Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review

Stanley A. White

DISTRIBUTED ARITHMETIC (DA) is so named because the arithmetic operations that appear in signal processing (e.g., addition, multiplication) are not "lumped" in a comfortably familiar fashion ("Aha, there's the multiplier over there," etc.), but are distributed in an often unrecognizable fashion. The most-often encountered form of computation in digital signal processing is a sum of products (or in vector analysis parlance, dot-product, or inner-product generation). This is also the computation that is executed most efficiently by DA.

Our motivation for using DA is its extreme computational efficiency. The advantages are best exploited in circuit design, but off-the-shelf hardware often can be configured effectively to perform DA. By careful design one may reduce the total gate count in a signal processing arithmetic unit by a number seldom smaller than 50 percent and often as large as 80 percent.

HISTORICAL PERSPECTIVE OF DISTRIBUTED ARITHMETIC

The first widely known description of DA was given at a presentation by Abraham Peled and Bede Liu on IIR digital filter mechanization in 1974 at the Arden House Workshop on Digital Signal Processing. Their work on both FIR and IIR digital filter mechanization was also published in the *IEEE ASSP Transactions* [1,2]. Earlier work (pre-1971) on DA had been performed in France by Croisier et al. [3]. The earliest documented work in the U.S. was done by Zohar [4,5,6], who had independently invented DA in 1968. Other early work in the United States was reported by Little [7]. From the Arden House description of DA, Bona and Terhan at Rockwell International designed an integrated-circuit DA compensator for control systems applications [8], and White generalized its control system application [9]. The April 1975 special digital-signal processing issue of *IEEE Proceedings* contained an article by Freeny on DA applications to the telephone system at Bell Laboratories [10] and an article by White on general vector dot-product formation using DA [11]. Later that year White et al. developed an AGM digital autopilot based on DA [12], Classen et al. at Philips in the Netherlands described communications systems applications [13], and Büttner and Schüessler at the University of Erlangen in Germany showed how to reduce the memory requirements [14]. In 1975, Kai-Ping Yiu at Hewlett-Packard showed a convenient rule for handling the sign bit [15]; H. Schröder of Siemens in Munich [16] and C. S. Burrus of Rice University [17] have given some suggestions and insight for speeding up the algorithms, and K. D. Kammeyer gave a survey/summary [18]. Mechanization studies on application of DA to digital filters were discussed by Burrus [19], Jenkins and Leon [20], Zeman and Nagel [21], Tam and Hawkins [22], Arjmand and Roberts [23] and White [24,25]. Kammeyer [26] and Taylor [27] have presented error analyses, Smith and White [28] have considered DA for coordinate transformations, and Burleson and Scharf have applied it to a rotator array [29]. Taylor applied DA to a Gray-Markel filter [30], and Zohar discussed a VLSI implementation of a correlator/filter [31]. DA is also discussed by Taylor in his text [32], in the text by Smith and Denyer [33], and by Mintzer in Elliott's handbook [34].

TECHNICAL OVERVIEW OF DISTRIBUTED ARITHMETIC

DA is basically (but not necessarily) a bit-serial computational operation that forms an inner (dot) product of a pair of vectors in a single direct step. The advantage of DA is its efficiency of mechanization. A frequently argued disadvantage has been its apparent slowness because of its inherent bit-serial nature. This disadvantage is not real if the number of elements in each vector is commensurate with the number of bits in each vector element, e.g., the time required to input eight 8-bit words one at a time in a parallel fashion is exactly the same as the time required to input (simultaneously on eight wires) all eight words serially. Other modifications to increase the speed may be made by employing techniques such as bit pairing or partitioning the input words into the most significant half and least significant half, the least significant half

0740-7467/89/0700-0004$1.00 © 1989 IEEE

of the most significant half, etc., thereby introducing parallelism in the computation. This will be described in Section 4.

As an example of direct DA inner-product generation, consider the calculation of the following sum of products:

$$y = \sum_{k=1}^{K} A_k x_k . \tag{1}$$

The A_k are fixed coefficients, and the x_k are the input data words. If each x_k is a 2's-complement binary number scaled (for convenience, not as necessity) such that $|x_k| < 1$, then we may express each x_k as

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \tag{2}$$

where the b_{kn} are the bits, 0 or 1, b_{k0} is the sign bit, and $b_{k,N-1}$ is the least significant bit (LSB).

Now let us combine Equations 1 and 2 in order to express y in terms of the bits of x_k:

$$y = \sum_{k=1}^{K} A_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right]. \tag{3a}$$

Equation 3a is the conventional form of expressing the inner product. Direct mechanization of this equation defines a "lumped" arithmetic computation. Let us instead interchange the order of the summations, which gives us:

$$y = \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0}) . \tag{3b}$$

This is the crucial step: Equation 3b defines a distributed arithmetic computation. Consider the bracketed term in Equation 3b:

$$\sum_{k=1}^{K} A_k b_{kn} . \tag{3c}$$

Because each b_{kn} may take on values of 0 and 1 only, expression (3c) may have only 2^K possible values. Rather than compute these values on line, we may precompute the values and store them in a ROM. The input data can be used to directly address the memory and the result, i.e., the $\sum_{k=1}^{K} A_k b_{kn}$, can be dropped into an accumulator. After N such cycles, the memory contains the result, y.

As an example, let $K = 4$, $A_1 = 0.72$, $A_2 = -0.30$, $A_3 = 0.95$, and $A_4 = 0.11$. The memory must contain all possible combinations ($2^4 = 16$ values) *and* their negatives in order to accommodate the term

$$\sum_{k=1}^{K} A_k (-b_{k0}) \tag{3d}$$

which occurs at the sign-bit time. As a consequence, we need to use a $2 \cdot 2^K$ word ROM. Figure 1a shows the simple structure (with a $2 \times 2^4 = 32$-word ROM) that can be used to mechanize these equations; Table 1 shows the contents of the memory. The T_s signal is the sign-bit timing signal. We assume that the data on the x_1, x_2, x_3, and x_4 lines (which with T_s comprise the ROM address words) are serial, 2's-complement numbers. Each is delivered in a one-bit-at-a-time (1BAAT) fashion, with LSBs $\{b_{k,N-1}\}$ first. The sign bits $\{b_{k0}\}$ are the last bits to arrive. The clock period in which the sign bits all simultaneously arrive is called the "sign-bit time." During the sign-bit time the control signal $T_s = 1$, otherwise $T_s = 0$. For the moment we will assume essentially zero time delay between the time of arrival of the address pattern to the ROM and the availability of its output. The delay around the accumulator loop is assumed to be one clock cycle and is concentrated in the summer. Switch SWA remains in Position 1 except during the clock cycle that follows the sign-bit time, when it toggles for one clock cycle to Position 2, and the fully formed result is output.

We may reduce the memory size by half to a 2^K word ROM by modifying the adder to an adder/subtractor and using T_s as the add/subtract-control line as shown in Figure 1b. This configuration may now be mechanized with a 16-word ROM. The stored table is simply the upper half of Table 1.

The memory size may be halved again to $\frac{1}{2} 2^K$ words. In order to understand how this works, we shall interpret (*not* convert, but just interpret) the input data as being cast not in a (0,1) straight binary code, but instead as being cast in a (−1,1) offset binary code. Suppose that we think of x_k as

$$x_k = \frac{1}{2}[x_k - (-x_k)], \tag{4}$$

and remember that in 2's-complement notation the negative of x_k is written as

$$-x_k = -\overline{b}_{k0} + \sum_{n=1}^{N-1} \overline{b}_{kn} 2^{-n} + 2^{-(N-1)} \tag{5}$$

where the overscore symbol indicates the complement of a bit. From Equations 2 and 5 we may rewrite Equation 4 as:

$$x_k = \frac{1}{2} \left[-(b_{k0} - \overline{b}_{k0}) + \sum_{n=1}^{N-1} (b_{kn} - \overline{b}_{kn}) 2^{-n} - 2^{-(N-1)} \right]. \tag{6}$$

In order to simplify our notation later, it is convenient to define the new variables

$$c_{kn} = b_{kn} - \overline{b}_{kn} \qquad n \neq 0 \tag{7}$$

and

$$c_{k0} = -(b_{k0} - \overline{b}_{k0}) \tag{8}$$

where the possible values of the c_{kn}, including $n = 0$, are ± 1. Now (6) may be rewritten as

$$x_k = \frac{1}{2} \left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right]. \tag{9}$$

By substituting (9) into (1) we obtain

$$y = \frac{1}{2} \sum_{k=1}^{K} A_k \left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right] \tag{10}$$

$$= \sum_{n=0}^{N-1} Q(b_n) 2^{-n} + 2^{-(N-1)} Q(0) \tag{11}$$

where

$$Q(b_n) = \sum_{k=1}^{K} \frac{A_k}{2} c_{kn} \quad \text{and} \quad Q(0) = \sum_{k=1}^{K} \frac{A_k}{2}. \tag{12}$$

Notice that $Q(b_n)$ has only $2^{(K-1)}$ possible amplitude values with a sign that is given by the instantaneous combination of bits. This is consistent with our earlier claim. The computation of y is mechanized using a $2^{(K-1)}$ word memory, a one-word initial condition register for $Q(0)$, and a single parallel adder/subtractor with the necessary control-logic gates. This is shown in Figure 1c, using the 8-word ROM, which contains the $Q(b_n)$.

