FPGA-based Approximate Multiplier for FP8

Ruiqi Chen[1], Yangxintong Lyu[1], Han Bao[1], Jiayu Liu[2], Yanxiang Zhu[3], Shidi Tang[4], Ming Ling[4,*], Bruno da Silva[1]
[1]ETRO, Vrije Universiteit Brussel, [2]University College London, [3]VeriMake Innovation Lab, [4]Southeast University
*Corresponding author: ruiqi.chen@vub.be, trio@seu.edu.cn, bruno.da.silva@vub.be

***Abstract*—The 8-bit floating-point (FP8) data format has been increasingly adopted in neural network (NN) computations due to its superior dynamic range compared to traditional INT8. However, FP8-based multiplication, a core operation in NNs, still incurs significant power consumption. To address this issue, this paper presents an FPGA-based approximate multiplier design for FP8. Firstly, we conduct a bit-level analysis of the approximation method. Based on this analysis, we implement a fine-grained optimized design on mainstream FPGAs (AMD and Altera) using primitives and templates combined with physical layout constraints. Then, the accuracy and resource utilization of the FP8 approximate multiplier are evaluated and analyzed. The results indicate that, compared to previous FPGA-based 8-bit designs, our design achieves the minimal LUT consumption. Finally, we integrate the design into the inference phase of a representative NN model, demonstrating its excellent power efficiency. To the best of our knowledge, this is the first FPGA-based FP8 approximate multiplier design, which can serve as a benchmark for future designs and comparisons of FPGA-based low-precision floating-point approximate multipliers. The code of this work is available in our GitLab.**

***Index Terms*—FPGA, FP8, multiplier, L-mul, Approximate computing**

I. Introduction

In recent years, the rapid development of neural networks has brought the issue of high energy consumption to the forefront [1]. An effective approach to address this challenge is the use of quantization techniques, which improves memory access and computational efficiency [2], [3]. Among these, 8-bit floating-point (FP8) numbers offer better dynamic range and computational precision compared with traditional 8-bit integer (INT8), making them widely adopted in neural network computations [4], [5]. 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) [6], [7] and integrating corresponding units into GPUs [8].

Nonetheless, this still cannot eliminate the multiplication operations in DNNs. 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. [9] proposed an optimally multi-level architecture that seamlessly integrates runtime configurability with parallel module execution. 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. [10] developed an 8x8 approximate multiplier tailored for NN designs by improving the design of logarithmic multipliers. HEAM [11] 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 sub-optimally 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. [12]–[14] proposed a series of FPGA-based approximate multipliers covering data bit-widths from 4-bit to 32-bit. More recently, their AxO series [15], [16] integrated the design of approximate multipliers into SNN accelerators. DyRecMul [17] 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. [18] 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 proposes an FPGA-based approximate multiplier for FP8. Our main contributions include:

- We introduce an approximate method for FP8 multiplication and analyze its principles at the bit-level. Based on this analysis, we implement the design on two mainstream FPGA platforms (AMD and Altera). Using primitives and templates combined with physical layout constraints, 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 make a comparison to the previous FPGA-based INT8 approximate multiplier designs, the result shows that 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 lowest power consumption.

