ON COMPARING HARDWARE IMPLEMENTATIONS OF FIXED POINT DIGITAL FILTERS

M. Arjmand and *R. A. Roberts*

Department of Electrical Engineering
UNIVERSITY OF COLORADO
Boulder, Colorado 80309

ABSTRACT

Distributed arithmetic structures are an alternative to the use of conventional multipliers in hardware implementations of digital filters. This paper compares the various methods of using distributed arithmetic in implementing fixed point digital filters. We introduce as a measure of hardware complexity the chip area needed to fabricate competing designs using nMOS technology. Comparisons of alternate realizations in hardware are also made. These comparisons are based on equating the output signal quality of each design. We show that traditional measures of complexity (such as the number of multipliers per output sample) do not agree with complexity measures based on chip area.

INTRODUCTION

The synthesis of a digital filter includes a number of steps. These are shown schematically in Figure 1. From a set of design specifications one first obtains an external characterization of the filter usually in the form of a transfer function H(z). This process can be carried out by any one of several methods and is often done by a computer program. Given the transfer function of the desired digital filter, the next step is to choose a realization of the transfer function. A realization is a specification of the internal structure of the filter and can be given in terms of a signal flow graph or (assuming there are no delay free loops) in terms of a state variable model {A,B,C,D} as defined in (1).

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k) \qquad (1)$$

In (1), x(k) is the vector with components equal to the contents of the internal storage registers (unit delays) of the filter.

How does one choose a particular realization for a given H(z)? In the design of fixed point digital filters the choice usually is based on minimizing the effects of finite register lengths. These effects include roundoff noise, coefficient sensitivity, overflow oscillations, and zero input limit cycles [1-5]. After synthesizing a particular realization {A,B,C,D}, one is faced with another design decision. What hardware implementation should be

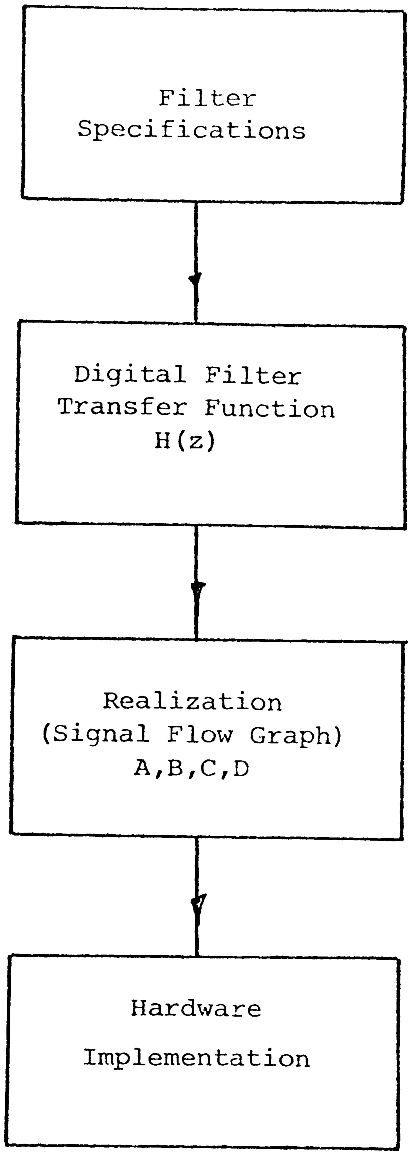
Filter Specifications
Digital Filter Transfer Function H(z)
Realization (Signal Flow Graph) A,B,C,D
Hardware Implementation

Figure 1. The design steps in symthesizing a digital filter for hardware implementation.

used? This choice is based on two primary factors: speed or data throughput and hardware cost or complexity.

Hardware designs which use monolithic multipliers offer few alternatives beyond rearrangement of the basic processes of multiplication, addition, and storage. Recognizing that a digital processor is a finite-state machine leads to another form of hardware implementation called distributed arithmetic implementation [6-7]. In these realizations the processes of multiplication and addition are replaced by a table look-up of precalculated partial products. This can be viewed as a trade of conventional multipliers for memory storage.

There are many alternative distributed arithmetic designs for a given realization $\{A,B,C,D\}$ [6-7]. The purpose of this paper is to compare several possible distributed arithmetic implementations against the usual multiplier designs for a second-order digital filter.