Notice from the memory values of Table 2 that those values in the lower half under "Q" are the mirror image of the values in the upper half, but with the signs re-

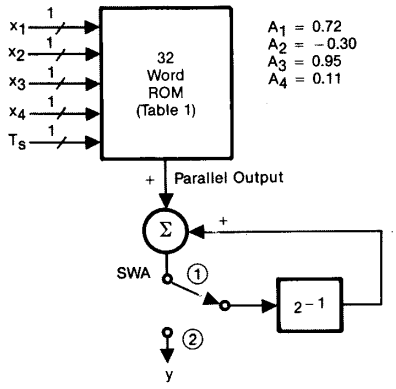

Figure 1a. Adder and Full Memory

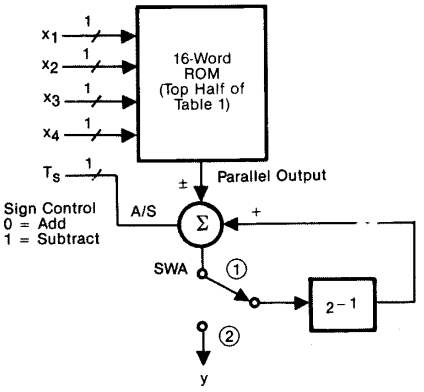

Figure 1b. Adder/Subtractor and Memory

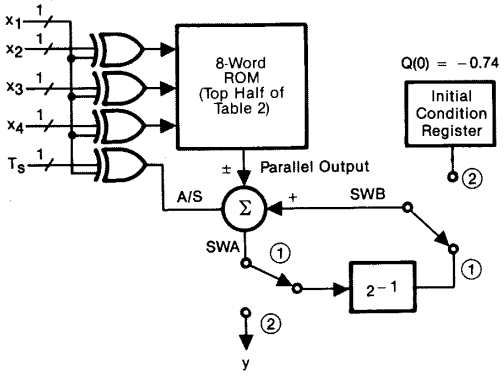

Figure 1c. Adder/Subtractor and Reduced Memory

Table 1

	Input Code T_s	b_{1n}	b_{2n}	b_{3n}	b_{4n}	32-Word Memory Contents
$1 \le n \le N-1$	0	0	0	0	0	0
	0	0	0	0	1	$A_4 = 0.11$
	0	0	0	1	0	$A_3 = 0.95$
	0	0	0	1	1	$A_3 + A_4 = 1.06$
	0	0	1	0	0	$A_2 = -0.30$
	0	0	1	0	1	$A_2 + A_4 = -0.19$
	0	0	1	1	0	$A_2 + A_3 = 0.65$
	0	0	1	1	1	$A_2 + A_3 + A_4 = 0.75$
	0	1	0	0	0	$A_1 = 0.72$
	0	1	0	0	1	$A_1 + A_4 = 0.83$
	0	1	0	1	0	$A_1 + A_3 = 1.67$
	0	1	0	1	1	$A_1 + A_3 + A_4 = 1.78$
	0	1	1	0	0	$A_1 + A_2 = 0.42$
	0	1	1	0	1	$A_1 + A_2 + A_4 = 0.53$
	0	1	1	1	0	$A_1 + A_2 + A_3 = 1.37$
	0	1	1	1	1	$A_1 + A_2 + A_3 + A_4 = 1.48$
$n = 0$	1	0	0	0	0	0
	1	0	0	0	1	$-A_4 = -0.11$
	1	0	0	1	0	$-A_3 = -0.95$
	1	0	0	1	1	$-(A_3 + A_4) = -1.06$
	1	0	1	0	0	$-A_2 = +0.30$
	1	0	1	0	1	$-(A_2 + A_4) = +0.19$
	1	0	1	1	0	$-(A_2 + A_3) = -0.65$
	1	0	1	1	1	$-(A_2 + A_3 + A_4) = -0.75$
	1	1	0	0	0	$-A_1 = -0.72$
	1	1	0	0	1	$-(A_1 + A_4) = -0.83$
	1	1	0	1	0	$-(A_1 + A_3) = -1.67$
	1	1	0	1	1	$-(A_1 + A_3 + A_4) = -1.78$
	1	1	1	0	0	$-(A_1 + A_2) = -0.42$
	1	1	1	0	1	$-(A_1 + A_2 + A_4) = -0.53$
	1	1	1	1	0	$-(A_1 + A_2 + A_3) = -1.37$
	1	1	1	1	1	$-(A_1 + A_2 + A_3 + A_4) = -1.48$

Table 2

Input Code b_{1n}	b_{2n}	b_{3n}	b_{4n}	8-Word Memory Contents, Q
0	0	0	0	$-1/2\,(A_1 + A_2 + A_3 + A_4) = -0.74$
0	0	0	1	$-1/2\,(A_1 + A_2 + A_3 - A_4) = -0.63$
0	0	1	0	$-1/2\,(A_1 + A_2 - A_3 + A_4) = 0.21$
0	0	1	1	$-1/2\,(A_1 + A_2 - A_3 - A_4) = 0.32$
0	1	0	0	$-1/2\,(A_1 - A_2 + A_3 + A_4) = -1.04$
0	1	0	1	$-1/2\,(A_1 - A_2 + A_3 - A_4) = -0.93$
0	1	1	0	$-1/2\,(A_1 - A_2 - A_3 + A_4) = -0.09$
0	1	1	1	$-1/2\,(A_1 - A_2 - A_3 - A_4) = 0.02$
1	0	0	0	$1/2\,(A_1 - A_2 - A_3 - A_4) = -0.02$
1	0	0	1	$1/2\,(A_1 - A_2 - A_3 + A_4) = 0.09$
1	0	1	0	$1/2\,(A_1 - A_2 + A_3 - A_4) = 0.93$
1	0	1	1	$1/2\,(A_1 - A_2 + A_3 + A_4) = 1.04$
1	1	0	0	$1/2\,(A_1 + A_2 - A_3 - A_4) = -0.32$
1	1	0	1	$1/2\,(A_1 + A_2 - A_3 + A_4) = -0.21$
1	1	1	0	$1/2\,(A_1 + A_2 + A_3 - A_4) = 0.63$
1	1	1	1	$1/2\,(A_1 + A_2 + A_3 + A_4) = 0.74$

Figure 1. DA mechanization of $y = A_1x_1 + A_2x_2 + A_3x_3 + A_4x_4$ for bit serial (1 BAAT) implementation.

versed. If we look at the bit patterns in the left-hand column, we discover that if we EXOR b_{in} with the remaining set of b_{2n}, b_{3n}, and b_{4n}, we properly address the 8-word memory to pull out the correct values of $Q\ldots$ except for the sign. By using the b_{1n} as the add/subtract control line for the accumulator input, we also now have the proper sign. During the sign-bit time the add/subtract command must be inverted. We therefore combine the b_{1n} and T_s signals through an EXOR gate in order to derive the proper add/subtract control signal.

The initial-condition memory that contains the value $Q(0)$ is shown on the extreme right side of Figure 1c. When the LSBs of the x_k are addressing the ROM, the value that is read out from the ROM must be corrected by the $Q(0)$ through switch SWB, which operates synchronously with switch SWA. This artifact of the binary-offset system can be seen in Equation 11. Subsequent values from the ROM are summed with the shifted previous sum. As before, we assume zero time delay between the application of the addressing bits and the availability of the contents of the ROM. There is a clock period of delay through the parallel adder, and the switches SWA and SWB are in Position 2 only for the clock cycle following the sign-bit time when $T_s = 1$. During the first clock cycle, the first output from the ROM, $Q(b_{N-1})$, is summed with $Q(0)$; during the second clock cycle it is right shifted and summed with $Q(b_{N-2})$ to produce $Q(b_{N-2}) + [Q(b_{N-1}) + Q(0)]2^{-1}$; during the third clock cycle it is again right shifted and summed with $Q(b_{N-3})$ to produce $Q(b_{N-3}) + Q(b_{N-2})2^{-1} + [Q(b_{N-1}) + Q(0)]2^{-2}$; up to the Nth clock cycle when we add $Q(b_0)$ to the right shifted previously computed quantity to produce $Q(b_0) + Q(b_1)2^{-1} + Q(b_2)2^{-2} + \cdots + Q(b_{N-2})2^{-(N-2)} + [Q(b_{N-1}) + Q(0)]^{2-(N-1)}$.

INCREASING THE SPEED OF DA MULTIPLICATION

One can see that ingesting the data serially, 1BAAT, results in a slow computation. If the input words are N bits in length, N clock cycles or periods are required in which to form the dot product. On the other hand, the equivalent of K separate products are being formed. If, therefore, $K > N$, the DA processor is faster than a single parallel multiplier/accumulator.

Additional speed may be bought in two ways; one at the expense of linearly increased memory plus more arithmetic operations, the other at the expense of exponentially increased memory. The speed may be increased by a factor of L by partitioning each input word into L subwords (L must be a factor of N). This effectively increases the dimension of the input vector by a factor of L. We can use L-times as many memories with an expanded-capacity accumulator for combining their results, or we can stay with a single memory, but its word capacity becomes $\frac{1}{2}2^{KL}$ and the lengths of the words grow by $\log_2 L$ bits. The first approach is obvious and is shown in Figure 2. The second is described below. Both are illustrated by example.

We seek a computationally simple means to introduce parallelism in the mechanization of Equation 11. The summation over n is next broken into two sums: the one over n sums from 0 to $(N/L) - 1$, and the second one sums l

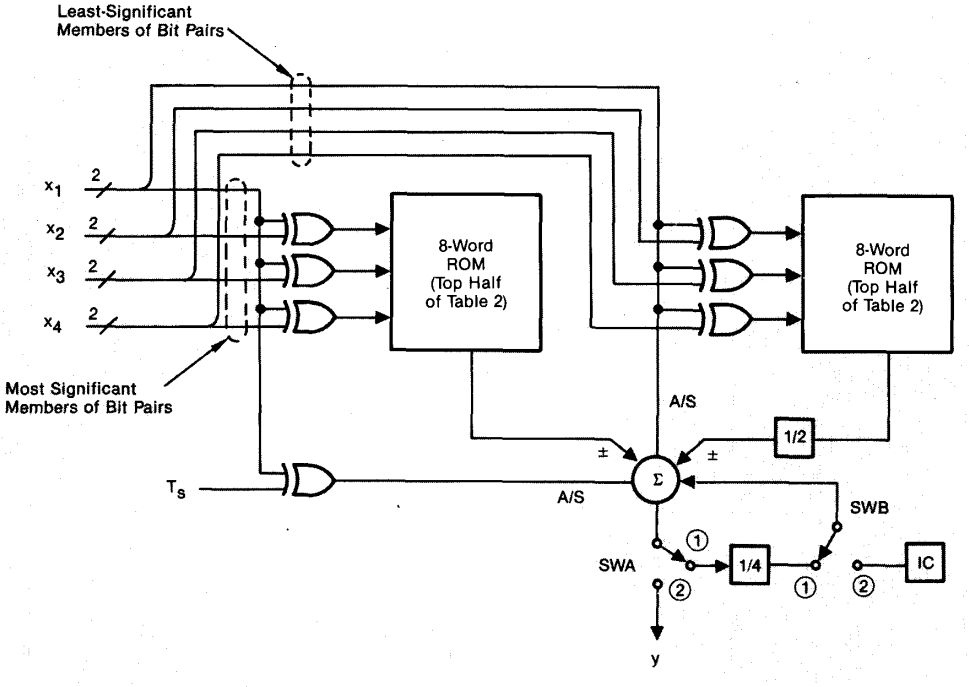

Figure 2. 2-Memory, 2 BAAT version of Figure 1C.

from 0 to $L-1$. Now Equation 11 becomes:

$$y = \left[2^{-(N-L)} P_{IC} + \sum_{n=0}^{(N/L)-1} P(b_k) 2^{-nL} \right] \tag{13}$$

where

$$P(b_k) = \sum_{k=1}^{K} \sum_{l=0}^{L-1} \frac{1}{2} A_k 2^{-l} c_{k,nL+l} \tag{14}$$

and

$$P_{IC} = -2^{-L} \sum_{k=1}^{K} A_k . \tag{15}$$

Only N/L rather than N clock times are required to form the inner product, therefore the reader can see that we have succeeded in increasing the speed of the computation by a factor of L.

