Area, Power, and Latency Optimized Approximate Integer Multipliers For Neural Networks In FPGAs

Vinu Sankar Sadasivan, Chandan Kumar Jha, and Joycee Mekie

***Abstract*—FPGAs are being used widely to cater to a variety of applications in recent years. Most of these applications have multiplications as one of the operations. For error resilient applications, approximate multipliers based on ASIC have been widely explored in recent years, as they provide energy, area and delay benfits, without significant loss in output quality. While AISC based designs have been extensively explored, in this paper we show how to design efficient approximate multipliers for FPGAs. We propose two novel 8-bit area, power, and latency optimized (APLO1 and APLO2) FPGA-based approximate multipliers to aid accelerated, energy-efficient inferencing of NN models, with an average accuracy dip as low as 2.2%. Our proposed 8-bit approximate multipliers occupy $\sim 38\%$ lesser area, consume $\sim 30\%$ lesser power and have $\sim 28\%$ lesser delay, on an average, as compared to the state-of-the-art designs. We propose a library of 81 different 16×16 multipliers using APLO1, APLO2, and accurate (8×8) multipliers as sub-modules. APLOLib allows designers to explore a wide space of multipliers with varying error, area, power, and delay to suit their requirements. Our approximate models are thoroughly analyzed and are used to deploy NNs on FPGA using a methodology exploiting quantization and normalization techniques for accelerating NNs on hardware.**

***Index Terms*—Approximate Computing, Multipliers, FPGAs, Neural Networks, Hardware Acceleration**

I. Introduction

NEURAL networks (NNs) have been shown to be robust towards inaccuracies in computations [1]. Many of the prior works have looked into algorithm level evaluation of the robustness of NNs towards the introduction of approximation [2]–[4]. We can thus obtain benefits in power, area, latency, and/or error depending upon the application for which the NNs are used. In medical diagnosis, minimizing error is of utter importance [5], whereas in mobile applications, power reduction is critical [6]. To cater to the wide plethora of demands, we need to come up with a wide range of designs with low error, power, area, and latency to allow design choice tailored to application's requirements. In this paper, we address the need for a library of approximate multiplier designs for implementing ML applications on FPGAs [7] using the Xilinx Kintex-7 KC705 board. Multiplications are performed frequently and form the heart of any NN. Approximate multiplier circuits for FPGAs have been relatively lesser explored as compared to ASIC based designs [1]. FPGAs have programmable gate arrays, which make them a great option for design prototyping, and can be exploited to perform highly parallelized tasks such as matrix multiplications in NNs. The digital signal processing blocks on FPGAs, which are optimized to perform multiplications and divisions tend to show performance degradation in some applications due to their fixed locations on the board [8]. This motivates us to efficiently utilize resources on FPGAs for applications where there are constraints in terms of power, area, and latency.

In this paper, we propose an Area, Power, and Latency Optimised library (APLOLib) containing a variety of approximate multiplier designs which allows design space exploration across wide ranges of delay, area, and power consumption with different error values. The main contributions of our work are:

- Two 8×8 approximate multiplier designs, APLO1 and APLO2, that exploit separate sets of hardware blocks on FPGA to predict each of the output product bit independently in a parallel fashion.
- A library of *81* different 16×16 design space explored multipliers using APLO1 (8×8), APLO2 (8×8), and accurate (8×8) multipliers as basic modules with a recursive approach.
- A methodology to deploy NNs using a hybrid of accurate and approximate multiplier units employing techniques like normalization and quantization to achieve area, latency, and power gains.

II. Related Works

Due to the growing interest in hardware ML, researchers have been trying to implement NNs using approximate multipliers to make fast and energy-efficient hardware [9]–[12]. In Xilinx FPGAs, configurable logic blocks are the fundamental blocks, and they are made up of look-up tables (LUTs) and carry-chains (CCs) [13]. Each LUT has an associated INIT value, which encodes the entire possible inputs for which the LUT outputs a value of 1. CCs serve the purpose of communication between LUT blocks on FPGA boards. In all our designs, we use LUTs with 6 inputs, that require a 64-bit INIT value. APLO multipliers have been designed carefully such that we can compute the results fast, using a minimal number of LUTs, and without the use of CCs, which has high power requirements. Kulkarni et al. [14] proposed an approximate 2×2 multiplier block using logic gates for non-error resilient applications. They also use a methodolgy [15] to build large multiplier blocks using their 2×2 multiplier blocks. [16] also uses a similar approach for their design space exploration.

