An Approach to the Implementation of a Discrete Cosine Transform

GUIDO BERTOCCI, MEMBER, IEEE, BRIAN W. SCHOENHERR, AND DAVID G. MESSERSCHMITT, SENIOR MEMBER, IEEE

Abstract—**An approach to the implementation of a discrete cosine transform (DCT) for application to coding speech is described. The approach is oriented toward single speech channel encoding. In addition, a detailed computer simulation of an adaptive transform coder is described.**

The purpose of the computer simulation is to determine the internal precision at various points in the implementation required to avoid subjective degradation. Specific recommmendations are made on the required internal precision in the implementation of the discrete cosine transform.

A breadboard implementation of the DCT using SSI and MSI TTL logic based on the results of the computer simulation is reported.

I. INTRODUCTION

ADAPTIVE transform coding is, together with subband coding, a promising method of encoding speech at bit rates below 16 kbits/s [1]. A significant obstacle to the widespread application of transform coding is, however, its great complexity. A computationally intensive portion of the transform coder is the front end discrete cosine transform (DCT). In this paper an architecture for the implementation of a DCT is recommended. A detailed computer simulation of a transform coder, including the bit allocation algorithm as well as DCT, was performed for the purpose of determining the required internal finite wordlength precisions.

This is a critical problem, since choosing too high a precision complicates the implementation, and insufficient precision will result in degradation of speech quality beyond that inherent in the encoding technique. The simulation was carefully designed to accurately reflect these finite precision effects. The simulation was run on actual speech followed by informal listening tests to determine the effects of insufficient precision.

This paper does not consider the implementation of the bit allocation algorithm in a transform coder. The bit allocation method used in the simulator was that recommended in [1]. The design of the bit allocation portion of the coder is the most challenging part, particularly from an algorithmic standpoint.

Manuscript received August 11, 1981; revised October 20, 1981. This work was supported in part by the National Science Foundation under Grant 78-16966 and by GTE Lenkurt. The work of G. Bertocci and B. W. Schoenherr was performed in partial fulfillment of the M.S. degree from the University of California, Berkeley, CA.

G. Bertocci is with Bell Laboratories, Holmdel, NJ 07733.

B. W. Schoenherr is with Bell Laboratories, North Andover, MA 08145.

D. G. Messerschmitt is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

As a ground rule in this study, it was assumed that a single channel was being encoded, as opposed to the encoding of a large group of channels simultaneously. The available circuit techniques and device speeds were assumed to be constrained by those available in MOS-LSI. The DCT was actually implemented using SSI and MSI TTL parts using the architecture recommended here.

The proposed implementation architecture is described in Section II, and the simulation and results are described in Section III. A brief description of the breadboarded DCT is given in Section IV.

II. AN APPROACH TO IMPLEMENTATION OF THE DCT CODER

It has been recommended that a discrete cosine transform (DCT) [2] is the most appropriate fixed (nonadaptive) transform for speech signals. It is given by

$$G_x(0) = \frac{2^{1/2}}{N} \sum_{m=0}^{N-1} X(m) \tag{2.1}$$

$$G_x(k) = \frac{2}{N} \sum_{m=0}^{N-1} X(m) \cos\left(2\pi \frac{(2m+1)k}{4N}\right)$$

$$k = 1, 2, \cdots, (N-1)$$

where $X(m)$, $0 \leqslant m < N$, are the N speech samples in the block, and $G_x(k)$ are the transform coefficients. The inverse DCT is given by an analogous equation.

Several approaches to the implementation of the DCT were considered, including the straightforward implementation of (2.1) using either an entirely digital or a partially analog approach using switched-capacitor techniques. "Fast DCT" algorithms which have been proposed [3] were also considered. At a sampling rate of 8 kHz, the multiply rate for a straightforward implementation of (2.1) is only a modest one million per second for the recommended value of $N = 128$ [1]. Thus, the added control complexity of a fast algorithm is obviously not justified for a single channel transform coder (although it would be valuable in a multichannel application). For a single channel coder, the switched capacitor techniques were not estimated to offer an appreciable die area advantage over an all-digital implementation, but would offer a significantly more difficult design. As a result we settled on a straightforward digital implementation of (2.1).

There are two possible methods of calculating the N transform coefficients from N successive speech samples.

