Efficient In-Memory Computing Architecture using SRAM

M.Tech. Thesis

By

AKASH PANDEY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

MAY 2025

Efficient In-Memory Computing Architecture using SRAM

A Thesis

Submitted in partial fulfillment of the

requirements for the award of the degree

of

M.Tech. Thesis

by

AKASH PANDEY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

MAY 2025

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Efficient In-Memory Computing Architecture using SRAM** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology** and submitted in the **Department of Electrical Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from July 2023 to May 2025 under the supervision of Dr. Santosh Kumar Vishvakarma, Professor, Indian Institute of Technology Indore, Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Akashpandey
19/05/2025

Signature of the Student with Date

(Akash Pandey)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

19/05/2025

Signature of the Supervisor of M.Tech. Thesis with Date

(Prof. Santosh Kumar Vishvakarma)

Akash Pandey has successfully given her M.Tech. Oral Examination held on **5th May 2025**

19/05/2025

Signature of Supervisor of M.Tech. Thesis

Date:

Saptarshi Ghosh

Signature of Convener, DPGC

Date: 19/05/2025

ACKNOWLEDGEMENTS

I am immensely grateful to my M.Tech. thesis supervisor and mentor, Prof. Santosh Kumar Vishvakarma, for consistently encouraging and supporting me in both my research and personal growth. His unwavering belief in my abilities and his invaluable guidance have served as constant motivation, pushing me to exceed my own limits. I owe him a debt of gratitude for granting me the freedom to explore my research interests and allowing my novel ideas to flourish.

I would also like to extend my sincere appreciation to all of my thesis evaluation committee. Their impartial evaluations and thought-provoking questions have contributed significantly to expanding my research perspective.

My family has played a major role in supporting my research work throughout the course of my master's. They have always boosted my confidence and always motivated me to push my limits. I will always be grateful to them for all their guidance, love and sacrifices. Their faith in me has brought me this far, and it will drive me further, as well, to achieve greater things. I deeply appreciate the Nanoscale Devices, VLSI Circuit System Design Lab (NSDCS) research group, especially Mr. Vikas Vishwakarma, Mr. Sagar Patel, Mr. Radheshyam Sharma, Ms. Komal Gupta for their continuous support and guidance. I am also grateful to my friends and labmates, Mrs. Neha Maheshwari, Mr. Sonu Kumar, Mr. Shashank Singh Rawat, Mr. Mukul Lokhande, Mr. Shivam Vaish, Mr. Omkar Kokane, Mr. Ankit Tenwar, Ms. Shivangi Mishra, and Mr. Akash Sankhe, whose camaraderie and encouragement made my time at the institute truly memorable.

Akash Pandey

This Thesis is Dedicated

to

My Parents, My Brother, My Grandparents
and the Almighty God

ABSTRACT

Compute-In-Memory (CIM) is a promising architectural paradigm that seeks to overcome the memory–compute bottleneck inherent in traditional von Neumann architectures. By integrating computation capabilities directly within the memory arrays, CIM enables localized data processing and significantly reduces energy and latency costs associated with frequent data movement between memory and processing units. This approach is particularly beneficial for data-intensive tasks such as neural network inference, where matrix-vector multiplication (VMM) is a dominant operation. Digital CIM architectures are favored for their accuracy, noise resilience, and compatibility with standard CMOS design flows, making them suitable for deployment in low-power edge AI systems.

In this thesis, a 16-Kb Digital Compute-In-Memory (DCIM) SRAM macro designed to perform vector-matrix multiplication between a 64-dimensional input vector and a 64×64 weight matrix, with both represented as 4-bit unsigned integers. The macro has a 9T NOR-bitcell that gives bitwise AND operations within the memory array, and an 18T full adder for accumulation of partial products from bitcell array. This architecture has been implemented using the TSMC 65-nm CMOS process, allowing for scalable integration and power-efficient operation.

The macro has 64 banks, each consisting of a 64×4 NOR bitcell array, peripheral logic, and an adder tree for accumulation. To improve the performance of this accumulation phase, three different adder tree topologies are evaluated. The first is a conventional "Standard 6-stage adder tree," which uses ripple-carry adder comprising of 18T Full adders. The second topology, is "Topology-1 6-stage adder tree," employs an interleaved structure of full adders across six stages to reduce the overall delay. In the first stage of the adder tree, 18T full adders are utilized to initiate the accumulation process with minimal latency. The second stage implements a ripple carry adder (RCA) composed of 28T standard cell full adders, providing more drive strength. This pattern of alternating between 18T and 28T standard cell full adders continues through the remaining stages, forming an interleaved architecture that

balances speed and area efficiency. The third topology, is "Topology-2 6-stage adder tree," having 28T standard cell full adders in the first stage. This is done due to the need to drive a large bitcell array in 1st stage of adder tree, where higher drive strength is needed to minimize delay. In subsequent adder tree stages, RCAs are implemented with 18T and 28T standard cell full adders alternatively, finally using 18T full adders in the 6th stage of adder tree.Topology-2 6-stage adder tree results in a slight increase in silicon area due to the 28T full adders being used in large stages of adder tree. Among all adder trees, Topology-2 6-stage adder tree gives optimum trade-off between performance and efficiency. Post-layout simulation results of the DCIM macro using Topology-2 6-stage adder tree has a peak throughput of 396 GOPS under 4b/4b precision, with power efficiency reaching 9.20 TOPS/W. When operating in 4b/1b mode, DIMC macro gives peak throughput of 1980 GOPS and power efficiency of 46.04 TOPS/W. The corresponding power consumption is 43mW.

To validate this macro on a CNN model, the LeNet-5 CNN model was quantized to 4-bit weights and 4-bit activations, matching the data format supported by the macro. The quantized model was trained and evaluated on the MNIST dataset, achieving a classification accuracy of 90.8%, which is comparable to its full-precision counterpart. Overall, the proposed DCIM macro, with the Topology-2 6-stage adder tree, is energy-efficient, provides high-throughput for edge AI hardware accelerator applications.

Contents

List of Figures

List of Tables

List of Abbrevations

CIM	-	Compute In Memory
PE	-	Processing Element
AI	-	Artificial Intelligence
DNN	-	Deep Neural Network
DRAM	-	Dynamic Random Access Memory
SRAM	-	Static Random Access Memory
MAC	-	Multiply and Accumulate
ReRAM	-	Resistive Random Access Memory
CNN	-	Convolution Neural Network
DIMC	-	Digital In-Memory Computing
AIMC	-	Analog In-Memory Computing
ALU	-	Arithmatic and Logical Unit
VMM	-	Vector-Matrix Multiplication
ADC	-	Analog to Digital Converter
DAC	-	Digital to Analog Converter
MOM	-	Metal-Oxide-Metal
PP	-	Partial Product
RCA	-	Ripple Carry Adder
DVFS	-	Dynamic Voltage and Frequency Scaling
PTL	-	Pass Transistor Logic
SGD	-	Stochastic Gradient Descent

Chapter 1

Introduction

1.1 Background and Motivation

As demand for processing large amounts of data continues to increase, traditional computing architectures are not able to produce the required performance. One of the most prominent challenges is the slowing of Moore's Law, which has driven semiconductor advancements for decades by predicting that the number of transistors on integrated circuits doubles approximately every two years. However, as transistors approach the 1nm scale, physical limitations become apparent, such as quantum effects like electron tunneling, increased leakage currents, and difficulties with heat dissipation. These issues, coupled with the escalating costs of building complex fabrication plants, have resulted in less returns in terms of performance gains from further transistor size scaling. As a result, the semiconductor industry is exploring alternatives such as three-dimensional (3D) chip stacking, quantum computing, and new memory technologies to continue pushing computational boundaries beyond the constraints of traditional transistor scaling.

In parallel, the von Neumann bottleneck poses another significant challenge in modern computing. This bottleneck arises from the separation between the CPU and memory in traditional architectures, which creates a mismatch in speed—CPUs can process data much faster than they can fetch it from memory, leading to delays and inefficiencies. As CPUs become increasingly powerful, the speed of memory access

has not kept pace, limiting overall system performance. In DNNs, each neuron engine typically demands several MAC operations, requiring significant computational power and hardware resources [21]. To overcome this, researchers are focusing on memory-centric computing models such as CIM, where computation is integrated directly with memory, reducing data transfer latency.

Key factors contributing to the von Neumann bottleneck are:

- **High Memory Access Delay:**Memory access latency refers to the time taken by CPU to retrieve data from memory. Even though modern CPUs are capable of executing billions of instructions per second, the time it takes to access data from memory can introduce significant delays. These delays are especially noticeable in applications that require frequent access to large datasets, as the CPU must pause its operations while waiting for the required data, slowing down overall processing speed.

- **Fetching Data and Instructions sequentially:** In the von Neumann architecture, both program instructions and data are stored in the same memory space and must share the same bus for transfer. This means that the CPU has to fetch instructions and data one after the other in a sequential manner. As a result, the CPU cannot fetch both instructions and data simultaneously, leading to additional delays. This sequential fetching process further limits the speed of computation and increases the overall processing time.

- **Less Data Bus Bandwidth:** The data bus is responsible for transferring information between the CPU and memory, but it has a fixed speed or bandwidth. As CPUs continue to become faster, the data bus often cannot keep up with the increasing speed of the processor, resulting in delays. This mismatch in speed between the CPU and the data bus creates a significant bottleneck, as the processor spends time waiting for data to be fetched from memory, which limits overall system performance.

- **Limited Cache Memory:** Modern systems use cache memory to temporarily store frequently accessed data closer to the CPU, helping to reduce memory

access time. However, cache memory is limited in size and cannot hold all the data needed for large or complex tasks. When the required data is not found in the cache (a cache miss), the CPU must access the slower main memory, causing delays. As data workloads grow, the limited capacity of cache becomes a more significant issue, contributing to the overall inefficiency of memory access in von Neumann architectures.

Some of the popular techniques to overcome von Neumann bottleneck are as follows:

- **Cache Memory:** Introducing multiple levels of cache between the CPU and main memory helps to narrow the speed gap. Cache memory, being much faster than main memory, temporarily holds frequently used data and instructions, thereby minimizing access latency.

- **In Memory Computing(IMC):** This approach moves computation closer to where data resides by embedding processing elements within memory itself. By reducing the need for data movement between memory and CPU, CIM architectures can significantly lessen the impact of the von Neumann bottleneck.

- **Parallel Computing:** Strategies such as pipelining, and multi-core architectures enable processors to handle multiple instructions at once, effectively reducing delays caused by slow memory access.

- **Prefetching Techniques:** Intelligent prefetching algorithms anticipate future data needs and load data into faster memory levels ahead of time, minimizing idle CPU cycles and improving overall system efficiency.

The von Neumann bottleneck arises from the limited bandwidth and high latency between a system's CPU and its memory, constraining data transfer rates and overall computational efficiency. SRAM plays a critical role in mitigating this issue due to its unique characteristics.

SRAM is significantly faster than DRAM because it does not require periodic refreshing to maintain stored data. This inherent speed advantage allows SRAM to

serve as a high-speed buffer between the processor and the slower main memory. By integrating SRAM-based cache memories (L1, L2, and sometimes L3 caches) within or near the CPU, frequently accessed data and instructions can be stored closer to the processing unit. This reduces the frequency and volume of main memory accesses which is slow, further decreasing latency and increasing throughput. This close integration of SRAM-based caches with the CPU enables faster instruction fetching, quicker data retrieval, and smoother execution pipelines, all of which reduces the impact of the von Neumann bottleneck.

Moreover, emerging CIM architectures are increasingly using SRAM arrays not just for storage but also for performing basic computations directly within the memory itself. SRAM-based CIM architectures eliminate the need to move data between the memory and the processor for certain operations, further reducing data transfer overheads and power consumption.

1.2 Overview of SRAM based Compute-in-Memory

Compute in Memory (CIM) refers to do computation directly within memory units, rather than relying on separate processing units. CIM represents a important shift in computing architecture, aiming to integrate computational capabilities directly within memory units rather than depending exclusively on separate processing cores. CIM reduces costly data movement between memory and compute units by enabling computation within the memory array itself [23].

Fundamentally, CIM focuses on performing computations in close proximity to data storage, thereby exploiting the inherent parallelism and high bandwidth of memory structures. By enabling computational tasks to be executed within the memory cells themselves, CIM architectures can significantly accelerate data processing, particularly for applications involving large-scale datasets and complex algorithms.

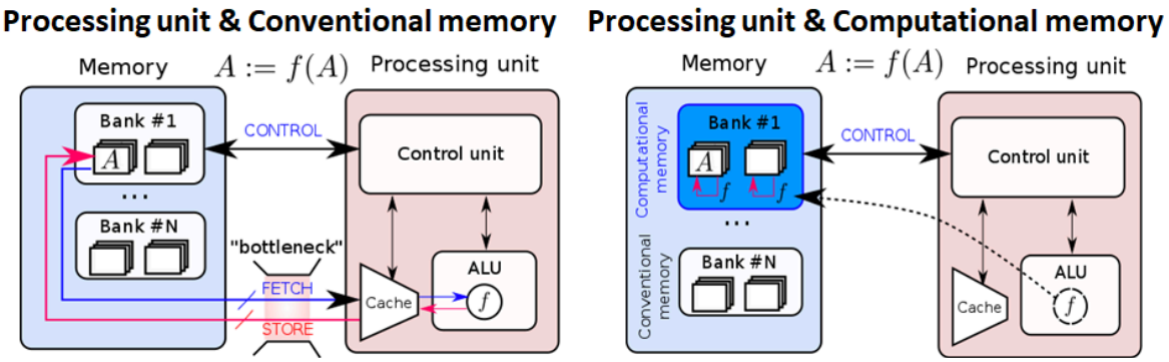

Figure 1.1: Processing unit and Conventional memory Vs Processing unit and Computational memory [1]

As depicted in Figure 1.1, in-memory computing eliminates the necessity of transferring data to external processing units. Instead, computation is achieved by leveraging the physical properties of memory devices, their array-based organization, associated peripheral circuitry, and integrated control logic. Digital near-memory compute architecture achieves a higher memory density while sacrificing performance metrics, such as throughput and computing latency, compared to the CIM [19]. But digital near-memory compute architecture has more area due to separate computing logic near memory block.

SRAM-based CIM emerges as a highly practical and efficient implementation of this concept, particularly suited for energy-constrained, latency-sensitive applications such as AI edge devices and AIoT systems. In SRAM-CIM architectures, standard SRAM arrays are modified to not only store data but also execute computational operations directly within the memory itself, leveraging the same principles of minimizing data movement and exploiting massive parallelism.

An IMC memory can be used in two modes: memory mode and computation mode. In memory mode, read or write operation is performed on an addressed word. In computing mode, the memory computes data over many addressed words [20].

SRAM, with its fast access times, high endurance, and compatibility with CMOS logic, can be used for designing CIM. By integrating computation within the SRAM arrays, operations such as logic functions, addition, and multiply-accumulate (MAC)

operations can be executed without the need to move data to external processing elements. This reduces memory access energy and latency compared to traditional von Neumann architectures, where frequent data transfer between memory and processors forms a performance bottleneck.

As explained earlier, conventional AI accelerators suffer from increased energy and latency overheads due to weight and activation data movement between on-chip and off-chip memories. SRAM-CIM directly addresses these challenges by embedding computational capabilities within the memory macros.

Unlike non-volatile memory-based CIM approaches, which face limitations in endurance and write energy, SRAM-CIM provides high write speeds, unlimited endurance, and easier integration with existing digital logic technologies. This makes SRAM-CIM highly suitable for small to medium capacity AI models that require frequent weight updates, configurability, and rapid inference at the edge.

Moreover, SRAM-CIM inherits the advantages of massive parallelism inherent to dense memory arrays, thereby improving the time complexity of computationally intensive tasks. By leveraging the parallel nature of millions of nanoscale SRAM cells functioning as compute units, SRAM-CIM enables high-throughput, low-latency computation, reinforcing its potential to redefine memory-centric computing architectures.

Thus, SRAM-based CIM represents a practical and scalable realization of the broader CIM vision, offering a path toward highly efficient, low-latency, and memory-centric AI processing in next-generation intelligent devices.

1.3 Motivation

The primary objective of this thesis is to design a Efficient In Memory Computing architecture based on SRAM for Edge AI and CNN applications. The main goals of this study are as follows:

- Designing a SRAM-based Efficient In Memory Computing architecture, with a

focus on improving speed, throughput,and area efficiency compared to conventional CIM designs.

