Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications

ERNEST L. HALL, MEMBER, IEEE, DAVID D. LYNCH, MEMBER, IEEE, AND SAMUEL J. DWYER, III, MEMBER, IEEE

Abstract—**An approximate method for rapid multiplication or division with relatively simple digital circuitry is described. The algorithm consists of computing approximate binary logarithms, adding or subtracting the logarithms, and computing the approximate antilogarithm of the resultant. Using a criteria of minimum mean square error, coefficients for the approximations are developed. An error analysis is given for three cases in which the algorithm is useful. Finally, applications to digital filtering computations are considered which illustrate that log-antilog multiplication is not simpler than an array multiplier for computing single products, but is useful for parallel digital filter banks and multiplicative digital filters.**

Index Terms—**Antilogarithm converter, binary-to-binary logarithm converter, computer multiplication, digital filter realization.**

INTRODUCTION

AN ALGORITHM for computer multiplication by binary logarithms was described by Mitchell [1] and expanded by Combet, Van Zonneveld, and Verbeek [2]. This method has only modest accuracy and limited application for general purpose computation. However, there is a class of digital filtering problems in which the speed, nature of the signals, and component count, offset accuracy considerations. Real-time digital filtering of radar video for moving target detection, synthetic aperture processing, and pulse compression are in this class. Because of the statistical nature of the sampled signals, the large amount of signal integration required, and the characterization of detection performance on a probabilistic basis, the accuracy of a single computation has less importance than the mean and variance of the operation on the signal ensemble. Radar video is characterized by broad bandwidth and corresponding high data flow rates which make real time multiplication with readily available logic very difficult. Furthermore, multiple filters are usually required because the noise is colored and the filter bandpass is but a small fraction of the actual signal bandwidth.

In Section I the binary-to-binary logarithm conversion is reviewed. In Section II the antilogarithm conversion is developed. In Section III a detailed error analysis is given. Finally, Section IV contains examples of the use of the log-antilog technique for digital filter applications.

Manuscript received May 1, 1968; revised July 14, 1969.

E. L. Hall is with the Department of Electrical Engineering, University of Missouri, Columbia, and Emerson Electric Company, St. Louis, Mo.

D. D. Lynch is with the Emerson Electric Company, Electronics and Space Division, St. Louis, Mo.

S. J. Dwyer, III, is with the Department of Electrical Engineering, University of Missouri, Columbia, Mo.

I. BINARY-TO-BINARY LOGARITHM CONVERSION

A simple method for the computation of the base two logarithm of a binary number was developed by Mitchell [1]. The method consists of encoding the binary number into a form from which the characteristic is easily determined and the mantissa is easily approximated.

Let N be a nonzero finite length binary number and let m and j represent the binary power of the most and least significant bits of N, respectively. The case of $N=0$ is easily handled separately. N may be written as

$$N = Z_m Z_{m-1} \cdots Z_{j+1} Z_j$$

with

$$Z_m = 0, 1; \quad m, j = 0, \pm 1, \cdots; \quad m \geq j.$$

Clearly,

$$2^{m+1} > N \geq 2^j.$$

N is also given by

$$N = \sum_{i=j}^{m} Z_i 2^i.$$

Now let Z_k be the most significant nonzero bit of N, $m \geq k \geq j$. Then,

$$N = 2^k + \sum_{i=j}^{k-1} Z_i 2^i.$$

Factoring out 2^k results in

$$N = 2^k \left\{ 1 + \sum_{i=j}^{k-1} Z_i 2^{i-k} \right\} = 2^k (1 + x)$$

where

$$x = \sum_{i=j}^{k-1} Z_i 2^{i-k} \quad \text{and} \quad 1 > x \geq 0 \text{ since } k \geq j.$$

Thus, N has been encoded into the form $N=2^k(1+x)$. The base two logarithm of N is

$$\log_2 N = k + \log_2 (1 + x).$$

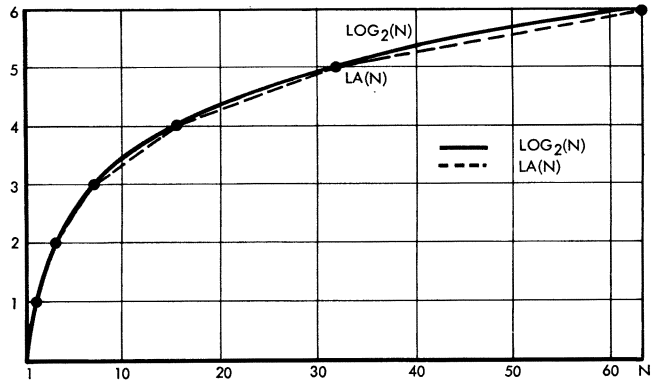

Fig. 1. Piecewise linear approximation to binary logarithm.

Since $1>x\geq 0$, the logarithm characteristic is k and the mantissa is only a function of x.

A linear approximation of $\log_2 N$ is of the form

$$LA(N) = k + ax + b.$$

The geometrical interpretation of this approximation is shown in Fig. 1 and consists of a piecewise linear approximation between the points where $\log_2 N$ attains integral values.

