2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia) | 978-1-6654-0857-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCE-Asia53811.2021.9641971

# FPGA-Based Approximate Multiplier for Efficient Neural Computation

Hao Zhang*, Hui Xiao*, Haipeng Qu* and Seok-Bum Ko†
*Faculty of Information Science and Engineering, Ocean University of China, China
†Department of Electrical and Computer Engineering, University of Saskatchewan, Canada

*Abstract*—**Neural networks are nowadays widely used in many fields of applications to provide better services. However, as neural networks are computation intensive, specialized hardware acceleration is needed. To make neural network computation applicable to consumer electronic devices, in this paper, field programmable gate array (FPGA) based approximate multipliers are proposed. The proposed architectures are designed based on the Karatsuba multiplication algorithm. Approximate computing method is introduced to further reduce resource and energy consumption. Two architectures are proposed where approximate computing is applied to lower order part and middle part of the multiplier, respectively. The proposed architectures are implemented in Xilinx Zynq Ultrascale+ device and up to 36% energy efficiency improvement is achieved. The proposed multipliers are applied in neural network computation and significant efficiency can be achieved with negligible accuracy degradation.**

*Index Terms*—**Approximate Multiplier, Field Programmable Gate Array, Karatsuba Algorithm, Neural Network Computation.**

## 1. Introduction

Approximate computing [1] is a new paradigm in computation that can be used to improve the performance, resource usage, and energy efficiency of certain error-resilient applications by using simplified logic. In applications, such as signal processing and image processing, a slightly degradation in result quality will usually not be distinguished by human perception and thus approximate computing can be applied to obtain better performance and energy efficiency. In the past few years, many approximate computing units, such as approximate adders [2], approximate multipliers [3], and approximate dividers [4], have been proposed in the literature.

Neural network computation is also error-resilient. On one hand, the results of the neural network prediction are usually generated by the comparison of the values of the output neurons. Therefore, as long as the ranking order of these neuron values is not changed, the neural network can still generate a correct result. On the other hand, many non-linear functions, such as ReLU and Sigmoid, are used in neural network computation. The computation results are limited within a certain range. Moreover, for applications, such as imaging captioning and image super-resolution, the result is quite subjective and more than one results can be treated as correct results. Therefore, approximate computing can be used in neural network computation for better performance and energy efficiency. In the literature, many approximate arithmetic units are designed specific for neural network computation [5] [6] [7].

To the best of our knowledge, most of the approximate units for neural network computation are designed based on application-specific integration circuit (ASIC) platform. The field-programmable gate array (FPGA) platform, due to its reconfigurable architecture, is widely used in neural network computation acceleration. On one hand, the logic density of recent FPGA devices is large enough to provide good performance for neural network computation. On the other hand, as FPGA is reconfigurable, the design can be easily modified to fit new neural network models. However, as the fundamental design elements of FPGA and ASIC are different, those ASIC based approximate units can not be directly applied in FPGA devices. Therefore, the design of FPGA specific approximate arithmetic unit for neural network computation becomes necessary.

In this paper, an FPGA based approximate multiplier architecture is proposed for efficient neural network computation. The proposed architecture is designed based on the Karatsuba algorithm [8] as it is very efficient for arithmetic units in FPGA devices. Two variants of approximate Karatsuba multipliers are proposed where the least significant part and the middle part of the multiplier are approximated, respectively. The proposed designs are implemented in Xilinx Zynq Ultrascale+ device and they show significant improvements in resource usage and energy efficiency. In addition, a case study is performed where the proposed designs are used in the computation of many popular neural network models. The results show that the proposed designs can achieve significant improvements in hardware metrics with only minor accuracy degradation.

The rest of the paper is organized as follows: Section 2 presents the Karatsuba multiplication algorithm. The proposed designs are presented in Section 3. In Section 4, the implementation results and their analysis, and the case study are presented. Finally, Section 5 concludes the whole paper.