We can parametrically explore the issue further. The total number of bits to be loaded is NK, and the number of clock periods required to read in these NK bits is N/L clocks. The number of input lines is

$$p = \frac{\text{total bits in}}{\text{number of clock periods}} = \frac{NK}{(N/L)} = KL . \tag{16}$$

There may be a relative cost

$$w = \frac{\text{importance of minimizing pins}}{\text{importance of minimizing time}} = \frac{(N/L)}{KL} = \frac{N}{KL^2} \tag{17}$$

We can solve this for L, the number of bits at a time (per input variable) that we are trying to load:

$$L = \left\lceil \sqrt{\frac{N}{wk}} \right\rceil . \tag{18}$$

The Gauss brackets tell us to round up to the next integer. DA is often most efficient when the number of input lines is commensurate with the number of clocks required to load the data, or equivalently, when $w = 1$. For our example

$$\sqrt{\frac{N}{wK}} = \sqrt{\frac{16}{1 \cdot 4}} = 2 ; \tag{19}$$

therefore, the data will be input 2 BAAT. Of each input bit pair, we identify the most significant bit (MSB) and the least significant bit (LSB). For all values of L, the gate at which the most significant bit appears receives special treatment. The T_s control signal is EXOR'd with the MSB for sign-bit time correction (because the sign bit is the MSB of the word). In Figure 3, we can see the configuration and also see that the LSB of the input pairs of x_1 controls the add/subtract line.

In order to demonstrate the validity of the approach, let us reconstruct by the same rules the structure for $L = 1$, as shown in Figure 4. Because for $L = 1$, a 1BAAT serial input line for each x_k uses the same line for the sign bit as for all other bits, all T_s correction must be EXOR'd with all x_k. The equivalence between Figures 1c and 4 should be easy for the reader to see because the ROM

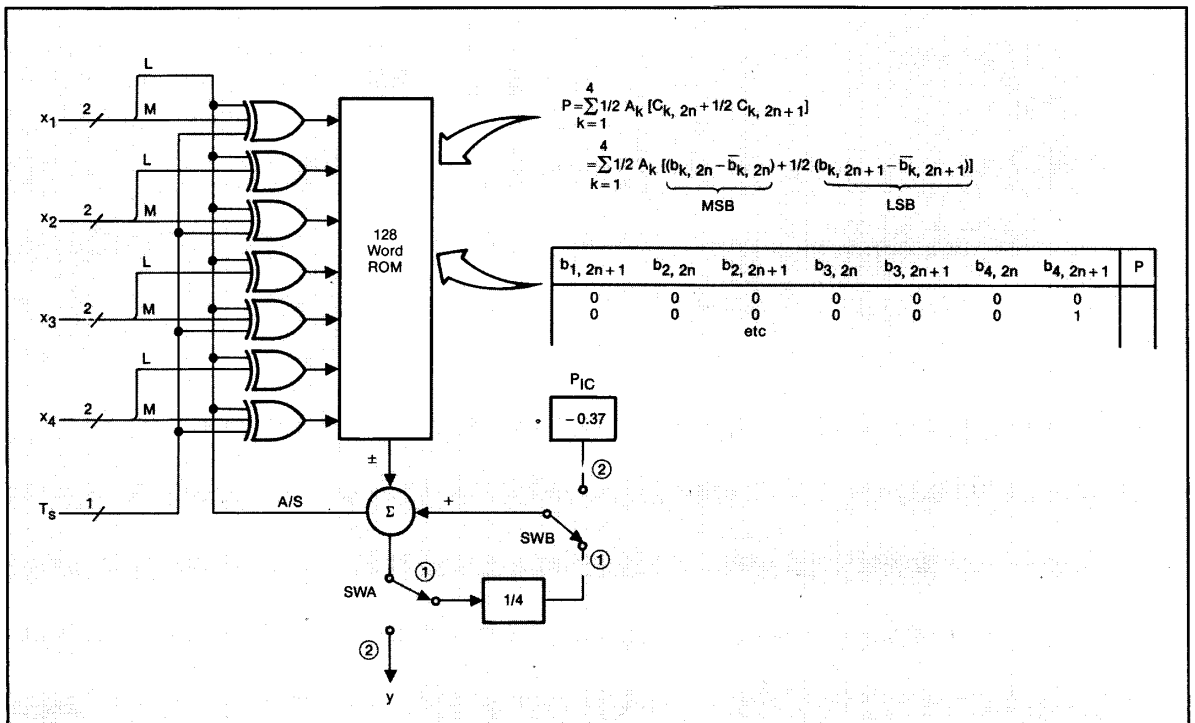

Figure 3. Single-memory, 2 BAAT version of Figure 1C.

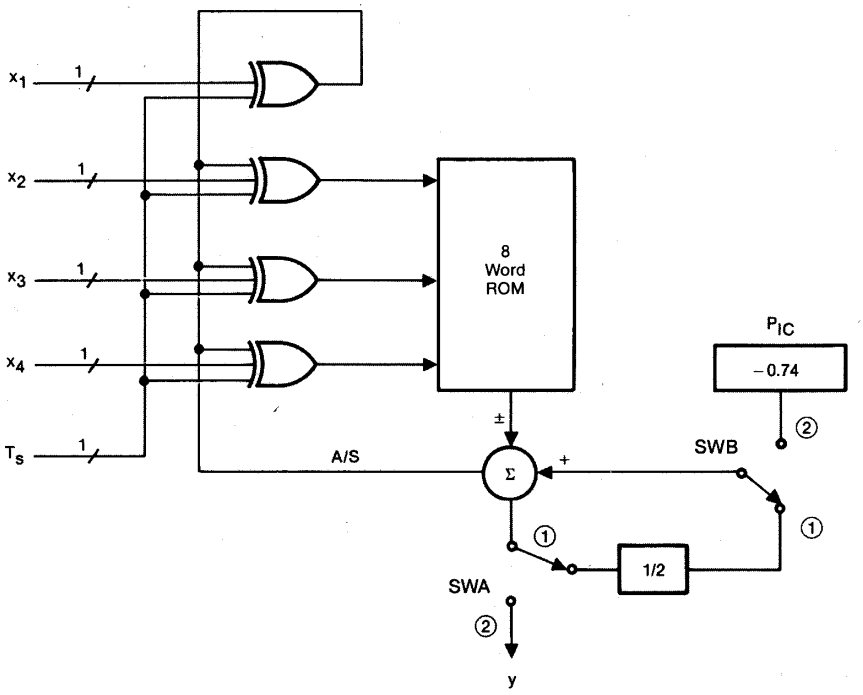

Figure 4. DA mechanization of $y = A_1x_1 + A_2x_2 + A_3x_3 + A_4x_4$ for 1 BAAT mechanization similar to Figure A3 showing alternate derivation of A/S (Add/Subtract) control.

address line in Figure 4 that is driven by x_k ($k = 2,3,4$) actually sees $x_k \oplus T_s \oplus (T_s \oplus x_1) = x_k \oplus x_1$. This is the same as in Figure 1c. The derivation of the A/S control lines is the same in both figures.

In Figure 5, we see a demonstration of an L = 4 design in order to show the structure. The memory cost is extremely great for such a small computation, but it is presented to illustrate the principle. Some more practical cases are shown below.

APPLICATION OF DA TO A BIQUADRATIC DIGITAL FILTER (AN EXAMPLE OF VECTOR DOT-PRODUCT AND VECTOR-MATRIX-PRODUCT MECHANIZATION)

A typical biquadratic digital filter has a transfer function of the form

$$\frac{Y(z)}{X(z)} = \frac{A_0 + A_1z^{-1} + A_2z^{-2}}{1 + B_1z^{-1} + B_2z^{-2}} \tag{20}$$

where the poles are determined by the B_1 and B_2, and the gain and zeros are determined by the A_0, A_1, and A_2. The time-domain description is

$$y_n = [A_0A_1A_2B_1B_2]^T[x_nx_{n-1}x_{n-2}y_{n-1}y_{n-2}] \tag{21}$$

where the coefficient vector is $[A_0A_1A_2B_1B_2]^T$ and the data vector is $[x_nx_{n-1}x_{n-2}y_{n-1}y_{n-2}]^T$. A direct DA mechanization of the filter is shown in Figure 6. Notice the extreme economy of this filter. This figure differs slightly from those that we have seen above. We have a 5-dimensional input vector to the addressing logic, but a scalar input, x_n, and a scalar output, y_n for the processor.

We have added a pair of delays (the z^{-1} blocks) in the input-signal path so that we could develop the delayed signals, x_{n-1} and x_{n-2} from the input, x_n; and we have added a third delay block so that we could develop the delayed output, y_{n-2}, from y_{n-1}. We obtain y_{n-1} from a parallel-to-serial register (P/S), since the output from the summer in the accumulator loop is a parallel word. The contents of the 16-word ROM are shown in Table 3. We may, of course, use parallelism to increase the speed, as was discussed in the previous section.

There is another important (because of low roundoff noise, low coefficient sensitivity and favorable limit-cycle behavior) form of digital filter (the normal form) that serves to illustrate the vector-matrix form of DA, and which we will now discuss. An excellent tradeoff study was recently presented by Barnes [35] to show how one could design a normal-form second order digital filter to meet prescribed performance criteria. In this section, an extremely efficient set of realizations is shown, one in which the speed and complexity can be effectively traded. (Much of the following is taken from References 36 and 37.)