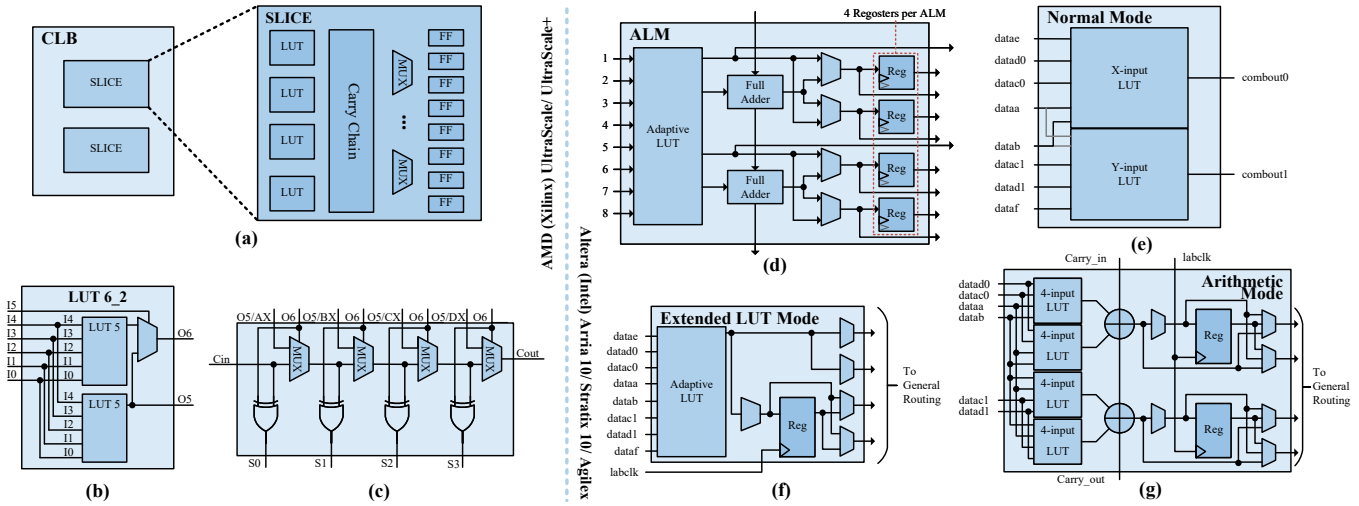

Fig. 1. The state-of-the-art FPGA basic structure from AMD (Xilinx) and Altera (Intel). (a) Typical AMD FPGA configurable logic block (CLB) structure [19]. (b) LUT6 structure. (c) Carry chain structure. (d) Typical Altera FPGA adaptive logic module (ALM) structure [20]. (e) The ALM runs in normal mode. (f) The ALM runs in extended LUT mode. (g) The ALM runs in arithmetic mode.

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

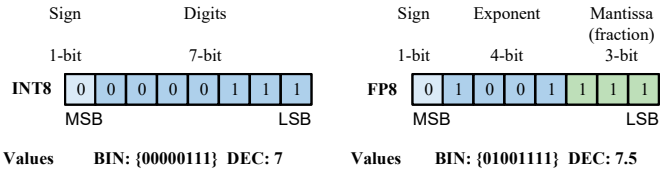

Fig. 2. The demonstration of the INT8 and FP8 (E4M3) defined in IEEE 754. MSB stands for most significant bit and LSB stands for least significant bit.

The paper is organized as follows: Section II provides the description of the FP8 formats and the introduction of FPGA structure. Section III introduces our efficient hardware implementation of FP8 approximate multiplier. Experimental results and discussion are given in Section IV. Conclusions are drawn in Section V.

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 [5]. Compared to traditional INT8, FP8 offers a larger dynamic range (the commonly used E4M3 format, as shown in Table I and Fig. 2). Moreover, FP8 achieves less accuracy loss during NN inference [21]. The FP8 format adheres to IEEE-754 conventions, where a real numbers is encoded by 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 is determined by the following formula:

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

Note that an implict 1, namely the $hiddenbit$, is concatenated to the fraction as an integer bit and forms 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 $NaNs$.

B. FPGA Structure

State-of-the-art FPGAs from AMD-Xilinx and Intel-Altera utilize basic logic cells such as multi-input LUTs, carry chains (adders), multiplexers, and D flip-flops to implement both combinational and sequential logic circuits. The method proposed in this paper is general, relying on LUTs, adders (carry chains), multiplexers, and D flip-flops. However, there are subtle differences depending on the characteristics of different devices. Therefore, we present two types of mainstream FPGA devices for illustration, as shown in Fig. 1.

A slice in the configurable logic block (CLB) of AMD's Virtex-7 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 UltraScale/UltraScale+) and eight flip-flops and multiplexers, as shown in Fig. 1(a). A LUT6_2 can be used to implement either a single 6-bit combinational function using the O6

