A Power-Efficient Hardware Implementation of *L-Mul*

Ruiqi Chen, Yangxintong Lyu, Han Bao, Bruno da Silva

Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel, Brussels, Belgium

Email:{ruiqi.chen, yangxintong.lyu, han.bao, bruno.da.silva}@vub.be

Abstract—Multiplication is a core operation in modern neural network (NN) computations, contributing significantly to energy consumption. The linear-complexity multiplication (*L-Mul*) algorithm is specifically proposed as an approximate multiplication method for emerging NN models, such as large language model (LLM), to reduce the energy consumption and computational complexity of multiplications. However, hardware implementation designs for *L-Mul* have not yet been reported. Additionally, 8-bit floating-point (FP8), as an emerging data format, offers a better dynamic range compared to traditional 8-bit integer (INT8), making it increasingly popular and widely adopted in NN computations. This paper thus presents a power-efficient FPGA-based hardware implementation (approximate FP8 multiplier) for *L-Mul*. The core computation is implemented using the dynamic reconfigurable lookup tables and carry chains primitives available in AMD Xilinx UltraScale/UltraScale+ technology. The accuracy and resource utilization of the approximate multiplier are evaluated and analyzed. Furthermore, the FP8 approximate multiplier is deployed in the inference phase of representative NN models to validate its effectiveness.

Index Terms—Approximate computing, multiplier, FPGA, FP8

I. INTRODUCTION

In recent years, the rapid development of neural network (NN) has brought the issue of high energy consumption to the forefront [1]. An effective approach to addressing this challenge is the use of approximate computing and quantization techniques. Approximate computing simplifies key operations in neural networks, such as multiplication and other nonlinear functions. The emerging approximate multiply method, linear-complexity multiplication (*L-Mul*) algorithm [2], has been deployed on GPUs for large language models (LLM), providing initial validation of its effectiveness. As for quantization techniques, they improve memory access and computational efficiency [3], [4]. Among these, 8-bit floating-point (FP8) numbers, compared to traditional 8-bit integer (INT8), offer better dynamic range and computational precision, making them widely adopted in neural network computations [5], [6]. To enhance the computational efficiency of FP8, specialized hardware modules for FP8 acceleration have become a new trend. For instance, designing application-specific integrated circuits (ASICs) [7], [8] and integrating corresponding units into GPUs [9].

As a fundamental computation in DNNs, various approximate multipliers have been proposed to improve efficiency and reduce energy consumption. These multipliers are designed to reduce latency, energy consumption, and area. Chen et al. [15] proposed an optimally multi-level architecture that seamlessly integrates runtime configurability with parallel module execu-

tion. An optimization strategy was applied to improve area efficiency, achieving a linear relationship with accuracy rather than the quadratic or exponential relationships seen in previous works. Ansari et al. [13] developed an 8×8 approximate multiplier tailored for NN designs by improving the design of logarithmic multipliers. HEAM [14] achieves automated design of approximate multipliers by minimizing the average error based on operand distribution and integrates these multipliers into DNN accelerators. However, the aforementioned approximation methods are primarily aimed at reducing power consumption and area utilization in ASIC implementations and may perform suboptimally on FPGAs. This is because FPGA reconfigurable logic is typically based on fixed-size lookup tables (LUTs). While FPGAs also integrate DSP hardware multiplier units, these units are physically fixed and limited in quantity. Therefore, improving the efficiency of LUT-based multiplication in terms of speed, power consumption, and resource utilization becomes particularly critical. Ullah et al. [16]–[18] proposed a series of FPGA-based approximate multipliers covering data bit-widths from 4-bit to 32-bit. More recently, their AxO series [19], [20] integrated the design of approximate multipliers into SNN accelerators. DyRecMul [21] introduced a dynamically reconfigurable INT8 approximate multiplier design, which includes a floating-point conversion unit. This design enables efficient floating-point conversion, reducing preprocessing operations and enhancing computational efficiency. Leon et al. [22] proposed a DSP-based approximate multiplier design for floating-point computations, which was integrated into a CNN accelerator. This approach achieved more efficient computation within the accelerator framework.

