2025 IEEE 6th India Council International Subsections Conference (INDISCON) | 979-8-3315-1504-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INDISCON66021.2025.11251525

Power and Area Efficient Processing Element (PE) of CNN Accelerator for Object Detection

Abinash Kumar Pala
Dept. of ECE
NIST University
Berhampur, India
abinashpala@gmail.com

Swastik Behera
Dept. of ETC
Parala Maharaja Engineering College
Berhampur, India
swastikbehera608@gmail.com

Raghunandan Swain
Dept. of ETC
Parala Maharaja Engineering College
Berhampur, India
raghu.etc@pmec.ac.in

Dinesh Kumar Dash
Dept. of ETC
Parala Maharaja Engineering College
Berhampur, India
dinesh.etc@pmec.ac.in

Sandipan Mallik
Dept. of ECE
NIST University
Berhampur, India
sandi.iitkgp@gmail.com

Abstract **- The proposed Research presents the architecture of an ASIC-based Processing Element (PE) designed to support and increasingly advance the computational needs of Convolutional Neural Networks (CNNs) as they extend to real-time applications of image processing, object recognition, and autonomous systems. The PE is specifically developed for convolution operations and has a pipelined datapath composed of a multiplier; three-layer convolution engine; first-in-first-out (FIFO) buffer; and an output register. The control elements of the component are computed using D-Flip Flops and include overflow detection, set-bounds logic, a counter and a MUX to enable step-level control and dynamic limits on the computation limits. The novel features ensure synchronized access and perform data transformations through convolution, pooling, and activation functions efficiently. It is a modular and scalable implementation, and can be integrated into larger CNN accelerators capable of performing multiple convolution operations in parallel across multiple PE's. Synthesis and simulation results substantiate that the architecture works effectively and shows significant gains in power, area, and performance when compared to conventional GPU-based systems.**

Keywords - ASIC, CNN, Processing Elements, CNN Accelerator, Convolution, Hardware Implementation, FIFO, Feature Accumulation, Pipelined Architecture.

I. INTRODUCTION

Artificial intelligence has established itself as a disruptive technology that spans virtually every area of application where machines can learn from data to enable them to make intelligent decisions. One of the prominent areas of artificial intelligence is known as deep learning [1]. Deep learning has made major strides in various fields due to an impressive formal performance in challenging tasks such as image recognition [2, 3], natural language processing [4, 5], and autonomous systems. At the heart of many deep learning models, like CNNs [6, 7], is the fact that they can analyze visual data. Often, CNNs perform multiply-accumulate (MAC) operations on very large data sets using convolutional layers. Because of this dependence on MAC operations, it is essential to ensure that hardware is efficient, especially given that a CNN is considered an expensive operation. With this efficiency in mind, Processing Elements (PEs) [8, 9] are field programmable hardware that can be utilized to share and/or perform the MAC operations amongst the processors. PEs represent the fundamental building blocks of CNN accelerators which deliver improvements in performance, speed, and efficiency on each individual MAC operation.

This paper discusses the architecture and design issues of PEs for CNN accelerators that are essential for high throughput and low-power deep learning inference. The following sections provide a structured overview of our work. Section II provides a detailed methodology for implementing the architecture that we have proposed. Section III describes our new Processing Element (PE), along

with an in-depth step-by-step data flow analysis. Section IV presents and analyzes the results from our proposed PE. Finally, Section V concludes the paper with a summary of findings and suggestions for future research directions.

II. METHODOLOGY

In figure 1, with an array representing pixel values, it enters the first Convolutional layer (Ci). In this layer, one can consider that there are a collection of trained filters - also called kernels or weights, which slide over the input image, one by one, in a convolutional operation where element-wise multiplication is performed and the output is summed together. In doing so, the network is able to find different features like edges, textures, etc [10], [11].

We then apply the ReLU (Rectified Linear Unit) [12], [13] activation function. The consequence of using an activation function is that it creates non-linearity in the model, allowing the model to better recognize complex patterns. It works by essentially setting all the negative values from the convolution to zero. Next comes the Pooling Layer (Pi). The pooling layer is known for reducing the spatial size of the feature maps from the convolutional layer. Standard pooling methods include max pooling - which selects the value from a window of size 'p×p' containing the largest value [11].

These Convolutional and Pooling (CP) layers are generally repeated several times, with each convolutional layer using the features learned by the previous layers to detect more complex features [14]. The Fig.1 demonstrates this repetition by way of, "iterate the CP layers according to the CNN models." This whole CP block is repeated multiple times (as indicated by CP1, CP2, ..., CPN−1), where each iteration begins to learn more complex features. The zoomed-in area at the bottom illustrates a simplified version of how the convolution operation is to be processed on hardware, with multipliers, adders and within an accumulator.