The linear coefficients a and b may be selected to maximize some return function. If simplicity is the desired return function, then $a=1$ and $b=0$ are the best coefficients. As shown by Mitchell, the maximum error

$$E = \log_2 N - LA(N)$$

with $a=1$, $b=0$, is 0.086.

If an easily computed set of coefficients is desired, then one may use the linear terms of a Taylor series expansion of $\log_2 (1+x)$ about the point $x=x_0$, $1>x_0\geq 0$ to obtain

$$a = \log_2 \epsilon/(1 + x_0)$$
$$b = \log_2 (1 + x_0) - x_0 \log_2 \epsilon/(1 + x_0).$$

The error in this approximation is

$$E = - x_0 \log_2 \epsilon - (1 + x_0) \log_2 (1 + x_0).$$

Combet *et al.* [2] described another method for selecting the coefficients. This method consists of partitioning the range of x into four parts and again making a piecewise linear approximation. The linear equations given in [2] were reportedly found by trial and error using a criteria of minimum error, and constraining the coefficients to be easily implemented with binary circuitry. That is, the coefficients were chosen to be fractions with integer numerators and power of two denominators. With a four subinterval partition, the single division error was reduced by a factor of six.

The authors propose that for many applications, including digital filtering, mean-square error is a desirable error criteria, although maximum error and easy implementation must also be considered. The coefficients for a linear least squares fit to $\log_2 (1+x)$ over, $1>x_2\geq x \geq x_1\geq 0$, will now be developed.

The mean-squared error is defined as

$$\overline{E}^2 = \frac{1}{x_2 - x_1}\int_{x_1}^{x_2} \{\log_2 (1 + x) - (ax + b)\}^2 dx.$$

To minimize $\overline{E}^2$ with respect to a and b it is necessary that

$$\frac{\partial \overline{E}^2}{\partial a} = 0 = \int_{x_1}^{x_2} -2x\{\log_2 (1 + x) - (ax + b)\}dx$$
$$\frac{\partial \overline{E}^2}{\partial b} = 0 = \int_{x_1}^{x_2} -2\{\log_2 (1 + x) - (ax + b)\}dx.$$

Let

$$I_1 = \int_{x_1}^{x_2} \log_2 (1 + x)dx$$
$$= \log_2 \epsilon\{ Y \log_\epsilon Y - Y\}\Big|_{1+x_1}^{1+x_2}$$
$$I_2 = \int_{x_1}^{x_2} x \log_2 (1 + x)dx$$
$$= \log_2 \epsilon\left\{\frac{Y^2}{2}\log_\epsilon Y - Y^2/4\right\}\Big|_{1+x_1}^{1+x_2} - I_1,$$

then

$$a = \{I_2 - (x_2 + x_1)I_1/2\} \Big/ \left\{\frac{x_2^3 - x_1^3}{3} - \frac{(x_2^2 - x_1^2)(x_2 + x_1)}{4}\right\}$$
$$b = I_1/(x_2 - x_1) - a(x_1 + x_2)/2.$$

Thus, for any partition of the interval [0, 1], the best linear mean-square coefficients can be determined and $\overline{E}^2$ evaluated. Also, the maximum error for any subinterval is easily determined. The coefficients, the mean-square error, and the maximum absolute error are given in Table I for 1, 2, 4, and 8 equispaced subinterval partitions.

For a particular realization, it is convenient to work with the linear equations in the form:

$$x + cx + d \quad \text{if } a \geq 1$$
$$x + c\bar{x} + d \quad \text{if } a < 1$$

TABLE I

MEAN-SQUARE ERROR AND COEFFICIENTS FOR LOGARITHM APPROXIMATION

Number of Subintervals	Subinterval	a	b	$\bar{E}^2$	$\lvert E_{\max}\rvert$
1	1	0.984255	0.065176	0.641074E-3	0.065176
2	1	1.163555	0.021303	0.192903E-4	0.021303
	2	0.827788	0.181567	0.581653E-5	0.010518
4	1	1.285610	0.006243	0.278225E-6	0.006243
	2	1.050957	0.063330	0.387113E-6	0.004141
	3	0.888761	0.143537	0.642476E-6	0.002186
	4	0.770244	0.231857	0.289856E-7	0.002186
8	1	1.359165	0.001681	0.173267E-7	0.001681
	2	1.215426	0.019368	0.550871E-7	0.001371
	3	1.099427	0.048200	0.814192E-7	0.001129
	4	1.003868	0.083914	0.297152E-7	0.000933
	5	0.923414	0.124049	0.130118E-6	0.000794
	6	0.854749	0.166916	0.128720E-6	0.000695
	7	0.806959	0.202033	0.186847E-6	0.001232
	8	0.734065	0.265769	0.150003E-6	0.001186

TABLE II

LOGARITHM EQUATIONS

Range	Mantissa
$0 \leq x < 1/4$	$x^* = x + 37x/128 + 1/128$
$1/4 \leq x < 1/2$	$x^* = x + 3x/64 + 1/16$
$1/2 \leq x < 3/4$	$x^* = x + 7\bar{x}/64 + 1/32$
$3/4 \leq x < 1$	$x^* = x + 29\bar{x}/128$

where

$$\bar{x} = 1 - x.$$

Also, for simple binary circuitry, it is necessary to obtain c and d as sums of binary fractions. In general the number of bits necessary for an input N as previously defined is $m-j-1$. However, for a given application one may be able to use a smaller number of bits with only a slightly larger maximum or mean-square error.

