International Journal on Advanced Science Engineering Information Technology

Vol.13 (2023) No. 3
ISSN: 2088-5334

Variable Precision Multiplier for CNN Accelerators Based on Booth Algorithm

Duck-Hyun Guem [a], Sunhee Kim [a,*]

[a] *Department of System Semiconductor Engineering, Sangmyung University, Chungcheongnam-do,31066, Republic of Korea*
Corresponding author: [*]happyshkim@smu.ac.kr

***Abstract*—As the utilization of CNN increases, many studies on lightweight, such as pruning, quantization, and compression, have been conducted to use CNN models in servers and edge devices. Studies have revealed that quantization greatly reduces the complexity of CNN models while lowering accuracy to a negligible level. CNN models with bit precision lowered from the existing 64/32 floating point to 16, 8, and 4 fixed points are being announced. Therefore, this paper proposes a variable precision multiplier that can select between 16 bits and 8 bits of precision. It consists of four 8-bit booth multipliers. When 16-bit multiplication is selected, the final product is calculated from four partial products, and when 8-bit multiplication is selected, four multiplications are possible simultaneously. The proposed multiplier was designed with Verilog HDL, and its function was verified in ModelSim. And it was synthesized for Altera Cyclone III EP3C16F484C6 using Quartus II 13.1.0 Web Edition. The proposed variable multiplier has increased combinational logic compared to general 8-bit/16-bit booth multipliers, and the clock speed is reduced by 65% and 82%, respectively. However, it can process four 8-bit multiplications within 1.68 times of normal 8-bit multiplication processing time and can process 16-bit multiplication within 75% of the normal 16-bit multiplication processing time. Therefore, the proposed multiplier is expected to increase speed and energy efficiency by selecting bit precision according to the layer in the CNN model.**

***Keywords*—Convolutional neural network; variable precision; booth multiplier.**

Manuscript received 8 Dec. 2022; revised 29 Jan. 2023; accepted 30 Mar. 2023. Date of publication 30 Jun. 2023.

I. INTRODUCTION

Recently, research on Convolutional Neural Networks (CNN) has been actively studied and used in various fields such as the automobile industry, medical care, and financial service industries [1], [2]. Deep learning applications using data collected from Internet of Things devices are one of them [3]–[5]. In order to prevent various problems such as network load and personal information leakage that may occur in the process of sending and receiving collected data to and from servers, the demand for edge computing that processes data on its own has been increased [6]–[8].

Generally, high-performance and high-capacity servers are required to process collected data and conduct AI learning for various applications. It may be difficult to process this process on an edge device [9], [10]. Most edge devices do not process the training process that requires processing large amounts of data and only proceed with inference. Nevertheless, the CNN layer becomes gradually deeper, and studies to reduce the amount of memory access and computation, such as pruning, quantization, and compression, have also been actively conducted [11]–[14].

Early CNN models used 64/32-bit floating points. As researchers began to study accelerators rather than GPUs, floating points were converted to fixed points to reduce computational complexity [15]–[17], and interest in quantization technology that lowered precision increased [18-20]. Many CNN models showed that if the precision is lowered, the hardware complexity is significantly reduced, and the accuracy reduction is negligible or low [21], [22]. So, different precisions are applied to each layer of CNN [23], [24], or CNN models that can select precision according to system performance are proposed [25]–[27].

CNN accelerators perform many Multiply-Accumulate (MAC) operations. CNN accelerators that can efficiently change precision for multiplication and MAC operations are being introduced [28], [29]. Therefore, this paper proposes a multiplier capable of selecting bit precision for CNN networks that require different precision for each layer. The proposed multiplier is based on the booth algorithm to increase the efficiency of multiplication operations, and multiplication can be performed by selecting one precision

between 16-bit and 8-bit multiplication. Using parallel processing technique in 8-bit unit, the processing time can be reduced compared to general 8-bit multiplication and 16-bit multiplication.

