Compute-in-Memory for MAC and Matrix Operations

Presenter: Durgam Saiprasad
Roll Number: 24m1210

Guide : Prof. Sachin B Patkar

Co-Guide : Prof. Pradeep R. Nair,

Prof. Veeresh Deshpande

High Performance Computing Laboratory
Solid State Devices
Department of Electrical Engineering,
Indian Institute of Technology, Bombay

Outline

Outline

The Big Problem: The "Memory Wall"

- The explosive growth of AI is pushing our computers to their limits.
- We face a fundamental problem: **data movement**, not just computation.
- In traditional systems, we waste massive energy (est. >60%) just shuttling data between the processor and memory.
- This is the **"von Neumann Bottleneck,"** the biggest barrier to more efficient AI.

The Solution: Compute-In-Memory (CIM)

- What if, instead of moving data to the processor, we move the compute **to the data**?
- This is the core idea of **Compute-In-Memory (CIM)**.
- By performing calculations **directly inside** the memory array, we can nearly eliminate this bottleneck.
- This promises to dramatically improve both speed and energy efficiency.

Outline

Literature Review

For my literature review, I looked into a couple of interesting approaches to tackling the memory bottleneck:

Literature Review

For my literature review, I looked into a couple of interesting approaches to tackling the memory bottleneck:

- First, the **SPAR-2** paper. It proposes a **Processing-in-Memory (PIM)** architecture using a SIMD array on FPGAs, targeting ML for IoT devices.

Literature Review

For my literature review, I looked into a couple of interesting approaches to tackling the memory bottleneck:

- First, the **SPAR-2** paper. It proposes a **Processing-in-Memory (PIM)** architecture using a SIMD array on FPGAs, targeting ML for IoT devices.
- Then, the **3D MIGCIM** paper. This uses **Compute-in-Memory (CIM)** with a novel **3D stacking** approach.[IIT Bombay]

Literature Review

For my literature review, I looked into a couple of interesting approaches to tackling the memory bottleneck:

- First, the **SPAR-2** paper. It proposes a **Processing-in-Memory (PIM)** architecture using a SIMD array on FPGAs, targeting ML for IoT devices.
- Then, the **3D MIGCIM** paper. This uses **Compute-in-Memory (CIM)** with a novel **3D stacking** approach.[IIT Bombay]
- Both aim to boost efficiency by computing closer to the data, just using different methods.

Outline

SPAR Architecture

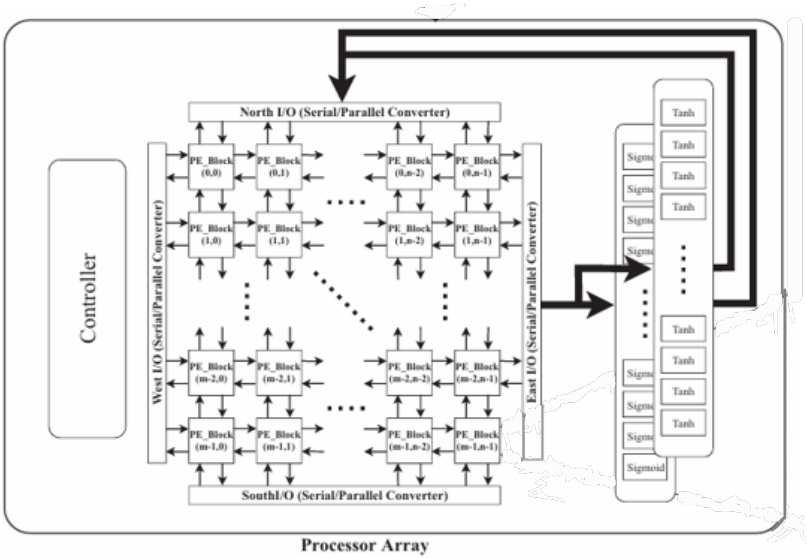
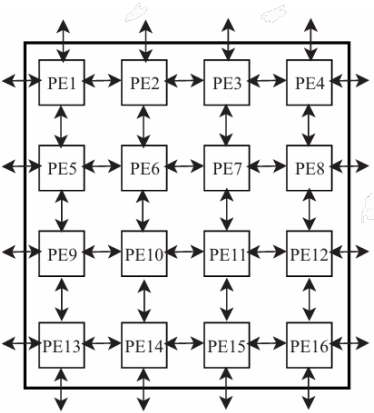