DISTRIBUTED ARITHMETIC IMPLEMENTATIONS

By way of background, consider the basic computation of an inner product of the form

$$y = \sum_{i=1}^{L} a_i x_i \tag{2}$$

In any digital implementation of (2), all scalars will be stored in finite length registers of B bits. If we assume x is represented in a fixed-point binary representation such as 2's complement, we can write (2) in the form

$$y = \sum_{i=1}^{L} a_i \sum_{k=0}^{B-1} 2^{-k} x_i^k$$

where x_i^k is the k—th bit of x_i (either 0 or 1). Interchanging summations gives

$$y = \sum_{k=0}^{B-1} 2^{-k} \sum_{i=1}^{L} a_i x_i^k \tag{3}$$

The inner sum over i can be interpreted as a function of ϕ of L binary arguments (the x_i^k). That is,

$$\phi(x^1, x^2, \ldots, x^L) = \sum_{i=1}^{L} a_i x_i^k \tag{4}$$

the function ϕ takes on 2^L possible values which are all the possible sums of the a_i's. We can thus rewrite our original inner product (1) as

$$y = \sum_{k=0}^{B-1} 2^{-k} \phi(x^1, x^2, \ldots, x^L) \tag{5}$$

Equation (5) has several possible hardware implementations. Perhaps the simplest consists of the following scheme. The 2^L values associated with ϕ are precalculated and stored in memory. The data vector x is used to generate an address for each of the B bit positions starting from the lowest order bit. The address is used to obtain the correct value of ϕ for each bit position. These B values of ϕ are then accumulated with the appropriate right shift as dictated by the factor 2^{-k} in (5). After B accumulations the result is y. Figure 2 is an implementation of (5) for $L = 3$.

There are many variations of this basic scheme [7]. The alternatives are combinations of two simple observations. These are:

(a) The sum of L terms in (2) can be broken down into several smaller sums, say, k sums of M_i terms each, $i=1, 2, \ldots, K$, with $L = \sum_{i=1}^{K} M_i$.

(b) The function ϕ can be addressed with one or more bits from each data word x_i on each clock cycle.

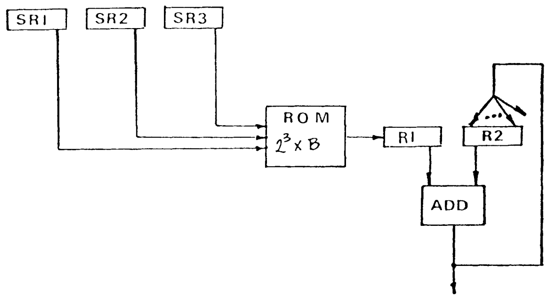

Figure 2. A distributed arithmetic structure for evaluation of a single state equation in (6). This implementation corresponds to realization #1 in Tables 1 and 3.

The partitioning of the sum of L terms into K sums of M_i terms reduces the memory required to store ϕ from 2^L words to $\sum_{i=1}^{K} 2^{M_i}$ words. This can be a significant savings in the storage needed to implement (2). For example, if $L = 10$ the storage needed for the direct implementation of a single sum is $2^{10} = 1024$ words. If, on the other hand, the sum is broken into 2 sums of 5 terms each, the storage needed is $2 \times 2^5 = 64$ words. The cost for this partitioning is an extra addition to add the results of the two sums together. This requires one additional clock cycle to obtain the final result.

Addressing the memory storage with more than one bit from each data sample x_i is another method of implementation. This is effectively pre-computing a larger table of partial results. The larger table increases the speed of the processor at the expense of an exponential growth in the memory size. If we address the memory with p bits from each data word and there are N groups in each register ($Np=B$), the required size of the memory is 2^{PL} words. If we have also partitioned the original sum of L terms into K sums of M_i terms, $i = 1, 2, \ldots, K$, the size of the memory is $\sum_{i=1}^{K} 2^{PM_i}$ words, where $L = \sum_{i=1}^{K} M_i$ and each address consists of P bits from each data word x_i. Figure 3 depicts a distributed arithmetic structure for the case $L = 3$, $K = 2$, and $P = 2$ for the computation in (2).

Of the many possible implementations using distributed arithmetic, which implementation is the fastest and least complex for a digital filter? How does an implementation using monolithic multipliers compare to the distributed arithmetic structures in terms of speed and complexity?