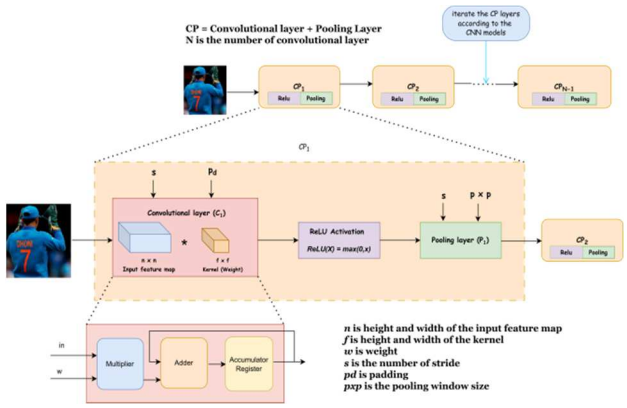

Figure 1. Training Model of Processing Element Operation (Convolution Operation)

III. ARCHITECTURAL FLOW OF PROCESSING ELEMENT

A Processing Element (PE) refers to the fundamental computation unit in a hardware accelerator. The PE performs the basic operations of a neural network, for example, multiply-accumulate (MAC) [15]. Processing Elements are designed for parallel processing, meaning many computations can be performed at the same time, which allows for a more efficient processing of huge sets of data.

There is Processing Element (PE) architecture because in general, convolutional operations need to be performed efficiently. As given in figure 2 the PE receives input data, weights, and bias and is filtered through a multiplier and a first convolutional layer. The PE also has a First-In-First-Out (FIFO) buffer to handle the intermediate results that are sent into the subsequent layers, such as another convolutional layer. The output from the convolutional layer is filtered into an output feature accumulator [16].

An overflow check mechanism is added to preserve data integrity, and its condition may be recorded in a D-Flip Flop. A bound module is defined by a bound level input, and this will give the output limited permit. This model has two control signals: step and en (enable) are registered in D-FFs and are used to drive a counter module. The output of the counter and a fixed value of (13'b0) are put into a multiplexer (MUX) which is also likely

switched on by the bound level. The output enable (out_en) is also registered. The design can support faster pixel-to-pixel execution on a convolutional neural network [17].

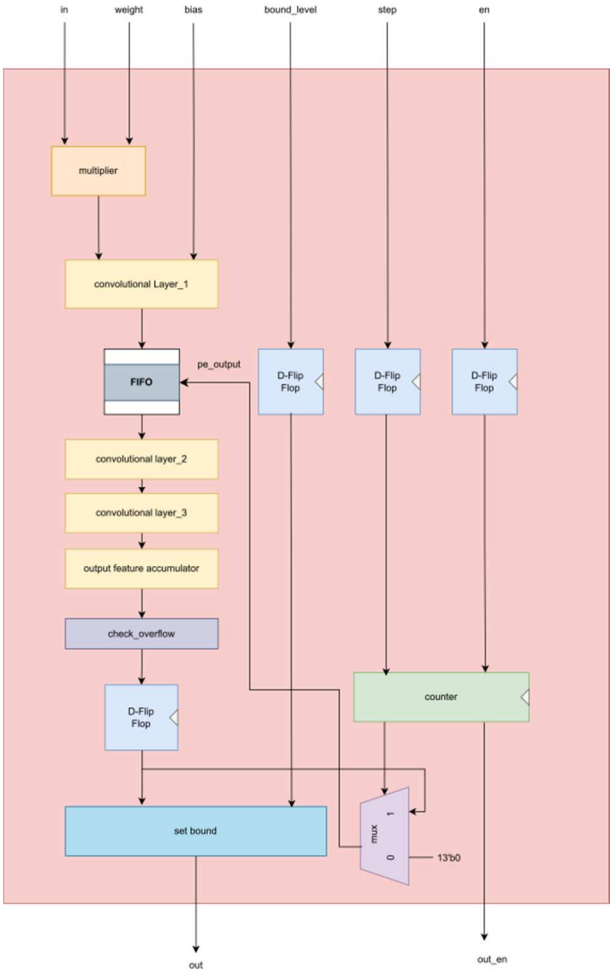

Figure 2. Proposed Processing Element (PE) Architecture

The workflow of the Processing Element (PE) enhances parallelism, pipelining, and data reuse, making the PE architecture highly efficient and scalable for deep CNN deployments in hardware accelerators as given in figure 3.

