An area-delay efficient Radix-8 12x12 Booth multiplier in CMOS for ML accelerator

Abhishek Choubey [1*], Shruti Bhargava Choubey [2]

[1, 2] EDP Sciences, Department of Electronics Communication Engineering , Sreenidhi Institute of science and technology, Hyderabad

Abstract. The multiplier is a significant module of graphics processing units (GPUs) and digital signal processing (DSP). These applications need low power consumption. This paper proposes a low-power radix-8 12-by-12 Booth multiplier. The proposed radix-8 Booth multiplier is implemented using an optimized Binary to 2-'s complement (B2C), convertor, and optimized multiplexer at each stage of the Booth multiplier architecture. The proposed architecture uses 23% less power and 12% less delay compared to existing architecture. To validate the results, all designs are synthesized using Cadence CMOS technology 45nm.

1 INTRODUCTION

Machine Learning (ML), Artificial Intelligence (AI), and Deep Learning (DL) have recently developed prominent technologies for a wide range of applications, including image classification. [2]-[3] and speech recognition [4], [5]. In a typical deep neural network (DNN) application, an enormous amount of multiplication computations is required. Several dedicated hardware has been proposed in the literature to accelerate the computations. The machine learning accelerator uses several multiply accumulation (MAC) for computation. In MAC computation, the multiplier consumes more power instead of the adder unit. To optimize the power of the multiplier various methods have been suggested.[1] The Booth multipliers are an effective choice when trying to find high-performance multipliers.[3] The feature of Booth multipliers is required halves of the partial products compared to other multipliers. The conventional Booth multiplier consumes more delay and power. The primary objective of this work is to reduce the power delay product of the conventional Booth multiplier. In this paper radix-8 12×12 Booth multiplier is developed which consumes less power-delay product.

In complex deep neural networks (DNNs), the convolution module functions as the key element of the AI accelerator, and thus, it requires a significant amount of power, which as a result requires a large-scale circuit for extensive processing.

For a low-power AI processor, you can optimize the circuit or modify the DNN parameters. Quantizing the weights of DNNs or their feature maps and converting them

* Corresponding author: abhishek.c@sreenidhi.edu.in

to small fixed-point forms such as 16, 8, 4, 2, or even 1 bit is a common way [1] to compress parameters. Quantization can drastically reduce the space needed for logic circuits, ON-chip memory, and AI accelerators. For some DNNs, it is not necessary to represent all weights and feature maps using the same fixed-point representations on different channels or layers. Multiplying an 8-bit weight by a feature map of 16 bits or a multiplier of 16 bits by an 8-bit weight, for example, results in significant wastes of logic resources and power. To overcome these challenges, next-generation AI accelerations require a hybrid bit convolution module with energy-efficient memories. In the literature, the approximate multiplier was also mentioned as a method that could be used to reduce the burden on AI accelerator hardware. Waris et al. [3] demonstrated AxBMs that use a radix-8 booth multiplier to reduce energy usage by up to 26.41 while maintaining an edge-detection rate of 98.45%. Jiang et. al. developed a circuit that reduces power consumption while increasing circuit size without compromising accuracy. [4] presents an approximate unsigned multiplier that uses an OR gate, and an approximate addition as an error recovery.

Ullah and others suggest an approximate design for a signed multiplier. [5] is based upon the radix-4-booth multiplier, and can reduce circuit space up to 63%. Instead of focusing on hybrid bit multiplication as with the approximation multipliers, the focus is instead on simple single-mode multiplicative. The Karatsuba multiplication is a variation of the divide and conquer multiplier with a lower computational cost [O(n log 23)]. Has a quick processing time and minimal energy requirements [6].

2 Proposed Radix-8 Booth multiplier

In Figure 1, the number of partial product rows is four. The partial product generator creates the partial product rows 0, A, 2, 3A, and 4A. To select one term, a 5:1 multiplexer with different word lengths is needed. Each row is composed of a 23-bit adder/subtractor. The subtractor's function is to generate negative partial product rows, such as -A, -2A, -3A, and -4A.

Stage 1
Inputs:

- 12-bit multiplicand x
- 3-bit least significant bits (LSBs) of the multiplier y

Process:

1) Apply the 3 LSBs of y to the encoder to control the 5:1 multiplexer, selecting between 0, x, 2x, 3x, or 4x.
2) Apply the output of the 5:1 multiplexer to the 4:1 multiplexer.
3) Control the 4:1 multiplexer using the 3 LSBs of y to choose the output from MUX 3:1, B2C, or a combination of both.
4) Extract the 2 LSBs from the output of the 4:1 multiplexer as the 2 LSBs of the final product P.
5) Combine the 2 LSBs of the 4:1 multiplexer output with the 2 LSBs of y to form the 2 LSBs of P.
6) Shift the remaining 7 bits of the 4:1 multiplexer output to the left by 2 bits.

