Reconfigurable VLSI Tsetlin Using Probabilistic State Updates for Intelligence Systems

Uppala Nikhil[1], Appala Sravan Kumar[1*]
[1]Department of Electronics & Communication Engineering, Vaagdevi Engineering College, Warangal, 506005, Telangana, India.
*Correspondence: Appala Sravan Kumar (appala.sravan@gmail.com)

Abstract
Machine learning accelerators have become increasingly important due to the rapid growth of intelligent edge devices, with global AI hardware deployments expected to exceed billions of connected systems and real-time data generation surpassing hundreds of zettabytes annually. However, existing Dynamic Tsetlin Machine (DTM) architectures employ deterministic clause evaluation, classification, and update mechanisms, which may limit learning flexibility, increase unnecessary parameter updates, reduce robustness to noisy data, and potentially affect generalization performance when handling complex datasets. To address these limitations, this work proposes a Probabilistic State Update-Dynamic Tsetlin Machine (PSU-DTM) training accelerator that introduces probabilistic intelligence into both learning and inference stages. The proposed architecture consists of a Feature Buffer, Partial Clause State Computation unit, Clause Buffer, Probabilistic Weight Multiplication module, Class Sum Computation unit, Probabilistic Classifier, Clause Feedback Generation module, Pseudo Random Number Generator (PRNG), Weight Update unit, and Tsetlin Automata (TA) State Update unit. The PRNG-driven probabilistic feedback mechanism selectively reinforces significant clauses while suppressing less informative ones, enabling adaptive clause learning and efficient weight optimization. Furthermore, probabilistic classification and state-transition strategies improve convergence speed, enhance robustness against overfitting, and increase classification accuracy. The resulting PSU-DTM architecture provides a scalable, hardware-efficient, and interpretable machine learning framework suitable for next-generation FPGA-based intelligent systems requiring low latency, high throughput, and adaptive learning capabilities.

Keywords: Dynamic Tsetlin Machine (DTM), Probabilistic State Update (PSU), FPGA Accelerator, Tsetlin Automata, Probabilistic Learning, Clause Feedback, Pseudo Random Number Generator (PRNG)

1. Introduction

The rapid advancement of artificial intelligence, machine learning, and edge computing technologies has significantly increased the demand for efficient hardware accelerators capable of performing intelligent decision-making tasks in real time. According to recent industry reports, the volume of global data generated annually has exceeded 180 zettabytes and is expected to continue growing due to the widespread deployment of Internet of Things (IoT) devices, autonomous systems, smart sensors, and industrial automation platforms. Simultaneously, semiconductor manufacturers are facing increasing pressure to design computing architectures that can process massive amounts of data while maintaining low latency, reduced power consumption, and efficient hardware utilization. These requirements have encouraged researchers and industries to explore specialized machine learning accelerators that can provide high computational efficiency while minimizing resource overhead.

Modern digital systems increasingly rely on hardware-based machine learning solutions to meet stringent performance requirements. Data centers, edge devices, automotive electronics, medical diagnostic equipment, and cybersecurity infrastructures continuously generate large volumes of data that require rapid analysis and classification. Traditional software-based processing approaches often encounter performance bottlenecks due to memory access delays, computational complexity, and increased energy consumption. As a result, hardware-oriented machine learning architectures implemented on FPGA and ASIC platforms have emerged as attractive alternatives because they offer

parallel processing capabilities, faster execution speeds, and improved energy efficiency. The growing adoption of intelligent hardware systems demonstrates the importance of integrating machine learning directly into digital circuits and embedded platforms. The semiconductor industry is witnessing a major transition toward intelligent VLSI architectures that combine data processing and learning capabilities within compact hardware frameworks. Organizations involved in integrated circuit design continuously seek solutions that provide scalability, reliability, and resource efficiency. Furthermore, the increasing complexity of modern applications requires architectures capable of handling large datasets without significantly increasing chip area or power requirements. Consequently, research in hardware-friendly machine learning has become an important area within VLSI design, enabling the development of intelligent computing systems that can satisfy the demands of next-generation electronic applications.

