AT&T Technical Journal
Vol. 64, No. 8, October 1985
Printed in U.S.A.

Single-Chip Implementation of Feature Measurement for LPC-Based Speech Recognition

J. G. ACKENHUSEN* and Y. H. OH†

(Manuscript received May 14, 1985)

A single-chip implementation of Linear Predictive Coding (LPC)-based feature measurement for speech recognition, called the Feature Extracting Digital Signal Processor (FXDSP), has been developed by programming the AT&T *DSP20*™ programmable Digital Signal Processor (DSP) and has been verified by both numerical simulation and system use. For identical input, the recognition distance between floating point simulation and the DSP implementation was found to be negligibly small when compared with distances for word matches. The feature-measurement technique is identical to that used in numerical simulations of LPC-based isolated- and connected-word recognition using combinations of dynamic time warping, vector quantization, and hidden Markov modeling. As a result, the FXDSP represents a single-chip common building block for real-time implementation of most speech recognition techniques under investigation at AT&T Bell Laboratories. The FXDSP performs eighth-order LPC analysis on speech received from a standard CODEC. In every frame period (15 ms) it produces a feature vector consisting of the log energy, nine amplitude-normalized autocorrelation coefficients, and nine LPC-based test-pattern coefficients. The feature-measurement program requires 1023 locations of the 1024 available in on-chip program ROM, 211 of 256 available RAM locations, and 75 percent of available real time.

I. INTRODUCTION

Most speech recognition work at AT&T Bell Laboratories has been based on a standard form of feature measurement first proposed by

* AT&T Bell Laboratories. † AT&T Bell Laboratories, now with Texas Instruments.

Itakura.[1] Speech recognition features are computed from an eighth-order Linear Predictive Coding (LPC) calculation on 45-ms analysis frames spaced by 15 ms. The autocorrelation method is used, and the speech is of telephone bandwidth (3.3 kHz) and is sampled at 6667 samples per second.

With this front end, numerical simulations have demonstrated successful word recognition algorithms based on dynamic programming for both isolated words[2] and connected words.[3] Real-time hardware that uses this front end for isolated-word recognition has been reported.[4] More recent simulations have used the same front end in recognizers that use vector quantization for isolated-[5] and connected-word recognition[6] and for recognizers using hidden Markov modeling.[7]

Comparative tests of the LPC front end with a variety of filter banks have found the LPC technique to provide superior performance for complex vocabularies over telephone bandwidths.[8]

This paper describes a real-time implementation of this LPC feature-measurement technique that is of single-chip complexity. The implementation uses a programmable signal processor, the AT&T Bell Laboratories Digital Signal Processor (DSP).[9] In this implementation, called the FXDSP (Feature Extracting Digital Signal Processor), the output continuously provides results of LPC analysis of whatever input signal is present with less than one frame (15 ms) of delay.

1.1 Relation to previous work

An implementation of LPC analysis using two DSP chips was previously described by Daugherty.[10] This used an older version of the programmable signal processor known as DSP-1. The DSP-1 operates at one half the speed (5-MHz clock) and has one half the RAM (128 20-bit words) as the *DSP20*™ signal processor used here, but has the same size program memory (1024 16-bit words). Thus, one *DSP20* signal processor is equivalent to two DSP-1 processors in speed and RAM, but is the same as one DSP-1 in program memory.

A major challenge of the work reported here was to reduce the program size by a factor of 2 to attain single-chip implementation. A second challenge was to combine two separate time scales, that of the input (150 μs) and that of the output (15 ms), which had previously been separated by two DSP-1 processors, into a single processor, the *DSP20* signal processor.

A microprocessor-based implementation of an isolated-word recognizer had partitioned the feature-measurement task between a slower general-purpose 16-bit microprocessor performing decision operations and a faster, special-purpose two-board signal processor performing high-speed repetitive arithmetic.[4] This arrangement is similar to the

original simulation environment of a minicomputer and array processor.

An implementation of 10th-order LPC analysis has been developed for the TMS320* signal processor.[11] The TMS320 signal processor uses a sampling rate of 8 kHz, a frame size of 30 ms, and a frame period of 20 ms. This combination of frame size and period results in a frame overlap of 33 percent, where each sample contributes to an average of 1-1/2 frames. The DSP implementation described here uses a frame overlap of 67 percent, and thus requires three frames of computation to be completed on each sample. However, an increase in recognition error rate accompanies the reduction in computation obtained by a reduction in frame overlap, as shown by numerical simulation.[12] In the TMS320 signal processor implementation, the same circuit also performs pattern matching for connected-word recognition.

In addition to realizations based on programmable signal processors, architectures for single-chip LPC feature extractors that use a custom-built processor have been described.[13]

1.2 Organization of paper

In Section II, we examine the equations of LPC feature measurement. Section III describes the DSP chip and the external circuitry required to do the feature measurement. Section IV describes the architecture of the FXDSP program, and Section V presents some details of program implementation. In Section VI, the comparison of the real-time FXDSP calculation with a floating point simulation is described.

II. LPC FEATURE MEASUREMENT

The requirement of LPC is to determine a unique set of predictor coefficients, a_k, $k = 1, 2, \cdots, p$, that minimize the sum of squared differences, E_n, between actual speech samples, $s(n)$, and approximated speech samples, $\tilde{s}(n)$. The approximated speech samples $\tilde{s}(n)$ are formed from a linear combination of speech samples over a short segment of the speech waveform. Thus, the approximate speech samples are given by

$$\tilde{s}(n) = \sum_{k=1}^{p} a_k s(n-k), \tag{1}$$

where $p = 8$ in this analysis. The task of minimizing the prediction error, E_n, is to choose a_k such that

*Trademark of Texas Instruments.

$$E_n = \sum_m e_n^2(m) \tag{2}$$

$$= \sum_m [s_n(m) - \tilde{s}_n(m)]^2 \tag{3}$$

$$= \sum_m [s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k)]^2 \tag{4}$$

is a minimum.

