Truncated Logarithmic Approximation

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*—The speed and levels of integration of modern devices have risen to the point that arithmetic can be performed very fast and with high precision. Precise arithmetic comes at a hidden cost—by computing results past the precision they require, systems inefficiently utilize their resources. Numerous designs over the past fifty years have demonstrated scalable efficiency by utilizing approximate logarithms. Many such designs are based off of a linear approximation algorithm developed by Mitchell. This paper evaluates a truncated form of binary logarithm as a replacement for Mitchell's algorithm. The *truncated approximate logarithm* simultaneously improves the efficiency and precision of Mitchell's approximation while remaining simple to implement.**

***Index Terms*—Truncated approximate binary logarithms, logarithm generation, anti-logarithm generation, computer arithmetic, mixed precision.**

I. Introduction

The speed and levels of integration of modern devices have risen to the point that arithmetic can be performed in hardware with high precision. While precise arithmetic is important for some applications, computing past the precision that applications require wastes power. Due to the increasing importance of power-efficient execution, such inefficiencies will become problematic for future power-constrained devices.

It has long been recognized that approximate binary logarithms can simplify costly arithmetic operations. Mitchell's algorithm [1] uses a linear approximation to estimate log conversion through leading-one detection and a series of shifts. Variants of Mitchell's linear approximation have been used to reduce the cost of multiplication, division, power, and square root [2], [3], [4]. However, Mitchell's approximation does not provide sufficient precision for many applications, and conversion to and from Mitchell's approximate logarithms still requires significant effort.

Much prior literature focuses on improving the precision of Mitchell's approximation [2], [5], [4], [6], [7]. While these designs address the limited precision of the algorithm, they typically add to its area, latency, and power consumption. This work extends Mitchell's approximation, simultaneously improving its efficiency and precision. The algorithm considered in this paper is also amenable to error-correcting circuits similar to those used in prior work, further improving the precision of approximation while maintaining superior efficiency.

Cost savings can be achieved over Mitchell's original approximation by truncating the least-significant bits of its fractional term. Such modified approximate logarithms are referred to as *truncated approximate logarithms* and are the main focus of this work. Section II describes Mitchell's approximation. The theory, cost, delay, and precision of truncated logarithms are discussed in Section III. Section IV explores how to improve the precision of truncated logarithms, both through prior as well as novel methods, and Section V evaluates truncated approximation as a replacement for Mitchell's algorithm.

II. Concepts and Background

Traditional approaches to approximate logarithmic conversion are reviewed. To perform high-level analysis of the cost and delay required by different components, this study makes use of a simple unit gate model described in Section II-A. The formation and precision of Mitchell's approximation are described in Section II-B, and its cost and delay are analyzed in Section II-C.

A. Unit Gate Model

Analyses in this paper make use of a simple 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. While the model roughly estimates the real cost for each component, it is useful for high-level analysis and does not depend strongly on any one technology generation, design tool, or cell library. 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 $[C = 2, T = 2]$
- m-input gates composed of a tree of 2-input gates

The unit gate model takes inversion and buffering to be free, both for simplicity and because synthesis tools add an element of uncertainty to the cost of these tasks. Also, circuits with low fan-outs that require minimal buffering are chosen to mitigate the impact of this simplifying assumption. The pre-charging of cell inputs is not taken into account for simplicity. Due to the complexities of circuit routing, wiring costs are not considered.

B. Mitchell's Logarithm Generation

A brief description of approximate logarithm generation follows, with an emphasis on Mitchell's approximation. The reader is referred to [1] for a more detailed explanation of Mitchell's method. Any fixed-point binary input, X, may be expressed as $X = 2^k * (1+f)$, where $0 \leq f < 1$. The value of k is referred to as the *characteristic* of X and is the position of the leading 1 in its unsigned binary representation ($0 < k < \log_2(X)$). The value f is referred to as the *fractional component* of the resultant logarithmic approximation and is formed by normalizing all non-leading bits to form a binary fraction. Given the values of k and f that uniquely

determine X, the binary logarithm of X may be expressed as $\log_2(X) = k + \log_2(1+f)$.

The values of k and f for an input X can be found using a leading-one detector, binary encoder, and shifter as shown in Figure 1a. Conversion back to binary from an approximate binary logarithm involves shifting (Figure 1b). The cost and delay of each component in Mitchell's algorithm are analyzed in the subsections that follow.

Given k and f, the complexity of determining $\log_2(X)$ is dominated by the term $\log_2(1+f)$, which itself requires the calculation of a binary logarithm. Approximate binary logarithmic approaches typically concern themselves with how to *approximate* $\log_2(1+f)$ with the highest precision and least cost. Mitchell estimates $\log_2(1+f)$ using a single straight-line approximation to the logarithm curve, $f \approx \log_2(1+f)$. Figure 2a illustrates this straight-line approximation. The notation $\log_{2\text{-}M}(X)$ is used throughout the remainder of the paper to denote Mitchell's approximation of $\log_2(X)$.

Mitchell's algorithm always underestimates the value of $\log_2(X)$ as can be seen from Figure 2. The maximum absolute error of Mitchell's approximation relative to $\log_2(X)$ is -0.08639. This error is independent of the value of k and occurs at $f = 0.4427$. The average error of Mitchell's approximation can be found through numerical integration to be -0.05731. This value is denoted by a horizontal dotted grey line in Figure 2b. The relative error of Mitchell's approximation decreases at larger values of k; its maximum over all values of k and f is -5.792% (at $k=0$, $f=0.3591$).

C. Cost of Mitchell Approximation

Mitchell's linear approximation approximates log (and anti-log) conversion using a leading-one detector, a binary encoder, and a series of shifters. The cost and delay of each component are analyzed in the following sections in order to form an analytical model for Mitchell's method.

