Coarse-Grained High-speed Reconfigurable Array-based Approximate Accelerator for Deep Learning Applications

Katherine Mercado
George Mason University
Fairfax, Virginia, USA
kmercad@gmu.edu

Sathwika Bavikadi
George Mason University
Fairfax, Virginia, USA
sbavikad@gmu.edu

Sai Manoj PD
George Mason University
Fairfax, Virginia, USA
spudukot@gmu.edu

***Abstract*—Deep Neural Networks (DNNs) are widely deployed in various cognitive applications, including computer vision, speech recognition, and image processing. The surpassing accuracy and performance of deep neural networks come at the cost of high computational complexity. Therefore, software implementations of DNNs and Convolutional Neural Networks (CNNs) are often hindered by computational and communication bottlenecks. As a panacea, numerous hardware accelerators have been introduced in recent times to accelerate DNNs and CNNs. Despite effectiveness, the existing hardware accelerators are often confronted by the involved computational complexity and the need for special hardware units to implement each of the DNN/CNN operations. To address such challenges, a reconfigurable DNN/CNN accelerator is proposed in this work. The proposed architecture comprises nine processing elements (PEs) that can perform both convolution and arithmetic operations through run-time reconfiguration and with minimal overhead. To reduce the computational complexity, we employ Mitchell's algorithm, which is supported through low overhead coarse-grained reconfigurability in this work. To facilitate efficient data flow across the PEs, we pre-compute the dataflow paths and configure the dataflow during the run-time. The proposed design is realized on a field-programmable gate array (FPGA) platform for evaluation. The proposed evaluation indicates 1.26× lower resource utilization compared to the state-of-the-art DNN/CNN accelerators and also achieves 99.43% and 82% accuracy on MNIST and CIFAR-10 datasets, respectively.**

I. Introduction

Deep learning algorithms including Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) have been widely adopted in a plethora of applications in recent times. These techniques exploit the internal correlation between the data samples in each of the internal hidden layers to perform the tasks such as classification or prediction with high accuracy. As a result, deep learning techniques require deeper networks leading to a large number of parameters requiring millions of multiply-and-accumulate operations. Implementing ML and other learning methodologies on traditional CPUs are facing a formidable challenge in terms of inference latency, memory accesses, and energy-efficiency due to a lack of temporal data locality and logic-memory communication Graphics processing units (GPUs) and custom-designed accelerators (ASICs) are designed for enhanced hardware performance. However, the performance and efficiency of such architectural paradigms are limited due to power consumption, costs, and reconfigurability [1], [2].

Field programmable gate array (FPGA)-based implementations are adopted recently for the deployment of numerous applications, including DNNs and CNNs due to their programmability, reconfigurability, and flexibility. Logic density on the state-of-the-art FPGAs allows good performance for these intensive computations. Another major advantage is FPGA's support for fine-grained and bit-level operations when compared to GPUs and ASICs makes this platform draw attention to low latency applications. Numerous FPGA-based DL accelerators are proposed in the literature [1], [3]–[7]. FPGA platforms though can enable reconfigurability and programmability, still incurs large resource utilization and latency when deployed for DL applications. To partially address this, the existing works have exploited the resilience of DNNs and CNNs despite utilizing low-precision data. In addition, approximate computing has been enabled to address the latency challenges [8]. In the literature, approximate arithmetic units such as dividers [9], adders [10], and multipliers [11] have been developed to implement DNN and CNN operations. The majority of these architectures are pre-configured for application-specific designs, confining their applicability to a specific architecture [12], [13] . Thus, the required resources and computational complexity can be reduced. However, the challenges of reconfiguration overheads and the computational complexity of performing multiplication and accumulation (MAC) operations still remain unanswered. In contrast, we exploit the reconfigurability of the FPGA architectures, arithmetic operations behind the approximation and MAC computations to propose the coarse-grained reconfigurable high-speed approximate accelerator for DL applications. For this purpose, we design a 3×3 tile structure of processing elements (PEs), reconfigurable through programming words to perform a wide variety of operations including add, subtract, multiply, divide, logarithm, and anti-logarithm. Mitchell algorithm [14] is employed to reduce the complexity of the resource-intensive multiply operations and enhance the involved computational latency. The high-speed reconfigurability and interconnectivity of FPGAs make the proposed design of PEs energy-efficient and reconfigurable with minimal overheads. The proposed architecture is also designed to facilitate programmability, reconfigurability, and

This work was partly supported by the US National Science Foundation (NSF) Grants CNS-2228239. The views, opinions, and/or findings of this article are those of the author(s). They should not be interpreted as representing the official views or policies, either expressed or implied, of the US NSF.

applicability to other applications with minimal overheads.