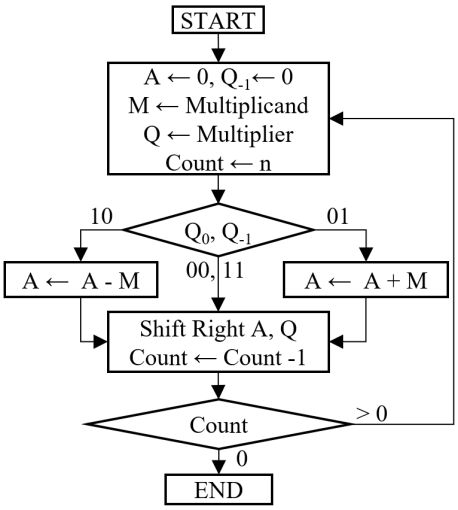
Section two of this paper presents the booth algorithm and architecture of the proposed variable precision multiplier. Section three reveals the results of verifying the multiplier designed with Verilog HDL with ModelSim and synthesizing it with FPGA. The last section of this paper is the conclusion.

II. Materials and Method

A. Booth Algorithm

Among arithmetic operators, the multiplier is an operator that requires a large area and a long calculation time due to its complicated calculation. Therefore, it is important to speed up the multiplier to improve the overall hardware performance. The booth multiplication algorithm is a multiplication algorithm faster than general multiplication methods [30]. General multiplication operations deal with unsigned numbers and partial product operations increase by the number of bits of the multiplier or by the number of 1s present in the multiplier. On the other hand, the booth algorithm deals with signed numbers, and when there are n (>=2) consecutive 1s in the multiplier, the operation for the partial product is limited to 2, which reduces the amount of operation. Therefore, we use the basic multiplier as the booth multiplier to design the variable precision multiplier.

A flowchart of the booth algorithm is shown in Fig. 1. The booth algorithm is based on the fact that when 1 occurs n times in a row ($b_n b_{n-1} \ldots b_1$), its value is equal to 2^n-1. Multiplication is performed by replacing the multiplier with -1 in the b_1 position and 2^n in the b_{n+1} position. Since b_1 is the starting point of a series of 1s, it must be compared with b_0 and similarly, b_n is the end point of a series of 1s, so it must be compared with b_{n+1}. Therefore, at the start of the calculation, a digit lower than the LSB of multiplier Q is made virtual and initialized to $Q_{-1} = 0$, and then the start, continuation, and end of 1 are distinguished by right-shifting the multiplier every clock.



n = # of bits (multiplier)

Fig. 1 Flowchart of the booth algorithm

B. Variable Precision Multiplier

The precision variable multiplier proposed in this paper is based on the Radix-2 booth algorithm. The proposed multiplier merged four 8-bit multipliers to construct a 16-bit multiplier. Depending on the selection conditions, 8-bit parallel multiplication, as well as 16-bit multiplication, is possible.

1) Signed 16-bit multiplier: The multiplication calculation for the 16-bit multiplicand and multiplier can be expressed as shown in Fig. 2. The 16-bit multiplicand and multiplier are divided into upper 8 bits and lower 8 bits, respectively, and expressed as A, B, and C, D. At this time, the multiplication of two numbers can be expressed by the following equation (1).

$$(A,B) \times (C,D) = (A \times 2^8 + B) \times (C \times 2^8 + D) = A \times C \times 2^{16} + (A \times D + B \times C) \times 2^8 + B \times D \quad (1)$$

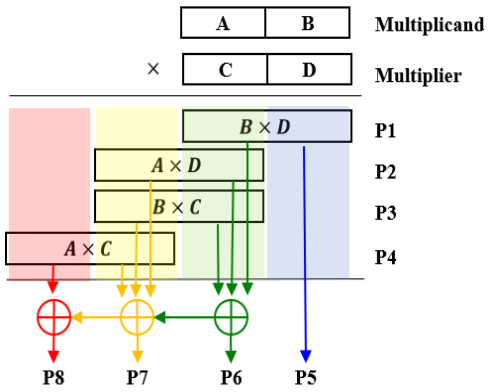


Fig. 2 Example of the multiplication operation

In Equation (1), the operand multiplied by a power of 2 can be simply expressed as a left-shift operation. The left-shift operation fills the lower bits with 0s, and 0 is the addition identity. Therefore, as shown in Fig. 2, The left-shift operation is actually not required in hardware implementation. The lower 8 bits of A×D and B×C are added with the upper 8 bits of B×D. The upper 8 bits of A×D and B×C are added to the lower 8 bits of A×C and the carry input. Finally, the upper 8 bits of A×C and the carry input signal are added. A 16-bit multiplier can be implemented using four 8-bit multipliers and 8-bit two-input, three-input, and four-input adders.

The radix-2 booth multiplication algorithm multiplies signed numbers. Therefore, when the multiplication of 16-bit signed numbers is divided into 8-bit booth multipliers, the signs of each 8-bit data must be considered. This is because the MSB of the lower 8 bits is not a sign bit.

