RTL-Based Design and Simulation of a Hardware-Aware CNN Accelerator Using Verilog and Python

Dhivya G B[1], Jai AdityaT[2], Sasmitha S P[3], Abirami S A[4], Mr. Manokaran J[5]
[1,2,3,4]*UG Scholar, Dept. of EE(VLSI), Sri Shakthi Institute of Engineering. And Tech., Coimbatore India*
[5]*Mentor, Sri Shakthi Institute of Engineering. And Tech., Coimbatore India*

***Abstract*—Convolutional Neural Networks (CNNs) have become one of the most widely used computational models in modern image processing and computer vision due to their ability to extract meaningful spatial features from input images. CNNs are applied in a broad range of fields including image classification, medical diagnosis, surveillance, object detection, robotics, and edge AI systems. However, the convolution operation that forms the core of CNN computation involves a large number of repetitive multiply and accumulate operations over the input feature map. When these operations are executed sequentially on a conventional processor, the result is high execution latency, increased memory access overhead, and reduced energy efficiency. These limitations motivate the need for hardware-aware computation models that can map convolution into efficient parallel structures.**

This paper presents the design and simulation of a hardware-aware CNN accelerator using both Python and Verilog HDL. The work adopts a hybrid methodology in which convolution is first demonstrated in software using grayscale image inputs and multiple filter kernels, followed by a hardware-oriented implementation using Multiply-Accumulate (MAC) based design. In the software domain, convolution is performed using a traditional loop-based method and an optimized computation method in order to compare performance and validate correctness. Edge detection and sharpening filters are applied to generate feature maps and observe the effect of kernel-based image transformation. The results show that the optimized method reduces execution time while preserving output correctness, thereby illustrating the advantage of computation-efficient design.

In the hardware domain, the same convolution concept is mapped into RTL using Verilog HDL. A MAC unit is designed as the fundamental arithmetic block, and a 3×3 convolution module is constructed by combining multiple MAC operations. The hardware design is synthesized and analysed using Intel Quartus II, and the RTL viewer is used to verify the structural representation of multipliers, adders, and registers. The implementation demonstrates how convolution operations can be translated into hardware-efficient structures suitable for CNN accelerator design. The proposed work provides a practical understanding of software-to-hardware mapping and offers a foundation for future development of VLSI-based AI accelerators.

***Index Terms*—Convolutional Neural Networks, CNN Accelerator, Hardware-Aware Design, Multiply-Accumulate Unit, Verilog HDL, RTL Design, Quartus II, Image Processing, Edge AI**

I. INTRODUCTION

Convolutional Neural Networks have emerged as a dominant technique in deep learning, particularly for image processing and computer vision applications. Their strength lies in the ability to automatically learn and extract hierarchical features from raw input images without manual feature engineering. CNNs are used in many modern systems, including face recognition, industrial inspection, autonomous navigation, medical image analysis, and intelligent surveillance. At the heart of CNN computation is the convolution operation, which applies a small kernel across an image or feature map to produce transformed output values that encode local spatial relationships. Although mathematically straightforward, this operation is computationally intensive because it requires repeated multiplication and accumulation over every valid position in the input matrix.

The demand for real-time processing in embedded and edge devices has made the efficient implementation of convolution increasingly important. Traditional

software-based execution on general-purpose processors often results in high latency because the operations are carried out sequentially. This approach is acceptable for small input sizes, but it becomes inefficient when the image dimensions increase or when the system needs to process multiple frames continuously. Furthermore, CNN applications are frequently deployed on low-power devices where energy consumption is a critical design constraint. These challenges have led to the development of dedicated hardware accelerators that exploit parallelism, pipelining, and specialized arithmetic units to improve performance.

A hardware-aware CNN accelerator seeks to bridge the gap between algorithmic convolution and hardware execution. Instead of treating convolution purely as a software operation, the computation is viewed as a dataflow problem that can be implemented using modular arithmetic blocks such as MAC units. In VLSI systems, MAC units are particularly important because they can perform multiplication and accumulation efficiently, which are the two primary operations required in convolution. By arranging these units in parallel or in a structured data path, it becomes possible to achieve much higher throughput than sequential execution.

