On the Hardware Implementation of Digital Signal Processors

ABRAHAM PELED, MEMBER, IEEE

Abstract—An approach to the machine organization of dedicated hardware digital signal processors is proposed that is based on a specialized representation of the processing coefficients derived from the canonical signed-digit code. This leads to a realization requiring the minimum number of add/subtract operations to mechanize the required multiplications and additions.

The proposed organization is shown to be highly modular and well suited to integrated circuit implementation, and offers a significantly better performance when compared with existing realizations using prepackaged multipliers.

I. Introduction

THE increasing availability of computing power at continually decreasing cost has made real-time digital signal processing possible and cost-effective in many areas, including speech processing, seismic exploration, vibration analysis, and radar and sonar detection, to mention only a few [1]-[4].

There exists a large class of applications in which the digital signal processing is performed in real time by dedicated hardware. This paper addresses such applications, as opposed to applications in which general purpose computers are used to perform the digital signal processing.

In this paper we suggest a possible implementation of such dedicated hardware signal processors based on the use of the canonical signed-digit code [5] to represent the set of constants required for the processing. This canonical coding will permit the mechanization of the multiplications required, using a minimum number of add/subtract operations. This code was used extensively in the early days of digital computers to achieve fast multiplication. Later on, due to the inefficiency involved in the conversion, more restricted versions were used [6] that still achieved a speed compatible with other operations to be performed in a general purpose computer, and not making the arithmetic unit the bottleneck limiting the speed of operation. This is, though, not the case in many of today's real-time digital signal processing systems. These systems are essentially "number crunchers" requiring very high computation rates, and in many cases several arithmetic units operating in parallel will be needed. Thus the cost of the arithmetic unit will be a major contributor to the total systems cost, significantly affecting its cost effectiveness.

Another factor that makes the canonical signed-digit code a potentially attractive way of representing the processing coefficients is that without loss of generality, and only at the expense of some storage increase, we can maintain those coefficients in this code in our machine and eliminate the inefficiency of having to convert the multiplicand before each multiplication from the standard binary code to the signed-digit code that was present in general purpose computers.

We now proceed to propose an architecture for the machine organization of such real-time digital signal processors that will exploit this special representation of the processing coefficients to achieve a significantly better multiplication rate per hardware expenditure than is achieved with standard multipliers. Although in this paper we will mainly base our comparison on standard available TTL medium-scale integration (MSI) IC's, the same improvements can be expected when large-scale integration (LSI) and other technologies are considered.

Manuscript received March 24, 1975; revised July 10, 1975.

The author is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

II. Representing the Processors Coefficients on the Basis of the Canonical Signed-Digit Code

The signed-digit code is well known, was especially prominent in the early days of electronic computers, and appeared in many publications in the 1950's with reference to fast multiplication. A complete treatment of the properties of this representation is given in [5] and a discussion of its use in fast multiplication is given in [7]. We repeat below, without proof, some of the more important properties of this representation.

Given an integer X in the range

$$-2^{B-1} \leqslant X < 2^{B-1}. \tag{1}$$

It can be represented in a 2's complement binary representation, or in a generalized representation by B signed binary coefficients β_i, according to

$$X = -x_B 2^{B-1} + \sum_{i=0}^{B-2} x_i \cdot 2^i = \sum_{i=0}^{B-1} \beta_i \cdot 2^i$$

$$x_i = 0, 1; \quad \beta_i = 0, \pm 1, B-1 \geqslant i \geqslant 0. \tag{2}$$

The representation of X in terms of β_i is the binary signed digit code for X. This code has the following properties.

1) Minimality and Uniqueness: Let T be defined by

$$T = \sum_{i=0}^{B-1} |\beta_i| \tag{3}$$

and we require that

$$\beta_i \cdot \beta_{i-1} = 0 \qquad n \geqslant i \geqslant 0, \tag{4}$$

i.e., no two consecutive digits are nonzero. Then there exists a unique set $\{\beta_i\}$ that satisfied (4) and there exists no other set $\{\beta_i\}$ for which T is less, that is, X is represented using a

minimum number of nonzero β_i's. A representation satisfying (4) is called the canonical signed digit binary code.

2) Existence: For any integer in the range of (1) there exists such a minimal decomposition.

3) Average Number of Nonzero Digits: Assuming a uniform probability density for X on the range of (1), then the probability of occurrence of a nonzero digit is given by

$$\Pr\{|\beta_i| = 1\} = \frac{1}{3} + \frac{1}{9B}\left(1 - \left(-\frac{1}{2}\right)^B\right) \tag{5}$$

and, therefore, for moderately large B the average number of nonzero coefficients β_i will be $B/3$ as opposed to $B/2$ in the usual (nonsigned) binary representations.

The following identity is basic to understanding the reduction in the number of add/subtract operations required when the multiplicand is expressed in the canonical signed digit binary code

$$2^{k+n+1} - 2^k = 2^{k+n} + 2^{k+n-1} + 2^{k+n-2} + \cdots + 2^k. \tag{6}$$

This identity indicates that a sequence of add operations will be replaced by an addition and substraction (e.g., the number 15 instead of being written as 01111 which requires four additions will be written as 1000-1 which requires only two). Equation (6) is also the key to the conversion from a regular binary representation to the canonical signed digit. Starting from the least significant bit and progressing towards the most significant bit, let x_i, x_{i+1} be two consecutive bits in the binary representation of X, and c_i the carry generated by the conversion in the ith step ($c_o = 0$). Then x_i is replaced by β_i according to the rules given in Table I. The conversion can be done also in parallel, which is straightforward based on (6).

Tables II-V give representative examples of the representation of the filter coefficients for various nonrecursive filters using this code. All nonrecursive filters shown here are symmetric with $h_i = h_{N-i}\, i = 0, \cdots [N/2]_I$, where h_i are the coefficients. Therefore we list only $[N/2]_I$ coefficients in the tables ($[X]_I$ denotes the integer part of X). Figs. 1-4 show the frequency response of these filters as obtained with the quantized coefficients. Since we started with equiripple filters, it is evident from the figures that quantization of the filter coefficients destroys this property. In Table II a 27-tap low-pass nonrecursive filter is given and the average number of nonzero digits per coefficient is 2.36, and since $B = 12$, this is $B/5.08$. Next we see a 33-tap low-pass filter using 10-bit coefficients, and here we get $B/4.37$ nonzero digits. Table IV shows a 56-tap bandpass filter using 12 bits. Again, here the average number of nonzero digits is $B/4.95 = 2.43$. Finally, Table VI shows a 100-tap multiband filter. Here we get $B/6.2 = 1.94$ nonzero digits. Thus, as we see from these examples, for nonrecursive filters the average number of nonzero digits is far less than $B/3$, and will in most cases average less than $B/4.5$, the exact number depending on the attenuation required. This is very significant. To illustrate that we mention that the 100-tap filter of Table VI will require in the standard approach 50 multiplications and 100 additions per input sample, taking into account the coefficient symmetry (i.e., if data come in at a 10-kHz rate we have to perform a multiplication and two additions every 2 μs), whereas if we use the signed-digit code we will be able to accomplish the same doing only 150 additions (50 + 50(1.94-1) + 50, see (7) and following text for explanation), which for the 10-kHz input rate means doing only three additions in 2 μs.

