Rochester Institute of Technology

RIT Scholar Works

Theses

10-1-1998

Design and implementation of a DSP based MPEG-1 audio encoder

Eric Hoekstra

Follow this and additional works at: <https://scholarworks.rit.edu/theses>

Recommended Citation

Hoekstra, Eric, "Design and implementation of a DSP based MPEG-1 audio encoder" (1998). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

Design and Implementation of a DSP Based MPEG-1 Audio Encoder

by

Eric Hoekstra

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

Principle Advisor

Muhammad Shaaban, Assistant Professor

Committee Member

Roy S. Czernikowski, Professor and Department Head,

Committee Member

Soheil A. Dianat, Professor, Electrical Engineering Department

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York
October 1998

RELEASE PERMISSION FORM

Rochester Institute of Technology

Design and Implementation of a DSP Based MPEG-1 Audio Encoder

I, Eric Hoekstra, hereby grant permission to any individual or organization to reproduce this thesis in whole or in part for non-commercial and non-profit purposes only.

Eric J. Hoekstra

Oct 16, 1998

Date

ABSTRACT

The speed of current PCs enables them to decode and play an MPEG bitstream in real-time. The encoding process, however, cannot be done in real-time. The purpose of this thesis is to produce a low-cost real-time Digital Signal Processor (DSP) implementation of an MPEG encoder. The DSP will provide an MPEG bitstream to the PC that can be saved to disk.

The input to the DSP will be an analog audio signal. A codec provides the DSP with 16-bit samples of the signal. The DSP compresses these 16-bit samples using MPEG-1 layer 1 compression. Then it formats the compressed data to the correct MPEG-1 bitstream, and transmits it to the PC over its byte-wide host interface. On the PC side, a program receives the data from the DSP and saves the MPEG data to the disk. An MPEG-1 layer 1 player can play the resulting file.

TABLE OF CONTENTS

List of Figures	iii
List of Tables	iv
List of Equations	v
Glossary	vi

1 AUDIO COMPRESSION METHODS	1
1.1. PCM.....	1
1.2. Time domain compression.....	2
1.2.1. μ -law Encoding.....	2
1.2.2. DPCM and ADPCM	3
1.3. Transform Coding.....	4
1.4. Subband Coding	5
2. QUANTIZATION.....	7
3. PSYCHOACOUSTICS.....	8
3.1. Absolute threshold.....	9
3.2. Masking	10
4. MPEG COMPRESSION.....	14
4.1. Overview	14
4.2. MPEG Frame Format	15
4.2.1. MPEG Header	16
4.2.2. Layer 1 Audio Data.....	19
4.3. Subband Processing.....	21
4.4. Psychoacoustic Model	24
4.4.1. FFT Analysis.....	24
4.4.2. Determination of the sound pressure level	25
4.4.3. Considering the threshold in quiet.....	26
4.4.4. Finding tonal and non-tonal components.....	26
4.4.4.1. Tonal.....	26
4.4.4.2. Non-Tonal	27
4.4.5. Decimation of tonal and non-tonal masking components.....	28
4.4.6. Calculation of the individual masking thresholds	28
4.4.7. Calculation of the global masking threshold.....	32
4.4.8. Determination of the minimum masking threshold.....	32
4.4.9. Calculation of the signal to mask ratio.....	33

4.5. Bit Allocation	33
4.6. Frame Packing	35
5. PC SOFTWARE IMPLEMENTATION	37
5.1. Graphics.....	37
5.2. Bit File	38
5.3. MPEG I/O.....	40
5.4. DSP Considerations.....	41
5.4.1. 24 Bit Mode, Scaling and FFT.....	41
5.4.2. Logarithms	43
6. DSP SOFTWARE IMPLEMENTATION	45
6.1. DSP Selection.....	45
6.2. Codec Interface.....	48
6.3. Pipeline	50
6.4. Foreground Processing	52
6.5. Simulation.....	53
6.6. Port B Testing.....	55
6.7. Bit Allocation	56
7. DSP - PC INTERFACE	59
7.1. Protocol.....	59
7.2. Hardware Needed	61
7.3. PC Implementation	62
7.4. DSP Implementation.....	63
7.5. Transfer Rates.....	65
8. CONCLUSION.....	66
8.1. Difficulties Encountered.....	69
8.2. Recommendations for Future Work	70
9. REFERENCES	71

LIST OF FIGURES

FIGURE 1: DPCM SIGNAL.....	3
FIGURE 2: ABSOLUTE THRESHOLD.....	10
FIGURE 3: ILLUSTRATION OF FREQUENCY MASKING	11
FIGURE 4: CRITICAL BAND RATE VS. FREQUENCY	13
FIGURE 5: MPEG ENCODER BLOCK DIAGRAM.....	14
FIGURE 6: MPEG AUDIO FRAME FORMAT	16
FIGURE 7: MPEG LAYER 1 FRAME FORMAT	20
FIGURE 8: MPEG LAYER 1 SUBBAND SAMPLE FORMAT.....	20
FIGURE 9: CALCULATION OF SUBBAND SAMPLES	22
FIGURE 10: PLOT OF THE $C[i]$ VECTOR.....	23
FIGURE 11: FREQUENCY RESPONSE OF SUBBAND FILTERS	23
FIGURE 12: INDIVIDUAL MASKING THRESHOLD: $X(z(j))+VF$	31
FIGURE 13: 24-BIT FIXED POINT REPRESENTATION.....	46
FIGURE 14: SSI DATA FRAME.....	49
FIGURE 15: FRAME PROCESSING	51
FIGURE 16: PORT B TESTING.....	56
FIGURE 17: FULL HANDSHAKE READ OPERATION	60
FIGURE 18: DOUBLE EDGE ACKNOWLEDGE.....	60
FIGURE 19: STATE MACHINE FOR DSP TRANSMITTER	64
FIGURE 20: DSP FOREGROUND LOOP.....	68

LIST OF TABLES

TABLE 1: MPEG HEADER	17
TABLE 2: TOTAL BIT-RATE OF MPEG FRAME	18
TABLE 3: CHANNEL MODE.....	18
TABLE 4: LAYER 1 AND 2 MODE EXTENSIONS.....	19
TABLE 5: LAYER 3 MODE EXTENSIONS	19
TABLE 6: LAYER 1 BIT ALLOCATIONS	20
TABLE 7: MPEG LAYER 1,2 SCALEFACTORS	21
TABLE 8: $X(k) - X(k+j) \geq 7 \text{ dB}$	27
TABLE 9: FREQUENCIES CORRESPONDING TO INDEX I	29
TABLE 10: LAYER I SIGNAL TO NOISE RATIOS	34
TABLE 11: DSP56302 vs. DSP56302.....	46
TABLE 12: STATE TABLE FOR PC RECEIVER.....	63
TABLE 13: STATE TABLE FOR DSP TRANSMITTER	64
TABLE 14: SLOT 2 IDLE TIMES AS A FUNCTION OF BIT-RATE.....	69

LIST OF EQUATIONS

EQUATION 1: CRITICAL BAND RATE AS A FUNCTION OF FREQUENCY	13
EQUATION 2: CRITICAL BANDWIDTH AS A FUNCTION OF FREQUENCY	13
EQUATION 3: SCALEFACTOR AS A FUNCTION OF INDEX.....	21
EQUATION 4: EXPRESSION FOR M MATRIX	22
EQUATION 5: HANN WINDOW	24
EQUATION 6: CALCULATION OF THE LOG-POWER DENSITY SPECTRUM	25
EQUATION 7: DETERMINATION OF SOUND PRESSURE LEVEL.....	26
EQUATION 8: CALCULATION OF TONAL SPL.	27
EQUATION 9: INDIVIDUAL MASKING THRESHOLD	30
EQUATION 10: MASKING INDEX.....	30
EQUATION 11: MASKING FUNCTION	31
EQUATION 12: GLOBAL MASKING THRESHOLD	32
EQUATION 13: BITS PER LAYER I FRAME.....	34
EQUATION 14: CONVERTING FROM POWER DOMAIN TO LOG DOMAIN	44
EQUATION 15: MODIFIED LOG DOMAIN CALCULATIONS	44
EQUATION 16: APPROXIMATION OF SMR	57

GLOSSARY

<i>ASIC</i>	Applications Specific Integrated Circuit	
<i>Bark</i>	Unit of frequency measure based on critical bandwidth	12
<i>Codec</i>	Coder/decoder - provides 16 bit samples of analog signal	
<i>DCT</i>	Discrete Cosine Transform	5
<i>dz</i>	Difference between two frequencies in barks.	12
<i>FIR</i>	Finite Impulse Response - filter based on the weighted sum of input values.	22
<i>FFT</i>	Fast Fourier Transform - time to frequency transform	4
<i>Intensity Stereo</i>	Subbands that are less than the bound are coded in stereo, while the sum of the two channels is coded for the higher subbands.	18
<i>ISR</i>	Interrupt Service Routine - code executed to service an interrupt	47
<i>kbps</i>	Kilo bits per second	
<i>LSB</i>	Least Significant Byte - lower 8 bits of a 16 bit value	40
<i>Psychoacoustic</i>	The science of how the ear and brain perceive sounds.	8
<i>Masking</i>	The effect of one tone or noise to make a test tone inaudible or less loud.	10
<i>Masking function</i>	The SPL of the surrounding frequencies that is masked out by the masking frequency	11
<i>MIPS</i>	Million Instructions Per Second	
<i>MNR</i>	Mask to Noise Ratio	33
<i>MPEG</i>	Moving Pictures Experts Group	
<i>MSB</i>	Most Significant Byte - upper 8 bits of a 16 bit value	40
<i>MS Stereo</i>	Encoding of stereo signals as their sum and difference	18
<i>Non-Tonal</i>	Noise like frequency components	26
<i>PC</i>	Personal Computer	
<i>PCM</i>	Pulse Code Modulation	1
<i>Quantization</i>	Continuous represented as discrete values.	7
<i>SMR</i>	Signal to Mask Ratio	14
<i>SNR</i>	Signal to Noise Ratio	7
<i>SPL</i>	Sound Pressure Level = $10 \log_{10}(\text{power})$	9
<i>Slot</i>	Part of a bitstream. 4 bytes in layer 1. 1 in layers 2 and 3	17
<i>Tonal</i>	Sinusoidal like frequency components	26

1 Audio Compression Methods

There are three forms of audio compression. Time domain compression transmits a reduced amount of information for each sample. Transform-based compression performs a transform on the data, compacting the information present in the audio signal, and transmits the transform information. Subband based compression breaks the input signal into multiple subbands of reduced frequency.

1.1. *PCM*

Pulse Code Modulation (PCM) represents an uncompressed audio signal. Each channel of the input signal is regularly sampled and the value of the analog signal is represented by a linear 2's complement value. The number of bits for each sample determines the number of quantization steps that are available. Typically, 8 or 16 bits represent the data. If the audio signal consists of more than one channel, then the bitstream is constructed by alternating samples from each channel. The data portion of .WAV files is in PCM format. CD audio is also stored as PCM. CD audio consists of two channels of 16 bit data sampled at 44.1 kHz. This requires a total bandwidth of $2 \times 16 \times 44.1k = 1.4112$ Mbps.