Techniques for calculating the linear prediction coefficients, a_k, from the speech samples, $s(n)$, are described in the literature.[14] The method used here is a block-processing technique based on the autocorrelation method and Durbin's recursion (Fig. 1).

Speech which has been bandlimited to 100 to 3300 Hz and sampled at 6667 samples per second is first preemphasized with a first-order network:

$$s'(n) = s(n) - as(n-1); \quad a = 0.95. \tag{5}$$

The preemphasized speech is then blocked into frames of 300 samples (45 ms) which are spaced by 100 samples (15 ms). Thus, the lth frame of speech, $\tilde{x}_l$, is given by

$$\tilde{x}_l = s'(Ml + n), n = 0, 1, \cdots, N-1; \qquad l = 0, 1, \cdots, L-1, \tag{6}$$

where $M = 100$ and $N = 300$ for an input sequence length of L frames. As a result of this choice of M and N, each speech sample contributes to three consecutive analysis frames.

Each frame is then smoothed by a Hamming window:

$$x_l(n) = w(n) \cdot \tilde{x}_l(n), \tag{7}$$

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right), \qquad N = 300. \tag{8}$$

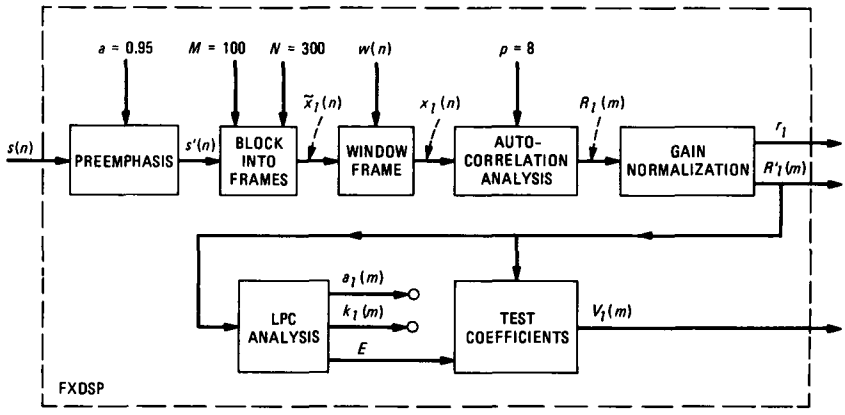

Fig. 1—Signal processing for extracting LPC features for recognition.

The resulting windowed frames of speech data are used to perform an autocorrelation calculation, given by

$$R_l(m) = \sum_{n=0}^{N-1-m} x_l(n)x_l(n+m); \qquad m = 0, 1, \cdots, 8. \tag{9}$$

The logarithm of the frame energy, $R_l(0)$, is then calculated:

$$r_l = \log_2 R_l(0). \tag{10}$$

The autocorrelation coefficients are gain-normalized such that $R_l'(0) = 1$, as follows:

$$R_l'(m) = \frac{R_l(m)}{2^{r_l}}. \tag{11}$$

This normalization is required so that later computation of Durbin's recursion uses the full integer precision of the machine. The log energy, r_l, is used for end-point detection and frame energy information during the recognition process.

Durbin's recursion is then applied to calculate a set of PARCOR coefficients, k_i, $i = 1, 2, \cdots, 8$, and a prediction residual from the $R_l'(m)$ for each frame as follows (the frame index l is suppressed):

$$E^{(0)} = R'(0). \tag{12}$$

For $i = 1, 2, \cdots, 8$, do eqs. (13) through (16):

$$k_i = \frac{\left[R'(i) - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} R'(i-j) \right]}{E^{(i-1)}} \tag{13}$$

$$\alpha_i^{(i)} = k_i \tag{14}$$

$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{i-1}; \qquad (j = 1, 2, \cdots, i-1; \quad i \neq 1) \tag{15}$$

$$E^{(i)} = (1 - k_i^2) E^{(i-1)}. \tag{16}$$

Extract final residual, E, and LPC coefficients a_j:

$$E = E^{(8)} \tag{17}$$

$$a_j = a_j^{(8)}. \tag{18}$$

Test-pattern coefficients are then formed by computing:

$$V_l(m) = \frac{R_l'(m)}{E}, \qquad m = 0, 1, \cdots, 8. \tag{19}$$

The FXDSP output consists of r_l, $R_l'(m)$, and $V_l(m)$ for $m = 0, 1, \cdots, 8$. The PARCOR coefficients k_i and LPC coefficients a_j are calculated as a result of calculating E; however, since they are not used directly in real-time pattern matching, they are discarded. Ref-

erence templates are made up of autocorrelations of a_j that are produced during a non-real-time vocabulary training session. In the current robust training algorithm, a reference pattern is made up of an autocorrelation average of two tokens that correspond to two different repetitions of a word.[15] Therefore, no use can be made of the LPC coefficients in real time.

III. HARDWARE

The hardware for this implementation consists of a μ-law CODEC with filters, which is run at a 6.667-kHz sampling rate, and the AT&T Bell Laboratories DSP, which is run at 10 MHz (Fig. 2). Separate oscillators control the sampling rate of the CODEC and the clock of the DSP.

A design alternative would have been to replace the 8-bit μ-law CODEC with a 12- or 13-bit linear analog-to-digital converter. Although a slight amount of quantization error is introduced by the μ-law conversion of the CODEC followed by the conversion back to 13-bit linear representation in the DSP, this error was seen to be minor. The benefit of the economy of a simple hardware interface between the DSP and the CODEC, the lower cost of the CODEC as compared with a 13-bit linear converter, and the fact that any telephone line input to the CODEC has probably already been subjected to conversions from analog to μ-law digital and back justified the slight degradation of waveform.

