An Approach to the Implementation of Digital Filters

LELAND B. JACKSON, Member, IEEE
JAMES F. KAISER, Associate Member, IEEE
HENRY S. McDONALD, Member, IEEE
Bell Telephone Laboratories, Inc.
Murray Hill, N. J.

Abstract

An approach to the implementation of digital filters is presented that employs a small set of relatively simple digital circuits in a highly regular and modular configuration, well suited to LSI construction. Using parallel processing and serial, two's-complement arithmetic, the required arithmetic circuits (adders and multipliers) are quite simple, as are the remaining circuits, which consist of shift registers for delay and small read-only memories for coefficient storage. The arithmetic circuits are readily multiplexed to process multiple data inputs or to effect multiple, but different, filters (or both), thus providing for efficient hardware utilization. Up to 100 filter sections can be multiplexed in audio-frequency applications using presently available digital circuits in the medium-speed range. The filters are also easily modified to realize a wide range of filter forms, transfer functions, multiplexing schemes, and round-off noise levels by changing only the contents of the read-only memory and/or the timing signals and the length of the shift-register delays. A simple analog-to-digital converter, which uses delta modulation as an intermediate encoding process is also presented for audio-frequency applications.

Manuscript received May 6, 1968.

Introduction

The basic theory underlying the analysis and design of digital filters is well advanced (although by no means complete) and quite a few summaries of theoretical results are now available in the engineering literature [1]–[4]. However, the impact of digital filtering theory has not yet been felt by most of the engineers and technicians who design and use the wide variety of filters presently constructed from RLC or crystal circuits. This has been due, in part, to a general unawareness of the possibilities of digital filtering and also, until recently, to the prohibitive complexity and cost of constructing most digital filters. Hence, digital filter implementation has been confined primarily to computer programs for simulation work or for processing relatively small amounts of data, usually not in real time. However, the rapid development of the integrated-circuit technology and especially the potential for large-scale integration (LSI) of digital circuits now promise to reverse this situation in many instances and to make many digital filters more attractive than their analog counterparts, from the standpoints of cost, size, and reliability.

In this paper, we present an approach to the physical implementation of digital filters which has the following features.

1) The filters are constructed from a small set of relatively simple digital circuits, primarily shift registers and adders.

2) The configuration of the digital circuits is highly modular in form and thus well suited to LSI construction.

3) The configuration of the digital circuits has the flexibility to realize a wide range of filter forms, coefficient accuraices, and round-off noise levels (i.e., data accuracies).

4) The digital filter may be easily multiplexed to process multiple data inputs or to effect multiple, but different, filters with the same digital circuits, thus providing for efficient hardware utilization.

After a brief review of general digital filter forms, the advantages of serial, two's-complement, binary arithmetic in the implementation of digital filters are discussed. The required arithmetic circuits (adder/subtractor, complementer, and multiplier) are then described, followed by the techniques for multiplexing. Finally, several examples are presented of multiplexed digital filters that have been constructed and tested. A description of a simple analog-to-digital converter for relatively low-frequency applications is also included.

Canonical Forms

The transfer characteristics of a digital filter are commonly described in terms of its z-domain transfer function [1],

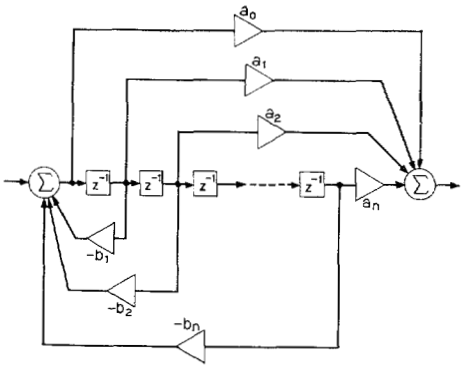


Fig. 1. The direct form for a digital filter.

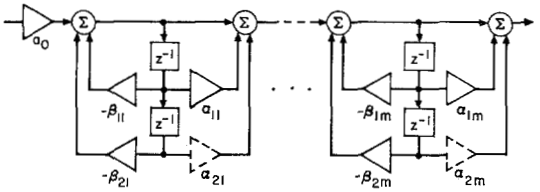


Fig. 2. The cascade form for a digital filter.

$$H(z) = \frac{\sum_{i=0}^{n} a_i z^{-i}}{1 + \sum_{i=1}^{n} b_i z^{-i}}, \qquad (1)$$

where z^{-1} is the unit delay operator. There are a multitude of equivalent digital circuit forms in which (1) may be realized, but three canonical forms, or variations thereof, are most often employed. These forms are canonical in the sense that a minimum number of adders, multipliers, and delays are required to realize (1) in the general case. The first of these forms, shown in Fig. 1, is a direct realization of (1) and as such is called the *direct form*. It has been pointed out by Kaiser [5] that use of the direct form is usually to be avoided because the accuracy requirements on the coefficients $\{a_i\}$ and $\{b_i\}$ are often severe. Therefore, although the implementation techniques presented here are applicable to any filter form, we will not specifically consider the direct form.