Figure 1: SPAR-2 Architecture Overview

Processing Element

- **Systolic Array:** PEs are connected to four neighbors for directional data flow (North, South, East, West).
- **Core Components:** Each PE contains a bit-serial ALU and a local Block RAM (BRAM) for data storage.
- **Data Handling:** I/O buffers along the array edges manage input vector loading and output vector retrieval.



Outline

Matrix Multiplication on SPAR-2

How SPAR-2 performs Matrix-Vector Multiplication?

- **Setup:** A 4x4 weight matrix W is pre-loaded into the BRAMs of a 4x4 PE array. Each PE (i, j) holds $W_{i,j}$.
- **Input Broadcast:** The 4x1 input vector X is loaded into the **North I/O Buffer**.

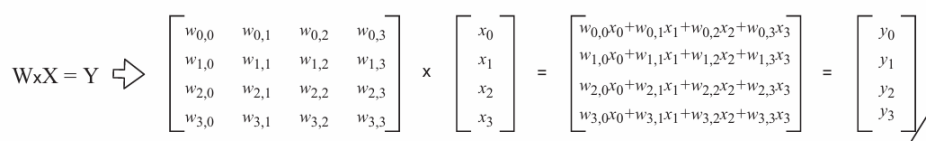


Figure 2: MVM

Matrix Multiplication

- **Matrix and Vector Mapping:** The 4x4 weight matrix W is distributed across BRAMs, and input vector X is replicated vertically into columns using move S operations.
- **Parallel Multiplication:** All Processing Elements (PEs) perform SIMD concurrent multiplication in one instruction, generating partial products in parallel.

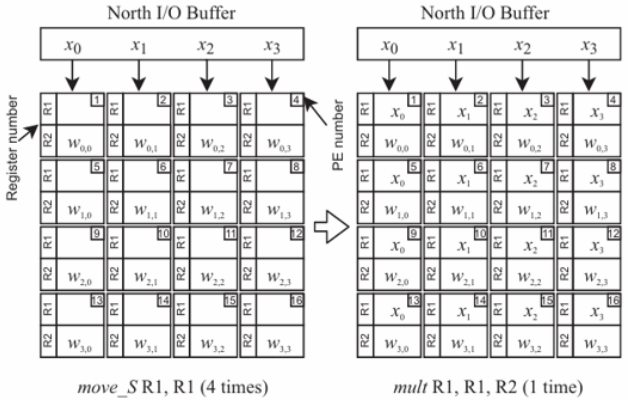


Figure 3: Vector Mapping and Parallel Multiplication

Matrix Multiplication

- **Partial Product Addition:** A binary reduction tree adds the partial products using $\log_2(n)$ additions and (n-1) move operations (e.g., for n=4 $\rightarrow$ 2 adds + 3 moves).
- **Final Output:** The multiply-accumulate result is written to the output buffer, completing the MVM computation efficiently in hardware.