The linear logarithm equations for a four subdivision realization are given in Table II. The coefficients are the minimum mean-square coefficients quantitized to seven bits. The maximum error, which was computed at the critical values and extrema, ranges over $-0.00782 < E_{\max} < 0.00994$. The realized maximum mean-squared error is $\bar{E}^2_{\max} = 3.33 \times 10^{-6}$.

II. ANTILOGARITHM CONVERSION

The following method of computing the antilogarithm is used by Mitchell [1].

Let

$$M = LA(N) = k + x$$
$$0 \leq x < 1, \quad k \text{ an integer.}$$

Then

$$2^M = 2^k \cdot 2^x.$$

Since k is an integer, multiplication by 2^k is simply a shift operation. An approximation to 2^M is given by $EA(M) = 2^k(1+x)$. This approximation is shown in Fig. 2 and consists of a piecewise linear approximation to 2^M between the points where M takes on integral values.

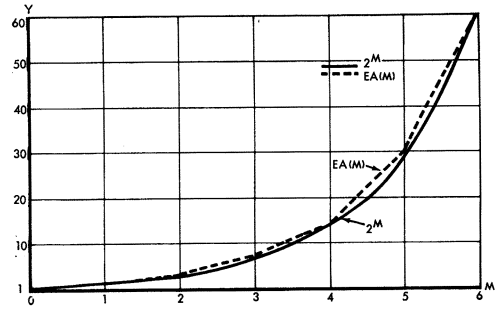

Fig. 2. Piecewise linear approximation to binary exponent.

The authors propose an improved approximation to the antilogarithm conversion which allows the error to be reduced to any desired level at the cost of increased complexity. The method is similar to the logarithm conversion and consists of partitioning the interval, $0 \leq x < 1$, into subintervals and making linear approximations over these subintervals. Mean-square error is again used as the error criteria. Maximum error is also computed.

The linear least squares fit to 2^x over the interval, $0 \leq x_1 \leq x \leq x_2 < 1$, will now be developed. The linear mean-squared error is defined as

$$\bar{E}^2 = \frac{1}{x_2 - x_1} \int_{x_1}^{x_2} \{2^x - (ax + b)\}^2 dx.$$

To minimize $\bar{E}^2$ with respect to a and b it is necessary that

$$\frac{\partial \bar{E}^2}{\partial a} = 0 = \int_{x_1}^{x_2} -2x\{2^x - (ax + b)\}\, dx$$

and

$$\frac{\partial \bar{E}^2}{\partial b} = 0 = \int_{x_1}^{x_2} -2\{2^x - (ax + b)\}\, dx.$$

TABLE III
MEAN-SQUARE ERROR AND COEFFICIENTS FOR ANTILOGARITHM APPROXIMATION

Number of Subintervals	Subintervals	a	b	E^{-2}	$\lvert E_{\max} \rvert$
1	1	0.992089	0.946650	0.656127E-3	0.061261
2	1	0.826852	0.988453	0.813603E-5	0.012333
	2	1.169200	0.813322	0.143051E-4	0.017487
4	1	0.756609	0.997295	0.894069E-7	0.002759
	2	0.900147	0.960905	0.283122E-6	0.003275
	3	1.068757	0.876172	0.23818E-6	0.004054
	4	1.273014	0.722413	0.730156E-6	0.004519
8	1	0.726814	0.999170	0.223517E-7	0.000839
	2	0.787594	0.991472	0.223517E-7	0.000846
	3	0.859510	0.973650	0.372529E-7	0.000872
	4	0.945204	0.941145	0.149011E-7	0.001242
	5	1.021377	0.902764	0.134110E-6	0.001095
	6	1.110733	0.847371	0.208616E-6	0.001372
	7	1.221244	0.764536	0.819563E-7	0.001323
	8	1.331134	0.667864	0.163912E-6	0.001400

TABLE IV
ANTILOGARITHM EQUATIONS

$0 \leq x < 1/4$	$Y = x + 1/4\bar{x} + 3/4$
$1/4 \leq x < 1/2$	$Y = x + 13/128\bar{x} + 55/64$
$1/2 \leq x < 3/4$	$Y = x + 9/128x + 7/8$
$3/4 \leq x < 1$	$Y = x + 35/128x + 23/32$

Let

$$I_1 = \int_{x_1}^{x_2} 2^x dx = \log_2 \epsilon \{2^{x_2} - 2^{x_1}\}$$

$$I_2 = \int_{x_1}^{x_2} x 2^x dx = 2^x \left[\frac{x \log_\epsilon 2 - 1}{(\log_\epsilon 2)^2}\right]\Bigg|_{x_1}^{x_2}.$$