0090-6778/82/0400-0635$00.75

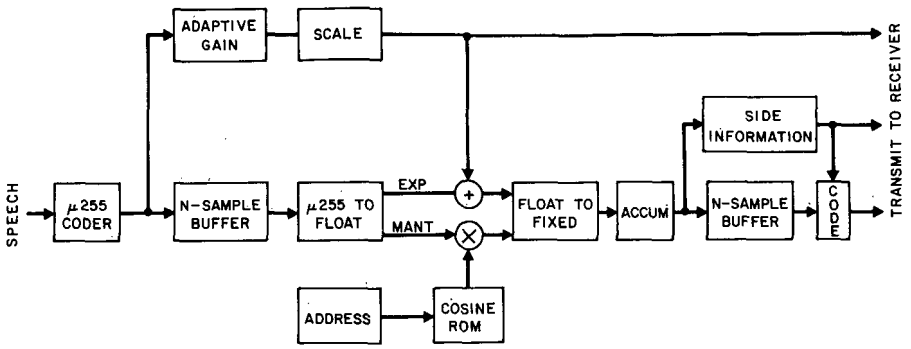


Fig. 1. Block diagram of a per-channel transform coder.

1) Calculate all N transform coefficients simultaneously, keeping, at any time, N partial accumulations. As each speech sample arrives, update each of the N partial accumulations.

2) Store in memory a block of N speech samples as they arrive. Simultaneously, calculate the transform coefficients of the previously stored block. This is done by calculating one transform coefficient per sampling interval, using all N stored speech samples.

The considerations in choosing one of these methods are as follows.

1) Method 2 requires storage locations for $2N$ speech samples, including those currently being received, as well as the last block on which the transform coefficients are currently being calculated. Method 1 requires no storage of speech samples.

2) Method 1 requires the storage of N partial accumulations, while method 2 has no partial accumulations. Both methods require memory for N transform coefficients for storage while the adaptive quantization side information is being calculated.

3) Both methods results in $2N$ sample periods of delay. In the case of method 1, the first block is used to calculate the transform coefficients, and the second to calculate the side information. For method 2, the first block is used to store the N speech samples, and the second is used to calculate both the transform coefficients and the side information.

4) Method 2 is compatible with adaptive feedforward quantization [1] in which a quantization scaling factor to be sent to the receiver as side information is calculated for the block of speech samples before the transform coefficients are calculated. This adaptive quantization results in a normalization of the transform coefficients, resulting in a reduction in the number of bits of precision in the accumulation and storage of transform coefficients and in the subsequent adaptive quantization algorithms which operate on a per-transform coefficient (as opposed to block) basis. In method 1, on the other hand, the transform coefficients are calculated prior to reception of the entire block of speech samples. Thus the accumulation requires greater precision.

In view of these considerations, the fourth point was considered overriding, and method 2 was chosen. This choice also results in fewer bits of required memory, in view of the fewer number of bits of precision required for speech samples as compared to partial accumulations and transform coefficients which have not been adaptively quantized.

This choice results in the configuration of Fig. 1. It is assumed that the speech is first encoded using the $\mu = 255$ encoding law commonly used in telephone networks. This type of coder is widely available in monolithic form and has adequate precision and dynamic range for this application. As the speech samples are read into an N-sample buffer, the adaptive feedforward quantization algorithm (the box labeled "adaptive gain") calculates some measure of the average speech power. This measure is then mapped into a scale factor in the box labeled "scale," which is applied before accumulation of the transform coefficients and is also transmitted as side information. As the samples are read from the N-sample buffer, they are converted into a true floating point representation (which is very similar to $\mu = 255$). Prior to accumulation, we must multiply by the values of the cosine as in (2.1). Since the cosine has a well-defined level, it can be stored in ROM in a fixed point representation. The multiply with the speech sample can be performed only on the mantissa of the speech sample floating point representation. The resulting values are then normalized by adding the previously determined scale factor to the exponent, and the resulting value is converted to fixed point prior to accumulation. The accumulation of the N values then determines one transform coefficient as in (2.1). The accumulation of these N values occurs during one speech sampling interval, and the N-sample buffer is read N times in order to determine the N transform coefficients.

As the N transform coefficients are calculated, they are stored in an N-sample buffer. Simultaneously, the adaptive quantization side information is calculated. Then, as the N transform coefficients are read from the buffer, they are coded using the appropriate step size and number of bits of precision, as determined by the side information.

Several aspects of the transform coder implementation deserve to be discussed in more detail. In the following sections we discuss the adaptive gain algorithm, the $\mu = 255$ to floating point conversion, and the generation of the cosine.

