Compute In Memory Architecture Using SRAM For Edge AI

M.Tech. Thesis

By

SAGAR PATEL

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2024

Compute In Memory Architecture Using SRAM For Edge AI

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree
of
Master of Technology

by

SAGAR PATEL

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JUNE 2024

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Compute In Memory Architecture Using SRAM For Edge AI** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY - VLSI DESIGN AND NANOELECTRONICS** and submitted in the DEPART-MENT OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore, is an authentic record of my own work carried out during the time period from July 2022 to June 2024 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.

Sagar
23/05/24

Signature of the student with date
(Sagar Patel)

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

[illegible]
24/05/2024

Signature of the Supervisor of
M.Tech. thesis with Date
(Prof. Santosh Kumar Vishvakarma)

Sagar Patel has successfully given his/her M.Tech. Oral Examination held on **7th May 2024.**

[illegible]

Signature(s) of Supervisor(s) of M.Tech. thesis
Date: 24/05/2024

Convener, DPGC
Date:

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 and friends 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 and System (NSDCS) Lab research group, especially Mr. Narendra Dhakad, and Mr. Vikash Vishwakarma for their continuous support and guidance. I am also grateful to my labmates Mrs. Neha Maheshwari, Mr. Shashank Singh Rawat, Mr. Ankit Tenwar, Mr. Radheshyam Sharma, Ms. Komal Gupta, Mr. Akash Sankhe Mr. Mukul Lokhande and Mr. Sonu Kumar whose camaraderie and encouragement made my time at the institute truly memorable

Sagar Patel

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

Abstract

This work presents the design and implementation of a novel 10T SRAM cell aimed at enhancing the Static Noise Margin (SNM) compared to conventional 6T SRAM cells. The new 10T SRAM cell incorporates additional transistors to improve stability and reduce noise susceptibility, thereby ensuring more reliable data storage and retrieval. Leveraging this improved SNM, a compute-in-memory (CIM) architecture is developed, which is fully digital, utilizes bit-serial computing, and offers reconfigurability to accommodate various input and weight precisions from 1 to 16 bits. This reconfigurability enhances the architecture's versatility and efficiency in processing neural networks and other data-intensive tasks. The entire design, including the 10T SRAM cell and the bit-serial computation framework, is simulated using Cadence Virtuoso with TSMC 65nm technology. Detailed schematics are created, and test benches are configured to evaluate key parameters such as read/write delays, power consumption, and overall stability. Simulation results demonstrate significant improvements in SNM, reliability, and energy efficiency, making the architecture suitable for edge-computing applications. This novel 10T SRAM cell-based CIM architecture offers a robust, high-performance, and energy-efficient computing solution, addressing the limitations of traditional 6T SRAM cells and analog CIM implementations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

ADE	-	Analog Design Environment
CDF	-	Cadence Design Framework
CMOS	-	Complementary Metal-oxide-semiconductor
DRC	-	Design Rule Check
MAC	-	Multiply and accumulate
CIM	-	Processing in memory
CIM	-	Compute in memory
SNM	-	Static noise margine
LVS	-	Layout vs Schematic
MC	-	Monte Carlo
MOS	-	Metal-oxide-semiconductor field-effect transistor
NMOS	-	N-type Metal-oxide-semiconductor
PDK	-	Design Kit
PMOS	-	P-type Metal-oxide-semiconductor
Polo	-	Post Layout
Prelay	-	Pre Layout
PSRR	-	Power Supply Rejection Ratio
PVT	-	Process,Voltage,Temparature
SerDes	-	Serializer/Deserializer
SOC	-	System On Chip
XL	-	Layout Accelerator (layoutXL)
DL	-	Deep Learning
AI	-	Artificial Intelligence
DNN	-	Deep Neural Network
ANN	-	Artificial Neural Network
RNN	-	Recurrent Neural Network
CNN	-	Convolutional Neural Network
MAC	-	Multiply and Accumulate
MLP	-	Multi-Layer Perceptron
SF	:	Softmax Function
ReLU	-	Rectified Linear Unit
RTL	-	Register Transfer Level

Chapter 1. Introduction and Related Work

1.1 Overview

Edge AI refers to the deployment of artificial intelligence (AI) algorithms and models directly on edge devices, such as smartphones, IoT devices, and other embedded systems, rather than relying on centralized cloud-based computing. This approach allows data to be processed locally on the device where it is generated, providing several key benefits.

1.2 Hardware Accelerator for Edge AI

Hardware accelerators for edge AI are specialized processing units designed to perform AI computations more efficiently than general-purpose processors, particularly in edge computing environments. These accelerators, such as GPUs, TPUs, FPGAs, and custom ASICs, are optimized to handle the intensive computational requirements of AI tasks, such as deep learning and neural network inference, directly on edge devices like smartphones, IoT devices, and embedded systems. By integrating these accelerators, edge devices can process data locally, reducing the latency associated with sending data to the cloud for processing. This local processing capability is crucial for applications requiring real-time decision-making, such as autonomous vehicles, smart cameras, and industrial automation.

Edge AI hardware accelerators are designed to be energy-efficient, making them suitable for battery-powered devices where power consumption is a critical concern. They achieve this efficiency through specialized architectures that perform parallel processing and optimize data movement, significantly reducing the energy required per computation compared to traditional CPUs. Additionally, these accelerators often incorporate on-chip memory, further enhancing performance by minimizing the need for data transfer to and from external memory. The use of hardware accelerators in edge AI not only enhances performance and energy efficiency but also improves privacy and security by keeping sensitive data on the device. As AI applications continue to grow, the development and deployment of hardware accelerators will play a vital role in enabling advanced AI capabilities at the edge.

1.3 Neural Network

1.3.1 Introduction

A neural network is a computational model inspired by the way biological neural networks in the human brain process information. These models are a cornerstone of artificial intelligence (AI) and are designed to recognize patterns, make decisions, and solve complex problems by learning from data. Neural networks consist of layers of interconnected nodes, or neurons, each performing simple calculations that contribute to the overall function of the network.

The basic structure of a neural network includes an input layer, one or more hidden layers, and an output layer. Each neuron in a layer receives inputs from neurons in the previous layer, processes them using weighted connections, and passes the result through an activation function to produce an output. These outputs then serve as inputs to the neurons in the next layer. The process continues until the final output layer produces the network's result

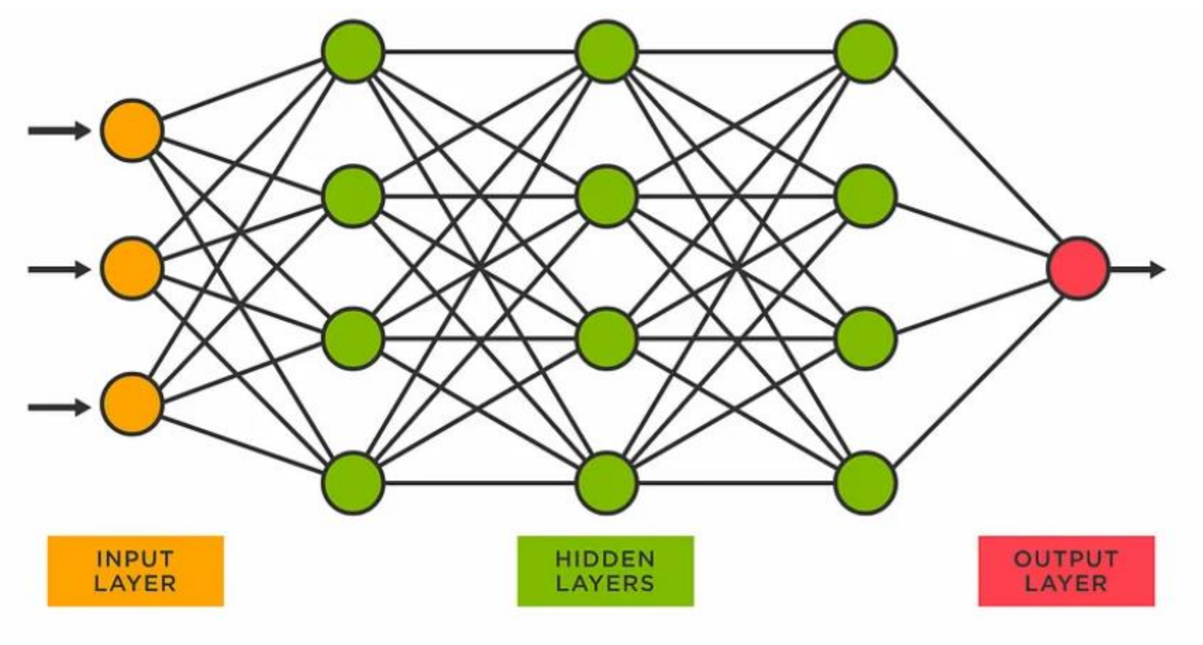


Fig. 1 Neural Network[21]

1.3.1 Neuron

A neuron, also known as a node or unit, is the fundamental building block of a neural network, modeled after the biological neurons found in the human brain. In an artificial neural network, a neuron performs several key functions. It begins by receiving multiple input signals, which can be raw data fed into the network or outputs from other neurons in the previous layer. Each input signal is associated with a weight, a numerical value representing the importance or strength of the input. These weights are initially set randomly and are adjusted during the training process. The neuron then computes a weighted sum of its inputs by multiplying each input by its corresponding weight and summing the results, often adding a bias term to the sum. This weighted sum is then passed through an activation function, which introduces non-linearity into the model, enabling the network to learn complex patterns and relationships in the data. Common activation functions include the sigmoid function, tanh function, and ReLU (Rectified Linear Unit). The result after applying the activation function is the output of the neuron, which can either be the final output of the neural network or serve as input to neurons in subsequent layers. Through this process of receiving inputs, weighting them, summing them, applying an activation function, and producing an output, neurons collectively transform input data into useful information, allowing the network to make predictions or decisions based on learned patterns.

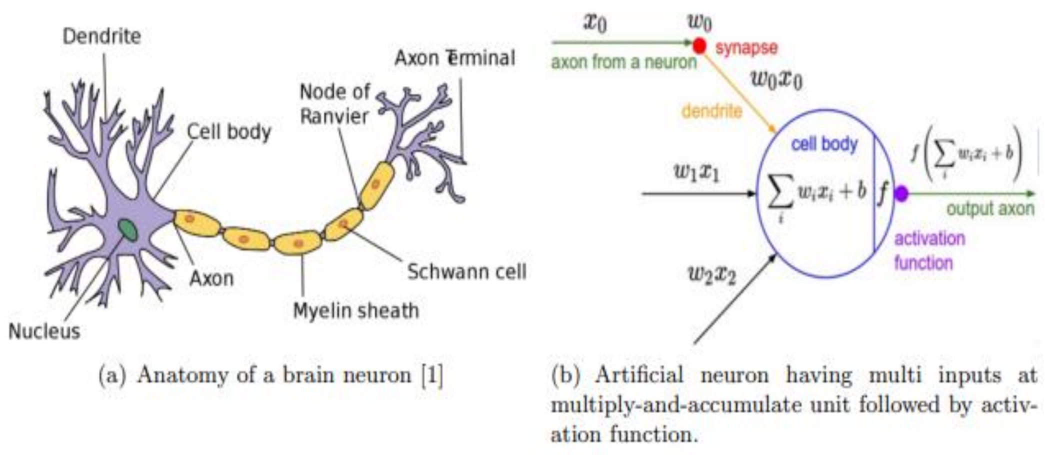


Fig. 2. Neuron Architecture[21]

1.3.2 MAC

In the context of neural networks, a multiply-and-accumulate (MAC) operation is a fundamental computational process. It involves multiplying pairs of input values and corresponding weights and then summing the results. This operation is crucial for neural network layers, particularly in convolutional and fully connected layers. During the forward pass of a neural network, each neuron receives multiple inputs, each of which is multiplied by a corresponding weight. The products of these multiplications are then accumulated to form a weighted sum. This sum is then passed through an activation function, which introduces non-linearity into the model, allowing it to learn and represent complex patterns in the data. The efficiency and speed of MAC operations significantly influence the overall performance of neural network training and inference, making them a critical aspect of hardware accelerators designed for deep learning tasks.

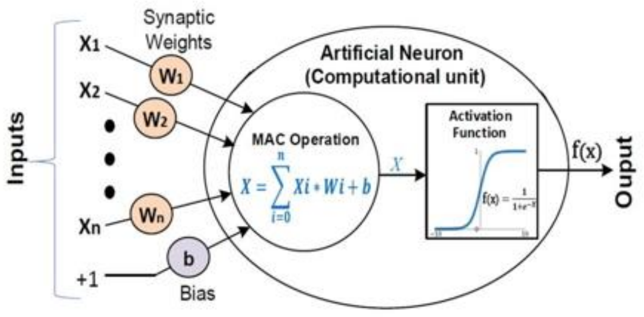

Inputs
X1
X2
Xn
+1
Synaptic Weights
W1
W2
Wn
b
Bias
Artificial Neuron (Computational unit)
MAC Operation
$X = \sum_{i=0}^{n} Xi * Wi + b$
X
Activation Function
$f(x) = \frac{1}{1+e^{-x}}$
f(x)
Ouput

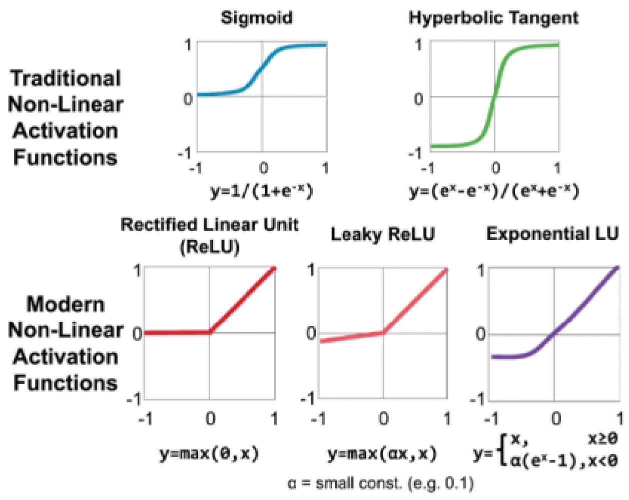
Fig. 3 Processing Elements of Neuron[9]

1.3.3 Activation Function

An activation function in a neural network is a crucial component that determines whether a neuron should be activated or not, effectively deciding whether the neuron's output should be passed to the next layer in the network. It introduces non-linearity into the model, allowing neural networks to learn and model complex data patterns. Without activation functions, a neural network would simply perform linear transformations, limiting its capacity to capture intricate relationships in the data.

There are several types of activation functions commonly used in neural networks, each with its unique characteristics and applications:

- **Sigmoid Function:** This function maps the input values to a range between 0 and 1, making it useful for binary classification tasks. However, it can suffer from vanishing gradient problems, where gradients become too small for effective learning in deeper networks.
- **Tanh Function:** Similar to the sigmoid function, the tanh function maps inputs to a range between -1 and 1. It is often preferred over the sigmoid function because its output is zero-centered, which can lead to faster convergence during training.
- **ReLU (Rectified Linear Unit):** ReLU is one of the most widely used activation functions in deep learning. It outputs the input directly if it is positive; otherwise, it returns zero. This simplicity helps in efficient computation and mitigates the vanishing gradient problem, although it can suffer from the "dying ReLU" problem, where neurons can sometimes become inactive permanently.
- **Leaky ReLU and Parametric ReLU:** These are variants of the ReLU function designed to address the "dying ReLU" issue. They allow a small, non-zero gradient when the input is negative, ensuring that neurons do not die.
- **Softmax Function**: Typically used in the output layer of a classification network, the softmax function converts the raw output scores into probabilities, which sum to one. This is particularly useful for multi-class classification problems



.

Fig. 4 Activation Functions[5]

1.4 Motivation and Problem Statement

2.3.1 Von Neumann Bottleneck

The Von Neumann architecture, proposed by John von Neumann in 1945, consists of a compute unit that executes user programs and a memory unit that stores both these programs and the necessary data. This architecture is foundational to most modern computer systems, including CPUs and GPUs. Over the years, improvements in technology have significantly enhanced compute performance, largely following Moore's Law, which predicts the doubling of transistors on a chip approximately every 18 months. However, while compute performance has seen rapid advancement, memory development has primarily focused on increasing capacity rather than performance. This growing disparity between compute and memory performance, often referred to as the "memory wall," has become a significant bottleneck in modern systems. The von Neumann bottleneck arises from the inherent separation between the compute unit and memory, leading to limitations in memory bandwidth and issues with data movement.

2.3.2 Latest AI Accelerators With High-Bandwidth Memories

The von Neumann bottleneck has been addressed by implementing a hierarchical memory structure. Processors now incorporate small but extremely fast on-chip SRAM caches to take advantage of temporal and spatial locality. Beyond the processor chip lies the main memory, which uses DRAM; it is faster and has a larger capacity than SRAM. Following the main memory, systems use solid-state drives (SSDs) for even greater storage capacity. However, as deep neural network (DNN) models grow in size to the terabyte range, machine learning workloads demand even higher bandwidth between the processor and main memory. Compounding this issue is the difficulty in further scaling process technology below the 10nm node, which signals the end of Moore’s Law.

To tackle both the von Neumann bottleneck and the challenges of process scaling, companies are developing array-type architectures to accelerate data-intensive machine learning tasks. They are also adopting high-bandwidth memory (HBM), a 3D-stacked DRAM technology, to enhance bandwidth between computing and memory units. For instance, Google has created the TPU for more cost-effective and energy-efficient inference and training in data centers. Intel has introduced the NNP-T and Habana Labs' Gaudi processors for training workloads. Start-ups like Graphcore and Groq are also utilizing this architecture. Despite the improvements brought by AI accelerators with HBM technology, which can achieve bandwidths up to several terabytes per second, these solutions still adhere to the von Neumann architecture and face issues such as high power dissipation and limited capacity.

