Truncated Error Correction for Flexible Approximate Multiplication

Michael B. Sullivan
School of Electrical and
Computer Engineering
University of Texas at Austin
Austin, Texas 78712
Email: mbsullivan@utexas.edu

Earl E. Swartzlander, Jr.
School of Electrical and
Computer Engineering
University of Texas at Austin
Austin, Texas 78712
Email: eswartzla@aol.com

***Abstract*—Binary logarithms can be used to perform computer multiplication through simple addition. Exact logarithmic (and anti-logarithmic) conversion is prohibitively expensive for use in general multipliers; however, inexpensive estimate conversions can be used to perform approximate multiplication. Such approximate multipliers have been used in domain-specific applications, but existing designs either offer superior efficiency or flexibility. This study proposes a flexible approximate multiplier with improved efficiency. Preliminary analysis indicates that this design provides up to a 50% efficiency advantage relative to prior flexible approximate multipliers.**

***Index Terms*—Truncated error correction, approximate binary logarithms, approximate multiplication, computer arithmetic.**

I. Introduction

Over the past decade, the energy efficiency of computation has become a primary design constraint. One possible avenue towards energy efficient arithmetic is to relax the stringent design requirements of exact computation and instead perform *approximate* arithmetic. Approximate arithmetic units have the potential to save power and latency relative to conventional arithmetic circuits.

Among approximate arithmetic units, multiplication is particularly attractive because of the cost and utilization of high-speed multipliers. Approximate multipliers have successfully been applied to diverse application areas, including computer graphics [1], neural networks [2], DSP filtering [3], and specialized circuitry for nuclear power plants [4].

Multiplication can be performed in the logarithmic domain with great speed and efficiency. For this reason, approximate logarithmic conversion is a popular approach for low-cost approximate multiplication [3], [5], [6], [7].

While various prior approximate logarithmic multipliers have been investigated, existing designs *either* offer superior efficiency or flexibility. Approximate logarithmic multipliers generally fall into two classes (*iterative* or *non-iterative*) that vary in their flexibility. Non-iterative multipliers compute using a fixed precision which is set at design-time, providing superior efficiency but limited runtime flexibility. Iterative approximate multipliers have the ability to refine results to an arbitrary precision, at a higher cost per bit of precision.

This paper investigates an iterative approximate multiplier based on a concept called *truncated error correction*. Truncated error correction extends prior work on iterative approximate logarithmic multiplication, resulting in a flexible approximate multiplier with improved efficiency.

A. Approximate Binary Logarithms for Multiplication

Before describing the main contribution of this paper, some basic properties and definitions of approximate binary logarithmic multiplication are reviewed. While high-speed multiplication is relatively expensive using fixed-point inputs, binary logarithms can be multiplied as in (1) using simple and efficient hardware. Therefore, if conversion to and from logarithmic format were inexpensive then fixed-point numbers could be efficiently multiplied as in (2).

$$\log_2(N_1 * N_2) = \log_2(N_1) + \log_2(N_2) \quad (1)$$

$$N_1 * N_2 = \log_2^{-1}\left(\log_2(N_1) + \log_2(N_2)\right) \quad (2)$$

Conversion between fixed-point binary numbers and exact binary logarithms is expensive and time consuming, making exact multiplication through logarithmic conversion prohibitively costly. However, inexpensive *approximate* conversions exist, and approximate logarithms can be used to cheaply approximate fixed-point multiplication. Such approximate logarithmic multiplication can be expressed by (3), where $\widetilde{\log_2}$ and $\widetilde{\log_2}^{-1}$ denote approximate binary logarithmic conversion and anti-conversion, respectively.

$$N_1 * N_2 \approx \widetilde{\log_2}^{-1}\left(\widetilde{\log_2}(N_1) + \widetilde{\log_2}(N_2)\right) \quad (3)$$

A W-bit fixed-point input, N, can be expressed as $N = 2^k * (1 + f)$. As a matter of notation, k is referred to as the *characteristic* and f is referred to as the *fractional component* of N.