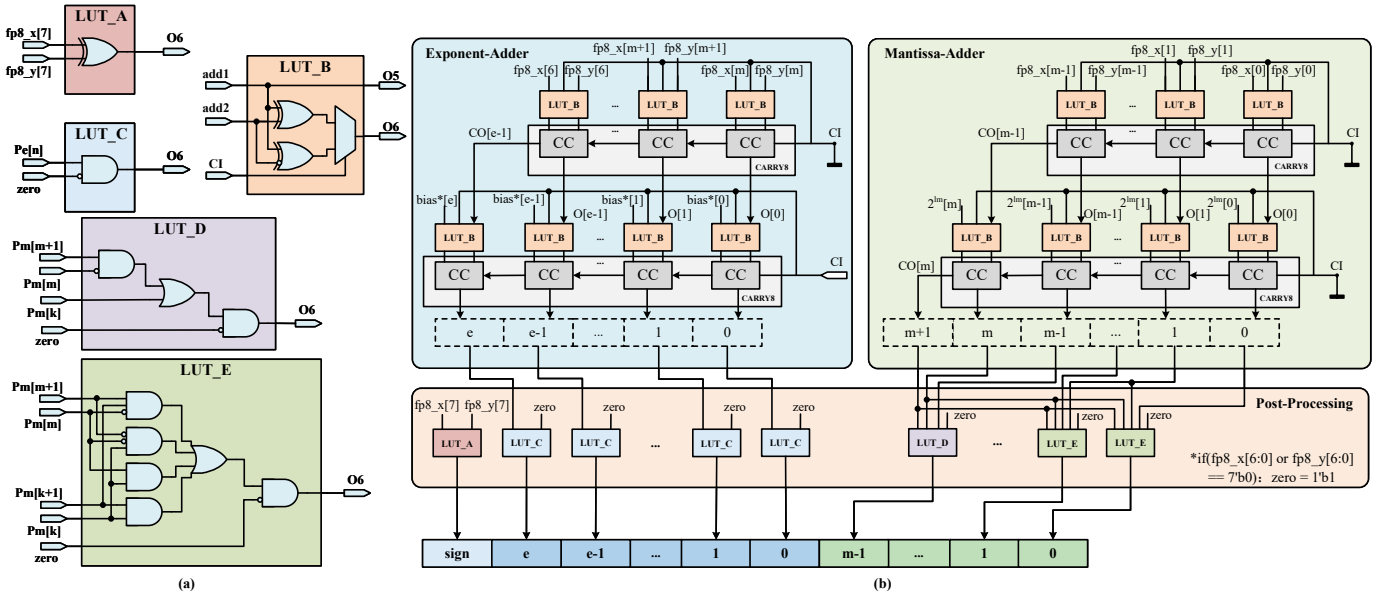

Fig. 3. Hardware fine-grained architecture for FP8 approximate multipler. (a) LUT-based basic components. (b) AMD-FPGA-based fine-grained architecture for FP8 approximate multipler.

output bit, or two 5-bit combinational functions using the O5 and O6 output bits, as shown in Fig. 1(b). To do it, an INT value is defined, 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. 1(c). The carry chain implements a carry-lookahead adder by 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.

The adaptive logic module (ALM) in Altera's Stratix 10, Arria 10, and Agilex series primarily includes an 8-input fracturable LUT, two dedicated embedded adders, and four dedicated registers, as shown in Fig. 1(d). Unlike AMD's CLB, Altera's ALM supports three different modes: normal mode, extended LUT mode, and arithmetic mode. In the normal mode, the adaptive LUT is divided into two multi-input LUTs: an X-input LUT and a Y-input LUT, as shown in Fig. 1(e). The two LUTs are connected through logical interconnections, allowing for different X:Y combinations. When the inputs are independent, the LUTs can support configurations such as 4:4 or 5:3, or even less inputs. However, when X:Y is 5:4 or 5:5, the inputs need to be shared between the two LUTs. In the extended LUT mode, the ALM can support a maximum of an 8-input LUT, as shown in Fig. 1(f). The ALM in arithmetic mode uses two sets of two 4-input LUTs along with two dedicated full adders, as shown in Fig. 1(g). The dedicated adders allow the LUTs to perform pre-adder logic. Therefore, each adder can add the output of two 4-input functions.

III. The Proposed Approximate Multiplier for FP8

A. Approximate Multiplication for FP8

According to the Eq. (1), the FP8 multiplication process of x and y can be represented as:

$$\begin{aligned} Mul(x,y) &= M_x \cdot 2^{E_x} \times M_y \cdot 2^{E_y} \\ &= (1+m_x) \cdot 2^{E_x} \times (1+m_y) \cdot 2^{E_y} \\ &= (1+m_x+m_y+m_x \cdot m_y) \cdot 2^{E_x+E_y}, \end{aligned} \tag{3}$$

we omit the sign bit as it can be handled through an XOR operation. One can note that in Eq. (3), only $m_x \cdot m_y$ involves a multiplication operation for hardware circuit design. The remaining operations can be implemented by using addition or other linear operations such as shift. To alleviate the potential bottleneck caused by mantissa multiplication, Luo et al. [22] propose the *L-Mul* algorithm, which can be designed to approximate the FP8 multiplication process:

$$\begin{aligned} \textit{L-Mul}(x,y) &= (1+m_x+m_y+2^{-l(m)}) \times 2^{E_x+E_y}, \\ l(m) &= \begin{cases} m & \text{if } m \le 3, \\ 3 & \text{if } m = 4, \\ 4 & \text{if } m \ge 4, \end{cases} \end{aligned} \tag{4}$$

where m denotes the bit-width of the mantissa. By using this piecewise function approximation, the original multiplication operation can be transformed into shift and addition operations. We will introduce the corresponding fine-grained FPGA-based hardware design in Section III-B.