Outputs:

- Two-bit LSBs of the final product P
- 9-bit shifted output of the 4:1 multiplexer
- Eliminating the Need for a Second Input in Stage 1

Since the LSB of y (y1) is always 0 in Stage 1, the encoder in Figure 1 can be simplified to handle only two inputs: the 12-bit multiplicand x and the 3 LSBs of the multiplier y. This eliminates the need for a second input in Stage 1.

It also controls the 4:1 multiplexer, which will determine the output from the 3:1 multiplexer, B2C. Instead of doing incremental additions to incorporate a value of 0, the two-bit least significant bit (LSB) from the output of stage-1 is directly transmitted to the two-bit LSB for the final output P. The shifter depicted in Figure 1 has the capability to be eliminated, allowing for the subsequent calculation of the output of the seven-bit stage and x to be carried out in a conventional manner during the subsequent phase.

The same architecture is used for stages 2/3/4, which includes the radix-8 Booth Encoder, a 13-bit multiplexer 3:2, and a 13-bit Adder/Subtractor. In a similar way to stage 1, 3 bits from y are used to control multiplexer 3, which selects either 0, x, or 2x for the input of the adder/subtractor. The result obtained from the previous stage provides the input for the adder/subtractor. The binary number will not increase by 2 bits in each stage, but will instead remain at 13 bits. Figure 2 shows that the last 13 bits of product P come from the output of stage four.
For the 13-bit addition/subtraction operation, a square-root carry select adder (CSLA) is utilized in order to reduce both delay and area consumption. Two pairs of ripple carry adders (RCAs) are utilized in this CSLA implementation, one for Cin = 0 and the other for Cin = 1. Following this, the output selection is carried out via a multiplexer At the beginning of the 13-bit SQRT CSA, two 1-bit adders are used. Each one can have a carry-in value of either 0 or 1. The second, third, and fourth stages consist of 2-bit, 3-bit, and 4-bit Ripple Carry Adders (RCAs), respectively. The carry-in (Cin) for each stage can be either 0 or 1. Figure 2 illustrates the typical Booth multiplier for a radix-8 implementation of 12 x 12With this multiplier, there are only four half products instead of twelve with the conventional multiplier. Every stage comprises a radix-8 booth encoding, a 15-bit adder/subtractor, and a 17-bit 3:1 multiplexer. Each stage is essentially identical from the others, and the subsequent stage's output is dependent on the outcomes of the preceding stage.

The Booth multiplier design illustrated in Figure 2 consists of four stages, with stage 1 being the primary optimization.
The proposed design of a four-staged 2 is shown in Fig 2. Booth multipliers are proposed to be made up of a Booth encoder, and a binary-to-2's complement converter (B2C), as shown in Figure 3. The B2C converter consists of two gates: a NOT gate, and an XOR. The proposed Booth multiplier has a word length of 12 bits and consists of 4 stages for partial product rows. Each stage is composed of a Mux 5:1, a Booth encoder, and an adder. Each partial product row has a word width of 12. The Mux inputs are 0, A, 2, 3A, and 4. The 12-bit adder is implemented with an RCA adder.