2. Literature Survey

Gang Mao et al. [1] introduced a DTM accelerator for on-chip training using FPGA platforms. The methodology employed a logic-based learning framework that integrates inference and training within the same hardware architecture. The design supports both Vanilla Tsetlin Machine and Coalesced Tsetlin Machine models through dynamically reconfigurable hardware modules. Clause computation, weight updates, and automata state transitions were implemented using FPGA-friendly logic structures to reduce computational complexity. The architecture utilized lookup-table-based processing and efficient memory organization to improve energy efficiency and training performance. Additionally, runtime reconfiguration enabled adaptation to different datasets and model configurations without requiring hardware resynthesis. The architecture still involves complex training-control circuitry, resulting in increased hardware overhead for large-scale implementations. Gang Mao et al. [2] developed an FPGA-based DTM training accelerator targeted for edge computing environments. Their approach combined finite-state automata-driven learning with logic-based inference to perform on-chip training and classification. The accelerator employed dynamic resource allocation and configurable clause computation modules to support varying model sizes. Efficient memory utilization strategies were incorporated to minimize block RAM usage while maintaining training capability. The design further emphasized energy-efficient operation by reducing multiply-accumulate computations typically found in neural network accelerators. Training latency increases as the number of clauses and automata grows, limiting scalability for highly complex datasets.

S. Kundu et al. [3] presented a comprehensive review of Tsetlin Machine methodologies, covering their theoretical foundations, learning mechanisms, hardware implementations, and practical applications. The study analyzed clause formation, Tsetlin Automata state transitions, feedback generation mechanisms, and different TM variants. It also evaluated the performance of Tsetlin Machines across IoT, healthcare, cybersecurity, and pattern-recognition applications. Furthermore, the review compared Tsetlin Machines with deep learning models in terms of interpretability, computational efficiency, and hardware suitability. The study primarily provides analytical insights and does not address practical hardware optimization challenges for real-time deployment. Omar Ghazal et al. [4] introduced an in-memory computing architecture based on Y-Flash technology for Coalesced Tsetlin Machine inference. The methodology integrated memory and computation within the same hardware structure to minimize data movement overhead. Shared clause storage and weighted class-sum computation were implemented directly inside memory arrays to improve throughput and energy efficiency. The architecture exploited parallel memory operations to accelerate inference while reducing external memory access requirements. Additionally, CoTM-specific optimizations were incorporated to support large-scale classification tasks. The design mainly focuses on inference acceleration and does not provide efficient support for on-chip learning and training operations.

Shtaiwi and Mustafa [5] formulated a framework for secure and adaptive AI hardware architectures targeting large language model applications. The methodology combined hardware optimization techniques with adaptive resource management to improve security and computational efficiency. The

framework incorporated workload-aware processing, dynamic architectural tuning, and hardware-level protection mechanisms. It also investigated the integration of AI-specific accelerators with secure execution environments to improve system reliability. The design aimed to balance performance, adaptability, and security requirements for next-generation AI systems. The framework introduces additional hardware complexity that may increase implementation cost and resource utilization. Santhosh Sivasubramani et al. [6] investigated design methodologies for skyrmion-based circuits and systems in AI-driven applications. Their methodology explored bidirectional integration between emerging spintronic devices and machine learning workloads. Device-level modeling, circuit design strategies, and system-level optimization techniques were employed to improve computational efficiency. The work also analyzed the suitability of skyrmion devices for implementing neuromorphic and AI accelerators. Performance evaluations were conducted with respect to energy consumption, scalability, and reliability. Practical fabrication challenges and device variability remain significant barriers to large-scale deployment.

