Improved Mitchell-Based Logarithmic Multiplier for Low-power DSP Applications

Duncan J. McLaren, Student Member, IEEE
Institute for System Level Integration, Livingston, Scotland
Cadence Design Systems, Livingston, Scotland
Sponsored By Cadence Design Systems Inc. and the Engineering & Physical Sciences Research Council (EPSRC)

Abstract

This paper presents a method to improve the accuracy of a logarithmic multiplier, based on Mitchell's algorithms for calculating logarithms and antilogarithms. The method developed offers an area saving of approximately 50% and a power saving of 71% for larger input widths. A filter based on the multiplier is also presented.

Introduction

Multiplication in hardware has always been a cumbersome process and many of the solutions produced have large areas and consume a lot of power. An alternative method is to convert the numbers into logarithms. The main advantage of this is that the multiplication is replaced by an addition, which requires significantly less logic, however the logarithms and antilogarithms need to be calculated. There is a trade-off in that the calculated logarithms are approximations, which leads to errors in the answer. For Digital Signal Processing (DSP) this is less of a problem as systems can often deal with the extra noise introduced.

This paper introduces a method of calculating logarithms and antilogarithms developed by Mitchell [1] in 1962, as well as various alternatives. The error generated by the Mitchell algorithms is analysed and is used to develop a method for improving the accuracy of the result. The area and power characteristics of the various multipliers are then given, which show that the Mitchell and Improved-Mitchell multipliers offer both area and power savings over the standard multiplier. The final section introduces a filter implemented using the Standard, Mitchell and Improved-Mitchell multipliers to show the benefits of using the multipliers in a real application.

Multiplication Using Binary Logarithms

The technique of multiplying two numbers using logarithms is a simple one. Take the logarithms of the two multiplicands, add the logarithms together and then take the antilogarithm of the resulting summation. The problem then becomes one of how to calculate the logarithms and antilogarithms.

Although the main focus of this paper is on an improved Mitchell-based multiplier, various alternative approaches for calculating logarithms were investigated to provide a comparison to the Mitchell and Improved-Mitchell multipliers. These were; a pure Look-Up Table (LUT); a LUT plus an interpolator, and a non-linear LUT and interpolator. The pure LUT stores a (pre-calculated) value for the logarithm of every possible input value. The logarithm is obtained by looking for its value in the table. The LUT and Interpolator use a similar table to the LUT, but do not store every value. Instead it stores half, for example, and uses linear interpolation to estimate the values between the look-up points. The interpolator formula is

$$f(x) = f(a) + (x-a)\left(\frac{f(b)-f(a)}{b-a}\right) \quad (1)$$

This formula still requires a multiplication. However, as it is working on numbers that have fewer bits than the full multiplication, the multiplier used here will be smaller in terms of area. For example, an 8-bit input logarithmic multiplier would us a 4-bit multiplier for the interpolation. The non-linear LUT and interpolator exploits the shape of the logarithm curve and store the logarithms for every power of 2 (i.e. 1_2, 10_2, 100_2, ...). These values are used because the logarithms have no fractional component, which means that the value stored in the LUT is exact.

Mitchell presented in his paper a method of calculating logarithms and antilogarithms which is summarised below:

Let N be a binary number. To calculate the logarithms first locate the Most Significant Bit (MSB) which gives the characteristic of the logarithm. Mitchell indicates that this can be done using a shift and count approach; however this would require multiple clock cycles to achieve. An alternative, purely combinatorial, circuit was developed which allows the characteristic to be found in a single clock cycle. The remaining bits are then shifted down (below the binary point) to give the binary fraction and merged with the characteristic to give the approximation. As an example, take N=25, which has a binary representation of 11001_2. The MSB is bit 4, which gives a characteristic of 100_2, and the remaining bits (1001_2) gives the binary fraction. This gives a value for the logarithm of 100.1001_2 ($=4.5625_{10}$). The correct value of $\log_2(25)$ is 4.6439.

A similar approach is used to calculate the antilogarithm. The characteristic of the logarithm is used to determine the MSB of the final result. The binary fraction is inserted to the right of the set bit. Take the

example above, where the logarithm is 100.1001_2. This gives an MSB of 10000_2, and 0.1001_2 is left-shifted by 4 to give 1001.0_2. When these two values are OR'd the answer is 11001_2, which is the number N.

Using the algorithms above, a circuit for calculating multiples was developed and is shown in Fig 1. The final block ("Check for Zero") is required to ensure that the output is zero when either of the inputs are zero.

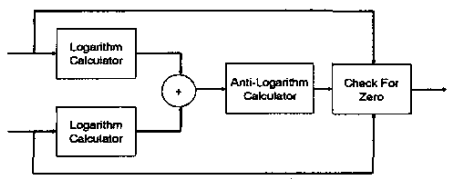

Fig 1. Multiplier Block Diagram.