1.2. Time domain compression

Time domain compression uses the current sample, and possible previous samples to transmit fewer bits for each sample in the audio stream. The simplest compression method discards the least significant bits of each sample. Transmitting the most significant 8 bits of a 16 bit audio stream, easily achieves a compression ratio of 2:1. However, reducing the number of quantization levels from 65536 to 256 reduces the quality of the signal.

1.2.1. μ -law Encoding

Most of the time, the audio signal is close to the zero line. μ -law distributes the possible values according to an eight bit log based code so more codes are located near the zero line. This decreases the amount of noise present at smaller values, but increases it for larger values. The codes with higher amplitudes are spaced further apart. The range of values coded with the eight-bit μ -law code need about 14 bits if represented as PCM. The following equation transfers from μ -law back to PCM. [1, p.1]

$$y = \begin{cases} 255 - \frac{127}{\ln(1 + \mu)} \times \ln(1 + \mu|x|) & \text{for } x \geq 0 \\ 127 - \frac{127}{\ln(1 + \mu)} \times \ln(1 + \mu|x|) & \text{for } x < 0 \end{cases}$$

1.2.2. DPCM and ADPCM

The next sample in an audio signal has some correlation to the previous samples. Differential Pulse Code Modulation (DPCM) and Adaptive DPCM compression methods take advantage of this. The encoder uses a predictor function to predict a value for the current sample from the knowledge of the previous samples. This predicted value is subtracted from the actual value and the resulting error is coded and placed in the bitstream. The decoding process is the reverse. Using the same predictor function, the decoder predicts what the next output sample should be and adds in the error coded in the bitstream.

Figure 1: DPCM Signal

The quality of this compression is based on how closely the predictor function matches the actual audio signal. A more accurate predictor generates smaller errors that require fewer bits to encode. The number of bits used for the error is usually

fixed, and much smaller than the 8 or 16 bits needed to represent the same signal with PCM. This limited number of bits leads to a problem when the error cannot be represented in the allocated bits. The extra error is then carried over into the next sample's error. This has the effect of delaying the data as shown in Figure 1. This example shows a sudden change in the input signal. The steps of the DPCM signal represent the maximum error value that can be coded. Since the change was large, the DPCM signal continues to be saturated at the maximum error for almost six values. ADPCM adapts the predictor to better match the audio signal. The parameters of the predictor need to be added to the bitstream so that the decoder can use the same predictor.

1.3. Transform Coding

Audio data is highly correlated. A transform of the audio data can be less correlated than the audio data. The transform places the energy of the signal in a more compact area. This more concise form of the signal information can be transmitted with much less data requirements.

One example of transform coding is the Discrete Fourier Transform (DFT). An audio signal that contains only one frequency component has energy distributed over the entire sample vector. The information in the DFT of this signal concentrates into one frequency component if the frequency component is a harmonic of the fundamental frequency. The magnitude and phase of this one frequency can be transmitted and the destination can reconstruct the exact waveform. Other transforms besides the DFT

can decorrelate the information present in the audio data. The discrete cosine transform (DCT) is a real domain transform that usually performs better than the DFT to compact the information. The JPEG compression standard uses a two-dimensional DCT to compress images, which are two-dimensional signals.

1.4. Subband Coding

According to the Nyquist sampling theorem, the largest frequency component present in a sampled signal equals half the sampling frequency (f_s). By analyzing N digital samples of a signal, there are $\frac{N}{2}$ frequency components that are present in the signal from DC to $\frac{f_s}{2}$ with each component being an integer multiple of the fundamental frequency of $\frac{f_s}{N}$. If the audio signal is filtered by a bandpass filter that has a passband width of $\frac{f_s}{2M}$, then the resulting signal only contains $\frac{N}{2M}$ frequency components. It can be shown that the resulting signal can be downsampled by a factor of M to produce a signal that has a bandwidth of $\frac{f_s}{2M}$ and contains $\frac{N}{M}$ samples. This reduced bandwidth signal is called a subband, because it contains the information for only a portion of the frequency spectrum of the original signal. [2, p.18]

Subband coding uses a number of bandpass filters to break the input signal into multiple subbands. Enough filters are used to cover the entire frequency spectrum

and are designed so there is no overlap between subbands. The resulting subband signals can be coded separately. Since each subband has information from a localized portion of the spectrum, bits can be allocated to subbands that represent the more important frequencies of the signal.

2. Quantization

An audio signal is a continuous analog voltage. In order for a digital device to process the signal, the analog signal must be converted to a digital signal. An Analog to Digital Converter (ADC) samples the voltage at fixed rate and converts the voltage level into a digital code. This process of conversion from analog to digital is called quantization. Quantization approximates a value to a code that consists of discrete levels. In the case of PCM the levels are evenly spaced, while μ -law encoding consists of a non-uniform code. For any code, using more quantization levels improves the quality of the approximation.

Quantization determines the quality of audio compression. Each additional bit used to code a signal, increases the signal to noise (SNR) by approximately 6 dB. Compression methods that break a signal up into its components distribute the bits so that a larger SNR is provided to components that contain the most important information.

Generally, using more bits to code audio data increases the SNR and improves the quality of the sound, while using a higher sampling rate increases the frequency resolution of the audio signal.

3. Psychoacoustics

Psychoacoustics is the science of human sound perception. The ear is an excellent microphone, capable of hearing frequencies from about 20 Hz up to 20kHz. The human auditory system is broken into three parts: the outer, middle and inner ear.

The outer ear channels sound to travel down the ear canal to hit the eardrum. The length of the outer ear canal is approximately 2 cm. This length is equal to one quarter of the wavelength of a 4kHz tone. The outer ear canal acts as an open pipe and resonates at that frequency to best transmit it to the eardrum. This contributes to the sensitivity of human hearing in the 4kHz range. [3, p.21]

The inner ear is filled with a salty fluid. The middle ear converts the vibrations of the air into vibrations of the liquid fluid. This can best be described as impedance matching. The eardrum, hammer, anvil and stirrup act as a mechanical lever to transfer the energy from one fluid (air) into another. The efficiency of the impedance matching is a function of frequency, and has a best match at frequencies around 1kHz. [3, p.22]

The inner ear, or cochlea, is shaped like a snail and is filled with fluid. This fluid transmits the oscillations to the basilar membrane, which runs the length of the cochlea and supports the organ of Corti. The organ of Corti converts the vibrations present on hairs to electrical signals that the auditory nerve can transmit. The cochlea

consists of 2.5 turns and makes the basilar membrane approximately 32 mm in length. The thickness of the basilar membrane increases along the length of the cochlea. [3, pp.22-23]

Different frequencies cause different places along the basilar membrane to vibrate. Higher frequencies cause the thinner basilar membrane located by the opening of the cochlea to vibrate. Lower frequencies cause the thicker basilar membrane located towards the end of the twisting cochlea to vibrate. The organ of Corti along the length of the basilar membrane senses only the oscillations that cause that portion of the basilar membrane to vibrate, and therefore is only sensitive to a small range of frequencies. The delay to detect a frequency increases along the length of the cochlea. High frequencies are detected first because they are closest to the opening of the cochlea. A 1.5kHz tone may have a delay around 1.5ms, and the low frequencies at the end of the cochlea have delays of almost 5ms. [3, pp.26-27]

3.1. Absolute threshold

The absolute threshold is the minimum Sound Pressure Level (SPL) required for a sound, in order for a person to hear it. The threshold, shown in Figure 2, varies with frequency. Human speech ranges from around 100 Hz up to 7 kHz. The figure shows the human ear is most sensitive to frequencies between 1kHz and 6 kHz. These frequencies are in the same range as human speech. Each person has a different hearing threshold. This curve has been obtained experimentally and is considered to

be an average threshold. There are many different ways to obtain this data, but the basic test is to play a test tone at a frequency and vary the power of the tone until the listener can hear or cannot hear the tone.

Figure 2: Absolute Threshold

3.2. Masking

Masking is the phenomenon of a tone or noise causing the loudness of a tone to decrease or be inaudible. This is best illustrated by an example. Two people are talking to each other by a street. The listener can clearly hear and understand what the speaker is saying. If a truck drives by, the sound of the truck masks out the voice of the speaker. The listener may have difficulty understanding what the speaker is saying, or may not hear the speaker's voice at all.

There are many different types of masking. The objective when studying masking is to see the effect of a masking tone or noise on a pure tone, or test tone. Masking can occur by broadband noise (white noise), high pass noise, low pass noise, narrow band noise, or a pure tone. Temporal effects exist that also cause masking. Once a masker is turned off, there may be a short delay before the tones that were masked are audible again. Each type of masker has a different masking function. The masking function shows the Sound Pressure Level (SPL) that the surrounding frequencies are masked by. If the SPL of a masked frequency is less than the masking function, then the frequency will be inaudible. If the SPL of the masked frequency is greater than the masking function then the frequency will be partially masked and will be heard at a reduced loudness.

Figure 3: Illustration of Frequency masking

Masking is a function of the type and frequency of the masker and the SPL of the masker. For most types of maskers, the shape of the masking function changes for different masking frequencies. Figure 3 illustrates the general shape of a masking function. Frequency is not the best scale to use when dealing with the psychoacoustic

properties of the human auditory system. Remember that the length along the basilar membrane determines what frequency the ear will hear. However, frequencies do not change linearly along the basilar membrane. Masking can be visualized as the vibrations caused by the masker effecting the neighboring frequencies that are within a certain distance along the basilar membrane.

It is theorized that when a tone is masked by a noise, then only the frequencies near to the tone contribute to the masking. This range of frequencies is a function of frequency and called the critical bandwidth. Critical bandwidth is approximately 100Hz for frequencies less than 500 Hz, and $0.2f$ for frequencies greater than 500Hz. Numbering the edges of adjacent critical bandwidths creates the critical band rate scale. The first complete bandwidth starts at 0 and goes to 100Hz, and the second starts at 100Hz and goes to 200Hz. The unit of Bark is defined as the transition between two of these bandwidths. Shown in Figure 4 is a plot of critical band rate in Barks (z) vs. frequency (Hz). The critical band rate better suites psychoacoustic studies because it has almost a linear relationship to the length along the basilar membrane. The shape of masking functions represented, as a function of dz (change in critical band rates) is independent of the masking frequency. Within the auditory frequency range, the value of the critical band rate at any frequency can be approximated by Equation 1. The approximate critical bandwidth at any frequency in the range is given by Equation 2. [3, p.147]

Figure 4: Critical band rate vs. Frequency

Equation 1: Critical band rate as a function of frequency

$$z / \text{Bark} = 13 * \arctan\left(\frac{0.76f}{\text{kHz}}\right) + 3.5 * \arctan\left(\frac{f}{7.5\text{kHz}}\right)$$

Equation 2: Critical bandwidth as a function of frequency

$$\Delta f_G / \text{Hz} = 25 + 75 \left[1 + 1.4 \left(\frac{f}{\text{kHz}} \right)^2 \right]^{0.69}$$

4. MPEG Compression

MPEG is a perceptually-lossless subband-based compression method. Since information is lost in the coding process, the original signal cannot be reconstructed from the compressed data. MPEG removes information from the audio signal that a listener would not be able to hear in the original signal.