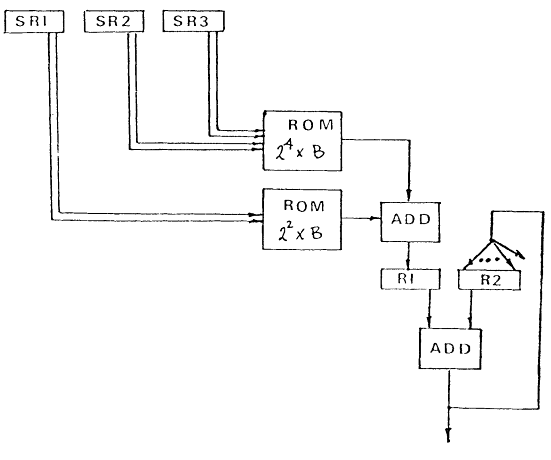

Figure 3. A distributed arithmetic structure for evaluation of a single state equation in (6). This implementation corresponds to realization #12 in Tables 1 and 3.

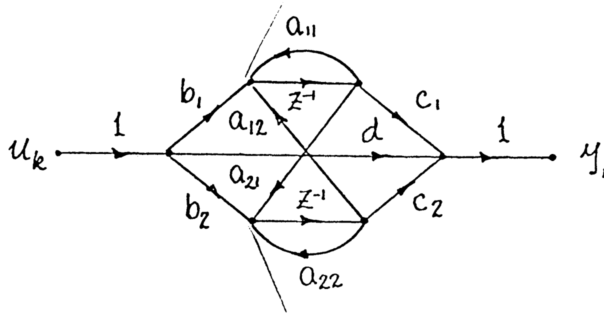

Figure 4. The signal flow graph for a general second-order state variable digital filter.

THE IMPLEMENTATION OF SECOND-ORDER DIGITAL FILTERS

In order to compare various hardware implementations, let's consider the problem of implementing a second-order digital filter. The first question we must answer is what realization $\{A,B,C,D\}$ should we choose to implement? The general state-space representation of a second order digital filter is given in (6), and the corresponding signal flow graph is shown in Figure 4.

$$x_1(k+1) = a_{11}x_1(k) + a_{12}x_2(k) + b_1u(k)$$

$$x_2(k+1) = a_{21}x_1(k) + a_{22}x_2(k) + b_2u(k)$$

$$y(k) = c_1x_1(k) + c_2x_2(k) + du(k) \qquad (6)$$

where $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $B = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, $C = [c_1\ c_2]$, $D = d$.

The commonly used direct form is included in this class by setting $a_{11} = 0$, $a_{12} = 1$, $b_1 = 0$. The direct form, in general, has the smallest number of multipliers. However, finite register effects are the most severe for these structures. Realizations in the form of (6) can be synthesized to minimize roundoff noise, reduce coefficient sensitivity, and eliminate autonomous overflow oscillations [4,8,9]. For these reasons we will base our comparisons on a realization $\{A,B,C,D\}$ given by (6) which requires, in general, 9 noninteger multiplies for each output sample y(k) instead of the 6 multiplies used by the direct form.

COMPARISONS OF DISTRIBUTED ARITHMETIC STRUCTURES

In order to determine the best distributed arithmetic structure and to compare these structures with the more usual multiplier structure, we must choose a method for measuring the complexity of a hardware implementation. We have used two methods for measuring the hardware complexity of our designs.

The first measure is based on the cost of off-the-shelf IC's (Schottky TTL chips). This analysis is, of course, dependent on current technology and the pricing structure of various IC's. Since our objective is to obtain an ordering of the various implementations, the absolute numbers are not of primary importance. It is rather the relative size of the numbers that matters. In order to include the speed of processing of each implementation in our comparisons, we shall use as a figure of merit F_1, where

F_1 = (cost of the implementation) ·
(time to compute an output sample) (7)

The results of this comparison are summarized in Table 1. All numbers pertain to implementing one of the equations in (6). An implementation of the complete filter is essentially a replication of the implementation of one equation (three times).