II. Prior Approximate Logarithmic Multipliers

The precision of an approximate multiplier can be set at design time, or it can be varied according to user needs. Non-iterative designs compute an approximate result with a fixed level of precision; iterative approximate multipliers can refine prior results to satisfy diverse precision needs.

A. Non-Iterative Approximate Multipliers

Non-iterative approximate multipliers seek to approximate logarithm and anti-logarithm conversion with enough fidelity for a given application. Precise non-iterative logarithmic approximation is achieved through piece-wise linear interpolation [3], [4], [8], higher-order interpolation [9], memory lookup [10], or a combination of methods [6].

The most notable benefit of non-iterative approximate multipliers is that they can provide a low cost per bit of precision.

However, because the precision of these designs is fixed at design time, any single static multiplier is likely to only serve a small number of user needs. In addition, a user with specific precision needs that do not match up precisely with an existing non-iterative design will likely have to combine multiple existing approaches in an ad-hoc manner in order to design an appropriate approximate multiplier.

B. Iterative Approximate Multipliers

In contrast to static approximate multipliers, iterative approximate logarithmic multipliers seek to refine a result until a desired precision is achieved. This level of precision can be varied at runtime, which may allow an iterative approximate multiplier to satisfy a diverse range of applications. While iterative approximate multiplication is flexible, existing designs suffer from a high cost per bit of precision. A review of existing iterative approximate binary logarithmic multipliers follows.

The iterative reduction of approximation error has been recognized since the beginnings of approximate logarithmic multiplication. The first approximate binary logarithmic multiplier, proposed by Mitchell over 50 years ago, includes an iterative scheme for flexibly meeting diverse precision needs [5]. Mitchell's scheme introduces a relatively large maximum relative error of 11.111% per iteration, but can use multiple iterations to provide any level of precision. Figure 1a shows a contour plot of the relative error introduced during the first iteration of Mitchell's approximation. This relative error depends only on the fractional component of each input, such that this error applies to inputs with any positive characteristics ($k1$ and $k2$).

While Mitchell's scheme supports iterative refinement, each iteration cannot begin until the previous iteration is nearly complete, making Mitchell's iterative refinement a long-latency operation. Figure 2a shows a block diagram of Mitchell's approximation. The components include a leading-one detector (with some control logic), a binary encoder, an adder, two shifters, and some masking logic to prepare the inputs for the next iteration. While this masking logic completes quickly, the next iteration cannot begin until the carry-out of the fractional component adder (CO') is available.

Babić *et al.* [7] have extended Mitchell's scheme to allow greater pipeline-level parallelism. Their design eliminates the near-serial dependency between successive iterations, improving the latency required to reach a high level of precision. However, due to internal simplifications, each stage of their iterative approximate multiplier is even less accurate than those of Mitchell's original algorithm–Babić's scheme has a maximum relative error of 25% per iteration. Figure 1b shows the relative error introduced during the first iteration of Babić's approximation, and Figure 2b shows a block diagram of the scheme. The major advantage of Babić's approximation is that successive iterations may begin immediately after leading-one detection. However, the limited precision per iteration of Babić's scheme means that a large number of pipelined iterations are needed to achieve a reasonable amount of precision.