- Developing efficient circuit designs to perform computation within SRAM memory bitcells, leveraging the speed of SRAM cell.
- Implementing a SRAM-based in-memory computing architecture and performing detailed performance analysis.
- Exploring various topology of Adder tree in order to enhance performance of CIM Architecture.
- Estimating accuracy of CNN model, LeNet-5 in reference to the data format strategy of the designed architecture.

1.4 Organization of Thesis

This thesis is organized into several chapters, each focusing on different aspects of the research conducted on SRAM based Compute-in-Memory architectures. The chapters are structured as follows:

Chapter 1: This chapter outlines the background and motivation behind the research, emphasizing the growing demands of data-intensive applications and the need for high-speed, energy-efficient solutions. It introduces the use of SRAM as a memory cell for computing to enable faster operations and higher throughput. The chapter concludes by presenting the thesis objectives and a brief overview of the thesis structure.

Chapter 2: This chpter provides a thorough overview of In-Memory Computing (IMC), covering existing IMC implementations and various types of IMC. This chapter lays the foundation for the contributions of this thesis by highlighting the limitations of current architectures and exploring the potential advancements enabled by SRAM-based In-Memory Computing Architecture.

Chapter 3: This chapter focuses on the design and implementation of an In-Memory Computing (DIMC) architecture using a novel SRAM-based bitcell. It

begins by introducing the proposed 9T NOR bitcell and its circuit analysis, followed by a discussion of the proposed 18T full adder and its detailed analysis. The chapter then outlines the architecture of the Digital In-Memory Compute system, including the pipeline structure and timing diagram for the Multiply-Accumulate (MAC) operation. Additionally, it covers the various subcircuits that make up the DIMC architecture and presents different topologies for the adder tree, including a standard 6-stage adder tree and two interleaved topologies.

Chapter 4: This chapter presents a comprehensive performance evaluation of the proposed CIM (Compute-in-Memory) macro, including pre- and post-layout simulations. It covers the layout design of a 64x4 memory bank and assesses the throughput, energy efficiency, and area efficiency of the CIM macro. A comparison with existing works highlights the advantages and potential improvements of the proposed CIM design. Additionally, the chapter includes an evaluation of the LeNet-5 CNN model's accuracy, providing insight into the model's performance within the context of the proposed architecture.

Chapter 5: The final chapter provides a summary of the key findings and contributions of the thesis, focusing on the implications of the 9T NOR Bitcell and various adder tree topologies for the future development of In-Memory Computing (IMC) architectures. It examines how these designs could shape the performance, efficiency, and scalability of IMC systems. Additionally, the chapter suggests potential directions for future research, aiming to further improve the performance and broader applicability of SRAM-based IMC architectures.

Chapter 2

In Memory Computing: Basics and Working

2.1 Overview to In Memory Computing

In-Memory Computing (IMC), also known as Computing-in-Memory (CIM), represents a rapidly emerging computational method that to overcome the inherent limitations of the traditional von Neumann architecture. In conventional computing systems, there is separation between memory units and processing elements (PEs). Due to this, frequent and energy-consuming data transfers between memory and processors happen which results in high latency and high energy consumption. These drawbacks become especially pronounced in applications such as artificial intelligence (AI) and the Internet of Things (IoT), where the requirements for real-time data processing, low power operation, and efficient energy usage are needed.

IMC addresses this challenge by enabling data processing to occur directly within the memory arrays themselves, thereby minimizing or even eliminating the need for constant data shuttling between memory and computational units. By merging memory and compute, IMC architectures are able to significantly reduce both data movement and the associated energy overhead.

In IMC fundamental computational operations such as basic logic functions, bitwise multiplication, and accumulation are performed inside memory bitcells. These

computations leverage properties of memory devices to perform parallel operations across entire arrays. IMC architecture brings about good improvements in both computational speed and energy efficiency. As a result, IMC is gaining significant attention for its potential in low-power, high-performance applications, particularly in edge computing and Artificial Intelligence of Things (AIoT) environments, where traditional data-centric processing architectures fall short.

The primary motivation behind In-Memory Computing (IMC) is to address the fundamental energy and performance limitations inherent in the traditional von Neumann architecture. In conventional computing systems, memory and processing units are physically separated, which leads to frequent and energy-intensive data transfers across the memory-processor interface [15]. This separation becomes a significant bottleneck, especially in applications requiring high throughput and low power, such as edge artificial intelligence (AI) and Internet of Things (IoT) devices.

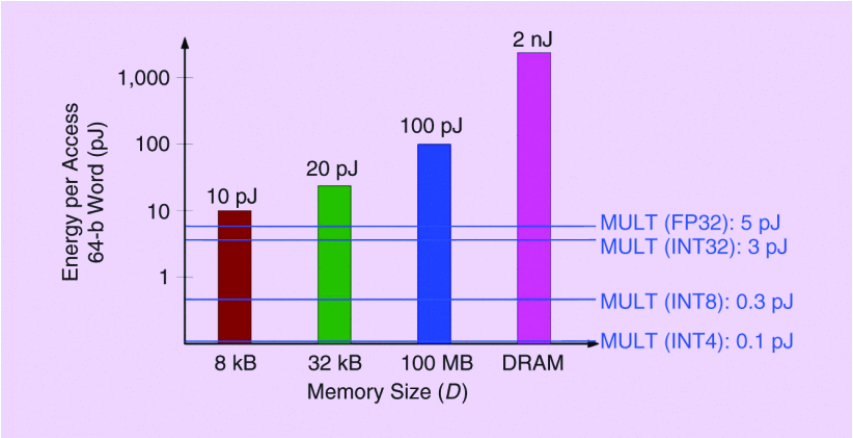

Figure 2.1: Comparison of Energy Consumption: Computation vs Data Movement. [2]

This figure 2.1 shows how energy per access increases with memory size, from on-chip SRAM (8 kB) to off-chip DRAM. Smaller memories consume less energy (10–100 pJ), while DRAM access consumes up to 2 nJ. Overlaying compute energy shows that even FP32 operations (5 pJ) are cheaper than accessing larger memory, emphasizing the energy cost of data movement. Thus indicating the need of efficient computing architecture like IMC architecture.

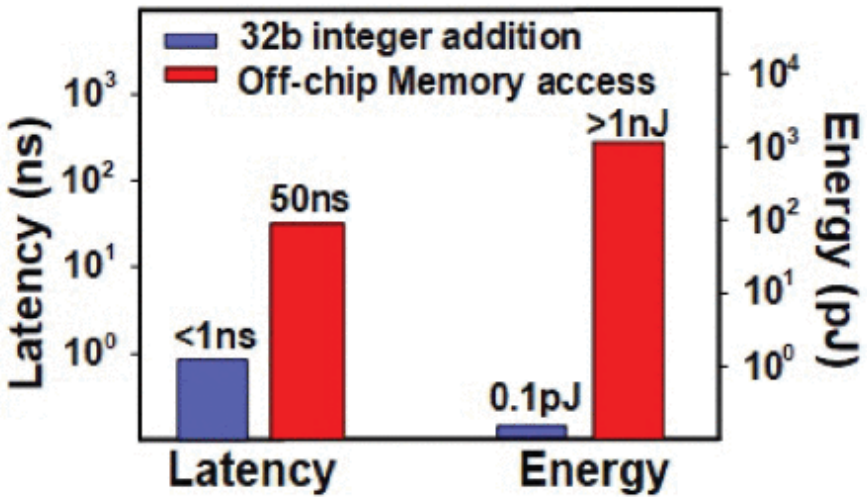

Figure 2.2: Energy & Latency comparison 32 bit ALU operation and off-chip access. [3]

Figure 2.2 compares a 32-bit integer addition with an off-chip memory access. The integer addition exhibits extremely low latency (1 ns) and energy consumption (0.1 pJ), whereas off-chip memory access suffers from much higher latency (50 ns) and energy (1 nJ). As per the conclusion drawn from above figures 2.2, 2.1 In-Memory Computing (IMC) is needed in place of Von-Neumann architecture as it enables computation directly within memory, thereby significantly reducing data movement, lowering energy consumption, and improving performance for data-intensive tasks.

Among the various memory technologies explored for implementing IMC, SRAM has emerged as a particularly attractive option. SRAM-based IMC designs benefit from the technology's low access latency, high write endurance, and compatibility with standard CMOS, making it ideal for on-chip integration in modern digital systems. These architectures are designed to meet design goals, including improved energy efficiency, reduced computational latency, minimized silicon area, enhanced computational precision under varying operating conditions.

By executing operations directly within the memory array, SRAM-based IMC significantly reduces energy consumption compared to traditional compute architectures. This efficiency is crucial for battery-powered and energy-constrained applications.

Additionally, by integrating computation and memory, IMC eliminates the delays associated with external memory access, resulting in lower overall latency and faster data processing. Compactness is also a major advantage, as these architectures reduce the need for large ALU and dedicated data movement circuitry, which is particularly valuable for area-constrained edge devices.

Moreover, SRAM-based digital IMC provides greater robustness and precision compared to analog counterparts. By utilizing bit-serial or bit-parallel computation techniques within or near the memory, these systems offer deterministic behavior and can maintain sufficient numerical accuracy for many AI inference tasks. Precision can be further optimized through quantization-aware training and hardware-aware model design, allowing designers to strike a balance between computational fidelity and resource efficiency.

SRAM-based IMC designs are inherently more resilient to environmental and process variations due to the maturity of CMOS fabrication processes. Nevertheless, designers must still consider challenges such as device mismatch, noise susceptibility, and operational variability when scaling such architectures. As a result, SRAM-based IMC continues to be a focal point of research and development, particularly for energy-efficient, high-performance AI processing at the edge.

Earlier, In-Memory Computing (IMC) was primarily applied to search-oriented tasks such as routing table lookups, pattern matching, and database query acceleration. These applications relied heavily on content-addressable memory (CAM), particularly binary CAM (BCAM) and ternary CAM (TCAM), which enabled high-speed associative search operations with minimal latency [17]. These early IMC systems leveraged the parallel comparison capabilities of CAM to rapidly identify matches, making them highly effective in networking and database environments.

Edge devices benefit from IMC's ability to tailor bit-precision, operations, and architecture on demand, making it suitable for power-sensitive AI applications [14].Thus, as the demand for machine learning and artificial intelligence (AI) workloads has grown, the focus of IMC has shifted significantly from simple search tasks to more compute-intensive operations, particularly Multiply-Accumulate (MAC) computa-

tions, which form the backbone of deep neural network (DNN) inference. This evolution has driven the development of IMC architectures capable of supporting arithmetic operations directly within the memory. In particular, SRAM-based IMC accelerators have emerged as a promising solution, enabling low-precision (e.g., 4-bit or 8-bit) arithmetic operations using either purely digital logic or hybrid mixed-signal techniques.

In digital SRAM-based IMC architectures, MAC operations are typically carried out using bitwise computation schemes combined with accumulation logic placed either adjacent to or integrated within the memory array. A common implementation involves streaming activation bits serially into the array while pre-storing the corresponding weights within the SRAM cells. Bitwise logic operations—such as AND or XNOR—are used to compute partial products, which are then accumulated over multiple cycles to produce the final output. This design approach is particularly advantageous for edge computing applications, as it offers a favorable trade-off between energy efficiency, silicon area, and computational throughput. Additionally, it aligns well with QNN models, which are designed to operate effectively using reduced-bit-width data, thereby minimizing both memory and compute requirements.

2.1.1 Different types of In Memory Compute Architectures

The increasing demand for low-power and high-throughput computation in artificial intelligence (AI) applications has led to growing interest in emerging architectures that overcome the limitations of the traditional von Neumann paradigm. In such conventional systems, the separation of memory and processing units introduces significant energy and latency penalties due to continuous data transfer. In-Memory Computing (IMC) addresses this inefficiency by embedding computation directly within memory arrays, thus significantly reducing data movement and enabling parallel processing at the memory level. Among the available memory technologies, Static Random-Access Memory (SRAM) has emerged as a particularly favorable option for implementing IMC due to its high speed, mature CMOS compatibility, and robust read-write endurance.

Two major types of IMC architectures have been proposed and widely studied: analog in-memory computing and digital in-memory computing. While both aim to reduce energy consumption and computational latency by localizing computation within memory arrays, they differ substantially in their mechanisms, levels of precision, design complexity, and application suitability.Analog IMC performs MAC operations via charge accumulation or current summation, while digital IMC leverages bit-serial logic for better precision and reliability [**?**]. In both cases, SRAM is often used to store synaptic weights, especially in AI accelerators performing inference tasks. This is because SRAM can support high-speed access and is amenable to both logic and analog extensions through custom circuit designs.

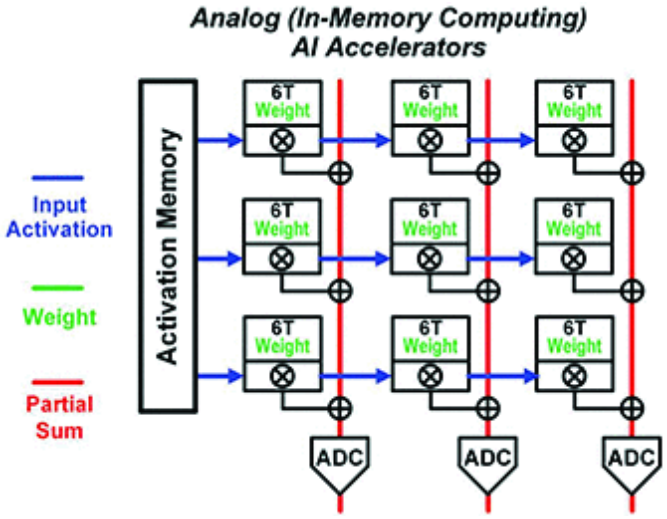

Figure 2.3: Conventional AIMC Architecture. [4]

As shown in figure 2.3 IMC leverages the intrinsic electrical properties of circuit components to perform operations such as MVM directly along the memory's bitlines. Input activations are applied as voltage or current signals on the wordlines, while the memory cells modulate these signals according to the stored weights. The aggregated current or voltage on the bitlines effectively represents the result of a MAC operation. These analog signals are then digitized by ADCs for further processing. This technique is inherently parallel, as multiple rows in the memory can be activated simultaneously, enabling large-scale, energy-efficient MVM execution [18]. As a

result, analog IMC is particularly attractive for applications involving low-precision inference, such as edge AI and ultra-low-power devices.

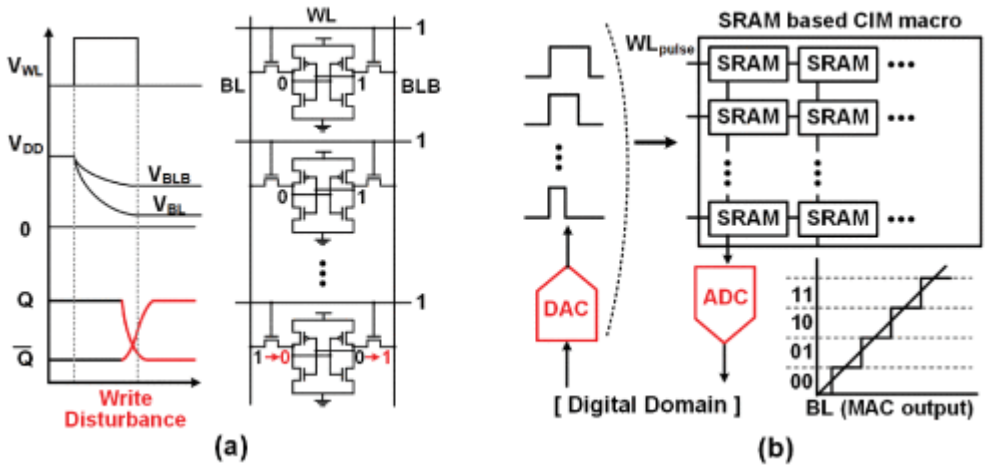

Figure 2.4: (a) Write Disturbance (b) DAC and ADC overhead in AIMC. [5]

As depicted in this figure 2.4 AIMC faces two major challenges: write disturbance as shown in 2.4(a) and ADC/DAC overhead as shown in 2.4(b). Write disturbance occurs when writing data to one SRAM cell unintentionally affects neighboring cells due to shared voltage lines, potentially leading to data corruption. This is caused by voltage coupling on bitlines and wordlines, which can disturb the state of adjacent memory cells. Another significant issue is the overhead introduced by digital-to-analog (DAC) and analog-to-digital (ADC) conversions. Since data is typically stored and processed in the digital domain, converting it to analog for computation and then back to digital for output adds latency, consumes extra energy, and occupies additional chip area.