The second canonical form corresponds to a factorization of the numerator and denominator polynomials of (1) to produce an $H(z)$ of the form

$$H(z) = a_0 \prod_{i=1}^{m} \frac{\alpha_{2i} z^{-2} + \alpha_{1i} z^{-1} + 1}{\beta_{2i} z^{-2} + \beta_{1i} z^{-1} + 1}, \qquad (2)$$

where m is the integer part of $(n+1)/2$. This is the *cascade form* for a digital filter, depicted in Fig. 2. Second-order factors (with real coefficients) have been chosen for (2) rather than a mixed set of first- and second-order factors for real and complex roots, respectively, to simplify the implementation of the cascade form, especially when multiplexing is employed. If n is odd, then the coefficients α_{2i} and β_{2i} will equal zero for some i. The α_{2i} multipliers are shown in dotted lines in Fig. 2, because for the very common case of zeros on the unit circle in the z-plane (corresponding to zeros of transmission in the frequency response of the filter) the associated α_{2i} coefficients are unity. Thus, for these α_{2i} coefficients, no multiplications are actually required.

The third canonical form is the *parallel form*, shown in Fig. 3, which results from a partial faction expansion of

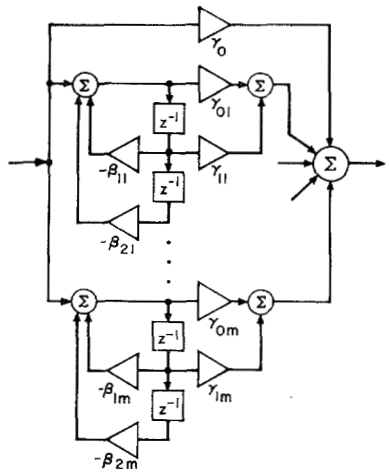
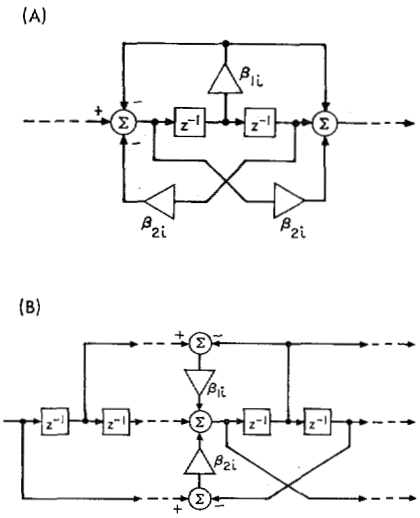


Fig. 3. The parallel form for a digital filter

Fig. 4. (A) Second-order section for digital all-pass filter in cascade form. (B) Alternate configuration for digital all-pass filter in cascade form.



(1) to produce

$$H(z) = \gamma_0 + \sum_{i=1}^{m} \frac{\gamma_{1i}z^{-1} + \gamma_{0i}}{\beta_{2i}z^{-2} + \beta_{1i}z^{-1} + 1}, \tag{3}$$

where $\gamma_0 = a_n/b_n$ and we have again chosen to use all second-order (denominator) factors. Note that all three canonical forms are entirely equivalent with regard to the amount of storage required (n unit delays) and the number of arithmetic operations required ($2n+1$ multiplications and $2n$ additions per sampling period). As previously noted, however, the cascade form requires significantly fewer multiplications for zeros on the unit circle and is thus especially appropriate for filters of the band-pass and the band-stop variety (including low-pass and high-pass filters).

Another interesting filter form may be derived for the special case of an all-pass filter (APF), i.e., a filter or "equalizer" with unity gain at all frequencies. The transfer function for a discrete APF has the general form [6]

$$H_A(z) = \frac{\sum_{i=0}^{n} b_{n-i}z^{-i}}{\sum_{i=0}^{n} b_i z^{-i}}. \tag{4}$$

Thus, with $a_i = b_{n-i}$ and $b_0 = 1$, the direct form can be used to implement (4). However, to reduce the accuracy requirements on the filter coefficients, a modified cascade form can be derived for the APF, corresponding to an $H_A(z)$ of the form

$$H_A(z) = \prod_{i=1}^{m} \frac{z^{-2} + \beta_{1i}z^{-1} + \beta_{2i}}{\beta_{2i}z^{-2} + \beta_{1i}z^{-1} + 1}. \tag{5}$$

Second-order sections for the cascade form of the APF are shown in Fig. 4. Fig. 4(A) is a straightforward modification of the standard cascade form in Fig. 2. Note that because the β_{1i} multiplier may be shared by both the feedforward and feedback paths, only three multiplications are required per second-order section rather than four. The number of multiplications may be further reduced by using the form of Fig. 4(B), which requires only two multiplications per second-order section. But now, two additional delays are required preceding the first second-order section to supply appropriately delayed inputs to the first section. Therefore, the cascade form of Fig. 4(B) requires a total of n multiplications and $n+2$ delays for an nth-order APF.

