Journal of Harbin Engineering University
ISSN: 1006-7043
Vol 44 No. 10
October 2023

ML Based Co-processor Verification in SoC Environment

[1]Pruthvi D., [2]Dr. Srividya P.
[1]MTech VLSI & ES Dept of ECE, RVCE Bengaluru, India
[2]Associate Professor Dept of ECE, RVCE Bengaluru, India

Abstract
A co-processor based on machine learning is a highly efficient parallel compute block that is especially intended for the efficient execution of complicated computations such as neural networks. This project's CNN-based ML co-processor uses it to do calculations more quickly. This co-processor takes the input of Data, weights and the biases in order to provide the output results by performing the machine learning operations such as Data Canvas, Quantization, Convolution, Padding, Stride, Pooling and Activation functions. The main reason for dedicated processors for machine learning is enhanced energy efficiency which provides faster performance and reduction in model size and complexity. This paper involves the System on Chip(SoC) verification of ML based Co-processor using languages such as C and System Verilog in the C_UVM verification environment at the SoC level. The test cases are written using C and UVM for verifying the functionality of the design. The Functional Verification of the ML based co-processor tools is done using such as Vim Editor tool is used to write and edit all the test cases using the C and System Verilog. Arm Tool chain used to convert the C assembly code to binary format. The Cadence Incisive Tool used for simulation of the test cases and the Cadence Sim Vision to analyze the waveform that are simulated for the written test cases.
Keywords: UVM, SoC, Machine Learning, Cadence, Verification, Convolution, Neural Network

I. INTRODUCTION

This section drives through the introduction to the ML based co-processor and SoC Verification. A coprocessor refers to an extra processor for a device that supports the activities of the CPU, or main processor. The coprocessor is capable of performing a variety of functions, including floating-point computations, graphics, signal processing, character processing, encryption, and I/O interaction with external devices. Co processors can improve system efficiency by offloading processor-heavy activities from the primary processor. A co processor does not constitute the system's primary processor. The operations that follow can be carried out by a coprocessor: floating point addition, subtraction, multiplication, calculating the logarithmic value of the given integer, computing its square root, etc. (for floating point values), or performing signal, string, graphical, or encryption/decryption operations, etc. Coprocessors make it possible to customize computers, so clients do not have to foot the bill when they do not need the extra performance. Its main goal was to reduce the amount of time applications that require intensive floating point computations needed to provide results. Applications' performance increased by 20 to 50% after the co-processor was added in addition to the primary processor.

II. ML BASED CO PROCESSOR

A processor designed expressly to handle the demands of neural networks is called a neural network accelerator. As the name suggests, it is incredibly effective at what it does, which involves quickly grouping and categorizing data. Engineers discovered that a finely tuned processor totally optimized just to accomplish a certain set of activities may operate exceptionally swiftly on minimal power and aid to make some algorithms run quicker. For instance, graphics and DSPs, which are designed to do signal processing rapidly and effectively.

A. Neural Network

Figure.1 shows a neural network. An algorithm designed to function like the human brain is called a neural network. A neural network's fundamental unit, the neuron contains distinct data characteristics. There are input and output neurons, known as Visible Layers, in a simple neural network, as well as a number of intermediary layers, known as Hidden Layers.

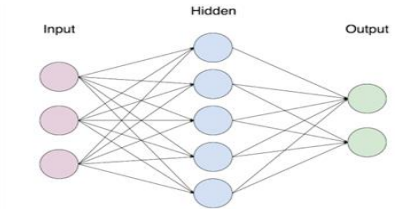

Fig. 1.A Neural network..

B. Data, Weights and Biases

Figure. 2 shows a Convolution operation performed by the co-processor. Figure. 3 shows the convolution of single element of input data. Data is the collection of input as well as values for data for which an output value must be predicted. Weights are connectivity controls between two fundamental neural network units. It is necessary to raise or reduce the relative weights for unit signals to educate neurons to advance in the network. neural network biases that prevent data from being sent to the intended output. The summation function's task is to combine each of the weights then inputs and get their sum.

