IET Computers and Digital Techniques

Research Article

Fast and Low-Power Leading-One Detectors for Energy-Efficient Logarithmic Computing

Mohammad Saeed Ansari, Shyama Gandhi, Bruce F. Cockburn, Jie Han

Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada.

✉ *{ansari2, smgandhi, cockburn, jhan8} @ ualberta.ca*

This work was supported financially by the Natural Sciences and Engineering Research Council of Canada (NSERC) (Project No. RES0018685 and RES0025211).

Abstract: The logarithmic number system (LNS) can be used to simplify the computation of arithmetic functions, such as multiplication. This article proposes three leading-one detectors (LODs) to speed up the binary logarithm calculation in the LNS. The first LOD (LOD I) uses a single fixed value to approximate the d least significant bits (LSBs) in the outputs of the LOD. The second design (LOD II) partitions the d LSBs into smaller fields and uses a multiplexer to select the closest approximation to the exact value. These two LODs help with error cancellation as they introduce signed errors for inputs $N < 2^d$. Additionally, a scaling scheme is proposed that scales up the input $N < 2^d$ to avoid large approximation errors. Finally, an improved exact LOD (LOD III) is proposed that only passes half of the input N to the LOD: the more significant half is passed if there is at least one '1' in that half; otherwise, the less significant half. Our simulation results show that the 32-bit LOD III can be up to 2.8× more energy-efficient than existing designs in the literature. The Mitchell logarithmic multiplier and a neural network are considered to further illustrate the practicality of the proposed designs.

1 Introduction

Data representation has a significant impact on the performance of arithmetic circuits [1, 2]. Using the logarithmic representation simplifies several important arithmetic operations, such as multiplication, division, roots and powers. In the logarithmic number system (LNS), these operations are converted to shift and addition/subtraction operations. Therefore, they can be implemented with less hardware complexity than conventional designs [3, 4].

Errors are inevitable while computing in an LNS due to: (1) the limited bit precision and (2) the approximation error when computing the logarithmic function [5, 6]. Hence, logarithmic arithmetic circuits could be used in error-tolerant applications for which there is a range of acceptable, good-enough results rather than a uniquely required result [7, 8], such as multimedia processing and machine learning. The main justifications for approximate arithmetic in these applications are human perceptual limitations and inherently noisy inputs [9–11]. Whenever it is necessary, piece-wise linear approximations over a finely subdivided input domain or iterative techniques can be used to compensate for the accuracy loss when computing $\log(x)$ at the cost of additional hardware, power, and latency [12, 13].

In an LNS, the binary logarithm of the input operand is computed using a leading-one detector (LOD), as phase 1. Then the required operations (shift and/or addition/subtraction) are performed during phase 2. Finally, the antilogarithm of the result is computed in phase 3. The two main techniques for designing a $2n$-bit LOD are: (1) Partition the $2n$ bits into groups of 4 bits and evaluate each group using a 4-bit LOD [12, 14]. Logical OR gates and 4-bit LODs are then used to select the outputs of the LODs [14]. (2) One continuous block evaluates the $2n$ input bits from the most significant towards the least significant bits. Once the most significant '1' is found, the remaining bits are cleared to zeros for a one-hot encoded output.

The first method, as described in the last paragraph, to design $2n$-bit LOD is much faster than the second one (according to our simulations, by about 3×) and, consequently, is a better approach for designing LODs. Note that the LNS is used to speed up multiplications and so faster designs are preferable.

According to our simulation results, phase 1 is the most hardware-inefficient phase in the Mitchell logarithmic multiplier, consuming the most hardware resources (i.e. more than 49% of the area), while being the most energy-inefficient (more than 54% of the entire energy) compared to the other two phases. Hence, it is important to reduce the complexity of this phase by designing more efficient components. The common element in phase 1 in all arithmetic units, which plays a significant role in the performance of the system, is the LOD [14, 15]. This article proposes two approximate LOD designs. We studied 32-bit LOD designs using 4-bit LOD slices and found that using approximation for the least significant LODs can significantly reduce the hardware cost while preserving the accuracy of a well-known logarithmic multiplier, the Mitchell multiplier [16]. Both designs approximate or use 16 least significant bits (LSBs) of the inputs to eliminate the four least significant 4-bit LODs

LOD I approximates the 16 LSBs with a constant bias, while LOD II approximates the 16 LSBs to one of four constant biases, which is selected using the output from a simple OR operation on the input bits of the four least significant LODs. In this way, LOD II partitions the 16 LSBs into four 4-bit fields and increases the accuracy by using a multiplexer that selects the closest approximation to the accurate LOD output.

The optimal number of bits to be approximated (i.e., 16) was obtained using the relative mean error distance (MRED) error metric on the Mitchell multiplier after considering 10 million randomly generated input combinations. The MRED was evaluated for different numbers of approximation bits and 16 appeared to be the most hardware-efficient choice that also preserves the accuracy of the resulting conventional Mitchell multiplier. Moreover, the sixteen LSBs of the adder that sums the base-2 logarithm of the input operands are also approximated with alternating ones and zeros for further hardware savings. Due to the use of approximation in the 16 LSBs, the new designs increase the signed errors for inputs of less than 2^{16}, which is a relatively small range (i.e. 0.001%) of the 32-bit input domain. But for larger inputs the output result is equal to that of the exact LOD, implying an accurate design. Thus a scaling scheme is proposed that scales up the inputs when desirable, i.e. for inputs $N < 2^{16}$, to avoid large errors and to match the accuracy of the exact LOD for the entire range of input numbers. Several scaling factors are considered for each design and the effects of scaling on the hardware costs are studied.

Moreover, the proposed approximate LOD designs I and II are applied in the conventional Mitchell logarithmic multiplier, which is then used to replace the exact multiplier in a neural network (NN) to analyze the effects of approximations in an error-tolerant application. A multilayer perceptron (MLP) was used to classify the Mixed National Institute of Standards and Technology (MNIST) dataset [17]. Our simulation results show that the proposed designs preserve the classification accuracy, while reducing the hardware cost.

Further, a novel design, LOD III, is proposed that uses an exact 16-bit LOD to find the position of the leading-one in a 32-bit input. Note that unlike LOD I and LOD II, LOD III is not an approximate design and is as accurate as the conventional 32-bit LOD. If the most significant '1' happens to be among the 16 MSBs, only the 16 MSBs are passed into the LOD; otherwise, the 16 LSBs are passed to the LOD. This significantly reduces the hardware complexity and speeds up the multiplication.