TABLE I
CONVERSION RULES FROM BINARY TO THE CANONICAL SIGNED DIGIT CODE

c_i	x_{i+1}	x_i	c_{i+1}	β_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	-1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	-1
1	1	1	1	0

Table VI shows the results for the cosine coefficients required in a 64-point discrete Fourier transform (DFT), obviously the sine coefficients will be the same. Table VII shows the average number of nonzero digits per coefficient for various length transforms and number of bits used to represent the coefficients. It is easy to see that *irrespective* of these parameters we average $B/3$ nonzero digits, as could be expected.

Tables VIII and IX show the case of recursive digital filters, first a sixth-order notch filter, and a tenth-order low-pass filter. Here we average $B/3$ nonzero digits, since we are generally dealing with large numbers. The average for the numerator is lower, but that is due to the fact that every 2 out of 3 coefficients are 1, and we would not count those as multiplications.

After we have presented these representative examples, we now proceed to point out another property of many digital signal processors that makes additional savings possible. In many cases the operation to be performed is

$$z = a \cdot x + y \tag{7}$$

where a is a constant coefficient, and x and y are either data points or intermediate results. As we will show in the next section, by choosing a suitable architectural structure for the arithmetic unit we will be able to combine this operation and make the number of add/substract operations (with concurrent shifting) required to perform this multiply-add operation $B/3$-1. It follows therefore that we will average only 0.94 adds per multiply for the nonrecursive filter of Table VI (for $B = 12$) and 2.6 adds per multiply for the FFT coefficients (if 12 bits are used to represent the coefficients).

The question that we now address is what is the most suitable representation for the processing coefficients that will be based on the canonical signed-digit code and minimize the control logic required in the arithmetic unit to derive the following information.

1) The total number of add/subtract operations required to multiply by this coefficient—this information is mainly needed

TABLE II
NONRECURSIVE DIGITAL FILTER—27 TAPS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING 12 BITS													
8.14712050E-03	17	0	0	0	0	0	0	0	1	0	0	0	1
8.82963880E-04	2	0	0	0	0	0	0	0	0	0	0	1	0
-1.21095780E-02	-25	0	0	0	0	0	0	-1	0	1	0	0	-1
-2.33753820E-04	0	0	0	0	0	0	0	0	0	0	0	0	0
1.94129390E-02	40	0	0	0	0	0	0	1	0	1	0	0	0
1.22882030E-03	3	0	0	0	0	0	0	0	0	0	0	1	0 -1
-3.27448250E-02	-67	0	0	0	0	0	-1	0	0	0	-1	0	1
-6.95225080E-04	-1	0	0	0	0	0	0	0	0	0	0	0	-1
5.33999200E-02	109	0	0	0	0	1	0	0	-1	0	-1	0	1
1.54158240E-03	3	0	0	0	0	0	0	0	0	0	1	0	-1
-1.00243210E-01	-205	0	0	0	-1	0	1	0	-1	0	1	0	-1
-9.82600960E-04	-2	0	0	0	0	0	0	0	0	0	0	-1	0
3.15881190E-01	647	0	0	1	0	1	0	0	0	1	0	0	-1
5.01667680E-01	1027	0	1	0	0	0	0	0	0	0	1	0	-1

NUMBER OF NONZERO DIGITS IS = 33 I.E. AN AVERAGE OF = 2.36 PER COEFFICIENT

TABLE III
NONRECURSIVE DIGITAL FILTER—33 TAPS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING 10 BITS										
-1.70975550E-03	-1	0	0	0	0	0	0	0	0	-1
-5.66305590E-03	-3	0	0	0	0	0	0	0	-1	0 1
-5.16583400E-03	-3	0	0	0	0	0	0	0	-1	0 1
2.10548940E-03	1	0	0	0	0	0	0	0	0	0 1
1.05457450E-02	5	0	0	0	0	0	0	0	1	0 1
6.72016290E-03	3	0	0	0	0	0	0	0	1	0 -1
-1.04736800E-02	-5	0	0	0	0	0	0	0	-1	0 -1
-2.07911390E-02	-11	0	0	0	0	0	-1	0	1	0 1
-3.13846630E-03	-2	0	0	0	0	0	0	0	0	-1 0
2.97799370E-02	15	0	0	0	0	0	1	0	0	0 -1
3.30440400E-02	17	0	0	0	0	0	1	0	0	0 1
-1.68231320E-02	-9	0	0	0	0	0	0	-1	0	0 -1
-7.23012090E-02	-37	0	0	0	0	-1	0	0	-1	0 -1
-4.32102380E-02	-22	0	0	0	0	-1	0	1	0	1 0
1.06116850E-01	54	0	0	0	1	0	0	-1	0	-1 0
2.94427570E-01	151	0	0	1	0	1	0	-1	0	0 -1
3.80510870E-01	195	0	1	0	-1	0	0	0	1	0 -1

NUMBER OF NONZERO DIGITS IS = 39 I.E. AN AVERAGE OF = 2.29 PER COEFFICIENT

TABLE IV
NONRECURSIVE DIGITAL FILTER—56 TAPS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING 12 BITS													
6.28501640E-04	1	0	0	0	0	0	0	0	0	0	0	0	1
-2.06151510E-03	-4	0	0	0	0	0	0	0	0	0	-1	0	0
-3.59398960E-03	-7	0	0	0	0	0	0	0	0	-1	0	0	1
9.19772080E-04	2	0	0	0	0	0	0	0	0	0	0	1	0
2.20607590E-03	5	0	0	0	0	0	0	0	0	0	1	0	1
-6.09823270E-04	-1	0	0	0	0	0	0	0	0	0	0	0	-1
5.80978770E-03	12	0	0	0	0	0	0	0	1	0	-1	0	0
5.97883390E-03	12	0	0	0	0	0	0	0	1	0	-1	0	0
-7.19704110E-03	-15	0	0	0	0	0	0	0	-1	0	0	0	1
-8.88548790E-03	-18	0	0	0	0	0	0	0	-1	0	0	-1	0
6.15206550E-04	1	0	0	0	0	0	0	0	0	0	0	0	1
-6.92458820E-03	-14	0	0	0	0	0	0	0	-1	0	0	1	0
-4.96438520E-03	-10	0	0	0	0	0	0	0	0	-1	0	-1	0
2.29928050E-02	47	0	0	0	0	0	1	0	-1	0	0	0	-1
1.85523180E-02	38	0	0	0	0	0	0	1	0	1	0	-1	0
-1.06724430E-02	-22	0	0	0	0	0	0	-1	0	1	0	1	0
-1.29632090E-03	-3	0	0	0	0	0	0	0	0	0	-1	0	1
-2.72547170E-03	-6	0	0	0	0	0	0	0	0	-1	0	1	0
-4.70883700E-02	-96	0	0	0	0	-1	0	1	0	0	0	0	0
-2.18418840E-02	-45	0	0	0	0	0	-1	0	1	0	1	0	-1
5.14126720E-02	105	0	0	0	0	1	0	-1	0	1	0	0	1
2.70580350E-02	55	0	0	0	0	0	1	0	0	-1	0	0	-1
6.36695330E-03	13	0	0	0	0	0	0	0	1	0	-1	0	1
8.06371570E-02	165	0	0	0	0	1	0	1	0	0	1	0	1
-6.57089050E-03	-13	0	0	0	0	0	0	0	-1	0	1	0	-1
-2.41418060E-01	-494	0	0	-1	0	0	0	0	1	0	0	1	0
-1.29671340E-01	-266	0	0	0	-1	0	0	0	0	-1	0	-1	0
2.71040200E-01	555	0	0	1	0	0	1	0	-1	0	-1	0	-1

