can be explored.

2.3 Phase II – Decoder Software Development

After completing phase I, software will be developed to interpret MP3 data frames and reconstruct the original signal. Like the encoding subsystem, understanding this phase of the project requires some familiarity with the MPEG-1 Layer III algorithm.

2.3.1 Standard MP3 Decoding Subsystem

Unlike the encoding subsystem, the standard MPEG-1 Layer III decoding subsystem will be fully implemented in the project. Since there should be no significant differences between the standard decoder and the decoder implemented in the project, a detailed comparison of the two is not necessary. The system block diagram of the decoder is illustrated below as Figure 4.

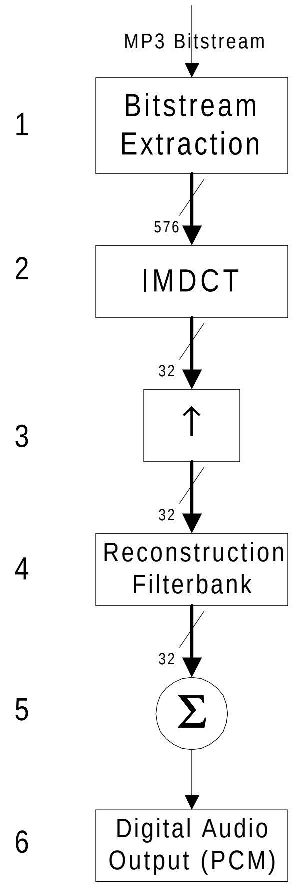


Figure 4 - Decoder Block Diagram

The decoding subsystem works by interpreting the MP3 bitstream produced by the encoding subsystem (Figure 2) and reconstructing the original signal. Since the subsystem is only concerned with the proper interpretation of the compressed data, it is significantly less complicated than the encoder.

The first step in recovering the original signal is bitstream extraction (Figure 4, Block #1). The extractor recovers the subband signals chosen by the encoding subsystem and provides this information to an inverse discrete modified cosine transform (IMDCT, Figure 4, Block #2). This transform is essentially the opposite of the MDCT -- it consolidates the 576 frequency bands back into the thirty-two bands generated by the encoding filterbank.

From the IMDCT, the subband signals are processed by an upsampler and reconstruction filterbank. The upsampler (Figure 4, Block #3) increases the number of samples in the signal by adding thirty-one zero-magnitude samples for every sample generated by the IMDCT. Upsampling is necessary to generate an output signal with the same signal bandwidth as the input signal. Next, the subbands are processed by the reconstruction filterbank (Figure 4, Block #4), which eliminates distortion caused by the upsampling process. Finally, the thirty-two subbands are added together (Figure 4, Block #5) to generate the single PCM coded signal (Figure 4, Block #6). Ideally, this signal should be perceptively identical to the original audio signal.

2.3.2 Software Development

The MP3 decoding algorithm detailed above will be implemented with a C program developed for the Texas Instruments TMS320C6x DSP evaluation module (EVM). Pre-compressed MP3 data frames will be uploaded from a host PC and stored in the memory available on the EVM. The decoding software will process these frames and produce a signal on an audio output port of the EVM. Since the decoding process is already broken into specific portions, the software development process will also occur in several well-defined steps: bitstream extractor, IMDCT, and reconstruction filterbank development. These portions are relatively self-contained, so developing them will be split between the project team members (see section 3 for a breakdown of tasks).

The most important step is developing an efficient bitstream extraction function because none of the other decoding subsystems will work correctly if they are not provided with accurate data. Though designing this algorithm may seem straightforward, it is complicated by the unusual nature of the MP3 data frames as illustrated below by Figure 5.

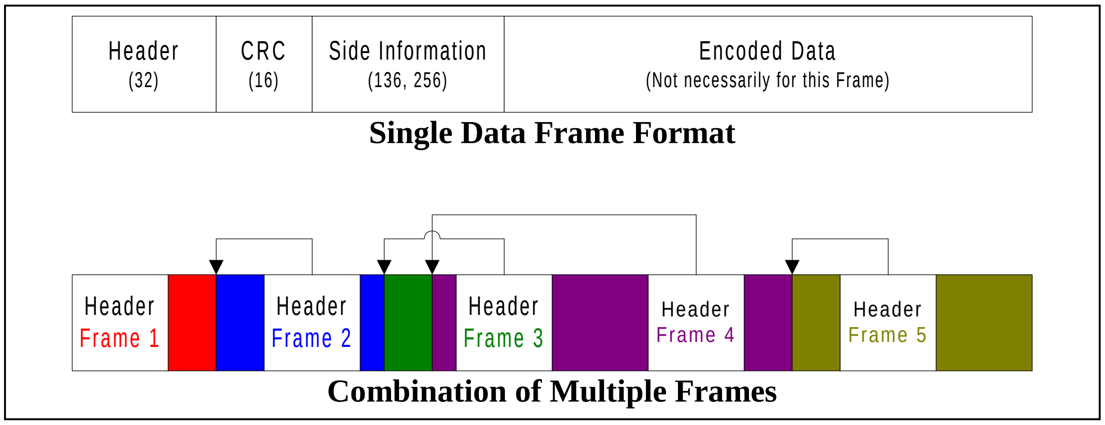


Figure 5 - MP3 Data Frame Format

As shown by the figure, each MP3 data frame has the same basic fields: a 32 bit header, an optional 16-bit cyclic redundancy check (CRC) value, some amount of miscellaneous (or "side") information, and encoded data. The header indicates what kind of audio data is being processed (stereo or mono), the bitrate, and the starting location of the encoded data. The CRC field is often included so the decoder can determine if the data frame has been corrupted. The decoding software will compute its own CRC from the data it receives and compare it with the CRC value in the data frame. If the numbers do not match, the data frame has been corrupted and is ignored.

Complications arise with the way that encoded data is handled in the data frames. Instead of allocating the same amount of data to every frame, the encoding algorithm can "borrow" data from adjacent frames if more accuracy is needed during the encoding process. This can occur when a frame containing few dominant components is adjacent to one containing many. In this case, the frame containing few dominant components requires less data to encode, so the encoder completes this frame's data field with data from the frame with more dominant components.

This process can become quite complicated, as illustrated in Figure 5. In the figure, Frame 2 is shown borrowing space from Frame 1, Frame 3 from Frame 2, Frame 4 from Frames 2 and 3, and Frame 5 from Frame 4. Though this process insures that all data frames are used efficiently, keeping track of the starting positions for multiple frames becomes quite frustrating.

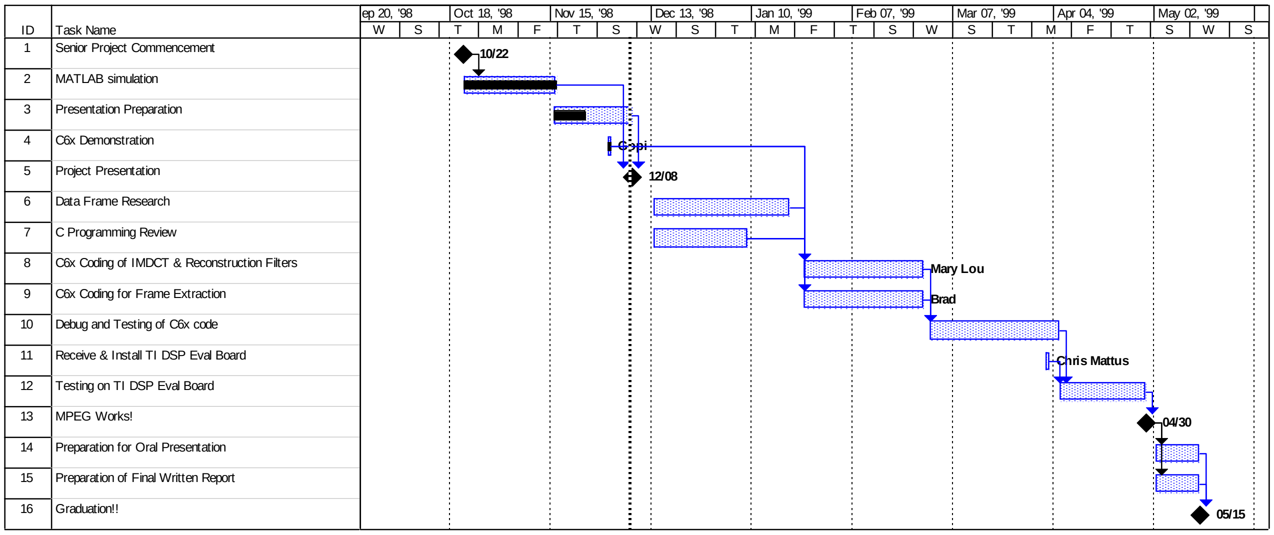
After the bitstream extractor, developing the IMDCT and reconstruction filterbank functions are the next programming challenge. From preliminary research into these components, both are implemented with some form of matrix arithmetic. Unfortunately, the details of these calculations continue to be a mystery – obviously, further research is required in these areas.