In contrast, digital IMC shown in figure 2.5 executes computation using standard binary logic operations embedded within or near the memory array. Rather than working with continuous voltage levels, digital IMC relies on discrete logic states (0s and 1s), which are more resilient to process and environmental variations. In SRAM-based digital IMC, operations such as bitwise AND, OR, XOR, and addition are implemented using either specially designed bitcells or peripheral circuits. One approach is to use bit-serial processing, where multi-bit MAC operations are broken

into binary sub-operations and carried out sequentially. Alternatively, designs based on 8T, 10T, or 12T SRAM cells enable enhanced logic capabilities by activating multiple wordlines and bitlines simultaneously.

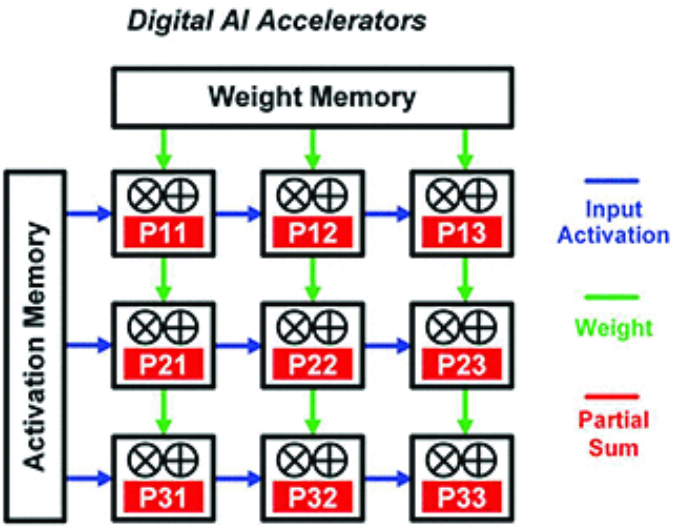

Figure 2.5: Conventional DIMC Architecture. [4]

The major strength of digital IMC lies in its precision and reliability. Because computations are performed using robust digital logic, it is easier to achieve higher accuracy and consistency across varying operating conditions. Moreover, digital IMC circuits integrate well with existing CMOS toolchains and verification methodologies, making them suitable for large-scale system design and fabrication. These systems can also be reconfigured to handle diverse AI operations, such as logic processing, quantized arithmetic, or even customized neural operations with programmable control.

Digital computations generally involve more transistor switching activity than analog, leading to increased energy consumption per operation. Additionally, while analog IMC can exploit inherent parallelism at the array level, digital IMC often requires serialization or time-multiplexed execution to maintain precision, which can reduce effective throughput. The integration of additional logic into the memory array can also increase cell size and limit the density advantages otherwise offered by SRAM bitcell.

Ultimately, both analog and digital SRAM-based IMC systems have benefits depending on the application. AIMC is well-suited for applications where extreme energy efficiency and area compactness are paramount and where modest precision is acceptable. DIMC, on the other hand, provides higher accuracy and robustness, making it preferable in scenarios where precision and digital compatibility are critical. As the field advances, there is growing interest in hybrid IMC architectures that aim to combine the high energy efficiency of analog processing with the precision and programmability of digital designs. These hybrid solutions promise to deliver flexible and scalable accelerators capable of efficiently supporting a wide range of AI workloads in energy-constrained environments.

2.1.2 Existing In Memory Computing Architectures

The growing demand for energy-efficient processing in edge-AI applications, particularly those involving convolutional neural networks (CNNs), has led to the proliferation of SRAM-based Computation-in-Memory (CIM) architectures. One significant advancement is a 4-Kb configurable CIM macro built using 6T SRAM, capable of operating across 1- to 8-bit precision levels. This design employs a hybrid computational approach that integrates in-memory binary product-sum (IMBPS) operations with near-memory multibit accumulation. Additionally, it introduces a self-referenced multilevel sensing scheme and input-aware bitline voltage compensation to enhance the read margin, energy efficiency, and reliability of operations. The architecture, fabricated in 55nm CMOS, demonstrates up to 40.2 TOPS/W energy efficiency across a broad input-weight precision range, maintaining robustness even under high-parallelism configurations. [9]

A different yet complementary approach focuses on charge-domain analog computing within standard 6T SRAM arrays. This architecture introduces a precision-programmable multiply-and-accumulate (MAC) operation, implemented using metal-oxide-metal (MOM) capacitors, allowing analog MACs to be accumulated via charge sharing. Supporting bit-serial computation, the design accommodates up to six levels of weight and eight levels of activation precision. A 7-bit charge-injection SAR ADC

is employed to improve linearity and avoid complex sample-and-hold mechanisms. The system achieves a peak efficiency of 49.4 TOPS/W and delivers accurate inference for CNNs trained on datasets like MNIST and CIFAR-10, confirming its relevance to real-world edge applications. [8]

Efforts have also been made to extend the scope of CIM to more general-purpose computing. A signed-input CIM macro, based on a true dual-port 8T SRAM bitcell, enables in-memory multiply-and-accumulate operations with arbitrary signed operands. This architecture supports both neural and signal processing tasks by allowing simultaneous computation with positive and negative inputs within the same cycle. It handles signed weights and activations, offering bit-level configurability and concurrent read/write operations, thus improving throughput and integration ease. With up to 41 TOPS/W energy efficiency, this architecture supports diverse workloads such as frequency-domain voice activity detection and CNN inference, all within the same hardware. [7]

In contrast to analog or hybrid schemes, another approach features a purely digital, programmable processing-in-SRAM (PSRAM) macro, operating at 1.23 GHz. Built on a 16-kb 8T SRAM array, it implements a wide set of Boolean and arithmetic functions such as XOR, majority logic, and full addition, all executed in a single memory cycle. The reconfigurable sense amplifier structure allows simultaneous realization of multiple logic functions, eliminating the need for multi-cycle data transfers. This system demonstrates high-speed and energy-efficient operation (34.98 TOPS/W), making it suitable for a range of applications including bitwise vector processing, deep learning inference, and cryptographic tasks like AES. [6]

Despite the considerable progress in SRAM-based CIM solutions, several critical limitations remain unaddressed. While certain architectures provide signed MAC functionality using dual-port SRAM cells [7], they typically target mid-to-high precision levels and struggle to efficiently scale to 1–4-bit regimes, which are needed for quantized CNN models. Moreover, analog and hybrid CIM designs suffer from conversion bottlenecks due to DAC/ADC blocks and limited scalability in highly parallel settings. These gaps point to the need for a fully digital, low-overhead,

and bit-precise CIM architecture that can natively support signed computations at ultra-low precision, while maintaining configurability, energy efficiency, and robustness—especially under the constrained power and area budgets of edge devices. **The main gaps identified in existing CIM architectures are as follows:**

- Existing solutions have lower throughput (GOPS), limiting real-time performance in edge-AI applications.
- Power efficiency remains suboptimal due to ADC/DAC overheads and inefficient support for low-precision CNN models.
- Area efficiency is limited by the use of complex analog/mixed-signal circuitry and larger SRAM bitcells.

2.1.3 Introduction to Convolution Neural Network

Convolutional Neural Networks (CNN) are a specialized type of deep learning model designed primarily for processing data, such as images. Unlike traditional fully connected neural networks, CNNs use spatial hierarchies within the data, making them effective for applications like image classification, object detection, and facial recognition. Their design is inspired by the way the human visual system processes information, focusing on extracting local patterns in images, such as edges, textures, and shapes, and gradually building more complex representations.CNNs consists of multiple convolution layer with additional pooling, normalization and fully connected layers mixed in [22]. Different from traditional neural networks where the relationship between input and output units is derived through matrix multiplication, convolutional neural networks (CNNs) reduce computational burden using sparse interactions, where smaller kernels are applied across the entire image [13].

Key Components of CNNs are as follows:

- The convolutional layer as shown in figure 2.6 applies a convolution operation to the input image (or the output of a previous layer), which involves sliding a small filter (also called a kernel) over the image and performing element-wise

multiplication followed by summing the results. This operation detects specific features such as edges, corners, or textures at different spatial locations in the image. The output of the convolution operation is called a feature map, which gives the presence of the detected features in the image. Multiple filters are typically used in a convolutional layer, each detecting a different feature.

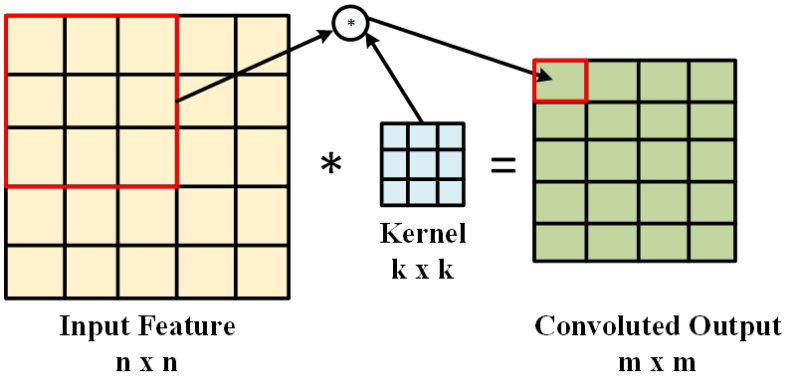

Figure 2.6: 2D Convolution Layer in CNN. [12]

- After the convolution operation, the output is passed through a non-linear activation function, typically the Rectified Linear Unit (ReLU). The ReLU function introduces non-linearity into the model, allowing CNNs to learn more complex patterns. ReLU operates by zeroing out all negative values in the feature map and passing the positive values unchanged, which helps maintain a sparse representation.

- The pooling layer is to reduce the spatial dimensions of the feature maps while retaining important features. Pooling operations, such as max pooling or average pooling, operate on small regions of the feature map and select the most important value (for max pooling) or average the values (for average pooling).

- After several convolutional and pooling layers, the CNN typically ends with one or more fully connected layers. These layers flatten the multi-dimensional feature maps into a one-dimensional vector, which is then passed through

a series of neurons connected to every neuron in the next layer. The fully connected layer allows the network to make final predictions based on the features extracted earlier in the network. This layer typically ends with a softmax activation function for classification tasks, which converts the output into probabilities that sum to 1.

- In classification tasks, the output layer produces a vector of class probabilities, with each value corresponding to the likelihood of the input image belonging to a particular class. For binary classification, a sigmoid function might be used, while for multi-class classification, the softmax function is commonly applied.

Figure 2.7: Images of Handwritten digit in MNIST Database. [11]

Training a CNN follows a similar process to that of other neural networks, involving forward propagation, loss calculation, and backpropagation to update the weights of the network. The CNN is trained using labeled datasets, where the network learns to minimize the difference between its predicted output and the true label through the loss function. Typically, the cross-entropy loss function is used for classification tasks.

During forward propagation, the input image passes through each layer of the network, and the output is computed. The loss is then calculated based on the error

between the predicted output and the ground truth. During backpropagation, the gradient of the loss with respect to the weights is computed using the chain rule, and the weights are updated through an optimization algorithm, such as stochastic gradient descent (SGD) or Adam.

The MNIST dataset as depicted in figure 2.7 is one of the most widely used datasets for training and testing machine learning models, particularly for image recognition tasks. The dataset consists of 70,000 images of handwritten digits, each of size 28x28 pixels. These digits range from 0 to 9, and the images are grayscale, making it a relatively simple dataset compared to more complex datasets such as ImageNet.

The MNIST dataset is split into two parts: 60,000 images are used for training the model, while the remaining 10,000 images are reserved for testing the model. The task is to classify each image into one of the 10 classes (digits 0-9).The MNIST dataset is commonly used to evaluate the performance of image classification models, and it serves as an excellent benchmark for CNN models in real-world applications.

Chapter 3

Proposed In-Memory Computing Architecture

In this chapter design methodology of the proposed DIMC optimized for bit-serial MAC operations is presented. It begins with the design and analysis of the 9T NOR-based bitcell, then 18T full adder circuit is discussed. The architecture of the digital IMC macro is then discussed in detail, including its pipelined dataflow structure and timing diagram of MAC operations. For performing efficient accumulation of PP, 6-stage adder tree topologies are explored. Among these, two interleaving adder tree topologies are proposed for optimum trade-off between delay, area, and power. This chapter concludes with the selection of the optimized topology-2 6-stage adder tree for post-layout implementation based on performance and energy,area efficiency.

3.1 Proposed 9T NOR Bitcell

9T NOR bitcell is proposed in this DIMC Macro, which performs logic operations within the memory array. Based on the conventional 6T SRAM cell, the 9T NOR Bitcell incorporates three additional transistors that allow it to perform Nor logic operation between an external input bit and the stored weight bit.

The 6T SRAM cell provides data storage functionality, holding the weight bit in inverted form at node QB. The additional three transistors —two NMOS and

one PMOS—form the logic layer that operates on this stored bit and an external input ($\overline{\text{IN}}$). The functional behavior of the cell is based on the Boolean logic: $\overline{\overline{\text{IN}} + \overline{\text{Q}}} = \text{IN} \cdot \text{Q}$

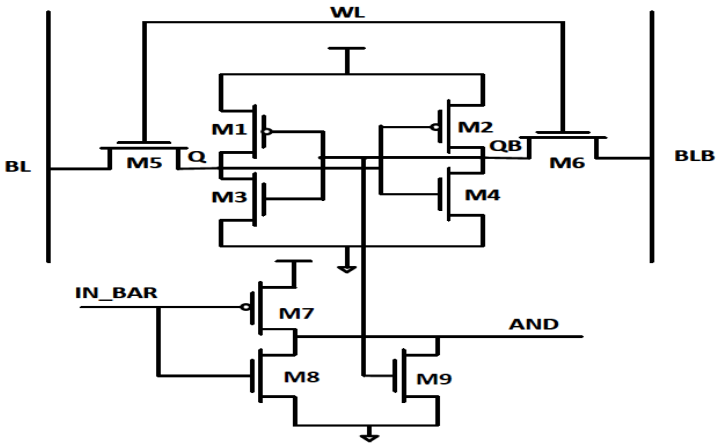

Figure 3.1: Schematic of Proposed 9T Nor-Bitcell.

9T Nor Bitcell allows the NOR operation of the inverted input and inverted weight (Q) to emulate a bitwise AND operation between the true values of input and weight. Thus, a single row of these NOR bitcells is capable of executing the bitwise multiplication of a 1-bit input vector with stored multi-bit weights, generating partial products that can then be accumulated by adder tree.

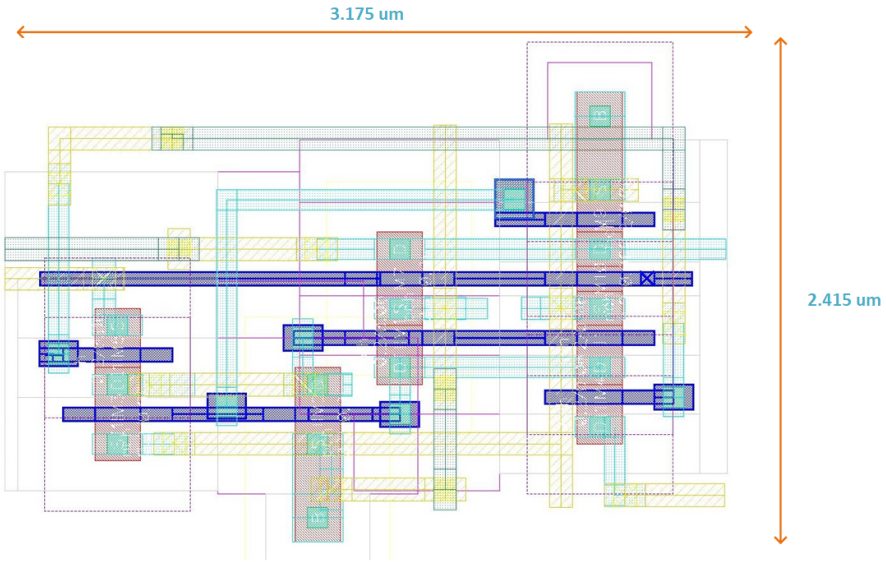

Figure 3.2: Layout of Proposed 9T Nor Bitcell.

Figure 3.2 shows the layout of the 9T NOR Bitcell having dimensions of $3.17\mu m$ in width and $2.41\mu m$ in height. Consequently, the area occupied by this single bitcell can be calculated as the product of its width and height, having total area of $(3.17\mu m) \times (2.41\mu m) = 7.67\mu m^2$.

Advantages Over Conventional 6T SRAM Cell:

- **Parallel Multiply Support:** Through row-level operation, the 9T NOR cells can perform simultaneous multiplications for the entire input vector, contributing to high parallelism and throughput in vector-matrix multiplications (VMMs).

