2026 Design, Automation & Test in Europe Conference (DATE 2026)

# AURORA - AUtomated 8T SRAM Wired-OR Logic Array for Boolean-Based Machine Learning

Komal Krishnamurthy*, Marcos L. L. Sartori*†, Shengyu Duan*†, Alex Yakovlev*, Rishad Shafik*
*Microsystems Research Group, Newcastle University, UK
{k.krishnamurthy3, Marcos.Sartori, shengyu.duan, alex.yakovlev, rishad.shafik}@newcastle.ac.uk
†Indicates equal contribution

***Abstract*—In-memory computing (IMC) addresses the data movement challenges of traditional von Neumann architectures. Among the available memory technologies, SRAMs are fast, robust, and CMOS compatible, making SRAM-based digital in-memory computing suitable for Boolean machine learning applications. However, most SRAM-based digital IMC architectures require specialized addressing strategies and demonstrate wired operations at schematic level. It is essential to analyze operand scalability for wired operations under layout-dependent parasitic effects. Moreover, generating custom memory macros is complex: proprietary or open-source memory compilers target standard memories and often require information obtained through tedious methods. This paper proposes AURORA - an automated 8T SRAM IMC array that performs multicast wired-OR across the array. AURORA primarily explores the impact of concurrent row switching on varying memory sizes through post-layout analysis. It is achieved through a custom memory compiler written in SKILL. The compiler automates arbitrarily sized memory array generation, which are thoroughly analyzed for functionality, performance, area, and power consumption. The array's wired-OR functionality aligns naturally with the Boolean-based inference and is demonstrated for the Tsetlin Machine (TM) as a case study. The proposed architecture offers two operation modes: i) memory mode, for standard read/write operation; ii) wired-OR/inference mode. AURORA achieves $10^3\times$ higher throughput and energy efficiency than similar architectures. It is 89 times faster, consumes $772\times$ less energy per data point and provides $37.5\times$ more TOPS/W than memristor-based TM implementations. It also has similar latency but uses $22\times$ less energy and offers $22.3\times$ higher TOPS/W compared to digitally synthesized TM implementations.**

***Index Terms*—Machine learning, In-memory compute, 8T SRAM, wired-OR, Tsetlin Machine.**

## I. Introduction

In-memory computing addresses the data transfer bottleneck in traditional computing systems by performing parallel computations within the memory. These attributes increase throughput and reduce energy consumption, making IMC ideal for ML edge applications [1], [2]. Several IMC techniques have been proposed in the literature based on different memory technologies such as non-volatile memories (NVM) and CMOS based memories. However, most state-of-the-art IMC architectures have been designed for multiply and accumulate operations, often in the analog domain. [3]–[8]. The main challenges include noise susceptibility, requiring complex peripherals and circuit calibration that limit scalability. This initiated a shift from MAC based IMC architecture toward boolean-based designs [9], [10].

To date, only a few works have explored IMC architectures for Boolean operations [11]–[21]. Some of these utilize memristors (resistive RAM) [11]–[13]; however, memristors introduce complexities in circuits and manufacturing process overlooking the natural suitability of CMOS technology. The CMOS-based IMC architectures—mainly built on SRAM—can efficiently support Boolean operations using wired logic schemes [17]–[21]. These architectures are limited to schematic only, employing specialized sense amplifiers (SA) [17]–[19] and segregated bitlines (differential bitlines BL/BLB for standard reads and read bitlines (RBL) for logic compute), requiring specialized addressing strategies [21]. Therefore, the impact of wired operations involving concurrent row switching on long metal wires (bitlines) needs to be investigated. Moreover, efficient placement and routing of memory elements in memory array layouts are crucial and can be facilitated by memory compilers. These compilers, offered by semiconductor foundries, companies like Synopsys, or open-source options like OpenRAM, generate hard macro memory arrays [22]. While open-source compilers allow porting to different PDKs, they complicate custom IMC array processes and limit design exploration for trade-offs in concurrent row operations.

This work proposes AURORA, an automated 8T SRAM wired-OR logic array for boolean ML. The IMC array is based on conventional 8- transistor SRAM (8T SRAM), capable of performing Boolean operations across concurrent rows while retaining standard memory functionality. Unlike existing 8T SRAM-based IMC architectures for Boolean logic, the proposed design eliminates the sense amplifier and incorporates a logic gate to differentiate between the standard read operation and wired-OR functionality on the same bitline. This research primarily explores how varying memory sizes are impacted by concurrent row switching through post-layout analysis. It is achieved with the support of a custom memory compiler that automates array generation. The compiler accepts user-defined parameters—such as array dimensions for layout generation and simulation input in binary format for testbench creation—enabling flexible and efficient design. The wired-OR functionality of the array is evaluated as a case study using the TM model on the MNIST dataset, demonstrating its scalability, low latency, and favorable area and power characteristics. The key contributions are as follows:

- An 8T SRAM design optimized for stability, compact area, and large-scale wired-OR capability.
- A design exploration studying the impact of concurrent row operations on different memory array sizes through post-layout analyses.
- A design automation flow that accepts user parameters

978-3-9826741-1-7/DATE26/

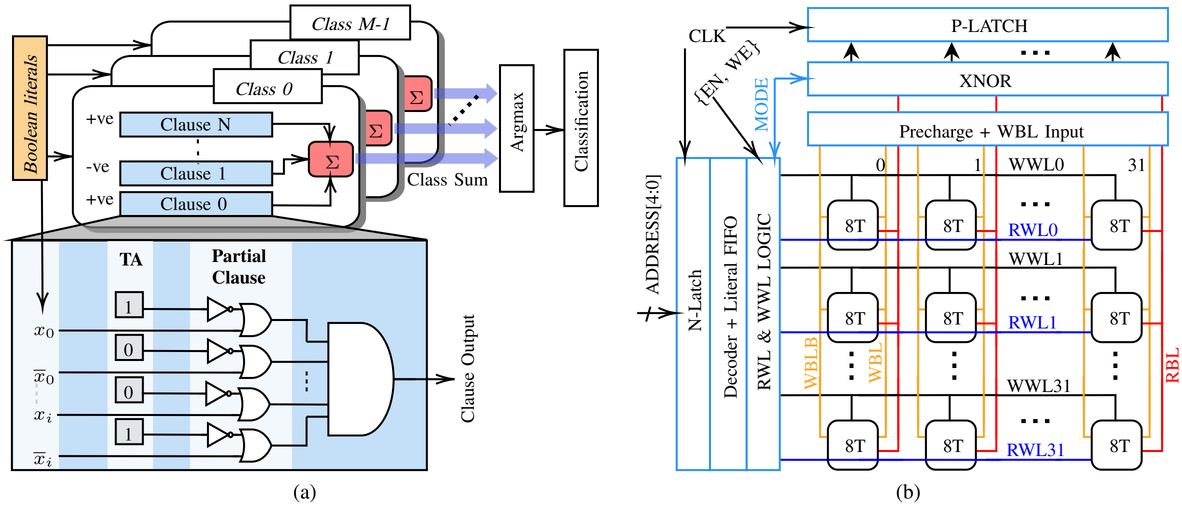



Fig. 1: (a) A TM inference architecture, and (b) AURORA 8T SRAM array architecture for bank size 32x32.

to generate DRC and LVS clean IMC array layout and testbench for its validation through simulation.

- A demonstration of the IMC array, with TM workload as a case study shows an order-of-magnitude improvement in energy efficiency and throughput compared to CMOS logic cell and memristor-based TM implementations.

The remainder of the paper is organized as follows. Section II introduces the fundamental working principle and architecture of TM. Section III presents the proposed AURORA IMC array for Boolean-based ML, including evaluations of optimal transistor sizing and bank configurations. Section IV outlines the design automation flow for IMC array generation. Section V details experimental results using the IMC array for TM inference. Finally, Section VI concludes the paper.

## II. Background

The proposed IMC architecture targets inference workloads. In this work, it is instantiated with a Tsetlin machine inference that does not require additional circuits or modifications. For completeness, a brief overview of the TM inference process is provided.

### A. Tsetlin Machine

The TM recognize patterns from Boolean data, represented using propositional logic. A Booleanization process is applied before both training and inference. Typically thresholds are applied on raw data to generate literals, which are Booleanized features and their complements. A TM is composed of a predefined number of clauses per class. All clauses are evenly split into positive and negative polarities, contributing to either supporting or opposing a class definition. TM training is centred around Tsetlin Automata (TAs), each of which determines whether a specific literal should be included in or excluded from a clause. After training, each clause captures a pattern formed by a conjunction (AND) of selected literals, corresponding to a specific classification task.

Fig.1 (a) illustrates the architecture of a TM during inference. Each partial clause responds based on the value of a specific literal and the corresponding TA action: it outputs 0 only when a literal with value 0 is included in the clause (i.e., literal = 0 and TA = 1). For a given datapoint (sample), classification is performed by summing all clause outputs using a popcount. In binary classification, the final decision is made through majority voting between the total outputs of positive and negative clauses. For multi-class classification, each class is represented by a separate TM, and the predicted class is determined by selecting the TM with the highest popcount (argmax). The remainder of the paper focus on the proposed IMC architecture and TM inference is a workload to demonstrate functionality and asses efficiency.