V. S. Sadasivan is with the Department of Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India e-mail: vinu.sankar@iitgn.ac.in.

C. K. Jha and J. Mekie are with the Department of Electrical Engineering, Indian Institute of Technology Gandhinagar, India e-mail: chandan.jha@iitgn.ac.in, joycee@iitgn.ac.in.

This work is supported by Visvesvaraya PhD scheme and Intel India PhD Fellowship.

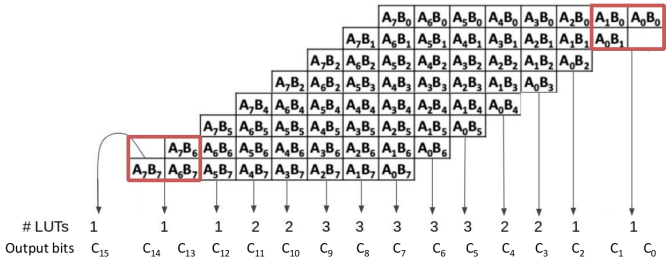

Fig. 1. Input bits grouped in columns and the total number of LUT(s) required to predict multiplier output bits in APLO2.

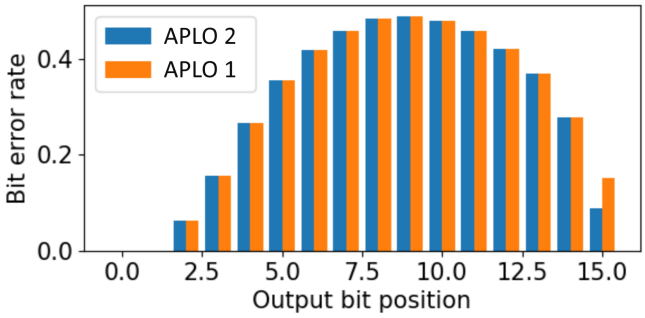

Fig. 2. Output bit error rates for APLO designs.

SMApproxLib [16] is the first open-source library of FPGA-based approximate multipliers. We have compared our designs with [16] as it is better in terms of area, power, and latency as compared to [17]. SMApproxLib with 3 8×8 designs (Approx1, Approx2, and Approx3), provide the state-of-the-art designs for approximate multipliers on FPGA. Hence, we choose SMApproxLib as our benchmark. The designs in SMApproxLib effectively use LUTs and CCs on FPGAs to optimize area and energy of multiplier units. They also propose a design exploration model, that can be used to combine accurate and approximate multiplier designs for improving performance. In [16], the authors report the relative error made by each of their designs after performing design exploration. Approx1, which is reported to be the best design in SMApproxLib, uses CCs to ripple carries. But the problem with this approach is that the error produced at an output bit will get cascaded to the next output bits. This makes the MSB prone to more error and hence worsening the approximation. The work [18] proposes approximate multiplier designs for ASIC while our APLO designs are tailored for FPGA in specific. Our proposed designs shows that simple LUT-based functions can be used to approximate integer multiplications in FPGA. In our work, we also perform an exhaustive design space exploration of 16×16 approximate multipliers built using APLO1 and APLO2 8×8 multiplier designs and analyze them based on their area, power, latency, and error benefits.

III. Neural Networks

A general l-layered MLP with one input layer, one output layer, and $(l-2)$ hidden layers is given by,

$$h_i = \sigma_i(W_i^T h_{(i-1)} + b_i) \tag{1}$$

where $x = h_0$ is the input and $y = h_{l-1}$ is the output, for $1 \leq i \leq l-1$. W_i's and b_i's are the weight matrices and the biases for the network. In the NN models in this paper, we choose all σ_i to be $ReLU$, except for $\sigma_{(l-1)}$ which is $softmax$. In this paper, we are interested in learning and inferencing using perceptron networks. Learning is a complex optimization task, wherein a back-propagation algorithm is employed to find the best weight and bias parameters, in a multi-dimensional space, that can be used to generalize the model well for a given training dataset.