1) Leading-One Detection: Leading-one detection can be implemented by passing the reversed, inverted input signal through a parallel AND prefix tree [8]. Approximate logarithmic conversion is likely to be on the critical path so a minimum-depth prefix graph is assumed. To provide conservative analyses later in the paper, a Kogge-Stone style prefix tree [9] is chosen for the cost model. Under the unit gate model, the cost of this prefix graph is maximal among all minimum-depth prefix graphs [10]. This ensures that the cost of leading-one detection is not underrepresented relative to the log and anti-log conversion shifters. The cost and delay of leading-one detection are given by (1) and (2).

$$C_{\text{LOD-M}}(n) = C_{\text{AND}} * \sum_{i=0}^{\log_2(n)-1} \frac{n}{2} = C_{\text{AND}} * \frac{n}{2} * \log_2(n) \quad (1)$$

$$T_{\text{LOD-M}}(n) = T_{\text{AND}} * \log_2(n) \quad (2)$$

2) One-Hot to Binary Encoder: The output of the leading-one detector is a one-hot encoded signal. In order to extract the characteristic, this signal is encoded into an unsigned binary value. One way to do this is with $\log_2(n)$ $\frac{n}{2}$-input OR gates [8]. Such an encoder is illustrated for 8 bits in Figure 3. Under the unit gate model, each $\frac{n}{2}$-input OR gate is formed out of $\left(\frac{n}{2}-1\right)$ 2-input OR gates. This leads to the cost and delay given by (3) and (4).

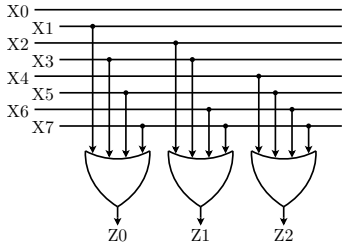


Fig. 3. An 8-bit one-hot to binary encoder.

$$C_{\text{ENC-M}}(n) = C_{\text{OR}} * log_2(n) * \left(\frac{n}{2} - 1\right) \quad (3)$$

$$T_{\text{ENC-M}}(n) = T_{\text{OR}} * (\log_2(n) - 1) \quad (4)$$

3) Approximate Logarithm Shifter: Following leading-one detection and the conversion of the characteristic into a binary number, a shifter is used to extract the $(n-1)$-bit fractional component (f) from X. Figure 4a shows a parallel prefix diagram of the 15-bit combinational shifter used for a 16-bit system. Black nodes represent two-input multiplexers, which are controlled by the (unsigned binary) shift amount. Some internal simplification occurs in the shifter due to the fact that constant 0 values are propagated into the least-significant bits of a shifted output. To account for this, grey nodes represent AND gates whose second input is connected to the inverted shift signal.

A shifter is determined by two parameters: its width, n, and its depth, d. The depth of a shifter is equivalent to the width of its binary shift amount. The number of black nodes at level i of Mitchell's $(n-1)$-bit combinational shifter is given by $b_{\text{SHIFT-M}}(i,n) = n - 2^i - 1$ such that the total number of black nodes in a shifter with depth d, $B_{\text{SHIFT-M}}(n,d)$, is given by (5). As such, the shifter used for approximate logarithm generation has $B_{\text{SHIFT-M}}(n, \log_2(n)) = n*\log_2(n) - \log_2(n) - n + 1$ black nodes.

$$B_{\text{SHIFT-M}}(n,d) = \sum_{i=0}^{d-1} b_{\text{SHIFT-M}}(i,n) = d*n - 2^d - d + 1 \quad (5)$$

The number of grey nodes at level i of Mitchell's log shifter, $g(i)$ is given by $g_{\text{SHIFT-M}}(i) = 2^i$ such that the total number of grey nodes in a shifter with depth d, $G_{\text{SHIFT-M}}(d)$, is given by (6). Accordingly, the number of grey nodes used for shifting during approximate logarithm generation is $G_{\text{SHIFT-M}}(\log_2(n)) = n-1$.

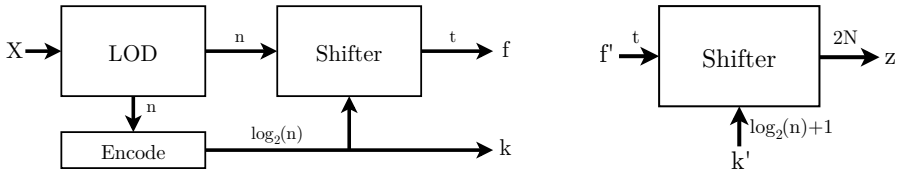
$$G_{\text{SHIFT-M}}(d) = \sum_{i=0}^{d-1} g_{\text{SHIFT-M}}(i) = 2^d - 1 \quad (6)$$

Equations (5) and (6) lead to the total cost of the approximate logarithm shifter with depth d given by (7). Therefore, the full-depth shifter used for logarithm conversion has a cost of $C_{\text{SHIFT-M}}(n, \log_2(n)) = C_{\text{MUX}}*(n*\log_2(n) - \log_2(n) - n + 1) + C_{\text{AND}}*(n-1)$ with a delay given by (8).

$$C_{\text{SHIFT-M}}(n,d) = C_{\text{MUX}} * (d*n - 2^d - d + 1) + C_{\text{AND}} * (2^d - 1) \quad (7)$$

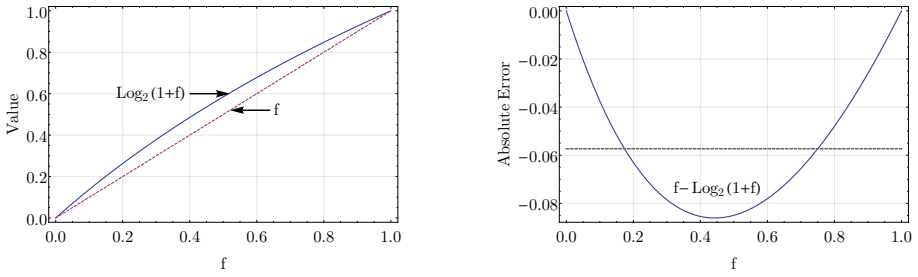
$$T_{\text{SHIFT-M}}(n) = T_{\text{MUX}} * \log_2(n) \quad (8)$$

4) Anti-Logarithm Conversion: The anti-logarithm of a Mitchell approximation is formed by setting the bit that