Victor Marot et al. [7] developed a nanoelectromechanical binary comparator for edge-computing applications. The methodology utilized nanoelectromechanical switching devices to perform comparison operations with ultra-low energy consumption. The architecture emphasized compact circuit design and reduced leakage power. Device-level optimization was carried out to enhance switching reliability and operational stability. The comparator was evaluated for suitability in low-power edge computing environments where energy efficiency is critical. Mechanical switching characteristics may limit operating speed compared to conventional CMOS-based comparators. Shahid Gulzar Padder et al. [8] reviewed data-driven approaches for estimating electric vehicle battery State-of-Charge (SoC) and State-of-Health (SoH). The methodology examined machine learning, deep learning, and hybrid predictive models for battery condition assessment. Various feature extraction, data preprocessing, and model training techniques were analyzed across different battery datasets. Comparative evaluations were performed based on estimation accuracy, robustness, and computational requirements. The study also discussed real-time implementation challenges for battery management systems. Many reviewed models require extensive training datasets and substantial computational resources for deployment.

Y. He et al. [9] presented ultra-low-voltage reconfigurable FETs in 22 nm FDSOI technology for dynamic circuit obfuscation and embedded security. The methodology utilized reconfigurable transistor structures capable of dynamically altering circuit functionality. Security enhancement was achieved through runtime circuit transformation and hardware obfuscation techniques. The design explored voltage-scalable operation to reduce energy consumption while maintaining security robustness. Experimental validation was performed using advanced FDSOI technology platforms. Reconfigurable security mechanisms introduce additional control circuitry and design complexity. Mejail and Nestares [10] investigated self-optimizing VLSI systems for sustainable embedded computing. Their methodology employed adaptive hardware monitoring and optimization strategies to dynamically adjust system operation. Resource allocation, power management, and workload balancing mechanisms were integrated into the architecture. The design continuously analyzed system conditions and adapted operating parameters to improve energy efficiency. The approach targeted sustainable embedded computing environments with varying workload requirements. Continuous monitoring and adaptation processes may incur additional hardware and computational overhead.

Maniyar Mohammed Saqlain et al. [11] developed a machine learning-based framework for power estimation in digital VLSI circuits. The methodology employed supervised learning algorithms to predict circuit power consumption during the design stage. Various circuit parameters were extracted and utilized as input features for model training. The framework analyzed the relationship between design attributes and power dissipation to generate accurate power estimates. Experimental evaluations demonstrated improved prediction accuracy compared with conventional analytical estimation

methods. Machine learning models were further optimized to reduce estimation time while maintaining reliability. The prediction accuracy is highly dependent on the quality and diversity of the training dataset. K. Malarkodi et al. [12] presented an SVM-based framework for precise on-chip power analysis in CMOS VLSI circuits. Their methodology utilized Support Vector Machine learning to model power consumption characteristics under different operating conditions. Feature extraction techniques were employed to capture circuit switching behavior and structural parameters. The trained SVM model generated power estimates without requiring exhaustive simulation procedures. The approach reduced computational effort while providing rapid power evaluation during design verification stages. The SVM training complexity increases significantly when handling large-scale circuit datasets.

Jae Hwan Shin et al. [13] introduced a scalable multi-site test architecture for chiplet-based systems operating on Automated Test Equipment platforms. The methodology incorporated parallel testing mechanisms to evaluate multiple chiplets simultaneously. Resource-sharing strategies were implemented to improve test throughput and reduce testing time. The architecture also utilized scalable communication interfaces to support heterogeneous chiplet integration. The architecture requires sophisticated synchronization mechanisms, increasing implementation complexity. Tanuj Mathur et al. [14] explored the use of generative diffusion models for test pattern synthesis in VLSI systems. The methodology employed diffusion-based generative learning to create optimized test vectors capable of improving fault coverage. Synthetic test patterns were generated through iterative denoising processes guided by fault characteristics. The generated patterns were evaluated for fault detection effectiveness and lifecycle management applications. The approach reduced dependence on traditional deterministic test generation techniques while improving coverage efficiency. Training diffusion models requires substantial computational resources and large datasets.