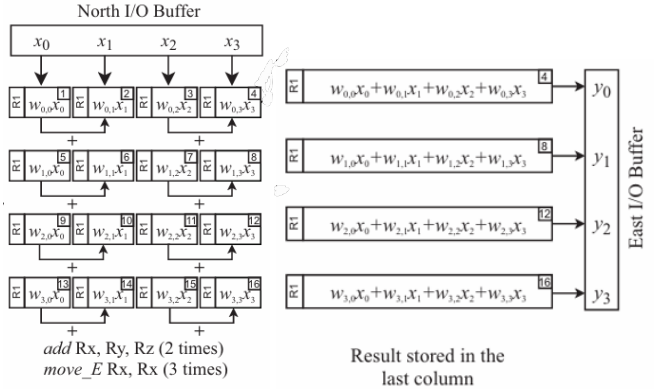


Figure 4: Partial Product Addition and Final Output

Outline

Block Diagram

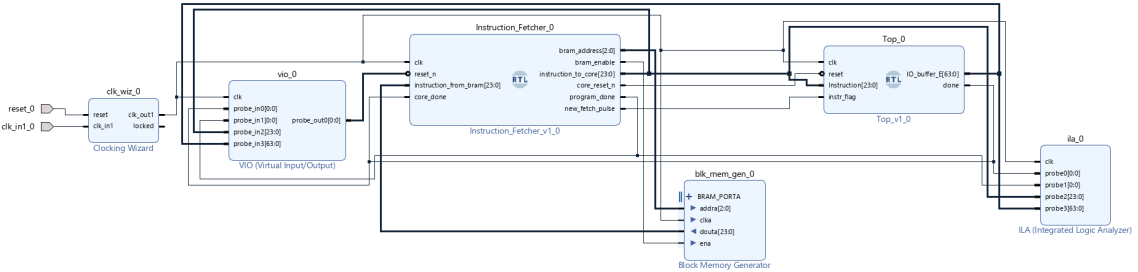


Figure 5: Block Diagram Overview

Implementation Overview

- **Top Module:** Integrates the controller with a 16-PE array, managing data flow and instruction broadcast.
- **Controller:** An FSM-based unit that decodes instructions and generates control signals for all PEs.
- **ALU:** Each PE contains a bit-serial ALU with a Booth multiplier for efficient signed multiplication.

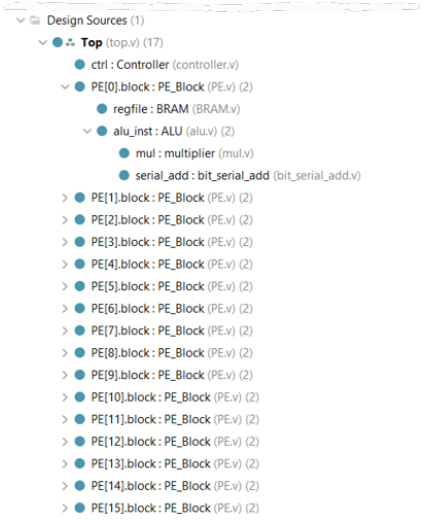


Figure 6: Design Hierarchy.

ALU Implementation: Booth Multiplier

- To efficiently multiply signed numbers in the SPAR-2 ALU, we implemented **Booth's algorithm**.
- Specifically, we used the **Radix-2 version**, guided by insights from the paper "Implementation of Modified Booth Algorithm " .
- This approach examines multiplier bits sequentially to determine whether to add, subtract, or do nothing with the multiplicand.

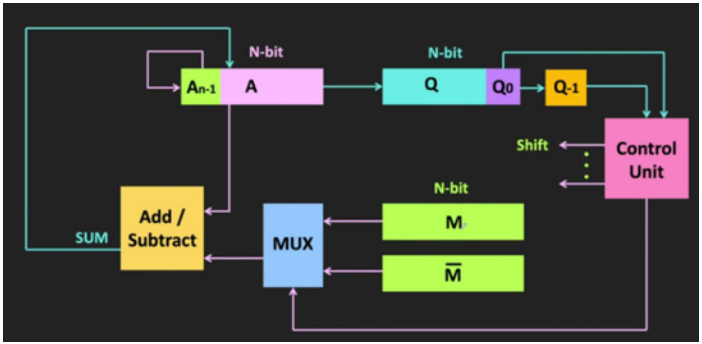


Figure 7: Booth's Algorithm logic.

