# CANN: Curable Approximations for High-Performance Deep Neural Network Accelerators

Muhammad Abdullah Hanif, Faiq Khalid, Muhammad Shafique
Technische Universität Wien (TU Wien), Vienna, Austria
{muhammad.hanif,faiq.khalid,muhammad.shafique}@tuwien.ac.at

## ABSTRACT

Approximate Computing (AC) has emerged as a means for improving the performance, area and power-/energy-efficiency of a digital design at the cost of output quality degradation. Applications like machine learning (e.g., using DNNs-deep neural networks) are highly computationally intensive and, therefore, can significantly benefit from AC and specialized accelerators. However, the accuracy loss introduced because of approximations in the DNN accelerator hardware can result in undesirable results. This paper presents a novel method to design high-performance DNN accelerators where approximation error(s) from one stage/part of the design is "completely" compensated in the subsequent stage/part while offering significant efficiency gains. Towards this, the paper also presents a case-study for improving the performance of systolic array-based hardware architectures, which are commonly used for accelerating state-of-the-art deep learning algorithms.

## KEYWORDS

Approximate Computing, Neural Network, MAC, DNN, High-Performance, Accelerator, Energy Efficiency, Power Efficiency

## 1 INTRODUCTION

Since technology scaling has started offering diminishing returns, Approximate Computing (AC) has emerged as an alternative paradigm for further improving the performance, power/energy, and area efficiency of the inherently error-resilient applications. This is achieved by relaxing the bounds of output accuracy and introducing tolerable quality loss for gaining significant advantage in terms of desired efficiency [15]. Multiple approximation techniques have been proposed at different abstraction layers of the computing stack [17]. At software layer, techniques like code perforation and code approximation are commonly employed while, at architecture/hardware layer, techniques like approximation of the functional units and voltage underscaling are commonly used [15].

In this paper, we focus on the hardware level techniques for improving the performance and energy/power efficiency of the designs. Multiple techniques have been proposed to design approximate circuits, for example, systematic logic synthesis of approximate circuits (SALSA) [20] and automated behavioral synthesis of approximate computing circuits (ABACUS) [16]. Other approximation techniques have also been proposed which build approximate accelerators using elementary approximate modules, such as approximate adders and approximate multipliers [3]. A method for building adaptive approximate datapaths have also been recently proposed in [14] which reduces the extent of approximation error by adaptively selecting the type of module in the subsequent stage/s. Techniques, such as [5], have also been introduced that tune the software models, in this case neural networks, in light of the underlying hardware approximations. Such methods put additional constraints on the training process and thereby limits the learning capability of the models, specifically in case of complex problems. *Although all these techniques result in significant improvement in the performance, area and power/energy efficiency of a system, they achieve this at the cost of some output quality loss which can be tolerated in many error-resilient applications, but cannot be tolerated in safety-critical applications where even a slight inaccuracy in output can result in catastrophic effects.* One such example is autonomous driving where machine learning algorithms are used for interpreting the surrounding environment of an autonomous vehicle and also used for decision making [12]. *This significantly limits the scope of approximate computing being employed for highly-critical applications.*

**Motivational analysis:** To illustrate the effects of hardware approximations on machine learning algorithms, we consider an image classification application using deep neural networks (DNNs) where the network is executed using hardware composed of approximate multipliers. For this analysis, we assumed the LeNet network (provided by MatConvNet [19]) trained on the cifar-10 dataset [9]. The network is quantized to 8-bit fixed point format, i.e., both the weights and activations are represented using 8-bit fixed point numbers, to reduce the complexity of the underlying hardware components. The multiplier designs used are based on the design presented in [4] where larger multipliers are constructed using smaller 2x2 accurate and approximate multipliers, while assuming accurate partial product accumulation. For building approximate multipliers the least significant 2x2 multipliers are implemented using the approximate 2x2 multiplier design proposed in [10]. Here we consider three approximate, i.e., type 1, 2, and 3, multipliers and one accurate multiplier. In approximate type 1, 2 and 3 multipliers, the least significant one, three, and four 2x2 multipliers were realized using the approximate 2x2 multiplier design, respectively. The error and hardware characteristics of the considered 8x8 multipliers are shown in Table 1. The approximation error of each multiplier is represented in terms of Mean Error Distance (MED) and is computed assuming uniform input distribution.

