HEWLETT PACKARD

A Fast Integer Implementation of an MPEG-I Audio Decoder

Mathieu C. Hans*, Vasudev Bhaskaran
Computer Research Laboratory
HPL-96-03
January, 1996

scaled Chen DCT,
fixed point
representation,
MPEG

We have developed a fast integer implementation of an MPEG-I audio decoder. We obtained a fast implementation using a combination of algorithmic enhancements and by modifying some of the filtering functions.

Firstly, we incorporated Chen's fast DCT algorithm to implement the maxtrixing operation in the subband filtering defined by the MPEG standard. Instead of using the Chen's DCT method as-is, we used a scaled DCT wherein multiples needed in the final stage of the DCT computations are absorbed in the stage after the DCT. This reduces the multiply count for the DCT from 116 to 84.

Secondly, we modified the filtering functions. Based on subjective tests, it was found that a considerable number of subband coefficients can be set to zero without degrading the reconstructed audio quality. Setting the subband coefficients to zero reduced the overall multiply count. Furthermore, subjective tests also indicated that setting some of the filter coefficients of the synthesis window to zero did not reduce overall audio quality. Zeroing out the filter coefficients also reduced the multiply and add count.

The combined effect of algorithmic enhancements and truncation of subband coefficients and filter coefficients yielded a 13% speed improvement over a naive integer implementation.

Internal Accession Date Only

*Georgia Institute of Technology, Atlanta, Georgia. Mathieu Hans worked as a SEED for HP Laboratories, 1995

1 Introduction

The ISO/MPEG-I audio coding standard [JSW93] can provide high-quality audio using reduced bit rates of 128 Kbit/s per audio channel instead of 706 Kbit/s required in uncompressed compact disc audio quality. It supports sampling rates of 32, 44.1, and 48 KHz, and bit rates per monophonic channel between 32 and 192 Kbit/s, or per stereophonic channel between 128 and 384 Kbit/s. Possible applications include the coding of audio for Digital Audio Broadcasting and for the storage of synchronized video-and-audio sequences on CD-ROM.

The MPEG audio standard mandates the syntax of the coded bit stream, defines the decoding process, and provides compliance tests. This guarantees that any fully compliant MPEG audio decoder will be able to decode any MPEG audio bit-stream with a predictable result.

In this paper, we study the problem of designing a fast and efficient integer-based MPEG-I audio decoder. The primary motivation for seeking an integer implementation is that floating point multiply operations are expensive on low-cost RISC processors, e.g. PA7100-LC, MIPS R4000. This paper is organized as follows. In Section 2, we provide a brief overview of the codec and describe the synthesis subband filter bank. In Section 3, we describe our integer implementation. Three approaches for reducing the total number of multiply-accumulates including the notion of using a scaled DCT are described in Section 4. Finally, a conclusion is given in Section 5.

2 The MPEG-I Audio Codec

The MPEG audio standard achieves compression by exploiting the spectral and temporal masking effects of the ear. It uses subband coding and a psychoacoustic model to eliminate information that is irrelevant from a human sound-perception viewpoint.

Figure 1 shows the block diagram of the standard MPEG audio encoder and decoder. The input PCM samples pass simultaneously through an analysis subband filter bank and a psychoacoustic model. The analysis subband filter bank divides the input into multiple subbands of frequency, and the psychoacoustic model determines the ratio of the signal energy to the masking threshold for each subband. This ratio is used to define the number of code bits that should be allocated for the quantization of each subband coefficients. Once these coefficients are quantized, they are formatted into a coded bit-stream and sent to the decoder.

The decoder parses this bit-stream and restores the quantized subband values, which are used to reconstruct the PCM samples using a synthesis filter bank. We describe this filter bank in the following paragraphs. Detailed overviews of the standard can be found in [Pan95, Nol93, Shl94, BS94].