A block diagram of the normal-form structure is shown in Figure 7. The multipliers b_1, b_2, c_1, and c_2 determine the pole locations of the filter. The input multipliers a_1 and a_2 are used to determine the input scaling and the multipli-

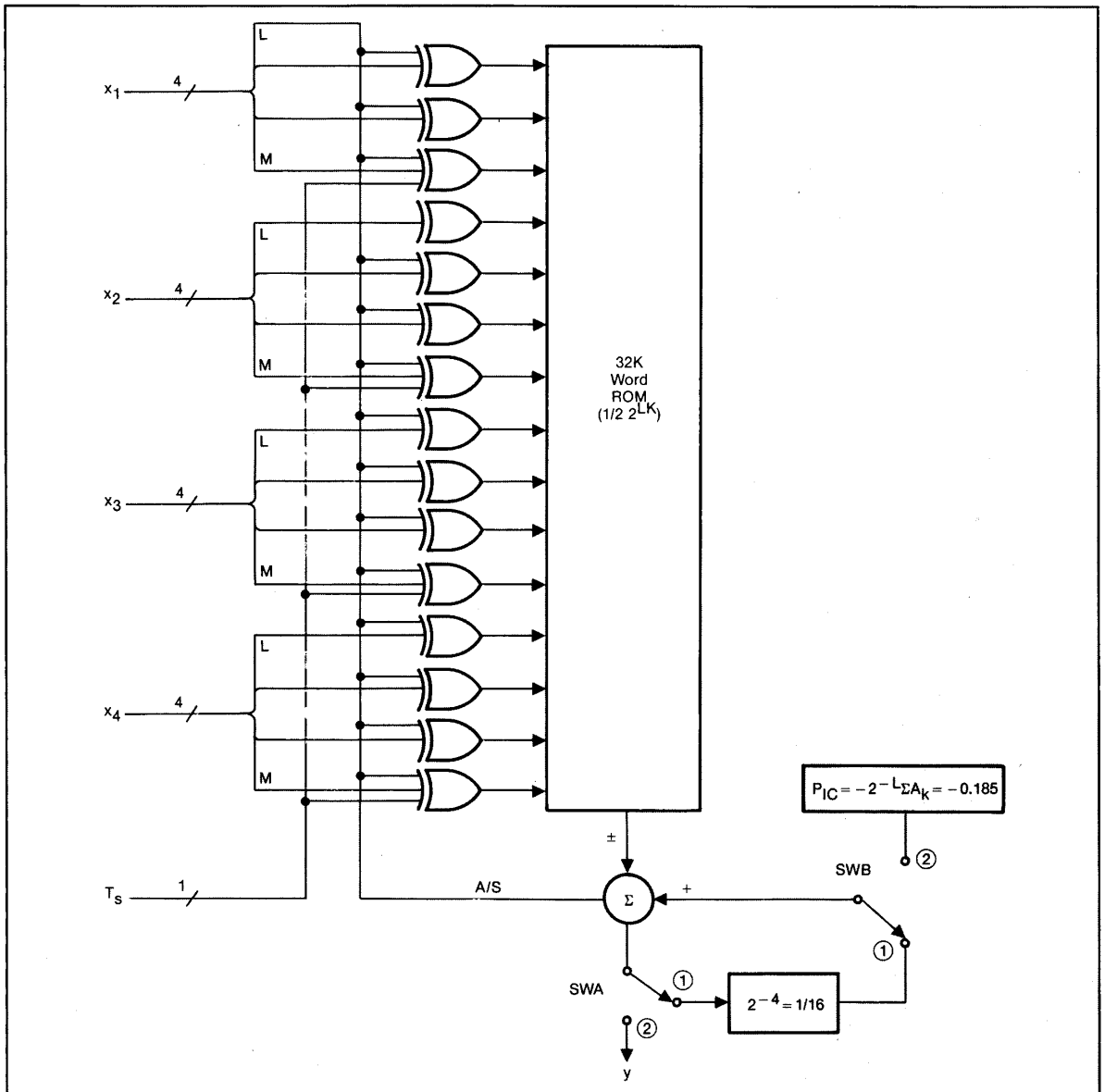

Figure 5. DA mechanization of $y = A_1x_1 + A_2x_2 + A_3x_3 + A_4x_4$ for 4 BAAT implementation using single Adder/Subtractor and minimum memory.

ers a_0, d_1, and d_2 to determine the zero locations of the filter. There are nine multipliers in total, as compared to the five that are required in the so-called direct mechanization. Barnes shows procedures whereby one may reduce this number of multipliers; however, we shall show how to eliminate them.

The vector matrix equation that describes the configuration of Figure 7 is given below:

$$\begin{bmatrix} u_n \\ v_n \\ y_n \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & c_2 & b_2 \\ a_0 & d_1 & d_2 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ u_{n-1} \\ v_{n-1} \end{bmatrix} \tag{22}$$

The relationships between Equations 20 and 22 are

$$\begin{aligned} A_0 &= a_0 \\ A_1 &= a_1d_1 + a_2d_2 - a_0(b_1 + b_2) \\ A_2 &= a_0(b_1b_2 - c_1c_2) + a_1(c_2d_2 - b_2d_1) \\ &\quad + a_2(c_1d_1 - b_1d_2) \\ B_1 &= b_1 + b_2 \\ B_2 &= -b_1b_2 + c_1c_2 . \end{aligned} \tag{23}$$

There are three common inputs to each of the "clumps" of Figure 7; so, for each output, we need to store $\frac{1}{2}2^3 = 4$

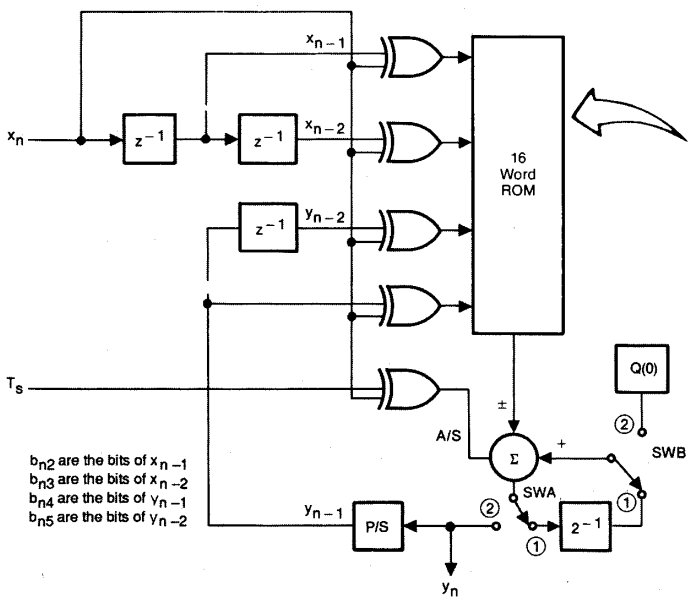

Table 3

Input Code b_{n2}	b_{n3}	b_{n4}	b_{n5}	Contents of 16-Word Memory, Q
0	0	0	0	$-1/2(A_0+A_1+A_2+B_1+B_2)$
0	0	0	1	$-1/2(A_0+A_1+A_2+B_1-B_2)$
0	0	1	0	$-1/2(A_0+A_1+A_2-B_1+B_2)$
0	0	1	1	$-1/2(A_0+A_1+A_2-B_1-B_2)$
0	1	0	0	$-1/2(A_0+A_1-A_2+B_1+B_2)$
0	1	0	1	$-1/2(A_0+A_1-A_2+B_1-B_2)$
0	1	1	0	$-1/2(A_0+A_1-A_2-B_1+B_2)$
0	1	1	1	$-1/2(A_0+A_1-A_2-B_1-B_2)$
1	0	0	0	$-1/2(A_0-A_1+A_2+B_1+B_2)$
1	0	0	1	$-1/2(A_0-A_1+A_2+B_1-B_2)$
1	0	1	0	$-1/2(A_0-A_1+A_2-B_1+B_2)$
1	0	1	1	$-1/2(A_0-A_1+A_2-B_1-B_2)$
1	1	0	0	$-1/2(A_0-A_1-A_2+B_1+B_2)$
1	1	0	1	$-1/2(A_0-A_1-A_2+B_1-B_2)$
1	1	1	0	$-1/2(A_0-A_1-A_2-B_1+B_2)$
1	1	1	1	$-1/2(A_0-A_1-A_2-B_1-B_2)$

Figure 6. DA implementation of direct form of biquadratic digital filter (1 BAAT).

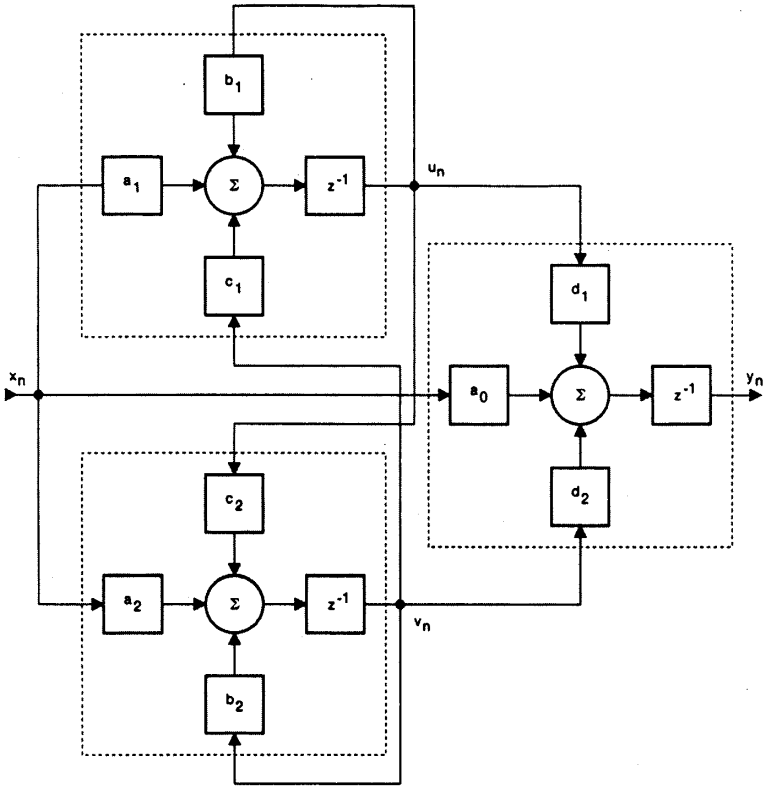

Figure 7. Normal form biquadratic filter organized into three clumps of identically driven inner products.