Fig. 1 shows the impact of approximation on the classification accuracy of the network. Note that, for this analysis, all the multipliers deployed at one time in the hardware are assumed to be of the same


*DAC '19, June 2–6, 2019, Las Vegas, NV, USA*

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317787

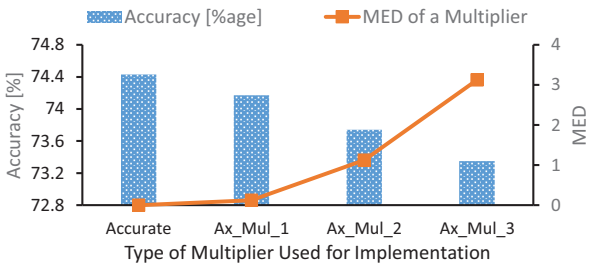



**Figure 1: An analysis illustrating the effects of approximation in the multipliers on the overall accuracy of the image classification application using the LeNeT network on the cifar-10 dataset.**

type. It can be seen from the figure that the classification accuracy of the network decreases with the increase in the approximation level. It can also be observed that the classification accuracy decreases even for the least approximate variant of the hardware. *Therefore, there is a need for designing approximate hardware such that the effects of approximations can be compensated internally, thereby allowing to achieve significant performance and/or energy/power efficiency while providing accurate/near-accurate results which have no impact on the output accuracy of the safety-critical applications.*

## 1.1 Novel Contribution

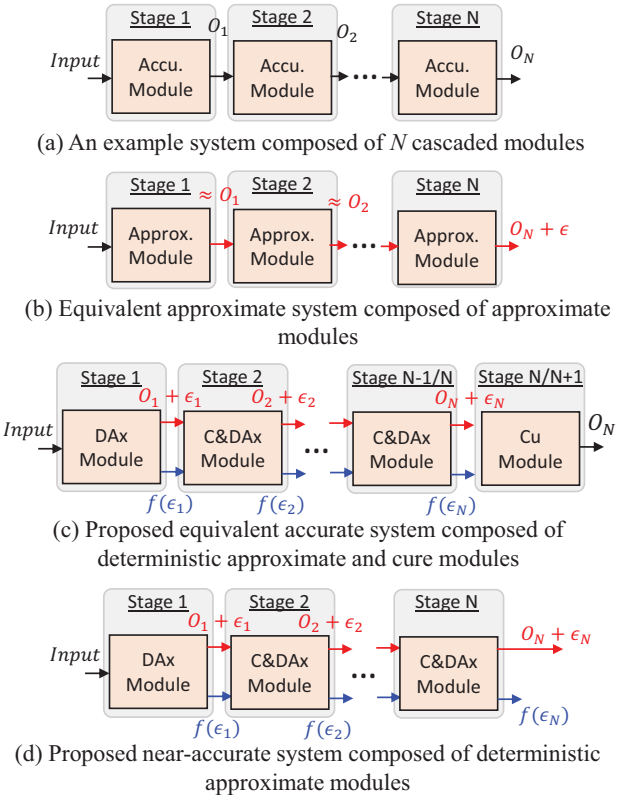
In the light of the above discussion, following are the main novel contributions of the paper.

(1) We present a novel method for designing accurate and near-accurate systems using approximate modules by designing modules with error curing characteristics.
(2) We present a case-study on systolic array-based specialized architectures which are highly effective for machine learning applications and can significantly benefit from the proposed method for improving the performance of machine learning based systems.
(3) We present novel designs of Multiply-and-Accumulate (MAC) unit for all types of modules required for building a systolic array-based hardware using the proposed methodology.

We also present results and analysis highlighting the effectiveness of the proposed technique.