First, A×C, which is the multiplication of the upper 8 bits, uses the booth multiplier as it is because both MSBs of A and C are sign bits. Next, consider the case where only one of the two inputs is an unsigned number, such as the case of A×D. In this case, the MSB of the unsigned number D must be checked. If the MSB of D is 0, the result of A×D multiplication may be used as it is. If the MSB of D is 1, D is determined to be a negative number in A×D booth multiplication, and the following equation is performed.

$$A \times D = A \times \left(-2^8 + abs(D)\right) = A \times (-2^8) + A \times abs(D) \quad (2)$$

D is an unsigned number, so to obtain the result of A×abs(D), A×(2^8) can be added to the multiplication calculation result. However, multiplying by 2^8 can be changed to an 8-bit shift left, and 0 is the identity of addition. Therefore, A can be added to the upper 8 bits of the multiplication result. Fig. 3 shows a block diagram of a multiplier for signed A and unsigned D.

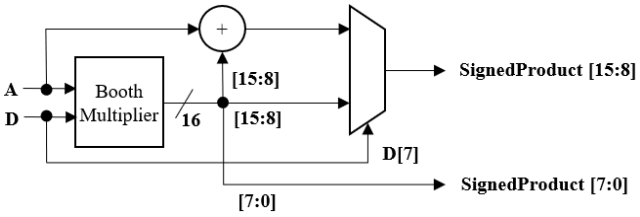


Fig. 3 Block diagram of a multiplier for signed A and unsigned D

Similarly, in the case of B×C, the MSB of B, which is an unsigned number, is checked. When it is 0, the multiplication result is used as it is, and when it is 1, C is added to the upper 8 bits of the multiplication result. B and D are the lower 8-bit data of multiplicand and multiplier, respectively, and both do not contain a sign. Therefore, when multiplying B and D, the MSB of both B and D should be checked. If the MSB of both numbers is 0, the result of the booth multiplier can be used as it is. If only the MSB of B is 1, as seen above, D can be added to the upper 8 bits of the result of the multiplier. Conversely, if only the MSB of D is 1, B can be added to the upper 8 bits of the result of the multiplier. Finally, if the MSB of both numbers is 1, the sum of B and D is added to the upper 8 bits of the multiplier result according to the following equation.

$$B \times D = \left(-2^8 + abs(B)\right) \times \left(-2^8 + abs(D)\right) = (2^{16}) + \left(abs(B) + abs(D)\right) \times (-2^8) + abs(B) \times abs(D) \quad (3)$$

Considering all these cases, the B×D multiplier can be expressed as shown in Fig. 4.

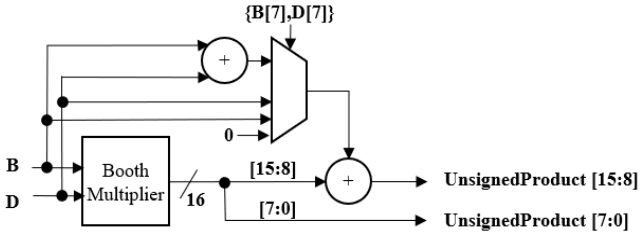


Fig. 4 Block diagram of a multiplier for unsigned B and unsigned D

Finally, in order to obtain a 16-bit signed product, four multiplication results must be added in bitwise alignment as in the multiplication of unsigned numbers. In this case, the multiplication result of B×D is an unsigned number, but the multiplication results of A×D, B×C, and A×C are signed numbers. Therefore, when adding for the upper 8-bit data of A×C, in multiplication of unsigned numbers, only the carry of addition of A×D and B×C was added, but in multiplication of signed numbers, in addition to the carry signal, after extending the sign bits of A×D and B×C to 8 bits each, the sign bits must be added together.

There are some things to consider when using the booth multiplication algorithm using two's complement. The booth multiplier compares Q_n and Q_{n-1} of multiplier Q and then adds or subtracts the multiplicand. Subtraction operations in hardware consist of adding the two's complements of the number to be subtracted. The two's complement of a number is obtained by adding 1 to the inverted value of each bit. However, due to the nature of two's complement, the two's complement of 8-bit 1000_0000_2 becomes the same value as 1000_0000_2. Therefore, when multiplicand has a value of 1000_0000_2 in the booth multiplier, the absolute value of the result is the same as the original multiplication value, but the sign is reversed. A negative result is obtained when the original result is positive, and a positive result is obtained when the original result is negative. Therefore, in the multiplier proposed in this paper, when the 8-bit multiplicand is 1000_0000_2, the sign of the output result of the booth multiplier is changed, and then the addition is performed. Fig. 5 shows a block diagram when the proposed multiplier operates in 16-bit mode.

