Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks

Min Soo Kim, Alberto A. Del Barrio, Leonardo Tavares Oliveira, Román Hermida, *Senior Member, IEEE*, and Nader Bagherzadeh, *Fellow, IEEE*

Abstract—This paper proposes energy-efficient approximate multipliers based on the Mitchell's log multiplication, optimized for performing inferences on convolutional neural networks (CNN). Various design techniques are applied to the log multiplier, including a fully-parallel LOD, efficient shift amount calculation, and exact zero computation. Additionally, the truncation of the operands is studied to create the customizable log multiplier that further reduces energy consumption. The paper also proposes using the one's complements to handle negative numbers, as an approximation of the two's complements that had been used in the prior works. The viability of the proposed designs is supported by the detailed formal analysis as well as the experimental results on CNNs. The experiments also provide insights into the effect of approximate multiplication in CNNs, identifying the importance of minimizing the range of error.The proposed customizable design at $w = 8$ saves up to 88 percent energy compared to the exact fixed-point multiplier at 32 bits with just a performance degradation of 0.2 percent for the ImageNet ILSVRC2012 dataset.

Index Terms—Arithmetic and logic units, low-power design, machine learning , computer vision, object recognition

1 Introduction

The breakthroughs in multi-layer convolutional neural networks (CNNs) have caused significant progress in the application of image classification and speech recognition. From the milestone network LeNet for the MNIST handwritten digit recognition [1] to ResNet achieving better than human classification [2], CNNs have been continually studied and improved to perform well even for the large-scale image classification [3]. The CNNs developed in this progress, such as AlexNet [4], VGG [5] and GoogLeNet [6], showed the trend where the amount of computations augmented as the number of layers increased for better accuracy. With such a large and growing number of computations, as well as the rising application of machine learning techniques to many areas, it is vital to develop efficient processing hardware units for CNNs. Particularly, CNNs involve many multiply-accumulate (MAC) operations, and it is important to implement efficient multipliers as they consume large amounts of power and area [7].

One distinction to make in neural network computation is the training versus inference computations. Training teaches neural networks to develop the classification capabilities through the gradient-based backpropagation algorithms performed on a large amount of supplied data. These algorithms involve delicate computations of gradient values and are performed with floating-point units. The DaDianNao project [8] studied applying fixed-point arithmetic to training and found that training required much more precision than inference. Hence, it is difficult to apply approximate computing to the training of CNNs.

On the other hand, many research papers have shown that it is possible to apply approximate computing to the inference stage of neural networks after training with exact arithmetic [7], [9], [10], [11]. Such techniques usually demonstrated small drops in performance but had significant reduction in resources. While the training phase involves much more computation, it may be performed offline in advance, and the approximated devices can be deployed to perform inferences using the trained network data. The resource reduction, especially the energy savings, would be beneficial for embedded systems and datacenters, as emphasized by the efforts from Google to create a custom TPU processor for machine learning in its datacenters [12].

The logarithmic multiplier is a promising topic of research for approximate neural network computation. This multiplier converts multiplications into additions by taking approximate logarithm. This algorithm is well known to have a significant benefit in area and power savings while maintaining a reasonably low error rate. Many research such as [13], [14], [15], [16] have recognized its benefits and attempted to improve it since the original proposal by John Mitchell in 1962 [17]. The original algorithm has a low worst case relative error that is proven to be as low as 11.1 percent, and this property is potentially important in neural networks that emulate firing of neurons. A large worst case error has a greater chance of incorrectly

- *M.S. Kim and N. Bagherzadeh are with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92697. E-mail: {minsk1, nader}@uci.edu.*
- *A.A. Del Barrio and R. Hermida are with Computers Architecture and Automation, Complutense University of Madrid, Madrid 28040, Spain. E-mail: {abarriog, rhermida}@ucm.es.*
- *L.T. Oliveira is with Computer Engineering, Universidade Federal de Sao Carlos, Sao Carlos 13565-905, Brazil. E-mail: leonardot1802@gmail.com.*

Manuscript received 29 May 2018, revised 31 Aug. 2018, accepted 13 Sept. 2018, Date of publication 11 Nov. 2018; date of current version 15 Apr. 2019. (Corresponding author: Min Soo Kim.)
Recommended for acceptance by J. D. Bruguera.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2018.2880742

firing a neuron, which would lead to a larger probability of incorrect classification.

We seek to make the following contributions in this paper. First, we present energy-efficient approximate log multipliers based on the Mitchell's logarithmic multiplication algorithm. We add various techniques to our prior work [18] to create customizable designs that are more energy-efficient and perform well on CNNs. Second, we perform various experiments on three different CNNs and datasets to evaluate the effects of the proposed designs and the other approximate multipliers. Not only do we establish that the proposed designs are a promising solution to reduce the energy consumption of CNN processing, but also provide an in-depth analysis of the workings of CNNs under approximate multiplication. We also provide significant results on the more complex dataset of ImageNet while the prior works focused on simpler datasets and networks. Lastly, we evaluate the cost of negative number handling as seen on many prior works, and propose the 1's complement sign conversion as a viable approximation of the previous approach. We provide the formal analysis of its effect on the log multiplier and perform the experiments to justify its benefits for CNNs.

The rest of the paper is organized as follows. Section 2 discusses the related work, particularly the prior state of the art when applying approximate multipliers to neural networks. Section 3 introduces the preliminary concepts related to the Logarithmic Number System (LNS) and the logarithmic multiplier presented in our prior work [18]. Section 4 presents the customizable logarithmic multiplier and Section 5 discusses the handling of negative numbers with the 1's complement approximation. Section 6 describes the techniques to unbias the errors that provide slight accuracy improvement on CNNs. The synthesis results of the proposed designs are presented in Section 7, and the experiments on the CNNs are discussed in Section 8. Lastly, the conclusion of the paper is presented in Section 9.

2 Related Work

There have been many works on the approximate computing and the efficient processing of CNNs, and we narrow the scope of relevancy to discuss the papers that specifically applied approximate multiplier designs to neural networks.

The application of logarithmic multiplication to neural networks had been studied in [9]. This paper used the 2-stage pipelined iterative logarithmic multiplication presented in [13], and the authors chose the iterative design over the original Mitchell algorithm because it had lower error rate and provided the opportunity for pipelining. They also claimed that the original Mitchell algorithm did not have a large reduction of power and area compared to their design. Our study takes a different direction where we optimize the Mitchell algorithm implementation and add the approximation techniques to have a significant power and area savings compared to the iterative design, and demonstrate that our design performs as well as theirs on CNNs despite the higher error.

Mrazek et al. used a heavy search-based method of Cartesian Genetic Programming (CGP) to achieve significant power reduction in CNNs [11]. They generated various approximate multipliers from the accurate multiplier by mutating the gates within the constraint of maximum target error. The CGP-based methodology requires a large design space exploration to find the optimal solution for each neural network and authors report an 11-bit multiplier as the largest one achievable, which could be too small for certain CNNs. On the other hand, their method is a gate-level approximation and can be considered complementary to our algorithm-level approximation. Our technique has the benefit of being independent of technology, can be easily scaled for the number of bits, and does not require searching the design space.

Du et al. have also achieved energy reduction through the gate-level approximation of Inexact Logic Minimization, where the bits are intentionally flipped to reduce logic [7]. They used heuristic-based exploration of the design space, and focused on approximating the multipliers where the most benefits were expected. They have also emphasized the importance of retraining the network after approximation to reduce the network performance degradation, and it complements our work. This search-based method is different than our work for the same reasons discussed in the previous paragraph.

Sarwar et al. proposed applying the Alphabet Set Multiplier to improve energy consumption and area, at the cost of small degradation in neural network performance [10]. This multiplier design precomputed multiples of the multiplier values as alphabets and used shifting and adding of these values to approximate the output. They found that they could eliminate the pre-computing of the alphabets and use only the multiplier value as the single alphabet to greatly reduce power and area consumption, at the cost of 2.83 percent drop in network accuracy in the MNIST dataset. This may be an acceptable drop in some cases, but it may be a significant drop for an application that requires near-perfect accuracy. They also found that the CNN accuracy drop was amplified as the applications became more complex.

There are other approximate multipliers that had not been applied to CNNs, including [19], [20], [21], [22], [23], [24], [25], [26] and [27]. DRUM [19] in particular compared well against [23] and [24] and showed good characteristics in our evaluations, so we applied the multiplier to CNN computations and compared against our own design.

There are other approaches of efficient CNN processing at the cost of accuracy degradation, such as [28], [29], [30], and [31] among many others. While all of them are very progressive and interesting research topics that propose different paradigms of efficient CNN processing, it is not easy to integrate them into the conventional systems. There is a gap between the data scientists who seek the ease of usage and the high CNN performance, and the engineering field that seeks the maximum energy efficiency. Improving the energy consumption with the approximate multiplier can potentially fill the gap as it can be easily integrated into the ASIC accelerators for CNN processing.

3 Preliminaries

In this section some basic concepts related to the Logarithm Number System will be introduced. Besides, the basics of

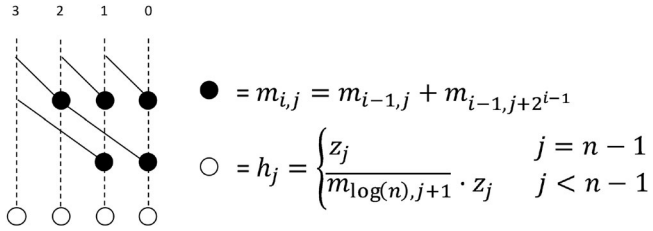

Fig. 1. 4-bit leading one detector.

the logarithmic multiplier presented in our prior work [18] will be described.

In order to transform an n-bit binary number Z to the LNS, the transformation depicted by Equation (1) is employed [9], [14], [17], [18].

$$Z = \sum_{i=0}^{n-1} 2^i z_i = 2^k \left(1 + \sum_{i=j}^{k-1} 2^{i-k} z_i\right), \quad k \geq j \geq 0, \qquad (1)$$

where k is the position corresponding to the leading one, z_i is a bit value at the ith position, and j depends on the number's precision. Then, the logarithm with the basis 2 of Z is expressed by Equation (2).

$$\log_2(Z) = \log_2\left(2^k\left(1 + \sum_{i=j}^{k-1} 2^{i-k} z_i\right)\right) = k + \log_2(1+x). \qquad (2)$$

Then, the expression $\log_2(1+x)$ is approximated with x, as $\forall x \in [0,1]$ both expressions provide close results [17]. On the other hand, the obtention of k, aka. *characteristic*, is essential for converting Z to the LNS. In order to do this, it is necessary to employ a Leading One Detector (LOD) in combination with an Encoder (ENC) module. In our prior paper [18] we proposed an efficient parallel implementation of the Leading One Detector whose output is provided in one-hot format, such that it is possible to implement the ENC module with a tree of logic OR gates instead of a costly priority encoder. As an example, a 4-bit LOD block is shown in Fig. 1.

Afterwards, the two operands in LNS format are added and converted back to binary thanks to an antilogarithm module. In parallel, once the outputs of the ENC blocks are known, it is possible to detect if the result will be zero. If this is the case, the outcome of the antilogarithm will be converted to zero.

