Low-power Implementation of Mitchell's Approximate Logarithmic Multiplication for Convolutional Neural Networks

Min Soo Kim
Dept. of EECS
University of California at Irvine, CA USA
minsk1@uci.edu

Alberto A. Del Barrio
Dept. of Computer Architecture and Automation
Universidad Complutense de Madrid, Spain
abarriog@ucm.es

Román Hermida
Dept. of Computer Architecture and Automation
Universidad Complutense de Madrid, Spain
rhermida@ucm.es

Nader Bagherzadeh
Dept. of EECS
University of California at Irvine, CA USA
nader@uci.edu

Abstract − This paper proposes a low-power implementation of the approximate logarithmic multiplier to improve the power consumption of convolutional neural networks for image classification, taking advantage of its intrinsic tolerance to error. The approximate logarithmic multiplier converts multiplications to additions by taking approximate logarithm and achieves significant improvement in power and area while having low worst-case error, which makes it suitable for neural network computation. Our proposed design shows a significant improvement in terms of power and area over the previous work that applied logarithmic multiplication to neural networks, reducing power up to 76.6% compared to exact fixed-point multiplication, while maintaining comparable prediction accuracy in convolutional neural networks for MNIST and CIFAR10 datasets.

Keywords: Approximate Multiplier, Convolutional Neural Networks, Logarithm, Power Reduction, MNIST, CIFAR-10

I. INTRODUCTION

The recent breakthroughs in multi-layer convolutional neural networks (CNNs) have caused significant progress in the applications of image classification and speech recognition. From the milestone network LeNet for the MNIST handwritten digit recognition [16], CNNs have been continually studied and improved to perform well even for the large-scale image classification. The CNNs developed in this progress, such as AlexNet and GoogLeNet [12, 15], 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-and-accumulate operations, and it is important to implement efficient multipliers as they consume large amounts of power and area [13].

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 [18] 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 [3,5,7,13]. 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 inference using the trained network data. The resource reduction, especially the power 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 on its datacenters [4].

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 [2], [9], [10], and [20] have recognized its benefits and attempted to improve it since the original proposal by John Mitchell in 1962 [1]. The original algorithm has a low worst case relative error that is proven to be as low as 11.1% [1], and this property is potentially important in neural networks that emulate firing of neurons. A large worst-case error would have a greater chance of incorrectly firing a neuron, which would lead to a larger probability of incorrect classification.

We seek to make the following contributions in this paper. Firstly, a power-efficient digital circuit implementation of the Mitchell logarithmic multiplication algorithm is presented. We explain the various optimization techniques we applied and demonstrate that our circuit design has a significant benefit in saving power and area compared to the exact multiplication and the prior state-of-the-art work on the logarithmic multiplication in neural networks. Secondly, the applicability of the logarithmic multiplier to CNN inference computation is evaluated. Our results prove that there is little to no degradation in network performance when the logarithmic multiplier is applied to the MNIST handwritten digit recognition and CIFAR-10 object recognition applications. Furthermore, a study relating the effect of the multiplications bitwidth on neural network performance is presented.

The rest of the paper is organized into the following sections. Section 2 discusses the related work, particularly the prior state of the art when applying approximate multipliers to neural network computation. Section 3 describes our implementation of the Mitchell algorithm, and Section 4 presents our evaluation of the power and area savings of the multiplier. Section 5 describes how we applied the approximate multipliers to CNNs and discusses the experimental results. Lastly, the conclusion of the paper is presented in Section 6.

II. RELATED WORK

The application of logarithmic multiplication to neural networks has been studied in [3]. This paper proposed a 2-stage pipelined iterative logarithmic multiplication, and the authors chose the iterative