Output

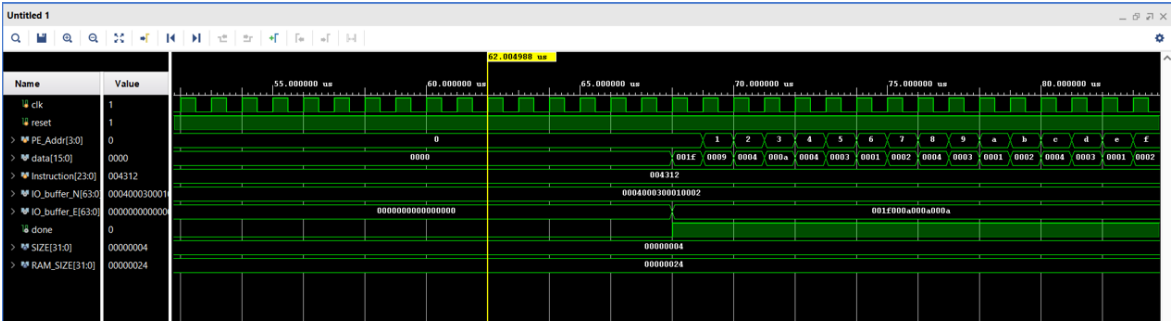


Figure 8: Simulation Waveform

Resource Utilization

- **BRAM Implementation:** The initial design used 11.5 BRAM blocks, resulting in a highly efficient, low-power implementation with minimal logic usage.
- **LUT/FF Implementation:** An alternative design built memory from logic, consuming thousands of LUTs and Flip-Flops, which significantly increased area and power.
- **Conclusion:** The BRAM-based approach is far more scalable and resource-efficient, making it the superior design choice for this architecture.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1 (active)	**constrs_1**	**synth_design Complete!**								**0**	**0**	**0.0**	**0**	**0**
impl_1	constrs_1	write_bitstream Complete!	27.028	0.000	0.031	0.000	0.000	0.222	0	3152	5237	11.5	0	0
Out-of-Context Module Runs														
design_1		Submodule Runs Complete												

Outline

The Paper's Proposal: 3D MIGCIM

- This paper introduces **Monolithic Integrated Gain-cell Compute-In-Memory**: a novel CIM accelerator.
- A key innovation is a **3D Monolithic** design, stacking the memory array directly on top of the logic.
- **On the bottom (FEOL):** Cutting-edge **2nm Silicon GAA logic** for fast peripheral circuits.
- **On the top (BEOL):** A novel **4T2C eDRAM memory array** using **Indium Tin Oxide (ITO)** transistors.