**Table 1: Error and hardware characteristics of different multipliers used for implementing the LeNet network for classifying the cifar-10 images. The hardware results are generated for 65 nm technology node using Cadence Genus tool with TSMC 65 nm library.**

| | Latency [$ps$] | Area [cell area] | Power [$\mu W$] | MED |
|---|---|---|---|---|
| Accurate | 1966.3 | 746 | 46.81 | 0 |
| Approx_Mul_1 | 1915.9 | 710 | 45.64 | 0.125 |
| Approx_Mul_2 | 1738.1 | 689 | 45.4 | 1.125 |
| Approx_Mul_3 | 1728.5 | 682 | 44.87 | 3.125 |




(a) An example system composed of $N$ cascaded modules

(b) Equivalent approximate system composed of approximate modules

(c) Proposed equivalent accurate system composed of deterministic approximate and cure modules

(d) Proposed near-accurate system composed of deterministic approximate modules

**Figure 2: Methods for building systems with cascaded modules. Here, $f(\epsilon_i)$ represents a reversible function of the error from the $i^{th}$ stage, i.e., $\epsilon_i$, which represents the error in a compressed form.**

## 1.2 Paper Organization

Section 2 presents our proposed methodology for designing accurate/near-accurate systems using approximate modules. In Section 3, we present a case study on building a DNN accelerator using the proposed methodology and also present the improvements compared to the conventional design. At the end, Section 4 concludes the paper.

# 2 METHODOLOGY FOR DESIGNING HARDWARE WITH CURABLE APPROXIMATIONS

In this section, we present our novel methodology for building approximate hardware. The methodology utilizes modules with curable error characteristics which accept error signal/s in compressed form from their previous stage along with the inputs, compensate for the error, and generate an approximate output with a compressed error signal containing the information about the error in the current stage, which should be compensated in the subsequent stage.

Fig. 2a shows a reference system composed of $N$ cascaded stages/modules. An approximate version of the system is illustrated in Fig. 2b, where all the modules are approximated to achieve

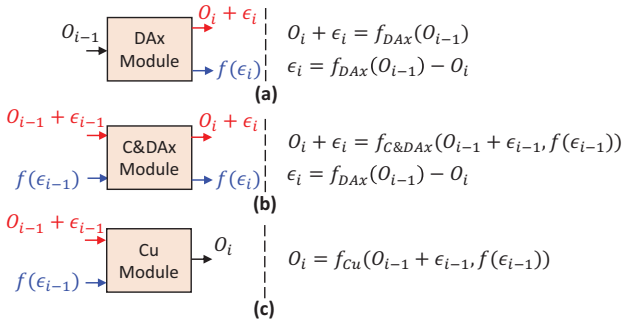



**Figure 3: Functionality of different modules used in Fig. 2. $O_i$ represents the accurate expected output and $\epsilon_i$ represents the approximation error generated by the $i^{th}$ stage. The functions $f_{DAx}(.)$ and $f_{C\&DAx}(.)$ are approximate variants of the corresponding accurate module and $f_{Cu}(.)$ can also be a variant of the corresponding accurate module or just an additional correction module. $f(\epsilon_i)$ represents a reversible function of the error from the $i^{th}$ stage.**

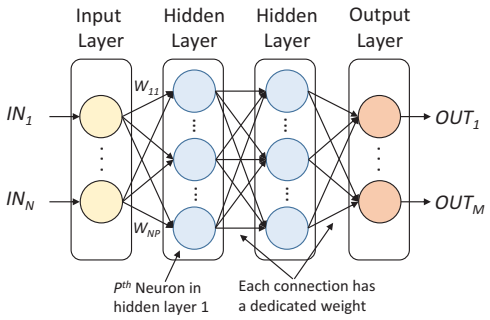



**Figure 4: Illustration of a fully connected neural network.**

significant efficiency gains. As illustrated in the figure, each module generates output with some level of inaccuracy and thereby adds some amount of uncertainty in the overall output. The resultant output of the system is not accurate and can deviate significantly from the desired output based on the number of stages and the amount of approximation in each stage. Therefore, such design methods are unusable for many safety-critical applications and other high precision computing scenarios.