B. Hardware Design

1) AMD-FPGA-Based 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:

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	$100.x_m$

TABLE III
THE $bias*$ VALUES FOR DIFFERENT TYPES OF FP8 FORMATS CORRESPOND TO SPECIFIC CONFIGURATIONS

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

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

To achieve this, we design five LUT configurations combined with carry chains to implement the 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.

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 and $CI = 1$ indicates subtraction. The O corresponds to the sum (S) in the full adder which can be 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

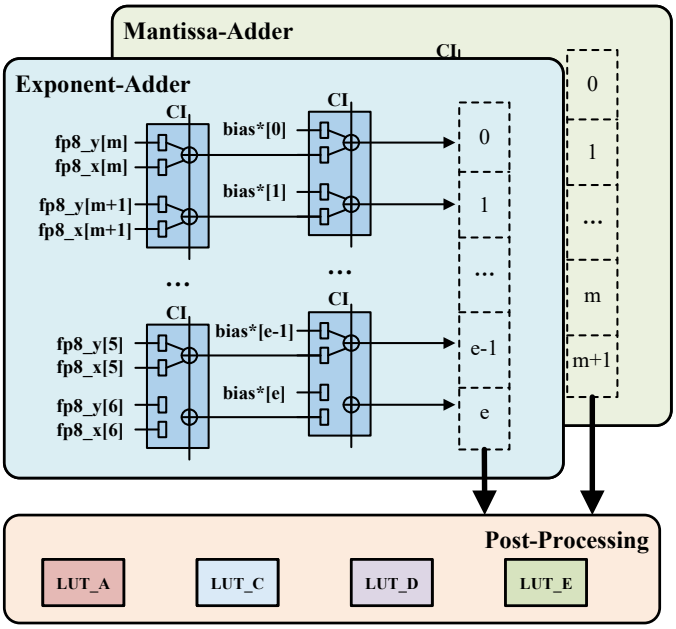

Fig. 4. Altera-FPGA-Based fine-grained architecture for FP8 approximate multipler.

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,

$$P'_m[m-1:0] = \begin{cases} 0, & \text{zero} = 1 \\ \{1'b1, P_m[m-1:1]\}, & P_m[m+1:m] = 2'b10 \\ P_m[m-1:0], & \text{others} \end{cases} \quad (6)$$

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

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 for the types of FP8 formats. LUT_C, LUT_D and LUT_E are used to implement Eq. (6) and (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.

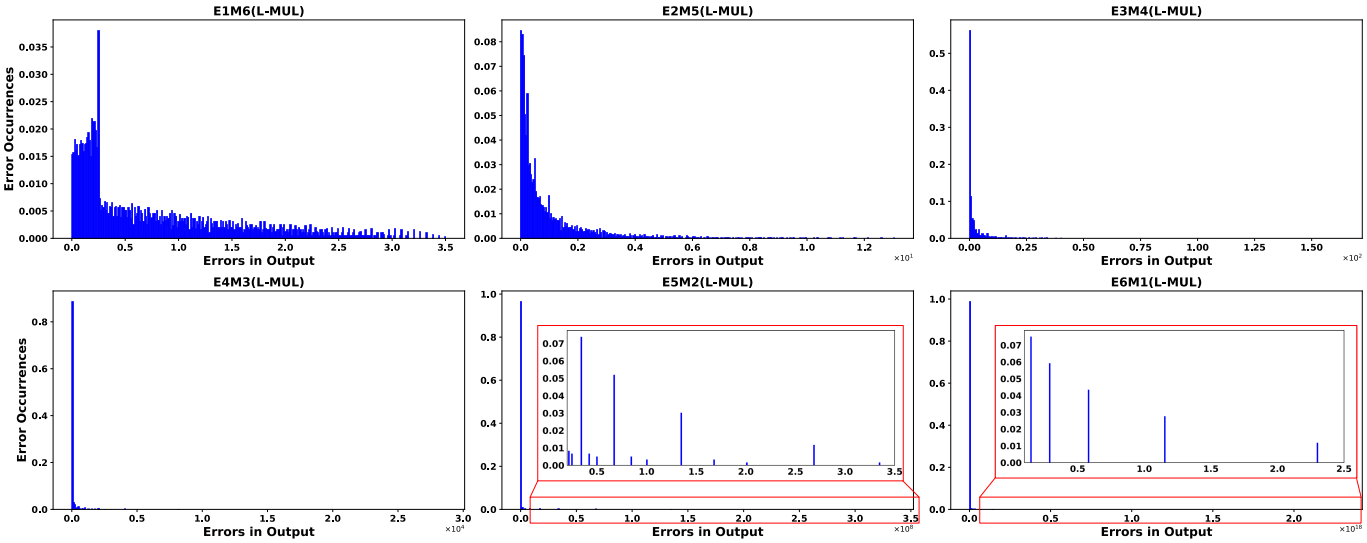

Fig. 5. The normalized number of unique error occurrences under different FP8 formats.