E. N. Darko et al. [15] presented a high-accuracy built-in self-test methodology for high-resolution data converters. The approach integrated on-chip testing circuitry with adaptive calibration mechanisms to improve test accuracy. Statistical analysis techniques were used to identify converter nonlinearity and performance degradation. The architecture minimized external testing requirements while maintaining measurement precision. Experimental validation confirmed effective fault detection with reduced testing cost. Additional built-in testing circuitry increases chip area overhead. Md Manan Mujahid et al. [16] investigated quantum machine learning techniques for performance analysis of low-power VLSI circuits in IoT devices. Their methodology combined quantum computing principles with machine learning algorithms to analyse circuit characteristics. Quantum-enhanced feature processing was utilized to evaluate power consumption and performance metrics. The framework examined optimization opportunities for energy-efficient circuit operation. Comparative analyses demonstrated improved prediction capability for complex VLSI behaviours. Practical deployment is limited by the current immaturity of quantum computing hardware.

3. Proposed System

The proposed PSU-DTM architecture as shown in Figure 1 enhances the conventional DTM by incorporating probabilistic learning mechanisms into clause evaluation, weight adaptation, and classification processes. The architecture consists of major modules including the Feature Buffer, Partial Clause State Computation Unit, Clause Buffer, Probabilistic Weight Multiplication Unit, Class Sum Computation Unit, Probabilistic Classifier, Clause Feedback Generation Module, PRNG, Weight Update Unit, and Tsetlin Automata (TA) State Update Unit. During inference, input numerical features are transformed into clause states, weighted probabilistically, and aggregated to generate class predictions. During training, random feedback generated by the PRNG is combined with clause outputs and classification results to probabilistically update both clause weights and automata states. This adaptive learning mechanism enables efficient hardware implementation while improving learning flexibility, reducing overfitting, and enhancing classification performance.

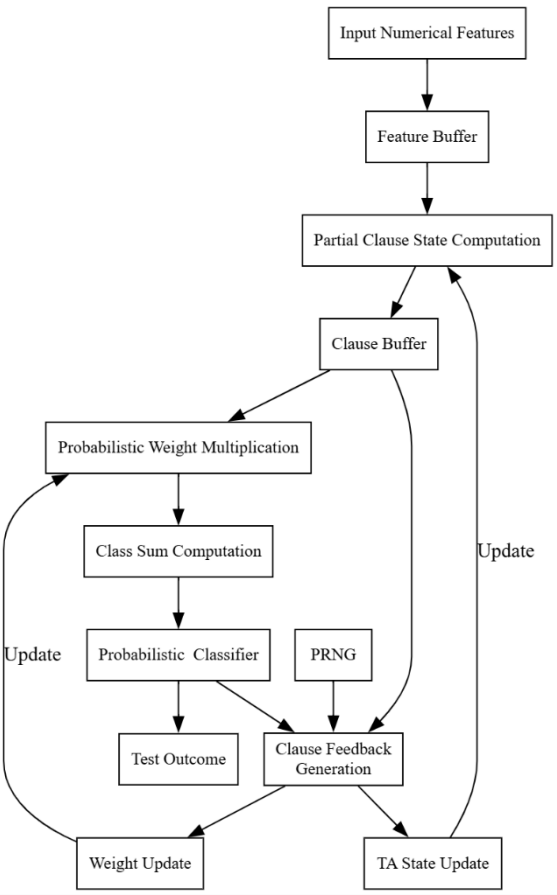

Figure. 1: Proposed PSU-DTM architecture.

Input Numerical Features and Feature Buffer: The operation begins by supplying numerical feature vectors to the Feature Buffer. This module temporarily stores the incoming data and organizes it for efficient parallel processing. The Feature Buffer acts as an interface between external memory and the clause computation engine, ensuring a continuous flow of feature values without processing interruptions. By buffering the data, the architecture reduces memory access latency and supports high-throughput execution.

Partial Clause State Computation: The buffered feature values are forwarded to the Partial Clause State Computation module. In this stage, feature literals are evaluated according to the current states of the Tsetlin Automata. Instead of generating only clause outputs, this module computes intermediate clause states that represent the activation condition of each clause. Multiple clause states are generated simultaneously through parallel hardware units, allowing efficient extraction of logical patterns from the input features.