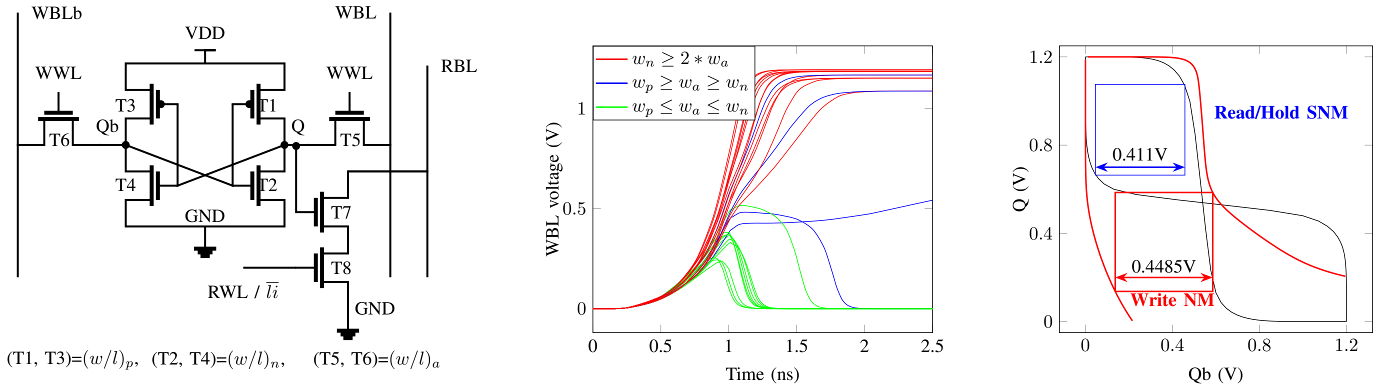
## III. In-memory 8T SRAM Architecture

The AURORA architecture is illustrated in Fig. 1 (b). As mentioned in the introduction, this architecture enables multicast wired-OR functionality across the array and offers two operational modes. The array can be addressed similarly to state-of-the-art 6T SRAMs. During standard read operations, the literal FIFO buffers are masked by custom logic cells. In contrast, during wired operations, these buffers are used to select operands or SRAM cells.

This section details the design of the 8T SRAM cell, emphasizing optimal transistor sizing for enhanced stability and reduced transient time. It also covers wired-OR and custom logic cells that differentiate between normal memory and inference reads, as well as the effects of simultaneous switching on concurrent rows.

### A. 8T SRAM Cell

The 8T SRAM is structurally similar to the standard 6T SRAM with two additional transistors (Fig.2 (a)). The 8T SRAM features distinct bit lines and word lines to select a cell during a write and read operation. The write operation is identical to that of the 6T SRAM. The write word line (WWL) selects the cell and the write bit lines (WBL/WBLB) write a ‘0’ or ‘1’ into the cell. During read operation, the read bit line (RBL) is precharged to VDD and read word line (RWL) selects




(a) (b) (c)

Fig. 2: (a) An 8T SRAM schematic, (b) write Bit line (WBL) variation during write for different transistor widths, (c) Read/Hold static noise margin and Write noise margin for 8T SRAM.

the cell from which the stored bit is read. The read circuit in the 6T SRAM cell requires an op-amp, which compares the voltage difference between BL and BLB. In contrast, the 8T SRAM cell is read through RBL using a XNOR. This memory architecture is constructed using TSMC 65nm CMOS mixed-signal RF PDK.

### B. Impact of W/L Ratio on 8T SRAM Cells

Cell stability is essential to ensure reliable read and write operations. It is achieved by selecting appropriate widths and lengths $(W/L)$ of the transistors. This section examines the impact of the transistor widths of the access transistor($wa$), pull-up transistor($wp$), and pull-down transistor($wn$) on the write operation. The channel length is kept to its minimum of $65nm$. The widths of the transistors range from $120nm$ to $240nm$. This analysis focuses on writability and illustrates WBL variations. For a successful writing process, the relationship $wp \leq wa \leq wn$ must hold. However, while writing a '0' to previously stored logic '1', if the width($wn$) exceeds twice that of the access transistor width (i.e. $wn > 2 * wa$), the bit line voltage rises to VDD, resulting in a write failure. Similarly, if the pull-up width ($wp$) is larger than $wn$ and $wa$, the write operation will also fail. The appropriate transistor widths with minimize voltage spikes occurs when all transistor widths are set to minimum ($wp = wa = wn = 120nm$) Fig.2 (b). Similarly, the transistor widths of T7 and T8 were varied from $120nm$ to $520nm$ to analyze their effects on the time taken to discharge a charged RBL. The widths of the transistors do not significantly impact the transient delay, and to minimize area, both T7 and T8 were set to the smallest width of 120 nm. Fig.2 (c)

SRAMs may experience noise in bit lines, leading to bit flips. The static noise margin (SNM) quantifies the maximum DC noise a cell can tolerate without bit flipping during read and hold operations. While write robustness is measured by write noise margin (WNM). The isolated read path improves read stability with a read and hold SNM of 0.411V, and a comparable WNM of 0.44V (Fig.2 (c)) [23]. Unlike 6T SRAM, which are susceptible to destructive reads without appropriate transistor sizing, the 8T SRAMs provide stable reads due to their isolated bit lines. This results in approximately 30% increased area per cell.