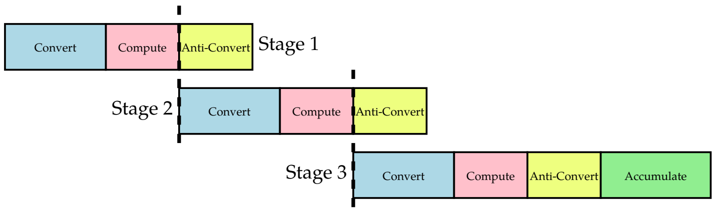
III. Truncated Error Correction

This work proposes an approximate multiplier that can efficiently pipeline iterative refinement (similar to [7]) while also providing high precision per iteration. Truncated error correction (TEC) places a small amount of error correcting circuitry along with the arithmetic in each iteration. This circuitry inexpensively replicates the effects of multiple Babić pipeline iterations for the most problematic inputs. Truncated error correction is characterized by two parameters:

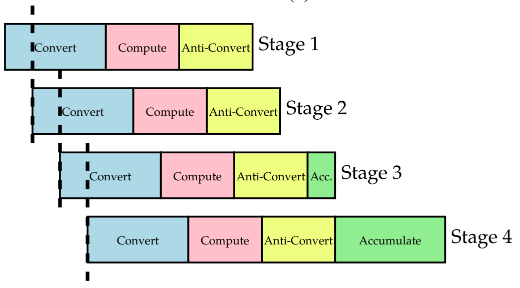
1) The truncation region width (t) determines the subset of inputs to consider for correction.
2) The truncation depth (d) limits the maximum number of successive Babić iterations to mimic.

Truncated error correction is based on the intuition that Babić's approximation suffers the worst approximation error for inputs whose fractional components approach 1. Handling these worst-case inputs in one iteration with TEC circuitry can replicate the maximum-error behavior of multiple Babić iterations with little added cost. A more in-depth explanation of the error behavior of truncated error correction is found below in Section III-A.

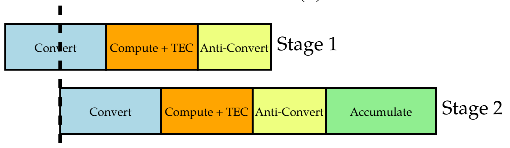
Figure 2c shows a block diagram of iterative multiplication using truncated error correction. Components that change relative to Babić's approximation are highlighted. These changes are described in more detail below in Section III-B.