Clause Buffer Storage: The computed clause states are stored in the Clause Buffer, which serves as temporary storage for intermediate clause information. The Clause Buffer synchronizes data transfer between clause computation, classification, and training modules. Since clause information is required by both inference and feedback generation stages, storing the clause states prevents redundant computations and improves processing efficiency.

Probabilistic Weight Multiplication: The clause states are transferred to the Probabilistic Weight Multiplication module. Here, each activated clause state is multiplied by its corresponding clause weight while incorporating probabilistic weighting behavior. This mechanism allows the architecture to dynamically adjust the contribution of individual clauses according to their learned significance. The probabilistic weighting process introduces adaptability into the classification procedure and helps emphasize highly informative clauses while reducing the influence of less relevant ones.

Class Sum Computation: The weighted clause contributions are accumulated in the Class Sum Computation module. All clause scores associated with a particular class are aggregated to generate a class sum value. These class sums represent the confidence levels of the model for each candidate class. The accumulation process integrates information from multiple clauses and provides a quantitative measure for final decision making.

Probabilistic Classification: The computed class sums are processed by the Probabilistic Classifier. Unlike conventional deterministic classifiers, this module incorporates probabilistic decision-making mechanisms that consider the class scores and learned uncertainty information. The classifier selects the most probable class label and generates the final prediction result. The resulting prediction is forwarded to the Test Outcome block and simultaneously supplied to the feedback generation module for learning purposes.

Test Outcome Generation: The selected class label is produced as the Test Outcome, representing the final classification result of the PSU-DTM architecture. During inference, this output corresponds to the predicted class of the input sample. During training, the prediction is further utilized to determine the correctness of the classification and to generate appropriate feedback signals for parameter updates.

Clause Feedback Generation: The Clause Feedback Generation module receives three major inputs: clause states from the Clause Buffer, prediction results from the Probabilistic Classifier, and random values generated by the PRNG. Based on these inputs, the module determines whether clauses should receive reward, penalty, or no update. The generated feedback signals guide the learning process and influence both weight adaptation and automata state transitions.

Pseudo Random Number Generator: The PRNG provides random values required for probabilistic learning operations. These random numbers determine whether specific feedback actions are applied during training. By introducing stochastic behavior into the update process, the architecture avoids excessive deterministic learning and improves generalization capability. The PRNG therefore plays a critical role in enabling probabilistic state updates and adaptive learning.

Weight Update Mechanism: The feedback signals generated during training are sent to the Weight Update module. This module modifies clause weights according to their contribution to classification accuracy. Clauses that support correct predictions receive positive weight adjustments, while clauses associated with incorrect decisions receive reduced weights. The updated weights are subsequently fed back into the Probabilistic Weight Multiplication module, enabling continuous refinement of clause importance.

Tsetlin Automata State Update: Simultaneously, the generated feedback is applied to the TA State Update module. This module updates the internal states of the Tsetlin Automata responsible for clause construction. Reward signals strengthen beneficial literal inclusion decisions, whereas penalty signals encourage alternative clause configurations. The updated automata states are fed back into the Partial Clause State Computation module, allowing the clause structures to evolve and adapt over successive training iterations.

Iterative Probabilistic Learning Process: The updated clause weights and automata states are continuously recycled back into the inference pipeline. Weight updates influence the Probabilistic Weight Multiplication stage, while TA state updates modify future clause state computations. Through repeated cycles of inference, feedback generation, probabilistic updating, and parameter refinement, the PSU-DTM architecture progressively improves classification accuracy and learning efficiency. This closed-loop framework enables scalable FPGA implementation while maintaining robust and adaptive rule-based machine learning performance.

3.1 Feature Buffer

The process begins with loading input feature vectors into the Feature Buffer (FB) as shown in Figure 2. Each feature is converted into a pair of literals (L): the original feature and its logical negation. These literals are distributed via Demultiplexer (DEMUX) structures to the clause computation units.