### C. 8T SRAM Wired-OR / Propositional Logic

The wired-OR functionality in the proposed architecture operates in two modes: i) standard memory mode (RBL inverted) and ii) wired-OR mode (RBL not inverted). In wired-OR mode, multiple rows are read simultaneously, behaving like a NOR operation. It satisfies TM's partial clause logic without additional structural changes (Fig.1 (a)). The Tsetlin automaton (TA) determines the 'include (1)' and 'exclude (0)' actions during training, which remain fixed during inference. Each memory cell represents a partial clause, with boolean literals controlling the RWLs. The output becomes '0' only when $li = 0(\overline{li} = 1)$ and $TA = 1$ (Fig.3).

As the 8T SRAM array supports both standard read and wired-OR operations, and it is essential to distinguish between these two modes. Custom logic cells are designed to differentiate operating modes and provide the necessary control signals. Two logic cells are appended to the memory array: one controls the WWL and the other controls the RWL. The common inputs to these logic cells are chip enable (EN), write enable (WE = $W/\overline{R}$), clock (CLK), and address (ADDR). Regardless of the operating mode, read and write operations can only be performed when EN = 1 and the clock signal is at a positive level. The logic cell that controls WWL is a four-input AND gate $(EN\&WE\&CLK\&ADDR)$.

The logic cell responsible for RWL has two additional inputs that specify the mode of operation (MODE) and a literal input

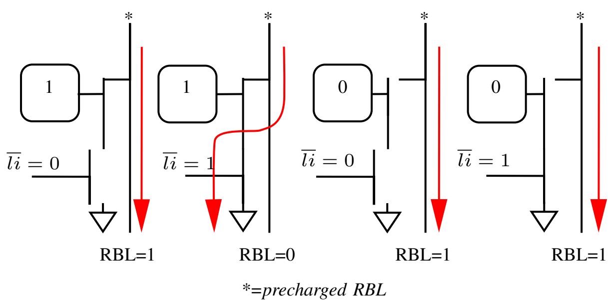



Fig. 3: TM partial clause cases in 8T SRAM cell

($\overline{li}$). For a normal read operation, MODE = 0, and RWL is driven by ADDR to read the data stored in the addressed memory cell. In this case, the output from the read bitline (RBL) needs to be inverted. Conversely, for inference reads, MODE = 1, and RWL is driven by $\overline{li}$, masking the ADDR. The MODE and RBL are XNOR'ed this allows for inverted RBL during standard read and not inverted for wired-OR.

$$RWL = CLK.EN.\overline{WE}.(MODE.\overline{li} + \overline{MODE}.ADDR) \tag{1}$$

### D. Impact of Column Size for Concurrent Readability

This section investigates the impact of concurrent row switching on RBL during wired-OR read. Alternatively, as each memory cell can perform a partial clause operation and each column represents a clause. The conjunction of partial clauses can be performed at once by activating all the RWL ($\overline{li}$) Fig. 1 (b). This arrangement causes RBL to discharge, even if a single discharge path exists in the column. The discharge time decides the inference speed.

The potential discharge paths in RBL is equal to the number of rows and controls the discharge time. The discharge and precharge times are affected by the parasitics. To analyze the effect on discharge and precharge time, SRAM column with different rows is simulated with a varying percentage of cells/rows meeting the discharge condition *TA=1, li=0 (activated cell)*. This analysis highlights the limitations on the row size when all RWLs are switched on simultaneously. Three memory sizes of 32, 256 and 512 rows are simulated to observe discharge and precharge times when 25%, 50% and 100% of the cells meet the discharge condition. The RBL behaviour is evaluated when activated cells are located in different positions in the memory column (Fig.4 (a) & (b)).

Post-layout simulation is performed at the typical corner with 1.2V, 27$^{o}$C, and a 5 ns read and precharge pulse. The discharge and precharge times are proportional to the number of rows as it increases the overall capacitance in the RBL. The observations show decrease in discharge time with more activated cells for the row sizes 32 and 256 (Fig. 4 (a)), but, is unlikely for 512 rows when completely activated. Simultaneous switching and increased capacitance lead to slower discharge. As RBL is a long metal wire and its resistance increases with length, ($R \propto L$), affects the discharge time. Likewise, precharge time (0 to VDD) of RBL should remain constant regardless of the activated cells(Fig.4 (b)). as RBL is disconnected from ground through RWLs. However, the precharge time tends to increase when all 512 cells are activated.

The worst case occurs when only one cell in a column meets the discharge condition. The discharge time is significant and takes around 0.6 and 2.3 ns for 32 and 256 cell rows, respectively. The discharge and precharge time for 512 rows is significantly large and fails for clock frequency of 100MHz. Depending on the constraint, row size of 256 is preferred for functional correctness.

The power consumption slightly increases with row sizes, but it is not significantly affected by the number of discharge paths. The power consumption values for row sizes of 32, 256, and 512 are 0.936 $\mu$W, 1.56 $\mu$W, and 2.13 $\mu$W, respectively.