NUMBER OF NONZERO DIGITS IS = 68 I.E. AN AVERAGE OF = 2.43 PER COEFFICIENT

TABLE V
NONRECURSIVE DIGITAL FILTER—100 TAPS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING 12 BITS

-2.2860471[illegible]E-04	0	0	0	0	0	0	0	0	0	0	0	0
4.6[illegible]E-04	1	0	0	0	0	0	0	0	0	0	0	1
1.721[illegible]E-04	0	0	0	0	0	0	0	0	0	0	0	0
-5.8[illegible]E-04	-1	0	0	0	0	0	0	0	0	0	0	-1
7.6[illegible]E-04	1	0	0	0	0	0	0	0	0	0	1	0
-1.1722877[illegible]E-04	0	0	0	0	0	0	0	0	0	0	0	0
-4.6[illegible]E-04	-1	0	0	0	0	0	0	0	0	0	0	-1
-2.2867473[illegible]E-04	0	0	0	0	0	0	0	0	0	0	0	0
-3.11307840E-04	-1	0	0	0	0	0	0	0	0	0	0	-1
-6.6404[illegible]E-04	-1	0	0	0	0	0	0	0	0	0	0	-1
-2.3507[illegible]E-04	0	0	0	0	0	0	0	0	0	0	0	0
1.2529[illegible]E-03	3	0	0	0	0	0	0	0	0	1	0	-1
-2.5075[illegible]E-03	-5	0	0	0	0	0	0	0	0	-1	0	-1
3.1[illegible]E-03	7	0	0	0	0	0	0	0	1	0	0	-1
3.0[illegible]E-03	6	0	0	0	0	0	0	0	1	0	-1	0
-2.8[illegible]E-03	-6	0	0	0	0	0	0	0	-1	0	1	0
2.7[illegible]E-03	6	0	0	0	0	0	0	0	1	0	-1	0
7.6[illegible]E-04	2	0	0	0	0	0	0	0	0	0	1	0
-8.4[illegible]E-05	0	0	0	0	0	0	0	0	0	0	0	0
-3.9[illegible]E-03	-3	0	0	0	0	0	0	0	-1	0	0	0
-1.3[illegible]E-03	-3	0	0	0	0	0	0	0	0	-1	0	1
-6.0[illegible]E-03	-12	0	0	0	0	0	0	-1	0	1	0	0
-2.1[illegible]E-03	-4	0	0	0	0	0	0	0	0	-1	0	0
8.7[illegible]E-03	18	0	0	0	0	0	0	1	0	0	1	0
-1.4[illegible]E-02	-29	0	0	0	0	0	-1	0	0	1	0	-1
5.6[illegible]E-03	12	0	0	0	0	0	0	1	0	-1	0	0
1.4[illegible]E-02	30	0	0	0	0	0	1	0	0	0	-1	0
3.3[illegible]E-03	7	0	0	0	0	0	0	0	1	0	0	-1
4.5[illegible]E-03	9	0	0	0	0	0	0	0	1	0	0	1
-4.4[illegible]E-04	-1	0	0	0	0	0	0	0	0	0	0	-1
6.6[illegible]E-03	14	0	0	0	0	0	0	1	0	0	-1	0
-9.0[illegible]E-03	-19	0	0	0	0	0	0	-1	0	-1	0	1
8.7[illegible]E-03	18	0	0	0	0	0	0	1	0	0	1	0
-3.6[illegible]E-02	-75	0	0	0	0	-1	0	-1	0	1	0	1
-2.7[illegible]E-02	-55	0	0	0	0	-1	0	0	1	0	0	1
3.4[illegible]E-02	71	0	0	0	0	1	0	0	1	0	0	-1
-2.6[illegible]E-02	-54	0	0	0	0	-1	0	0	1	0	1	0
-3.6[illegible]E-03	-7	0	0	0	0	0	0	0	-1	0	0	1
7.6[illegible]E-03	16	0	0	0	0	0	0	1	0	0	0	0
4.0[illegible]E-02	83	0	0	0	0	1	0	1	0	1	0	-1
3.7[illegible]E-02	77	0	0	0	0	1	0	1	0	-1	0	1
2.8[illegible]E-02	58	0	0	0	0	1	0	0	-1	0	1	0
4.8[illegible]E-03	10	0	0	0	0	0	0	0	1	0	1	0
-7.1[illegible]E-02	-147	0	0	0	0	-1	0	0	-1	0	-1	0	1
1.3[illegible]E-01	272	0	0	0	1	0	0	0	1	0	0	0	0
-9.9[illegible]E-02	-204	0	0	0	-1	0	1	0	-1	0	1	0	0
-2.3[illegible]E-01	-481	0	0	-1	0	0	0	1	0	0	0	0	-1
-9.3[illegible]E-03	-19	0	0	0	0	0	0	0	-1	0	-1	0	1
-1.3[illegible]E-01	-272	0	0	0	-1	0	0	0	-1	0	0	0	0
3.3[illegible]E-01	688	0	1	0	-1	0	-1	0	-1	0	0	0	0

NUMBER OF NONZERO DIGITS IS = 97 I.E. AN AVERAGE OF = 1.94 PER COEFFICIENT

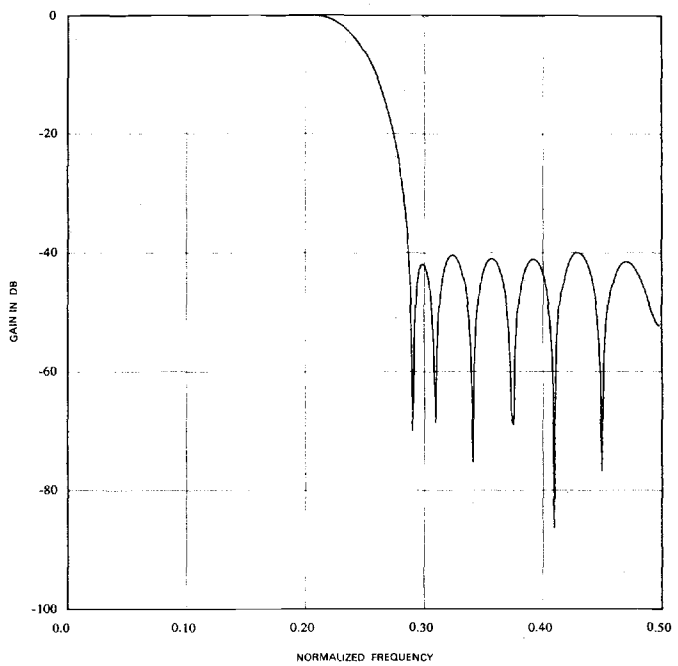

Fig. 1. Frequency response of 27-tap nonrecursive filter (12-bit coefficients).

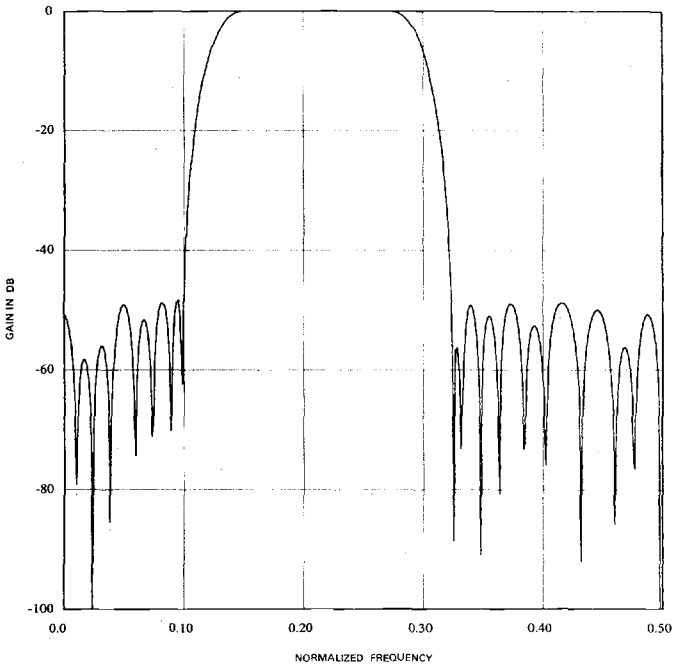

Fig. 3. Frequency response of 56-tap nonrecursive bandpass filter (12-bit coefficients).