Step 1: Fetching Input Data

The processing begins by fetching the input data, denoted as "in," which serves as the input feature map for the convolutional operations.

Step 2: Fetching Weights (Kernel) Data

Simultaneously, the corresponding weights (kernel) data, labeled as "weight," are fetched. These weights will be used in the convolution operations.

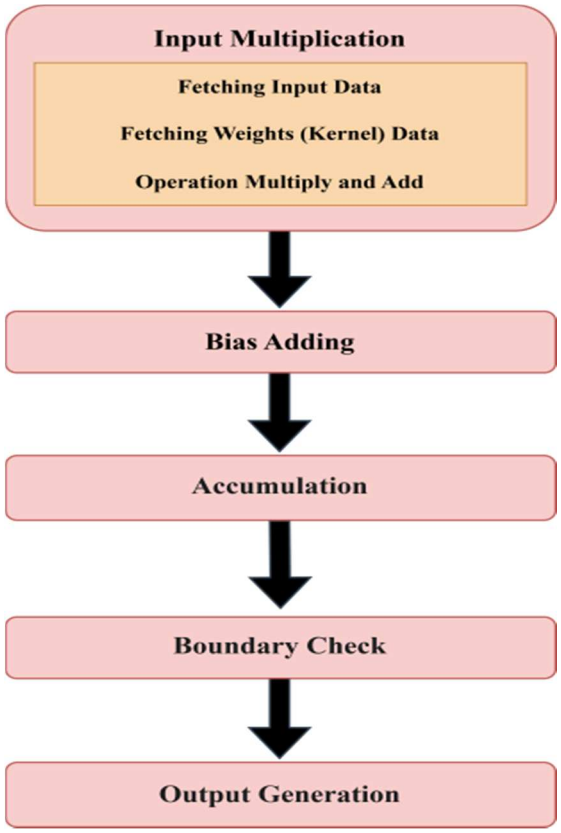

Figure 3. Workflow of the Processing Element

Step 3: Operation Multiply and Add

Each of the input data is first sent through a multiplier block where they are shifted to the proper order relative to the weight data which has been preloaded sequentially from memory into the weight buffer. The order of inputs and weights is provided to the "multiplier" block. The output of this multiplier is combined with accumulated values and is processed through a sequence of multiply - accumulate (MAC) operations. At the completion of six clock cycles, the multiplication output is sent to a parallel-out adder tree which retrieves the sums from each layer [18]. Dummy padding is used, if

necessary, for output count consistency. Desired outputs are selected based on clock cycle timing.

Mathematically, this operation is expressed as:

$$O[z][y][x]=\sum_{k=0}^{C_i-1}\sum_{j=0}^{K_y-1}\sum_{i=0}^{K_x-1} I[K][U_y+j][U_x+i]\times W[z][k][j][i] \quad (1)$$

$$O \le z \le C_0; O \le x \le W_0; O \le y \le H_0$$

Where:

U is Stride; W is Kernel (Weights);
*O[z][y][x]*represent the output feature map;
I[k] represent the input feature map;
[x] is horizontal position of the element;
[y] is vertical position of the element;
[z] is index of the output channel;
[k] is index of the input channel;
*[i]*is horizontal coordinate within the kernel;
*[j]*is vertical coordinate within the kernel;
U_y+*j*is vertical coordinate of the pixel in the input patch;
U_x+*i*is horizontal coordinate of the pixel in the input patch.

Step 4: Bias Adding

A bias term, labeled as "bias," is added to the accumulated result. This addition introduces an offset to the feature map[18].

Mathematically, this operation is expressed as:

$$O[z][y][x]=B[z]+\sum_{k=0}^{C_i-1}\sum_{j=0}^{K_y-1}\sum_{i=0}^{K_x-1} I[K][U_y+j][U_x+i]\times W[z][k][j][i] \quad (2)$$

Where:

B[z] is bias.

Step 5: Accumulation

The outputs from the multiplication and bias stages are passed through a set of convolutional layers (e.g., convolutional_layer_1, _2, _3). Intermediate outputs are temporarily stored in a FIFO buffer, ensuring continuous data flow and avoiding pipeline stalls. The final feature values from these layers are accumulated using an output feature accumulator.

Step 6: Boundary Check

After accumulation, the "check overflow" block detects any data overflow, followed by a D-Flip Flop that stabilizes the signal before a "set bound" operation adjusts the output values. Control signals like "bound level," "step," and "en," along with D-Flip Flops and a counter, manage the timing of the pipeline.