Serial Arithmetic

Using any of the canonical forms described in the preceding section, all of the coefficient multiplications and many of the additions during a given Nyquist interval may be performed simultaneously. Therefore, a high degree of parallel processing is possible in the implementation of a digital filter and this may be achieved by providing multiple adders and multipliers with appropriate interconnections. Economy is then realized by using serial arithmetic, and by sharing the adders and multipliers (using the multiplexed circuit configurations to be described) insofar as circuit speed will allow.

In addition to a significant simplification of the hardware, serial arithmetic provides for an increased modularity and flexibility in the digital circuit configurations. Also, the processing rate is limited only by the speed of the basic digital circuits and not by carry-propagation times in the adders and multipliers. Finally, with serial arithmetic, sample delays are realized simply as single-input single-output shift registers.

The two's-complement representation [7] of binary numbers is most appropriate for digital filter implementation using serial arithmetic because additions may proceed (starting with the least significant bits) with no advance knowledge of the signs or relative magnitudes of the numbers being added (and with no later corrections of the obtained sums as with one's-complement). We will assume a two's-complement representation of the form

$$\delta_0 \cdot \delta_1\delta_2 \cdots \delta_{N-1}, \tag{6}$$

which represents a number (δ) having a value of

$$\delta = -\delta_0 + \sum_{i=1}^{N-1} \delta_i 2^{-i}, \tag{7}$$

where each δ_i is either 0 or 1. Thus, the data is assumed to lie in the interval

$$-1 \leq \delta < 1, \tag{8}$$

with the sign of the number δ being given by the last bit (in time) δ_0.

An extremely useful property of two's-complement representation is that in the addition of more than two numbers, if the magnitude of the correct total sum is small enough to allow its representation by the N available bits, then the correct total sum will be obtained regardless of the order in which the numbers are added, even if an overflow occurs in some of the partial sums. This property is illustrated in Fig. 5, which depicts numbers in two's-complement representation as being arrayed in a circle, with positive full scale ($1-2^{-N+1}$) and negative full scale (-1) being adjacent. The addition of positive addends produces counterclockwise rotation about the circle, whereas negative addends produce clockwise rotation. Thus, if the correct total sum satisfies (8), no information is lost with positive or negative overflows and the correct total sum will be obtained.

This overflow property is important for digital filter implementation because the summation points in the filters often contain more than two inputs (see Fig. 3);

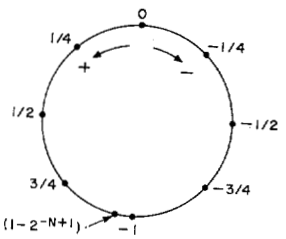
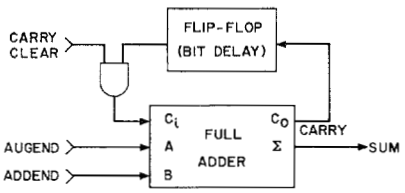


Fig. 5. Illustration of overflow property in two's-complement binary representation.

Fig. 6. Serial two's-complement adder.



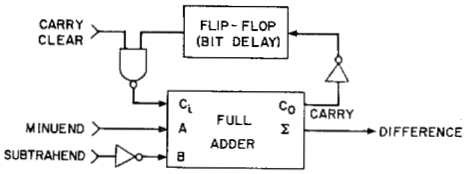
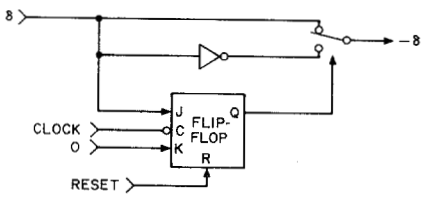


Fig. 7. Serial two's-complement subtractor.

Fig. 8. Serial two's complementer.



although it may be possible to argue that because of gain considerations the output of the summation point cannot overflow, there is no assurance that an overflow will not occur in the process of performing the summation. Note that this property also applies when one of the inputs to the summation has itself overflowed as a result of a multiplication by a coefficient of magnitude greater than one.

Arithmetic Unit

The three basic operations to be realized in the implementation of a digital filter are delay, addition (or subtraction), and multiplication. As previously mentioned, serial delays (z^{-1}) are realized simply as single-input single-output shift registers. Realizations for a serial adder (subtractor) and a serial multiplier are described in this section. The adders and multipliers, including their interconnections, will be said to comprise the arithmetic unit of the digital filter.