Fig. 2c illustrates our proposed variant of the reference system. The system is composed of three types of modules: 1) Deterministic Approximate (DAx) module (fig. 3a); 2) Cure & Deterministic Approximate (C&DAx) module (fig. 3b); and 3) Cure (Cu) module (fig. 3c). The DAx module generates approximate output along with a compressed yet deterministic error signal which can be used by the subsequent stage to decipher and compensate for the exact amount of error occurred in the previous stage. The C&DAx module compensates for the error in the previous stage based on the compressed signal and generates an output and a compressed error signal for the subsequent stage. To generate an accurate output, the last stage is required to be a cure, i.e., Cu, stage which compensates for the error produced in the second to the last stage. Note that in some cases the Cu stage can be the $N^{th}$ stage while in others, where it is not possible to design a cure stage while meeting the required functionality, an additional stage, i.e., $N + 1^{th}$, is introduced to generate the accurate output. However, an alternative to this can be to not use the cure stage. This introduces a small error equivalent to the approximation error of the last stage, as shown in Fig. 2d. *Using the proposed methodology, unlike the system in Fig. 2b, the system in Fig. 2c (Fig. 2d) produces accurate (near-accurate) output while benefiting from the approximations in the modules.*

# 3 CASE STUDY

In this section, we discuss the implementation of an accurate high-performance and energy-efficient systolic array-based DNN accelerator using approximate modules by exploiting the proposed methodology. In the upcoming subsections, we first present a brief overview of the neural networks (Section 3.1) which is followed by the designs of different types of MAC units required in the implementation based on the proposed methodology (Section 3.2). Using the elementary modules, a high-performance systolic array architecture is proposed in section 3.3 which is followed by the results in Section 3.4.

## 3.1 Overview of Neural Networks

A Neural network is an interconnected network of nodes called neurons where the operation of a neuron can be represented by Eq. 1.

$$Output = F(\sum_{i=0}^{M} W_i * A_i + b) \quad (1)$$

Here, $W_i$s represents the weights, $b$ represents the bias, and $A_i$s represents the input activations of a neuron. The $F(.)$ represents the activation function for introducing non-linearity in the network model. An example illustration of a fully connected neural network is shown in Fig. 4.

There are many types of neural networks specialized for different applications. However, without any loss of generality, in this paper, for explanations we consider Convolutional Neural Networks (CNNs), as used in Section 1.

CNNs are composed of multiple types of layers, i.e., convolutional, fully-connected, pooling, and activation layers, which are connected in cascade to form a network. Out of all the layers the convolutional and fully-connected layers are the most computationally intensive. The basic operation used in these layers is a multiply-and-accumulate (MAC) operation, as represented by Eq. 1. Therefore, the state-of-the-art accelerators [13][8][2][11] mainly focus on accelerating the convolutional and fully-connected layers using arrays of processing elements that can perform large number of MAC operations in parallel. A more detailed overview of the neural networks and DNN hardware accelerators can be found in [18].

## 3.2 Designs of Required MAC units

To apply the proposed methodology on DNN accelerators, we first design required MAC units and later integrate them for building a complete computational array similar to one of the state-of-the-art DNN accelerators, i.e., Tensor Processing Unit (TPU) [8]. For this case study, we consider 8-bit fixed-point multiplication and

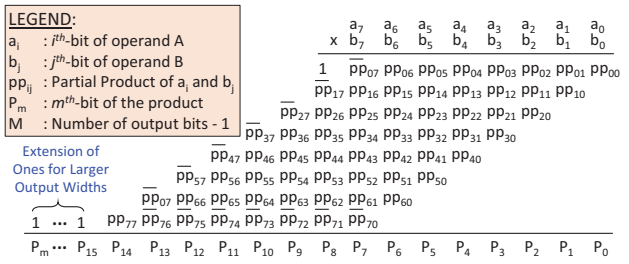



**Figure 5: An 8x8 signed multiplication based on Baugh-Wooley algorithm.**

the maximum possible partial sum size (within the array) to be 19-bit, which is sufficient for an array of size 8x8. Note that, in neural networks, 8-bit weights and activations are considered sufficient for achieving significant accuracy [7][6]. An exemplar computational array is shown in Fig. 7b and will be discussed in the next subsection.

Fig. 5 shows the multiplication of two 8-bit signed operands using Baugh-Wooley multiplication algorithm [1]. The sign extension can be performed by either extending the $P_{15}$ bit directly or by adding 1*s* at the most significant location of the last partial product. We make use of the latter along with the Wallace tree architecture to design different required types of high-performance signed MAC units.