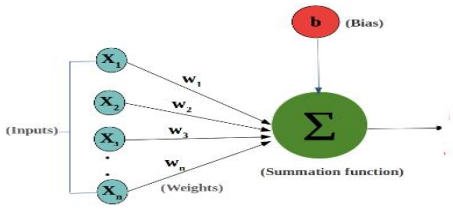

Fig. 2. Convolution operation of Co-processor

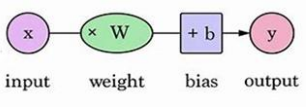

Fig. 3. Convolution of single element of data

III. FEATURES OF CO-PROCESSOR

The features of the co-processor include the Data reading by the Data Canvas, Dequantization, Pooling such as Max Pooling, Average Pooling, Padding, Stride and Activation functions like Sigmoid, Tanh, ReLU, ReLUx, and Leaky ReLUx.

A. Data Canvas

Data canvas is used for reading of the input data. Data Canvas acts as a circular buffer to store the input data. During computation of input data, the reuse of data can be done by the circular scroll of the data and sent to the pipeline for the computation. This results in the faster computation. The Figure. 4 shows the data reading through the data canvas. The input data is updated into the data canvas and stored within the data canvas. The data is being read and it is sent to the pipelines for the computations.

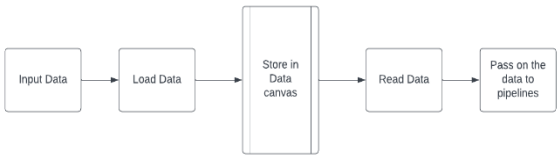

Fig. 1.Data Reading through the Data Canvas

B. Quantization

In machine learning (ML) Quantization involves the method of translating information from FP32 (floating point 32, bits) to a lesser accuracy such INT8 (Integer 8 bits) and execute all key processes like The process of convolution using INT8 and then, in the end, transform the lower precision result to higher accuracy in FP32. This looks straightforward yet since transforming floating point numbers to integers results in inaccuracy that can rise up over computations, preserving the correctness is crucial. With Quantization, precision exceeds 1% of its original accuracy.

With regards of how they perform, as the data processed is 8 bit rather of 32 bits of information technically this is 4 times quicker. In practical applications, an acceleration of no less than 2x is noticed. Further how the precision is restored in Quantization as well as the computation for INT8 or FP32 yields a comparable result is described. There are two primary procedures in Quantization that involve Quantize i.e., transform the information to a lower accuracy such INT8 & Dequantize i.e., to restore information to a greater precision like FP32.The table. 1 offers you the concept of the decrease in data size and enhancement in mathematical capability based on the data type as follows.

TABLE I. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

DATA TYPE	ACCUMULATION	DATA SIZE REDUCED	MATH POWER
INT4	INT32	8x	32x
INT4	INT32	4x	16x
FP16	FP16	2x	8x
FP32	FP32	1x	1x

C. Convolution

A mathematical procedure called convolution is used to take features out of a picture. An image kernel defines the convolution. A 3x3 kernels matrix is often encountered. In the Figure. 5 below, the green moving matrix shows the original picture, while yellow moving matrix, known as the kernel. It is used to learn the many aspects of the original image, as seen in Figure. 5

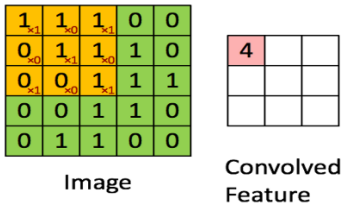

Fig. 2.Illustration of the convolution operation

Using the example of the condensed picture in Figure . 6 may assist you comprehend the idea of edge detection. The result of a 6x6 matrix and a 3x3 matrix being convolved is a 4x4 matrix. To generalize this, suppose a n × n kernel and a m x m image are convolved., the resultant output image is the size of (m - n) + 1 x (m - n) + 1.

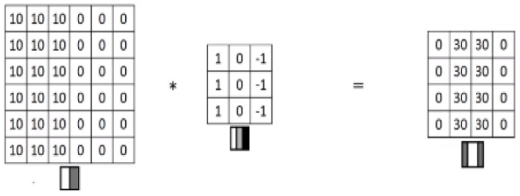

Fig. 3.A 6x6 image convolved with 3x3 kernel

D. Padding

In image classification tasks, there are numerous convolution layers, therefore performing multiple convolution operations will make the image to reduce. The second problem is that, as the kernel performs the convolution it overlaps with the image center but contacts the border of picture less frequently. So, padding is a brand-new idea that is created in order to address these two problems. The original data's size is kept when padding is used.