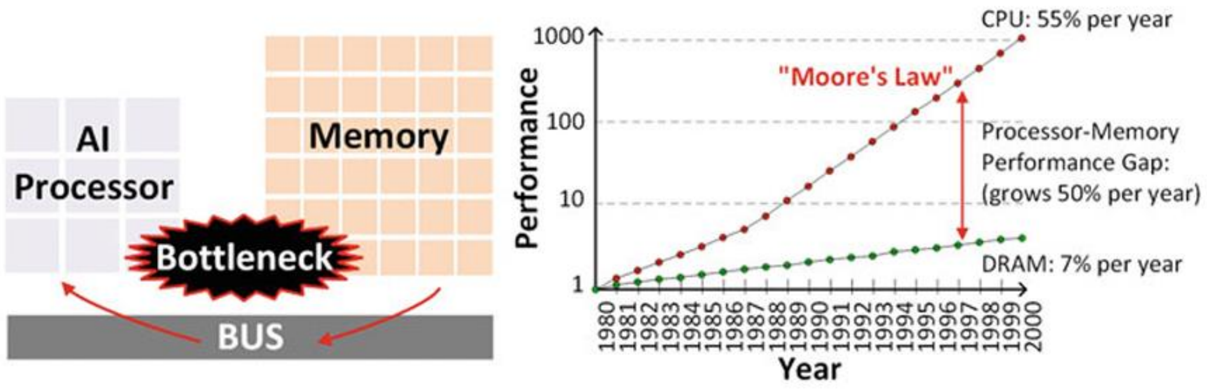


Fig. 5 von Neumann Bottleneck and memory wali problem[8]

1.5 Compute-In-Memory Architecture

In a Compute-In-Memory (CIM) architecture, instead of transferring data from the memory unit to the compute unit, data remains in the memory while integrated logic performs computations directly in place. This represents a significant departure from traditional computer architectu re. Whereas conventional and near-memory architectures utilize the same memory hierarchy to efficiently manage external memory bandwidth, CIM combines computing and memory units to eliminate issues related to external data movement. This shift from a compute-centric to a memory-centric or hybrid architecture is gaining considerable attention as a solution to the von Neumann bottleneck, particularly for data-intensive applications such as AI and machine learning. Besides enhancing performance, CIM architecture also significantly reduces energy consumption by replacing costly external data transfers with more efficient on-chip data movements.

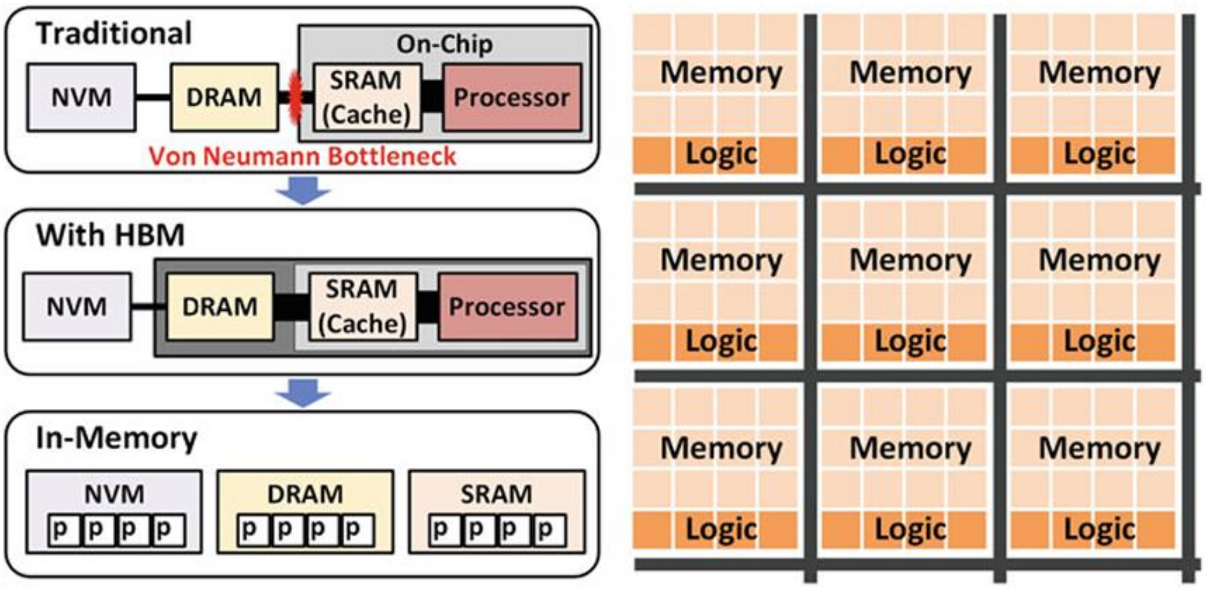


Fig. 6 Processing in Memory Achitecture[7]

1.6 Challenges of CIM Architecture

While CIM architecture holds promise, it faces several significant challenges due to the need to integrate logic units within the memory module. Three primary challenges in CIM design are accessibility to the manufacturing process, designing within physical constraints, and developing a compatible software stack for usability.

Among various memory technologies, SRAM is unique in that it can be built using commercially available logic processes. This is why many CIM prototypes utilize SRAM—it can be fabricated with a logic process and easily customized in both the memory cell and peripheral circuits. Additionally, the larger cell size of SRAM minimizes area constraints for integrating logic. However, processes for DRAM and non-volatile memory (NVM) are less accessible. Memory manufacturers like Samsung, SK Hynix, and Micron have proprietary processes that are not available to external developers. Without access to these process design kits (PDKs), researchers struggle to even simulate basic circuits. Although there have been numerous CIM architecture proposals for DRAM, these designs are typically evaluated through performance simulations rather than physical designs. The significant differences between DRAM and logic processes, which focus on maximizing cell capacity and density, make it difficult to confirm the feasibility of these CIM architectures solely through simulations.

In CIM design, chip designers must carefully select which functions to integrate into the memory, as the silicon area is limited. Implementing a wide range of functions or overly generic logic is impractical due to these area constraints. Additionally, integrating logic reduces the available memory capacity. The logic design must also be physically aligned with the memory cell design to maximize internal bandwidth, presenting another layer of complexity in the design process.

. The final challenge in CIM design lies within the software stack. For CIM to achieve widespread adoption as a new technology, it is crucial to address this aspect comprehensively. Unlike traditional memory devices, CIM is not merely a passive component; it can perform logic operations simultaneously with memory functions. This necessitates a fundamental shift in the software domain as well. To fully optimize CIM systems, the entire software stack must be re-evaluated, including programming languages, compilers, drivers, and runtime environments. Without such changes, CIM will struggle to surpass the performance and usability of existing von Neumann architectures.

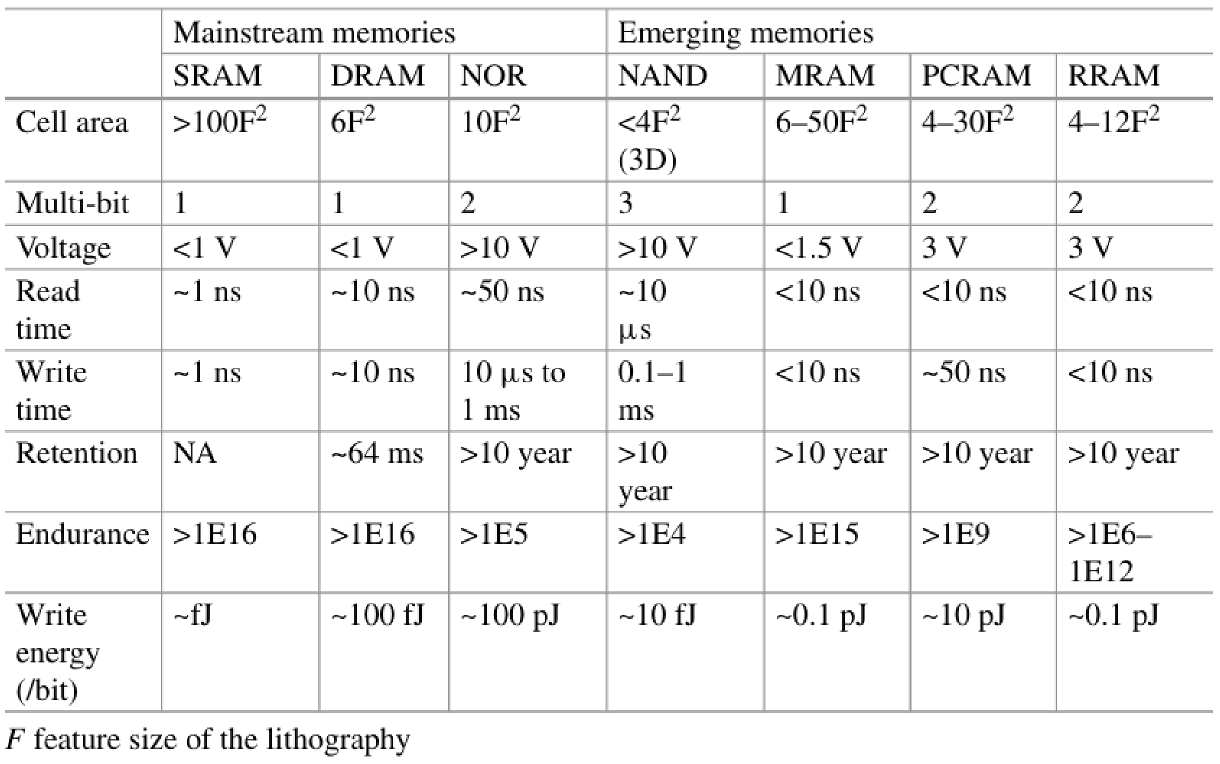
Chapter 2. Theoretical Framework

2.1 Basic Memory Operation

Dynamic random-access memory (DRAM) and static random-access memory (SRAM) have been essential components in modern VLSI systems. Advances in semiconductor technology have led to increased memory density and enhanced computing power, driving progress in electronic systems. However, as semiconductor technology continues to scale down, DRAM and SRAM face several design challenges, including increased leakage currents and reduced sensing margins. Extensive research and development efforts have been made to address these issues and meet the market's demand for high-performance, low-power memory solutions.

In the realm of mobile computing devices, there is a significant demand for nonvolatile memory solutions that can retain key data even without a power supply. FLASH memory has seen substantial development due to the rapid growth of mobile electronics. However, FLASH is primarily used for storage rather than computing purposes. It is well-known that FLASH memory has a rewrite endurance of around 10^6 cycles, which is considerably lower than that of SRAM and DRAM. Additionally, FLASH lags behind DRAM and SRAM in terms of write speed and power consumption. Although various technologies have been developed to enhance the endurance and reduce the write power consumption of FLASH memory, no breakthrough has been achieved to make FLASH memory comparable to DRAM and SRAM in these aspects.

Table 1 Device characteristics of mainstream and emerging memory technologies[3]



	Mainstream memories			Emerging memories			
	SRAM	DRAM	NOR	NAND	MRAM	PCRAM	RRAM
Cell area	>100F^2	6F^2	10F^2	<4F^2 (3D)	6–50F^2	4–30F^2	4–12F^2
Multi-bit	1	1	2	3	1	2	2
Voltage	<1 V	<1 V	>10 V	>10 V	<1.5 V	3 V	3 V
Read time	~1 ns	~10 ns	~50 ns	~10 µs	<10 ns	<10 ns	<10 ns
Write time	~1 ns	~10 ns	10 µs to 1 ms	0.1–1 ms	<10 ns	~50 ns	<10 ns
Retention	NA	~64 ms	>10 year	>10 year	>10 year	>10 year	>10 year
Endurance	>1E16	>1E16	>1E5	>1E4	>1E15	>1E9	>1E6–1E12
Write energy (/bit)	~fJ	~100 fJ	~100 pJ	~10 fJ	~0.1 pJ	~10 pJ	~0.1 pJ

F feature size of the lithography

Recently, various resistive nonvolatile memory devices, such as magnetic RAM (MRAM), ferroelectric RAM (FeRAM), phase change RAM (PCRAM), and resistive RAM (ReRAM), have been introduced. Although these memory technologies operate based on different physical mechanisms, they all function by utilizing two distinct resistance values. Among these, ReRAM has attracted significant interest due to its simple structure and compatibility with CMOS technology. Additionally, ReRAM is more reliable, faster, and consumes less power than FLASH memory. While ReRAM's endurance is still lower than that of DRAM and SRAM, it is suitable for mobile applications that require non-volatility and moderate computing power.

2.2 SRAM Basics

Static random-access memory (SRAM) is commonly used as an embedded memory solution in computing systems due to its high performance, robustness, and cost-effectiveness. SRAM is faster than DRAM because the cross-coupled inverters in SRAM cells produce quicker and larger voltage swings on the bitlines. Additionally, SRAM can simultaneously receive row and column addresses, unlike DRAM, which processes these addresses separately using the same address pins. As a result, SRAM exhibits lower latency compared to DRAM. The cross-coupled inverters in SRAM cells also automatically maintain stored data when the wordlines are turned off, eliminating the need for refresh and write-back operations.

Another significant advantage of SRAM is its full compatibility with CMOS process technology, which facilitates easy integration with computing blocks. However, as CMOS technology continues to scale, SRAM faces several challenges, including reduced stability margins, increased leakage currents, and difficulties in supply voltage scaling. Various design techniques have been developed to address these issues and maintain SRAM's performance and reliability.

2.2.1 Working of SRAM

The conventional 6T SRAM cell consists of six transistors: two cross-coupled inverters and two access transistors. During a write operation, data is first loaded onto the bitline pair, and then the wordline is activated. The data on the bitline pair is transferred to the SRAM cell nodes through the access transistors. For instance, if the bitline holds a "0" and the complementary bitline holds a "1," the node Q will be pulled low through the access transistor, and the complementary node QB will be pulled high. Consequently, the SRAM cell stores Q = "0." The write operation is primarily constrained by the difficulty of writing a "0" because NMOS access transistors can pass low voltage more effectively than high voltage. Therefore, the access transistors need to be stronger than the PMOS transistors to ensure that Q is lowered below the trip point of the inverters in the SRAM cell.

During a read operation, the wordline is activated after precharging the bitline pairs. One of the differential bitlines will decrease depending on the data stored in the SRAM cell. For example, if Q is "0," the bitline will decrease while the complementary bitline remains at the supply voltage (VDD). A sense amplifier then amplifies the differential voltage between the bitlines to generate the output signal.

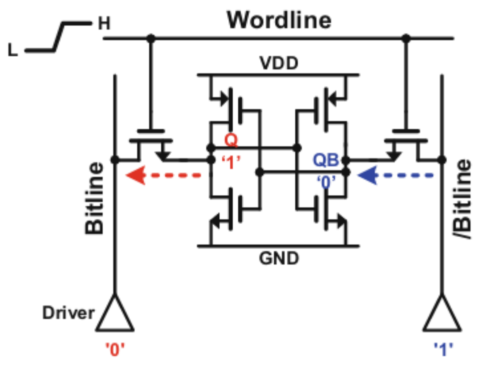


Fig. 7 SRAM cell Operation write[15]

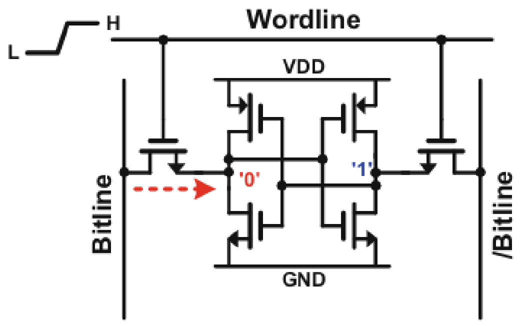


Fig. 8 SRAM cell Operation read[15]

Below figure illustrate a typical SRAM architecture, which includes an array of cells, row decoding, column multiplexing, sense amplifiers, write drivers, and a controller. During a read operation, the accessed cell generates a differential voltage across a bitline pair. This differential voltage is routed to a sense amplifier through a column multiplexer. Unlike DRAM, SRAM uses sense amplifiers that are shared among multiple columns, so only one column is connected to a sense amplifier at a time for signal amplification. There is no need for a write-back operation in the unselected columns since SRAM cells can regenerate stored data via the cross-coupled inverters.

During a write operation, write drivers send data to the selected bitlines through the column multiplexer. However, the access transistors in the unselected columns remain active, which can lead to unintended write operations. To prevent this, the bitlines of the unselected columns are precharged to the supply voltage (VDD), ensuring that SRAM cells in the selected row but unselected columns undergo a read operation instead. This approach helps maintain the integrity of the data stored in the unselected columns.

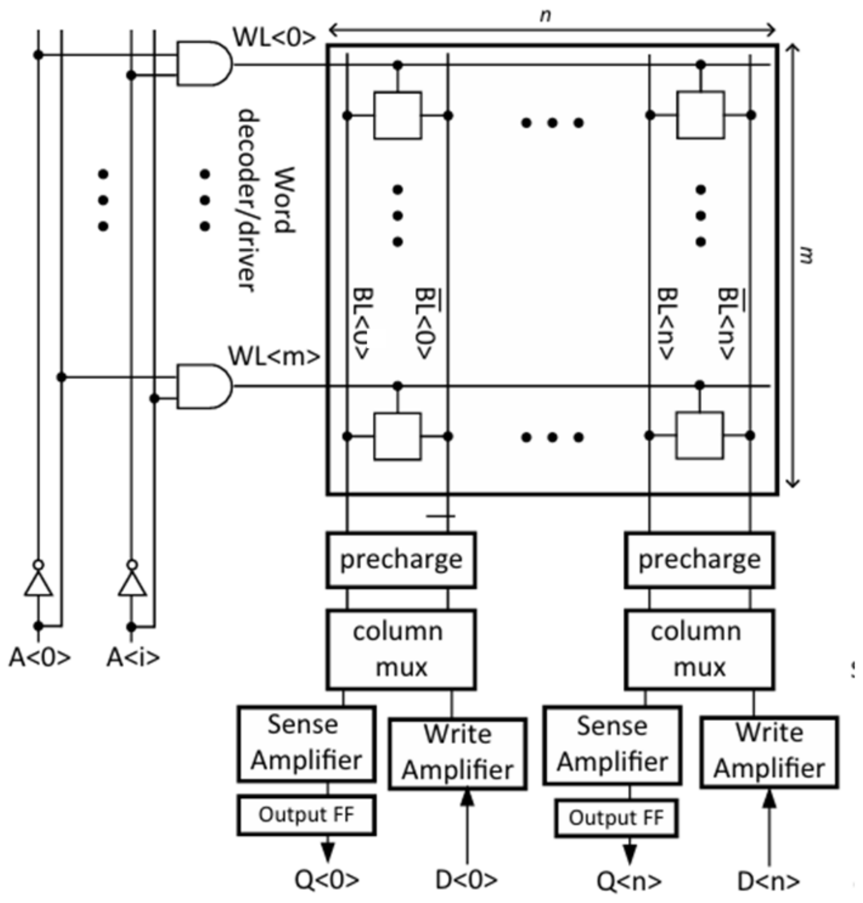


Fig. 9 SRAM Array[22]

2.3 CIM Fundamentals

Processing-in-memory (CIM) has recently emerged as a promising alternative computer architecture for energy-efficient processing, particularly for massively parallel arithmetic operations required in artificial intelligence and machine learning applications. This architecture is especially well-suited for battery-operated edge computing devices. One of the critical operations in CIM is the multiply-and-accumulate (MAC), which is essential for processing artificial neural networks like convolutional neural networks (CNNs). CNNs often require billions of MAC operations to process a single image classification, highlighting the importance of efficient MAC operation units in hardware accelerators based on CIM architecture.

Several emerging memory devices, such as resistive random-access memory (ReRAM), magneto resistive RAM (MRAM), and phase-change RAM (PCRAM), are considered for CIM implementation. These devices are compact and can provide high storage capacity along with massively parallel MAC operations. However, despite their potential, these emerging memories are not yet mature and are relatively costly. As a result, traditional static RAM (SRAM) is still the most commonly used memory technology in CIM implementations. Despite having a larger bitcell size compared to emerging memories, SRAM offers advantages such as scalability, compatibility with logic design, cost-effectiveness, and reliability, making it a preferred choice for CIM implementations.