Although previous works have made efforts in FPGA-based approximate multiplier designs, there is still a lack of specialized approximate multipliers targeting the FP8 format. Therefore, this paper present a power-efficient hardware implementation for *L-Mul* and the main contributions include:

- We implement *L-Mul*, an approximate multiplication method for FP8 computations, on FPGA. Using LUT and carry-chain primitives, we achieve fine-grained optimization to minimize resource usage and power consumption. To the best of our knowledge, this is the first FPGA-based FP8 approximate multiplier design.
- We validate our design on AMD UltraScale/UltraScale+ devices. Compared to previous FPGA-based 8-bit approximate multiplier designs, our approach reduces resource consumption by an average of 10% while maintaining

comparable performance. Additionally, we integrate our design into a CNN accelerator, and experiments demonstrate that, among 8-bit designs, ours achieves the highest accuracy, energy efficiency, and the lowest latency.

II. PRELIMINARIES

A. FP8 Formats

FP8 is a natural progression from the FP16 representations, effectively reducing memory consumption and improving memory access and computational efficiency [6]. Compared to traditional INT8, FP8 offers a larger dynamic range (the commonly used E4M3 format, as shown in Table I and Fig. 1). Moreover, FP8 delivers good results for neural network inference processes [12]. The FP8 format adheres to IEEE-754 conventions, where a real numbers is encoded using a 1-bit sign S, an e-bit integer exponent E, and an m-bit fractional (mantissa M),

$$x_{\text{DEC}} = (-1)^S \times 2^E \times M, \tag{1}$$

where $E = e - bias$ and $M = 1 + m$. The $bias$ in this context varies with the number of bits in the exponent, and it is determined by the following formula:

$$bias = 2^{e-1} - 1. \tag{2}$$

Note that an implict 1, the $hiddenbit$, is concatenated to the fraction as an integer bit, forming the significand. A FP number with $E = 0$ has no implicit 1 in the significand so zero and subnormal values can be represented. In addition, exponent $E = 2^{e-1} - 1$ is reserved for the representation of $\pm\infty$ and $NaNs$.

TABLE I
COMPARISON OF INT8 AND FP8 (E4M3)

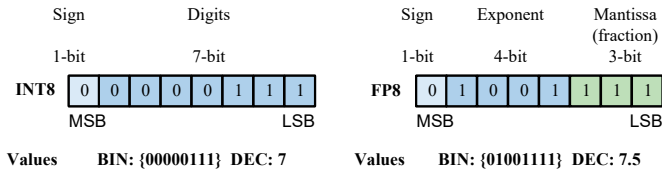| Data Type	INT8	FP8 (E4M3)
Bit Width	8 bits	8 bits
Minimum Value	-128	-448
Maximum Value	127	448
Decimal Precision	Fixed (1)	Dynamic

Fig. 1. The demonstration of the INT8 and FP8 (E4M3) defined in IEEE 754. MSB stands for most significant bit and LSB stands for least significant bit.

B. FPGA Structure

State-of-the-art FPGAs from AMD (Xilinx) and Altera (Intel) utilize basic logic cells such as 6-input LUTs, carry chains, multiplexers, and D flip-flops to implement both combinational and sequential logic circuits. The typical structure of AMD's
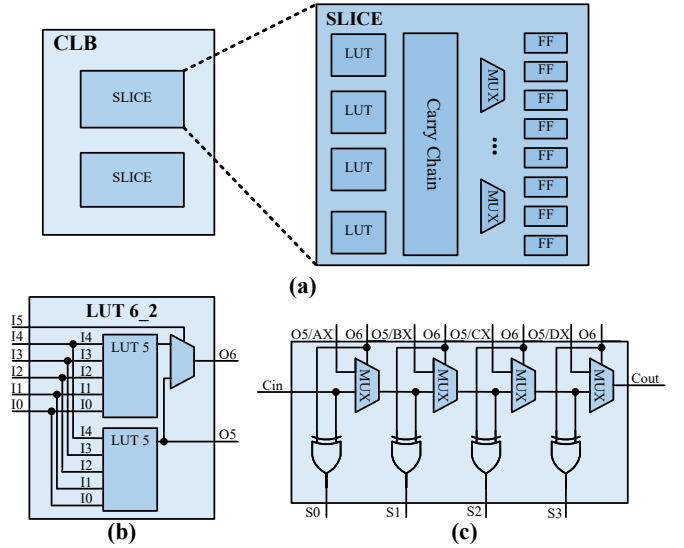

Fig. 2. Typical AMD FPGA configurable logic block (CLB) structure [10]. (a) Logic cell within the CLB. (b) LUT6 structure. (c) Carry chain structure.