4.1. Overview

Figure 5: MPEG Encoder Block Diagram

The encoder block diagram for the MPEG ISO/IEC 11172-3 standard is shown in Figure 5. The filter bank divides the input signal into 32 subbands. The psychoacoustic model calculates the Signal to Mask Ratio (SMR) for each of the subbands. The bit allocation and quantization stage uses this SMR to determine how each subband should be quantified. The final stage properly formats the information into an MPEG frame. The standard allows for three different layers of compression. Each layer has a different frame format and increasing encoder complexity. The

audio data to be compressed can be originally sampled at 32kHz, 44.1kHz or 48kHz. For each layer, there are 14 possible bit-rates for the compressed data.

The audio data is broken down into 32 subbands of reduced frequency. These subbands are equally spaced in layer 1 and 2. In layer 3, the bandwidths of the subbands are more closely related to critical bandwidth. MPEG audio compression uses psychoacoustic principles to determine the signal to mask ratio for each subband. This information is used to dynamically distribute the available bits among the subbands.

The number of samples used in a layer 2 or 3 frame is three times that of layer 1. This provides more information for the psychoacoustic model and better frequency resolution. Except for more samples being used in the processing, layer 2 does not have any psychoacoustic improvements over layer 1. The coding of bit allocation and scalefactors data is improved to reduce the amount of data. In addition to improved MPEG compression, Layer 3 also provides Huffman coding on the resulting MPEG frames.

4.2. MPEG Frame Format

MPEG audio data is a frame-based bitstream. All information presented here regarding the MPEG frame format is from [4], the ISO/IEC 11172-3 standard. For a given bit-rate, the maximum size of a frame is fixed. Each frame represents the coded

information from 384 samples of audio data for layer 1 or 1152 samples in layer 2 and 3. Figure 6 shows the general format for all layers of MPEG audio compression. All layers of MPEG audio share the same 32-bit header. The error check is an optional 16 bit CRC that can be used to detect errors present in the bitstream. Each layer has a different format for the data portion of the MPEG frame. Any bits that are remaining in the frame after filling in the header, Error Check, and audio data is ignored, and can be used for ancillary data.

Header	Error Check	Audio Data	Ancillary Data

Figure 6: MPEG audio frame format

4.2.1. MPEG Header

The header contains all the information needed for a MPEG decoder to interpret the fields of the rest of the MPEG frame. Table 1 shows the format of the MPEG header and the number of bits for each field. The syncword is the bit string '1111 1111 1111'. When the decoder encounters twelve ones in a row in the bitstream it interprets it as the start of a new frame. Care is taken in the data portion of the frame so that the syncword does not appear accidentally.

Table 1: MPEG header

Field	Number of Bits
Syncword	12
ID	1
Layer	2
Protection bit	1
bitrate_index	4
Sampling frequency	2
padding bit	1
private bit	1
Mode	2
mode extension	2
Copyright	1
original/copy	1
Emphasis	2

ID is a '1' to indicate ISO/IEC 11172-3 audio. The ID of '0' is reserved. Layer indicates what compression layer was used to encode the frame and indicates the format for the data portion of the frame. '11', '10', and '01' are used to represent layer 1, 2 and 3 respectively. The layer of '00' is reserved. A protection bit set to '0' indicates that the MPEG frame includes a 16 bit CRC. The 4 bit bit-rate index is used as an index into Table 2 to determine the total bit-rate of the MPEG frame. The 2 bit sampling frequency indicates the original audio data sample rate. '00', '01', and '10' indicate 44.1kHz, 48kHz and 32kHz respectively. The sampling frequency of '11' is reserved. For a sample rate of 44.1kHz, it is necessary to pad the audio data so that the mean bit-rate is equal to the bit-rate indicated by the bitrate_index. The padding bit is set to '1' if the current frame contains an extra slot for padding.

Table 2: Total Bit-rate of MPEG frame

bitrate_index	bit-rate specified (kbits/sec)		
	Layer 1	layer 2	layer 3
0000	Free	free	free
0001	32	32	32
0010	64	48	40
0011	96	56	48
0100	128	64	56
0101	160	80	64
0110	192	96	80
0111	224	112	96
1000	256	128	112
1001	288	160	128
1010	320	192	160
1011	352	224	192
1100	384	256	224
1101	416	320	256
1110	448	384	320
1111	forbidden	forbidden	forbidden

The private bit is not used by the standard and can be used for private use. MPEG-1 has support to encode up to two channels of audio. The mode and the mode extension indicate what number and type of channels are being encoded, and how the decoder should interpret those channels. The copyright bit is set if the bitstream is copyrighted material. The original/copy bit is set if the bitstream is original.

Table 3: Channel Mode

Mode	Mode specified
00	Stereo
01	Joint Stereo (intensity stereo and/or MS stereo)
10	Dual channel
11	Single channel

Table 4: Layer 1 and 2 Mode Extensions

Mode	
00	Subbands 4-31 in intensity stereo, bound = 4
01	Subbands 8-31 in intensity stereo, bound = 8
10	Subbands 12-31 in intensity stereo, bound = 12
11	Subbands 16-31 in intensity stereo, bound = 16

Table 5: Layer 3 Mode Extensions

Mode	Intensity stereo	MS Stereo
00	off	off
01	on	off
10	off	on
11	on	on

4.2.2. Layer 1 Audio Data

Each layer has a different format for the audio data section of the frame. Only the format for a layer 1 frame containing single channel data is described here. The format for the MPEG-1 layer 1 frame can be seen in Figure 7. MPEG audio consists of 32 subbands. All values are inserted into the bitstream in order from lowest subband to highest subband. The allocation section specifies how many bits are allocated for each subband. The four bit allocation value is used as an index into Table 6 to obtain the number of bits for each subband. If a subband has a bit allocation of zero, then no scalefactor or subband sample values for that subband are present in the frame. For each of the transmitted subbands, a six bit scalefactor index is placed in the bitstream. This index corresponds to the entry in Table 7 that all the subband samples of that subband have been normalized to. The values in the scalefactor table can be represented by Equation 3. Each scalefactor in the table is a

effectively a one third bit shift to the left of the previous scalefactor. The subband samples are added to the bitstream in the order that they were calculated. The first set of subband sample values calculated is output in increasing subband order. The second set follows the third and so on, until the twelfth set of subband samples is output. Figure 8 shows this graphically. In the figure, sb0..sb31 represents the subband samples that were calculated at the same time, and have bits allocated to them.

Header 32 bits	Optional CRC 16 bits	Allocation 32 x 4 bits	Scalefactors 6 bits / subband	Subband Samples 12 x 2-15 bits / subband

Figure 7: MPEG layer 1 frame format

sample 0	sample 1	sample 2	...	sample 10	sample 11
sb0..sb31	sb0..sb31	sb0..sb31	...	sb0..sb31	sb0..sb31

Figure 8: MPEG layer 1 Subband Sample Format

Table 6: Layer 1 Bit Allocations

Allocation	Bits per sample	Allocation	Bits per sample
0	0	8	9
1	2	9	10
2	3	10	11
3	4	11	12
4	5	12	13
5	6	13	14
6	7	14	15
7	8	15	forbidden

Equation 3: Scalefactor as a function of index

$$scalefactor(index) = 2.0 * \left(\sqrt[3]{\frac{1}{2}} \right)^{index}$$

Table 7: MPEG Layer 1,2 scalefactors

inde x	Scalefactor	inde x	Scalefactor	inde x	scalefactor
0	2.000000000000000	21	0.015625000000000	42	0.00012207031250
1	1.58740105196820	22	0.01240157071850	43	0.00009688727124
2	1.25992104989487	23	0.00984313320230	44	0.00007689947814
3	1.000000000000000	24	0.007812500000000	45	0.00006103515625
4	0.79370052598410	25	0.00620078535925	46	0.00004844363562
5	0.62996052494744	26	0.00492156660115	47	0.00003844973907
6	0.500000000000000	27	0.003906250000000	48	0.00003051757813
7	0.39685026299205	28	0.00310039267963	49	0.00002422181781
8	0.31498026247372	29	0.00246078330058	50	0.00001922486954
9	0.250000000000000	30	0.001953125000000	51	0.00001525878906
10	0.19842513149602	31	0.00155019633981	52	0.00001211090890
11	0.15749013123686	32	0.00123039165029	53	0.00000961243477
12	0.125000000000000	33	0.000976562500000	54	0.00000762939453
13	0.09921256574801	34	0.00077509816991	55	0.00000605545445
14	0.07874506561843	35	0.00061519582514	56	0.00000480621738
15	0.062500000000000	36	0.000488281250000	57	0.00000381469727
16	0.04960628287401	37	0.00038754908495	58	0.00000302772723
17	0.03937253280921	38	0.00030759791257	59	0.00000240310869
18	0.031250000000000	39	0.00024414062500	60	0.00000190734863
19	0.02480314143700	40	0.00019377454248	61	0.00000151386361
20	0.01968626640461	41	0.00015379895629	62	0.00000120155435

4.3. Subband Processing

There are 384 samples in an MPEG layer 1 frame. These 384 samples are critically filtered to produce 384 subband samples. Every 32 samples, the most recent 512 points are filtered by a polyphase filter. The polyphase filter contains 32 equally spaced non-overlapping bandpass filters. The filter produces one output for each of the subbands. This process repeats 12 times for each frame, to produce 12 filtered

points for each subband. This effectively runs the 32 bandpass filters on the same data and downsamples the filtered data to produce critically sampled subbands. Figure 9 shows the method of calculating the subband samples as stated in the standard. It can be shown that each output $S[i]$ can be represented by a 512 point FIR filter. [5, p. 64] The frequency response of the first four subband filters is shown in Figure 11. The top plot shows the complete frequency spectrum from 0 to π , and the bottom plot shows a close up in the area where the filters begin to overlap. These figures were produced by matlab scripts to examine the frequency response of the subband filters.

Figure 9: Calculation of Subband Samples

Equation 4: Expression for M matrix

$$M[i, k] = \cos \left[\frac{(2i+1)(k-16)\pi}{64} \right] \quad \text{for } i = 0..31, k = 0..63$$

Figure 10: Plot of the $C[i]$ vector

Figure 11: Frequency Response of Subband Filters

4.4. Psychoacoustic Model

The psychoacoustic model portion of the encoder determines the SMR for the first 30 subbands. The SMR provides information about how much of the signal is not being masked in each subband. A large SMR indicates that the global masking threshold masked less of the signal than a subband that has a smaller SMR. There are two psychoacoustic models used to calculate the SMR. Both models can be used for any of the three layers of compression, but Model 1 is generally used for layer 1 and 2 and Model 2 is used for layer 3. Following are the steps to the psychoacoustic model 1 as suggested by the ISO/IEC 11172-3 standard.

4.4.1. FFT Analysis

MPEG layer 1 uses a 512 point Fast Fourier Transform (FFT) to determine the frequency spectrum of the current frame. Since there are only 384 samples in a frame, an additional 128 points need to be used to calculate the FFT. The 384 samples of the current frame are centered between the last 64 samples from the previous frame and the first 64 samples of the next frame. The 512 samples are windowed by the modified Hann window shown in Equation 5 before calculating the FFT.