Others [3,4] have used Mitchell's algorithms to implement a multiplier similar to the one above, however they do not implement the error correction function.

Improvements to the Mitchell-Based Multiplier

Mitchell showed in his paper that the error in the product (using his method) is dependant only on the binary fraction components of the two multiplicands. The important points are repeated here.

A binary number, N, can be written

$$N = \sum_{i=j}^{k} 2^i z_i \tag{2}$$

or $$N = 2^k (1+x) \tag{3}$$

where $x = \sum_{i=j}^{k-1} 2^{i-k} z_i$

Note that k represents the characteristic and x the binary fraction, with x in the range $0 \le x < 1$. The true logarithm and the approximation using the Mitchell method are

$$(\log_2 N)_{true} = k + \log_2 (1+x) \tag{4}$$

$$(\log_2 N)_{approx} = k + x \tag{5}$$

The logarithm of a product is equal to the sum of the logarithms of the multiplicands.

$$(\log_2 P)_{true} = k_1 + \log_2 (1+x_1) + k_2 + \log(1+x_2) \tag{6}$$

$$P_{true} = 2^{(k_1+k_2)} (1+x_1)(1+x_2) \tag{7}$$

The subscripts $_1$ and $_2$ refer to the two multiplicands. The approximation is

$$(\log_2 P)_{approx} = k_1 + x_1 + k_2 + x_2 \tag{8}$$

This is split into two separate equations to account for the case were $x_1+x_2 < 1$ (i.e. no carry from the binary fraction to the characteristic) and where $x_1+x_2 \ge 1$ (i.e. there is a carry from the binary fraction).

$$(\log_2 P)_{approx} = k_1 + k_2 + (x_1 + x_2) \qquad x_1+x_2 < 1 \tag{9}$$

$$(\log_2 P)_{approx} = 1 + k_1 + k_2 + (x_1 + x_2 - 1) \qquad x_1+x_2 \ge 1 \tag{10}$$

The antilogarithms of (9) and (10) are

$$P_{approx} \approx 2^{(k_1+k_2)} (1+x_1+x_2) \qquad x_1+x_2 \ge 1 \tag{11}$$

$$P_{approx} \approx 2^{(1+k_1+k_2)} (x_1+x_2) \qquad x_1+x_2 \ge 1 \tag{12}$$

The error in the multiplication, E_m, is defined as

$$E_m = \frac{P_{approx} - P_{true}}{P_{true}} = \frac{P_{approx}}{P_{true}} - 1 \tag{13}$$

Substituting (7) and (11), and (7) and (12) into (13)

$$E_{m1} = \frac{1+x_1+x_2}{(1+x_1)(1+x_2)} \qquad x_1+x_2 < 1 \tag{14}$$

$$E_{m2} = \frac{2(x_1+x_2)}{(1+x_1)(1+x_2)} \qquad x_1+x_2 \ge 1 \tag{15}$$

(14) and (15) show that the error in the multiplication is due to the value of the binary fraction of the two multiplicands. Mitchell showed that this error ranged from 0% to 11.1111%, and the maximum occurred when $x_1 = x_2 = ½$.

The above results were used as the basis for providing a more accurate approximation for the multiplication. The formula gives the relative error that is generated for every possible value of x_1 and x_2. To correct the error the following is used

$$P_{correct} = \frac{P_{approx}}{(1-E_m)} \tag{16}$$

Taking logarithms of (16) gives

$$\log_2 (P_{correct}) = \log_2 (P_{approx}) + \log_2 \left(\frac{1}{1-E_m} \right) \tag{17}$$

This shows that to provide the correct answer, an error correction factor should be added to the summation before the antilogarithm is calculated. This correction is only dependent upon the fractional part and so can be repeated for all characteristics. Theoretically this would require a different value for every combination of x_1 and $x2$; however this would be impractical. The approach is to average the value of the correction factor over a range of x-values, and add this to the summation. This results in a multiplier of improved accuracy. The two fractional parts are split into 8 ranges, from 0 to 1 in steps of 0.125. This means that the 3 most significant bits of x can be used to determine the error correction factor (which is pre-calculated). The inputs were split into 8 as this gave sufficient granularity in the selection of the error correction factor, whilst keeping the logic needed to a minimum. The figure below shows the factors for each combination of x (in decimal).