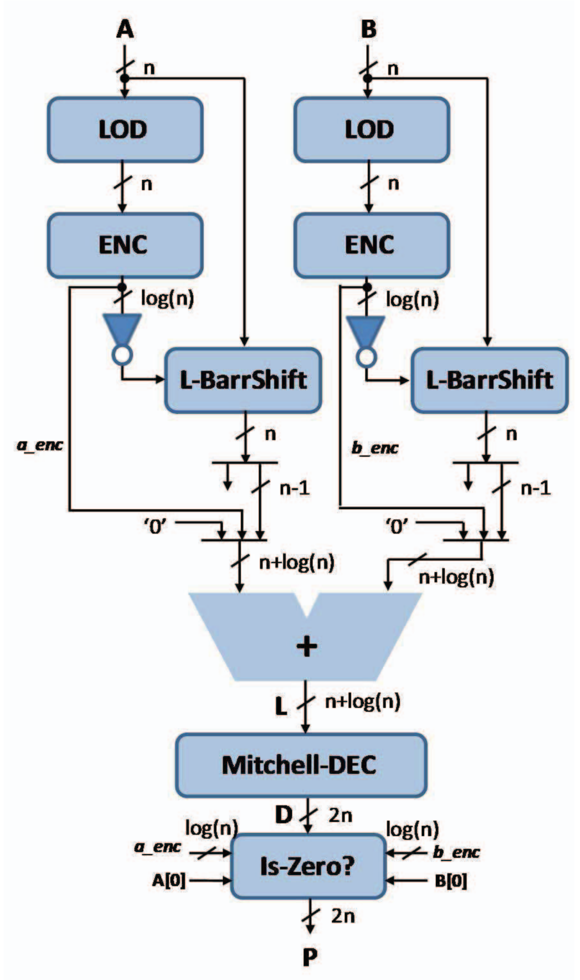

Fig. 1. Mitchell Logarithmic Multiplier according to Algorithm 1

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 to have a significant power and area savings compared to their design, and demonstrate that our design performs as well as theirs on CNNs despite the higher error rate. Throughout this paper, we present the comparison of our proposed design against their iterative multiplier.

There have been other research projects that have applied different approximate multipliers to neural networks. Mrazek et al. used a heavy search based method of Cartesian Genetic Programming (CGP) to achieve significant power reduction in CNNs [7]. They generated various approximate multipliers from the accurate multiplier by mutating the gates within the constraint of the maximum target error. In this study, the authors report an 11-bit multiplier as the largest one achievable with the CGP-based methodology, which could be too small for certain CNNs. On the other hand, while their method is a gate-level optimization, our proposal is an algorithm-level optimization; not only their work is lower level of abstraction and fundamentally different from ours, but could be considered a complimentary approach to what we have proposed in this paper. Our algorithm-level 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. has also achieved energy reduction through the gate-level approximation of Inexact Logic Minimization, where the bits are intentionally flipped to reduce logic [13]. 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, but our work does not require one because our results did not show performance degradation for the given applications. Moreover, Du et. al. did not apply their method to convolution.

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 [5]. 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 precomputing 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% drop in network accuracy in the MNIST problem. This may be an acceptable drop in some cases, but it may be a significant drop for an application that requires near-perfect accuracy. We speculate that the significant accuracy drop is due to the large worst case relative error that is more than 40%, much higher than the 11% reported by the logarithmic multipliers [1]. In fact, experiments show that our logarithmic multiplier has a low performance degradation.

III. THE APPROXIMATE MULTIPLIER

The Mitchell's algorithm and the proposed implementation is described in this section.

A. *Mitchell's Algorithm*

The first approach to logarithmic multipliers was presented by J. Mitchell in 1962 [1]. The logarithmic multiplication of two numbers $A*B$ requires converting them to the Logarithm Number System (LNS), then adding both logarithms and finally computing the antilogarithm of the result. Equation 1 represents Z, an n-bit number.

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

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

$$\begin{aligned} \log_2(Z) &= \log_2\left(2^k\left(1 + \sum_{i=j}^{k-1} 2^{i-k} z_i\right)\right) \\ &= \log_2\left(2^k(1+x)\right) \\ &= k + \log_2(1+x). \end{aligned} \tag{2}$$

The expression $log_2(1+x)$ is then approximated with x, as $\forall x \in [0,1]$ both expressions provide close results. Due to space restrictions, the rest of foundations of Mitchell's algorithm will not be described.