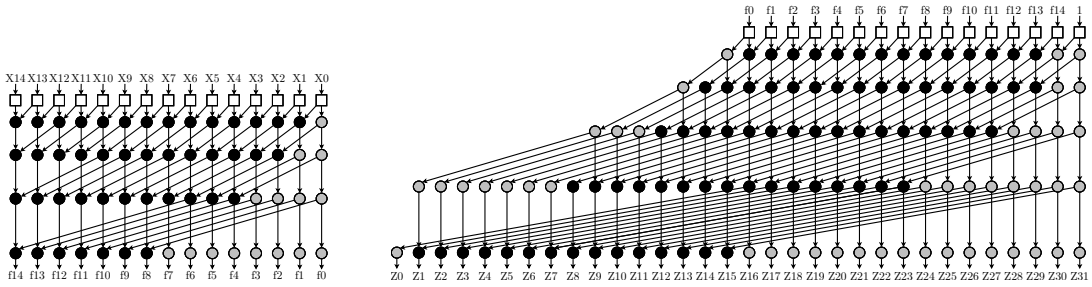
(a) Logarithm Generation (b) Anti-log Generation

Fig. 1. Approximate log and anti-log conversion. For Mitchell's algorithm, $t = n$. Truncated approximation chooses $t < n$ (see Section III). Sizes of components are not to scale with respect to their costs or delays.



(a) Values (b) Absolute Error

Fig. 2. The error in Mitchell's log approximation. The dotted grey line in (b) denotes the average absolute error.



(a) Logarithm Shifter ($n=16$) (b) Anti-Logarithm Shifter (after multiplying two 16-bit operands)

Fig. 4. Combinational shifters for approximate logarithmic and anti-logarithmic conversion.

corresponds to the characteristic (2^k) and inserting the fractional component immediately after this bit. In practice, this effect can be achieved by inserting a 1 value after the most-significant bit of the fraction and right-shifting the concatenated value to the correct position. The concatenated value needs to be shifted $(n-k-1)$ spaces to the right, a shift amount equivalent to the one's complement of k.

Figure 4b shows the shifter used to convert an approximate 16-bit multiplication back to binary[1]. Such an anti-logarithm generator is selected because approximate logarithms are often used to simplify multiplication. As before, black nodes represent two-input multiplexers. Grey nodes either represent AND gates (some with a complemented input) or OR gates; in any case, grey nodes all share the unit cost of a simple gate.

The number of black nodes at level i of the anti-log conversion shifter is given by (9), such that the total number of black nodes in the anti-log shifter with depth d, $B_{\text{ALOG-M}}(n,d)$ is given by (10). As such, the shifter used for approximate logarithm generation has $B_{\text{ALOG-M}}(n,\log_2(n)+1) = n*\log_2(n)+n-3$ black nodes.

$$b_{\text{ALOG-M}}(i,n) = \begin{cases} n-2, & \text{if } i=0 \\ n, & \text{if } 0<i<d \\ n-1, & \text{if } i=d \end{cases} \tag{9}$$

[1]The anti-log right shifter is shown as a left-shift of the reversed input.

$$B_{\text{ALOG-M}}(n,d) = \sum_{i=0}^{d-1} b_{\text{ALOG-M}}(i,n) = d*n-3 \tag{10}$$

The number of grey nodes at level i of the anti-log conversion shifter with depth d is given by (11) such that the total number of grey nodes in the anti-log shifter, $G_{\text{ALOG-M}}(d)$, is given by (12). Accordingly, the number of grey nodes used for approximate anti-logarithm shifting is $G_{\text{ALOG-M}}(\log_2(n)+1) = 3*n-\log_2(n)+1$.

$$g_{\text{ALOG-M}}(i) = \begin{cases} 3, & \text{if } i=0 \\ 2^{(i+1)}-1, & \text{if } 0<i<d \\ 2^i+1, & \text{if } i=d \end{cases} \tag{11}$$

$$\begin{aligned} G_{\text{ALOG-M}}(d) &= 3+\sum_{i=1}^{d-2} 2^{(i+1)} - 1 + 2^{(d-1)} + 1 \\ &= 2^d + 2^{(d-1)} - d + 2 \end{aligned} \tag{12}$$

Equations (10) and (12) can be combined to derive the total cost and delay of the anti-log shifter, given by (13) and (14).

$$\begin{aligned} C_{\text{ALOG-M}}(n) = {} & C_{\text{MUX}} * (n*\log_2(n)+n-3) \\ & + C_{\text{GATE}} * (3*n-\log_2(n)+1) \end{aligned} \tag{13}$$

$$T_{\text{ALOG-M}}(n) = T_{\text{MUX}} * (\log_2(n)+1) \tag{14}$$

5) Total Cost of Mitchell Approximation: It can be seen from Figure 1a that the critical path of Mitchell's log generation goes through the leading-one detector, binary encoder, and shifter such that the unit gate delay of the total circuit

is additive over these three components. This leads to the total cost given by (15) and delay given by (16) for Mitchell's approximation. Appropriate values can be substituted from the equations above and from the unit gate model.

$$C_{\text{LOG+ALOG-M}}(n) = C_{\text{LOD-M}}(n) + C_{\text{ENC-M}}(n) + C_{\text{SHIFT-M}}(n) + C_{\text{ALOG-M}}(n) \quad (15)$$

$$T_{\text{LOG+ALOG-M}}(n) = T_{\text{LOD-M}}(n) + T_{\text{ENC-M}}(n) + T_{\text{SHIFT-M}}(n) + T_{\text{ALOG-M}}(n) \quad (16)$$

The above cost and delay models are used in Section V to explore the scaling properties of Mitchell's algorithm and to investigate the cost benefits of truncated logarithm generation. Next, truncated logarithms will be described and an analytical model will be derived for the cost and delay of truncated conversion.

III. Truncated Logarithmic Approximation

The maximum absolute error in Mitchell's fraction is between 2^{-3} and 2^{-4}, yet at large input sizes many less-significant bits are retained. Truncated logarithmic approximation is based on the intuition that the size of an approximate fraction should be proportional to its precision. Rounding off the least-significant bits of approximate log and anti-log conversion reduces their costs, and can actually be manipulated to improve the average precision of the result.