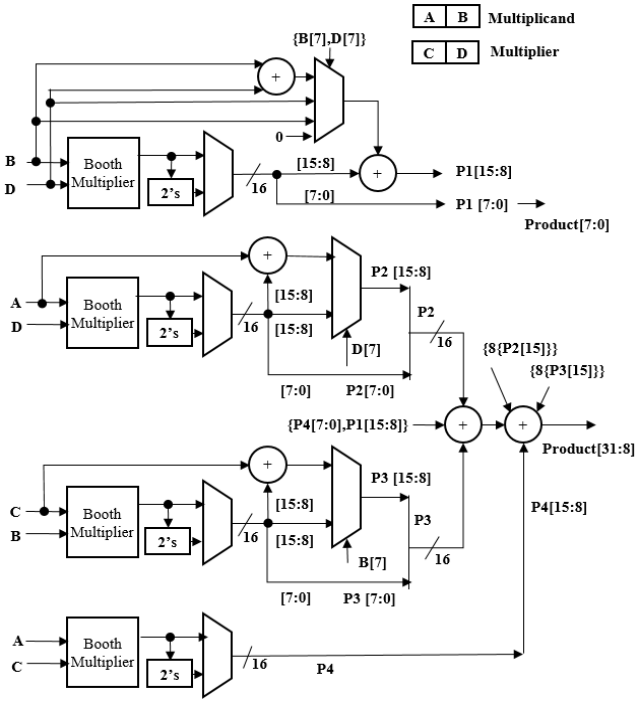


Fig. 5 Block diagram of the proposed multiplier in 16-bit mode

2) Four signed 8-bit multipliers: Since the proposed multiplier consists of four signed 8-bit booth multipliers, it can simultaneously process four multiplications for 8-bit signed multiplicand and multiplier. Unlike in the 16-bit mode, in the 8-bit mode, there is no additional process after each multiplication, and the number of bits of input and output data is changed.

First, the number of bits in the output data is increased from 32 bits to 64 bits. In 16-bit mode, the multiplication result is 32 (=16x2) bits. In the 8-bit multiplication mode, when four multiplications are performed simultaneously, there are a total of four 16 (= 8x2) bit results, so the multiplication result is 64

bits in total. Therefore, output ports must be added to operate in 8-bit multiplication mode.

The input part is again divided into two structures. The first case is when four 8-bit multipliers operate on independent data. In 16-bit mode, two 16-bit data are input. That is, four 8-bit data are input. Therefore, using the same input port as in 16-bit mode, only two independent 8-bit multipliers are possible at the same time, using only two A×C (P4) and two B×D (P1), or only two A×D (P2) and B×C (P3). To use all four 8-bit multipliers simultaneously, four 8-bits data, that is, 32-bits data are added to the input data. Depending on the number of bits of multiplicand and multiplier, the method of connecting input data to the boot multiplier is different.

The second structure of the input part is to use the same structure as the 16-bit mode. As mentioned above, four independent multiplication results cannot be obtained if 8-bit multiplication is performed with a 16-bit mode structure. This is because the multipliers and multiplicands of the two input pairs cross each other and the multiplication results in four cases are output. This operation can be usefully used in 2-D convolution or matrix operation. Fig. 6 shows an example of convolution operations that are widely used in CNNs. To obtain 3×3 a-data by convolving 5×5 C-data and 3×3 W-data, multiplication and addition are repeated while W-data moves. As in the example of Fig. 6, if 8-bit C2 and C3 are input to the 16-bit multiplicand (A, B) and 8-bit W1 and W2 are input to the 16-bit multiplier (C, D), P1, P2, P3, and P4 all become meaningful values.