Then

$$a = \left\{I_2 - \frac{(x_2 + x_1)}{2} I_1\right\} \Bigg/ \left\{\frac{(x_2^3 - x_1^3)}{3} - \frac{(x_2^2 - x_1^2)(x_1 + x_2)}{4}\right\}$$

$$b = \left\{I_1 - a\frac{(x_2^2 - x_1^2)}{2}\right\} / (x_2 - x_1).$$

Thus, for any partition of $[0, 1]$, the linear mean-square error coefficients a and b may be determined and $\overline{E}^2$ evaluated. The absolute maximum error may also be determined over each subinterval. These values are given in Table III for 1, 2, 4, and 8 subinterval partitions.

The linear antilogarithm equations for a four subdivision realization are given in Table IV. The coefficients are the mean-square coefficients quantized to seven bits. The maximum error ranges over

$$-0.00327 < E_{\max} < 0.00796;$$

the realized maximum mean-squared error is $\overline{E}^2_{\max} = 1.475 \times 10^{-6}$.

III. Error Analysis

In this section, an error analysis is given for several cases in which a log-antilog conversion would be desirable. The first case considered will be the product of two variables. Next, a special case of a product of a constant and a variable is considered. This special case arises in many digital filter applications. Finally, the error for a quotient of two variables is considered.

This analysis is mainly concerned with errors due to the approximation. Quantization error, truncation error, and coefficient error have been dealt with in other papers and would depend on the actual hardware used to implement the conversions. For one of the authors' applications [3] an exact digital simulation was made to study these effects.

A. Product Error

Suppose that the log-antilog conversion was used to approximate the product of two numbers. How much error would be incurred? Let M_1 and M_2 represent two binary numbers which are encoded into the form:

$$M_1 = 2^{k_1}(1 + x_1) \quad \text{where } 0 \leq x_1 < 1, \ k_1 \text{ an integer}$$

$$M_2 = 2^{k_2}(1 + x_2) \quad \text{where } 0 \leq x_2 < 1, \ k_2 \text{ an integer.}$$

The approximate binary logarithms, denoted by LA, are given by

$$LA(M_1) = k_1 + x_1 + y_1$$

$$LA(M_2) = k_2 + x_2 + y_2$$

where y_1 and y_2 are the linear correction terms, i.e.,

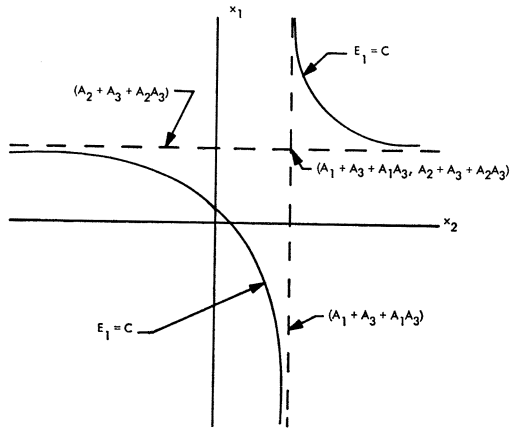

Fig. 3. Curve of constant error $E_1 = C$.

$$y_1 = a_1x_1 + b_1$$
$$y_2 = a_2x_2 + b_2.$$

The values of a_i and b_i, $i=1, 2$, are constants over a certain interval of x_i and are given in Table II. No carry can occur in the sum x_i+y_i with the coefficients given in Table II so that

$$0 \leq x_1 + y_1 < 1$$
$$0 \leq x_2 + y_2 < 1.$$

Adding the approximate logarithms of M_1 and M_2 gives the approximate logarithm LP of the product $P = M_1M_2$. Thus,

$$LP = k_1 + k_2 + x_1 + y_1 + x_2 + y_2.$$

Since a carry from the mantissa to the characteristic can occur, there are two cases to consider.

For Case 1, no carry: $x_1+y_1+x_2+y_2<1$.

For Case 2, carry: $x_1+y_1+x_2+y_2\geq 1$.

Case 1: The approximate binary exponent $EA(P)$ is given by

$$EA(P) = 2^{k_1+k_2}(1 + x_1 + x_2 + y_1 + y_2 + z_{12})$$

where z_{12} is the linear correction term, i.e.,

$$z_{12} = a_3(x_1 + x_2 + y_1 + y_2) + b_3.$$

Case 2:

$$EA(P) = 2^{k_1+k_2+1}(x_1 + x_2 + y_1 + y_2 + z_{12})$$

where z_{12} is the linear correction term, i.e.,

$$z_{12} = a_3(x_1 + y_1 + x_2 + y_2 - 1) + b_3.$$

The coefficients a_3 and b_3 are constants over certain regions and are given in Table IV. The error in using the approximate product is given by

$$E = P - EA(P) = M_1M_2 - EA(P).$$

Case 1:

$$E = 2^{k_1+k_2}\{(1 + x_1)(1 + x_2) - (1 + x_1 + y_1 + x_2 + y_2 + z_{12})\}$$
$$E = 2^{k_1+k_2+1}\{x_1 \cdot x_2 - (y_1 + y_2 + z_{12})\}.$$