Fig. 6 shows the designs of accurate, DAx, and C&DAx MAC units by illustrating the accumulation steps for accumulating the partial-products and the partial sum. The partial-products are generated from the multiplication of a weight and an activation based on the Baugh-Wooley algorithm (as shown in Fig. 5) and the partial sum (19-bit number) is the output from another MAC unit. Note that here we consider a merged MAC design where the partial-products (from the multiplication) and the partial sum are added simultaneously rather than performing the complete multiplication first and then adding the partial sum. Each MAC design has five accumulation steps where the first four steps are the compression steps followed by the final addition step. The compression steps use full and half adders as compressors for partial product reduction and the final step uses a multi-bit Ripple Carry Adder (RCA) for adding the final two arrays of bits. Fig. 6a shows the accumulation steps of the *Accurate Merged* MAC. For building DAx (Fig. 6b), the final addition (in the $5^{th}$ step) of the accurate merged MAC is divided into two parts to improve the performance of the MAC unit, as the final addition is the most delay intensive step. The compressed error signal $f(\epsilon)$, which is the carry out from the least-significant addition, is generated along with the approximate output for the subsequent module. For building C&DAx MAC (Fig. 6c) from accurate merged MAC, the error bit ($f(\epsilon)$) is added in the $1^{st}$ compression step at the corresponding significance location by replacing a half adder with a full adder. Through this the error generated in the previous stage is compensated. However, to improve the performance, the final addition (in the $5^{th}$ step) is truncated, similar to DAx MAC shown in Fig. 6b. For this scenario, the cure (Cu) module is composed of an RCA equivalent to the length of the most significant RCA in the $5^{th}$ step of the DAx and the C&DAx MAC units, i.e., almost half the length of the RCA used in the accurate merged MAC.

## 3.3 Systolic Array Design

Fig. 7b shows the systolic array design similar to the one used in the TPU architecture [8]. The systolic array is composed of multiple processing elements (PEs). The architecture of a single PE is shown in fig. 7a. In the array, each PE is connected to its neighboring PE in a manner that it receives activations from its left neighboring PE (or input) and partial sums from the above neighboring PE. The weights are communicated to the respective PEs using vertical channels, i.e., from top to bottom, and are stored inside the PEs during the computation.

Fig. 7d shows the modified design of the systolic array based on the proposed methodology. The architecture of the PEs used in it is shown in Fig. 7c. As the partial sums are communicated from top to bottom in the array, the first row of the PEs is replaced with approximate PEs composed of DAx MAC shown in Fig. 6b. Rest of all the PEs are replaced with approximate PEs with compensation capability composed of C&DAx MAC shown in fig. 6c. An additional row is added at the bottom of the processing array to compensate for the error generated in the last row of approximate PEs containing C&DAx MAC units. The additional row is composed of Cu modules which are adders of size equivalent to the size of the most-significant adder in step 5 of fig. 6b and c.

Note that the adopted approximations are orthogonal to most of the architecture-level approximations and, therefore, can also be used in conjunction with them for significantly improving performance and power/energy efficiency in case of error-resilient applications.

## 3.4 Results

In this subsection, we provide the performance, area and power results of the proposed systolic-array design (from henceforth referred to as *Proposed* systolic array), which is based on the proposed methodology. To compare the methodology with the state-of-the-art, we consider the conventional systolic array design (from henceforth referred to as *Conventional* systolic array) based on [8], also shown in Fig. 7b. We also consider a systolic array composed of PEs containing accurate merged MAC units (from henceforth referred to as *Merged Accurate* systolic array) and an array composed of approximate PEs (from henceforth referred to as *Approximate* systolic array). The approximate systolic array uses type 3 approximate multipliers (*Approx_Mul_3*) from section 1 in the PEs. The overall architecture of the array is shown in Fig. 8.

*3.4.1 Accuracy Comparison.* As mentioned in the earlier sections, the proposed methodology results in a system which can offer accuracy equivalent to the reference system. To validate the results, we implemented an equivalent model of the systolic arrays in MATLAB and simulated the LeNet network for image classification task on the cifar-10 dataset. The classification accuracy came out to be 74.43% for the conventional, merged accurate and the proposed systolic arrays, i.e., the same as the accuracy of the accurate simulations presented in Fig. 1. However, the accuracy achieved using the approximate systolic array is 73.35%.

