##### A Mini Project/Internship Report on

**Scalable NOC-Based AI Accelerator For Deep Learning Workloads** A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

**Bachelor of Technology In**

**Electronics & Communication Engineering**

Submitted by

D.SHRAVANI - 22H51A04M5

V.SYAMTARUN - 22H51A0463

T.AKHILA - 22H51A0457

A.SIDDHU - 22H51A04K6

S.AKHIL - 22H51A04J3

Under the esteemed guidance of Dr.R.BHARGAV RAM

Associate professor



**Department of ELECTRONICS & COMMUNICATION ENGINEERING**

### CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

\*Approved by AICTE \*Affiliated to JNTUH \*NAAC Accredited with A+ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

### 2024- 2025

**CMR COLLEGE OF ENGINEERING & TECHNOLOGY**

(UGC Autonomous)

\*Approved by AICTE \*Affiliated to JNTUH \*NAAC Accredited with A+ Grade

**Department of Electronics & Communication Engineering**

****

### CERTIFICATE

This is to certify that the Mini Project/ Internship report entitled "Scalable NOC- Based AI Accelerator For Deep Learning Workloads" being submitted by T.AKHILA (22H51A0457),V.SYAM TARUN (22H51A0463), S.AKHL (22H51A04J3), A.SIDDU (22H51A04K6),D.SRAVANI (22H51A04M5), in partial fulfillment for the award of Bachelor of Technology in ELECTRONICS & COMMUNICATION ENGINEERING is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree.

**Mini Project Co-Ordinator Head of the Department Dr. R. BHARGAV RAM Dr. P. Raveendra Babu**

**Associate professor Professor & Head of Dept of ECE CMRCET**

**EXTERNAL EXAMINER**

#### DECLARATION

We hereby declare that the project work titled **“Scalable NOC-Based AI Accelerator For Deep Learning Workloads”** is an original work carried out by us in partial fulfilment for the award of Bachelor of Technology in ELECTRONICS & COMMUNICATION ENGINEERING. This project report has been prepared based on our research and findings and has not been submitted to any other university or institute for the award of any degree or diploma.

We confirm that the work presented in this report is authentic and free from any form of plagiarism. Any references or sources used have been duly acknowledged.

Team Members:

T.AKHILA – 22H51A0457

V.SYAM TARUN – 22H51A0463

S.AKHIL – 22H51A04J3

A.SIDDU – 22H51A04K6

D.SRAVANI – 22H51A04M5

### ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express our heartfelt gratitude to all the people who helped in making this project a grand success.

We are grateful to **Dr.R.BHARGAV RAM ,Associate Professor**, Department of Electronics & Communication Engineering, for his/her valuable technical suggestions and guidance during the execution of this project work.

We express our sincere gratitude to **Mr.B.Venkataramanaiah**, Project Coordinator, Electronics & Communication Engineering Department, CMR College of Engineering & Technology, for his continuous guidance, encouragement, and support throughout the completion of our project.

We would like to thank, **Dr. P. Raveendrababu**, Head of the Department of Electronics & Communication Engineering, CMR College of Engineering & Technology, who is the major driving forces to complete our project work successfully.

We are very grateful to **Dr. Ghanta Devadasu**, Dean-Academics, CMR College of Engineering & Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Dr. A Seshu Kumar**, Principal, CMR College of Engineering & Technology, for unwavering support, and encouragement throughout the duration of this project.

We wish to extend our deepest gratitude to the esteemed Director **Major Dr. V A Narayana**, CMR College of Engineering & Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Electronics & Communication Engineering for their co-operation.

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary& Correspondent, CMR Group of Institutions, and **Shri Ch Abhinav Reddy**, CEO, CMR Group of Institutions for their continuous care and support.

Finally, we extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly or indirectly in completion of this project work

T.AKHILA 22H51A0457

V.SYAM TARUN 22H51A0463

S.AKHIL 22H51A04J3

A.SIDDU 22H51A04K6

D.SRAVANI 22H51A04M5

**TABLE OF CONTENTS**

**CHAPTER**

**NO. TITLE PAGE NO.**

LIST OF FIGURES vi

[LIST OF TABLES vii](#_TOC_250017)

[ABSTRACT viii](#_TOC_250016)

1. INTRODUCTION 1
   1. [Introduction 2](#_TOC_250015)
   2. [Problem Statement 3](#_TOC_250014)
   3. [Project Objective 3](#_TOC_250013)
   4. [Project Scope and Limitations 4](#_TOC_250012)
2. LITERATURE SURVEY 5
   1. [ASICs (Application-Specific Integrated Circuits) 6](#_TOC_250011)
      1. [Introduction 6](#_TOC_250010)
      2. Merits, Demerits and Challenges 7
      3. Implementation of ASICs (Application-Specific Integrated 7

Circuits)

* 1. [FPGA Based NOC AI Accelerator 8](#_TOC_250009)
     1. [Introduction 8](#_TOC_250008)
     2. Merits,Demerits and Challenges 8
     3. Implementation of FPGA Based NOC AI Accelerator 9
  2. [GPU Based NOC AI Accelerator 10](#_TOC_250007)
     1. [Introduction 10](#_TOC_250006)
     2. Merits,Demerits and Challenges 10
     3. Implementation of GPU Based Noc Ai Accelerator

**3 PROPOSED SOLUTION**

* 1. [Objective of Proposed Model 13](#_TOC_250005)
  2. [Algorithms/Block diagram of Proposed Model 13](#_TOC_250004)
  3. [Designing 14](#_TOC_250003)
  4. [Stepwise Implementation 15](#_TOC_250002)

4 RESULTS DISCUSSION 17

* 1. [Performance metrics 18](#_TOC_250001)

5 CONCLUSION 20

5.1 Conclusion and Future Enhancement 21

[REFERENCES 24](#_TOC_250000)

**APPENDIX**

|  |  |  |
| --- | --- | --- |
| **FIGURE** | **List of Figures** |  |
| **NO.** | **TITLE** | **PAGE NO.** |
| 3.2 | Scalable NoC-Based AI Accelerator for Deep Learning Workloads | 13 |
| 4.1 | waveforms | 19 |

##### List of Tables

|  |  |  |
| --- | --- | --- |
| **FIGURE**  **NO.** | **TITLE** |  |
| 5.1 | Pin Interface |  |

###### PAGE NO.

23

#### ABSTRACT

As deep learning models become more complex, there is a growing need for specialized hardware that can offer high performance, scalability, and energy efficiency. This project introduces an AI accelerator based on a Scalable Network-on-Chip (NoC) architecture, designed to efficiently handle demanding workloads such as Convolutional Neural Networks (CNNs) and transformer models. The system is built using a modular array of Processing Elements (PEs), each equipped with a multiply-accumulate (MAC) unit, local memory, and a dedicated NoC router. These PEs work concurrently to accelerate matrix-heavy operations that are fundamental to AI inference tasks.