	0 → 0.125	0.125 → 0.25	0.25 → 0.375	0.375 → 0.5	0.5 → 0.625	0.625 → 0.75	0.75 → 0.875	0.875 → 1
0 → 0.125	0.0048	0.0131	0.0198	0.0254	0.0301	0.0342	0.0377	0.0266
0.125 → 0.25	0.0131	0.0360	0.0548	0.0705	0.0838	0.0952	0.0910	0.0328
0.25 → 0.375	0.0198	0.0548	0.0837	0.1080	0.1287	0.1324	0.0811	0.0250
0.375 → 0.5	0.0254	0.0705	0.1080	0.1397	0.1527	0.1093	0.0601	0.0187
0.5 → 0.625	0.0301	0.0838	0.1287	0.1527	0.1186	0.0774	0.0428	0.0133
0.625 → 0.75	0.0342	0.0952	0.1324	0.1093	0.0774	0.0507	0.0282	0.0088
0.75 → 0.875	0.0377	0.0910	0.0811	0.0601	0.0428	0.0282	0.0157	0.0049
0.875 → 1	0.0266	0.0328	0.0250	0.0187	0.0133	0.0088	0.0049	0.0015

Fig 2. Calculated Error Correction Values

Fig 3 shows the error spread for the original and improved Mitchell multipliers.

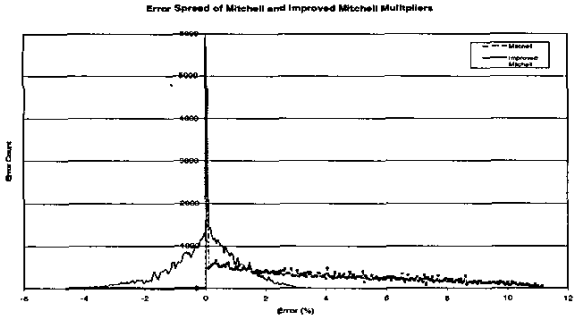

Fig 3. Error Spread of Original & Modified Mitchell Multipliers

The mean and standard deviation of the original method are 3.614 and 3.107 respectively. With the modifications to the method the mean and standard deviation are now -0.0363 and 1.175 respectively. This shows that there are more errors around zero (with about 68% of all errors between -1.211 and 1.139 for the modified version, as opposed to between 0.507 and 6.721 for the original version). Note that there are now negative errors, which gives the error-spread a more Gaussian shape.

An alternative method for improving the accuracy of the Mitchell algorithms is presented in [2], however this approach uses an adaptive linear interpolator at the logarithm and anti-logarithm calculations to ensure the approximation is more accurate. This method is more complex than the method presented here, and would be difficult to implement in hardware.

Characteristics of Multipliers

Each multiplier was synthesised at 50MHz using a 0.18µm technology library. The resultant areas are shown in Fig 4. The Mitchell and Improved-Mitchell based multiplier offer significant area improvements over the standard multiplier (i.e. the multiplier obtained when the code is "c = a*b") for input widths above 12 bits. The improvement to the Mitchell-based multiplier increases the gate count of the Improved-Mitchell multiplier over the Mitchell multiplier by approximately 30%.

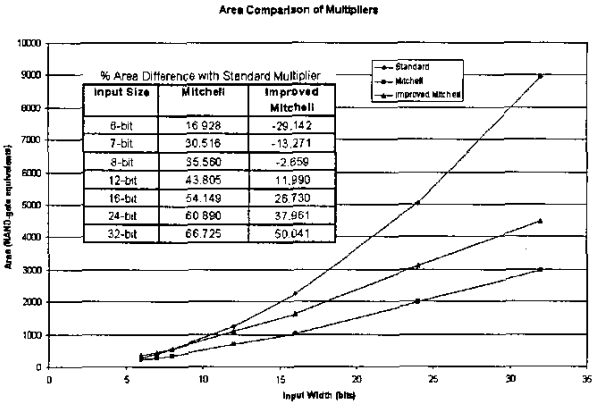

Fig 4. Area Comparison of Logarithmic Multipliers

The other methods of calculating logarithms were also synthesised; however they were significantly larger than the above designs. For example, the 8-bit Look-Up-Table implementation was approximately 14k gates, whereas the designs above were all in the region of 500 gates.

Fig. 5 shows the power (in mW) that the standard, Mitchell and Improved-Mitchell multipliers consumed when running a testbench that performs 1,048,576 calculations (i.e. the two inputs go from 0 to 1023).

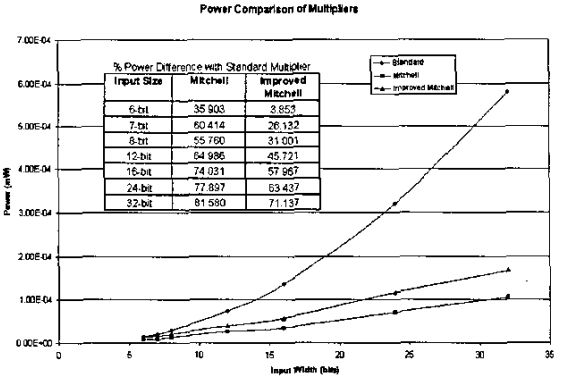

Fig 5. Power Comparison of Logarithmic Multipliers

Again the Mitchell and Improved-Mitchell multipliers offer significant power savings over the standard multiplier, with greater savings for larger input widths. The Improved-Mitchell multiplier increases the power by about 40% over the Mitchell multiplier.