- **Energy Efficiency:** By eliminating the need to transfer data between memory and compute units, the 9T NOR bitcell significantly reduces power consumption which is an essential feature for edge AI applications.

- **Edge AI Suitability:** Computation within memory aligns well with the low-power, low-latency requirements of applications such as image classification, object detection.

- **IMC Capability:** Unlike 6T SRAM Bitcell, which only stores data and requires separate compute logic, the 9T NOR bitcell also performs Nor operation, reducing data movement and enabling compute parallelism within memory array.

3.1.1 Circuit Analysis of 9T NOR Bitcell

Figure 3.1 refers to the Bitcell circuit where the NMOS (M8), PMOS (M7) and NMOS (M9) help to evaluate the NOR logic. When QB = 1 (i.e., Q is '0') and IN = 1 (i.e., $\overline{IN}$ is '0'), both M9 (NMOS) and M7 (PMOS) turn ON, creating a contention between pull-up and pull-down paths. Due to default transistor sizing and higher NMOS mobility compared to PMOS, the pull-down path dominates, resulting in the NOR node being pulled down. However, the node does not reach full logic '0' due to the contention, leading to non-full-swing outputs.

This incomplete logic swing presents a challenge in downstream logic stages, but is effectively compensated by the robust 18T full adder used in the adder tree, which tolerates degraded input levels and restores them during accumulation. This co-optimization between memory cell and arithmetic block ensures reliable end-to-end processing.

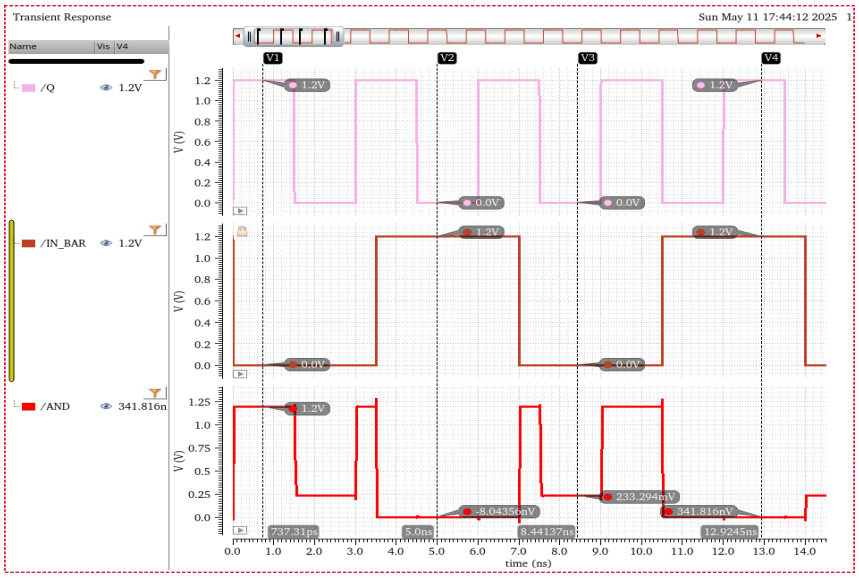

Figure 3.3: Transient simulation of 9T Nor-Bitcell.

The figure 3.3 shows the transient simulation results of 18T full adder circuit. Q and $\overline{IN}$ are toggled to verify all possible input combinations. Outputs AND transition correctly for all input combinations, verifying 9T Nor-Bitcell functionality.

Parameter	**Metric**	**Value**
Delay	T_{rise}	10.67 ps
	T_{fall}	27.53 ps
Average Power	-	20.39 µW

Table 3.1: 9T Nor Bitcell performance metrics @ 1.2V

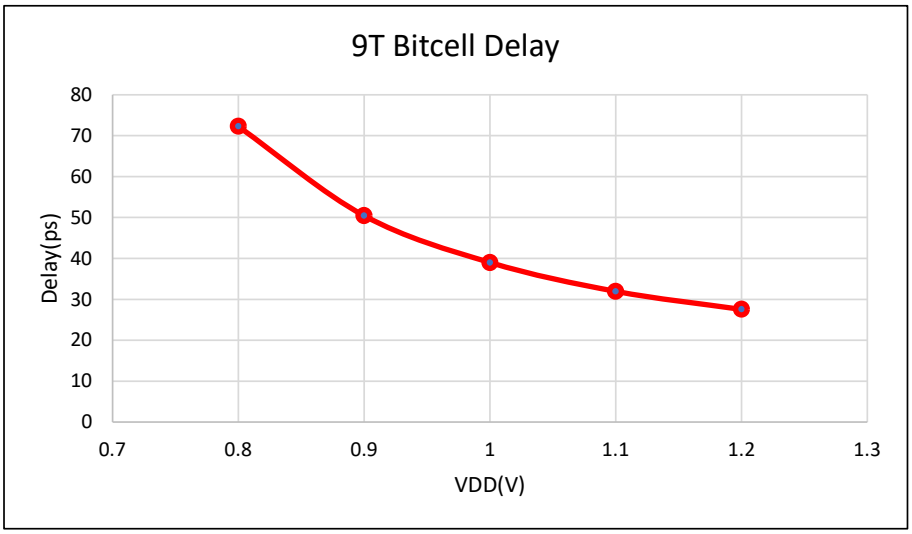

Figure 3.4: Variation in 9T Nor-Bitcell delay with respect to VDD.

As per the figure 3.4 delay of the 9T NOR bitcell decreases as the supply voltage (VDD) increases. As VDD is varied from 0.8 V to 1.2 V, the delay reduces from 72.26 ps to 27.54 ps. This behavior is due to the increased drive strength of the transistors at higher VDD, which allows faster switching. The reduction in delay with increasing voltage indicates improved performance, but the trade-off is the increased power consumption associated with higher VDD.

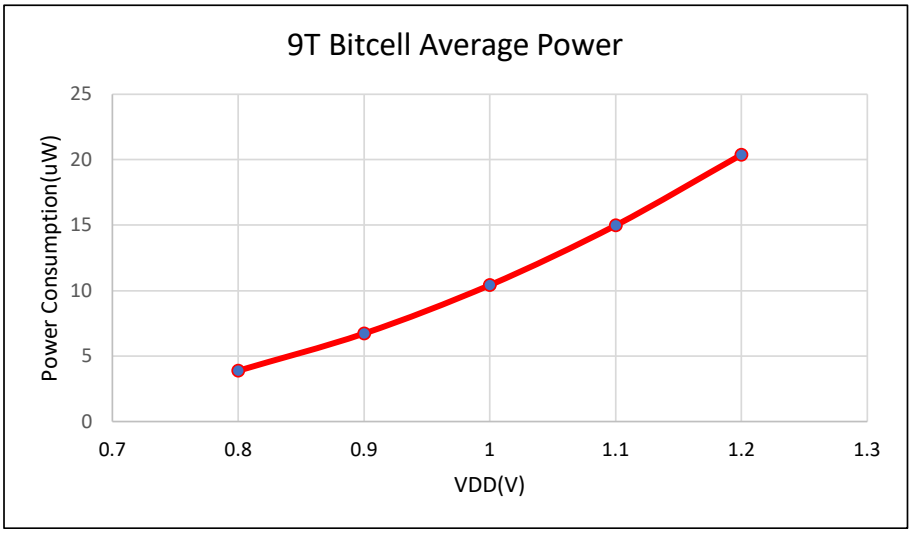

Figure 3.5: Variation in 9T Nor-Bitcell power consumption with respect to VDD.

Figure 3.5 power consumption of the 9T NOR bitcell increases significantly as

the supply voltage (VDD) rises. The power varies from 3.88 μW at 0.8 V to 20.39 μW at 1.2 V. This increase in power is primarily due to the higher dynamic power dissipation as a result of faster switching and higher capacitance charging at higher VDD.

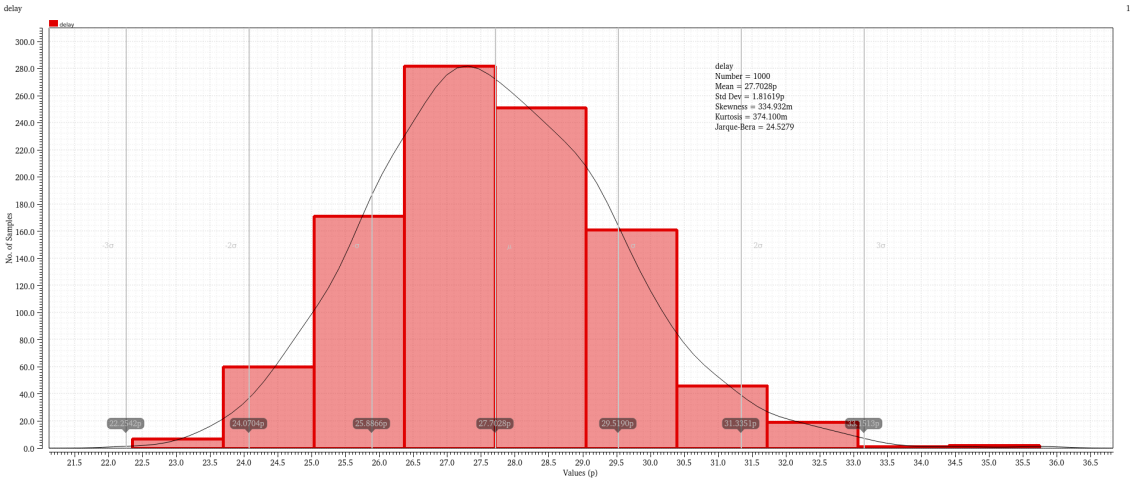

Figure 3.6: Monte-Carlo simulation of 9T Nor-Bitcell.

The Monte Carlo simulation shown in Figure 3.6 analyzes how the delay of the 9T NOR Bitcell changes due to process variations, using 1000 different samples at a supply voltage of 1.2V. The histogram indicates a concentration of delay values around the lower end of the range, with a gradual spread towards higher delays, leading to a positively skewed distribution. On average, the delay is 27.7ps, with a standard deviation of 1.81ps.

3.2 Proposed 18T Full Adder

18T Full Adder is proposed for efficient design of adder tree for accumulating the partial product coming from 9T Nor Bitcell array.Pass-transistor logic (PTL) is used for implementing XOR and XNOR gates, which are essential for generating the Sum output efficiently. The Full Adder implements the standard arithmetic logic:

$$\mathbf{Sum = A \oplus B \oplus Cin}$$

$$\mathbf{Carry = A{\cdot}B + Cin{\cdot}(A \oplus B)}$$

These expressions are realized using cascaded logic stages where PTL optimizes the number of transistors used in the XOR/XNOR stages, and CMOS logic ensures output signal integrity. This approach keeps the design compact while ensuring optimum tradeoff with performance. 18T full adder achieves a balance between drive strength, compact area, and energy efficiency.

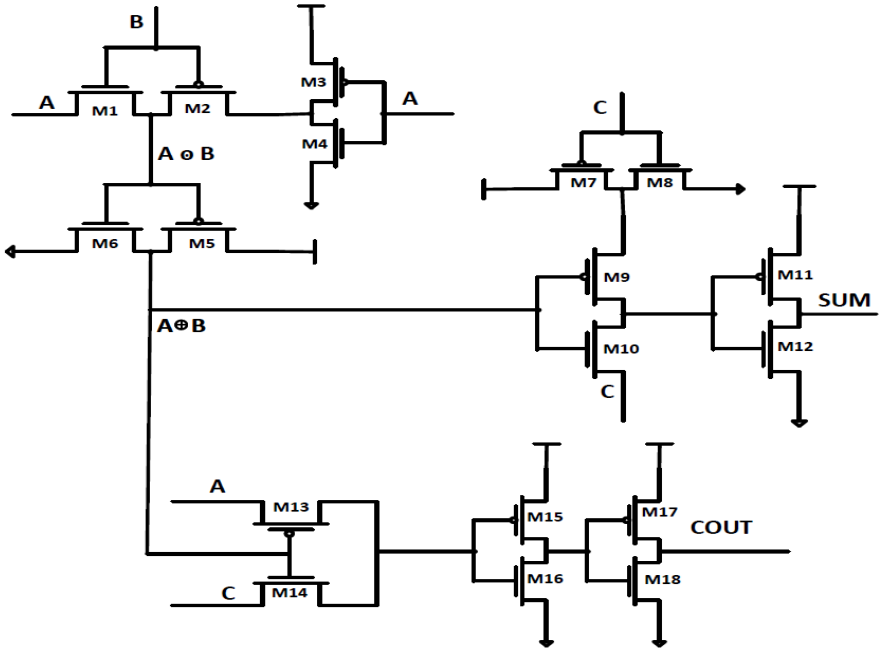

Figure 3.7: Schematic of Proposed 18T Full Adder circuit.

Unlike traditional Full Adders such as the 28T standard cell Full Adder, which offers robust functionality but occupies significant silicon area, the 18T design significantly reduces the transistor count while maintaining correct arithmetic behavior. This makes it highly suitable for hardware accelerators where area and power efficiency is required.

Comparison with the 28T standard cell Full Adder:

- **Area Efficiency:** With 10 fewer transistors than the 28T standard cell Full Adder, the 18T design occupies significantly less silicon area, enabling the integration of more arithmetic units in the same chip footprint.
- **Balanced Trade-Off:**The 18T Full Adder strikes an ideal trade-off between

performance, power, and area, unlike the 28T standard cell Full Adder, which is robust but area- and energy-intensive.

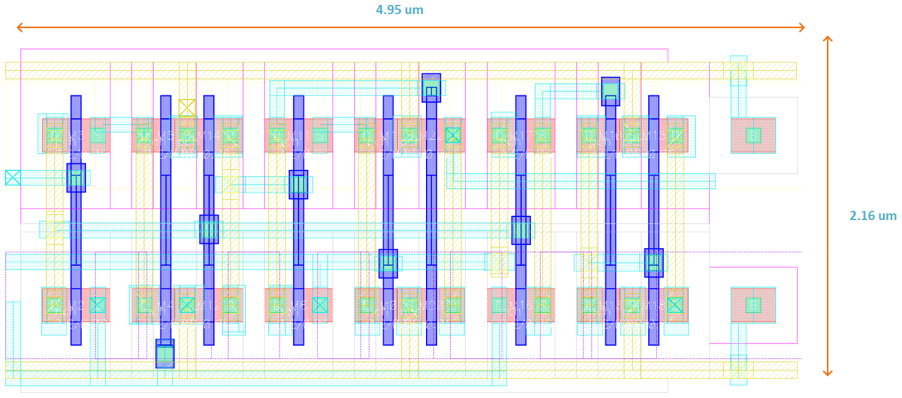

Figure 3.8: Layout of Proposed 18T Full Adder circuit.

Figure 3.8 shows the layout of the 9T NOR Bitcell having dimensions of $4.95\mu m$ in width and $2.16\mu m$ in height. Consequently, the area occupied by this single bitcell can be calculated as the product of its width and height, having total area of $(4.95\mu m) \times (2.16\mu m) = 10.69\mu m^2$.

3.2.1 Circuit Analysis of 18T Full Adder Circuit

The proposed 18T full adder circuit 3.7 effectively performs the addition of three single-bit binary inputs: A, B, and the carry-in C. The circuit generates two outputs: the sum (SUM) and the carry-out (COUT). Functionally, the circuit can be decomposed into distinct stages. The initial stage, comprising transistors M1, M2, M5, and M6, implements the exclusive-OR (XOR) operation between the input bits A and B, giving the intermediate result $A \oplus B$. Further, this intermediate result is fed into the second stage, having transistors M3, M4, M7, M8, M9, M10, M11, and M12. This stage performs another XOR operation between $(A \oplus B)$ and the carry-in C, thereby producing the final sum output, SUM $= (A \oplus B) \oplus C$. The carry-out signal (COUT) is generated by the final stage of the circuit, involving transistors M13, M14, M15, M16, M17, M18. This logic block implements COUT $= (A \cdot B) + (C \cdot (A \oplus B))$,

which can also be expressed as COUT = $AB + AC + BC$. COUT becomes high when at least two of the input bits (A, B, or C) are high.