IV. APLO Approximate Multipliers

Conventional shift-and-add multipliers use compressors [1], that consume a large amount of power and area, to find the sum of partial products to obtain results. Additionally, the carry value that needs to be propagated from the least significant bit to the most significant bit decides the critical path, making the conventional multipliers very slow. We solve both these problems of conventional multipliers in APLO designs and show how these designs reduce the consumption of power, area, and computation time significantly on a Xilinx Kintex-7 KC705 FPGA board, with errors rates comparable to the state-of-the-art approximate multiplier designs [16] on FPGA. We compare the approximations in each design in terms of average relative error, which is given as,

$$Average\ Relative\ Error\ = \frac{1}{N}\sum_{i=1}^{N}\frac{|\ Accurate\ result_i - Approximate\ result_i\ |}{Accurate\ result_i} \tag{2}$$

where N denotes the total number of distinct combinations of two inputs that can be input to the multiplier unit, given both of the inputs are non-zero.

Our key idea is to get rid of the CCs in exact multipliers, which is the bottleneck to latency and power in product computations. Moreover, approximations with a reduced number of LUTs will help us minimize the area of the design. Either by neglecting or approximately predicting the carry, we can avoid the use of CCs. To reduce the LUT usage and avoid CCs, we have tailor-made the logic functions which approximately mimic a compressor. APLO designs take inputs A (8 bits) and B (8 bits) to compute the approximate output C (16 bits). APLO1 design for obtaining output bits $[C_7...C_0]$ is shown in Table I, with LUTs and their INIT values. The accurate product bits $\{D_i\}$, where $D = A \times B$, can be computed using the help of logical functions $\{m_i\}$ given as,

$$D_i = \begin{cases} m_i(A_i,\ ...,A_0,\ B_i,\ ...,\ B_0) & \text{if } 0 \leq i \leq 7 \\ m_i(A_7,\ ...,A_0,\ B_7,\ ...,\ B_0) & \text{if } 8 \leq i \leq 15 \end{cases} \tag{3}$$

In this paper, we try to approximate each $m_i(x)$ using $m_i^{APLO}(x)$ by minimizing LUTs and avoiding CCs, where $x \in \{0,1\}^{min(2i+2,\ 32)}$. APLO multiplier LUT designs for computing $C_i = m_i^{APLO}(A_i, ..., A_0, B_i, ..., B_0)$ is given in Table I. We split the accurate multiplier into columns to group partial products as shown in Figure 1, and employ independent approximate adders to compute each of the approximate output product bits. This makes our multipliers faster due to its ability to process output bits in a parallel fashion.

TABLE I
LUTS WITH INPUT PINS $I_0 - I_5$, AND OUTPUT PINS O_5 AND O_6 WITH THEIR INIT VALUES USED FOR APLO 8×8 DESIGNS. A AND B ARE THE INPUTS AND C IS THE OUTPUT. * REPRESENTS THE LUT USED FOR PREDICTING THE MSB IN THE APLO2 DESIGN.