## 2. Background

When designing large multipliers, the recursive method is usually applied. For two operands $A$ and $B$ of $2n$-bit, we can decompose them into higher order parts, $A_H$ and $B_H$, and lower order parts, $A_L$ and $B_L$:

$$\begin{aligned} A &= A_H \cdot 2^n + A_L \\ B &= B_H \cdot 2^n + B_L \end{aligned} \tag{1}$$

Thus the product of $A \times B$ can be generated by

$$A \times B = A_H B_H \cdot 2^{2n} + (A_H B_L + A_L B_H) \cdot 2^n + A_L B_L \tag{2}$$

where four multiplications are required.

In Karatsuba algorithm, the computation of the middle term $A_H B_L + A_L B_H$ is modified as

$$A_H B_L + A_L B_H = A_H B_H + A_L B_L - (A_H - A_L)(B_H - B_L) \tag{3}$$

Therefore, only three multiplications, $A_H B_H$, $A_L B_L$, and $(A_H - A_L)(B_H - B_L)$, are required for the large multiplication. However, as shown in equation (3), extra addition/subtraction and shifters are needed to compute the middle term. As addition/subtraction and shifting operation are much cheaper than multiplication in hardware, Karatsuba algorithm can still reduce the overall resource usage.

When applying approximate computing, as $A_H B_H$ is in the most significant position, its accuracy is more important for the quality of the final results. Therefore, $A_H B_H$ is not approximated. For the other two multiplications, approximate computing method is applied in this paper and their error metrics and performance in neural network computation are evaluated.

## 3. The Proposed Approximate Multipliers

In this paper, two approximate multiplier architectures are proposed. The first one, which is named AxC-KT-LSB, is the design when $A_L B_L$ is approximated. As $A_L B_L$ is also participated in the middle term computation, as shown in equation (3), the middle term is also approximated. The second one, which is named AxC-KT-MID, is the design when only $(A_H - A_L)(B_H - B_L)$ is approximated.

In the proposed design, truncation of least significant parts with error compensation is used as the approximate design method. In addition, only look-up tables (LUTs) are used which is reasonable for FPGA devices with limited amount of DSP blocks.

As 16-bit fixed-point format is enough for many neural network inference operations [9], 16-bit approximate multipliers are designed and implemented.

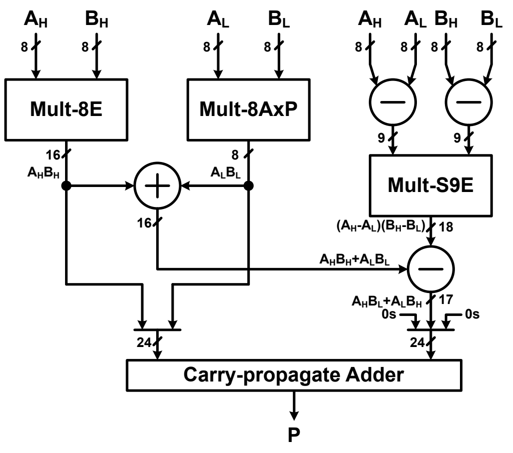
### 3.1. The Design of Small Multipliers

In order to reduce the resource usage, modified Booth multiplication [10] is used for the design of small multipliers. As the operands are divided into two parts, 8-bit or 9-bit multipliers are required for each small multiplications.

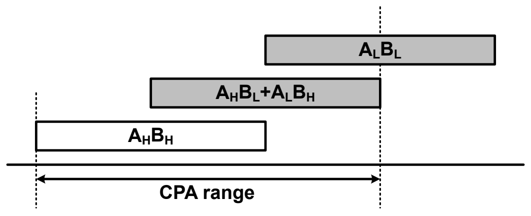
Preliminary experiments of radix-4 and radix-8 modified Booth multiplications are performed. The resource usage, delay, and power consumption are summarized in Table 1. As shown in Table 1, radix-8 Booth multiplier perform better than radix-4 Booth multiplier for 8-bit cases. Therefore, in the proposed design, radix-8 Booth multiplier is used for the three small multiplications.