2.4 Phase III – Hardware Test

In the final phase of the project, the C program developed in Phase II will be uploaded to a TMS320C6x evaluation board for testing. Test MP3 data of a known signal will be loaded into the evaluation board’s RAM, decoded by the C program, and reproduced on the board’s audio output port. Comparing the signal produced with the original signal will test the decoder. Though they may be different due to slight inaccuracies in the decoding algorithm or limitations of the DSP, both should be perceptually identical to the listener.

3 Project Schedule

The tasks mentioned in section 2.3 are detailed in the chart below. The resources and partner assignments are noted next to each major task. Tasks with no partner assignments will be completed together.



ID	Task Name
1	Senior Project Commencement
2	MATLAB simulation
3	Presentation Preparation
4	C6x Demonstration
5	Project Presentation
6	Data Frame Research
7	C Programming Review
8	C6x Coding of IMDCT & Reconstruction Filters
9	C6x Coding for Frame Extraction
10	Debug and Testing of C6x code
11	Receive & Install TI DSP Eval Board
12	Testing on TI DSP Eval Board
13	MPEG Works!
14	Preparation for Oral Presentation
15	Preparation of Final Written Report
16	Graduation!!

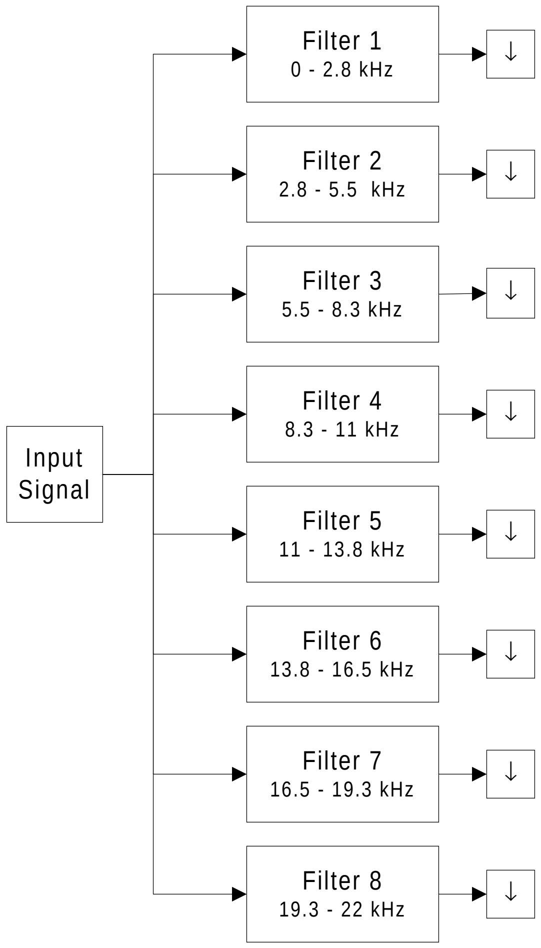
4 Current Results

As indicated by the project schedule, the MATLAB simulation phase of the project is complete. As promised, the MATLAB code is able to read PCM coded digital audio from a standard Windows .WAV file and process the signal into eight equally-spaced, downsampled subbands. By using coefficients supplied by the user, the program is able to simulate encoding the subbands with varying degrees of accuracy. To reconstruct the original signal, the downsampled subband signals are upsampled and processed by a simple reconstruction filterbank. While this approach is not nearly as effective as that employed in the standard encoding system, this simulation offers a good demonstration of the encoding process. The MATLAB code, in its entirety, is included as Appendix B.

4.1 Parallel vs. Cascade Filterbank Architecture

During the course of developing the MATLAB simulation, two different approaches to constructing the initial filterbank were explored: a “parallel” or “cascaded” implementation. The different architectures are illustrated below in Figures 6 and 7. The parallel architecture has the advantage of requiring fewer filters, but each of the filters require different coefficients. On the other hand, the parallel architecture requires only two distinct filters – each of which are applied several times throughout the filterbank.

Both filterbank architectures were tested with a “frequency ramp” signal (Figures 8 and 9) to determine how well they performed. In essence, the ramp signal simulates a frequency sweep from DC to 22 kHz. The signals resulting from the application of the different filterbanks are shown in Figures 10 through 13. In the FFT analyses (Figures 11 and 13) of the output signals, the cutoff frequencies of the eight bands are easily distinguishable. Since the cutoffs illustrated on the FFTs agree with the designed cutoff frequencies, the filterbanks are working correctly. However, by comparing the spectral views (Figures 10 and 12), it is obvious that the cascaded architecture results in less distortion of the original signal.



Parallel Filterbank Architecture
(8x downsamplers)