A. $\mu = 255$ to Floating Point Conversion

The conversion of a $\mu = 255$ sample to floating point to expedite a subsequent multiplication has been used in [4]. In this section the details of this conversion are developed. The $\mu = 255$ output level is given by [5]

$$X = 2^L(V + 16.5) - 16.5 \tag{2.2}$$

where X is the output (analog) level corresponding to $\mu = 255$

code (L, V), L is the segment number, $0 \leqslant L < 8$, and V is the level on a segment, $0 \leqslant V < 16$. The effect of the sign bit has not been incorporated into (2.2). We can find a floating point representation for X by setting it equal to $2^L V'$, where L is the exponent (the same as the segment number) and V' is the mantissa. Solving for V':

$$V' = V + 16.5(1 - 2^{-L}). \tag{2.3}$$

It is straightforward to see that $0 \leqslant V' < 32$, and that a full precision representation of V' would require 13 bits. However, if we used only 5 bits, the accuracy in V' would be ± 0.5, which would result in an accuracy in X of $\pm 2^{L-1}$. This is exactly half a $\mu = 255$ step size, since on segment L the step size is 2^L. Thus, a 5 bit precision on V' results in a maximum error of half a step size in the $\mu = 255$ law, and adding bits to V' reduces the error correspondingly.

The actual conversion from V to V' can be done by simple combinatorial logic. It should also be apparent that if the original speech should be encoded in fixed point rather than $\mu = 255$, the conversion to floating point would also be advantageous, although slightly more complicated.

B. Adaptive Gain

It is recommended in [1] that the input speech samples be normalized by their sample variance prior to calculation of the transform coefficients. However, once the speech samples have been expressed in floating point form, it should be adequate (and much simpler) to normalize them in terms of the exponents only. In particular, if we calculate the sample mean of the exponents in the block of N speech samples

$$L = \frac{1}{N} \sum_{i=0}^{N-1} L_i \tag{2.4}$$

where L_i is the exponent of sample i, and then subtract L from the exponent of each speech sample, the block will be normalized such that the average exponent is zero. This is roughly equivalent to normalizing by the geometric mean of the block of speech samples. Also note that L_i is the same as the segment number of the $\mu = 255$ sample, so that this normalization factor can be calculated directly from the $\mu = 255$ samples as they are being stored in the buffer (as shown in Fig. 1).

C. Generation of the Cosine

From (2.1), we see that the cosine must be generated at the $4N$ points uniformly spaced on a circle shown in Fig. 2. Obviously, only $N + 1$ values in one quadrant need be stored in the ROM, and the others can easily be inferred by symmetry. In addition, since N will undoubtedly be a power of two, it is convenient to store only N values. Fortuitously, the three angles marked with a question mark can never occur for N a power of two[1] (this is suggested by (2.1) and has been verified by a computer program for $N = 128$). This implies that the ROM need only have N addresses. In addition, angle zero is

[1] This was pointed out to the authors by H.-H. Lu.

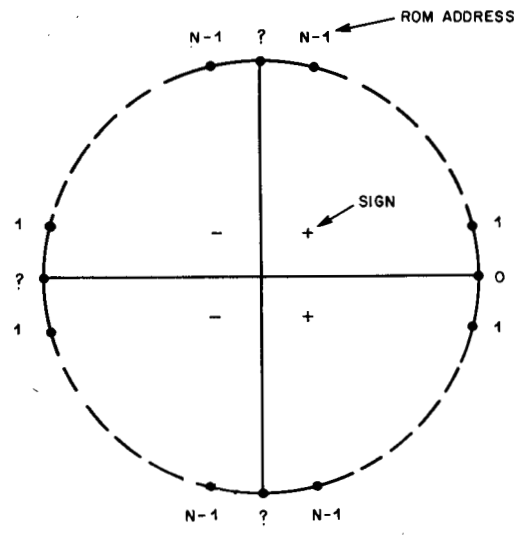


Fig. 2. $4N$ angles for which cosine must be generated.

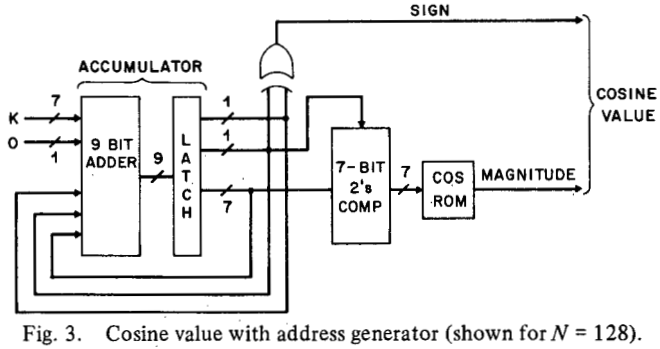


Fig. 3. Cosine value with address generator (shown for $N = 128$).

only addressed in calculation of the $k = 0$ transform coefficient, and hence ROM address zero can contain the value $2^{-1/2}$ rather than unity. The ROM addresses are then shown on the diagram for all four quadrants, and the required signs are also shown.