A serial adder for two's-complement addition is extremely simple to construct [7]. As shown in Fig. 6, it consists of a full binary adder with a single-bit delay (flip-flop) to transfer the carry output back to the carry input. A gate is also required in the carry feedback path to clear the carry to zero at the beginning of each sample. Accordingly, the carry-clear input is a timing signal, which is zero during the least significant bit of each sample and is one otherwise.

A serial two's-complement subtractor is implemented by first complementing (negating) the subtrahend input and then adding the complemented subtrahend to the minuend. To complement a number in two's-complement representation, each bit of the representation is inverted and a one is then added to the least significant bit of the inverted representation (i.e., 2^{-N+1} is added to the inverted number). The corresponding serial subtractor circuit is shown in Fig. 7. The subtrahend is inverted and a one is added to the least significant bit by clearing the initial carry bit to one, rather than to zero as in the adder. This is accomplished by means of two inverters in the carry feedback path, as shown.

A separate two's complementer (apart from a subtractor) may also be constructed; such circuits are required in the multiplier to be described. This operation is implemented with a simple sequential circuit which, for each sample, passes unchanged all initial (least significant) bits up to and including the first "1" and then inverts all succeeding bits. A corresponding circuit is depicted in Fig. 8.

A serial multiplier may be realized in a variety of configurations, but a special restriction imposed by this implementation approach makes one configuration most appropriate. This restriction is that no more than N bits per sample may be generated at any point in the digital network because successive samples are immediately adjacent in time and there are no "time slots" available for more than N bits per sample. Hence, the full $(N+K)$-bit product of the multiplication of an N-bit sample by a K-bit (fractional) coefficient may not be accumulated before rounding is performed. However, using the multiplication scheme described below, it is possible to obtain the same rounded N-bit product without ever generating more than N bits per sample. Rounding is usually preferable, rather than truncation, to limit the introduction of extraneous low-frequency components (dc drift) into the filter.

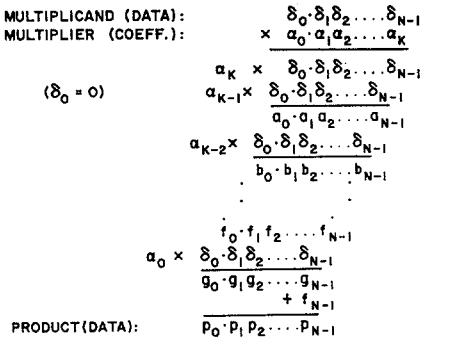


Fig. 9. Serial multiplication using no more than N bits per data word.

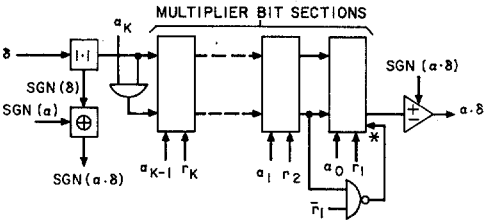
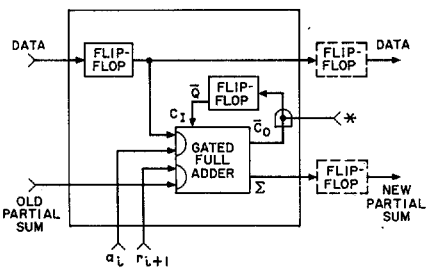


Fig. 10. Serial multiplier, showing modularity.

Fig. 11. A multiplier bit section.



The serial multiplication scheme is depicted in Fig. 9. To simplify the required hardware, both the multiplicand (data) and the multiplier (coefficient) are constrained to be positive, with appropriate sign changes being made before and after the multiplication. Thus $\delta_0 = 0$ in Fig. 9 and the sign of the multiplicand (δ) is stored separately as SGN δ. For convenience, the multiplier (α) will be assumed to lie in the interval.

$$-2 < \alpha < 2. \tag{9}$$

Although (9) is not necessarily applicable in the general case, it does hold for the denominator coefficients of the cascade and parallel forms, and usually for the numerator coefficients of the cascade form as well. The magnitude of the multiplier is thus represented in Fig. 9 as

$$\alpha_0 \cdot \alpha_1 \alpha_2 \cdots \alpha_K, \tag{10}$$

which represents a value of

$$|\alpha| = \sum_{i=0}^{K} \alpha_i 2^{-i}. \tag{11}$$

The restriction in (9) and the resulting representation in (10) and (11) are in no way essential to the serial multipliers to be described, but are meant only to be representative of the multiplication scheme. The sign of the multiplier is also stored separately as SGN α.