(a) Mitchell

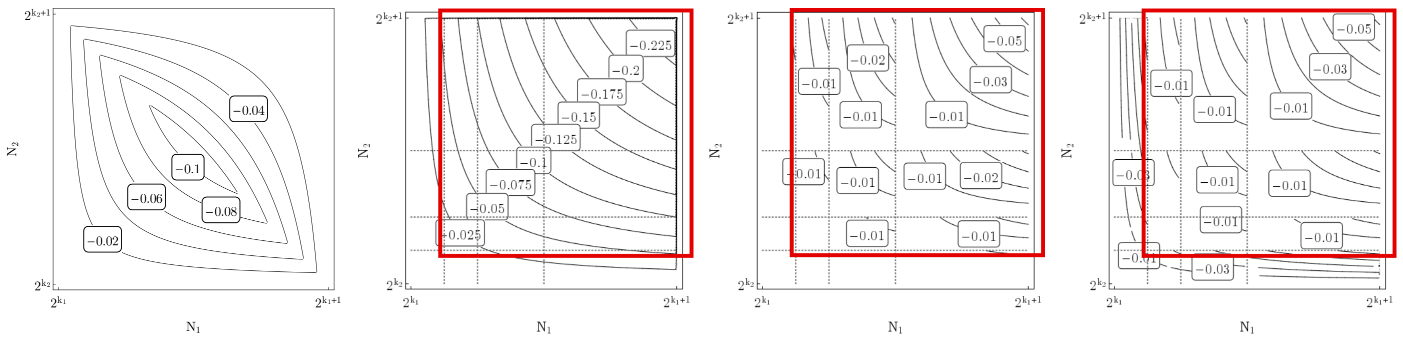


(b) Babić



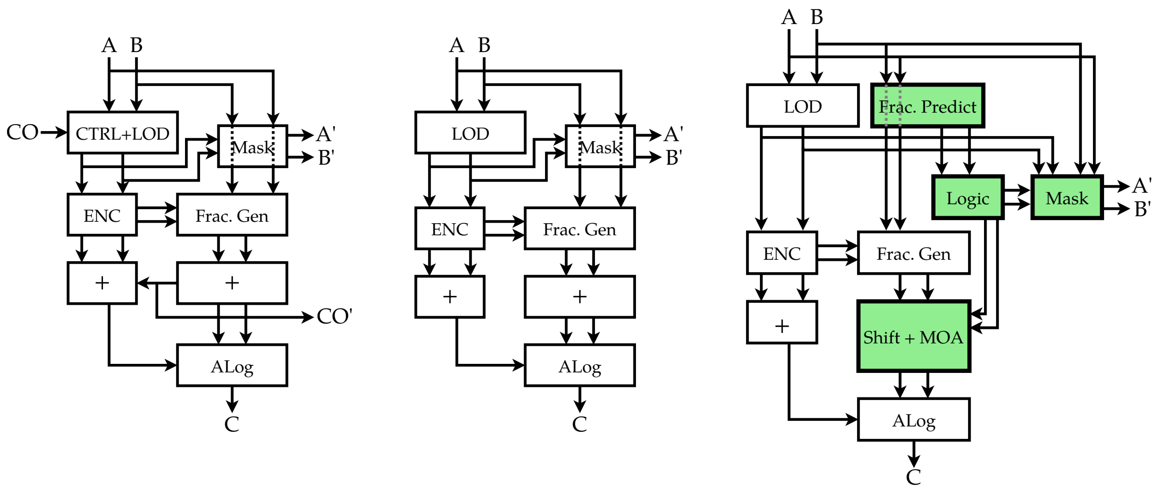
(c) Truncated Error Correction

Fig. 3. An illustration of different pipelined iterative approximate multipliers with roughly equivalent approximation errors. The time spent in each phase of computation is approximate; the scale of some phases has been altered for readability.



(a) Mitchell, 1st iteration (b) Babić, 1st iteration (c) Babić, 2nd iteration (d) TEC ($t\!=\!3, d\!=\!1$), 1st iteration

Fig. 1. A comparison of error behaviors between different iterative approximate logarithmic multipliers.



(a) Mitchell (b) Babić (c) Truncated EC

Fig. 2. Functional diagrams of different iterative approximate logarithmic multipliers.

Figure 3 shows an illustration of different pipelined iterative multipliers, with pipeline iterations accumulated through multi-operand addition. The near-serial dependency in Mitchell's algorithm increases its latency, and the low precision per iteration of Babić's scheme increases its cost. In contrast, truncated error correction completes in fewer iterations than either approach while effectively exploiting pipeline-level parallelism, providing superior efficiency.

A. TEC Error Behavior

The approximation error of Babić's scheme increases monotonically with the magnitude of the fractional components of its inputs; the worst approximation errors occur for inputs whose fractional components both approach 1. Figure 1b illustrates this phenomenon–the highlighted area contains those inputs where the leading-bit of each fractional input falls within a truncation region of $t = 3$ bits ($f_1 \geq 0.125$ and $f_2 \geq 0.125$). Truncated error correction inexpensively mimics the behavior of multiple Babić iterations for these problematic inputs–focusing TEC control logic on a small number of leading bits in the fractional components avoids much of the overheads associated with multiple full Babić iterations. As an illustrative example, Figure 1d shows the approximation error of one TEC iteration that mimics two Babić iterations for any inputs falling within a truncation window of 3 bits ($t\!=\!3$ and $d\!=\!1$). The highlighted region is corrected, and has an approximation error equal to that of the second iteration of Babić's scheme (Figure 1c). However, any inputs outside of the truncation window are not corrected and remain as they were following the first Babić iteration. While inputs outside the truncation window remain uncorrected, the maximum approximation error is equal to that following two full Babić iterations.

