IMPLEMENTATION OF LOW POWER CONSUMPTION USING MITCHELL'S APPROXIMATE LOGARITHMIC MULTIPLICATION FOR CONVOLUTIONAL MULTIPLEXER

[1]M.Renuka, Assistant Professor, Dept of ECE, Vidya Jyothi Institute of Technology, Aziz Nagar.
[2]G.Ravi Kumar, Assistant Professor, Dept of ECE, Vidya Jyothi Institute of Technology, Aziz Nagar.
[3]V.Sridhar, Assistant Professor, Dept of ECE, Vidya Jyothi Institute of Technology, Aziz Nagar.
[4]Dr.K.V.Ranga Rao, Professor, Dept.of CSE, Neil Gogte Institute of Technology, Kachwansingaram,

Abstract:

This paper proposes implementation of low power consumption using Mitchell's approximate logarithmic multiplication for convolutional multiplexer image classification, taking advantage of its intrinsic tolerance to error. The approximate logarithmic multiplier is based on the Mitchell algorithm that converts multiplications to additions in the logarithm domain 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 compared to exact fixed-point multiplication, while maintaining comparable prediction accuracy in convolutional neural networks for MNIST and CIFAR10 applications. 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, CNNs have been continually studied and improved to perform well even for the large-scale image classification.

Keywords: Mitchell's, MNIST, convolution neural networks, milestone network LeNet.

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 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 Alex Net and GoogleNet 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.

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 back propagation 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 recent 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 in the logarithm domain. 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], and [10] 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 bit width on neural network performance is presented. The rest of the paper is organized into the following sections.

II Related Work:

This paper proposed a 2-stage pipelined iterative logarithmic multiplication, and 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 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.

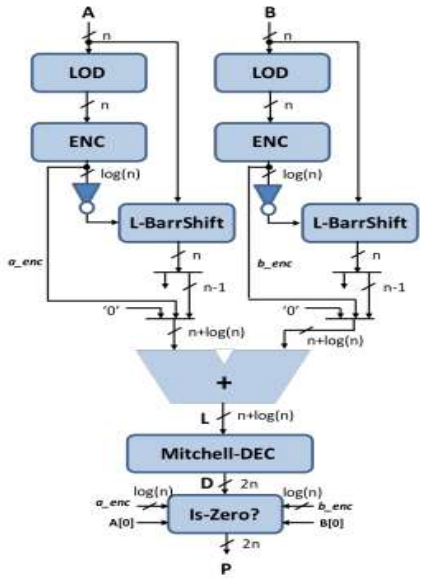

Fig. 1. Mitchell Logarithmic Multiplier according to Algorithm 1

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.

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 CNNs. 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.

This multiplier design recomputed 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 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 first 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.

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

where k is the position corresponding to the leading one, zi is a bit value at the i th position, and j depends on the number's precision (it is 0 for integer numbers). Then, the logarithm with the basis 2 of Z is then expressed by Equation 2.

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

The expression log2(1+x) is then approximated with x, as "x Î [0,1] both expressions provide close results. Due to space restrictions, the rest of foundations of Mitchell's algorithm will not be described.

Our implementation:

Our proposal is detailed by Algorithm 1. Fig. 1 shows the design of our logarithmic multiplier according to Algorithm 1. 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. kA=LOD(A), kB=LOD(B)
2. xA=A << (n – kA-1), xB= B << (n – kB-1)
3. L = ('0' & kA & xB[n-2..0]) + ('0' & kB & xB[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.

$$h_j = z_j \cdot m_j, \quad 0 \le j < 4$$

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

where is the 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.

$$m_j = z_j \cdot \overline{\sum_{i=j+1}^{n-1} z_i}.$$

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.