2.3.1 Implementation Of CIM Macro

The CIM macro can be implemented by utilizing a classical two-dimensional array composed of different types of memory cells (such as SRAM, DRAM, or ReRAM), as depicted below figure For instance, the operation of a standard SRAM in the context of MAC (multiply and accumulate) is described here in detail. In this setup, a standard six-transistor (6T) SRAM cell serves as a binary CIM unit for executing MAC operations. Each macro row receives a binary input (either 0 or +1), which serves as the multiplicand for all SRAM cells in the same row. The binary weight (either -1 or +1) is stored in an SRAM cell and is multiplied by the input applied to its wordline (WL). Accumulation is carried out column by column, with the accumulated result being a voltage difference between a bitline (BL) and its complement (BLb).

In this configuration, a '0' input results in a WL high voltage, creating a discharging path from BL (or BLb) to the ground via an SRAM internal node Q (or Qb). Prior to the MAC operation, all bitlines (BLs and BLbs) are precharged to a high voltage. Ideally, inputs are applied and outputs are generated in parallel, enabling massively parallel binary SRAM CIM operations that maximize throughput and minimize latency. However, the actual performance of the designed CIM macro is heavily influenced by the essential data conversions for input (digital-to-analog) and output (analog-to-digital). These conversions ultimately determine the overall performance of the implemented CIM macro.

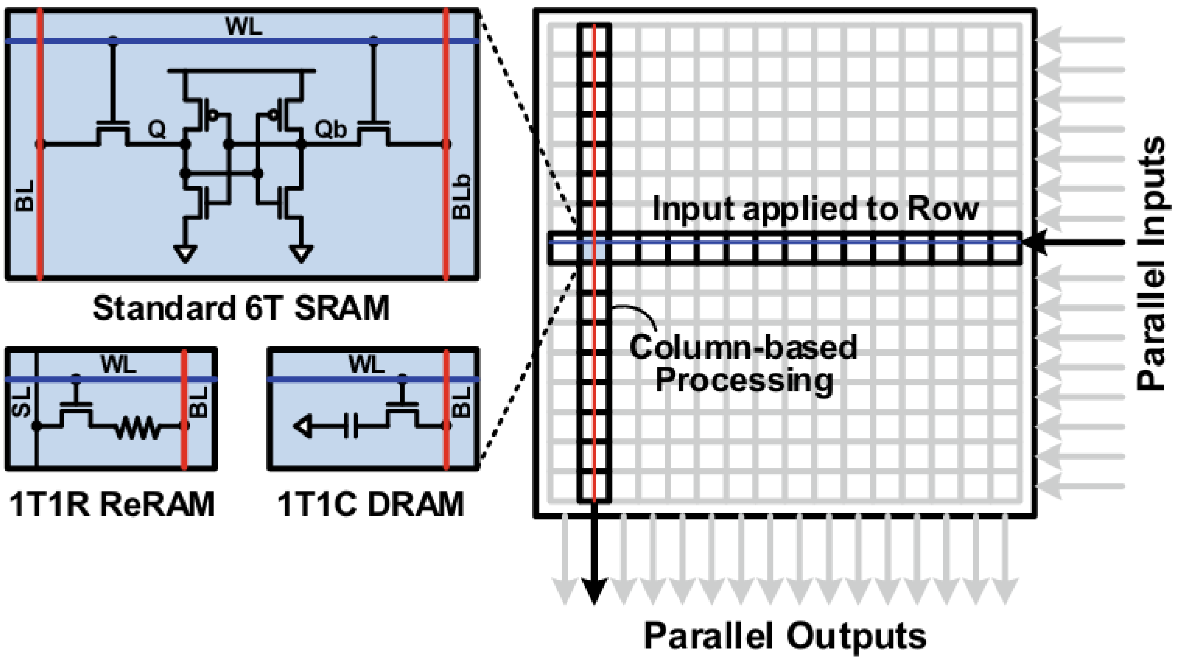


Fig. 10 CIM macro using common memory cells[6]

2.3.2 CIM Macro Mapping For DNN

The CIM macro depicted in Figure 13 can be utilized to map and process the fundamental arithmetic operations of a fully connected layer within deep neural networks (DNNs). Figure 11 illustrates a pair of binary inputs and weights mapped to an SRAM cell and an input pair, respectively. Binary multiplication is executed within the SRAM cell, resulting in a unit analog accumulation represented as a voltage difference across the vertical bitlines (BL and BLb).

A collection of input and weight pairs constitutes a dot-product, as depicted in the left side of Figure 12. This dot-product is then mapped to a column of the CIM macro, as shown on the right side of Figure 12. The unit voltage differences originating from SRAM cells accumulate within the column, which shares a pair of BL and BLb. Finally, a vector-matrix multiplication (i.e., the fully connected layer itself) is mapped to the entire CIM macro. In this configuration, all multiplications and accumulations are executed in parallel, enabling extensive parallelism, as illustrated in Figure 13.

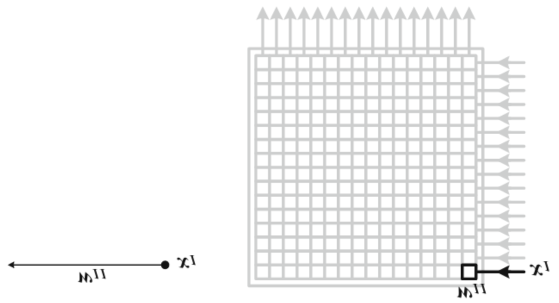


Fig. 11 Processing a fully connected layer using SRAM based macro multiplication[11]

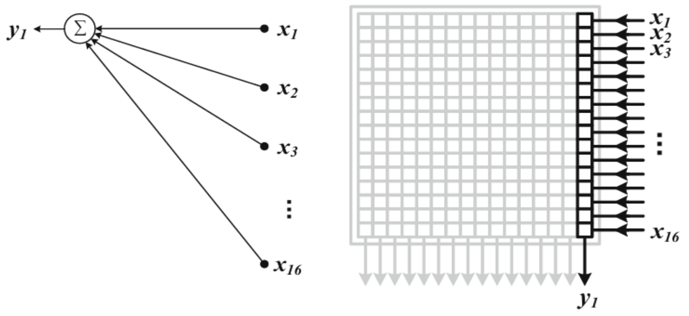


Fig. 12 Processing a fully connected layer using SRAM based macro dot-product[11]

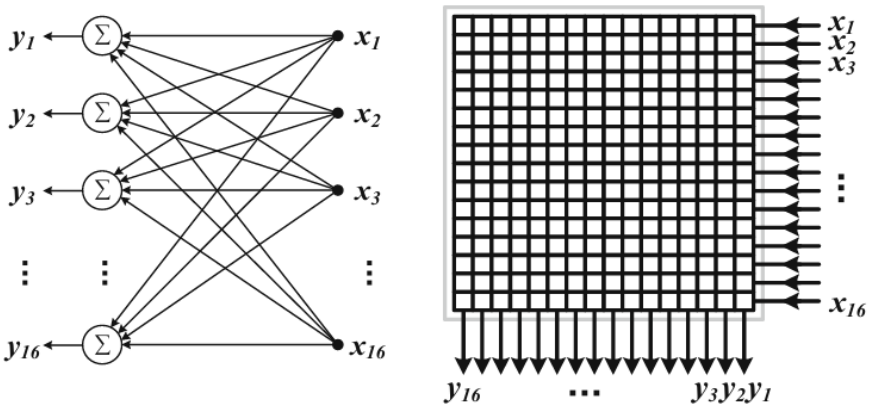


Fig. 13 Processing a fully connected layer using SRAM based macro vector matrix multiplication[11]

2.3.3 CIM Macro Mapping For CNN

The CIM macro can be utilized to assign a convolutional layer by unrolling and mapping high-dimensional filter weights and input feature maps into the macro. Figure 14 illustrates the mapping of an input and weight pair from a convolutional layer configuration into the macro. The two-dimensional (2D) filter weights are unrolled and mapped into a column of four bitcells, as shown in Figure 15. For three-dimensional (3D) filter weights and input feature maps, which consist of multiple channels of 2D filters and input feature maps, the entire column of the CIM macro is utilized for mapping, as depicted in Figure 16. If the number of filter and input feature map element pairs exceeds the number of bitcells in a single macro column, the 3D filter and input feature map can be mapped to multiple macro columns.

Expanding on the convolutional layer processing, Figure 17 introduces another dimension (output channels or the channels of 3D filters) that can be processed in parallel using multiple columns in the CIM macro. Each column output corresponds to a pixel of each 2D output feature map. To generate the complete 3D output feature map, the same CIM macro is reused while sliding the window of the input feature map to process and complete the 3D output feature map, as illustrated in Figure 18

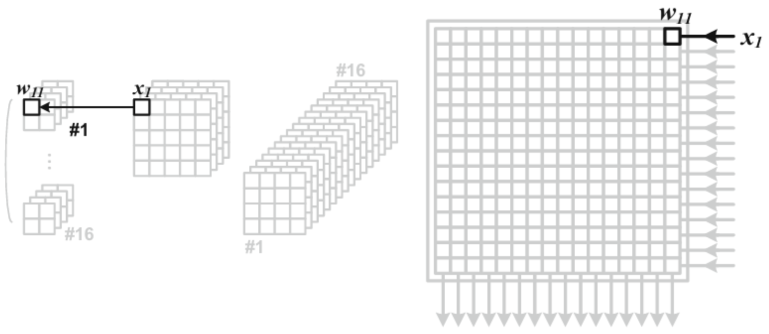


Fig. 14 Processing a fully convolutional layer using SRAM based CIM macro multiplication[11]

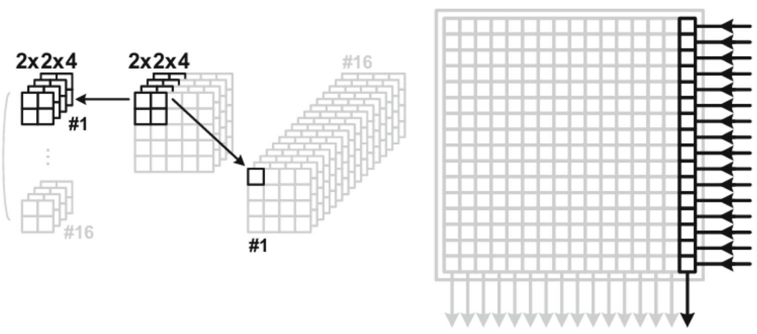


Fig. 15 Processing a fully convolutional layer using SRAM based CIM macro a dot product for 2D filter[11]

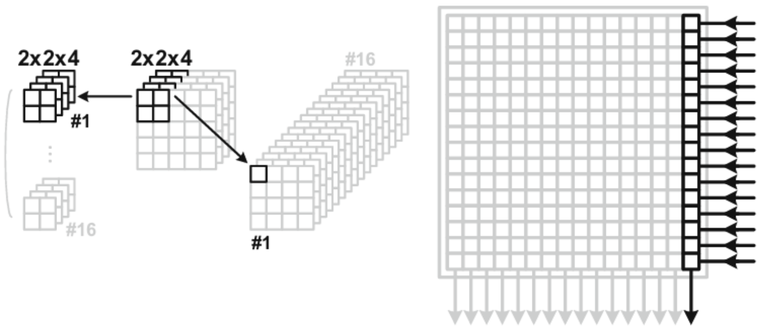


Fig. 16 Processing a fully convolutional layer using SRAM based CIM macro a dot product for 3D filter[11]

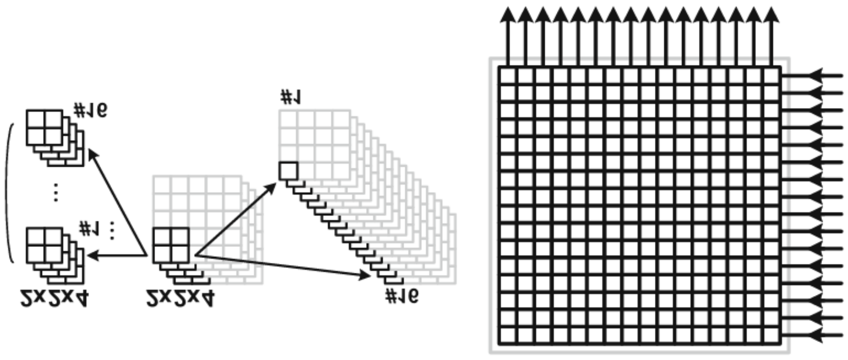


Fig. 17 Processing a fully convolutional layer using SRAM based CIM macro a vector matrix for 4D filter[11]

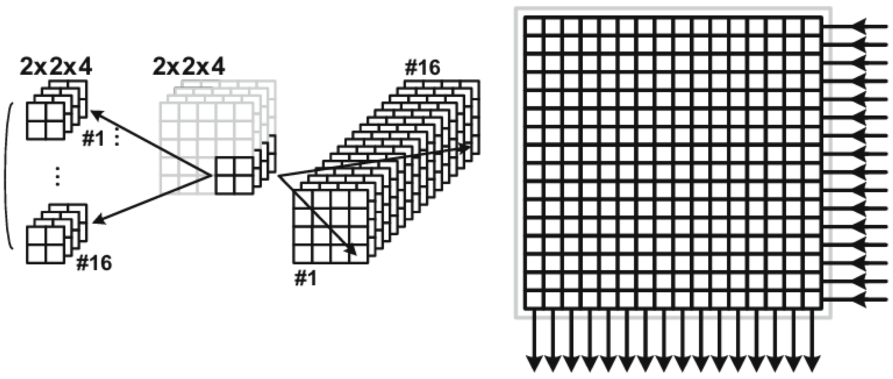


Fig. 18 Processing a fully convolutional layer using SRAM based CIM macro after 16 cycles of vector matrix operation[11]

2.4 CIM Design Challenges

While analog CIM macros offer exceptional efficiency, they also face significant design challenges. The most notable issues include computation nonlinearity caused by process, temperature, and voltage (PVT) variations, as well as the overhead associated with DAC/ADC conversions.

Figure 19 illustrates the input offset error in analog circuits within the CIM macro, such as bitcells, sense amplifiers (SA), and ADCs, which is induced by process variations. The left side of Figure 19 shows the error distribution of the output ADC code for identical MAC operations. Despite the regular structure of memory bitcells in the CIM array, differences in MAC results arise due to process variations during the fabrication of these cells. The right side of Figure 19 depicts the variation in both a single bitcell and an entire column. The top right section shows the distribution of discharge current when a bitcell processes a multiplication operation using current discharge, while the bottom right section displays the bitline (BL) voltage allocation after completing the dot-product operation in a column-based neuron.

Process variation leads to fluctuations in the bitline voltage representing the dot-product result, increasing the likelihood of producing incorrect output ADC codes. In the context of neural networks, this incorrect output code becomes the new input activation for the next layer and is used to calculate subsequent dot-products. Consequently, errors in one layer's output can propagate through multiple computations, ultimately leading to classification errors and reducing the accuracy of the application.

Computation nonlinearity occurs when multiple rows are activated in parallel to enhance computational efficiency, as depicted in Figure 20. When more "1"s are added in the column, the bitline voltage representing dot-product results decreases, leading to a dynamic range limit. If the bitline voltage drops too low, the accumulation linearity is substantially degraded, as indicated by the red dotted line on the right side of Figure 20.

The overhead associated with digital-to-analog and analog-to-digital converters (DAC/ADC) for data transmission is a major concern for CIM macros. As illustrated in Figure 21, DAC/ADC circuits consume a substantial amount of area and energy, and they also increase the latency of the neural network accelerator. Furthermore, typical ADCs have fixed bit precision, which limits the system's reconfigurability.

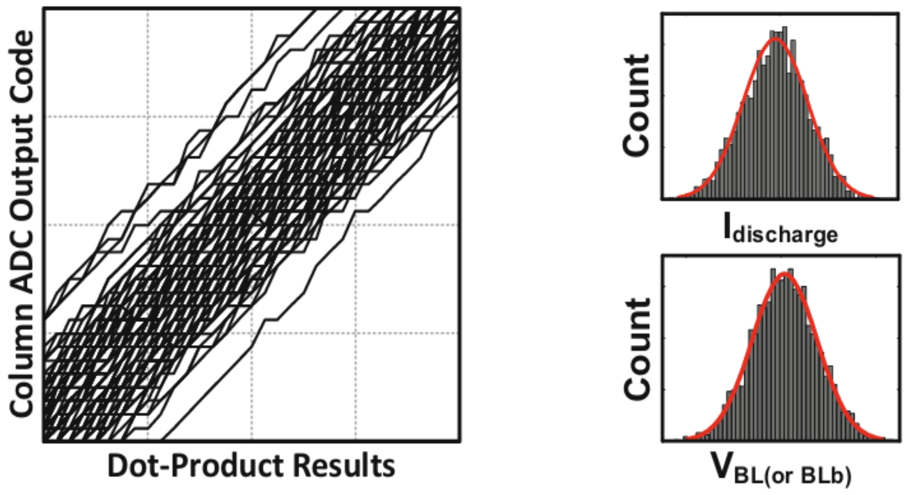


Fig. 19 Challenges of analog CIM macro process variation[3]

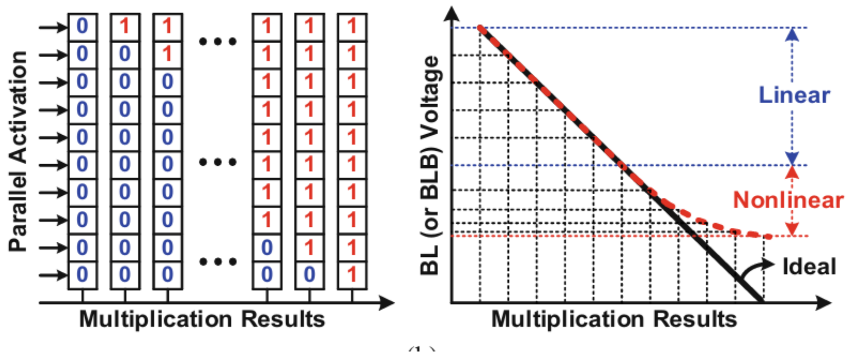
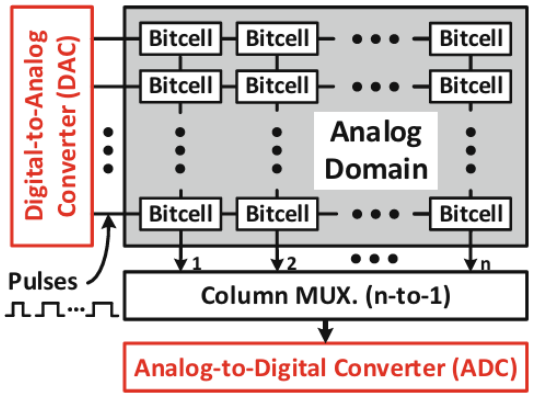


Fig. 20 Challenges of analog CIM macro non linearity[3]



Digital-to-Analog Converter (DAC)
Bitcell
Analog Domain
Pulses
Column MUX. (n-to-1)
Analog-to-Digital Converter (ADC)

Fig. 21 Challenges of analog CIM macro ADC overhead[3]

Digital CIM macros face their own set of critical issues, namely low area efficiency and high power consumption. Figure 2.20a illustrates a modern neural network accelerator that includes a complete array of digital processing elements (PEs) designed to handle massive MAC operations synchronously. By utilizing a hierarchical memory system and data reuse strategies, this design enhances computational efficiency and reduces energy consumption, as memory access energy typically exceeds the energy used in MAC operations.