Let the normalized error be defined as

$$E_1 = E/2^{k_1+k_2}.$$

Case 2:

$$E = 2^{k_1+k_2+1}\{(1 + x_2)(1 + x_1) - 2(x_1 + x_2 + y_1 + y_2 + z_{12})\}$$
$$E = 2^{k_1+k_2}\{(1 - x_1)(1 - x_2)/2 - (y_1 + y_2 + z_{12})\}.$$

For this case, let the normalized error be defined as

$$E_2 = E/2^{(k_1+k_2+1)}$$

A direct attempt at finding the critical points of E_1 and E_2 would involve setting the partial derivatives equal to zero. For Case 1,

$$\frac{\partial E_1}{\partial x_1} = x_2 - a_1 - a_3 - a_1a_3$$
$$\frac{\partial E_1}{\partial x_2} = x_1 - a_2 - a_3 - a_2a_3$$

which would indicate that the critical point is

$$(x_1, x_2) = (a_2 + a_3 + a_2a_3, a_1 + a_3 + a_1a_3).$$

However, this point never falls in the interval of interest for the given coefficients. This fact is clearly indicated by the E_1=constant curves shown in Fig. 3. A similar result holds for E_2. If

$$E_1 = C = x_1x_2 - (a_1 + a_3 + a_1a_3)x_1 - (a_2 + a_3 + a_2a_3)x_2 - (b_1 + b_2)(1 + a_3) - b_3,$$

then

$$x_1 = \frac{(a_2 + a_3 + a_2a_3)\left\{x_2 + \dfrac{C + (b_1 + b_2)(1 + a_3) + b_3}{a_2 + a_3 + a_2a_3}\right\}}{x_2 - (a_1 + a_3 + a_1a_3)}.$$

A graph of this equation is shown in Fig. 3.

In fact, E is a hyperbolic paraboloid and is a monotonic function over the regions of interest, and therefore, it attains its maximum and minimum values at the boundary of the region.

The maximum absolute values of the normalized error E over the 16 regions of x_1 and x_2 are given in Table V. The maximum error is 0.01907 and occurs at $x_1 = x_2 = \frac{1}{4}$. The values in Table V were arrived at by computing E at 4096 equispaced points in the (x_1, x_2) plane. The maximum product error is only $\frac{1}{4}$ as large as the product error computed by Mitchell [1] for single interval approximations to the logarithm and exponential function.

B. Product Error—Special Case

The special case of the product of a variable and a constant will now be considered. This case arises in all linear constant coefficient digital filter applications. It is assumed that the logarithm of the constant is exact. As one would expect, the resulting product error is smaller.

Let N and C represent two binary numbers encoded into the form

$$N = 2^{k_1}(1 + x_1)$$
$$C = 2^{k_2}(1 + x_2).$$

The approximate logarithm of N is given by

$$LA(N) = k_1 + x_1 + y_1 \quad \text{where } y_1 = a_1x_1 + b_2.$$

The binary logarithm of C is given by

$$\log_2 (C) = k_2 + \log_2 (1 + x_2).$$

The approximate logarithm of the product $P = CN$ is given by

$$LP = k_1 + k_2 + x_1 + y_1 + \log_2 (1 + x_2).$$

Two cases must again be considered.

For Case 1, no carry: $x_1 + y_1 + \log_2 (1 + x_2) < 1$.

For Case 2, carry: $x_1 + y_1 + \log_2 (1 + x_2) \geq 1$.

The approximate binary exponent of P is given by:

Case 1:

$$EA(P) = 2^{k_1+k_2}\{1 + x_1 + y_1 + \log_2 (1 + x_2) + z_{12}\}$$

where

$$z_{12} = a_3\{x_1 + y_1 + \log_2 (1 + x_2)\} + b_3.$$

Case 2:

$$EA(P) = 2^{k_1+k_2}\{x_1 + y_1 + \log_2 (1 + x_2) + z_{12}\}$$

where

$$z_{12} = a_3\{x_1 + y_1 + \log_2 (1 + x_2) - 1\} + b_3.$$

The product error is defined as

$$E = P - EA(P).$$

Case 1:

$$E = 2^{k_1+k_2}\{(1 + x_1)(1 + x_2) - (1 + x_1 + y_1 + \log_2 (1 + x_2) + z_{12})\}.$$

Let the normalized error be defined as

$$E = E/(2^{k_1+k_2}).$$

Case 2:

$$E = 2^{k_1+k_2+1}\{(1 + x_1)(1 + x_2)/2 - (x_1 + y_1 + \log_2 (1 + x_2) + z_{12})\}.$$

Let the normalized error be defined by

$$E_2 = E/[2^{k_1+k_2+1}].$$

The normalized error is again a monotonic function over certain regions and attains its maximum and minimum values at the boundaries of these regions. The maximum absolute values of the error are shown in Table VI. The largest error is 0.01321 which occurs at $x_1 = 25/64$, $x_2 = 14/32$ and is substantially smaller than the product error for the general case. Although this point (x_1, x_2) is not on the boundary of one of the 16 main regions, it is on a product boundary since a_3 changes inside the region.