Some of our preliminary work was published in [18]. In this article, the two approximate LOD designs in [18] are modified to improve their accuracy. Specifically, an up-scaling technique is added to both designs and we show how the performance of each design changes with different scaling factors, ranging from 2 to 14 with a step size of 2. Additionally, a novel design (LOD III) is proposed to use only a 16-bit LOD instead of the conventional 32-bit LOD for a 32-bit input. Despite halving the LOD width, the proposed LOD III is still a fully-accurate design. Note that LOD III can be seen as the logical progression of scaling applied to approximate LOD designs. In fact, we realized that for a scaling factor of 16, we do not need to approximate the lower half of the input and the resulting LOD III design can be made exact.

The rest of this article is organized as follows: Section 2 reviews the conventional approximate Mitchell multiplier and 4-bit LOD designs. Section 3 describes the three proposed LODs. Section 4 presents the accuracy and hardware analysis for the proposed designs. The Mitchell logarithmic multiplier and an MLP for classifying the MNIST dataset are considered in Section 5 as applications for the proposed LODs. Finally, the main conclusions are given in Section 6.

2 Background

2.1 *Conventional 4-bit LODs*

A common strategy for building larger LODs, such as 16-bit and 32-bit designs, is to use 4-bit LOD slices [14]. A 4-bit LOD, as shown in Fig. 1, produces a one-hot 4-bit output with one bit set to '1' at the position of the leading-one and the remaining bits cleared to '0'. The

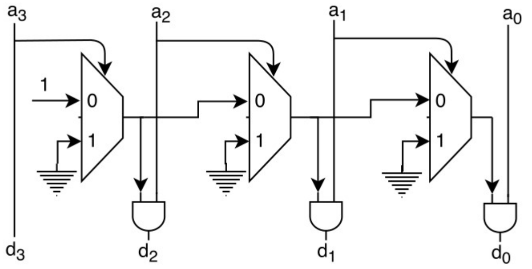


Fig. 1: A 4-bit leading-one detector [14].

input to this LOD is a 4-bit vector a and the output is a 4-bit vector d. Multiplexers are used to find the position of the leading-one. The LOD circuit evaluates in a ripple fashion from the most significant bit (MSB) to the LSB. One can verify that the worst-case delay of the circuit corresponds to the 4-bit input binary word "0001".

In another 4-bit LOD design, the multiplexers are replaced with 2-input logic gates, i.e. AND, OR, and NOT, to improve the performance by reducing the delay [19]. The higher-level 4-bit LOD as in Fig. 1 is used as our main building block for larger LOD designs.

2.2 *The Conventional Mitchell Multiplier*

Let A and B be the two positive integer inputs to the multiplier. There exist positive integers, k_1 and k_2, for which A and B can be factored as [16]:

$$A = 2^{k_1}(1 + m_1),\ 0 \le m_1 < 1. \tag{1}$$

$$B = 2^{k_2}(1 + m_2),\ 0 \le m_2 < 1. \tag{2}$$

The leading-one positions for A and B and their corresponding mantissas are denoted by k_1 and k_2 and m_1 and m_2, respectively [16]. Since A and B are unsigned integers, m_1 and m_2 can be considered to be given by the bits to the right of the leading-one for both inputs, i.e. bit positions $k_1 - 1$ down to 0 for A and, similarly, bit positions $k_2 - 1$ down to 0 for B.

Mitchell proposed approximating $\log_2(1 + x)$ with x. Following this approximation, the base-2 logarithm of the two inputs A and B can be calculated as:

$$\log_2 A \approx k_1 + m_1. \tag{3}$$

$$\log_2 B \approx k_2 + m_2. \tag{4}$$

Since $\log_2(A \times B) = \log_2 A + \log_2 B$, the binary logarithm of the product $A \times B$ is approximated by adding (3) and (4):

$$\log_2(A \times B) \approx k + m, \tag{5}$$

where $k = k_1 + k_2$ and $m = m_1 + m_2$. The approximated product is then obtained by taking the anti-logarithm of (5). To do so, the position of the most significant '1' is determined by the value of k and then the mantissa m is given by the bits appended to the right of the most significant '1'. Note that the carry out from the sum $m_1 + m_2$ of the mantissa fields is used as the carry-in to determine the position of leading one vector obtained by addition of k_1 and k_2. Hence, the Mitchell logarithmic algorithm can be represented as:

$$A \times B \approx \begin{cases} 2^k(1 + m), & m < 1, \\ \\ 2^{k+1}m, & 1 \le m < 2. \end{cases} \tag{6}$$

Three cases can occur based on the position of the most significant '1': (1) There are not enough output bits to store all of the mantissa bits. Hence, some less significant bits of m must be discarded. (2) The mantissa bits exactly fit into the available bits. (3) The number of output bits is more than the width of m. The excess bits are filled with zeros.

Fig. 2 shows an illustrative example of an 8-bit Mitchell multiplication with inputs 40 and 139. Following the above discussion, $k_1 = 5$, $k_2 = 7$, $m_1 = 01000$, and $m_2 = 0001011$ in Fig. 2. Note that two zeros are appended to mantissa m_1 to make its width equal to that of m_2. The next step is concatenating k_1 with m_1 and k_2 with m_2 to obtain the approximate base-2 logarithm of each of the input operands. The approximate logarithms are then added to produce k (shown in dark grey) and m (shown in light grey) as specified in (5). The approximated product is then '1' at bit position 12, followed by the mantissa $m = 0101011$, and five zeros of padding.

2.3 *MLP NN and the MNIST dataset*

MLPs are a class of feed-forward NNs (i.e., NNs wherein the connections between neurons do not form a cycle) that consist of an input layer, an arbitrary number of hidden layers, and an output layer. They are trained using the back-propagation supervised learning technique. Fig. 3 shows a simple MLP with n, k, and m neurons in the input, hidden, and output layers, respectively.

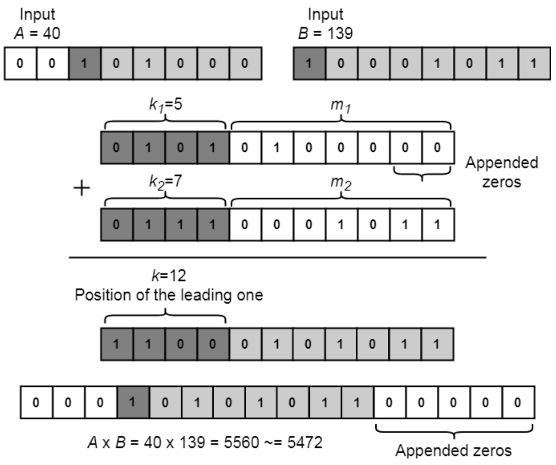


Fig. 2: Example of Mitchell approximate multiplication.