Equation 5: Hann Window

$$h(i) = \sqrt{8/3} * 0.5 * \left(1 - \cos\left(\frac{2\pi i}{512}\right)\right)$$

The result from the FFT is a complex vector of size 512. Since the input to the FFT is a real signal, the k^{th} frequency component will be equal to the $k+N/2$ frequency component. The upper half of components is discarded and the lower half is squared to determine the power frequency spectrum. This spectrum is converted to the log power domain and normalized to a maximum value of 96 dB. Equation 6 shows the calculations performed to determine the log-power domain FFT. This frequency spectrum is used in the rest of the psychoacoustic model stage to calculate the Signal to Mask Ratio (SMR).

Equation 6: Calculation of the log-power density spectrum

$$X(k) = 10 * \log_{10} \left| \frac{1}{N} \sum_{n=0}^{N-1} h(n) * x(n) * e^{-j \frac{2\pi}{N} nk} \right|^2 \text{ dB} \quad \text{for } k = 0.. \frac{N}{2}$$

4.4.2. Determination of the sound pressure level

The sound pressure level is calculated for each of the subbands. The sound pressure level indicates how much energy is present in each subband. The sound pressure level is determined by finding the maximum frequency spectrum in the subband and comparing it to log-power representation of the scalefactor for that subband. As shown in the standard, the expression for the sound pressure level of subband n can be represented by Equation 7. The expression $\text{scalefactor}(n) * 32768$ represents the largest subband sample possible for subband n. The -10 is needed to adjust the maximum sample value to a RMS value. .

Equation 7: Determination of Sound Pressure Level

$$L_{sb} = MAX[X(k), 20 * \log_{10}(scalefactor(n) * 32768) - 10] \quad \text{for } X(k) \text{ in subband } n$$

4.4.3. Considering the threshold in quiet

The threshold in quiet is the minimum sound level that can be heard by a person. Sounds lower than this threshold are inaudible to the listener and can be removed from the audio signal. There is an offset of -12dB added to the threshold in quiet when the bit rate is greater or equal to 96 kbps. This downward shift of the threshold in quiet allows for less of the masking components to be removed. With the higher bit rates, more masking components are needed so there more data is available for building the frame.

4.4.4. Finding tonal and non-tonal components

Compression uses many aspects of psychoacoustics. MPEG layer 1 and 2 only use two types of maskers -- tonal and non-tonal maskers -- when determining the global masking threshold. Each of the two types of maskers has a different shape for the masking function.

4.4.4.1. Tonal

Tonal frequency components are sinusoidal. A local maximum or peak in the frequency spectrum indicates a tonal frequency component. The frequency spectrum consists of 257 values indexed by k from 0 to 256. A frequency component in the

spectrum that is larger than the previous and next frequency component is marked as local maximum. A frequency marked as a local maximum is only marked as a tonal component if its SPL is 7 dB larger than neighboring frequency components within a relative distance (j). The relative distance is determined by the k^{th} index of the frequency component and is shown in Table 8.

Table 8: $X(k) - X(k+j) \geq 7$ dB

Relative distance (j)	k index of local maximum
-2,+2	$2 \leq k < 63$
-3,-2,+2,+3	$63 \leq k < 127$
-6,-5,-4,-3,-2,+2,+3,+4,+5,+6	$127 \leq k \leq 250$

Once a frequency component is marked as tonal, the sound pressure level is calculated for the frequency component. Adding the power of the two adjacent spectral lines with the tonal component does this. Since the frequency spectrum is represented in the log power domain, the calculation of the SPL must be done using Equation 8. All of the frequency components within the relative distance are set to a power of negative infinity decibels. This prevents those frequencies from being used in the non-tonal calculations.

Equation 8: Calculation of Tonal SPL.

$$SPL(X(k)) = 10 * \log_{10} \left(10^{\frac{X(k-1)}{10}} + 10^{\frac{X(k)}{10}} + 10^{\frac{X(k+1)}{10}} \right)$$

4.4.4.2. Non-Tonal

One non-tonal frequency component is assigned for each of the critical bands. A non-tonal component is a representation of the noise level present in the critical band. The

SPL of the non-tonal component is the sum of all of the frequency components in each critical band not zeroed out by the tonal calculations or picked as tonal components. The non-tonal component is assigned the k index that most closely relates to the geometric mean of the critical band.

4.4.5. Decimation of tonal and non-tonal masking components

Decimation is used to reduce the number of tonal and non-tonal components that will be used when calculating the global masking threshold. Tonal and non-tonal components lower than the threshold in quiet at the components frequency are removed from the masking components list. Tonal components that are close together will produce similar effects when calculating the global masking threshold. Tonal components that are within 0.5 bark of each other are compared. The tonal component with the lower SPL is removed from the list.

4.4.6. Calculation of the individual masking thresholds

Not all of the 256 frequency components are used when calculating the global masking threshold. A reduced number of them (108 in layer 1) are used for the calculations. This reduced set of frequencies is shown in Table 9, and is indexed by i . For the first six subbands (3kHz), all frequencies are used as i indices. For the next six subbands, every other frequency component is used, and for the remaining

subbands, every fourth frequency is used. Tonal and non-tonal masking components are assigned an index i that is closest to the frequency of the masking component.

Table 9: Frequencies corresponding to index i

i	Frequency	i	Frequency	i	Frequency	i	Frequency
1	62.5	28	1750.0	55	3875.0	82	8500.0
2	125.0	29	1812.5	56	4000.0	83	8750.0
3	187.5	30	1875.0	57	4125.0	84	9000.0
4	250.0	31	1937.5	58	4250.0	85	9250.0
5	312.5	32	2000.0	59	4375.0	86	9500.0
6	375.0	33	2062.5	60	4500.0	87	9750.0
7	437.5	34	2125.0	61	4625.0	88	10000.0
8	500.0	35	2187.5	62	4750.0	89	10250.0
9	562.5	36	2250.0	63	4875.0	90	10500.0
10	625.0	37	2312.5	64	5000.0	91	10750.0
11	687.5	38	2375.0	65	5125.0	92	11000.0
12	750.0	39	2437.5	66	5250.0	93	11250.0
13	812.5	40	2500.0	67	5375.0	94	11500.0
14	875.0	41	2562.5	68	5500.0	95	11750.0
15	937.5	42	2625.0	69	5625.0	96	12000.0
16	1000.0	43	2687.5	70	5750.0	97	12250.0
17	1062.5	44	2750.0	71	5875.0	98	12500.0
18	1125.0	45	2812.5	72	6000.0	99	12750.0
19	1187.5	46	2875.0	73	6250.0	100	13000.0
20	1250.0	47	2937.5	74	6500.0	101	13250.0
21	1312.5	48	3000.0	75	6750.0	102	13500.0
22	1375.0	49	3125.0	76	7000.0	103	13750.0
23	1437.5	50	3250.0	77	7250.0	104	14000.0
24	1500.0	51	3375.0	78	7500.0	105	14250.0
25	1562.5	52	3500.0	79	7750.0	106	14500.0
26	1625.0	53	3625.0	80	8000.0	107	14750.0
27	1687.5	54	3750.0	81	8250.0	108	15000.0

Each of the tonal and non-tonal masking components has an individual masking function that contributes to the global masking function. The standard gives equations to approximate the individual masking function of a tonal or non-tonal masker. The equations are functions of the critical band rate, the SPL of the masker and the critical band rate of the frequency being masked. Tonal and non-tonal

maskers have different masking properties. In the equations that follow, a subscript of *tm* indicates tonal masker, and a subscript of *nm* indicates non-tonal masker. The index *j* represents the *i*-index of the masker and an index of *i* represents the *i*-index of the frequency being masked. The expression of $z(j)$ represents the critical band rate for the frequency component with an *i*-index of *j*. Equation 9 shows the equations that summarize the calculations for the individual masking threshold. $X[z(j)]$ is the SPL of the tonal or non-tonal masker. The term av is called the masking index and is a function of the critical band rate of the masker. Equation 10 shows the expression for the masking index. The masking index for a tonal masker is always a larger negative number than a non-tonal masker at the same frequency. This shows that a non-tonal masker has more masking ability than a tonal masker of the same SPL does.

Equation 9: Individual Masking Threshold

$$\begin{aligned} LT_{tm}[z(j), z(i)] &= X_{tm}[z(j)] + av_{tm}[z(j)] + vf[z(j), z(i)] \\ LT_{nm}[z(j), z(i)] &= X_{nm}[z(j)] + av_{nm}[z(j)] + vf[z(j), z(i)] \end{aligned}$$

Equation 10: Masking Index

$$\begin{aligned} av_{tm} &= -1.525 - 0.275 * z(j) - 4.5 \\ av_{nm} &= -1.525 - 0.175 * z(j) - 0.5 \end{aligned}$$

The final term, vf , of the individual masking threshold calculation is the masking function. This masking function is a piecewise linear approximation of the actual masking function. A masker at the critical band frequency of $z(j)$ masks only the surrounding frequencies to the left by 3 barks and to the right by 8 barks. The vf term is a function of the SPL of the masker at $z(j)$ and the distance the surrounding

frequencies are from the masker. The equations for vf are shown in Equation 11. Except for the av term, the calculation of the individual masking function is identical for tonal and non-tonal maskers and is not a function of the masker frequency. Figure 12 shows the common masking properties for a masking tone of 0,25,50,75, and 100 dB. The effects of the SPL of the masker on the shape of the masking function can be seen in the figure.

Equation 11: Masking Function

$$\begin{aligned}
 vf &= 17 * (dz + 1) - (0.4 * X[z(j)] + 6) & \text{for } -3 \leq dz < -1 \\
 vf &= (0.4 * X[z(j)] + 6) * dz & \text{for } -1 \leq dz < 0 \\
 vf &= -17 * dz & \text{for } 0 \leq dz < 1 \\
 vf &= -(dz - 1) * (17 - 0.15 * X[z(j)]) - 17 & \text{for } 1 \leq dz < 8 \\
 dz &= z(i) - z(j)
 \end{aligned}$$

Figure 12: Individual Masking Threshold: $X(z(j)) + vf$

4.4.7. Calculation of the global masking threshold

The global masking threshold is only calculated at frequencies indexed by i in Table 9. The global masking threshold is the masking level that results from the threshold in quiet, the tonal maskers, and the non-tonal maskers. Overlapping masking functions have an additive effect. This means that if there are two maskers present, the effective masking level at a frequency will be the sum of the two masking levels. To calculate the global masking threshold, the masking level is calculated for each of the frequencies indexed by i . At each index i , the global masking threshold is equal to the sum of the threshold in quiet ($LT_q(i)$), and the individual masking effects from all of the tonal and non-tonal masking components.