FPGA is presented as an example for design implementation. However, the proposed methodology is generic and can be implemented on FPGAs from other vendors, including Altera and Anlogic, which also use 6-input LUTs, carry chains, multiplexers, and D flip-flops.

A slice in the configurable logic block (CLB) of AMD's 7-series and UltraScale/UltraScale+ FPGAs contains four 6-input LUTs (commonly referred to as LUT6_2), two 4-bit carry chains (a 8-bit carry chain within Ultra-Scale/UltraScale+), eight flip-flops and multiplexers, as shown in Fig. 2. A LUT6_2 can be used to implement either a single 6-bit combinational function, using the O6 output bit, or two 5-bit combinational functions, using the O5 and O6 output bits, as shown in Fig. 2. This is done by defining an INT value, which describes all the possible input combinations for which a logic value "1" is required at the output. For example, an INT value of 0000000000000002 (hex) for LUT6_2 defines to produce outputs O5 = 1 and O6 = 0 for input combination 100001. Besides the implementation of single 6-bit combinational functions, these LUT6_2 are also used for controlling the associated carry chain, as shown in Fig. 2. The carry chain implements a carry-lookahead adder, using O5 as the carry-generate signal and O6 as the carry-propagate signal. The carry-generate signals for the carry chain can also be provided by the external bypass signals AX – DX.

III. THE HARDWARE IMPLEMENTATION FOR L-Mul

A. L-Mul for FP8

According the introduction in Section II-A, the FP8 multiplication process can be represented as:

$$Mul(x, y) = (1 + m_x) \cdot 2^{E_x} \times (1 + m_y) \cdot 2^{E_y}$$
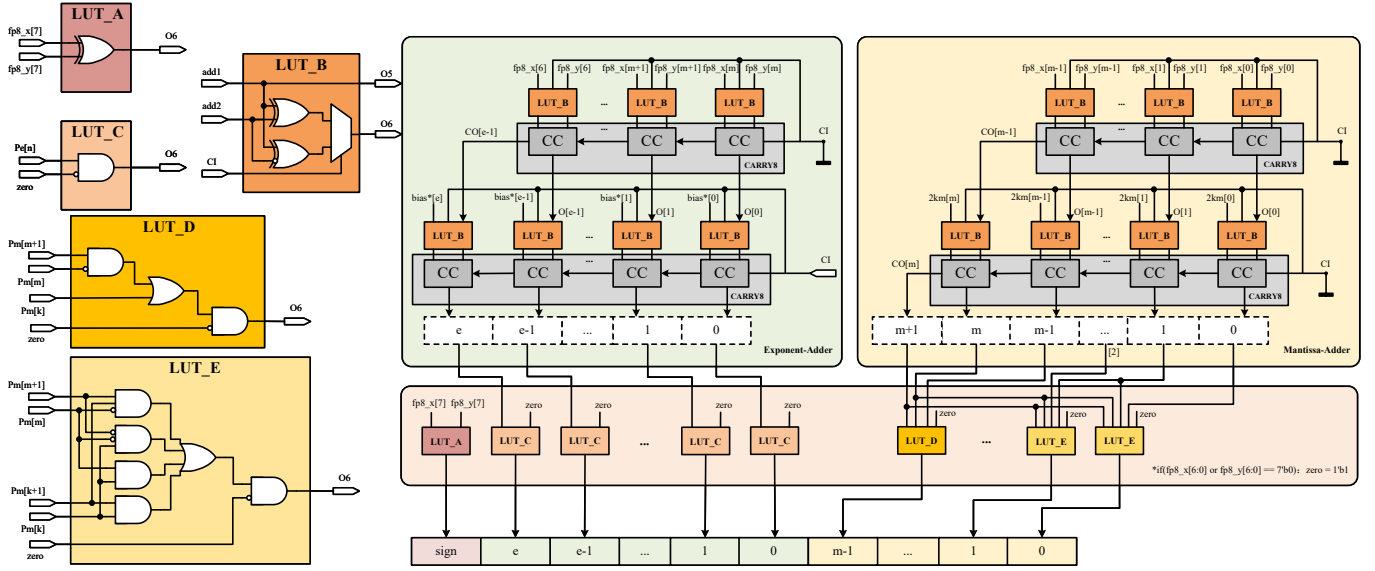$$= (1 + m_x + m_y + m_x \cdot m_y) \cdot 2^{E_x + E_y}, \tag{3}$$