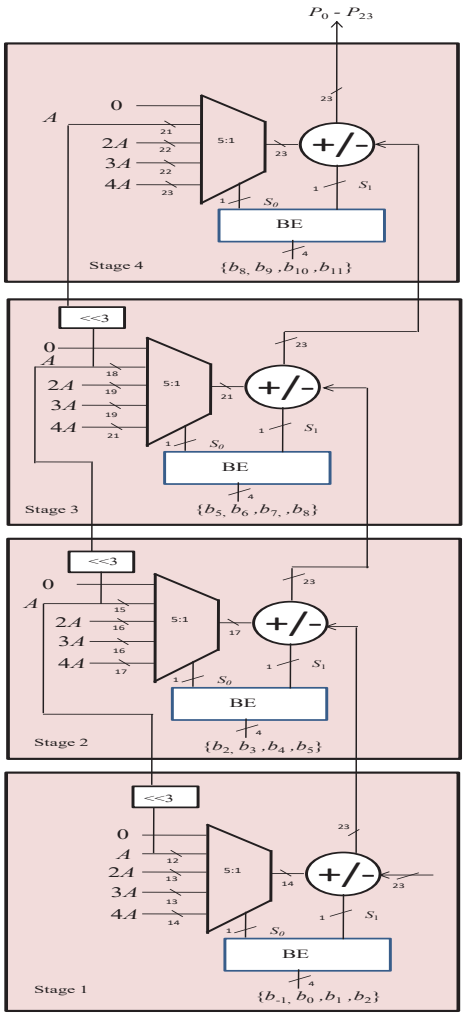

Figure 1 Conventional Radix-8 Booth multiplier

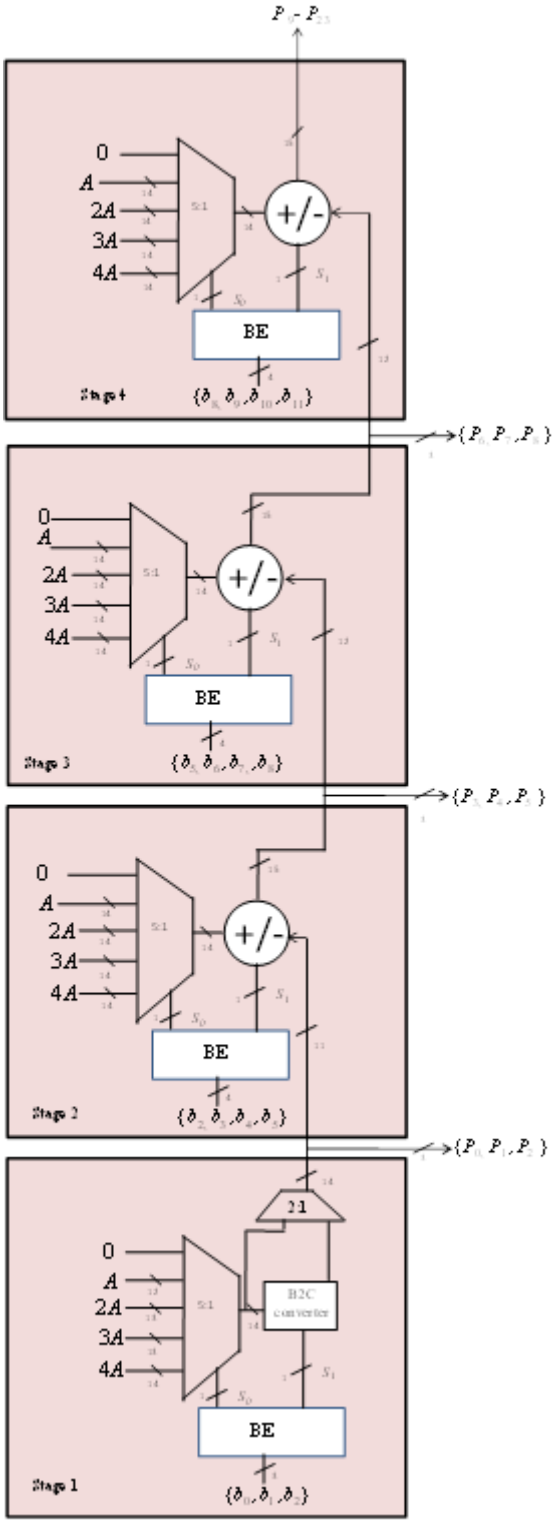

Figure 2 Proposed Radix-8 Booth multiplier

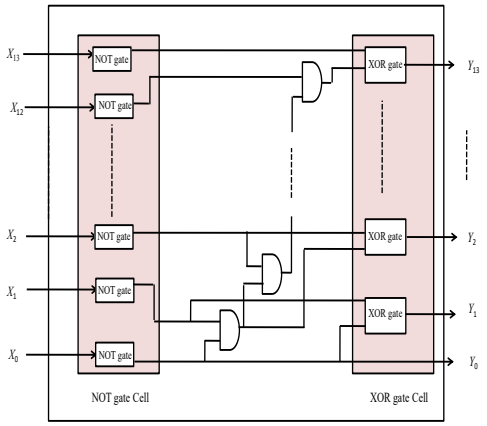

Figure 3. Proposed Radix-8 Booth Encoder

3 Result and Discussion

The radix-8 booth multiplier structure consists of one MUX, one BEU, and one Adder/subtractor unit. Hardware complexity is estimated at n=12. Adder and subtractor units are 12-bit. The proposed Booth radix-8 multiplication structure is significantly smaller and faster than the existing radix-8 structure [4]. Both the proposed multiplier as well as the conventional multiplier were coded in VHDL. Cadence reports the area, critical path delay, and power. The proposed structure had a 15.8% smaller area, 25.6 ADP, and 24.6% lower power than the conventional Booth multiplier.