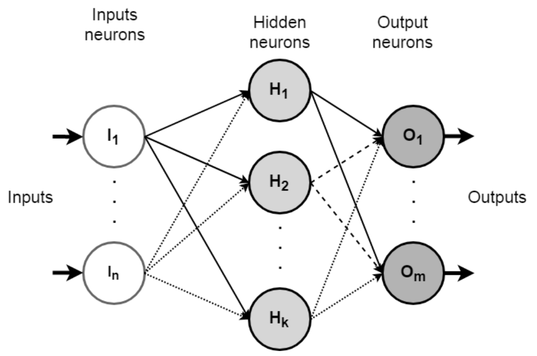


Fig. 3: Structure of a feed-forward NN.

MNIST is a dataset of handwritten numbers that consists of a training set of 60,000 and a test set of 10,000 28×28 images and their labels [17]. We used an MLP network with 784 input neurons (one for each pixel of the monochrome image), 300 neurons in the hidden layer and 10 output neurons. The outputs are interpreted to be the probability of classification of an input image into the 10 target classes for the digits 0 to 9.

3 Proposed LOD Designs

The exact 32-bit LOD can be implemented using 4-bit LOD circuits, as shown in Fig. 4. The 32-bit input is partitioned into 8 fields of 4-bits each that are evaluated by the first stage of eight 4-bit LODs operating in parallel. The two 4-bit LODs in the second stage determine the leading-one in the two 16-bit fields d_{31} to d_{16} and d_{15} to d_0. Finally, the 2-bit LOD in the third stage determines whether the leading-one lies in d_{31} to d_{16}, or in d_{15} to d_0.

The output of the 2-bit LOD in the third stage (i.e., "00","01" or "10") provides the select bits for the two groups of 4-bit multiplexers. The output of the 4-bit LOD in the second stage is passed through if the select line is '1' or else "0000" will be the output of the multiplexers. The combined 8-bit output from the multiplexers forms the select inputs for the eight groups of 4-bit multiplexers in the last stage. Only one bit from the 8-bit select word will be '1' and the multiplexer will pass the 4-bit decoded word obtained from that first-stage 4-bit LOD. The final 32-bit output has only one bit set to '1' (at the leading one position) with the remaining bits cleared to '0'.

3.1 LOD design I

This design approximates the 16 LSBs of the LOD with a single fixed bias. If the input to the LOD is greater than 2^{16}, there will be no error in calculating the position of the leading one. Otherwise, inputs of less than 2^{16} will often lead to errors in the leading-one position. Simulation was performed to determine the maximum number of bits that can be approximated in the LOD without causing any loss of accuracy in the final Mitchell product. Based on the accuracy evaluation results presented in the next section, we found that the four least significant 4-bit LODs in Fig. 4 can be safely approximated with a hexadecimal constant bias of "0400", see Fig. 5.

With this bias, sometimes the obtained approximated product will be higher and sometimes lower than the Mitchell product. This feature lets us benefit from a double-sided error distribution and, consequently, error cancellation, which can be helpful in iterative and repetitive applications. Since the four least significant 4-bit LODs in the first stage have been eliminated (due to a known fixed bias), the least significant 4-bit LOD in the second stage that is shown as "LOD A" in Fig. 4 and the OR-gates are also removed to further simplify the structure.

3.2 LOD design II

In LOD I, fixing the 16 LSBs to a constant value for some input pairs can cause a relatively large error. The second LOD design proposed in this subsection is not as hardware-efficient as LOD I but it can increase the accuracy. LOD II approximates the outputs of the four least significant 4-bit LODs to one of the four constant biases 0x"0004", 0x"0040", 0x"0400" and 0x"4000" based on the input bits to these four LODs. The biases were chosen based on the priority of the OR gates, with OR4 having the highest and OR1 having the lowest priorities, as shown in Fig. 6. Table 1 clearly shows how the priority of the OR-gates affect the 16-bit bias selection, where $O4$, $O3$, $O2$, and $O1$ represent the output of each OR gate, as shown in Fig. 6.

Table 1 Bias selection for approximate LOD II.

O4	O3	O2	O1	Bias for LOD approximation
1	X	X	X	0x"4000"
0	1	X	X	0x"0400"
0	0	1	X	0x"0040"
0	0	0	1	0x"0004"

Note that once the one 4-bit segment with a nonzero output is found, the output of the LOD in that segment is approximated by the bias "0100". When the segment is selected, three cases may occur: (1) the outputs of the exact LOD for some inputs (4 out of 16) are less than this bias, (2) the outputs of the exact LOD for some inputs (8 out of 16) are greater than this bias, and (3) the outputs of the exact LOD for some inputs (4 out of 16) equal this bias. Table 2 shows the three cases within a 4-bit segment. This selection helps with the double-sided error distribution of the proposed LOD.

Table 2 The three error cases with the bias for approximate LOD II.

4-bit input	Exact LOD output	Fixed bias	Error
0000	0000	0100	High
0001	0001	0100	High
0010 – 0011	0010	0100	High
0100 – 0111	0100	0100	Exact
1000 – 1111	1000	0100	Low

3.3 Scaling Technique

As explained above, when the inputs to the multiplier are less than 2^{16}, the approximation error could be relatively large. Thus a scaling technique is proposed to overcome this issue.

A scaling factor s, where $s \in \{2, 4, 6, ..., 12, 14\}$ is selected and used to scale up the small input operands. The two multiplier inputs are each compared with the threshold 2^{31-s}. If either of them is less than or equal to the threshold, that input is shifted to the left by s