There are 19 distributed arithmetic structures compared in Table 1. The structures are the result of combinations of partitioning the sum of three terms into smaller sums and the use of one or more bits as the address for the memory storage of the ϕ function. We assume the word length is 12 bits. P is the number of bits in a word used to address the memory. N is the number of groups in each word so that NP is always 12. We have restricted P to those values that divide 12 with no remainder. In principle this restriction is not necessary. One could, for example, address the ϕ function in two steps; first with 7 bits and then with 5 bits, add the results with the correct shift, 2^{-7}, to obtain y in two clock cycles. It is easy to show, however, that this latter method is more complex than using first 6 bits, followed by 6 bits again. To address with 7 bits followed by 5 bits requires a memory of $(2^7 + 2^5)$ words. If one uses 6 bits, followed by 6 bits to address the memory, the memory required is reduced to $(2^6 + 2^6)$ words. Thus, since both schemes require the same amount of time, the scheme using the smallest amount of memory is better.

The implementations can be subdivided into groups which are characterized by the number of sums. There are three possibilities. One sum of three terms (A), three "sums" of one term (B), and two sums consisting of one and two terms (C). In addition there are three implementations which are highly parallel and do not fit into the above three categories. These are entries 17, 18 and 19.

There are some obvious limitations of this analysis based on a figure of merit that employs cost as a measure of complexity. Because of our desire to increase speed as much as possible, we have used high speed Schottky TTL ROM's. The largest size memory currently available on a single chip with 50 nanosecond access time has a maximum of 10 address lines.

If $PL > 10$ (where $L = \sum_{i=1}^{B} M_i$), several memory chips must be connected together with an enabling pulse generated by an address decoder to select the correct memory address. This arrangement increases memory access time and the cost over a single chip memory with the correct number of address lines. Entries 4-6, 11, 15 and 16 in Table 1 suffer from this constraint.

	Configuration			Pkg.	Speed	Cost	F_1	Ranking
					(word rate)		time · cost to compute	
	No.	N	P	Count	MHz	$	one word	F_1
	1	12	1	21	2.56	43.40	16.92	1
	2	6	2	22	2.86	68.04	23.81	3
	3	4	3	22	3.08	80.49	26.16	6
A	4	3	4	44	3.03	260.00	85.80	15
	5	2	6	603	4.545	13,183.00	2900.00	17
	6	1	12	154×10^6	6.58	3440×10^6	522.8×10^6	19
	7	6	2	33	2.86	120.93	36.03	12
	8	4	3	33	4	120.93	30.23	9
B	9	3	4	33	5	120.93	24.19	4
	10	2	6	36	4.76	140.88	29.58	8
	11	1	12	102	6.67	716.73	107.51	16
	12	6	2	27	2.38	73.16	30.73	10
	13	4	3	28	4.0	97.81	24.45	5
	14	3	4	28	5.0	97.81	19.56	2
	15	2	6	48	3.97	283.36	71.41	14
C	16	1	12	38×10^3	8.43	840×10^3	1.008×10^6	18
	17			43	8.0	226.74	28.34	7
	18			87	7.69	388.50	50.51	13
	19			57	7.69	249.78	32.47	11

Table 1

A: 1 group of 3 registers
B: 3 groups of 1 register
C: 2 groups containing 1 and 2 registers, respectively

For IIR digital filters, the recursive nature of the equations in (6) means one must compute an entire word before reloading the data registers to generate the next address for the stored ϕ function. Some overlapping of computations can be accomplished in some of the parallel structures by including additional registers in the adder tree which is needed whenever the sum of three terms in (6) is broken into smaller sums. By including these additional registers, the shift and add process can be performed concurrently with the table look-up. The details of this pipelining depend on the actual distributed arithmetic design selected. We've included this pipelining into the designs presented here.

Comparisons Based on the Figure of Merit F_1

An examination of Table 1 indicates that the best distributed arithmetic structure is the design that employs a single bit from each of three data registers to address a stored ϕ function of $2^3 = 8$ entries. The designs that use more than one bit from each data register to increase the data throughput cannot make up in speed for the more complex hardware needed.

Entries 17, 18 and 19 deserve a special word of explanation. These three realizations are highly parallel implementations and offer the designer the highest word rates with a slight premium in complexity. (Entry 16 is also very fast, but much too costly. We've included it only to indicate the large variations that are possible with distributed arithmetic structures.) If one is interested primarily in data throughput without regard to cost, these realizations are good candidates. Entries 17 and 18 are shown in Figure 5 and 6, respectively. Entry 19 is similar except that 2 bits from each register go to each ROM, resulting in 6, 64 x 12 ROM's.