Fig. 3. Hardware fine-grained architecture for *L-Mul*.

the sign bit could be omitted, as it can be handled through an XOR operation. Observing the above equation, it is evident that for hardware circuit design, only $m_x \cdot m_y$ involves a multiplication operation. The remaining operations can be implemented using addition or linear operations such as shifting. To alleviate the potential bottleneck caused by mantissa multiplication, Luo et al. [2] propose the *L-Mul* algorithm, designed to approximate the FP8 multiplication process:

$$\text{L-Mul}(x, y) = (1 + m_x + m_y + 2^{-l(m)}) \times 2^{E_x + E_y},$$
$$l(m) = \begin{cases} m & \text{if } m \leq 3, \\ 3 & \text{if } m = 4, \\ 4 & \text{if } m \geq 4. \end{cases} \quad (4)$$

Where, m represents the bit-width of the mantissa. Using this piecewise function approximation, the original multiplication operation can be transformed into shift and addition operations. In the following subsection, we introduce the corresponding fine-grained FPGA-based hardware design.

B. Hardware Design

The combination of the *L-Mul* algorithm (Eq. 4) and the FP8 format conversion relationships (Eq. 1 and Eq. 2) provides the bit-level representation of the *L-Mul* algorithm in binary operations:

$$\text{L-Mul}_{\text{BIN}}(x, y) = \left(1 + \frac{x[m-1:0] + y[m-1:0] + 2^{l(m)}}{2^m} \right)$$
$$\times 2^{x[6:m] + y[6:m] - \text{bias}_x - \text{bias}_y}. \quad (5)$$

To achieve this, we design five LUT configurations combined with carry chains to implement the *L-Mul* FP8 approximate multiplier, as shown in Fig. 3. The design primarily consists of three parts: the Exponent-Adder, the Mantissa-Adder, and the Post-Processing unit.

TABLE II
THE REPRESENTATION OF THE MANTISSA FOR DIFFERENT CARRY

[m+1,m]	Mantissa
2'b00	$1.x_m$
2'b01	$10.x_m$
2'b10	$11.x_m$
2'b11	$100.x_m$

The Exponent-Adder and Mantissa-Adder are implemented using LUT_B and CARRY8. The Exponent-Adder includes an m-bit adder and an $m+1$-bit adder, while the Mantissa-Adder includes an e-bit adder and an $e+1$-bit adder. LUT_B primarily functions as a half-adder. The output $O5$ corresponds to the sum (S). When the LSB of carry-in (CI) is 0, the operation is $O_5 = add1 \oplus add2$. Otherwise the operation is $O_5 = add1 \oplus (\sim add2)$. The output (O_6) corresponds to the C in a half-adder. In other words, $O_6 = add1 \cdot add2$. CARRY8 is used to implement addition operations. Each CARRY8 unit contains eight basic units (CC), and each CC can combine with LUT_B to function as a full adder. The CI represents the carry input from the previous stage. When the CC unit is the LSB, $CI = 0$ indicates addition, while $CI = 1$ indicates subtraction. The O corresponds to the sum (S) in the full adder, calculated as: $O = (add1 \oplus add2) \oplus CI$. In summary, a total of N LUT_B and CC units can be combined to form an N-bit adder.