From (2.1), the ROM address can easily be derived from the quantity

$$I(m, k) = (2m + 1)k \text{ modulo } 4N \tag{2.5}$$

where m is the sample number in the block, and k is the transform coefficient number. If N is a power of two, the modulo operation is simply accomplished by using finite precision arithmetic in the address generation and ignoring any overflow. Of course, k is simply generated by a counter. The $I(m, k)$ can be generated sequentially from the relation

$$I(m + 1, k) = I(m, k) + 2k \tag{2.6}$$

as shown in Fig. 3. The accumulator simply adds $2k$ to the last value of $I(m, k)$ at each new speech sample. The resulting 9 bit address (for $N = 128$) specifies which of the $4N$ points is desired on Fig. 2. The two most significant bits specify the quadrant. The exclusive-or gate then determines the correct sign bit. In quadrants one and three, the seven least significant bits are used as the ROM address, and in quadrants two and four the two's complement of the 7 bit address is used.

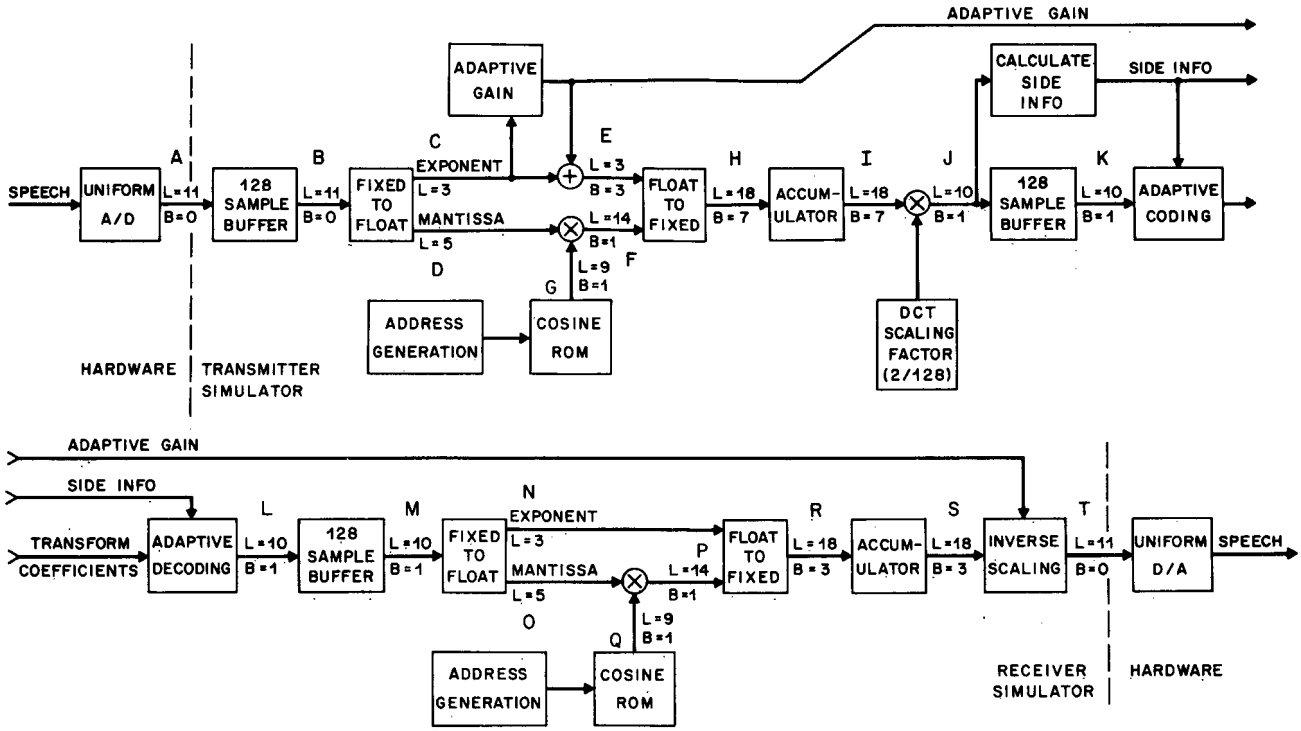


Fig. 4. Transform coder simulator architecture (L = bit length, B = binary point location).