Equation 12: Global Masking Threshold

$$LT_g(i) = 10 \log_{10} \left(10^{\frac{LT_q(i)}{10}} + \sum_{j=1}^m 10^{\frac{LT_{tm}(z(j), z(i))}{10}} + \sum_{j=1}^n 10^{\frac{LT_{nm}(z(j), z(i))}{10}} \right)$$

4.4.8. Determination of the minimum masking threshold

The global masking threshold is a vector of 108 SPL values spread over the first 30 subbands. The minimum masking threshold is the amount of masking that all of the frequency components contained in a subband are masked by. Some frequency components may have more masking, but it is necessary to find the SPL that can be removed from all of the frequencies contained within the subband. The minimum

masking threshold of all the frequencies in each subband determines the masking threshold of the subband. The following expression summarizes this process.

$$LT_{\min}(n) = \text{MIN}[LT_g(i)] \quad \text{for all } i \text{ in subband } n$$

4.4.9. Calculation of the signal to mask ratio

The calculation of the signal to mask ratio (SMR) is the final step of the psychoacoustic model. The SMR quantifies how much of the original signal energy is left in a subband after masking has occurred. To calculate the SMR, the minimum masking threshold ($LT_{\min}(n)$) is subtracted from the sound pressure level $L_{sb}(n)$ for each subband.

$$SMR(n) = L_{sb}(n) - LT_{\min}(n)$$

4.5. Bit Allocation

The size of an MPEG frame is fixed based on the target bit-rate. The target bit-rate and the sampling rate (as shown in Equation 13) determine the total number of bits available for the frame. Of these bits, 32 bits are reserved for the header, the subband bit allocations use $32 \times 4 = 128$ bits, and an optional 16 bits contain the CRC. The subband scalefactors and subband samples must be coded within the remaining bits. The bit allocation process is iterative. Each time through the loop, the number of bits for the subband with the minimum Mask to Noise Ratio (MNR) is increased. Initially, zero bits are allocated to each subband. The MNR is calculated for each

subband by subtracting the SNR from the SMR. Table 10 shows the SNR for the corresponding number of steps used. The subband that has the minimum MNR is chosen. It is checked to see if the number of additional bits required exceeds the number of available bits. If the subband has zero bits allocated to it, changing the bit allocation to 2 bits would require 30 bits: six bits for the scalefactor index and 24 bits for the twelve 2-bit samples. If the subband has a non-zero bit allocation, then adding one more bit of precision to the subband samples requires an additional 12 bits. Subbands are no longer picked for minimum MNR when they contain 15 bits per sample, or the additional bit requirements is larger than the available bits. This process is repeated until no more subbands can be added, and no more subbands can be increased by one bit.

Equation 13: Bits per Layer I Frame

$$\text{total bits} = 12 * \frac{\text{bitrate}}{\text{sampling\_frequency}} * 4$$

Table 10: Layer I Signal to Noise Ratios

Number of Steps	SNR (dB)
0	0.00
3	7.00
7	16.00
15	25.28
31	31.59
63	37.75
127	43.84
255	49.89
511	55.93
1023	61.96
2047	67.98
4095	74.01
8191	80.03
16383	86.05
32767	92.01

4.6. Frame Packing

Not all of the bits of the samples are transmitted. This reduction of bits reduces the precision of the values that can be transmitted. The subband samples are transmitted as fractions from -1 inclusive to 1 non-inclusive. The most significant bits of the sample are transmitted so the receiver can properly restore the sample. Many times the most significant bits of a number are all zero or one and do not provide the majority of the information stored in the sample. By dividing the samples by a number slightly larger than the maximum sample, the resulting values are normalized and closer to the value of 1 or -1. This has the effect of moving more information about the samples to the more significant bits of the number. In order to reverse this normalization process, the encoder and the decoder must use the same scalefactors. The scalefactors used for MPEG-1 Layer 1 and 2 are shown in Table 7.

All the samples in each subband are normalized to the smallest entry in the table that is larger than the maximum of the twelve samples of the subband. The resulting values will be between the values of -1 and 1 non-inclusive. To correct small changes around zero quantizing to different quantization levels, the samples go through a linear quantizer. The final step in preparing the subband samples for the bitstream inverts the most significant bit of every sample. This is done to help prevent the syncword of '1111 1111 1111' from appearing in the data.

Once the subband samples have been normalized and quantized, the frame can be built and added to the bitstream. First, the 32 bits of the header go into to the bitstream. The optional 16 bit CRC follows. For each section of the MPEG frame, information is listed in order from lower frequency subbands to higher frequency subbands. Next, the bit allocations are inserted. Four bits for each subband indicate the number of bits used to represent each subband sample. After this point in the bitstream only information about subbands that have bits allocated to them are present. Six bits are used to indicate which scalefactor was used when normalizing the subband samples. The final portion of the MPEG frame is the subbands samples. Starting with the lowest subband with allocated bits, the first subband sample is inserted into the bitstream, followed by the first subband sample of the next subband with allocated bits. This is repeated until all of the first samples for each subbands that have bits allocated to them are inserted into the bitstream. This process is repeated for the remaining 11 samples for all subbands.

5. PC Software Implementation

A PC MPEG encoder was written in C to examine the facets of the encoding process and test the functionality of the DSP algorithms. The code was written using the standard [4] as a guide. Clarification on the meaning of some of the points present in the standard came from the Working Draft of the MPEG/Audio Technical Report [6]. This contained source code for implementing an MPEG-1 layer 1 or 2 encoder and decoder. Using a high level language, such as C, allowed for quick changes to the encoder algorithm. All code was first tested for functionality in C before being ported to DSP assembly. The PC encoder was compiled as a 16 bit DOS application using Borland C++ version 3.1.

5.1. *Graphics*

The information present in an audio signal is best represented in a graphical format. Graphics routines were written to plot an array of values to the screen. The ability to visually see the results of the individual encoding steps aided in understanding the encoding process. The benefits of visual representation of the data were first realized when examining the figures presented in [7] as a tutorial on MPEG compression. The Borland BGI graphics libraries were selected for visual output for the encoding process because of the simplicity of API calls. The maximum resolution of 640x480 with 16 colors was sufficient to view multiple plots on the screen. It was necessary to

have the ability to plot both integer arrays and double precision floating-point arrays. Two identical plotting functions, `ploti` and `plotd`, were written to plot an array of integers and doubles, respectively. The two plotting functions were not written for speed, and a considerable speed improvement could be gained by rewriting them to use double buffering methods and by using a faster graphics library. The function prototypes for the plotting functions are shown below. The functions plot the data pointed to by the `data` parameter in the color defined by the `color` parameter. The parameter `numPoints` specifies how many data points of the input data are plotted. The parameter `offset` is added to each element of data before plotting. This shifts the plot up or down, adding the ability to perform multiple plots without overlap. The parameters `minval` and `maxval` are used for scaling and represent the bottom and top of the screen, respectively.

```
void ploti(int *data, int offset, int minval, int maxval,
           uint numPoints, int color);

void plotd(double *data, double offset, double minval, double maxval,
           uint numPoints, int color);
```

5.2. Bit File

In order to generate the MPEG bitstream, file I/O routines were needed to read and write bits to a file. The `bitfile` code simplified the writing of the bitstream. One to sixteen bits of an unsigned integer can be written to a file. The bits are written left to right and can be taken from the most significant bits or the least significant bits. The

bitfile code was also not written for performance. The bitstream was constructed using a shift and OR for each bit. In the low level I/O, bytes are transferred to or from the disk one at a time. Improvements in the bitstream routines would greatly increase the speed of the program. The function prototypes for the Bit File routines are shown below.

```
BitFile* OpenFile(char *fname, BF_mode mode);
void      CloseFile(BitFile *bFile);

uchar WriteBits(BitFile *bFile, uint data, uchar numBits, uchar dir);
uint  ReadBits(BitFile *bFile, uchar numBits, uchar dir);

uchar BF_EOF(BitFile *bFile);
uint  Read_LSB_MSB(BitFile *bFile);
void  ByteAlign(BitFile *bFile);
char* PrintBits(uint data, uchar numBits, uchar dir);
```

The first two functions, `OpenFile` and `CloseFile`, provide a way to open and close a bitstream file. `OpenFile` returns a pointer to a newly created `BitFile` structure and requires a filename and the mode of bitstream file. The `BitFile` structure contains the buffer used to translate from the bitstream to the byte based file I/O routines. All of the bitfile routines use this structure to access the bitstream file. A bitstream file can be opened in read or write mode. `ReadBits` and `WriteBits` read and write bits in a bitstream file. The `numBits` parameter indicates how many bits should be read from or written to the bitstream. When writing to the bitstream, the `dir` parameter indicates the side of the data to be written. A `dir` set to `BF_LEFT` writes the most significant bits of the data to the bitstream, while a `dir` set to `BF_RIGHT` write the least significant bits. When reading from the bitstream, `dir` indicates the destination side of the data. Setting `dir` to `BF_LEFT` is used to read or write fixed point fractional integer values, while `BF_RIGHT` is better suited for integer values.

Processing bitstream files requires four more helper functions. `BF_EOF` returns a non-zero value when the end of file has been reached when reading a bitstream. For Intel based 16 bit audio formats such as .PCM and .WAV files, the least significant byte (LSB) of the 16-bit audio sample is listed first in the bitstream. `Read_LSB_MSB` reads 16 bits from the bitstream and places the first 8 bits in the LSB and the second 8 bits in the Most Significant Byte (MSB). MPEG audio frames contain an integer number of bytes and always start on a byte aligned boundary in the bitstream. If the bitstream is not currently on a byte boundary, `ByteAlign` will flush the `BitFile` buffer and correctly align the bitstream to a byte boundary. The final function, `PrintBits`, is used for display purposes. It constructs a character string that represents the bits that would be read or written by the `BitFile` routines.

5.3. MPEG I/O

The aforementioned bitfile routines provide a means to write a variable number of bits to a bitstream. The encoder required variable bitfile routines because the MPEG frame contains fields that are shorter than eight bits, and fields that can change width every frame. Data structures to store all the information needed to construct an MPEG frame were used throughout the MPEG encoder. The MPEG I/O functions write and read an entire MPEG frame to and from disk. The encoder properly fills the MPEG frame data structure with the required data. Then the write MPEG frame

function calls the proper bitfile routines to write the MPEG frame data structure to disk in the correct MPEG format.

5.4. DSP Considerations

The hardware implementation of the encoder utilized a fixed point fractional DSP. The DSP chosen uses 24 bit numbers when performing calculations. Besides validating the encoder algorithms, the PC version of the encoder examined the effects of 24 bit fixed point calculations on the encoding process.

5.4.1. 24 Bit Mode, Scaling and FFT

Many times during the encoding process, a vector must be normalized to one. When writing the PC encoder it is easy to scale the largest value to one by searching for the largest value and dividing all elements in the vector by that largest value. The instruction set of the DSP contains instructions to perform fast normalization and shifting. Normalization on a DSP is done using shifting. The value is shifted left until one more shift would cause a sign change. This shifting multiplies the number by 2 until it is greater than one half. For a fractional number between -1 and 1, normalization produces a result greater than 0.5 or less than -0.5. The worst case produces a value of 0.5 instead of 1.0. When converting the result to the log power domain, the amount of error equals -3 dB. All frequency components share the same error. Since most of the psychoacoustic calculations are performed as the difference

of two SPLs, the error does not appear. The DSP has instructions to perform a divide iteration, but a divide can take up to 24 times more instruction cycles to execute. The PC program's 24-bit mode, in combination with graphic visualizations, allowed tests to determine if the inaccuracy present in the DSP normalization algorithm would have any effect on the encoding process.

