

DESIGN AND IMPLEMENTATION OF 8T SRAM-BASED IN MEMORY MULTIPLY-ACCUMULATE OPERATIONS AND HAMMING DISTANCE CALCULATION

Prof. P. V Sridevi M.E, PhD¹, Nagulakonda Achyut Sai², Nakka Sravya Sri³, Nelli Preethika⁴, Gilson Durao Nunes⁵

¹HOD, Department of Electronics And Communication Engineering, Andhra University College of Engineering(A), Andhra Pradesh, India

^{2,3,4,5}(Students, Department of Electronics And Communication Engineering), Andhra University College of Engineering(A), Andhra Pradesh, India

Abstract - This paper presents the design and implementation of an 8T SRAM based in memory computing architecture for performing multiply accumulate operations and Hamming distance computation along with logic functionality. Conventional computing systems suffer from high latency and power consumption due to continuous data transfer between processor and memory. To overcome this limitation, the proposed approach performs computation directly within the memory array. An 8 by 8 array of 8T SRAM cells is designed where the decoupled read path ensures improved stability and enables simultaneous multi row activation. The computation is achieved through read bit line discharge, where the voltage variation represents the accumulation of bit level operations. The MAC result represents the number of zero bits, which is further used to implement logic operations such as AND, NOR, XOR and XNOR efficiently. Hamming distance is computed by interpreting bit differences using the same framework. The design is implemented and validated in a schematic capture tool Xschem using circuit level simulations in spice simulator Ngspice with Skywater 130nm technology PDK. The results demonstrate accurate operation with reduced hardware complexity and improved efficiency, making the proposed architecture suitable for low power and high performance in memory computing applications.

Key Words: In-Memory Computing, 8T SRAM, Multiply Accumulate (MAC), Hamming Distance, MAC based Logical operations, SRAM-Based Computing Architecture

1. INTRODUCTION

Modern computing systems are based on the conventional von Neumann architecture, where memory and processing units are physically separated. In such systems, data must be continuously transferred between the processor and memory, resulting in increased latency and power consumption. This limitation is commonly referred to as the von Neumann bottleneck and becomes more severe in data-intensive applications such as artificial intelligence and signal processing.

To overcome this limitation, in-memory computing (IMC) has emerged as a promising approach in which computation is performed directly within the memory array. By reducing data movement, IMC significantly improves speed and energy efficiency while enabling parallel processing.

Static Random Access Memory (SRAM) is widely used for implementing IMC due to its fast access time and compatibility with CMOS technology. However, conventional 6T SRAM cells suffer from read disturbance issues, making them unsuitable for computation-based operations. To address this, the 8T SRAM cell is used, which provides a separate read path and improved stability.

In this work, an 8T SRAM-based in-memory computing architecture is proposed to perform multiply accumulate operations and Hamming distance calculation. The design utilizes read bit line discharge for computation and employs a simplified decoding scheme using comparators and priority encoders. The proposed approach also enables efficient implementation of logic operations directly within the memory.

2. PROPOSED ARCHITECTURE

2.1 8T SRAM Cell

The 8T SRAM cell consists of eight transistors, including a standard 6T storage core and two additional transistors forming a separate read path. This separation ensures that the stored data is not disturbed during read operations which is important as multiple word lines will be activated and data in 6T SRAM cell may flip.

The read operation is performed through the Read Word Line (RWL) and Read Bit Line (RBL), while the write operation is carried out using the Write Word Line (WL) and bit lines (BL, BLB).

2.2 SRAM Array Design

An 8×8 SRAM array is implemented, consisting of 64 memory cells. Each row represents an 8-bit word, and all

columns share common bit lines. A 3:8 decoder is used to select a row to read or write.

During operation:

- RBL is set to a low voltage
- Multiple rows are activated simultaneously
- Parallel computation is achieved

This structure enables efficient in-memory processing by allowing multiple cells to contribute to computation simultaneously.