Figure 22 shows a CIM column equipped with a parallel adder tree that performs massively parallel accumulation operations without needing additional registers to store input activations and partial sums. This setup improves energy efficiency through bit-serial multiplication, albeit at the cost of increased operation latency. The digital approach avoids the compute nonlinearity and poor scaling associated with analog circuits. However, fully digital PEs require more arithmetic circuits, leading to larger area occupancy and higher static and dynamic energy consumption compared to the bitcells in analog CIM.

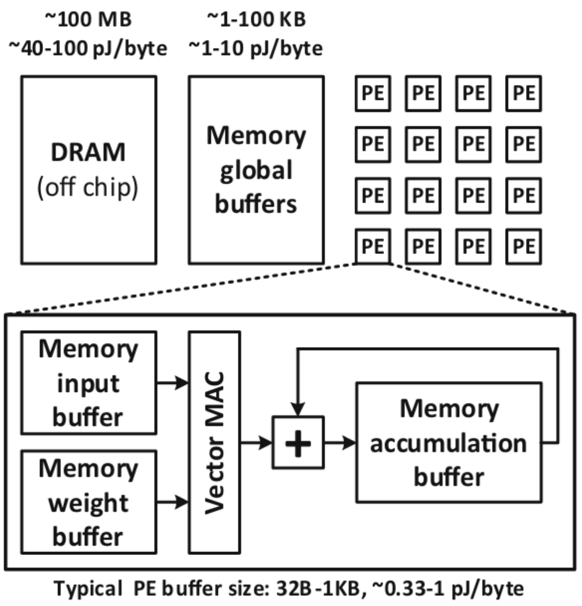


Fig. 22 Simplified block diagram of a typical digital DNN accelerator[5]

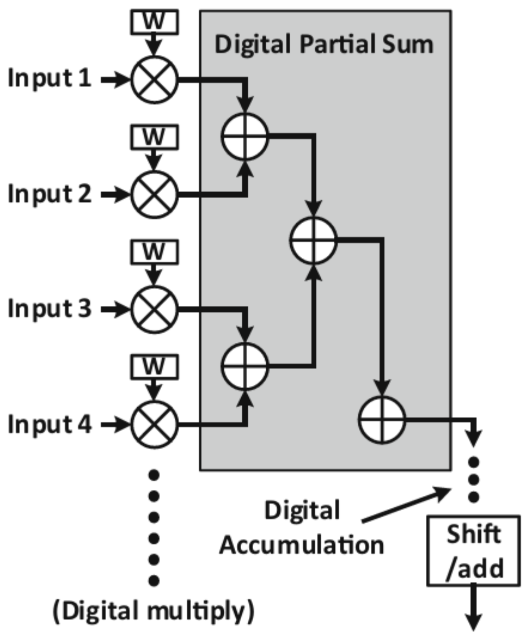


Fig. 23 A column based dot-product circuit using digital CIM[5]

Chapter 3. Proposed Architecture

3.1 Introduction

Previous digital hardware accelerators tend to use a significant portion of their total energy accessing OFF-chip memory, despite efforts to minimize data movement. Nonetheless, digital accelerators offer several advantages over their analog counterparts. One key benefit is their robustness; digital designs are less sensitive to process variations and various noise sources due to their core computation mechanism, which includes an abstraction layer over the analog signal values. Another significant advantage is that digital accelerators eliminate the need for data conversions, leading to reductions in both energy and area consumption. This not only improves energy and area efficiency but also simplifies computation and alleviates performance bottlenecks caused by time-multiplexed operations. In this section, we will explore how we leverage the aforementioned benefits of digital accelerators to minimize memory access.

3.2 Weight-Stationary Systolic Architecture

A systolic array of processing elements (PEs) is an efficient data processing architecture known for achieving high throughput due to its parallel computations and natural input-output data flow. In this setup, both inputs and partial sums move through a 2-D array while pipelined parallel multiply-accumulate (MAC) operations are carried out by the distributed digital PEs. Similarly, a 2-D PE array is used in the analog compute-in-memory (CIM) macro. However, unlike the digital version, all operations in the analog CIM macro occur in parallel without pipelining because of its smaller size and faster computation speed. The analog macro performs parallel MAC operations and also stores weights within its memory array (i.e., it is weight-stationary).

The proposed digital CIM macro utilizes a weight-stationary systolic architecture, as depicted in Fig. 24. This design merges the high throughput of the systolic array with the low latency and high energy efficiency of the CIM macro. Additionally, the CIM macro can handle input and weight precisions ranging from 1 to 16 bits. The proposed architecture is a digital bitcell array that adopts the bit-parallel systolic PE array's operation directionality. It functions as a precision reconfigurable bit-serial digital CIM macro, offering significant area savings compared to the traditional systolic PE array.

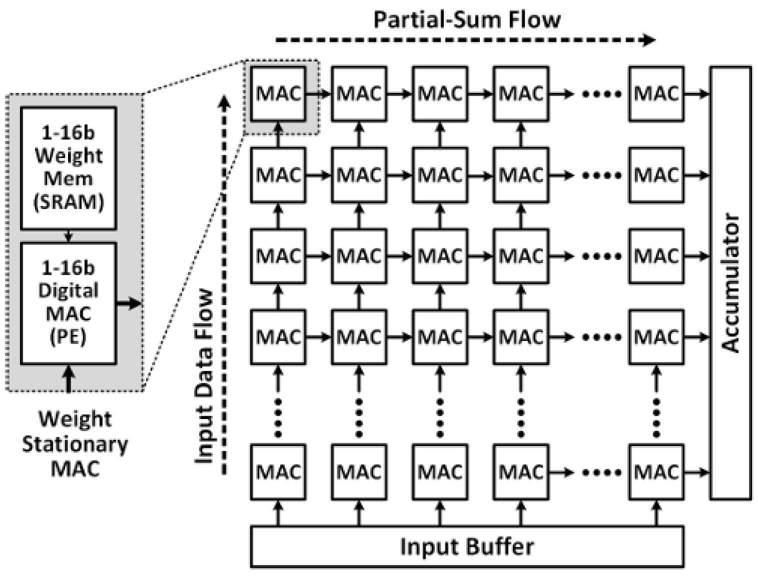


Fig. 24 Weight-Stationary Systolic Architecture of the proposed digital CIM macro with 1-16 bit reconfigurable MAC precision[22]

3.3 Bit-Serial Computing and Reconfigurability

Figure 25 illustrates a conventional bit-parallel ALU, which includes two 4-bit multipliers and an 8-bit adder. In this setup, pairs of 4-bit inputs are multiplied, and the resulting 8-bit outputs are subsequently added. It's important to note that the power and area requirements of bit-parallel digital multipliers increase quadratically with input precision. To address this issue and reduce the MAC area, Stripes [19] introduces bit-serial computing, which significantly saves area by serializing one of the multibit inputs and replacing the large multiplier circuit with more compact bitwise ALUs, as shown in Figure 6(b). The area savings of the bit-serial ALU circuit increase with precision since the complexity of bit-serial computation grows linearly with bit precision. However, bit-serial computing requires an additional circuit to accumulate partial sums from each operation cycle. Inputs are serialized from the least significant bit (LSB) to the most significant bit (MSB), with each bit generating a partial sum. Another limitation of conventional bit-parallel ALUs is their lack of reconfigurability. 25 shows BitFusion [20], which consists of reconfigurable ALU units that can handle fine-grained bit precision. These low-precision ALU units can be grouped to function as a higher-precision ALU.

In addition to minimizing OFF-chip memory access and preserving the benefits of digital architecture, the proposed digital CIM macro also reduces area consumption by adopting the bit-serial ALU computing paradigm. Furthermore, it incorporates a BitFusion-like regular two-dimensional digital bitcell structure, which can be reconfigured from 1 to 16 bits to meet various performance and energy requirements.

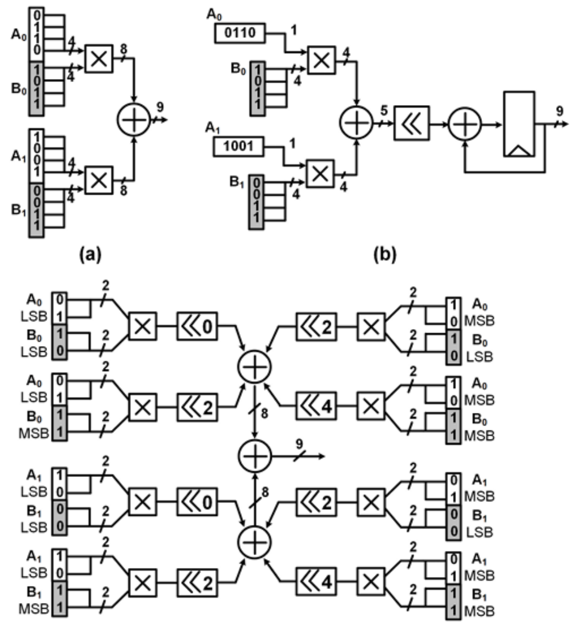


Fig. 25 Digital ALU architecture. (a) Conventional bit-parallel, (b) bit-serial, and (c) reconfigurable bit-precision[18]

3.4 Proposed 10T SRAM cell

3.4.1 Circuit Description

The proposed 10T current-based SRAM bitcell features a design that includes a sub-circuit of transistors M1-M2-M3-M4, resembling a conventional 6T SRAM design. This configuration forms a bi-stable transistor structure composed of two CMOS inverters connected in a back-to-back fashion, creating a feedback loop that maintains a particular logical state (0 or 1) as Q (true value) or QB (complementary value). For write access, the cell uses two access transistors, M5 and M6, driven by the writing wordline signal (WWL) and connected to the write bitlines (BL and BLB) on either side.

Additionally, transistors M7-M8-M9-M10 form the read circuitry for the design. Their gate terminals are connected to the cell’s storage nodes, Q and QB, and controlled by two read wordline signals (BL and BL_BAR), which are connected to the read bitlines (RBL and RBLB). This configuration enables independent read and write operations due to the separate write and read access ports, enhancing the read stability compared to conventional 6T cell

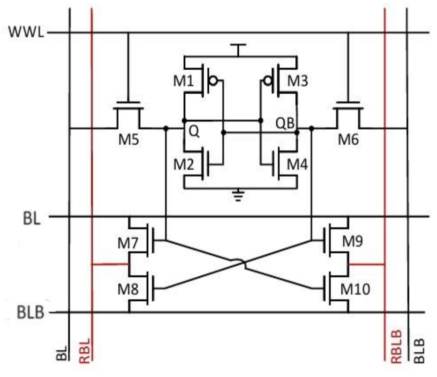


Fig. 26 Proposed 10T SRAM cell

3.4.2 Working Principle

Table 2 enlists the control signals for the proposed 10T cell during different operating conditions. As can be seen, the write operation is similar to the conventional 6T cell. Here, based upon the type of input applied to BL and BLB, data is written inside the cell, using write access transistors, driven by WWL. Hold operation is equivalent to not selecting the cell. It works on the logic to detect no voltage difference on any output bitline. Therefore, to make the cell hold a particular logic state, both of its read bitlines, RBL and RBLB are precharged to logic high and the read wordlines are also made to stay at a logic high state, resulting in no voltage change on read bitlines. Also, signal WL is deactivated in order to disconnect write bitlines from storage node, inferring a hold operation.

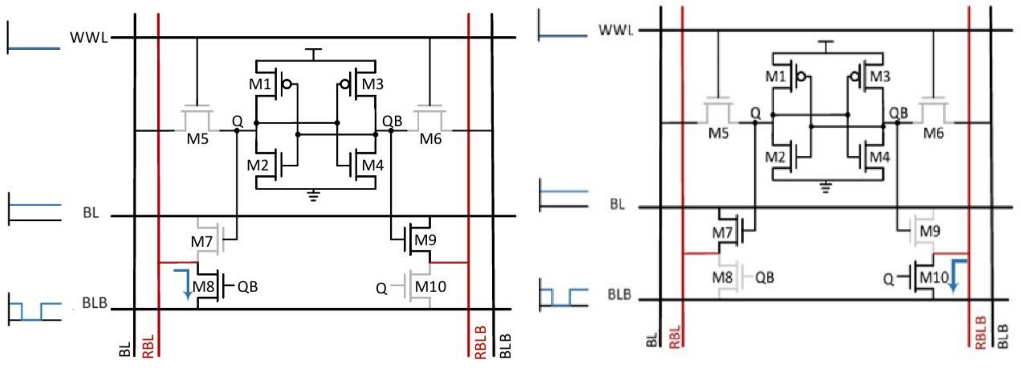
To understand the read operation, we first need to examine the role of the read wordline signals, BL and BL_BAR, in reading the desired output on the read bitlines, RBL and RBLB. The biasing conditions detailed in Table 1 clarify the operational concept for reading data from the proposed cell, presenting two scenarios: one for reading Q on bitline RBL and another for reading QB on bitline RBLB.

Initially, both read bitlines are precharged to a logic high level. The output to be read on RBL and RBLB is then determined by the inputs applied to RWL0 and RWL1. Figure 27 illustrates four cases that help to understand the read-data operation, each corresponding to different input biasing conditions on the wordlines, BL and BLB.

Table 2 Biasing for different memory mode operations[23]

	WWL	BL	BLB	BL	BLB	RBL	RBLB	Output
Hold	L	H	H	X	X	X	X	Q and QB holds their state
Write 0/1	H	H	H	L/H	H/L	X	X	Q = L/H; QB = H/L
Read 0/1 (RBL)	L	H	L	X	X	Initialized to H		RBL = Q; RBLB = QB
Read 1/0 (RBLB)	L	L	H	X	X	Initialized to H		RBL = QB; RBLB = Q

H = High, L = Low, X = Don't care



(a) Read 0 on RBL

(b) Read 1 on RBL

Fig. 27 Read Operation for proposed 10T SRAM cell

3.4.3 Read Stability

The implementation of 6T SRAM cells offers the advantage of low static power dissipation. However, a significant issue with 6T SRAM cells is potential instability during read operations, where a stored 0 can be inadvertently overwritten by a 1. This occurs due to a positive feedback mechanism, where the voltage at node Q exceeds the threshold voltage of PMOS M1, causing node QB to drop to 0 and subsequently pulling node Q up to 1. To address this issue, the proposed design uses separate read/write wordlines to isolate the data retention element from the data output element, preventing data storage disruption during read operations.

Maintaining data retention in SRAM cells during standby mode and read access is a critical functional constraint, especially in advanced technology nodes. With technological scaling, the stability of the cell decreases as the supply voltage is reduced, leading to increased leakage currents and variability. The stability is typically defined by the Static Noise Margin (SNM), which is the maximum value of DC noise voltage that the SRAM cell can tolerate without altering the stored bit. Figure 4 illustrates the SNM comparison between the proposed 10T cell and a conventional 6T SRAM cell. The read SNM of the proposed 10T cell is 395mV at VDD = 1V, whereas the conventional 6T SRAM cell exhibits a read SNM of 155mV at VDD = 1V. The RSNM of the decoupled 10T cell is 2.54 times that of the conventional 6T cell

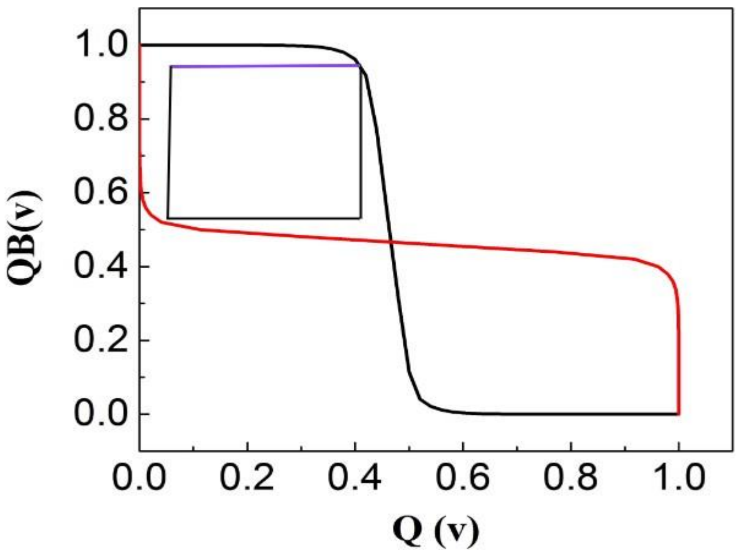


Fig. 28 SNM of 10T SRAM cell

3.4.4 XNOR Operation

When binary values are used for computation purposes, the dot product oper- ation between weights and activation functions can be reduced to bit-wise operations; binary values being -1 or +1. These are encoded with logic '1' for +1, and logic '0' for -1. Table 9 illustrates how multiplication on binary values can be interpreted as performing an XNOR operation on binary encoded logic values.

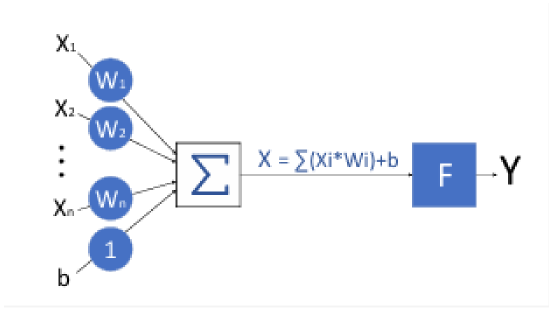


Fig. 29 Basic neuron architecture showinf MAC operation[23]

Table 3 XNOR Operation Equivalent to DOT- Product[23]

Encoding (Value)		XNOR (Multiply)
0 (-1)	0 (-1)	1 (+1)
0 (-1)	1 (+1)	0 (-1)
1 (+1)	0 (-1)	0 (-1)
1 (+1)	1 (+1)	1 (+1)

3.4.5 MAC(Multiply-and-accumulate) using 10T SRAM

In today's world, neural networks are important tools for achieving cutting edge results in a wide range of autonomous applications. DNNs have traditionally been used for this purpose, as they use 32-bit floating-point integers. Convolution in neural networks is based on the multiply-accumulate principle, which entails computing the dot product of two matrices, one holding weight and another storing input, which is a relatively common operation. The major issue arises when it comes to managing 32-bit numbers, which involve computations to be performed, requiring high storage and thus, are expensive. Because of this it becomes difficult for today's edge applications to handle sucha scenario. BNNs were proposed to tackle this problem by limiting weights and input activations (IAs) to +1 and 1. This reduced storage and computation need to a major extent. As a result, the simple XNOR-pop count operation replaces the dot-product operation in BNNs.