The t-bit downward-rounded approximation of X, $\log_{2\text{-}T\downarrow}(X,t)$, is given by (17). Functionally, it is equivalent to Mitchell's approximation retaining only the t most-significant fractional bits.

$$\log_{2\text{-}T\downarrow}(X,t) = k + (f \bmod 2^{-t}) \approx \log_2(X) \quad (17)$$

The t-bit upward-rounded approximation of X, $\log_{2\text{-}T\uparrow}(X,t)$, is given by (18). Upward rounding may be desirable as it tends to introduce positive error into the fraction. Since Mitchell's algorithm inherently underestimates the true logarithm, positive rounding counter-biases the truncated approximation. This leads to an estimate that (for $t \geq 4$) has a maximum absolute error no worse than Mitchell's algorithm and also decreases the average error relative to Mitchell's approximation.

$$\log_{2\text{-}T\uparrow}(X,t) = k + (f \bmod 2^{-t}) + 2^{-t} \approx \log_2(X) \quad (18)$$

While upward rounding has precision advantages, it requires incrementation for proper operation ($+2^{-t}$). The cost and delay of this incrementation may be significant depending on implementation details. The increment operation is ignored in the cost and delay model below for two reasons. First, it may be possible to subsume the incrementation into the approximate functional unit. For example, a truncated approximate constant multiplier could incorporate this increment into the constant inputs of its internal adder. As such, it is difficult to derive a proper cost and delay model without considering the unit internals. Also, more precise extensions to Mitchell's algorithm often incorporate addition with a constant term; as such, the costs of this incrementation may be agglomerated with that of the error correction circuitry.

A visualization of upward and downward-rounded truncation is shown in Figure 5. It can be seen from Figure 5a that the value of $\log_{2\text{-}T\uparrow}(X,t)$ always falls between Mitchell's approximation and a shifted form of Mitchell's curve. Likewise, the value of $\log_{2\text{-}T\downarrow}(X,t)$ (Figure 5c) is bounded from above by Mitchell's approximation and from below by a shifted form of the curve. These relationships are expressed in (19) and (20). The absolute error of truncated approximation is similarly bounded by an error envelope as shown by Figures 5b and 5d. An upward-rounded approximate logarithm attains a maximum positive error at $f = 0$; this error falls linearly from the upper error envelope to that of Mitchell's estimate over each 2^{-t}-width interval. The error of a downward-rounded approximate log attains a maximum negative error at the same point as does Mitchell's approximation ($f = 0.4427$) and falls linearly from the error of Mitchell's estimate to the lower error envelope over each 2^{-t}-width interval.

$$\log_{2\text{-}M}(X) - 2^{-t} \leq \log_{2\text{-}T\downarrow}(X,t) \leq \log_{2\text{-}M}(X) \quad (19)$$

$$\log_{2\text{-}M}(X) \leq \log_{2\text{-}T\uparrow}(X,t) \leq \log_{2\text{-}M}(X) + 2^{-t} \quad (20)$$

A. Cost and Delay of Truncated Approximation

Truncated approximate logarithms use a similar conversion process to Mitchell's algorithm and can easily be incorporated in a design as an improvement over Mitchell's method. Cost benefits result from a simplification of the truncated log and anti-log shifters. Cost models for these two components are derived below.

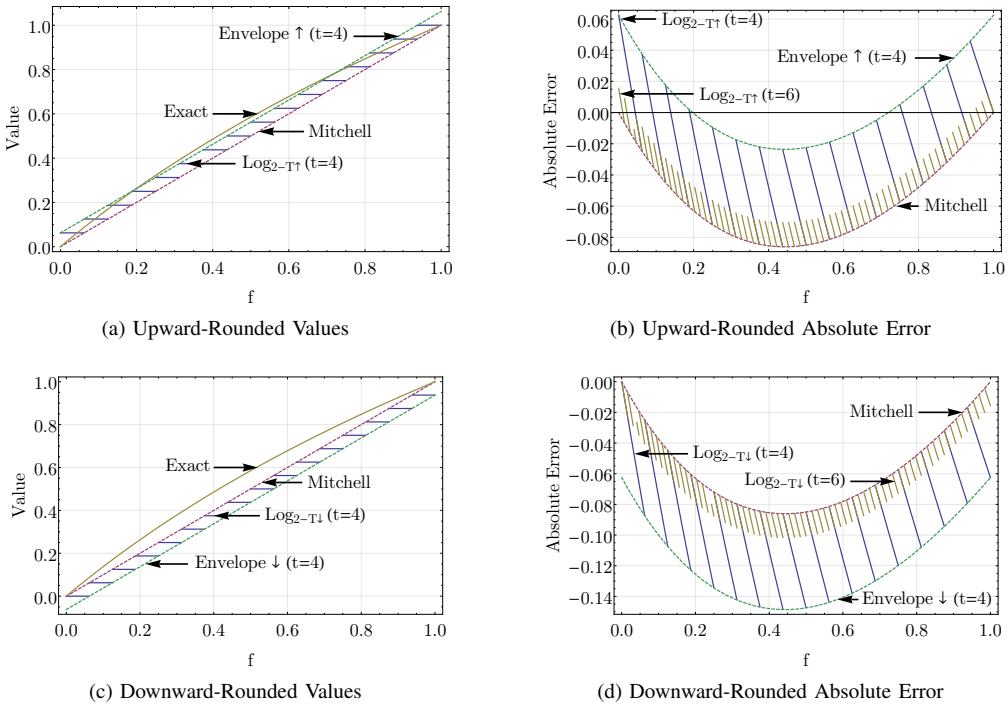
1) Truncated Logarithm Shifter: Truncation can significantly reduce the cost of the log conversion shifter. Figure 6 shows a range of 16-bit truncated approximate logarithm shifters. It can be seen that the number of both grey and black nodes are greatly reduced relative to Mitchell's approximation (Figure 4a) for small truncation widths (t). Cost and delay models for the truncated shifter follow.

The truncated logarithm shifter with depth d is equivalent to Mitchell's log shifter for the first $tb(n,t) = \lfloor \log_2(n) - \log_2\left(\frac{n}{t}\right) \rfloor$ levels. This design parameter is referred to as the *truncation boundary* for the remainder of the section and is used pervasively in the cost model. The cost of any level, i, preceding the truncation boundary ($i < tb$) is derived previously in Section II-B. Accordingly, only the cost of the truncated log shifter at levels in the interval $tb \leq i < \log_2(n)$ is derived below. This interval is referred to as the *truncation-specific region* of the shifter. Both the truncation boundary and the truncation-specific region are illustrated for a truncated shifter ($n = 16$, $t = 4$) in Figure 6c.