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.

2) Altera-FPGA-Based Design: Similarly, the design based on bit-level Eq. (5) is divided into three parts: the Exponent-Adder, the Mantissa-Adder, and the Post-Processing unit, as shown in Fig 4. As introduced in Section II-B, due to the integration of full adders within the ALMs of Altera FPGAs, the designs of Exponent-Adder and Mantissa-Adder need to be modified, while the Post-Processing unit can be directly reused. In other words, only the LUT_B design needs adjustment, while other LUT-based basic components (as shown in Fig. 3(a)) can be directly applied to Altera FPGAs. We configure the ALMs to operate in arithmetic mode, where each ALM includes two full adders. Thus, the combination of LUT_B + CC from AMD FPGAs can be directly replaced by the ALM implementation. Additionally, since carry cascading is supported between ALMs, the cascade can be achieved by enforcing physical placement constraints on adjacent ALMs during layout. Under this configuration, one ALM serves as an equivalent replacement for two LUT_B and CC combinations. Therefore, Exponent-Adder and Mantissa-Adder consume e ALMs and m ALMs, respectively. The differences in design between AMD and Altera for this implementation will be analyzed in Section IV-C.

TABLE IV
ERROR METRICS AND CORRESPONDING FORMULAS

Metric	Formula
Error Probability (EP)	$EP = \frac{1}{2^N}\sum_{i=0}^{2^N-1} ED_i \neq 0$
Mean Absolute Error (MAE)	$MAE = \frac{1}{2^N}\sum_{i=0}^{2^N-1} ED_i$
Mean Relative Error (MRE)	$MRE = \frac{1}{2^N}\sum_{i=0}^{2^N-1} \frac{ED_i}{Exact_i}$
Mean Squared Error (MSE)	$MSE = \frac{1}{2^N}\sum_{i=0}^{2^N-1} (ED_i)^2$
Normalized Error Distance (NED)	$NED = \frac{1}{2^N}\sum_{i=0}^{2^N-1} \frac{ED_i}{\max(ED)}$

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 in Table IV.

The FP8 approximate multiplier design is described in Verilog and implemented using AMD Vivado 2022.2 and Quartus Prime Pro 23.3 for logic synthesis and placement constraints. The design is deployed and validated on the ZCU104 Evaluation Kit of UltraScale+ FPGA and Apollo Agilex SOM, respectively. 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

TABLE V
AMD FPGA AND ALTERA FPGA IMPLEMENTATION RESULTS FOR DIFFERENT FP8 FORMATS

Data type	AMD UltraScale+ FPGA				Altera Agilex FPGA		
	LUTs	FFs	CARRY	Max Frq (MHz)	ALM	Logic Register	Max Frq (MHz)
E6M1	22	25	4	606	27	25	655
E5M2	21	25	4	610	26	25	674
E4M3	22	25	4	617	27	25	694
E3M4	22	25	4	610	27	25	623
E2M5	22	25	4	568	24	25	519
E1M6	22	25	4	585	25	25	577

TABLE VI
ERROR EVALUATION ACROSS DIFFERENT FP8 FORMATS & COMPARISON WITH RELATED WORK

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
DyRecMul [17]	0.5157	397	0.0680	96336	0.00005

are used to calculate 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 FP8 approximate multiplier on a typical DNN accelerator to validate its superiority in terms of energy efficiency.

B. Error Evaluation

Table VI presents the error metrics of *L-Mul* in different formats of FP8. For the commonly used E4M3 format, *L-Mul* demonstrates better performance in terms of MAE and MRE compared to DyRecMul [17], which is the latest INT8 approximate design. This is an acceptable outcome given the better data range of the FP8 format. Moreover, 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 5.