A central Finite State Machine (FSM) coordinates the entire operation, overseeing data movement, computation, and output storage by directing traffic across the NoC. Thanks to its configurable and scalable design, the accelerator can be tailored to support various AI model sizes and complexities. It is suitable for both edge and cloud applications, making it a flexible solution for real-time AI in embedded systems, IoT devices, and high-performance data center environments. Simulation and synthesis results confirm that the architecture delivers notable gains in throughput, reduced latency, and better energy efficiency compared to traditional hardware solutions.

**CHAPTER-1**

## INTRODUCTION

### CHAPTER-1

**INTRODUCTION**

###### INTRODUCTION

The AI Accelerator is a specialized hardware design aimed at executing matrix computations efficiently, especially for neural network inference in deep learning models like CNNs and DNNs. As these models grow in complexity, they demand hardware with low latency, high throughput, and optimized energy usage. To address this, the accelerator incorporates a **Network-on-Chip (NoC)** architecture, enabling rapid distribution of data and weights to multiple **Processing Elements (PEs)** for parallel computation.

The architecture processes **128-bit input activations** and **128-bit weight data** concurrently across **four PEs**, with each PE producing a **64-bit output**, resulting in a combined **256-bit output**. Its **modular and scalable design**, along with an efficient routing mechanism, ensures seamless data flow and high-performance computation—making it well-suited for real-time deep learning applications.

* + - **DUT:** AI Accelerator

###### SUBMODULES:

* + - * NoC\_Router
      * Processing\_Elements

##### Features:

* + - * Dynamic data and weight routing
      * Multiply-accumulate operations
      * Parallel processing with scalable design

**Objective:** Efficient execution of deep learning workloads using a modular NoC-based architecture.

* 1. Problem Statement

Modern AI systems face major challenges in efficiently running deep learning models, primarily due to their intensive computational requirements and the cost of moving large volumes of data.

Conventional hardware architectures often fall short in terms of scalability and power efficiency. Although Network-on-Chip (NoC) designs provide a foundation for parallel processing, building an AI accelerator that delivers high performance and low latency—while maintaining efficient resource allocation and optimized data routing—remains a complex endeavor, especially for real-time inference tasks.

* 1. Project Objective

1. To design and develop a scalable Network-on-Chip (NoC)-based AI accelerator specifically optimized for deep learning inference tasks.
2. To construct processing elements (PEs) capable of performing efficient fixed-width multiply- accumulate (MAC) operations.
3. To implement a NoC router that enables high-throughput and low-latency data communication among interconnected PEs.
4. To validate the functionality and timing performance of the complete system through rigorous simulation and synthesis processes.
   1. Project scope and Limitations

**Scope:**

1. RTL design and simulation of NoC-based architecture with PEs and routers.
2. Support for fixed data sizes suited for CNN/DNN inference.
3. Performance analysis using synthesis and testbench results.

Limitations:

1. Limited to inference tasks; excludes training and floating-point operations.
2. No integration with processor cores or real peripherals.
3. Power analysis based on estimation tools, not physical implementation.

# CHAPTER-2

## LITERATURE SURVEY

### CHAPTER-2

**LITERATURE REVIEW**

* 1. ASICs (Application-Specific Integrated Circuits)
     1. Introduction

Application-Specific Integrated Circuits (ASICs) are custom-designed hardware chips created to perform a specific function or set of functions with maximum efficiency. In the context of artificial intelligence (AI) and deep learning, ASICs are developed to accelerate key operations such as matrix multiplications and activation functions that are commonly used in neural networks.

* + 1. Merits

**Design and Implementation**: Custom-built for a specific task, which allows for optimized performance and efficiency in deep learning workloads.

**User Experience**: Delivers faster computation with lower latency, resulting in real-time responsiveness—ideal for time-critical AI applications.

**Maintenance**: Once fabricated, ASICs are stable and require little to no maintenance; ideal for large-scale deployments.

**Power Consumption:** Consumes significantly less power compared to CPUs and GPUs, making it suitable for both data centers and portable device

Demerits

**Upgradability:** Cannot be easily updated or modified once manufactured, requiring complete redesign for any changes in functionality.

**Time to Market**: Takes a long time to design and manufacture, which can delay deployment in fast-changing AI environments.

**Flexibility**: Designed for specific tasks, making them difficult or impossible to reprogram or adapt to new models or algorithms.

**Use Cases**: Less suitable for dynamic applications where adaptability, frequent updates, or support for multiple AI models is required.

Challenges

**High Initial Investment:** Designing and fabricating ASICs requires significant financial resources, making it less accessible for startups or small projects.

**Long Development Time:** ASIC development involves complex design, verification, and fabrication processes, which can take several months or even years.

**Limited Flexibility:** Once manufactured, ASICs cannot be reprogrammed or reconfigured, making them unsuitable for evolving AI models or changing requirements.

**Design Complexity:** Requires specialized design tools, hardware expertise, and extensive testing to ensure functionality and efficiency.