Table 1. Comparison of existing and proposed design

Multiplier design	Area(μ.sqm.)	DAT(ns)	ADP (μ.sqm.s)	Power(mW)
Conventional Radix-8 multiplier	6607.9	13.97	92313.36	31.3
Proposed radix-8 multiplier	5562.36	13.17	73256.93	23.6

4 Conclusion

We propose a radix-8 Booth multiplier by optimizing the MUX complexity in this paper. A radix-8 Booth multiplication design is proposed based on the optimization of MUX. This offers a savings of 26,7% ADP compared to the conventional radix-8 multiplier design. The radix-8 Booth multiplication is used to design the ML accelerator.

REFERENCES

[1] Mohanty, B.K., Choubey, A. Efficient Design for Radix-8 Booth Multiplier and Its Application in Lifting 2-D DWT. Circuits Syst Signal Process 36, 1129–1149 (2017).
[2] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in Int. Symp. on Comp. Arch. (ISCA), 2017, p. 1–12.
[3] A. Samajdar et al., "A systematic methodology for characterizing scalability of DNN accelerators using scale-sim," in IEEE Int. Symp. on Perf. Analysis of Systems and Software (ISPASS), 2020, pp. 58–68.
[4] X. Wei et al., "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in Design Automation Conference (DAC), 2017.
[5] B. Asgari, R. Hadidi, and H. Kim, "Meissa: Multiplying matrices efficiently in a scalable systolic architecture," in IEEE Int. Conf. on Computer Design (ICCD), 2020, pp. 130–137.
[6] I. Ullah, K. Inayat, J.-S. Yang, and J. Chung, "Factored radix-8 systolic array for tensor processing," in Design Automation Conf. (DAC), 2020.
[7] C. Peltekis, D. Filippas, G. Dimitrakopoulos, C. Nicopoulos, and D. Pnevmatikatos, "ArrayFlex: A Systolic Array Architecture with Configurable Transparent Pipelining," in Design Automation and Test in Europe (DATE), 2023.
[8] R. Xu et al., "Configurable multi-directional systolic array architecture for convolutional neural networks," ACM TACO, vol. 18, no. 4, July 2021.
[9] J. Lee, J. Choi, J. Kim, J. Lee, and Y. Kim, "Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array npus," in Design Automation Conference (DAC). IEEE, 2021, pp. 247–252.
[10] Google. (2022) The bfloat16 numerical format. [Online]. Available: https://cloud.google.com/tpu/docs/bfloat16
[11] D. Filippas, C. Peltekis, G. Dimitrakopoulos, and C. Nicopoulos, "Reduced-Precision Floating-Point Arithmetic in Systolic Arrays with Skewed Pipelines," in IEEE Intern. Conference on Artificial Intelligence Circuits and Systems (AICAS), 2023.
[12] N. Jouppi et al., "Ten lessons from three generations shaped Google's tpuv4i: Industrial product," in Intern. Symp. on Computer Architecture (ISCA). IEEE, 2021, pp. 1–14.
[13] M. Andersch et al. (2022) NVIDIA Hopper architecture. [Online]. Available: https://developer.nvidia.com/blog/ Nvidia-hopper-architecture-in-depth
[14] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," Intern. Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015.
[15] S. Wimer and I. Koren, "Design flow for flip-flop grouping in data-driven clock gating," IEEE Trans. on VLSI Systems, vol. 22, no. 4, pp. 771–778, 2014.
[16] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power i/o," IEEE Trans. on VLSI systems, vol. 3, no. 1, pp. 49–58, 1995.
[17] Y. Shin, S.-I. Chae, and K. Choi, "Partial bus-invert coding for power optimization of application-specific systems," IEEE Trans. on VLSI Systems, vol. 9, no. 2, pp. 377–383, 2001.
[18] L. Benini, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Synthesis of low-overhead interfaces for power-efficient communication over wide buses," in Design Automation Conference (DAC), 1999, pp. 128–133.
[19] A. Acquaviva and R. Scarsi, "A spatially-adaptive bus interface for low switching communication," in Intern. Symp. on Low Power Electronics and Design (ISLPED), 2000, pp. 238–240.

[20] K. He et al., “Deep residual learning for image recognition,” in IEEE CVPR, 2016, pp. 770–778.
[21] A. G. Howard et al., “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv:1704.04861, 2017.
[22] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, “S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration,” in IEEE Intern. Symp. on High-Performance Computer Architecture (HPCA), 2022, pp. 573–586.