The PC encoder revealed that scaling performed before the FFT calculations greatly increased the usability of the FFT results. Low power audio signals contain PCM samples with very small magnitudes. Smaller magnitudes only contain information in the least significant bits and the more significant bits are unused. The FFT process performs a large number of multiplies. Each multiply can easily lose these lower bits in the rounding and clipping. Normalizing the input signal before performing FFT calculations improves the signal and moves the information in the samples from the less significant bits to the greater significant bits. Normalization effectively multiplies all input samples by the same number. Since the FFT is a linear transform, this scales the results of the FFT by the same factor. The first step after performing the FFT is to normalize the results. The effects caused by the pre-FFT normalization do not have to be reversed before proceeding.

The DSP performs calculations with 24 fixed-point fractional number, while the PC performs calculations with double floating point values. This enables the PC to produce results that contain more precision than the DSP. One purpose of the PC program was to analyze the results using 24 bit coefficients and the effects of

rounding errors during 24 bit FFT calculations. The PC encoder simulated the DSP's precision by using an optional mode of calculations where all floating point multiplies were immediately quantized to 24 bits and then converted back to floating point. Graphics compared the results from both the floating-point and fixed-point calculations. The displays and output from the PC program showed that 24 bit fixed point calculations were sufficient for performing the encoding process.

5.4.2. Logarithms

One of the major drawbacks of using a fixed point DSP is that there is the lack of dynamic range for accurately calculating logarithms. The 24 bits available for calculations on the DSP has dynamic range of 69 dB. In the psychoacoustic portion of the MPEG encoding process, an FFT is performed on the last 512 audio samples. Psychoacoustic calculations are performed with the logarithm of the FFT. Since these logs are normalized to 96dB before calculations, the available dynamic range had to be translated to the larger dynamic range needed for the psychoacoustic model. The PC software served as a testbench for observing the effects of many different logarithm implementations.

In order to normalize a list of log values to 96 dB, first the largest value in the list is subtracting from 96dB. This offset is added to every element in the list. This moves all the values in the list up or down until the maximum value is equal to 96dB. In the case of the fixed point calculations used on the DSP, all values are between -1 and 1.

The logs of these values will always be less than zero. If all values are normalized to 1 before converting to the log power domain, then the values can quickly be converted to the log power domain by taking the log of the values, multiplying by 10 and then adding 96dB. Equation 14 shows the equations for converting from the power domain to the log domain and back. These equations produce values that are normalized to 96 dB if the power domain values are normalized to 1.

Equation 14: Converting from power domain to log domain

$$X_{\text{dB}} = 96 + 10 * \log(X)$$

$$X = \text{pow}(10, (X_{\text{dB}} - 96) / 10)$$

Using these equations without modification only produce logs in the range from about 26 to 96. Psychoacoustic comparisons require more than this limited range of values. Equation 15 shows a modified version of the equations that stretch the values to obtain a larger range. The parameter dB_scale controls the amount of scaling used for the conversions. In order for Equation 15 to have a negative result, dB_scale must be at least 1.39. The PC encoder tested values of dB_scale from 2 to 4. The scaling of the logs slightly distorts the calculations of the SMR, but does not effect the quality of the subband samples. Larger values of dB_scale cause peaks in the spectrum to be more easily marked as local maximums. If the dB_scale is too large, smaller FFT values will be converted to log values that are marked as inaudible because they are lower than the threshold in quiet.

Equation 15: Modified log domain calculations

$$X_{\text{dB}} = 96 + \text{dB\_scale} * 10 * \log(X)$$

$$X = \text{pow}(10, (X_{\text{dB}} - 96) / 10 / \text{dB\_scale})$$

6. DSP Software Implementation

In a real time encoder, all processing on one frame must take less time than the number of samples for that frame. Pipelining and ping-pong buffers allow calculations to be performed in parallel with the input of the samples and the output of the bitstream. Past work has been done with implementing a DSP based MPEG-1 encoder as in [8]. A single floating point TI DSP was used to implement an encoder and decoder. The analog input samples were encoded, decoded, and then output as an analog signal. During this process, the MPEG bitstream was never created or transmitted off chip. This thesis produces the MPEG bitstream and then sends it to a PC for immediate use.

6.1. DSP Selection

A DSP has many advantages over fabricating an ASIC. It is cheaper and reduces the amount of hardware design that is required. With a low-cost solution in mind, a Motorola DSP56302EVM was chosen for the MPEG encoder. The DSP56302EVM is an evaluation module that contains a DSP, codec, external memory, and a PC serial interface. The DSP56302 is a 24 bit fixed point digital signal processor. All numbers have a value between -1 and 1 and are represented as shown in Figure 13. This thesis required a DSP which could transmit a high volume of data to the PC. The DSP56k family of processors contain a byte wide host interface. Various vendors provide an

ISA card that can interface with the DSP to acquire data in real-time. Initial testing with the DSP56002EVM determined that the parallel port could be used to receive data from the encoder, eliminating the extra cost of the ISA host interface card and increasing the portability of the encoder.

Figure 13: 24-bit fixed point representation

Initially, Motorola's DSP56002EVM was chosen for the encoder. This DSP did not have enough on-chip memory or high enough MIPS rating. The DSP56002EVM was replaced by the DSP56302EVM. The DSP56302 is capable of running all DSP56002 code with minimal changes for enhanced peripherals. Table 11 summarizes some of the differences between the DSP56302 and DSP56002. Both DSPs operate at the same clock frequency. The instruction cycle of the '002 is two clock cycles while the instruction cycle of the '302 is one clock cycle. Therefore, the '302 has twice the MIPS rating for the same clock frequency.

Table 11: DSP56302 vs. DSP56302

	DSP56002	DSP56302
Program Memory	512	20480
X and Y Memory (each)	256	7168
Clock	66 MHz	66 MHz
MIPS	33	66
Serial Interface	SSI	2 ESSI
Other		56 bit Barrel Shifter

The internal RAM of the '002 does not satisfy the code and data requirements of the encoder. Both evaluation modules have 32k of external SRAM which is shared for program and data memory. However, using the off-chip memory causes a performance hit. Accessing the off chip RAM requires the pipeline of the DSP to be stalled for one instruction cycle on the '002 and two cycles on the '302. An opcode may need to access memory up to three times: once to fetch the opcode and one each for a parallel X or Y memory move. The previous and next instructions might also access the off chip memory, possibly requiring the three level deep '002 pipeline to stall twice. On the '002, running code from the off chip memory while accessing off chip data may require a pipeline delay of five instruction cycles. The larger on-chip memory of the '302 enable all code to be run from internal RAM so no off-chip fetches are required for data calculations or code execution.

The DSP communicates with the codec through a Synchronous Serial Interface (SSI). The SSI is interrupt driven. A frame signal indicates the start of a new frame. When communicating with the serial codec on the EVM, the DSP interrupts at a frequency four times faster than the sampling rate. The Enhanced SSI (ESSI) on the DSP56302 has an additional interrupt for the frame signal. This helps maintain the synchronization between the DSP and the device. The Interrupt Service Routine (ISR) for the ESSI can be a few instructions shorter. This technique saves a considerable amount of processing time since the ISR code runs 128,000 times per second. The DSP56302 has numerous architectural improvements over the DSP56002 such as: 56 bit barrel shifter, instruction cache, Position Independent

Code, 16-bit arithmetic support, 24 bit address registers, and 6 channel DMA support. Of these, the encoder only utilizes the 56 bit barrel shifter. The barrel shifter allows for multi-bit shifts in a single instruction. Some of the added instructions insert and extract a variable number of bits to or from a register. These additional instructions were useful when formatting the fields of the bitstream.

6.2. Codec Interface

The evaluation module provides a Crystal Semiconductor's CS4215 codec. The codec is a multifunction device used to provide the DSP with 16 bit stereo samples of an analog audio source. The codec is connected to the DSP through an Enhanced Synchronous Serial Interface (ESSI) running in network mode. There are four basic signals in a SSI consisting of the serial clock (sclk), frame sync (fsync), serial receive data (SRD) and serial transmit data (STD). The clock signal (sclk) runs at the frequency of the bit-rate of the data lines. Data is transmitted on the rising edge of sclk and data is received on the falling edge of sclk. When a SSI is in network mode, there is a fixed number of bits per frame. The frame sync signal (fsync) set for one sclk period indicates the start of a frame.

After reset, the codec defaults to slave mode and must be configured by the DSP. The DSP places the codec in control mode and programs the codec for proper operation. At this time, the sampling rate is set to 32kHz, the data format set to 16 bit linear stereo, and the frame size set to 64 bits.. The codec is configured to operate as the

master in data mode. As master, the codec drives the fsync and sclk signals. The codec produces 64 bits of data at the frequency equal to the sampling rate. This data is sent in one SSI network frame. Once the codec is properly configured, the DSP places the codec in data mode and the codec begins to transmit audio data. Figure 14 shows the data transmitted by the codec when it is transmitting stereo data. Each field in the figure represents 16 bits of information.

Left sample	Right sample	Output settings	Input settings

Figure 14: SSI data frame

The ESSI on the DSP is also configured in network mode. The word size is set to 16 bits. When the SSI shifts in 16 bits of data, the SSI generates a receive data interrupt and places the 16 bit value in the receive data register. Therefore, the DSP needs to service four ESSI receive data interrupts for each frame of data. The ESSI also produces a receive last- slot interrupt when the last slot of the SSI frame has occurred. This indicates that all data from the current frame has been serviced and is used to reset data pointers used when receiving the audio data. This interrupts the DSP one time each frame.

The encode process is data-driven. Samples are added to the FIFO when they arrive, subband samples are calculated every 32 samples and frames are created every 384 samples. Since the ESSI interrupts occur when the new samples arrive, it is useful to use the ESSI receive interrupts to process individual samples and to control the subband and frame calculations.

Each sample that is received generates a receive last slot (rxls) interrupt. The purpose of this interrupt is to synchronize the input buffers with the codec. The next SSI interrupt that occurs after the rxls is the receive (rx) interrupt. This interrupt corresponds to the time slot containing the data for the left sample. The second receive interrupt will correspond to the data for the right sample. The average of the left and right channels produces a single mono signal that is added to the FIFO. In the rxls interrupt, the samples are counted to know when to perform calculations. A radix 32 counter is used to know when 32 samples have been received. At this point, a bit is set to indicate the next rx interrupt should perform subband calculations after adding the next sample value into the FIFO. Every time the radix 32 counter resets to zero, a radix 12 counter is incremented. This counter keeps track of the state of the encoder. When the counter is equal to 0 or 1 the encoder is performing pre-frame processing. Otherwise the encoder is performing psychoacoustic calculations and filling the bitstream buffer.