B. *Our implementation*

Our proposal is detailed by Algorithm 1. Fig. 1 shows the design of our logarithmic multiplier. 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.

```
Algorithm 1 (Mitchell Algorithm)
A, B: n-bits, P: 2n-bits approximate product
1. k_A = LOD(A), k_B = LOD(B)
2. x_A = A << (n – k_A –1), x_B = B << (n – k_B –1)
3. L = ('0' & k_A & x_B[n-2..0]) + ('0' & k_B & x_B[n-2..0])
4. charac=L[n+log(n)-1..n-1], mant=L[n-2..0], s=charac[log(n)]
5. IF s == '1' THEN // Large characteristic
       D = ('1' & mant) << (('0' & charac[log(n)-1..0])+1)
   ELSE // Small characteristic
       D = ('1' & mant) >> (~charac[log(n)-1..0])
6. IF Is-Zero(A,B) THEN // A or B are zero
       P = 0
   ELSE
       P = D
```

Step 1 is implemented through the Leading One Detectors (LODs) and the Encoders (ENCs). It must be noted that the LOD module produces a one-hot representation of the leading one. Hence, the encoder is composed of just a set of OR gates, instead of the priority encoder employed in [2,3]. Our LOD module leverages the implementation provided in [2,3], where the proposed 4-bits LOD blocks can be modeled with Equations 3 and 4.

$$h_j = z_j \cdot m_j, \quad 0 \le j < 4 \tag{3}$$

$$m_j = \begin{cases} 1, & j = 3 \\ \overline{z_{j+1}} \cdot m_{j+1}, & 0 \le j < 3 \end{cases} \tag{4}$$

where h_j is the j^{th} bit belonging to the hot one representation (H) of the leading one in Z. Expanding Equation 4 for a generic bitwidth n, it would be possible to obtain the expression given in Equation 5.

$$m_j = z_j \cdot \overline{\sum_{\iota=j+1}^{n-1} z_\iota}. \tag{5}$$

Using Equation 5 and constructing an or-tree in the same fashion as the Kogge-Stone adder [19] calculates the carry-in signals of an addition, it is possible to obtain Equations 6 and 7, which model our fully parallel LOD.

$$m_{i,j} = \begin{cases} z_j, & i = 0 \\ m_{i-1,j}, & i > 0, (n-1-j) < 2^{i-1} \\ m_{i-1,j} + m_{i-1,j+2^{i-1}}, & i > 0, (n-1-j) \ge 2^{i-1} \end{cases} \tag{6}$$
$$\forall i, 0 \le i \le \log(n), \forall j, 0 \le j < n$$

$$h_j = \begin{cases} z_j, & j = n-1 \\ \overline{(m_{\log(n),j+1})} \cdot z_j, & j < n-1 \end{cases} \tag{7}$$

where $\overline{(m_{\log(n),j+1})}$ is a signal that indicates whether or not there is a '0's chain at the left of z_j, h_j is the j^{th} bit belonging to the one-hot representation (H) of the leading one in Z. Finally, Equation 8 gives the value of e_i, if it is the i^{th} bit of the encoded value E.

$$e_i = \sum_{\substack{j=0 \\ (j \bmod 2^{i+1}) \ge 2^i}}^{n-1} h_j\ , \qquad \forall i, 0 \le i \le \log(n). \tag{8}$$

In order to perform Step 2 and compute the mantissas x_i, the shift amount is computed utilizing one's complement arithmetic, as n-k_i-1 is equivalent to $not(k_i)$ when n is a power of 2. Afterwards two left barrel shifters generate the mantissas, which concatenated with the corresponding characteristics compose the two operands added to compute L.

Afterwards, the result L needs to be decoded. Fig. 2 depicts the structure of our Mitchell Decoder. In Step 5 of Algorithm 1, two cases are distinguished depending on the most significant bit of L (labelled as s), which is also the most significant bit of the characteristic of L. If s='1', the mantissa of L must be shifted at least n positions to the left. That is why a 2n-bit left barrel shifter is employed for such purpose. It must be noted that when this left shifter is being used, the shift amount is always increased by 1. Hence, this shifter has been customized to always shift one extra position to the left. In this way an addition is avoided at zero cost. On the other hand, when s='0', its mantissa must be shifted to the left by n-1 positions at most, which is equivalent to shift to the right n-1-*shamt*, *shamt* being the shift amount. Besides, employing one's complement arithmetic, n-1-*shamt* is equivalent to *not*(*shamt*). As can be observed, the right barrel shifter bitwidth is just n-bits. Thus, while the least significant bits of D must be selected using a multiplexer (MUX), the most significant ones can be obtained through an AND gate with the most significant bit of L.