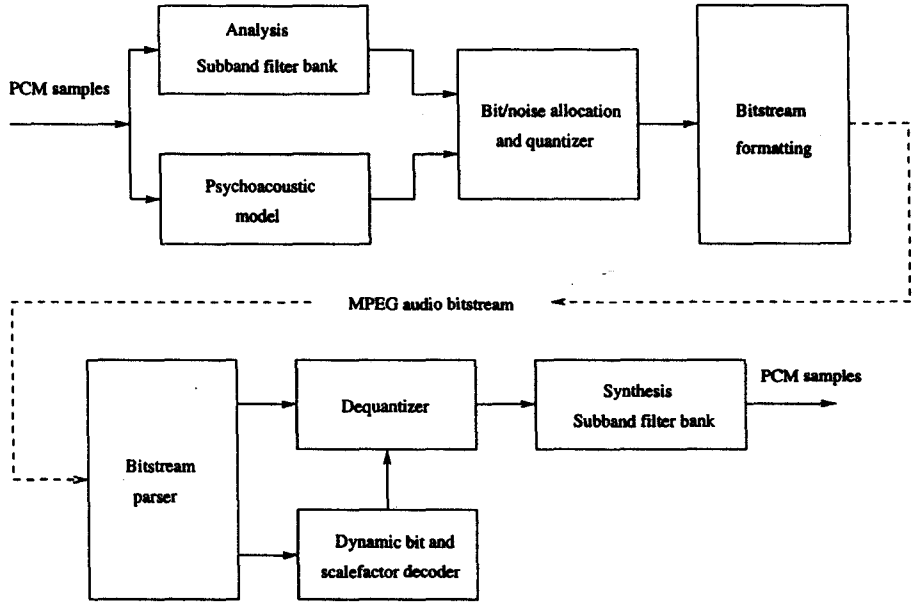

Figure 1: Block structure of the Layer II MPEG-I Audio codec

2.1 The Synthesis Subband Filter Bank

Table 1 displays the profiling output of a software implementation for an MPEG-I decoder[1]. This Table indicates that most of the execution time is spent in the synthesis subband filter bank shown in Figure 1. Therefore, any improvements in the design of this block should yield a faster decoder.

Block	%
Synthesis Subband filter bank	71%
Dequantizer	13%
Bitstream parser	9%
Dynamic bit and scalefactor decoder	1%
Miscellaneous (saving results, etc.)	6%

Table 1: Approximate percentage of the total time of the software MPEG-I decoder accounted for each block.

Figure 2 is a block diagram of the synthesis subband filter bank. In this diagram, there are two blocks which are computationally intensive: the Matrixing, and the Windowing block. Note that the Shifting block can be implemented using a circular buffer.

[1] We studied the MPEG-I public source code decoder written by Fraunhofer Institute and available on the Internet on the FTP site ftp.fhg.de in the directory pub/iis/layer3/public_c

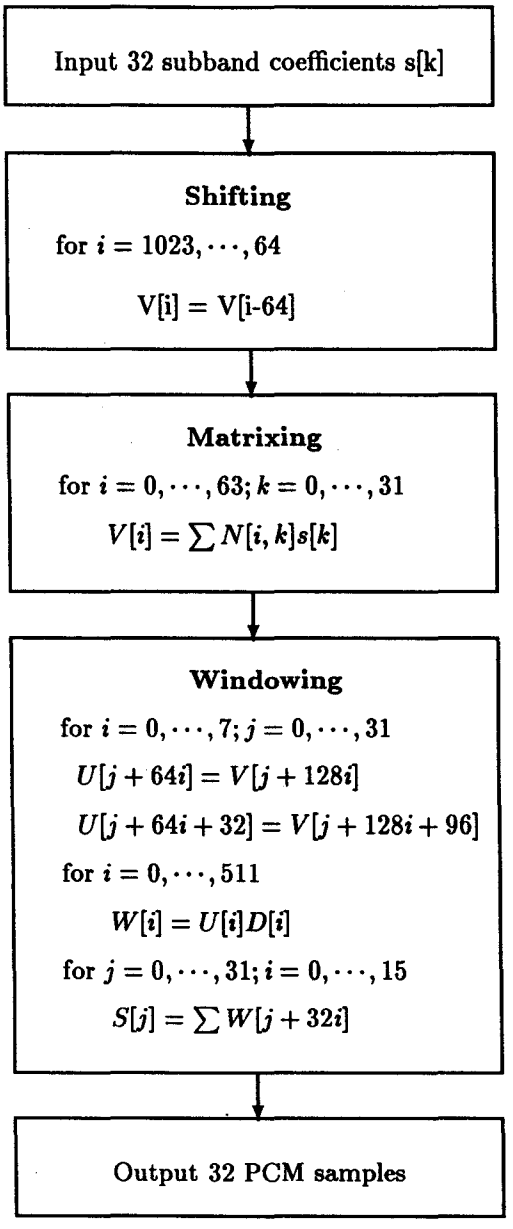

Figure 2: Synthesis subband filter bank block diagram

2.1.1 Matrixing Block

From Figure 2, the matrixing operation is defined as

$$V(i) = \sum_{k=0}^{k=31} N[i,k]s(k) = \sum_{k=0}^{k=31} \cos[\frac{\pi}{64}(2k+1)(i+16)]s(k), i = 0,1,\cdots,63 \qquad (1)$$