* + 1. Implementation

1. Define AI workload requirements and design a scalable architecture with multiple Processing Elements (PEs) connected via a suitable NoC topology.
2. Develop specialized PEs optimized for AI operations with efficient arithmetic units and parallelism.
3. Design low-latency, high-bandwidth NoC routers and integrate them with PEs for efficient data flow.
4. Implement and verify the design in HDL, ensuring functionality and performance with AI workloads.
5. Synthesize, place-and-route, and optimize the design for power, timing, and signal integrity in the chosen ASIC technology.
6. Fabricate the chip, test with real AI tasks, deploy in systems, and iterate based on performance feedback.
   1. FPGA Based NOC AI Accelerator
      1. Introduction

FPGA-based NoC AI accelerators leverage the flexibility and reconfigurability of Field- Programmable Gate Arrays (FPGAs) to implement scalable AI processing architectures. By integrating a Network-on-Chip (NoC) within the FPGA fabric, multiple processing elements can communicate efficiently, enabling parallel execution of complex AI workloads such as convolutional neural networks and transformers. This approach allows rapid prototyping and customization for evolving AI models, offering a balance between performance and adaptability

* + 1. Merits

**Flexibility:** FPGA-based accelerators can be reprogrammed to support different AI models and algorithms, allowing rapid updates and customization.

**Rapid Prototyping:** Ideal for testing new AI architectures and algorithms before committing to expensive ASIC fabrication.

**Parallelism:** Supports highly parallel execution of AI workloads through configurable processing elements and interconnects like NoC.

**Lower Development Cost:** Compared to ASICs, FPGAs have lower upfront costs and faster time- to-market.

Demerits

**Upgradability:** Cannot be easily updated or modified once manufactured, requiring a full redesign for functional changes.

**Time to Market:** Long design and fabrication cycles delay deployment, which is challenging in rapidly evolving AI fields.

**Flexibility:** Fixed-function design limits adaptability to new AI models or algorithms.

**Use Cases:** Less suited for dynamic applications needing frequent updates or support for diverse AI workloads

Challenges

**Limited Performance Compared to ASICs:** FPGAs generally deliver lower peak performance and throughput than custom ASIC implementations.

**Higher Power Consumption:** FPGA architectures tend to consume more power for the same AI workload compared to optimized ASICs.

**Resource Constraints:** Limited on-chip resources (logic blocks, memory) can restrict the size and complexity of AI models implemented.

**Design Complexity and Toolchain:** Programming and optimizing NoC and AI accelerators on FPGAs require specialized knowledge and complex toolflows.

* + 1. Implementation

1. Define AI workload requirements and design a scalable architecture with multiple configurable Processing Elements (PEs) connected via a flexible NoC topology.
2. Develop parameterizable PEs optimized for AI operations, leveraging FPGA resources like DSP blocks and on-chip memory.
3. Design low-latency, high-bandwidth NoC routers within the FPGA fabric and integrate them with PEs for efficient data communication.
4. Implement and verify the design using HDL or high-level synthesis (HLS) tools, simulating AI workloads for functionality and performance.
5. Map and optimize the design on the FPGA, managing resource utilization, timing closure, and power consumption through place-and-route tools.
6. Deploy the FPGA design, test with real AI applications, and update or reconfigure the accelerator as needed based on feedback.
   1. GPU Based NOC AI Accelerator
      1. Introduction

GPU-based NoC AI accelerators utilize the massive parallelism and high computational throughput of Graphics Processing Units (GPUs), enhanced by internal Network-on-Chip (NoC) interconnects. Modern GPUs feature hundreds to thousands of cores connected via scalable mesh or ring-based NoCs, enabling fast and efficient data exchange across processing units. This architecture is well-suited for deep learning tasks such as training large neural networks, where high memory bandwidth and core-to-core communication are critical.

* + 1. Merits

**Massive Parallelism**: Thousands of cores enable simultaneous execution of AI operations like matrix multiplications and convolutions.

**Efficient On-Chip Communication**: Integrated NoC (mesh or ring topology) ensures high-speed data transfer between cores and memory blocks, reducing latency.

**High Throughput for AI Workloads**: Ideal for training large-scale models (e.g., transformers, CNNs) due to strong compute and memory capabilities.

**Programmability and Flexibility**: Supports a wide range of AI models and frameworks (e.g., TensorFlow, PyTorch), making it easy to adapt to new algorithms.

Demerits

**High Power Consumption**: GPUs consume significant power, especially under full load, making them less suitable for power-constrained edge devices.

**Latency in Real-Time Applications**: Despite high throughput, latency can be higher than ASICs or FPGAs in real-time, time-critical AI applications.

**Cost and Size**: High-performance GPUs are expensive and often require large physical space and cooling infrastructure.

**General-Purpose Overhead**: GPUs are designed for general parallel processing, which may introduce inefficiencies compared to task-specific hardware.

Challenges

**Scalability Bottlenecks**: As models grow larger, NoC interconnects may face congestion, limiting efficient scaling across all GPU cores.

**Memory Bandwidth Limitations**: Although GPUs have high memory bandwidth, it may still be insufficient for large AI models, leading to data transfer bottlenecks.

**Inefficiency for Sparse or Irregular Workloads**: GPUs are optimized for dense, regular computations; performance drops with sparse or dynamic AI tasks.

**Limited Real-Time Suitability**: GPUs may not meet the strict timing requirements of real-time AI applications due to scheduling and queuing overhead.

* + 1. Implementation

1. Define AI workload requirements and choose a GPU architecture with suitable internal NoC (e.g., mesh or ring topology).
2. Configure GPU cores and thread blocks to parallelize AI operations effectively.
3. Utilize the GPU’s internal NoC for efficient data transfer between cores and memory units.
4. Develop and optimize AI kernels using CUDA/OpenCL for compute and memory efficiency.
5. Manage memory and execution flow using streams, shared memory, and unified memory models.
6. Deploy, test on real AI workloads, and optimize based on performance metrics.