Finally, when any of the operands is zero, the result P must be zero. It has been demonstrated by Mrazek et al. in [7] that it is important to produce the accurate zero result when one of the operands is zero, and we implemented a zero detection unit to correctly handle it. In order to implement it, we leverage the following property: if an operand is zero, the value provided by ENC is zero and the least significant bit of the operand is '0'. This is shown in Fig. 3 and the logic within the darkened area produces valid results just after the encoded values of the one-hot representations are computed.

C. *Complexity*

TABLE I contains a detailed summary of the complexity between our proposal and the one presented in [2,3]. It must be noted that the latter is an iterative design that combines several basic blocks (BBs) to obtain different accuracies. Thus, the data shown in the table corresponds with just one BB, while our Mitchell's multiplier data correspond with the whole multiplier.

TABLE I: Complexity Comparison

	LOD	ENC	SHT	ADD	Zero
[2,3] (BB)	4-bit blocks [10]	Priority	2 Left (n)	1 (2n), 2 (log(n))	--
Ours	Parallel	Or-tree	2 Left (n), 1 Right (n) 1 Left (2n)	1 (log(n)+n)	Or-tree

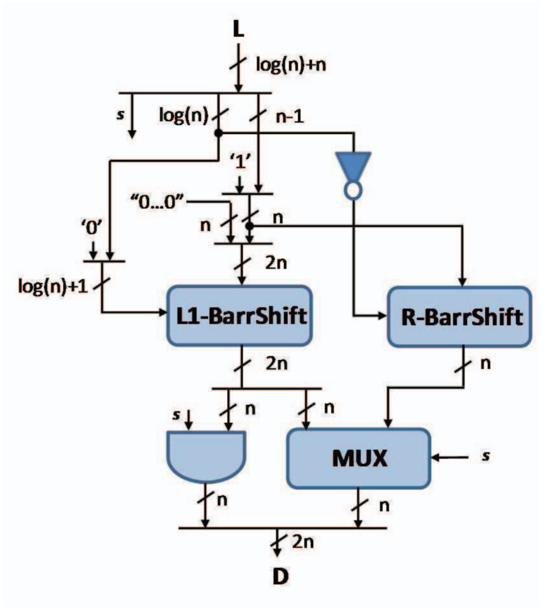

Fig. 2. Mitchell Decoder

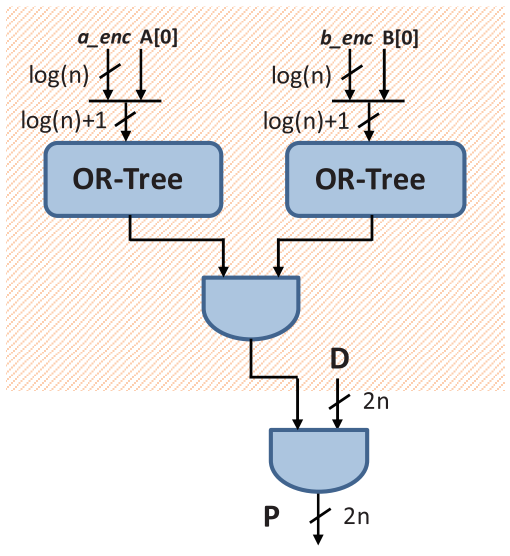

Fig. 3. Is-Zero block structure: performs both zero detection and correct result generation

TABLE II: Comparison of Power and Area after Synthesis

	8-bit			16-bit			32-bit		
	Exact	Our Design	Iterative Log	Exact	Our Design	Iterative Log	Exact	Our Design	Iterative Log
Mean Relative Error	0 %	3.77 % [9]	0.83 % [2]	0 %	3.83 % [9]	0.99 % [2]	0%	3.87 % [9]	N/A
Worst Case Relative Error	0 %	11.11% [1]	6.25 % [2]	0 %	11.11% [1]	6.25 % [2]	0%	11.11% [1]	6.25 % [2]
Cell Area (um^2)	403	312	872	1681	909	2189	6409	2161	7220
Critical Path (ns)	1.07	1.13	1.75	2.23	2.31	3.77	3.78	3.70	4.00
Total Power (mW)	0.269	0.197	0.544	1.240	0.549	1.310	6.02	1.41	4.64
Power Savings (vs Exact)	**--**	**26.8 %**	**−102.2 %**	**--**	**55.7 %**	**−5.6 %**	**--**	**76.6 %**	**22.9%**