4 The Customizable Logarithmic Multiplier

Our proposal is detailed by Algorithm 1 and Fig. 3. It must be noted that & stands for the concatenation symbol and $x[b..a]$ represents the bits that range from positions b to a belonging to signal x. As in the multiplier presented in [18], we will focus on bitwidths that are a power of 2. If n is not a power of two, there are some optimizations that cannot be applied.

The main difference with respect to our prior work [18] is the introduction of the parameter w. This parameter indicates that only the most significant w bits of the operand will be taken into account. As the leading one is encoded into the characteristic, this means that only the most significant $w-1$ mantissa bits are considered to be added. Some approaches in literature [10], [19] have shown that under the paradigm of approximate computing, the most significant part of a value may be sufficient to provide an acceptable approach. Fig. 2 shows the truncation schemes presented at [19] and ours. Figs. 2a and 2b illustrate how [19] unbiases and truncates an operand using w bits, namely: the leading one located at the kth position, $w-2$ bits, and an extra '1' for approximating the least significant part. This operand will be driven to a wxw multiplier then. On the other hand, Fig. 2c and 2d describe the resulting mantissa after left-shifting the operand, and how the $w-1$ most significant bits (excluding the leading one) are used in combination with the characteristic k to compose the operand that will be driven to the adder within the logarithmic multiplier. These differences ultimately stem from the fact that our designs perform the truncation in the approximate logarithmic domain while authors in [19] operate in the linear domain.

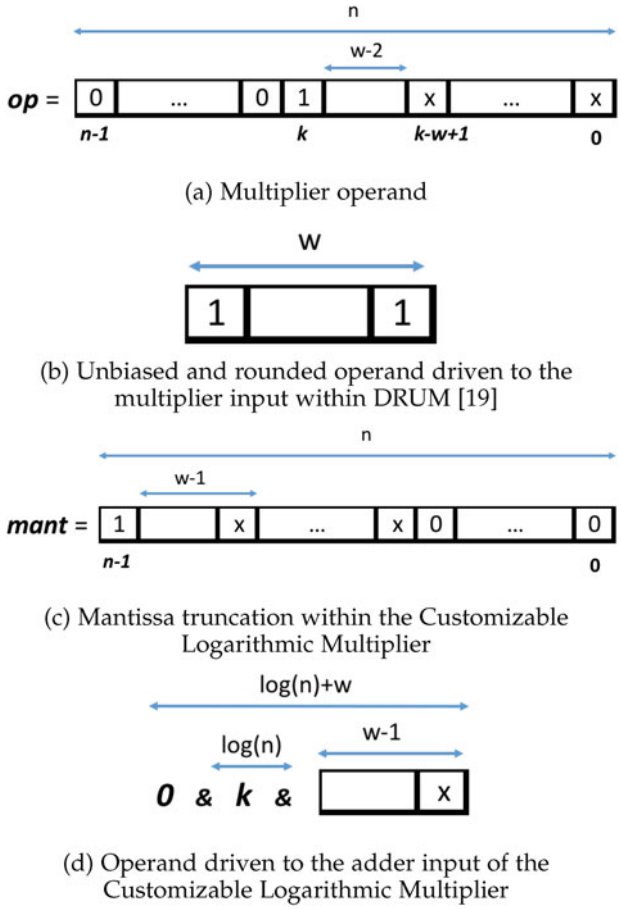

Fig. 2. Operands truncation methods.

The use of the aforementioned truncated summands implies using a customized antilogarithm block. The design of this module is shown in Fig. 4. As observed, utilization of the w parameter has a large impact on power and area of this block. The original customized left barrel shifter shown at [18] decreases its size from $2n$ to $n+w$, while the right barrel shifter and the multiplexer decrease their sizes from n to w. Reducing the size of these blocks has a positive impact on power and area, but at the expense of losing some accuracy. In the following sections there is a study about this accuracy loss.

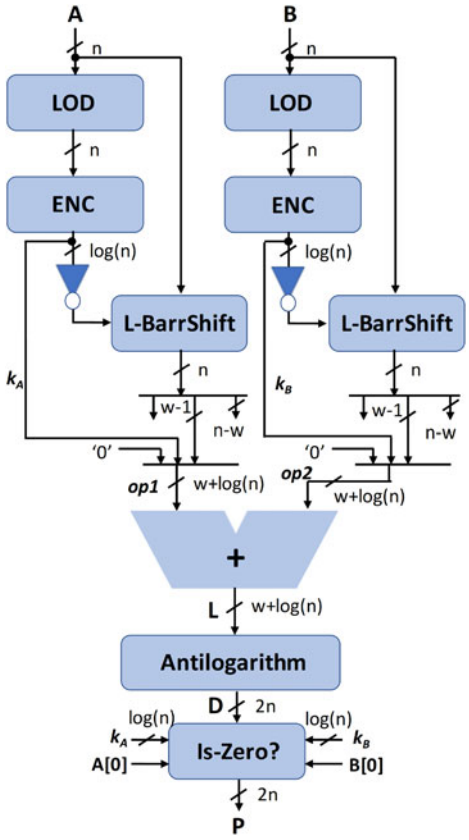

Fig. 3. Customizable log approximate multiplier, mitch-w.

Algorithm 1. Customized Mitchell's Algorithm (Mitch-w)

Input: A, B: n-bits, $w \in [0,n\text{-}1]$
Output: P: 2n-bits ▷ P is an approximate product

▷ Logarithm

$h_A \leftarrow LOD(A), h_B \leftarrow LOD(B)$
$k_A \leftarrow ENC(h_A), k_B \leftarrow ENC(h_B)$
$x_A \leftarrow A << (n - k_A - 1), x_B \leftarrow B << (n - k_B - 1)$

▷ Addition in the LNS domain

$op1 \leftarrow$ '0' & k_A & $x_A[n-2..n-w]$
$op2 \leftarrow$ '0' & k_B & $x_B[n-2..n-w]$
$L \leftarrow op1 + op2$

▷ Antilogarithm

$charac \leftarrow L[w + \log_2(n) - 1..w-1]$
$lr \leftarrow charac[\log_2(n)]$
$m \leftarrow$ '1' & $L[w-2..0]$
if $lr =' 1'$ **then** ▷ Large characteristic
  $shamtL \leftarrow$ ('0' & $charac[\log_2(n) - 1..0]) + 1$
  $D \leftarrow m << shamtL$
else ▷ Small characteristic
  $shamtR \leftarrow n - charac[\log_2(n) - 1..0] - 1$
  $D \leftarrow m >> shamtR[\log_2(n)..\log_2(n) - \log_2(w)]$
end if

▷ Check if the result should be zero

if $A = 0 \vee B = 0$ **then**
  $P \leftarrow 0$
else
  $P \leftarrow D$
end if

Finally, as in our prior work, it must be noted in Algorithm 1 that some improvements can be done to provide an implementation as optimized as possible. For instance, by applying one's complement arithmetic it is possible to compute $n - k_{Op} - 1$ as $not(k_{Op})$, n being power of 2. The same applies when calculating $shamtR$, which can be obtained as $not(charac[\log_2(n) - 1..0])$. Another interesting feature is the realization of the zero condition at the output, which is implemented by checking whether or not the operand characteristics (k_A or k_B) are zero. If a characteristic is zero, in order to certify that an operand is zero, it is enough to check that the least significant bit (LSB) of the operand is '0' as well. This reduces the sizes of the comparators with zero. This design will hereafter be referred to as Mitch-w.

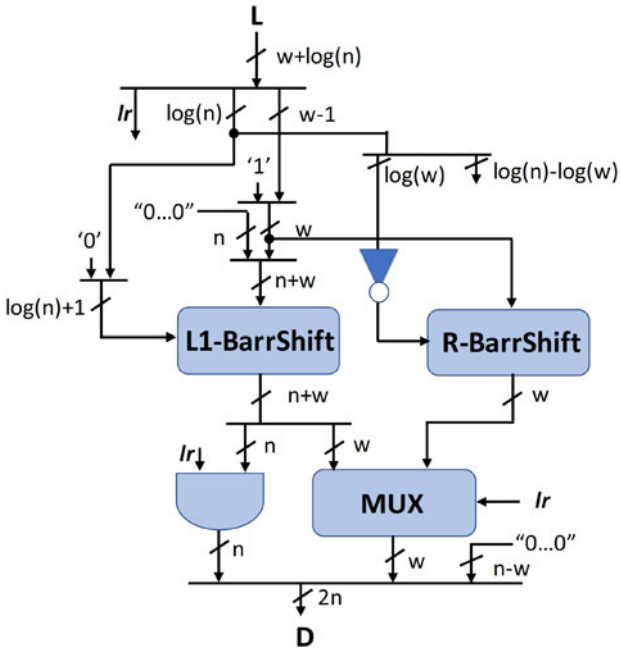

Fig. 4. Customizable antilogarithm block.

4.1 Error Study

In this section, an analysis on the use of the customizable logarithmic multiplier is presented. This error (E_D) is the one produced at the output of the antilogarithm block, as truncating some bits on the operand mantissas does not affect the zero-checking block. Thus, it must be noted that the error with respect to a conventional Mitchell logarithmic multiplier is first produced when truncating the operands.

$$\widehat{m} = m/2^{n-1} < 1, \tag{3}$$

$$y_A = x_A \wedge (2^{n-1} - 1) \tag{4}$$

$$y_B = x_B \wedge (2^{n-1} - 1) \tag{5}$$

$$y'_A = x_A \wedge (2^{n-1} - 2^{n-w}) > y_A - 2^{n-w}, \tag{6}$$

$$y'_B = x_B \wedge (2^{n-1} - 2^{n-w}) > y_B - 2^{n-w}, \tag{7}$$

$$y_L = (y_A + y_B) \wedge (2^{n-1} - 1), \tag{8}$$

$$y'_L = (y'_A + y'_B) \wedge (2^{n-1} - 1), \tag{9}$$

$$k_L = k_A + k_B + (((y_A + y_B) \wedge (2^{n-1})) >> (n-1)), \tag{10}$$

$$k'_L = k_A + k_B + (((y'_A + y'_B) \wedge (2^{n-1})) >> (n-1)), \quad (11)$$

$$\begin{aligned} L &= k_A * 2^{n-1} + y_A + k_B * 2^{n-1} + y_B \\ &= k_L * 2^{n-1} + y_L, \end{aligned} \quad (12)$$

$$\begin{aligned} L' &= k_A * 2^{n-1} + y'_A + k_B * 2^{n-1} + y'_B \\ &= k'_L * 2^{n-1} + y'_L, \end{aligned} \quad (13)$$

$$D = antilog(L) = 2^{k_L} * (1 + \widehat{y_L}), \quad (14)$$

$$D' = antilog(L') = 2^{k'_L} * (1 + \widehat{y'_L}). \quad (15)$$

First, according to Equation (3) let us define $\widehat{m}$ as the normalized version of a generic mantissa m, so $\widehat{m}$ will always be $\in$ [0, 1). Let x_A and x_B be the mantissas as defined in Algorithm 1, i.e., after left shifting. And let y_A and y_B be the mantissas after removing the leading one, i.e., the most significant bit, as Equations (4) and (5) indicate. And let y'_A and y'_B be the truncated version of those, as Equations (6) and (7) point, respectively. Let us define L and L' as Equations (12) and (13) indicate, respectively.