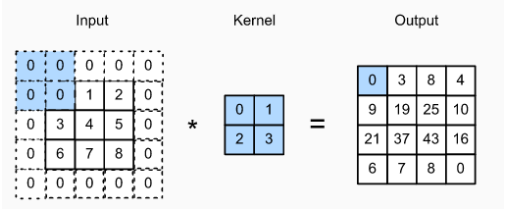

Fig. 4.Padded image convolved with 2x2 kernel

The Figure. 7 shows the Padding of the matrix data. As a result, the size of the output picture when a mxm matrix is convolved with a nxn matrix with padding p is (m + 2p - n + 1) x (m + 2p - n + 1), where the value of p = 1 in the given instance.

E. Strides

Stride is a component of convolutional neural networks, which are neural networks specifically designed for the decompression of image and video data. The amount of activity is present across the image or video is determined by the stride parameter of the neural network filter. For example, if a neural network's stride parameter is set to 1, its filter will move forward one pixel or unit at a time.

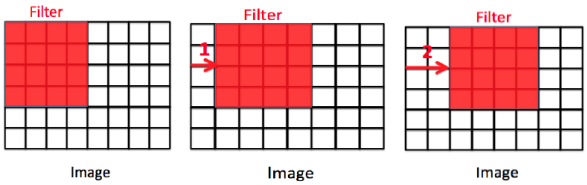

Fig. 5.Striding for stride = 1 and stride = 2

The Figure. 8 shows the striding operation performed for stride = 1 and stride = 2. The left picture displays a stride of zero, the center one, and the right one, a stride of two. The total number of pixels that are shifted across the input data matrix called stride. The output result dimensions for pad p, of the filter size f x f, the input data size n x n, and

stride s is expected to be [(n + 2p - f + 1) / s + 1] x [((n + 2p - f + 1) / s) + 1].

F. Pooling

In a neural network, pooling layers are employed, followed by a convolution layer. By reducing the number of feature maps, pooling is mainly used to speed up computation by reducing the total number of training parameters. After the pooling layer, the feature map's size is: ((l - f + 1) / s) x ((w - f + 1) / s) x c where: w = width of the feature map, l = length of the feature map, c = number of channels of the feature map, f = dimensions of the filter and s = stride.

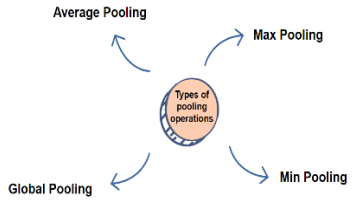

Fig. 6.Different pooling operations

The Figure. 9 shows the different pooling operations. Each of the types are explained below.

G. Max Pooling

In this kind of pooling, the largest value inside an area serves as a representation of the characteristics in that region as a whole. Since max pooling tends to select brighter pixels, it is typically employed whenever the picture has a dark backdrop. The Figure. 10 shows the 4x4 Max Pooling.

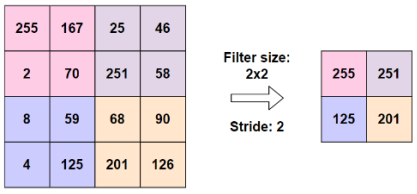

Fig. 7.Max Pooling

H. Min Pooling

In this kind of pooling, the lowest number in an area serves as a representation for an overview of the characteristics in that region. Since min pooling tends to select darker pixels when the picture has a bright backdrop, it is typically employed in that situation. The Figure. 11 shows the 4x4 Min Pooling.

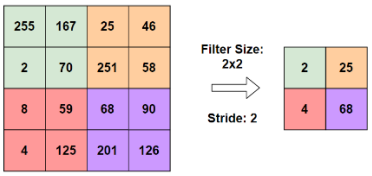

Fig. 8.Min Pooling

I. Average Pooling

In the third type of pooling, the average value for the cover serves as a representation of the characteristics in that region as a whole. When these borders are not crucial, average pooling is used to soften the sharp edges of an image. The Figure. 12 shows the 4x4 Average Pooling.