The novel contributions of this work can be outlined in a three-fold manner as follows:

- Mitchell algorithm-inspired reconfigurable PEs are designed to perform the MAC operations for DNNs and CNNs. The proposed design employs 8-bit operands to further minimize the computational overheads without impacting the performance of DNN/CNN implementation.
- A coarse-grained reconfigurable DNN/CNN accelerator on the FPGA platform is proposed to minimize the reconfiguration overheads and enable adaptability to a wide range of applications.
- A weight-stationary approach is employed for seamless dataflow across the PE cores in the proposed architecture. Look-up-Table(LUT) based log and antilog blocks are designed to support Mitchell's algorithm-based approximate multipliers with low latency.

We have evaluated the proposed coarse-grained reconfigurable architecture on the Zynq UltraScale ZCU126 FPGA platform for DNN and CNN networks. The proposed accelerator is at least 3.24× faster than previous work [4] with 8-bit precision and 4.86× faster than [5], [6] with 16 and 32-bits, respectively. Moreover, it shows improvement in both area and energy.

The rest of the paper is organized as follows. Section II provides an overview of related work. Section III describes the proposed architecture and the different operational modes it supports. Section IV presents the implementation and results. Finally, conclusions are provided in section V.

II. Related Work

Deep learning techniques, including DNNs and CNNs compose millions of MAC operations. These operations can be performed in a parallel manner. As such, FPGA platforms are one of the best-suited platforms for DL acceleration by exploiting their inherent parallelism. Numerous works have proposed FPGA-based accelerators for DL applications [4], [5], [6]. The great challenge for a hardware accelerator design is to find the best trade-off between power, performance, and reconfigurability. Reconfigurability brings forth advantages, as having more functionality employing fewer resources helps cost savings. Moreover, it may extend the useful life of hardware by updating its purpose and achieving faster development. Coarse-grained reconfigurable architecture is based on functional units such as PEs in a mesh-style network. This type of architecture may perform complex operations while providing low power consumption, less configuration, and routing overhead. The advantage of PEs is to avoid the combination of configurable logic blocks (CLBs) compared to pure FPGA-based hardware accelerators, ensuring a decrease in area and routing overhead, such as the DReAm [15] and MORA [16] architectures.

The present memory bottleneck found in FPGA-based hardware accelerators is localized in data movement, which could result in more energy consumption than the computation. One goal of this paper is to alleviate such concerns by implementing weight-stationary dataflow to maximize access to computation results from the different PEs and thus, minimize energy consumption. Coarse Grain Reconfigurable Architecture (CGRA) based architectures such as MORA [16], employ pipelined computational dataflow organized in two levels while eliminating the need for a centralized routing controller. Contrary to DreAm [15], which possesses a global unit for this purpose. Other works also, [17], [3], exploit the hierarchical dataflow concept for on-chip and inter-PE communication respectively, for both convolutional data and MAC operations. Another FPGA-based accelerator such as [18], proposed an adaptable reconfigurable datapath that allows depending on the operand, parallel or sequential dataflow for multiple operations.

FPGAs require a considerable amount of data reconfiguration for their programmable routing network. This is translated into a larger configuration time when multiple hardware configurations are involved in a single architecture. In order to overcome this, multiple-bit arithmetic processing elements, such as those working with 8 or 16 bits, as our proposed architecture, may be used where high efficiency is achieved for DNN/CNN computations while reducing power and area. Arithmetic hardware accelerator units are developed to carry neural network computations such as [14], [4], [5], [6], multipliers and dividers are the most frequent types of independent arithmetic units. Few designs execute both operations where the lack of support for division may generate a large overhead in the design [19], [11]. The proposed architecture integrates the flexibility benefits that an FPGAs platform provides, as well as characteristics of a reconfigurable coarse-grained based processing element architecture for multiple hardware configurations.