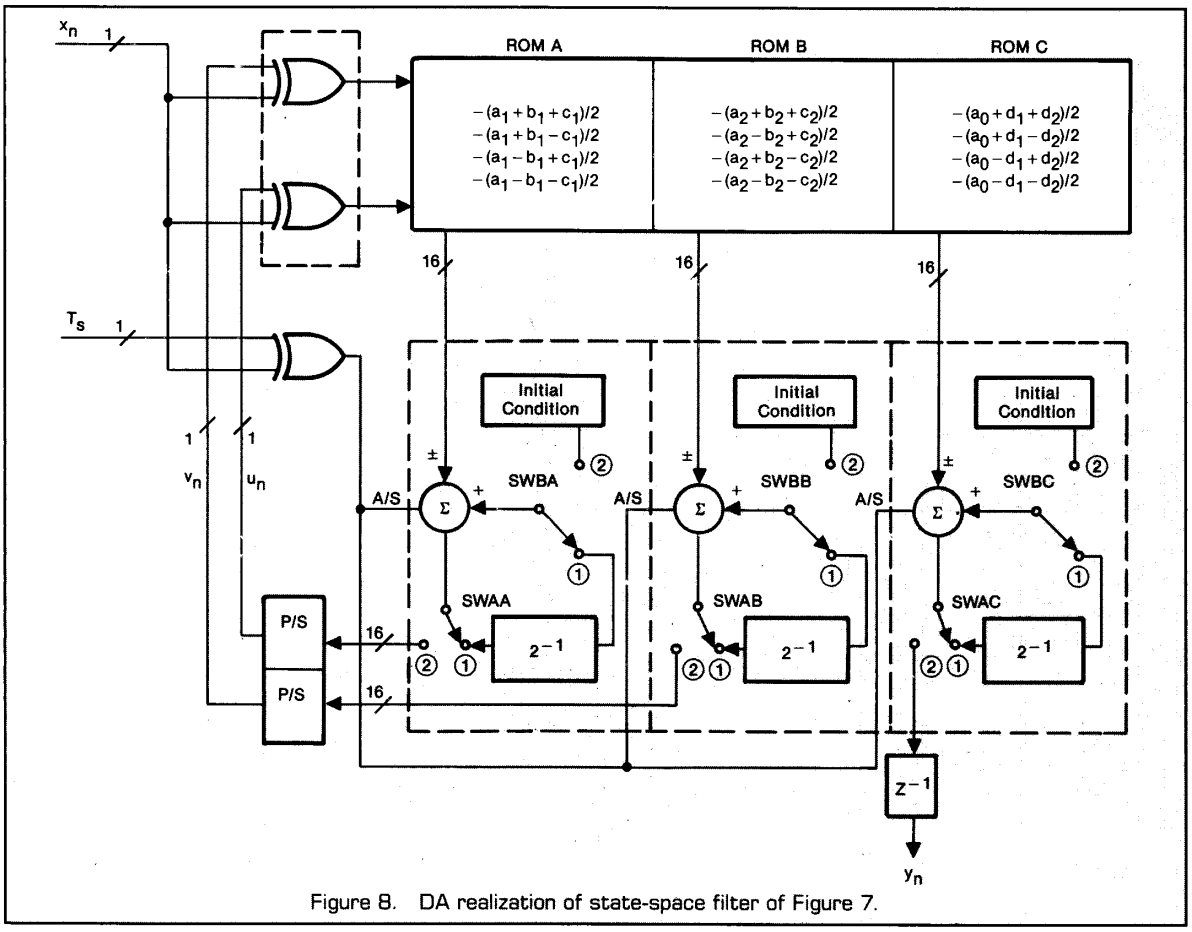

Figure 8. DA realization of state-space filter of Figure 7.

words. If the words that are stored are 16 bits long, then our three outputs together call for $3 \times 16 = 48$ bits per stored word. The total number of bits stored, however, is a modest $4 \times 48 = 192$. The somewhat detailed DA realization is illustrated in Figure 8. Notice how the addressing section has reduced to a pair of EXOR gates. The 4-word by 48-bit memory is shown for clarity as comprising three memories. In fact, the ROM may physically consist of three identically addressed 4-word by 16-bit memories or a single 4-word extended length (48-bit) memory. Each 16-bit output segment drives a separate accumulator loop, each with its own initial-condition register, just as we encountered earlier. The add/subtract control line is common, and the outputs of two of the accumulator loops are converted to serial form in their parallel-to-serial registers to be fed back to the memory addressing gates.

In order to simplify our subsequent development, it will be useful to redraw Figure 8 as shown in Figure 9. We can see the essence and utter simplicity of the structure, which is somewhat startling when one realizes that this is a realization of the 9-multiplier configuration of Figure 7.

Figure 10 shows a factor-of-four speedup over the circuit of Figure 9 by using the data 4 bits at a time (4BAAT). The parallel-to-serial registers shown do not output a serial data stream, but rather provide a sequence of four 4-bit wide segments. The time required to perform the filtering function has been reduced to 4 clock periods. This increase in speed demands that the ROM size be increased to $(\frac{1}{2})\ (2^3)^4 = 2048$ words. One would like to be able to make the throughput rate equal to the clock rate, rather than just a quarter of that rate. By quadrupling the memory of the configuration of Figure 10 and complicating the 3 adders, we can create a very fast but memory-hungry (8K word by 48 bit) filter structure that can perform the filtering function in a single clock period, as shown in Figure 11.

An alternate approach, which is shown in Figure 12, may be used in which eight ROMs are addressed by the three data streams, 2 bits at a time (2BAAT). Each memory is now a more modest $\frac{1}{2}\ (2^3)^2 = 32$ words $\times$ 48 bits for a total memory requirement of $8 \times 32 \times 48 = 12{,}288$ bits. The three adders are each of a complexity less than that of a 16-bit modified-Booth multiplier. In this approach, we have reduced the memory size at the expense of complicating the adders.

The outputs from the memories are 16 (or whatever,

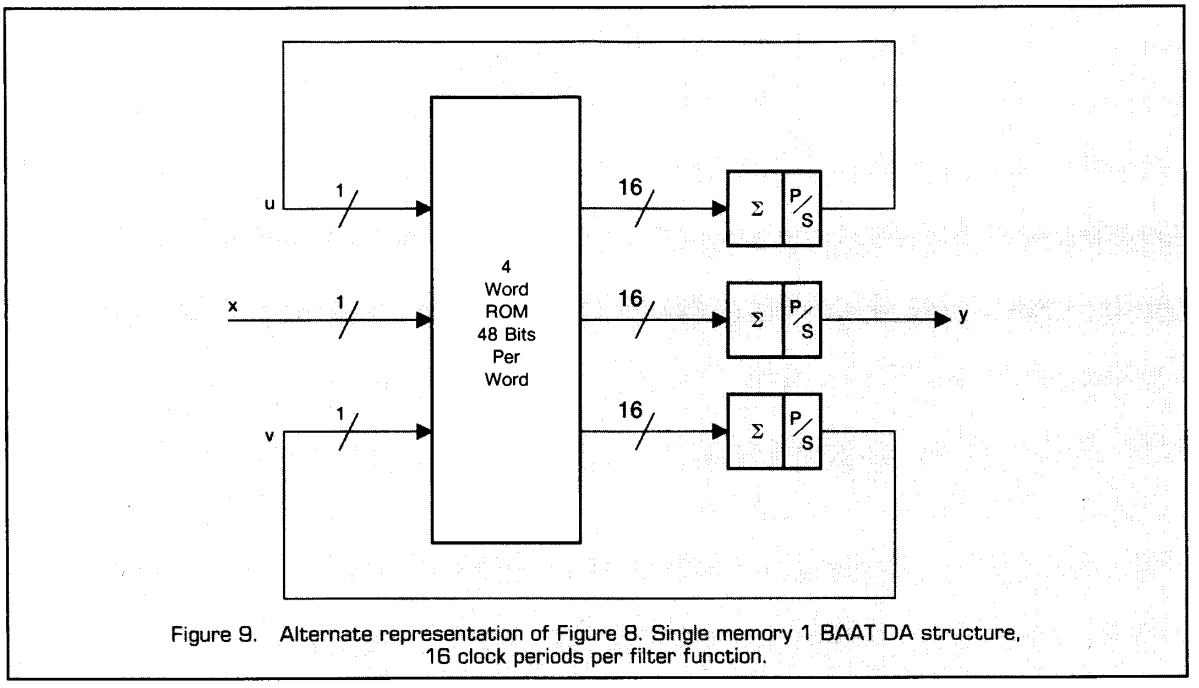

Figure 9. Alternate representation of Figure 8. Single memory 1 BAAT DA structure, 16 clock periods per filter function.

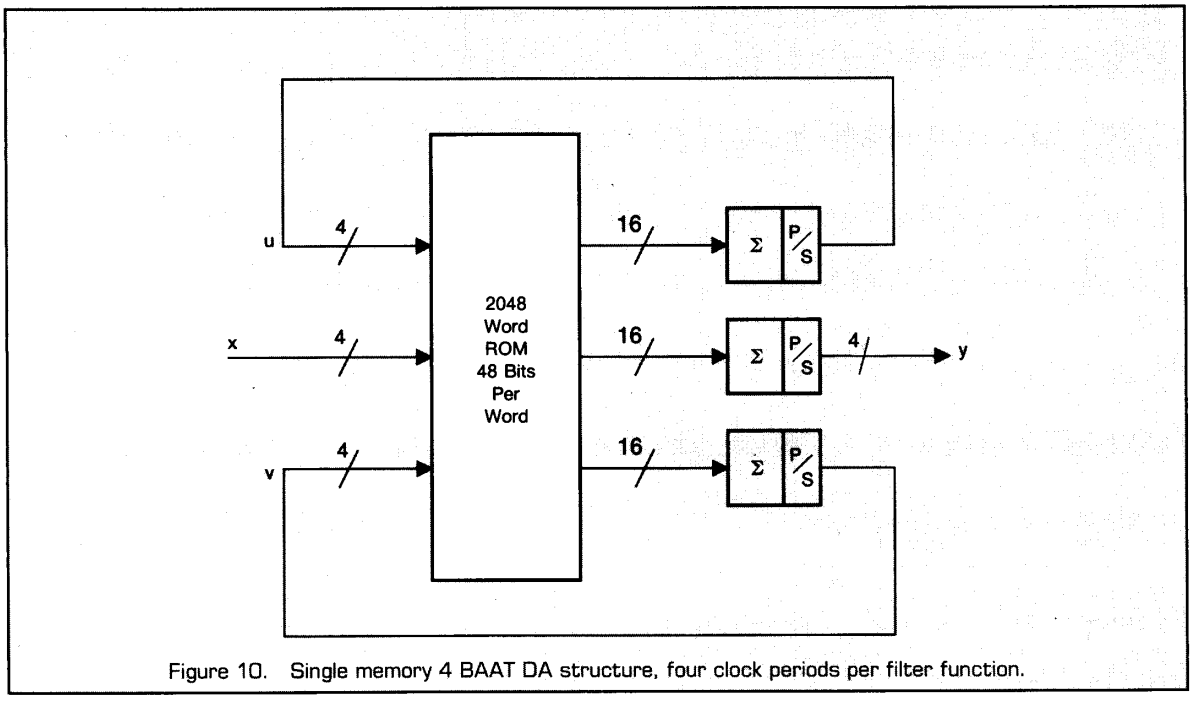

Figure 10. Single memory 4 BAAT DA structure, four clock periods per filter function.

say N, number of) bits. In the shift-and-add process that occurs in the accumulators, the accuracy of the results degrades because of the least-significant bits that are lost through the trauma of quantization. Figure 13 illustrates the problem; as the data circulates through the accumulator loop, LSB's are lost at the shift stage. (These lost bits are often modeled as an additive error.) From the point in the filter that this error is introduced (the accumulator outputs), we follow them around the recursive loop of the filter and see the reinforcement addition that is the cause of noise gain within recursive filters. We will create a parallel error path, now, with error subtractions that will nearly cancel the error additions.