Figure 6 – Parallel Filterbank Architecture

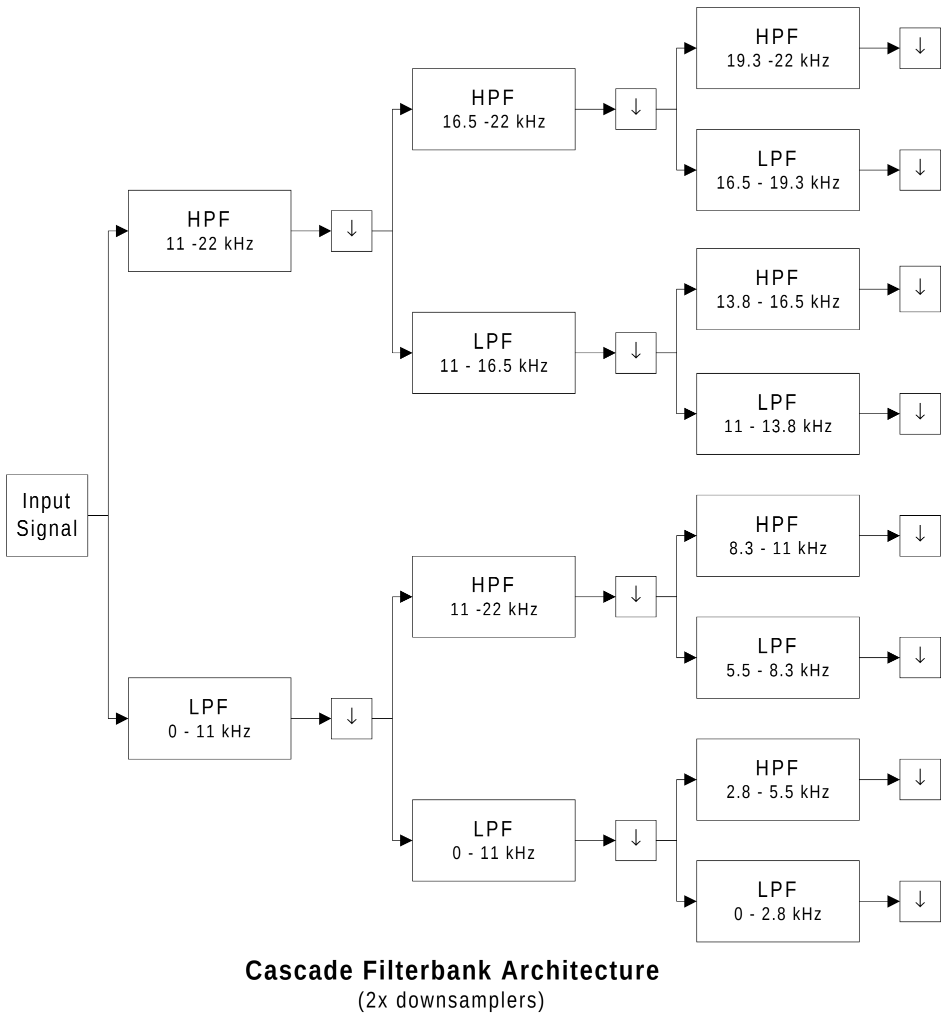
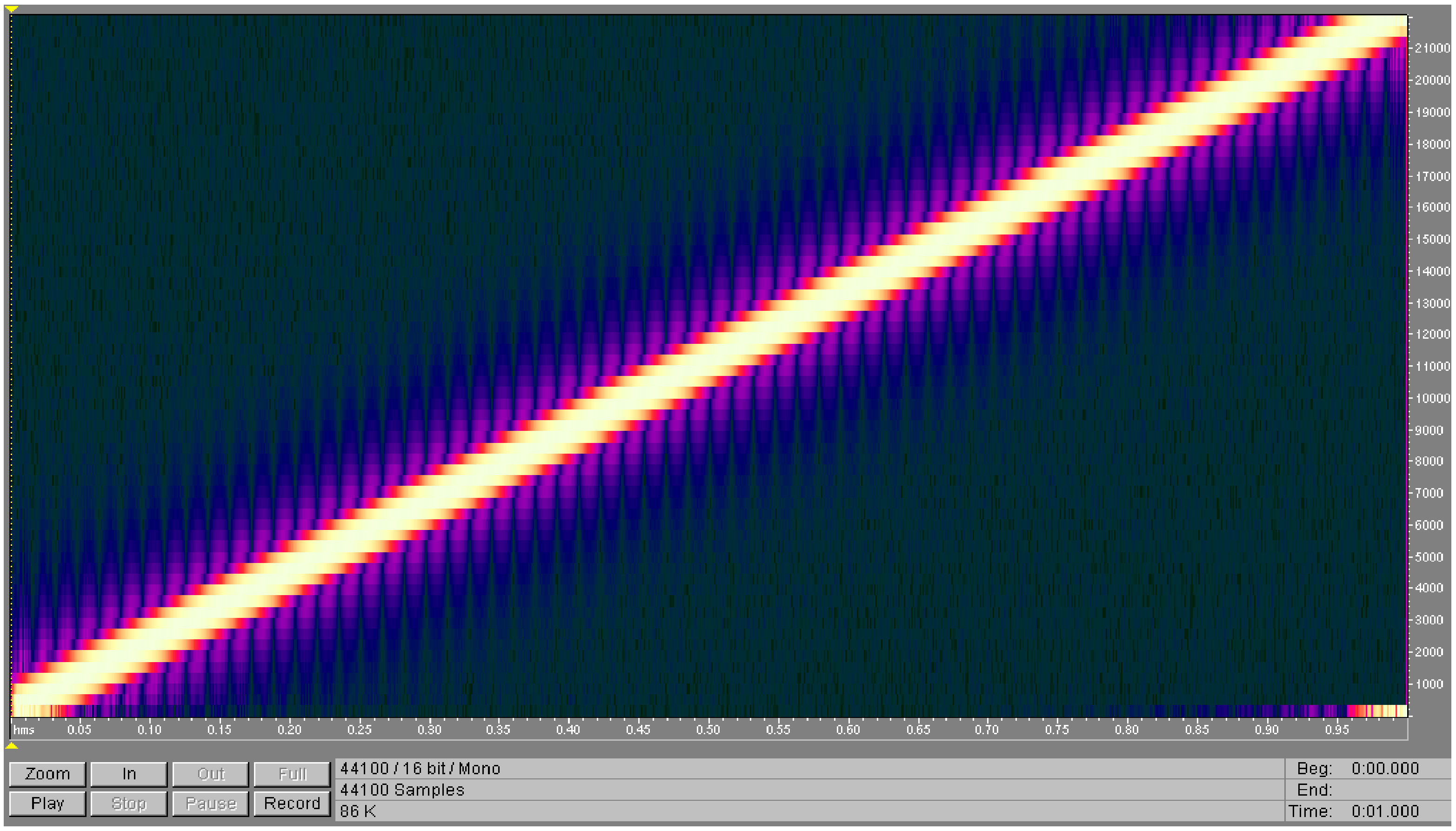


Figure 7 – Parallel Filterbank Architecture



44100 / 16 bit / Mono
44100 Samples
86 K
Beg: 0:00.000
End:
Time: 0:01.000

Figure 8 - Spectral View of Input Signal

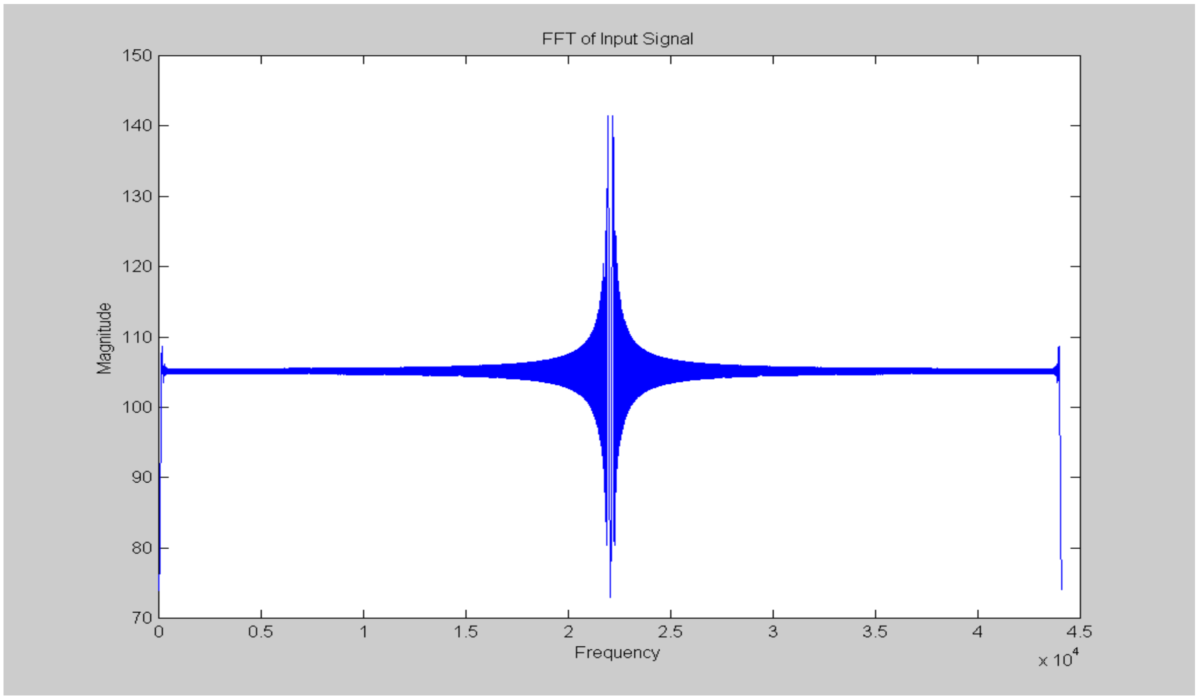
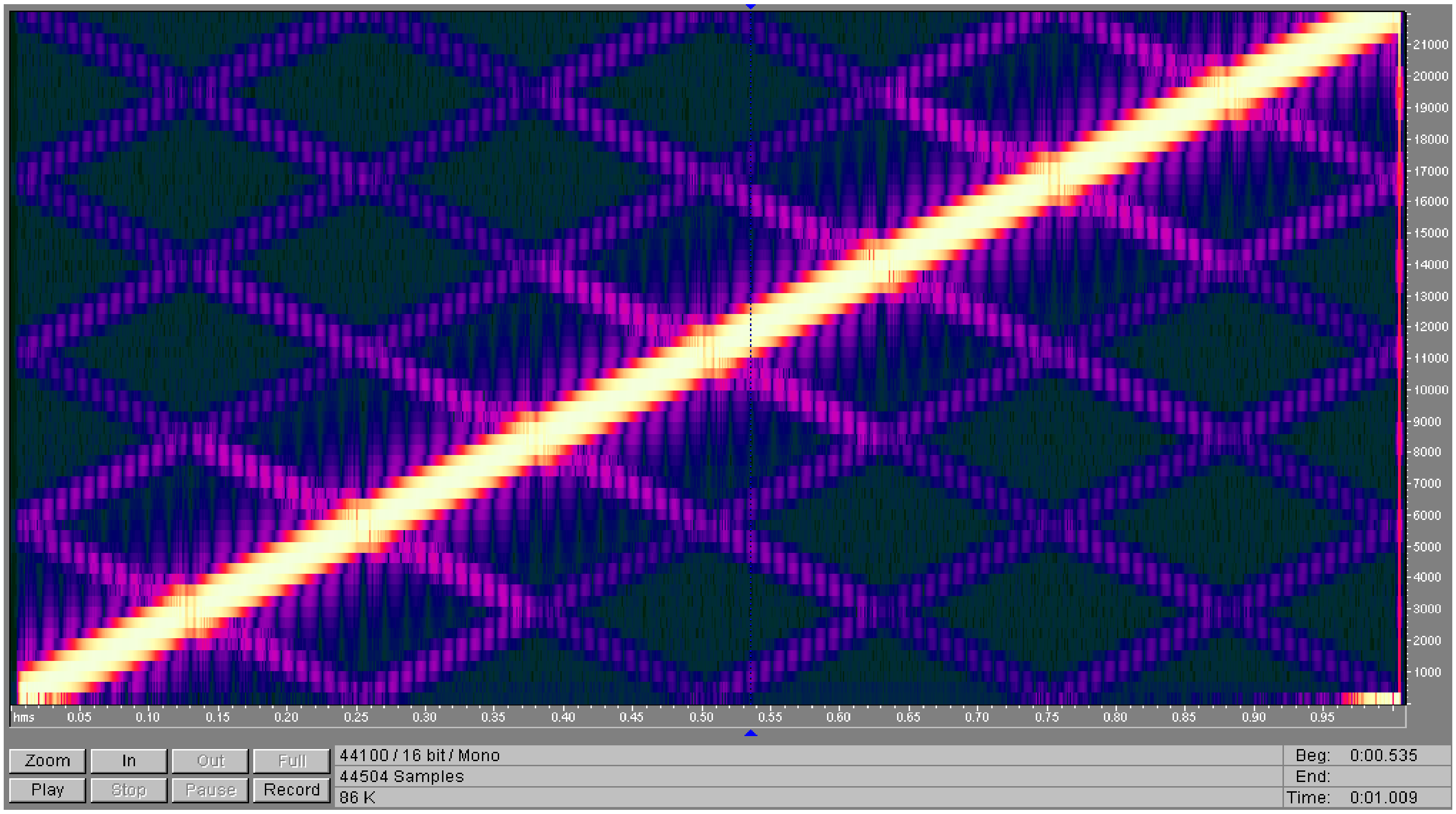