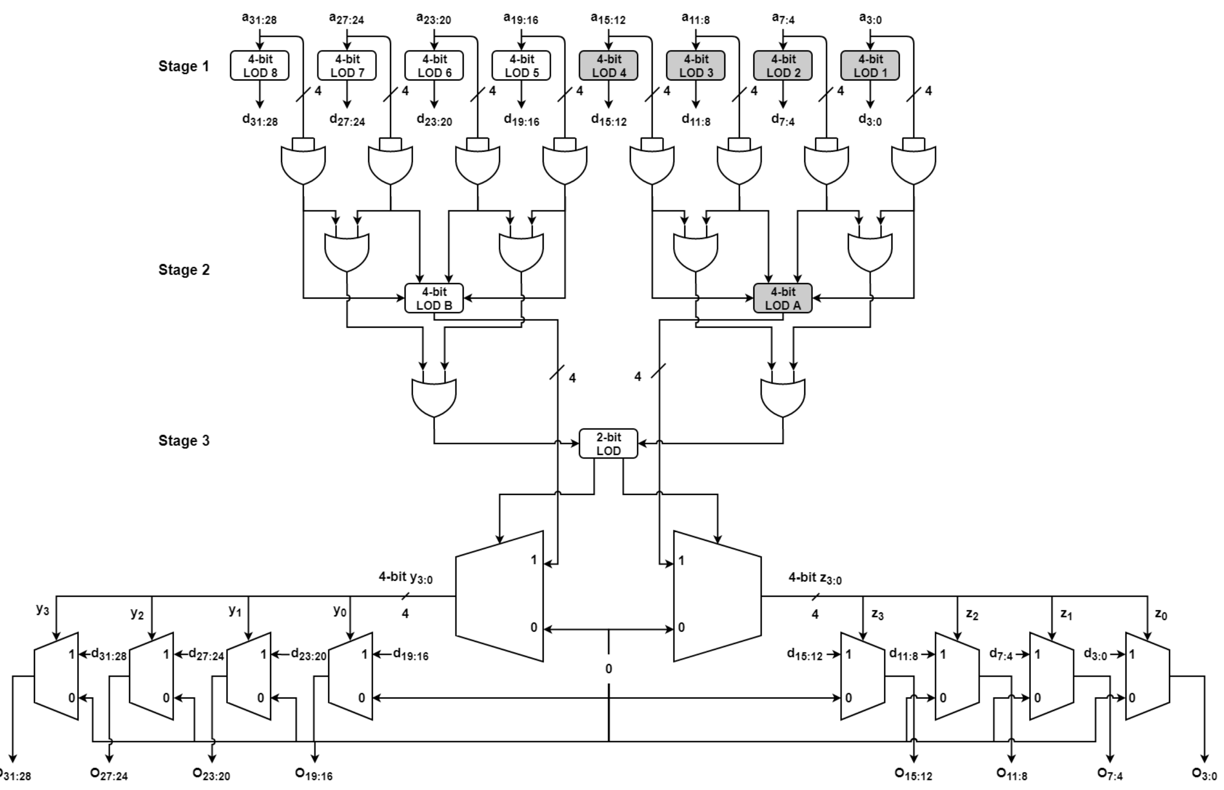

Stage 1
Stage 2
Stage 3
4-bit LOD 8
4-bit LOD 7
4-bit LOD 6
4-bit LOD 5
4-bit LOD 4
4-bit LOD 3
4-bit LOD 2
4-bit LOD 1
4-bit LOD B
4-bit LOD A
2-bit LOD
4-bit $y_{3:0}$
4-bit $z_{3:0}$
$a_{31:28}$ $a_{27:24}$ $a_{23:20}$ $a_{19:16}$ $a_{15:12}$ $a_{11:8}$ $a_{7:4}$ $a_{3:0}$
$d_{31:28}$ $d_{27:24}$ $d_{23:20}$ $d_{19:16}$ $d_{15:12}$ $d_{11:8}$ $d_{7:4}$ $d_{3:0}$
y_3 y_2 y_1 y_0 z_3 z_2 z_1 z_0
$O_{31:28}$ $O_{27:24}$ $O_{23:20}$ $O_{19:16}$ $O_{15:12}$ $O_{11:8}$ $O_{7:4}$ $O_{3:0}$

Fig. 4: A 32-bit leading-one detector [14].

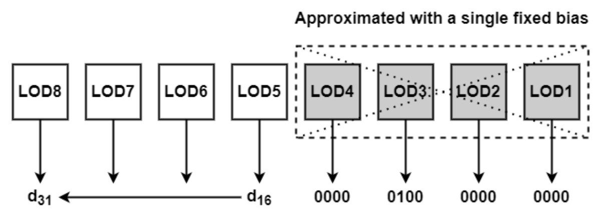


Fig. 5: Approximation of the LOD using a single fixed bias.

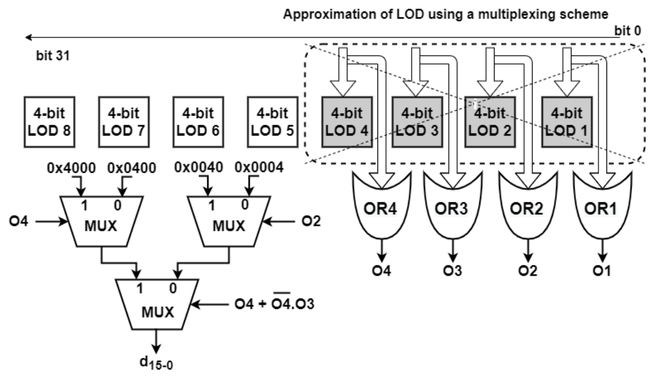


Fig. 6: Approximation of a LOD using a multiplexing scheme.

bits before being input to the LOD. If input A is shifted by q_1 bits and input B is shifted by q_2 bits, the final product needs to be shifted to the right by $q_1 + q_2$ bits. The scaling factor $s = 0$ will denote no scaling.

Comparing each input operand with the threshold value is a bottleneck of the proposed scaling technique. We solved this issue by using a simple, hardware-efficient solution that uses logical OR gates. The OR logic is performed on the $s \geq 1$ MSBs of the inputs. If the result is zero, that input can safely be shifted to the left by s bits. Otherwise, one or a few MSBs would be discarded by shifting, which would introduce a huge error. An example of this process is shown in Fig. 7 for $s = 6$.

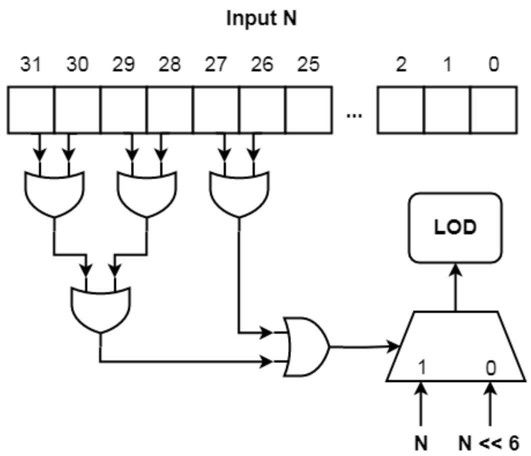


Fig. 7: An example of the proposed scaling technique for $s = 6$.

As shown in Fig. 7, the final output of the logical OR gate-tree is used as a multiplexer's select signal. If the select is '1', there is a '1' among the 6 MSBs and, therefore, the input operand N should not be shifted and we have to pass N to the LOD. On the other hand, a select of '0' means that there is no '1' in the 6 MSBs and we can safely input the shifted version of N to the LOD. Note that the LOD in Fig. 7 could be either an approximate or a fully-accurate design.