It must be noted that k_L and k'_L do not necessarily match, as if $y_L \neq y'_L$, the carry-out propagating towards the characteristic part of L and L' may be different too. Then, the error E_D performed when computing D' with respect to D in Algorithm 1 is as pointed by Equation (16).

$$\begin{aligned} E_D = D - D' &= 2^{k_L} * (1 + \widehat{y_L}) - 2^{k'_L} * (1 + \widehat{y'_L}) \\ &= 2^{k_L} * ((1 + \widehat{y_L}) - 2^{k'_L - k_L} * (1 + \widehat{y'_L})). \end{aligned} \quad (16)$$

As $k_L \geq k'_L$, because L' comes from the truncated operands, two cases may happen then: either $k_L = k'_L$, or $k_L > k'_L$. If $k_L = k'_L$, then both y_L and y'_L propagate, or do not propagate, a carry-out to the characteristic parts k_L and k'_L, respectively. Then, $y_L = y_A + y_B - carry$, $y'_L = y'_A + y'_B - carry'$, and Equation (17) holds.

$$\begin{aligned} E_D &= 2^{k_L} * ((1 + \widehat{y_L}) - (1 + \widehat{y'_L})) \\ &= 2^{k_L} * (\widehat{y_L} - \widehat{y'_L}) \\ &= 2^{k_L} * (\widehat{y_A} - \widehat{y'_A} + \widehat{y_B} - \widehat{y'_B}) \\ &< 2^{k_L} * (2^{-w+1} + 2^{-w+1}) = 2^{k_L - w+2}. \end{aligned} \quad (17)$$

It must be noted that as Equation (6) indicates, $y_A - y'_A$ is lower than 2^{n-w}, and then $\widehat{y_A} - \widehat{y'_A}$ is lower than 2^{-w+1}. Analogously for Equation (7). On the other hand, if $k_L > k'_L$, then $k_L = k'_L + 1$. As the only difference between L and L' is a carry-out being propagated to the non-truncated part, there can only be one unit of difference in the characteristic part of L. Furthermore, if in L such carry propagation exists, while in L' does not, $\widehat{y'_L}$ will be composed of just '1's, and $\widehat{y_L}$ will be transformed into a chain of '0's. Under these conditions, Equation (18) holds.

$$\begin{aligned} E_D &= 2^{k_L} * ((1 + \widehat{y_L}) - 2^{k'_L - k_L} * (1 + \widehat{y'_L})) \\ &= 2^{k_L} * (1 - (1 + \widehat{y'_L})/2) \\ &= 2^{k_L} * \left(1 - \left(1 + \sum_{i=n-w+1}^{n-1} 2^{i-n}\right)/2\right) \\ &= 2^{k_L} * (1 - (1 + 1 - 2^{-w+1})/2) \\ &= 2^{k_L} * (2^{-w}) = 2^{k_L - w}. \end{aligned} \quad (18)$$

Thus, E_D is never larger than $2^{k_L - w+2}$. Or, in relative terms, it is $2^{k_L - w+2}/(2^{k_L} * (1 + \widehat{y_L})) = 2^{-w+2}/(1 + \widehat{y_L})$. In order to finally bound the error, in the worst case $\widehat{y_L} = 0$, so it can be concluded that the relative error, with respect to the conventional logarithmic multiplier, is always lower than 2^{-w+2}. Hence, the shorter the w, the larger the error.

5 Handling Negative Numbers

In this section a proposal for handling the negative values will be described. The CNNs often involve negative weights and require the arithmetic of signed numbers. One problem with the prior design in [18] was that it did not handle signed numbers naturally, and required the 2's complements sign conversion (hereafter referred to as C2) before and after the design so that the inputs to the log multiplier are always positive. Each operand went through C2 if it was negative, and the output result also went through C2 depending on the expected output sign determined by the input signs. The inability to process signed numbers is found in many approximate multipliers, and the ones based on locating the leading one in the operands all suffer the same problem because the leading one is always placed at the leftmost bit for the negative numbers. The previous approaches either had assumed the sign-magnitude representation of signed numbers [9] or inserted C2 before and after their designs as we did [10], [19]. None of the prior works had evaluated the associated costs to fully investigate the issue.

Our proposal consists of leveraging the approximate computing scenario introduced by the CNNs. In this way, both inputs of the multiplier will be in two's complement. In this format there are several well-known facts:

- If $X \in \mathbb{Z}$, then $C2(X) = C1(X) + 1$. Thus, if a number is negative, it can be approached with its one's complement (hereafter referred to as C1) at the expense of introducing an error of one *unit in the last place* (ulp).
- In C2, the sign of a number is given by the most significant bit (MSB). Thus, if $X \geq 0$ the MSB is '0', and on the contrary if $X < 0$, the MSB is '1'.

The proposed design is shown in Fig. 5. As observed, first both operands are XOR-ed with their sign bit in order to convert the negative numbers into the positive domain by using the C1 transform. Consequently, the sign of the result will be the logical XOR between the operand signs. If such bit is '1', then the result provided by the antilogarithm block must be XOR-ed to produce a negative result. In this case, the zero handling is performed differently in comparison with our prior work [18], although a similar idea is followed: the result is not zero if both operands are not zero. Let k_{O+} be the characteristic of an operand O after being converted to the positive domain. And let msb_O (i.e., sign) and lsb_O be the most and least significant bits of an operand O. Then, the conditions under which an operand is not zero are given by Table 1.

- If $k_{O+} > 0$, this means that the operand is not zero (cases 4 to 7).
- If $k_{O+} = 0$ and the operand is negative ($msb_O = 1$), the operand is not zero (cases 2 and 3).

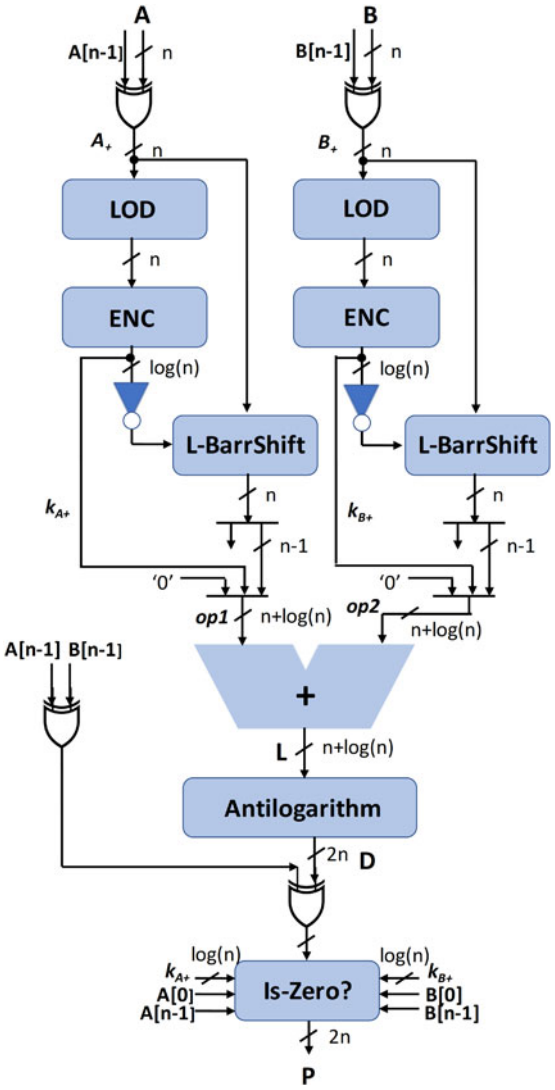

Fig. 5. Logarithmic multiplier with negative numbers handling.

- If $k_{O+} = 0$ and the operand is positive ($msb_O = 0$), the operand is not zero if lsb_O is 1 (case 1), and zero otherwise (case 0).

Therefore, the condition under which the result is not zero is as Equation (19) indicates.

$$notZero_D = notZero_A \wedge notZero_B = (k_{A+} > 0 \vee msb_A \vee lsb_A) \wedge (k_{B+} > 0 \vee msb_B \vee lsb_B). \tag{19}$$

5.1 Error Study

The error produced by our approach when A and B have different signs and when A and B are negative is generally low. Intuitively, there is only 1 unit of difference between computing C1 or C2 of an operand. In this section, the error introduced by the proposed scheme to handle negative numbers will be studied in detail. Provided that the operands are not zero, the error occurs at the output of the antilogarithm block. Four cases may arise depending on the sign of both operands A and B. If both are positive, there is no error in comparison to the conventional logarithmic multiplier.

TABLE 1
Truth Table for Determining Whether an Operand Is Zero or Not

$k_{O+} > 0$	msb_O	lsb_O	$notZero_O$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

5.1.1 A and B Possess Different Sign

Without loss of generality, let us suppose that $A < 0$ and $B \geq 0$. If $A \geq 0$ and $B < 0$, the analysis is analogous. In this scenario, let us define the variables indicated by Equations (20) to (28).

$$y_L = (y_{C2(A)} + y_B) \wedge (2^{n-1} - 1), \tag{20}$$

$$y'_L = (y_{C1(A)} + y_B) \wedge (2^{n-1} - 1), \tag{21}$$

$$k_L = k_{C2(A)} + k_B + (((y_{C2(A)} + y_B) \wedge (2^{n-1})) >> (n-1)), \tag{22}$$

$$k'_L = k_{C1(A)} + k_B + (((y_{C1(A)} + y_B) \wedge (2^{n-1})) >> (n-1)), \tag{23}$$

$$L = k_{C2(A)} * 2^{n-1} + y_{C2(A)} + k_B * 2^{n-1} + y_B \tag{24}$$

$$= k_L * 2^{n-1} + y_L\ , \tag{25}$$

$$L' = k_{C1(A)} * 2^{n-1} + y_{C1(A)} + k_B * 2^{n-1} + y_B = k_L * 2^{n-1} + y'_L, \tag{26}$$

$$P = C2(2^{k_L} * (1 + \widehat{y_L})) = C2(U),$$

$$P'_{Neg1} = C1(2^{k'_L} * (1 + \widehat{y'_L})) = C1(V), \tag{27}$$

$$E_{Neg1} = P - P'_{Neg1}. \tag{28}$$

$$\begin{aligned} E_{Neg1} &= P' - P'_{Neg1} = C2(U) - C1(V) \\ &= C2(U) - (C2(V) - 2^{-n+1}) = V - U + 2^{-n+1} \\ &= 2^{k'_L} * (1 + \widehat{y'_L}) - 2^{k_L} * (1 + \widehat{y_L}) + 2^{-n+1} \\ &= 2^{k_L} * (2^{k'_L - k_L} * (1 + \widehat{y'_L}) - (1 + \widehat{y_L})) + 2^{-n+1}. \end{aligned} \tag{29}$$

As in the prior theorem, two cases arise: $k_L = k'_L$ and $k_L > k'_L = k_L - 1$. If $k_L = k'_L$, then by definition of two's and one's complement $k_{C2(A)} \geq k_{C1(A)}$. If $k_{C2(A)} = k_{C1(A)}$ and $k_{C2(B)} = k_{C1(B)}$, the logical AND in the definition of y_L and $y_{L'}$ has no effect and then Equation (30) holds. If $k_{C2(B)} > k_{C1(B)}$, the situation will be analogous as described by Equation (34).