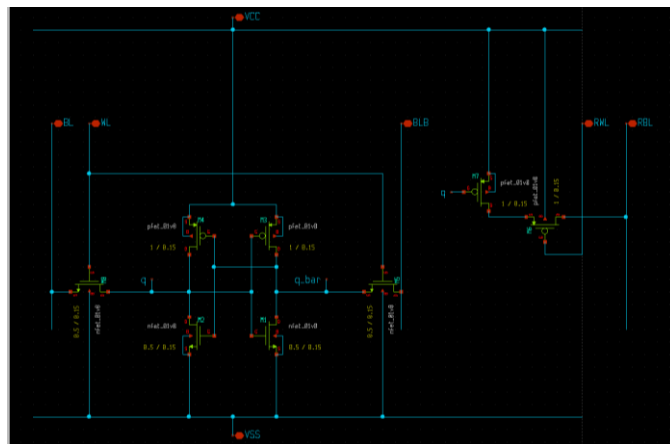


Fig -1: 8T SRAM Cell

2.3 Voltage decoder

The Voltage decoder circuit is used to convert the Read Bit Line(RBL) voltage values to digital outputs. It has of eight voltage comparators. The voltage threshold of each comparator is set according to the RBL voltage corresponding to each MAC count result. The comparator 3 stages: preamplifier stage, regeneration stage and latch stage. It consists of double tail comparator(Fig. 3) for preamplifier and regeneration stages. RS latch(Fig. 4) is used for latch stage

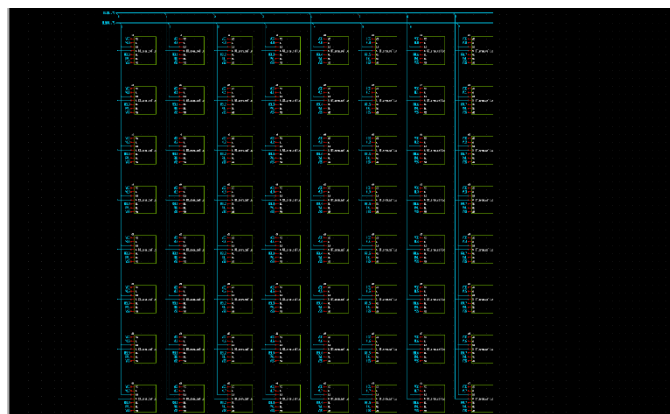


Fig -2: SRAM Array Design

The analog RBL voltage is converted into digital form by comparator. These comparators classify the voltage into different regions corresponding to MAC values.

The RS latch captures the output and holds it constant until the next evaluation cycle. This ensures that the final MAC output remains consistent across the computation.

Finally, a priority encoder generates the final digital MAC output by selecting the highest valid signal.