LUT ID	INIT value	I_0	I_1	I_2	I_3	I_4	I_5	O_5	O_6
L_0	6CA0888888888888	A_0	B_0	B_1	A_1	1	1	C_0	C_1
L_1	8777788878887888	A_2	B_0	A_1	B_1	A_0	B_2	-	C_2
L_2	8777788878887888	A_3	B_0	A_2	B_1	A_1	B_2	-	-
L_3	8777788878887888	A_0	B_3	$L_2(O_6)$	$L_2(O_6)$	1	1	-	C_3
L_4	8777788878887888	A_4	B_0	A_3	B_1	A_2	B_2	-	-
L_5	8777788878887888	A_1	B_3	A_0	B_4	$L_4(O_6)$	$L_4(O_6)$	-	C_4
L_6	8777788878887888	A_5	B_0	A_4	B_1	A_3	B_2	-	-
L_7	8777788878887888	A_2	B_3	A_1	B_4	A_0	B_5	-	-
L_8	8777788878887888	$L_6(O_6)$	$L_6(O_6)$	$L_7(O_6)$	$L_7(O_6)$	1	1	-	C_5
L_9	8777788878887888	A_6	B_0	A_5	B_1	A_4	B_2	-	-
L_{10}	8777788878887888	A_3	B_3	A_2	B_4	A_1	B_5	-	-
L_{11}	8777788878887888	A_0	B_6	$L_9(O_6)$	$L_9(O_6)$	$L_{10}(O_6)$	$L_{10}(O_6)$	-	C_6
L_{12}	8777788878887888	A_7	B_0	A_6	B_1	A_5	B_2	-	-
L_{13}	8777788878887888	A_4	B_3	A_3	B_4	A_2	B_5	-	-
L_{14}	7888877787777888	A_1	B_6	A_0	B_7	$L_{12}(O_6)$	$L_{13}(O_6)$	-	C_7
L_{15}	8777788878887888	A_7	B_1	A_6	B_2	A_5	B_3	-	-
L_{16}	8777788878887888	A_4	B_4	A_3	B_5	A_2	B_6	-	-
L_{17}	8777788878887888	A_1	B_7	$L_{15}(O_6)$	$L_{15}(O_6)$	$L_{16}(O_6)$	$L_{16}(O_6)$	-	C_8
L_{18}	8777788878887888	A_7	B_2	A_6	B_3	A_5	B_4	-	-
L_{19}	8777788878887888	A_4	B_5	A_3	B_6	A_2	B_7	-	-
L_{20}	8777788878887888	$L_{18}(O_6)$	$L_{18}(O_6)$	$L_{19}(O_6)$	$L_{19}(O_6)$	1	1	-	C_9
L_{21}	8777788878887888	A_7	B_3	A_6	B_4	A_5	B_5	-	-
L_{22}	8777788878887888	A_4	B_6	A_3	B_7	$L_{21}(O_6)$	$L_{21}(O_6)$	-	C_{10}
L_{23}	8777788878887888	A_7	B_4	A_6	B_5	A_5	B_6	-	-
L_{24}	8777788878887888	A_4	B_7	$L_{23}(O_6)$	$L_{23}(O_6)$	1	1	-	C_{11}
L_{25}	8777788878887888	A_7	B_5	A_6	B_6	A_5	B_7	-	C_{12}
L_{26}	6CA0888888888888	A_6	B_6	B_7	A_7	1	1	C_{13}	C_{14}
L_{27}*	8000800080008000	A_7	B_7	A_6	B_6	1	1	-	C_{15}

The number of LUTs used for predicting each column output bit is shown in Figure 1 with their INIT values in Table I. A single LUT is used to accurately compute both C_0 and C_1. The LUT used to predict C_2 performs computations to add the XOR sum of 3 partial products given by $A_2.B_0 \oplus A_1.B_0 \oplus A_0.B_2$. This LUT assumes the carry from the first column (that predicts C_0), given by $A_1.B_0.A_0.B_1$, to be 0 since there is only 6.25% probability for the carry to be 1 (when $A_0 = A_1 = B_0 = B_1 = 1$). This is interesting as a single 6 input LUT is able to predict the output correctly with an error rate of just 0.0625 (see Figure 2). This motivates us to use the same LUT function $I_0.I_1 \oplus I_2.I_3 \oplus I_4.I_5$ in a cascading manner to predict more significant output bits as shown in Table I. C_{15} is always forced to be 0 in APLO1. APLO2 is obtained by adding an extra LUT to APLO1, to predict the MSB (C_{15}). The function used for predicting MSB in APLO2 is shown by L_{27} in Table III. It is visible that an addition of one LUT to predict the MSB improves the average relative error by 0.039 as shown in Table II. But the error rate can be further improved by employing more such LUTs to predict the MSB. This can lower the error rates to a greater degree, but at the same time increase power and area consumption.

V. APLOLIB

We have performed a design space exploration of 16×16 multipliers. A 16×16 multiplier can be built using four 8×8 multipliers as sub-modules. Since we have proposed 2 different designs, APLO1 and APLO2, for each of the four sub-module multipliers, we can have 3 possible options (APLO1, APLO2, accurate) to gives us a total of *81* 16×16 approximate multipliers. Similarly, we also implemented *256* approximate multiplier design space of [16] and compared them to our designs in Figure 3 and 4. It is clear from the design space exploration shown in Figure 3 and 4 that some designs obtained using APLOLib clearly outperform the designs obtained from SMApproxLib in terms of area, power, and latency for similar error rates. Some of the designs obtained from SMApproxLib have some of the lowest error rates in the design space, but the energy requirements are very high for these. In this era of EdgeML, when researchers are trying to deploy ML algorithms on edge devices with resource constraints by developing efficient NN architectures like Bonsai [19], we show that approximations can be made at inference level too to aid the purpose. We provide a library of designs from which a designer could choose designs to trade-off between area, power, latency, and accuracy constraints. We obtain power benefits with some drop in the classification accuracies of the NNs as being analyzed in Table II.