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

$$\forall i, 0 \le i \le \log(n), \forall j, 0 \le j < n$$

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

where nlog(n),j is a signal that indicates whether or not there is a ‘0’s chain at the left of zj, hj is the j the bit belonging to the one-hot representation (H) of the leading one in Z. Finally, Equation 8 gives the value of ect, if it is the bit of the encoded value E

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

In order to perform Step 2 and compute the mantissas xi, the shift amount is computed utilizing one’s complement arithmetic, as n-ki-1 is equivalent to not(ki) 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, being shamt the shift

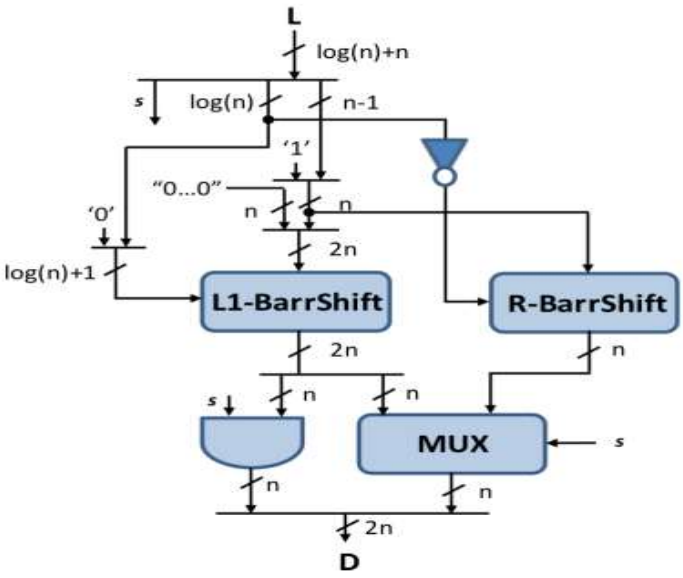

Fig 2:Mitchell decoder

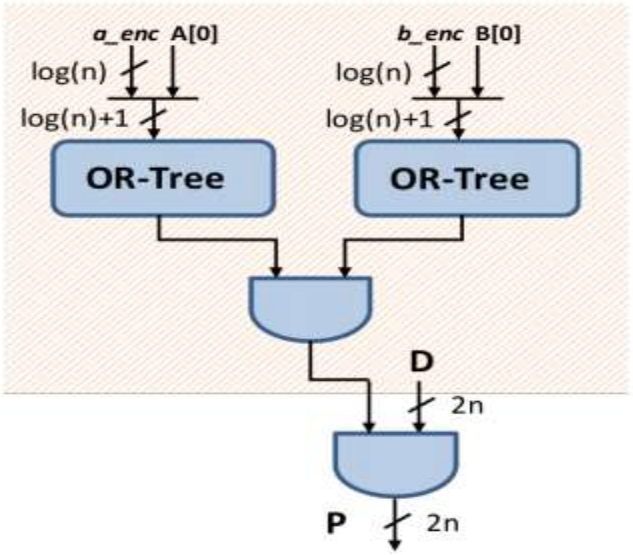

Fig 3:zero block structure :performs both zero detection and correct

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.

VI Results:

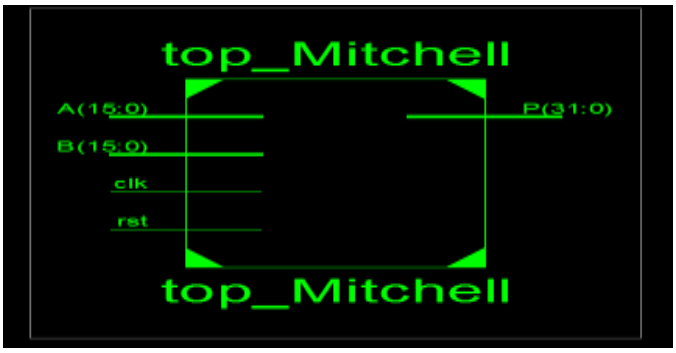

Fig 4: RTL schematic

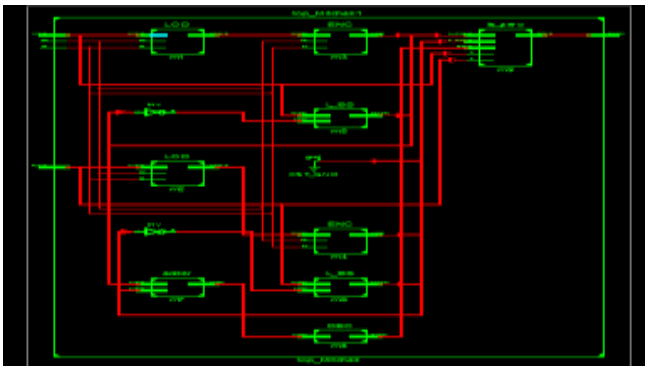

Fig 5: Technology Schematic

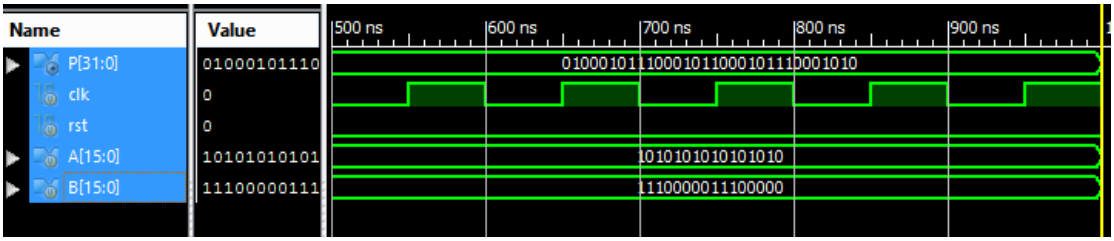

Fig 6: Simulation Results

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	154	4656	3%
Number of Slice Flip Flops	16	9312	0%
Number of 4 input LUTs	276	9312	2%
Number of bonded IOBs	66	232	28%
Number of GCLKs	1	24	4%

Fig 7: Design Summary

```
Data Path: m4/o_0 to P<6>
                            Gate     Net
  Cell:in->out      fanout  Delay    Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   FDRSE:C->Q           30   0.514   1.224  m4/o_0 (m4/o_0)
   LUT3:I0->O            2   0.612   0.383  Sh203_SW0 (N67)
   LUT4:I3->O            3   0.612   0.481  Sh183 (Sh18_bdd2)
   LUT4:I2->O            2   0.612   0.383  Sh26260 (Sh26_bdd0)
   LUT4:I3->O            1   0.612   0.426  Sh2611 (Sh26)
   LUT2:I1->O            1   0.612   0.000  m7/Madd_s_lut<10> (m7/Madd_s_lut<10>)
   MUXCY:S->O            1   0.404   0.000  m7/Madd_s_cy<10> (m7/Madd_s_cy<10>)
   MUXCY:CI->O           1   0.052   0.000  m7/Madd_s_cy<11> (m7/Madd_s_cy<11>)
   MUXCY:CI->O           1   0.052   0.000  m7/Madd_s_cy<12> (m7/Madd_s_cy<12>)
   MUXCY:CI->O           1   0.052   0.000  m7/Madd_s_cy<13> (m7/Madd_s_cy<13>)
   MUXCY:CI->O           1   0.052   0.000  m7/Madd_s_cy<14> (m7/Madd_s_cy<14>)
   MUXCY:CI->O           1   0.052   0.000  m7/Madd_s_cy<15> (m7/Madd_s_cy<15>)
   XORCY:CI->O          26   0.699   1.223  m7/Madd_s_xor<16> (L<16>)
   LUT3:I0->O            2   0.612   0.449  m8/Sh42_SW1 (N35)
   LUT3:I1->O            3   0.612   0.481  m8/Sh40 (m8/Sh40)
   LUT3:I2->O            3   0.612   0.454  m8/Sh116_SW0 (m8/Sh11220)
   LUT4:I3->O            3   0.612   0.454  m8/Sh116 (D<20>)
   LUT4:I3->O            1   0.612   0.000  m8/mux<5>211 (m8/mux<5>21)
   MUXF5:I1->O           1   0.278   0.357  m8/mux<5>21_f5 (D<5>)
   OBUF:I->O                 3.169          P_5_OBUF (P<5>)
  ----------------------------------------
  Total                     17.758ns (11.441ns logic, 6.317ns route)
                                     (64.4% logic, 35.6% route)
```

Fig 8: Timing Report

V Conclusion:

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 little as 8 bits for the simpler networks, and potentially has bigger power savings as more bits are required for more complex networks and strict performance requirements.

VI REFERANCES:

[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.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.-.

[9] V. Mahalingam and N. Ranganathan, "An efficient and accurate logarithmic multiplier based on operand decomposition," 19th International Conference on VLSI