# CHAPTER-3

## PROPOSED SYSTEM

### CHAPTER-3

**PROPOSED SYSTEM**

* 1. Objective of Proposed Model

1. Develop a scalable NoC architecture for efficient, low-latency communication among multiple AI processing cores.
2. Implement flexible packet-based routing to support diverse deep learning workloads.
3. Optimize power efficiency through intelligent data transfer and routing mechanisms.
4. Design configurable MAC units to enhance adaptability across various neural network models.
5. Ensure seamless integration with standard interfaces like AXI4-Lite for robust system control.
   1. Algorithms/Block Diagram of Proposed Model

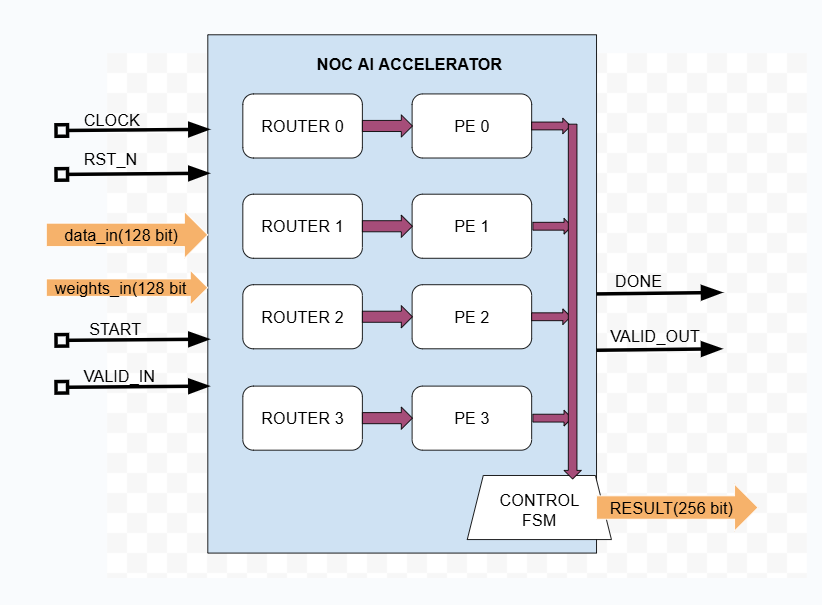


Fig.3.2.1 Scalable NoC-Based AI Accelerator for Deep Learning Workloads

Algorithms used for Proposed Model

1. **Write Transaction Algorithm (for NoC Write Channels)**

Objective: Receive write requests with addresses and data from masters, route packets through NoC to target nodes, store data, and send acknowledgment.

1. Read Transaction Algorithm (for NoC Read Channels)

Objective: Receive read requests from masters, route to target nodes, retrieve data, and send response back.

Common Algorithm Characteristics:

* + Read and write channels operate independently with handshake protocols.
  + Address decoding done via routing tables or mapping logic.
  + Nodes respond only to incoming requests, not initiate them.
  + Flow control and congestion management ensure smooth data transfer
  1. Designing

1. Input Interface: Takes in 128-bit data and weights along with control signals (Clock, Reset, Start, Valid In).
2. Router-PE Structure: Four routers (Router 0–3) distribute data to four processing elements (PE0– PE3) for parallel processing.
3. Processing Elements: Each PE performs neural computations (e.g., MAC operations) on the incoming data and weights.
4. Control FSM: Manages the operation flow, synchronizes processing, and collects results from all PEs.
5. Output Interface: Produces a 256-bit final result with Done and Valid Out signals to indicate completion and validity.
   1. Stepwise Implementation
6. The system is first initialized by applying the clock signal and asserting the reset signal (RST-N) low to ensure all modules are in a known state. After a few clock cycles, the reset signal is deasserted to allow normal operation to begin.
7. Computation is initiated by asserting the START signal high. At the same time, input data (DATA IN) and weights (WEIGHTS IN), both 128-bit wide, are provided to the system. The VALID IN signal is also asserted to indicate that the input values are valid and ready for processing.
8. The input data and weights are routed through four routers (Router 0 to Router 3), each responsible for forwarding the respective input streams to the corresponding processing elements (PE 0 to PE 3).
9. Each processing element (PE) receives its allocated data and weights and performs computations in parallel. Typically, these computations involve multiply-accumulate (MAC) operations, commonly used in neural network inference.
10. The Control FSM (Finite State Machine) plays a critical role by managing the overall flow of operations. It synchronizes the activity of all routers and PEs, tracks the computation status, and ensures that partial results are correctly accumulated and processed.
11. Once all computations are complete, the Control FSM gathers the outputs from each processing element and consolidates them into a single 256-bit result. The system then asserts the VALID OUT signal to indicate that the output is ready and the DONE signal to mark the completion of the computation cycle.
12. The accelerator is now ready to either accept new input data for the next computation cycle or return to an idle state until reactivated by another START signal.

# CHAPTER-4

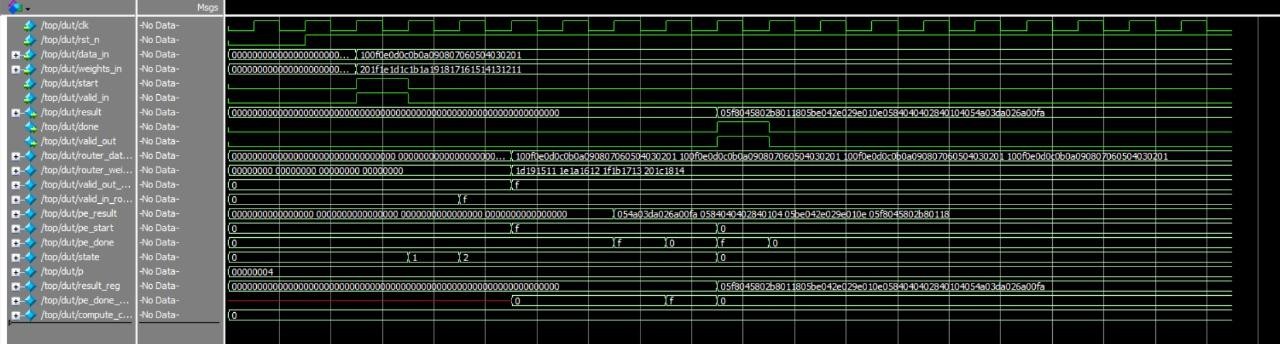
## RESULTS AND DISCUSSIONS

### CHAPTER -4

**RESULTS AND DISCUSSION**

* 1. Performance Metrics
     1. Throughput – The rate at which the accelerator processes data or completes operations, typically measured in operations per second.
     2. Latency – The time taken from input to output for a single inference or computation task.
     3. MAC Efficiency – The percentage of time the multiply-accumulate units are actively performing useful computations.
     4. Network Latency & Bandwidth – Measures the communication delay and data transfer capacity between processing elements over the NoC.
     5. Scalability – The ability of the architecture to maintain or improve performance as the number of processing elements increases.
     6. Power Consumption – The total electrical power used by the accelerator during operation.
     7. Energy Efficiency – The amount of computation achieved per unit of energy consumed, often expressed in TOPS/W.
     8. Area Utilization – The amount of hardware or silicon space required to implement the accelerator design.
     9. Accuracy Retention – The extent to which the original model’s prediction accuracy is preserved after hardware optimization or quantization.