VI. RESULTS

A. Approximate multipliers

All multiplier designs are implemented using Verilog HDL in Xilinx Vivado on Xilinx Kintex-7 KC705 FPGA board. We used Vivado simulator and Power Analyzer for calculating power, area, and latency values. In this paper, we have

TABLE II
COMPARISON OF AREA, POWER, LATENCY, AND AVERAGE RELATIVE ERROR VALUES OF 8 × 8 MULTIPLIER DESIGNS ALONG WITH ACCURACIES (IN %) ON VARIOUS TEST DATASET.

Designs	Area (LUTs)	Power (W)	Latency (ns)	Average relative error	MNIST		Fashion-MNIST		ISOLET		USPS	
					H_0	H_1	H_0	H_1	H_0	H_1	H_0	H_1
Accurate	56	0.980	11.809	**0.0**	**93.0**	**98.8**	**87.8**	91.4	**94.6**	95.8	**87.8**	**90.3**
a [16]	43	0.664	7.621	0.540	89.8	98.0	84.2	90.2	91.4	88.4	87.0	82.8
d [16]	35	0.595	7.276	0.527	89.4	98.2	84.2	89.8	91.6	88.6	87.0	83.0
o [16]	37	0.638	7.219	0.527	89.8	98.4	84.0	90.0	91.6	88.6	87.0	83.0
aacc [16]	55	0.923	10.716	0.019	93.2	**98.8**	**87.8**	**91.8**	94.4	95.8	**87.8**	**90.3**
dacc [16]	47	0.780	10.200	0.028	92.8	**98.8**	87.6	91.6	**94.6**	**96.0**	87.7	**90.3**
oacc [16]	46	0.827	10.686	0.028	**93.0**	**98.8**	87.4	91.4	**94.6**	95.8	**87.8**	**90.3**
APLO1	**27**	0.519	**6.408**	0.311	89.6	98.6	84.2	89.6	91.4	92.4	86.6	88.7
APLO2	**27**	**0.510**	6.438	0.272	89.6	98.6	84.4	89.6	92.0	92.2	86.9	89.0

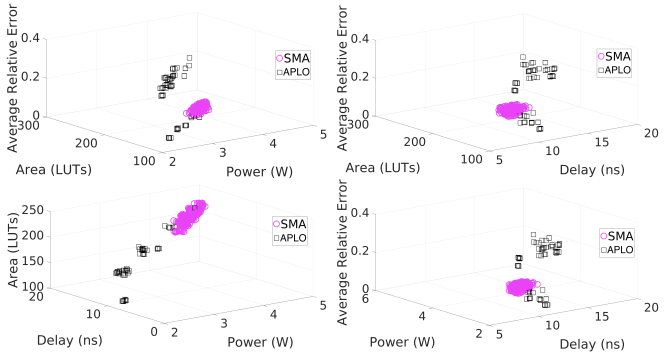

Fig. 3. 3D plots comparing design explored space of APLOLib (*81* designs) and SMApproxLib (*256* designs).

performed an in-depth comparison of APLO1 and APLO2 with six different designs proposed in [16]. The a, d, o [16] in use approximate adders and aacc, dacc, oacc [16] in use accurate adders. Results (in Table II) indicate that savings for our designs in area, power, and delay are ∼38%, ∼30%, and ∼28%, respectively as compared to the state-of-the-art.

B. Approximate NNs

We use Python 3.6 for implementing approximate NNs using the approximate multipliers discussed in the paper to take advantage of their error-resilient property for area, power, and latency gains. We deploy the multipliers and analyze them on 3 different datasets MNIST, ISOLET, Fashion-MNIST, and USPS to show their robustness as shown in Table II. For each dataset, we design a single layer feed-forward fully-connected perceptron network and a 2-layer MLP. Since we use 8×8 approximate integer multiplier units to replace accurate 32-bit float multiplier units for matrix multiplications in NNs, we perform matrix normalization and quantization, as discussed in Section III. The weight matrices of the pre-trained networks are normalized and quantized to range 8-bit integers using the accurate quantizing unit, as shown in Figure 5. In this manner, we perform all the 2D matrix multiplications using approximate multiplier units and perform vector normalization and quantization using accurate units. $ReLU$ can be implemented

TABLE III
NN ARCHITECTURES FOR EACH DATASET.