Figure 9 – FFT of Ramp Input Signal



44100 / 16 bit / Mono
44504 Samples
86 K
Beg: 0:00.535
End:
Time: 0:01.009

Figure 10 – Spectral View of Output Signal (Parallel Filterbank)

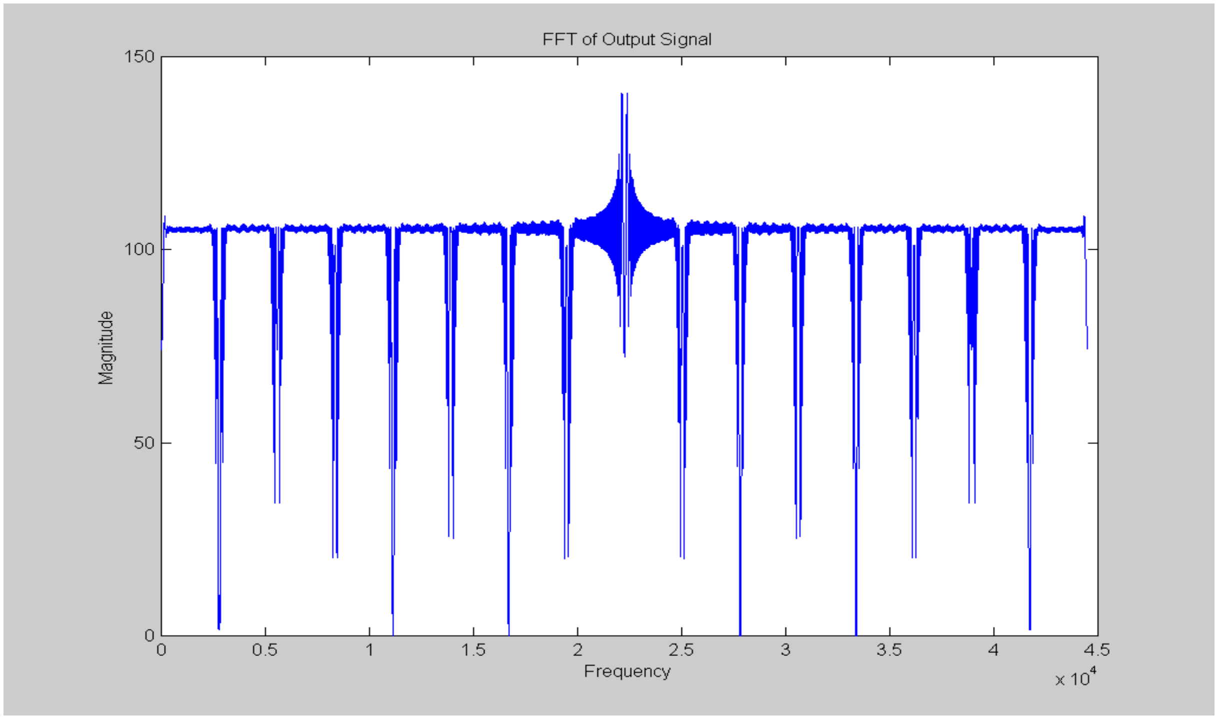
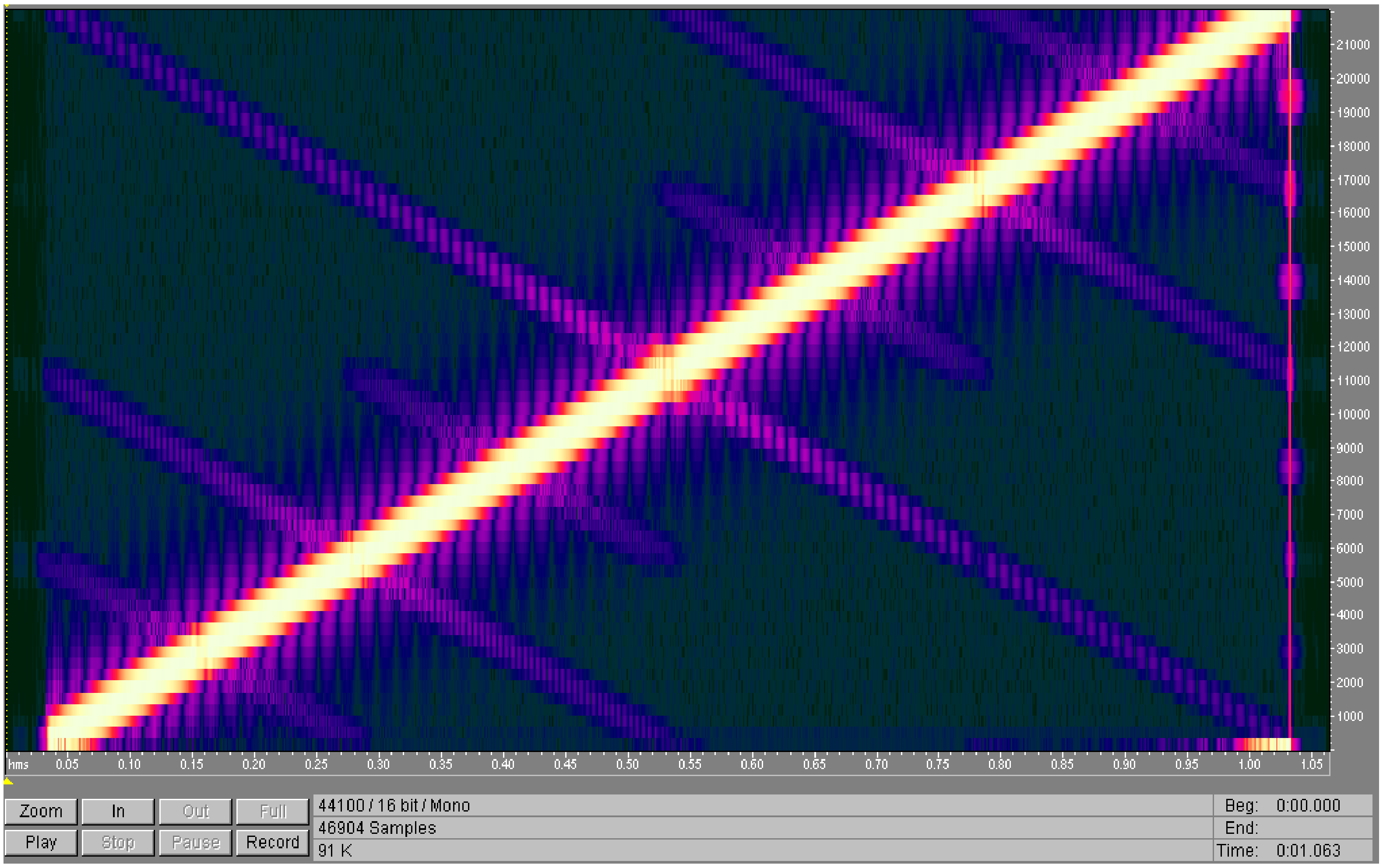