**OUTPUT:**

****

# CHAPTER-5

## CONCLUSION

### CHAPTER -5 CONCLUSION

* 1. Conclusion and Future Enhancements CONCLUSION

The proposed scalable Network-on-Chip (NoC)-based AI accelerator is engineered to efficiently interconnect multiple processing elements (PEs), enabling fast and seamless communication that is critical for handling the demands of deep learning inference workloads. By leveraging a high- throughput, low-latency NoC infrastructure, the system ensures rapid data exchange between PEs, which significantly boosts overall computational performance. Each PE is equipped with a configurable multiply-accumulate (MAC) unit and local memory, allowing the architecture to be tailored for various model sizes and precision requirements.The modular design supports flexible routing algorithms, enabling efficient load balancing and reducing congestion within the network. This adaptability allows the accelerator to maintain optimal performance across different deep learning architectures, such as Convolutional Neural Networks (CNNs) and transformer-based models. Through its balanced approach to scalability, efficiency, and performance, this NoC-based accelerator enhances AI inference capabilities and offers a robust solution for integration into next- generation System-on-Chip (SoC) platforms. Its versatility and efficiency make it a strong candidate for deployment in a wide range of AI-powered applications, from smart IoT devices to large-scale cloud-based inference engines.

Future Enhancement:

1. **Deep Learning Inference:**

Optimizes matrix multiplications for CNNs/DNNs used in tasks such as facial recognition, voice commands, and autonomous systems.

Example: Handles 128-bit input activations and 32-bit weight values in convolution operations.

1. IoT and Edge Applications:

Designed for energy-efficient processing in edge environments, ideal for real-time analytics on sensor data from smart devices.

1. Data Center Acceleration:

Enables large-scale AI workloads through multiple parallel processing units, supporting intensive computations for cloud-based platforms.

1. Embedded Intelligence:

Can be integrated into embedded systems for smart robotics, supporting functions like object tracking and decision-making with low delay.

**Pin Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin Name** | **Direction** | **Width** | **Description** |
| Clk | INPUT | 1 | System clock for synch operation |
| Rst\_n | INPUT | 1 | Active low reset |
| Data\_in | INPUT | 128 | Input activations data |
| Weights\_in | INPUT | 128 | Input weights(32 bits per PE) |
| Start | INPUT | 1 | Initiates computation |
| Valid\_in | INPUT | 1 | Initiates valid input data |
| Result | OUTPUT | 256 | Computation Result(4x64 bits) |
| Done | OUTPUT | 1 | Signals computation completion |
| Valid\_out | OUTPUT | 1 | Indicates valid result output |

#### REFERENCES

* 1. **Bjerregaard, T., & Mahadevan, S. (2025).** A survey of research and practices of Network-on-Chip. *ACM Computing Surveys (CSUR)*.
  2. **Chen, Y., Emer, J., & Sze, V. (2025).** Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*.
  3. **Benini, L., & De Micheli, G. (2025).** Networks on Chips: A New SoC Paradigm. *IEEE Computer Society*.

#### APPENDIX

**Source Code:**

#### NOC\_AI\_TOP

module ai\_accelerator ( input clk,

input rst\_n,

input [127:0] data\_in, // 128-bit activations (same for all PEs) input [127:0] weights\_in, // 128-bit weights (32 bits per PE) input start,

input valid\_in,

output [255:0] result, // 256-bit output (4 × 64 bits) output reg done,

output reg valid\_out

);

wire [127:0] router\_data\_out [0:3];

wire [31:0] router\_weights\_out [0:3];

wire [3:0] valid\_out\_router;

reg [127:0] router\_data\_in [0:3];

reg [31:0] router\_weights\_in [0:3];

reg [3:0] valid\_in\_router;

reg [1:0] dest\_xy\_reg [0:3];

reg [127:0] pe\_data\_in [0:3];

reg [31:0] pe\_weights [0:3];

wire [63:0] pe\_result [0:3];

reg [3:0] pe\_start;

wire [3:0] pe\_done;

reg [2:0] state;

localparam IDLE = 0, LOAD = 1, COMPUTE = 2, STORE = 3;

integer p;

reg [255:0] result\_reg; // 256 bits

reg [3:0] pe\_done\_packed;

reg [1:0] compute\_count;

// Router instantiations noc\_router u\_router\_0 (

.clk(clk),

.rst\_n(rst\_n),

.data\_in(router\_data\_in[0]),

.weights\_in(router\_weights\_in[0]),

.src\_xy(2'b00),

.dest\_xy(dest\_xy\_reg[0]),

.valid\_in(valid\_in\_router[0]),

.data\_out(router\_data\_out[0]),

.weights\_out(router\_weights\_out[0]),

.valid\_out(valid\_out\_router[0])

);

noc\_router u\_router\_1 (

.clk(clk),

.rst\_n(rst\_n),

.data\_in(router\_data\_in[1]),

.weights\_in(router\_weights\_in[1]),

.src\_xy(2'b01),

.dest\_xy(dest\_xy\_reg[1]),

.valid\_in(valid\_in\_router[1]),

.data\_out(router\_data\_out[1]),

.weights\_out(router\_weights\_out[1]),

.valid\_out(valid\_out\_router[1])