TABLE 1. COMPARISON OF RADIX-4 AND RADIX-8 8-BIT BOOTH MULTIPLIER

| Design (8-bit) | Resource | | | Latency | Power |
|---|---|---|---|---|---|
| | LUTs | Registers | Carry8 | $(ns)$ | $(W)$ |
| Radix-4 Booth | 92 | 32 | 2 | 1.34 | 0.012 |
| Radix-8 Booth | 88 | 32 | 2 | 1.30 | 0.011 |



(a) Datapath of the proposed design.



(b) Diagram of the multiplication (gray parts are approximated).

Figure 1. Datapath and diagram of the proposed design when $A_L B_L$ is approximated.

### 3.2. The AxC-KT-LSB Design

The datapath of the AxC-KT-LSB is shown in Figure 1(a), where an exact 8-bit multiplier, Mult-8E, an approximate 8-bit multiplier, Mult-8AxP, and an exact 9-bit signed multiplier, Mult-S9E, are used. $A_H B_H$ and $A_L B_L$ are computed by Mult-8E and Mult-8AxP, respectively. In parallel, the subtraction between $A_H$ $(B_H)$ and $A_L$ $(B_L)$ is performed and the results are multiplied by Mult-S9E. These three parts are combined using an adder and a subtractor to generate the middle term, which is then shifted to is right position. $A_H B_H$ and $A_L B_L$ are not overlapped in bit position and can be concatenated directly. A final carry-propagate adder is used to add the concatenated two products and the middle term.

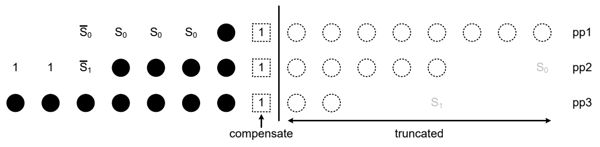


Figure 2. Approximation for the 8-bit radix-8 Booth multiplier.

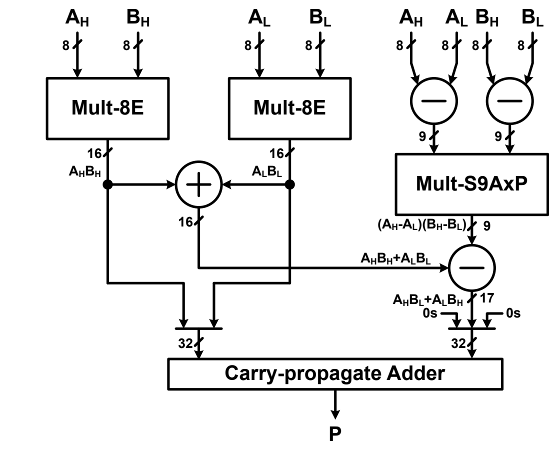


Figure 3. Datapath of the proposed design when $(A_H - A_L)(B_H - B_L)$ is approximated.

The approximate method used for the 8-bit multiplier is shown in Figure 2, where $s$ represents the sign of the partial product according to the Booth algorithm. First, those bits in the least significant 8-bit positions are truncated. This will lead to a biased negative error where the approximated value is always smaller than the exact value. In neural network computation, this error may be accumulated and affect the final result. To compensate this error, the least significant bit of the non-truncated part is set to 1. And thus, the error now becomes unbiased and during accumulation, those errors may be canceled out with each other. This will be helpful in approximate applications [11].

As the Mult-8AxP has only 8-bit effective output, the bit-width of the final carry-propagate adder can be reduced to 24-bit instead of 32-bit as shown in Figure 1(a).

The overall diagram of the multiplication is shown in Figure 1(b). As $A_L B_L$ is approximated, two parts of the multiplication are affected. As the affected positions are in the least significant part, the error of this approximate computation is expected to be small.