C. Quotient Error

The error incurred in a division operation will now be considered. Let the dividend D_1 and the divisor D_2 be binary numbers encoded into the form

$$D_1 = 2^{k_1}(1 + x_1), \qquad 0 \leq x_1 < 1$$
$$D_2 = 2^{k_1}(1 + x_2), \qquad 0 \leq x_2 < 1.$$

The approximate logarithms of D_1 and D_2 are given by

$$LA(D_1) = k_1 + x_1 + y_1, \qquad y_1 = a_1x_1 + b_1$$
$$LA(D_2) = k_2 + x_2 + y_2, \qquad y_2 = a_2x_2 + b_2.$$

Subtracting these logarithms gives the approximate logarithm LQ of the quotient $Q = D_1/D_2$:

$$LQ = k_1 - k_2 + x_1 - x_2 + y_1 - y_2.$$

TABLE V
MODULUS VALUES OF PRODUCT ERROR

X_2 \ X_1	$0 \le X_1 < \frac{1}{4}$	$\frac{1}{4} \le X_1 < \frac{1}{2}$	$\frac{1}{2} \le X_1 < \frac{3}{4}$	$\frac{3}{4} \le X_1 < 1$
$0 \le X_2 < \frac{1}{4}$	0.01907	0.01280	0.01059	0.01270
$\frac{1}{4} \le X_2 < \frac{1}{2}$	0.01280	0.01758	0.01163	0.00838
$\frac{1}{2} \le X_2 < \frac{3}{4}$	0.01059	0.01163	0.00773	0.00994
$\frac{3}{4} \le X_2 < 1$	0.01277	0.00838	0.00994	0.01531

TABLE VI
MODULUS VALUES OF PRODUCT ERROR SPECIAL CASE

X_2 \ X_1	$0 \le X_1 < \frac{1}{4}$	$\frac{1}{4} \le X_1 < \frac{1}{2}$	$\frac{1}{2} \le X_1 < \frac{3}{4}$	$\frac{3}{4} \le X_1 < 1$
$0 \le X_2 < \frac{1}{4}$	0.01046	0.01035	0.01010	0.00932
$\frac{1}{4} \le X_2 < \frac{1}{2}$	0.01229	0.01321	0.01150	0.00728
$\frac{1}{2} \le X_2 < \frac{3}{4}$	0.01168	0.01103	0.00677	0.00994
$\frac{3}{4} \le X_2 < 1$	0.01276	0.00742	0.01000	0.01100

TABLE VII
MODULUS VALUES OF QUOTIENT ERROR

X_2 \ X_1	$0 \le X_1 < \frac{1}{4}$	$\frac{1}{4} \le X_1 < \frac{1}{2}$	$\frac{1}{2} \le X_1 < \frac{3}{4}$	$\frac{3}{4} \le X_1 < 1$
$0 \le X_2 < \frac{1}{4}$	0.00985	0.00741	0.00692	0.00590
$\frac{1}{4} \le X_2 < \frac{1}{2}$	0.01500	0.00724	0.00490	0.00364
$\frac{1}{2} \le X_2 < \frac{3}{4}$	0.01779	0.00724	0.00657	0.00497
$\frac{3}{4} \le X_2 < 1$	0.01776	0.00779	0.00612	0.00594

Again, two cases must be considered depending on the occurrence of a borrow from the characteristic to the mantissa.

Case 1, no borrow: $x_1+y_1 \ge x_2+y_2$.

Case 2, borrow occurs: $x_1+y_1 < x_2+y_2$.

Case 1:

$$EA(Q) = 2^{k_1-k_2}\{1 + x_1 + y_1 - x_2 - y_2 + z_{12}\}$$

where

$$z_{12} = a_3\{x_1 + y_1 - x_2 - y_2\} + b_3.$$

Case 2:

$$EA(Q) = 2^{k_1-k_2-1}\{2 + x_1 + y_1 - x_2 - y_2 + z_{12}\}$$

where

$$z_{12} = a_3\{1 + x_1 + y_1 - x_2 - y_2\} + b_3.$$

The actual quotient is given by

$$Q = D_1/D_2 = 2^{k_1-k_2}\left\{\frac{(1 + x_1)}{(1 + x_2)}\right\}.$$

The error is defined as

$$E = Q - EA(Q)$$

which for the two cases is the following.