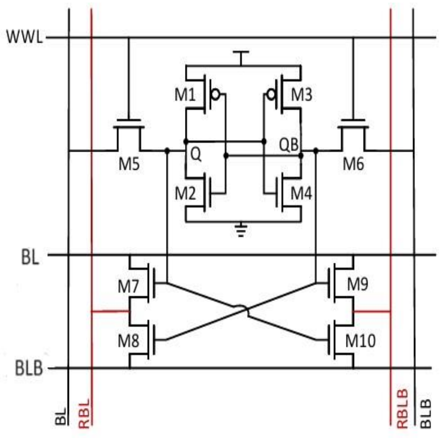


Fig. 30 10T SRAM cell

Table 4 MAC Operation using 10T SRAM cell

BL(Input)	Q(Weight)	XNOR(RBL)
0(-1)	0(-1)	1
0(-1)	1	0(-1)
1	0(-1)	0(-1)
1	1	1

3.5 Bitcell For Proposed Architecture

Fig. 31 shows a block diagram of the proposed digital bitcell. A bitcell is composed of three major building blocks: a full custom designed proposed 10T SRAM cell for a bitwise multiplication, and a full-adder for accumulating partial sum. Two 2:1 multiplexers (MUXs)are added for the selection of internal signals in different configurations. MUXs are used to configure the operating mode of the bitcell in column MACs and to determine the LSB bitcell which is located at the top of each column MAC. Fig. 8 describes an operation example of two columns with five bitcells per each to form a cascaded two 4 bit bit-serial MAC units. Each bitcell is configured to one of the two different functional models (i.e., Type-A and Type-B) based on its location within the column bitcell array. Type-A bitcell enables all three building blocks in a bitcell, while Type-B only enables full-adder for accumulate-only operation. The bit-precision of weights is configured by the number of Type-A bitcells in a column (e.g., 4 bit in Fig. 8), while type-B bitcells are added to extend the output precision which also depends on the number of columns. For example, the number of Type-B bitcells are 7 for each column MAC in the 128-column array and the output precision at 1 bit, 4 bit and 16 bit weights are 8 bit, 11 bit, and 23 bit, respectively. Each cycle of bit-serial MAC operation, a serialized input on the shared bitline is multiplied to a 4 bit weight stored in SRAM cells in a 4 bit column MAC. Bitwise multiplication results are then accumulated at the following ripple carry adder that is formed by vertically connected full-adders from each bitcells. The bit-serial MAC computations are performed through all the column MACs in the same row. Once a partial sum output value on the far right of the column is settled, then it is further post-processed for merging partial sum results from each cycle of bit-serial operation.

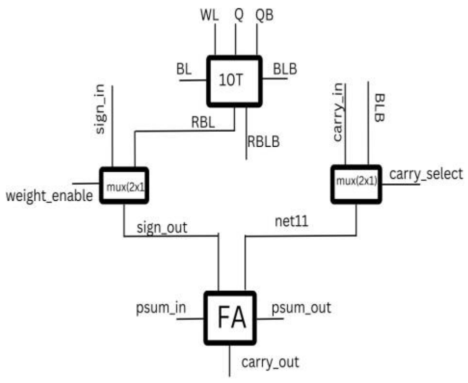


Fig. 31 BitCell Using 10T SRAM cell

3.5.1 Basic Operation Using Bitcell

Fig. 32 describes an operation example of two columns with five bitcells per each to form a cascaded two 4 bit bit-serial MAC units. Each bitcell is configured to one of the two different functional models (i.e., Type-A and Type-B) based on its location within the column bitcell array. Type-A bitcell enables all three building blocks in a bitcell, while Type-B only enables full-adder for accumulate-only operation. The bit-precision of weights is configured by the number of Type-A bitcells in a column (e.g., 4 bit in Fig. 32), while type-B bitcells are added to extend the output precision which also depends on the number of columns. For example, the number of Type-B bitcells are 7 for each column MAC in the 128-column array and the output precision at 1 bit, 4 bit and 16 bit weights are 8 bit, 11 bit, and 23 bit, respectively. Each cycle of bit-serial MAC operation, a serialized input on the shared bitline is multiplied to a 4 bit weight stored in SRAM cells in a 4 bit column MAC. Bitwise multiplication results are then accumulated at the following ripple carry adder that is formed by vertically connected full-adders from each bitcells. The bit-serial MAC computations are performed through all the column MACs in the same row. Once a partial sum output value on the far right of the column is settled, then it is further post-processed for merging partial sum results from each cycle of bit-serial operation.

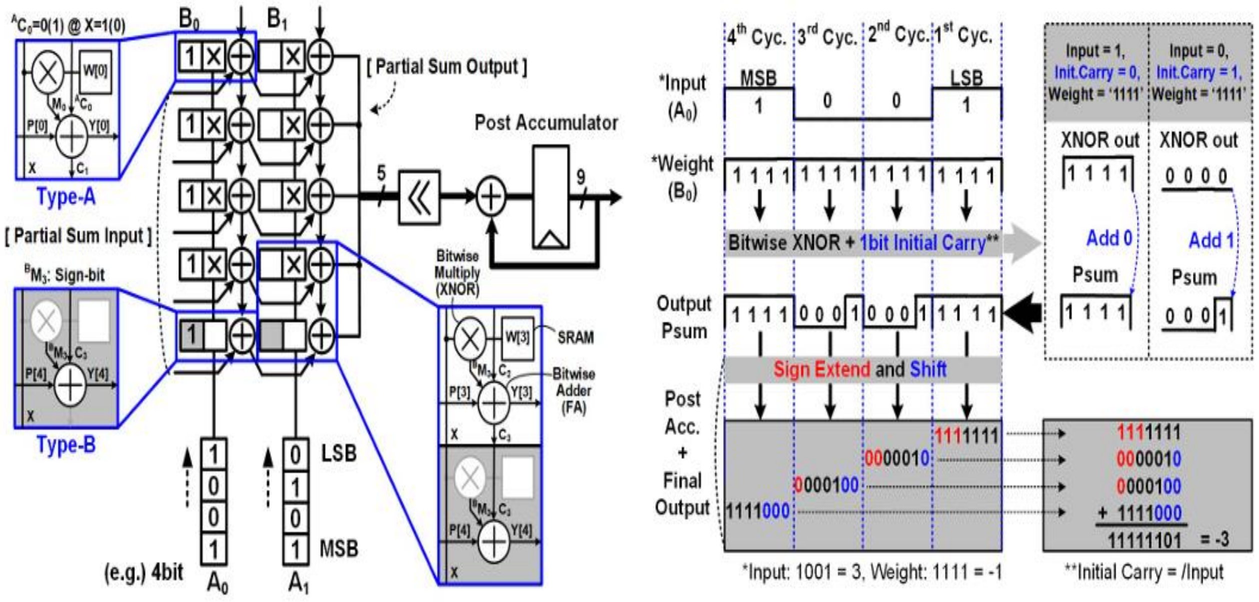
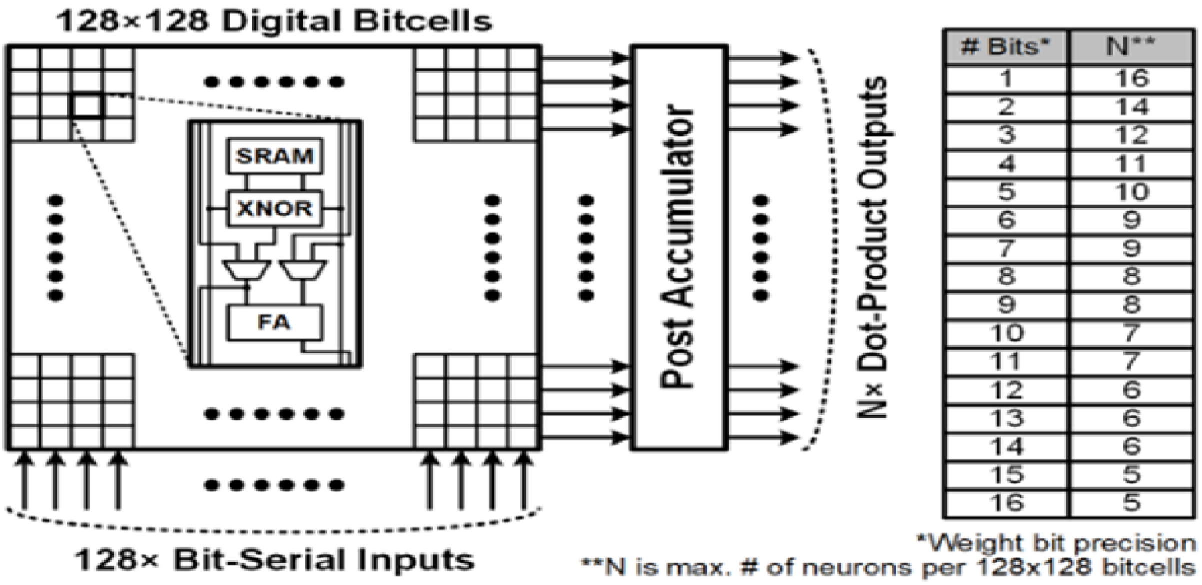


Fig. 32 Building a column MAC array using bitcells. In the example on the left, 10 bitcells are used for building a dot-product with 4 bit weight/input. The example shown on the right describes bit-serial multiplication of 4 bit two’s complement weight and 4 bit binary weighted signed number input[22]

3.5.2 Reconfigurability

Fig. 33 shows a complete CIM macro comprising of 128 ×128 digital bitcells and a post-accumulator. The bitcell array can be readily reconfigured to operate as parallel dot products. Each row of the reconfigured column MACs per forms a dot-product computation with variable bit-precisions. We assign the number of Type-A bitcells to represent the weight precision while the number of Type-B bitcells is determined based on the dynamic range of partial-sum, which depends on the number of columns. For instance, we can assign 1 Type-A and 7 Type-B bitcells as a single column MAC when reconfiguring the CIM macro into sixteen 1 bit dot-products, as shown in Fig. 34, left. The macro can be reconfigured to eight 9 bit dot-products by changing the number of Type-A bitcells per column MAC from 1 to 9, as shown in Fig. 34, right. Note that the number of bit-serial operation cycles programs the input precision, and hence 9 bit dot-products require 9× more operation cycles than that of 1 bit dot-products

A 16 bit partial-sum (i.e., Pi [15:0]) is generated from each cycle (i = 0 to 8) of the dot-product operation between 128×9 bit weights and inputs. Each cycle, the binary-weighted inputs are serialized and used for computing dot-product partial-sums. Hence, the partial-sums are left shifted and accumulated at the following post accumulator, as shown in Fig.35.



# Bits*	N**
1	16
2	14
3	12
4	11
5	10
6	9
7	9
8	8
9	8
10	7
11	7
12	6
13	6
14	6
15	5
16	5

Fig. 33 CIM micro with 128X128 bitcells for N x product [22]

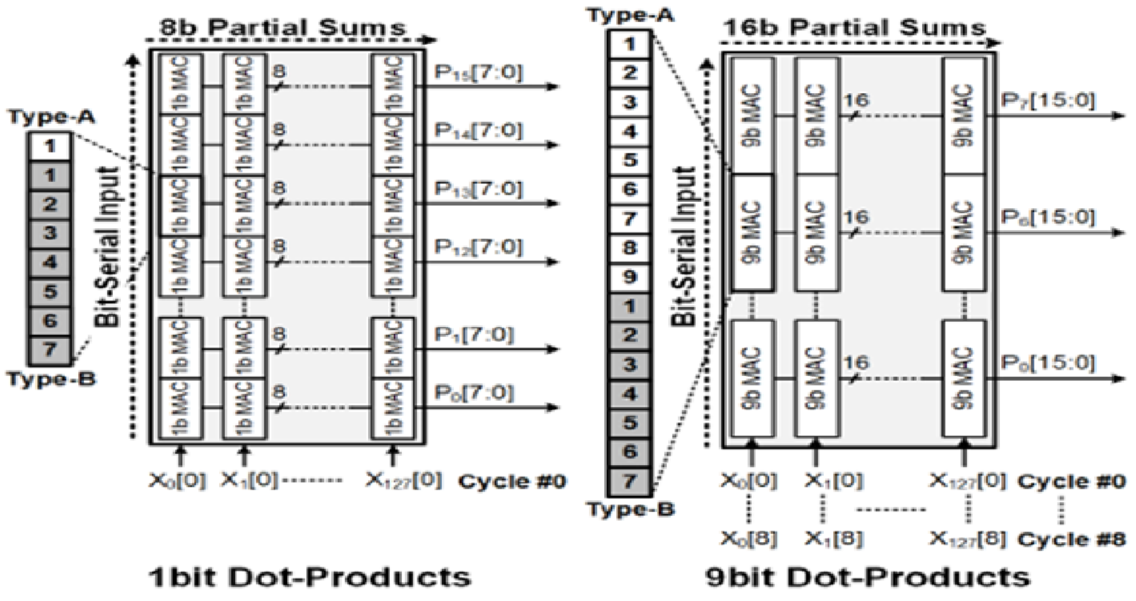


Fig. 34 Reconfigured CIM macros with 128 × 128 bitcells. Each column MAC requires M Type-A cells and 7 Type-B cells[22]

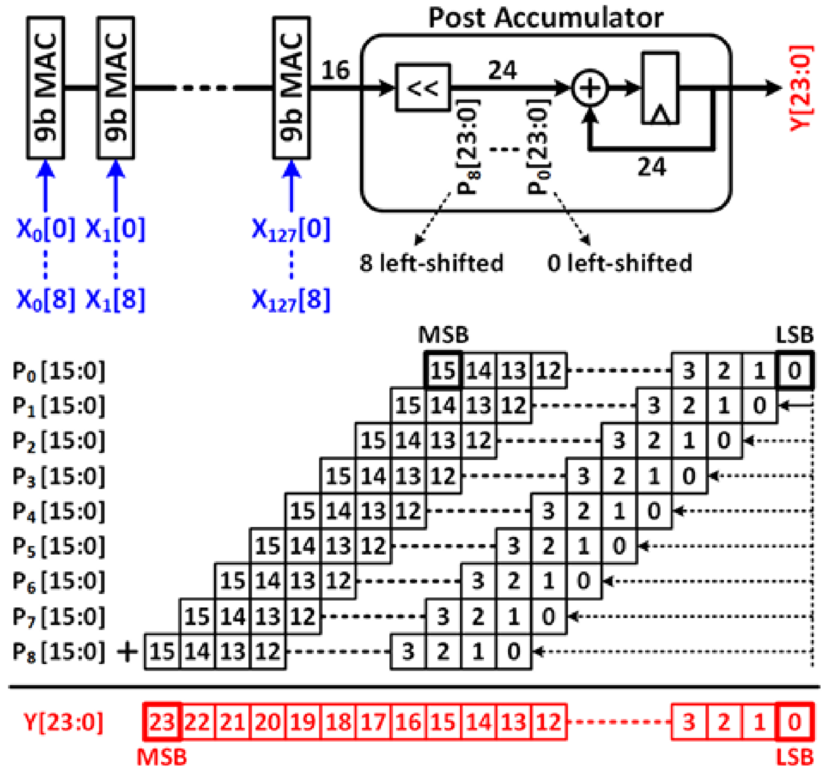


Fig. 35 Postaccumulator combines individual partial-sums from each operation cycle to complete a dot-product with multibit precision[22]

3.6 Top Level Of Proposed Architecture

CIM architectures generally comprise an array of memory cells surrounded by essential peripheral circuits and control logic. Figure 2 illustrates the memory array as the central focus, with other main blocks including the address decoder, word line drivers, column multiplexer, precharge circuitry, write drivers, sense amplifier, and control logic. The subsequent sections detail the operation of each individual block within the SRAM, followed by a high-level explanation of how these diverse blocks interact to facilitate the functioning of a memory device.

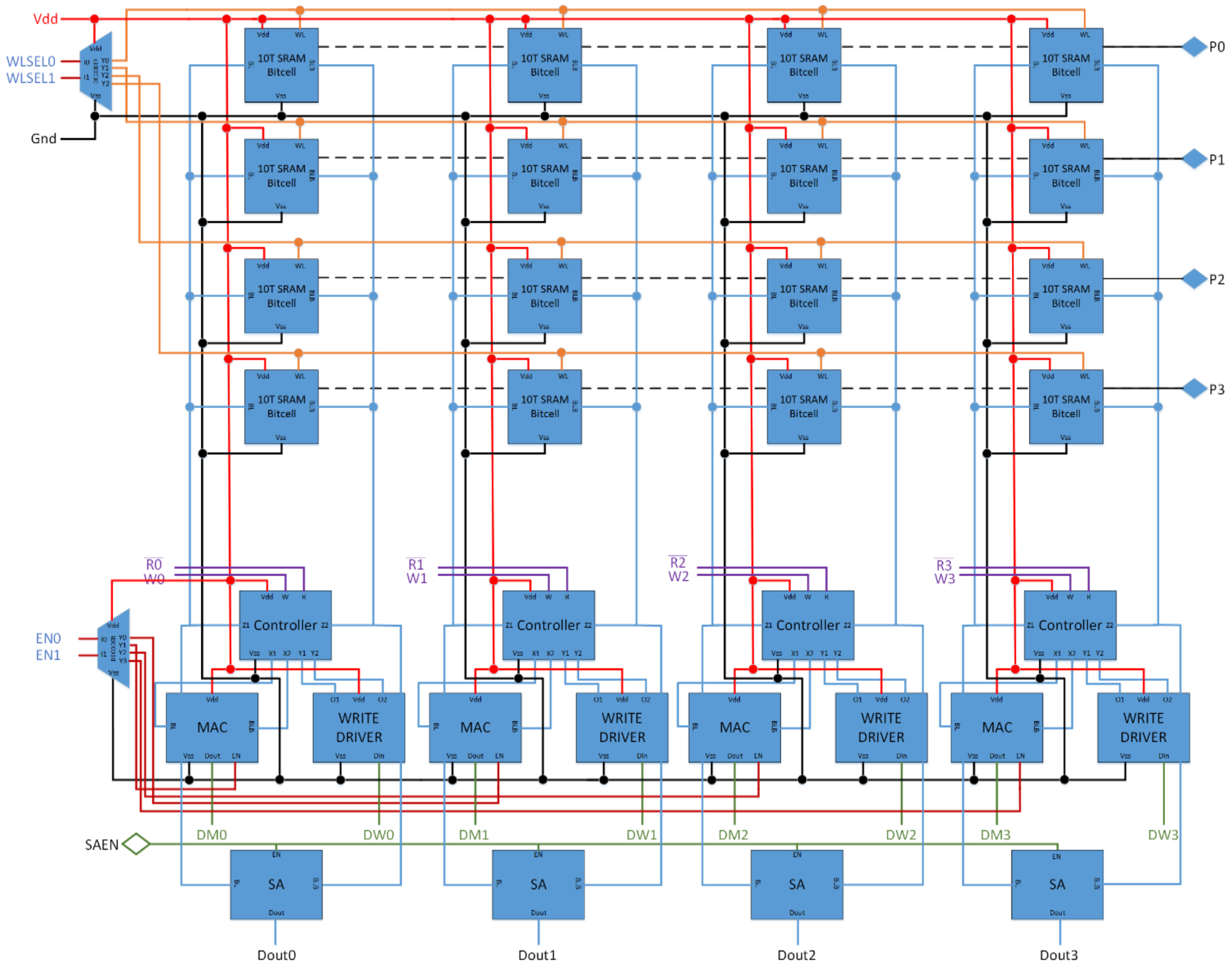


Fig. 36 Proposed CIM architecture

3.6.1 Precharge Circuitry

The precharge circuit is an integral component used in both read and write operations within the SRAM architecture. It plays a crucial role during the initial phase of the clock cycle. Illustrated in Figure 37, this circuit is relatively straightforward, comprising three PMOS transistors.