As shown in TABLE I, the leading one detection(LOD) stage is different. While the LOD implementations in [2,3,10] are based on 4-bit blocks interconnected through carry signals, our implementation is fully parallel and therefore faster. Second, as our LOD produces a one-hot representation, the encoding phase (ENC) is realized through OR-trees instead of complex priority encoders. In terms of shifters (SHT) our proposal requires more instances of these resources, but less adders (ADD). It must be noted that in both SHT and ADD columns the size of every resource is specified among parentheses, as they vary depending on the multiplier implementation. Finally, our proposal includes a zero detection unit based on the encoded signals, whose bit width is *log(n)*, which is simpler than employing two *n*-bit comparators with zero. This unit is critical for the implementation of neural networks.

Finally, as will be shown in the experiments section, in terms of CNN accuracy the proposed logarithmic multiplier performs as well as the 2 BB, also known as the 2-stage, iterative multiplier.

IV. EVALUATION OF THE MULTIPLIERS

To evaluate the power and area benefits of the proposed design, we performed synthesis using Synopsys Design Compiler and compared it against the synthesis results of the exact fixed-point multiplier and the 2-stage iterative logarithmic multiplier. The exact multiplier is automatically synthesized by Design Compiler from the simple Verilog multiplication, and the 2-stage iterative logarithmic multiplier presented in [3] was modified to remove all pipeline registers and add the zero detection unit. 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, because we wanted to see the maximum power savings without being constrained by the timing. The 32-bit multipliers had the latencies close to 4 ns, and the synthesis tool attempted trading off area for timing with a faster clock, making the power suboptimal and disrupting the trend between the numbers of bits. The 2-stage iterative multiplier at 32 bits was time constrained and had an increased area, but we did not lower the clock speed of the experiment because of the inefficiency of the design. We also compared the referenced error rates of these multipliers.

TABLE II shows the synthesis results of the multipliers to compare their power and area. Although our optimized design has the worst error rate, it is significantly smaller and consumes less power than the other multipliers. The critical path length of the proposed design is comparable to that of the exact multiplier and clearly shorter than the critical path of the iterative design proposed in [3]. Nevertheless, in the neural networks scenario we will rather focus on the low-power design. In this sense, our logarithmic multipliers show better reduction of power and area as the number of bits increases, thus possessing better scalability. Compared to the exact fixed-point multiplier, it saves up to 76.6% of power at 32 bits and shows a clear saving of 26.8% even at 8 bits. We will demonstrate in the next section that these savings do not cause degradation in neural network performance despite the error associated with the approximation.

TABLE III: Network Description

Dataset	CNN	Layers
MNIST	LeNet	Conv[5x5] → Pooling[2x2, stride 2] → Conv→ Pooling→ Inner Product (IP) [500 output] → ReLU → IP[10 output]
CIFAR-10	Cuda-convnet	Conv[5x5] → Pooling[3x3, stride 2] → ReLU → LRN[3x3] → Conv → ReLU → Pooling → LRN → Conv → ReLU → Pooling → IP[10 output]

The proposed design achieves larger power and area savings compared to the 2-stage iterative multiplier that had been applied to the neural networks in [3]. In fact, the iterative multiplier design seems inefficient and consumes more power and area than the exact multiplier at 8 and 16 bits. We divided the 2-stage iterative logarithmic multiplier to create the 1-stage multiplier without the error correction, but it was still larger than our proposed design and consumed 0.689 mW at 16 bits, while having the worst case relative error of 25% [2]. It must be noted that the authors of [3] had a different aim than ours. They focused on pipelining the iterative design and compared it against the matrix multiplier in Xilinx Spartan 3 FPGA. Nevertheless, in this dark silicon era where we are limited primarily by power, our proposed logarithmic multiplier suits better than the state-of-the-art for the implementation of neural networks.