It is noted that there are t total nodes in the last level of the truncated log shifter and that each higher level of the truncation-specific region has two times as many nodes as the preceding one. This leads to a total of $t_{\text{SHIFT-T}}(i,n,t) = 2^{\log_2(n)-i-1} * t$ nodes (either black or grey) at level i within the truncation-specific region. Every level below the truncation boundary is composed entirely of black nodes. The truncation boundary itself consists of a number of black nodes followed by one or more grey nodes. Equation (21) expresses this relationship, where $b_{\text{SHIFT-T}}(i,n,tb,t)$ is the number of black nodes at level i in the truncation-specific region and $g_{\text{SHIFT-T}}(tb,t)$ is the number of grey nodes at the truncation boundary.

$$b_{\text{SHIFT-T}}(i,n,tb,t) = \begin{cases} t_{\text{SHIFT-T}}(i,n,t) - g_{\text{SHIFT-T}}(tb,t), & \text{if } i = tb \\ t_{\text{SHIFT-T}}(i,n,t), & \text{if } tb < i < d \end{cases} \quad (21)$$

Only the first level of the truncation-specific region (the



(a) Upward-Rounded Values

(b) Upward-Rounded Absolute Error

(c) Downward-Rounded Values

(d) Downward-Rounded Absolute Error

Fig. 5. The value and absolute error of truncated logarithmic approximation.

truncation boundary) contains any grey nodes. To determine the number of grey nodes at this level, it is noted that the *position* of the leading grey node on the truncation boundary is $lg_{\text{SHIFT-T}}(tb) = 2^{tb} - 1$. The node at this position is grey, and every existent node to the right of it is grey, as well. The lowest existing node position on the truncation boundary is referred to as the trailing node; its position is given by $tn_{\text{SHIFT-T}}(tb, t) = 2^{tb+1} - t - 1$. The number of grey nodes on the truncation boundary equals the number of nodes between the leading and trailing node, given by (22).

$$\begin{aligned} g_{\text{SHIFT-T}}(tb, t) &= lg_{\text{SHIFT-T}}(tb) - tn_{\text{SHIFT-T}}(tb, t) + 1 \\ &= t + 1 - 2^{tb} \end{aligned} \tag{22}$$

Substituting (22) into (21) and summing over all levels leads to the total number of black and grey nodes in the truncated shifter, expressed by (23) and (24), respectively. $B_{\text{SHIFT-M}}(n, d)$ and $G_{\text{SHIFT-M}}(d)$, which were derived previously for Mitchell's approximation, are used for those levels *outside* of the truncation-specific region.

$$B_{\text{SHIFT-T}}(n, tb, t) = 2^{-tb} * n * t + tb * n - tb - 2 * t \tag{23}$$

$$G_{\text{SHIFT-T}}(tb, t) = 2^{tb} - 2^{tb+1} + t + 1 \tag{24}$$

Equations (23) and (24) lead to the total cost of the truncated approximate logarithm shifter, given by (25). Truncation leaves the critical path of the approximate log shifter untouched. As such, the delay remains the same as that of Mitchell's approximation and $T_{\text{SHIFT-T}} = T_{\text{SHIFT-M}}$.

$$\begin{aligned} C_{\text{SHIFT-T}}(n, tb, t) = C_{\text{MUX}} * (2^{-tb} * n * t + tb * n - tb - 2 * t) \\ + C_{\text{AND}} * (2^{tb} - 2^{tb+1} + t + 1) \end{aligned} \tag{25}$$

2) Truncated Anti-Log Conversion: A truncated approximate logarithm is converted back to binary in the same manner as Mitchell's approximation. However, due to the truncated logarithm's shorter fraction, the complexity of the resultant anti-log shifter is greatly reduced. Figure 6d shows the shifter used to convert the multiplication of two truncated approximate logarithms ($t = 4$) back to binary. When compared to Mitchell's method (Figure 4b), it is apparent that the truncated anti-log shifter has fewer total nodes and many fewer black nodes. To illustrate how the cost of truncated anti-log conversion changes with the truncation width, Figure 6b shows a smaller and less precise truncated anti-log shifter ($t = 2$).

During anti-logarithmic conversion, black nodes represent two-input multiplexers and grey nodes represent simple gates (either AND or OR gates). The number of black nodes at level i of the truncated anti-log shifter is given by (26) such that the total number of black nodes in the anti-log shifter with depth d, $B_{\text{ALOG-T}}(n, d, t)$, is given by (27). As such, the shifter used for approximate logarithm generation has $B_{\text{ALOG-T}}(n, \log_2(n) + 1, t) = t * log_2(n) + t - 1$ black nodes.

$$b_{\text{ALOG-T}}(i, t) = \begin{cases} t - 1, & \text{if } i = 0 \\ t, & \text{if } i > 0 \end{cases} \tag{26}$$

$$B_{\text{ALOG-T}}(n, d, t) = \sum_{i=0}^{d-1} b_{\text{ALOG-T}}(i, t) = t * d - 1 \tag{27}$$