The Post-Processing Unit is primarily responsible for handling the sign bit and managing the carry of the mantissa. The LUT_A is used to determine the sign bit of the product. Specifically, it processes the bit of the inputs $x[7]$ and $y[7]$. There might be a carry occur, requiring the carry value from the mantissa to be added to the exponent. For the mantissa, we follow the carry principles of typical FP multipliers, representing the mantissa in the form of $1.x_m$. The corresponding carry

FP8 Type	[m+1,m]	bias	FP8 Type	[m+1,m]	bias
E6M1	2'b00	-31	**E5M2**	2'b00	-15
	2'b11	-29		2'b11	-13
	others	-30		others	-14
E4M3	2'b00	-7	**E3M4**	2'b00	-3
	2'b11	-5		2'b11	-1
	others	-6		others	-2
E2M5	2'b00	-1	**E1M6**	2'b00	0
	2'b11	1		2'b11	2
	others	0		others	1

Data type	EP	MAE	MRE	MSE	NED
E6M1	1	2.1×10^{15}	0.319	2×10^{33}	0.001
E5M2	0.938	8.58×10^{5}	0.111	9.12×10^{13}	0.002
E4M3	0.968	141	0.068	7.56×10^{5}	0.005
E3M4	0.992	3.04	0.069	90.7	0.019
E2M5	0.997	0.991	0.072	3.23	0.076
E1M6	0.999	0.765	0.073	1.18	0.218

handling is shown in Table II. When the carry value is $2'b00$, the final product's mantissa is $P_m[m-1:0]$, and no carry is added to the exponent. When the carry value is $2'b01$, the mantissa is represented as $10.x_m$, requiring the decimal point to shift left by one position, i.e., the exponent is incremented by 1. In this case, the mantissa is $P_m[m-1:0]$. Similarly, when the carry value is $2'b10$, the exponent is incremented by 1, and the mantissa becomes $1'b1, P_m[m-1:1]$. For a carry value of $2'b11$, the exponent is incremented by 2, and the mantissa is $P_m[m-1:0]$. Since the product of 0 and any number is 0, the final product's mantissa and exponent can be expressed using the following formulas

$$P_m'[m-1:0] = \begin{cases} 0, & zero = 1 \\ \{1'b1, P_m[m-1:1]\}, & P_m[m+1:m] = 2'b10 \\ P_m[m-1:0], & others \end{cases} \tag{6}$$

$$P_e'[e:0] = \begin{cases} 0, & zero = 1 \\ P_e, & P_m[m+1:m] == 2'b00 \\ P_e + 2, & P_m[m+1:m] == 2'b11 \\ P_e + 1, & others. \end{cases} \tag{7}$$

To reduce the usage of adders, we combine the bias with various carry scenarios from Table II and treat it as a constant, $bias*$. The corresponding values are shown in Table III. LUT_C, LUT_D, and LUT_E are used to implement Equation 6, 7, and the remaining corresponding operations. These operations compute the final product's exponent bits, the highest mantissa bit, and the remaining mantissa bits excluding the highest bit, respectively.

To enhance the performance of the FP8 approximate multiplier, we implemented the hardware design using LUTs and carry chain primitives. Furthermore, to shorten the connection paths between LUTs and carry chains, we applied strict physical placement constraints. Specifically, as described in Section II-B, each carry chain can connect directly to four LUTs. Therefore, we constrained the physical placement at the CLB level, ensuring that the LUTs are connected to the carry chain within the same CLB. The input FFs are placed in adjacent CLBs to guarantee the shortest possible data paths.

IV. RESULTS AND DISCUSSION

A. Experimental setup

We first evaluate the error of *L-Mul* using five metrics: Error Probability (EP), Mean Absolute Error (MAE), Mean Relative Error (MRE), Mean Squared Error (MSE), and Normalized Error Distance (NED). For unsigned arithmetic, these metrics are defined as follows:

$$EP = \frac{1}{2^N} \sum_{i=0}^{2^N-1} ED_i \neq 0, \tag{8}$$

$$MAE = \frac{1}{2^N} \sum_{i=0}^{2^N-1} ED_i, \tag{9}$$

$$MRE = \frac{1}{2^N} \sum_{i=0}^{2^N-1} \frac{ED_i}{Exact_i}, \tag{10}$$

$$MSE = \frac{1}{2^N} \sum_{i=0}^{2^N-1} (ED_i)^2, \tag{11}$$

$$NED = \frac{1}{2^N} \sum_{i=0}^{2^N-1} \frac{ED_i}{\max(ED)}. \tag{12}$$

Then, We use Verilog for the *L-Mul* FP8 approximate multiplier design and use AMD Vivado 2022.2 for logic synthesis and placement constraints. The design is deployed and validated on the ZCU104 Evaluation Kit of UltraScale+ FPGA. We perform multiple synthesis iterations, applying different critical path constraints in each iteration to implement each design multiple times. This approach ensures accurate measurements of area and maximum operating frequency. The Vivado simulator and power analysis tools are used to calculate dynamic power consumption. As this is the first FPGA-based FP8 approximate multiplier design, we ensure fairness by selecting previous FPGA-based INT8 approximate multipliers for comparisons of resource consumption, power consumption, and critical path delay. Finally, we deploy the *L-Mul* FP8 approximate multiplier on 2 typical DNN accelerators to validate its superiority in terms of energy efficiency.
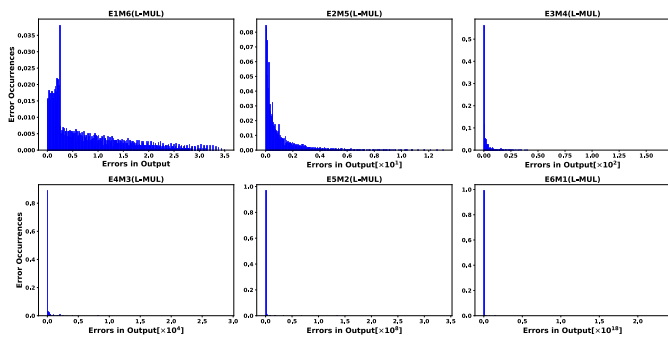
Fig. 4. The normalized number of unique error occurrences under different FP8 formats.

TABLE V
HARDWARE IMPLEMENTATION RESULTS OF *L-Mul* FOR DIFFERENT FP8 FORMATS

Data type	LUT	FF	CARRY8/4	WNS (ns)
E6M1	22	25	4	1.65
E5M2	21	25	4	1.64
E4M3	22	25	4	1.62
E3M4	22	25	4	1.64
E2M5	23	25	4	1.71
E1M6	22	25	4	1.76

B. Error Evaluation

Table IV presents the error metrics of *L-Mul* in different formats of FP8. It is important to note that when the exponent is allocated a larger bit-width, the range of representable numbers increases, which can lead to significantly larger MAE and MSE values due to the greater magnitude of errors. Moreover, to provide a more intuitive representation of the normalized number of unique error occurrences for the proposed multipliers, we visualize the data in Fig 4.

C. Hardware Implementation and Evaluation

We use Verilog to implement the *L-Mul* hardware design for different FP8 formats, as described in Section 3. The corresponding resource consumption and latency for each format is collected from AMD Vivado 2022.2 and shown in Table V. It can be observed that our design consumes an average of fewer than 23 LUTs. To further highlight the advantages of our design in terms of resource utilization and power consumption, we compare it with previous FPGA-based approximate multipliers and AMD's IP core. We use the deployment results under the E4M3 format, which is the most commonly used FP8 format. Although the multiplication rules for FP8 and INT8 differ, we consider this comparison valuable due to their identical data bit-width. Table VI presents the comparison results. For each item, it can be observed that our design exhibits the lowest resource consumption. Additionally, compared to the FP8-compatible RR_DyRecMul [21], our design achieves a lower delay. The fastest frequency, reported in [24], benefits from the simplicity of unsigned INT8 operations. It is important to note that references [23], [17], and [18] are implemented on AMD-Xilinx 7-series FPGAs,

TABLE VI
IMPLEMENTATION RESULTS OF DIFFERENT 8-BIT MULTIPLIERS

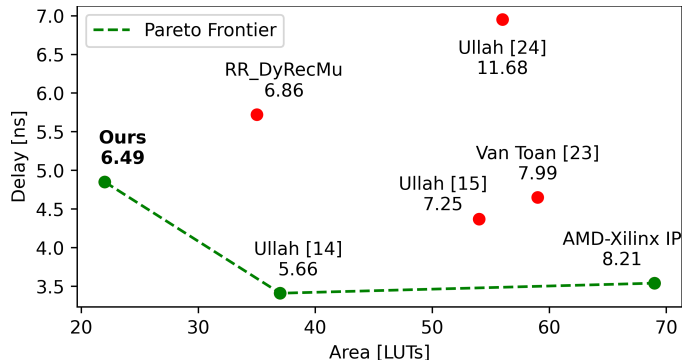| Designs	LUTs	Max Frq (MHz)	Delay (ns)	Power (mW)
Ours (FP8 E4M3)	**22**	617	4.85	1.34
Ullah [23] (INT8 Unsigned)	56	/	6.95	1.68
Van Toan [24] (INT8 Unsigned)	59	**759**	4.65	**0.432**
Ullah [17] (INT8 Unsigned)	37	/	**3.41**	/
Ullah [18] (INT8 Signed)	54	/	4.37	1.66
RR_DyRecMul [21] (FP8 to INT8 Signed)	35	699	5.72	/
AMD-Xilinx (Exact)	69	730	3.54	2.32