Figure 14 shows the state-space filter of Figure 7 with

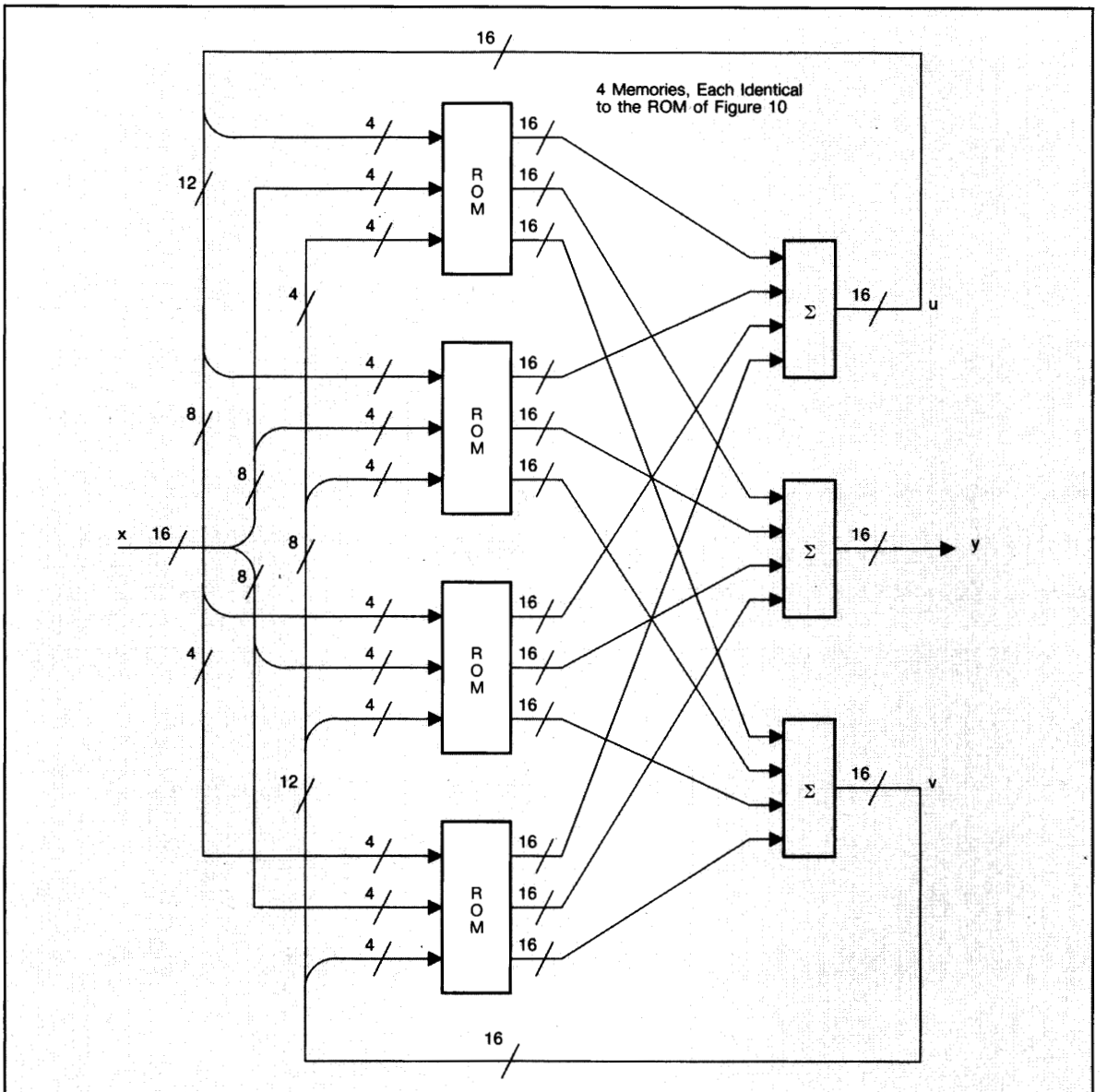

Figure 11. Four memory 4 BAAT DA structure, one clock period per filter function.

noise sources (quantization effects) ϵ_1, ϵ_2, and ϵ_3 added at the outputs of the accumulators. We have also added, within the dotted lines, noise-cancellation paths whose gains are shift approximations to the actual gains. This technique is known as error feedback and has long been used in DA filters [39]. We mechanize the noise cancellation paths by modifying Figure 8 as shown in Figure 15. Since at most the tilde-marked gains consist of a shift, those gains shown in Figure 15 are physically trivial. They feed a serial adder in which they are bit-by-bit combined with the initial conditions (ICs), then loaded into a serial-to-parallel register; that register now contains a chimerical initial condition, which includes the error feedback. If the entity within the dotted line of Figure 15 is redefined as the adder and parallel-to-serial register, then Figure 9 is valid for the error-feedback mechanization. If the adders that are fed by the ICs are L BAAT adders, then this structure can be generalized for use with structures that are shown in Figures 10–12.

APPLICATIONS IN TRANSFORMERS

DA has found important use in low-complexity, high-performance FFT structures [39, 40, 41]. The effective use

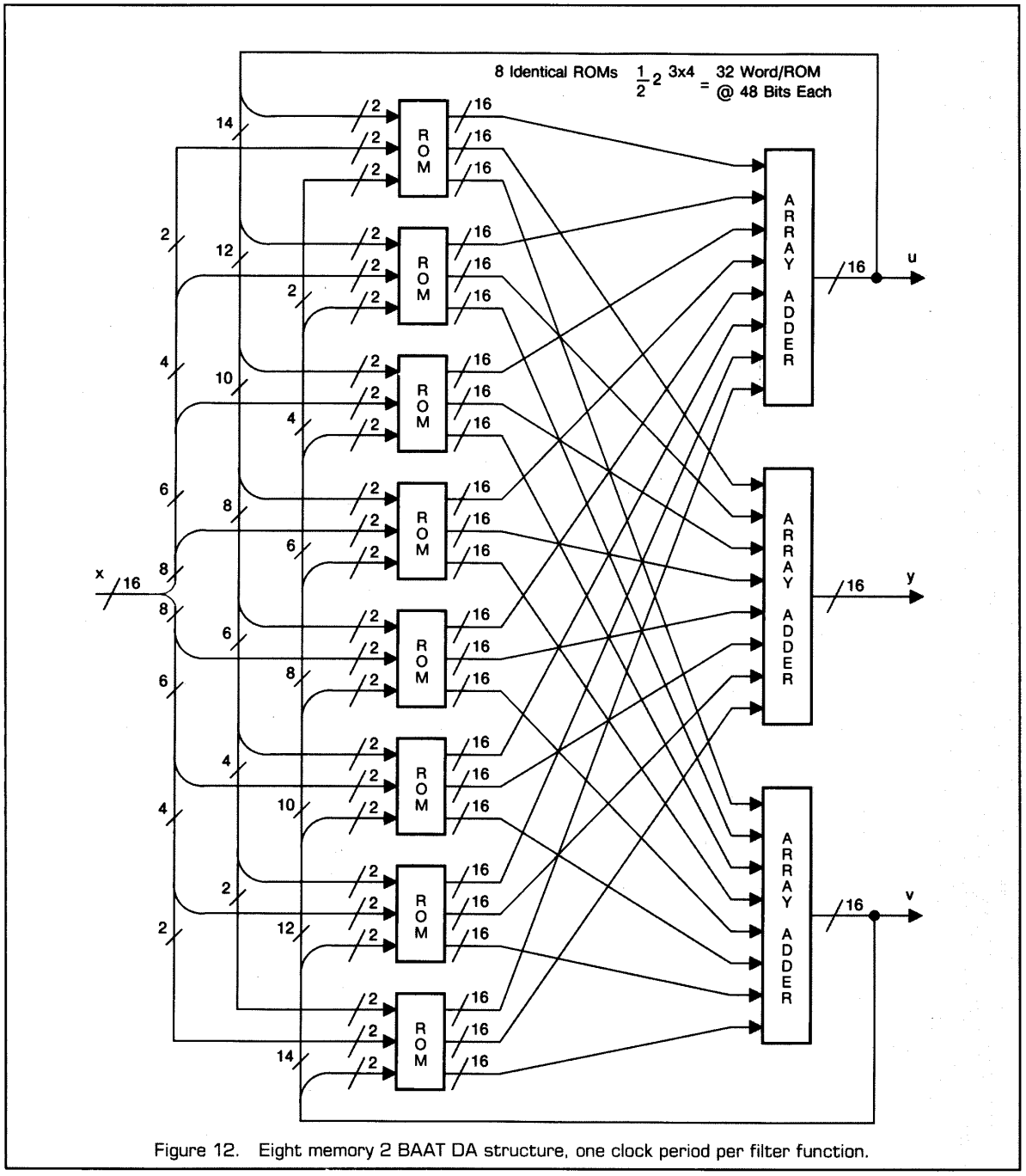

Figure 12. Eight memory 2 BAAT DA structure, one clock period per filter function.

of DA in the mechanization of a simple, direct, high-performance complex multiplier has been the reason for its success in FFT applications, and has spurred additional work in the development of efficient complex multipliers [42]. The development path for complex multipliers took an interesting twist with the advent of radix-3 and radix-6 FFT's [43, 44]. Their charm is that they give larger-radix multiplier-free building blocks that lead to increased efficiency in transform processing. By the simple expedient of turning to non-orthogonal coordinate systems for complex arithmetic, the resulting FFT structures [45] gave birth to a new complex multiplier [46] to perform the reduced number of "twiddles" on the interstage coupling.

Image processing has turned to the discrete-cosine

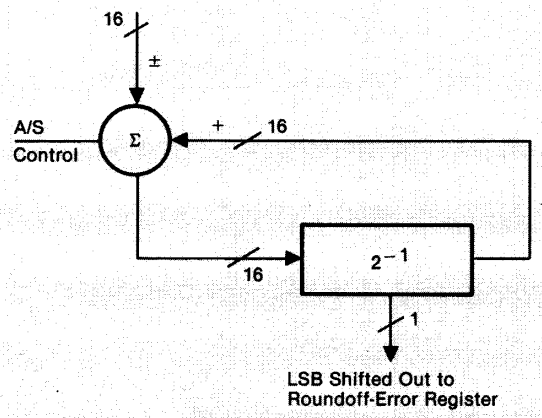

Figure 13. Accumulator loop showing LSB being dropped out on the right shift operation.