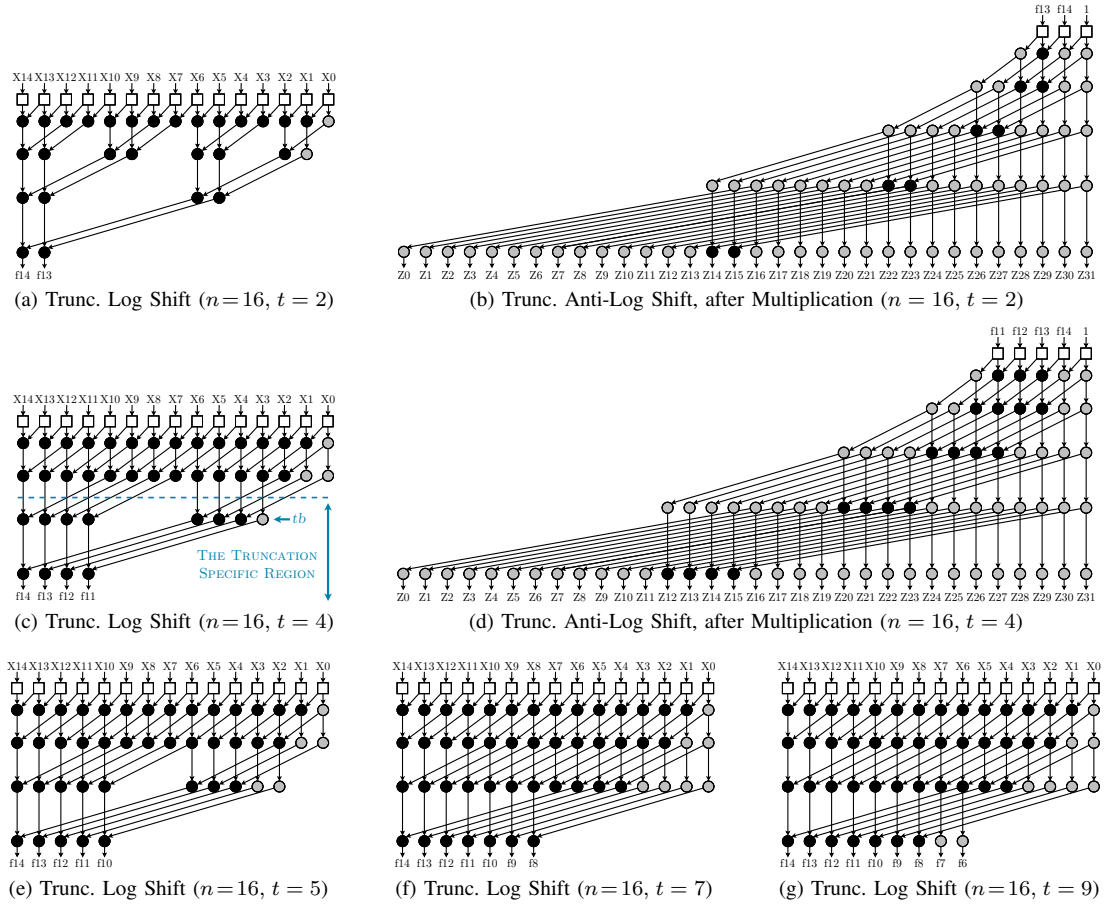
The number of grey nodes at level i of a truncated anti-log shifter with depth d is given by (28); summation produces the total number of grey nodes in the shifter, given by (29).

$$g_{\text{ALOG-T}}(i, d, t) = \begin{cases} 3, & \text{if } i = 0 \\ 2^{(i+1)}, & \text{if } 0 < i < d \\ 2^i - t, & \text{if } i = d \end{cases} \tag{28}$$

$$G_{\text{ALOG-T}}(n, t) = 4 * n - t - 1 \tag{29}$$

Equations (27) and (29) lead to the total cost of the truncated approximate anti-logarithm shifter, given by (30). Truncation does not change the critical path of the anti-logarithm circuit such that $T_{\text{ALOG-T}} = T_{\text{ALOG-M}}$.

$$\begin{aligned} C_{\text{ALOG-T}}(n, t) = C_{\text{MUX}} * (t * log_2(n) + t - 1) \\ + C_{\text{GATE}} * (4 * n - t - 1) \end{aligned} \tag{30}$$



(a) Trunc. Log Shift ($n=16$, $t=2$)

(b) Trunc. Anti-Log Shift, after Multiplication ($n=16$, $t=2$)

(c) Trunc. Log Shift ($n=16$, $t=4$)

(d) Trunc. Anti-Log Shift, after Multiplication ($n=16$, $t=4$)

(e) Trunc. Log Shift ($n=16$, $t=5$)

(f) Trunc. Log Shift ($n=16$, $t=7$)

(g) Trunc. Log Shift ($n=16$, $t=9$)

Fig. 6. The impact of truncation on the shifters used for approximate logarithmic and anti-logarithmic conversion.

3) Total Cost of Truncated Log Approximation: Truncated logarithmic approximation lessens the cost of Mitchell's log and anti-log conversion shifters; the other components remain the same, as does the critical-path delay. This results in the total cost and delay for a t-bit ($t < n$) truncated logarithmic approximation given by (31) and (32). Models are taken from the cost analysis of Mitchell's approximation where appropriate. Values may be computed from the cost and delay models above, using the unit gate model.

$$\begin{aligned} C_{\text{LOG+ALOG-T}}(n,t) = {} & C_{\text{LOD-M}}(n) + C_{\text{ENC-M}}(n) \\ & + C_{\text{SHIFT-T}}(n, tb(n,t)) \\ & + C_{\text{ALOG-T}}(n,t) \end{aligned} \tag{31}$$

$$\begin{aligned} T_{\text{LOG+ALOG-T}}(n) = {} & T_{\text{LOD-M}}(n) + T_{\text{ENC-M}}(n) \\ & + T_{\text{SHIFT-M}}(n) + T_{\text{ALOG-M}}(n) \end{aligned} \tag{32}$$

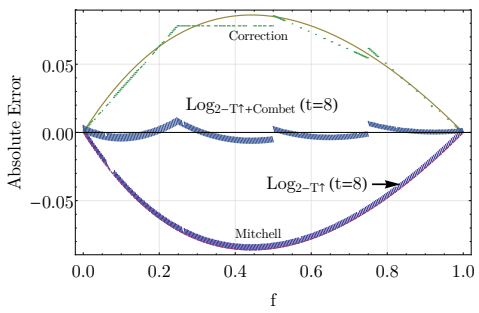
IV. Error Behavior and Correction

Because of its simplicity, Mitchell's approximation is commonly used as the basis for more complex and precise approximate logarithm generators. A popular approach is to apply piecewise linear schemes to improve upon Mitchell's simple linear interpolation [2], [5], [4], [6], [7], with increased implementation costs. It is demonstrated that truncated approximation is a viable replacement strategy for Mitchell's algorithm in such designs by showing that prior error correction schemes effectively correct truncated logarithms. In addition, it is demonstrated that there exist *truncation-specific* error correction circuits that improve the precision of truncated logarithms while adding little cost or delay.