Upon receiving the input signal from the cell, known as PCLK, all three transistors are activated. Two of these transistors, M1 and M2, are responsible for charging the bit lines, BL and BL_bar, respectively, to the supply voltage (Vdd). The third transistor, M3, aids in equalizing the voltages observed on the bit lines.

The primary purpose of equalizing the bit line voltages during the pre-charge phase is to prepare them for subsequent operations. By ensuring that both bit lines have equal voltages, any voltage discrepancies that occur during subsequent phases become more pronounced. This facilitates quicker detection of voltage differences by the sense amplifier, which is essential for accurate data retrieval

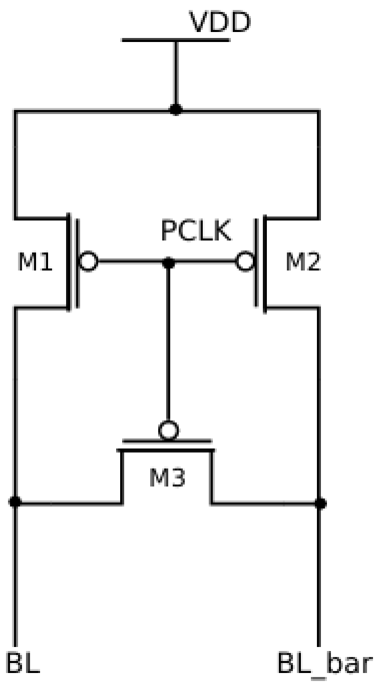


Fig. 37 Precharge Circuitry[8]

3.6.2 Address Decoder And Word Line Drivers

The address decoder in CIM architectures plays a crucial role in selecting the appropriate word line based on the row address bits received from the address bus. With an n-bit input, the address decoder can control 2^n word lines. Figure 38 demonstrates a 2-to-4 dynamic NAND decoder, functioning as follows: During the initial clock phase, while the clock signal is low, PMOS transistors enabled by the PCLK signal precharge all internal word lines to Vdd. In the subsequent clock phase, the PMOS transistors are disabled. Depending on the input address, a specific internal word line is pulled down to ground. Output inverters ensure that only one word line is asserted during the second clock phase, optimizing data access within the CIM architecture.

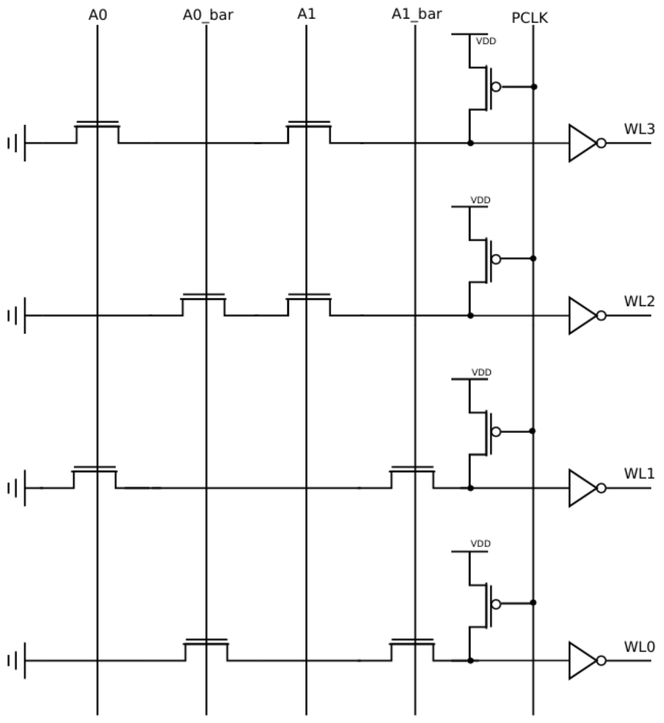


Fig. 38 Address Decoder[8]

Table 5 Truth Table for 2 to 4 NAND decoder[8]

Asserted WL	Inp1	Inp2	Binary
WL0	A0_bar	A1_bar	00
WL1	A0	A1_bar	01
WL2	A0_bar	A1	10
WL3	A0	A1	11

The truth table for the 2-to-4 decoder is depicted by Table 5. From the table it can be seen that the inputs are connected to the address bits in a binary reduction pattern. This pattern can be exploited to easily scale the dynamic decoder up to handle an array with more rows. Word line drivers are inserted, as bffers, in-between the word line output of the address decoder and the input of the Bit- cell. The word line drivers ensure that as the size of the memory array increases, and the word line capacitan ce increases, the signal is still able to turn on the access transistors in all Bit- cells

3.6.3 Column Multiplexer

The column multiplexer takes in n-bits from the address bus and can select 2^n bit line pairs associated with one word in the memory array. The schematic for a 4-to-1 tree multiplexer is shown in Figure 6. This type of tree multiplexer is bi-directional and is used for both the read and write operations; it connects the bit lines of the memory array to both the sense amplifer and the write driver.

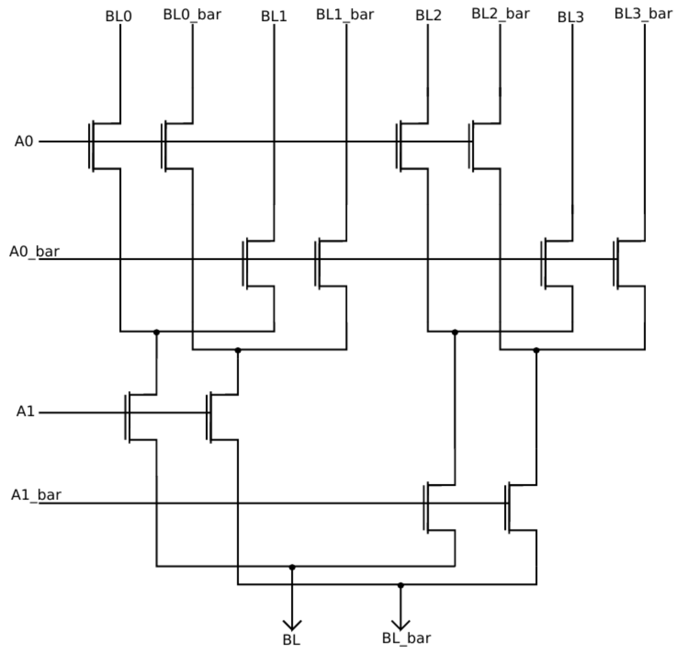


Fig. 39 Column Multiplexer[10]

As seen in Figure 39, the column mux is built of NMOS transistors in a tree-like structure. The depth of the decoder is determined based on the number of words per row in the memory array. The most basic column mux has a depth one which means that there are two words per row. If there is only one word per row in the array, then no column mux is needed.As The number of words per row in the memory array increases, the depth of the column mux grows. The depth of the column mux is equal to the number of bits in the column address bus.

Table 6 Binary reduction pattern for 4-to-1 tree column mux[10]

BL Pair	Inp1	Inp2	Binary
BL0/BL0_bar	A0_bar	A1_bar	00
BL1/BL1_bar	A0	A1_bar	01
BL2/BL2_bar	A0_bar	A1	10
BL3/BL3_bar	A0	A1	11

Figure 39 illustrates a column mux with a depth of two. This means that there are four words per row in the memory array and two select bits from the address bus are needed to choose the bit line pairs for one of the four words. A binary reduction pattern, shown in Table 6, is used to select the appropriate bit lines. In level one, A0, and its complement A0_bar, select either the even numbered words or the odd numbered words in the row. In level two, the most significant bit A1, and its complement A1_bar, then select one of the words passed down from the previous level. Relative to other column mux designs, such as pass transistor based decoders with NOR pre-decoders, the tree mux uses significantly less devices. However, this type of design can provide poor performance if a large decoder with many levels is needed. The delay of of a tree mux quadratically increases with each level. Due to this fact, other types of column decoders should be considered for larger memory arrays

3.6.4 Sense Amplifier

The sense amplifier is used to sense the difference between the bit lines (BL and BL_bar) while a read operation is performed. A sense amplifier is necessary to recover the signals from the bit lines because they do not experience full voltage swing. As the size of the memory array grows, the capacitive load of the bit lines increase and the voltage swing is limited by the small memory cells driving this large load. A differential sense amplifier is used to sense the small voltage difference between the bit lines and accelerates the read operation**.**

The schematic for the sense amp is shown in Figure 40. The sense amplifier is enabled by the SCLK signal, which initiates the read operation. Before the sense amplifier is enabled, the bit lines are precharged to Vdd by the precharge unit. When the sense amp is enabled, one of the bit lines experiences a voltage drop based on the value stored in the memory cell.

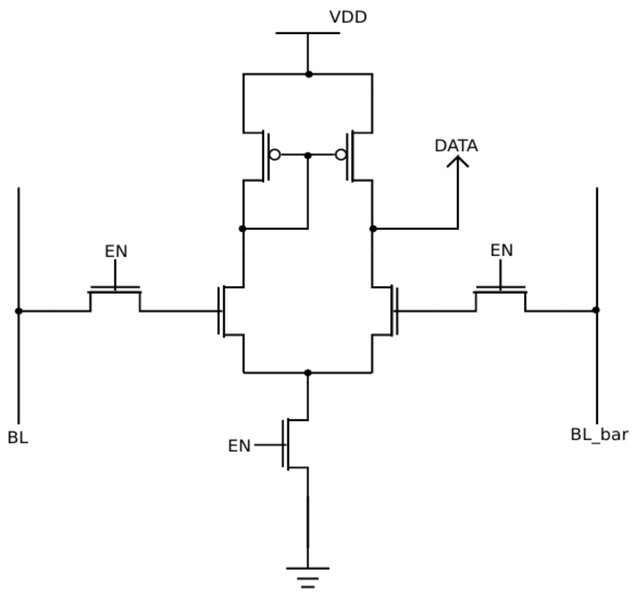


Fig. 40 Sense Amplifier[8]

If a zero is stored, the BL voltage drops. If a one is stored, the BL_bar voltage drops. The voltage difference between BL and BL_bar is sensed and the output signal is then taken to a true logic level and latched to the data bus

3.6.5 Write Driver And MAC Driver

The write driver is responsible for driving the input signal into the memory cell during a write operation. As depicted in Figure 41, the write driver comprises two tristate buffers: one inverting and the other non-inverting. It receives a data bit from the data bus and outputs this value to the bit line, while simultaneously outputting its complement to the bit line bar. Both tristate buffers are controlled by the EN signal. Ensuring that the bit lines always carry complementary values is crucial for corre ctly storing data in the Bitcell. Additionally, the drivers must be properly sized to accommodate the increasing bit line capacitance as the memory array expands.

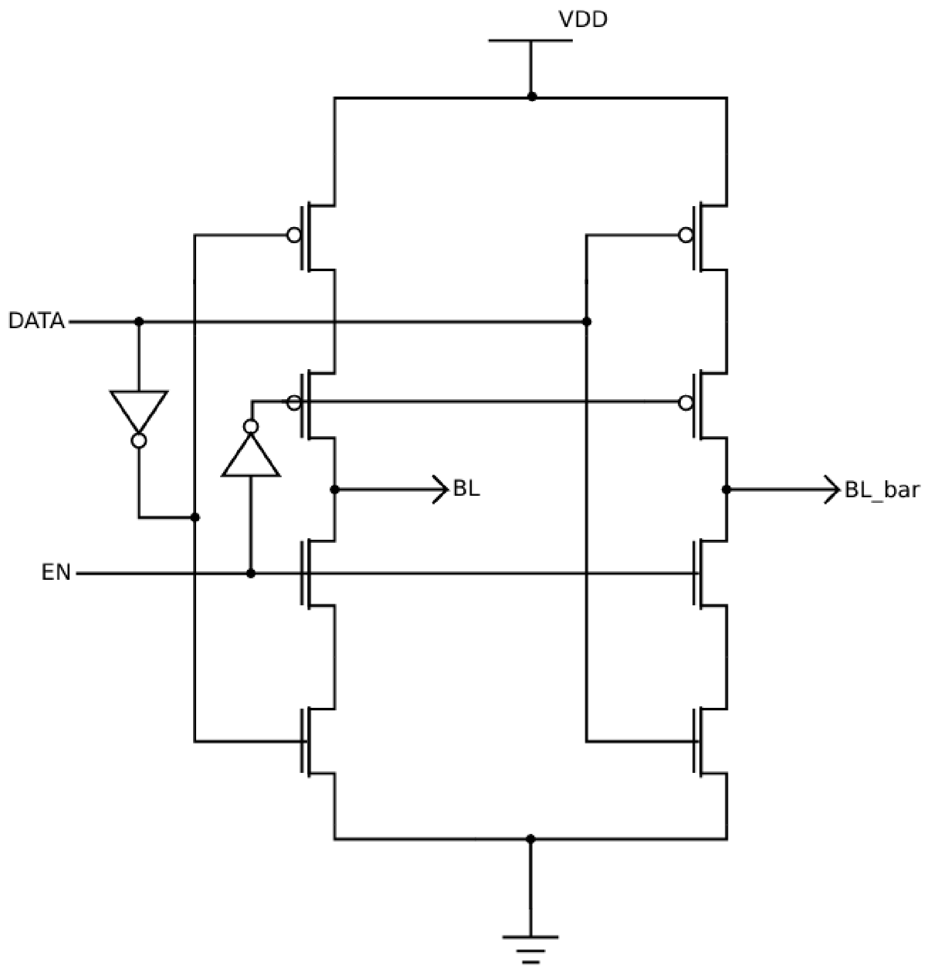


Fig. 41 Write Driver[8]

Chapter 4. Simulation Result & Analysis

4.1 Introduction

To design and analyze a proposed Processing-In-Memory (CIM) architecture using Cadence Virtuoso with TSMC 65nm technology, begin by setting up the Cadence Virtuoso environment. Ensure that Cadence Virtuoso is installed correctly and import the TSMC 65nm Process Design Kit (PDK) into Virtuoso. Create a new library for your CIM design and attach it to the TSMC 65nm technology file. In the schematic editor, design the CIM architecture by placing necessary components such as transistors, resistors, and capacitors, and ensure proper connectivity, including power (VDD) and ground (GND) connections. If needed, create a symbol for your design for hierarchical design purposes.

Next, set up the simulation by creating a test bench to simulate the CIM architecture. Define input sources for data and control signals, and set appropriate load conditions to mimic the actual operating environment. Use the Analog Design Environment (ADE) in Virtuoso to configure the simulation, selecting the appropriate simulator (e.g., Spectre), specifying the correct TSMC 65nm model files, and setting up the type of analysis (DC, AC, transient, parametric) along with simulation parameters such as run time, accuracy, and temperature.

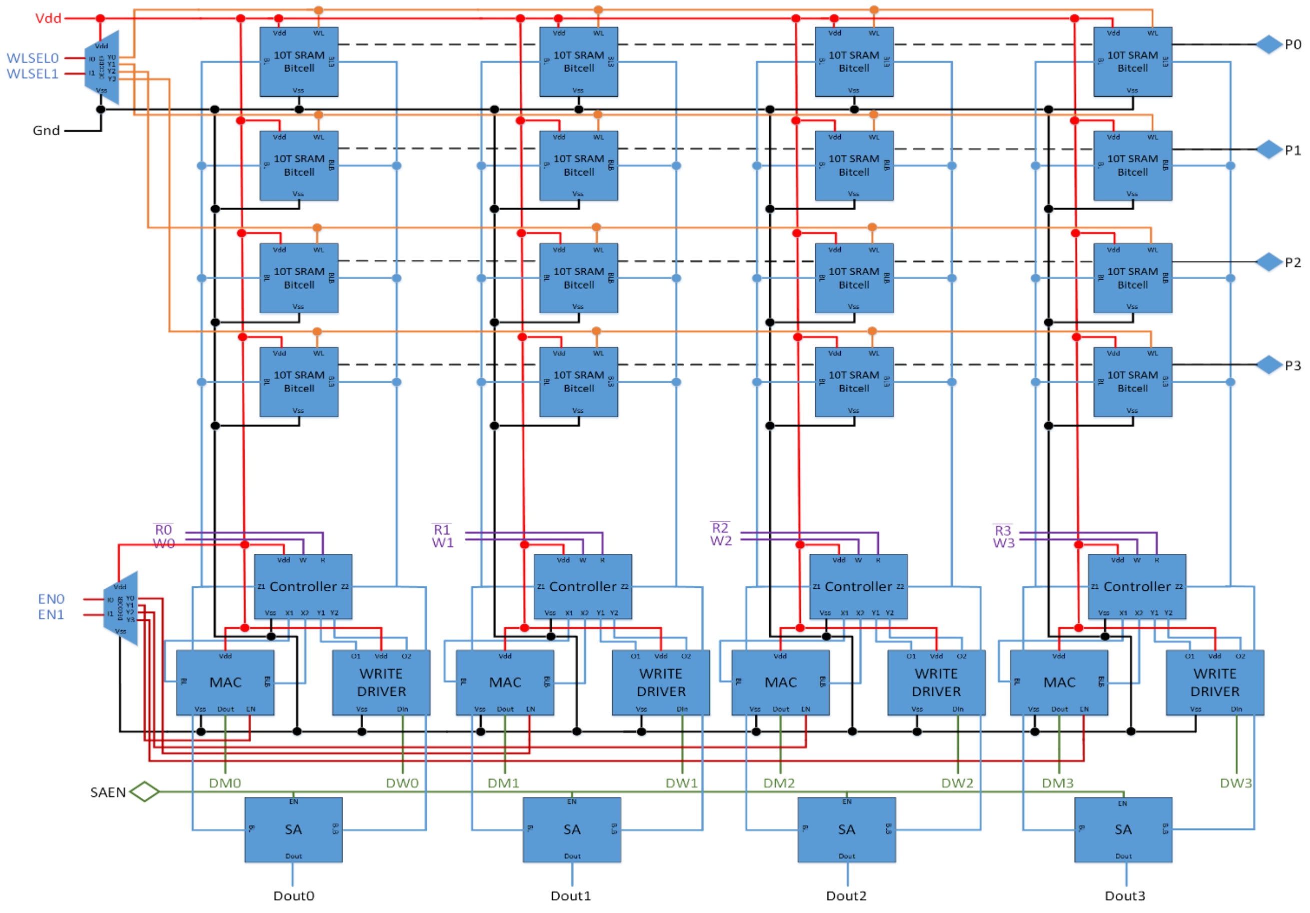


Fig. 42 Proposed CIM Architecture

4.2 Simulation Results Of Proposed 10T SRAM Cell

To perform read and write operations in a 10T SRAM cell, precise input stimuli must be applied. For a write operation, create a pulse signal for the word line (WL) to enable writing; the WL should go high, allowing data to be written into the cell. Apply the data bit to the bit line (BL) and its complement to the bit line bar (BLB), ensuring that the WL pulse is synchronized with the data inputs and remains high long enough to complete the write cycle. For a read operation, first precharge the bit lines (BL and BLB) to a specific voltage level, typically VDD/2, before initiating the read. Then, generate a pulse signal for the WL to go high, enabling the stored data to transfer to the bit lines. The duration of the WL pulse should be adequate for the cell to transfer the data. Monitor the bit line voltages: the bit line with a lower voltage indicates a stored '0', while the higher voltage indicates a stored '1'. This controlled application and monitoring ensure accurate write and read operations in the 10T SRAM cell.