III. GOAL OF COMPUTER SIMULATION

The quality of the speech processed by a transform coder should depend only on the bit rate of the transmitted speech as determined by the bit-allocation algorithm. Determining the minimum precision required to implement a transform coder is important in order to reduce to a minimum the memory requirements and complexity of the transform coder.

The goal of the computer simulation is to determine experimentally the required precision at each point of the transform coder. In particular, it is important to determine the required precision to implement the discrete cosine transform since most of the memory needed for a transform coder lies in implementing the DCT.

Since the simulator was not a goal in itself but a tool to determine required wordlength precisions, one compromise was made to accelerate the implementation of the transform-coder simulator. The compromise lies in the use of a 12 bit uniform coder instead of a $\mu = 255$ coder. A 12 bit uniform coder interfaced to a PDP 11/40 computer was available, greatly simplifying the collection of digitized voice samples for processing through the simulator.

A. Architecture of Computer Simulator

Fig. 4 shows the architecture of the computer simulator. As mentioned previously, there is one basic difference, the use of a 12 bit uniform coder. This substitution causes another small change. The adaptive gain is calculated using the exponent instead of using the $\mu = 255$ segment number and scaling. The result of these two processes is, however, identical.

1) DCT Scaling Factor: The DCT scaling factor is defined to be

$$\frac{2}{N} \quad \text{where } N = \text{block length.} \tag{3.1}$$

The $2/N$ factor comes from (2.1). By introducing the $2/N$ factor after all the partial coefficients have been accumulated, leading zeros are not carried through the entire discrete cosine transform.

Since N is a power of 2, multiplying by $2/N$ simply shifts the binary point. For the simulator, the location of the binary point is a very important parameter and must be maintained properly for each coefficient throughout the simulation (Section III-B discusses number representation). In actual hardware the location of the binary point is implicit, thus eliminating the necessity to multiply by $2/N$.

Thus, even though there appears to be an additional change in architecture from Fig. 1 (at point I in Fig. 4) this change is necessary only for the simulation and not in a hardware implementation.

B. Number Representation

The heart of the finite precision computer simulator is the ability to specify the number of bits in a coefficient and the location of the binary point. The location of the binary point is very important since it determines the tradeoff between the range of numbers that can be represented and the precision. Adding a bit to the left of a binary point increases the range and reduces the probability of an overflow. Adding a bit to the right of a binary point increases the precision.

For each coefficient, the binary point and bit length must

be specified. The binary point is defined relative to the most significant bit. For example, the coefficient 1.01 would be represented as $L = 3$ and $B = 1$, which means a wordlength of 3 bits and 1 bit to the left of the binary point. In addition, a sign bit is required. In Fig. 4 at points A-T the minimum precision needed to implement a one-channel transform coder is shown. The required wordlengths as indicated in Fig. 4 are given by

$$\text{total wordlength in bits} = L + 1. \tag{3.2}$$

This precision was determined from the simulation as described in Section III-D.

The simulator uses signed magnitude arithmetic for ease of implementation. Signed magnitude arithmetic creates one problem that does not exist with two's complement arithmetic: there is a duplicate representation of zero. For coefficients with bit lengths less than 4 there is a significant loss of information. For example, a coefficient with two bits can only represent three distinct values with signed magnitude arithmetic as opposed to four distinct values with two's complement arithmetic.

Since most of the transmitted coefficients are short, an average length of 2 for 16 kbit/s speech, the redundancy factor is important. To determine the minimum required precision for the DCT for 16 kbit/s speech, a higher bit rate is necessary to compensate for the redundant zero.

C. Overflows

One of the critical problems is overflowing a transform coefficient during accumulation. An overflow is defined to be a loss of most significant bits due to an insufficient number of allocated bits to the left of the binary point. An overflow can occur at two critical points. One is during the calculation of the DCT or IDCT and the other is during the scaling of the transform coefficients before truncating to a specified bit length for transmission.

Overflows during the calculation of the DCT or IDCT can be easily avoided by allocating bits to the left of the binary point. One subtle point to consider is that it is possible for a transform coefficient to exceed its final value during the accumulation of partial transforms. This is due to the possibility that several positive partial transforms are accumulated before negative partial transforms or vice versa.

Overflows during the scaling of the transform coefficients are very critical. The purpose of scaling the coefficients before truncating to an assigned bit length is to remove all leading zeros. However, the algorithm to determine the proper scaling factor is based on linearly interpolating between the averaged side information coefficients [1]. As a result, the scaling factor for a particular coefficient may cause a transform coefficient to overflow.