Parameter	Metric	Value
Delay	T_{rise}	36.74 ps
	T_{fall}	52.64 ps
Average Power	-	2.41 µW

Table 3.2: 18T Full Adder circuit performance metrics @ 1.2V

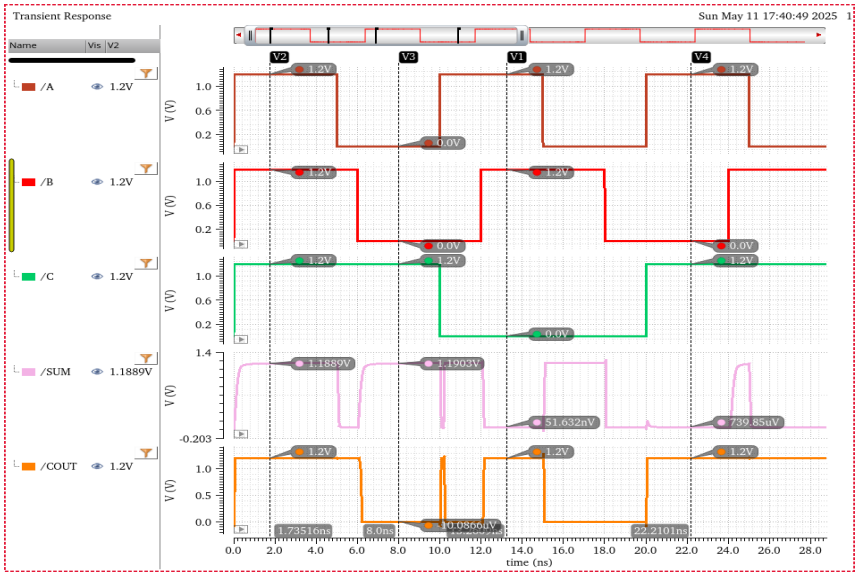

Figure 3.9: Transient simulation of 18T Full Adder circuit.

The waveform 3.9 shows the transient simulation results of 18T full adder circuit. Inputs A, B, and C are toggled to verify all possible input combinations. Outputs SUM and COUT transition correctly for all input combinations, verifying full adder functionality.

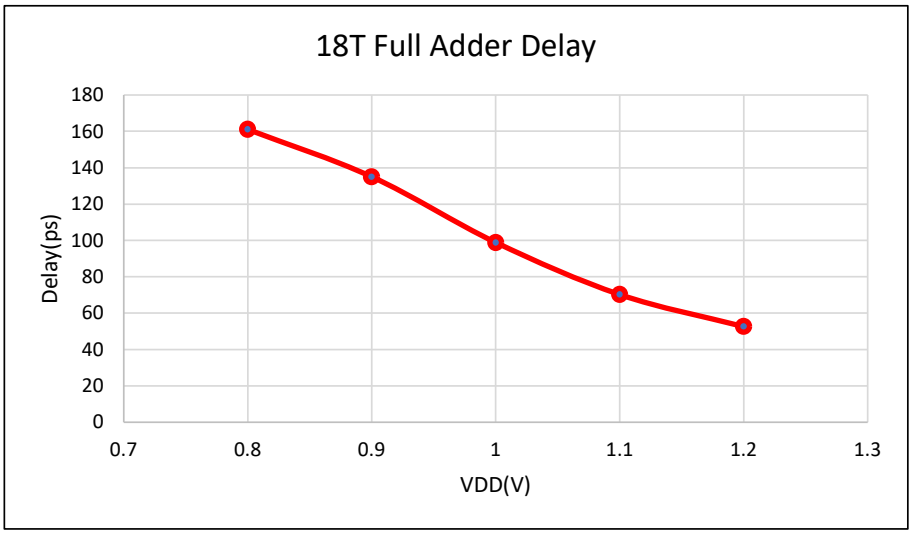

Figure 3.10: Variation in 18T Full Adder delay with respect to VDD.

As per the figure 3.10 delay of the 18T Full Adder decreases as the supply voltage (VDD) increases. As VDD is varied from 0.8 V to 1.2 V, the delay reduces from 161 ps to 52.64 ps. This behavior is due to the increased drive strength of the transistors at higher VDD, which allows faster switching. The reduction in delay with increasing voltage indicates improved performance, but the trade-off is the increased power consumption associated with higher VDD.

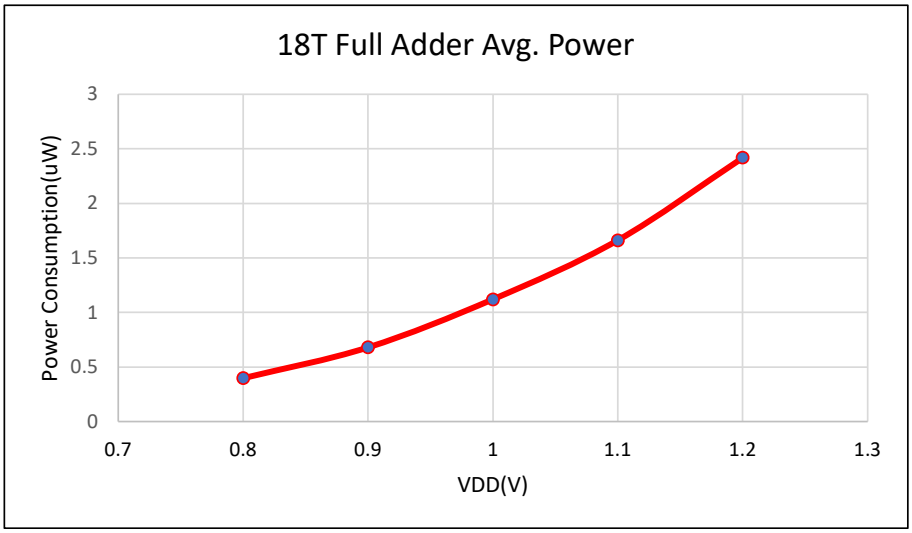

Figure 3.11: Variation in 18T Full Adder power consumption with respect to VDD.

Figure 3.11 power consumption of the 18T Full Adder increases significantly as

the supply voltage (VDD) rises. The power varies from 0.39 μW at 0.8 V to 2.41 μW at 1.2 V. This increase in power is primarily due to the higher dynamic power dissipation as a result of faster switching and higher capacitance charging at higher VDD.

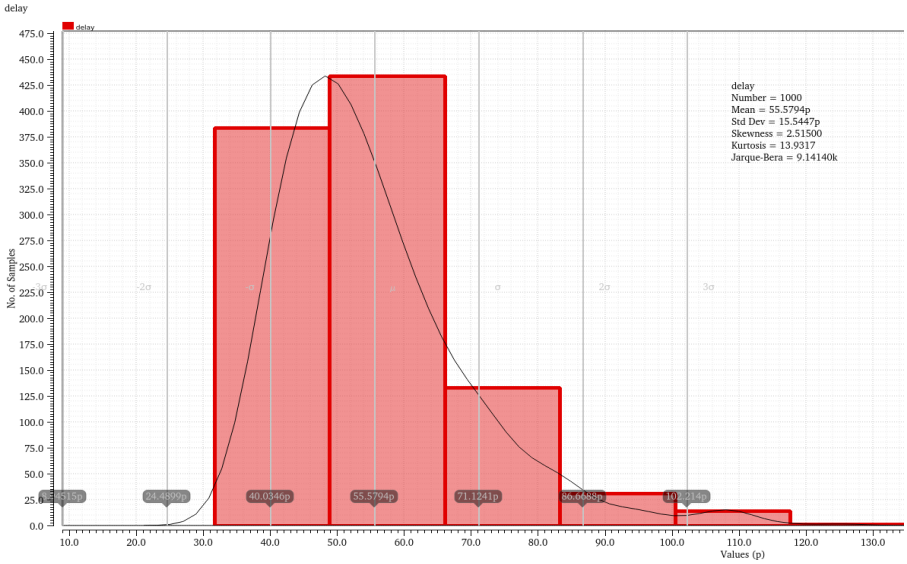

Figure 3.12: Monte-Carlo simulation of 18T Full Adder circuit.

The Monte Carlo simulation shown in Figure 3.12 analyzes how the delay of the 18T full adder changes due to process variations, using 1000 different samples at a supply voltage of 1.2,V. The histogram shows that most samples have lower delay values, while a few take longer, which results in a positively skewed curve. On average, the delay is 55.57ps, with a standard deviation of 15.54 ps.

3.3 Proposed Architecture of Digital In Memory Compute

The proposed 16-Kb DIMC macro is an IMC architecture optimized for MAC operation in neural network processing. The DIMC macro is has 64 banks, enabling simultaneous computation for multiple input features and increasing the throughput significantly.

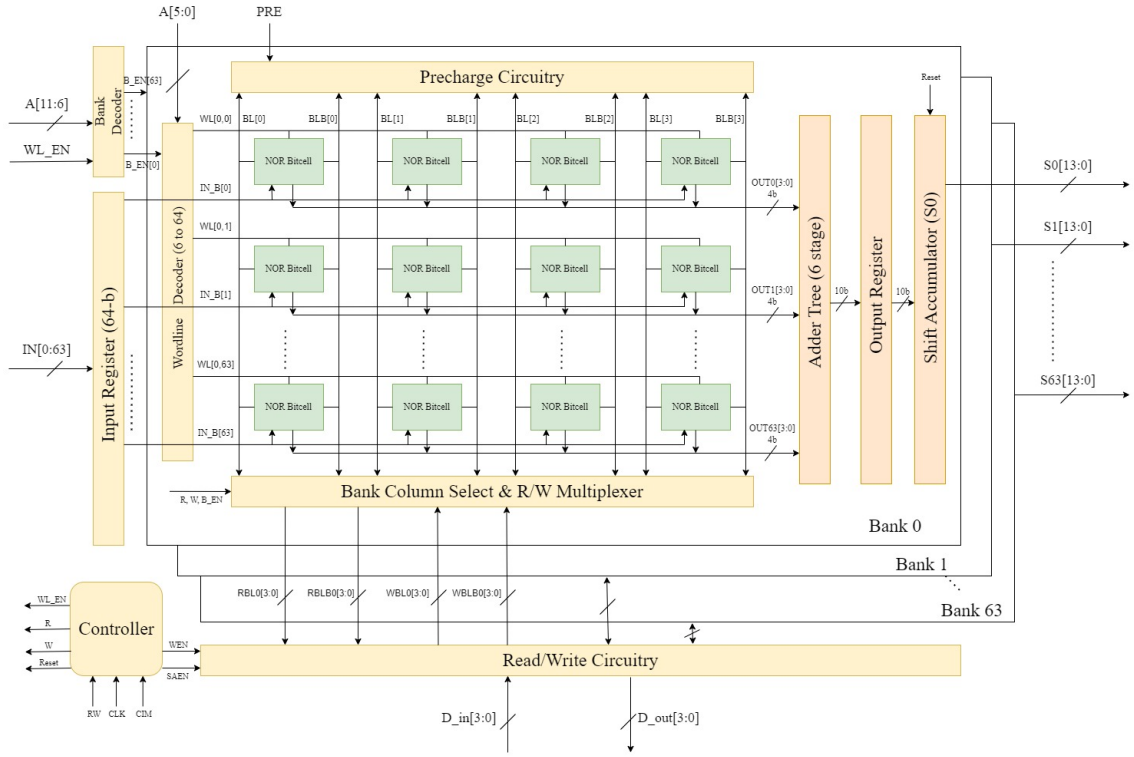

Figure 3.13: Proposed DIMC Architecture.

Each of the 64 banks comprises a 64×4 NOR bitcell array, which is main processing element in the architecture. This bitcell array is designed to perform bit-wise multiplication between 1-bit inputs and 4-bit weights stored within the array. Each bank stores one column of the weight matrix, with each row containing a 4-bit weight. This configuration ensures that the DIMC Macro can perform mac operations for 64 1bit inputs in one cycle in each bank out of 64 banks simultaneously. Bit-serial architectures process one bit per cycle per weight but achieve high parallelism,

making them efficient for low-precision DNN inference [16].

Alongside the 64x4 Nor Bitcell array, each bank includes:

- A six-stage adder tree implemented using for accumulation of partial products from NOr Bitcell array, optimized for both speed and area.
- A 10-bit output register that acts as a pipelining buffer between the adder tree and the next stage.
- A 14-bit shift accumulator that collects and accumulates 10bit partial sum across multiple input cycles, ensuring that the final output maintains the correct bit-precision.
- Standard SRAM peripheral circuits, including a 6-to-64 row decoder, read/write multiplexer, and precharge circuitry for standard memory access operations.

Weight Storage and Data Input Scheme: DIMC macro is designed to store the weight matrix in a bank-wise column format, such that each 4-bit weight is distributed across four columns of the array. The weight bits (QB) are used by 9T Nor bitcell to perform logical NOR operations with the $\overline{\text{IN}}$. The input data is provided row-wise in a bit-serial fashion, starting from the least significant bit (LSB) to the most significant bit (MSB). Each row of the array receives one input bit, and all four bitcells in the row (each representing a bit of the 4-bit weight) operate in parallel. The NOR logic used in each bitcell computes the logical NOR of the input bit and the stored QB, which functionally corresponds to a logical AND between the true input and weight bits, effectively achieving bitwise multiplication.This parallel computation across all banks results in the generation of 64 partial products, one from each row, which are then fed into 6-stage adder tree for accumulation.

6-stage Adder Tree: The six-stage adder tree performs accumulation the partial products generated by the 64x4 Nor Bitcell array. It starts with 64 inputs, each being a 4-bit partial product from one row. These inputs are reduced stage by stage using ripple-carry adders, eventually producing a 10-bit output that represents the accumulated sum for the given input bit cycle.

Output Register and Shift Accumulator: Once the adder tree completes the summation, the 10-bit result is latched into the output register, which serves as a pipeline register. This helps isolate the timing between the adder tree and the next stage, reducing critical path delays and allowing the macro to operate at a higher clock frequency.

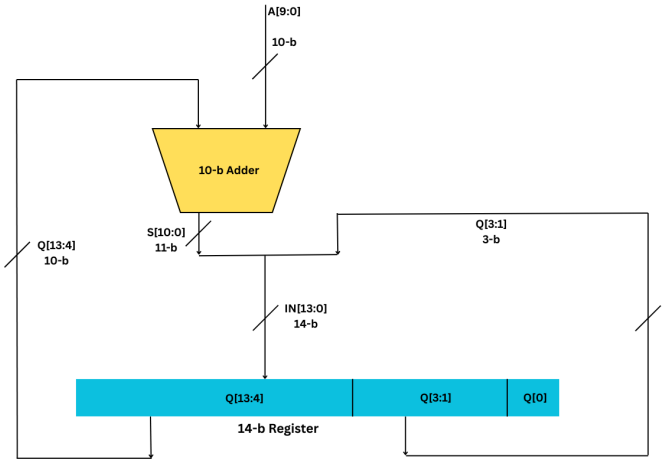

Figure 3.14: Schematic of Shift Accumulator.

The output from the 10bit pipeline register is then fed to 14-bit shift accumulator, which aligns the outputs from each input bit cycle and performs the final accumulation to compute the total product of the input vector and the stored weight matrix column.

As shown in figure 3.14, the output is accumulated by shifting the result register to the right in each clock cycle, while adding the current input. Unlike the conventional bit-serial method, where the input must be left-shifted by varying amounts every cycle, this technique simplifies the data path by keeping the input fixed. As a result, the design eliminates the need for a barrel shifter or dynamic shift logic for input alignment. The right-shifting of the result register inherently aligns the bits corresponding to their positional weight. Consequently, this reduces hardware complexity by avoiding multiple shifting stages.

3.3.1 Pipeline structure of Architecture

The proposed DIMC macro adopts a pipelined architecture to have high-throughput matrix-vector multiplication (MVM) operations using bit-serial processing. As illustrated in Figure 3.15, the design consists of three main pipeline stages: the input register, the bitcell array with an adder tree, and the shift-accumulation unit.

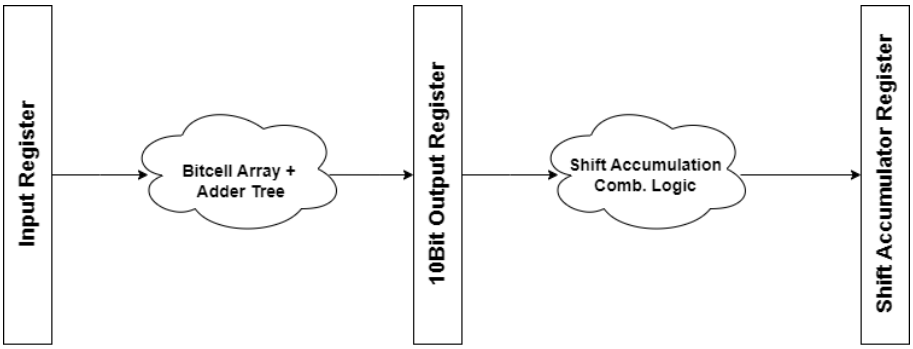

Figure 3.15: Pipeline structure of CIM Bank.