A block diagram of the DSP is shown in Fig. 3. The version used here, known as the *DSP20* signal processor, is an improved version of the original signal processor described in Ref. 9 in which both speed and RAM size have been doubled.

The *DSP20* signal processor has a 400-ns instruction cycle time. The processor consists of a read/write memory of 256 20-bit words and a mask-programmable program ROM of 1024 16-bit words. Alter-

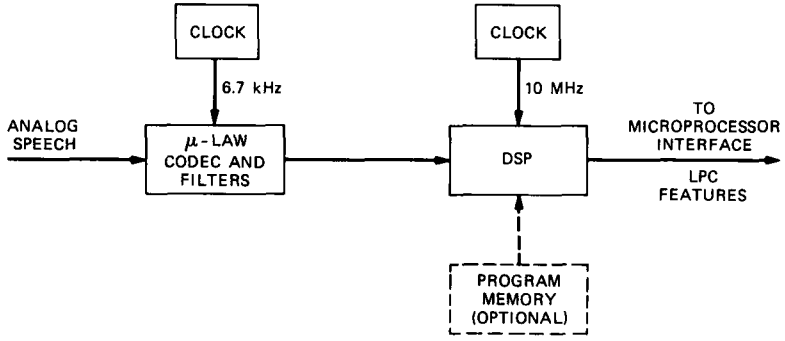

Fig. 2—LPC feature measurement hardware.

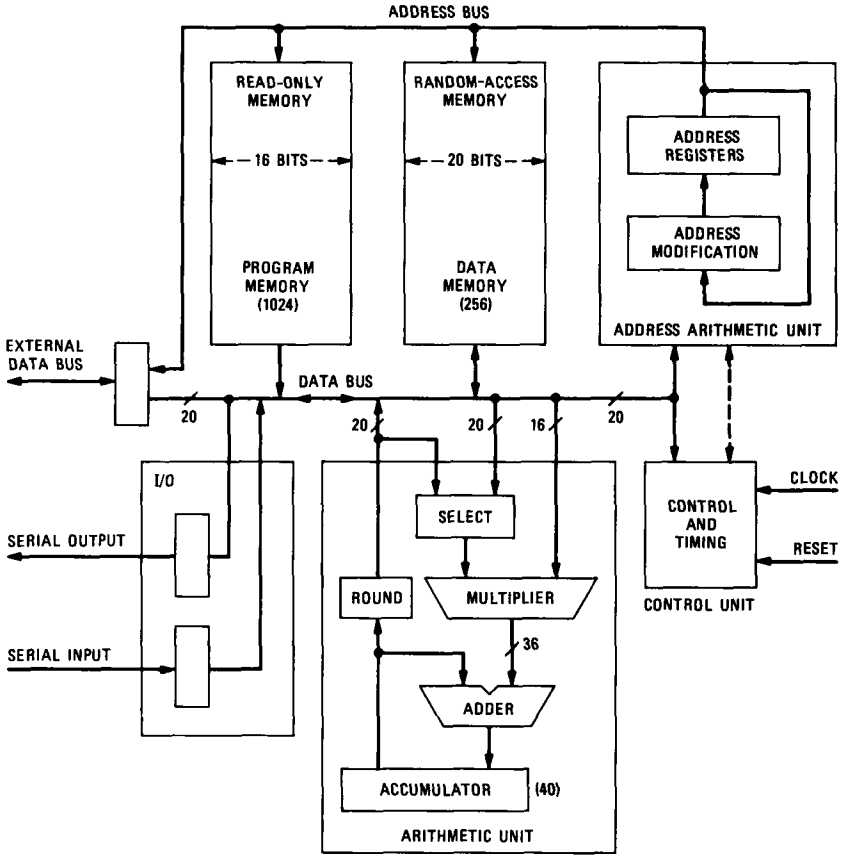

Fig. 3—Block diagram of DSP.

natively, the DSP can be run from 1024 words of external program memory, usually made of erasable programmable ROM, or RAM that can be down loaded. An address arithmetic unit contains registers for controlling memory access. A data arithmetic unit contains a 16-bit × 20-bit multiplier, a 40-bit accumulator, a 40-bit adder, and a 20-bit rounding-overflow circuit. Input and output occur through two serial data pins.

In one 400-ns machine cycle, the DSP can (1) decode an instruction, (2) fetch data and perform a multiplication, (3) accumulate output products from the multiplier, and (4) store data in memory.

IV. PROGRAM ARCHITECTURE

A conflict arises between the input time scale of the FXDSP, one sample every 150 μs, and its output time scale, 19 coefficients of a

feature vector every 15 ms. The FXDSP is required to process a new sample every 150 μs regardless of any other operation in progress, else the input sample is lost and the resulting frame feature vector is incorrect. Thus, two time scales exist, a sample time scale and a frame time scale.

As a result of the two time scales, the program architecture really consists of two separate programs, a sample update program that updates autocorrelation vectors every four samples [eqs. (5) through (9)] and a frame-recursion program that calculates the output feature vector from the autocorrelation vectors from the previous frame [eqs. (10) through (19)]. The frame-recursion program is divided into smaller pieces that are interposed with repeated executions of the sample update program (Fig. 4).