In an attempt to reduce the overflows during scaling, the simulator counts the number of overflows for each block of transform coefficients after each coefficient is scaled. If the number of overflows exceeds a predetermined tolerance, the transform coefficients are rescaled with a smaller scaling factor (each coefficient has one less leading zero truncated). This process is repeated until the number of overflows is below the tolerance. The number of fewer leading zeros truncated is sent to the receiver as side information.

In [1], the $\log_2$ of the variance of the transform coefficients is proposed as the side information. In an attempt to simplify the algorithm, the simulator uses the number of leading zeros for each coefficient instead of squaring the entire transform coefficient. This is very similar to using the exponent of a floating point number.

TABLE I
SIMULATION RESULTS ("WHY DO I OWE")

Label	Simul 1	Simul 2[a]	Simul 3
D	15	5	4
F	30	14	12
G	15	9	8
I	30	18	15
J	30	10	10
L	30	10	10
O	30	5	4
Q	15	9	8
P	30	14	12
R	30	18	15
SNR (dB)	16.57	16.16	14.87

[a] Values recommended for minimum required precision as shown in Fig. 4.

D. Simulation Results

Three phrases low-pass filtered at 3500 Hz were used for processing. The three phrases are "Why do I owe," "Why not be louder" and the letter "e."

The strategy for determining the required precision was to first set all bit lengths to the maximum allowable by the simulator. The number of overflows allowed during scaling of the transform coefficients was varied until the SNR was maximized. Three overflows per block of 128 appeared to be close to optimal for all three samples.

The next step was to reduce the precision at each point in the coder, starting from the maximum wordlength of 30 or 15 depending on the coefficient, until there was noticeable degradation in the SNR and quality of the speech (the latter determined by informal listening tests comparing the original uncoded speech with the processed speech).

Table I shows the resulting SNR (not segmental SNR) for three simulations using different wordlengths. Each of the three simulations were conducted at 16 kbits/s. Simulation 1 uses the maximum precision allowed by the simulator. Simulations 2 and 3 show a threshold at which performance, as measured by SNR, drops dramatically with a small reduction in coefficient precision.

The labels in Table I correspond to the labels (A-T) in Fig. 4. Only the values that changed from simulation to simulation are shown. All nonlisted values are those shown in Fig. 4. In Fig. 4 "L" denotes the length of each word in bits and "B" denotes the location of the binary point relative to the most significant bit.

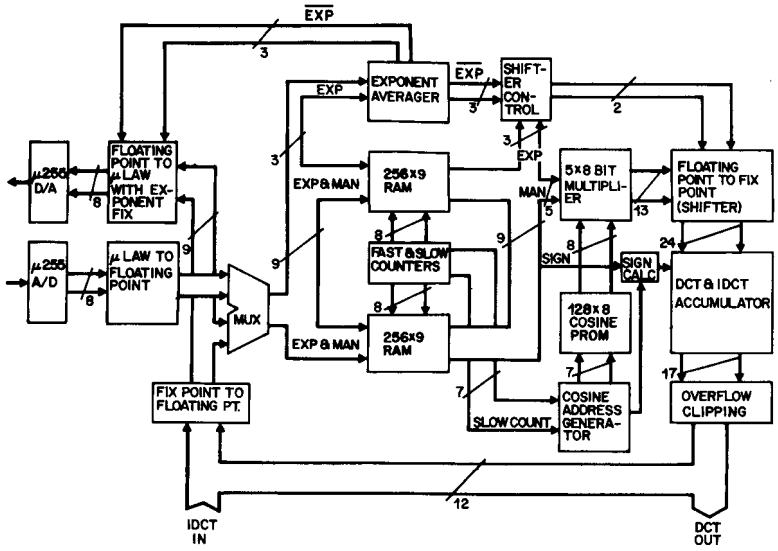


Fig. 5. DCT/IDCT hardware architecture.

The transform coder output bit rate used for the simulations was 16 kbits/s. This bit rate does not take into account the side information or the inefficiencies due to the use of signed magnitude notation. These two factors have an opposing effect. Including the side information would increase the bit rate for the same SNR while using a more efficient representation of coefficients would decrease the bit rate for the same SNR.

Once the minimum required precision for 16 kbits/s was determined, simulations were conducted to determine if the same precision was also adequate for 24 kbits/s. The SNR for 24 kbits/s using the same precision as shown in Fig. 4 was only 0.7 dB lower than the SNR obtained with the maximum precision allowable by the simulator at 24 kbits/s. This result indicates that the precision as shown in Fig. 4 is sufficiently conservative to allow for improvements in the coding algorithm at 16 kbits/s without being limited by the DCT implementation.