V. NETWORK PERFORMANCE

In this section, we discuss the classification performance of two CNNs when the approximate multipliers are applied, and analyze what is the appropriate number of bits for the neural networks. 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 [8]. We performed our experiments on the MNIST and CIFAR-10 applications. MNIST is the handwritten digit recognition dataset, and it is composed of 60,000 training examples and 10,000 test examples of handwritten digits. LeNet was the milestone CNN that was very successful with the classification of the MNIST database [16]. We used the modified version of LeNet

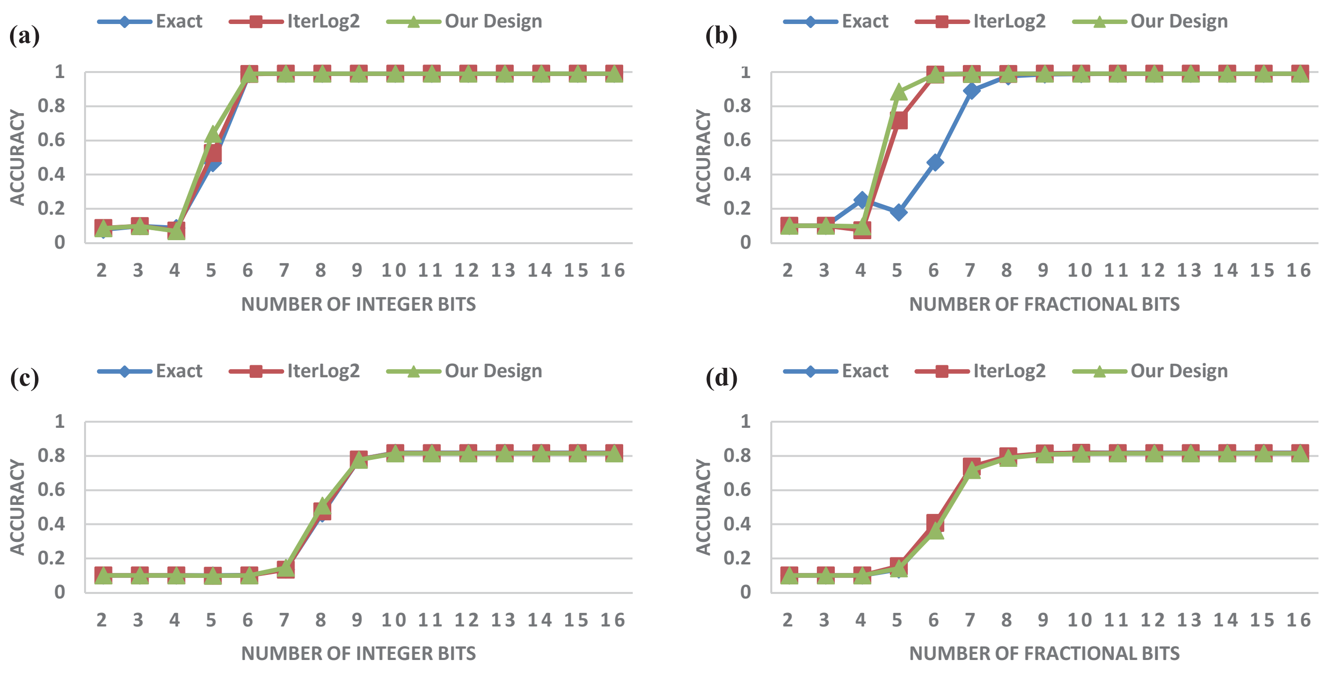

Fig. 4. The comparisons of the Top-1 classification accuracies between the multipliers, with varying number of integer and fractional fixed-point bits. The MNIST results are shown in (a) and (b), and the CIFAR-10 results are shown in (c) and (d)

TABLE IV: The comparison of Top-1 accuracies with 10 integer bits and 22 fractional bits