where $s(k), k = 0,\cdots,31$ denote the input subband coefficients. Using this formula to evaluate the $V(i)$ requires $32 \cdot 64 = 2,048$ multiply-accumulates. [Kon94] proved that a more efficient way to compute these values is to use a 32-point DCT (Discrete Cosine Transform). When fast 32-point DCTs are implemented, a dramatic reduction in the number of additions and multiplications can be obtained, as shown in Table 2.

Note that the standard DCT in Table 2 refers to the direct implementation of the Discrete Cosine Transform definition, also called the forward DCT-II in [RY90, pp.15].

Algorithm	Additions	Multiplications
Direct matrixing	1,984	2,048
Standard DCT	992	1,024
Chen's DCT [CSF77]	194	116
Lee's DCT [Lee84]	209	80

Table 2: Number of additions and multiplications for the Matrixing block for different algorithms.

2.1.2 Windowing Block

From Figure 2, the windowing operation is defined as

$$S[j] = \sum_{i=0}^{15} W[j+32i] = \sum_{i=0}^{15} U[j+32i]D[j+32i], j = 0,1,\cdots,31, \qquad (2)$$

where $D[i], i = 0,\cdots,511$ are window coefficients defined by the MPEG standard with 10 significant digits, and U is defined as

$$\begin{aligned} U[j+64i] &= V[j+128i] \\ U[j+64i+32] &= V[j+128i+96] \\ & i = 0,1,\cdots,7; j = 0,1,\cdots,31. \end{aligned} \qquad (3)$$

The windowing operation requires $32 \cdot 16 = 512$ multiply-accumulate operations. A plot of the coefficients $D[i]$ is given in Figure 3.

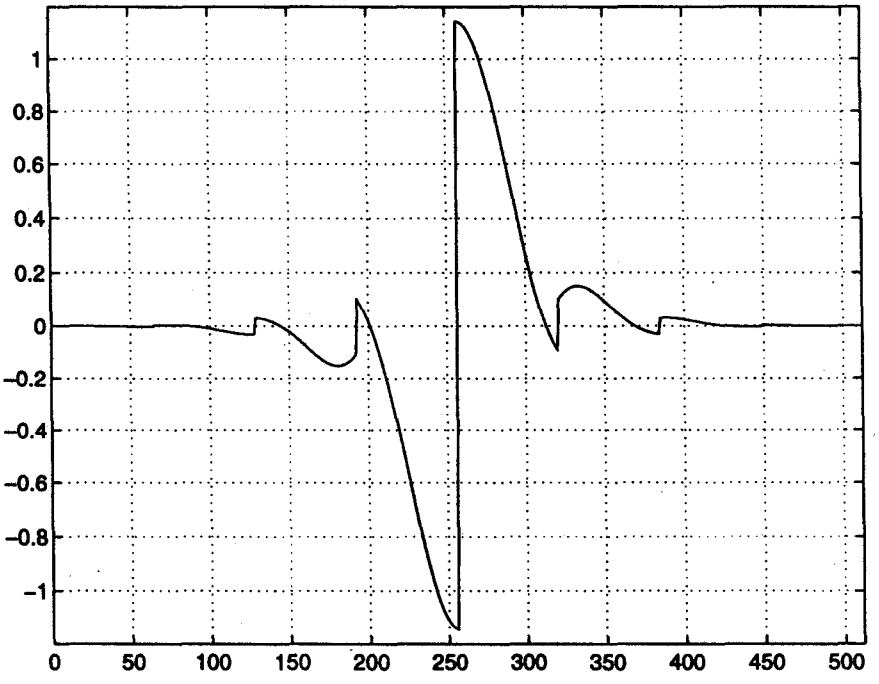

Figure 3: Coefficients $D[i]$ of the synthesis window.

3 Integer Implementation

For the integer implementation design, we used the fixed point representation. Before exposing our implementation, let us recall the definition of such a representation.