$$\begin{aligned} E_{Neg1} &= 2^{k_L} * ((1 + \widehat{y'_L}) - (1 + \widehat{y_L})) + 2^{-n+1} \\ &= 2^{k_L} * (\widehat{y'_{C1(A)}} - \widehat{y'_{C2(A)}} + \widehat{y_B} - \widehat{y_B}) + 2^{-n+1} \\ &= 2^{k_L} * (\widehat{y'_{C1(A)}} - \widehat{y'_{C2(A)}}) + 2^{-n+1}\ . \end{aligned} \tag{30}$$

The difference between $\widehat{y_{C2(A)}}$ and $\widehat{y_{C1(A)}}$ is as studied in Equations (31) and (32), where the logical AND of

Equations (20) and (21) has no effect.

$$\begin{aligned} y_{C2(A)} - y_{C1(A)} &= (C2(A) << \neg(k_{C2(A)})) \wedge (2^{n-1}-1) \\ &\quad - (C1(A) << \neg(k_{C1(A)})) \wedge (2^{n-1}-1) \\ &= (C2(A) - C1(A)) << n - k_{C1(A)} - 1 \\ &= 2^{n-k_{C1(A)}-1}, \end{aligned} \tag{31}$$

$$\widehat{y_{C2(A)}} - \widehat{y_{C1(A)}} = 2^{n-k_{C1(A)}-1}/2^{n-1} = 2^{-k_{C1(A)}}. \tag{32}$$

Introducing Equation (31) into Equation (30) it is possible to get Equation (33).

$$E_{Neg1} = 2^{k_L} * (-2^{-k_{C1(A)}}) + 2^{-n+1} \approx -2^{k_L - k_{C1(A)}}. \tag{33}$$

In this case, the relative error is $(-2^{k_L-k_{C1(A)}})/(2^{k_L} * (1+\widehat{y_L})) > -1/2^{k_{C1(A)}}$.

If $k_{C2(A)} = k_{C1(A)} + 1$, then $C2(A)$ is a power of two and $y_{C2(A)} = 0$, and y_L does not propagate any carry. Thus the logical AND of its definition has no effect. On the other hand, if $k_L = k'_L$, then $k_{C2(B)} = k_{C1(B)}$ and y'_L must propagate to compensate $k_{C2(A)} = k_{C1(A)} + 1$, i.e., $y'_L = y_{C1(A)} + y_B - 1$. In this scenario, Equation (34) holds.

$$\begin{aligned} E_{Neg1} &= 2^{k_L} * ((1 + \widehat{y_{C1(A)}} + \widehat{y_B} - 1)/2 - (1 + \widehat{y_B})) + 2^{-n+1} \\ &< 2^{k_L} * ((\widehat{y_{C1(A)}} + \widehat{y_B}) - (1 + \widehat{y_B})) + 2^{-n+1} \\ &\approx 2^{k_L} * (\widehat{y_{C1(A)}} - 1). \end{aligned} \tag{34}$$

Therefore, the relative error is $(2^{k_L} * (\widehat{y_{C1(A)}} - 1))/\ (2^{k_L} * (1+\widehat{y_L})) = (\widehat{y_{C1(A)}} - 1)/(1+\widehat{y_L}) \to 0$. As $C2(A)$ is a power of two, $y_{C2(A)} = 0$ and consequently $y_{C1(A)}$ will possess several '1's in the most significant positions. For instance, consider $n = 8$ bits, $A = -64$, $C2(A) = 64$ and then $k_{C2(A)} = 6$, $y_{C2(A)} = 0$ and $\widehat{y_{C2(A)}} = 0$. On the other hand, $C1(A) = 63$, $k_{C1(A)} = 5$, $y_{C1(A)} = 120 = 11111000$b and $\widehat{y_{C1(A)}} = 0.9375$.

If $A \geq 0$ and $B < 0$, similar expressions can be obtained.

5.1.2 A and B Are Negative

In the scenario where $A < 0$ and $B < 0$, let us define the variables indicated by Equations (35) to (43).

$$y_L = (y_{C2(A)} + y_{C2(B)}) \wedge (2^{n-1} - 1), \tag{35}$$

$$y'_L = (y_{C1(A)} + y_{C1(B)}) \wedge (2^{n-1} - 1), \tag{36}$$

$$\begin{aligned} k_L &= k_{C2(A)} + k_{C2(B)} \\ &\quad + (((y_{C2(A)} + y_{C2(B)}) \wedge (2^{n-1})) >> (n-1)), \end{aligned} \tag{37}$$

$$\begin{aligned} k'_L &= k_{C1(A)} + k_{C1(B)} \\ &\quad + (((y_{C1(A)} + y_{C1(B)}) \wedge (2^{n-1})) >> (n-1)), \end{aligned} \tag{38}$$

$$\begin{aligned} L &= k_{C2(A)} * 2^{n-1} + y_{C2(A)} + k_{C2(B)} * 2^{n-1} + y_{C2(B)} \\ &= k_L * 2^{n-1} + y_L, \end{aligned} \tag{39}$$

$$\begin{aligned} L' &= k_{C1(A)} * 2^{n-1} + y_{C1(A)} + k_{C1(B)} * 2^{n-1} + y_{C1(B)} \\ &= k_L * 2^{n-1} + y'_L, \end{aligned} \tag{40}$$

$$P = 2^{k_L} * (1 + \widehat{y_L}), \tag{41}$$

$$P'_{Neg2} = 2^{k'_L} * (1 + \widehat{y'_L}), \tag{42}$$

$$E_{Neg2} = P - P'_{Neg2} = 2^{k_L} * ((1 + \widehat{y_L}) - 2^{k'_L - k_L} * (1 + \widehat{y'_L})). \tag{43}$$

Again, two possible scenarios arise: either $k_L = k'_L$, or $k_L > k'_L$. If $k_L = k'_L$, then by definition of two's and one's complement $k_{C2(A)} \geq k_{C1(A)}$ and $k_{C2(B)} \geq k_{C1(B)}$. If $k_{C2(A)} = k_{C1(A)}$ and $k_{C2(B)} = k_{C1(B)}$, then the logical AND in Equations (35) and (36) have no effect and applying Equation (32) it is possible to prove that Equation (44) holds.

$$\begin{aligned} E_{Neg2} &= 2^{k_L} * (\widehat{y_{C2(A)}} - \widehat{y_{C1(A)}} + \widehat{y_{C2(B)}} - \widehat{y_{C1(B)}}) \\ &= 2^{k_L} * (2^{-k_{C1(A)}} + 2^{-k_{C1(B)}}). \end{aligned} \tag{44}$$

In this case, the relative error would be $2^{k_L} * (2^{-k_{C1(A)}} + 2^{-k_{C1(B)}})/(2^{k_L} * (1+\widehat{y_L})) < 2^{-k_{C1(A)}} + 2^{-k_{C1(B)}}$.

If $k_L = k'_L$ and $k_{C2(A)} > k_{C1(A)} = k_{C2(A)} - 1$, then $C2(A)$ is a power of two and then $y_{C2(A)} = 0$. In this scenario, $k_{C2(B)} = k_{C1(B)}$, as otherwise $(k_{C2(A)} + k_{C2(B)}) - (k_{C1(A)} + k_{C1(B)}) = 2$, and this cannot be compensated by a carry out propagated from y'_L to k'_L. On the other hand, a difference of just 1 can be compensated if y'_L propagates, and then $\widehat{y'_L} = \widehat{y_{C1(A)}} + \widehat{y_{C1(B)}} - 1$. Thus, Equation (45) holds.

$$\begin{aligned} E_{Neg2} &= 2^{k_L} * ((1 + \widehat{y_{C2(B)}}) - (1 + \widehat{y_{C1(A)}} + \widehat{y_{C1(B)}} - 1)) \\ &= 2^{k_L} * (1 - \widehat{y_{C1(A)}} + 2^{-k_{C1(B)}}) < 2^{k_L - k_{C1(B)}}. \end{aligned} \tag{45}$$

It must be noted that if $C2(A)$ is a power of 2, then $C1(A)$ and consequently $y_{C1(A)}$ will possess several '1's in the most significant positions. Thus, $(1 - \widehat{y_{C1(A)}}) \to 0$. Hence, the relative error is around $2^{k_L - k_{C1(B)}}/(2^{k_L} * (1+\widehat{y_L})) < 2^{-k_{C1(B)}}$. A similar expression can be found if $k_L = k'_L$ and $k_{C2(B)} > k_{C1(B)} = k_{C2(B)} - 1$.

If $k_L > k'_L$, several scenarios may happen depending on the relation between $k_{C2(A)}$ and $k_{C1(A)}$, and $k_{C2(B)}$ and $k_{C1(B)}$. If $k_{C2(A)} = k_{C1(A)}$ and $k_{C2(B)} = k_{C1(B)}$, then y_L propagates a carry-out while y'_L does not. Thus, $\widehat{y_L} = \widehat{y_{C2(A)}} + \widehat{y_{C2(B)}} - 1$, $k_L = k'_L + 1$ and Equation (46) holds.

$$\begin{aligned} E_{Neg2} &= 2^{k_L} * ((1 + \widehat{y_{C2(A)}} + \widehat{y_{C2(B)}} - 1) \\ &\quad - (1 + \widehat{y_{C1(A)}} + \widehat{y_{C1(B)}})/2)) \\ &= 2^{k_L} * (2^{-k_{C1(A)}} + 2^{-k_{C1(B)}} - 1/2) \\ &< 2^{k_L} * (2^{-k_{C1(A)}} + 2^{-k_{C1(B)}}). \end{aligned} \tag{46}$$

In this case again the relative error is lower than $2^{-k_{C1(A)}} + 2^{-k_{C1(B)}}$. If $k_L > k'_L$, $k_{C2(A)} > k_{C1(A)} = k_{C2(A)} - 1$ and $k_{C2(B)} = k_{C1(B)}$, then $y_{C2(A)}$ is zero, so y_L does not propagate a carry-out and neither does y'_L, otherwise $k_L = k'_L$. Thus, $k_L = k'_L + 1$. Hence, the logical AND in Equations (35) and (36) has no effect and Equation (47) holds.

$$\begin{aligned} E_{Neg2} &= 2^{k_L} * ((1 + \widehat{y_{C2(B)}}) - (1 + \widehat{y_{C1(A)}} + \widehat{y_{C1(B)}})/2)) \\ &= 2^{k_L - 1} * (1 + 2 * \widehat{y_{C2(B)}} - \widehat{y_{C1(B)}} - \widehat{y_{C1(A)}}) \\ &< 2^{k_L - 1} * (1 + \widehat{y_{C2(B)}} - \widehat{y_{C1(B)}} - \widehat{y_{C1(A)}}) \\ &= 2^{k_L - 1} * (1 + 2^{-k_{C1(B)}} - \widehat{y_{C1(A)}}) \\ &\approx 2^{k_L - k_{C1(B)} - 1}. \end{aligned} \tag{47}$$

In this case the relative error is lower than $(2^{k_L - k_{C1(B)}-1})/(2^{k_L} * (1+\widehat{y_L})) < 2^{-k_{C1(B)}-1}$. If $k_{C2(B)} > k_{C1(B)} = k_{C2(B)} - 1$ and $k_{C2(A)} = k_{C1(A)}$, an analogous expression can be found.