Application	Reference Float-point	Our Design	Exact Fixed-point	Iterative Logarithm (2-stage)
MNIST	99.02 %	99.02 %	99.02 %	99.02 %
CIFAR-10	81.43 %	81.43 %	81.89 %	81.71 %

provided in the Caffe framework that replaced the sigmoid activations with Rectified Linear Units (ReLU). CIFAR-10 [6] is the dataset for object recognition collected by the creators of AlexNet, another milestone CNN [15]. It involves recognizing the 10 objects such as airplane and automobile from 32x32 color images, and is made of 50,000 training images and 10,000 test images. The network model we used for CIFAR-10 is Alex Krizhevsky's cuda-convnet [6], which was also provided with Caffe. These two applications and the network models are suitable for quickly evaluating the applicability of our proposed design in CNNs. The details of the network models used are shown in TABLE III.

As mentioned earlier, the approximate multiplier was not suitable for the training of the networks and we trained the networks first using floating-point arithmetic. For the inference stage, we implemented the exact fixed-point multiplication, the iterative logarithmic multiplication, and the proposed low-power design in C++ using a fixed-point class library, and replaced the matrix multiplication of Caffe with our own code that calls the implemented procedures. The Top-1 CNN inference accuracy for each multiplier has been measured across all test examples of the datasets. Because the multipliers perform fixed-point arithmetic, the number of bits used is important to their precision and resource usage. Hence, the numbers of integer and fractional fixed-point bits were varied to evaluate the CNNs at different numerical ranges and precisions.

Fig. 4 shows the comparison of the Top-1 classification accuracies among the multipliers, with varying number of integer and fractional fixed-point bits. Also, TABLE IV shows the inference performance of each multiplier when 10 integer bits and 22 fractional bits are used, to show the maximum performance with sufficient number of bits. The performance of the original floating-point multiplication in Caffe is given as the reference. Fig. 4 and TABLE IV show that, given sufficient number of bits, our proposed low-power design does not cause performance degradation despite having the highest error out of the compared multipliers. The computational error did not affect the performances of the CNNs, because they are measured by the correctness of the discrete outputs. The CNNs compute the probability score of each output option and select the most probable one as the discrete output. The absolute score computed for each option is not as important as the relative order of the scores. For example, when the CNN correctly recognizes an airplane in the inference stage, it does not matter if the computed score is exactly 9 or 8; it only needs to be the highest score in order to be evaluated as one correct inference. In the CNNs used, the differences between the scores were usually large enough so that applying the approximate multiplier did not result in changing the final outputs. This property of the discrete classification makes CNNs resilient against approximations and reduced precisions of computation. However, it does not mean that any amount of error is acceptable. We can see in [5] that the large amount of error associated with the multiplier can reduce the network accuracy significantly.

Fig. 4 shows that all the multipliers caused sharp drops in the network performance when insufficient number of bits were provided for either integer or fractional parts. We found that having insufficient number of integer bits saturated too many values and produced incorrect results, while having insufficient number of fractional bits caused the loss of precision and differentiability of the values. It was also noticed that the necessary number of bits is different for each application and network; our design applied on MNIST LeNet required 6 integer bits and 8 fractional bits to reach its full potential, while it required about 10 integer and 10 fractional bits on CIFAR-10 cuda-convnet. This supports the conclusion that different network models, datasets, quantization techniques and training methods all affect the numerical values in the network, and the number of bits must adjust to the range and precision required to represent the distribution of the values. This conclusion agrees with the findings of other research such as [14], [11], [3] and [18], where different numbers of bits were required for the different neural networks. Stripes [14] in particular clearly proved the network dependency and showed the trend where the most complex networks required more bits.

There are other research projects that reported less number of bits compared to our results. The research in [14] and [11] applied dynamic quantization to each network layer instead of using a uniform quantization to reduce the number of bits required, but it would require the cost of additional circuitries and design efforts. SqueezeNet [17] and Google Tensor Processor Unit [4] used only 8 bits and significantly reduced the hardware resource, but at the cost of small performance degradation which may not be acceptable in some applications. They have their advantages as well as disadvantages, and are not counterexamples to our experiments. After all, it is not our aim to minimize the number of bits required through various techniques. Our primary aim is to show that our logarithmic multiplier can produce substantial power reduction for the CNNs without performance degradation. Our low-power design can produce significant power savings at as little as 8 bits, and the savings will be larger when more bits are required for more complex networks.