C. Hardware Implementation and Evaluation

The resource consumption and latency for different FP8 formats implemented on AMD FPGAs and Altera FPGAs are shown in Table V. For AMD UltraScale+ FPGA, by leveraging fine-grained primitive designs and physical placement and routing constraints, all components are mapped to the resources within six adjacent CLBs, as shown in Fig. 6. Meanwhile, the carry chain consumption aligns with the design intent, requiring four carry chains. It can be observed that our design consumes fewer than 22 LUTs on average. In the case of Altera Agilex FPGA, we utilize templates and logic region

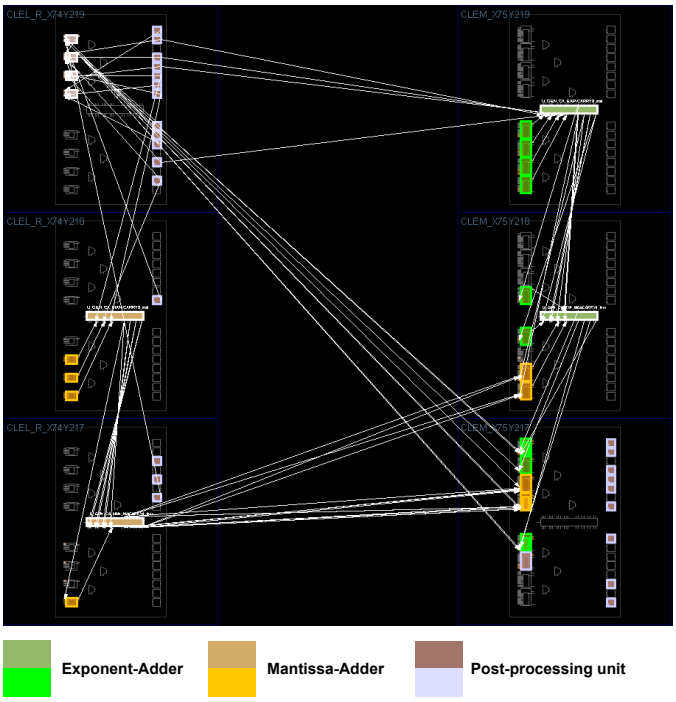

Fig. 6. The layout of our design on the AMD UltraScale+ FPGA.

constraints to compensate for the absence of fine-grained primitives. All components are mapped to the resources within three adjacent logic array blocks, as shown in Fig. 7(b). Using the resource property viewer tool, we confirm that the ALMs function in arithmetic mode and achieve carry cascading, as intended in Section III-B2 and shown in Fig. 7(c).

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-Xilinx's IP core. It should be noted that previous work has primarily been based on AMD (Xilinx) FPGAs, and there are certain differences in resource calculation between AMD and Altera FPGAs. Therefore, in this comparison, we only present the results for AMD FPGAs. 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 VII presents the comparison results, these values are reported or estimated from original reports. It should be noted that the data in the table VII is

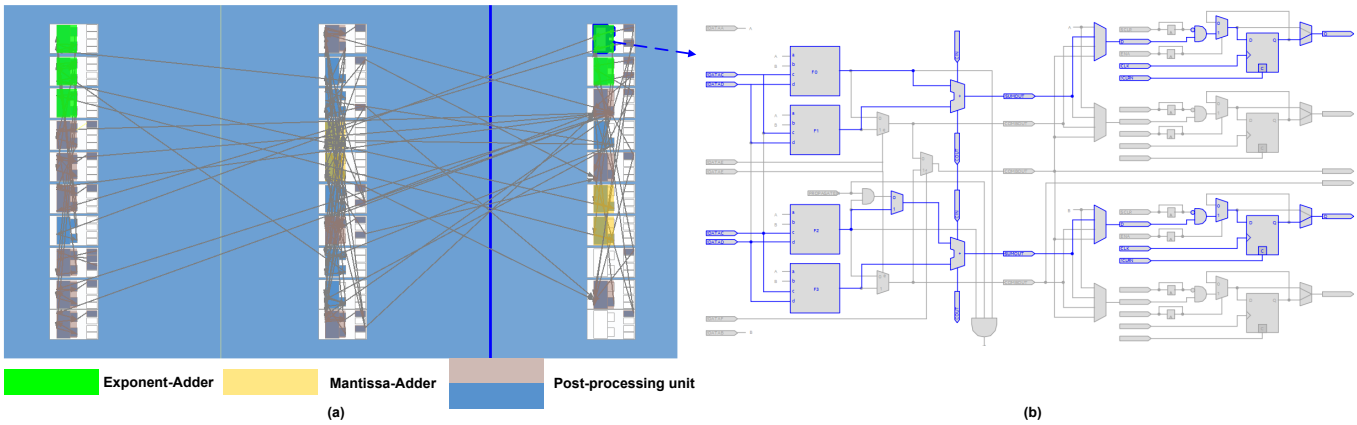

Fig. 7. (a) The layout of our design on the Altera Agilex FPGA. (b) The demonstration of ALM runs in arithmetic mode.