The Input Register holds the serial input vector bits, which are supplied bit-by-bit to the in-memory compute array. This array comprises a 64x4 NOR-based bitcell array for digital compute operations, bitwise AND operations between stored weights and input bits. These partial results are then accumulated using a six-stage adder tree. The result of each cycle is a 10-bit partial sum, captured by the 10-bit Output Register, which serves as a pipeline buffer between adder tree and shift accumulator.

The Shift Accumulation Logic performs bit-serial accumulation of partial sum results. This logic includes a 14-bit shift-accumulator that accumulates partial sums based on the current bit position of the input vector. This step enables accurate bit-level summation across multiple clock cycles, ensuring correct MAC operation result. The final accumulated results are stored in the Shift Accumulator Register.

This pipelined design ensures that while one bit-slice of computation is being processed by the Nor Bitcell array, previously computed partial sum is concurrently being accumulated, leading to improved throughput and power efficiency for edge-AI applications.

3.3.2 Timing Diagram for MAC operation

Figure 3.16 shows the timing diagram of MAC operation in the DIMC macro across multiple clock cycles. It shows a method of input vector fed serially into the DIMC macro and corresponding outputs accumulate over multiple clock cycles.

The CLK signal is reference for the operation of the macro. At clock cycle 0, a reset (RST) pulse initializes the system, clearing the internal registers and preparing the macro for computation. From cycle 1 onward, the IN[63:0] signal represents the bit-serial input, where 64 input bits are applied simultaneously for a single-bit slice. Each input slice corresponds to a single bit position (from LSB to MSB) of the 4-bit quantized activation vector.

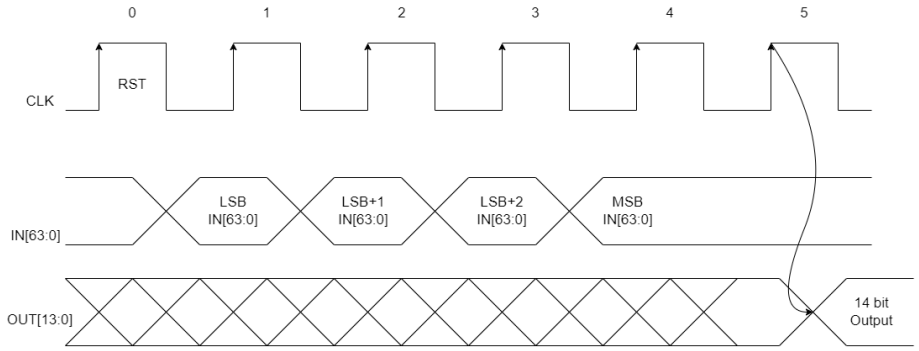

Figure 3.16: Timing Waveform of 4b/4b MAC operation in DIMC Macro.

The bit-serial data is supplied over multiple clock cycles: the Least Significant Bit (LSB) is processed in the first cycle, followed by the next higher-order bits in subsequent cycles, up to the Most Significant Bit (MSB). At each clock edge, the 64-bit wide input vector corresponding to the current bit position is fed to Nor Bitcell array, which performs a bitwise AND with the stored weights, followed by accumulation using adder tree and shift accumulator.

The OUT[13:0] waveform shows accumulated result over multiple clock cycles. Each new partial sum is shifted based on its input bit significance before being added to the previous result. This ensures correct bit alignment. The final 14-bit output becomes valid after the last bit slice (MSB) is processed, shown at the rising edge of

clock cycle 5.

3.4 Different Topologies of Adder Tree

The adder tree is a fundamental component of the Digital In-Memory Compute (DIMC) architecture, designed to accumulate the partial products generated by NOR bitcell array. Each of the 64 rows in a Nor Bitcell array produces a 4-bit partial product when a 1-bit input is applied to a 4-bit stored weight. These 64 parallel partial products are summed to produce the final dot product output for that cycle. To achieve this with precision and speed, a six-stage pipelined adder tree is implemented using multi precision RCA designed using full adders.

The adder tree follows a structure where output of each stage performs binary reduction of the outputs from the previous stage, progressively reducing the number of adders while increasing the output bit-width to accommodate carries. In the first stage, 64 four-bit inputs are added in pairs using 32 adders, each producing a 5-bit output. These 32 outputs are then added in the second stage using 16 adders that generate 6-bit results. Similiarly, the third stage uses 8 adders with 6-bit inputs producing 7-bit outputs; the fourth uses 4 adders for 7-bit inputs resulting in 8-bit outputs; the fifth uses 2 adders for 8-bit inputs yielding 9-bit outputs. Finally, a single 9-bit adder in the sixth stage combines the remaining two inputs to produce the final 10-bit output.

This adder tree design provides high throughput, enabling one complete accumulation per cycle for 1 bit of input. The increase in output bit-width ensures full bit precision maintained throughout the computation. As a result, the architecture supports high accuracy, which is essential for inference tasks in neural network applications.

Limitations of adder tree are that he number of full adders and increasing bit-width across stages results in higher delay and power consumption, which may be critical MAC operations. Thus, multiple adder tree topologies are explored to get best performance with optimum area and power consumption.

3.4.1 Standard 6-stage Adder Tree

The Digital In-Memory Compute (DIMC) macro employs a six-stage interleaving adder tree to accumulate the outputs of a 64×4 NOR bitcell array within each bank. Each row of the array produces a 4-bit partial product (PP), resulting from bitwise multiplication between a 1-bit input and a 4-bit stored weight. The goal of the adder tree is to sum these 64 partial products and produce a final 10-bit digital output.

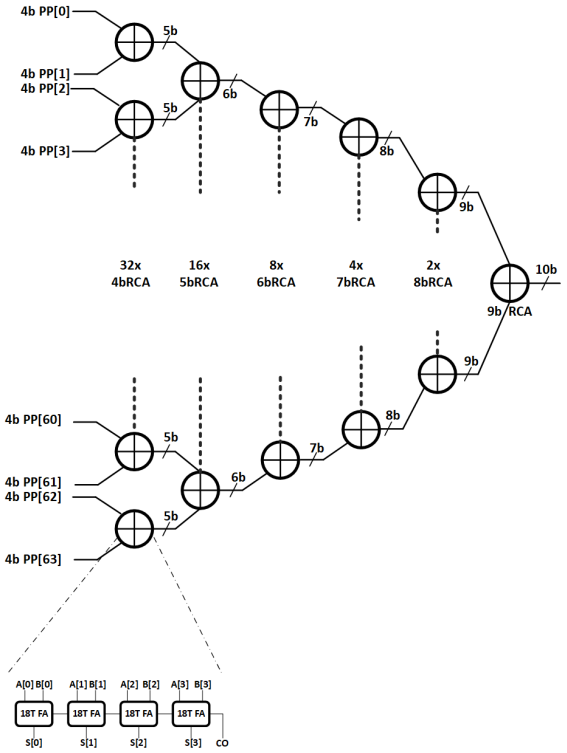

Figure 3.17: Standard 6-stage Adder Tree.

The adder tree shown in figure 3.17 is composed of Ripple Carry Adders (RCAs) arranged in an interleaved fashion, where 32 four-bit RCAs are used in the first stage to reduce 64 inputs to 32 outputs. The following stages use progressively wider RCAs—5-bit, 6-bit, up to 9-bit—culminating in a final 9-bit RCA that generates a 10-bit sum. This interleaved topology enables efficient data flow by balancing parallelism and depth across stages. At the circuit level, each RCA is constructed from custom-designed 18-transistor (18T) full adders, which offer compact area and energy-efficient logic operations using a combination of pass-transistor and CMOS

logic.

Parameter	**Value @ 1.2V**
T_{clk} (ns)	25
Max. Frequency(MHz)	40
Average Power (mW)	47.07
GOPS (4b W / 1b I)	327.68
GOPS (4b W / 4b I)	65.53
TOPS/W (4b W / 1b I)	6.97
TOPS/W (4b W / 4b I)	1.48

Average Power = Average Power (1 Bank) × 64
Throughput (4b/4b) = (64 × 64 × 2) / (5 × T_{clk})
Throughput (4b/1b) = (64 × 64 × 2) / T_{clk}
Power Efficiency = Throughput / Average Power

Table 3.3: Performance evaluation of the CIM macro with standard 6-stage adder tree @ 1.2V

Table 3.3 presents performance metrics of the proposed CIM macro implemented with the Standard 6-stage adder tree at a supply voltage of 1.2V. The DIMC macro achieves a clock period of 25ns (40MHz), delivering up to 327.68GOPS for 4b/1b precision and 65.53GOPS for 4b/4b, with corresponding energy efficiencies of 6.97TOPS/W and 1.48TOPS/W at an average power of 47.07mW.

The proposed adder tree offers full digital precision without the need for DACs, eliminating its area overheads. Its modular and scalable design supports flexible integration across different array sizes.

This adder tree from increased latency due to the limited driving strength of 18T full adders, especially in higher stages of the adder tree where fanout load increases. To address this, two interleaving adder tree topologies are proposed:

Topology-1 (18T–28T Interleaving) adder tree: The early stages of the

adder tree utilize area-efficient 18T full adders, while the later stages uses 28T standard cell full adders. This combination aims to balance area and performance.

Topology-2 (28T–18T Interleaving) adder tree: In this topology, high-performance 28T standard cell full adders are used in the initial stages to minimize initial delay due to Nor Bitcell array load, followed by 18T full adders in the later stages to reduce area and power.

3.4.2 Topology-1 Interleaving 6-stage Adder Tree

To reduce the latency and area overhead due to standard 6-stage adder tree, Topology-1 6-stage adder tree shown in figure 3.18 has an interleaved structure of 18T and 28T full adders across the six stages of the adder tree. This topology alternates the two types of full adders to get a balance between area efficiency and logic robustness.

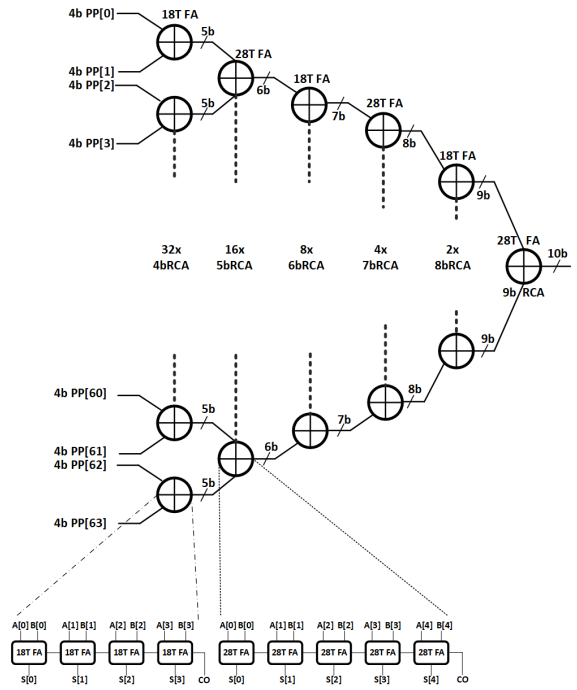

Figure 3.18: Topology-1 interleaving 6-stage adder tree.

In this architecture, the first stage consists of 18T full adders, offering compact area and lower power. The following stages—5-bit to 10-bit adders—alternate

between 28T and 18T full adders, gradually introducing stronger drivers to support higher bit-width operations and growing fan-out. The aim is to allow compact adders where possible and use stronger logic where necessary to maintain functionality and performance. This interleaving strategy attempts to reduce critical path delay with trading off area efficiency, especially in later stages of the adder tree.

Parameter	**Value @ 1.2V**
T_{clk} (ns)	19
Max. Frequency(MHz)	52.6
Average Power (mW)	44.9
GOPS (4b W / 1b I)	431.15
GOPS (4b W / 4b I)	107.78
TOPS/W (4b W / 1b I)	9.58
TOPS/W (4b W / 4b I)	1.91

Average Power = Average Power (1 Bank) × 64
Throughput (4b/4b) = (64 × 64 × 2) / (5 × T_{clk})
Throughput (4b/1b) = (64 × 64 × 2) / T_{clk}
Power Efficiency = Throughput / Average Power

Table 3.4: Performance evaluation of the CIM macro with Topology-1 6-stage adder tree @ 1.2V

Table 3.4 presents performance metrics of the proposed CIM macro implemented with the Topology-1 6-stage adder tree at a supply voltage of 1.2V. The DIMC macro achieves a clock period of 19ns (52.6MHz), delivering up to 431.15GOPS for 4b/1b precision and 107.78GOPS for 4b/4b, with corresponding energy efficiencies of 9.58TOPS/W and 1.91TOPS/W at an average power of 44.9mW.

Despite the use of 28T standard cell full adders in intermediate stages of adder tree, Topology-1 has gained limited performance over standard 6-stage adder tree. This is due to the bottleneck in first stage of adder tree where all additions are

performed by 18T full adders which lack the driving strength required to drive the high capacitive load of Nor bitcell array. Therefore, even with increased area due to the 28T standard cell full adders in later stages, the performance improvement is not substantial. The system remains bottlenecked by the weak initial stage.

To address this, Topology-2 6-stage adder tree is proposed where the first stage employs 28T standard cell full adders to drive the Nor Bitcell array load, and later stages use 18T full adders and 28T standard cell full adders in alternate method.This improves the overall timing and throughput of the adder tree with trading off area.

3.4.3 Topology-2 Interleaving 6-stage Adder Tree

Due to the limitations in Topology-1 adder tree, the Topology-2 adder tree shown in figure 3.19 uses 28T standard cell full adders in the first stage, followed by use of 18T full adders and 28T standard cell full adders in later stages.

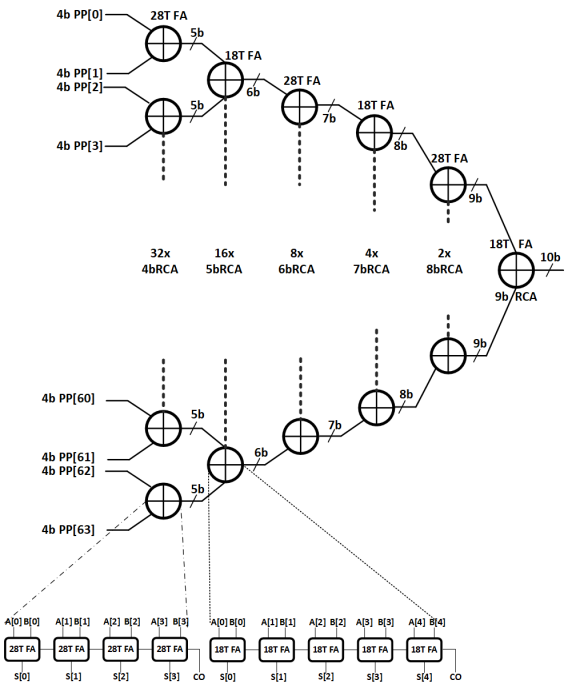

Figure 3.19: Topology-2 interleaving 6-stage adder tree.

In Topology-2 adder tree, this issue is directly addressed by using robust 28T standard cell full adders in the first stage, which is capable of driving large capacitive

loads due to which critical path delay is significantly reduced.

The further stages in Topology-2 adder tree use 18T full adders and 28T standard cell full adders in alternate method. Use of 18T full adders saves area and power. This strategy forms a balanced tradeoff between performance, area and power.

Topology-2 adder tree incurs slightly more area overhead due to use of 28T standard cell full adders in big stages of adder tree where more numbers of full adders are required but this tradeoff helps in achieving the required performance and throughput for DIMC macro.

Parameter	**Value @ 1.2V**
T_{clk} (ns)	1.9
Max. Frequency(MHz)	526.31
Average Power (mW)	38.4
GOPS (4b W / 1b I)	4311.57
GOPS (4b W / 4b I)	862.31
TOPS/W (4b W / 1b I)	112.28
TOPS/W (4b W / 4b I)	22.45

Average Power = Average Power (1 Bank) × 64
Throughput (4b/4b) = (64 × 64 × 2) / (5 × T_{clk})
Throughput (4b/1b) = (64 × 64 × 2) / T_{clk}
Power Efficiency = Throughput / Average Power

Table 3.5: Performance evaluation of the CIM macro with Topology-2 6-stage adder tree @ 1.2V

Table 3.5 presents performance metrics of the proposed CIM macro implemented with the Topology-2 6-stage adder tree at a supply voltage of 1.2V. The DIMC macro achieves a clock period of 1.9ns (526.31MHz), delivering up to 4311.57GOPS for 4b/1b precision and 862.31GOPS for 4b/4b, with corresponding energy efficiencies of 112.28TOPS/W and 22.45TOPS/W at an average power of 38.4mW.