The sample update program operates on four samples each time it is executed. This four-sample operation is a compromise between fully block processing, in which autocorrelation vectors are calculated on a

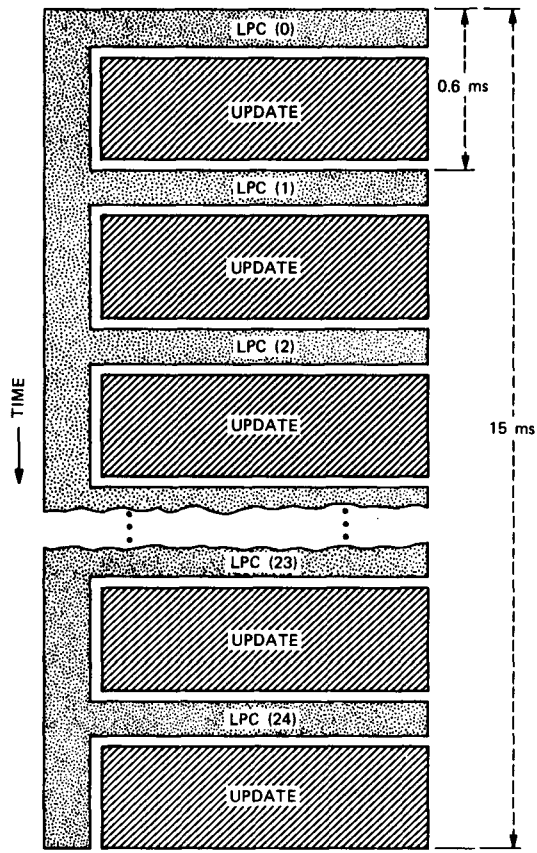

Fig. 4—Interleaving of sample update and frame-inversion programs.

frame of 300 samples all at once, and fully stream processing, in which the autocorrelation vectors are updated upon receipt of each new sample.[10] Fully block processing, however, requires enough read/write memory to store all 300 samples, which is more memory than the single-chip DSP has available. Fully stream processing, which has been used in other implementations,[4,11] executes too slowly for real-time analysis on the DSP.[10] This is because before any autocorrelation update occurs, address pointers must be set up for accessing samples and autocorrelation vectors, and each autocorrelation coefficient must be accessed and placed in the accumulator of the arithmetic unit. These overhead operations are necessary for any number of samples used in the update, and can only be tolerated in real time if the updates occur for more than one sample at a time.

The frame period of 100 samples and the updating of autocorrelation vectors by four samples at a time require that the update program be executed 25 times per frame period. Therefore, an output operation of one frame coefficient is added to the sample update program to provide 25 output coefficients per frame, spaced at four sample intervals. The 19 frame coefficients (r_l, $R_l(m)$, and $V_l(m)$, $m = 0, 1, \cdots, 8$) and six consecutive zeroes are output for each frame. The sequence of six zeroes provides a synchronization marker for identification of the 19 coefficients by the processor that receives the output of the FXDSP.

Figure 5 shows a more detailed view of the timing of operations. The frame recursion is divided into 25 pieces numbered LPC(0) through LPC(24). Between the first and second samples of the group of four sample inputs, one piece of the frame recursion program is

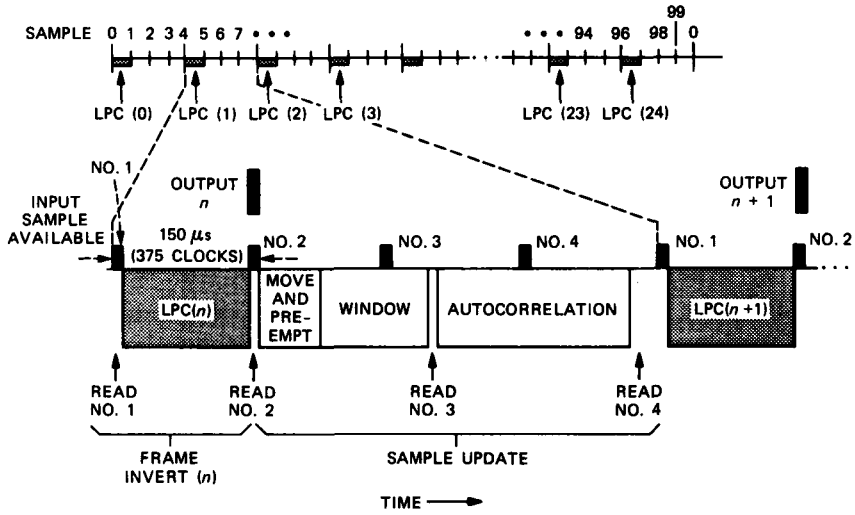

Fig. 5—Timing of input, output, and program sections.

executed. Each piece is completed within the sample period of 150 μs. In Table I, the function of each piece of the frame recursion is shown, as well as the time required for execution and the number of 16-bit words in program ROM required for that piece. As shown at the bottom of Table I, the final 11 time slots, LPC(15) through LPC(24), are unused.

During the time following the second, third, and fourth samples, the sample update operation is performed. The operations associated with the sample update operation are described in Table II. A sample is available every 150 μs and is placed in the FXDSP input buffer by the CODEC. The update program reads that sample at a convenient time, but before the next sample, arriving 150 μs later, overwrites it. Each sample is immediately converted from μ-law to linear encoding by the FXDSP and is then written into a four-sample buffer without any further processing until all four samples are obtained.