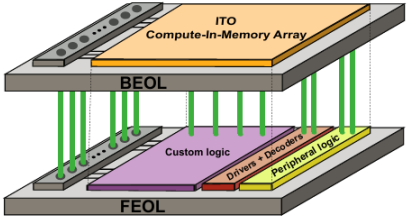


Figure 9: 3D Stack

Architecture

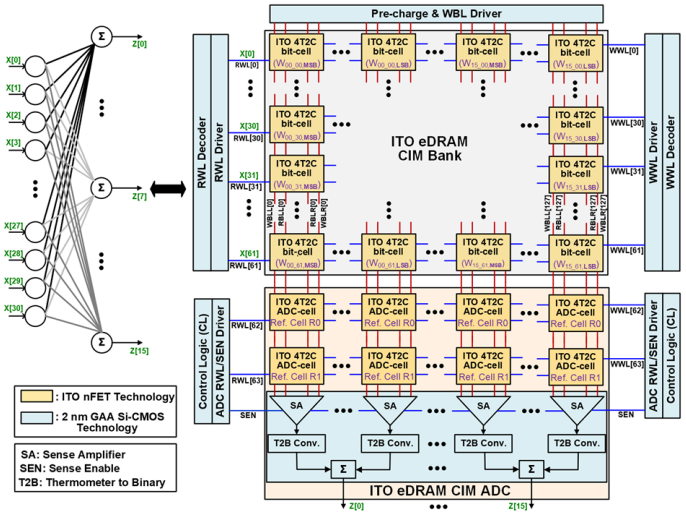


Figure 10: Architecture Overview

Write Operation: Storing Weights

- The 4T2C cell stores ternary weights: **-1, 0, +1**.
- It uses two storage nodes: V_{SNL} and V_{SNR} .
- Each node stores a binary value: High or Low .
- **Storing Ternary Weights:**
  - **'-1':** Write V_{SNL} = 'H', V_{SNR} = 'L'.
  - **'0':** Write V_{SNL} = 'L', V_{SNR} = 'L'.
  - **'+1':** Write V_{SNL} = 'L', V_{SNR} = 'H'.
- **Array Write:**
  - WWL Decoder selects the row.
  - WBLL and WBLR lines set values simultaneously.