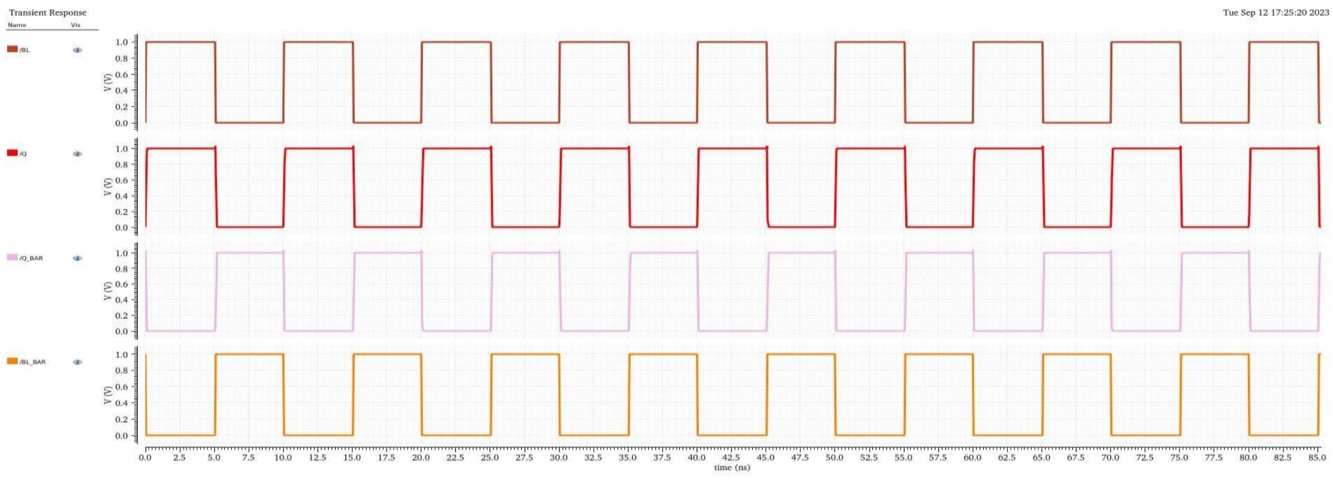


Fig. 43 Write Operation

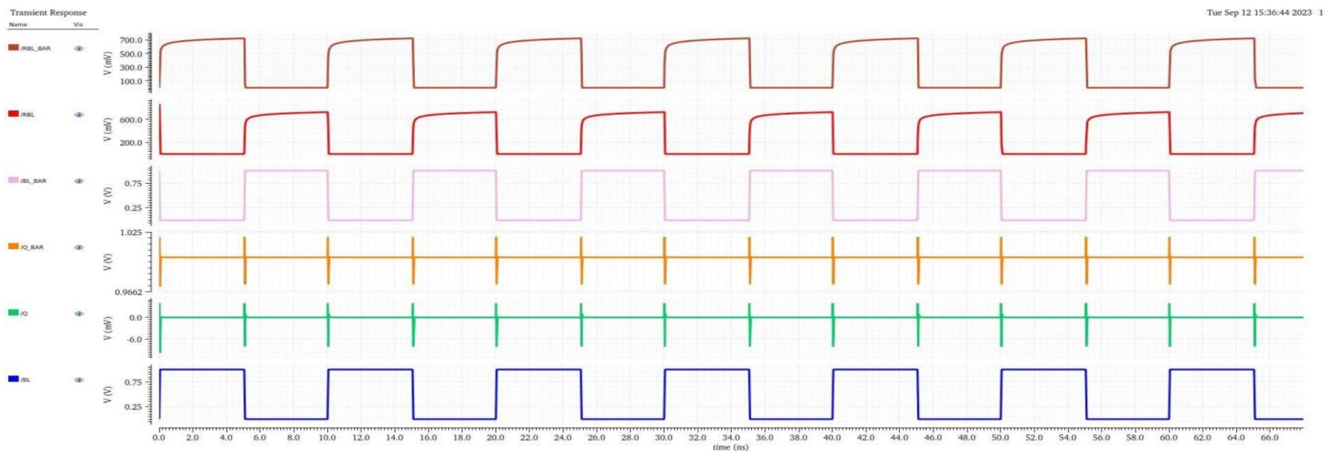


Fig. 44 Read Operation

4.2.1 Static Noise Margin(SNM):

Static Noise Margin (SNM) is a critical parameter in the operation of SRAM (Static Random-Access Memory) cells, reflecting their robustness against noise and disturbances. SNM is defined as the maximum noise voltage that a memory cell can tolerate before it becomes unstable and erroneously flips its stored state. It is typically measured using the "butterfly curve" method, which involves plotting the voltage transfer characteristics (VTC) of the cell's cross-coupled inverters. The largest square that fits within the lobes of this curve represents the SNM, with the side length of the square being the SNM value. SNM is evaluated in different operational modes: read SNM, write SNM, and hold SNM. During read operations, the read SNM is usually lower because the access transistors are activated, connecting the cell to the bit lines and potentially disturbing the stored data. Write SNM is concerned with the ability to overwrite existing data, ensuring reliable writes even in noisy conditions. Hold SNM, measured when the cell is idle, ensures data retention stability. Factors affecting SNM include transistor mismatches due to manufacturing variations, supply voltage levels, temperature, and cell design, particularly the sizing of the transistors. High SNM values indicate more reliable memory cells, contributing to better manufacturing yield and overall performance. Balancing SNM with other performance metrics like access speed and power consumption is crucial for optimal memory design.

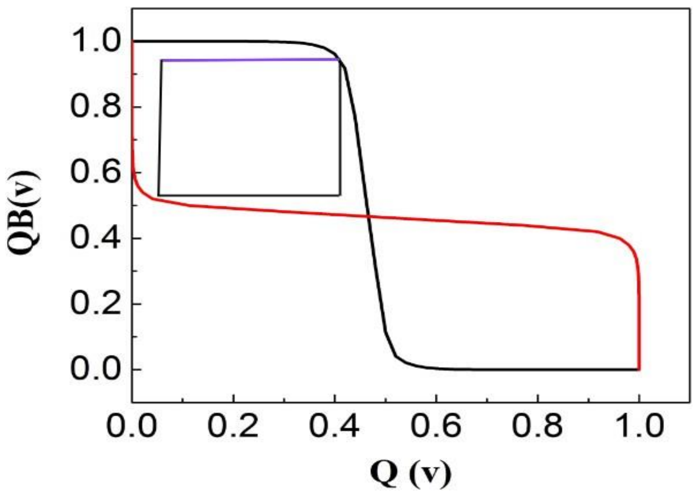


Fig. 45 SNM of 10T SRAM cell

4.2.2 Delay

Read and write delays are critical performance metrics for memory cells, including SRAM, as they determine the speed at which data can be accessed and stored. The read delay is the time it takes for a memory cell to transfer the stored data to the output after the read command is issued. This involves the activation of the word line (WL) and the subsequent propagation of the signal through the bit lines (BL and BLB) until the data is sensed by the read circuitry. Write delay, on the other hand, is the time required to write new data into the memory cell after the write command is activated. This process involves driving the bit lines with the data and its complement, enabling the word line, and ensuring that the data is successfully latched into the cell. To calculate these delays in Cadence Virtuoso, set up a transient analysis in the Analog Design Environment (ADE). For read delay, precharge the bit lines to a specific voltage level, pulse the word line, and measure the time taken for the bit line voltage to reach a defined threshold that indicates a successful read. For write delay, apply the data signals to the bit lines, pulse the word line, and measure the time required for the cell to store the new data. By analyzing the waveforms generated during these simulations, you can accurately determine the read and write delays, which are crucial for optimizing memory performance.

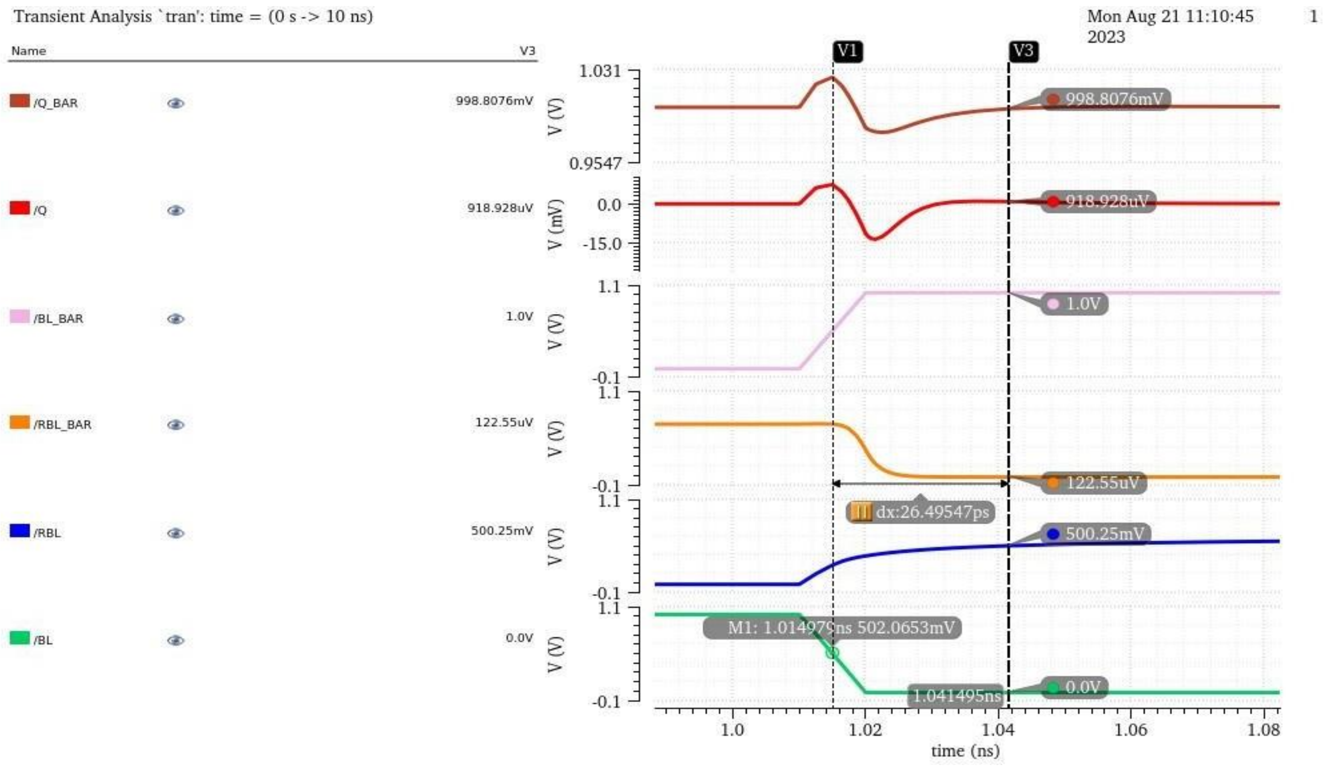


Fig. 46 Delay Analysis of 10T SRAM Cell

4.3 Simulation Result Of Bitcell For Proposed Architecture

To simulate the bitcell for a column Multiply-Accumulate (MAC) operation using Cadence Virtuoso, several steps need to be followed. First, the schematic for the bitcell must be designed, incorporating the custom-designed 10T SRAM cell for bitwise multiplication and a full-adder for accumulating partial sums. Additionally, two 2:1 multiplexers (MUXs) are included to select internal signals in different configurations, crucial for configuring the bitcell's operating mode within column MACs and determining the Least Significant Bit (LSB) bitcell placement at the top of each column MAC. Once the schematic is complete, a test bench schematic is created to instantiate the bitcell for simulation. Input stimuli are defined to test the MAC operation, including input data patterns for bitwise multiplication and full-adder operation. The MUXs are configured to select appropriate signals for the desired operation mode. In the Analog Design Environment (ADE), simulation parameters are set, including the simulator selection (e.g., Spectre) and simulation type (e.g., transient). The simulation is then executed, and waveforms and signals are monitored to verify the correct functionality of the bitcell during MAC operation. Any observed issues are debugged, and circuit parameters are optimized if necessary. Finally, the simulation setup, results, and observations are documented in a comprehensive report, ensuring that the bitcell meets the requirements for efficient MAC operation in the intended application.

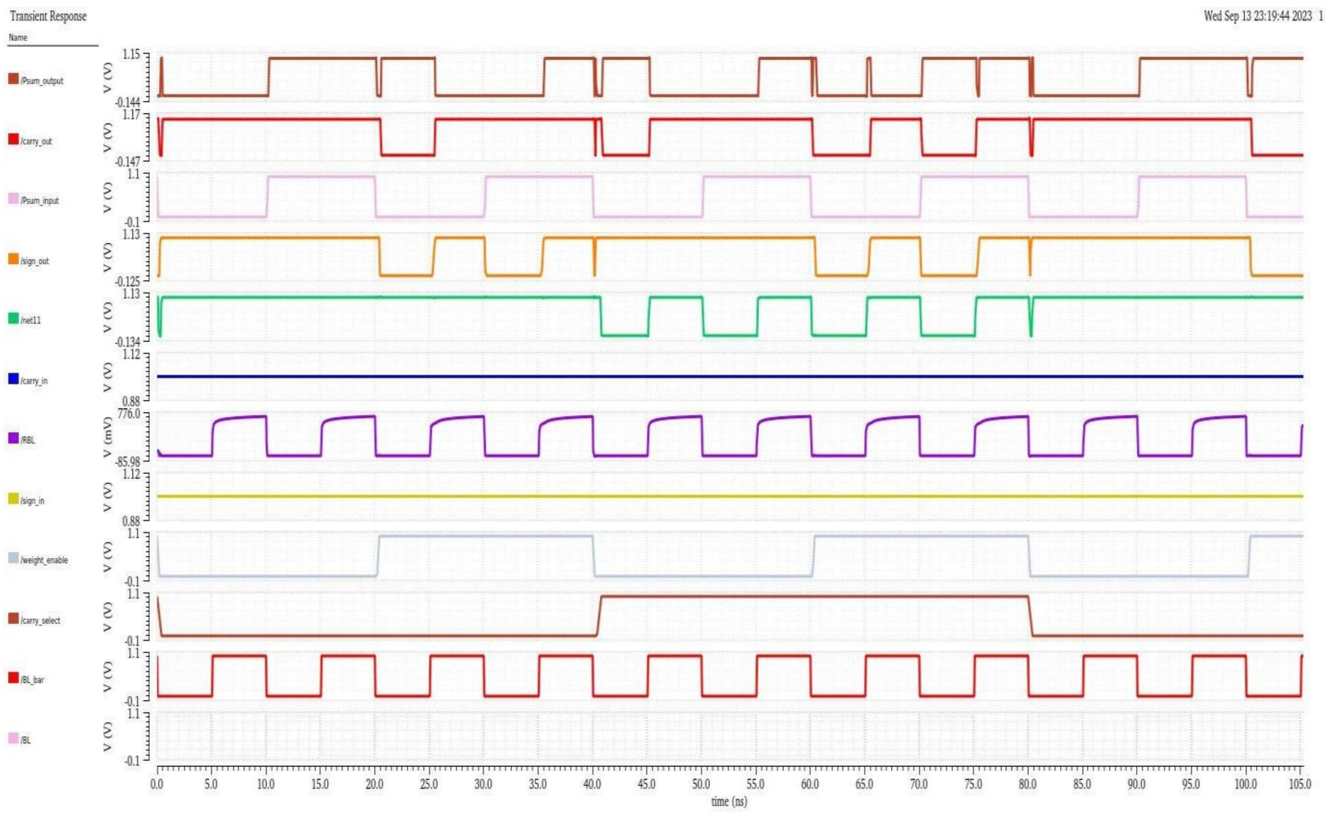


Fig. 47 BitCell output waveform

4.4 Simulation Result Of Different Operation Of Proposed Architecture

To simulate a Processing-In-Memory (CIM) computation architecture capable of executing read, write, and Multiply-Accumulate (MAC) operations using Cadence Virtuoso, a systematic approach is essential. Firstly, the architecture of the CIM system needs to be designed, encompassing the memory array, processing units, and interconnects, ensuring efficient support for both read and write operations. Following this, schematics are created for each component using the Cadence Virtuoso schematic editor, ensuring proper connectivity and functionality. A test bench schematic is then developed to instantiate the CIM architecture for simulation, incorporating input stimulus generators and defining the timing and sequencing of operations. In the Analog Design Environment (ADE), the simulation is configured with appropriate parameters, and simulations for read, write, and MAC operations are conducted. During read simulations, input patterns are defined to observe correct data retrieval, while write simulations validate data storage accuracy. MAC operation simulations involve configuring input data patterns and control signals to compute MAC operations in the processing units, with subsequent analysis to validate correctness and efficiency. Performance metrics such as latency, throughput, and energy efficiency are measured and analyzed. Any encountered issues are debugged, and optimizations are made to enhance overall efficiency. Finally, comprehensive documentation of the simulation setup, results, and observations is prepared, providing insights into the CIM architecture's performance and functionality, aiding in informed decision-making before hardware implementation.