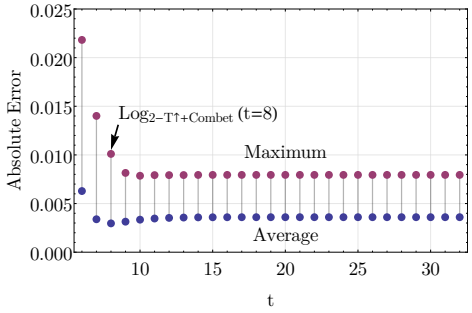
Truncated approximate logarithms are amenable to error correction techniques that were originally applied to Mitchell's approximation. This reinforces the ability of truncated logs to improve a Mitchell-based design without drastic changes to the underlying circuitry. To demonstrate the application of prior error correction techniques to truncated logs, the well-known Mitchell-based correction applied by Combet et al. [5] is applied to truncated logarithms. Figure 7a shows the precision of the resultant circuit, and Figure 7b shows the maximum and average absolute error at various truncation widths. The precision benefits of increasing truncation widths quickly diminish past the effective precision of the error correction scheme. Combet's scheme achieves 6 to 8 bits of precision; choices of t past this point quickly lose utility.

It was noted previously that an upward-rounded approximate logarithm with $t = 4$ is more precise than Mitchell's algorithm. A simple error correction scheme for this design replicates the t_{ec} most-significant bits of the truncated and rounded fractional component, appending them to form a modified $(t+t_{ec})$-bit fraction. Figure 7c shows that the resultant absolute errors for this circuit are less than those of Mitchell's approximation or the upward-rounded truncated logarithm. An exploration of the t_{ec} design space for this circuit is shown in Figure 7d. Any design with $t_{ec} > 2$ improves upon the maximum error of Mitchell's algorithm, and larger t_{ec} values lead to further precision improvements.

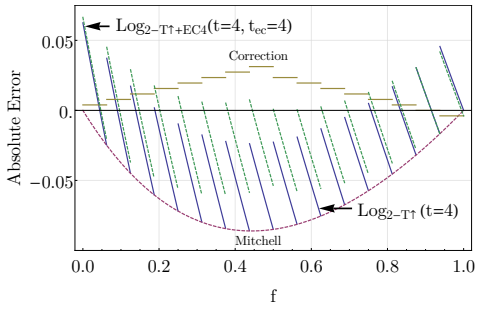
The preceding error correction circuit incurs minimal overhead because the truncated logarithm's short fraction makes addition of the correction term with the fraction unnecessary—



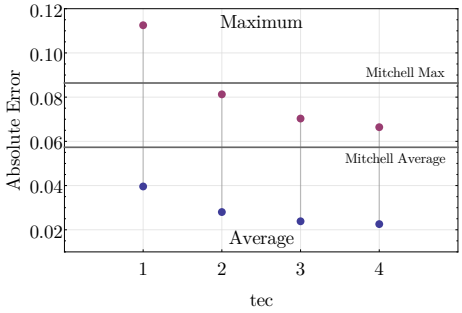
(a) Truncated Combet Error Correction



(b) Truncated Combet t Exploration



(c) $\log_{2\text{-}T\downarrow+EC4}$ ($t_{ec}=4$)



(d) t_{ec} Design Space Exploration ($t=4$)

Fig. 7. Error correction applied to truncated logarithms.

TABLE I
PRECISION OF TRUNCATED MITCHELL ANALOGUES.

Technique	Min. Error	Max. Error	Max. Abs. Error	Average Abs. Error
Mitchell	-0.08639	0	0.08639	0.05731
$log_{2-T\uparrow}(t=4)$	-0.08639	0.06250	0.08639	0.03457
$\log_{2-T\uparrow+EC4}(t_{ec}=2)$	-0.08126	0.07813	0.08126	0.02802
$\log_{2-T\uparrow+EC4}(t_{ec}=3)$	-0.06564	0.07031	0.07031	0.02384
$\log_{2-T\uparrow+EC4}(t_{ec}=4)$	-0.06089	0.06641	0.06089	0.02258

the two values need only be concatenated. In total, the correction term requires t_{ec} XOR gates (for selective complementation) and it increases the length of the truncated fraction passed to the anti-log shifter. The same error correction circuit would not be cost efficient for Mitchell's approximation—due to the full length of Mitchell's fraction, addition is necessary for all useful correction terms. This demonstrates that there are efficient error-correcting schemes that are specific to truncated approximate logarithms. However, a full exploration of such specialized error-correcting schemes is left for future work.

V. Cost Analysis of a Truncated Mitchell Replacement

The viability and cost savings of replacing Mitchell's approximation with a truncated logarithm are considered. Table I shows the precision of different truncated log designs with worst and average precision equal to or better than Mitchell's approximation. Truncated logarithms with simple truncation-specific error correction provide superior precision. Analyses below demonstrate that these designs also offer significant cost savings over Mitchell's algorithm.

Figure 8c shows the superior scalability of a truncated logarithm over Mitchell's approximation. The relative cost savings of truncation (at a fixed t) grow with the input length and range from -34% (at $n=16$) to -52% (at $n=128$). Figure 8b shows the added cost required for error correction during truncated approximate logarithmic conversion. The costs are negligible at large input sizes and represent at most a 8.765% increase over $\log_{2-T\updownarrow}$ (at $n=16, t=4$).