Fig. 5. The Pareto analysis under area and latency.

which introduces some power consumption differences. Additionally, the power data from [24] is recorded at 100 MHz, which is significantly lower than the results from other designs running at their maximum frequencies. To better demonstrate the power-efficiency advantages of our design, we performed a Pareto analysis to visually illustrate the differences between our design and others, as shown in Fig. 5. It shows that our design lies on the Pareto frontier. Additionally, due to its outstanding power-efficiency, our design achieves the second smallest Power-Delay Product (PDP) among the compared implementations.

D. Deployment and Evaluation on Computation of DNN Models

To further validate the performance and power efficiency of our design, we integrate it into both a CNN accelerator and a GCN accelerator. We evaluate inference accuracy on three representative CNN datasets (MNIST, CIFAR-10, and ImageNetV2) and three typical GCN datasets (CORA, CiteSeer, and Pubmed). Table VII shows the average accuracy loss for the corresponding models under different data formats. FP8, with its superior dynamic range, incurs the least accuracy loss. Although *L-Mul* exhibits the highest accuracy loss, it eliminates multiplication operations, achieving significantly better hardware deployment efficiency. We design CNN inference accelerators based on INT8 and *L-Mul* (FP8, E4M3) using the quantization parameters shown in Table VII, with Faster R-CNN as the backbone network. The resulting resource

Models	FP32	FP8 (E4M3)	INT8	*L-Mul* (E4M3)
CNN	0	-0.04%	-0.10%	-0.96%
GCN	0	-1.96%	-2.77%	-3.01%

Multiplier	Model	LUT	DSP	Power (W)
INT8	CNN	117,067	1,156	9.46
(Exact)	GCN	161,529	512	8.61
Ours	CNN	143,702	**0**	**8.08**
(FP8, E4M3)	GCN	173,387	**0**	**8.23**

consumption and power usage are shown in Table VIII. Thanks to the approximate multiplier design, we achieve a DSP-free implementation and reduce power consumption by 14.59% at the same operating frequency (250 MHz). Similarly, we implement a GCN inference accelerator based on *L-Mul* (FP8, E4M3) using LW-GCN (INT8) [25]. This design also achieves a DSP-free implementation at the same operating frequency and reduces overall power consumption.

V. CONCLUSIONS

This paper presents a power-efficient hardware deployment for the *L-Mul* algorithm. By analyzing the CLB structure of AMD FPGAs, we achieve fine-grained optimization through primitive-based design. We demonstrate and analyze the deployment results for FP8, showing that our design achieves the lowest LUT consumption and power usage compared to other 8-bit designs. Furthermore, we deploy our design in the inference phase of typical NNs, validating its effectiveness and power efficiency. In the future, we plan to integrate this design into LLMs and diffusion model to further demonstrate its advantages.

REFERENCES

[1] M. Dampfhoffer, T. Mesquida, A. Valentian, and L. Anghel, "Backpropagation-based learning techniques for deep spiking neural networks: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 9, pp. 11 906–11 921, 2024.

[2] H. Luo and W. Sun, "Addition is all you need for energy-efficient language models," *arXiv preprint arXiv:2410.00907*, 2024.

[3] Q. Han, Y. Hu, F. Yu, H. Yang, B. Liu, P. Hu, R. Gong, Y. Wang, R. Wang, Z. Luan, and D. Qian, "Extremely low-bit convolution optimization for quantized neural network on modern computer architectures," in *Proceedings of the 49th International Conference on Parallel Processing*, ser. ICPP '20. New York, NY, ACM, 2020.

[4] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates, Inc., 2022, pp. 27 168–27 183.