### 3.3. The AxC-KT-MID Design

The datapath of the AxC-KT-MID design is shown in Figure 3. The overall datapath is similar to that of the AxC-KT-LSB design, except the 9-bit multiplier is replaced with an approximated one and both 8-bit multipliers become exact designs. Therefore, the output of the second 8-bit multiplier become 16-bit. As the approximated part is in the middle position, the bit-width of the final carry-propagate adder can not be reduced.

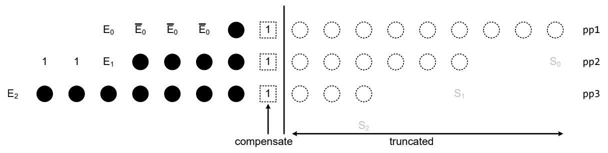


Figure 4. Approximation for the 9-bit radix-8 signed Booth multiplier.

The approximate method is shown in Figure 4, where $s$ is the sign of the partial product and $e$ represents whether the partial product and the multiplicand has the same sign according to Booth algorithm. Still, the lower half positions are truncated and errors are compensated as in AxC-KT-LSB design. For signed Booth multiplication, there is an extra sign bit $s_3$ under the last partial product and it will need an extra stage of carry save adder in partial product accumulation (assume only (3, 2) compressor is used). In the proposed design, as $s_3$ is truncated, that extra stage of carry save adder is also eliminated.

## 4. Results and Analysis

The proposed designs are implemented using Verilog and are then simulated, synthesized, and place-and-routed in Xilinx Zynq Untrascale+ (xczu9eg-ffvb1156-2-e) using Vivado 2021.1. The simulation process is performed with 1 million sets of 16-bit random vectors. Signal activity file is recorded for power analysis. The implementation results of both proposed designs and some reference designs are shown in Table 2. The reference designs include a normal 16-bit radix-8 Booth multiplier (Normal Mult), a recursive 16-bit multiplier designed with 8-bit multipliers (Recursive Mult), and exact 16-bit multiplier designed with Karatsuba algorithm (Exact-KT).

Compared to the normal multiplier and the recursive multiplier, the Exact-KT can effectively reduce the resource usage although its critical path delay is larger. The critical path goes through the right part of Figure 1(a) or Figure 3 which includes two subtractors, one 9-bit multiplier, and the carry-propagate adder. The proposed AxC-KT-LSB and AxC-KT-MID achieve 23.6% and 21.0% LUT reduction compared to the Exact-KT. As the LSB 8-bit are truncated in AxC-KT-LSB, the amount of registers is also reduced. Power consumption of both proposed designs are also reduced compared to the Exact-KT.

The AxC-KT-LSB has similar delay compared to Exact-KT as it does not improve the critical path significantly. Whereas, the AxC-KT-MID achieves smaller delay due to the approximation in the 9-bit multiplier. Due to the improvements in resource and power consumption, the proposed approximate multipliers achieves 18% (36.3%) and 17.1% (34.7%) improvements in ADP (PDP) compared to the Exact-KT.

The model of both proposed multipliers are built in MATLAB to evaluate their error metrics. The mean error distance (MED), normalized MED (NMED), and mean relative error distance (MRED) are compared as shown in

TABLE 2. Comparison of proposed designs and reference designs.

| Design | Resource | | | | Delay | Power | ADP | PDP |
|---|---|---|---|---|---|---|---|---|
| (16-bit) | CLBs | LUTs | Registers | Carry8 | ($ns$) | ($W$) | (CLB$\cdot ns$) | ($pJ$) |
| Normal Mult | 63 | 390 | 64 | 7 | 2.10 | 0.039 | 132.3 | 81.9 |
| Recursive Mult | 68 | 439 | 64 | 11 | 2.40 | 0.041 | 163.2 | 98.4 |
| Exact-KT | 61 | 376 | 64 | 16 | 3.10 | 0.033 | 189.1 | 102.3 |
| AxC-KT-LSB | 50 | 287 | 56 | 13 | 3.10 | 0.021 | 155.0 | 65.1 |
| AxC-KT-MID | 54 | 297 | 64 | 14 | 2.90 | 0.023 | 156.6 | 66.7 |