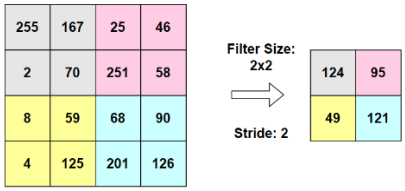

Fig. 9.Average Pooling

J. Global Pooling

Each channel of the feature map is reduced to one parameter. The kind of global pooling—which might be any of the categories previously described—determines the value. Global pooling resembles applying a filter with the identical feature map dimensions.

K. Activation functions

A neural network's activation function explains how the nodes or networks in a certain layer within a network convert the input's weighted average into an output. The co-processor supports for Sigmoid, Tanh, ReLU for short, ReLUx, and Leaky ReLUx activation functions. The activation functions utilised for all kinds of layer are described in detail below.

L. Activation for Hidden Layers

In a neural network, a hidden layer is a layer that gets data from another layer and sends output to a different layer. ReLU or Rectified Linear Activation, Logistic Activation (Sigmoid), and Hyperbolic Tangent (Tanh) are three activation functions that one would wish to take into consideration for usage in hidden layers.

M. Sigmoid Function

A sigmoid function is a function of mathematics with a recognizable "S"-shaped curve, commonly referred to as a sigmoid curve. The definition of a sigmoid function refers to a constrained, differentiable real function with exactly one bending point, a positive derivative for each point, and definitions that take into account any actual input values. The logistic function depicted in the first picture and denoted by the equation 1 is a typical instance of a sigmoid function.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} = 1 - S(-x) \quad (1)$$

Sigmoid functions typically display a returned value (y axis) between 0 and 1. The range of -1 to 1 is another frequently used range. The Figure. 13 shows the Sigmoid function.

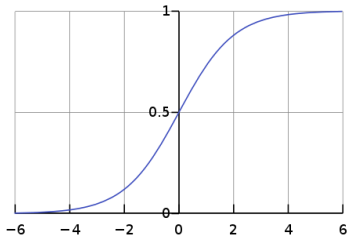

Fig. 10. Sigmoid Function

N. Tanh Function

Tanh Activating function is a neural network activation function: The following is a common illustration of a tanh function. Figure. 14 and given by the equation

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (2)$$

Tanh function gained popularity over sigmoid function formerly because it was more effective for neuronal networks with several layers. With the inclusion of ReLU activations, the vanishing gradient problem that sigmoids faced was more successfully overcome.

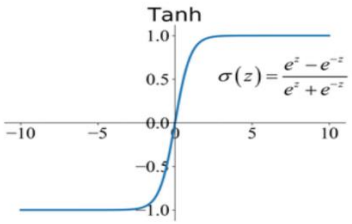

Fig. 11. Tanh Activation Function

Tanh is the updated version of logistic sigmoid. The tanh function has a ranges from (-1 to 1). Tanh also has an s-shaped sigmoidal form. Figure. 15 shows the comparision between both the plots of tanh and the sigmoid functions.

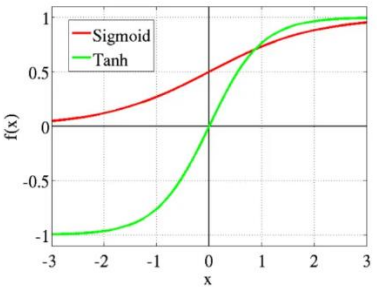

Fig. 12. Tanh v/s sigmoid functions

O. ReLU Activation Function

The ReLU is now the activation function that is utilised most frequently worldwide. as almost all deep learning and convolutional neural network systems use it.