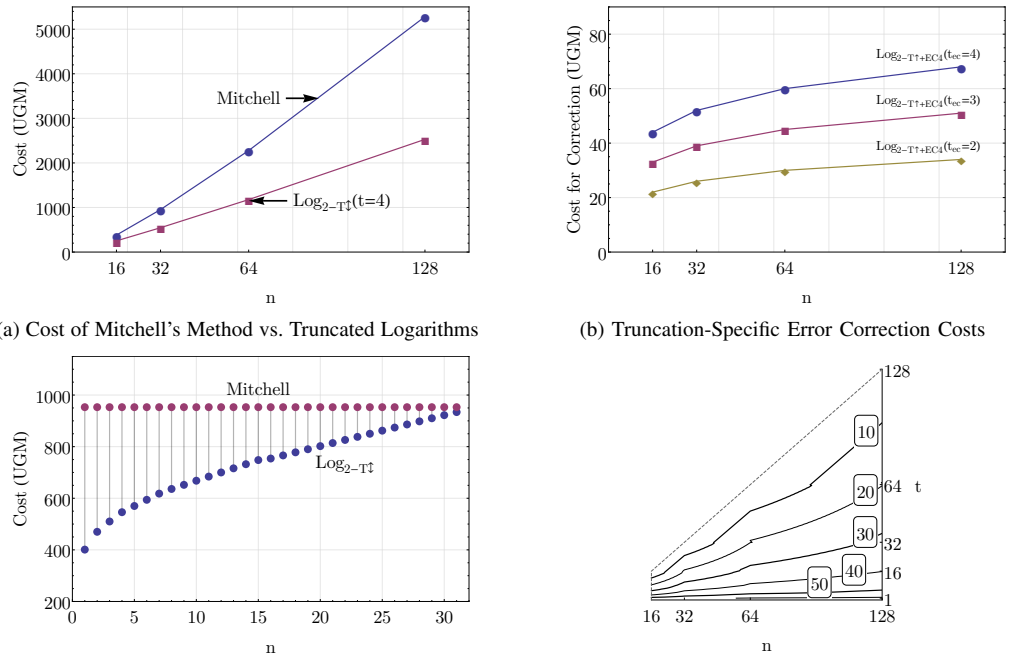
The delay of Mitchell's approximation and truncated logarithmic conversion are equal; both are given by Equation (16). This delay increases logarithmically with increasing input length. The error correction circuitry used for $\log_{2\text{-}T\uparrow+EC4}$ adds a constant T_{XOR} delay[2]. The impact of this extra XOR gate delay is relatively minor—it incurs an 8% increase in time at 16-bits, 6.452% at 32-bits, 5.405% at 64-bits, and 4.651% at 128-bits.

The truncated log with $t=4$ is an appropriate replacement for Mitchell's algorithm. However, it was shown earlier (in Section IV) that a larger truncation width is appropriate once stronger error correction is applied. The cost savings of truncated logarithms across a range of truncation widths are explored in Figure 8c. Truncated log generation has large cost savings over the 32-bit Mitchell approximation, especially at small to moderate truncation widths. The cost of a truncated log generator increases approximately linearly past $t=4$. Similar trends are found across different input widths—Figure 8c shows a contour plot of the percent cost savings due to truncation across different values of n and t. It can be seen that truncated logarithms have a significant cost advantage relative to Mitchell's algorithm, especially at small truncation widths or large input sizes. Past $t=4$ to $t=8$, the cost of truncated log conversion increases approximately linearly until converging with Mitchell's algorithm at $t=n$.

VI. Future Research

This paper presents an initial description and analysis of truncated approximate logarithms. There are a number of enticing areas of future exploration related to this research.

[2]In the log/anti-log; approximate unit internals are not considered.



(a) Cost of Mitchell's Method vs. Truncated Logarithms

(b) Truncation-Specific Error Correction Costs

(c) Cost of Approximate Conversion, Across t ($n = 32$)

(d) % Cost Savings Due to Truncation, Across n and t

Fig. 8. The relative costs of truncated Mitchell analogues.

This paper compares against Mitchell's algorithm and demonstrates the use of truncation in piecewise linear-error correction schemes. In addition to these linear methods, prior work has sought to form approximate logarithms through table-based approximations [11], [12], higher order interpolation [13], and hybrid methods using both combinational and table-based correction [14], [15]. The extension of truncation to these domains may provide useful insight and could lead to designs with superior precision proportionality.

Truncated logarithms not only reduce the cost of log and anti-log approximation, but also simplify operations on the numbers in approximate logarithmic form. This effect is not considered or explored in this paper. Cost and error analysis of common approximate operations (such as fixed-point multiplication and division) is an obvious extension of this work.

VII. Conclusion

Truncated approximate logarithms are a modification of Mitchell's algorithm for computing the approximate binary logarithm. Truncated approximate logarithms decrease the cost and improve the precision of Mitchell's method, and can serve as a drop-in replacement for circuits utilizing Mitchell's approximation, easing their adoption. In addition, it is shown that truncated approximate logarithms are amenable to both existing and truncation-specific error correction techniques. Combining approximate logarithms with error correction could lead to a family of designs that demonstrate superior precision proportionality across a range of precisions.

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. The authors would like to thank Mehmet Başoğlu for helping to prepare this paper for publication.

References

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

[2] 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.

[3] H.-Y. Lo, "Binary logarithms for computing integral and non-integral roots and powers," *International Journal of Electronics*, vol. 40, no. 4, pp. 357–364, 1976.

[4] Y.-H. Lee, Y.-S. Cho, and S. Moon, "Design of a high precision logarithmic converter in a binary floating point divider," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 4, pp. 342–353, 2012.

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

[6] S. SanGregory, C. Brothers, D. Gallagher, and R. Siferd, "A fast, low-power logarithm approximation with CMOS VLSI implementation," in *Midwest Symposium on Circuits and Systems*, vol. 1, 1999, pp. 388–391.

[7] 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.

[8] R. Zimmermann, "VHDL library of arithmetic units," in *The International Forum on Design Languages*, 1998, pp. 267–272. [Online]. Available: http://www.iis.ee.ethz.ch/~zimmi/arith_lib

[9] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

[10] S. Knowles, "A family of adders," in *Proceedings of the IEEE Symposium on Computer Arithmetic*, 2001, pp. 277–281.

[11] 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.

[12] M. Arnold, T. Bailey, and J. Cowles, "Error analysis of the Kmetz/Maenner algorithm," *The Journal of VLSI Signal Processing*, vol. 33, pp. 37–53, 2003.

[13] 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, dec. 1972.

[14] "A linear approximation based hybrid approach for binary logarithmic conversion," *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 353–361, 2002.

[15] D. Mclaren, "Improved Mitchell-based logarithmic multiplier for low-power DSP applications," in *Proceedings of the International Systems-on-Chip Conference (SoC)*, 2003.