Chapter 4

Performance Evaluation & Lenet-5 CNN model Accuracy Estimation

In this chapter, a detailed evaluation of the performance of the designed Compute-In-Memory (CIM) macro, along with an analysis of the accuracy estimation of the LeNet-5 Convolution Neural Network (CNN) model is presented. The first part of this chapter focuses on the simulation results and performance evaluations of the CIM macro. Both pre- and post-layout performance evaluations are discussed to give idea about DIMC Macro's functional verification, power consumption, area, and timing performance. Performance of proposed DIMC macro is also compared with other similar works to get insight of DIMC macro.

Further, the application of the CIM macro in accelerating neural network LeNet-5 CNN model, which is used for image classification of handwritten digits. The accuracy of the model is estimated based on the hardware implementation, taking into account the quantization effects.

4.1 Simulation and Performance Evaluation

This section presents the simulation and performance evaluation of the DIMC macro, for checking the overall functionality and efficiency of the system. The performance evaluations are for both both pre-layout and post-layout simulations

to ensure the DIMC macro's performance. This analysis provides insights into the DIMC macro having enabling high-performance, energy-efficient computing. The results discussed here are essential for validating the CIM macro's application in edge AI systems.

4.1.1 Functional verification of CIM Macro

To validate the functional correctness of the designed 64×4 Compute-In-Memory (CIM) bank macro, a comprehensive simulation was carried out using predefined vectors for input activations and weights. Both vectors consisted of 64 elements each, where every element is a 4-bit unsigned integer ranging from 0 to 15.

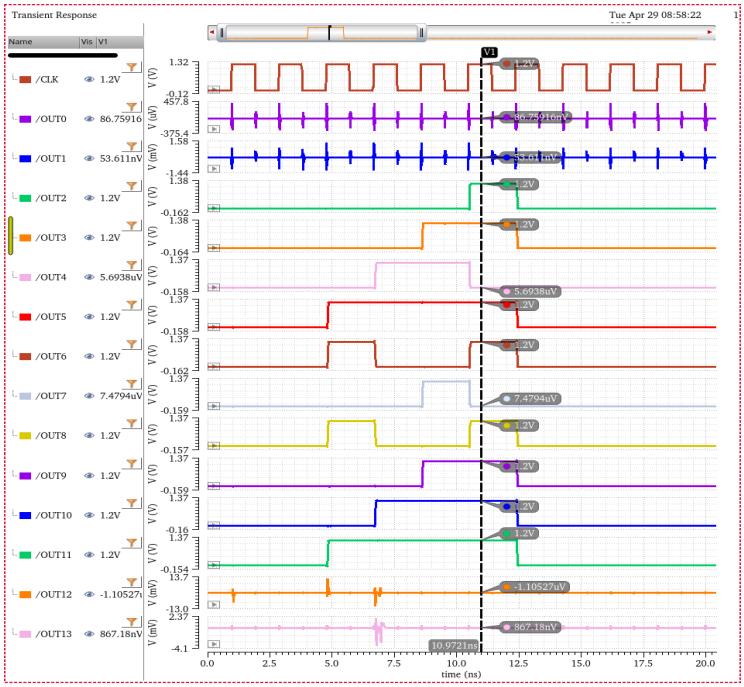

Figure 4.1: Functional verification of Mac operation in CIM Bank.

The input activation vector used for the simulation was: [6, 12, 10, 8, 8, 15, 15, 15, 4, 11, 6, 14, 5, 9, 9, 5, 15, 9, 2, 0, 6, 10, 2, 11, 9, 14, 1, 12, 6, 11, 11, 3, 8, 2, 13, 0, 3, 5, 11, 10, 9, 14, 8, 11, 6, 12, 4, 6, 7, 9, 6, 0, 7, 5, 15, 1, 1, 9, 1, 5, 15, 7, 14, 15], while the corresponding weight vector was: [15, 7, 7, 0, 9, 11, 1, 3, 10, 8, 11, 10, 14, 12, 14, 10, 12, 7, 12, 5, 3, 4, 6, 0, 6, 7, 7, 9, 10, 6, 12, 6, 6, 2, 12, 5, 15, 15, 9, 13, 1, 1,

11, 3, 0, 11, 9, 1, 10, 10, 10, 12, 0, 15, 10, 10, 10, 9, 1, 11, 5, 13, 7, 5].

The transient simulation shown in figure 4.1 validates the Multiply-and-Accumulate (MAC) operation performed within the CIM bank, where each input element is multiplied by its corresponding weight and the resulting partial products are accumulated to produce a final 14bit output. After performing this MAC operation, the computed output was obtained as a 14-bit binary value: OUT[13:0] = 00111101101100. Converting this binary value to its decimal equivalent yields 3948.This matches with the expected MAC result verifies the functionality of CIM bank.

4.1.2 Post-layout simulation of CIM Macro

The figure 4.2shows layout of four key sub-circuits used in a Compute-In-Memory (CIM) bank. Sub-circuit (a) represents the 64×4 NOR-based bitcell array, occupying an area of 13.25 µm × 128.84 µm, which results in a total area of 1707.13 μm^2. This bitcell array is responsible for storing 4-bit weights used during Multiply-and-Accumulate (MAC) operations.

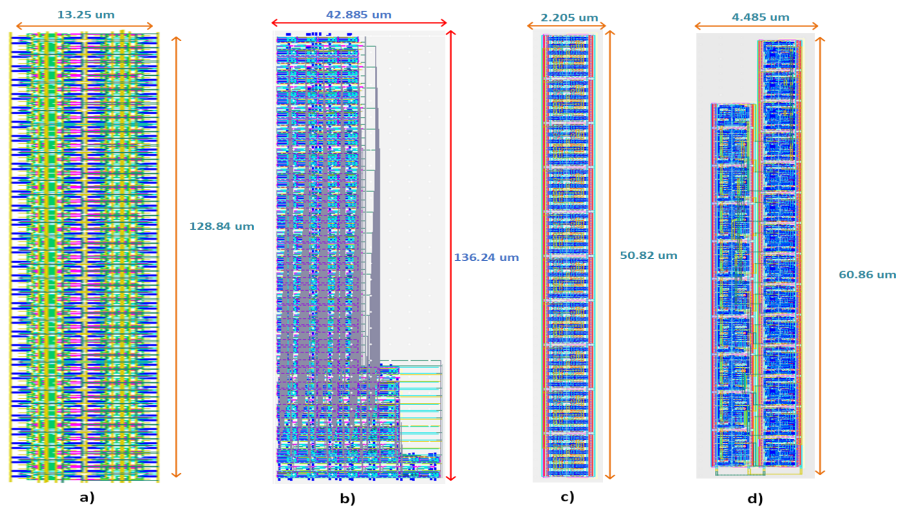

Figure 4.2: Layout of CIM Bank Subcircuits (a) 64x4 Bitcell array (b) Topology-2 Adder tree (c) 10bit pipeline Output Register (d) Shift Accumulator.

Sub-circuit (b) is the adder tree, which vertically accumulates partial products

from the bitcell array into a 10-bit result. Its layout spans 42.885 µm in width and 136.24 µm in height, yielding an area of 5842.65 µm^2. Sub-circuit (c), the 10-bit output register, is designed to hold the result from the adder tree at each clock cycle. It is laid out in a compact form with dimensions of 2.205 µm × 50.82 µm, resulting in an area of 112.06 µm^2. Finally, sub-circuit (d) shows the shift accumulator, which performs accumulation of partial sums over multiple clock cycles. Its layout measures 4.485 µm in width and 60.86 µm in height, giving an area of 272.95 µm^2.

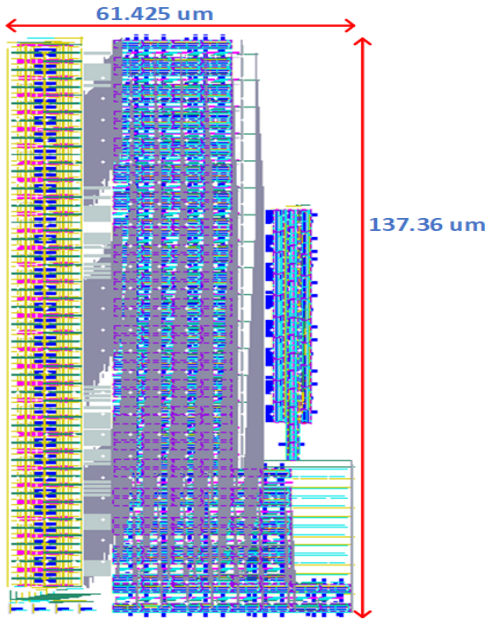

Figure 4.3: Layout of 64x4 CIM Bank.

The figure 4.3 shows the complete layout of a single CIM bank, which integrates the bitcell array, adder tree, 10-bit output register, and shift accumulator as discussed in previous sections. The overall dimensions of the bank layout are 61.425 µm in width and 137.36 µm in height, resulting in a total area of 8437.33 µm^2. The bank operates by receiving input activations from an off-chip input register, which are then applied to the prestored weights within the bitcell array. The partial products generated in the array are accumulated vertically using the adder tree. These results are temporarily stored in the 10-bit pipeline register and subsequently accumulated over multiple clock cycles by the shift accumulator to complete the MAC operation. It is important to note that off-chip components such as the input register and

control logic are not included in this layout. This modular design allows seamless tiling of multiple such banks to build higher-level DIMC macros.

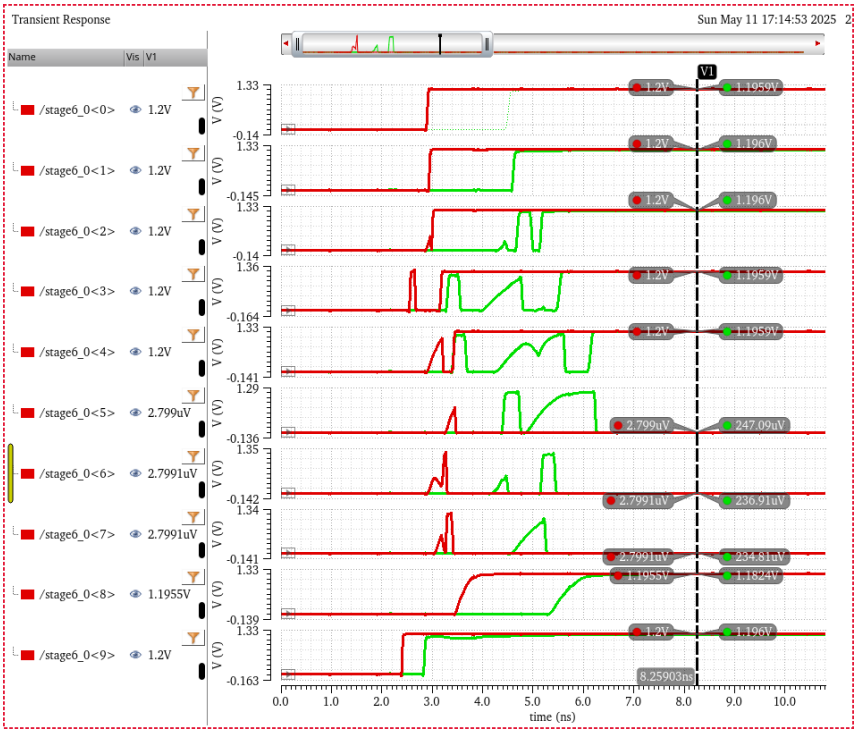

Figure 4.4: Post-layout simulation of CIM Bank(4b Weight/1b Input).

The figure 4.4 shows the post-layout transient simulation of the DIMC bank for a 4-bit weight and 1-bit input configuration. 10bit output for 4b weight and 1bit input is plotted, with red representing pre-layout and green representing post-layout waveforms. The close overlap between red and green signals validates functional correctness after layout, with minimal degradation.

4.1.3 Pre-Layout Performance Estimation of CIM Macro

Table 4.1 summarizes the pre-layout performance metrics of the proposed 16Kb Digital In-Memory Compute (DIMC) macro at various supply voltages ranging from 1.2V to 0.8V. It includes the clock period (T_{clk}), maximum operating frequency, and total average power estimated across 64 banks. The throughput values are calculated for two precision configurations: (4-bit weight, 1-bit input) and (4-bit weight, 4-bit

input), based on a bit-serial VMM operation model. Power efficiency is shown in TOPS/W, representing energy efficiency at each voltage for both configurations. Area efficiency is calculated in GOPS/mm², highlighting performance relative to silicon area. As expected, higher voltages offer better performance and efficiency, while lower voltages reduce power consumption at the cost of throughput.

Voltage (V)	T_{clk} (ns)	Max Clock Freq. (MHz)	Avg. Power (mW)	Throughput (GOPS) (4b/1b)	Throughput (GOPS) (4b/4b)	Power Eff. (TOPS/W) (4b/1b)	Power Eff. (TOPS/W) (4b/4b)	Area Eff. (GOPS/mm²) (4b/1b)	Area Eff. (GOPS/mm²) (4b/4b)
1.2	1.9	526.31	38.4	4311.57	862.31	112.28	22.45	7984.56	1596.91
1.1	2.5	400.00	30.9	3276.80	655.36	106.00	21.20	6068.27	1213.65
1.0	4.8	208.33	22.2	1706.66	341.33	76.85	15.36	3160.55	632.11
0.9	10.0	100.00	13.4	819.20	163.84	60.90	12.19	1517.06	303.11
0.8	21.0	47.61	7.8	390.09	78.01	49.96	9.99	722.41	144.48

Average Power = Average Power (1 Bank) × 64

Throughput (4b/4b) = (64 × 64 × 2) / (5 × T_{clk})

Throughput (4b/1b) = (64 × 64 × 2) / T_{clk}

Power Efficiency = Throughput / Average Power

Table 4.1: Pre-layout results of the CIM macro using Topology-2 adder tree under varying VDD

4.1.4 Post-Layout Performance Estimation of CIM Macro

Table 4.2 provides a post-layout evaluation of the CIM macro's performance under multiple supply voltage conditions ranging from 1.2 V down to 0.8 V. The clock period (T_{clk}) increases with lower voltage levels due to reduced drive strength and slower transistor switching behavior, which in turn lowers the maximum operating frequency of the macro. As observed from the post-layout results, operating the DIMC macro at higher supply voltages leads to better performance and energy efficiency due to faster signal propagation and reduced delays. However, this comes with increased power consumption. On the other hand, running at lower voltages significantly reduces power usage, making it suitable for energy-constrained applications, but results in a drop in throughput and overall performance.

Voltage (V)	T_{clk} (ns)	Max Clock Freq. (MHz)	Avg. Power (mW)	Throughput (GOPS) (4b/1b)	Throughput (GOPS) (4b/4b)	Power Eff. (TOPS/W) (4b/1b)	Power Eff. (TOPS/W) (4b/4b)	Area Eff. ($GOPS/mm^2$) (4b/1b)	Area Eff. ($GOPS/mm^2$) (4b/4b)
1.2	4.137	241.72	43.00	1980.17	396.03	46.04	9.20	3667.07	733.41
1.1	6.255	159.87	35.13	1309.67	261.93	37.27	7.45	2425.36	485.07
1.0	10.81	92.50	25.53	757.81	151.56	29.67	5.93	1403.39	280.67
0.9	21.739	46.00	17.98	376.83	75.36	20.95	4.19	697.85	139.57
0.8	52.78	18.94	11.07	155.21	31.04	14.01	2.80	287.43	57.48

Average Power = Average Power (1 Bank) × 64

Throughput (4b/4b) = (64 × 64 × 2) / (5 × T_{clk})

Throughput (4b/1b) = (64 × 64 × 2) / T_{clk}

Power Efficiency = Throughput / Average Power

Table 4.2: Post-layout results of the CIM macro using Topology-2 adder tree under varying VDD

4.2 Comparison of CIM Macro with Other Works

Table 4.3 presents a comparative analysis of the proposed Compute-In-Memory (CIM) design with different state-of-the-art architectures from recent literature. All designs vary in technology nodes, precision formats, array sizes, and operational frequencies. The proposed work is implemented in 65 nm CMOS technology with a 16 Kb array, supporting both 4-bit weight/4-bit input and 4-bit weight/1-bit input configurations. The flexibility in input precision allows a trade-off between throughput and energy efficiency. For the 4b/1b configuration, the design achieves a peak throughput of 1980.17 GOPS and an energy efficiency of 46.04 TOPS/W, while the 4b/4b configuration yields 396.03 GOPS and 9.20 TOPS/W. In comparison, ESSCIRC'22 [6], which uses 1b/1b precision, offers higher frequency (1230 MHz) but slightly lower energy efficiency at 34.98 TOPS/W. JSSC'21 [7] and JSSC'21 [8] adopt lower-bit precision and achieve moderate throughput and efficiency, with JSSC'21 [8] showing 49.4 TOPS/W for variable precision inputs. JSSC'2020 [9], while operating in 55 nm technology with only 4 Kb array, achieves 40.2 TOPS/W but has poor area efficiency at 0.055 $TOPS/mm^2$. The proposed design exhibits a strong area efficiency of 3.66 $TOPS/mm^2$ (4b/1b), significantly outperforming others, especially in higher-bit precision configurations. All results in this work are

derived from post-layout simulation. Overall, the design achieves a favorable balance across performance, energy, and area metrics, proving its utility for efficient edge AI applications.