This design works with the scaling factor $s \in \{2, 4, 6, ..., 12, 14\}$, however we realized that the LOD design can be notably simplified for $s = 16$, resulting in a hardware-efficient accurate LOD, which is discussed below.

3.4 LOD design III

The main idea of the LOD III design is to use an exact n-bit LOD to find the position of the leading-one of a $2n$-bit number. The $2n$-bit input number N is divided into two halves, i.e. the more significant half ($N_H = N_{2n-1:n}$) and the less significant half ($N_L = N_{n-1:0}$). An OR gate tree, similar to that shown in Fig. 7, is used to find out whether or not there is a '1' in N_H. The two possible scenarios are as follows:

• There is a '1' in N_H. In this case, N_H is the input to an exact n-bit LOD. Since the output bit width of a conventional LOD for a $2n$-bit number is $2n$ bits, n zeros are appended to the lower half of the result as LSBs.
• There is no '1' in N_H. In this case, N_L is input to an exact n-bit LOD. With the same analogy as in the former case, n zeros are appended to the upper half of the result as MSBs.

The block diagram of LOD III is shown in Fig. 8. Note how 16 MSBs are evaluated using an OR gate tree and the final result, i.e. signal sel, is used as the select line of a multiplexer. Similar to the scaling technique, $sel =$ '1' means that there is at least one '1' within the 16 MSBs (N_H) and, therefore, N_H is the input to the 16-bit LOD. Otherwise, N_L is used as the input to the 16-bit LOD. Note that in Fig. 8, the symbol '&' denotes vector concatenation and not logical bit-wise AND.

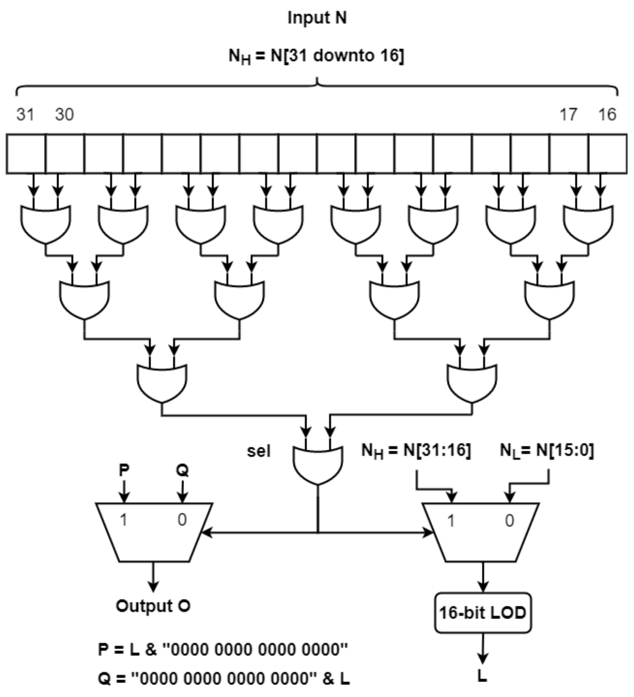


Fig. 8: LOD III, which uses a 16-bit LOD to find the position of the leading one in a 32-bit number.

Although LOD III is only evaluated in a logarithmic multiplier in this article, it is applicable to all of the other logarithmic arithmetic units that use a LOD, such as logarithmic squaring and square root functions and logarithmic dividers.

4 Simulation Results

This section provides simulation results for the proposed LODs. This is the first work on approximate LODs and, therefore, we can only compare LOD I and LOD II in the accuracy analysis. For the hardware cost analysis, on the other hand, the two existing references in the literature are considered for comparison purposes.

4.1 Accuracy analysis

The accuracy evaluations were performed using *MATLAB* models of LOD I and LOD II. We evaluated their accuracy using the Average Error (AE), which considers the error sign, and the Mean Relative Error distance (MRED) for 32-bit LODs. Since the both designs generate accurate results for inputs greater than 2^{16}, and considering that 2^{16} is only 0.001% of the 32-bit input domain, almost all the error metrics were zero (no error), even for 10^9 randomly generated inputs. Hence, we limited the maximum input to 2^{18}, 2^{20}, and 2^{22} and the results are provided in Table 3.

Table 3 Accuracy analysis of the two approximate LOD I and LOD II designs.

LOD Circuit	max. input=2^{18}		max. input=2^{20}		max. input=2^{22}	
	AE	MRED	AE	MRED	AE	MRED
LOD I	-1.0005	0.0713	-0.2493	0.0178	-0.0626	0.0045
LOD II	-0.0665	0.0147	-0.0167	0.0036	-0.0042	0.0009

As shown in Table 3, the error behavior improves as the maximum input increases. Also, the results in this table clearly show that the LOD II is more accurate than the LOD I.

Fig. 9 plots the error, $N - 2^{logN}$, for when 2^{logN} is calculated using the exact LOD, and three variants (with different scaling factors) of approximate designs LOD I and LOD II. Note that LOD III is an exact design and so its accuracy matches the conventional LOD. Thus, LOD III is not included in these two figures.

As shown in the four sub-figures of Figs. 9, Mitchell Multiplier using LOD I and II results in larger approximation errors than the conventional Mitchell's design for inputs of up to 2^{16}, but the errors are identical for all three designs for inputs of greater than 2^{16}. Although the error introduced by using approximate LODs is larger than that of the exact LOD for inputs less than 2^{16}, the maximum error over the entire input range, i.e. $0 \leq N < 2^{32}$, is actually the same for both the approximate and exact LOD designs. It is also shown in Fig. 9 how the scaling improves the accuracy and helps the approximate LODs to behave more like the exact LOD.

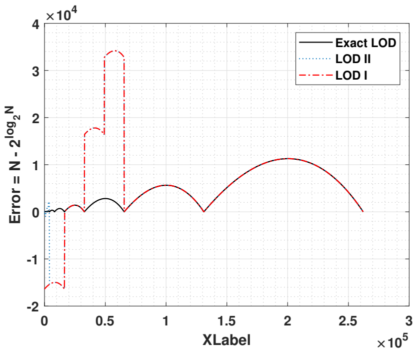
4.2 Hardware analysis

The three proposed designs are compared against three LOD designs in the literature in terms of their hardware characteristics. The only three LODs in the literature, to the best of our knowledge, are those reported in [14], [19], and [20]. Table 4 provides the results of this comparison with respect to the area, critical path delay, power consumption, and power-delay product (PDP).