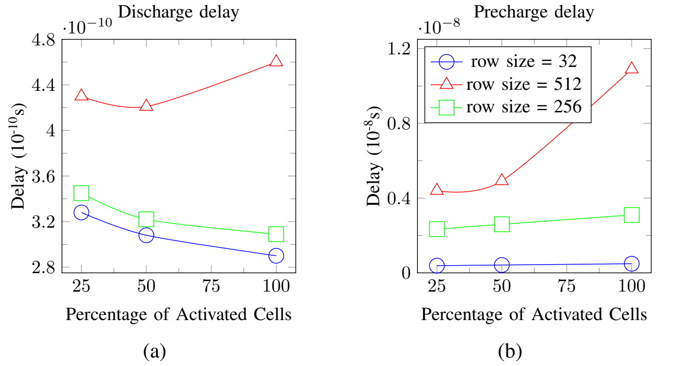


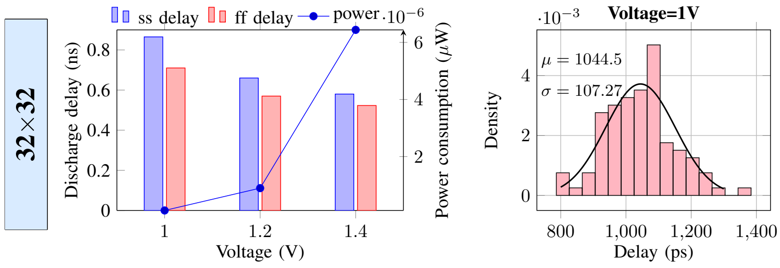

Fig. 4: RBL (a) discharge time and (b) precharge time for 25%, 50%, and 100% of cells meet discharge condition in a column.

### E. PVT and Monte Carlo Analysis of Memory Banks

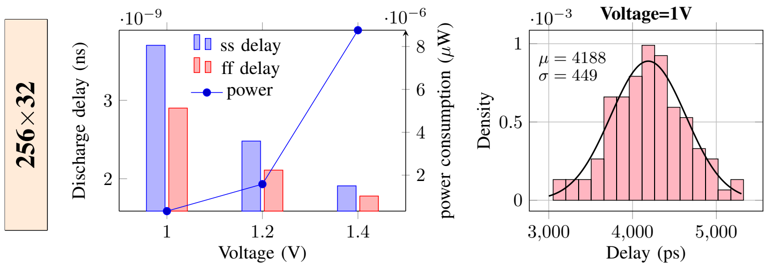
With the appropriate row size based on previous analysis, bank sizes of 32×32 and 256×32 are chosen. The worst-case scenario occurs when only one discharge path is available at the top of the banks. The memory banks are simulated for the process *(slow-slow (ss) and fast-fast (ff))*, voltage *(1V, 1.2V, and 1.4V)*, and temperature *(27$^{o}$C)*, Fig. 5. The discharge times for both memory bank sizes indicate an expected outcome of ff corner discharging slightly faster than the ss corner. The power consumption values for both the ss and ff processes are nearly identical, so an average is used to represent them. For the 32 $\times$ 32 bank size, the average power values are approximately 0.11 $\mu$W, 0.9 $\mu$W, and 6.4 $\mu$W. Similarly, for 256 $\times$ 32 the average power values are 0.32$\mu$W, 1.6$\mu$W and 8.7$\mu$W. The worst-case delay occurs in the *ss* process. Further, through Monte Carlo simulations, the memory banks are tested for device-to-device mismatch and variations in the *ss* process.

Monte Carlo simulations for 32×32 banks at $V_{DD} - \Delta V$ (1V) demonstrate a mean delay of 1 ns, with a standard deviation and variability of $\pm$0.108 ns and $\pm$10.3%. For $V_{DD}$ (1.2V), the mean delay is 0.712 ns. Its standard deviation and variability are $\pm$0.06 ns and $\pm$8.8%. In the case of $V_{DD} + \Delta V$(1.4V), the mean delay is 0.56 ns, with a standard deviation of $\pm$0.045 ns and variability of $\pm$8%. These distributions exhibit a slight positive skewness of approximately 0.2, 0.1, and 0.09, suggesting that most delay values are below the mean. However, the distributions remain nearly symmetric, indicating stable circuit behaviour.

For the 256×32 memory bank, the mean delay at $V_{DD} - \Delta V$ is 4.2 ns, with a standard deviation of $\pm$0.45 ns and a variability of $\pm$10%. For $V_{DD}$, the mean delay is 2.79 ns, with a standard deviation of $\pm$0.27 ns and a variability of $\pm$9%. For $V_{DD} + \Delta V$, the mean delay is 2.16 ns, with a standard deviation of $\pm$0.19 ns and a variability of $\pm$8%. The skewness of these distributions is close to zero, with values of -0.017, 0.08, and 0.02. This suggests that the delay distributions are approximately symmetric and are closer to an ideal Gaussian distribution, indicating no bias toward unusual slow or fast transients. This consistency points to stable timing behaviour under variations in process and voltage.