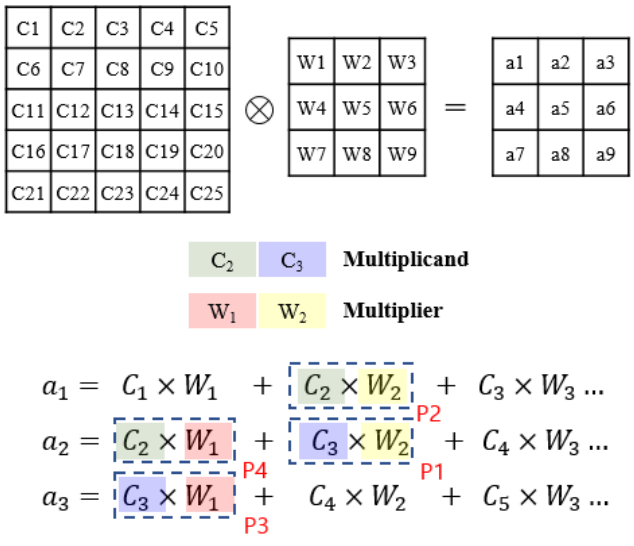


Fig. 6 Example of 2-D convolution operation

Fig. 7 shows the block diagram of the proposed multiplier. Input and output data are 64 (= 8×2×4) bits, and the multiplier can operate in 16-bit, 8-bit, independent, and 8-bit association modes depending on operation selection.

III. Result and Discussion

A. HDL design and simulation results

The proposed variable precision multiplier was designed with Verilog-HDL, and the functions were confirmed with ModelSim. Multiplication results were confirmed for all cases. Table 1 and Fig. 8 show the simulation results for several cases. It included exceptional situations to consider, and a simple test data was selected to facilitate understanding of the results.

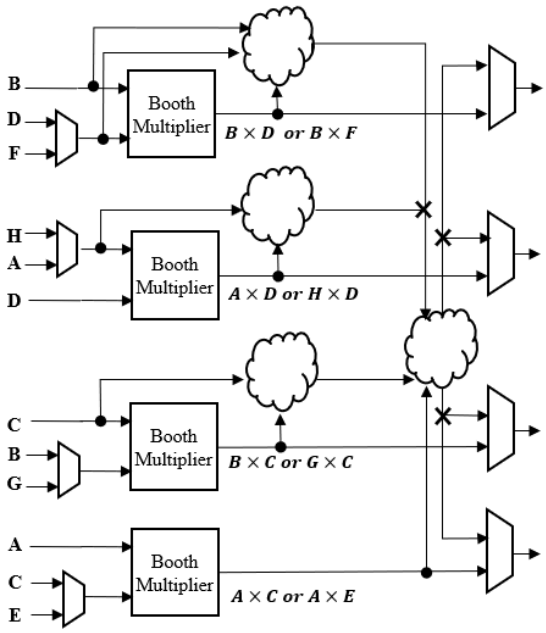


Fig. 7 Block diagram of the proposed multiplier

TABLE I
Multiplication Results for 5 Cases

Case	Multiplicand Hex (Dec)	Multiplier Hex (Dec)	Product result Hex (Dec)
1	0001 (1)	0001 (1)	0000_0001 (1)
2	0101 (257)	FFFF (-1)	FFFF_FEFF (-257)
3	FFFF (-1)	0101 (257)	FFFF_FEFF (-257)
4	80FF (-32513)	80FF (-32513)	3F01FE01 (1,057,095,169)
5	8080 (-32640)	0101 (257)	FF800080 (-8,388,480)

In cases 1-4, the multiplication of unsigned B and unsigned D was tested. In cases 1, 2, 3, and 4, the MSB of B and the MSB of D are 00_2, 01_2, 10_2, and 11_2, respectively. The processing of booth multiplication results is different for the four cases, and it is confirmed that it operates correctly in each case.

In cases 2 and 3, when unsigned D is multiplied by signed A, the MSB of D is 1 and 0, respectively. Conversely, in cases 2 and 3, when unsigned B and signed C are multiplied, the MSB of B is 0 and 1, respectively. When multiplying an unsigned number and a signed number, the process, which checks the MSB of the unsigned number and then adds the signed number to the upper bit of the multiplication result according to the MSB, is confirmed.

In case 5, the operation was confirmed when both multiplicand A and B values were 1000_0000_2. Signed A is calculated as -128, and unsigned B is calculated as +128. Finally, case 4 is the case where both multiplicand A and multiplier C are 1000_0000_2. It was confirmed that post-processing for 1000_0000_2 was performed only for multiplicand.