The design of the multiplier allows itself to be easily pipelined, which enables it to run at higher speeds. When this is done to the 16-bit Improved Mitchell, with the pipeline stages being; the logarithm calculation; the summation and error correction, and the antilogarithm calculation, then the complete multiplier will run at approximately 345MHz. Table I shows the maximum speeds at which the various stages of the multiplier can run.

Table I – Maximum Speed of Multiplier Stages

Stage	Maximum Speed
Logarithm Calculator	450 MHz
Error Correction	345 MHz
Antilogarithm Calculation & Zero Check	500 MHz

Filter Implementation

To test the multiplier further, it was used as part of a real application, in this case a Finite Impulse Response (FIR) Filter. The filter was an 11-tap low-pass FIR, with a normalized cut-off frequency of 0.25. The filter was implemented in Verilog using the standard multiplier, the un-modified Mitchell multipliers and the Improved Mitchell multipliers. The input was 16-bit and the output was 32-bit. The figure below shows the magnitude response from each of the three implementations.

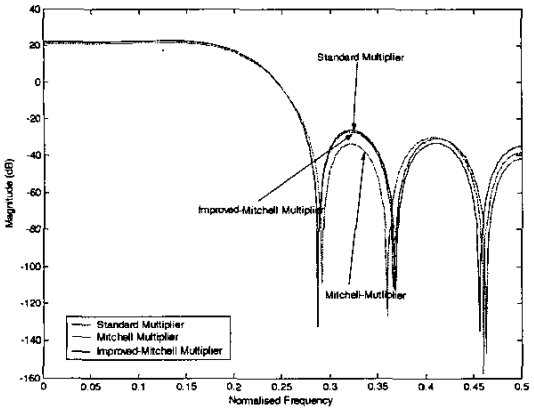

Fig 6 –Filter Magnitude Response

A simple two-tone signal was inputted to the filter, with the first tone at a (normalized) frequency of 0.05, and the second tone at 0.25 (the cut-off frequency). To show the differences between the 3 implementations, a close up of the peak of the filtered signal is shown in Fig 8.

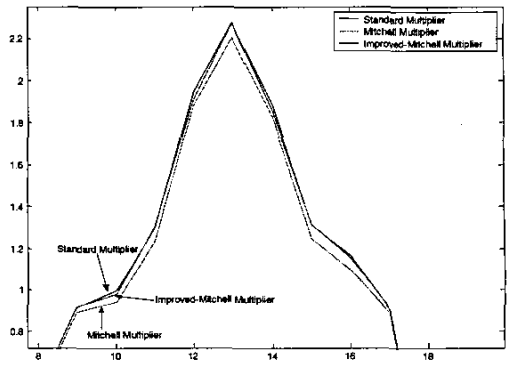

Fig 7 – Close up of peak

The filters were synthesised at 50MHz. Table II shows the area of the three filter implementations.

Table II – Area of Filter Implementation

Multiplier Implementation	Area (gates)	% Saved
Standard	37 k	N/A
Mitchell	22 k	40.95
Improved-Mitchell	28 k	24.27

The final column indicates the area saving with respect to the standard multiplier implementation.

Conclusion

This paper has introduced a simple method to improve the accuracy of logarithmic multiplier based on Mitchell's algorithms. The improved method increases the area and power by about 30% over an unmodified Mitchell-based multiplier, but this is still significantly less than the area and power of a standard multiplier for larger input widths.

The multiplier can also run at high speeds by pipe-lining the 3 main stages (namely logarithm calculation; summation and error correction; and antilogarithm calculation).

References

(1) J. N. Mitchell, "Computer multiplication and division using binary logarithms", *IRE Transactions on Electronic Computers*, pp. 512-517, August 1962.

(2) E. L. Hall, D. D. Lynch and S.J. Dwyer III, "Generation of products and quotients using approximate binary logarithms for digital filtering applications", *IEEE Transactions on Computers*, vol. C-19, pp. 97-105, February 1970

(3) K. H. Abed, R. Siferd, "CMOS VLSI Implementation of 16-bit logarithm and anti-logarithm converters", *IEEE Midwest Symposium on Circuits and Systems*, August 2000.

(4) S Ramaswamy, R. Siferd, "CMOS VLSI implementation of a digital logarithmic multiplier", *Proceedings of the IEEE National Aerospace and Electronics Conference*, vol 1, pp 291-294, May 1996