The multiplication scheme in Fig. 9 proceeds as follows. The multiplicand is successively shifted (delayed) and multiplied (gated) by the appropriate bit (α_i) of the multiplier. These delayed gated instances of the multiplicand are then added in the order indicated. After each addition (including the "addition" of $\alpha_K\delta$ to 0), the least significant bit of the resulting partial sum (i.e., $\delta_{N-1}, a_{N-1}, b_{N-1}, \cdots, f_{N-1}$) is truncated to prevent the succeeding partial sum from exceeding N bits in length. Note that these bits may be truncated because the full unrounded product would be

$$g_0 \cdot g_1 g_2 \cdots g_{N-1} f_{N-1} \cdots b_{N-1} a_{N-1} \delta_{N-1} \tag{12}$$

and to round (12) to N bits, only the value of the bit f_{N-1} is required. Thus, before truncating f_{N-1}, its value is stored elsewhere to be added in the final step to g, as shown in Fig. 9, to obtain the rounded product (p).

The serial multiplier corresponding to the scheme described above is shown in Fig. 10. The absolute value of each incoming datum (δ) is taken and its sign (SGN δ) is added modulo-2 to the coefficient sign (SGN α) to determine the product sign (SGN $\alpha \cdot \delta$). The (positive) multiplicand is then successively delayed and gated by the appropriate multiplier bits (α_i) and the partial sums are accumulated in the multiplier bit sections. A single multiplier bit section is shown in Fig. 11. The least significant bit of each partial sum is truncated (gated to zero) by the appropriate timing signal r_{i+1}. Rounding is accomplished by adding in the last truncated bit (f_{N-1}) via the * input to the last bit section. Finally, the sign of the product is inserted using a two's-complementer such as that in Fig. 8. At high data rates, it may be necessary to insert extra flip-flops between some or all of the multiplier bit sections, as shown in dotted lines in Fig. 11, to keep the propagation delay through the adder circuits from becoming excessive.

Several observations concerning the serial multiplier should be made at this point. First, there is a delay of K bits in going through the multiplier and this delay must be deducted from a delay (z^{-1}) that precedes the multiplier. (If the extra flip-flops in Fig. 11 are required, then the multiplier will yield a delay of up to $2K$ bits.) In addi-

tion, the absolute value operation at the first of the multiplier requires a delay of N bits (to determine the sign of each incoming datum) and this must be deducted from a preceding delay as well. Thus, to use this serial multiplier, the z^{-1} delays of the digital filter must be at least $N+K$ (or up to $N+2K$) bits in length. This in turn implies, as we shall see in the next section, that some form of multiplexing is required if the multipliers are to be implemented in this manner.

Another observation is that the adders in the multiplier bit sections do not require carry-clear inputs because only positive numbers are being added. However, output product overflows (in the sense of Fig. 5) are possible with coefficients (α) of magnitude greater than one. It may thus be necessary to restrict the amplitude of the data into certain multipliers to prevent output overflows; while in certain other multipliers, these overflows may be perfectly allowable as discussed in the preceding section. In general, however, the inputs to a summation must be scaled so that an overflow will not occur in the *final* output of the summation. Such overflows represent a severe nonlinearity in the system, and very undesirable effects can result in the output of the filter.

Multiplexing

Having realized the three basic digital filter components (delays, adders, and multipliers), the filter itself may be implemented by simply interconnecting these components in a configuration corresponding to one of the digital forms, canonical or otherwise, for the filter. However, if the input rate bit (sampling rate times bits per sample) is significantly below the capability of the digital circuits, the digital filter can be multiplexed to utilize the circuits more efficiently. The various multiplexing schemes are of two main types: 1) the multiplexed filter may operate upon a number of input signals simultaneously or 2) the multiplexed filter may effect a number of (different) filters for a single input signal. A combination of these two types is also possible.

To multiplex the filter to process M simultaneous inputs (type 1), the input samples from the M sources are interleaved sample by sample and fed (serially) into the filter. The bit rate in the filter is thus increased by a factor of M. The shift-register delays must also be increased by a factor of M to a length of MN bits. Otherwise, the filter is identical in its construction to the single-input case. In particular, the arithmetic unit containing the adders and multipliers is the same; it just operates M times faster. The output samples emerge in the same interleaved order as the input and are thus easily separated. Type-1 multiplexing is depicted in Fig. 12.

If the M channels in Fig. 12 are to be filtered differently or if type-2 multiplexing is also employed, the filter coefficients are stored in a separate read-only coefficient memory and are read-out as required by the multiplexed

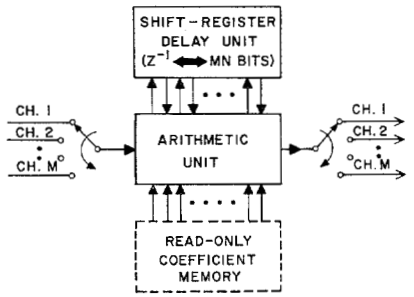


Fig. 12. Type-1 multiplexing for M input channels.

filter. A diode matrix provides a very fast and inexpensive form of read-only memory (ROM) for this purpose. If, however, all M channels are to be filtered identically and no type-2 multiplexing is employed, the coefficients may be wired into the multipliers and no ROM is then required. This is indicated by the dotted lines enclosing the ROM in Fig. 12. In this case, adders must be included only in those multiplier bit sections of the arithmetic unit for which the corresponding multiplier bits (α_i) equal one.