Figure 11 – FFT of Output Signal (Parallel Filterbank)



44100 / 16 bit / Mono
46904 Samples
91 K
Beg: 0:00.000
End:
Time: 0:01.063

Figure 12 – Spectral View of Output (Cascade Filterbank)

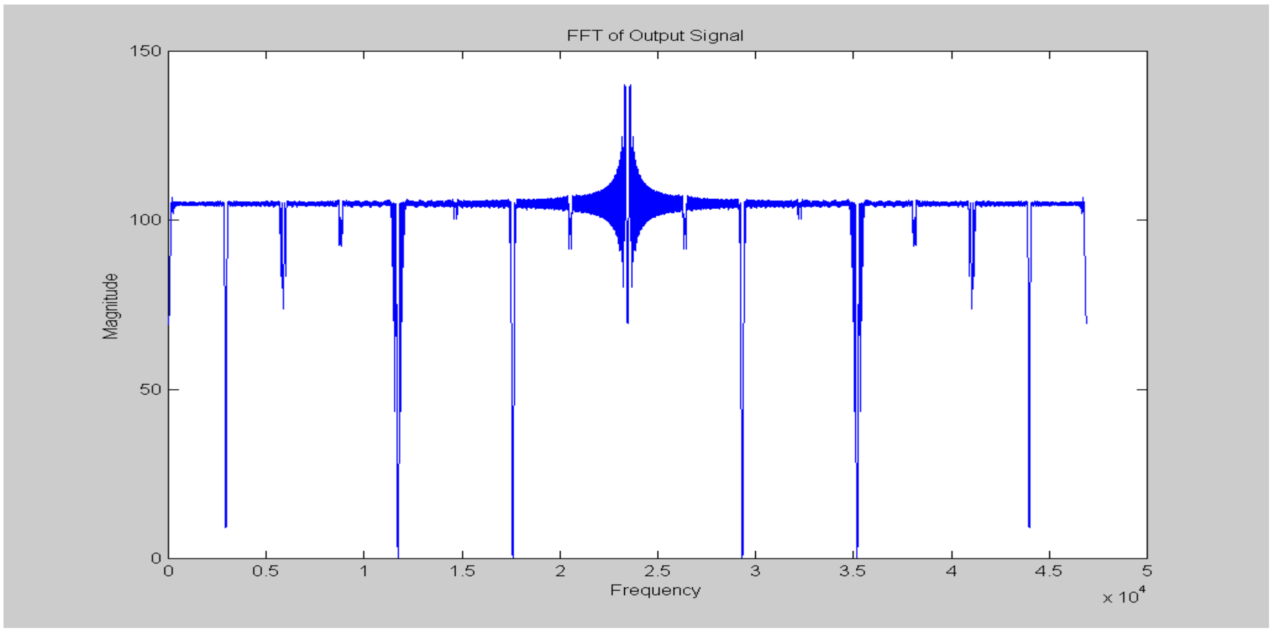
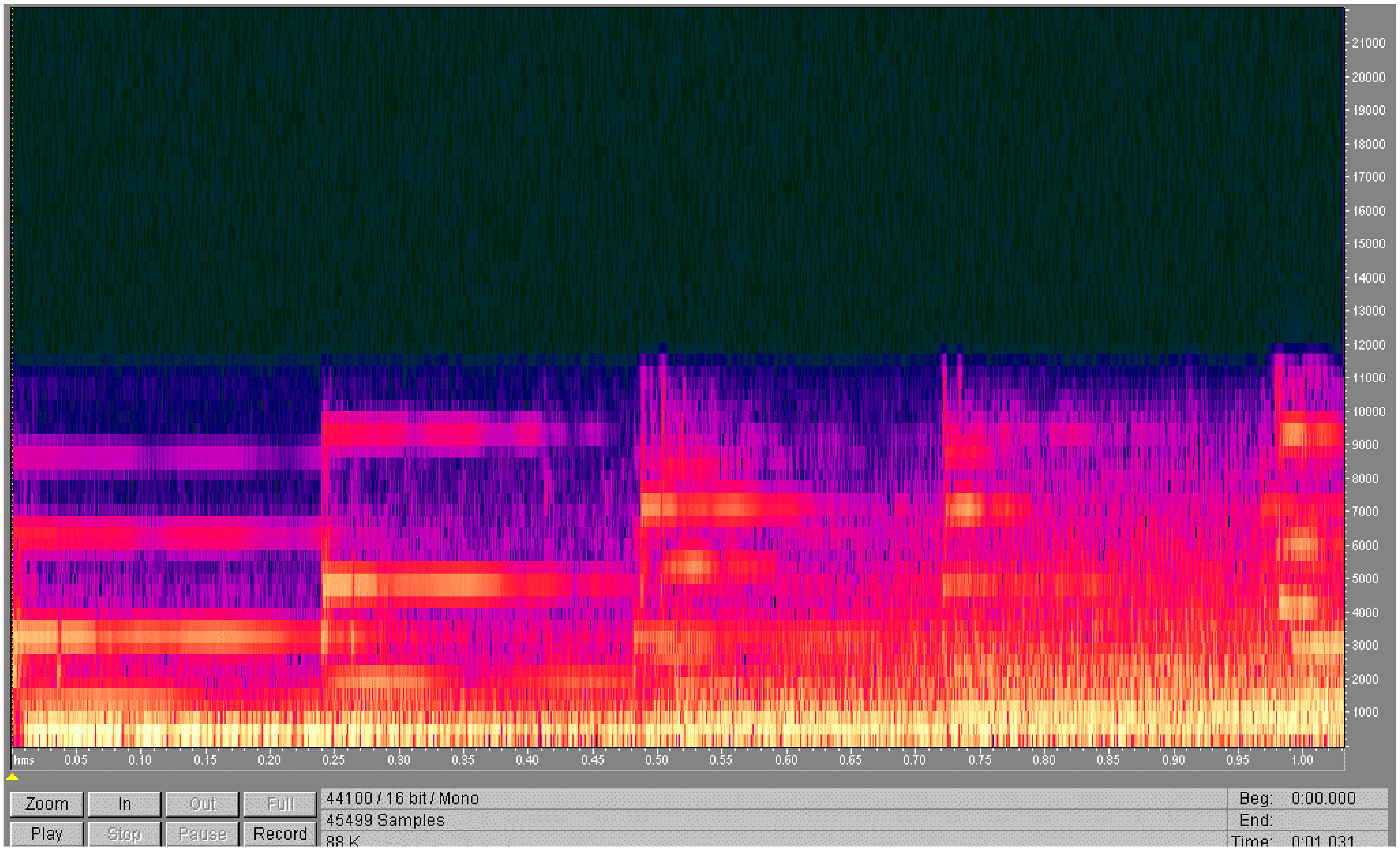


Figure 13 – FFT of Output (Cascade Filterbank)

4.2 Encoder Simulation

After verifying the functionality of the filterbanks, a different test signal was used to simulate the psychoacoustic model and encoding subsystems of the compression algorithm. The spectral and FFT plots of this signal appear below in Figures14 and 15, and those of the output signal generated by the simulation appear in Figures 16 and 17.