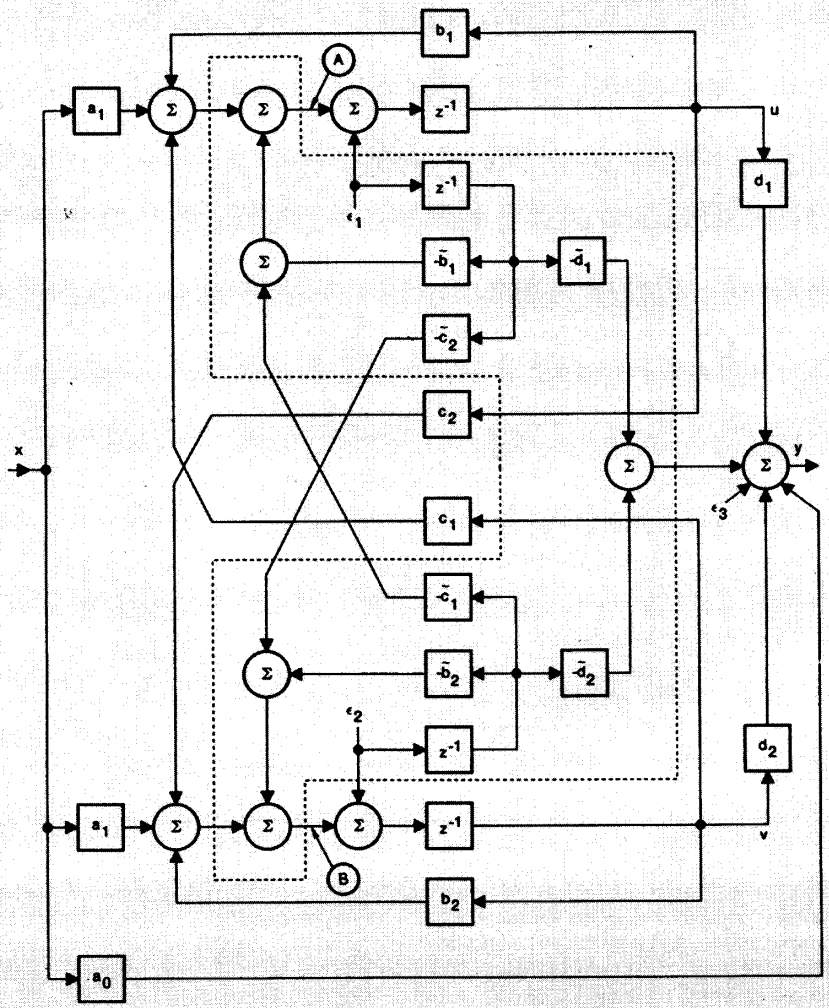

Figure 14. State space digital filter showing roundoff noise sources and cancellation loops.

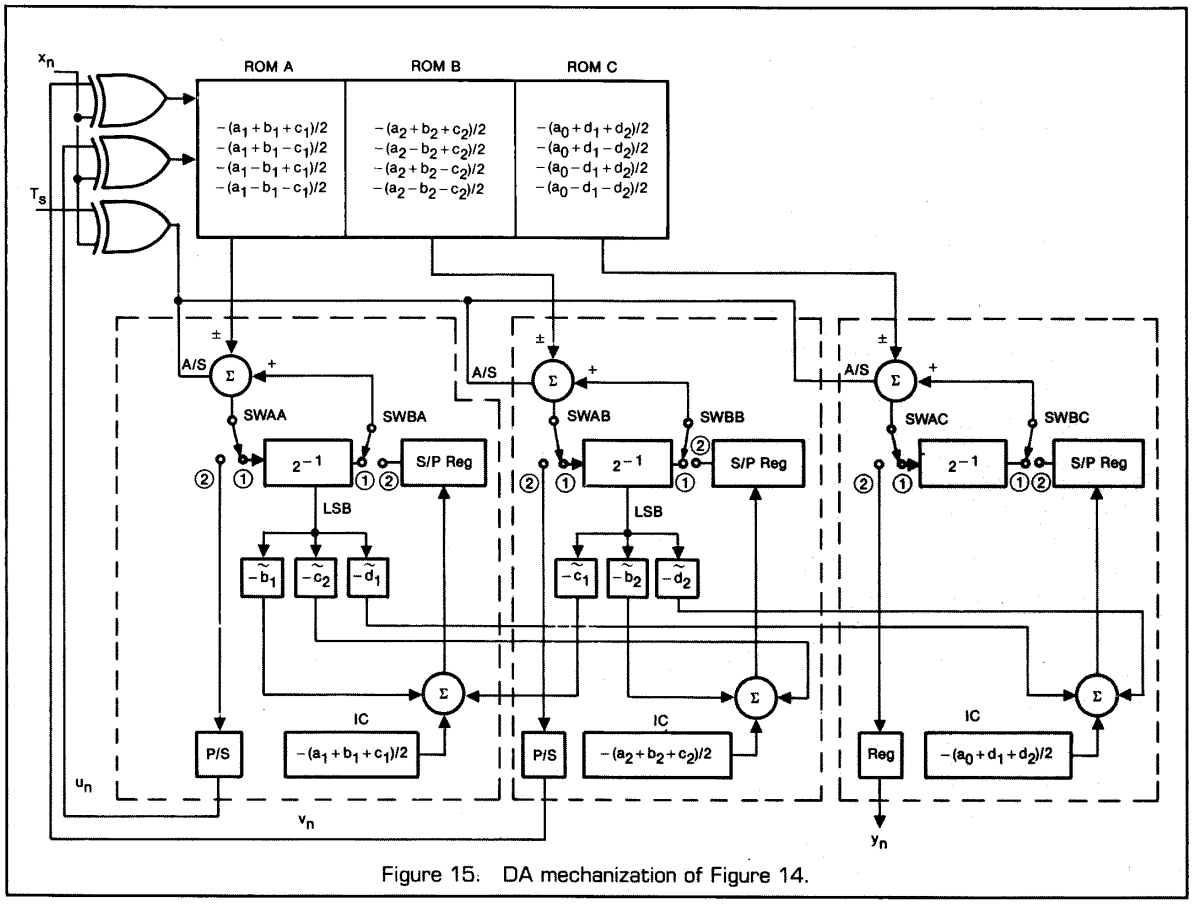

Figure 15. DA mechanization of Figure 14.

transform (DCT) for more efficient processing. There, again, DA has found a home [47, 48].

NONLINEAR AND/OR NONSTATIONARY PROCESSING WITH DA

We have only considered the use of DA in linear, time-invariant systems. It is not so restricted. For variable coefficients, we may use RAM's rather than ROM's. In fact, one of the trailblazers in this approach was Schroder [16].

In 1981 Cowan and Mavor [49] described an 8-tap adaptive transversal filter that employed DA, and in 1983 an expanded version was published by Cowan, Smith, and Elliott [50]. Andrews [51] compared it favorably in a mechanization study involving traditional and nontraditional arithmetic mechanizations of adaptive filters. The capacity of the DA adaptive structure can be increased by using block-processing concepts [52, 53].

The mechanization of nonlinear difference equations by DA was presented by Sicuranza in [54], who represented the filter by a truncated discrete Volterra series. Chiang et al. in [55] report the results of a mechanization trade-off study of various implementations of quadratic filters. They conclude that DA is as efficient as matrix decomposition implemented with systolic arrays, but that DA lacks the modularity.

Satisfactory DA adaptive nonlinear filters have been investigated and reported by Sicuranza and Ramponi [56], and by Smith et al. [57].

CONCLUSIONS

DA is a very efficient means to mechanize computations that are dominated by inner products. As we have seen, the coefficients of the equations can be time varying, and the equations themselves can be nonlinear. When a great many computing methods are compared, DA has always fared well, not always (but often) best, and never poorly. As a consequence, whenever the performance/cost ratio is critical (especially in custom designs), DA should be seriously considered as a contender.

ACKNOWLEDGMENTS

I want to express my thanks to the many workers in the field who have quickly and generously shared their results with me; to my coworkers who, over the years, have patiently simulated, analyzed, built, and tested endless DA concepts, and corrected my errors; and to the reviewers who offered very helpful suggestions and led me to references that were new and unfamiliar to me.

REFERENCES

[1] A. Peled and B. Liu, "A New Approach to the Realization of Nonrecursive Digital Filters," *IEEE Trans. Audio and Electroacoustics*, Vol. AU-21, No. 6, pp. 477–485, December 1973.

[2] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-22, pp. 456–462, December 1974.

[3] A Croisier, D. J. Esteban, M. E. Levilion, and V. Rizo, "Digital Filter for PCM Encoded Signals," U.S. Patent 3,777,130, December 3, 1973.

[4] S. Zohar, "New Hardware Realization of Nonrecursive Digital Filters," *IEEE Trans. on Computers*, Vol. C-22, pp. 328–338, April 1973.

[5] S. Zohar, "The Counting Recursive Digital Filter," *IEEE Trans. on Computers*, Vol. C-22, pp. 338–347, April 1973.

[6] S. Zohar, "A Realization of the RAM Digital Filter," *IEEE Trans. on Computers*, Vol. C-25, pp. 1048–1052, October 1976.

[7] W. D. Little, "A Fast Algorithm for Digital Filters," *IEEE Trans. on Communications*, Vol. C-23, pp. 466–469, May 1974.

[8] B. E. Bona and F. C. Terhan, "A Special-Purpose Digital Control Compensator," *Proc. 8th Asilomar Conference on Circuits, Systems, and Computers*, Pacific Grove, California, pp. 454–457, December 1974.

[9] S. A. White, "Applications of Digital Signal Processing to Control Systems," *Proc. 8th Asilomar Conference on Circuits, Systems, and Computers*, Pacific Grove, California, pp. 278–284, December 1974.

[10] S. L. Freeny, "Special-Purpose Hardware for Digital Filtering," *Proc. IEEE*, Vol. 63, pp. 633–648, April 1975.

[11] S. A. White, "On Mechanization of Vector Multiplication," *Proc. IEEE*, Vol. 63, pp. 730–731, April 1975.

[12] S. A. White, W. P. Engler, J. P. Davis, S. L. Smith, J. V. Henning, "A Programmable Digital Servo Controller," Invention Disclosure PF75E145, October 16, 1975.

[13] T. R. C. M. Classen, W. F. G. Mecklenbrauker, and J. D. H. Peek, "Some Considerations on the Implementation of Digital Systems for Signal Processing," Philips Research Report 30, pp. 73–84, 1975.

[14] M. Buttner and H. W. Schuessler, "On Structures for the Implementation of the Distributed Arithmetic," *NTZ Communication Journal*, Vol. 6, June 1975.

[15] Kai-Ping Yiu, "On Sign-Bit Assignment for a Vector Multiplier," *Proc. IEEE*, Vol. 64, pp. 372–373, March 1976.

[16] H. Schroder, "High Word-Rate Digital Filters with Programmable Table Look-Up," *IEEE Trans. on Circuits and Systems*, Vol. CAS-24, No. 5, pp. 277–279, May 1977.

[17] C. S. Burrus, "Digital Filter Realization by Distributed Arithmetic," International Symposium on Circuits and Systems, Munich, April 1976.