There are two other interesting observations of the experimental results. Firstly, the authors of [3] reported small performance degradation when applying the 2-stage iterative logarithmic multiplier to their neural networks, but our results did not show any. As mentioned in Section 4, we added the zero detection unit to make a fair comparison, and it is likely the cause of the performance improvement. It confirms the benefit of accurate zero multiplication in neural network computation. Secondly, as shown in TABLE IV, the exact fixed-point multiplier and the iterative logarithmic multiplier have better network performances than the reference floating-point multiplication in CIFAR-10. We can see a similar phenomenon in [7]. This is possibly due to the element of chance associated with neural network inference; the error and the lack of precision associated with those multipliers can lead to correct classifications when the exact computation would have resulted in incorrect ones. This explains the better performances of the logarithmic multipliers in Fig. 4(b) over the exact multiplier at insufficient fractional bits. In this instance of limited precision, the logarithmic multipliers with negative errors produced differently quantized values compared to the exact fixed-point multiplier and avoided producing the same incorrect outputs during convolution. This phenomenon disappeared in Fig. 4(d) for CIFAR-10 because the input and kernel values were different.

VI. CONCLUSIONS

In this paper, we have presented the optimized implementation of the logarithmic multiplication algorithm, and demonstrated that it can reduce significant amount of power for CNNs. Our logarithmic multiplier saves up to 76.6% of power compared to the exact multiplier at 32 bits, and has a low worst-case error that is less disruptive in CNNs. Our design shows a significant improvement in power reduction compared to the previous study that applied logarithmic multiplication to neural networks, and our experiments performed on MNIST and CIFAR-10 show no degradation in the image classification performance.

Furthermore, our studies have revealed that the number of bits required for CNN computation is dependent on the network. Our low-power design shows benefits at as few as 8 bits for the simplest networks, and potentially has larger power savings as more bits are required for more complex networks and strict performance requirements.

ACKNOWLEDGEMENTS

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 and the UCM-Banco Santander Grant PR26-16/20B-1.

VII. REFERENCES

[1] J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," in *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512-517, Aug. 1962.

[2] Z. Babić, A. Avramović, P. Bulić, An iterative logarithmic multiplier, Microprocessors and Microsystems, Volume 35, Issue 1, February 2011, Pages 23-33.

[3] U. Lotrič and P. Bulić. 2012. Applicability of approximate multipliers in hardware neural networks. *Neurocomput.* 96 (November 2012), 57-65.

[4] Jouppi, Norman P., et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." arXiv preprint arXiv:1704.04760 (2017).

[5] S. S. Sarwar, S. Venkataramani, A. Raghunathan and K. Roy, "Multiplier-less Artificial Neurons exploiting error resiliency for energy-efficient neural computing," *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2016, pp. 145-150.

[6] Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "The CIFAR-10 dataset." 2013-11-14]. http://www. cs. toronto. edu/~ kriz/cifar. html (2014).

[7] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, 2016, pp. 1-7.

[8] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014.

[9] V. Mahalingam and N. Ranganathan, "An efficient and accurate logarithmic multiplier based on operand decomposition," *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*, 2006, pp. 6 pp.-.

[10] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," in *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1421-1433, Nov. 2003.

[11] Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yang, H. (2016). Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*, pp. 26–35.

[12] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[13] Du, Z., Palem, K., Lingamneni, A., Temam, O., Chen, Y., & Wu, C. (2014). Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 201–206.

[14] Judd, Patrick, et al. "Stripes: Bit-serial deep neural network computing." Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016.

[15] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012

[16] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[17] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size" *arXiv preprint arXiv:1602.07360*(2016).

[18] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Temam, O. (2015). DaDianNao: A Machine-Learning Supercomputer. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO, 2015,* (January), 609–622.

[19] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," in *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786-793, Aug. 1973.

[20] Weiqiang Liu, Jiahua Xu, Danye Wang, and Fabrizio Lombardi. 2017. Design of Approximate Logarithmic Multipliers. In *Proceedings of the on Great Lakes Symposium on VLSI 2017* (GLSVLSI '17). ACM, New York, NY, USA, pp. 47-52.