*3.4.2 Performance, Area, and Power Evaluation of MACs.* Table 2 shows the hardware results of proposed MAC designs shown in Fig. 6, the conventional MAC shown in Fig. 7a and the approximate MAC shown in Fig. 8. It can be seen from the table that power

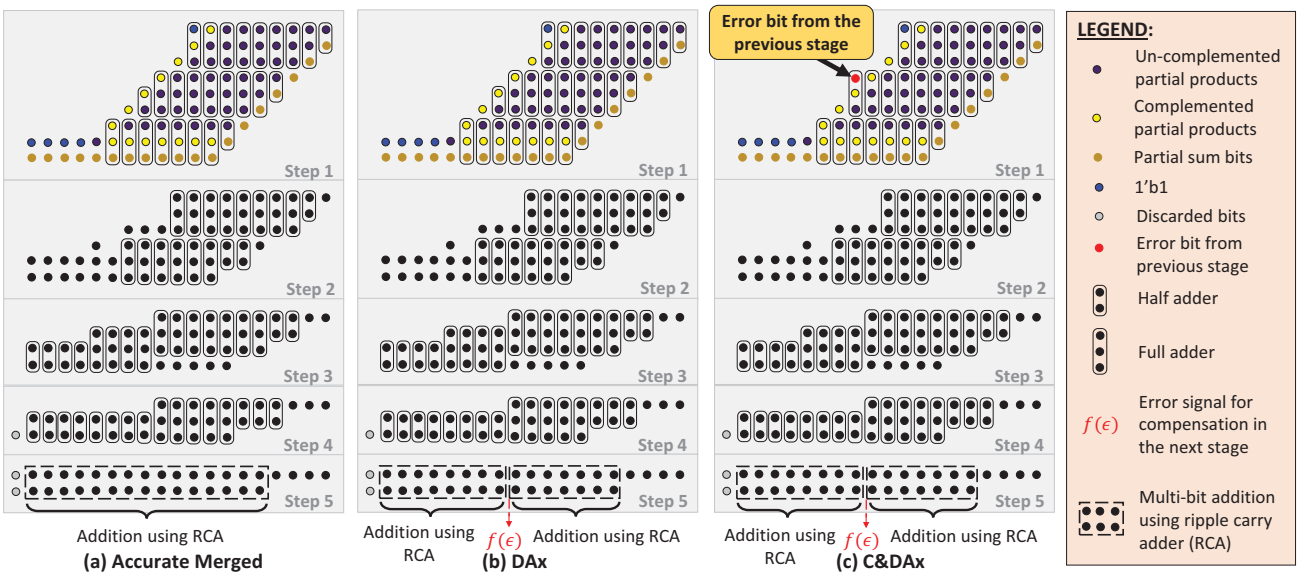



**Figure 6: Design of different types of MAC units based on Bough-Wooley multiplication algorithm and Wallace tree architecture. The multiplicand and the multiplier are assumed to be 8-bit wide and the partial sums are assumed to be 19-bit wide. (a) Accurate Merged MAC. (b) Deterministic Approximate (DAx) MAC. (c) Cure amd Deterministic Approximate (C&DAx) MAC.**