);

noc\_router u\_router\_2 (

.clk(clk),

.rst\_n(rst\_n),

.data\_in(router\_data\_in[2]),

.weights\_in(router\_weights\_in[2]),

.src\_xy(2'b10),

.dest\_xy(dest\_xy\_reg[2]),

.valid\_in(valid\_in\_router[2]),

.data\_out(router\_data\_out[2]),

.weights\_out(router\_weights\_out[2]),

.valid\_out(valid\_out\_router[2])

);

noc\_router u\_router\_3 (

.clk(clk),

.rst\_n(rst\_n),

.data\_in(router\_data\_in[3]),

.weights\_in(router\_weights\_in[3]),

.src\_xy(2'b11),

.dest\_xy(dest\_xy\_reg[3]),

.valid\_in(valid\_in\_router[3]),

.data\_out(router\_data\_out[3]),

.weights\_out(router\_weights\_out[3]),

.valid\_out(valid\_out\_router[3])

);

// PE instantiations

pe u\_pe\_0 (.clk(clk), .rst\_n(rst\_n), .data\_in(pe\_data\_in[0]),

.weights(pe\_weights[0]), .start(pe\_start[0]), .result(pe\_result[0]), .done(pe\_done[0]));

pe u\_pe\_1 (.clk(clk), .rst\_n(rst\_n), .data\_in(pe\_data\_in[1]),

.weights(pe\_weights[1]), .start(pe\_start[1]), .result(pe\_result[1]), .done(pe\_done[1]));

pe u\_pe\_2 (.clk(clk), .rst\_n(rst\_n), .data\_in(pe\_data\_in[2]),

.weights(pe\_weights[2]), .start(pe\_start[2]), .result(pe\_result[2]), .done(pe\_done[2]));

pe u\_pe\_3 (.clk(clk), .rst\_n(rst\_n), .data\_in(pe\_data\_in[3]),

.weights(pe\_weights[3]), .start(pe\_start[3]), .result(pe\_result[3]), .done(pe\_done[3]));

// Router to PE connections

always @(posedge clk or negedge rst\_n) begin if (!rst\_n) begin

for (p = 0; p < 4; p = p + 1) begin pe\_data\_in[p] <= 128'b0; pe\_weights[p] <= 32'b0;

end

end else begin

if (valid\_out\_router[0]) begin

pe\_data\_in[0] <= router\_data\_out[0]; // Full 128-bit data from router pe\_weights[0] <= router\_weights\_out[0];

end

if (valid\_out\_router[1]) begin pe\_data\_in[1] <= router\_data\_out[1]; pe\_weights[1] <= router\_weights\_out[1];

end

if (valid\_out\_router[2]) begin pe\_data\_in[2] <= router\_data\_out[2]; pe\_weights[2] <= router\_weights\_out[2];

end

if (valid\_out\_router[3]) begin pe\_data\_in[3] <= router\_data\_out[3]; pe\_weights[3] <= router\_weights\_out[3];

end end

end

// Main control logic

always @(posedge clk or negedge rst\_n) begin if (!rst\_n) begin

state <= IDLE; done <= 0;

valid\_out <= 0; pe\_start <= 4'b0; result\_reg <= 256'b0; compute\_count <= 0;

for (p = 0; p < 4; p = p + 1) begin router\_data\_in[p] <= 128'b0; router\_weights\_in[p] <= 32'b0; valid\_in\_router[p] <= 1'b0; dest\_xy\_reg[p] <= 2'b00;

end

end else begin case (state)

IDLE: begin done <= 0;

valid\_out <= 0; pe\_start <= 4'b0; compute\_count <= 0;

if (start && valid\_in) state <= LOAD;

end

LOAD: begin

// Feed full data\_in and unique weights to routers router\_data\_in[0] <= data\_in;

router\_weights\_in[0] <= {weights\_in[103:96], weights\_in[71:64], weights\_in[39:32], weights\_in[7:0]};

router\_data\_in[1] <= data\_in;

router\_weights\_in[1] <= {weights\_in[111:104], weights\_in[79:72], weights\_in[47:40], weights\_in[15:8]};

router\_data\_in[2] <= data\_in;

router\_weights\_in[2] <= {weights\_in[119:112], weights\_in[87:80], weights\_in[55:48], weights\_in[23:16]};

router\_data\_in[3] <= data\_in;

router\_weights\_in[3] <= {weights\_in[127:120], weights\_in[95:88], weights\_in[63:56], weights\_in[31:24]};

valid\_in\_router <= 4'b1111; dest\_xy\_reg[0] <= 2'b00; // Local delivery dest\_xy\_reg[1] <= 2'b01;

dest\_xy\_reg[2] <= 2'b10; dest\_xy\_reg[3] <= 2'b11; state <= COMPUTE;

end

COMPUTE: begin pe\_start <= 4'b1111;

pe\_done\_packed <= {pe\_done[3], pe\_done[2], pe\_done[1], pe\_done[0]}; if (&pe\_done\_packed) begin

pe\_start <= 4'b0000;

for (p = 0; p < 4; p = p + 1) begin

result\_reg[p\*64 +: 64] <= pe\_result[p]; // Full 64 bits per

PE

end

done <= 1;

valid\_out <= 1; state <= IDLE;

end end

endcase end

end

assign result = result\_reg; endmodule

noc\_router :-

module noc\_router( input clk,

input rst\_n,

input [127:0] data\_in,

input [31:0] weights\_in,

input [1:0] src\_xy,

input [1:0] dest\_xy, input valid\_in,

output reg [127:0] data\_out,

output reg [31:0] weights\_out, output reg valid\_out

);

always @(posedge clk or negedge rst\_n) begin if (!rst\_n) begin

data\_out <= 128'b0; weights\_out <= 32'b0; valid\_out <= 0;

end else begin

if (valid\_in && src\_xy == dest\_xy) begin data\_out <= data\_in;

weights\_out <= weights\_in; valid\_out <= 1;

end else begin valid\_out <= 0;

end end

end

endmodule

##### processing element:-

module pe( input clk, input rst\_n,

input [127:0] data\_in,

input [31:0] weights,

input start,

output reg [63:0] result, output reg done

);

reg [15:0] temp0, temp1, temp2, temp3; reg [1:0] compute\_stage;

reg processing;

always @(posedge clk or negedge rst\_n) begin if (!rst\_n) begin

result <= 64'b0; done <= 0; temp0 <= 16'b0; temp1 <= 16'b0; temp2 <= 16'b0; temp3 <= 16'b0;

compute\_stage <= 2'b00; processing <= 0;

end else begin

if (start && !processing) begin processing <= 1; compute\_stage <= 2'b01; done <= 0;

end

if (processing) begin case (compute\_stage)