(a)




(b)

Fig. 5: Discharge delay and power under process and voltage variations are analyzed for memory banks of (a) 32×32 and (b) 256×32. For each fixed supply voltage in both cases, the ratio of the mean discharge delay to its standard deviation remains consistent. Similarly, under each fixed process variation corner, the discharge delay exhibits a comparable trend as the supply voltage increases. These results indicate that the impact of process and voltage variations remains generally consistent across different bank sizes.

## IV. Automated Array Generation

Generating memory arrays is a complex process. It requires careful execution to minimize area wastage, ensure efficient placement and routing, and achieve Design Rule Check (DRC) and Layout Versus Schematic (LVS) cleanliness. A custom memory compiler is proposed to automate memory placement and testbench creation based on user inputs for simulation using Spectre, Fig.6. The memory array is organized into user-defined memory banks, with customizable memory rows and columns.

The proposed architecture employs full-custom cells; therefore, custom standard cells were created. Initially, the necessary schematics were designed using the Cadence Virtuoso Schematic Editor and were analyzed for functionality, performance, and variations based on parameters, e.g. transistor size. The main objective is to achieve the correct functionality with reduced latency, power consumption, and area. After schematic simulations and considering design rules, layout cells were created and validated through LVS checks and simulations.

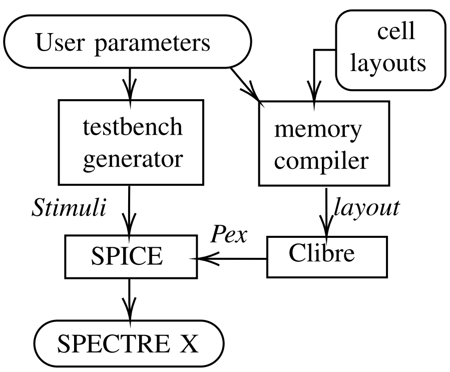



Fig. 6: Automation flow

The SKILL programming language was then used to build the memory array, perform power planning and necessary routing, allowing customization according to user-defined requirements. Additionally, the memory compiler can generate decoder circuit using dynamic logic for the memory array. Depending on the type of integration (digital or analog on top), bare memory or memory with the decoder can be generated.

Using Siemens Calibre tool, parasitic netlist in SPICE was extracted from the compiled memory layout for post-layout simulation. Furthermore, the simulation environment in **SPICE** is generated by integrating Python in the front-end. It requires the memory size, memory states, addressing inputs, operation mode, and the duration of the read pulse as user inputs in binary format. The generated testbench defines the stimuli to simulate the circuit. For fast simulation without decoders, the memory cells can be simulated using nodeset conditions. The results are then analyzed to verify the impact of parasitic effects and overall circuit functionality.

## V. 8T SRAM Implementations and Comparative Evaluations

The previous sections show the array design, automation process, and its analysis for functionality and stability. This section compares the proposed work with existing similar 8T SRAM IMC and TM architectures. The comparison with TM architectures is performed for MNIST dataset (as a case study) as shown in Table I. The TM MNIST model includes 10 classes and 1,568 literals. For training the TM model on this dataset, 100 clauses were assigned per class and requires memory array dimension of 1,568×1,000. With this constraint, TM achieves an accuracy of 96% and can be further enhanced by tuning the number of clauses and TM hyperparameters as in [24].

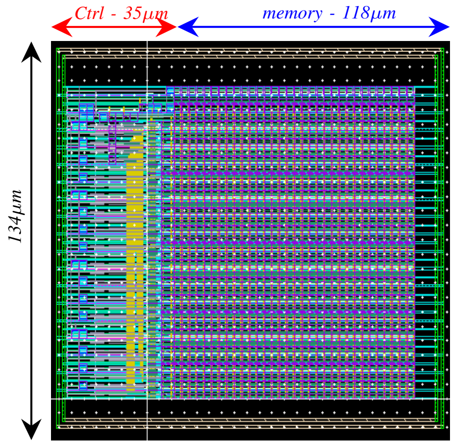



Fig. 7: Layout of a 32×32 Memory Bank.

Due to tool constraints, lengthy simulation times, and requirement of a large memory database, the memory banks were consolidated to accommodate all 1,568 literals for a single class (1568×100). Fig. 7 shows the layout of one 32×32 memory bank. While this requires 49 banks, a 256×32 configuration needs only seven and four arranged horizontally. As both give similar results, only the 256×32 results are shown in Table I.

The proposed architecture is compared against two baselines: a digitally synthesized CMOS TM using the same $65nm$ CMOS standard cell library [25], and a memristor-based analog IMC architecture [13]. All three implementations use the same TM model for MNIST, with 100 clauses per class. It is important to note that AURORA and [13] does not include pop count and argmax. In CMOS TM, the TAs are stored in the flip-flops, and the basic gates achieve the partial clause. Due to the logic reduction from the synthesis tool, the average number of transistors required to perform a partial clause is 7. The 8T SRAM cell reduces area by 81% per stored TA and partial clausing compared to CMOS TM with an area of $3\mu m^2$.