IV. BREADBOARD IMPLEMENTATION

The DCT-AQF section of the transform coder was implemented with SSI and MSI TTL hardware. Because similar functions are required by both the DCT and IDCT, the hardware that performed these functions was operated at twice its normal frequency so that data from both the DCT and IDCT could be interleaved. Hardware that is shared by both the DCT and IDCT includes the memories, the cosine value generator, the multiplier, the floating point to fixed point converter, the accumulator, and the overflow detection circuitry.

The hardware implementation architecture is shown in Fig. 5. Fig. 5 clearly shows the shared structure for the DCT and IDCT. The word widths recommended in Fig. 4, derived by the software simulation, were used for the implementation except for three modifications.

The first modification involved the cosine value generator PROM. The PROM has 8 bit word widths representing values less than unity. Each of these values had its least significant bit rounded to account for lesser significant bits, except for the second through sixth values. These should have been rounded to 1.00000000 but, because the multiplier used only numbers less than unity, these values were truncated to 0.1111111. This error should be insignificant because, at most, this represents 5/128 of an accumulation where the least significant seven bits are truncated.

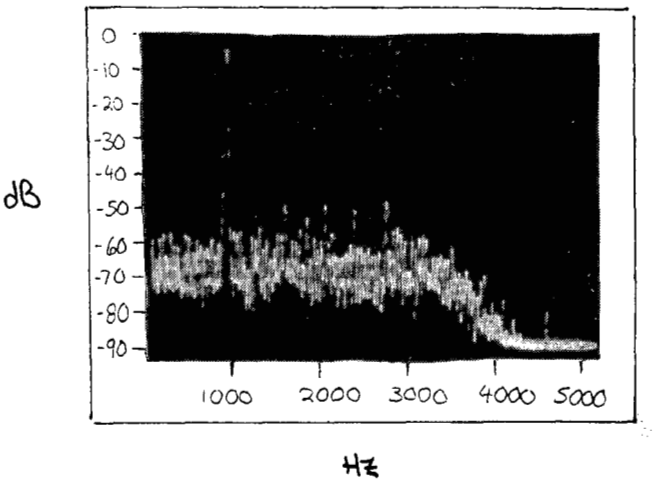
The second modification involved the floating point to fixed point converter. Because of the time required to input a value to the accumulator, only shifts up to six positions were allowed. This is also of minor consequence because the adaptive quantization of the input makes the occurrence of seven shifts highly unlikely.

The third modification involved the accumulator. It was determined that an additional five chips would be needed to implement a full precision accumulator with 24 bits instead of 18 as suggested by the simulations. Therefore, the additional cost was determined to be worthwhile in light of the fact that the hardware could be used for other purposes than 16 kbit/s transform coding.

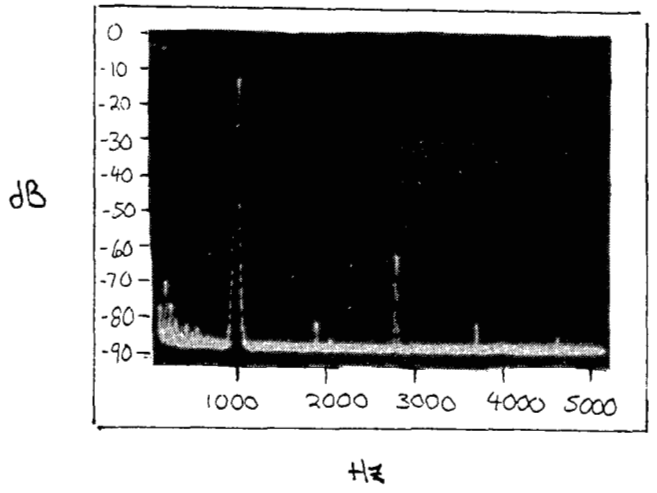
The circuit implementation required 139 assorted SSI and MSI TTL integrated circuits, as well as seven MOS integrated circuits (codec, filters, and memories). Three power supplies were required (−5 V, +5 V, and +12 V). A 16.384 MHz clock was used.

The performance in looped operation, the DCT output connected to the IDCT input, was found to be limited only to that of the input and output coder-decoder, with a 2 block (32 ms) delay. Fig. 6 shows a mainly 920 Hz input spectrum and the resulting output spectrum. The SNR as calculated from the output spectrum, using a 10 Hz window with a spectrum analyzer, is approximately 38 dB which is characteristic of a codec.