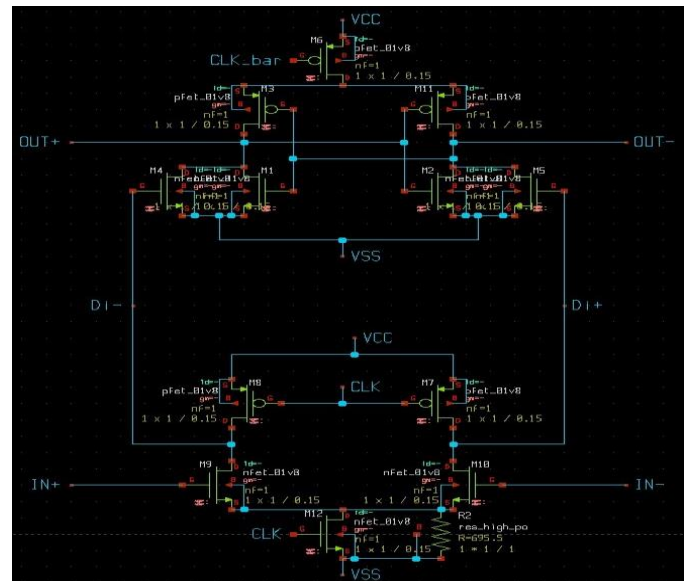


Fig -3: Double Tail Comparator

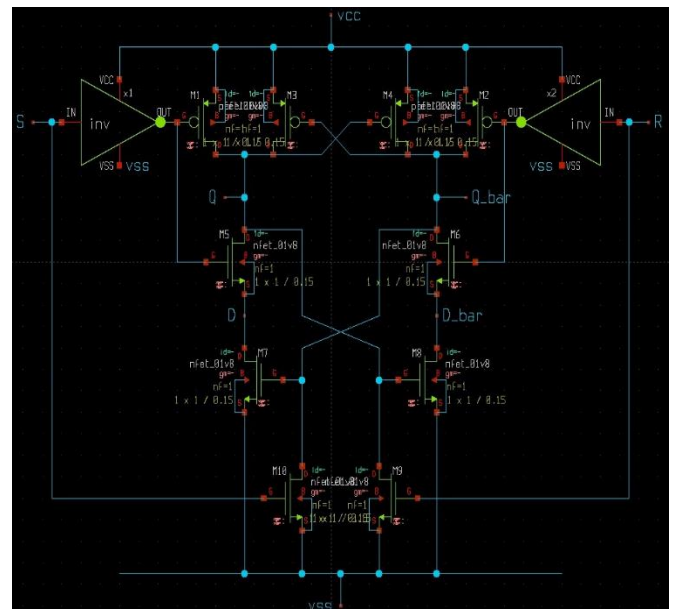


Fig -4: Design of RS Latch

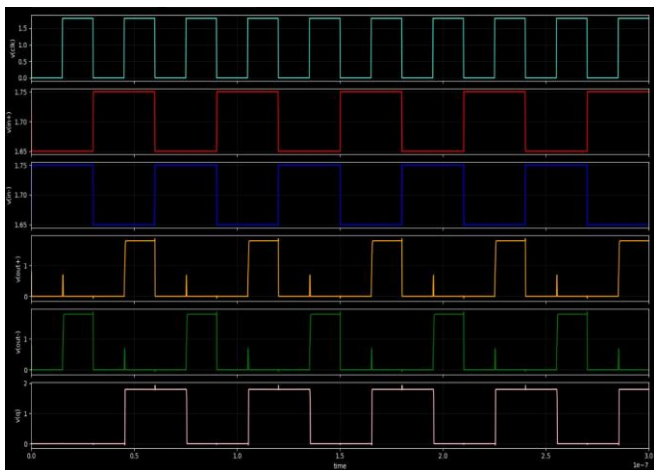


Fig -5: Comparator Waveform

2.4 MAC Operation using RBL

The multiply accumulate operation is performed on a column using the capacitive behaviour of the Read Bit Line. When multiple rows are activated, cells storing logic '0' create a charge path to 1.8V, causing a voltage rise on the RBL. The amount of voltage rise is proportional to the number of active cells. In this design, the MAC output represents the **number of zeros** present in the array.

For example MAC count of 5 means only five cells in the column store logic '0'. MAC count of 8 means all eight cells in the column store logic '0'. MAC count of 0 means none of the cells store logic '0' or all cells store logic '1' in that column.

A Table representing the relation between number of zero stored cells and MAC voltages is shown in Table-1

Table -1: Variation of RBL Voltage with MAC Count

Data in the row	MAC count	RBL voltage(Volts)
11111111	0	1.783
01111111	1	1.755
00111111	2	1.712
00011111	3	1.629
00001111	4	1.477
00000111	5	1.227
00000011	6	0.876
00000001	7	0.472
00000000	8	0.012

The computation process is as follows:

- Step 1: RBL is set to 0V
- Step 2: Multiple rows are activated
- Step 3: Cells storing '0' charge RBL
- Step 4: Voltage rise represents zero count

This analog representation is then converted into a digital output.