Step 7: Output Generation

A multiplexer selects between the bounded and original output. D-Flip Flops pipeline the data and control signals. A counter and multiplexer generate addresses for accessing input data or weights. The out en signal is asserted to indicate when the PE output is valid and ready to be read by the next processing block or memory stage.

IV. RESULT

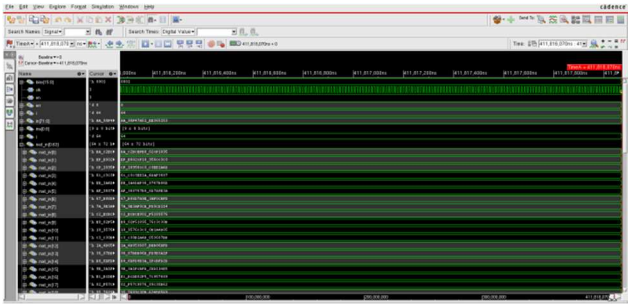

Figure 4. Simulation Waveform of Processing Element (PE)

The figure 4 illustrates the detailed simulation waveform of the Processing Element (PE), showcasing the verification of its functional correctness and timing behavior. The input and output waveforms of each individual modulesuch as "Multiplier," "Convolutional Layer1," "Convolutional Layer2," "Convolutional Layer3," are meticulously examined to ensure proper synchronization and seamless data flow throughout the system. Furthermore, timing analysis and resource utilization metrics are evaluated to confirm that the PE meets the desired performance benchmarks in terms of speed and efficiency.

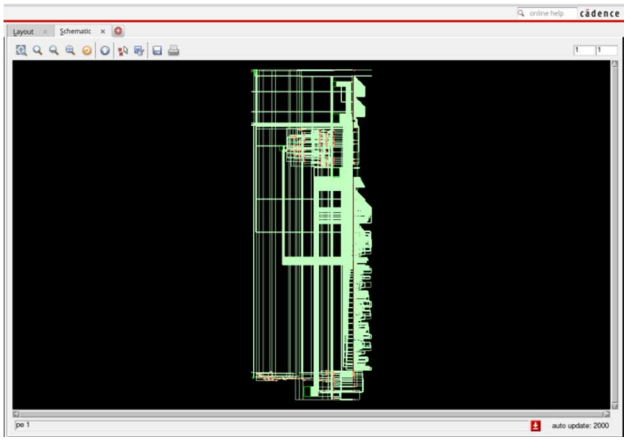

Figure 5. Schematic Diagram of Processing Element (PE)

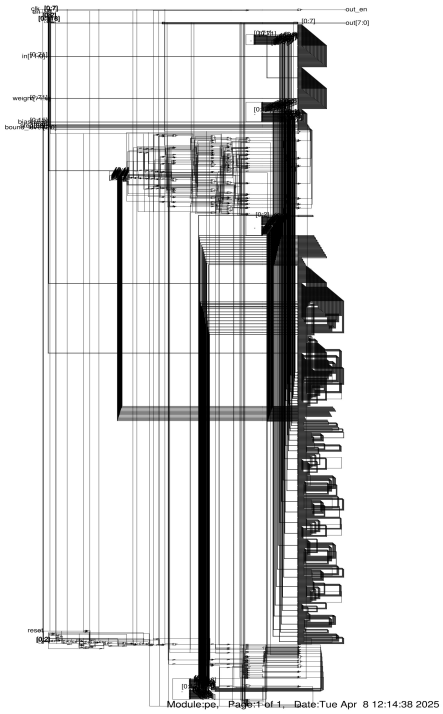

Figure 6. Technology View of Processing Element (PE)

This figure 5 highlights architectural feature, supports analysis of performance metrics like power and timing, and demonstrates the synthesized output derived from the RTL description. The figure 6 presents a detailed schematic diagram of a processing element, likely synthesized using the Cadence Genus Synthesis Solution. This visualization provides a structural representation of the processing element's components and their interconnections at a specific level of abstraction, potentially after Register-Transfer Level (RTL) synthesis. The hierarchical design structure organizes components into sub-modules such as "Multiplier," "Convolutional Layer1," "Convolutional Layer2," "Convolutional Layer3," and "Output Feature Accumulator,". The central layout "PE" view illustrates the intricate interconnectivity between various logic elements, reflecting the data and control flow.

The presented Processing Element achieves an area utilization of 3.5×10^3nm² and operates with a power consumption of 2.19×10^5nW, as summarized in theTable I. These optimized performance metrics demonstrate its potential as a building block for advanced digital processing units.