Simultaneously, the Literal Counter (LC) tracks the index of each literal. These signals are forwarded to the TA RAM to retrieve current Tsetlin Automata actions, which dictate whether a literal is "Included" or "Excluded" from a specific clause.

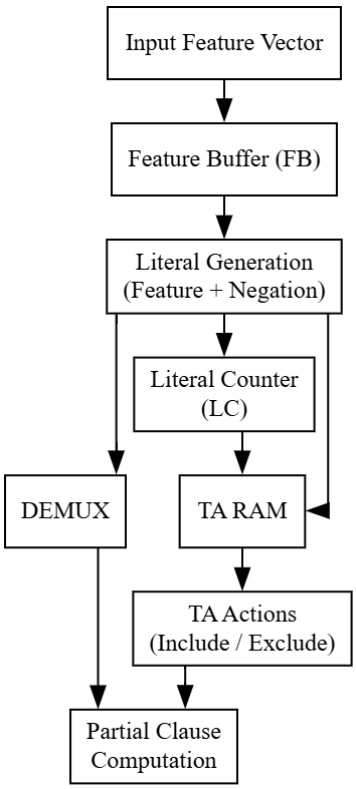

Figure. 2: Feature buffer flowchart.

Step 1: The process begins by loading the input feature vector into the Feature Buffer (FB), which temporarily stores feature values required for clause computation.

Step 2: Each feature stored in the FB is converted into two literals (lit) representing the feature itself and its negated form.

Step 3: The generated literals are distributed to multiple clause computation paths using the Demultiplexer (DEMUX) network.

Step 4: The Literal Counter (LC) tracks the sequence of literal processing and provides indexing information for clause evaluation.

Step 5: The literal signals are forwarded to the Tsetlin Automata RAM (TA RAM) where the stored Tsetlin Automata (TA) states are accessed.

Step 6: Based on the automata states, the corresponding TA actions (ta) are generated to determine whether a literal should be included or excluded in clause formation.

Step 7: The literals and TA actions are then transmitted to the Partial Clause computation units for clause evaluation.

3.2 Partial Clause State Computation

The retrieved automata actions and literal signals are processed by Partial Clause (PC) state units as shown in Figure 3. Each PC unit evaluates a subset of a clause by performing logical conjunction operations based on the TA decisions. By utilizing multiple PCs operating in parallel, the architecture can construct complex clause components simultaneously, significantly reducing the clock cycles required for the logical combination of selected literals.

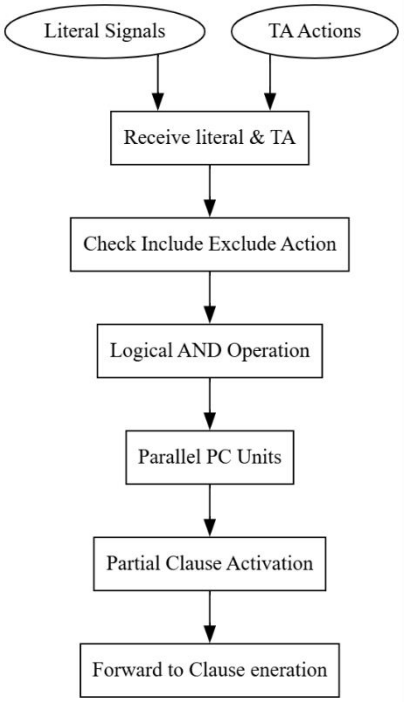

Figure. 3: Partial clause state computation

Step 1: The Partial Clause (PC) units receive literal signals (lit) and automata actions (ta) from the Feature Buffer and TA RAM.

Step 2: Each PC unit checks whether the automata action indicates inclusion of the corresponding literal in clause formation.

Step 3: Logical conjunction operations are performed between the selected literals to produce intermediate clause signals.

Step 4: Multiple PC units operate in parallel to evaluate different clause components simultaneously.

Step 5: The intermediate clause results generated by the PC units represent partial clause activations.

Step 6: These partial clause outputs are forwarded to the clause generation stage for final clause evaluation.