This project focuses on the design and simulation of a simplified CNN accelerator using Python for software validation and Verilog HDL for hardware realization. The software component demonstrates convolution using image matrices and filter kernels, while the hardware component translates the same mathematical principle into RTL form through a MAC unit and a 3×3 convolution block. The overall aim is to provide a clear understanding of how convolution-based computations can be optimized and mapped into VLSI-oriented accelerator structures. The work is intentionally modular and educational in nature, making it suitable for academic learning while still reflecting the architecture of practical CNN accelerators.

II. LITERATURE REVIEW

Convolutional Neural Networks have been extensively studied due to their strong performance in feature extraction and pattern recognition tasks. The convolution layer, which applies a set of filters over an input image, is the most computationally dominant part of a CNN architecture. As the size of input images and the depth of neural networks continue to increase, the computational burden of convolution becomes more significant. This has motivated researchers to investigate a variety of acceleration techniques spanning software optimization, parallel computing, and hardware implementation. In general, the literature agrees that CNN acceleration is necessary for achieving low-latency and energy-efficient execution in real-world applications.

Many existing studies show that general-purpose processors are not well suited for convolution-heavy workloads because they execute arithmetic operations sequentially and incur high memory access overhead. To overcome this limitation, specialized hardware platforms such as GPUs, FPGAs, and ASICs are commonly used. GPUs offer massive parallelism for matrix-based operations, while FPGA-based and ASIC-based solutions provide better control over power, timing, and data reuse. Among these options, FPGA and RTL-based implementations are often preferred in academic projects because they clearly demonstrate the mapping of algorithmic behaviour into hardware structures, while also allowing synthesis and RTL verification.

A major theme in CNN hardware research is the use of Multiply-Accumulate units. Since convolution is fundamentally composed of multiplication followed by summation, MAC units are ideal for implementing convolution efficiently. Researchers have shown that by organizing MAC units in a parallel or pipelined structure, it is possible to increase throughput and reduce the number of clock cycles required per output feature. Some accelerator designs also include line buffers, sliding-window logic, and local storage blocks to reduce redundant memory reads. These techniques improve data reuse and are especially important in embedded and edge devices where bandwidth is limited.

Another important concept in the literature is the translation of convolution from mathematical expression to hardware-oriented dataflow. Software simulation is often used to validate the functional behaviour of the algorithm before hardware

implementation. Once the algorithm is verified, the same logic can be rewritten in Verilog or VHDL and synthesized into a register-transfer-level model. This process helps designers' study both correctness and structural implementation. It also provides insight into how high-level deep learning operations can be represented using low-level digital design blocks such as adders, multipliers, registers, and control logic.

The literature consistently shows that efficient CNN accelerators rely on structured hardware design, efficient memory access, and parallel arithmetic units. However, many advanced implementations are too complex for introductory academic work, particularly when the goal is to demonstrate fundamental understanding rather than full industrial-scale optimization. This project therefore adopts a simplified but technically meaningful approach. It combines Python-based convolution with a Verilog-based MAC and convolution block to demonstrate the essential principles of CNN acceleration and RTL mapping in a form that is accessible, verifiable, and academically relevant.

III. PROBLEM STATEMENT

The main problem addressed in this project is the high computational cost of convolution operations in CNNs when executed sequentially on a general-purpose processor. Convolution requires sliding a kernel across the input image and performing repeated multiplication and accumulation at every valid location. This creates a large number of arithmetic operations, especially for larger images or multiple filter stages. As a result, software implementations often become slow, and the delay becomes more pronounced when real-time processing is required. In embedded systems and edge AI applications, this issue is even more critical because the available processing power and energy budget are limited.

A second problem is the lack of visibility into how convolution is implemented at the hardware level. While software models can compute correct outputs, they do not reveal how the operation can be mapped onto digital hardware blocks. In practical VLSI systems, convolution is implemented using arithmetic datapaths, MAC arrays, registers, and control logic. Without such a mapping, it is difficult to understand the architecture of modern CNN accelerators or to evaluate how they achieve faster performance than software execution alone.