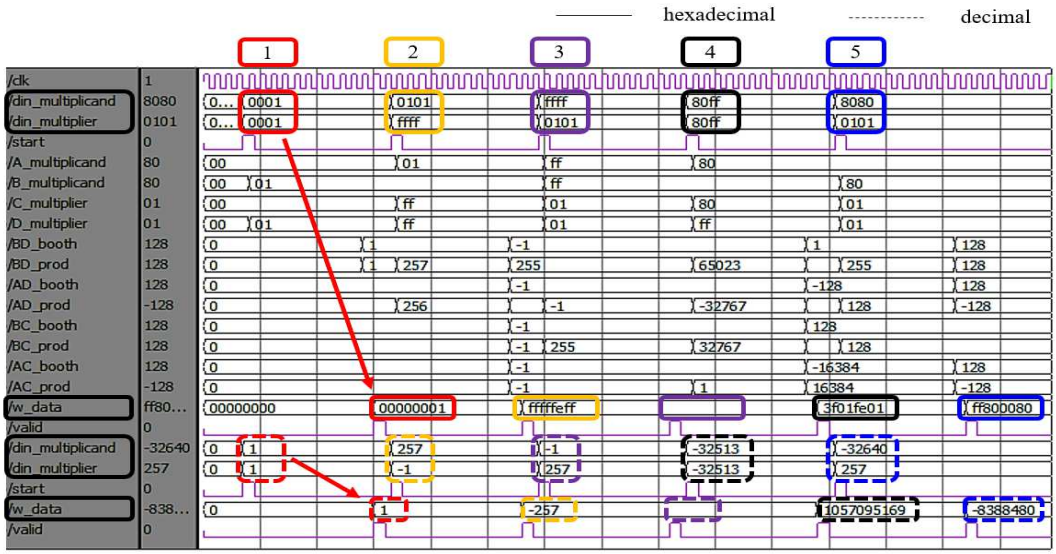


Fig. 8 Simulation results for 5 cases

In the simulation results shown in Fig. 8, the output data w_data for two input data, multiplicand, and multiplier, are marked with arrows. Moreover, the solid line is the part where each data is expressed in hexadecimal, and the dotted line is the part where the same data is expressed in decimal. Intermediate result values are shown between the input and output parts marked with a solid line, showing the results of the four booth multipliers and the modified results considering exceptional cases. Since the booth multiplier has to shift as many as the number of bits of the multiplier after the multiplication input data are given, the total latency is 11 clock cycles, including flip-flops of the input/output ports.

B. FPGA Implementation and Results

After functional verification of the Verilog-HDL design, the proposed variable precision multiplier was synthesized for Altera Cyclone III EP3C16F484C6 using Quartus II 13.1.0 Web Edition. To compare the performance of variable precision multipliers, a 16-bit booth multiplier was additionally designed.

TABLE II

COMPARISON OF IMPLEMENTATION RESULTS WITH FPGA (FAMILY: CYCLONE III, DEVICE : EP3C16F484C6, QUARTUS II VERSION : 13.1.0 WEB EDITION)

	8-bit booth	**16-bit booth**	**This paper**
Total combinational functions	67	118	327
Dedicated logic registers	48	89	199
Total logic elements	68	123	352
Total registers	48	89	199
Max clock rate (MHz)	269.61 (3.71ns)	214.87 (4.65ns)	176.77 (5.65ns)
Latency (clock cycle)	10	18	11
Latency (ns)	37.1	83.7	62.15

As can be seen from Table II, the variable precision multiplier has about four times more total registers than the 8-bit booth multiplier and about five times more total combinatorial functions. This multiplier has about twice as many total registers as 16-bit booth multipliers and about three times as many total combination functions. Since the variable precision multiplier consists of four 8-bit booth multipliers, the total register number represents a proportional relationship with a single boot multiplier. However, a single boot multiplier uses the multiplication result as the output, while a variable precision multiplier includes logic circuit that combines four multiplication results into one result. Therefore, the variable precision multiplier uses more total combinatorial functions than the single bit multiplier.

The clock speed of the proposed multiplier is slower than the 8-bit booth multiplier and the 16-bit booth multiplier. As explained above, the proposed multiplier requires logic circuit to be added after the booth multiplier operation is finished. However, comparing the latency, the 8-bit booth multiplier has a total of 37.1 ns with 10 clock cycles, the 16-bit booth multiplier has a total of 83.7 ns with 18 clock cycles, and the proposed circuit is 62.15 ns. When the proposed circuit operates in 16-bit mode, the latency is lower than that of a general 16-bit booth multiplier because it performs parallel calculations by dividing into four 8-bit multiplications. In addition, when operating in an 8-bit mode, four multiplication results can be obtained only in about 1.68 times the operating time of a general 8-bit booth multiplier.