2'b01: begin

result[15:0] <= data\_in[7:0] \* weights[7:0] + data\_in[15:8] \* weights[15:8] + data\_in[23:16] \* weights[23:16] + data\_in[31:24] \* weights[31:24];

result[31:16] <= data\_in[39:32] \* weights[7:0] + data\_in[47:40]

\* weights[15:8] + data\_in[55:48] \* weights[23:16] + data\_in[63:56] \* weights[31:24];

result[47:32] <= data\_in[71:64] \* weights[7:0] + data\_in[79:72]

\* weights[15:8] + data\_in[87:80] \* weights[23:16] + data\_in[95:88] \* weights[31:24];

result[63:48] <= data\_in[103:96] \* weights[7:0] + data\_in[111:104] \* weights[15:8] + data\_in[119:112] \* weights[23:16] + data\_in[127:120] \* weights[31:24];

done <= 1;

processing <= 0;

end

endcase end else begin

done <= 0; end

end

end endmodule

##### TestBench :-

`include "../rtl/PE.v"

`include "../rtl/NoC\_Router.v"

`include "../rtl/NoC\_Mesh.v"

module top; reg clk; reg rst\_n;

reg [255:0] mem\_data\_in; reg start;

reg valid\_in;

wire [255:0] result; wire done;

wire valid\_out; integer i;

ai\_accelerator dut (

.clk(clk),

.rst\_n(rst\_n),

.data\_in(mem\_data\_in[127:0]),

.weights\_in(mem\_data\_in[255:128]),

.start(start),

.valid\_in(valid\_in),

.result(result),

.done(done),

.valid\_out(valid\_out)

);

initial begin clk = 0;

forever #5 clk = ~clk; end

initial begin rst\_n = 0;

start = 0;

valid\_in = 0; mem\_data\_in = 256'b0;

#10;

@(posedge clk) rst\_n = 1;

@(posedge clk) begin

for (i = 0; i < 32; i = i + 1) mem\_data\_in[i\*8 +: 8] = i + 1; valid\_in = 1;

start = 1; end

@(posedge clk) begin start = 0;

valid\_in = 0; end

16]);

wait(valid\_out); @(posedge clk) begin

$display("Results for Matrix Set 1:");

for (i = 0; i < 16; i = i + 1) $display("result[%0d] = %d", i, result[i\*16 +: end

#40;

@(posedge clk) begin

mem\_data\_in[7:0] = 8'd10; mem\_data\_in[15:8] = 8'd20; mem\_data\_in[23:16] = 8'd30; mem\_data\_in[31:24] = 8'd40; mem\_data\_in[39:32] = 8'd50; mem\_data\_in[47:40] = 8'd60; mem\_data\_in[55:48] = 8'd70; mem\_data\_in[63:56] = 8'd80; mem\_data\_in[71:64] = 8'd90; mem\_data\_in[79:72] = 8'd100; mem\_data\_in[87:80] = 8'd110; mem\_data\_in[95:88] = 8'd120; mem\_data\_in[103:96] = 8'd130; mem\_data\_in[111:104] = 8'd140; mem\_data\_in[119:112] = 8'd150; mem\_data\_in[127:120] = 8'd160;

mem\_data\_in[135:128] = 8'd1; mem\_data\_in[143:136] = 8'd2; mem\_data\_in[151:144] = 8'd3; mem\_data\_in[159:152] = 8'd4; mem\_data\_in[167:160] = 8'd5; mem\_data\_in[175:168] = 8'd6; mem\_data\_in[183:176] = 8'd7; mem\_data\_in[191:184] = 8'd8; mem\_data\_in[199:192] = 8'd9; mem\_data\_in[207:200] = 8'd10; mem\_data\_in[215:208] = 8'd11; mem\_data\_in[223:216] = 8'd12; mem\_data\_in[231:224] = 8'd13; mem\_data\_in[239:232] = 8'd14; mem\_data\_in[247:240] = 8'd15; mem\_data\_in[255:248] = 8'd16; valid\_in = 1;

start = 1; end

@(posedge clk) begin start = 0;

valid\_in = 0; end

16]);

wait(valid\_out); @(posedge clk) begin

$display("Results for Matrix Set 2:");

for (i = 0; i < 16; i = i + 1) $display("result[%0d] = %d", i, result[i\*16 +: end

#90

@(posedge clk) begin mem\_data\_in[7:0] = 8'd1; mem\_data\_in[15:8] = 8'd1; mem\_data\_in[23:16] = 8'd1; mem\_data\_in[31:24] = 8'd1; mem\_data\_in[39:32] = 8'd1; mem\_data\_in[47:40] = 8'd1; mem\_data\_in[55:48] = 8'd1; mem\_data\_in[63:56] = 8'd1; mem\_data\_in[71:64] = 8'd1; mem\_data\_in[79:72] = 8'd1; mem\_data\_in[87:80] = 8'd1; mem\_data\_in[95:88] = 8'd1; mem\_data\_in[103:96] = 8'd1; mem\_data\_in[111:104] = 8'd1; mem\_data\_in[119:112] = 8'd1; mem\_data\_in[127:120] = 8'd1; mem\_data\_in[135:128] = 8'd1; mem\_data\_in[143:136] = 8'd1; mem\_data\_in[151:144] = 8'd1; mem\_data\_in[159:152] = 8'd1; mem\_data\_in[167:160] = 8'd1; mem\_data\_in[175:168] = 8'd1; mem\_data\_in[183:176] = 8'd1; mem\_data\_in[191:184] = 8'd1;