TABLE 3. Error metrics of the proposed multipliers.

| Design | MED | NMED | MRED |
|---|---|---|---|
| AxC-KT-LSB | $1.5016 \times 10^5$ | $4.50 \times 10^{-3}$ | $6.76 \times 10^{-2}$ |
| AxC-KT-MID | $3.0356 \times 10^5$ | $9.10 \times 10^{-3}$ | $9.55 \times 10^{-2}$ |

TABLE 4. Accuracy of neural network models under different computation methods.

| | AlexNet | VGG-16 | ResNet-18 | MobileNet-V2 |
|---|---|---|---|---|
| Exact-FLP32 | 91.63% | 94.12% | 95.38% | 95.58% |
| Exact-INT16 | 90.23% | 93.56% | 93.22% | 94.56% |
| AxC-16-LSB | 90.21% | 93.50% | 93.06% | 94.28% |
| AxC-16-MID | 88.56% | 90.21% | 92.58% | 92.21% |

Table 3. The AxC-KT-MID has significant larger AxC-KT-LSB.

To show the effectiveness of the proposed multipliers in neural network computation, four neural network models, AlexNet, VGG-16, ResNet-18, and MobileNet-V2, are used. They are fine-tuned for the CIFAR-10 dataset and then quantized and the proposed multipliers are then utilized. The accuracy of these models using different computation methods is shown in Table 4. Both proposed design are feasible to be used in neural network computation. AxC-KT-MID has relatively the lowest accuracy but it is still acceptable for some applications.

## 5. Conclusion

In this paper, FPGA-based approximate fixed-point multipliers are proposed for efficient neural network computation. The proposed design is based on the Karatsuba algorithm and radix-8 Booth multiplier is used for each small multiplications. Two variants of the approximate designs are proposed. Implementation results show significant resource and energy improvement. When used in neural network computation, significant energy improvement can be obtained with only minor accuracy degradation. In the future, other approximate design methods will be applied and evaluated and other machine learning algorithms will also be considered.

## Acknowledgments

The authors would like to thank Ocean University of China and the University of Saskatchewan for their financial support for this project.

## References

[1] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, “Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications,” *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec 2020.

[2] W. Liu, L. Chen, C. Wang, M. O’Neill, and F. Lombardi, “Design and Analysis of Inexact Floating-Point Adders,” *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 308–314, Jan 2016.

[3] S. Venkatachalam, E. Adams, H. J. Lee, and S.-B. Ko, “Design and Analysis of Area and Power Efficient Approximate Booth Multipliers,” *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1697–1703, Nov 2019.

[4] E. Adams, S. Venkatachalam, and S.-B. Ko, “Approximate Restoring Dividers Using Inexact Cells and Estimation From Partial Remainders,” *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 468–474, Apr 2020.

[5] M. S. Ansari, B. F. Cockburn, and J. Han, “An Improved Logarithmic Multiplier for Energy-Efficient Neural Computing,” *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 614–625, 2021.

[6] M. Ahmadinejad and M. H. Moaiyeri, “Energy- and Quality-Efficient Approximate Multipliers for Neural Network and Image Processing Applications,” *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.

[7] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, “Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2020.

[8] A. Karatsuba and Y. Ofman, “Multiplication of Many-Digital Numbers by Automatic Computers,” in *Proceedings of the USSR Academy of Sciences*, vol. 145, 1962, pp. 293–294.

[9] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, “Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–6, 2018.

[10] A. D. Booth, “A Signed Binary Multiplication Technique,” *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[11] S. Hashemi, R. I. Bahar, and S. Reda, “DRUM: A Dynamic Range Unbiased Multiplier for approximate applications,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 418–425.