TABLE VII
RESOURCE AND PERFORMANCE COMPARISON OF 8-BIT APPROXIMATE MULTIPLIERS BASED ON REPORTED DATA IN PRIOR WORK

Designs	Year	Device Family	LUTs	FFs	CARRY	DSPs	Max Frq (MHz)	Delay (ns)	Power (mW)
AMD-Xilinx[1] (Exact)	2015	UltraScale+ (16nm)	69	16	14	0	**730**	3.54	2.32
Ullah [23] (INT8 Unsigned)	2018	Virtex-7 (28nm)	56	**0**	4	0	/	6.95	1.68
Van Toan [24] (INT8 Unsigned)	2020	Spartan-6 (45nm)	59	/	4	0	/	4.65	**0.432**[2]
Ullah [13] (INT8 Unsigned)	2020	Virtex-7 (28nm)	37	35	4	0	/	**3.41**	1.65[3]
Ullah [14] (INT8 Signed)	2021	Virtex-7 (28nm)	54	32	9	0	/	4.37	1.66
DyRecMul [17] (FP8 to INT8 Signed)	2024	UltraScale+ (16nm)	35	/	**0**	0	699	5.72	1.20[3]
Ours (FP8 E4M3)	2025	UltraScale+ (16nm)	**22**	25	4	0	617	4.85	1.34[4]

[1] The IP version is Multiplier v12.0 and values reported at 200 MHz
[2] Values reported from the original report (operating at 1.2V and 100 MHz)
[3] Values estimated from the reported PDP in the original report
[4] Values reported at 200MHz

sourced from the original reports of each respective paper. As a result, the comparisons involve different devices and operating frequencies. For each item, it can be observed that our design exhibits the lowest resource consumption. Additionally, compared to the FP8-compatible DyRecMul [17], our design achieves a lower delay. Benefiting from the simplicity of unsigned INT8 operations, our design achieves the fastest frequency compared with [24]. It is important to note that references [23], [13], and [14] are implemented on AMD-Xilinx 7-series FPGAs, which introduces some power consumption differences. Additionally, the power data reported in [24] is recorded at 1.2 V supply and 100 MHz frequency, which is significantly lower than the results from other design. 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. 8. The result 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.

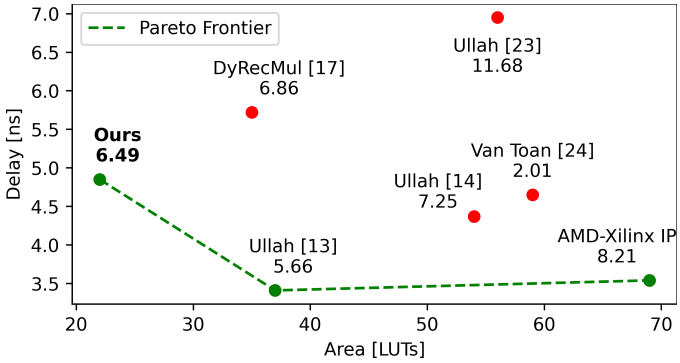

Fig. 8. The Pareto analysis under area and delay. The data under the design point represents the Power-Delay Product (PDP).

D. Case Study: DNN Accelerator Integration

To further validate the performance and power efficiency of our design, we integrate it into a CNN accelerator. The CNN model is built using PyTorch and quantized using a post-

TABLE VIII
EVALUATION OF ACCURACY LOSS FOR DIFFERENT DATA FORMATS ACROSS VARIOUS DATASETS

	Datasets	MNIST	CIFAR-10	ImageNetV2
Exact	FP32	0	0	0
	FP8 (E4M3)	-0.04%	-0.36%	-0.18%
	INT8	-0.1%	-1.69%	-0.35%
Approx	Ullah [23] (INT8 Unsigned)	-1.81%	-3.1%	-1.52%
	Ullah [13] (INT8 Unsigned)	-0.88%	-2.83%	-5.01%
	Ullah [14] (INT8 Signed)	-1.33%	-1.56%	-0.49%
	DyRecMul [17] (INT8 Unsigned)	-1.66%	-7.25%	-0.21%
	L-Mul (E4M3)	-0.96%	-1.56%	-0.83%

TABLE IX
HARDWARE DEPLOYMENT RESULTS FOR NEURAL NETWORK INFERENCE USING DIFFERENT MULTIPLIERS, OBTAINED THROUGH OUR OWN EXPERIMENTS ON A UNIFIED PLATFORM