As mentioned previously, this simplified encoder uses coefficients supplied by the user to encode the subband signals with varying degrees of accuracy. Manipulating these coefficients simulates encoding the subbands with fewer bytes of data, resulting in compression. By comparing the spectral graphs of the two signals (Figures 14 and 16), it is obvious that the encoding system ignored a considerable amount of unneeded data that occurred at frequencies above 11 kHz. Additionally, the subbands from 2.8 kHz to 11kHz were “encoded” with less resolution than the subband from DC to 2.8 kHz. Observing the FFT plots (Figures 15 and 17) confirms that little signal power is concentrated in these areas, so they are good candidates for exclusion (ex, bands of 11 kHz and above) or less precise encoding (ex, bands from 2.8 kHz to 11 kHz). In this case, the first subband (DC to 2.8 kHz) was “encoded” at full 16-bit accuracy, the next two (2.8 kHz to 8.3 kHz) at 14-bit, the next (8.3 kHz to 11 kHz) at 13-bit, and the remaining subbands were ignored. The compression realized from this very simple demonstration approaches 2.25:1 – much better compression would be possible with a more sophisticated psychoacoustic model, better encoding techniques, and greater filterbank resolution.



44100 / 16 bit / Mono
45499 Samples
88 K
Beg: 0:00.000
End:
Time: 0:01.031

Figure 14 – Spectral View of Input Music Signal

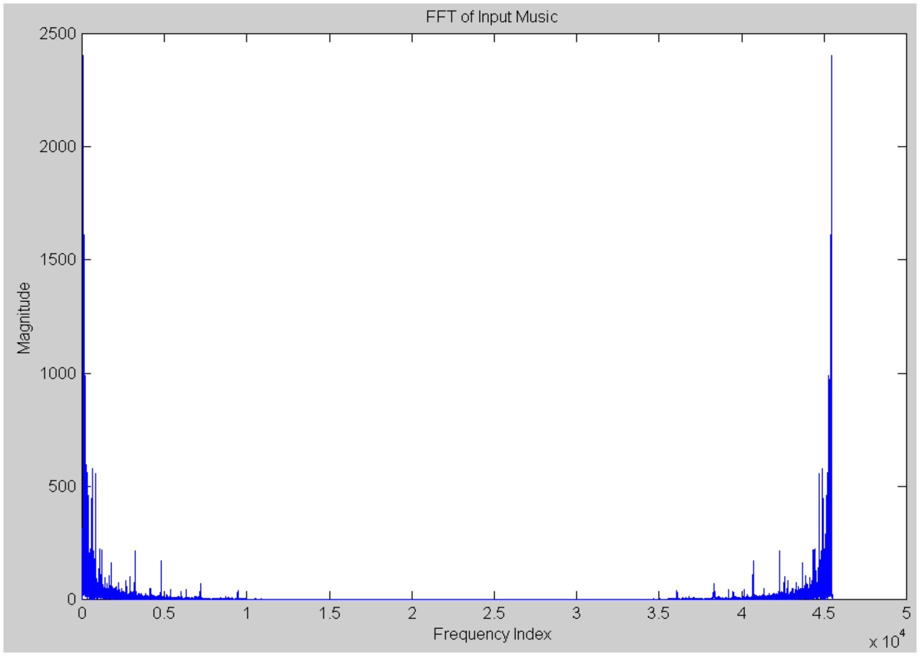
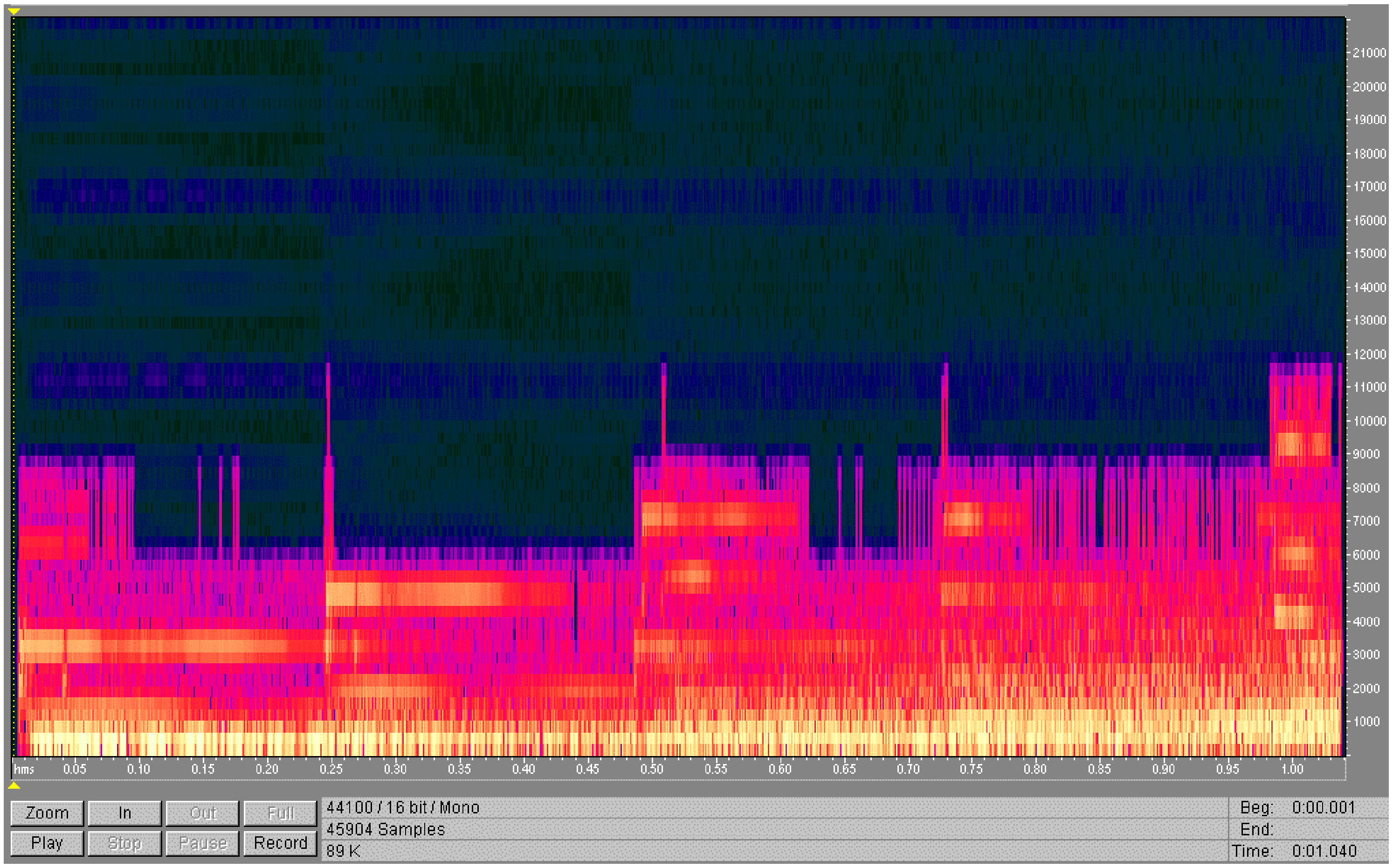


Figure 15 – FFT of Input Music Signal



44100 / 16 bit / Mono
45904 Samples
89 K
Beg: 0:00.001
End:
Time: 0:01.040

Figure 16 – Spectral View of Output Signal

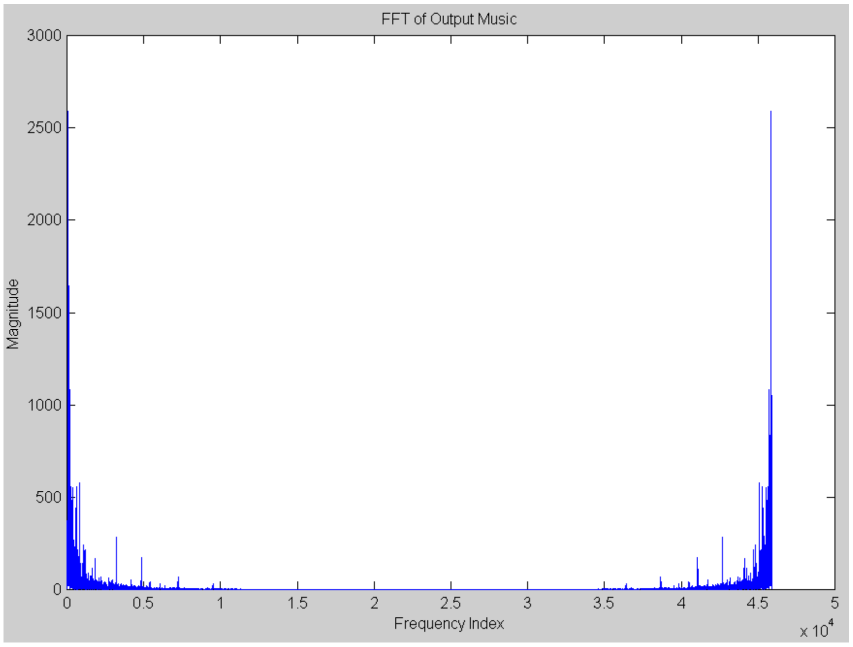


Figure 17 – FFT of Output Signal

5 Equipment Required

Part Number	Manufacturer	Description
TMDX326006201	Texas Instruments	TMS320C62x PCI evaluation board with support software and code generation tools
N/A	N/A	Pentium Pro or Pentium II class desktop PC with available PCI expansion slot

Note: According to the specifications for the TMS320C62x evaluation board (http://www.ti.com/sc/docs/dsps/tools/c6000/c62xevm/features.htm), there are sufficient memory and input/output subsystems on the evaluation board to implement the decoding algorithm without additional hardware.