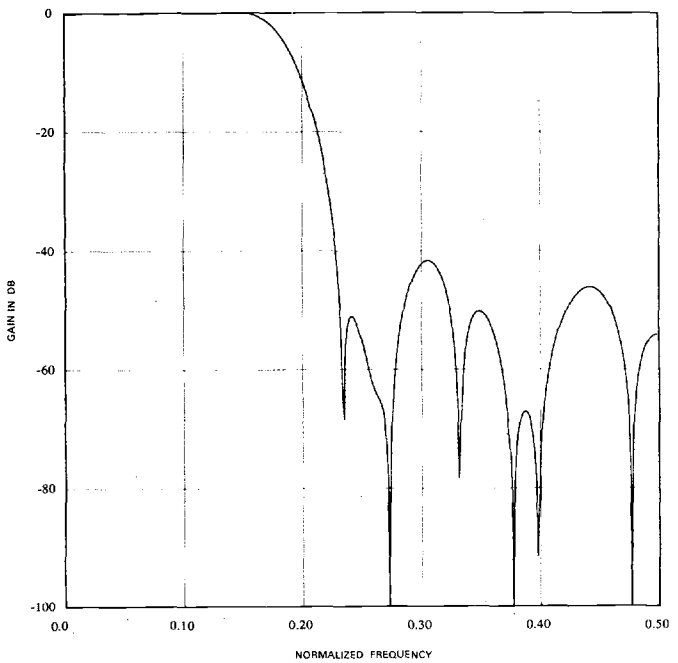

Fig. 2. 33-tap nonrecursive low-pass filter (10-bit coefficients).

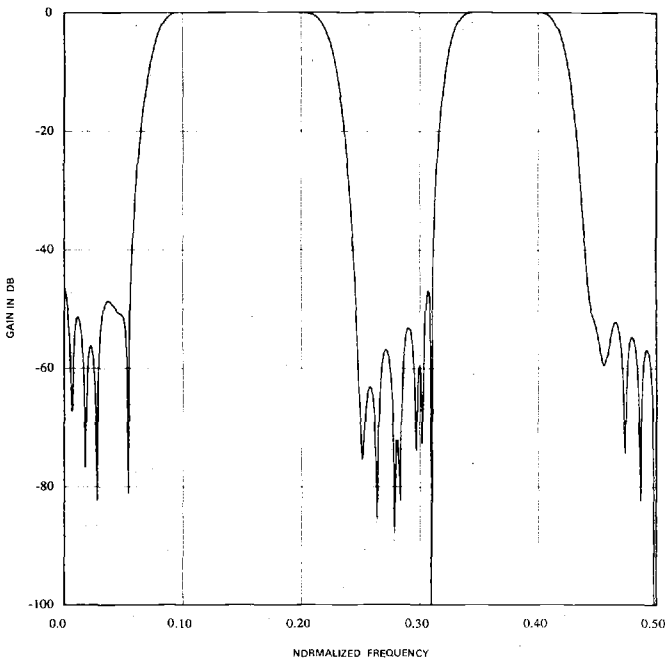

Fig. 4. 100-tap nonrecursive multiband filter (12-bit coefficients).

by the microprogram to know when to get the result and continue processing;

2) the number of positions to be shifted before each add/subtract;

3) to control the type of arithmetic operation, i.e., should an addition or subtraction be performed.

A representation that will satisfy these requirements is to use a field of $[\log_2 (B/2)]_I$ bits as a header for each coefficient which will contain the total number a_N of nonzero digits in this coefficient, followed by a_N fields of length $[\log_2 B]_I + 1$ bits in which the first $\log_2 B$ bits will determine how many positions are to be shifted and the last bit will be 0 or 1 depending on whether we should add or subtract. Therefore, the average coefficient will require

$$N_B = (B/3) \cdot [[\log_2 B]_I + 1] + [\log_2 (B/2)]_I. \quad (8)$$

Obviously, since only integer numbers are allowed for the fields, and furthermore, memory is usually available in fixed configuration, some inefficiency in memory allocation will result. Consider a specific example where B is between 9 and

TABLE VI
64-POINT FFT COSINE COEFFICIENTS IN 12 BITS

FFT COEFFICIENTS ARE EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE

1.000000000E+00	2048	1	0	0	0	0	0	0	0	0	0	0
9.951847791E-01	2038	1	0	0	0	0	0	0	0	-1	0	-1	0
9.807852310E-01	2009	1	0	0	0	0	0	-1	0	-1	0	0	1
9.569404135E-01	1960	1	0	0	0	-1	0	1	0	1	0	0	0
9.238795568E-01	1892	1	0	0	0	-1	0	-1	0	0	1	0	0
8.819213513E-01	1806	1	0	0	-1	0	0	0	1	0	0	-1	0
8.314697156E-01	1703	1	0	-1	0	1	0	1	0	1	0	0	-1
7.730106124E-01	1583	1	0	-1	0	0	1	0	-1	0	0	0	-1
7.071068668E-01	1448	1	0	-1	0	-1	0	1	0	1	0	0	0
6.343933515E-01	1299	0	1	0	1	0	0	0	1	0	1	0	-1
5.555704241E-01	1138	0	1	0	0	1	0	0	-1	0	0	1	0
4.713973406E-01	965	0	1	0	0	0	-1	0	0	0	1	0	1
3.826833573E-01	784	0	1	0	-1	0	0	0	1	0	0	0	0
2.902857078E-01	595	0	0	1	0	0	1	0	1	0	1	0	-1
1.950911268E-01	400	0	0	1	0	-1	0	0	1	0	0	0	0
9.801769268E-02	201	0	0	0	1	0	-1	0	0	1	0	0	1
3.139787741E-07	0	0	0	0	0	0	0	0	0	0	0	0	0
-9.801768632E-02	-201	0	0	0	-1	0	1	0	0	-1	0	0	-1
-1.950859579E-01	-400	0	0	-1	0	1	0	0	-1	0	0	0	0
-2.902842168E-01	-595	0	0	-1	0	0	-1	0	-1	0	-1	0	1
-3.826823638E-01	-784	0	-1	0	1	0	0	0	-1	0	0	0	0
-4.713959658E-01	-965	0	-1	0	0	0	1	0	0	0	-1	0	-1
-5.555697088E-01	-1138	0	-1	0	0	-1	0	0	1	0	0	-1	0
-6.343922828E-01	-1299	0	-1	0	-1	0	0	0	-1	0	-1	0	1
-7.071061738E-01	-1448	-1	0	1	0	1	0	-1	0	-1	0	0	0
-7.730100758E-01	-1583	-1	0	1	0	0	-1	0	1	0	0	0	1
-8.314688218E-01	-1703	-1	0	1	0	-1	0	-1	0	-1	0	0	1
-8.819267558E-01	-1806	-1	0	0	1	0	0	0	-1	0	0	1	0
-9.238794068E-01	-1892	-1	0	0	0	1	0	1	0	0	-1	0	0
-9.569399368E-01	-1960	-1	0	0	0	1	0	-1	0	-1	0	0	0
-9.807852728E-01	-2009	-1	0	0	0	0	0	1	0	1	0	0	-1
-9.951846608E-01	-2038	-1	0	0	0	0	0	0	0	1	0	1	0