The inference times for the CMOS TM and memristor array are $3ns$ and $35ns$, respectively, while the proposed 8T SRAM architecture has an inference time of about $4ns$. It is 30% larger than the CMOS TM and provides a 90% latency reduction compared to IMBUE. AURORA shows significant energy efficiency, with reductions of 22.22× and 772× in energy consumption per data point, achieving 22.3× and 37.5× TOPS/W compared to CMOS TM and IMBUE, respectively. The 8T SRAM and CMOS TM have similar inference times, but the 8T SRAM uses less power for storing TA, with wired-OR speed comparable to combinational logic.

In a memristor-based TM architecture, the clause's output current is proportional to the sum of the currents flowing through the column, which increases with dataset size. The condition (TA=1, li=0) corresponds to the low resistance state (LRS) of the memristor leading to larger current flow. The current flow increases proportionally to the number of includes, increasing the circuit's power. Therefore, power is significantly influenced by both the dataset size and the number of inclusions. In contrast, the power in the proposed in-memory architecture is not influenced by the number of includes (cells that meet the discharge condition, discussed previously), and its impact on the dataset is less (energy increases due to an increase in memory cells) than in a memristor-based architecture.

TABLE I: Comparison of AURORA with other TM inference architecture.

| | Area per TA + PC | Latency | TOPS/W | Energy per Datapoint (nJ) |
|---|---|---|---|---|
| CMOS TM [25] | 35T + 7T | 3ns | 387.3 | 0.4 |
| IMBUE [13] | 1T1R | 35ns | 230 | 13.9 |
| This work* | 8T | 4ns | 8638 | 0.018 |

TABLE II: Parametric comparisons of AURORA with other 8T arrays.

| | **AURORA** | 8T [7] | 8T [8] | 8T [19] | 6T [18] |
|---|---|---|---|---|---|
| **Bank size** | 256×32 | 16×8 | 128×8 | 128×128 | 512×512 |
| **Operation(s)** | NOR, TM PC | MAC | MAC | AND, OR, XOR | (N)AND, (N)OR, X(N)OR, NOT, Row CPY |
| **Throughput** | 3413 TOPS | 0.29 GOPS | 240 GOPS | – | – |
| **TOPS/W** | 2917 | 5.07 | 48.08 | 0.04 | 769.3 |
| **Stage** | Post layout | Post Si | Post Si | Schematic | Schematic |

Similarly, in Table II, different 8T SRAM IMC implementations are compared. The operations performed in [7] and [8] are current-based MAC for multi-layer perceptron and binary multiplication similar to BNN. [19] performs logic operations for BNN and [18] demonstrate logic operations in 6T SRAM. The magnitude of the difference for throughput and energy is of the order $10^3$. AURORA outperforms the other architectures in throughput and energy efficiency.

## VI. Conclusions

This paper presents AURORA, an automated 8T SRAM wired-OR logic array. The limitations imposed by concurrent row operations on the row size were simulated and discussed in detail. Two bank configurations, 32×32 and 256×32, were analyzed for their inference latency and power consumption. Furthermore, PVT analysis revealed the best and worst corners for latency and power metrics. Monte Carlo simulations demonstrated the stability of the designed banks under the worst-case conditions, with a skewness of 0.1 for 32×32 and almost zero for 256×32. The experimental results with and without decoder did not show significant differences. The memory arrays were effectively generated using an internally developed custom memory compiler. The area of a single cell is $3\times3\mu m^2$.

The concurrent row operation of the memory array was validated using the MNIST dataset within the Boolean machine learning model known as the Tsetlin Machine. In primary comparisons with other TM, AURORA demonstrated the fastest inference time with a greater TOPS/J ratio, indicating its high suitability for Boolean inference acceleration. In secondary comparisons with existing 8T SRAM architectures, it exhibited improved energy efficiency alongside a larger memory size.

## VII. Acknowledgement

We gratefully acknowledge the support of our sponsors through the following projects: EPSRC ESTEEM (EP/X039943/1) and the EPSRC iCASE Award “Analogue Asynchronous Co-Design for On-Chip Machine Learning”.

## REFERENCES


[1] Z.-T. Lin, Z. Tong, J. Zhang, F. Wang, T. Xu, Y. Zhao, X. Wu, C. Peng, W. Lu, Q. Zhao, and J. Chen, "A review on sram-based computing in-memory: Circuits, functions, and applications," *Journal of Semiconductors*, vol. 43, p. 031401, 03 2022.

[2] A. Sebastian, M. L. Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[3] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[5] Z. Chen, Z. Yu, Q. Jin, Y. He, J. Wang, S. Lin, D. Li, Y. Wang, and K. Yang, "Cap-ram: A charge-domain in-memory computing 6t-sram for accurate and precision-programmable cnn inference," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 6, pp. 1924–1935, 2021.