III. Proposed Approximate Computing-based High-Speed Reconfigurable Accelerator

We present the proposed high-speed reconfigurable approximate computing-based hardware accelerator architecture in Figure 1. The proposed architecture comprises nine processing elements (PEs), termed cores. Each of the PEs is designed to perform multiplications based on Mitchell's algorithm [14], discussed later in Section III-B. For the purpose of reconfigurability, the CGRA PEs are controlled using the codewords, defined through control bits (4-bits in our design). Depending on the control bits, the datapath and the PEs are configured. The reconfiguration time is reduced through the control words of the CGRA paradigm. Therefore, it allows the usage of configuration memory more efficiently. The details of individual blocks are discussed below.

A. System Architecture

The overall structure of the proposed architecture is presented in Figure 1. It comprises nine PEs arranged in a tiled manner. In the highest level of description, this architecture considers two operands A and B of 8-bit precision along with a code word of 4-bits, referred to as *Mode* signal. The output of this PE is 8-bits. The codewords and the corresponding operation are represented in Table I. In the current design, each operand is represented in a fixed-point format using 4-bits for the integer and 4-bits for the fractional part. The rationale to choose the fixed-point representation is its wide adoption for neural network applications, as it helps to reduce the logic usage and power consumption on FPGAs platforms.

TABLE I
CODEWORDS AND OPERATIONS

	Log Block	*2's C. Block*	*Adder Block*	*Antilog Block*
Add			0001	
Sub		0010	0010	
Mult	0100		0100	0100
Div	1000	1000	1000	1000

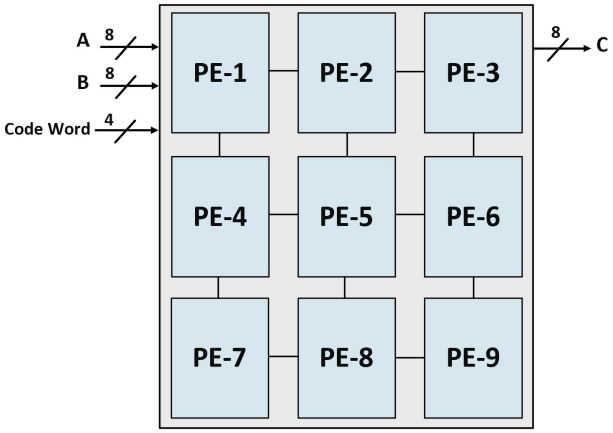

Fig. 1. Proposed accelerator architecture with cores in a mesh-style network.

B. Processing Element Architecture

One of the common and computationally intensive operations in the DL applications as well as the generic application workloads is the multiplication operation. The complexity further increases for fixed-point and floating-point data types. To address this challenge, we design our PE focusing on multiplication operations. However, designing only a multiplier makes the design inefficient, as other operations such as additions and subtractions are critical in DL and generic workloads.

Considering the reconfigurability of the underlying FPGAs, we design the PE to support multiplications as well as other arithmetic operations in this project. For this purpose, we design the multiplier based on Mitchell's algorithm [14] as described below. As per Mitchell's algorithm, a multiplication operation is defined as follows:

$$\begin{aligned} A \cdot B &= antilog(\log A + \log B) \\ A/B &= antilog(\log A - \log B) \end{aligned} \tag{1}$$

where A and B are the operands.

Based on (1), we design our PE architecture, as shown in Figure 2. Thus, the PE encompasses an adder, two's complement, log, and antilog blocks. Depending on the codeword, the individual units of the PE are enabled or disabled. For instance, to perform the multiplication operation, the log, adder, and antilog blocks will be enabled. Similarly, the codeword enables the log, adder, two's complement, and antilog blocks to perform the division operation.

The proposed architecture thus is capable of performing the add, subtract, log, antilog, multiply, and divide operations in a seamless manner. This is facilitated through the reconfigurability of the underlying FPGA architecture. To perform the reconfigurability through the codewords in a CGRA manner, we design a finite state machine (FSM). The FSM is responsible to decode the codewords and enable the data flow in a dynamic manner. As shown in Table I, depending on the codeword, the corresponding computational blocks in the PE will be enabled or disabled. Logarithm and antilogarithm functions are complex functions to be designed through standard CMOS designs. For this purpose, we employ Taylor's Series approximation and design the hardware accordingly.