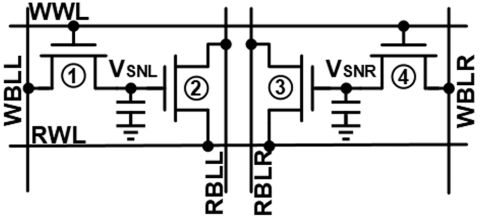


Figure 11: CIM Cell

Output

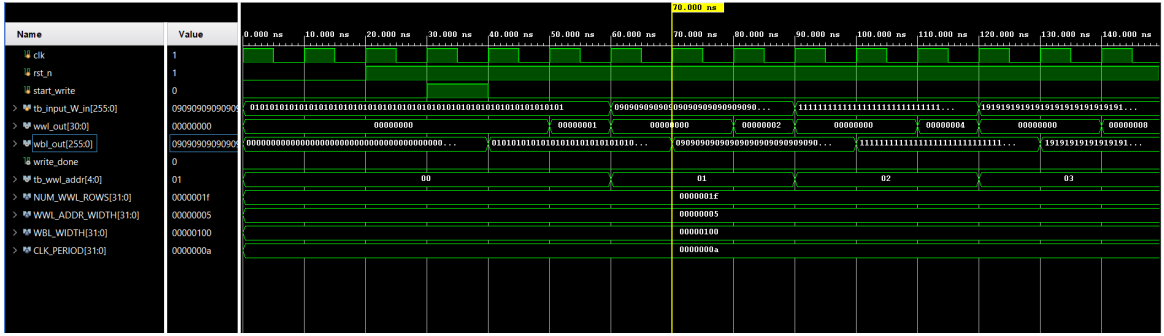


Figure 12: Write Operation

Outline

Positive Weight

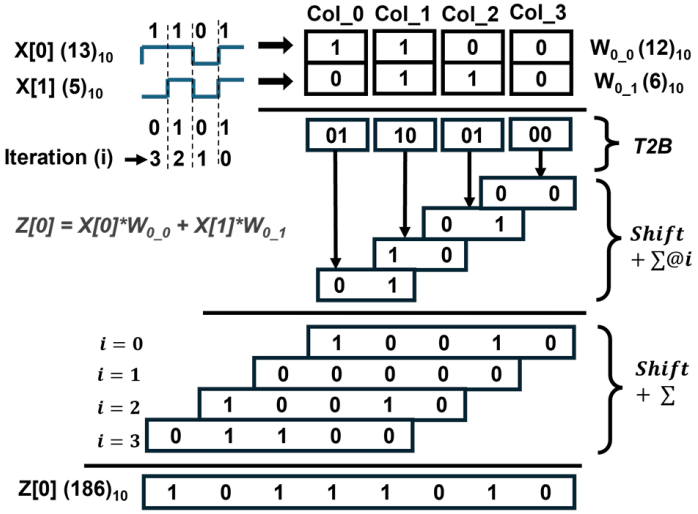


Figure 13: MAC Operation

Negative Weight

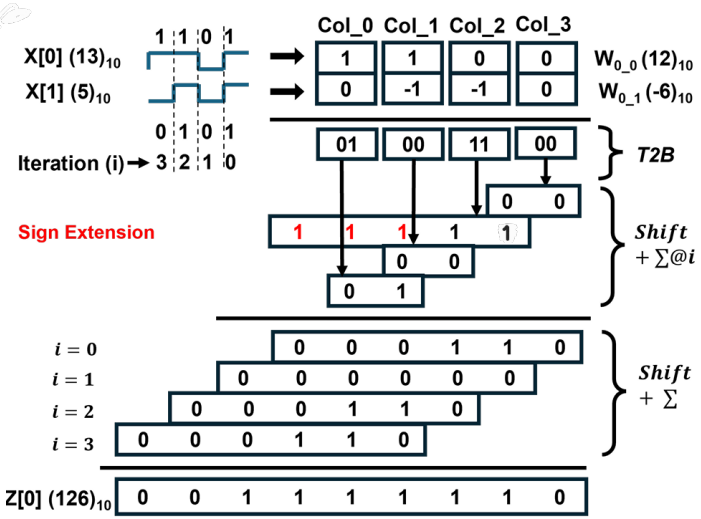


Figure 14: MAC Operation

Input Activation & Analog Compute

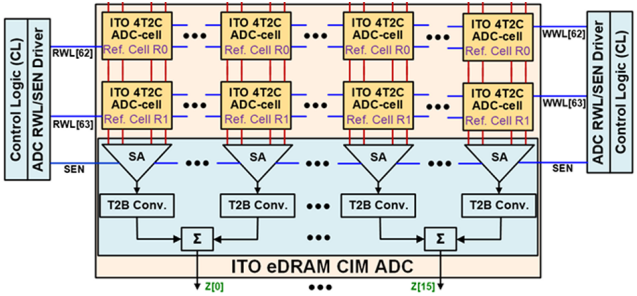
- Input vector bits (X[0] to X[30]) determine which RWLs get pulsed.
- The **RWL Decoder & Driver** send simultaneous input bits to selected rows.
- Each active cell performs its dot product, discharging RBLL/RBLR based on the stored weight (-1, 0, +1).
- The **analog differential voltage** ($V_{RBLL} - V_{RBLR}$) develops across the bitlines, representing the sum.

ADC Magnitude Conversion: The 32 Steps

1. **Sequential Discharge (1 cycle):**
   - ADC Driver pulses the RWL of the *single* ADC reference cell.
   - This changes the voltage difference between RBLL and RBLR by a small step (ΔV).
2. **Sense & Compare (part of 2 cycles hold):**
   - ADC Driver asserts Sense Enable (SEN).
   - SA compares the *new* V_{RBLL} and V_{RBLR}.
   - SA outputs '1' or '0' based on which voltage is higher.
3. **Latch Output (part of 2 cycles hold):**
   - The SA output ('1' or '0') is latched into a DFF.

ADC Magnitude Conversion: Final Steps

- **Thermometer Code Generation:**
  - After 32 steps, the 32 latched SA outputs form a pattern (thermometer code).
  - This pattern reflects how many steps were taken before the voltage difference crossed a threshold.
  - Example (Original $V_{diff} = +3\Delta V$): '111000...0'
- **Binary Conversion (T2B):**
  - The **T2B converter** receives the 32-bit thermometer code.
  - It interprets this pattern to determine the magnitude of the original voltage difference.