Table I—Frame recursion timing and program memory (by function)

Label	Function	Execution Time (μs)	Program Locations
LPC (0)	Read $R_l(m)$ to frame recursion input buffer, shift window	95	97
LPC (1)	Calculate r_l	60	143
LPC (2)	Calculate $R_l'(m)$; $m = 1, 2, 3, 4$	144[1]	50
LPC (3)	Calculate $R_l'(m)$; $m = 5, 6, 7, 8$	144	8[2]
LPC (4)	Set up for Durbin's recursion [$E_0 = R_l'(0)$]	50	32
LPC (5)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 1$)	128	226
LPC (6)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 2$)	128	6[2]
LPC (7)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 3$)	128	6
LPC (8)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 4$)	128	6
LPC (9)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 5$)	128	6
LPC (10)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 6$)	128	6
LPC (11)	Calculate $1/E_{i-1}$ and Durbin's recursion ($i = 7$)	128	6
LPC (12)	Calculate $1/E_{i-1}$ and Durbins's recursion ($i = 8$)	128	6
LPC (13)	Calculate $1/E$	128	6
LPC (14)	Calculate $V_l(m)$, $m = 0, 1, \cdots, 8$	12	41
LPC (15) thru LPC (24)	Idle	12 each	26
	Total (% used of available)	1777 (12%)	688[3] (67%)

[1] For signal 51 dB down from peak; shorter execution time for stronger signals.
[2] Locations include only the subroutine call; subroutine previously counted.
[3] Total includes 17 locations of the power-up initialization routine not listed above.

Table II—Sample update timing and program memory (by function)

Label	Function	Execution Time (μs)	Program Locations
Read #1	Read and μ-to-linear convert sample	3	11
Output	Output one frame feature coefficient	6	29
Read #2	Read and μ-to-linear convert samples		
Move and pre-emp	Shift sample buffer by four samples and preemphasize four samples	60	54
Window	Calculate window values and apply three times to four samples	111	123
Read #3	Read and μ-to-linear convert sample		
Autocorrelation	Use four samples to update nine autocorrelation vectors for three overlapped frames	193	118
Read #4	Read and μ-to-linear convert sample		
	Total (% used of available)	337 (62%)	335 (33%)

As a result, the sample update program has a pipeline delay of four samples. The frame recursion program calculates on the frame just completed and produces the output of a feature vector within one frame period after the end of the corresponding frame.

V. PROGRAM IMPLEMENTATION

This section describes several novel programming techniques that were required to implement the FXDSP. The most scarce resource was program memory; execution time and read/write memory were available in sufficient quantities. Therefore, most innovations were directed toward reducing the amount of program memory required at the expense of increasing execution time or read/write memory requirements. The specifics of program module size, execution time, and execution sequence are covered in Tables I and II.

One major problem, the negotiation between the input sample time scale of 150 μs and the output frame time of 15 ms, was solved by the program architecture discussed in the previous section.

A second problem was the Hamming window computation. Because of the frame size and overlap, each sample falls into the first third of one analysis frame, the second third of the previous analysis frame, and the final third of the twice previous frame. Additionally, after every 100 samples—when one of the three frames is completed—the relationship of the three analysis windows rotates cyclically. As a result, the Hamming window presented both the problem of producing

the cosine-based values and rearranging the segments of the window upon completing a frame.

In earlier implementations,[4,10] the Hamming window was stored as a table in program memory. In this implementation, program memory was too scarce, so a Taylor series expansion was used instead. Each third of the Hamming window (100 samples) was computed from a third-order Taylor series expansion about its midpoint (sample 50, 150, and 250). A comparison of the exact and approximate Hamming window is shown in Fig. 6 in both the time and frequency domain. The approximated window has been slightly shifted up to each comparison—its peak value is actually identical to the peak of the exact window.

To conserve program memory, several pieces of program modules were shared for multiple functions, sometimes with multiple exit points. For example, to perform the division required by eq. (13), the reciprocal of the energy E was calculated. An efficient reciprocal routine developed by Daugherty[16] was used, but required that the number for which the reciprocal was being formed be between 1 and 2. To build a general-purpose reciprocal routine, the number was first normalized to fall within the desired range. The reciprocal was readjusted to its true value to compensate for the normalization. The reciprocal normalization is the same operation as the amplitude nor-

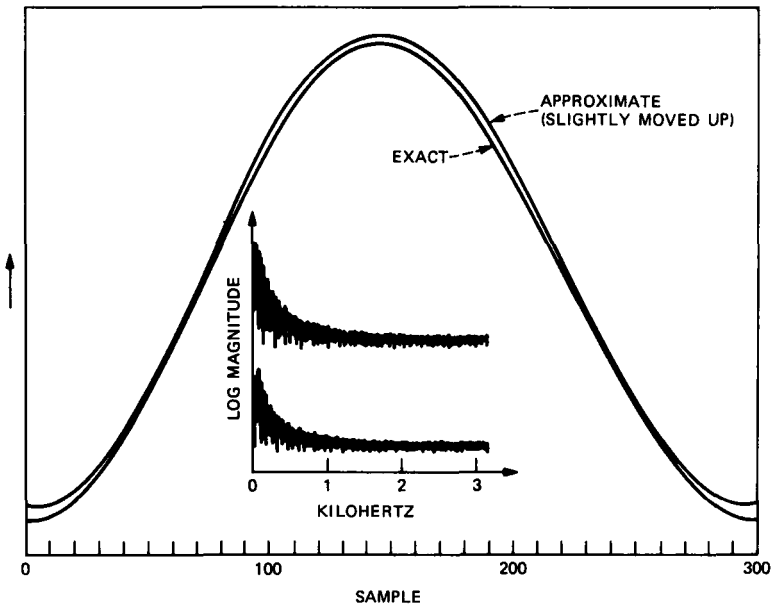

Fig. 6—Comparison of exact and Taylor series approximation of Hamming window.

malization and log energy calculation performed at LPC(1) [eq. (10)], and the same program performs both functions. However, for amplitude normalization, the program is exited before the reciprocal calculation is executed. Thus, the reciprocal routine, which requires 181 program locations, shares 99 of these locations with the gain normalization program, saving on overall program space.