IV. CONCLUSION

This paper proposes an 8/16 variable precision multiplier based on the boot algorithm. As the network layer of CNN becomes deeper, quantization techniques are being studied to reduce computation. Moreover, even if the precision is lowered in many CNN models, the performance is sufficiently maintained. Therefore, a variable precision multiplier suitable for CNN models with different precision for each layer was studied. By configuring four 8-bit boot multipliers in parallel, four 8-bit multiplications may be processed simultaneously.

Currently, four multipliers can obtain four multiplication results for four different data pairs. Alternatively, the two multipliers perform multiplications for two different pairs of data, and the other two multipliers perform multiplication by changing only the multiplier of the two pairs of data. In order to operate with 16-bit multiplication, the calculation is performed by dividing the multiplicand and multiplier into upper 8 bits and lower 8 bits, performing partial

multiplication, and then combining them. After designing the proposed variable precision multiplier with Verilog HDL, the operation verification was completed for all input data using ModelSim.

After it was synthesized for Altera Cyclone III EP3C16F484C6 using Quartus II 13.1.0 Web Edition, the area, and speed were compared with general 8-bit and 16-bit boot multiplication. The number of registers and combinational functions were four times and five times more than the 8-bit multiplier, respectively, and two times and three times more than the 16-bit multiplier, respectively. The combinational function is more than the value proportional to the number of bits because the proposed multiplier adds a logic circuit in the process of combining them after partial multiplication. The added combinational function makes the clock speed slower than other multipliers.

However, because it is based on parallel processing, four 8-bit multiplications can be processed within 1.68 times the processing time of one 8-bit multiplication, and 16-bit multiplication can be performed at 75% of the processing time of the 16-bit multiplier. Therefore, the proposed multiplier is expected to increase speed and energy efficiency by selecting bit precision according to the layer in the CNN model that requires different precision for each layer.

REFERENCES

[1] T. S. Alemayehu, and We. -D. Cho, "Distributed Edge Computing for DNA-Based Intelligent Services and Applications: A Review," *Journal of Information Processing Systems*, vol. 9. no. 12, pp. 291-306, 2020, doi: 10.3745/KTCCS.2020.9.12.291.

[2] J. H. Hong, K. C. Lee, and S. Y. Lee, "Trends in Edge Computing Technology," *Electronics and Telecommunications Trends*, vol. 35, no. 6, pp. 78-87, Dec. 2020, doi: 10.22648/ETRI.2020.J.350608.

[3] Y. B. Zikria, M. K. Afzal, S. W. Kim, A. Marin, and M. Guizani, "Deep learning for intelligent IoT: Opportunities, challenges and solutions," *Computer Communications*, vol. 164, pp. 50-53, Dec. 2020, doi: 10.1016/j.comcom.2020.08.017.

[4] H. Li, K. Ota and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Network*, vol. 32, no. 1, pp. 96-101, Jan.-Feb. 2018, doi: 10.1109/MNET.2018.1700202.

[5] T. Han, K. Muhammad, T. Hussain, J. Lloret and S. W. Baik, "An Efficient Deep Learning Framework for Intelligent Energy Management in IoT Networks," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3170-3179, 1 March1, 2021, doi: 10.1109/JIOT.2020.3013306.

[6] J. Kim, J. Jeon, M. Kee and G. H. Park, "The Method Using Reduced Classification Models for Distributed Processing of CNN Models in Multiple Edge Devices," *Journal of KIISE*, vol. 47, no. 08, pp. 787-792, Aug. 2020, doi: 10.5626/jok.2020.47.8.787.

[7] K. Cao, Y. Liu, G. Meng and Q. Sun, "An Overview on Edge Computing Research," *IEEE Access*, vol. 8, pp. 85714-85728, 2020, doi: 10.1109/ACCESS.2020.2991734.

[8] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655-1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.

[9] T. Sledevič, and A. Serackis, "mNet2FPGA: A Design Flow for Mapping a Fixed-Point CNN to Zynq SoC FPGA," *Electronics*, vol. 9, no. 11: 1823, 2020, doi: 10.3390/electronics9111823.

[10] M. Merenda, C. Porcaro, D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," *Sensors*, vol. 20, no. 9, 2533, 2020, doi: 10.3390/s2009253.