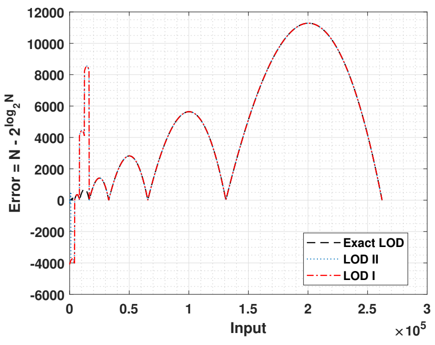
Table 4 Hardware comparison of the exact and approximate LODs.

LOD Circuit	Area (μm^2)	Delay (nS)	Power (μW)	PDP (fJ)
LOD I	86.00	0.38	6.22	2.36
LOD II	105.42	0.38	8.22	3.12
LOD III	**51.24**	**0.28**	**5.94**	**1.66**
LOD [14]	117.83	0.42	11.33	4.75
LOD [19]	109.67	0.43	11.36	4.88
LOD [20]	95.30	0.38	8.837	3.35

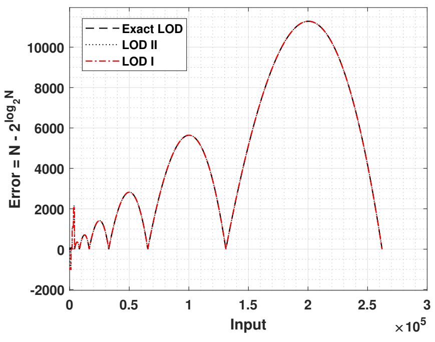
As can be seen, the proposed exact LOD III is the most hardware-efficient design. Unlike approximate designs LOD I and LOD II, LOD III does not use the scaling approach and, therefore, it is smaller, faster, and more power-efficient than the other two proposed designs. It is worth mentioning that the proposed approximate LOD I, although less hardware-efficient than LOD III, is still smaller and more energy-efficient than the exact LOD designs. LODs I and II can be used in applications (with or without the scaling part), where small signed inaccuracies can be beneficial, e.g. neural networks [21].



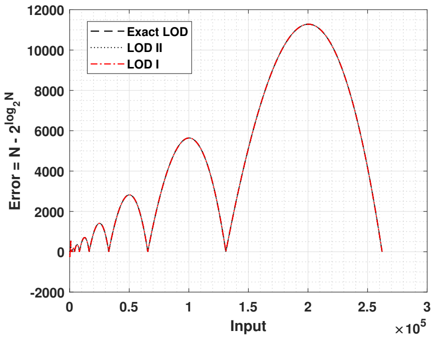
(a) Scaling factor $s = 0$



(b) Scaling factor $s = 2$



(c) Scaling factor $s = 4$



(d) Scaling factor $s = 6$

Fig. 9: Error in the logarithm approximations for four different scaling factors.

5 Applications of the LODs

Two applications are considered in this section for the proposed LODs: (1) the logarithmic multiplier, and (2) neural networks.

5.1 Logarithmic multiplier

Perhaps the most common application of logarithms to the implementation of arithmetic, judging from the literature, is multiplication. Mitchell is the most well-known logarithmic multiplier that is often used as the baseline design for many logarithmic arithmetic circuits. Although the benefits of using the proposed LODs are only shown by using them in the Mitchell multiplier, they are also applicable to any other logarithmic circuit that uses LODs.

The accuracy evaluations were performed using *MATLAB* models of the logarithmic multipliers. We evaluated their accuracy using the MRED for 32-bit multiplier designs with 10^4, 10^5, 10^6 and 10^7 randomly chosen unsigned integer pairs. The MRED metric gives the average relative error distance between the approximate and the exact products. From the simulation results, it was found that the obtained value of MRED = 0.0385 does not change after 10^6 iterations for all of the designs. With this MRED value, the four least significant LODs in Fig. 4 can be approximated.

The adder that sums the logarithms is also a significant part of the Mitchell multiplier and it can be approximated to further reduce the hardware cost. The authors in [12] showed that approximating the adder that sums the base-2 logarithms of the input operands does not reduce the accuracy of the Mitchell multiplier, and instead it improves the accuracy. Therefore, for the adder, the approximation is done by fixing the optimum number of output LSBs with a bias of alternating ones and zeros. Using simulation we determined that the 16 LSBs can be approximated in this way with no loss of accuracy in the product. By choosing an alternating sequence of ones and zeros, the Mitchell multiplier will sometimes overestimate and sometimes underestimate the true product resulting in a roughly symmetrical error distribution. This is a better approach than using all zeros (a.k.a. truncation) or all ones as it preserves the double-sided error distribution in the final product [7].

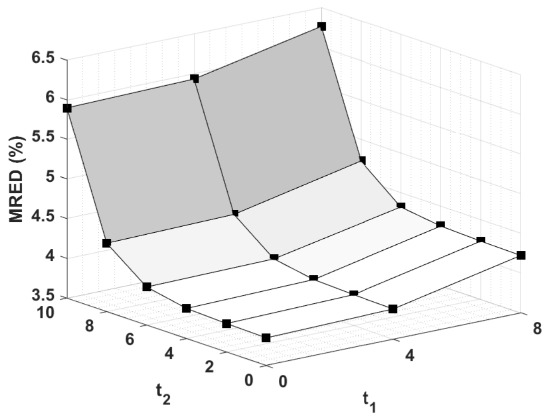


Fig. 10: MRED for the 16-bit Mitchell multiplier with LOD II.

Fig. 10 plots the MRED of the Mitchell multiplier with LOD II for different t_1 and t_2 parameters, where t_1 is the number of LSBs in LODs to be approximated and t_2 is the number of LSBs approximated for the adder. As t_1 and t_2 increase, the MRED increases. The optimum value of these parameters is chosen to match the accuracy (in MRED) of the Mitchell multiplier. For example, $t_1 = 4$ and $t_2 = 6$ are the optimum values for the 16-bit multiplier in Fig. 10. The MRED is plotted for a 16-bit version of the proposed design since it provides a clearer visualization than the 32-bit version. We observed that LOD I has a similar behavior and, therefore, the results are only shown for LOD II.