3. IMPLEMENTATION OF LOGIC AND HAMMING DISTANCE

3.1 Logic Operations

The MAC output is represented using 9 lines (MAC0 to MAC8), where only one line is active at a time based on the number of zeros. Logic operations can be realised with specific MAC count values

- MAC0 → all bits are 1
- MAC8 → all bits are 0

Logic operations are realised as follows:

- AND operation: Activated when MAC0 is high
- NOR operation: Activated when MAC8 is high

For XOR operation:

- Output depends on parity
- Even number of zeros → output is 0
- Odd number of zeros → output is 1

XNOR is the complement of XOR.

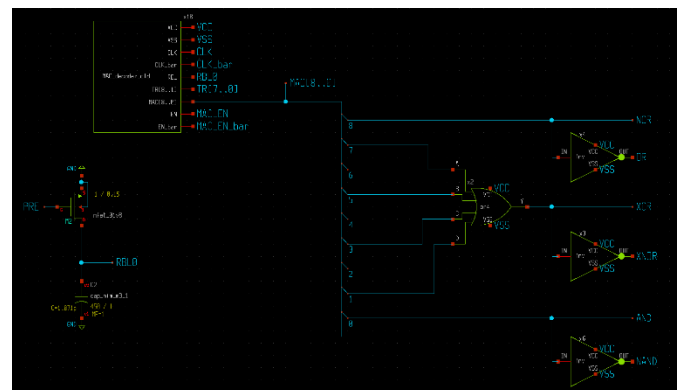


Fig -6: Implementation of Logic Gates

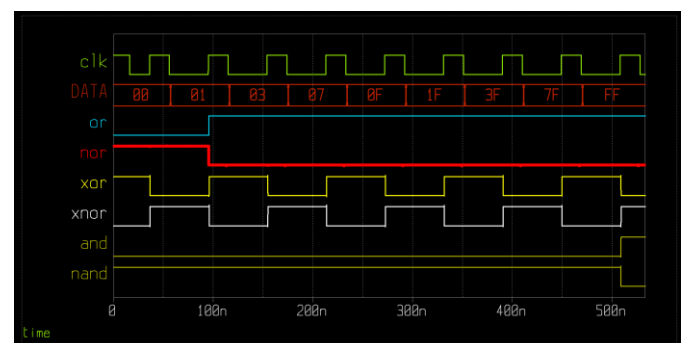


Fig -7: Simulation of realised logic gates

3.2 Hamming Distance Computation

Hamming distance represents the number of bit positions where two binary inputs differ. In this design, Hamming distance is computed using XOR-based comparison using MAC 1 outputs of all columns. We load the two operands in 1st and 2nd rows. We set all other rows to logic '1'.

Each bit comparison produces:

- 1 → if bits are different
- 0 → if bits are same

The XOR result is stored in separate row and MAC count of the row is determined. In the fig 8, ROW_HAM is extra row for storing XOR result of the operands.

The total number of differences is obtained using MAC operation within that row. Thus, Hamming distance is computed efficiently without transferring data outside the memory.

4. RESULTS AND DISCUSSION

The proposed design is implemented using schematic capture tool Xschem and simulated with spice simulator NGSPICE with Skywater 130nm technology PDK. The simulation results verify the correct operation of the SRAM array, MAC computation, and logic implementation.

The RBL waveform shows a proportional voltage drop based on the number of active cells. The comparator successfully detects voltage levels and produces accurate digital outputs.