Decoder Output

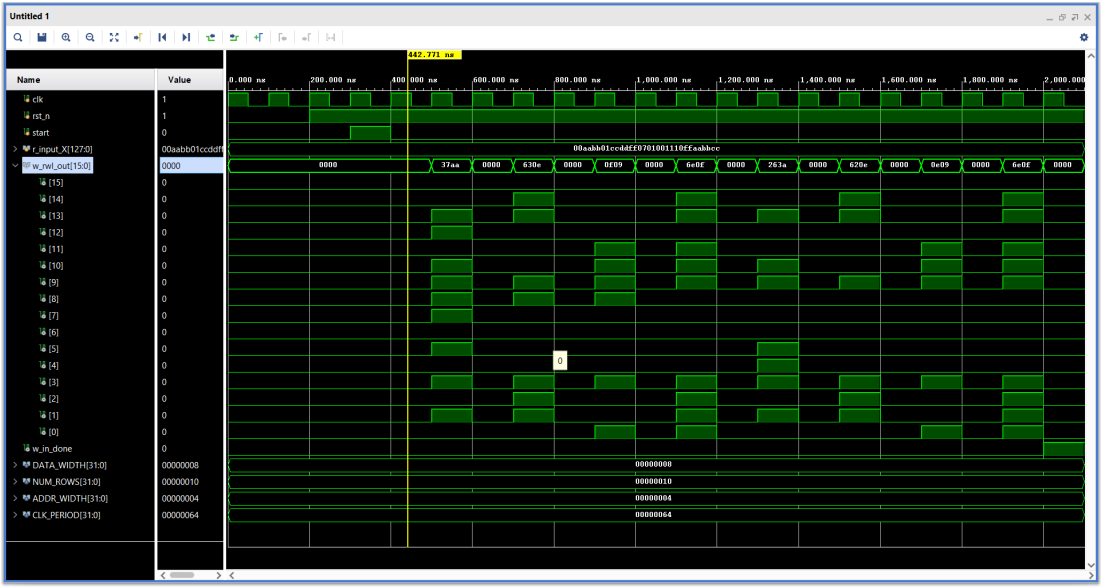
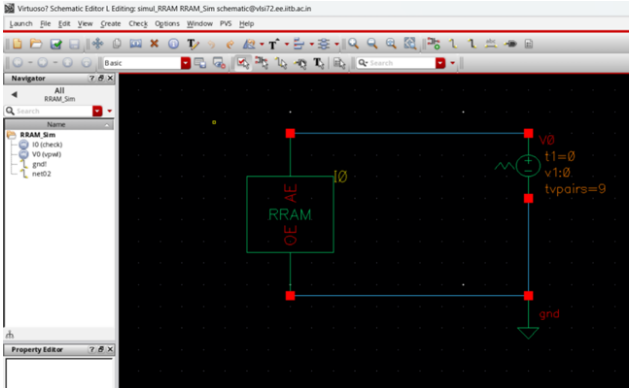


Figure 15: Waveform

Outline

RRAM Device Simulation

- To understand the RRAM behavior, device-level simulations were performed using **Cadence Spectre**.
- We utilized the **JART memristor model** , a Verilog-A implementation based on insights from referenced paper(s).
- This allowed us to successfully simulate the fundamental **I-V (current-voltage) characteristics** of the RRAM cell, verifying its expected electrical behavior.



RRAM Simulation (Transient Response)