Furthermore, the Mitchell multiplier with the conventional LOD and several variants of the proposed LODs were implemented using the VHDL hardware description language using the Vivado design tool and then synthesized using the Synopsys Design Compiler for ST Micro's 28-nm CMOS process. The hardware measurements for four key metrics, area, critical path delay, power consumption, and

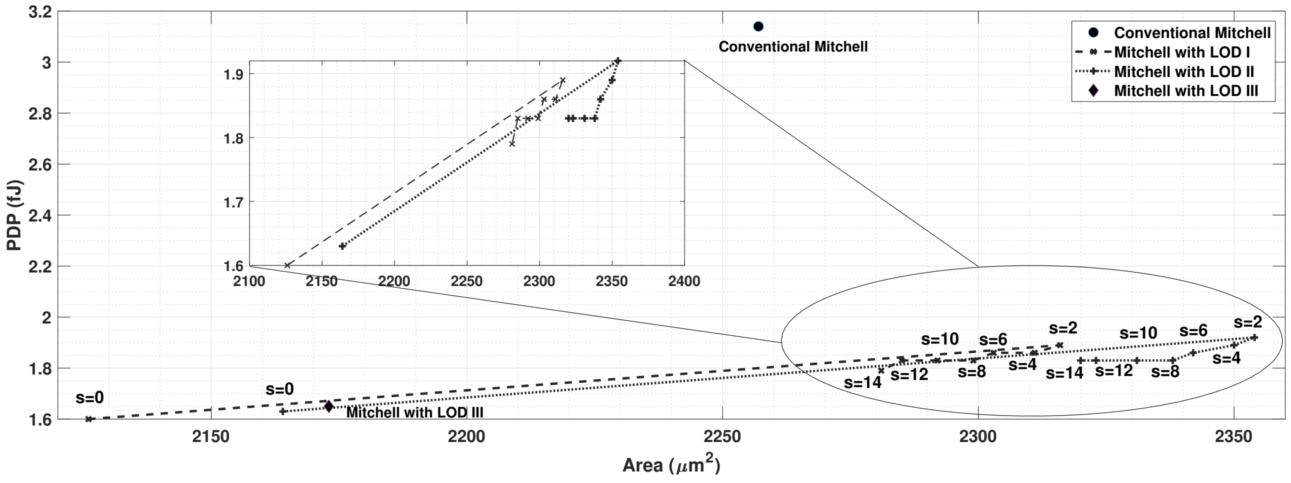


Fig. 11: PDP vs. area for Mitchell multiplier with four different LOD designs.

power-delay product (PDP), were extracted. Approximation of four least significant LODs in the first stage and of one least significant LOD in the second stage together with 16 LSBs approximated for the adder yields the hardware savings shown in Table 5, where the index e in *Mitchell with LOD I-e* denotes the scaling factor.

As shown in Table 5, the Mitchell multiplier with approximate and exact LODs is 24.47% and 30.35% faster than the Mitchell with the LOD proposed in [14], respectively. In terms of area, approximate LOD I and LOD II with scaling result in a larger multiplier than the conventional Mitchell multiplier; however, the baseline LOD I, LOD II, and the exact LOD III all result in smaller multipliers than the standard Mitchell multiplier. Finally, all of the proposed LOD designs reduce the power consumption of the Mitchell multiplier.

Table 5 Hardware characteristics of the Mitchell multiplier with different LODs.

LOD circuit	Power (mW)	Delay (nS)	Area (μm^2)	PDP (pJ)
LOD [14]	0.74	4.25	2257.21	3.14
LOD [20]	0.73	4.25	2213.48	3.12
LOD I-0	0.50	3.21	2126.00	1.60
LOD I-2	0.59	3.21	2316.13	1.89
LOD I-4	0.58	3.21	2311.23	1.86
LOD I-6	0.58	3.21	2303.73	1.86
LOD I-8	0.57	3.21	2299.16	1.83
LOD I-10	0.57	3.21	2292.63	1.83
LOD I-12	0.57	3.21	2285.12	1.83
LOD I-14	0.56	3.21	2281.20	1.79
LOD II-0	0.51	3.21	2164.84	1.63
LOD II-2	0.60	3.21	2354.97	1.92
LOD II-4	0.59	3.21	2350.08	1.89
LOD II-6	0.58	3.21	2342.57	1.86
LOD II-8	0.57	3.21	2338.00	1.83
LOD II-10	0.57	3.21	2331.47	1.83
LOD II-12	0.57	3.21	2323.96	1.83
LOD II-14	0.57	3.21	2320.05	1.83
LOD III	0.56	2.96	2173.66	1.65

For better visualization of the reported results in Table 5, the PDP vs. the area for all of the designs in Table 5 are plotted in Fig. 11. All of the variants of the proposed designs (with different scaling factors) are shown in this figure. Smaller designs with lower energy consumption, i.e. the designs in the bottom-left corner are preferable.

The normalized MRED-PDP product can be used as a metric for comparing approximate digital circuits with respect to both hardware and error metrics [10, 11]. However, all of the proposed designs in this article have the same accuracy (in terms of MRED) and, therefore, the MRED-PDP product is not a useful discriminating metric. Thus the PDP alone is used to identify the best of the proposed designs. The results in Table 5 show that using LOD III not only reduces the area, it also improves the energy efficiency of the Mitchell multiplier by 47.69%. Although all of the designs have the same MRED, LOD III is an exact design and, therefore, it is selected as the best of the proposed designs. As mentioned before, the main idea behind LOD III, i.e. using n-bit LOD for a $2n$-bit number, can be used to design more energy efficient LODs of different sizes.

5.2 *Neural networks*

An MLP with 784 input neurons (the images for the MNIST dataset are 28×28 and, therefore, 784 input neurons are required), a hidden layer of 300 neurons and a 10-neuron output layer is considered to classify the MNIST dataset.

The exact multipliers were replaced with several configurations of the Mitchell logarithmic multiplier, i.e. with different LODs and the Top-1 classification accuracy comparison is plotted in Fig. 12. As mentioned before, the LODs are designed so that all of the resulting multipliers have the same MRED and, therefore, only the conventional and base LOD I and LOD II (i.e. without scaling) are considered in Fig.12.

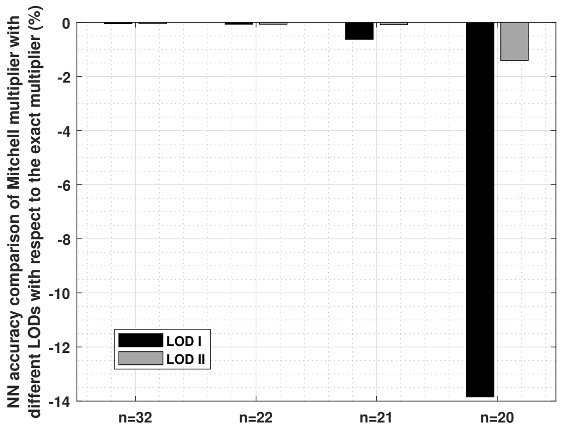


Fig. 12: Comparison of the Top-1 classification accuracy of the MNIST dataset with Mitchell multiplier with different LODs.