3.1 Fixed point representation

In theory[2], any real number x whose magnitude is less than or equal to X_m can be represented with infinite precision in two's-complement form as

$$x = X_m(-b_0 + \sum_{i=1}^{\infty} b_i 2^{-i}) \tag{4}$$

where X_m is an arbitrary scale factor and the b_i are either 0 or 1, and the quantity b_0 is the sign bit, such that: if $b_0 = 0$, then $0 \leq x \leq X_m$, and if $b_0 = 1$, then $-X_m \leq x < 0$. In

[2]For a complete overview on numerical effects of finite-precision in Digital Signal Processing, refer to [OS89, pp. 328–373].

practice, only a finite number of bits can be used to represent a real number. When $(B+1)$ bits are available, the above representation must be modified to

$$\hat{x} = X_m(-b_0 + \sum_{i=1}^{B} b_i 2^{-i}) \tag{5}$$

This shows that $\hat{x}$ is a quantized representation of x, and that the quantization step is $\delta = X_m 2^{-B}$.

It also suggests that if a number is larger than X_m (overflow), a policy needs to be defined to determine the quantized result. With two's complement numbers (which we used in our software implementations), two possibilities are given: clipping or no clipping. In a software implementation, clipping adds cycle overhead, because conditional statements, such as [`if (x > MAX) clip(x);`] need to be included in the data flow for every addition. On the other hand, no clipping does not introduce cycle overhead, and allows us to use the following nice property: if several two's-complement numbers whose sum would not overflow are added, then the result of two's-complement accumulation of these numbers is correct even though intermediate sums might overflow [OS89, pp. 332].

In our preferred integer implementation of the synthesis subband filter block, B was set to 15 to allow the results of multiplications to hold in 32 bit words, and the no clipping method was used except at the output of this block, where saturation was used if needed.

3.2 Our integer implementation

To understand how we designed our integer implementation, let us follow the computational path taken by the data.

We already noted that the block diagram from Figure 2 can be divided into two computational blocks. Each block implements a linear filtering operation. The Matrixing block filters the 32 subband input data, and the Windowing block filters the vector U of length 512.

Because the filters involved have fixed coefficients, we decided to compute their fixed-point representation only once. The value of the scale factor X_m of the fixed point representation defined in equation (5) depended on the filter used.

For the Matrixing block, 1.0 was chosen when the standard DCT, and the Chen's fast DCT were used. In the case of Lee's fast DCT, no scale factor was found. Indeed, for this algorithm, a large scale factor is required because it involves coefficients with magnitude as large as 10.19. Unfortunately, this large scale factor introduced a large quantization step for the filtered data, which added a distinct quantization noise in the audio output. Hence, even though in Table 2 we note that Lee's DCT requires fewer multiplies than Chen's DCT,

the latter is more amenable to an integer implementation. Moreover, we will show in the following section that a scaled version of Chen's DCT can be used, which contains fewer operations than Lee's DCT.

For the Windowing block, the scale factor was set to 2.0 to accommodate all the coefficients.

As for the data, we defined the scale factor to 1.0 after plotting the histograms of subband coefficient values of our example files and found that *all* values were inside the $[-1.0, 1.0]$ segment[3].

Using these scale factors, no overflows were detected during the flow of computation. Therefore, we decided that our integer software implementation would only check for overflows at the end of the synthesis subband filter block and clip if necessary.

3.3 Quality measures

To measure the quality of the implementations, two methods were used. First, we subjectively tested the decoder by listening to the output audio quality. Second, we measured the Signal-to-Noise Ratio defined as:

$$SNR = -10 \cdot \log_{10} \left(\frac{\frac{1}{M} \sum_{i=1}^{M} (S[i] - \hat{S}[i])^2}{\frac{1}{M} \sum_{i=1}^{M} S[i]^2} \right) \quad (6)$$

where $S[i]$ are the original PCM samples before any encoding and decoding, and $\hat{S}[i]$ are the PCM samples after the encoding and decoding process.

The results of our experiments show that a 16-bit fixed point representation yields the *same* audio quality as a double implementation. In all the examples we tested, the SNR of the 16-bit fixed point and double implementations were very close. Figure 4 is an example of such measure. Note that the two SNR curves are overlapping.

4 Reducing the number of multiply-accumulates

To design a faster decoder, we looked for ways to reduce the total number of multiply-accumulates inside the synthesis subband filter block. Three different approaches were found:

- A Scaled Chen DCT: by modifying the window coefficients $D[i]$, we were able to reduce the number of multiplications involved in the Matrixing block by 27%,

[3] The standard does not define the subband coefficients to be smaller than 1.0 in norm, but we believe that their norm will rarely be over 1.0.