[5] H. Shen, N. Mellempudi, X. He, Q. Gao, C. Wang, and M. Wang, "Efficient post-training quantization with fp8 formats," in *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. D. Sa, Eds., vol. 6, 2024, pp. 483–498.

[6] D. R. Lutz, A. Saini, M. Kroes, T. Elmer, and H. Valsaraju, "Fused fp8 4-way dot product with scaling and fp32 accumulation," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. IEEE, 2024, pp. 40–47.

[7] S. K. Lee, A. Agrawal, J. Silberman, M. Ziegler, M. Kang, S. Venkataramani, N. Cao, B. Fleischer, M. Guillorn, M. Cohen *et al.*, "A 7-nm four-core mixed-precision ai chip with 26.2-tflops hybrid-fp8 training, 104.9-tops int4 inference, and workload-aware throttling," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 1, pp. 182–197, 2021.

[8] S. K. Venkataramanaiah, J. Meng, H.-S. Suh, I. Yeo, J. Saikia, S. K. Cherupally, Y. Zhang, Z. Zhang, and J.-S. Seo, "A 28-nm 8-bit floating-point tensor core-based programmable cnn training processor with dynamic structured sparsity," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 7, pp. 1885–1897, 2023.

[9] A. C. Elster and T. A. Haugdahl, "Nvidia hopper gpu and grace cpu highlights," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 95–100, 2022.

[10] AMD Xilinx. Ultrascale architecture configurable logic block user guide. Accessed: 2024-11-11. [Online]. Available: https://docs.amd.com/v/u/en-US/ug574-ultrascale-clb

[11] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[12] M. van Baalen, A. Kuzmin, S. S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soriaga *et al.*, "Fp8 versus int8 for efficient deep learning inference," *arXiv preprint arXiv:2303.17951*, 2023.

[13] 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, 2020.

[14] S. Zheng, Z. Li, Y. Lu, J. Gao, J. Zhang, and L. Wang, "Heam: High-efficiency approximate multiplier optimization for deep neural networks," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 3359–3363.

[15] C. Chen, S. Yang, W. Qian, M. Imani, X. Yin, and C. Zhuo, "Optimally approximated and unbiased floating-point multiplier with runtime configurability," in *Proceedings of the 39th international conference on computer-aided design (ICCAD)*, 2020, pp. 1–9.

[16] S. Ullah, S. S. Murthy, and A. Kumar, "Smapproxlib: Library of fpga-based approximate multipliers," in *Proceedings of the 55th Annual Design Automation Conference (DAC)*. New York, NY, ACM, 2018, pp. 1–6.

[17] S. Ullah, H. Schmidl, S. S. Sahoo, S. Rehman, and A. Kumar, "Area-optimized accurate and approximate softcore signed multiplier architectures," *IEEE Transactions on Computers*, vol. 70, no. 3, pp. 384–392, 2020.

[18] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-performance accurate and approximate multipliers for fpga-based hardware accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 211–224, 2021.

[19] Y. Liu, S. Ullah, and A. Kumar, "Bitsys: Bitwise systolic array architecture for multi-precision quantized hardware accelerators," in *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2024, pp. 220–220.

[20] S. Ullah, S. S. Sahoo, and A. Kumar, "Axospike: Spiking neural networks-driven approximate operator design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 3324–3335, 2024.

[21] S. Vakili, M. Vaziri, A. Zarei, and J. P. Langlois, "Dyrecmul: Fast and low-cost approximate multiplier for fpgas using dynamic reconfiguration," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[22] V. Leon, T. Paparouni, E. Petrongonas, D. Soudris, and K. Pekmestzi, "Improving power of dsp and cnn hardware accelerators using approximate floating-point multipliers," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5, pp. 1–21, 2021.

[23] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, "Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18, 2018.

[24] N. Van Toan and J.-G. Lee, "Fpga-based multi-level approximate multipliers for high-performance error-resilient applications," *IEEE Access*, vol. 8, pp. 25 481–25 497, 2020.

[25] Z. Tao, C. Wu, Y. Liang, K. Wang, and L. He, "Lw-gcn: A lightweight fpga-based graph convolutional network accelerator," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 1, pp. 1–19, 2022.