The classification accuracy using the exact multipliers is 97.81% in our simulations. This value is used as the golden result in the comparisons in Fig. 12. The Mitchell multiplier with a conventional exact LOD (any of the previously discussed designs, including LOD III, [20], [14], and [19]) achieves 97.76% accuracy. Note that in a 32-bit design, the inputs and synaptic weights should be scaled to 32-bit values, however, we scaled them to 20-bit, 22-bit, and 32-bit values (n in Fig. 12) to better show the differences. Since the accuracy of the Mitchell with the exact LOD is independent of n, it is not plotted in this figure and only the LOD I and II based designs are compared.

As the results in Fig. 12 show, LOD I causes more degradation in the MLP accuracy. This was expected as LOD I, unlike LOD II, uses a constant bias. Clearly, worse accuracies are achieved as n decreases: there is only 0.02% accuracy degradation going from 32-bit to 22-bit designs (97.76% vs. 97.74%) and then the accuracy rapidly drops, especially for LOD I, for $n < 21$. The results in this figure show that the approximate LODs can be safely (with proper scaling) used in an error-tolerant application without causing a noticeable loss in the output accuracy.

6 Conclusion

This article proposes two fast and low-power approximate LOD designs that can be used instead of an exact LOD. LOD I approximates the four least significant 4-bit LODs with a single fixed bias. LOD II, on the other hand, approximates the four least significant 4-bit LODs with one of four fixed biases chosen using a priority encoder controlled multiplexer. The adder that is used to sum the base-2 logarithms of the input operands is also approximated by using alternating ones and zeros for the 16 output LSBs. For smaller inputs, depending on the input scaling factor, approximate designs, LOD I and II can have large errors. To overcome the large errors for small numbers, an input scaling scheme is proposed. Considering a scaling factor s, if there is no '1' in the s MSBs of the 32-bit input N, N is shifted to the left by s bits. We observed that this helps the approximate LOD designs to better match the accuracy of the exact LOD. Moreover, an exact 16-bit LOD is proposed (LOD III) that is used to find the position of the leading-one in a 32-bit number N. The more significant half of N, N_H, is searched for '1'. If at least one '1' is found, N_H is input to the 16-bit LOD III; otherwise, the less significant half of N, N_L, is used as the input to LOD III.

The proposed LODs with different scaling factors are used in the Mitchell logarithmic multiplier, where the approximate LODs I and II achieve PDP reductions of 19.74% and 18.83%, respectively. The exact LOD III, however, reduces the PDP by 24.89% compared to the same conventional Mitchell multiplier. Furthermore, LOD III makes the Mitchell multiplier 1.4× faster. The Mitchell multiplier with different LODs is then used in an MLP that classifies the MNIST dataset. Our simulation results show that the approximate LOD I and II only slightly reduce the classification accuracy for input bit-widths exceeding 20 compared to when the exact LOD is used in the Mitchell multiplier.

7 References

1 Pramod Kumar Meher. *Arithmetic Circuits for DSP Applications*. John Wiley & Sons, 2017.

2 F. Taylor. An extended precision logarithmic number system. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(1):232–234, 1983.

3 E. E. Swartzlander and A. G. Alexopoulos. The sign/logarithm number system. *IEEE Transactions on Computers*, 24(12), 1975.

4 Vassilis Paliouras and Thanos Stouraitis. Low-power properties of the logarithmic number system. *15th IEEE Symposium on Computer Arithmetic*, pages 229–236, 2001.

5 Tomio Kurokawa, J Payne, and S Lee. Error analysis of recursive digital filters implemented with logarithmic number systems. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6):706–715, 1980.

6 Khalid H Abed and Raymond E Siferd. VLSI implementations of low-power leading-one detector circuits. In *Proceedings of the IEEE in SoutheastCon.*, pages 279–284, 2005.

7 Mohammad Saeed Ansari, Bruce Cockburn, and Jie Han. A hardware-efficient logarithmic multiplier with improved accuracy. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 922–925, 2019.

8 Aleksej Avramović, Zdenka Babić, Dušan Raič, Drago Strle, and Patricio Bulić. An approximate logarithmic squaring circuit with error compensation for DSP applications. *Microelectronics Journal*, 45(3):263–271, 2014.

9 Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. *18th IEEE European Test Symposium (ETS)*, pages 1–6, 2013.

10 M. Saeed Ansari, Honglan Jiang, Bruce F. Cockburn, and Jie Han. Low-power approximate multipliers using encoded partial products and approximate compressors. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8 (3):404–416, 2018.

11 Honglan Jiang, Francisco J. H. Santiago, M. Saeed Ansari, Leibo Liu, Bruce F. Cockburn, Fabrizio Lombardi, and Jie Han. Characterizing approximate adders and multipliers optimized under different design constraints. *Proceedings of the ACM Great Lakes Symposium on VLSI*, pages 393–398, 2019.

12 Weiqiang Liu, Jiahua Xu, Danye Wang, Chenghua Wang, Paolo Montuschi, and Fabrizio Lombardi. Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(9):2856–2868, 2018.

13 Tso-Bing Juang, Sheng-Hung Chen, and Huang-Jia Cheng. A lower error and ROM-free logarithmic converter for digital signal processing applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 56(12):931–935, 2009.

14 Khalid H. Abed and Raymond E. Siferd. CMOS VLSI implementation of a low-power logarithmic converter. *IEEE Transactions on Computers*, 52(11): 1421–1433, 2003.

15 Joshua Low, Lih Yung, and Ching Chuen Jong. Unified Mitchell-based approximation for efficient logarithmic conversion circuit. *IEEE Transactions on Computers*, 64(6):1783–1797, 2015.

16 John N. Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, (4):512–517, 1962.

17 Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

18 Shyama Gandhi, M. Saeed Ansari, Bruce F. Cockburn, and Jie Han. Approximate leading one detector design for a hardware-efficient Mitchell multiplier. *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 205–208, 2019.

19 K. Kunaraj and R. Seshasayanan. Leading one detectors and leading one position detectors-an evolutionary design methodology. *Canadian Journal of Electrical and Computer Engineering*, 36(3):103–110, 2013.

20 Min Soo Kim, Alberto A Del Barrio, Román Hermida, and Nader Bagherzadeh. Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks. *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 617–622, 2018.

21 Mohammad Saeed Ansari, Vojtech Mrazek, Bruce F Cockburn, Lukas Sekanina, Zdenek Vasicek, and Jie Han. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(2):317–328, 2019.