An important way to conserve program space was the development of a means of computing Durbin's recursion with a common piece of code for all orders, $i = 1, 2, \cdots, 8$. Although the recursion is readily executed as a subroutine in microprocessor and Fortran implementations, it is difficult to perform as a subroutine in a programmable signal processor. This is because a DSP does not allow enough addressing capability to handle the two-dimensional array of α and the one-dimensional arrays of k, E, and R. A DSP typically provides only indirect addressing with the ability to increment one of two or three pointer registers by a fixed amount. An implementation of Durbin's recursion, if strung out, requires 536 program locations (not including the reciprocal calculation). With the iteration-independent form used here, that figure drops to 119 program locations.

By careful assignment of memory locations and proper sequencing through the arrays α, k, E, and R, all address calculation was rendered to be sequential within one iteration, that is, only in increments or decrements of one location.[17] This type of address sequencing is within the capability of the signal processor, and makes possible the single subroutine for all iterations. This allowed the frame-recursion program and the sample update program to fit together in the 1024 locations of program memory.

The LPC test coefficients $V_l(m)$ produced by the recursion are scaled by a power of 2 before output to obtain $\tilde{V}_l(m)$:

$$\tilde{V}_l(m) = 2^{-n(m)} \cdot V_l(m), \tag{20}$$

where

$$n(m) = 0; \quad m = 0, 1, 2, 3 \tag{21}$$

$$= 1; \quad m = 4, 5 \tag{22}$$

$$= 2; \quad m = 6 \tag{23}$$

$$= 3; \quad m = 7 \tag{24}$$

$$= 4; \quad m = 8. \tag{25}$$

This scaling is to compensate for a scaling performed on reference coefficients by a factor of $2^{n(m)}$ to allow each reference coefficient to be represented in 12 bits of memory. The values $n(m)$ are based on statistical analysis of the dynamic range of reference coefficients.[12]

To conclude the examination of program implementation, it is important to examine the arithmetic precision used in the signal processing. The μ-law speech is immediately converted to 13-bit linear encoding and is multiplied by 32 to attain an 18-bit word length. All sample update processing before autocorrelation—that is, eqs. (5) through (8)—is performed with 16 bits of precision, with the only approximation being introduced by the Taylor series expansion of the Hamming window. The autocorrelation calculation, eq. (9), is performed with 34 bits of precision, which represents full accuracy for the 13-bit speech samples. Double-precision storage is used on the 34-bit autocorrelation vectors.

A completed frame of autocorrelation vectors is normalized and then truncated to 15 significant bits [eqs. (10) and (11)]. This allows the remaining LPC recursion to be computed on single-precision data. Fifteen-bit precision has been shown to be adequate for fixed-point implementation of Durbin's recursion.[18] The LPC recursion [eqs. (12) through (16)], including the reciprocal calculation, is computed to at least 16 bits of precision. Often, for computations such as the accumulation of sums, eq. (13), the full 40-bit accumulator is used before rounding the sum to the single-word size.

As a result of maintaining full precision throughout the calculation, the difference between the LPC calculation, as computed by the FXDSP and as computed by full-precision floating point simulation, is minimal, as will now be described.

VI. COMPARISON WITH FLOATING POINT SIMULATION

To evaluate the performance of the FXDSP, a comparison of LPC feature measurement as calculated by the real-time FXDSP hardware was compared to LPC feature measurement as calculated by a floating point Fortran simulation running in non-real-time. The input to both routines was a common file of digitized speech, and final comparison was made using the log likelihood spectral distance used in speech recognition. This allowed relative comparison of errors introduced by the FXDSP to typical speech recognition scores.

The two-path program flow is shown in Fig. 7. Input at the left is a linear-encoded, 16-bit-per-sample speech file that had been band-limited to 3.2 kHz and sampled at 6667 samples/s. The program module FORMAT produced two speech files, one in format suitable for down loading into a DSPMATE—a hardware development tool for the AT&T Bell Laboratories DSP—and the other a standard integer speech file for Fortran simulation. Because the FXDSP is intended for use with a μ-law CODEC, one step in the DSPMATE formatting is the conversion of the speech file from linear to μ-law

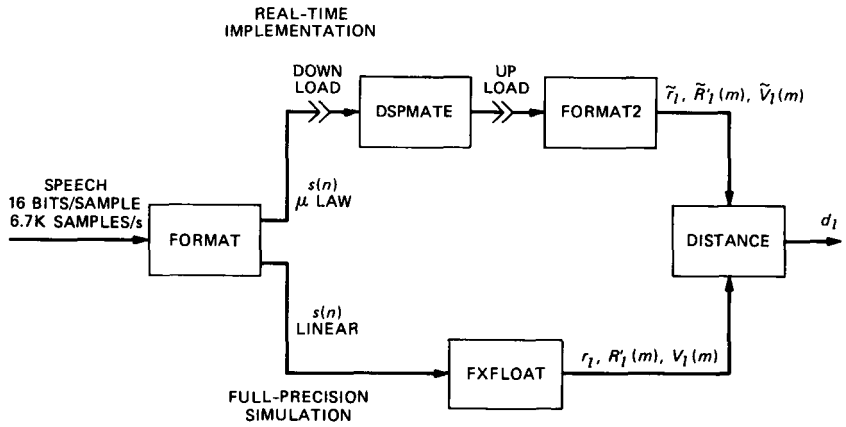

Fig. 7—Program module architecture for FXDSP verification.

encoding. The FXDSP immediately converts each sample back from μ-law to linear.

The upper path represents the route through the DSPMATE. The file capable of being down loaded is sent to the DSPMATE, where it is presented as file input in real time to a DSP chip running the LPC feature-measurement program. The resulting outputs, consisting of log energy $\tilde{r}_l$, gain-normalized autocorrelations $\tilde{R}'_l(m)$, and LPC test-pattern coefficients $\tilde{V}_l(m)$, $m = 0, 1, \cdots, 8$, are then up loaded, reformatted for Fortran simulation (FORMAT2), and input to a log likelihood distance computation program (DIST). The tilde over a quantity indicates that it was calculated by the FXDSP.