[6] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, "C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, 2020.

[7] K.-Y. Chung, H. Kim, Y. An, K. Seong, D.-H. Shin, K.-H. Baek, and Y. Shim, "8T-SRAM based process-in-memory (PIM) system with current mirror for accurate MAc operation," *IEEE Access*, 2024.

[8] H.-J. Shin and S.-H. Jo, "Low-power 8T SRAM compute-in-memory macro for edge AI processors," *Applied Sciences*, vol. 14, no. 23, 2024.

[9] Z. Lin, Z. Tong, J. Zhang, F. Wang, T. Xu, Y. Zhao, X. Wu, C. Peng, W. Lu, Q. Zhao, and J. Chen, "A review on sram-based computing in-memory: Circuits, functions, and applications," *Journal of Semiconductors*, vol. 43, p. 031401, mar 2022.

[10] K. Yoshioka, S. Ando, S. Miyagi, Y.-C. Chen, and W. Zhang, "A review of sram-based compute-in-memory circuits," *Japanese Journal of Applied Physics*, vol. 63, p. 120802, Dec. 2024.

[11] Z.-R. Wang, Y.-T. Su, Y. Li, Y.-X. Zhou, T.-J. Chu, K.-C. Chang, T.-C. Chang, T.-M. Tsai, S. M. Sze, and X.-S. Miao, "Functionally complete boolean logic in 1T1R resistive random access memory," *IEEE Electron Device Letters*, vol. 38, no. 2, pp. 179–182, 2016.

[12] Y. Huang, T. Ando, A. Sebastian, M.-F. Chang, J. J. Yang, and Q. Xia, "Memristor-based hardware accelerators for artificial intelligence," *Nature Reviews Electrical Engineering*, 2024.

[13] O. Ghazal, S. Singh, T. Rahman, S. Yu, Y. Zheng, D. Balsamo, S. Patkar, F. Merchant, F. Xia, A. Yakovlev, and R. Shafik, "IMBUE: In-Memory Boolean-to-CUrrent Inference ArchitecturE for Tsetlin Machines," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2023.

[14] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, "8T SRAM cell as a multibit dot-product engine for beyond Von Neumann computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[15] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, D. Blaauw, and D. Sylvester, "A 0.3v vddmin 4+2t sram for searching and in-memory computing using 55nm ddc technology," in *2017 Symposium on VLSI Circuits*, pp. C160–C161, 2017.

[16] D. Elliott, M. Stumm, W. Snelgrove, C. Cojocaru, and R. Mckenzie, "Computational ram: implementing processors in memory," *IEEE Design & Test of Computers*, vol. 16, no. 1, pp. 32–41, 1999.

[17] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-SRAM: Enabling in-memory boolean computations in CMOS static random access memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4219–4232, 2018.

[18] N. Rohbani, M. A. Soleimani, B. Salami, O. Unsal, A. C. Kestelman, and H. Sarbazi-Azad, "Bimax: A bitwise in-memory accelerator using 6t-sram structure," in *2025 Design, Automation and Test in Europe Conference (DATE)*, pp. 1–7, 2025.

[19] A. Jaiswal, A. Agrawal, M. F. Ali, S. Sharmin, and K. Roy, "i-sram: Interleaved wordlines for vector boolean operations using srams," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4651–4659, 2020.

[20] K. Krishnamurthy, S. Duan, J. Ojukwu, O. Ghazal, A. Yakovlev, and R. Shafik, "9t sram cell based wired-or logic arrays for tsetlin machine inference," in *2024 International Symposium on the Tsetlin Machine (ISTM)*, pp. 1–4, 2024.

[21] J. Chen, W. Zhao, Y. Wang, Y. Shu, W. Jiang, and Y. Ha, "A reliable 8t sram for high-speed searching and logic-in-memory operations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 6, pp. 769–780, 2022.

[22] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, 2016.

[23] C. D. Arandilla, A. B. Alvarez, and C. R. K. Roque, "Static noise margin of 6T SRAM cell in 90-nm CMOS," in *2011 UkSim 13th International Conference on Computer Modelling and Simulation*, pp. 534–539, 2011.

[24] O. Tarasyuk, T. Rahman, R. Shafik, A. Yakovlev, A. Gorbenko, O.-C. Granmo, and L. Jiao, "Systematic search for optimal hyper-parameters of the Tsetlin Machine on MNIST dataset," in *2023 International Symposium on the Tsetlin Machine (ISTM)*, pp. 1–8, 2023.

[25] S. A. Tunheim, Y. Zheng, L. Jiao, R. Shafik, A. Yakovlev, and O.-C. Granmo, "An all-digital 65-nm tsetlin machine image classification accelerator with 8.6 nj per mnist frame at 60.3 k frames per second," *arXiv preprint arXiv:2501.19347*, 2025.