4.4.1 Signals

In order to explain the read and write operations of a CIM Architecture, it is necessary to summarize the internal and external signals as well as the important timing considerations

The typical top-level signals for a CIM Architecture are:

- DATA - the bi-directional data bus
- WL - - the address bus
- Carry_select - for mux1 of bitcell
- Weight_enable – for mux2 of bitcell
- P0,P1,P2,P3 - Output of MAC operation
- Write_enable - column mux for write operation
- Read_enabel - column mux for read operation
- MAC_enable - column mux for MAC operation
- PCLK - to activate precharge circuit
- SCLK - to activate sense amplifier circuit
- WD_EN - enables the write driver during a write operation

4.4.2 Timing Considerations:

In a Processing-In-Memory (CIM) architecture, timing considerations for SRAM cells play a crucial role in ensuring the reliability and efficiency of memory operations. Key timing parameters include the setup and hold times for input signals, memory read and write delays, and the minimum clock period. Setup and hold times define the duration for which an input signal must remain stable before and after the clock edge that triggers the memory operation, respectively. These parameters are essential for proper data capture and stability during read and write operations. The write delay indicates the time taken from the clock edge of a write operation until valid data is driven into a memory cell. Similarly, the read delay refers to the time elapsed from the clock edge until valid data appears as an output of the sense amplifier. These delays are critical for ensuring accurate data retrieval and propagation within the memory array. In a CIM architecture, optimizing these timing parameters is crucial to achieving high-speed and reliable memory operations, ultimately enhancing the overall performance of the system.

4.4.3 Read Operation:

In an proposed architecture, the read operation is a critical process for retrieving stored data from memory cells. It initiates with the activation of the word line corresponding to the targeted memory cell or row. This activation grants access to the selected memory cells within the designated row. Subsequently, the bit lines, BL and BLB, are precharged to a predefined voltage level, typically VDD/2, to establish a stable reference voltage. Following precharging, the bit lines are connected to the memory cell via access transistors, facilitating the transfer of stored data onto the bit lines. The sensed data is then amplified by sense amplifiers, detecting any voltage disparity between BL and BLB. Based on this voltage difference, the stored data in the memory cell is determined; a higher voltage on BL compared to BLB signifies a logical '1', while the opposite indicates a '0'. The sensed data is finally outputted from the sense amplifiers for further processing or utilization by the system. Once the read operation concludes, the word line is deactivated, safeguarding the integrity of the stored data within the memory cell. Overall, the read operation in an proposed architecture ensures efficient and accurate retrieval of data from the memory array, essential for the system's overall functionality

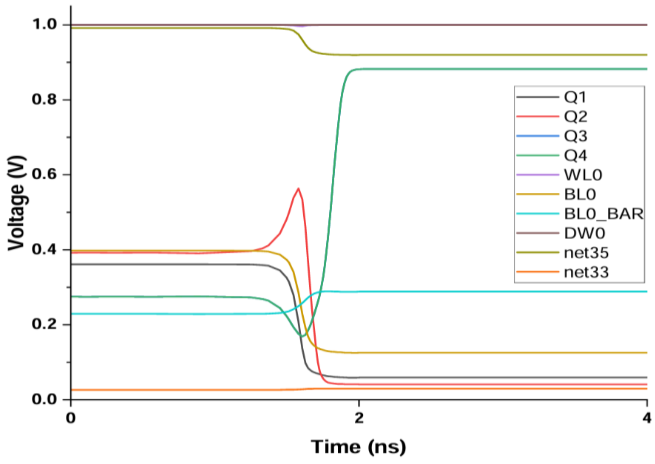


Fig. 48 Ouput Waveform Write Operation

Table 7 Read operation of Proposed Architecture

Signal	**State of input**
Write_enable	Low
Read_enable	High
MAC_enable	Low
Input_weight	Low
Input_MAC	Low
Word line	High
Bit line	High

4.4.4 Write Operation:

In an SRAM-based architecture, the write operation is a crucial process for storing new data into the memory cells. The write operation typically begins with the activation of the word line (WL) corresponding to the desired memory cell or row. This activation enables access to the selected memory cells within that row. Simultaneously, the bit lines (BL and BLB) associated with the targeted column are driven with the data bit and its complement. The activated word line allows the transistors in the access path of the selected memory cell to become conductive, connecting the memory cell to the bit lines. The stored data in the memory cell is determined by the voltage levels of the bit lines; if BL is at a higher voltage level compared to BLB, the memory cell stores a logical '1', and vice versa for a '0'. Once the write operation is completed for the selected cell, the word line is deactivated, isolating the memory cell from the bit lines to maintain the integrity of the stored data. Overall, the write operation in an SRAM-based architecture ensures efficient and accurate storage of data in the memory cells, essential for the system's overall functionality.

Table 8 Write Operation For Proposed Architecture

Signal	**State of input**
Write_enable	High
Read_enable	Low
MAC_enable	Low
Input_weight	High
Input_MAC	Low
Word line	High
Bit line	High

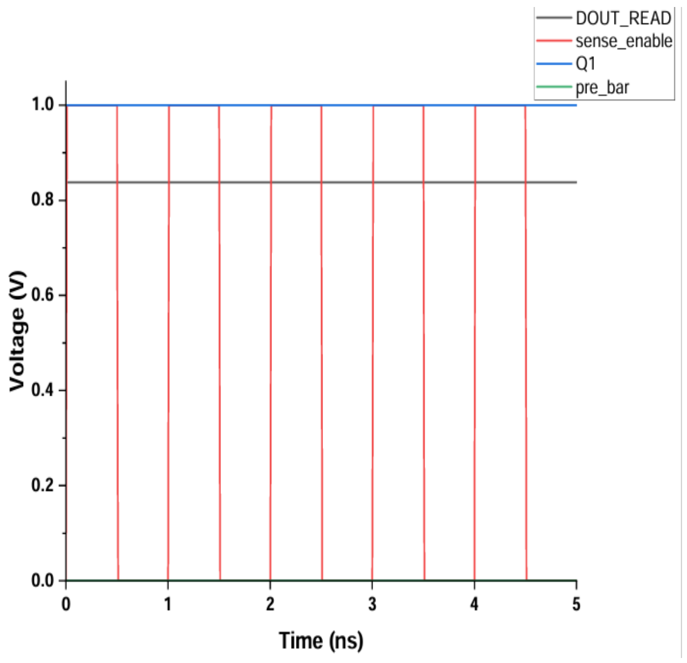


Fig. 49 Output Waveform Read Operation

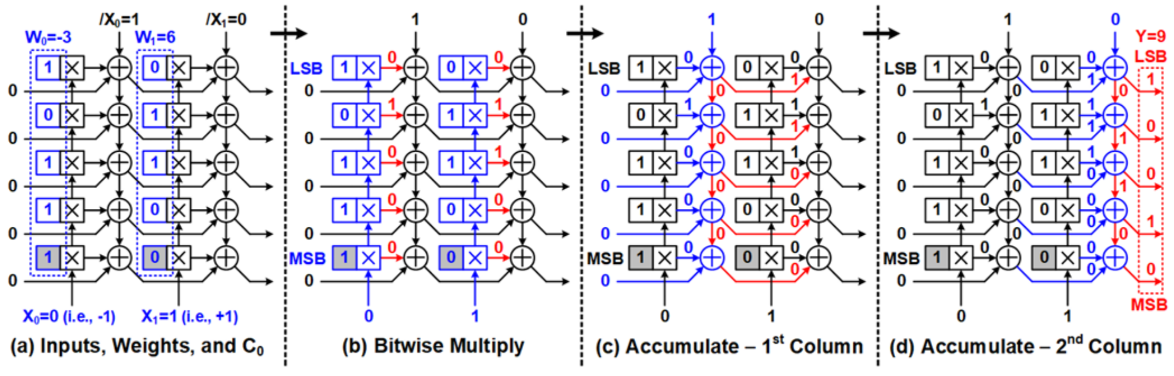
4.4.5 MAC Operation:

Table 9 shows the number representation scheme for input output, weight, and the first carry-in (C0) of the MAC unit. The weight is stored as a two's complement signed number while the input is encoded to a binary-weighted signed . number, which realizes the bit-serial input as the serialized binary value of +1/−1. Compared to the traditional binary weighted two's complement number, the proposed encoding scheme provides simpler operation by not having to spend more compute cycles for the sign bit and encoding stages to express the signed number. For instance, the encoded 4 bit serial input "0110" represents a decimal number −3 since "$-3 = -2^3 + 2^2 + 2^1 - 2^0$," where 0 represents −1and 1 represents +1. Each cycle of MAC operation can be broken into three steps

Table 9 Weight and Input Number Representation[22]

	Number Representation & Abstraction
Input	(0, 1) represents (-1, +1) [e.g.] $1001 = 2^3 - 2^2 - 2^1 + 2^0 = 3$
Output / Weight	Two's Complement
First Carry-In ($C_0 = /X_{IN}$)	$C_0 = 0$ @ $X_{IN} = 1$ (i.e., +1) $C_0 = 1$ @ $X_{IN} = 0$ (i.e., -1)

Figure 50 illustrates a detailed example of a four-step Processing-In-Memory (CIM) dot-product operation, utilizing two 4-bit column Multiply-Accumulate (MAC) units. Each MAC unit comprises four Type-A bitcells and one Type-B bitcell. The operation involves computing the dot product of two sets of 4-bit weights (W0 = -3, W1 = 6) and binary inputs (X0 = -1, X1 = +1). Initially, the weights and inputs are prepared for bitwise multiplications (Fig. 50). Note that the Type-B bitcell extends the sign from the above Type-A bitcell (MSB of the weight). In the first step, the two's complement multiply operations require the addition of the first carry (input bar) to produce correct results (Fig. 12b). When the input is 0 (-1), the sign-extended two's complement weight is inverted using XNOR-gate-based bitwise multipliers and then added to the first carry-in, which is 1 (X0 = 1). Conversely, the 5-bit weight is buffered while the first carry-in is 0 when the input is 1 (+1). The bitwise multiplication results are then accumulated in a ripple carry adder in the first column MAC (Fig. 12c). Finally, the 5-bit partial-sum result is combined with the bitwise multiplication results from the second column MAC on the right (Fig. 12d), completing the dot-product operation.



*4-Step CIM Dot-Product w/ two 4bit weights (W_0=-3 & W_1=6) & two binary Inputs (X_0=-1 & X_1=+1) → $Y=W_0 \cdot X_0 + W_1 \cdot X_1 = 9$

Fig. 50 Detailed operation of column MACs with 4 bit weight and 2× columns (i.e. 5× bitcells per column MAC). The input activation is 1 bit, and hence it takes 1× cycle to complete a full dot-product between two pairs of 4 bit weights and 1 bit input activations[22]

Table 10 Mac Operation for proposed Architecture

Signal	**State of input**
Write_enable	High
Read_enable	Low
MAC_enable	Low
Input_weight	High
Input_MAC	Low
Word line	High
Bit line	High

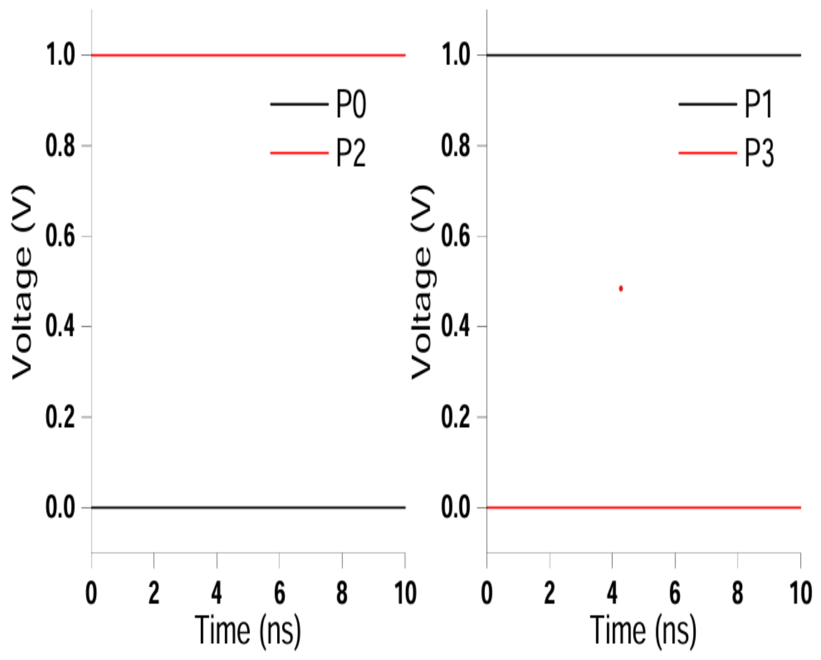


Fig. 51 Waveform Of MAC operation

Chapter 5. Conclusion And Future Scope

5.1 Summary

The design of a novel 10T SRAM cell aims to enhance the Static Noise Margin (SNM) compared to the conventional 6T SRAM cell. This new 10T SRAM cell incorporates additional transistors to improve stability and reduce susceptibility to noise, thereby ensuring more reliable data storage and retrieval. By leveraging the improved SNM, the 10T SRAM cell provides a robust foundation for developing a compute-in-memory (CIM) architecture.

The proposed CIM architecture is fully digital, utilizing bit-serial computing and offering reconfigurability to accommodate various input and weight precisions from 1 to 16 bits. This reconfigurability allows the architecture to adapt to different computational requirements, enhancing its versatility and efficiency in processing neural networks and other data-intensive tasks.

To implement and verify the performance of this CIM architecture, the entire design, including the novel 10T SRAM cell and the bit-serial computation framework, is simulated using Cadence Virtuoso with TSMC 65nm technology. The simulation process involves creating detailed schematics of the 10T SRAM cell, integrating it into the CIM macro, and configuring test benches to evaluate the functionality and performance of the system. Key parameters such as read/write delays, power consumption, and overall stability are analyzed to ensure the design meets the desired specifications.

5.2 Work Conclusion

Conventional digital accelerators have become unsuitable for machine learning tasks in edge-computing due to excessive energy consumption from off-chip memory access and data movement. This issue has driven the development of compute-in-memory (CIM) architectures, which integrate compact memory macros with embedded analog computing circuits in each bitcell. However, analog architectures face significant challenges, including process variation, data conversion overhead, noise susceptibility, and scalability issues, which are less prevalent in digital systems. While digital architectures have evolved to minimize off-chip memory access and data movement, a digital adaptation of CIM has been lacking.

The article introduces a novel digital CIM macro architecture designed to address the specific concerns of digital accelerators and resolve the critical issues faced by SRAM-based analog CIM macros. This CIM architecture features a fully reconfigurable digital CIM bitcell and a bit-serial CIM macro, consisting of a two-dimensional digital bitcell array and a post-accumulator. This innovative architecture allows both input and weight bit-precision to be programmed from 1 to 16 bits. Although bit-serial computation offers area efficiency, it comes with trade-offs in latency and throughput. The CIM architecture aims to provide a balanced solution that leverages the strengths of digital systems while overcoming the limitations of analog CIM implementations.

5.3 Future Scope of Work

The future scope of the novel 10T SRAM cell-based compute-in-memory (CIM) architecture is vast and promising. One potential direction involves further optimizing the 10T SRAM cell design to increase memory density, thereby enhancing its suitability for applications requiring extensive data storage and processing. Additionally, as semiconductor technology progresses, adapting the design to smaller process nodes, such as 45nm or 28nm, could significantly improve speed, power consumption, and overall performance. Another intriguing avenue is integrating the 10T SRAM-based CIM architecture with emerging memory technologies like resistive RAM (RRAM) or magnetoresistive RAM (MRAM), creating a hybrid system that leverages the strengths of various memory types for enhanced capabilities.

Future research could also focus on architectural enhancements, such as incorporating error correction codes (ECC) to boost data reliability and fault tolerance, and optimizing interconnects to reduce latency and improve throughput. Developing software and algorithms specifically optimized for the bit-serial CIM architecture could unlock new levels of performance, particularly in machine learning and neural network applications. Moreover, extending testing to real-world applications, such as edge computing devices, IoT systems, and autonomous vehicles, would provide valuable insights and demonstrate the architecture's practical viability.

Exploring the integration of energy-harvesting techniques to power the 10T SRAM-based CIM architecture could pave the way for low-power, self-sustaining applications in remote or inaccessible locations. Lastly, incorporating robust security features to protect data integrity and prevent unauthorized access is crucial for many applications, warranting further research into developing tailored security protocols. By pursuing these research directions, the novel 10T SRAM-based CIM architecture can continue to evolve, addressing new challenges and expanding its applicability across a broader range of high-performance and energy-efficient computing scenarios.

References

[1] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1,” 2016.

[2] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” 2015.

[3] I. Hubara, M. Courbariaux, and D. Soudry, “Quantized neural networks: Training neural networks with low precision weights and activations,” J. Mach. Learn. Res., vol. 18, pp. 1–30, Jan. 2018.

[4] J. Wang et al., “A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing,” *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, Jan. 2020.

[5] G.K.Chen,R.Kumar, H.E. Sumbul, P.C. Knag, and R. K. Krishnamurthy, “A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS,” *IEEE J. Solid-State Circuits,* vol. 54, no. 4, pp. 992–1002, Apr. 2019.

[6] J. Park, J. Lee, and D. Jeon, “7.6 A 65 nm 236.5 nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers,* pp. 140–142, Feb. 2019.

[7] Y. Chen et al., “DaDianNao: A machine-learning supercomputer,” in Proc. 47th Annu. *IEEE/ACM Int. Symp. Microarchitecture,* pp. 609–622, Dec. 2014,.

[8] S. Han et al., “EIE: Efficient inference engine on compressed deep neural network,” in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA),* pp. 243–254 Jun. 2016.

[9] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, “Stripes: Bit-serial deep neural network computing,” in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO),* pp. 1–12, Oct. 2016.

[10] H. Sharma et al., “Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network,” in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, pp. 764–775, Jun. 2018,.

[11] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers,* pp. 10–14, Feb. 2014,.

[12] B. Moons and M. Verhelst, "An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS," *IEEE J. Solid-State Circuits,* vol. 52, no. 4, pp. 903–914, Apr. 2017.

[13] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018.

[14] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR),* Apr. 2018.

[15] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *Proc. Int. Conf. Learn. Represent. (ICLR),* Apr. 2017.

[16] S. Yin et al., "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE J. Solid-State Circuits,* vol. 53, no. 4, pp. 968–982, Apr. 2018.

[17] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA),* Jun. 2015, pp. 92–104.

[18] C. Eckert et al., "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA),* pp. 383–396, Jun. 2018.

[19] B. Reagen et al., "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA),* pp. 267–278, Jun. 2016.

[20] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers,* pp. 240–242, Feb. 2017.

[21] Gopal Raut, Saurabh Karkun, and Santosh Kumar Vishvakarma, "An Empirical Approach to Enhance Performance for Scalable CORDIC-Based Deep Neural Networks", *ACM Trans. Reconfigurable Technol. Syst*. 2023.

[22] Kim, H., Yoo, T., Kim, T.T.H. and Kim, B., 2021. Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks. *IEEE Journal of Solid-State Circuits,* 56(7), pp.2221-2233, July 2021.

[23] Dhakad, N.S., Chittora, E., Sharma, V. and Vishvakarma, S.K., 2023. R-inmac: 10T SRAM based reconfigurable and efficient in-memory advance computation for edge devices. Analog Integrated Circuits and Signal Processing, 116(3), pp.161-184. 2021.