Table I shows the maximum and average relative error of different truncated error correcting configurations. Because approximate logarithmic multiplication calculates the characteristic exactly, approximation error is localized to the least-significant bits of the fractional component. Relative error can be expressed either as a percentage or as the minimum number of correct bits in the fractional component of a result. Accordingly, one iteration of Mitchell's and Babić's approximation provide a maximum relative error of 11.111%/3.47 bits and 25%/2 bits, respectively.

It can be seen that in order for a single TEC iteration to achieve a maximum error equivalent to I iterations of Babić's approximation, t and d must be chosen such that $t \geq (I\!+\!1)$ and $d \geq (I-1)$. The least costly combinations of t and d that have a maximum error equivalent to 2, 3, and 4 Babić iterations are bolded in Table I. One TEC iteration can provide 4 bits of precision (at $t\!=\!3$ and $d\!=\!1$), 6 bits of precision (at $t = 4$ and $d = 2$), or 8 bits of precision (at $t = 5$ and $d = 3$).

TABLE I
THE ERROR BEHAVIOR OF TRUNCATED ERROR CORRECTION.

Depth, d	Max Err. (%/bits)	Width, t	Ave. Err. (%/bits)
0	25/2	-	9.28/3.43
1	**6.25/4**	**3**	**1.36/6.20**
	6.25/4	4	1.05/6.57
	6.25/4	5	0.980/6.67
2	6.25/4	3	0.777/7.01
	1.56/6	**4**	**0.304/8.36**
	1.56/6	5	0.156/9.32
3	6.25/4	3	0.764/7.03
	1.56/6	4	0.260/8.59
	0.391/8	**5**	**0.086/10.18**

TABLE II
ADDED COST OVERHEAD RELATIVE TO A SINGLE BABIĆ ITERATION.

t / d	W	Logic (%)	Shift (%)	MOA (%)	Mask (%)	Total (%)
3 / 1	32	1.07	33.0	14.7	17.1	72.4
	64	0.457	28.7	12.8	15.2	63.0
	128	0.200	25.4	11.3	13.6	55.6
4 / 2	32	2.54	95.4	29.3	22.1	164
	64	1.09	83.1	25.6	19.8	143
	128	0.478	73.4	22.6	17.8	126
5 / 3	32	5.03	187	44.0	27.0	290
	64	2.16	163	38.4	24.4	251
	128	0.945	144	33.9	21.9	221

Equivalently, the maximum approximation error decreases by a factor of 4 each time the truncation depth increases by 1 (when $t = (d+2)$). Higher-precision TEC iterations are possible, but are not analyzed in this study.

B. TEC Circuitry

Truncated error correction places additional circuitry in each pipeline stage. The major components of this circuitry are briefly described below.

Truncated error correction utilizes specialized **fraction prediction logic** to obtain the low-cost, low latency prediction of the top bits of the fractional component.

A **shared logic** block is used by both the masking logic and the main datapath. This logic operates on the predicted fractional components of each input to identify the position of bit-pairs that fall completely within the truncation region.

The fractional components of any bit-pairs in the truncation region are passed through shifters before being accumulated by a multi-operand adder. The maximum distance that each operand may be shifted depends on the (small) truncation width, such that fast **short-distance shifters** made out of selector-multiplexers suffice for the task. These shifters accept a one-hot encoded shift amount that is extracted by the aforementioned shared logic block.

Multi-operand addition is used to accumulate these shifted operands with the original fractional components; a tree of (n;2) compressors composed of full adder cells is used for this accumulation.

Modified **masking logic** is needed to prepare the inputs of each iteration for the next stage of pipelined approximation. This logic masks out the leading-one bit from the current computation, as well as any bit-pairs that fall completely within the truncation region.