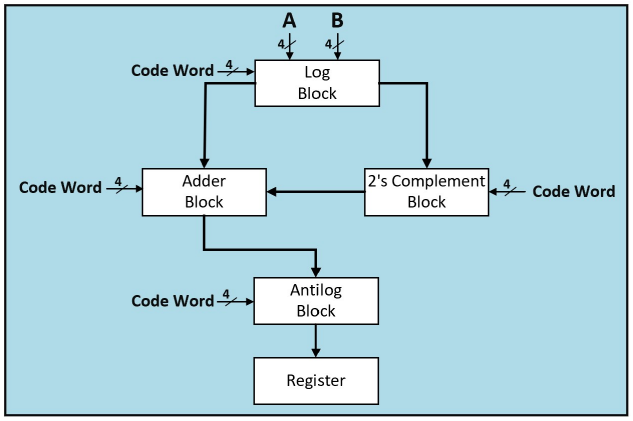

Fig. 2. Processing element architecture showing main four arithmetic blocks and register.

C. Taylor Series Algorithm for Log and Antilog Operations

The relationship between logarithm and antilogarithm to obtain a multiplication or a division is such that, the antilogarithm of the addition or subtraction of logarithms of A and B is the multiplication or the division of A and B respectively (1).

The logarithm function is achieved by the approximation of the Taylor Series for the natural logarithm. The function $log(x)$ is the approximation of the natural logarithm where a is the point where the function is centered and n represents the number of terms as shown in (2). Using 10 terms, the computation achieves an error of 1.5%.

$$log(x) = \frac{\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{na^n}(x-a)^n}{ln10} \tag{2}$$

To compute the antilogarithm for instance, for the multiply, the approximation between the logarithm and the exponential functions is made in (3).

$$antilog = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{na^n}(A-a)^n + \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{na^n}(B-a)^n \tag{3}$$

As observed from (2), direct implementation of logarithm and antilogarithm operations through CMOS design is inefficient due to the involved complexity. As such, we design a look-up-table (LUT)-based log and antilog units in this work. As FPGAs realize the design through LUTs, the design of logarithm and antilogarithm blocks are realized using the LUTs. This enables faster lookup, reduced computational complexity, and compatibility with the underlying FPGA architectures. Though Taylor's series and Mitchell's algorithms lead to the introduction of a certain amount of error in the computations, our analysis has shown that the introduced errors can be neglected for the considered application and datasets. However, error-correcting techniques can be adopted to minimize the error for other applications. This can also lead to additional complexity and hardware overheads.

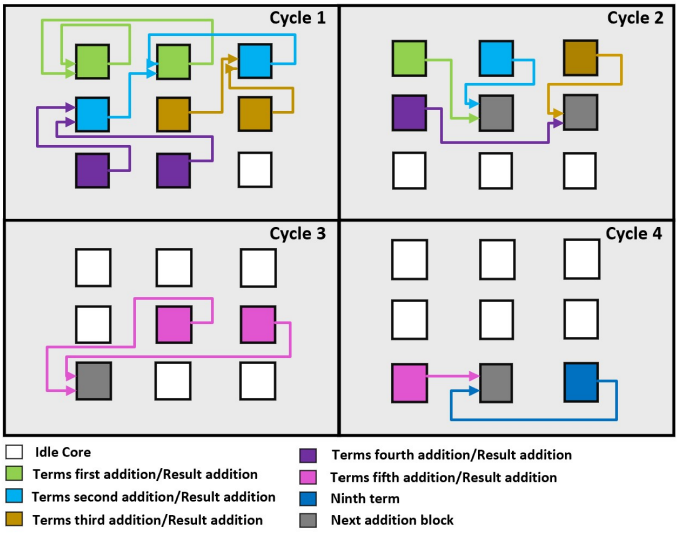

Fig. 3. Accelerator dataflow. Cycle 1: each color corresponds with one addition. The first addition operation takes place in the top left PE. The following additions continue right to the next PE until 4 additions. Cycle 2: each color corresponds with one result from previous additions, which are summed right to the next PE in pairs, shown in grey. Cycle 3: similar to the first case, the pink color corresponds with the terms of the next addition, in this case, there is only one addition Cycle 4: the addition of the previous result and the last PE are summed right to the next PE, shown in grey. The PEs are configured as shown in Table I.