Multiplier	LUT	FF	DSP	Power	Frq
INT8 (Exact)	117,067	95,317	1,156	9.46	200
Ullah [23]	188,324	136,434	0	10.33	173
Ullah [13]	166,534	152,056	0	9.56	192
Ullah [14]	186,602	157,106	0	10.21	187
DyRecMul [17]	250,960	169,079	0	12.69	200
Ours	143,702	122,020	0	9.08	200
Ours (default[1])	163,540	152,157	0	9.54	189

[1] The design is directly deployed by Vivado without any primitives and physical placement constraints.

training quantization (PTQ) strategy [25]. The model consists of 13 convolutional layers, 1 Region Proposal Network, 1 RoI Pooling layer, and 2 fully connected layers. We evaluate the inference accuracy on three representative CNN datasets (MNIST, CIFAR-10, and ImageNetV2). Table VIII shows the average accuracy loss for the corresponding models under different data formats. FP8, with its superior dynamic range, incurs less accuracy loss compared with INT8. For the comparison of 8-bit approximation methods, *L-Mul* demonstrates the best robustness. Specifically, across three datasets, *L-Mul* achieves the lowest average accuracy drop (-1.12%) and the smallest standard deviation (0.39) among all methods.

To demonstrate the high energy efficiency of our approximate multiplier in hardware deployment, we integrate it into a CNN accelerator to replace the original DSP blocks, and evaluate the corresponding resource usage and power consumption. For comparison, prior works are also integrated and deployed using the same methodology. Considering that most baseline designs are developed on the Virtex-7 platform, we adopt the VC709 Connectivity Kit as the target platform. This choice does not affect our implementation, as introduced in Section II-B, the Virtex-7 and UltraScale+ devices feature the same CLB architecture. The synthesis and implementation are conducted using AMD Vivado 2022.2. As shown in Table IX, both resource utilization and power consumption are obtained from the post-implementation reports generated by Vivado. Power consumption is obtained as the sum of GTH, hard IP, and dynamic components. Among them, GTH and hard IP are used for PCIe communication, and this portion of the power overhead is identical across all accelerators. We design CNN inference accelerators based on INT8 exact multiplier and 8-bit approximate multiplier using the quantization parameters shown in Table VIII, with Faster R-CNN as the backbone network. The data shows that all 8-bit approximate multipliers enable DSP-free designs; however, only our design and DyRecMul are able to maintain the original operating frequency of 200 MHz. Among all 8-bit approximate designs, our approach consumes the lowest LUTs and FFs, leading to reduced power consumption. Specifically, compared to the original INT8 design, our method achieves a 4.02% reduction in power, and compared to the state-of-the-art DyRecMul, power is reduced by 28.45%. In addition, the table includes a default accelerator implementation as part of the ablation study. This design is directly derived from Eq. (5) for RTL implementation, without applying any primitives-based fine-grained optimizations or physical placement constraints. The results show that under the default strategy, the default implementation incurs higher resource usage and exhibits inferior operation frequency. To sum up, these results demonstrate the superiority of our design especially in power efficiency.

V. CONCLUSIONS AND FUTURE WORK

This work presents an FPGA-based approximate multiplier design for FP8. More specifically, we analyze the features of different FPGA structures and achieve fine-grained optimizing hardware implementation. The experimental results demonstrate that our design achieves the lowest LUT consumption and power usage compared with the existing FPGA-based 8-bit designs. Furthermore, we deploy the proposed design in the inference phase of typical DNN accelerators, validating its effectiveness and power efficiency. We are conducting further research in the following: 1. Integrating the proposed design into emerging DNN models, such as graph neural network models, large language models and diffusion models, to further demonstrate its advantages. 2. Performing fine-grained optimizations on additional FPGA platforms, such as Microchip and Lattice. 3. Enhancing the current approximation methods to reduce accuracy loss.

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] 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 (ICPP)*. New York, NY, USA: ACM, 2020.

[3] 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.

[4] 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*, vol. 6, 2024, pp. 483–498.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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)*. ACM, 2020, pp. 1–9.

[10] 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.

[11] 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.

[12] 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, USA: ACM, 2018, pp. 1–6.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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*, vol. 20, no. 5, pp. 1–21, 2021.

[19] 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

[20] Altera Intel. Agilex™ 7 FPGAs and SoCs Device Overview. Accessed: 2024-11-11. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683458/current/adaptive-logic-module-in-fpgas-and-socs.html

[21] 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.

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

[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 (DAC)*. ACM, 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] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization0," *arXiv preprint arXiv:2106.08295*, 2021.