One obvious question at this point is how do these distributed arithmetic structures compare to a more conventional multiplier structure? Again assuming we wished to realize the recursive equations (8) with a 12 bit word length, we could realize the state equations in two ways. One method would be to employ three multipliers working in parallel. Disregarding the control circuitry, this realization would result in a figure of merit of $F_1 = 84.15$ — five times as poor as the best distributed arithmetic structure. Using one multiplier multiplexed in time increases the figure of merit to 94.5. In either case, the distributed arithmetic structures are more attractive. The multiplier structures do offer one very important advantage — they are more flexible. The flexibility of distributed arithmetic structures is enhanced by using RAM instead of ROM. We have asumed the use of ROM to increase speed and reduce complexity.

Chip Area as a Measure of Hardware Complexity

As a second method of comparing hardware structures, we propose to measure complexity of hardware implementations by the chip area needed to fabricate competing designs in an nMOS process. This measure of complexity has the appealing feature that all designs are reduced to their essential building blocks. This measure of complexity depends on the particular technology used in fabrication. However, relative sizes of the areas are approximately the same for different technologies. Table 2 summarizes the amount of chip area needed for various functions in nMOS. We've added 40% of the area for interconnections. (This is a standard design rule-of-thumb for interconnection area.)

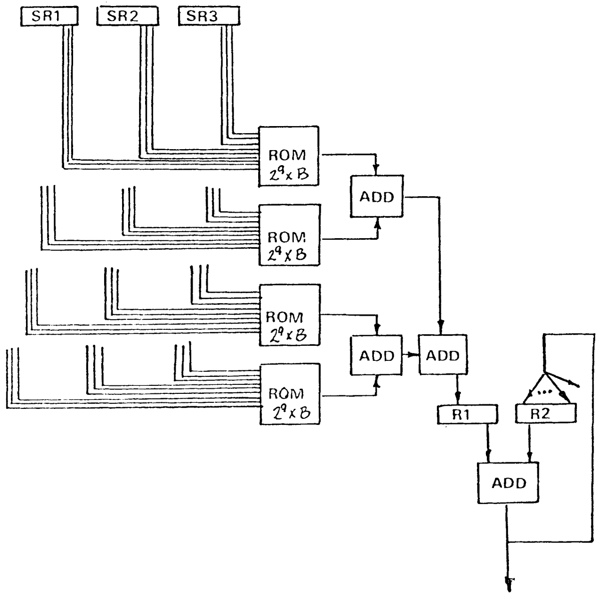

Figure 5. A distributed arithmetic structure corresponding to realization #17 in Tables 1 and 3. This highly parallel structure contains 4 ROM's which store 2^9 x B bits resulting in a total memory of 4608 x B bits. (B is the number of bits in a data word. This structure implements a single state equation in (6).)

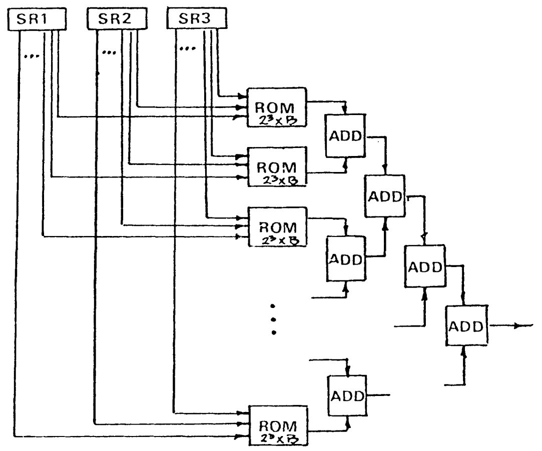

Figure 6. A distributed arithmetic structure corresponding to realization #18 in Tables 1 and 3. This structure contains 12 ROM's of 8 x B bits each. The total memory is 96 x B bits. (This structure implements a single state equation in (6).)

Table 2

nMOS Area Needed to Implement Some Typical Hardware Functions

Function Description	nMOS Area Needed
Inverter	.002 mm^2
1 Bit Full Adder	.031 mm^2
Bonding Pad	.09 mm^2
8 Bit Parallel Register	.045 mm^2
8 Bit Full Adder	.29 mm^2
32 Bit ROM	.36 mm^2
16 x 16 Bit Multiplier	1.96 mm^2

Table 3 summarizes the comparisons in terms of the figure merit F_2 where

F_2 = (time to compute one word) • (chip area in nMOS) (8)

There are some changes in the orderings of the top distributed arithmetic realizations. However, the changes are minor. The N = 12, P = 1 realization is the best implementation under both figures of merit. The top five realizations as a group under both measures are the same, but for one exception. Interestingly, it appears that the off-the-shelf price of a chip is closely related to the chip area. Entries 5, 6 and 16 indicate that there can be very large variations in complexity using distributed arithmetic structures.