NUMBER OF NONZERO DIGITS IS =127 I.E. AN AVERAGE OF = 3.969

TABLE VII
AVERAGE NUMBER OF NONZERO DIGITS FOR FFT COEFFICIENTS

B \ N	16	32	64	128	256	512	1024	2048
8	2.71	2.73	2.75	2.80	2.71	2.72	2.73	2.73
10	3.47	3.68	3.58	3.45	3.39	3.37	3.38	3.37
12	3.41	3.89	3.99	4.01	4.01	4.03	4.03	4.03
14	3.63	4.24	4.48	4.66	4.65	4.68	4.68	4.69
16	4.13	4.87	5.23	5.32	5.37	5.34	5.33	5.33

N = transform length; B = number of bits

16; 3 bits will be required for the header field and 4 + 1 = 5 bits for each nonzero digit, making $N_B = 23$, which implies an approximate doubling of the storage capacity for the coefficients. Furthermore, if we assume that the memory is available in a 8-bit wide configuration, we will have to use $N_B = 32$ which corresponds to an almost tripling of the memory required for coefficients.

Table X shows the FFT coefficients of Table VI represented in the form described above. In the next section, dealing with the architectural organization of the processor, we shall discuss how to deal with the unequal lengths required for the different coefficients as illustrated in Table X.

III. AN ARCHITECTURE FOR THE MACHINE ORGANIZATION OF DIGITAL SIGNAL PROCESSORS

Fig. 5 illustrates the functional block diagram of a digital signal processor. The basic units are as follows.

1) Random-access memory (RAM) in which the input data to be processed resides and where the output data are transferred.

2) An arithmetic unit capable of performing the operation $a \cdot x + y$.

3) High-speed scratch-pad storage for intermediate results.

4) A control section containing the string of microinstructions necessary to perform the required signal processing.

5) Memory in which the processing coefficients $\{a_i\}$ reside, which may be read-only memory (ROM) or RAM for programmable processors.

This structure is not unique. In fact it is quite similar to what is available today in microprocessors and minicomputers. The only obstacle to using these general purpose processors is that their computational rate is by far inadequate even for the more modest signal processing tasks.

For the sake of simplicity and clarity we proceed to define a machine organization in which the processing coefficients will have an accuracy of up to 16 bits, which is adequate for

TABLE VIII

SIXTH-ORDER RECURSIVE NOTCH FILTER

NUMERATOR COEFFICIENTS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING, 12 BITS

1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-1.61803461E+00	-1657	-1	0	1	0	-1	0	0	0	1	0	0	-1
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-1.61803461E+00	-1657	-1	0	1	0	-1	0	0	0	1	0	0	-1
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-1.61803461E+00	-1657	-1	0	1	0	-1	0	0	0	1	0	0	-1
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0

TOTAL NUMBER OF NONZERO DIGITS IS = 21 I.E. AN AVERAGE OF = 2.33 PER COEFFICIENT

DENOMINATOR COEFFICIENTS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING, 12 BITS

-1.51803461E+00	-1554	-1	0	1	0	0	0	0	-1	0	0	-1	0
9.21598269E-01	944	0	1	0	0	0	-1	0	-1	0	0	0	0
-1.53713288E+00	-1574	-1	0	1	0	0	0	-1	0	-1	0	1	0
9.02500000E-01	924	0	1	0	0	-1	0	1	0	0	-1	0	0
-1.58713281E+00	-1625	-1	0	1	0	-1	0	1	0	1	0	0	-1
9.21598269E-01	944	0	1	0	0	0	-1	0	-1	0	0	0	0

TOTAL NUMBER OF NONZERO DIGITS IS = 25 I.E. AN AVERAGE OF = 4.17 PER COEFFICIENT

TABLE IX

TENTH-ORDER ELLIPTIC RECURSIVE LOW-PASS FILTER

NUMERATOR COEFFICIENTS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING, 12 BITS

1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-1.41961670E+00	-1454	-1	0	1	0	0	1	0	1	0	0	1	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-1.37234500E+00	-1405	-1	0	1	0	1	0	0	0	0	1	0	-1
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-1.22251130E+00	-1252	0	-1	0	-1	0	0	1	0	0	-1	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
-7.31235500E-01	-749	0	-1	0	1	0	0	0	1	0	1	0	-1
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0
1.55649950E+00	1594	1	0	-1	0	0	1	0	0	-1	0	1	0
1.00000000E+00	1024	0	1	0	0	0	0	0	0	0	0	0	0

TOTAL NUMBER OF NONZERO DIGITS IS = 34 I.E. AN AVERAGE OF = 2.27 PER COEFFICIENT

DENOMINATOR COEFFICIENTS

FILTER COEFFICIENTS EXPRESSED IN THE CANONICAL SIGNED DIGIT CODE USING, 12 BITS

-1.50277710E+00	-1539	-1	0	1	0	0	0	0	0	0	-1	0	1
9.82435230E-01	1006	0	1	0	0	0	0	0	-1	0	0	-1	0
-1.49814610E+00	-1534	-1	0	1	0	0	0	0	0	0	0	1	0
9.36639790E-01	959	0	1	0	0	0	-1	0	0	0	0	0	-1
-1.50927350E+00	-1545	-1	0	1	0	0	0	0	0	-1	0	0	-1
8.57742310E-01	878	0	1	0	0	-1	0	0	-1	0	0	-1	0
-1.53313450E+00	-1570	-1	0	1	0	0	0	-1	0	0	0	-1	0
7.37310410E-01	755	0	1	0	-1	0	0	0	-1	0	1	0	-1
-1.55649950E+00	-1594	-1	0	1	0	0	-1	0	0	1	0	-1	0
6.26565930E-01	642	0	0	1	0	1	0	0	0	0	0	1	0

TOTAL NUMBER OF NONZERO DIGITS IS = 38 I.E. AN AVERAGE OF = 3.80 PER COEFFICIENT

most processing purposes. The arithmetic will be done in 16-bit wide registers, corresponding to multiplying and adding with a 16-bit precision. All arithmetic is in fixed point. Increasing the arithmetic precision can be easily done by using 20-bit registers and a 20-bit adder.