Case 1:

$$E = 2^{k_1-k_2}\left\{\frac{(1 + x_1)}{(1 + x_2)} - (1 + x_1 + y_1 - x_2 - y_2 + z_{12})\right\}.$$

Let the normalized error be defined by

$$E_1 = E/2^{k_1-k_2}.$$

Case 2:

$$E = 2^{k_1-k_2}\left\{\frac{(1 + x_1)}{(1 + x_2)} - 1/2(1 + x_1 + y_1 - x_2 - y_2 + z_{12})\right\}.$$

Again let the normalized error be defined by

$$E_2 = E/(2^{k_1-k_2}).$$

The maximum value of E for the 16 regions of the x_1, x_2 plane is listed in Table VII. This error is five times smaller than the quotient error computed by Mitchell.

IV. APPLICATIONS TO DIGITAL FILTERING

To illustrate the applications in which the log-antilog conversion would be advantageous, three examples are given. The first example shows that for a single multiplication a cobweb array is simpler. The second example illustrates how the log-antilog conversion can be used advantageously for a parallel filter bank. The last example of a multiplicative filter illustrates a situation in which a log-antilog conversion is necessary.

Example 1—Nonrecursive Digital Filter: The difference equation of a nonrecursive digital filter [4] may be written as

$$Y_n = \sum_{i=0}^{N-1} a_i x_{n-i}.$$

Using the log-antilog conversion the computation may be performed by

$$Y_n = \sum_{i=0}^{N-1} \exp\{\log|a_i| + \log|x_{n-i}|\}.$$

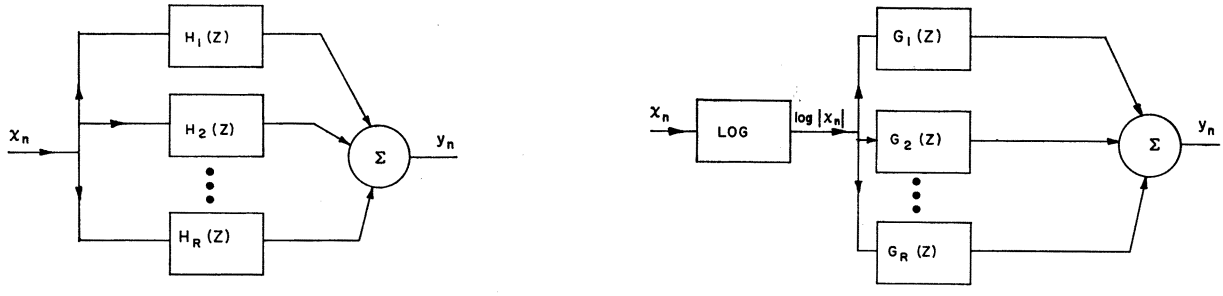

Fig. 4. Parallel filter bank.

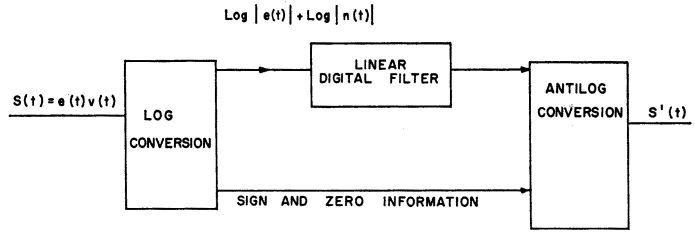

Fig. 5. Multiplicative digital filter.

The special cases of $a_i \leq 0$ or $x_{n-i} \leq 0$ are easily handled by either complementing or clearing the antilogarithm result. Also, the computation of $\log |a_i|$ may be done a priori.

The tradeoff between the two computations is: a direct multiplication versus a log conversion, an addition, and an exponentiation. A comparison of hardware complexity and computation time can be made for the particular example of multiplying two six-bit numbers. An indicator of hardware complexity is the number of full adder circuits required. For a cobweb array multiplier, 30 adders are required if all product bits are retained; however, only 21 adders are required if the product is truncated to six bits. For the log conversion, twelve adders are required, for the log addition eight are used, and for the antilog conversion ten are needed, or a total of 30 adders are required to obtain 6-bit accuracy. Thus, for a multiplication the cobweb array is less complex and simpler than the log-antilog conversion.

Example 2—Parallel Digital Filter Bank: A particular digital filter configuration which often arises is the parallel filter bank which may be described by a z-transfer function of the form

$$H(z) = H_1(z) + H_2(z) + \cdots + H_r(z).$$

The block diagram of this filter is shown in Fig. 4(a). If each of the H_i are of the nonrecursive type, then the log conversion of x_i may be performed first as shown in Fig. 4(b). The resulting filters, G_i, would then require only an addition and an exponentiation for each multiplication. If any of the H_i are of the recursive [5] type described by a difference equation of the form

$$Y_n = \sum_{i=1}^{N-1} a_i x_{n-i} - \sum_{i=1}^{M} b_i Y_{n-i},$$

then, again the computation of log x_i may be moved ahead of the remaining filters.

Once more assuming 6-bit numbers, the cobweb array truncated to 6 bits would require 21 adders for each multiplication. An addition and exponentiation require 18 adders. Thus, if the number of filters R is four or more, the log-antilog conversion would require less hardware.