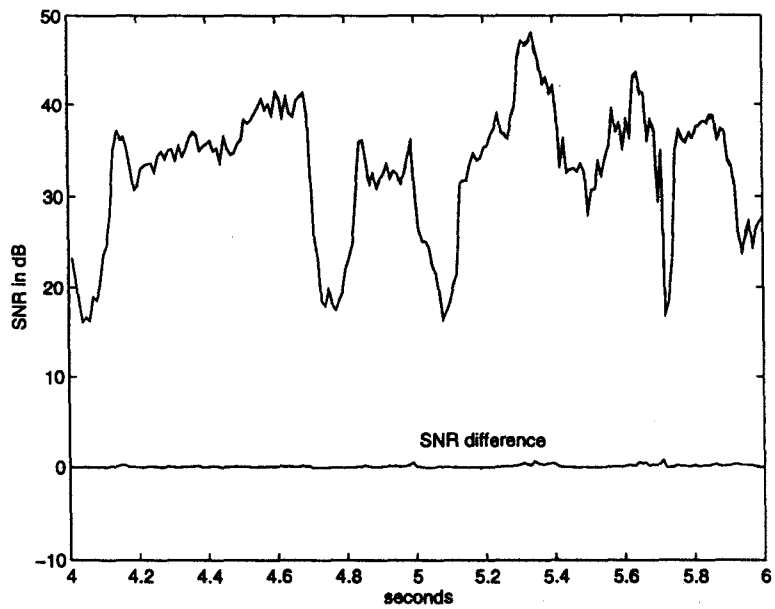

Figure 4: Typical SNR for the double implementation, and the 16-bit fixed point representation for a 2 second clip of an audio file. The difference between both SNR plots is also represented. Note that SNR are almost identical.

- Subband Coefficients and Window Pruning: after noticing no loss in audio quality when setting subband coefficients to zero, and truncating the window D, we reduced the number of additions and multiplications in both Matrixing and Windowing blocks.

We describe these approaches in the following paragraphs.

4.1 A Scaled Chen DCT

Figure 5 shows the structure of Chen's fast DCT algorithm which we used in our integer implementation. Note that every coefficient has magnitude less than 1.0. If we call C the matrix representation of Chen's fast DCT represented in this Figure, then we can write

$$C = Q_{32} \cdot C^* \tag{7}$$

where Q_{32} is a diagonal matrix defined with the cos and sin values showed on the right of Figure 5 next to the output coefficients $X(i)$, and C^* is a scaled version of Chen's fast DCT.

Let us show that the diagonal matrix Q_{32} can be absorbed into the windowing coefficients D.