The last case to be studied is $k_L > k'_L$, $k_{C2(A)} > k_{C1(A)} = k_{C2(A)} - 1$ and $k_{C2(B)} > k_{C1(B)} = k_{C2(B)} - 1$. In this scenario, $C2(A)$ and $C2(B)$ are power of two, so $y_{C2(A)}$, $y_{C2(B)}$ and y_L are zero, while $y_{C1(A)}$ and $y_{C1(B)}$ will have several '1's in the most significant positions (at least one) so y'_L will propagate a carry-out to k'_L and then $k_L = k'_L + 1$. Thus, $\widehat{y'_L} = \widehat{y_{C1(A)}} + \widehat{y_{C1(B)}} - 1$ and Equation (48) holds.

$$\begin{aligned} E_{Neg2} &= 2^{k_L} * (1 - (1 + \widehat{y_{C1(A)}} + \widehat{y_{C1(B)}} - 1)/2)) \\ &= 2^{k_L} * (1 - (\widehat{y_{C1(A)}} + \widehat{y_{C1(B)}})/2)). \end{aligned} \tag{48}$$

For the same reasons as in Section 5.1.1, $y_{C1(A)}$ and $y_{C1(B)}$ will have several '1's in the most significant positions and will be close to 1. Thus, $(\widehat{y_{C1(A)}} + \widehat{y_{C1(B)}})/2 \rightarrow 1$, and then the error shown in Equation (48) will tend to zero, and so will do the relative error.

5.1.3 Corner Cases

Despite the fact that the general low error when employing the one's complement, there are some special cases that need careful attention. In the prior proofs it has been considered that, without loss of generality, when $C2(A)$ is a power of two, then $y_{C2(A)}$ is 0 and consequently $C1(A)$ and $y_{C1(A)}$ will possess several '1's in the most significant positions. This is valid but for two special cases, namely: -1 and -2.

Following Fig. 5, it is possible to observe that when $A = -1$, $C1(A) = A_+ = 0$, $k_{C1(A)} = k_{A_+} = 0$ and $op1 = 0$. Hence, multiplying by -1 works as adding the neutral element in the LNS domain. It must be noted, that according to Table 1 the result will not be converted to zero, despite the fact that $A_+ = 0$. Furthermore, if $B = -1$ and consequently $op2 = 0$, the antilogarithm will produce $D = 1$ anyway.

When $A = -2$, $C1(A) = A_+ = 1$, $k_{C1(A)} = k_{A+} = 0$ and $op1 = 0$ and $y_{C1(A)} = 0$. Hence, a similar situation arises, but with the drawback of producing a larger error, as we are actually multiplying by -2. Once again, if $B = -1$ or $B = -2$, the result given by the antilogarithm will be $D = 1$.

This situation is different when we have larger power of two numbers, because $k_{A+} > 0$ and there will be at least a MSB equal to '1' in $y_{C1(A)}$. For instance, $A = -4$, $C1(A) = A_+ = 3$, $k_{A+} = 1$ and $y_{C1(A)} = 2^{n-2}$, i.e., $\widehat{y_{C1(A)}} = 0.5$.

5.1.4 Error Interpretation

In order to summarize this section, it must be noted that all the errors depend on the one's complement of an operand, and as the operand is smaller in absolute terms, the error grows. In other words, the relative error is high for the smallest operands, but in absolute terms the difference is not high because the operands are small. In general, this will not have a large impact on the CNNs, as small results will not contribute considerably to the MAC output of each convolution and fully connected node. This claim is supported by the experiments and the analysis presented in Section 8.4.

6 Unbiasing the Errors

The proposed log multiplier only produces negative errors, which means that it produces approximated results that are equal to or less than exact multiplication. It never produces a result with positive error, which is larger than the correct result. Therefore, it has the biased error distribution that increases the values of the mean error and the worst case error (hereafter abbreviated as WCE). Thus, our method is to evaluate the effect of having a low mean error and the error distribution centered around zero. The w truncation and approximate logarithm are the two sources of error in the proposed designs and both of them produce only negative errors. To create the unbiased designs and have the mean error closer to zero, both of the error sources must be addressed.

The entire error of DRUM multiplier comes from the truncation error and it uses rounding up to make the error unbiased as shown earlier in Fig. 2b [19]. We adapted their technique in our Mitch-w designs, so that the LSB of each adder operand is truncated and replaced with '1', which is equivalent to removing the least significant full adder and using a carry-in equal to '1'. Moreover, this technique slightly decreases the cost of other parts within the designs, as it truncates one bit and reduces the number of bits associated with the barrel shifters and the multiplexer within the antilogarithm block.

The detailed description of the approximate logarithm is presented in [17]. The error from approximate logarithm can be unbiased by adding a constant mantissa to the added log result so that the linear approximation of the mantissa is shifted up. This causes some results to have positive errors where the log curve is below the shifted linear approximation. We have empirically found with RTL simulations that adding the binary mantissa '0.0001' to the result of the addition makes the designs unbiased to have the small mean error of 0.4 percent for 16 and 32 bits.

7 The Synthesis Results

To evaluate the energy and area savings of the proposed designs, we performed synthesis using Synopsys Design Compiler and compared it against the synthesis results of the exact fixed-point multiplier. The exact multiplier is automatically synthesized by Design Compiler from the simple Verilog multiplication. We used a 32nm digital standard cell library from Synopsys, and repeated the synthesis for 8, 16, and 32 bits. The synthesis was performed with Ultra effort at the clock frequency of 250 MHz. Critical path, area, and total power are reported by the Design Compiler, and energy is calculated from the critical path and power. The error values are obtained through RTL simulations using QuestaSim. The simulations are performed with 1,000,000 random input values and reported to the nearest 10th of a percent. All energy savings reported in the tables are in comparison to the exact fixed-point multiplier with the same number of bits.

Table 2 shows the comparison of the synthesis results of Mitch-w designs, along with the error characteristics. The error values are relative to the results of exact multiplication, and PWCE stands for the positive worst case error while NWCE stands for the negative worst case error. The Mitch multiplier refers to our prior design in [18] that did not apply the w truncation. Our multiplier saves significant energy and area compared to the exact fixed-point multiplier, and we also improve significantly upon our prior

TABLE 2
The Comparison of the Synthesis Results of Mitch-w Designs

	N = 8					N = 16						N = 32					
	Exact	Mitch	Mitch-w			Exact	Mitch	Mitch-w				Exact	Mitch	Mitch-w			
			$w=5$	$w=6$	$w=7$			$w=5$	$w=6$	$w=7$	$w=8$			$w=5$	$w=6$	$w=7$	$w=8$
Mean Err. (%)	0	−3.8	−6.5	−4.7	−4	0	−3.8	−7.9	−5.9	−4.9	−4.4	0	−3.9	−7.9	−5.9	−4.9	−4.4
PWCE (%)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
NWCE (%)	0	−11.1	−17.3	−13.8	−12.0	0	−11.1	−18.0	−14.6	−12.9	−12.0	0	−11.1	−18.0	−14.7	−12.9	−12.0
Critical Path (ns)	1.07	1.13	0.97	1.13	1.14	2.23	2.31	1.53	1.59	1.70	1.66	3.78	3.70	1.67	1.98	1.99	2.03
Design Area (um^2)	474	389	305	353	376	2032	1150	592	693	760	824	8627	2890	1314	1346	1496	1559
Tot. Power (mW)	0.27	0.20	0.15	0.17	0.18	1.24	0.55	0.26	0.31	0.34	0.38	6.02	1.41	0.47	0.53	0.58	0.62
Energy (pJ)	0.29	0.23	0.15	0.19	0.21	2.77	1.27	0.40	0.49	0.58	0.63	22.76	5.22	0.78	1.05	1.15	1.26
Energy Savings		22%	50%	34%	29%		54%	86%	82%	79%	77%		77%	97%	95%	95%	94%

work. Compared to the exact multiplier, our parametrized design at $w = 8$ can save up to 94 percent of energy at 32-bits, and have potentially more cost savings with smaller w. It is important to note that the w customizable truncation results in a lot of additional energy savings compared to the original Mitchell's log multiplier without substantial increase in error.

The multiplier in our prior work [18] had the critical path that was comparable with the exact fixed-point multiplier. With the introduction of the w truncation, we reduce the critical path lengths of the designs, as the adders and the barrel shifters get simplified. We see the highest benefits in the 32-bit designs, where the critical path reduces as much as 46 percent, almost doubling the potential processing speed and energy reduction.

Table 3 shows the synthesis results for the original and unbiased versions of Mitch-w as described in Section 6. The unbiased designs have the positive errors as well as the negative errors, and the error ranges are slightly increased while the mean errors and the WCE are decreased. Adding the constant to the result of the adder within the log multiplier introduced the additional circuitry that increased the critical path length for all designs except $w = 6$ at N = 32.

Table 4 shows the synthesis results of the proposed designs that include the C2 and C1-based signed number handling. The error characteristics are omitted in this table because the corner cases produce large errors that make the comparison difficult. Table 4 shows that the power and area costs of C2 negative number handling are significant, but the C1 approximation reduces the costs considerably.

TABLE 3
The Synthesis Results of the Unbiased Designs

	N = 16				N = 32			
	Original		Unbiased		Original		Unbiased	
	$w=6$	$w=8$	$w=6$	$w=8$	$w=6$	$w=8$	$w=6$	$w=8$
Mean Err. (%)	−5.9	−4.4	0.4	0.4	−5.9	−4.4	0.4	0.4
PWCE (%)	0	0	12.4	7.7	0	0	12.4	7.7
NWCE (%)	−14.6	−12.0	−11.1	−8.2	−14.7	−12.0	−11.1	−8.2
Critical Path (ns)	1.59	1.66	1.81	1.98	1.98	2.03	1.93	2.45
Area (um^2)	693	824	660	819	1346	1559	1394	1556
Tot. Power (mW)	0.31	0.38	0.30	0.38	0.53	0.62	0.51	0.63
Energy (pJ)	0.49	0.63	0.54	0.75	1.05	1.26	0.98	1.54
Energy Savings	82%	77%	80%	73%	95%	94%	96%	93%

Energy savings are calculated with respect to the exact multiplier.

The C2 conversions also add significant delays to the critical path and make the designs slower. The C1 approximation removes the step of adding one, and the compiler performed better optimizations so that the critical path lengths of the C1-based designs are significantly shortened, even shorter than the unsigned version for 16 bits. The signed designs with C1 perform faster than the exact fixed-point multipliers, while the C2 designs do not. Combined with power, the C1-based designs are significantly more energy-efficient than the C2 counterparts.

8 CNN Experiments

We used the Caffe framework to evaluate the applicability of our approximate multiplier on CNNs. Caffe is a well-known deep learning framework that provides the tools to train and test CNN models for the visual recognition applications [32]. We implemented various multiplier designs in C++ using a fixed-point class library, and replaced the matrix multiplication of Caffe with our own code that calls the implemented procedures. All CNN experiments in this paper were performed with 16 integer bits and 16 fractional bits, as they were sufficient for all three network instances we used. CNN computations involve signed numbers and the C2 conversion was used before and after the approximate multipliers to correctly handle them. The results with the C1 conversion are analyzed in Section 8.4.