This project addresses these challenges by developing a hardware-aware CNN accelerator model that combines software simulation and RTL-based design. The software component verifies the mathematical behaviour of convolution and measures performance improvement. The hardware component maps the same operation into Verilog HDL using a MAC unit and a 3×3 convolution module. The resulting system demonstrates how CNN computation can be made more efficient through hardware-oriented implementation.

IV. PROPOSED METHODOLOGY

The proposed methodology follows a hybrid design flow that combines software verification with hardware realization. The first stage begins with image preprocessing in Python. A grayscale input image is represented as a matrix of pixel values, and convolution is applied using predefined kernel matrices. The project uses both a slow loop-based implementation and an optimized computation method to compare execution time and validate output correctness. This software stage helps establish a clear understanding of how convolution transforms an image and how different filters extract different features from the same input.

The second stage moves from software to hardware. The convolution operation is translated into Verilog HDL by designing a Multiply-Accumulate unit. This MAC unit performs the essential arithmetic required in convolution, namely multiplication of input pixel values with kernel coefficients followed by accumulation of the partial products. After validating the MAC unit, a 3×3 convolution module is constructed by connecting multiple multiply and add operations in a structured RTL form. This module represents a complete hardware block for one convolution window.

The final stage is synthesis and verification using Intel Quartus II. The Verilog modules are compiled, and RTL visualization is used to inspect the generated hardware structure. The presence of multipliers, adders, and register elements confirms that the

convolution operation has been correctly mapped into a hardware-friendly architecture. This methodology is effective because it demonstrates both the algorithmic behaviour of CNN convolution and its hardware realization in a clear and modular manner.

V. SOFTWARE IMPLEMENTATION

The software implementation is carried out using Python and standard scientific computing libraries. The input image is converted into grayscale and normalized so that its pixel values can be processed numerically. Convolution is performed using a sliding-window approach, where a 3×3 kernel is moved over the image matrix and element-wise multiplication is followed by summation. The project includes multiple filters such as vertical edge detection, horizontal edge detection, and sharpening. These filters are used to generate feature maps and illustrate the effect of different kernel values on the same image.

Two different computational styles are used in the Python implementation. The first is a direct nested-loop method, which follows the straightforward mathematical definition of convolution. The second is an optimized approach that minimizes redundant computation and uses efficient matrix operations. The execution time of both methods is measured to show the performance difference. The output images are then visualized using Matplotlib, allowing easy inspection of feature extraction results. This stage is important because it validates the convolution concept before the same logic is mapped to hardware.

The software model is not intended to represent the full complexity of a deep CNN, but it accurately captures the core operation on which CNNs are built. By using grayscale images and small kernels, the system demonstrates feature extraction in a simplified and understandable form. This makes it useful as a learning and verification model for understanding how convolution works in image processing and how acceleration improves performance.

VI. HARDWARE IMPLEMENTATION

The hardware implementation translates convolution into RTL using Verilog HDL. The core element of the design is the Multiply-Accumulate unit, which performs the multiplication of input values and kernel coefficients and accumulates the resulting products. Since convolution is fundamentally a MAC operation repeated over multiple input positions, the MAC unit serves as the primary building block of the accelerator. In the implemented design, the MAC unit is synthesized and verified independently before being integrated into the larger convolution module.

The 3×3 convolution module combines multiple MAC operations to compute the output value corresponding to one kernel window. The design takes nine input pixel values and nine kernel values, multiplies them pairwise, and adds the results to obtain the final convolution output. This module reflects the hardware behaviour of a small CNN accelerator block. It is simple enough to be clearly understood, yet representative of the actual structures used in larger accelerator architectures.

The Verilog code is synthesized using Intel Quartus II, and the RTL viewer is used to analyse the generated hardware structure. The RTL diagrams confirm the presence of combinational multiplication logic, accumulation logic, and output registers. These results validate the correct hardware mapping of the convolution operation. Although the design is simplified, it provides a strong conceptual bridge between software-defined CNN computation and real hardware acceleration.