IV. TRUNCATED ERROR CORRECTION EVALUATION

Truncated error correction complicates the input masking circuitry, increasing the cost and critical path between pipeline iterations. TEC circuitry also increases the cost and critical path of computation through multi-operand addition, shifting, and arithmetic control logic. However, a pipelined TEC design requires fewer iterations in order to achieve the same precision. Using fewer pipeline iterations saves power and latency, outweighing the added cost and delay of truncated error correction. The cost, delay, and efficiency of truncated error correction are analyzed below.

A. Methodology

Analyses in this paper make use of a simplified unit gate model that characterizes each design element according to two metrics. The first is an abstract concept of *cost*, denoted by C, which is assumed to be proportional to both area and dynamic power consumption. The second is *time*, T, which is proportional to the delay of a design element. Each circuit design is hierarchically decomposed into simple components in order to analytically characterize its cost and delay. Some basic assumptions of the unit gate model follow.

- Simple 2-input gates (AND, OR) $[C = 1, T = 1]$
- 2-input XOR gates, MUXes, and HA cell $[C = 2, T = 2]$
- Full-adder standard cell $[C = 4, T = 4]$
- Complex gates (AOI, etc.) composed of 2-input gates
- m-input gates composed of a tree of 2-input gates

The unit gate model does not consider buffering or wiring, both for simplicity and because synthesis tools add an element of uncertainty to the cost and delay of these elements. In order to compensate for added buffering and wiring costs due to truncated error correction, a conservative 10% cost overhead is added to the TEC circuitry.

B. Experimental Results

Table II gives the added cost of TEC circuitry[1] relative to the cost of a single iteration of Bulić's approximation. The total cost of TEC circuitry is slightly larger than the sum of each component, due to the correction for wiring and buffering costs. It is apparent that TEC circuitry is less costly than the equivalent number of Babić stages (which would incur 100% overhead per stage), and that TEC approximation scales favorably with the input width (W).

Table III gives the additional latency incurred by TEC circuitry relative to the delay of a single iteration of Bulić's approximation. Both the path between successive pipeline iterations (Mask) and the path through an iteration (Output) are given. The added latency between successive pipeline iterations is significant, and demonstrates an important area of future optimization for truncated error correction.

Despite the added cost and latency per TEC iteration due to correction circuitry, truncated error correction significantly improves the efficiency of pipelined designs. The reason for this is twofold. First, as was noted before, the overall cost

[1]Fractional component prediction is included in the masking logic cost.

TABLE III
ADDED DELAY OVERHEAD RELATIVE TO A SINGLE BABIĆ ITERATION.

t / d	*W*	Mask (%)	Output (%)
3 / 1	32	167	18.2
	64	157	15.4
	128	150	13.3
4 / 2	32	233	18.2
	64	213	15.4
	128	200	13.3
5 / 3	32	350	45.5
	64	314	34.6
	128	288	26.7

TABLE IV
BENEFITS OF TEC RELATIVE TO PIPELINED BABIĆ APPROXIMATION.

Min. Prec. (bits)	*t / d*	*W*	Cost (%)	Delay (%)	*C*D* (%)
4	3/1	32	-30.8	-20.0	-44.6
		64	-34.0	-21.1	-48.0
		128	-36.6	-21.8	-50.5
8	5/3	32	-16.1	-21.0	-33.7
		64	-23.8	-25.5	-43.3
		128	-29.8	-29.0	-50.1
10	3/1+4/2	32	-15.0	-4.60	-18.9
		64	-20.1	-5.94	-24.9
		128	-24.1	-6.96	-29.4
12	4/2+4/2	32	-15.0	-6.45	-20.4
		64	-20.9	-8.33	-27.5
		128	-25.6	-9.76	-32.9

of TEC circuitry is less than the equivalent number of Babić stages. Second, TEC designs require fewer stages, which in turn requires less circuitry to accumulate the results of each pipeline stage. Table IV shows the cost, delay, and efficiency (cost-delay product) of a pipelined TEC design relative to the equivalent Babić approximation. Scalable efficiency gains up to 50% can be seen due to truncated error correction. Efficiency is improved in every design, but gains are lowered to 20-30% in pipelined TEC designs, due to the added latency between TEC iterations.