We used three different datasets to evaluate our designs: MNIST, CIFAR-10, and ImageNet ILSVRC2012 validation data. Table 5 describes the CNNs used to perform inferences for each dataset. The approximate multipliers were not suitable for the training of the networks, because the amount of error was too large for the gradient descent of backpropagation to converge. For the MNIST and CIFAR-10 datasets, we trained the networks first using floating-point arithmetic and applied the multiplier models to the inference stage. For these datasets, the Top-1 CNN inference accuracy has been measured across all 10,000 test images. For AlexNet, we simply used the pre-trained model that was available in Caffe without further training. Performing experiments on all 50,000 images of ILSVRC2012 validation set proved to be very time-consuming, because the C++ models of the approximate multipliers were much slower to simulate than the conventional multiplication performed in hardware. Therefore, we primarily used the first 5,000 images from the shuffled set to perform various

TABLE 4
The Synthesis Results of the Signed Designs

	N = 8			N = 16						N = 32					
	Unsigned	C2	C1	Unsigned		C2		C1		Unsigned		C2		C1	
	$w = 6$	$w = 6$	$w = 6$	$w = 6$	$w = 8$	$w = 6$	$w = 8$	$w = 6$	$w = 8$	$w = 6$	$w = 8$	$w = 6$	$w = 8$	$w = 6$	$w = 8$
Crit. Path (ns)	1.21	1.99	1.32	1.81	1.98	3.84	3.69	1.75	1.90	1.93	2.45	3.86	3.86	2.19	2.50
Area (um^2)	380	624	483	660	819	1236	1287	922	1135	1394	1556	3028	3168	1815	2092
Tot. Power (mW)	0.18	0.33	0.26	0.30	0.38	0.60	0.70	0.50	0.61	0.51	0.63	1.52	1.64	0.90	1.08
Energy (pJ)	0.22	0.66	0.34	0.54	0.75	2.30	2.58	0.88	1.16	0.98	1.54	5.87	6.33	1.97	2.70
Energy Savings	25%	−127%	−19%	80%	73%	17%	7%	68%	58%	96%	93%	74%	72%	91%	88%

Energy savings are calculated with respect to the exact multiplier.

experiments, and then used the entire set of 50,000 images in Section 8.6 to further prove our designs.

AlexNet is not as contemporary as some of the other networks like ResNet [2], but the ILSVRC2012 on AlexNet was the well-established milestone in the visual recognition research, and it serves as the important intermediate step for the research of approximate multipliers on CNNs. The prior works on applying approximate multipliers to CNNs are performed in the simpler datasets, and none of them has demonstrated significant benefits in ImageNet. It is significantly more challenging than MNIST and CIFAR-10, because the images are more complex and there are 1000 possible outputs as opposed to 10.

The performance of CNNs for visual object classification is typically measured by the Top-1 and Top-5 accuracies. Given an image, a CNN produces the inference scores of all object classes, and the Top-1 accuracy measures the rate of the object with highest score matching the correct answer. For the Top-5 accuracy, the correct answer is searched among the top five scores instead of just one.

8.1 Impact on CNN Performance

Table 6 shows the Top-1 accuracies on MNIST and CIFAR-10 with different multipliers. Mitchell refers to our prior work in [18], and Mitch-w employs the proposed design with $w = 6$. As can be observed, our low-energy design does not cause significant performance degradation despite the error from approximation.

TABLE 5
The Descriptions of CNNs

Dataset	CNN	Layers
MNIST	LeNet	Conv[5x5] → Pooling[2x2, stride 2] → Conv→ Pooling→ Fully Connected (FC) [500 output] → ReLU → FC [10 output]
CIFAR-10	Cuda-convnet	Conv[5x5] → Pooling[3x3, stride 2] → ReLU → LRN[3x3] → Conv → ReLU → Pooling → LRN → Conv → ReLU → Pooling → FC [10 output]
ImageNet	AlexNet	Conv[11x11] → ReLU → LRN → Pooling[3x3] → Conv[5x5] → ReLU → LRN → Pooling [3x3] → Conv[3x3] → ReLU → Conv[3x3] → ReLU → Conv[3x3] → ReLU → Pooling[3x3] → FC [4096 output] → ReLU → FC [4096 output] → ReLU → FC [1000 output]

Fig. 6 shows the sample convolution layer outputs and the final raw scores from one of the MNIST inferences. The Mitchell multiplier produces the convolution outputs that are very close to the floating-point multiplier in terms of the relative magnitude. Mitchell's algorithm always produces negative errors, meaning that the magnitude of the values are always reduced, but it is applied to all values so that the convolution can still locate the abstract features in the images.

The final layer output shows the effect of approximate multiplication more clearly, as the scores are reduced in magnitude. The reductions are more than the −4 percent mean error of the Mitchell's multiplier and indicate that the errors have been accumulated across the layers. Despite the difference in the absolute magnitude of results, the log multiplier still correctly recognizes "2", because all the outputs are reduced at the same time so that the order between them is preserved. The effect of the log multiplier is as if the image was reproduced with smaller pixel values; lighter image but still clearly recognizable. In fact, CNNs are designed to have the tolerance for such small differences in the input images to successfully classify as much images as possible. The computational error did not affect the performances of the CNNs, because they are measured by the correctness of the discrete outputs. The absolute score computed for each option is not as important as the relative order of the scores. This property of the discrete classification makes CNNs resilient against approximations and reduced precisions of computation.

The exact fixed-point multiplication showed better performance in CIFAR-10 than the reference floating-point as shown in Table 6. Even though the Mitchell's multiplier shows exactly the same accuracies as the reference in MNIST and CIFAR-10, a closer examination revealed that they did not produce the same predictions for all test images. The floating-point multiplier produced better predictions for some images while the Mitchell's multiplier did better for others. This happens because inexact

TABLE 6
The Top-1 Accuracies for LeNet and Cuda-Convnet

Network	Float	Fixed	Mitchell	Mitch-w
LeNet	99.0%	99.0%	99.0%	99.0%
cuda-convnet	81.4%	81.9%	81.4%	81.3%

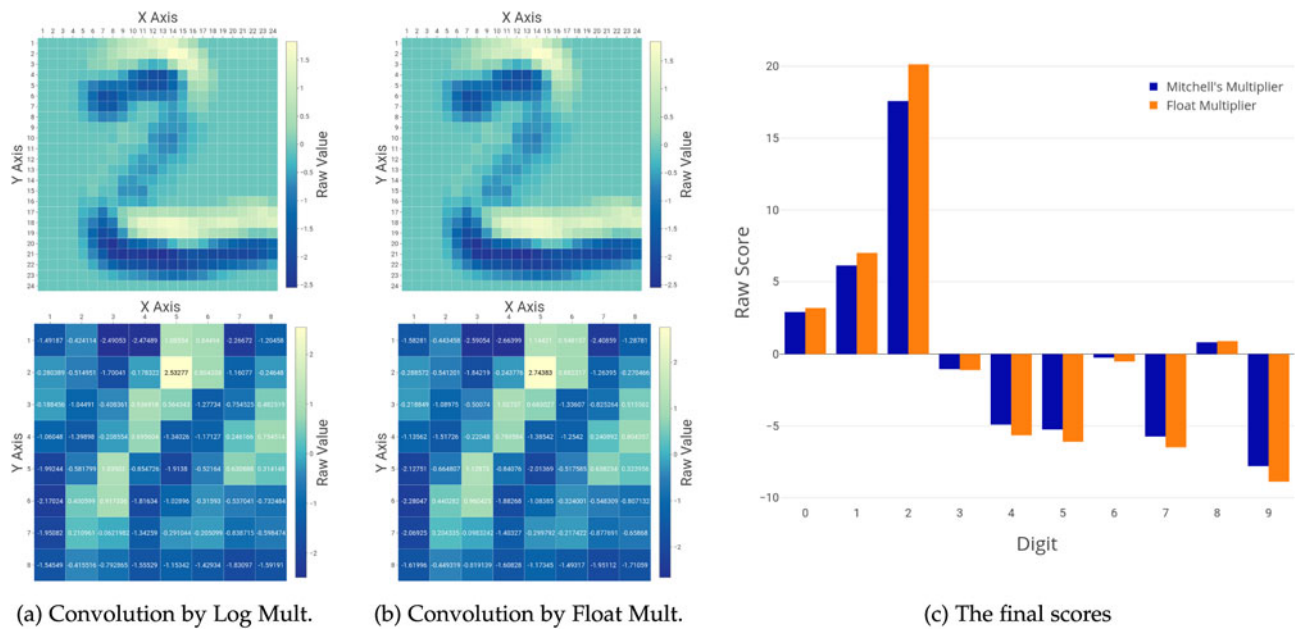

Fig. 6. The convolution outputs and the final raw scores of a sample inference from LeNet.

computations can result in correct predictions when exact computations would have resulted in the incorrect ones.

Table 7 shows the Top-1 and Top-5 accuracies for AlexNet on ImageNet dataset. Fig. 7 shows the top 10 scores of a sample inference sorted by the reference floating-point score. The results from AlexNet demonstrate the same concepts presented in MNIST. The negative errors of approximate designs decrease magnitudes of the final scores, but all the output options are affected so that the relative order between them are mostly preserved.

The ImageNet [3] is more challenging than the MNIST for the approximate multiplication primarily because there are many more possible outputs and their raw scores are much closer together. The variations in error accumulation can change the order of the output more easily. We can also see in Fig. 7 that the amount of error accumulation shown by the value reduction is higher than the MNIST example, because AlexNet has more layers and computations compared to LeNet.

Yet, despite the difficulties the proposed designs perform reasonably well even for the ImageNet. The relative order in the top 10 are slightly modified between the floating-point and Mitch-w in Fig. 7, but the example still demonstrates the same top 5 scores, with the same emphasis on the first score. Also, it should be noted that there are images that the floating-point incorrectly classifies while the Mitch-w does correctly, because the inexact computations can result in correct predictions. Overall, the proposed Mitch-$w8$ design produces the Top-1 accuracy that is only 0.3 percent less than the floating-point multiplier, and the Top-5 accuracy is only 0.1 percent less.

TABLE 7
The Top-1 and Top-5 Accuracies for AlexNet

	Float	Fixed	Mitch-$w6$	Mitch-$w8$
Top-1	58.3%	58.3%	58.0%	58.0%
Top-5	80.2%	80.2%	80.0%	80.1%

From the presented examples, we can deduce the cause of incorrect inferences and the desirable properties of the approximate multiplication in CNNs. Reducing the mean error of the multiplier reduces the error accumulation going across the layers, but it is not the dominant factor because the same mean error accumulation is applied to all scores. On the other hand, the dispersion of the error is much more important, because an incorrect prediction occurs when the computational errors affect two scores differently, thereby reversing their relative order.