VII. EXPERIMENTAL RESULTS

The experimental evaluation includes both software outputs and hardware synthesis results. In the software domain, the grayscale input image is successfully processed using different convolution kernels. The resulting output images clearly show the effect of edge detection and sharpening. The vertical edge filter highlights intensity changes in the horizontal direction, while the horizontal edge filter emphasizes boundaries in the vertical direction. The sharpen filter enhances the contrast and detail in the image. These results confirm that the convolution logic is correctly implemented.

Performance measurements show that the optimized Python implementation executes faster than the basic loop-based version. This confirms that reducing

redundant computation and using more efficient processing methods can significantly improve performance. While the software implementation is still limited by sequential execution on a general-purpose processor, the speed improvement demonstrates the motivation for hardware acceleration.

On the hardware side, the Verilog modules are synthesized successfully in Quartus II. The RTL diagrams show the MAC unit and the 3×3 convolution module as structured hardware blocks with arithmetic datapaths. The multiplier and adder combinations visible in the RTL viewer confirm that the system is implemented as intended. This verifies that the convolution process can be represented at the hardware level using basic digital design elements. The combination of software outputs and hardware RTL results demonstrates both functional correctness and architectural feasibility.

VIII. ADVANTAGES AND APPLICATIONS

The proposed system offers several advantages. It provides a simple but effective demonstration of how CNN convolution works in software and how the same operation can be mapped into hardware. It also helps in understanding the relationship between algorithmic image processing and RTL design. Since the system is modular, it can be extended easily into larger accelerator structures. The use of Python makes the software behaviour easy to visualize, while Verilog and Quartus II provide a realistic view of hardware synthesis.

This type of accelerator model is relevant to multiple application domains. It can be used in educational environments to teach CNN basics and VLSI mapping concepts. It is also applicable to edge devices that require image processing with limited resources, such as surveillance cameras, small embedded boards, and portable AI systems. The architecture also reflects the principles used in more advanced deep learning accelerators, making it a useful starting point for future design work.

IX. FUTURE SCOPE

Although the present work focuses on a simplified CNN accelerator model, several extensions are possible in future research. The current design can be expanded to support larger kernels and multiple convolution layers, which would bring it closer to a full CNN pipeline. Parallel MAC arrays can also be introduced to further increase throughput and demonstrate more advanced accelerator behaviour. In addition, memory optimization techniques such as line buffering and data reuse can be added to reduce redundant image reads and improve hardware efficiency.

Future work may also include FPGA implementation and real-time image processing experiments. This would provide a stronger hardware validation of the design and enable practical performance measurement. The system can be extended for applications in embedded AI, edge computing, and lightweight vision systems. These enhancements would move the current model from an educational accelerator demonstration toward a more complete practical platform.

X. CONCLUSION

This paper presented the design and simulation of a hardware-aware CNN accelerator using Python and Verilog HDL. The project demonstrates the essential operation of convolution in CNNs and shows how it can be implemented both in software and in RTL hardware form. The software implementation verifies the functional behaviour of convolution using grayscale images and different filters, while the hardware implementation maps the same operation into a MAC-based architecture using a 3×3 convolution module.

The results confirm that convolution can be efficiently represented as a structured hardware computation problem. The Python experiments show the benefit of optimized processing, and the Quartus synthesis results validate the RTL structure of the MAC and convolution blocks. Together, these components provide a clear understanding of how CNN operations are translated into accelerator-based design.

The project serves as a practical foundation for further exploration of VLSI-based AI accelerators. It also demonstrates the usefulness of combining high-level algorithmic simulation with low-level hardware

implementation. Overall, the design successfully bridges the gap between CNN theory and hardware realization, making it relevant for education, research, and future accelerator development.

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press, 2016.
[2] David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann, 2017.
[3] Intel Quartus II Documentation, Intel Corporation.
[4] NumPy Documentation, https://numpy.org
[5] Matplotlib Documentation, https://matplotlib.org
[6] Pillow Documentation, https://python-pillow.org
[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, “Deep Learning,” Nature, 2015.
[8] Research articles and technical references on CNN accelerators, MAC-based architectures, and RTL design.