16]);

mem\_data\_in[199:192] = 8'd1; mem\_data\_in[207:200] = 8'd1; mem\_data\_in[215:208] = 8'd1; mem\_data\_in[223:216] = 8'd1; mem\_data\_in[231:224] = 8'd1; mem\_data\_in[239:232] = 8'd1; mem\_data\_in[247:240] = 8'd1; mem\_data\_in[255:248] = 8'd1; valid\_in = 1;

start = 1; end

@(posedge clk) begin start = 0;

valid\_in = 0; end wait(valid\_out);

@(posedge clk) begin

$display("Results for Matrix Set 2:");

for (i = 0; i < 16; i = i + 1) $display("result[%0d] = %d", i, result[i\*16 +:

end

#90 $finish;

end endmodule

### Testbench:

//`timescale 1ns / 1ps

`include "../rtl/axi4\_lite\_master.sv"

`include "../rtl/axi4\_lite\_slave.sv"

`include "../rtl/axi4\_lite\_top.sv"

module axi4\_lite\_tb;

// Parameters

localparam CLK\_PERIOD = 10;

// Signals reg clk;

reg reset\_n;

reg write\_en;

reg read\_en;

reg [31:0] write\_data\_in;

reg [31:0] write\_addr\_in;

reg [31:0] read\_addr\_in;

reg [3:0] strobe\_in;

wire [31:0] read\_data\_out;

wire [1:0] write\_response\_out;

wire [1:0] read\_response\_out; wire write\_done;

wire read\_done;

// DUT Instantiation axi4\_lite\_top dut (

.clk (clk),

.reset\_n (reset\_n),

.write\_en (write\_en),

.read\_en (read\_en),

.write\_data\_in (write\_data\_in),

.write\_addr\_in (write\_addr\_in),

.read\_addr\_in (read\_addr\_in),

.strobe\_in (strobe\_in),

.read\_data\_out (read\_data\_out),

.write\_response\_out (write\_response\_out),

.read\_response\_out (read\_response\_out),

.write\_done (write\_done),

.read\_done (read\_done)

);

// Clock Generation initial begin

clk = 0;

forever #(CLK\_PERIOD/2) clk = ~clk; end

// Test Procedure initial begin

// Initialize reset\_n = 0;

write\_en = 0;

read\_en = 0;

write\_data\_in = 0;

write\_addr\_in = 0;

read\_addr\_in = 0;

strobe\_in = 0;

#20;

reset\_n = 1;

#20;

// Test 1: Reset Behavior

$display("\nTest 1: Reset Behavior"); #10;

if (read\_data\_out != 0 || write\_response\_out != 0 || read\_response\_out != 0 || write\_done != 0 || read\_done != 0)

$display("ERROR: Outputs not reset"); else

$display("Reset OK");

// Test 2: Random Write

$display("\nTest 2: Random Write"); write\_en = 1;

write\_data\_in = 32'hABCD\_1234; write\_addr\_in = 32'h0000\_0004; strobe\_in = 4'b1111;

#10;

write\_en = 0;

wait (write\_done);

$display("Write Response=%b (Expected 00)", write\_response\_out); if (write\_response\_out != 2'b00) $display("ERROR: Write failed");

// #20; #(CLK\_PERIOD); #(CLK\_PERIOD);

//test 3 : MUltiple writes write\_en = 1;

write\_data\_in = 32'hABCD; write\_addr\_in = 32'h0000\_000C; strobe\_in = 4'b1111;

#10;

write\_en = 0;

wait (write\_done);

$display("Write Response=%b ", write\_response\_out);

//#20; #(CLK\_PERIOD); #(CLK\_PERIOD);

write\_en = 1; write\_data\_in = 32'hEF10;

write\_addr\_in = 32'h0000\_0008; strobe\_in = 4'b1111;

#10;

write\_en = 0;

wait (write\_done);

$display("Write Response=%b ", write\_response\_out);

//$display("Write Response=%b (Expected 00)", write\_response\_out);

//if (write\_response\_out != 2'b00) $display("ERROR: Write failed");

// Test 3: Random Read #20;

$display("\nTest 3: Random Read"); read\_en = 1;

read\_addr\_in = 32'h0000\_0004; #10;

read\_en = 0;

wait (read\_done);

$display("Read Data=%h (Expected ABCD1234), Read Response=%b (Expected 00)", read\_data\_out, read\_response\_out);

if (read\_data\_out != 32'hABCD\_1234 || read\_response\_out != 2'b00)

$display("ERROR: Read failed");

// Test 4: Write with Strobes #20;

$display("\nTest 4: Write with Strobes"); write\_en = 1;

write\_data\_in = 32'hCAFE\_BADD; write\_addr\_in = 32'h0000\_0008; strobe\_in = 4'b0011;

#10;

write\_en = 0;

wait (write\_done);

$display("Write Response=%b (Expected 00)", write\_response\_out); if (write\_response\_out != 2'b00) $display("ERROR: Write failed");

// Test 5: Read to Verify Strobes

#20;

$display("\nTest 5: Read to Verify Strobes"); read\_en = 1;

read\_addr\_in = 32'h0000\_0008; #10;

read\_en = 0;

wait (read\_done);

$display("Read Data=%h (Expected 0000BADD), Read Response=%b (Expected 00)", read\_data\_out, read\_response\_out);

if (read\_data\_out[15:0] != 16'hFBADD || read\_response\_out != 2'b00)

$display("ERROR: Strobe write/read failed");

// Test 6: Error Response #20;

$display("\nTest 6: Error Response"); write\_en = 1;

write\_data\_in = 32'hFFFF\_FFFF; write\_addr\_in = 32'h0000\_0400; strobe\_in = 4'b1111;

#10;

write\_en = 0;

wait (write\_done);

$display("Write Response=%b (Expected 10)", write\_response\_out); if (write\_response\_out != 2'b10)

$display("ERROR: Expected SLVERR not detected");

// Finish #100;

$display("Simulation Complete");

$finish; end

endmodule