V. FUTURE WORK

Preliminary analyses indicate that approximate multiplication with truncated error correction is more flexible than static approximation and is more efficient than prior iterative approximate multipliers. More extensive experimentation is needed to fully understand the benefits of truncated error correction. Future work will use synthesis-based efficiency estimates and will compare the precision proportionality of truncated error correction relative to both static and iterative approaches. Also, future design space exploration will investigate the behavior of higher precision TEC iterations and will more fully analyze the average-case error behavior of competing approaches.

Further design optimizations are possible for the TEC scheme considered in this work. Cost results in Table II show that the cost of short-distance shifting prior to multi-operand addition dominates the TEC overheads. Fast shifters are used in this study, despite the fact that this computation is off of the critical path for all but the last pipeline iteration. Future work will include alternative shifter designs, with an emphasis on balancing out the delay of pipelined TEC approximation. Delay optimizations will also be explored for the path between successive TEC iterations, in order to increase the efficiency of pipelined designs.

Approximate logarithms can be used to efficiently approximate operations other than two-operand multiplication. Future work may extend truncated error correction to related arithmetic operations, including approximate constant multiplication, multi-operand multiplication, and approximate exponentiation[2].

VI. CONCLUSION

This work proposes a flexible approximate multiplier based on a concept called truncated error correction. Initial analyses indicate that iterative approximate multiplication with truncated error correction provides benefits over all existing approaches. Iterative approximate multiplication with truncated error correction provides greater flexibility than static approximate multiplication, and has latency, cost, and precision benefits over prior iterative multipliers. The flexibility and efficiency of truncated error correction may make it an attractive alternative to fixed-point multiplication for application-specific designs.

ACKNOWLEDGMENTS

Michael Sullivan's research was supported by the Temple Foundation through an MCD Fellowship in Engineering. Earl Swartzlander is supported in part by a grant from AMD, Inc.

REFERENCES

[1] H. Tian, S. Lam, and T. Srikanthan, “Implementing Otsu's thresholding process using area-time efficient logarithmic approximation unit,” in *International Symposium on Circuits and Systems*, vol. 4, 2003.

[2] U. Lotri and P. Buli, “Logarithmic multiplier in hardware implementation of neural networks,” in *Adaptive and Natural Computing Algorithms*. Springer Berlin / Heidelberg, 2011, pp. 158–168.

[3] E. Hall, D. Lynch, and S. Dwyer, “Generation of products and quotients using approximate binary logarithms for digital filtering applications,” *IEEE Transactions on Computers*, vol. C-19, no. 2, pp. 97–105, 1970.

[4] M. Combet, H. Van Zonneveld, and L. Verbeek, “Computation of the base two logarithm of binary numbers,” *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 6, pp. 863–867, 1965.

[5] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[6] D. Mclaren, “Improved Mitchell-based logarithmic multiplier for low-power DSP applications,” in *IEEE International Systems-on-Chip Conference*, 2003.

[7] Z. Babić, A. Avramović, and P. Bulić, “An iterative logarithmic multiplier,” *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[8] K. Abed and R. Siferd, “CMOS VLSI implementation of a low-power logarithmic converter,” *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1421–1433, 2003.

[9] D. Marino, “New algorithms for the approximate evaluation in hardware of binary logarithms and elementary functions,” *IEEE Transactions on Computers*, vol. C-21, no. 12, pp. 1416–1421, 1972.

[10] T. Brubaker and J. Becker, “Multiplication using logarithms implemented with read-only memory,” *IEEE Transactions on Computers*, vol. C-24, no. 8, pp. 761–765, 1975.

[2]Division and rooting are also possible through approximate binary logarithmic conversion. However, the approximation error of these operations is not well characterized by multiplication, such that truncated error correction may not be applicable