From the implementation of the DCT and IDCT it became clear that most of the cost was in the multiplications and format conversions and not in the accumulation of the coefficients. This result is partially due to the fact that semiconductor memories are available in units that are powers of two (i.e., a 16 bit accumulator would have saved significantly more chips than an 18 bit accumulator).



(a)



(b)

Fig. 6. (a) Output spectrum (10 Hz window). (b) Input spectrum (10 Hz window).

REFERENCES

[1] R. Zelinski and P. Noll, "Adaptive transform coding of speech signals," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, Aug. 1977.

[2] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, Jan. 1974.

[3] J. Makhoul, "A fast cosine transform in one and two dimensions," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, Feb. 1980.

[4] D. L. Duttweiler, "A twelve-channel digital echo canceler," *IEEE Trans. Commun.*, vol. COM-26, May 1978.

[5] H. Kaneko, "A unified formulation of segment companding laws and synthesis of codecs and digital companders," *Bell Syst. Tech. J.*, vol. 49, Sept. 1970.

[6] J. Tribolet and R. Crochiere, "Frequency domain coding of speech," *IEEE Trans. Acoust., Speech, Signal Processing*,, vol. ASSP-27, pp. 512–530, Oct. 1979.

[7] ——, "An analysis/synthesis framework for transform coding of speech," in *Proc. IEEE Conf. Acoust., Speech, Signal Processing*, 1979, pp. 81–84.

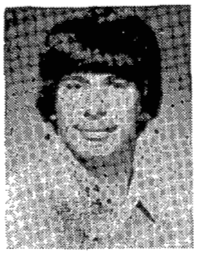
[8] J. Huang and P. Schultheiss, "Block quantization of correlated Gaussian random variables," *IEEE Trans. Commun. Syst.*, vol. CS-11, pp. 289–296, Sept. 1963.

[9] S. Campanella and G. Robinson, "A comparison of orthogonal transformations for digital speech processing," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 1045–1049, Dec. 1971.

[10] R. Zelinski and P. Noll, "Approaches to adaptive transform speech coding at low bit rates," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, Feb. 1979.

[11] B. Kernighan and D. Ritchie, *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

★



Guido Bertocci (S'79–M'79–S'80–M'80) was born in Milan, Italy, in 1957. He received the B.S. degree with university honors in electrical engineering from Carnegie–Mellon University, Pittsburgh, PA, in 1979 and the M.S. degree in electrical engineering and computer science from the University of California, Berkeley, in 1980.

Since 1979, he has been a member of the Technical Staff at Bell Laboratories, Holmdel, NJ. He is currently working on low bit rate algorithms for voice and voiceband data.

Mr. Bertocci is a member of Eta Kappa Nu.

★

Brian W. Schoenherr was born in Kingston, NY, in 1958. He received the B.E.E. degree with distinction from the Institute of Technology, University of Minnesota, Minneapolis, in 1980 and the M.S. degree in electrical engineering and computer science from the University of California, Berkeley, in 1981.

Since 1980, he has been a member of the Technical Staff at Bell Laboratories, Merrimack Valley, MA, where he is working in the area of digital signal processing.

Mr. Schoenherr is a member of Tau Beta Pi and Eta Kappa Nu.

★

David G. Messerschmitt (S'65–M'68–SM'78) received the B.S. degree from the University of Colorado, Boulder, in 1967, and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1968 and 1971, respectively.

He is Professor of Electrical Engineering and Computer Science at the University of California at Berkeley. From 1968 to 1974 he was a member of the Technical Staff and from 1974 to 1977 a Supervisor at Bell Laboratories, Holmdel, NJ, where he did systems engineering, development, and research on digital transmission lines and terminals, digital speech interpolation, and digital signal processing, particularly as it relates to both low and high rate encoding of speech. His current research interests are analog and digital signal processing, with applications to voice encoding; digital transmission; phase-locked loops; multiprocessor approaches to signal processing and circuit simulation; and adaptive filtering. He has published 35 papers and has eight patents issued or pending in these fields. Since 1977, he has served as a Consultant to industry, including among other companies TRW Vidar, Hughes Aircraft, Acumenics, IBM, Intelsat, and Intel.

Dr. Messerschmitt is a member of Eta Kappa Nu, Tau Beta Pi, and Sigma Xi. He is currently serving as Editor for Transmission Systems for the IEEE TRANSACTIONS ON COMMUNICATIONS.