In many cases, a number of different, but similar, filters or subfilters are required for the same input signal. For example, all of the second-order subfilters comprising the cascade or parallel forms are similar in form, differing only in the values for the multiplying coefficients (see Figs. 2 and 3). Type-2 multiplexing refers to the implementation of these different (sub)filters with a signal multiplexed filter. An example of a multiplexed second-order filter is shown in Fig. 13. As with type-1 multiplexing, the combining of M separate filters into one multiplexed filter requires that the bit rate in the filter be increased by a factor of M and that the shift-register delays (z^{-1}) also be increased by a factor of M to MN bits in length. The coefficients are supplied from the read-only coefficient memory, which cycles through M values for each coefficient during every Nyquist interval. Data are routed in, around, and out of the filter by external routing switches, which are also controlled from the ROM.

As an example of type-2 multiplexing, consider the implementation of a 12th-order filter in cascade form, using the multiplexed second-order filter in Fig. 13. Here $M=6$, so the bit rate in the filter must be (at least) $6N$ bits per Nyquist interval. During the first N bits of each Nyquist interval, the input sample is introduced into and is processed by the arithmetic unit with the multiplying coefficients (α_1, α_2, β_1, β_2) of the first subfilter in the cascade form. The resulting output is delayed by N bits ($z^{-1/M}$) and fed back via the input routing switch to become the input to the filter during the second N-bit portion of the Nyquist interval. This feedback process is repeated four

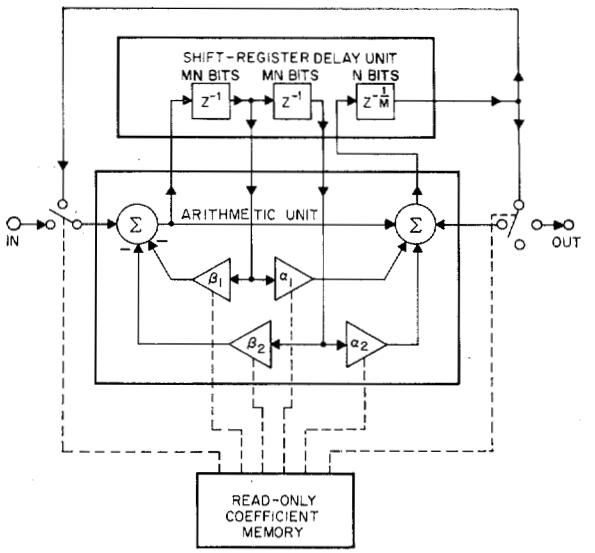


Fig. 13. General second-order filter for type-1 and type-2 multiplexing.

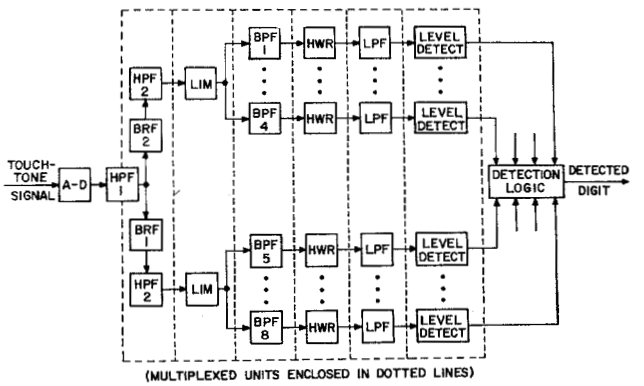


Fig. 14. Digital touch-tone receiver, showing multiplexed filters and nonlinear units.

more times, with the filter coefficients from the ROM being changed each time to correspond to the appropriate subfilter in the cascade form. The sixth (last) filter output during each Nyquist interval is the desired 12th-order filter output. The parallel form, or a combination of cascade and parallel filters, may be realized using the filter in Fig. 13 by simply changing the bits in the ROM which control the switching sequences of the input and output routing switches.

Sample System

As an example of this approach to the implementation of digital filters, we will take an experimental, all-digital touch-tone receiver (TTR) which has been designed and constructed at Bell Telephone Laboratories, Inc. The digital TTR is depicted in block-diagram form in Fig. 14 This is a straightforward digital version of the standard analog TTR described elsewhere [8]. Without going into the detailed operation of the system, we simply note that the combined high-pass filters (HPF's) are third order, the band-rejection filters (BRF's) are each sixth order, the bandpass filters (BPF's) are each second order, and the low-pass filters (LPF's) are each first order. The other signal-processing units required are the limiters (LIM's), half-wave rectifiers (HWR's), and level detectors. These nonlinear operations are, of course, easily implemented in digital form.