[18] K. D. Kammeyer, "Digital Filter Realization in Distributed Arithmetic," *Proc. European Conf. on Circuit Theory and Design*, Genoa, Italy, September 1976.

[19] C. S. Burrus, "Digital Filter Structures Described by Distributed Arithmetic Filters," *IEEE Trans. on Circuits and Systems*, Vol. CAS-24, No. 12, pp. 674–680, December 1977.

[20] W. K. Jenkins, and B. J. Leon, "The Use of Residue Number System in the Design of Finite Impulse Response Filters," *IEEE Trans. on Circuits and Systems*, Vol. CAS-24, No. 4, April 1977.

[21] J. Zeman, and H. T. Nagle, Jr., "A High-Speed Microprogrammable Digital Signal Processor Employing Distributed Arithmetic," *IEEE Trans. on Computers*, Vol. C-29, No. 2, pp. 134–144, February 1980.

[22] B. S. Tam, and G. J. Hawkins, "Speed-Optimized Microprocessor Implementation of a Digital Filter," *IEEE Proc.*, Vol. 69, No. 3, pp. 85–93, May 1981.

[23] M. Arjmand and R. A. Roberts, "On Comparing Hardware Implementations of Fixed-Point Digital Filters," *IEEE Circuits and Systems Magazine*, Vol. 3, No. 2, 1981, pp. 2–8.

[24] S. A. White, "Architecture for a Digital Programmable Image Processing Element," *Proc. 1981 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Atlanta, pp. 658–661, March 30–April 1, 1981.

[25] S. A. White, "An Architectural for a High-Speed Digital Signal Processing Device," *Proc. 1981 IEEE International Symposium on Circuits and Systems*, Chicago, pp. 893–896, April 28–29, 1981.

[26] K. D. Kammeyer, "Quantization Error of the Distributed Arithmetic," *IEEE Trans. on Circuits and Systems*, Vol. CAS-24, No. 12, pp. 681–689, December 1977.

[27] F. J. Taylor, "An Analysis of the Distributed-Arithmetic Digital Filter," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-35, No. 5, pp. 1165–1170, Oct. 1986.

[28] S. G. Smith and S. A. White, "Hardware Approaches to Vector-Plane Rotation," *Proc. 1988 IEEE International Conference on Acoustics, Speech, and Signal Processing*, New York, pp. 212–213, April 11–14, 1988.

[29] W. P. Burleson and L. L. Scharf, "A VLSI Implementation of a Cellular Rotator Array," *Proc. 1988 IEEE Custom Integrated Circuits Conference*, pp. 8.1.2–8.1.4.

[30] F. J. Taylor, "A Distributed Gray-Markel Filter," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-31, No. 3, pp. 761–763, June 1983.

[31] S. Zohar, "A VLSI Implementation of a Correlator/Digital Filter Based on Distributed Arithmetic," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-37, No. 1, pp. 156–160, Jan. 1989.

[32] F. J. Taylor, *Digital Filter Design Handbook*, Marcel Dekker, Inc., pp. 678–697, June 1983.

[33] S. G. Smith and P. B. Deyer, *Serial-Data Computation*, Kluwer Academic Publishers, 1988.

[34] D. F. Elliott (ed.), *Handbook of Digital Signal Processing*, Academic Press, pp. 964–972, 1987.

[35] C. W. Barnes, "A Parametric Approach to the Realization of Second-Order Digital-Filter Sections," *IEEE*

Trans. on Circuits and Systems, Vol. CAS-32, No. 6, pp. 530–539, June 1985.

[36] S. A. White, "High-Speed Distributed-Arithmetic Realization of a Second-Order Normal-Form Digital Filter," *IEEE Trans. on Circuits and Systems,* Vol. CAS-33, No. 10, pp. 1036–1038, October 1986.

[37] S. A. White, "High-Speed Distributed-Arithmetic Realization of a Second-Order State-Space Digital Filter," *Proc. 20th Asilomar Conference on Signals, Systems, and Computers,* Pacific Grove, California, pp. 359–362, November 10–12, 1986.

[38] T. L. Chang and S. A. White, "An Error Cancellation Digital-Filter Structure and Its Distributed Arithmetic Implementation," *IEEE Trans. on Circuits and Systems,* Vol. CAS-28, No. 4, pp. 339–342, April 1981.

[39] S. A. White, "A Simple FFT Butterfly Arithmetic Unit," *IEEE Trans. on Circuits and Systems,* Vol. CAS-28, No. 4, pp. 352–366, April 1981.

[40] I. R. Mactaggart and M. A. Jack, "A Single Chip Radix-2 FFT Butterfly Architecture Using Parallel Data Distributed Arithmetic," *IEEE J. Solid State Circuits,* Vol. SC-19, pp. 368–373, June 1984.

[41] S. A. White, "An Architecture for a 16-Point FFT GaAs Building Block," *Proc. 21st Asilomar Conference on Signals, Systems, and Computers,* Pacific Grove, California, pp. 928–932, November 1987.

[42] S. G. Smith and P. B. Denyer, "Efficient Bit-Serial Complex Multiplication and Sum-of-Products Computation Using Distributed Arithmetic," *Proc. 1986 International Conference on Acoustics, Speech, and Signal Processing,* Tokyo, pp. 2203–2206, April 1986.

[43] E. Dubois and A. N. Venetsanopoulos, "A New Algorithm for the Radix-3 FFT," *IEEE Trans. on A.S.S.P.,* Vol. ASSP-26, No. 3, pp. 222–225, June 1978.

[44] S. Prakash and V. V. Rao, "A New Radix-6 FFT Algorithm," *IEEE Trans. on A.S.S.P.,* Vol. ASSP-29, No. 4, pp. 939–941, August 1981.

[45] S. A. White, "Results of a Preliminary Study of a Novel IC Arithmetic Unit for an FFT Processor," *Proc. 18th Asilomar Conference on Circuits, Systems, and Computers,* Pacific Grove, California, pp. 67–71, November 5–7, 1984.

[46] S. A. White, "A Complex Multiplier for Binary Two's-Complement Numbers," U.S. Patent No. 4,680,727, July 14, 1987.

[47] N. Demassieux, G. Concordel, J.-P. Durandeau, and F. Jutand, "An Optimized VLSI Architecture for a Multiformat Discrete Cosine Transform," *Proc. 1987 IEEE International Conference on Acoustics, Speech, Signal Processing,* pp. 547–550.

[48] A. M. Gottlieb, M. T. Sun, and T. C. Chen, "A Video Rate 16 × 16 Discrete Cosine Transform IC," *Proc. 1988 IEEE Custom Integrated Circuits Conference,* pp. 8.2.1–8.2.4.

[49] C. F. N. Cowan and J. Mavor, "New Digital-Adaptive Filter Implementation Using Distributed-Arithmetic Techniques," *IEE Proc.,* Vol. 128, Pt. F, No. 4, pp. 225–230, Aug. 1981.

[50] C. F. N. Cowan, S. G. Smith, and J. H. Elliott, "A Digital Adaptive Filter Using a Memory-Accumulator Architecture: Theory and Realization," *IEEE Trans. on A.S.S.P.,* Vol. ASSP-31, No. 3, pp. 541–549, June 1983.

[51] M. Andrews, "A Systolic SBNR Adaptive Signal Processor," *IEEE J. Solid State Circuits,* Vol. SC-21, No. 1, pp. 120–128, Feb. 1986.

[52] C.-H. Wei, and J.-J. Lou, "Multimemory Block Structure for Implementing a Digital Adaptive Filter Using Distributed Arithmetic," *IEE Proc.,* Vol. 133, Pt. G, No. 1, pp. 19–26, Feb. 1986.

[53] Y.-Y. Chiu and C.-H. Wei, "On the Realization of Multimemory Block Structure Digital Adaptive Filter Using Distributed Arithmetic," *J. Chinese Institute of Engineers,* Vol. 10, No. 1, pp. 115–122, 1987.

[54] G. L. Sicuranza, "Nonlinear Digital-Filter Realization by Distributed Arithmetic," *IEEE Trans. on A.S.S.P.,* Vol. ASSP-33, No. 4, pp. 939–1321, Aug. 1985.

[55] H.-H. Chiang, C. L. Nikias, and A. N. Venetsanopoulos, "Efficient Implementations of Quadratic Digital Filters," *IEEE Trans. on A.S.S.P.,* Vol. ASSP-34, No. 6, pp. 1511–1528, Dec. 1986.

[56] G. L. Sicuranza, "Adaptive Nonlinear Digital Filters Using Distributed Arithmetic," *IEEE Trans. on A.S.S.P.,* Vol. ASSP-34, No. 3, pp. 518–526, June 1986.

[57] M. J. Smith, C. F. N. Cowan, and P. F. Adams, "Nonlinear Echo Cancellers Based on Transpose Distributed Arithmetic," *IEEE Trans. on Circuits and Systems,* CAS-35, No. 1, pp. 6–18, Jan. 1988.

Stanley A. White (S'55, M'57, SM'69, F'82) was born in Providence, Rhode Island in 1931. He is Senior Scientist at Rockwell International Corporation's Autonetics Electronics Systems in Anaheim, California, and Adjunct Professor of Electrical Engineering at the University of California, Irvine. He received his B.S.E.E., M.S.E.E., and Ph.D. (with letter of commendation) degrees from Purdue University in 1957, 1959, and 1965, respectively. He is also a Fellow of the American Association for the Advancement of Science (AAAS), the New York Academy of Sciences (NYAS), and the Institute for the Advancement of Engineering (IAE). He is a member (and long-time Autonetics Chapter President) of Sigma Xi, Tau Beta Pi, the Audio Engineering Society, and has been International Director of Eta Kappa Nu. He received the Purdue University Distinguished Engineering Alumnus Award in 1988, the Leonardo Da Vinci Medallion in 1986, and was selected as Rockwell International Engineer of the Year in 1985. In 1984 he received both the Engineer of the Year Award from the Orange County (California) Engineering Council and the IEEE Centennial Medal. He has held several distinguished lecturerships and received the NEC Distinguished Lecturer Award.

Dr. White has over 100 publications in the open literature, holds 30 U.S. patents (with more pending issue), and is a registered Professional Engineer in both California and Indiana.

He is General Chairman of ISCAS '92; was General Chairman of ICASSP '84, Vice Chairman of ISCAS '83; was both General Chairman of the Asilomar Conference on Circuits, Systems, and Computers, and Technical Program Chairman of the IEEE Region 6 Conference in 1982; and was Technical Program Chairman of the Asilomar Conference in 1981. Dr. White is listed in *American Men and Women of Science, Who's Who in Biomedical Engineering, Who's Who in Engineering, Who's Who in America, Who's Who in The World,* and other standard biographical reference works.