The exact distribution of error that produces the best results for CNNs is hard to define or quantify, because it depends on each dataset and network instance. Depending on the values present in CNNs, there are error patterns that perform particularly well for a given instance. However, the range of error given by the difference between maximum and minimum possible values (or PWCE - NWCE) is proposed as a useful guideline when evaluating the approximate multipliers for CNNs. CNNs suffer performance degradation when the top choices are replaced by others due to different error accumulations, and the range of error represents the possible extent of this difference. The WCE alone is incomplete because it hides the existence of either

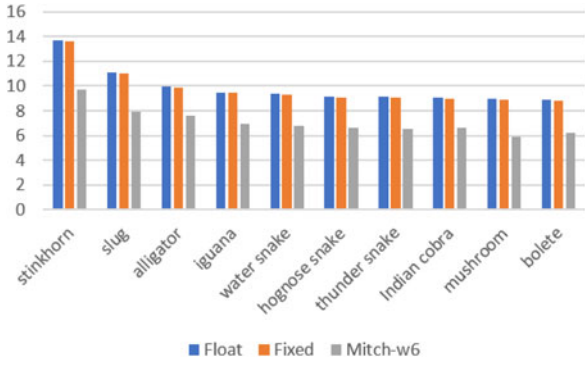

Fig. 7. The top 10 scores of a sample AlexNet inference, sorted by the highest floating-point raw scores.

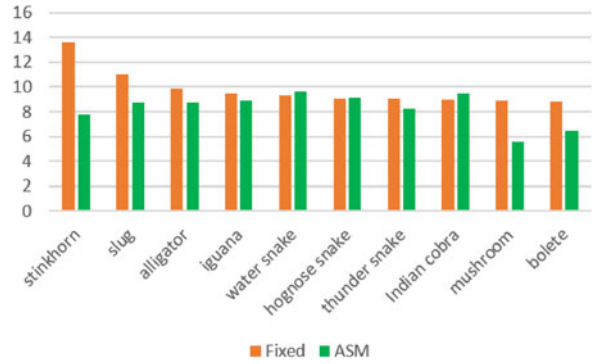

Fig. 8. The final raw scores of AlexNet of the ASM.

PWCE or NWCE and underestimates the variation of error. The standard deviation of error is not as representative as the range of error, because it is the small number of large worst errors that have the significant impact on the final outputs [33].

Fig. 8 shows the final scores of the same inference with the 1-Alphabet ASM [10]. The ASM methodology has a mean error of 1.5 percent, PWCE of 33 percent and NWCE of −47 percent, and authors reported performance degradation when the multiplier was applied to more complex datasets. In Fig. 8, some of the final scores are increased in magnitude while the others are significantly decreased, as the effect of having both high positive and negative errors. The amount of error variation is so large that the order of the scores is heavily disrupted to produce incorrect inferences. The ASM performed poorly in AlexNet and we present the accuracies in Section 8.5 where we make the comprehensive comparisons between different approximate multipliers.

8.2 Evaluation of the Truncation Parameter

To evaluate the effect of w values on CNN performance, we measured the Top-1 and Top-5 accuracies in AlexNet for various w values. Table 8 shows the Top-1 and Top-5 accuracies for increasing w. The performance improves as expected when the multiplier accuracy is improved with higher w. The performance improvement after $w = 6$ shows diminishing return so that the w truncation of our designs is justified. Proper choice of w depends on the application, dataset and network instance, and our parameterized designs can accommodate the requirements of different applications.

8.3 Evaluation of the Unbiased Designs

The experiments were performed with the ImageNet dataset to test the unbiased designs presented in Section 6.

TABLE 8
The Top-1 and Top-5 Accuracies of AlexNet with the Mitch-w Design with Varying w

	Top-1	Top-5
Mitch-w3	54.5%	78.0%
Mitch-w4	57.0%	79.8%
Mitch-w5	57.7%	79.8%
Mitch-w6	58.0%	80.0%
Mitch-w7	57.9%	80.1%
Mitch-w8	58.0%	80.1%

TABLE 9
The Effect of Unbiased Error on Top-5 Accuracy for AlexNet

	Original	Unbiased
Mitch-w5	79.8%	80.1%
Mitch-w6	80.0%	80.3%
Mitch-w7	80.1%	80.2%
Mitch-w8	80.1%	80.2%

Table 9 shows the comparisons of the Top-5 accuracies for different w. The performances were comparable, though the unbiased designs performed slightly better than the original biased designs. Fig. 9 shows the final outputs of the sample inference, also sorted from the left by the floating-point reference values. The mean error of the unbiased designs is a small positive value and the accumulated effect is evident in the final scores. We can see the same error accumulation across the layers that is applied to all scores, and the order of Top-5 is preserved while the order of top 10 shows small variations.

The CNN accuracy improves with the unbiased designs because the small mean error prevents the compression of numerical range observed with the negative mean error. There are as many as 1,000 object classes in ImageNet, and CNNs produce the final scores that are much closer together than for the simpler datasets. When the magnitudes of scores are reduced because of negative mean error, absolute differences between the scores become even smaller, and relative order between the outputs becomes more susceptible to changes. In summary, unbiasing the approximate multiplier negates the small adverse effect of negative mean errors on CNNs.

8.4 Evaluation of the 1's Complement Sign Conversion

The effect of the C1 sign handling approximation in the CNNs is discussed in this section. Table 10 shows the comparisons of the Top-5 accuracies using the C2 and the C1 sign conversions, for different values of w. We can see that both approaches produce comparable performances in AlexNet, with small drops for lower values of w. Nevertheless, employing the C1 transform reduces the energy, area, and critical path, as shown in Table 4.

As discussed in Section 5.1.3, the proposed designs with C1 produce very high relative errors with the small negative

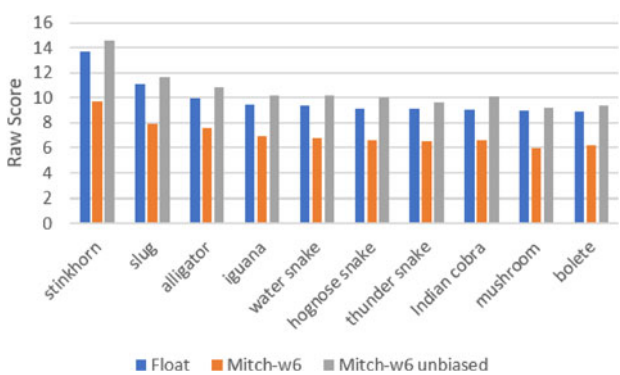

Fig. 9. The final scores of the sample inference from AlexNet with the unbiased Mitch-w.

TABLE 10
The Effect of the C1 Approximation on Top-5 Accuracy for AlexNet

	2's comp	1's comp
Mitch-w5	80.1%	80.0%
Mitch-w6	80.3%	80.2%
Mitch-w7	80.2%	80.2%
Mitch-w8	80.2%	80.2%

operands, because they are offset by the absolute value of one. The high relative error from the small operands would be unacceptable, if the application involved a chain of multiplications that makes the relative error important. However, the convolution and the fully connected layers of CNNs perform MAC operations. Each multiplication is followed by the accumulation, and the absolute error of the multiplication is accumulated instead of the relative error. This makes the corner cases of small operands have less impact on the error accumulation in the CNNs, because the results and the errors from small operands tend to be small in absolute magnitude as mentioned in Section 5.1.4.

To further support the claim, the difference in each output value from the convolutional and fully connected layers is measured between the designs using the C2 and C1 conversions. The average differences for AlexNet layers at $w = 6$ are displayed in Table 11, along with the number of accumulations for each output. Each layer has a different range of values, so the differences are taken as relative values to display the pattern more clearly. To prevent any possible confusion, this is not the same as the relative error of the multiplier, because it is normalizing the outputs of the MAC operations.

Table 11 clearly shows that the fully connected layers have higher differences than the convolutional layers. There is a sharp increase from the last convolutional layer to the first fully connected layer. One factor of the behavior is that the later layers take the outputs from the previous layers as inputs and therefore have more accumulated differences. The dominant factor, however, is the number of accumulations for each output. Each convolutional layer has a smaller number of accumulations that is the convolution kernel size, and the fully connected layers have much larger numbers as they receive all outputs from the previous layer as the inputs. The same pattern is also observed in the other CNNs, and the results are displayed in Table 12. Tables 11 and 12 not only demonstrate the process of error accumulation described in the claim, but they also show that the impact of C1 is much less than what the high relative error of the corner cases suggest. They support that the high error from the C1 conversion with the small operands do not have as much impact in the CNN values.

These tables show the differences in the outputs compared to C2, but the differences do not automatically mean the degradation in CNN performance. Table 10 already demonstrated a comparable performance. Fig. 10 shows the comparison of the top 10 scores from the sample inference, between the two sign handling techniques. We can see once again that all scores are affected at the same time so that the impact on the relative order is small. In conclusion, taking C1 for the proposed signed multiplier is a viable and cost-effective technique for the CNN computations.

TABLE 11
The Average Differences in the Outputs of Each Layer for AlexNet, between C2 and C1 Sign Handling

	Conv1	Conv2	Conv3	Conv4	Conv5	FC1	FC2	FC3
Avg. Diff(%)	8.7	7.2	9.5	12.3	6.7	15.5	15.7	15.5
Num. Accum.	121	25	9	9	9	10816	4096	4096

TABLE 12
The Average Differences in the Outputs of the Layers for the CNNs, between C2 and C1 Sign Handling

	MNIST	CIFAR	ImageNet
Conv	0.6%	1.6%	8.6%
FC	5.7%	6.9%	15.6%

8.5 Comparison Against Other Approximate Multipliers

In this section, the comparisons are made against the other approximate multipliers. We specifically compared against the 2-stage iterative log multiplier and the 1-Alphabet ASM because they had been applied to neural networks previously as discussed in Section 2, and we made the comparison to DRUM because the approximate multiplier showed very good accuracies and cost effectiveness.

Some modifications are made to the designs for the comparisons. The 2-stage iterative log multiplier presented in [9] was a pipelined design and lacked the zero detection unit so that they reported some CNN performance degradations. To compare the energy consumptions of the designs, we removed the pipeline registers and added the zero detection unit to make a fair comparison in our framework. We had observed that the addition of the zero detection unit improved the CNN performances significantly.

The 1-Alphabet ASM design presented in [10] only had 8 and 12-bit versions, but we extended the techniques to 16 and 32-bits for the comparison. Their methodology involved the retraining of the CNNs to round the unsupported values to the nearest values supported by the ASM. The retraining of the CNNs is unexplored in our work, and we instead added the rounding logic to the C++ ASM model. We did not include the rounding logic when performing the synthesis to evaluate energy and area.

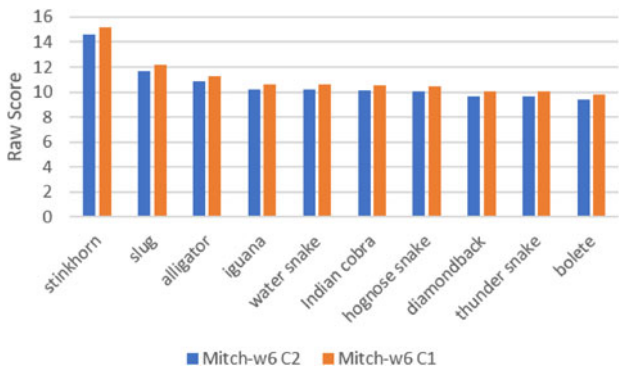

Fig. 10. The final scores for the C2 and C1 sign handling from the sample inference of AlexNet.