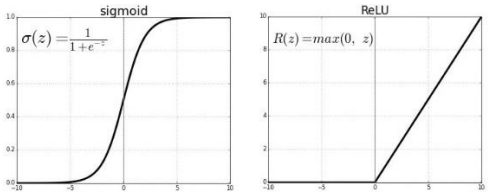

Fig. 13. Relu v/s sigmoid functions

Figure. 16 shows the comparison between both the plots of Relu and the sigmoid functions. As can be observed, the ReLU has been somewhat fixed. when z is more than or equal to zero, f(z) equals z, and Whenever z is below zero, f(z) equals zero, [From 0 to infinity]. Any negative input data that goes into the activation function of the ReLU is updated to zero in the representation of the graph.

P. Leaky ReLU Function

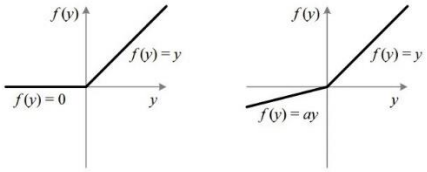

Fig. 14. Relu v/s Leaky ReLU Function

Figure. 17 shows the comparison between both the plots of Relu and the sigmoid functions. It is an effort to address the fading ReLU issue. The leak extends the range of the ReLU function. Usually, A has an average value of about 0.01. When an is not 0.01, it is commonly referred to as Randomized ReLU. The range of the Leaky ReLU then changes to (-infinity to infinity).

IV. VERIFICATION METHODOLOGY

Before tape-out, a design is evaluated against a specified design specification as part of the SOC Verification process. The ML Based Co-processor's SOC levels verification flow is shown in Figure. 18. Here are the steps of a typical SoC level Verification pipeline to make sure a good tape out of SOCs.

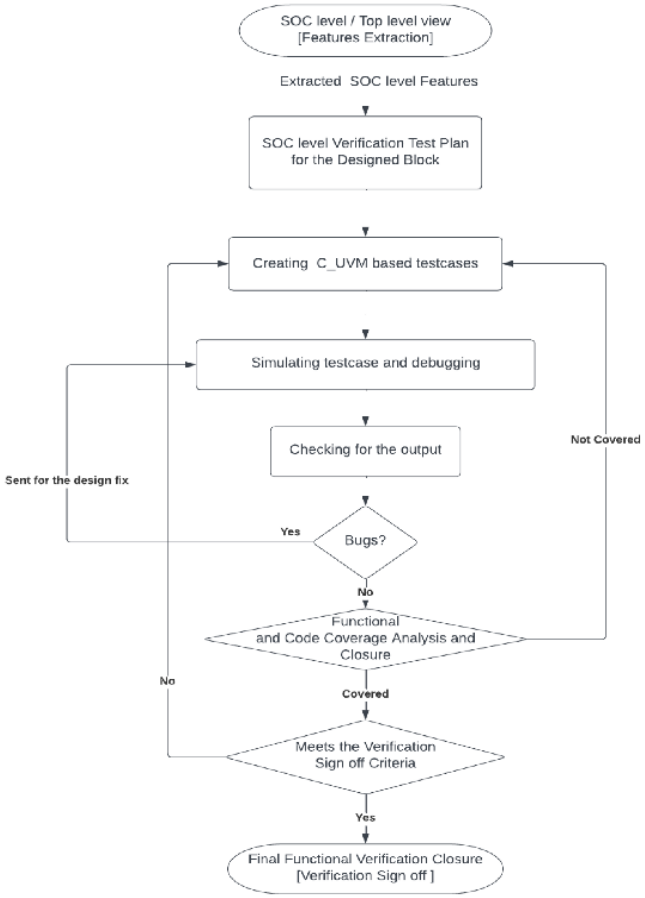

Fig. 15. SoC Verification Methodology

A. Features Extraction

One must assess the concept at the highest level feasible and determine actual SoC-level features at the specification research stage of SoC verification. A thorough understanding the SoC's function and architecture is essential at this stage since failing to comprehend the specification might result in errors and waste your time on issues that aren't RTL-related.

B. SOC Verification Test Plan

When developing the SoC levels verification strategy, it is important to explicitly define/identify the capabilities that need to be validated both on the SoC layer and at the sub-block, sub-IP, and sub-cluster level in order to draw a distinction between SOC and IP.

C. SoC Verification Environment, Verification and Debug

The finest verification approach must be used to perform SoC verification in order to thoroughly verify it. Along with this, if it is accessible and if someone intends to reuse it, it is also checked for reusability and compliance with the former verification environment.