TABLE I. AREA AND POWER COMPARISON

Parameter	Power (nW)	Area (nm²)	Ref
CPA-factored	6.27×10^9	2.1×10^6	[17]
CNN accelerator	2.2×10^{11}	31.02×10^6	[19]
Proposed Processing Element	2.19×10^5	3.5×10^3	This work

V. CONCLUSION AND FUTURE WORK

The Processing Element that has been constructed has useful characteristics of both low area (3.5×10^3 nm²) and low power consumption (2.19×10^5nW) that will be useful for integration in small area and low power VLSI systems. The results show that the Processing Element has a small size and enough capability to process useful calculations. This is critically important for integration into low power and low area SoCs. The Processing Element could be used as a starting point for optimizing use as a CNN accelerator, a low power parallel processing platform used within the AI and edge computing space. A low power, low area Design will allow a designer to explore large, scalable and efficient CNN architectures in the development of real time, high performance inference engines in a multitude of applications such as image recognition and autonomous systems.

ACKNOWLEDGMENT

The authors express their gratitude to the MeitY C2S project (having Grant no. EE-9/2/2021-R&D-E) at Dept. of E&TC, Parala Maharaja Engineering College, Berhampur (761003), Odisha for its generous support.

REFERENCE

[1] Y. Hu, Y. Liu, and Z. Liu, "A survey on convolutional neural network accelerators: GPU, FPGA and ASIC," in *2022 14th International Conference on Computer Research and Development (ICCRD)*, 2022, pp. 100–107.

[2] C. T. Sari and C. Gunduz-Demir, "Unsupervised feature extraction via deep learning for histopathological classification of colon tissue images," *IEEE Trans. Med. Imaging*, vol. 38, no. 5, pp. 1139–1149, 2018.

[3] M. Chun, H. Jeong, H. Lee, T. Yoo, and H. Jung, "Development of korean food image classification model using public food image dataset and deep learning methods," *IEEE Access*, vol. 10, pp. 128732–128741, 2022.

[4] V. More, A. Chaudhari, B. Puli, V. Vuppala, J. Dofe, and W. Danesh, "Natural Language Processing Meets Hardware Trojan Detection: Automating Security of FPGAs," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024, pp. 775–778.

[5] S. Raman and E. R. Shaji, "ASIC design of a matching unit for NLP," *Microprocess. Microsyst.*, vol. 19, no. 6, pp. 327–340, 1995.

[6] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D convolutional neural networks and applications: A survey," *Mech. Syst. Signal Process.*, vol. 151, p. 107398, 2021.

[7] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE Trans. neural networks Learn. Syst.*, vol. 33, no. 12, pp. 6999–7019, 2021.

[8] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, p. 101887, 2021.

[9] S. M. A. H. Jafri, A. Hemani, and D. Stathis, "Can a reconfigurable architecture beat ASIC as a CNN accelerator?," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2017, pp. 97–104.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "How to evaluate deep neural network processors: Tops/w (alone) considered harmful," *IEEE Solid-State Circuits Mag.*, vol. 12, no. 3, pp. 28–41, 2020.

[11] C. Y. Lo, C.-W. Sham, and C. Fu, "Novel CNN accelerator design with dual Benes network architecture," *IEEE Access*, vol. 11, pp. 59524–59529, 2023.

[12] P. Purwono, A. Ma'arif, W. Rahmaniar, H. I. K. Fathurrahman, A. Z. K. Frisky, and Q. M. ulHaq, "Understanding of convolutional neural network (cnn): A review," *Int. J. Robot. Control Syst.*, vol. 2, no. 4, pp. 739–748, 2022.

[13] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towar. Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[14] S. Deepika, V. Arunachalam, and A. N. J. Raj, "A Review on Hardware Accelerators for Convolutional Neural Network-based Inference engines: Strategies for Performance and Energy-efficiency Enhancement," *Microprocess. Microsyst.*, p. 105146, 2025.

[15] M. S. Kumar, D. A. Kumar, and P. Samundiswary, "Design and performance analysis of Multiply-Accumulate (MAC) unit," in *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*, 2014, pp. 1084–1089.

[16] Y. Xu, J. Luo, and W. Sun, "Flare: An FPGA-Based Full Precision Low Power CNN Accelerator with Reconfigurable Structure," *Sensors*, vol. 24, no. 7, p. 2239, 2024.

[17] K. Inayat and J. Chung, "Carry-propagation-adder-factored gemmini systolic array for machine learning acceleration," *Electronics*, vol. 10, no. 6, p. 652, 2021.

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

[19] X. Qin, X. Liu, and J. Han, "A CNN hardware accelerator designed for YOLO algorithm based on RISC-V SoC," in *2021 IEEE 14th International Conference on ASIC (ASICON)*, 2021, pp. 1–4.