6.3. Pipeline

The DSP encoder works in a pipelined manner. There are three levels to the pipeline. The subbands samples are calculated in the first stage. The frame processing and bitstream formatting is done in the second stage, and the bitstream is transmitted in the final stage. This requires the use of ping-pong buffers. The first stage fills a subband sample buffer. The second stage reads a subband sample buffer and writes a bitstream buffer. The final stage reads a bitstream buffer. In order for all the stages

to have data to process and a place to store the results, two subband sample buffers and two bitstream buffers are required.

There are 384 samples for each MPEG frame. Every 32 samples, one subband sample is calculated for each of the 32 subbands. These results are stored in one of the subband sample buffers. After 384 samples occur, there are 12 samples stored in each of the subbands and the subband buffer is full. Then the subband sample buffers are swapped. The buffer that was used to store the results from subband calculations in the previous frame is used to construct the next bitstream buffer. Two pointers for each set of buffers represent which buffer is currently being written (subsamp_stor / bstream_stor) and read (subsamp_read / bstream_read).

	0	1	2	3	4	5	6	7	8	9	A	B	0	1	2	3	4	5	6	7	8	9	A	B	0	1	2	3	4	5	6	7	8	9	A	B
sb	x	x	x	x	x	x	x	x	x	x	x	x	y	y	y	y	y	y	y	y	y	y	y	z	z	z	z	z	z	z	z	z	z	z	z	z
sf	w	w											x	x											y	y										
fp			w	w	w										x	x	x	x									y	y								
sh	v	v	v	v	v	w	w	w	w	w	w	w	w	w	w	w	w	w	x	x	x	x	x	x	x	x	x	x	y	y	y	y	y	y	y	

Figure 15: Frame Processing

Figure 15 shows an overview of the events occurring within the encoder. The top row represents the current value of the radix 12 counter incremented in the rxls interrupt. The letters (wxyz) represent the frames currently being processed. The labels along the left side of the rows represent the calculation being performed on the current frames. Subband and scalefactor calculations are represented by 'sb' and 'sf', respectively. Psychoacoustic calculations and frame construction occur in the 'fp'

stage. In the final stage, 'sh', the frame is sent to the host as described in chapter 7. Sections 'sb' and 'sh' are executed within the ESSI receive interrupts, and sections 'sf' and 'fp' are executed in the foreground.

While in the subband calculation phase (sb), subband calculations are performed as shown in section 4.3 and results are placed in the subband buffer pointed to by the subsamp_stor pointer. The switching of the subband buffers occurs when the radix 12 counter resets to zero and subband sample calculations for the next frame begins. In the foreground, scalefactor determination, SMR calculation, and frame construction use the subband buffer pointed to by subsamp_read and store the resulting bitstream in the buffer pointed to by bstream_stor. The time it takes for the foreground to finish constructing the next frame is indeterminate. This is indicated on the figure as a variable length 'fp' stage. As soon as the foreground is done building a new bitstream buffer, the bitstream pointers are swapped so the newly constructed frame may be transmitted. The send_host (sh) routine transmits the bitstream buffer pointed to by bstream_read.

6.4. *Foreground Processing*

There are two phases to the foreground processing. Both phases are synchronized to events that occur in the background. The scalefactor calculations require the subband read buffer to be filled. This requirement is met when the subband buffers are swapped, which occurs when the radix 12 counter resets to zero. To perform the 512

point FFT needed for the psychoacoustic calculations, a delay of 64 samples is needed to center the 384 samples representing the current frame. Since each increment of the radix 12 counter represents 32 samples, the foreground must wait until the counter has reached 2 before it can begin calculations on the second phase.

During this 64 sample delay, the subband samples are searched to find the scalefactor to use for normalization. Subband samples can also be normalized at this point. If normalization is done at this time, then it must be done for all subbands and at maximum resolution (15 bits). Normalization can be delayed until after bits have been determined for each subband. At this time, normalization only has to be performed for the subbands that have bits allocated to them, and the division only has to be performed for the number of bits allocated to the subband. By delaying the normalization, the number of divides is reduced. It therefore takes less CPU time to calculate the normalized subband samples. However, this CPU time is taken away from the frame processing. By performing the quantization at full precision before the frame processing, more CPU time is used but the DSP would otherwise be sitting idle at that time. During the pre-frame processing time, any calculations performed increase the amount of CPU time that is available for frame processing.

6.5. *Simulation*

Motorola provides a free assembler, compiler, and simulator for their DSP products. Sections of the code were tested by filling a buffer with known data and then running

a simulation of the code. The resulting output buffers were then examined with the desired output. The difficulty lies in figuring out what the desired output should be. The calculations were on large data sets that required many operations, so tracing through the steps of the calculations by hand would be impractical. Matlab performed identical calculations on the same data as the simulator and generated the proper output. Matlab provides a means to represent highly mathematical algorithms in a simple scripting language. Matlab scripts were written for both the subband calculations and the subband sample normalization.

Matlab is ideal for testing the mathematical portion of the encode process, but it does not contain any functions for testing the bitstream formatting. The MPEG I/O routines were tested while writing the PC version of the encoder, and are known to be correct. The best way to test the bitstream formatting portion of the DSP encoder is to capture the state of the DSP before formatting the bitstream and output the data using the MPEG I/O routines. This was done through the use of the simulator. A simulation was performed to fill the bitstream buffer. Dummy data was used to fill the data structures on the simulator, and then the contents of the simulator memory were dumped to disk. A C program was written to parse the memory dump and to output the correctly formatted frame to disk. The contents of this file were then compared with the results of the simulator.

6.6. Port B Testing

Motorola provides a hardware debugger with the EVM board. The EVM connects to the PC through a serial port. The full-featured debugger includes such features as breakpoints, watches, disassembly, and single step execution. Memory dumps can be viewed in plot form. This is very useful for visualizing the data stored in memory. Another useful feature was the window update. The debugger updates all windows at a specified interval when the update windows feature is enabled. Used together, these last two features are useful for viewing rapidly changing data. A memory dump of the input FIFO can be displayed as a graphics plot. By using the update mode, the contents of the memory can be viewed while the DSP is running. This method of viewing the memory is, however, obstructive to the normal operation of the DSP. The needed data transfers interrupt the processing of the DSP. This can sometimes be heard as interruptions in the sound while the transfer is occurring.

A method was needed to examine the state of the DSP without interrupting normal operation of the encoder. The only available parallel I/O on the DSP is Port B. This is used in the encoder as a parallel interface to the PC. A quick way to allow for real-time non-obtrusive debugging is to disable the DSP to PC transfer and output status information to Port B. A logic analyzer captures the output from Port B, allowing the state and timing of the internal DSP operation to be examined. Example output from the logic analyzer is show below in Figure 16.

Figure 16: Port B testing

Since the current encoder skips the psychoacoustic calculations, it is necessary to know how many free cycles are available for the extra calculations needed for the psychoacoustic model. While the foreground waits for the slot0 bit to be set, a counter is incremented. By multiplying the counter by the number of instruction cycles it takes to execute the busy wait loop, the number of unused processor cycles can be calculated. The value of this counter is saved to memory so the debugger can examine it and see how many times the loop was executed. The foreground performs the same action while waiting for the slot2 bit to be set.

6.7. Bit Allocation

With the psychoacoustic model removed from the DSP encoder, the bit allocation stage does not receive the SMR that the psychoacoustic model normally produces. The bit allocation stage needs the SMR to calculate the MNR for all the subbands. Bits are allocated to the subbands to minimize the MNR. Initially the bit allocation portion was also removed and a fixed bit allocation was assigned to all the subbands.

The number of bits for each subband was predetermined and placed in the source file. The bits were distributed so that the lower frequencies were allocated the largest number of bits. Typical audio signals contain more energy in the lower frequencies than higher frequencies, and this bit distribution attempts to provide a higher SNR to the lower frequencies than the higher frequencies. This method of bit allocation is fixed and does not adapt to the energy present in the signal.

A method was needed to generate a SMR that represents the energy present in the signal without performing the FFT calculations. The SMR is calculated by subtracting the mask level from the signal level. In psychoacoustic calculations, the FFT is used to determine the masking levels for each subband. With the FFT removed, an expression for the masking level for each subband needs to be determined. The solution is to set the masking level for all the subbands equal. The masking level can then be removed from the SMR calculations because the SMR is now independent of the masking level.

Equation 16: Approximation of SMR

$$SMR(sb) = 20 * \log_{10}(scalefactor(sb) * 32768)$$

The signal power is calculated using the expression shown in Equation 7. The signal power is the maximum of the scalefactor energy and the spectral energy in the subband. In place of the FFT being calculated, the power of the scalefactor energy can be used as an approximation of the signal power. With equal masking for all the

subbands, the SMR can be approximated by Equation 16. By using this expression for the SMR, the bit allocation distributes bits dynamically to subbands based only on the scalefactor energy present in the subbands.

7. DSP - PC Interface

One of the major goals for this thesis is to provide a means to encode an MPEG signal and transmit it to a PC in real time. It is also desired to transmit to the PC using a minimum amount of additional PC hardware. By reducing the amount of PC hardware needed, the portability of the encoder increases. The bit-rates for MPEG-1 layer 1 range from 32kbps to 448kbps. The target bit-rate for the encoder was between 32kbps and 256kbps. The serial port of the PC does not have sufficient bandwidth to receive the amount of data transmitted by the encoder. The parallel port was used in the bi-directional mode to perform the transfers. The eight-bit data bus was used for the bitstream data. Handshaking is done through the status and control lines of the printer port.

7.1. Protocol

Speed was the highest priority of the DSP to PC transfer. A low overhead protocol was needed to transmit the data and provide a means of synchronization. Figure 17 shows the waveforms to perform a full handshaking read operation. The falling edge of the STR signal indicates that the receiver is ready for new data. The transmitter outputs its data and lowers the ACK line to indicate that the data is valid. Once the receiver has obtained the data, it raises its STR line to indicate that it received the data. When the transmitter detects the rising edge of the STR line, it stops driving the

data bus and raises the ACK line to indicate that it is ready for another operation. This is the normal protocol used when transferring data. It requires two transitions on both of the control lines.

Figure 17: Full Handshake Read Operation

Figure 18: Double Edge Acknowledge

The devised protocol for the transmission of the bitstream, shown in Figure 18, uses both edges of the control lines for acknowledgement. Because this protocol does not have a "ready" state like the above protocol does, a third control signal (INIT) is required to synchronize both the ACK and the STR lines to the high state. This protocol has the transmitter in control. From the initialization state, the transmitter sets the data line and then toggles the state of the ACK line. Once the receiver sees that the ACK line has changed state, it reads the data and toggles the STR line to acknowledge receiving the data. Either the transmitter or the receiver can have control of the INIT line. The device in control of the INIT line is responsible for determining when the transmission falls out of synchronization. This can be done as a timeout on a transmit or a receive. Since the protocol is driven by the transmitter, if

a timeout is performed by the receiver then a fourth control line (BUSY) must be added. This signal is controlled by the transmitter, and is high whenever the transmitter is busy or is not transmitting. This is useful in the case of the encoder. The transmission speed is fast enough to allow the DSP to transmit the data faster than it can fill up the next buffer. The BUSY line is set to prevent the receiver from performing a synchronization, because that may cause a frame of data to be lost.