D. Application Mapping on the Proposed Accelerator

Convolutional layers are the building blocks for CNN, where convolution calculations are employed for feature extraction. An element-wise product is performed between an input tensor and a kernel array. After this, all terms are summed to obtain the value in that specific position of the output feature map. The correspondent codewords for CNN computation activate the PEs and the behavior involves further steps as seen in Figure 3. Once the PEs have completed nine multiplications and have passed through the registers; these values are redirected back to the PEs to perform addition operations by utilizing and taking advantage of the same resources. Eight values are moved to the first four PEs, and every two terms for each addition are designated with the same color label as illustrated in Figure 3. The result of these four additions is again redirected to the next PEs, activating the following two in the next cycle. In the third cycle, the two PEs results become the two next terms for the following addition. Finally, our last result can be summed with the result of the ninth PE.

This illustration demonstrates that the multiply calculation among the PEs will be executed in parallel independently of the number of inputs involved. After that, the time computation increases according to the number of additions that may be performed in each clock cycle. With different operations, when a PE is inactive, its output value is registered to be dispatched for immediate use while the remaining modules may start orchestrating other operations according to their codewords.

E. Applicability to Other Workloads

As aforementioned, one of the advantages of the proposed reconfigurable architecture is its applicability to a wide range of applications in addition to the DL acceleration. In order to employ this architecture for non-ML applications, the aforementioned application mapping needs to be reprogrammed. In other words, by redefining the associated data flow with each of the code words one can apply the proposed architecture to non-ML applications. For instance, instead of convolution operation, general matrix multiplication (GEMM) is a common operation encountered in a plethora of applications and workloads. To perform GEMM operation on the proposed architecture, the operand mapping and sum dataflow have to be redefined compared to the convolution operation. This can be performed in a non-complex manner through the programmability of the FPGAs. GEMM reuses the data during computation, therefore, data movement and storage are drastically reduced, which leads to improved architectures.

IV. Results and Discussion

A. Simulation Setup

The proposed architecture with nine processing element (PE) cores is implemented on Xilinx Zynq UltraScale ZCU126 FPGA. The PEs are described using the very high-speed integrated circuit hardware description language (VHDL) and simulated using Modelsim Intel software. Further, it is synthesized as an IP block and verified in Xilinx Vivado 2021.1. The evaluation is performed in terms of the FPGA resources consumed for the deployment of the proposed architecture. The ML performance evaluation is performed on MNIST [20] and CIFAR-10 [21] datasets. We perform a comparison of proposed work with other FPGA accelerators (Table II), a comparison of individual approximate computing units (Table III), as well as a comparison with works where programming is performed to reconfigure the hardware for multiple functionalities (Table IV).

B. Evaluation of Processing Element

The data computation has been pipelined to reach the maximum possible frequency of operation. Furthermore, the scope of the data of both logarithm and antilogarithm computations is limited in the state machine to refine the code and reduce cycles. Area overhead has been decreased by reducing the precision of the data under performance constraints. Table II summarizes the proposed architecture results. As Delay and resources have been obtained from Vivado synthesis. Power consumption is that of all the processing elements and their communication network. As one can observe from Table II the proposed approximation-based system architecture can operate at a higher frequency with lower power and resource consumption. Compared to [6], the proposed architecture employs a slightly higher number of LUTs, but a significantly lower number of DSPs. The main benefit in terms of resources comes from the fact of reconfigurability and re-utilization of blocks to perform various operations.