	Configuration No.	N	P	Speed Word rate (MHz)	Chip Area (mm^2)	F_2 Time • area	Rankings (F_2)	(F_1)
A	1	12	1	2.56	.84	.3276	1	1
	2	6	2	2.86	1.3125	.4594	4	3
	3	4	3	3.08	5.0925	1.655	12	6
	4	3	4	3.03	35.3325	11.66	16	15
	5	2	6	4.545	2213	486	17	17
	6	1	12	6.58	579x10^6	88x10^6	19	19
B	7	6	2	2.86	1.8785	.6575	7	12
	8	4	3	4.	1.98	.4950	6	10
	9	3	4	5.	2.1825	.4365	3	4
	10	2	6	4.76	3.3975	.7135	8	8
	11	1	12	6.67	36.3375	5.45	14	16
C	12	6	2	2.38	1.4438	.4043	2	10
	13	4	3	4.	1.8825	.4706	5	5
	14	3	4	5.	3.9525	.7905	9	2
	15	2	6	3.97	36.375	9.1665	15	14
	16	1	12	8.43	141x10^3	16.17x10^3	18	18
	17			8.	19.425	2.4281	13	7
	18			7.69	6.435	.8366	11	13
	19			7.69	6.255	.8132	10	11

Table 3

A: 1 group of 3 registers
B: 3 groups of 1 register each
C: 2 groups containing 1 and 2 registers, respectively

COMPARISONS OF DIGITAL FILTER STRUCTURES BASED ON F_2

We have compared the implementation of state variable structures as given by (6) using monolithic multipliers and distributed arithmetic. There are any number of digital filtering structures we could use as a basis for comparison. We could ask which structure among all structures implemented using distributed arithmetic has the best figure of merit F_2? This question cannot be answered without some additional constraints. The essential problem is this: the realizations one wishes to implement have varying internal noise properties (caused by finite realization length). Thus given two realizations one should adjust the word lengths so they have the same output noise caused by finite register length. These two realizations (with perhaps different word lengths) implemented using distributed arithmetic (or some other method) give a fair comparison of the hardware complexity of the two realizations.

This method of comparison will depend on the external specifications of the filter. We know that finite register effects become more severe as the poles of the filter are brought closer together. A narrowband low pass filter is the most difficult filter to implement in terms of finite length register effects [1,2]. For example, let's assume we wish to build a sixth-order Butterworth filter with a bandwidth of 2% of the foldover frequency $f_s/2$. Let's compare a cascade of second-order direct forms with a cascade of second-order optimal low-noise forms [1,2]. (These structures have the same topology as those in (6).) Assuming a word length of 12 bits for the direct form, the corresponding low-noise forms can use 4 bits/register less and obtain the same noise performance. (This figure can be obtained from the graphs in [2].)

The direct form is simpler in terms of the number of multipliers used. However, the word lengths are longer. The word length difference increases as the bandwidth of the filter decreases. Using F_2 as a figure of merit for the direct form we obtain

$$\text{nMOS area} = 1.478 \text{ mm}^2$$
$$\text{word rate} = 390 \text{ nsec}$$
$$\therefore F_2 \text{ (Direct Form)} = .576$$

The above figure of merit is for a single isolated second-order section implemented using distributed arithmetic.

The corresponding figure of merit for the low-noise structure is

$$\text{nMOS area} = 1.388 \text{ mm}^2$$
$$\text{word rate} = 270 \text{ nsec}$$
$$\therefore F_2 \text{ (Low Noise Form)} = .375$$

And so, even though the direct form has three fewer multipliers, the difference in word lengths means the low-noise structure is less complex (in terms of chip area). This advantage will disappear as the bandwidth of the filter is increased. The crossover point in complexity between these two different realizations occurs for a bandwidth of approximately 5% of the foldover frequency. The complexity advantage of the low-noise structures vis-a-vis direct forms is reduced as bandwidth increases because the internal roundoff noise is less severe. This means the difference in word lengths of the two structures is reduced as bandwidth of the filter increases. Table 4 relates the roundoff noise performance difference between these two forms in terms of bits saved.