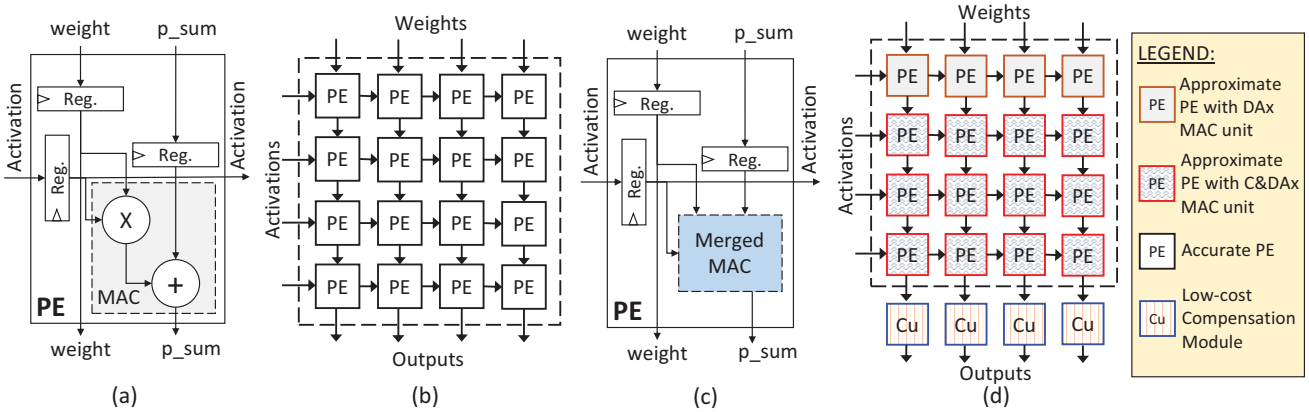



**Figure 7: (a) Processing Element (PE) design with conventional MAC. (b) Conventional systolic array design similar to the systolic array of the TPU [8]. (c) Processing Element (PE) design with merged MAC. (d) Modified systolic array design based on the proposed methodology.**

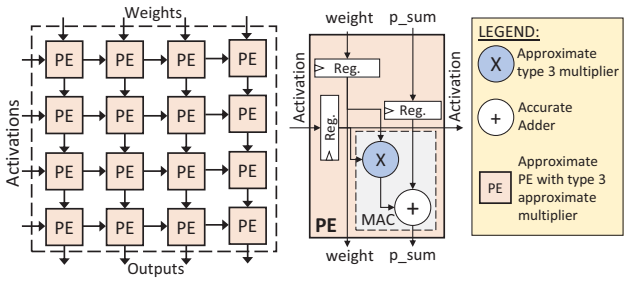



**Figure 8: An approximate systolic array design based on type 3 approximate multiplier from section 1.**

consumption of all the designs is almost the same with the approximate and the conventional accurate MACs having slightly less power consumption. The area consumption of the proposed MAC designs is also the same, however, the conventional MAC and the approximate MAC require approximately 19% and 10% more area as compared to the proposed MACs, respectively. The delay of the proposed DAx and C&DAx MAC units is the same and is slightly less than 50% of the delay offered by the conventional MAC and around 65% of the delay offered by the accurate merged MAC. The approximate MAC offers delay which is higher than that of the accurate merged MAC due to the fact that its design is based on the conventional MAC. Note that all the hardware results are generated

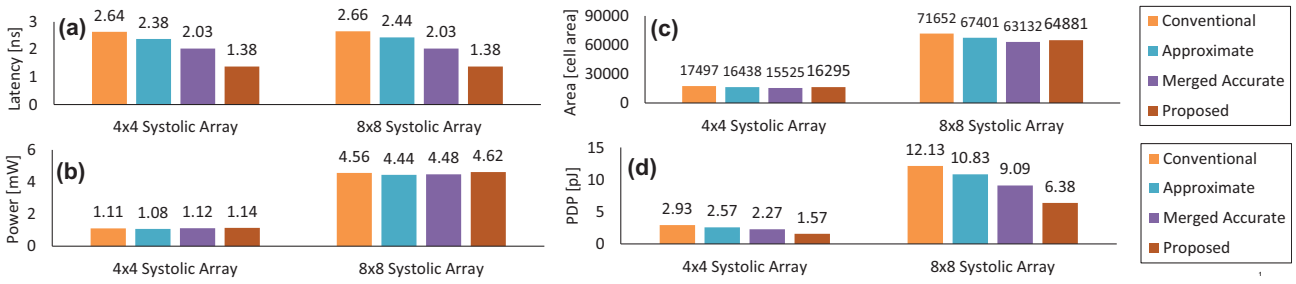



**Figure 9: Comparison of the hardware characteristics of four different designs of the systolic arrays for 4x4 and 8x8 sizes.**

**Table 2: Hardware characteristics of different types of MAC units.**

| | Latency [$ps$] | Area [Cell Area] | Power [$\mu W$] |
|---|---|---|---|
| Accurate MAC (Merged) | 1871.1 | 746 | 66.56 |
| DAx MAC | 1214.2 | 744 | 66.3 |
| C&DAx MAC | 1214.2 | 746 | 68.13 |
| Accurate MAC (Conventional) | 2470.9 | 889 | 62.73 |
| Approx MAC with Approx_Mul_3 | 2274.2 | 822 | 61.14 |

for 65 nm technology node using Cadence Genus (Encounter) tool with TSMC 65 nm library.