6 Related Resources

6.1 Journals and Books

Brandenburg, K. "ISO-MPEG-1 Audio: A Generic Standard for Coding of High-Quality Digital Audio." *J. Audio Eng. Society*. **42**: #10. October 1994. p 780-792.

Konstantinicles, K. "Fast Subband Filtering in MPEG." *IEEE Signal Processing Letters*. **1**: #2. Febuary 1994. p. 26.

Liu, C. & Lee, W. "The Design of a Hybrid Filter Bank for the Psychoacoustic Model in ISO/MPEG Phases 1,2 Audio Encoder." *IEEE Transactions on Consumer Electronics.* **43**: #3. August 1997. p 586.

Mitchell, Joan L., Pennebaker, William B., Fogg, Chad E., and LeGall, Didier J., eds. *MPEG Video Compression Standard*. New York : Chapman & Hall, 1997.

Pan, Davis. "A Tutorial on MPEG/Audio Compression." *IEEE Multimedia Journal*. Summer 1995.

Pan, Davis Yen. "Digital Audio Compression." *Digital Technical Journal*. Vol. 5, No.2. 1993.

Princen, J.P. "Analysis/Synthesis Filter Bank Design Based on Time Domain Alias Cancellation." *IEEE Transactions on Acoustics, Speech, and Signal Processes.* **34**: #5. October 1986. p 1153.

Solari, Stephen J. *Digital Video and Audio Compression*. New York : McGraw-Hill, 1997.

Watkinson, John. *Compression in Video and Audio*. Oxford : Focal Press, 1995.

Zwicker, E. & Fastl, H. *Psychoacoustics.* Springer-Verlag. Berlin: 1990.

6.2 Standards

ISO/IEC 11172-1992 "Coding of Moving Pictures and Associated Audio for Digital Storage Media at rates up to 1.5M bits per second."

ISO/IEC 13818-2-1996 "Information Technology - Generic Coding of Moving Pictures and Associated Audio Information"

6.3 Websites

MPEG.ORG (http://www.mpeg.org) – A site offering numerous links to information concerning both MPEG video and audio encoding.

MP3.COM (http://www.mp3.com) – A site offering MP3 news and downloadable MP3 data.

Appendix A - Psychoacoustics

To understand how MPEG-1 and other audio compression algorithms work, one must be familiar with the properties of the human auditory system.

Generally, humans can hear sounds ranging from frequencies of 20Hz to 20,000Hz. However, due to complex interactions between sound signals and the auditory system, a strong signal component of a given frequency is able to override weaker signal components of different frequencies – an effect called spectral masking (see Figure 18).

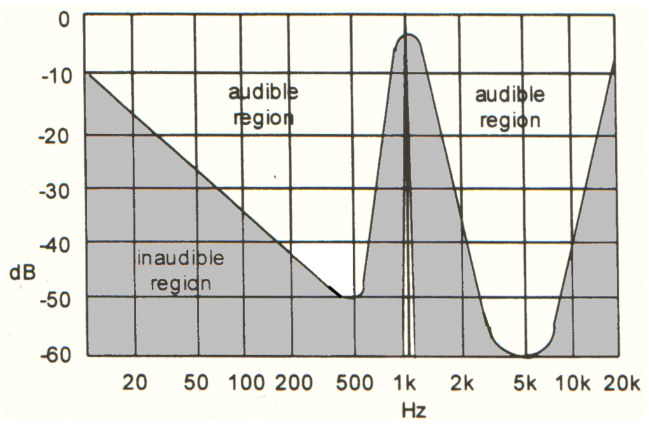


Figure 18 - Spectral Masking with a 1KHz Tone

In the example displayed in Figure 18, a strong signal component at 1KHz masks other components that fall within the shaded region. Though these masked components are present, the auditory system is able to distinguish only the dominant 1KHz component. In this case, the recessive components can be completely eliminated without any noticeable difference to the listener.

In addition to spectral masking, the human auditory system exhibits behavior described as temporal masking (see Figure 19).

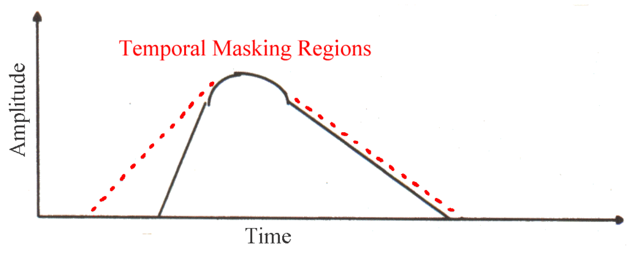


Figure 19 - Temporal Masking (regions indicated by dashed line)

As illustrated in Figure 19, temporal masking occurs during the time immediately before and after a sound is generated. The size of the masking regions is a function of the amplitude and frequency of the dominant component. In certain circumstances, a dominant component can mask a recessive component even if the recessive component starts before and ends after the dominant component. Similar to spectral masking, the recessive component can be discarded without any noticeable difference to the listener.

Significant compression is possible by identifying and ignoring recessive components of a given audio signal. The MPEG-1 algorithm achieves much of its compression by taking advantage of this principle.

Appendix B - MATLAB Code

Included below are the MATLAB functions used for Phase I of the project.

B.1 Parallel Filterbank Code

```
% Bandpass Filter Banks
% Demo of downsampling and upsampling

fs     = 44100; % sampling frequency
fnyq   = fs/2;  % Nyquist rate
order  = 200;   % FIR filter order
R_down = 8;     % downsampling ratio
R_up   = 8;     % upsampling ratio

[z]=wavread('sound.wav');

% Following 8 filters break the spectrum into
% 8 equal partitions with bandwidth of 2756.25 Hz

W0 = [0.0001/fnyq 2756.25/fnyq];
B0 = fir1(order, W0);

W1 = [2756.25/fnyq 5512.5/fnyq];
B1 = fir1(order, W1);

W2 = [5512.5/fnyq 8268.75/fnyq];
B2 = fir1(order, W2);

W3 = [8268.75/fnyq 11025/fnyq];
B3 = fir1(order, W3);

W4 = [11025/fnyq 13781.25/fnyq];
B4 = fir1(order, W4);

W5 = [13781.25/fnyq 16537.5/fnyq];
B5 = fir1(order, W5);

W6 = [16537.5/fnyq 19293.75/fnyq];
B6 = fir1(order, W6);

W7 = [19293.75/fnyq 22049/fnyq];
B7 = fir1(order, W7);

% Apply the filters to the input
```

```
y0 = conv(B0, z);
y1 = conv(B1, z);
y2 = conv(B2, z);
y3 = conv(B3, z);
y4 = conv(B4, z);
y5 = conv(B5, z);
y6 = conv(B6, z);
y7 = conv(B7, z);

% Add result together to compare to input
% (testing only)
% filter = y0+y1+y2+y3+y4+y5+y6+y7;

% Downsample the bands

d0 = downsample(y0, R_down);
d1 = downsample(y1, R_down);
d2 = downsample(y2, R_down);
d3 = downsample(y3, R_down);
d4 = downsample(y4, R_down);
d5 = downsample(y5, R_down);
d6 = downsample(y6, R_down);
d7 = downsample(y7, R_down);

% Add result together
% (testing only)
% down_sample = d0+d1+d2+d3+d4+d5+d6+d7;
% wavwrite(down_sample,fs/R_down,'dnsamp.wav');

% upsample the downsampled bands

u0 = upsample(d0, R_up);
u1 = upsample(d1, R_up);
u2 = upsample(d2, R_up);
u3 = upsample(d3, R_up);
u4 = upsample(d4, R_up);
u5 = upsample(d5, R_up);
u6 = upsample(d6, R_up);
u7 = upsample(d7, R_up);

% apply anti-aliasing filters

f0 = conv(B0, u0);
f1 = conv(B1, u1);
f2 = conv(B2, u2);
f3 = conv(B3, u3);
f4 = conv(B4, u4);
```

```
f5 = conv(B5, u5);
f6 = conv(B6, u6);
f7 = conv(B7, u7);

% add results together to obtain original signal

output= 8*(f0+f1+f2+f3+f4+f5+f6+f7);
wavwrite(output,fs,'output.wav');
```