Example 3—A Multiplicative Digital Filter: Recently, Oppenheim *et al.* [6] presented a general method for nonlinear filtering of multiplied and convolved signals. The multiplicative techniques were applied to audio dynamic range compression and expansion and image enhancement. The block diagram of a multiplicative filter is shown in Fig. 5. If the input signal $S(t)$ consists of the product of two components $e(t)$ and $b(t)$, then the logarithm conversion reduces the process to the familiar additive process which may be filtered using linear techniques. The antilogarithm conversion reconstitutes the filtered signals.

The log and antilog conversions developed in this paper could be used for any digital realization of the multiplicative filter.

V. Summary

Algorithms for approximate binary logarithms and exponents and some applications of these algorithms to digital filtering have been described. Since the maxi-

mum product and division errors occur as percentages of the operands, these algorithms are suited for high-speed hardware rather than GP computer applications.

One such application is real time digital filtering. The computations involved are usually sums of products of a variable and constant coefficients. Using the log-antilog algorithms gives complete freedom of coefficient selection. For single multiplications a cobweb array multiplier is simpler. However, for other configurations, such as a parallel digital filter bank, the log-antilog technique is less complex. Also, there are applications, such as multiplicative digital filters, where log and exponent conversions are necessary.

References

[1] J. N. Mitchell, Jr., "Computer multiplication and division using binary logarithms," *IRE Trans. Electronic Computers*, vol. EC-11, pp. 512–517, August 1962.

[2] M. Combet, H. Van Zonneveld, and L. Verbeek, "Computation of the base two logarithm of binary numbers," *IEEE Trans. Electronic Computers*, vol. EC-14, pp. 863–867, December 1965.

[3] E. L. Hall, D. D. Lynch, and R. E. Young, "A digital modified discrete Fourier transform Doppler radar processor," *1968 EASCON Rec.*, pp. 150–159.

[4] J. F. Kaiser and F. Kuo, *System Analysis by Digital Computer*. New York: Wiley, 1966, pp. 218–277.

[5] C. M. Rader and B. Gold, "Digital filter design techniques in the frequency domain," *Proc. IEEE*, vol. 55, pp. 149–171, February 1967.

[6] A. V. Oppenheim, R. W. Schafer, and T. G. Stockham, Jr., "Nonlinear filtering of multiplied and convolved signals," *Proc. IEEE*, vol. 56, pp. 1264–1291, August 1968.

A Generalization of the Fast Fourier Transform

J. A. GLASSMAN

***Abstract*—A procedure for factoring of the $N \times N$ matrix representing the discrete Fourier transform is presented which does not produce shuffled data. Exactly one factor is produced for each factor of N, resulting in a fast Fourier transform valid for any N. The factoring algorithm enables the fast Fourier transform to be implemented in general with four nested loops, and with three loops if N is a power of two. No special logical organization, such as binary indexing, is required to unshuffle data. Included are two sample programs, one which writes the equations of the matrix factors employing the four key loops, and one which implements the algorithm in a fast Fourier transform for N a power of two. The algorithm is shown to be most efficient for N a power of two.**

***Index Terms*—Cooley-Tukey algorithm, discrete Fourier transform, fast Fourier transform, mixed radix, spectral analysis.**

THE fast Fourier transform, extensively covered in current journals [1]–[7] and popularly termed the Cooley–Tukey algorithm [4], can be generalized for any number of coefficients N by a factoring which does not shuffle the data. This factoring has three major effects on the fast Fourier transform. First, the transform is more easily explained, since the complexity of the tree graph to trace data is eliminated. Second, the mechanization is simplified since the final data need not be unshuffled, and third, the fast Fourier transform may be conveniently applied to any number of coefficients, although an application to anything but a power of two or four may be only of academic interest. The application of the fast Fourier transforms to any N, which may be termed a mixed radix algorithm, has appeared in recent articles [7]–[9] but no other algorithm has been found which does not involve scrambled data. This paper develops the basic matrix, demonstrates the algorithm for its factorization, and illustrates the factoring and the resulting fast Fourier transform with programs for time-sharing operation.

Manuscript received January 16, 1969; revised August 4, 1969.
The author is with Hughes Aircraft Company, Canoga Park, Calif.

The Discrete Fourier Transform

The Fourier transform $Y(w)$ of a function $x(t)$ is defined by the relation

$$Y(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}dt. \tag{1}$$

If $x(t)$ is sampled μ times, a sampled function $X^*(t)$ is produced, defined by

$$X^*(t) = \sum_{k=0}^{\mu-1} x(t)\delta(t - kT) \tag{2}$$

where $\delta(t)$ is the Dirac delta function. The Fourier transform of $X^*(t)$ is the discrete Fourier transform $Y^*(w)$:

$$\begin{aligned} Y^*(w) &= \int_{-\infty}^{\infty} x^*(t)e^{-jwt}dt \\ &= \int_{-\infty}^{\infty} \sum_{k=0}^{\mu-1} x(t)\delta(t - kT)e^{-jwt}dt \\ Y^*(w) &= \sum_{k=0}^{\mu-1} x(kT)e^{-jwkT}. \end{aligned} \tag{3}$$