Let the processing coefficients be represented as described in Section II (see Table X) and assume we have a ROM 8 bits wide (e.g., Signetics 8205 512 × 8 bits or 8204 256 × 8 or 8223 32 × 8, depending on the number of coefficients required). The coefficients are stored sequentially according to their index. Each coefficient will be assigned as many locations as nonzero digits. The first location will contain the 3-bit field

TABLE X
AN EXAMPLE OF USING A SPECIALIZED REPRESENTATION TO FACILITATE THE CONTROL OF THE SHIFT MATRIX AND ADDER/SUBTRACTOR

FFT COEFFICIENTS EXPRESSED ON THE BASIS OF THE SIGNED DIGIT CODE

1	0 0 1	0 0 0 0 0						
2	0 1 1	0 0 0 0 0	1 0 0 0 1	1 0 1 0 1				
3	1 0 0	0 0 0 0 0	0 1 1 0 1	1 0 0 0 1	1 0 1 1 0			
4	1 0 0	0 0 0 0 0	0 1 0 0 1	0 1 1 0 0	1 0 0 0 0			
5	1 0 0	0 0 0 0 0	0 1 0 0 1	0 1 1 0 1	1 0 0 1 0			
6	1 0 0	0 0 0 0 0	0 0 1 1 1	0 1 1 1 0	1 0 1 0 1			
7	1 1 0	0 0 0 0 0	0 0 1 0 1	0 1 0 0 0	0 1 1 0 0	1 0 0 0 0	1 0 1 1 1	
8	1 0 1	0 0 0 0 0	0 0 1 0 1	0 1 0 1 0	0 1 1 1 1	1 0 1 1 1		
9	1 0 1	0 0 0 0 0	0 0 1 0 1	0 1 0 0 1	0 1 1 0 0	1 0 0 0 0		
10	1 0 1	0 0 0 1 0	0 0 1 1 0	0 1 1 1 0	1 0 0 1 0	1 0 1 1 1		
11	1 0 0	0 0 0 1 0	0 1 0 0 0	0 1 1 1 1	1 0 1 0 0			
12	1 0 0	0 0 0 1 0	0 1 0 1 1	1 0 0 1 0	1 0 1 1 0			
13	0 1 1	0 0 0 1 0	0 0 1 1 1	0 1 1 1 0				
14	1 0 1	0 0 1 0 0	0 1 0 1 0	0 1 1 1 0	1 0 0 1 0	1 0 1 1 1		
15	0 1 1	0 0 1 0 0	0 1 0 0 1	0 1 1 1 0				
16	1 0 0	0 0 1 1 0	0 1 0 1 1	1 0 0 0 0	1 0 1 1 0			
17	0 0 0	0 0 0 0 0						
18	1 0 0	0 0 1 1 1	0 1 0 1 0	1 0 0 0 1	1 0 1 1 1			
19	0 1 1	0 0 1 0 1	0 1 0 0 0	0 1 1 1 1				
20	1 0 1	0 0 1 0 1	0 1 0 1 1	0 1 1 1 1	1 0 0 1 1	1 0 1 1 0		
21	0 1 1	0 0 0 1 1	0 0 1 1 0	0 1 1 1 1				
22	1 0 0	0 0 0 1 1	0 1 0 1 0	1 0 0 1 1	1 0 1 1 1			
23	1 0 0	0 0 0 1 1	0 1 0 0 1	0 1 1 1 0	1 0 1 0 1			
24	1 0 1	0 0 0 1 1	0 0 1 1 1	0 1 1 1 1	1 0 0 1 1	1 0 1 1 0		
25	1 0 1	0 0 0 0 1	0 0 1 0 0	0 1 0 0 0	0 1 1 0 1	1 0 0 0 1		
26	1 0 1	0 0 0 0 1	0 0 1 0 0	0 1 0 1 1	0 1 1 1 0	1 0 1 1 0		
27	1 1 0	0 0 0 0 1	0 0 1 0 0	0 1 0 0 1	0 1 1 0 1	1 0 0 0 1	1 0 1 1 0	
28	1 0 0	0 0 0 0 1	0 0 1 1 0	0 1 1 1 1	1 0 1 0 0			
29	1 0 0	0 0 0 0 1	0 1 0 0 0	0 1 1 0 0	1 0 0 1 1			
30	1 0 0	0 0 0 0 1	0 1 0 0 0	0 1 1 0 1	1 0 0 0 1			
31	1 0 0	0 0 0 0 1	0 1 1 0 0	1 0 0 0 0	1 0 1 1 1			
32	0 1 1	0 0 0 0 1	1 0 0 0 0	1 0 1 0 0				

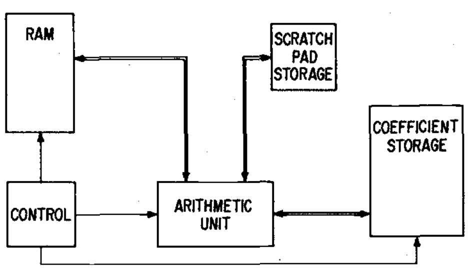

Fig. 5. Block diagram of a general purpose digital signal processor.

indicating how many nonzero digits there are a_N (a maximum of 8 for 16-bit coefficients) and the first 5-bit field indicating how many positions should be shifted and whether an addition or subtraction is required. The subsequent $(a_N - 1)$ locations contain the rest of the $a_N - 1$ 5-bit fields of this coefficient. Thus, a serious inefficiency is caused by using an 8-bit memory to store 5 bits, a better storage efficiency could be achieved if we had available memory that is 5 bits wide, thus wasting only 2 bits on the first field of each coefficient.

This arrangement of the coefficients will make it easy for the control to keep track of where the next coefficient should come from. In the control there will be a location counter pointing to the address of the next processing coefficient, and each time a new coefficient is fetched the value of a_N (the first 3-bit field) is added to the location counter to give the address of the next coefficient. This is illustrated in Fig. 6.

Fig. 6 also shows the proposed organization of the arithmetic unit which consists of the two input registers R_A and R_B (each 16 bits), a shift-matrix S capable of shifting the value in B during one clock cycle the number of positions indicated by the control register R_H which is loaded from the coefficient memory. This shift matrix can be implemented either by custom IC's containing $B^2/2 = 128$ two-input AND gates and suitable decoding to implement a mask-skew circuit, or by using such a commercially available IC as the Signetics 8243 8-bit position scaler which performs exactly the required func-

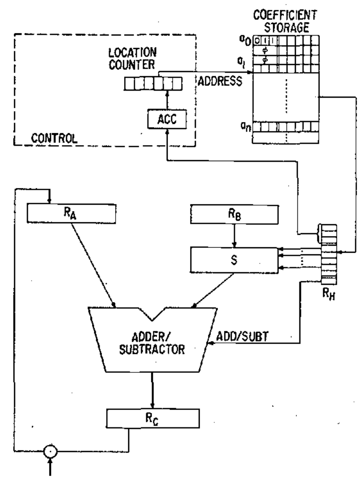

Fig. 6. Proposed organization of the arithmetic unit in a digital signal processor that uses a special representation for the coefficients.

tion for an 8-bit shift matrix (except for an inversion which will require an additional IC containing inverters) or Advanced Micro Devices AM25510 4-bit shifter. In general, such a shift matrix could be implemented using several levels ($\log_2 B$). The number of gates required will be proportional to $B \log_2 B$, rather than B^2. The arithmetic unit also has a 16-bit adder/subtracter who will add or subtract under the control of the last bit in R_H. This adder can be implemented using 4 standard 4-bit ALU's, however, they provide more functions than we need. Finally we have the output register R_C which can be loaded into either the RAM, the scratch pad, or wrapped around R_A or R_B to perform accumulation.

It is easy to see that the throughput rate of the arithmetic unit can be quite high since the shift operation is static and, depending on the number of levels in which it is achieved, will require as many gate propagation delays, which for Schotthy TTL gates will be some 20 ns for 6–8 levels, and the addition time using 5 IC's (4 ALU's as 74S181 and a look-ahead carry generator as 74S182) will require 28 ns, thus our throughput rate per shift/add will be easily 60 ns. Thus, in general, we will achieve an average multiply–add of a 12 × 16 bits with a 16-bit result in 240 ns and a 16 × 16 bits with a 16-bit result in 300 ns. This will require some nine IC's. An even higher throughput rate can be achieved by concurrent shift and add which will require a look-ahead fetch. This will double the speed to, respectively, 120 and 150 ns. Although we will give detailed comparisons later, we only mention here that using the Advanced Micro Devices AM2505 4 × 2 TTL MSI multiplier it is possible to perform a 12 × 16 bit multiplication in 275 ns using 18 IC's and a 16 × 16-bit multiplication again with a 16-bit result in 275 ns using 22 IC's. However, as we demonstrated in Section II, in many cases the average number of adds that will be required will be less than $B/3$ leading to even better results.