[11] Y. Byun, M. Ha, J. Kim, S. Lee and Y. Lee, "Low-Complexity Dynamic Channel Scaling of Noise-Resilient CNN for Intelligent Edge Devices," in *2019 DATE*, Florence, Italy, 2019, pp. 114-119, doi: 10.23919/DATE.2019.8715280.

[12] M. Shao, J. Dai, J. Kuang, and D. Meng, "A dynamic CNN pruning method based on matrix similarity," *SIViP*, vol. 15, pp. 381–389, March 2021, doi: 10.1007/s11760-020-01760-x

[13] S. K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K. R. Müller, and W. Samek, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, July 2021, 107899, doi: 10.1016/j.patcog.2021.107899.

[14] Y. Liang, L. Lu, Y. Jin, J. Xie, R. Huang, J. Zhang, and W. Lin, "An Efficient Hardware Design for Accelerating Sparse CNNs With NAS-Based Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 597-613, March 2022, doi: 10.1109/TCAD.2021.3066563.

[15] C. Y. Lo, F. C. M. Lau and C. Sham, "Fixed-Point Implementation of Convolutional Neural Networks for Image Classification," in *2018 International Conference on Advanced Technologies for Communications*, Ho Chi Minh City, Vietnam, 2018, pp. 105-109.

[16] T. Sledevič, A. Serackis, "mNet2FPGA: A Design Flow for Mapping a Fixed-Point CNN to Zynq SoC FPGA," *Electronics*, vol. 9, no. 11, 1823, 2020, doi: 10.3390/electronics9111823.

[17] Z. Nie, Z. Li, L. Wang, S. Guo, Y. Deng, R. Deng and Q. Dou, "Laius: an energy-efficient FPGA CNN accelerator with the support of a fixed-point training framework," *International Journal of Computational Science and Engineering*, vol. 21, no. 3, pp. 418-428, 2020, doi: 10.1504/IJCSE.2020.106064.

[18] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 2849–2858.

[19] Z. Bao, G. Fu, W. Zhang, K. Zhan and J. Guo, "LSFQ: A Low-Bit Full Integer Quantization for High-Performance FPGA-Based CNN Acceleration," *IEEE Micro*, vol. 42, no. 2, pp. 8-15, 1 March-April 2022, doi: 10.1109/MM.2021.3134968.

[20] M. Sailesh, K. Selvakumar, P. Narayanan, "A novel framework for deployment of CNN models using post-training quantization on microcontroller," *Microprocessors and Microsystems*, vol. 94, 104634, 2022, doi: 10.1016/j.micpro.2022.104634.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. EI-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017. [Online]. Available: https://dl.acm.org/citation.cfm?id=3122009.3242044.

[22] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W convolutional network accelerator," *IEEE Trans. Circuits Syst. Video Technol. (TCSVT)*, vo. 27, no. 11, pp. 2461–2475, Nov. 2017, doi: 10.1109/TCSVT.2016.2592330.

[23] Y. Park, Y. Kang, S. Kim, E. Kwon, and S. Kang, "GRLC: Grid-based Run-length Compression for Energy-efficient CNN Accelerator", in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, Boston, MA, USA, Aug. 2020, pp. 91-96, doi: 10.1145/3370748.3406576.

[24] D. Kim, E. Park, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Los Angeles, CA, USA, July 2018, pp. 688–698, doi: 10.1109/ISCA.2018.00063.

[25] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU:An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019. doi: 10.1109/JSSC.2018.2865489.

[26] J. Wang, S. Fang, X. Wang, J. Ma, T. Wang and Y. Shan, "High-Performance Mixed-Low-Precision CNN Inference Accelerator on FPGA," *IEEE Micro*, vol. 41, no. 4, pp. 31-38, 1 July-Aug, 2021, doi: 10.1109/MM.2021.3081735.

[27] S. -N. Tang, "Area-Efficient Parallel Multiplication Units for CNN Accelerators With Output Channel Parallelization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 3, pp. 406-410, March 2023, doi: 10.1109/TVLSI.2023.3235776.

[28] W. Liu, J. Lin, and Z. Wang, "A Precision-Scalable Energy-Efficient Convolutional Neural Network Accelerator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 10, pp.3484-3497, Oct. 2020, doi: 10.1109/TCSI.2020.2993051.

[29] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp.697-711, Dec. 2019, doi: 10.1109/JETCAS.2019.2950386.

[30] A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly *Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236-240, 1951.