7.2. Hardware Needed

At the time of the proposal, it was not known if the PC parallel port was capable of receiving the amount of data being produced by the DSP encoder. An ISA host interface card is available that connects to the host interface of the DSP. If the parallel port could not receive the data fast enough, the ISA card would be needed. Once the EVM board was received, the first priority was to see if the extra hardware was necessary for the project's operation. Initial testing concluded no extra hardware, other than the DSP EVM board and a bi-directional parallel port, was needed for the project. The I/O pins for Port B of the DSP do not have the driving capabilities to transmit across the parallel cable. A set of line drivers (74LS244 + 74LS245) provides the necessary driving capability. In the PC's parallel port hardware, some of the status pins are inverted. This required the addition of some inverters (74LS14) to change the sense of the control signal, so a read of Port B on the DSP would correspond to the same bit pattern written to the parallel port on the PC. Both the inverters and the line drivers have an amount of hysteresis useful in reducing any

noise that might be present in the data. Two inverters on the ACK signal guarantee the settlement of the data before the parallel port detects its transition. This allowed for the DSP to change the state of the ACK signal at the same time as changing the data and removed any software delays necessary to settle the data.

7.3. *PC Implementation*

The PC program works as a gateway for the DSP to write the MPEG bitstream to disk. The program was written in C. The Inportb and outportb functions were used to access the parallel port through its data, status and control registers. All reads of the registers are done twice to make sure that the data on the port had settled. The program continuously polls the parallel port for new data. Once new data is available, it adds the data to a buffer in memory. Periodically, the buffer is written to disk. The PC program is in control of the INIT line. A simple method of detecting a synchronization error was implemented with counters. The INIT signal is set high when ever a timeout occurs. When waiting for data from the DSP, if the BUSY line is high, the timeout mechanism is disabled.

The state of the PC receiver is based on the previous ACK signal read from the status register. Table 12 shows the state table used by the PC program. The DSP toggles the value of ACK whenever new data is available. The PC program saves the contents of the data register whenever it detects a change of the ACK line, and sends back a response by setting the STR signal to the new value of ACK.

Table 12: State Table for PC Receiver

<i>State</i>	<i>Action</i>
Timeout Occurred on read	INIT='1' STR='1'
Previous ACK high	If ACK = '0' Add data to buffer STR = '0'
Previous ACK low	If ACK = '1' Add data to buffer STR = '1'

7.4. DSP Implementation

The DSP transfers 8 bits of the bitstream for every transfer to the PC. In order to achieve a bit-rate of 512kbps, the DSP must transmit 8 bits 64k times per second. A timer could be used on the DSP to interrupt the processor 64k times a second and execute the code for bitstream transfer. This would require extra cycles to save the registers and the added complexity of another interrupt. The SSI interrupts used for communications with the Codec interrupt the DSP 128k times per second. By executing the DSP to PC transfer code at the end of the SSI interrupt, the code would execute 128k times per second. This reduces the number of context switching necessary, and executes enough times to provide sufficient throughput.

Since the code is executed 128k times per second, it should be as fast as possible. Every time the code is reduced by one instruction, a possible savings of 128,000 instructions per second exists. The DSP transmitter is a state machine based on the current value of ACK. The INIT signal synchronizes the transmitter with the

receiver, and causes the DSP to initialize the ACK to '1'. The transmitter places the first byte on the data lines and lowers the ACK signal to indicate the data is valid. After that point, the transmitter checks until the STR line is equal to the new state of ACK. The transmitter then knows the receiver has acknowledged the data. The transmitter can then transmit the next byte of the bitstream and toggle the state of the ACK. Table 13 and Figure 19 show the state table and the state diagram for the DSP transmitter.

Table 13: State Table for DSP Transmitter

<i>State</i>	<i>Action</i>
INIT high	Data = '0' ACK = '1'
ACK high	If STR = '1' Data = next data ACK = '0'
ACK low	If STR = '0' Data = next data ACK = '1'

Figure 19: State Machine for DSP Transmitter

7.5. *Transfer Rates*

The highest priority when developing the DSP-PC transfer was to see if the parallel port could receive the bit-rates needed for the encoder. To test the maximum transfer rate, a circular buffer on the DSP was filled with 256 values from 0x00 to 0xFF. The flow control portion of the DSP transmit code on the DSP was disabled. This caused the DSP to transmit data whenever the PC was ready for more. Since the data was known, the received data was analyzed to see if any data was lost or corrupted. Testing showed that when transferring approximately three to four megabytes of information, no data loss was encountered and the data rate obtained was approximately 500 kbps. This throughput rate is high enough for any of the bit-rates defined in the MPEG-1 standard.

8. Conclusion

The increasing amount of multimedia content demands the need for real-time processing of video and audio signals. MPEG audio provides high compression of an audio signal with little removal of audio quality. The general-purpose processors found in today's PCs are not suited for real-time encoding of an audio source. Producing a custom ASIC to perform the encoding requires extensive design and higher cost for small production volumes. The DSP is quickly becoming the tool used to process data in real-time. In addition to being optimized to process data in real-time, development on a DSP can be done with assembly or high level languages. Both the DSP hardware and the development tools are inexpensive.

The goal of this thesis was to design a low cost real-time MPEG audio encoder that transmits the MPEG bitstream to a PC. An additional concern was to minimize the amount of additional hardware needed to interface with the PC. Requiring the interface to use additional PC hardware makes the encoder less portable. A standard bi-directional parallel port was the only PC hardware requirement to interface with the DSP encoder.

The current implementation of the encoder does not perform any psychoacoustic calculations for determining the SMR. This has an audible effect on the resulting bitstream. More bits are allocated to the subbands that have the largest sample values.

This heuristic does not take into consideration the masking effects present in the signal.

The DSP hardware implementation was able to encode MPEG bitstreams between 32kbps and 320kbps. The DSP-PC transfer rate limited the maximum bit-rate to be 320kbps. Higher bit-rates could be encoded by the DSP, but the host transfer would have to be improved. Figure 20 shows the DSP56302 assembly used for the main foreground loop. Lines 3 and 11 are where the foreground synchronizes with the background. A counter is used to count the number of idle cycles available. It takes 5 clock cycles to execute one iteration of the wait loop. An MPEG frame encoded at a sampling rate of 32kHz is 12 milliseconds long. With the DSP running at 66MHz, there is a total of 792,000 clock cycles in each frame period. Since the execution of the code on lines 6 and 7 is independent of the bit-rate, the value of `wait2_cnt` is constant and is equivalent to approximately 76800 clock cycles. This corresponds to 9.7 percent of the time for one frame.

```

1      loop      clr      a
2      wait_0    inc      a                                (1)
3                      jclr  #FG_slot0_B,x:FIFO_status,wait_0    (4)
4                      move  a,x:wait0_cnt
5
6                      jsr    find_scalefactor_indices
7                      jsr    normalize_samples_slot0
8
9      wait_2    clr      a
10     wait_2    inc      a
11                      jclr  #FG_slot2_B,x:FIFO_status,wait_2
12                      move  a,x:wait2_cnt
13
14                      jsr    psychoacoustic_calculations
15                      jsr    bit_allocation
16                      jsr    write_frame
17
18                      jmp    loop

```

Figure 20: DSP foreground loop

The execution times of lines 14 through 16 are affected by the target bit-rate of the encoder. This causes the values of wait0_cnt to change with different bit-rates. Table 14 shows the number of clock cycles when the foreground process is waiting for the start of the next frame. These numbers are needed for any future additions to the encoder. Any code added to the encoder, such as the FFT needed for psychoacoustic calculations, will run during this idle time. With a minimum of 270,000 clock cycles, there is enough processing time available to add the calculations needed for a full MPEG-1 layer 1 audio encoder. The results from using the idle counters indicate that the DSP is idle from approximately 44% to 67% of the time.

Table 14: Slot 2 Idle times as a function of bit-rate

Bit-rate	Cycles	% idle
32	458,240	57.9%
64	440,320	55.6%
96	420,480	53.1%
128	400,000	50.5%
160	378,880	47.8%
192	355,840	44.9%
224	334,080	42.2%
256	313,600	39.6%
288	291,840	36.8%
320	272,000	34.3%

8.1. Difficulties Encountered

There were a number of small difficulties encountered in the implementation of the encoder. Initial understanding of the encoding process made the psychoacoustic portion appear more difficult than it actually was. A defective codec on the initial DSP56302EVM board produced many untraceable errors before its final malfunction requiring its replacement. The limited dynamic range of the fixed-point digital signal processor raised many issues about the effects of scaling when performing logarithms on the results of psychoacoustic algorithms. The larger dynamic range of a floating point DSP would have allowed for unscaled logarithms to be performed.

8.2. Recommendations for Future Work

This implementation of an MPEG audio encoder is the minimum needed to produce a valid MPEG-1 layer 1 bitstream and to transmit it to a PC. The first step for any future work is the completion of the psychoacoustic model portion of the encoder. This would greatly improve the sound quality of the resulting bitstream. Future work may include the implementation of a layer 2 or layer 3 encoder, stereo encoding, or the ability to encode at different sampling rates.

Speed was chosen over data integrity when designing the DSP-PC transfer. The current implementation of the DSP-to-PC transfer leaves approximately four I/O pins available for future use. These available data lines could be used to transfer Error Correcting Codes (ECC) along with the data that could be used by the PC to correct any transfer errors that may occur in the encode process. Each frame of MPEG audio is approximately 12 ms of audio data. When a corrupt frame is played, an audible "chirp" can sometimes be heard. The PC receiver could be expanded to fix corrupt frames by removing them, repeating the previous frame, or interpolating a frame from the surrounding frames.

This thesis implements a real-time MPEG-1 layer 1 audio encoder on a fixed point DSP. The ability to transmit the MPEG bitstream to a PC with minimal hardware shows the feasibility of this implementation as a cost-effective consumer product capable of encoding and audio source in real-time.

9. References

- [1] Davis Pan, "Digital Audio Compression", *Digital Technical Journal*, Vol. 5, No. 2, Spring 1993.
- [2] John W. Woods, editor, Subband Image Coding, Kluwer Academic Publishers, Boston, 1991.
- [3] E. Zwicker and H. Fastl, Psychoacoustics: Facts and Models, Springer-Verlag, Berlin Heidelberg, 1990.
- [4] ISO/IEC International Standard 11172-3 "Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s - Part 3:Audio", Switzerland, Aug 1993.
- [5] Davis Pan, "A Tutorial on MPEG/Audio Compression", *IEEE Multimedia*, Vol. 2, No. 2, Summer 1995, pp. 60-74.
- [6] Davis Pan, Working Draft of MPEG/Audio Technical Report, ISO/IEC JTC1/SC29/WG 11, Nov. 1992.
- [7] Seymour Shilen, "Guide to MPEG-1 Audio Standard", *IEEE Transactions on Broadcasting*, Vol. 40, No. 4, Dec 1994, pp. 206-218.
- [8] C. D. Murphy and K. Anandakumar, "Real-Time MPEG-1 Audio Coding and Decoding on a DSP Chip", *IEEE transactions on Consumer Electronics*, Vol. 43, No. 1, February 1997, pp. 40-47.