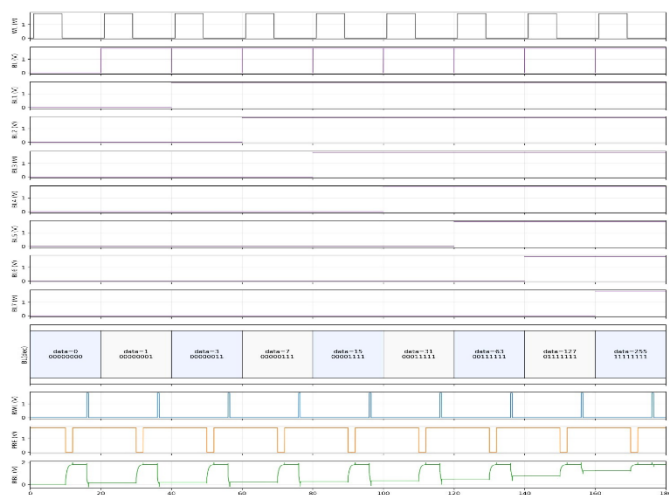


Fig -8: RBL Discharge Mechanism Waveform

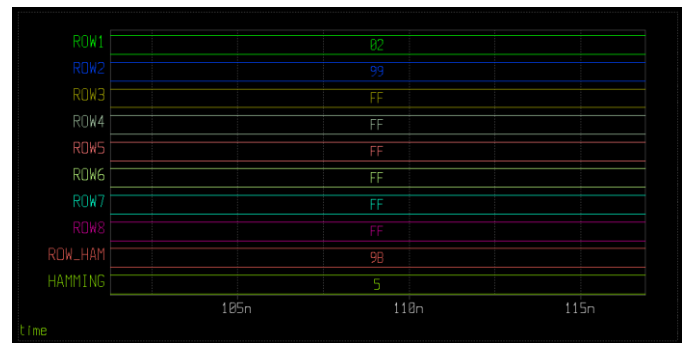


Fig -9: Final output of Hamming Distance Computation

The MAC decoder correctly activates one of the output lines (MAC0–MAC8), which is further used to implement logic operations. XOR, XNOR, AND, and NOR operations are verified using simulation results.

The proposed design demonstrates efficient computation with reduced hardware complexity along with logic-based decoding.

5. CONCLUSION

This paper presents an 8T SRAM-based in-memory computing architecture for performing multiply accumulate operations, logic operations and Hamming distance computation. The design utilizes RBL charging behavior for computation and employs a decoding approach using comparator and priority encoder. Every logic function in the proposed design is derived from a single MAC computation. This helps minimize both chip area and power usage, making the approach well-suited for edge AI applications.

The proposed system reduces data movement, improves parallel processing capability, and minimizes hardware complexity. The simulated results in 130nm process confirm correct operation and demonstrate the effectiveness of the design for low power and high performance in-memory computing applications.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the Department of Electronics and Communication Engineering, Andhra University College of Engineering, for providing guidance and support throughout the project.

REFERENCES

[1] B. Khailany, H. Nguyen, and R. Ho, "In-memory computing architectures for machine learning applications," *IEEE Micro*, vol. 40, no. 6, pp. 50–58, Nov. 2020.

- [2] S. Kang, W. Zhao, and Y. Cao, "SRAM-based in-memory computing for artificial intelligence: A review," *IEEE Transactions on Circuits and Systems I*, vol. 68, no. 7, pp. 2817–2830, July 2021.
- [3] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, Jan. 2019.
- [4] M. E. Sinangil, B. Erbagci, and A. S. Yavuz, "A 28nm SRAM-based in-memory computing architecture for deep neural networks," in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 64–66.
- [5] M, A. K. and S, S. M., "A Novel 8T SRAM-Based In-Memory Computing Architecture for MAC-Derived Logical Functions", *arXiv e-prints*, Art. no. arXiv:2512.00441, 2025 doi:10.48550/arXiv.2512.00441.
- [6] K. Zhang et al., "A 65nm 8T SRAM-based computing-in-memory macro for low-power neural network applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1445–1456, June 2020.
- [7] Z. Lin et al., "In situ storing 8T SRAM-CIM macro for full-array Boolean logic and copy operations," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 5, pp. 1472–1486, 2022
- [8] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, Apr. 1950.
- [9] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York, NY, USA: McGraw-Hill, 2001.
- [10] M. Alioto, "Energy-efficient SRAM design for modern digital systems," *IEEE Circuits and Systems Magazine*, vol. 18, no. 1, pp. 30–45, 2018.