The lower route from FORMAT is passed through a floating point computation (FXFLOAT) that produces the values of r_l, $R'_l(m)$, and $V_l(m)$ in a file that is in a format identical to that produced by FORMAT2.

Program DIST computes the log likelihood distance of Itakura[1] for test coefficients produced by the FXDSP and reference coefficients produced by the floating point simulation. Reference coefficients $F_l(m)$ are produced from the LPC coefficients of eq. (18) as follows:

$$F_l(0) = \sum_{j=0}^{8} a_j^2 \tag{26}$$

$$F_l(m) = 2 \cdot 2^{n(m)} \cdot \sum_{j=0}^{8-m} a_j a_{j+m}; \qquad m = 1, 2, \cdots, 8. \tag{27}$$

The values of $n(m)$ are given in eqs. (21) through (25).

The distance calculated by DIST is for test and reference frames taken from the same sequence of speech samples and is given by

$$d_l = \log \sum_{i=0}^{8} F_l(i) \cdot \tilde{V}_l(i). \tag{28}$$

For test and reference coefficients computed with full precision from the same speech samples, $d_l = 0$.

The comparison was performed on 161 frames of speech taken from spoken digits. The dynamic range of the speech was 38 dB.

In Fig. 8a, a histogram of distances computed according to eq. (28) is displayed. The negative distances are a normal result of taking the log of a quantity that is slightly less than 1 due to round-off error. The average of the distances is 0.021.

This distance is negligibly small compared with the distances associated with the variation in word pronunciation shown by scores for correct word recognition. In Fig. 8b, the error histogram of Fig. 8a is overlaid on the histogram for correct word recognition using the same

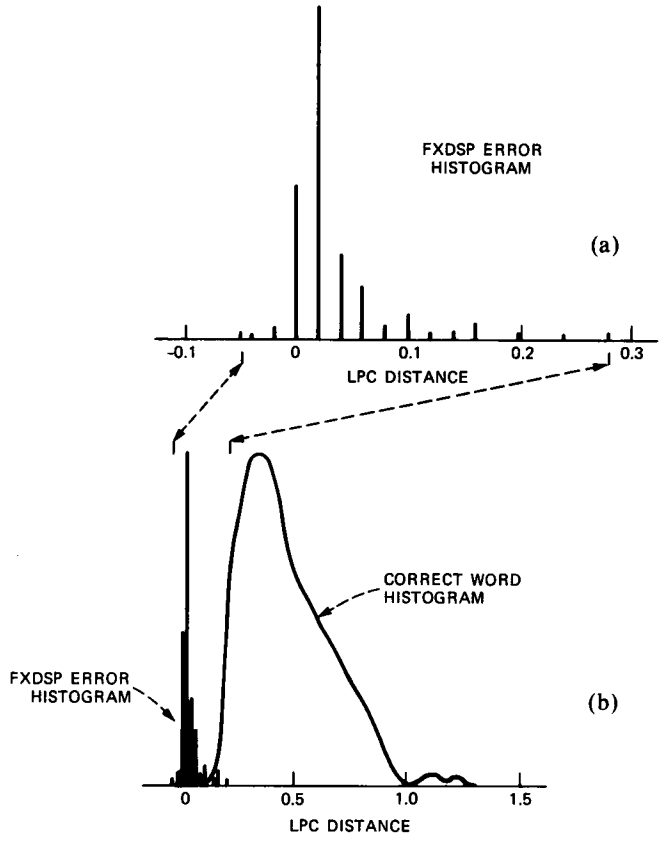

Fig. 8—Comparison of LPC distance from FXDSP to distance of correct word matches.

distance measurement.[19] The average of the correct recognition scores is about 0.45.

The distance of the FXDSP calculation from the true floating point computation is significantly less than the distance arising from variation when a given analog waveform is digitized at randomly varying phase. This distance, measured by repetitively playing taped speech with a professional quality recorder into a digital speech recognizer, averages about 0.035.[20]

The major source of error between floating and FXDSP computation arises from the linear-to-μ-law-to-linear conversion that is performed on the FXDSP path through Fig. 7, but not on the floating point path. Table III shows a sequence of particularly large distances that contributed to Fig. 8 in the left column. In the right column are the much smaller distances that result from performing linear-to-μ-law conversion, followed by μ-law-to-linear conversion, on the speech at a point immediately preceding the floating point LPC analysis (FXFLOAT). The average distance here drops from 0.09 to 0.012. A preliminary investigation on more speech frames suggests that about 75 percent of the distance between floating point simulation and FXDSP implementation is because of the linear-to-μ-to-linear conversion.

VII. SUMMARY

A single-chip basic building block for LPC-based connected- and isolated-word recognition systems has been described. The single chip is an appropriately programmed digital signal processor of AT&T Bell Laboratories.

Because the major limitation in attaining single-chip implementation was the amount of program memory available, several novel programming techniques were used to conserve program memory. These included (1) development of a program architecture that interleaved a background mainframe inversion program with a foreground

Table III—Comparison of FXDSP-to-floating point distances—with and without linear-μ-law-linear conversion in floating point computation

Frame	Without	With
1	0.100	0.013
2	0.147	0.008
3	0.055	0.035
4	0.226	0.008
5	0.052	0.020
6	0.010	0.001
7	0.009	0.002
AVG	0.090	0.012

sample update program, (2) development of a form of Durbin's recursion suitable for implementation as an iteration-independent subroutine, (3) use of overlaid subprograms with multiple exit points, and (4) use of a Taylor series expansion, rather than a look-up table, to store and permute segments of a Hamming window.