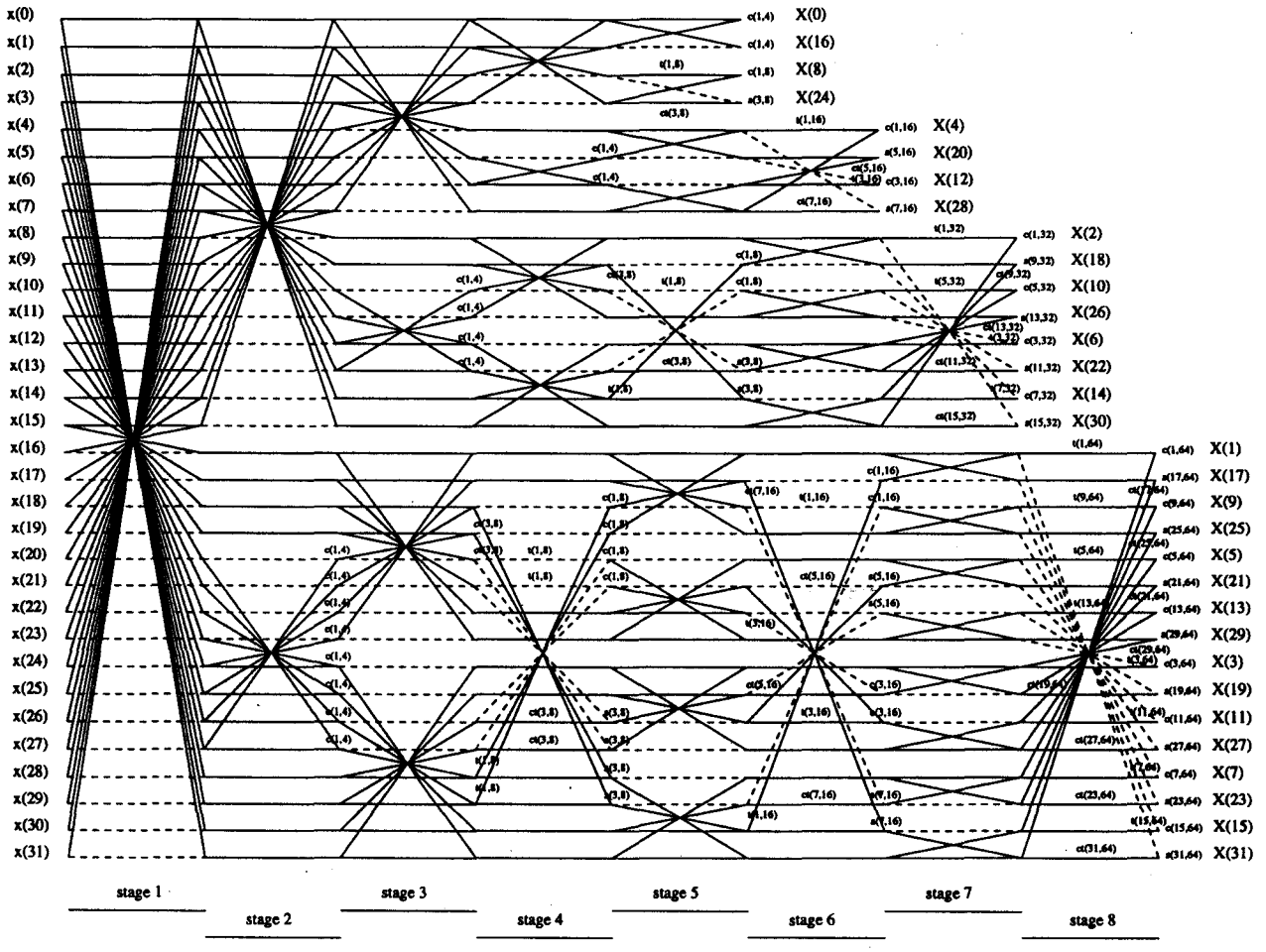

Figure 5: Chen's Fast DCT, note that no coefficient has a magnitude greater than 1.0 - $c(i,j) = \cos(\frac{\pi i}{j}), s(i,j) = \sin(\frac{\pi i}{j}), t(i,j) = \tan(\frac{\pi i}{j}), ct(i,j) = \cot(\frac{\pi i}{j})$. A dotted line represents a -1 multiplier.

Let V'' be the output of the 32-point fast DCT, S the input subband vector defined in Figure 2, then

$$V'' = C \cdot S \tag{8}$$

In the following, the diagonal matrix Q_{32} is defined as follows

$$Q_{32} = \begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{31} \end{pmatrix} \tag{9}$$

Using equation (7) in equation (8), and defining $V''^* = C^* \cdot S$, we have

$$V'' = Q_{32} \cdot V''^* \tag{10}$$

Introducing the results from [Kon94], and defining V to be the output of the Matrixing block defined in Figure 2, we can easily derive the following equality

$$V = T \cdot \begin{pmatrix} V'' \\ V'' \end{pmatrix} \tag{11}$$

where T is a 64 by 64 matrix defined as follows

$$T = \left(\begin{array}{c|c|c|c} & I_{16} & & \\ \hline & & & \Delta_{-1} \\ \hline \Delta_{-1} & O_{-1} & & \\ \hline & & -I_{16} & \end{array}\right) \tag{12}$$

where I_{16} is the 16 by 16 identity matrix, and Δ_{-1}, O_{-1} are also 16 by 16 matrices, defined by the following equations.

$$\Delta_{-1} = \begin{pmatrix} & & & & 0 \\ & & & 0 & -1 \\ & & \cdot & \cdot & \\ & 0 & -1 & & \\ 0 & -1 & & & \end{pmatrix} \tag{13}$$

$$O_{-1} = \begin{pmatrix} -1 & 0 & \cdots & 0 \\ 0 & 0 & & \\ \vdots & & \ddots & \\ 0 & & & 0 \end{pmatrix} \tag{14}$$

Grouping equations (10) and (11) yields

$$V = T \cdot \left(\begin{array}{c|c} Q_{32} & \\ \hline & Q_{32} \end{array} \right) \cdot \begin{pmatrix} V''^{*} \\ V''^{*} \end{pmatrix} = T \cdot Q_{64} \cdot \begin{pmatrix} V''^{*} \\ V''^{*} \end{pmatrix} \tag{15}$$

where Q_{64} is a 64 by 64 diagonal matrix.