*3.4.3 Performance, Area, and Power Evaluation of Systolic Arrays.* Fig. 9 shows the overall hardware characteristics of four different systolic array designs (i.e., *Conventional*, *Approximate*, *Merged Accurate* and *Proposed*) for two different systolic array sizes (i.e., 4x4 and 8x8). As can be seen in fig. 9a, the *Proposed* design offers less critical path delay compred to all other designs which allows it to operate at-least at 1.91x the frequency of the *Conventional*, 1.72x the frequency of the *Approximate*, and 1.47x the frequency of the *Merged Accurate* design. The Area (in Cell Area unit) and Power are shown in Fig. 9c and b, respectively. It can be observed that the overall power and the area of all the designs are approximately similar with *Approximate* offering a bit better power and *Accurate Merged* offering a bit better area characteristics. However, if we analyze the PDP (Power Delay Product) of the designs, shown in fig. 9d, it can be observed that the *Proposed* design offers approximately 46% reduction in PDP as compared to the *Conventional*, 38% reduction in PDP as compared to the *Approximate*, and 30% reduction in PDP as compared to the *Merged Accurate* design.

## 4 CONCLUSION

In this paper, we proposed a novel methodology for designing high-performance accurate systems using approximate components with curative properties. Based on the methodology, we presented a case study on building a high-performance systolic array for deep neural network acceleration. The results showed that the systolic array design based on the proposed methodology provides better performance and PDP characteristics when compared to the conventional state-of-the-art systolic array and the array built using conventional approximate modules.

## REFERENCES


[1] C. R. Baugh and B. A. Wooley. 1973. A Two's Complement Parallel Array Multiplication Algorithm. *IEEE Trans. Comput.* C-22, 12 (Dec 1973), 1045–1047. https://doi.org/10.1109/T-C.1973.223648

[2] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan 2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[3] W. El-Harouni, S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz, and M. Shafique. 2017. Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1384–1389.

[4] M. A. Hanif, A. Marchisio, T. Arif, R. Hafiz, S. Rehman, and M. Shafique. 2018. X-DNNs: Systematic Cross-Layer Approximations for Energy-Efficient Deep Neural Networks. *Journal of Low Power Electronics* 14, 4 (2018), 520–534.

[5] X. He, L. Ke, W. Lu, G. Yan, and X. Zhang. 2018. AxTrain: Hardware-Oriented Neural Network Training for Approximate Inference. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '18)*. ACM, New York, NY, USA, Article 20, 6 pages. https://doi.org/10.1145/3218603.3218643

[6] Google Inc. [n. d.]. Tensorflow Lite. https://www.tensorflow.org/mobile/tflite/. ([n. d.]). https://www.tensorflow.org/mobile/tflite/

[7] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877* (2017).

[8] N. P. Jouppi, C. Young, N. Patil, and et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[9] A. Krizhevsky and G. Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[10] P. Kulkarni, P. Gupta, and M. Ercegovac. 2011. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In *2011 24th Internatioal Conference on VLSI Design*. 346–351. https://doi.org/10.1109/VLSID.2011.51

[11] H. Kwon, A. Samajdar, and T. Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. ACM, New York, NY, USA, 461–475. https://doi.org/10.1145/3173162.3173176

[12] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. 2011. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 163–168.

[13] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li. 2017. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 553–564. https://doi.org/10.1109/HPCA.2017.29

[14] S. Mazahir, O. Hasan, and M. Shafique. 2018. Adaptive Approximate Computing in Arithmetic Datapaths. *IEEE Design Test* 35, 4 (Aug 2018), 65–74. https://doi.org/10.1109/MDAT.2017.2772874

[15] S. Mittal. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)* 48, 4 (2016), 62.

[16] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. 2014. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–6. https://doi.org/10.7873/DATE.2014.374

[17] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel. 2016. Cross-layer approximate computing: From logic to architectures. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 99.

[18] V. Sze, Y. Chen, T. Yang, and J. S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[19] A. Vedaldi and K. Lenc. 2015. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 689–692.

[20] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. 2012. SALSA: Systematic logic synthesis of approximate circuits. In *DAC Design Automation Conference 2012*. 796–801. https://doi.org/10.1145/2228360.2228504