B.2 Cascade Filterbank Code

```
% MPEG Bandpass Filter Banks
fs       = 44100;
fnyq     = fs/2;
order    = 200; %FIR filter order
LPF      = [0.0001 0.4999];
low_pass = fir1(order,LPF);
layers   = 3;
R_down   = 2;
R_up     = 2;

[z]=wavread('sound.wav');

WL = [0.0001 0.499999];
WH = [0.5 .999999];

BL = fir1(order, WL);  % Wn < fs/2
BH = fir1(order, WH);

a0 = conv(BH, z);
a1 = conv(BL, z);
a0 = downsample(a0,R_down);
a1 = downsample(a1,R_down);

b0 = conv(BH, a0);
b1 = conv(BL, a0);
b2 = conv(BH, a1);
b3 = conv(BL, a1);
b0 = downsample(b0,R_down);
b1 = downsample(b1,R_down);
b2 = downsample(b2,R_down);
b3 = downsample(b3,R_down);

c0 = conv(BH, b0);
c1 = conv(BL, b0);
c2 = conv(BH, b1);
c3 = conv(BL, b1);
```

```
c4 = conv(BH, b2);
c5 = conv(BL, b2);
c6 = conv(BH, b3);
c7 = conv(BL, b3);
c0 = downsample(c0,R_down);
c1 = downsample(c1,R_down);
c2 = downsample(c2,R_down);
c3 = downsample(c3,R_down);
c4 = downsample(c4,R_down);
c5 = downsample(c5,R_down);
c6 = downsample(c6,R_down);
c7 = downsample(c7,R_down);

% test purposes only..
% down_sample = c0+c1+c2+c3+c4+c5+c6+c7;
% wavwrite(down_sample,fs/(R_down^layers),'dnsamp.wav');

cu0 = upsample(c0, R_up);
cu1 = upsample(c1, R_up);
cu2 = upsample(c2, R_up);
cu3 = upsample(c3, R_up);
cu4 = upsample(c4, R_up);
cu5 = upsample(c5, R_up);
cu6 = upsample(c6, R_up);
cu7 = upsample(c7, R_up);
cu7 = conv(BL, cu7);
cu6 = conv(BH, cu6);
cu5 = conv(BL, cu5);
cu4 = conv(BH, cu4);
cu3 = conv(BL, cu3);
cu2 = conv(BH, cu2);
cu1 = conv(BL, cu1);
cu0 = conv(BH, cu0);

bu0 = cu0+cu1;
bu1 = cu2+cu3;
bu2 = cu4+cu5;
bu3 = cu6+cu7;

bu0 = upsample(bu0, R_up);
bu1 = upsample(bu1, R_up);
bu2 = upsample(bu2, R_up);
bu3 = upsample(bu3, R_up);
bu0 = conv(BH, bu0);
bu1 = conv(BL, bu1);
bu2 = conv(BH, bu2);
bu3 = conv(BL, bu3);
```

```
au0 = bu0+bu1;
au1 = bu2+bu3;

au0 = upsample(au0, R_up);
au1 = upsample(au1, R_up);
au0 = conv(BH, au0);
au1 = conv(BL, au1);

output=8*(au0+au1);
wavwrite(output,fs,'output.wav');
```

B.3 Compression Simulation Code

```
% Bandpass Filter Banks
% Demo of downsampling and upsampling

fs     = 44100; % sampling frequency
fnyq   = fs/2;  % Nyquist rate
order  = 200;   % FIR filter order
R_down = 8;     % downsampling ratio
R_up   = 8;     % upsampling ratio
divisor_0 = 1;  % divisors for each subband.
divisor_1 = 4;  % these are used to simulate lossy
compression.
divisor_2 = 4;
divisor_3 = 8;
divisor_4 = 256;
divisor_5 = 256;
divisor_6 = 256;
divisor_7 = 256;


[z]=wavread('sound2.wav');

% Following 8 filters break the spectrum into
% 8 equal partitions with bandwidth of 2756.25 Hz

W0 = [0.0001/fnyq 2756.25/fnyq];
B0 = fir1(order, W0);

W1 = [2756.25/fnyq 5512.5/fnyq];
B1 = fir1(order, W1);

W2 = [5512.5/fnyq 8268.75/fnyq];
B2 = fir1(order, W2);
```

```
W3 = [8268.75/fnyq 11025/fnyq];
B3 = fir1(order, W3);

W4 = [11025/fnyq 13781.25/fnyq];
B4 = fir1(order, W4);

W5 = [13781.25/fnyq 16537.5/fnyq];
B5 = fir1(order, W5);

W6 = [16537.5/fnyq 19293.75/fnyq];
B6 = fir1(order, W6);

W7 = [19293.75/fnyq 22049/fnyq];
B7 = fir1(order, W7);

% Apply the filters to the input

y0 = conv(B0, z);
y1 = conv(B1, z);
y2 = conv(B2, z);
y3 = conv(B3, z);
y4 = conv(B4, z);
y5 = conv(B5, z);
y6 = conv(B6, z);
y7 = conv(B7, z);

% Add result together to compare to input
% (testing only)
%filter0 = y0+y2+y4+y6;
%filter1 = y1+y3+y5+y7;

% Downsample the bands

y0 = y0 * 256;
y1 = y1 * 256;
y2 = y2 * 256;
y3 = y3 * 256;
y4 = y4 * 256;
y5 = y5 * 256;
y6 = y6 * 256;
y7 = y7 * 256;

d0 = round((downsample(y0, R_down))/divisor_0);
d1 = round((downsample(y1, R_down))/divisor_1);
d2 = round((downsample(y2, R_down))/divisor_2);
d3 = round((downsample(y3, R_down))/divisor_3);
d4 = round((downsample(y4, R_down))/divisor_4);
```

```
d5 = round((downsample(y5, R_down))/divisor_5);
d6 = round((downsample(y6, R_down))/divisor_6);
d7 = round((downsample(y7, R_down))/divisor_7);

% Add result together
% (testing only)
% down_sample = d0+d1+d2+d3+d4+d5+d6+d7;
% wavwrite(down_sample,fs/R_down,'dnsamp.wav');

% upsample the downsampled bands

d0 = d0 / 256;
d1 = d1 / 256;
d2 = d2 / 256;
d3 = d3 / 256;
d4 = d4 / 256;
d5 = d5 / 256;
d6 = d6 / 256;
d7 = d7 / 256;

u0 = upsample(d0, R_up);
u1 = upsample(d1, R_up);
u2 = upsample(d2, R_up);
u3 = upsample(d3, R_up);
u4 = upsample(d4, R_up);
u5 = upsample(d5, R_up);
u6 = upsample(d6, R_up);
u7 = upsample(d7, R_up);

% apply anti-aliasing filters

f0 = conv(B0, u0);
f1 = conv(B1, u1);
f2 = conv(B2, u2);
f3 = conv(B3, u3);
f4 = conv(B4, u4);
f5 = conv(B5, u5);
f6 = conv(B6, u6);
f7 = conv(B7, u7);

% add results together to obtain original signal

output=
8*(divisor_0*f0+divisor_1*f1+divisor_2*f2+divisor_3*f3+divi
sor_4*f4+divisor_5*f5+divisor_6*f6+divisor_7*f7);
wavwrite(output,fs,'output.wav');
```

B.4 Associated Functions

```
function [down_sample] = downsample(input, ratio)
i = 1;
element = 1;
n = length(input);
for i = 1:ratio:n,
   down_sample(element) = input(i);
   element = element +1;
end

function [up_sample] = upsample(input, ratio)
element = 1;
n = length(input);
up_sample = zeros(ratio*n,1);
for i = 1:1:n,
   up_sample(element) = input(i);
   element = element + ratio;
end
```