Using equation (15) to find a 64 by 64 diagonal matrix K_{64} such that

$$V \;=\; K_{64} \cdot T \cdot \begin{pmatrix} V''^{*} \\ V''^{*} \end{pmatrix} \tag{16}$$

we identified K_{64} to be

$$K_{64} = \left(\begin{array}{c|c} \begin{array}{ccccccc} \lambda_0 & & & & & & \\ & \ddots & & & & & \\ & & \lambda_{15} & & & & \\ & & & 0 & & & \\ & & & & \lambda_{31} & & \\ & & & & & \ddots & \\ & & & & & & \lambda_{17} \end{array} & \\ \hline & \begin{array}{ccccccc} \lambda_{16} & & & & & & \\ & \lambda_{15} & & & & & \\ & & \ddots & & & & \\ & & & \lambda_1 & & & \\ & & & & \lambda_0 & & \\ & & & & & \ddots & \\ & & & & & & \lambda_{15} \end{array} \end{array} \right) \tag{17}$$

where $\lambda_0, \lambda_1, \cdots, \lambda_{31}$ are defined in equation (9).

Before looking at vector U defined in the Windowing block Figure 2, let

$$V^{*} \;=\; T \cdot \begin{pmatrix} V''^{*} \\ V''^{*} \end{pmatrix} \tag{18}$$

and rewrite equation (16) as follows

$$V = K_{64} \cdot V^* \tag{19}$$

Vector U, which has 512 elements, is constructed with 16 vector V. Calling these vectors $V_0, V_1, \cdots, V_{15}$, with V_0 the last output of the Matrixing block, we may write

$$U = \left(\begin{array}{c} R_1 \cdot V_0 \\ \hline R_2 \cdot V_1 \\ \hline R_1 \cdot V_2 \\ \hline R_2 \cdot V_3 \\ \hline \vdots \\ \hline R_2 \cdot V_{15} \end{array} \right) \tag{20}$$

where matrices R_1 and R_2 are 32 by 64 matrices defined with the 32 by 32 identity matrix I_{32}, and the zero matrix 0 as follows

$$R_1 = \left(\, I_{32} \, | \, 0 \, \right) \tag{21}$$

$$R_2 = \left(\, 0 \, | \, I_{32} \, \right) \tag{22}$$

Using equation (19), we may write U as follows

$$U = \left(\begin{array}{c|c|c|c} R_1 \cdot K_{64} & & & \\ \hline & R_2 \cdot K_{64} & & \\ \hline & & \ddots & \\ \hline & & & R_2 \cdot K_{64} \end{array} \right) \cdot \left(\begin{array}{c} V_0^* \\ V_1^* \\ \vdots \\ V_{15}^* \end{array} \right) \tag{23}$$

which simplifies to

$$U = \left(\begin{array}{c|c|c} K_{64} & & \\ \hline & \ddots & \\ \hline & & K_{64} \end{array} \right) \cdot \left(\begin{array}{c} V_0^* \\ V_1^* \\ \vdots \\ V_{15}^* \end{array} \right) = Diag(K_{64}, \cdots, K_{64}) \cdot U^* \tag{24}$$

where K_{64} appears 8 times in the diagonal.

Finally, the vector W defined in Figure 2, can be written as the product of a diagonal matrix defined with coefficients of window D, and the vector U as follows

$$W = \begin{pmatrix} D[0] & & & \\ & D[1] & & \\ & & \ddots & \\ & & & D[511] \end{pmatrix} \cdot U \tag{25}$$

and introducing the result from equation (24), we have

$$\begin{aligned} U &= Diag(D[0], D[1], \cdots, D[511]) \cdot Diag(K_{64}, \cdots, K_{64}) \cdot U^* \\ &= Diag(\lambda_0 D[0], \lambda_1 D[1], \cdots, \lambda_{15} D[511]) \cdot U^* \end{aligned} \tag{26}$$

Equation (26) proves that the scaling factors from the fast DCT can be absorbed into the window coefficients D.

This results in reducing the 116 multiplications of the standard Chen's fast DCT to 84.

4.2 Subband Coefficients Pruning

Subjective tests were conducted to assess the feasibility of discarding some of the subband coefficients. Our motivation for discarding some of the coefficients is that it would permit us to use a pruned DCT instead of a full 32-point DCT and thereby reduce the number of multiply-accumulates.

These tests used three monophonic audio files which are known to be difficult to code, and are commonly used to test high-quality digital audio codecs.