Dataset	MNIST	Fashion-MNIST	ISOLET	USPS
$\mathbf{H_0}$	784-10	784-10	617-26	256-10
$\mathbf{H_1}$	784-512-10	784-256-10	617-128-26	256-500-10

on FPGAs using comparators with ease. The fully-connected network architectures with number of neurons in each layer (H_0 and H_1) used for training for each dataset are shown in Table III. Output layers use $softmax$ activation and hidden layers use $ReLU$ activation. As discussed in Section III, $softmax$ activation is not added to NNs while deploying them. Softmax cross-entropy loss [20] is used to train all the networks.

VII. CONCLUSION

In this paper, we have proposed two novel 8×8 approximate multiplier designs for FPGAs with area, power, and latency optimization and showed the application of these multipliers in NN models. Our multipliers, when deployed on NNs, show only a dip of $\mathbf{1.8}\%$, $\mathbf{2.7}\%$, $\mathbf{3.2}\%$, and $\mathbf{1.3}\%$ accuracy on MNIST, Fashion-MNIST, ISOLET, and USPS datasets, respectively. We use a hybrid of accurate and approximate multiplier units to design a methodology to deploy NNs for exploiting the benefits of the designed approximate multiplier units using techniques like approximate computing, normalization, and quantization. NNs with zero (H_0) and one (H_1) hidden layer show a drop of $\mathbf{2.7}\%$ and $\mathbf{1.7}\%$ accuracy, respectively. Hence, the area, power, and latency gains ($\sim \mathbf{38}\%$, $\sim \mathbf{30}\%$, and $\sim \mathbf{28}\%$, respectively) of our approximate multipliers over the accurate multipliers sum up to give significant benefits for hardware NNs with a negligible drop in accuracy performance of the NN models. We also provide APLOLib, a library of *81* 16×16 approximate multipliers based on the APLO designs, with a wide range of area, power, and latency values for different errors.

REFERENCES

[1] Honglan Jiang, Leibo Liu, Fabrizio Lombardi, and Jie Han, *Approximate Arithmetic Circuits: Design and Evaluation*, pp. 67–98, Springer International Publishing, Cham, 2019.