TABLE II
FPGA-BASED ACCELERATOR PERFORMANCE COMPARISON

	[4]	[5]	[6]	**Proposed**
FPGA	Startix-V	Zynq	Arria-10	Zynq
Frequency (Mhz)	150	100	100	486
Precision	8-16bits	16-bits	32-bits	8-bits
LUTs	161K	155K	118K	127K
DSPs	1518	824	784	196
Power (W)	21.2	9.4	9.4	8.7

C. Comparative Performance

We also compare the speed and the area of the arithmetic units from our proposed architecture with distinct arithmetic units as shown in Table III. Many architectures found in the literature are focused on a specific arithmetic computation such as addition or multiplication [10], [11], [22], [23]. Additionally, there are some works that integrate adder and multiplier units in a single architecture [24]. As such, for a fair comparison, we configure the proposed architecture either as an adder, multiplier, or divider and compare it with the corresponding state-of-the-art works. However, it needs to be noted that irrespective of the configured functionality, the proposed system can be reconfigured to perform other operations, which is one of the advantages compared to state-of-the-art works.

As can be seen from Table III, the area increases for more complex computations such as multiplications and divisions compared to additions. The proposed architecture shows an area improvement in the adder, multiplier, and divider computation when comparing to individual units [9]–[11], [22], [23], [25]. For instance, the best arithmetic unit implemented in [11] has more significant LUTs than the proposed PE unit.

However, the proposed architecture has a wider benefit over existing architectures with multiple heterogeneous arithmetic blocks, as shown in Table IV. Observing a generic block for both multiply and addition such as [24], the proposed solution shows an improvement of 1.17× speedup in terms of speed and 1.09× reduction in area. Similarly, for the next arithmetic block in [24] that can perform multiply and subtraction operations through reconfiguration, our proposed system leads to a 1.19× speedup. Additionally, for both the configurable dynamic range multiplier shown in [26] and the configurable-error multiplier in [27], the proposed work is at least 2.4× faster.

In Table II, we compare our proposed architecture with previous FPGA-based accelerators of 8, 16, and 32 bits of precision. Previous works such as [5] reduce the amount of off-chip data transfer by the optimization of its dataflow with an increment in BRAM usage, an improvement over the previous architecture [6]. This is demonstrated with a better power efficiency when compared to [4]. However, there is still power required for the off-chip memory. Our solution optimizes power efficiency with optimized weight-stationary datapath, which further decreases energy consumption. The proposed architecture is 3.24× faster than [4] and 4.86× faster than [5], [6]. Implementation has been optimized for power and communication, leading to lower use of resources for both LUTs and DSPs compared to [4]–[6].

TABLE III
ARITHMETIC UNITS COMPARISON

Reference	Speed (Mhz)	Area (LUT)
Adder [10]	275	557
Adder [22]	71.017	932
Proposed Adder	486	243
Multiplier [11]	19.8	6971
Multiplier [23]	142.8	63400
Proposed Multiplier	486	780
Divider [9]	38	1060
Divider [25]	67.150	2472
Proposed Divider	486	815

TABLE IV
ARITHMETIC BLOCKS COMPARISON

Reference	Speed (Mhz)	Area (LUT)
Multiplier/adder [24]	415	1121
Proposed block	486	1023
Multiplier/Subtractor [24]	407	664
DRUM-4 [26]	218	53
MBM [27]	202	204
Proposed block	486	883

D. Inference Accuracy

We evaluate our proposed architecture for various state-of-the-art deep neural networks such as LeNet, AlexNet, and ResNet -18,-34,-50. These deep learning algorithms are implemented on the proposed hardware accelerator using MNIST ($28\times28\times1$ dimensions), and the CIFAR-10 dataset ($32\times32\times3$ dimensions). Both datasets consist of about 60,000 training and 10,000 testing images belonging to 10 classes. The goal is to classify the given input image into the correct class.

Figure 4 shows the Top-5 accuracy comparison plots for 16-bit floating-point (FP), 8-bit fixed-point data precision for both datasets. It is observed that the accuracies obtained on the evaluated networks are very similar for 16-bit and 8-bit precision data (inputs and weights). The Top-1 accuracy obtained for the MNIST dataset when implemented on AlexNet is 98.89% and 99.43% for 16-bit and 8-bit precision respectively. On the other hand, Top-1 accuracy obtained for the CIFAR-10 dataset when implemented on AlexNet is 83.5% and 82% for 16-bit and 8-bit precision respectively.