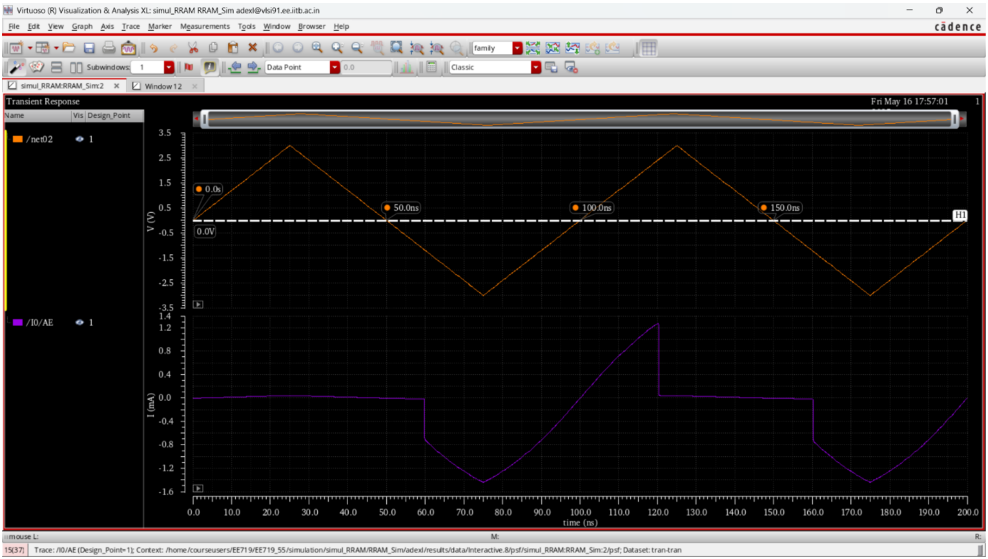


Figure 17: Transient Response (Voltage and Current)

I-V Characteristics

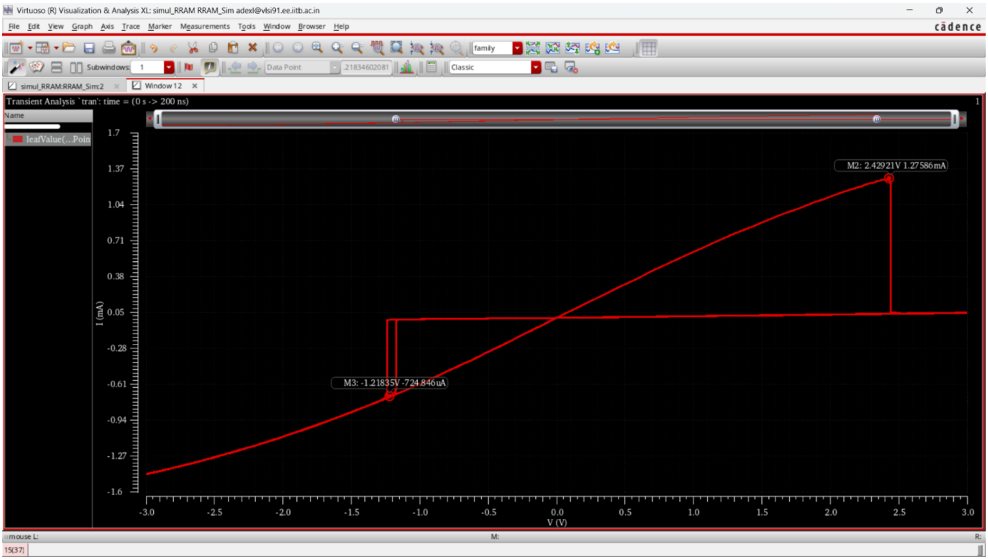


Figure 18: I-V Characteristics of RRAM Cell

Outline

Future Work: Integration and Verification

- Having designed the digital peripheral modules (decoders, drivers, controller).
- Next step is **integration and co-simulation** with the target Compute-in-Memory array to **verify the end-to-end functionality** of the complete CIM macro (write, compute, ADC operations).

Outline

References

1. Suhail Basalama, Atiyehsadat Panahi, 2020, SPAR-2: A SIMD Processor Array for Machine Learning in IoT Devices
2. 3D MIGCIM Paper:3d-monolithic-integrated-4t2c-ito-gain-cell-compute-in-memory
3. Sukhmeet Kaur, Suman, 2013, Implementation of Modified Booth Algorithm (Radix 4) and its Comparison with Booth Algorithm (Radix-2).
4. JART VCM v1 Model User Guide
5. JART Model Website

Thank You!