A multiplexing factor of $M=8$ is employed in the experimental TTR to combine all of the units enclosed in dotted lines into single multiplexed units. In particular, all of the HPF's and BRF's are multiplexed into one second-order filter (combined type-1 and type-2 multiplexing), the eight BPF's are multiplexed into another second-order filter (type-1 multiplexing with ROM coefficients), and the eight LPF's are multiplexed into one first-order section (type-1 multiplexing with wired-in coefficients). The nonlinear units are readily multiplexed as well and operate directly upon the interleaved output samples from the filters.

Some of the parameters of the experimental TTR design are as follows: the sampling rate is 10 K samples/second with an initial quantization (A/D conversion) of 7 bits/sample; the data word length (N) within the filter is 10 bits/sample; the filter coefficients have 6-bit fractional parts (K); and, as previously stated, the multiplexing factor (M) is eight. Thus, the bit rate within the filter (sampling rate $\times$ bits/sample $\times$ M) is 800 K bits/second. The number of bits required to represent the data and the coefficients of the TTR were determined through computer simulation of the system. The hardware required to implement this design consists primarily of about 40 serial adders and 400 bits of shift-register storage.

Analog-to-Digital Converter

In most applications of digital filters, the initial input signal is in analog form and must be converted to digital form for processing. It may or may not be necessary to reconvert the digital output signal to analog form, depending upon the application. Digital-to-analog (D/A) conversion is a relatively straightforward and inexpensive

process, but the initial analog-to-digital (A/D) conversion is often quite a different situation. For audio-frequency applications, however, a simple and very accurate A/D converter may be implemented using delta modulation (Δ-mod) as an intermediate encoding process. This A/D converter will now be described.

The A/D converter is depicted in Fig. 15, with a Δ-mod encoder being shown in Fig. 16. The Δ-mod encoder produces a series of bivalued pulses (0's and 1's) which, when integrated, constitute an approximation to the input analog signal. The number of 1's (or 0's) occurring during each (eventual) sampling interval is accumulated in the counter as a measure of the change in signal amplitude during that interval. At the end of the interval, this number is transferred to the storage register and the counter is then reset to its initial value to begin counting during the next interval. The appropriate initial value for the counter is minus one-half the number of Δ-mod pulses per sampling interval.

The number stored in the storage register for each sampling interval is the difference between the desired sample value and the preceding sample value. Thus, if these difference samples are accumulated in a simple first-order accumulator, as shown, the full digital sample values result. A small "leak" is introduced into the accumulator by making the feedback gain slightly less than one ($1-2^{-K}$) to keep the dc gain of the accumulator from being infinite. This prevents a small dc bias in the Δ-mod output from generating an unbounded accumulator output. The accumulator leak should be matched by a similar leak in the integrator of the Δ-mod encoder.

Since the accumulator is itself a first-order digital filter, it can be implemented and multiplexed using the same circuits and techniques as previously described. The multiplexing may be either with other A/D conversion channels or with other filters, or both. Note, however, that if the digital filter following the A/D converter has, or can have, a zero at $z=1$ (corresponding to zero of transmission at dc), this zero would cancel the pole supplied by the accumulator (with no leak). Therefore, in this case, the accumulator may be eliminated from the A/D converter (along with the zero at $z=1$ from the following filter), making the A/D conversion even simpler.

The accuracy of A/D conversion implemented in this manner is a function of the following factors: 1) the ratio of the Δ-mod rate to the sampling rate, 2) the sensitivity of the input comparison amplifier in the Δ-mod encoder, and 3) the match between the accumulator leak (if an accumulator is required) and the integrator leak in the Δ-mod encoder. There is also a maximum-slope limitation with delta modulation and an accompanying slope overload noise results if this slope limitation is exceeded [9]. Assuming that the error resulting from 2) and 3) and from slope overload is negligible, a useful rule of thumb for the A/D conversion accuracy is that the number of quantization levels effected equals approximately the ratio of the Δ-mod rate to the sampling rate. Thus, for example, to effect 10-bit A/D conversion using this scheme, the Δ-mod rate must be approximately 1000 times the sampling rate.

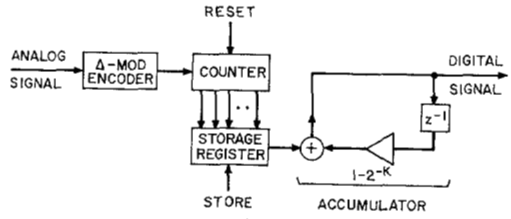
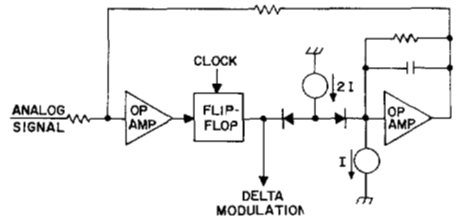


Fig. 15. Simple A/D converter using delta modulation.