It is also observed that the accuracies of LeNet, AlexNet, ResNet-18, -34, and -50 on the CIFAR-10 dataset are noticeably lower when compared to the MNIST dataset, as shown in Figure 4. The accuracy of the CIFAR-10 dataset, in general, is significantly lower than MNIST dataset due to the comparatively higher complexity of the CIFAR-10 dataset. Although higher accuracy with CIFAR-10 is reported in the literature, it is with higher data precision than those adopted in this work [28]. Such high accuracy comes at the cost of additional hardware resources and processing overheads.

The baseline accuracy of the networks with the 8-bit precision data in comparison with 6-bit precision (referred to as approximation data) is shown in Figure 5. For the evaluation purpose, the approximation strategy adopted is to disregard the 2 least significant bits in the 8-bit fixed point input data before implementing the neural networks. The accuracies for AlexNet with 8-bit input and weights, 6-bit input and weights of approximated data, 99.43%, 99.11% respectively on the MNIST and 82%, 81.7% respectively on the CIFAR10 dataset, as demonstrated in Figure 5. The performance degradation between the 8-bit fixed point precision and to 6-bit approximation strategy is around $< 0.05\%$- $< 0.35\%$ for all the CNNs deployed on both datasets. We have evaluated with 6-bit precision to validate the performance impact, as we have observed a nearly 9.4% reduction in the required resources as compared with the 8-bit implementation reported in Table II.

V. CONCLUSION

This paper presents a novel reconfigurable accelerator architecture aimed at CNN computations. Its programmability allows

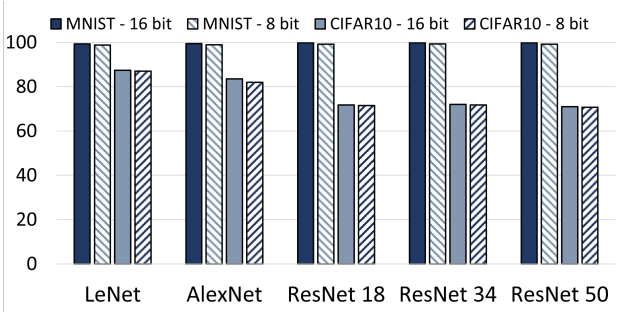

Fig. 4. Comparison of Top-5 accuracies of LeNet, AlexNet, ResNet-18, -34 and -50 on MNIST, CIFAR-10 datasets for 16-bit, 8-bit data precision.

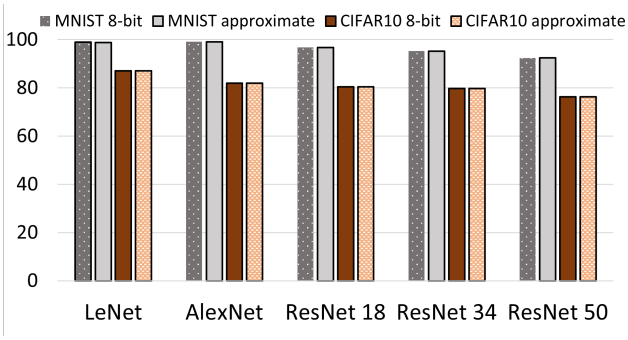

Fig. 5. Accuracy comparison of LeNet, AlexNet, ResNet-18, -34 and -50 on MNIST, CIFAR-10 datasets for exact 8-bit precision data and approximate data with 6-bit precision.

it to perform addition, subtraction, multiplication, and division operations individually through a given mode. Its processing elements or cores are displayed following a coarse-grained reconfigurable architecture employing a weight-stationary approach as a datapath. Each core can perform low-precision operations of 8 bits word-length. Performance is evaluated with Zynq UltraScale ZCU126. Compared with different arithmetic units aimed at CNN computations, the architecture shows improvement in the area, 1.22× lower resource utilization compared to the standard DNN/CNN accelerators. Further, it shows enhanced performance being 3.24× and 4.86× faster. Finally, it demonstrates 99.43 % and 82% accuracy on MNIST and CIFAR-10 datasets, respectively.

REFERENCES

[1] S. Bavikadi, A. Dhavlle, A. Ganguly, A. Haridass, H. Hendy, C. Merkel, V. J. Reddi, P. R. Sutradhar, A. Joseph, and S. M. Pudukotai Dinakarrao, "A survey on machine learning accelerators and evolutionary hardware platforms," *IEEE Design & Test*, vol. 39, no. 3, pp. 91–116, 2022.