If large LSI is considered, since the arithmetic unit described above contains approximately 1000 gates, it could be fit on one integrated circuit. Depending upon the specific technology used, different speeds would be achieved.

It is clear that the proposed structure is quite general and can be used to implement any signal processing function we may require. The only difference from currently available

processors would be to insert or convert upon input the processing coefficients, be they filter coefficients or FFT sine/cosine values, to the representation described in Section II. Since the insertion or modification of coefficients, even in programmable signal processors is done off line, or at least at a rate several orders of magnitude slower than the real-time processing speed, this will not affect the computational rate that can be achieved.

An additional important advantage of the proposed organization is the fact that we can easily mix coefficients with different word lengths, which is in many cases desirable, and gain a computational advantage from shorter word-length coefficients. This is not the case for systems using a general multiplier.

Although the exact savings that will be achieved in a particular application depend on the specific coefficients needed in that application, the comparison made above with a multiplier and the examples that we present in Section V for an FFT processor and implementation of digital filters, tend to indicate that the proposed method will indeed yield results superior to a standard approach.

IV. A Comparison with High-Speed Multipliers

In this section we attempt to compare the performance of standard high-speed multipliers [9], [10], with the equivalent multiply/add arithmetic unit described in Section III. This comparison does not take into account the increased storage required for the coefficients, since this depends on the number of coefficients and their specific values. The examples in the next section show that the comparison results are essentially the same when this is taken into consideration.

From [11] it follows that if Schottky TTL gates are used to implement high-speed multipliers using Wallace's scheme or Dada's scheme, the time required to multiply two B-bit numbers is

$$T_M = 18.84 \log_2 B + T_A(2B) + 20 \text{ ns} \tag{9}$$

where $T_A(2B)$ is the time required for addition of $2B$-bit numbers, and the number of gates required is

$$M_G = 13B^2 - 16B \text{ gates.} \tag{10}$$

The performance obtained from the arithmetic unit proposed in Section III is

$$T_{EM} = \frac{B}{a_N} T_A(2B) \text{ ns} \tag{11}$$

$$EM_G = 33B + 3.63\, B \log_2 B \text{ gates} \tag{12}$$

where a_N is determined by the average number of nonzero digits (usually 3). The number of gates corresponds to a $2B$-bit adder with carry look ahead and a B-bit shift matrix.

For the purposes of comparison we consider the multiplication rate over the number of gates required to achieve it for both cases and define an efficiency ratio as

$$E_{M/EM} = \frac{1/T_M/M_G}{1/T_{EM}/EM_G} = \frac{T_{EM} \cdot EM_G}{T_M \cdot M_G}. \tag{13}$$

From (9)-(13) it follows that

$$E_{M/EM} = \frac{1}{a_N} \cdot \frac{T_A(2B)\,[33 + 3.63 \log_2 B]}{\left[13 - \frac{16}{B}\right][18.84 \log_2 B + 20 + T_A(2B)]}. \tag{14}$$

Addition time will be $T_A = 28$ ns for $8 < B \leqslant 32$, and letting $a_N = 3$, (14) reduces to

$$E^{(B)}_{M/EM} \cong \frac{11 + 1.21 \log_2 B}{22.29 + 8.75 \log_2 B} \qquad 8 < B \leqslant 32 \tag{15}$$

which would be the range of interest for B in digital signal processors. From (15) we find that $E_{M/EM}$ (16) = 0.28 and $E_{M/EM}$ (32) = 0.26, which implies that the proposed scheme is almost four times as efficient as the Wallace multiplier for the case of digital signal processors, where no conversion to the signed-digit code is required as described above.

Table XI lists the efficiency ratio as defined above based on a comparison with two standard multipliers; one the Advanced Micro Devices AM2505 that uses carry-save combined with examination of two multiplier bits at once, and the second a Texas Instruments implementation of a Wallace multiplier using SN74LS261, SN745275, and SN745274. It is worth mentioning that the AM2505 although slower, has the advantage that it does a multiply/add as in (7), and requires no correction for 2's complement multiplication.

V. Implementing the Processors—Some Examples

In this section we present some examples of the implementation of digital signal processors using the organization proposed in Section III and compare it to existing realizations.

We adopt a figure of merit F_T which is defined as

$$F_T = R/N_{IC} \tag{16}$$

where R is the data rate throughput in kHz and N_{IC} is the number of MSI TTL IC's required. F_T is a good measure of the cost of processing digitally, data coming in at a given rate.

A. A 1024-Point FFT Processor

For a radix-2 decimation-in-time algorithm [8], the arithmetic unit will repeatedly perform the operation

$$a'_R = a_R + (c_R \cdot \cos(2\pi k/N) + c_I \cdot \sin(2\pi k/N))$$

$$c'_R = a_R - (c_R \cdot \cos(2\pi k/N) + c_I \cdot \sin(2\pi k/N))$$

$$a'_I = a_I + (c_I \cdot \cos(2\pi k/N) - c_R \cdot \sin(2\pi k/N))$$

$$c'_I = a_I - (c_I \cdot \cos(2\pi k/N) - c_R \cdot \sin(2\pi k/N))$$

where a_R, a_I, c_R, c_I are the real and imaginary parts of the input to the "butterfly," a'_R, c'_R, c'_I, a'_I are the outputs, k is an integer depending on how many steps of the algorithm have been executed, and N is the number of points, 1024 in our case.

We assume that $B = 12$ bits are used to represent the sine/cosine coefficients, 16 bits for the data, and 20 bits are retained in the arithmetic to allow a reasonable dynamic range and accuracy.

If T is the time required to perform the computation indicated in (10) then, assuming that fast enough memory RAM is

TABLE XI
EFFICIENCY RATIO FOR TWO STANDARD MULTIPLIERS VERSUS THE PROPOSED SCHEME FOR TWO VALUES OF THE AVERAGE NUMBER OF NONZERO DIGITS

	$E_{M/EM}$ (AM2505)		$E_{M/EM}$ (SN745761)	
B	$a_N = 3$	$a_N = 5$	$a_N = 3$	$a_N = 5$
8	0.14	0.08	0.30	0.18
12	0.18	0.11	0.32	0.19
16	0.15	0.09	0.28	0.17
24	0.13	0.08	0.27	0.16
32	0.11	0.07	0.26	0.16

used for the data to keep the arithmetic unit continuously busy, a $2/(T \log_2 N)$ complex data rate throughput will be achieved if one arithmetic unit is used. By adopting a pipeline architecture for the processor [12], [13] and using $\log_2 N$ arithmetic units, a complex data rate of up to $2/T$ can be achieved.

From Table VII we see that the sine/cosine coefficients for this case average 4.03 nonzero digits. Thus to compute (7) we require an average of 20.12 adds, making $T = 1207.2$ ns. To do this we need some 12 IC's consuming 6 W. To allow more flexibility in considering various speeds of operation and to permit a fair comparison with existing realizations, we include the IC's needed for storage of the coefficients. As evident from Section III, we need 30 bits for each coefficient (taking into account the inefficiency caused by using 8-bit wide ROM), thus requiring four IC's of 512 X 8 each for storage.