Comparison with numerical simulations shows that the error introduced by the implementation is negligible. This good match renders the chip suitable for use in systems that use quantities calculated in floating point on general-purpose computers, such as statistically clustered templates or frames for speaker-independent work recognition or for recognition based on vector quantization or hidden Markov modeling.

REFERENCES

1. F. Itakura, "Minimum Prediction Residual Principle Applied to Speech Recognition," IEEE Trans. Acoust., Speech, Signal Processing, *ASSP-23* (February 1975), pp. 67–72.
2. B. Aldefeld et al., "Automated Directory Listing Retrieval System Based on Isolated Word Recognition," Proc. IEEE, *68*, No. 11 (November 1980), pp. 1364–79.
3. C. S. Myers and L. R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition," IEEE Trans. Acoust., Speech, Signal Processing, *ASSP-29* (April 1981), pp. 284–97.
4. J. G. Ackenhusen and L. R. Rabiner, "Microprocessor Implementation of an LPC-Based Isolated Word Recognizer," Proc. IEEE ICASSP-81 (1981), pp. 746–9.
5. L. R. Rabiner, M. M. Sondhi, and S. E. Levinson, "A Vector Quantizer Incorporating Both LPC Shape and Energy," Proc. IEEE ICASSP-84 (1984), pp. 17.1.1–4.
6. S. C. Glinski, "On the Use of Vector Quantization for Connected-Digit Recognition," AT&T Tech. J., *64*, No. 5 (May–June 1985), pp. 1033–45.
7. L. R. Rabiner, S. E. Levinson, and M. M. Sondhi, "On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent, Isolated Word Recognition," B.S.T.J., *62*, No. 4 (April 1983), pp. 1075–105.
8. B. A. Dautrich, L. R. Rabiner, and T. B. Martin, "On the Effect of Varying Filterbank Parameters on Isolated Word Recognition," IEEE Trans. Acoust., Speech, Signal Processing, *ASSP-31* (August 1983), pp. 793–807.
9. Special Issue on the Digital Signal Processor, B.S.T.J., *60*, No. 7, Pt. 2 (September 1981), pp. 1431–709.
10. J. W. Daugherty, unpublished work.
11. T. Schalk and M. McMahan, "Firmware-Programmable μC Aids Speech Recognition," Electron. Des., *30* (July 22, 1982), pp. 143–7.
12. L. R. Rabiner, J. G. Wilpon, and J. G. Ackenhusen, "On the Effects of Varying Analysis Parameters on an LPC-Based Isolated Word Recognizer," B.S.T.J., *60*, No. 6 (July–August 1981), pp. 893–911.
13. Y. H. Oh et al., "Architecture for a Real-Time LPC-Based Feature Measurement Integrated Circuit," Proc. IEEE ICASSP-84 (1984), pp. 25B.2.1–4; also B. P. Tao and M. Oijala, "Architecture for a VLSI Implementation of an LPC-Based, Isolated Word Recognition System," Proc. IEEE ICASSP-84 (1984), pp. 34B.5.1–4.
14. L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, N. J.: Prentice Hall, Inc., 1978.
15. L. R. Rabiner and J. G. Wilpon, "A Simplified, Robust Training Procedure for Speaker-Trained, Isolated Word Recognition Systems," J. Acoust. Soc. Amer., *68*, No. 5 (November 1980), pp. 1271–6.
16. J. W. Daugherty, unpublished work.
17. J. G. Ackenhusen, unpublished work.
18. J. D. Markel and A. H. Gray, *Linear Prediction of Speech*, Berlin: Springer-Verlag, 1976.
19. M. K. Brown and L. R. Rabiner, "On the Use of Energy in LPC-Based Recognition of Isolated Words," B.S.T.J., *61*, No. 10 (December 1982), pp. 2971–87.
20. K. L. Shipley, unpublished work.

AUTHORS

John G. Ackenhusen, B.S. (Physics), B.S.E. (Nuclear Engineering), M.S. (Physics), M.S.E. (Nuclear Engineering), 1976; Ph.D. (Nuclear Engineering), 1977, University of Michigan; AT&T Bell Laboratories, 1978—. After serving as Interim Director of the University of Michigan Laser Plasma Interaction Laboratory, Mr. Ackenhusen joined AT&T Bell Laboratories in 1978 and began working in the field of optics for lightwave communications systems. His present activity in computer speech recognition began in 1979 with his interest in real-time hardware for speech recognition, in which he designed the first special-purpose computer for performing speech recognition using the computationally demanding techniques of linear predictive coding and dynamic time warping. In 1981 he became Supervisor of the Speech Recognition Group, where he leads an effort concerned with the development of efficient algorithms, hardware, software, and silicon for real-time speech recognition. Senior Member, IEEE. Member, ASSP Technical Committees on Speech and VLSI, ASSP Conference Board.

Young Hwan Oh, B.S., 1971, M.S., 1974, Ph.D., 1974 (Electrical Engineering), University of New Mexico; M.B.A. Program, 1971–1972; GTE Automatic Electric Laboratories, 1978–1981; AT&T Bell Laboratories, 1981–1984; Texas Instruments, 1984—. While at GTE Automatic Electric Laboratories, Mr. Oh was involved in hardware design, debugging, and testing for an I/O module for the GTD5-EAX, Class 5 Digital End Office Switching System. Also, he was a responsible engineer for converting the analog Dual-Tone Multifrequency (DTMF) receiver to the digital DTMF receiver for No. 5 application. At AT&T Bell Laboratories, he worked in the Speech Recognition Group, where he was involved in speech synthesis and recognition projects. In 1984 he joined Texas Instruments Advanced Technology Laboratory, where he is Director of the Speech Processing Laboratory. His dissertation was in the area of digital filtering and performance analysis. Member, IEEE, Tau Beta Pi.