TABLE 13
The Comparison of the Area and Energy Against the Other Approximate Multipliers

	N = 16					N = 32				
	Fixed-point	Mitch-w8	2-Stage Iter. Log.	1-Alphabet ASM	DRUM6	Fixed-point	Mitch-w8	2-Stage Iter. Log.	1-Alphabet ASM	DRUM6
Crit. Path (ns)	2.23	1.90	3.88	2.64	2.64	3.78	2.50	4.00	4.00	3.96
Area (um^2)	2032	1135	3335	1543	1375	8627	2092	11218	7642	2917
Tot. Power (mW)	1.24	0.61	1.79	1.04	0.88	6.02	1.08	5.70	5.81	1.54
Energy (pJ)	2.77	1.16	6.95	2.75	2.32	22.76	2.70	22.80	23.24	6.10
Energy Savings	0%	58%	−151%	1%	16%	0%	88%	0%	−2%	73%

Mitch-w multipliers are unbiased and consider the C1 transform for negative numbers.

Table 13 shows the synthesis results of the approximate multipliers to compare their costs, especially the energy savings compared to the exact fixed-point multiplier. Table 14 shows the Top-1 and Top-5 accuracies in the AlexNet experiment.

The 2-stage iterative log multiplier had been proposed because it was more accurate than the original Mitchell's multiplication, but our proposed design performs as well as the iterative log multiplier in our experiments. The iterative log multiplier consumes much more power compared to the proposed design, and adding the sign handling for the CNNs negates the power and energy benefits against the exact fixed-point multiplication.

The 1-alphabet ASM showed 16 percent power savings at 16 bits and only 3 percent power savings at 32 bits, and worse energy savings. The power savings are larger for the smaller bits as the original work only presented 8 and 12-bit designs. However, the real problem when applying the ASM to CNNs is the high amount of accuracy degradation. The work on the ASM reported the trend where the performance degradation increased for more complex applications [10], and the continuation of the trend was observed for the ImageNet with a significant performance degradation. We had demonstrated earlier that the poor performance was due to the high error dispersion. The question may be raised as to why the ASMs with more alphabets were not considered. The ASM with more alphabets requires the pre-computed alphabet values and introduces the memory accesses that add significant costs to the system.

DRUM reported good characteristics and compared well against the ESSM8 and Kulkarni multipliers [19]. It was also based on the leading one detection like the log multipliers and naturally handled zero correctly, so the design received our attention though it had never been applied to a CNN. Our experiments show that DRUM6 produces the same good inference results as the proposed design, but the proposed design is significantly smaller and consumes less energy. DRUM6 had WCE of 6.3 percent that is both positive and negative, and it behaved similarly to the proposed unbiased design at $w = 8$, which has slightly higher WCE of 8.1 percent. Despite the fact that DRUM did not require the zero detection unit, the presented log multiplication algorithm was more effective at reducing the energy consumption.

TABLE 14
The Top-1 and Top-5 Accuracies of the Different Approximate Multipliers on AlexNet

	Fixed	Mitch-w8	IterLog2	ASM	DRUM6
Top-1	58.3%	58.2%	58.2%	41.6%	58.2%
Top-5	80.2%	80.2%	80.2%	67.0%	80.2%

TABLE 15
The AlexNet Accuracies with the Entire ILSVRC2012 Validation Set

	Top-1	Top-5
Float	56.8%	80.0%
Fixed	56.8%	79.9%
Mitchell's Mult	56.6%	79.7%
Mitch-w6	56.5%	79.7%
Mitch-w8	56.5%	79.7%
Unbiased Mitch-w6	56.5%	79.8%
Unbiased Mitch-w8	56.6%	79.8%
Unbiased Mitch-w6 with C1	56.5%	79.8%
Unbiased Mitch-w8 with C1	56.6%	79.8%

8.6 Verification with More Images

In this section, the proposed designs are used to perform the AlexNet inference for the entire ILSVRC2012 validation set, in order to confirm that the observations made about the design decisions hold true for the entire 50,000 images. Table 15 shows the Top-1 and Top-5 accuracies on AlexNet with the proposed designs. The original Mitchell's algorithm showed little performance degradation compared against the exact multipliers. The w truncation was introduced and decreased the Top-1 accuracy slightly but provided significant improvement in the energy consumption. The unbiasing techniques are then introduced to improve the accuracies slightly as it prevented the final score values from being compressed together. We can see that using more significant bits with $w = 8$ results in slightly higher accuracies, but the designs at $w = 6$ still perform reasonably well. Lastly, the results show that the proposed C1 approximation of the C2 sign conversion does not degrade the CNN performances, and is a more energy-efficient alternative. Table 15 shows that the ideas presented with the subset of 5,000 images remain true for the larger number of images.

9 Conclusion

In this paper, we have developed the customizable log multiplier designs based on the Mitchell's algorithm that can save significant amount of energy for CNNs. The low-energy implementation of the Mitchell multiplier was created with

the improved LOD block and the C1-based shift amount calculations, as well as the optimization of the decoder and the introduction of the zero detection unit to improve the CNN performances. We have introduced the additional approximating techniques of the w truncation and the C1 sign handling, and provided the formal analysis of the errors as well as the experimental results to show that they are viable for CNNs. We have also showed the unbiasing of the designs that provide a slight improvement to CNN performances. The proposed design at $w = 8$ consumes 88 percent less energy compared to the exact fixed-point multiplier at 32 bits, at the performance degradation of just 0.2 percent for the entire ILSVRC2012 validation dataset.

While evaluating the proposed designs for CNNs, we have also made various observations that provide deeper understanding into the effect of approximate multiplication in CNNs. Specifically, we have identified that the variations in error accumulation can impact the inference accuracies of CNNs, and suggested that the approximate multipliers should seek to minimize the range of error to perform well, though some corner cases may be tolerable. The observations presented should be helpful for future research into the topic, including the expansion to other approximate multipliers and the more contemporary CNNs.

Acknowledgments

This work has been partially supported by the Center for Pervasive Communications and Computing at UCI, the EU (FEDER) and the Spanish MINECO under grant TIN 2015-65277-R, Sao Paulo Research Foundation (FAPESP) grant 2018/00096-1 and the UCM-Banco Santander Grant PR26-16/20B-1.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst. - Vol. 1*, 2012, pp. 1097–1105.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, pp. arXiv:1409.1556, 2015.

[6] C. Szegedy, et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[7] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *Proc. 19th Asia South Pacific Des. Autom. Conf.*, 2014, pp. 201–206.

[8] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 609–622.

[9] U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks," *Neurocomput.*, vol. 96, pp. 57–65, Nov. 2012.

[10] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2016, pp. 145–150.

[11] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2016, pp. 1–7.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers, et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. 2017 ACM/IEEE 44th Annual Int. Symp. Comput. Architecture (ISCA)*, 2017, pp. 1–12.

[13] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocess. Microsyst.*, vol. 35, no. 1, pp. 23–33, Feb. 2011.

[14] V. Mahalingam and N. Ranganathan, "An efficient and accurate logarithmic multiplier based on operand decomposition," in *Proc. 19th Int. Conf. VLSI Des. Held Jointly 5th Int. Conf. Embedded Syst. Des.*, 2006, p. 6, doi: 10.1109/VLSID.2006.42.

[15] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1421–1433, Nov. 2003.

[16] W. Liu, J. Xu, D. Wang, and F. Lombardi, "Design of approximate logarithmic multipliers," in *Proc. Great Lakes Symp. VLSI*, 2017, pp. 47–52.

[17] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 4, pp. 512–517, Aug. 1962.

[18] M. S. Kim, A. A. D. Barrio, R. Hermida, and N. Bagherzadeh, "Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks," in *Proc. 23rd Asia South Pacific Des. Autom. Conf.*, 2018, pp. 617–622.

[19] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2015, pp. 418–425.

[20] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, Apr. 2015.

[21] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. IEEE Int. Conf. Electron Dev. Solid-State Circuits*, 2010, pp. 1–4.

[22] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2014, pp. 1–4.

[23] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Int. Conf. VLSI Des.*, 2011, pp. 346–351.

[24] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 6, pp. 1180–1184, Jun. 2015.

[25] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "Roba multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 2, pp. 393–401, Feb. 2017.

[26] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, Sep. 2018.

[27] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE J. Emerging Select. Top. Circuits Syst.*, vol. 8, no. 3, pp. 1–1, Sep. 2018.

[28] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[29] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[30] D. Kim, J. Kung, and S. Mukhopadhyay, "A power-aware digital multilayer perceptron accelerator with on-chip training based on approximate computing," *IEEE Trans. Emerging Top. Comput.*, vol. 5, no. 2, pp. 164–178, Apr.-Jun. 2017.

[31] Y. Shim, A. Sengupta, and K. Roy, "Low-power approximate convolution computing unit with domain-wall motion based spin-memristor for image processing applications," in *Proc. 53nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2016, pp. 1–6.

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[33] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Quality control for approximate accelerators by error prediction," *IEEE Des. Test*, vol. 33, no. 1, pp. 43–50, Feb. 2016.

Min Soo Kim received the BASc degree in engineering science from the University of Toronto, Canada, in 2008, and the MS degree in computer engineering from the University of California, Irvine, in 2011. He returned to the University of California, Irvine, in 2015 where he is currently working toward the PhD degree in computer engineering. His current research interests include the power-efficient hardware acceleration of convolutional neural networks through approximate computing, circuit design, and architectural exploration.

Alberto A. Del Barrio received the PhD degree in computer science from the Complutense University of Madrid (UCM), Madrid, Spain, in 2011. Since 2017, he has been an interim associate professor of computer science with the Department of Computer Architecture and System Engineering, UCM. His research interests include design automation, arithmetic as well as video coding optimizations.

Leonardo Tavares Oliveira is currently working toward the BSc degree in computer engineering at the Federal University of São Carlos (UFSCar), Brazil. He received a fellowship from São Paulo Research Foundation (FAPESP) for a short term research at the University of California, Irvine, in 2018 to complement and enhance his Scientific Initiation research at UFSCar. His research interests include machine learning hardware acceleration, FPGA implementation of power efficient circuitry and heterogeneous computing.

Román Hermida received the PhD degree from the Complutense University of Madrid (UCM), Spain, in 1984. He is currently a professor with the Department of Computer Architecture and Systems Engineering, UCM. His current research interests include design automation, computer architecture, reconfigurable computing, and embedded systems. He has been actively involved in program committees of several international conferences, and served as the program chair of the International Symposium on System Synthesis in 2000, and general chair of the same symposium in 2001. He received the 2002 IEEE VLSI Transactions best paper award. He has also served as Track Chair for CODES+ISSS and DATE. He is a senior member of the IEEE.

Nader Bagherzadeh received the PhD degree from the University of Texas at Austin, in 1987. He is a professor of computer engineering with the Department of Electrical Engineering and Computer Science, University of California, Irvine, where he served as a chair from 1998 to 2003. He has been involved in research and development in the areas of: computer architecture, reconfigurable computing, VLSI chip design, network-on-chip, 3D chips, sensor networks, computer graphics, memory and embedded systems. He is a fellow of the IEEE.