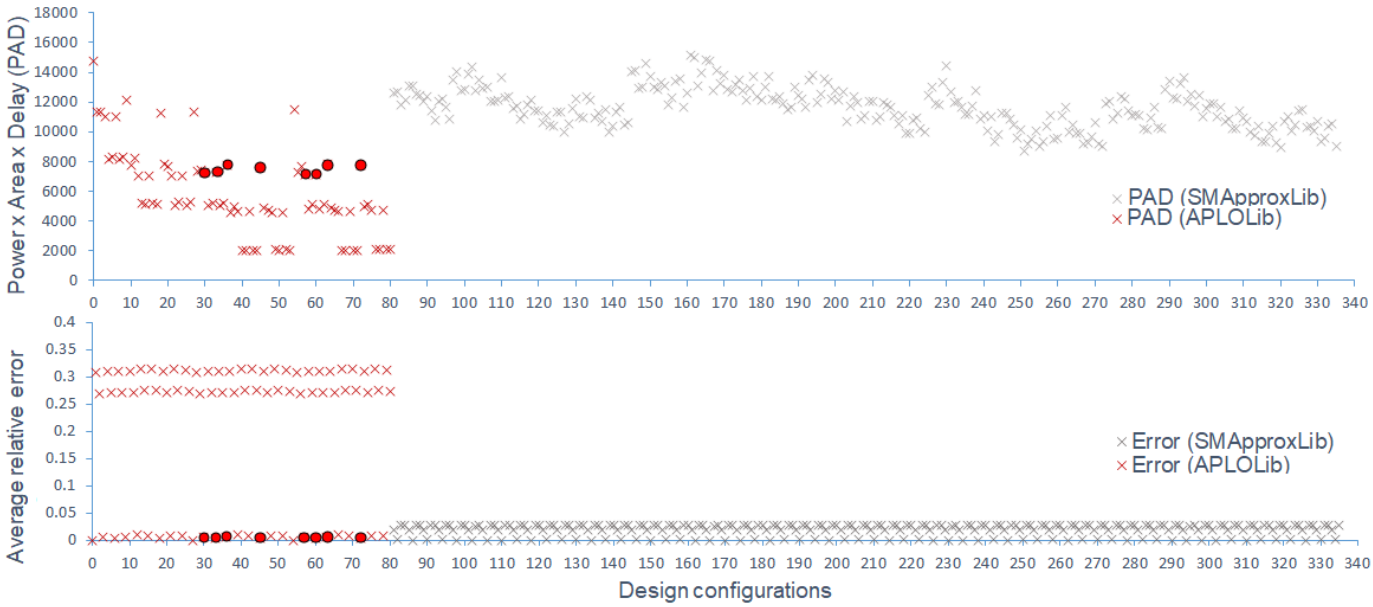

Fig. 4. Comparing Power×Area×Delay (PAD) and average relative error of all the explored design configurations. APLOLib designs with error lower than 0.005 and lower PAD than SMApproxLib are marked in red circles.

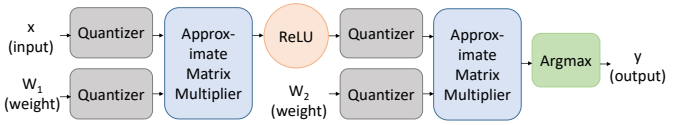

Fig. 5. NN deployment methodology for FPGAs using approximate multiplier and adder units for matrix operations.

[2] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2014, pp. 27–32.

[3] Patrick Judd, Jorge Albericio, Tayler H. Hetherington, Tor M. Aamodt, Natalie D. Enright Jerger, Raquel Urtasun, and Andreas Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," *CoRR*, vol. abs/1511.05236, 2015.

[4] Adarsha Balaji, Salim Ullah, Anup Das, and Akash Kumar, "Design methodology for embedded approximate artificial neural networks," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, New York, NY, USA, 2019, GLSVLSI '19, pp. 489–494, ACM.

[5] V. S. Sadasivan and C. S. Seelamantula, "High accuracy patch-level classification of wireless capsule endoscopy images using a convolutional neural network," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, April 2019, pp. 96–99.

[6] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[7] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2015, FPGA '15, pp. 161–170, ACM.

[8] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[9] Uroš Lotrič and Patricio Bulić, "Applicability of approximate multipliers in hardware neural networks," *Neurocomputing*, vol. 96, pp. 57 – 65, 2012, Adaptive and Natural Computing Algorithms.

[10] Z. Liu, A. Yazdanbakhsh, T. Park, H. Esmaeilzadeh, and N. S. Kim, "Simul: An algorithm-driven approximate multiplier design for machine learning," *IEEE Micro*, vol. 38, no. 4, pp. 50–59, Jul 2018.

[11] Adarsha Balaji, Salim Ullah, Anup Das, and Akash Kumar, "Design methodology for embedded approximate artificial neural networks," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, New York, NY, USA, 2019, GLSVLSI '19, pp. 489–494, ACM.

[12] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–7.

[13] *Xilinx. 2017. Xilinx 7 Series Configurable Logic Block. https://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf.*

[14] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, Jan 2011, pp. 346–351.

[15] Israel Koren, *Computer Arithmetic Algorithms*, A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2001.

[16] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar, "Smapprox-lib: Library of fpga-based approximate multipliers," in *Proceedings of the 55th Annual Design Automation Conference*, New York, NY, USA, 2018, DAC '18, pp. 157:1–157:6, ACM.

[17] Salim Ullah, Semeen Rehman, Bharath Srinivas Prabakaran, Florian Kriebel, Muhammad Abdullah Hanif, Muhammad Shafique, and Akash Kumar, "Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, New York, NY, USA, 2018, DAC '18, pp. 159:1–159:6, ACM.

[18] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, Nov 2018.

[19] Ashish Kumar, Saurabh Goyal, and Manik Varma, "Resource-efficient machine learning in 2 KB RAM for the internet of things," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1935–1944.

[20] Zhilu Zhang and Mert Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 8778–8788. Curran Associates, Inc., 2018.