Fig. 16. Delta-modulation encoder.

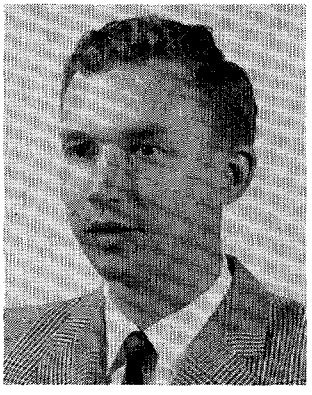


Conclusions

An approach to the implementation of digital filters has been described that employs a small set of relatively simple digital circuits in a highly regular and modular configuration, well suited to LSI construction. By using parallel processing and serial, two's-complement arithmetic, the required arithmetic circuits (adders and multipliers) are greatly simplified, and the processing rate is limited only by the speed of the basic digital circuits and not by carry-propagation times. The resulting filters are readily multiplexed to process multiple data inputs or to effect multiple, but different, filters (or both) using the same arithmetic circuits, thus providing for efficient hardware utilization. A multiplexing factor of 100 or so is possible in audio-frequency applications, using presently available digital logic in the medium-speed range. The filters are also easily modified to realize a wide range of filter forms, transfer functions, multiplexing schemes, and round-off noise levels (i.e., data accuracies) by changing only the contents of the read-only coefficient memory and/or the timing signals and the length of the shift-register delays. For audio-frequency applications, a simple A/D converter may be implemented using delta modulation as an intermediate encoding process.

REFERENCES

[1] J. F. Kaiser, "Digital filters," in *System Analysis by Digital Computer*, J. F. Kaiser and F. F. Kuo, Eds. New York: Wiley, 1966, pp. 218–85.
[2] C. M. Rader and B. Gold, "Digital filter design techniques in the frequency domain," *Proc. IEEE*, vol. 55, pp. 149–171, February 1967.
[3] R. M. Golden, "Digital filter synthesis by sampled-data transformation," this issue, pp. 321–329.
[4] B. Gold and C. M. Rader, *Digital Processing of Signals*. New York: McGraw-Hill, 1969.
[5] J. F. Kaiser, "Some practical considerations in the realization of linear digital filters," *1965 Proc. 3rd Allerton Conf. on Circuit and System Theory*, pp. 621–633.
[6] R. B. Blackman, unpublished memorandum.
[7] Y. Chu, *Digital Computer Design Fundamentals*. New York: McGraw-Hill, 1962.
[8] R. N. Battista, C. G. Morrison, and D. H. Nash: "Signaling system and receiver for touch-tone calling," *IEEE Trans. Communications and Electronics*, vol. 82, pp. 9–17, March 1963.
[9] E. N. Protonotarios, "Slope overload noise in differential pulse code modulation systems," *Bell Sys. Tech. J.*, vol. 46, pp. 2119–2161, November 1967.



Leland B. Jackson (M'65) was born in Atlanta, Ga., on July 23, 1940. He received the B.S. and M.S. degrees in electrical engineering in 1963 from the Massachusetts Institute of Technology, Cambridge. He is presently working towards the Sc.D. degree in electrical engineering at the Stevens Institute of Technology, Hoboken, N. J.

In 1961 and 1962 he was associated with Bell Telephone Laboratories, Inc., under the M.I.T. cooperative program in electrical engineering. From 1964 to 1966, he was employed by Sylvania Electronic Systems, Inc., Mountain View, Calif., where he studied and designed signal processing techniques for ionospheric research. In 1966 he joined Bell Telephone Laboratories, Inc., Murray Hill, N. J., where he has been primarily concerned with the analysis and design of digital filters and related systems.

Mr. Jackson is a member of Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.

James F. Kaiser (S'50–A'52), for a photograph and biography, please see page 183 of the June, 1968, issue of this TRANSACTIONS.

Henry S. McDonald (A'52–M'57) received the B.S. degree in electrical engineering in 1950 from the Catholic University of America, Washington, D.C., and the M.S. and Doctor of Engineering degrees in 1953 and 1955 from The Johns Hopkins University, Baltimore, Md.

He served as an Instructor in Electrical Engineering and as a Research Staff Assistant while pursuing his academic work. Since 1955 he has been with the Research Division of Bell Telephone Laboratories, Inc., Murray Hill, N. J. His interests and work have ranged over speech processing, vocoders, television bandwidth reduction, digital signal processing, computer input–output devices, real-time and graphical aspects of computing, computer design, and switching system design. He is currently Assistant Director of the Communication Principles Research Laboratory at Murray Hill.