Result of these tests showed that at all sampling frequencies allowed by the standard (e.g. 32, 44.1, and 48KHz), and at bit rates 192 Kbit/s and 32 Kbit/s, a very good playback audio quality was found when setting the last 16 subband coefficients to zero.

4.3 Window Pruning

To further reduce the number of multiply-accumulates, we experimented with truncating the synthesis window D shown in Figure 3.

Additional subjective tests indicated that setting $\frac{1}{4}$ of the filter coefficients of window D, and only keeping coefficients $D[64], \cdots, D[447]$, did not reduce the overall audio quality.

By integrating the scaled Chen DCT, truncation of subband coefficients and window pruning into the design of a decoder, the total number of multiply-accumulates is reduced. For Chen's fast DCT, the first 32 additions disappear, reducing the total number of additions for this block by 16%. As for the Windowing block, the total number of operations is reduced by 25%.

4.4 Experimental results

The scaled Chen's fast DCT showed in Figure 5 was integrated into the software decoder. Two new functions were designed: chendct16, and chendct32 where successively 16 point, and 32 point inputs were considered to be non-zero.

Using the timing information given by the Unix `time` command, we measured the speed of execution for the decoding of the 19.49 seconds mono audio file sampled at 48KHz. The averages of 5 decoding time for the different integer software implementations are given in Table 3.

Note that in the case where the 16 highest subband coefficients are set to zero, and the window D is truncated, the total number of multiply-accumulates is reduced by 24%.

	No DCT	DCT	Chen DCT	
			32	16
without Window pruning	13.75	11.15	**8.71**	8.48
with Window pruning			7.81	7.63

Table 3: Decoding time of a 19.49 second audio sequence for the integer decoder implementation. The values reflect the average decoding time over 5 runs.

Comparing the speed execution of the 32 point Chen DCT without pruning window D, and the 16 point Chen DCT with pruning of window D, we find a 13% speed improvement.

Presently, simulations are being performed to assess the efficiency of this integer implementation compared with floating point implementation.

5 Conclusion

We designed a fast integer implementation of an MPEG-I audio decoder using a scaled version of Chen's fast DCT to implement the Matrixing operation in the subband filtering defined by the MPEG standard.

In the process, we found three new ways to reduce the number of multiply-accumulates. First we proved that 32 multiplications, out of the 116, of Chen's algorithm could be absorbed in

the windowing operation of the subband filtering. Second, we used the results of subjective tests, which showed that it is possible to set subband coefficients to zero and truncate the synthesis window without subjective loss of audio quality, and reduce by 24% the operations in the subband filtering.

Using these results improved by 13% the speed of our integer implementation decoder.

6 References

[BS94] Karlheinz Brandenburg and Gerhard Stoll. "ISO-MPEG-1 Audio: A Generic Standard for Coding of High-Quality Digital Audio". *Journal of the Audio Engineering Society*, 42(10):780–792, October 1994.

[CSF77] Wen-Hsiung Chen, C. Harrison Smith, and S. C. Fralick. "A Fast Computational Algorithm for the Discrete Cosine Transform". *IEEE Transactions on Communications*, 25(9):1004–1009, September 1977.

[JSW93] ISO/IEC JTC 1/SC 29/WG 11. *"ISO/IEC 11172-3, Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s - Part 3: Audio"*, 1993.

[Kon94] Konstantinos Konstantinides. "Fast Subband Filtering in MPEG Audio Coding". *IEEE Signal Processing Letters*, 1(2):26–28, February 1994.

[Lee84] Byeong Gi Lee. "A New Algorithm to Compute the Discrete Cosine Transform". *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 32(6):1243–1247, December 1984.

[Nol93] Peter Noll. "Wideband Speech and Audio Coding". *IEEE Communications Magazine*, pages 34–44, November 1993.

[OS89] Alan V. Oppenheim and Ronald W. Schafer. *"Discrete-Time Signal Processing"*. Prentice Hall Signal Processing Series, 1989. ISBN 0-13-216771-9.

[Pan95] Davis Pan. "A Tutorial on MPEG/Audio Compression". *IEEE Multimedia*, 2(2):60–74, Summer 1995.

[RY90] K. R. Rao and P. Yip. *"Discrete Cosine Transform, Algorithms, Advantages, Applications"*. Academic Press, 1990. ISBN 0-12-580203-X.

[Shl94] Seymour Shlien. "Guide to MPEG-1 Audio Standard". *IEEE Transactions on Broadcasting*, 40(4):206–218, December 1994.