Metrics	This Work*	ESSCIRC'22 [6]	JSSC'21 [7]	JSSC'21 [8]	JSSC'2020 [9]
Technology	65nm	65nm	40nm	65nm	55nm
Array Size	16Kb	16Kb	32Kb	64Kb	4Kb
Weight/Input precision	(4b/4b) & (4b/1b)	1b/1b	2b/2b	1b-4b	1b-4b
Supply Voltage (V)	1.2V	1.2V	0.9V	1.2V	0.9V
Frequency (MHz)	241.72	1230	65	70	285.7*
Throughput (GOPS)	1980.17 (4b/1b), 396.03 (4b/4b)	1259 (1b/1b)	122 (2b/2b)	573.4 (2b/4b)	329 (1b/1b)
Energy Efficiency (TOPS/W)	46.04 (4b/1b), 9.20 (4b/4b)	34.98 (1b/1b)	41 (2b/2b)	49.4 (1b/4b)	40.2 (1b/1b)
Area Efficiency (TOPS/mm^2)	3.66 (4b/1b), 0.73 (4b/4b)	0.583 (1b/1b)	2.4 (2b/2b)	3.4 (2b/4b)	0.055(1b/1b)

* Calculated Value from Access time 3.5ns given.

Table 4.3: Post-layout result comparison of the CIM macro using Topology-2 adder tree with prior state-of-the-art designs

4.3 Lenet-5 CNN model Accuracy Estimation

The LeNet-5 architecture is quantized as per 4bit weight and 4b input precision supported in DIMC Macro and implemented using PyTorch in with the Brevitas library, which provides support for quantization-aware training of DNN models.

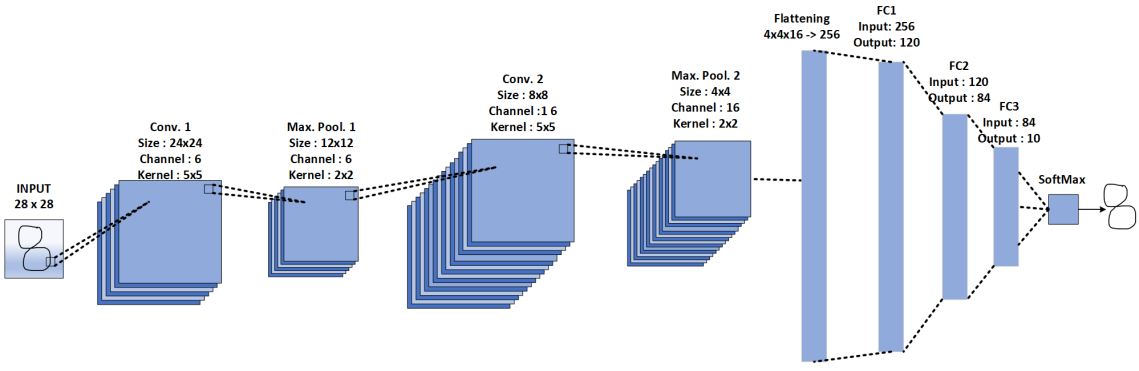

Figure 4.5: Lenet-5 CNN model architecture. [10]

In this implementation, both the convolutional (Conv2D) and fully connected (Linear) layers were quantized to 4-bit unsigned integers for weights and activations, thereby significantly reducing memory footprint and computational complexity. The model was trained on the MNIST dataset, which consists of grayscale images of handwritten digits (0–9) with a resolution of 28×28 pixels. A total of 100 training epochs were executed using a batch size of 100. For optimization, the SGD algorithm was employed along with a learning rate of 0.005. The SGD optimizer updates the network's weights based on the error gradients computed over mini-batches, enabling efficient convergence. The loss function used was Cross Entropy Loss, which effectively penalizes incorrect class predictions by comparing the predicted probability distribution with the true labels. With this configuration, the quantized LeNet-5 model achieved a classification accuracy of 90.8% on the test set under the 4-bit weight and 4-bit activation (4b/4b) setup. Additionally, the model demonstrated an average loss of 0.2920.

Layer	**Kernel**	**Output**	**MAC Operations**
Input	-	$28 \times 28 \times 1$	-
Conv. 1	$5 \times 5 \times 6$	$24 \times 24 \times 6$	86.4K
Max Pool 1	2×2	$12 \times 12 \times 6$	-
Conv. 2	$5 \times 5 \times 16$	$8 \times 8 \times 16$	153.6K
Max Pool 2	2×2	$4 \times 4 \times 16$	-
Flatten	-	256×1	-
FC1	$1 \times 1 \times 120$	120	30.72K
FC2	84	10	10.08K
FC3	10	1	0.84K

Table 4.4: Layer-wise Breakdown of Lenet-5 CNN Model

Table 4.4 provides a layer-wise breakdown of a Convolutional Neural Network (CNN) architecture, starting with a 28×28×1 input image. It includes two convolution layers (Conv. 1 and Conv. 2) with 5×5 kernels, each followed by 2×2 max pooling layers. After flattening, the data passes through three fully connected (FC) layers with decreasing node counts. The table also shows the number of multiply-accumulate (MAC) operations required at each stage, totaling approximately 281.64K operations. This compact architecture is well-suited for low-complexity tasks such as MNIST digit classification.

4.4 Summary

This chapter presented the performance evaluation of the proposed DIMC) macro and estimation of accuracy of quantized LeNet-5 CNN model for inference tasks. The functionality of the designed CIM macro was first verified through pre-layout simulations to validate MAC operations. Subsequently, post-layout simulations were conducted to ensure signal integrity with parasitics load. Pre-layout performance estimation was performed at a supply voltage of 1.2,V, where the macro achieved a throughput of 4311.57GOPS for the (4b/1b) configuration and 862.31 GOPS for (4b/4b), with energy efficiency values of 112.28 TOPS/W and 22.45 TOPS/W. Post-layout analysis showed a throughput of 1980.17 GOPS (4b/1b) and 396.03 GOPS (4b/4b), with corresponding energy efficiencies of 46.04 TOPS/W and 9.20 TOPS/W, respectively. The chapter also includes a comparative study of the proposed macro with other state-of-the-art CIM designs. Finally, the LeNet-5 CNN model, quantized to 4-bit weights and activations using the Brevitas library, was trained on the MNIST dataset and achieved a test accuracy of 90.8% with an average loss of 0.2920, demonstrating the effectiveness of the proposed CIM architecture in edge AI applications.

Chapter 5

Conclusion and Future Scope

5.1 Summary of Contribution

This thesis presents the design, optimization, and evaluation of a 16 Kb DIMC macro, optimized for VMM IMC operations, with a focus on its application in edge AI devices. The primary goal of this work is to design an efficient IMC architecture that can be utilized in hardware accelerator capable of addressing the performance and power requirements of edge AI applications, where real-time processing and energy efficiency are critical.

- **Design and Optimization of a 16-Kb DIMC Macro:** In thesis 16-Kb DIMC macro is implemented that performs both IMC and memory functions to carry out VMM IMC operations. By minimizing the need to move data back and forth between memory and processing units power usage is reduced compared to conventional systems. Thus, DIMC macro provides optimum throughput, latency and energy efficiency. Thus, it offers a good solution for AI inference at the edge, where performance and power efficiency are critical.

  The DIMC macro was validated through both pre-layout and post-layout simulations, ensuring its functionality. Post-layout simulations validates the DIMC macro's high throughput and low latency with optimum power and area efficiency, with post-layout parasitics.

- **9T NOR Bitcell and 18T Full Adder Design for DIMC Architecture:** In this work 9T NOR bitcell and a 18T full adder are designed for compute-in-memory operations to support energy-efficient IMC operation in edge AI devices. The 9T NOR bitcell performs logic-in-memory operations by integrating logic functionality within the memory bitcell which significantly reduces data movement and enabling parallel computation in memory arrays. 9T Nor Bitcell circuit ensures read and write stability, making it suitable for IMC with minimal loss in reliability.

  To accumulate the partial product given by 9T Nor Bitcell, an 18T full adder is designed. 18T Full adder performs accumulation operations with reduced power and delay. Optimized for speed and energy efficiency, this full adder gives optimum balance between propagation delay and power consumption, contributing to faster MAC operations and improving the throughput of DIMC Macro.

  Together, the integration of the 9T NOR bitcell and the 18T full adder enables MAC operation directly within the DIMC Macro. Thus, it helps in building scalable and low-power IMC architectures that are required for AI inference in edge devices.

- **Evaluation of Different Adder Tree Topologies in DIMC Architecture:** In this thesis multiple 6-stage adder tree topologies within the DIMC Macro are explored to optimize accumulation of partial production in mAC operation. There three topologies of adder tree used in DIMC macro named Standard 6-stage adder tree, Topology-1 6-stage interleaving adder tree , and Topology-2 6-stage interleaving adder tree. All three adder tree are analyzed to observe trade-offs between latency, area, and power consumption. The Topology-2 interleaving adder tree provided optimum computation speed and energy efficiency of DIMC Macro, having trade off with area. Thus it shows architecture's utility for Neural Network processing.

- **Accuracy Benchmarking with Lenet-5 CNN Model:** To check the

accuracy when used for real neural network , the macro was evaluated using a quantized Lenet-5 CNN model on the MNIST dataset. The model operated with 4-bit weights and 1-bit to 4-bit inputs, showcasing the macro's ability to handle varying levels of precision—a key requirement for edge inference. The inference accuracy remained within acceptable limits despite quantization, validating the ability of the macro for CNN applications.

5.2 Future Scope

The work presented in this thesis lays a strong foundation for efficient IMC architectures in edge AI systems. However, area remains for further research to enhance the performance, scalability, and broader applicability of the proposed 16-Kb DIMC macro. Future work can focus on optimizing existing techniques and exploring new technologies to drive greater efficiency and functionality in edge AI applications. **Key areas for future exploration include:**

- **Increase in Bit Width for Enhanced Accuracy:** While the current implementation uses 4-bit weights and variable input precision ranging from 1-bit to 4-bit, future work could explore increasing the bit width for weights and inputs. This would improve the accuracy of the model, allowing for more precise computations and better performance in complex neural network tasks, while still balancing power and memory efficiency.

- **Integration of Emerging Memory Technologies:** Explore integrating emerging memory technologies, such as ReRAM or MRAM, into the DIMC macro to further improve its performance, scalability, and energy efficiency.

- **Increase Reconfigurability for Optimum Performance:** Further work could focus on increasing the reconfigurability of the architecture to allow for increased precision levels. This would enable the system to adapt in real-time to the specific demands of different AI workloads, optimizing performance and

energy efficiency across various edge AI applications while ensuring greater flexibility and scalability.

- **Hardware-Software Co-Design for Optimization:** Develop methods for the co-design of hardware and software to streamline the deployment of AI models on edge devices. This would optimize real-time processing and ensure efficient execution of AI tasks on resource-constrained platforms.

By addressing these future directions, the advancements presented in these works can be further improved, contributing to the development of more efficient, reliable, and high-performance IMC architectures for edge AI applications.

References

[1] Abu Sebastian, Tomas Tuma, Nikolaos Papandreou, Manuel Le Gallo, Lukas Kull, Thomas Parnell, and Evangelos Eleftheriou. "Temporal correlation detection using computational phase-change memory." *Nature Communications*, 8(1):1115, Oct 2017.

[2] N. Verma et al., "In-Memory Computing: Advances and Prospects." *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43-55, Aug 2023.

[3] Vishal Sharma, Hyunjoon Kim, and Tony Tae-Hyoung Kim. "A 64 kb reconfigurable full-precision digital ReRAM-based compute-in-memory for artificial intelligence applications." *IEEE Transactions on Circuits and Systems I: Regular Papers*", 69(8):3284-96, Apr 2022.

[4] J. -s. Seo et al., "Digital Versus Analog Artificial Intelligence Accelerators: Advances, trends, and emerging designs." *IEEE Solid-State Circuits Magazine*, 14(3):65-79, Aug 2022.

[5] H. Kim, T. Yoo, T. T. -H. Kim and B. Kim, "Colonnade: A Reconfigurable SRAM-Based Digital Bit-Serial Compute-In-Memory Macro for Processing Neural Networks." *IEEE Journal of Solid-State Circuits*, 56(7):2221-33, Mar 2021.

[6] Sridharan, Amitesh, et al. "A 1.23-ghz 16-kb programmable and generic processing-in-sram accelerator in 65nm." *ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC)*, pp. 153-156, Sep 2022.

[7] Jain, Saurabh, Longyang Lin, and Massimo Alioto."±CIM SRAM for signed in-memory broad-purpose computing from DSP to neural processing." *IEEE Journal of Solid-State Circuits*", 56(10):2981-92, Jul 2021.

[8] Chen, Zhiyu, et al."CAP-RAM: A charge-domain in-memory computing 6T-SRAM for accurate and precision-programmable CNN inference." *IEEE Journal of Solid-State Circuits*, 56(6):1924-35, May 2021.

[9] Chiu, Yen-Cheng, et al. "A 4-Kb 1-to-8-bit configurable 6T SRAM-based computation-in-memory unit-macro for CNN-based AI edge processors." *IEEE Journal of Solid-State Circuits*, 55(10):2790-801, Jul 2020.

[10] Xu, Siqiu, et al. "A high-precision implementation of the sigmoid activation function for computing-in-memory architecture." *Micromachines*, 12(10):1183, Sep 2021.

[11] Sze, Vivienne, et al. "How to evaluate deep neural network processors: Tops/w (alone) considered harmful." *IEEE Solid-State Circuits Magazine* , 12(3):28-41, Aug 2020.

[12] Sze, Vivienne, et al. "Efficient processing of deep neural networks: A tutorial and survey." *Proceedings of the IEEE*, 105(12):2295-329, Nov 2017.

[13] Liu, Weibo, et al. "A survey of deep neural network architectures and their applications." *Neurocomputing 234*, 234:11-26, Apr 2017.

[14] Rajput, Anil Kumar, Alok Kumar Tiwari, and Manisha Pattanaik. "An energy-efficient hybrid SRAM-based in-memory computing macro for artificial intelligence edge devices." *Circuits, Systems, and Signal Processing*, 42(6):3589-616, Jun 2023.

[15] Philippe, A., et al. "An Automated Design Methodology for Computational SRAM Dedicated to Highly Data-Centric Applications." *Proceedings of the 24th ACM/IEEE Workshop on System Level Interconnect Pathfinding*, pp. 1-7, Nov 2022.

[16] Liu, Yihe, et al. "Design and implementation of a charge-sharing in-memory-computing macro with sparse feature for quantized neural network." *Microelectronics Journal*, 154:106470, Dec 2024.

[17] Karam, Robert, et al. "Emerging trends in design and applications of memory-based computing and content-addressable memories." *Proceedings of the IEEE*, 103(8):1311-30, Jul 2015.

[18] Gebregiorgis, Anteneh, et al. "A survey on memory-centric computer architectures." *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(4):1-50, Oct 2022.

[19] Kim, Hyunjoon, et al. "A 1-16b reconfigurable 80Kb 7T SRAM-based digital near-memory computing macro for processing neural networks." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(4):1580-90, Jan 2023.

[20] Dhakad, Narendra Singh, et al. "R-inmac: 10T SRAM based reconfigurable and efficient in-memory advance computation for edge devices." *Analog Integrated Circuits and Signal Processing*, 116(3):161-84, Sep 2023.

[21] Raut, Gopal, Saurabh Karkun, and Santosh Kumar Vishvakarma. "An empirical approach to enhance performance for scalable cordic-based deep neural networks." *ACM Transactions on Reconfigurable Technology and Systems*, 16(3):1-32, Jun 2023.

[22] Eckert, Charles, et al. "Neural cache: Bit-serial in-cache acceleration of deep neural networks." *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 383-396, Jun 2018.

[23] Yu, Shimeng, et al. "Compute-in-memory chips for deep learning: Recent trends and prospects." *IEEE circuits and systems magazine*, 21(3):31-56, Aug 2021.