The throughput rate will be $R = \cong 165$ kHz, making the figure of merit $F_T = 10.31$.

If a standard prepackaged multiplier chip such as AM2505 is used, we will need 20 IC's to multiply the 12 X 16 bit numbers and it will take 275 ns for a multiply and add. However, since we need to add and subtract a_R from the multiplication result, we are faced with the choice of adding additional hardware (a 20-bit adder) or wasting two multiply cycles, just to perform two additions. Assume we add the additional five IC's for an adder, then T in this case will be 1100 ns. Storage for the coefficients is here only two IC's. Thus the throughput rate here will be 182 kHz, making the figure of merit for this case $F_T = 6.74$, i.e., nearly two times less than in our proposed implementation.

Since we did include the memory in the arithmetic unit, F_T will remain constant when adding arithmetic units to increase the throughput rate. A fully pipelined structure having 10 arithmetic units will in both cases permit a ~1.7 MHz complex throughput rate, however, in our implementation requiring 160 IC's versus the 270 needed by the standard approach.

Finally, we mention that although it is true that a stand alone FFT processor may be required to do multiplication by other than fixed coefficients (e.g., squaring to compute the power spectrum), usually the rate at which such multiplications have to be performed is significantly slower than the FFT computations, and they can be mechanized in the proposed machine as shift/add with skip across zeros.

B. Digital Filtering

Here we consider a processor that must perform a series a filtering operations on an input data stream, which would be either a high-frequency signal, or a stream derived from multiplexing several lower rate channels. One such useful processor (a digital frequency division multiplexer/demultiplexer) is described in [4], [14].

Let us consider, therefore, a hypothetical processor that will implement, the 10th-order low-pass recursive filter of Table IX, the 56th-order nonrecursive bandpass filter of Table IV, the 100-tap multiband nonrecursive filter of Table V, and the 6th-order recursive notch filter of Table VIII. For simplicity we assume that all coefficients will use 12 bits, and data 16 bits, with 16 bits rounded arithmetic results. For this processing task a total of 102 multiplications and 188 additions must be made for each incoming data point. (Taking into account the symmetry of the nonrecursive filters and not counting as multiplications the recursive filter coefficients that are unity.) From the aforementioned tables, we see that only 361 add/subtract operations will have to be performed in the proposed implementation.

Thus the throughput rate that can be achieved by a processor using the architecture proposed in Section III is $R = 1/(361 \cdot 60) \cong 46$ kHz and the arithmetic unit will have nine IC's consuming about 4.5 W. Storing the filtering coefficients will require only one 4096-bit IC, although we will use an average of 32 bits per coefficient. This makes the figure of merit of this hypothetical processor $F_T = 4.6$.

If we consider now implementing the same processor with the prepackaged multipliers, then 18 IC's are needed to perform the multiplications, one IC for coefficient storage, and four IC's for an adder to account for the 86 more add's than multiplies. The throughput rate $R = 1/(102 \cdot 275) = 36$ kHz, making the figure of merit for this case $F_T = 1.57$, which is nearly three times less than in the proposed realization. The even better performance here is due to the lower average of nonzero digits for filter coefficients than for FFT coefficients.

VI. CONCLUSION

We have proposed a machine organization for the implementation of dedicated hardware digital signal processors that is based on a specialized representation of the processing constants. This organization is shown to be highly modular and well suited to integrated circuit implementation. Upon comparison with existing realizations, the proposed implementation was shown to be significantly more efficient in terms of achievable computing power per hardware expenditure. This is done without any loss of generality, and only at the expense of a fixed increase in storage requirements for the coefficients.

Finally, we mention that a recent approach to the hardware implementation of digital signal processors that also does not require multipliers was suggested recently by B. Liu and this author [15], [16], and the approach proposed in this paper are essentially complementary. While the former approach

addresses mainly high-speed processing applications and the increase in speed is achieved at the expense of an exponential increase in coefficient storage requirements, the approach proposed in this paper is intended mainly for low- and medium-speed applications and requires only a constant increase in coefficient storage.

In the relatively short time that digital signal processing has evolved it has attracted increased attention and found ever widening applications. Numerous approaches to the hardware implementation of such processors have appeared, most notably the pioneering work by Jackson *et al.* [17]. These have opened up new options and widened the areas in which digital signal processing has become an effective and economical means of performing a variety of tasks. It is hoped that this paper will offer yet another powerful and efficient option to the system designer, whose ultimate responsibility it is to choose the most cost-effective approach to satisfy his specific requirements.

Acknowledgment

The author wishes to thank his colleagues J. Cocke and A. Chang for introducing him to the canonical signed digit code and the stimulating discussions on the subject matter.

References

[1] J. F. Kaiser, "The digital filter and speech communication," *IEEE Trans. Audio Electroacoust. (Special Issue on Speech Communication and Processing–Part II)*, vol AU-16, pp. 180–183, June 1968.

[2] D. Silverman, "The digital processing of seismic data," *Geophysics*, vol. XXXII, pp. 998–1002, Dec. 1967.

[3] R. R. Shivley, "A digital processor to generate spectra in real time," *IEEE Trans. Comput.*, vol. C-17, pp. 485–491, May 1968.

[4] S. L. Freeny, R. B. Kierburtz, K. V. Mina, and S. K. Tewksbury, "Systems analysis of a TDM-FDM translator/digital *A*-type channel bank," *IEEE Trans. Commun. Technol. (Special Issue on Signal Processing for Digital Communications–Part I)*, vol. COM-19, pp. 1050–1059, Dec. 1971.

[5] G. W. Reitweisner, "Binary arithmetic," in *Advances in Computers*, vol. 1, F. L. Alt, Ed. New York: Academic, 1960, pp. 232–308.

[6] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "Model 91 floating point execution," *IBM J. Res. Develop.*, vol. 11, pp. 34–53, Feb. 1967.

[7] H. L. Garner, "Number systems and arithmetic," in *Advances in Computers*, vol. 6, F. L. Alt, Ed. New York: Academic, 1965, pp. 163–168.

[8] B. Gold and C. M. Rader, *Digital Processing of Signals*. New York: McGraw-Hill, 1969, pp. 173–196.

[9] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, Feb. 1964.

[10] R. C. Ghert, "A 2's complement digital multiplier the AMS505," Advanced Micro Devices, Sunnyvale, CA, Application Note.

[11] A Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. Comput.* (Short Notes), vol. C-19, pp. 153–157, Feb. 1970.

[12] H. L. Gorginsky and G. A. Works, "A pipeline fast Fourier transform," *IEEE Trans. Comput.*, vol. C-19, pp. 1015–1019, Nov. 1970.

[13] M. J. Corinthios, "The design of a class of fast Fourier transform computers," *IEEE Trans. Comput.*, vol. C-20, pp. 617–623, June 1971.

[14] M. G. Bellanger and J. L. Daguet, "TDM-FDM transmultiplexer: Digital polyphase and FFT," *IEEE Trans. Commun. (Special Issue on Communications in Europe)*, vol. COM-22, pp. 1199–1204, Sept. 1974.

[15] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456–462, Dec. 1974.

[16] B. Liu and A. Peled, "A new hardware realization of high-speed fast Fourier transforms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-23, pp. 543–547, Dec. 1975.

[17] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to the implementation of digital filters," *IEEE Trans. Audio Electroacoust. (Special Issue on Digital Filters: The Promise of LSI Applied to Signal Processing)*, vol. AU-16, pp. 413–421, Sept. 1968.