D. Integration of Sub-Blocks

The subblocks, sub-IPs, and sub-clusters must first be integrated or stitched into the SoC levels verification framework before the real SoC verification can begin. Additionally, connection checkers that confirm appropriate or improper sub-block integration into SoC must be developed.

E. Functional Coverage and Code Coverage Closure

One of the key steps to a successful SoC tape out is the function and code coverage closure. It will be possible to close it to 100% with proper functional/code coverage analysis and evaluation.

F. Verification Sign-Off/ Final Verification Closure

All of the functional verification requirements outlined on the functional verification approval lists must be satisfied in order to achieve the ultimate functional verification closure. Examples include complete functional or code coverage, a certain amount of clean regressions over time, the closure or correction of all flaws, etc.

G. SoC Verification Debug

Debugging is time-consuming, particularly when working with the complex SoC verification. A solid understanding of the complex architecture or the structural elements that must be checked at the SoC level will greatly reduce the amount of unnecessary debugging work. When reporting a bug or any RTL issue, it's important to include all the little details that will help the designer quickly resolve the problem, such as the steps to replicate the problem and the waveform being affected, if applicable.

V. RESULTS AND DISCUSSIONS

Performance of an IP is calculated for the latency at which the IP can read or write the data onto the bus. Performance is calculated for the IP with and without arbitration.

TABLE II. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

No. of transactions for Dbus to SRAM1	No. of transactions for Math co-processor to SRAM1	Latency (in clock cycle)
18	6	0
12	8	1
1	12	2
1	10	3
Average Latency = 0 clock cycle	Average Latency= 2 clock cycles	

TABLE III. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

No. of transactions for Dbus to SRAM2	No. of transactions for Math co-processor to SRAM2	Latency (in clock cycle)
7	2	0
784	58	1
258	110	2
2	684	3

Average Latency = 1 clock cycle	Average Latency= 3 clock cycles	

TABLE IV. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

No. of transactions for Rbus to Flash Memory	No. of transactions for Math co-processor to Flash Memory	Latency (in clock cycle)
945	15	0
973	84	1
10	923	2
2	710	3
2	116	4
2	2	5
3	3	6
Average Latency = 1 clock cycle	Average Latency= 2 clock cycles	

Performance without arbitration includes the evaluation of performance at which the IP alone accessing the bus for the read and write transactions. Performance with arbitration includes the evaluation of performance at which the IP along with the multiple IPs or multiple masters in the SoC are accessing the bus simultaneously where sometimes priorities can be given to other masters for the read and write transactions. Figure. 19 shows the integration of ML Coprocessor to SoC.

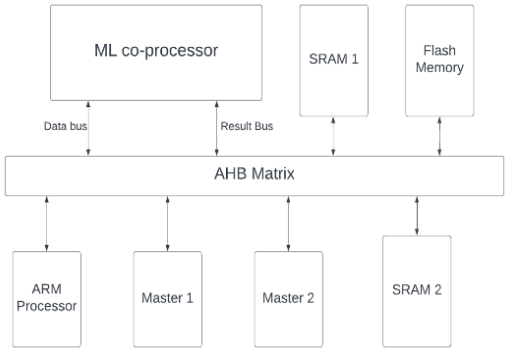

Fig. 16. Integration of ML Coprocessor to SoC

Performance is calculated for the Data(D) bus and Result(R) bus. Data(D) bus can access only SRAM1 and SRAM2 since input data to be computed is stored in SRAM1 and SRAM2. Result(R) bus can access only Flash memory since the computation result is stored in flash memory so that the input data and results should not collide. For both bus the performance is evaluated and compared with a simple Math co-processor. Performance of the IP without arbitration – No other masters in SoC are active. Performance of the IP with arbitration – Other masters in SoC are active.