Table 4

The number of bits saved as a function of filter bandwidth for low-noise structures as compared to the direct form structure

Bandwidth as a percentage of the foldover frequency	Number of bits saved using optimal low noise structures vis-a-vis direct form structures
50%	.17
20%	.87
10%	1.67
5%	2.39
2%	4.35
1%	5.00
.5%	5.55

This comparison, we believe, introduces an idea that has heretofore been ignored in comparing the complexity of hardware structures. One cannot meaningfully compare hardware structures without including some measure of signal quality (which quantifies the effects of finite register length). Traditional measures of complexity (such as the number of multiplies per output sample) may not be adequate as this example demonstrates.

SUMMARY

There are many methods of hardware implementation of a given realization (signal flow graph) for a digital filter. We have compared, in some detail, distributed arithmetic implementations of an IIR, state-variable, second-order digital filter. Our comparisons are based on two figures of merit which attempt to quantify the speed and hardware complexity of a given implementation. Our results indicate that distributed arithmetic implementations are better then the more usual implementations based on monolithic multipliers.

We have also considered comparisons of hardware implementation of different realizations of the same digital filter. This comparison suggests that one must quantify the signal quality of a realization before one attempts to compare the hardware complexity.

REFERENCES

1. C. T. Mullis and R. A. Roberts, "Synthesis of Minimum Roundoff Noise Fixed-Point Digital Filters," *IEEE Trans. on Circuits and Systems,* Vol. CAS-23, pp 551-561, Sept. 1976.
2. C. T. Mullis and R. A. Roberts, "Roundoff Noise in Digital Filters: Frequency Transformations and Invariants," *IEEE Trans. on Acoustics, Speech, and Signal Processing,* Vol. ASSP-24, pp 538-549, Dec. 1976.
3. L. B. Jackson, A. G. Lindgren, and Y. Kin, "Optimal Synthesis of Second-Order State Space Structures for Digital Filters," *IEEE Trans. on Circuits and Systems,* Vol. CAS-26, No. 3, pp 149-153, March 1979.
4. C. W. Barnes, "Roundoff Noise and Overflow in Normal Digital Filters," *IEEE Trans. on Circuits and Systems,* Vol. CAS-26, No. 3, pp 154-159, March 1979.
5. S. Y. Hwang, "Minimum Uncorrelated Unit Noise in State Space Digital Filtering." *IEEE Trans. on Acoustics, Speech, and Signal Processing,* Vol. ASSP-25, pp 273-281, August 1977.
6. A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing,* Vol. ASSP-22, pp 456-462, Dec. 1974.

7. C. S. Burrus, "Digital Filter Structures Described by Distributed Arithmetic," *IEEE Trans. on Circuits and Systems,* Vol. CAS-24, pp 674-680, Dec. 1977.
8. W. L. Mills, C. T. Mullis, and R. A. Roberts, "Digital Filter Realizations without Overflow Oscillations," *IEEE Trans. on Acoustics, Speech, and Signal Processing,* Vol. ASSP-26, No. 4, pp 334-338, Aug. 1978.
9. L. B. Jackson, "Roundoff Noise Bounds Derived from Coefficient Sensitivities for Digital Filters," *IEEE Trans. on Circuits and Systems,* Vol. CAS-23, pp 481-485, Aug. 1976.

Masud M. Arjmand was born on January 27, 1953, in Khewra, Pakistan. He received his B.S. degree in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1975. In 1975-76 he was with Engineers International Limited, Lahore, Pakistan, where he was involved in the planning and design of power-line-carrier communication systems for a large utility network. He then joined the University of Colorado at Boulder where he received his M.S. and Ph.D. degrees, both in electrical engineering, in 1979 and 1981, respectively. He is currently with the Central Research Laboratories of Texas Instruments in Dallas. His research interests include finite word-length effects and development of efficient hardware implementations for digital signal processing.

Richard A. Roberts received his B.S. degree from the University of California, Berkeley, in 1957. In 1961 and 1965 he received the M.S. and Ph.D. degrees from the University of Michigan, repectively. He is presently Professor of Electrical Engineering at the University of Colorado, Boulder. His research interests include digital signal processing structures for implementation in VLSI, applications of digital signal processing to problems in biomedical data processing, and spectral estimation.