[2] S. Bavikadi, P. R. Sutradhar, K. N. Khasawneh, A. Ganguly, and S. M. Pudukotai Dinakarrao, "A review of in-memory computing architectures for machine learning applications," in *GLSVLSI*, 2020.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[4] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017.

[5] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas," in *Annual Design Automation Conf.*, 2017.

[6] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2016.

[7] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[8] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE European Test Symp. (ETS)*, 2013.

[9] G. Sutter and J.-P. Deschamps, "High speed fixed point dividers for fpgas," in *Int. Conf. on Field Programmable Logic and Applications*, 2009.

[10] K. R. Gavali, N. Choudhary, S. Mishra, and S. Dubey, "A parallel pipelined adder suitable for fpga implementation," in *Int. Conf. on Smart City and Emerging Technology (ICSCET)*, 2018.

[11] C. Hanuman and J. Kamala, "Hardware implementation of 24-bit vedic multiplier in 32-bit floating-point divider," in *Int. Conf. on Electrical, Electronics and System Engineering (ICEESE)*, 2018.

[12] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. K. Prajapati, M. A. Indovina, S. M. Pudukotaidinakarrao, and A. Ganguly, "Look-up-table based processing-in-memoryarchitecture with programmable precision-scalingfor deep learning applications," *IEEE TPDS*, 2021.

[13] S. Bavikadi, P. R. Sutradhar, M. A. Indovina, A. Ganguly, and S. M. P. Dinakarrao, "Polar: Performance-aware on-device learning capable programmable processing-in-memory architecture for low-power ml applications," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 889–898.

[14] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. on Electronic Computers*, no. 4, pp. 512–517, 1962.

[15] J. Becker, T. Pionteck, C. Habermann, and M. Glesner, "Design and implementation of a coarse-grained dynamically reconfigurable hardware architecture," in *IEEE Workshop on VLSI*, 2001.

[16] M. Lanuzza, S. Perri, P. Corsonello, and M. Margala, "A new reconfigurable coarse-grain architecture for multimedia applications," in *NASA/ESA Conf. on Adaptive Hardware and Systems*, 2007.

[17] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *IEEE Int. Conf. on Computer Design (ICCD)*, 2013.

[18] M. Lanuzza, S. Perri, M. Margala, and P. Corsonello, "Low-cost fully reconfigurable data-path for fpga-based multimedia processor," in *Int. Conf. on Field Programmable Logic and Applications, 2005.* IEEE, 2005, pp. 13–18.

[19] Z. Ebrahimi, S. Ullah, and A. Kumar, "Simdive: Approximate simd soft multiplier-divider for fpgas with tunable accuracy," in *Great Lakes Symp. on VLSI*, 2020.

[20] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, Nov 2012.

[21] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (canadian institute for advanced research)," 2009.

[22] S. Ghosh, P. Bhattacharyya, and A. Dutta, "FPGA-based implementation of a double precision ieee floating-point adder," in *Int. Conf. on Intelligent Systems and Control (ISCO)*, 2013.

[23] S. Beohar and S. Nemade, "VHDL implementation of self-timed 32-bit floating point multiplier with carry look ahead adder," in *Int. Conf. on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2016.

[24] L. S. A. Hamid, K. Shehata, H. El-Ghitani, and M. ElSaid, "Design of generic floating point multiplier and adder/subtractor units," in *Int. Conf. on Computer Modelling and Simulation*, 2010.

[25] P. Malik, "High throughput floating-point dividers implemented in FPGA," in *IEEE Int. Symp. on Design and Diagnostics of Electronic Circuits & Systems*, 2015.

[26] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: a dynamic range unbiased multiplier for approximate applications," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, 2015.

[27] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 2623–2635, 2018.

[28] K. Vasquez, Y. Venkatesha, A. Bhattacharjee, A. Moitra, and P. Panda, "Activation Density based Mixed-Precision Quantization for Energy Efficient Neural Networks," *arXiv e-prints*, p. arXiv:2101.04354, Jan. 2021.