TABLE V. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

No. of transactions for Dbus to SRAM1	No. of transactions for Math co-processor to SRAM1	Latency (in clock cycle)
1	0	0
1	1	1
35	2	2
20	5	3
5	25	4
Average Latency = 2 clock cycles	Average Latency= 4 clock cycles	

TABLE VI. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

No. of transactions for Dbus to SRAM2	No. of transactions for Math co-processor to SRAM2	Latency (in clock cycle)
1	0	0
2	1	1
16	10	2
789	110	3
Average Latency = 3 clock cycle	Average Latency= 4 clock cycles	

TABLE VII. COMPARISION OF DATA SIZE AND MATH POWER WITH CHANGE IN DATA TYPE

No. of transactions for Rbus to Flash Memory	No. of transactions for Math co-processor to Flash Memory	Latency (in clock cycle)
68	10	0
944	216	1
968	680	2
22	710	3
6	550	4
5	86	5
3	3	6
Average Latency = 2 clock cycle	Average Latency = 3 clock cycles	

VI. CONCLUSION

This work is mainly focused on the Functional and Integration verification of ML based co-processor in SoC Environment. The objectives of the project revolved around evaluating the SOC level performance criteria of ML Co-processsor with and without arbitration. The integration verification of ML co-processor IP at SOC level was performed. Functional verification of co-processor for the ML based targeted application scenarios like unsigned dot product, signed convolution, matrix multiplication, Dequantization, Data Canvas, Activation functions like, ReLU, Max Pooling, were verified. The gate level simulation was performed to check for the proper functionality for HtoL corner

without any timing violation in the design. The work was carried out in the C_UVM SoC environment. The C code will initiate particular bus transactions which are being evaluated while the UVM tests are streaming background activity on the bus. All test cases are written and edited using the GVIM Editor tool using System Verilog and C. The C assembly language code has been converted to binary representation using the Arm tool chain. The written test cases were simulated using the Cadence Incisive Tool, and the Cadence SimVision was utilised to analyse the waveforms in each case.

From the RTL simulation 100% of the tests are passing for all the 50 verification scenarios that were considered under the testplan for the integration verification of ML based co-processor. From the gate level simulation of the co-processor it was found that there were 0 setup and 0 hold violations and the design is ready for signoff. With comparision to simple math co-processor, the average latency of the ML based co-processor for dbus is 2 clock cycles less and 1 clock cycle less for rbus without arbitration. Also with arbitration the average latency of the ML based co-processor for dbus is 2 clock cycles less and 1 clock cycle less for rbus. In conclusion to the performance, the choice between a simple math co-processor and an ML-based co-processor, ML-based co-processor offers greater flexibility and adaptability, making it a compelling choice for a wide array of data-driven tasks.

Declaration

Decalarion	Suggestions
Funding/ Grants/ Financial Support	No, I did not receive.
Conflicts of Interest/ Competing Interests	No conflicts of interest to the best of our knowledge.
Ethical Approval and Consent to Participate	No, the article does not require ethical approval and consent to participate with evidence.
Availability of Data and Material/ Data Access Statement	Not relevant.
Authors Contributions	All authors have equal participation in this article.

REFERENCES

[1] M. Khan, V. Kumar Kodavalla, and A. Uddin Sameer, “Verification methodology of vison based hardware accelerators in system-on-chip,” in 2020 International Conference on Industry 4.0 Technology (I4Tech), 2020, pp. 39–45.

[2] Y. Wang, H. Li, and X. Li, “A case of on-chip memory subsystem design for low-power cnn accelerators,” IEEE Transactions on Computer-

Aided Design of Integrated Circuits and Systems, vol. 37, no. 10, pp. 1971–1984, 2018.

[3] P. Subramanyan, B.-Y. Huang, Y. Vizel, A. Gupta, and S. Malik, “Template-based parameterized synthesis of uniform instruction-level abstractions for soc verification,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 8, pp. 1692–1705, 2018.

[4] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, “Formal verification of arithmetic circuits by function extraction,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 12, pp. 2131–2142, 2016.

[5] J. Kumar, Y. Miyasaka, A. Srivastava, and M. Fujita, “Formal verification of integer multiplier circuits using binary decision diagrams,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 4, pp. 1365–1378, 2023.

[6] S. El-Ashry, M. Khamis, H. Ibrahim, A. Shalaby, M. Abdelsalam, and M. W. El-Kharashi, “On error injection for noc platforms: A uvm-based generic verification environment,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 5, pp. 1137–1150, 2020.

[7] M.-C. Chen, A. Sengupta, and K. Roy, “Magnetic skyrmion as a spintronic deep learning spiking neuron processor,” IEEE Transactions on Magnetics, vol. 54, no. 8, pp. 1–7, 2018.

[8] H. Xu, Z. Li, Z. Li, et al., “Reducing sram reading power with column data segment and weights correlation enhancement for cnn processing,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 11, pp. 2237–2250, 2021.

[9] C. Lin and A. Kumar, “A cnn-based framework for comparison of contactless to contact-based fingerprints,” IEEE Transactions on Information Forensics and Security, vol. 14, no. 3, pp. 662–676, 2019.

[10] S. Zheng, X. Zhang, D. Ou, et al., “Efficient scheduling of irregular network structures on cnn accelerators,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3408–3419, 2020.

[11] A. Li, H. Mo, W. Zhu, et al., “Bitcluster: Fine-grained weight quantization for load-balanced bit-serial neural network accelerators,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 11, pp. 4747–4757,2022.

[12] D. Kang and S. Ha, “Datapath extension of npus to support nonconvolutional layers efficiently,” IEEE Design Test, vol. 39, no. 5, pp. 54–61, 2022.

[13] P. Dasgupta, M. K. Srivas, and R. Mukherjee, “Formal hardware/software co-verification of embedded power controllers,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 12, pp. 2025–2029, 2014.

[14] R. Claris´ o, C. A. Gonz´ alez, and J. Cabot, “Smart bound selection for the verification of uml/ocl class diagrams,” IEEE Transactions on Software Engineering, vol. 45,no. 4, pp. 412–426, 2019.

[15] X. Wang, C. Wang, J. Cao, L. Gong, and X. Zhou, “Winonn: Optimizing fpga-based convolutional neural network accelerators using sparse winograd algorithm,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 4290–4302, 2020.

[16] G. Zhou, J. Zhou, and H. Lin, “Research on nvidia deep learning accelerator,” in 2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID), 2018, pp. 192–195.

[17] D. Kang, J. Kang, H. Kwon, H. Park, and S. Ha, “A novel convolutional neural network accelerator that enables fully-pipelined execution of layers,” in 2019 IEEE 37th International Conference on Computer Design (ICCD), 2019, pp. 698–701.

[18] Z. Li, Y. Chen, and D. Zhao, "A method of verification in chisel based deep learning accelerator design," in 2020 IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA), vol. 1, 2020, pp. 789–792.

[19] H. Vladimir, L. Eugenia, and P. Irina, "Soc verification infrastructure," in 2010 IEEE Region 8 International Conference on Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010, pp. 80–85.

[20] W. Yang, M.-K. Chung, and C.-M. Kyung, "Current status and challenges of soc verification for embedded systems market," in IEEE International [Systems-on-Chip] SOC Conference, 2003. Proceedings., 2003, pp. 213–216

Authors Profile

Pruthvi D, is a student pursuing M Tech 2nd year in VLSI and Embedded Systems at RV College of Engineering. Her research interests primarily focus on digital design and Verification. She is eager to contribute to the advancement of VLSI technology and aims to pursue a career in semiconductor industry research and development.

Dr.Srividya P currently working as Associate Professor in the department of electronics and communication engineering, RVCE, Bangalore, India. Has 19 years of teaching experience. Obtained Bachelor of Engineering Degree in Telecommunication Engineering in 2003, M.Tech degree in VLSI Design and embedded system in 2007 and Ph.D in 2016 all from Visveswaraiah Technological University, Belgaum, Karnataka, India. Has 2 patents published and nearly 22 papers in international conferences, 15 papers in International journals and 6 book chapters in different books published by CRC press. Guided around 30 UG projects and 20 PG projects. Is editor for 2 books published by renowned publishers. Paper titled "Challenges in p-type oxide based thin film transistors" published along with a research scholar received best paper award in International technical conference on control in Electric vehicles and smart grid with renewable energy synergies, organized by school of Electrical Engineering, VIT in October 2020.

Area of interest includes Analog VLSI design, Digital VLSI design and embedded systems.