4. Results and Discussion

Figure 4 illustrates the functional simulation results of the proposed PSU-DTM architecture. The waveform verifies the correct operation of key control signals, including clk, reset, ctrl, enable, up_down, and load. Initially, the system starts with an unknown state, represented by “xxxx”, before initialization. After the reset sequence, the scan_in[15:0] input successfully loads the hexadecimal value 000A, confirming correct data acquisition and feature loading functionality. The enable signal remains active throughout the execution period, allowing continuous operation of the learning mechanism. The clock signal exhibits stable periodic transitions across the entire simulation interval of approximately 8 µs, ensuring synchronized execution of clause evaluation and state updates. Furthermore, the bist_out signal transitions successfully after initialization, demonstrating correct BIST functionality and validating the integrity of the proposed hardware design. The simulation results confirm that the probabilistic learning, state update, and classification modules operate correctly under the specified control conditions, thereby validating the functional correctness of the PSU-DTM accelerator.

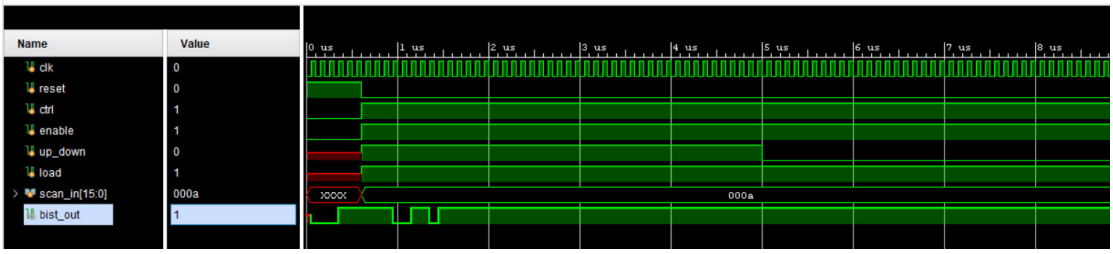

Figure. 4: Proposed simulation outcome.

Figure 5 presents the FPGA resource utilization summary of the proposed PSU-DTM architecture. The design utilizes 1,637 LUTs out of the available 133,800 LUTs, corresponding to only 1.22% utilization, indicating a highly optimized combinational logic implementation. The architecture employs 851 Flip-Flops (FFs) from the available 267,600 FFs, resulting in 0.32% utilization, which reflects the storage requirements associated with probabilistic state maintenance and feedback tracking. The proposed design uses 23 IO pins out of 500 available IO resources, producing 4.60% utilization, while 2 BUFG global clock buffers are utilized from the available 32 resources, corresponding to 6.25% utilization. Compared with the existing architecture, the LUT utilization is significantly reduced from 3,676 to 1,637 LUTs, demonstrating the hardware efficiency achieved through probabilistic clause selection and optimized state-update mechanisms. The low overall resource consumption confirms the scalability of the PSU-DTM architecture for FPGA-based intelligent systems.

Resource	Utilization	Available	Utilization %
LUT	1637	133800	1.22
FF	851	267600	0.32
IO	23	500	4.60
BUFG	2	32	6.25

Figure. 5: Proposed area outcome

Figure 6 shows the power consumption characteristics of the proposed PSU-DTM architecture. The total on-chip power is dominated by dynamic power consumption of 29.347 W, which accounts for approximately 99% of the total power, while static power consumption is only 0.247 W, representing about 1% of the total power dissipation. Among the dynamic power components, logic resources consume 15.406 W (52%), making them the largest contributor due to probabilistic clause evaluation and adaptive state-transition operations. Signal routing contributes 13.865 W (47%), indicating that interconnect activity remains a significant factor in overall power consumption. The I/O power consumption is only 0.076 W, accounting for less than 1% of total dynamic power. The static power component consists entirely of PL Static power (0.247 W). These results demonstrate that the proposed architecture maintains low static power requirements while efficiently supporting probabilistic learning operations, making it suitable for high-performance intelligent hardware accelerators.

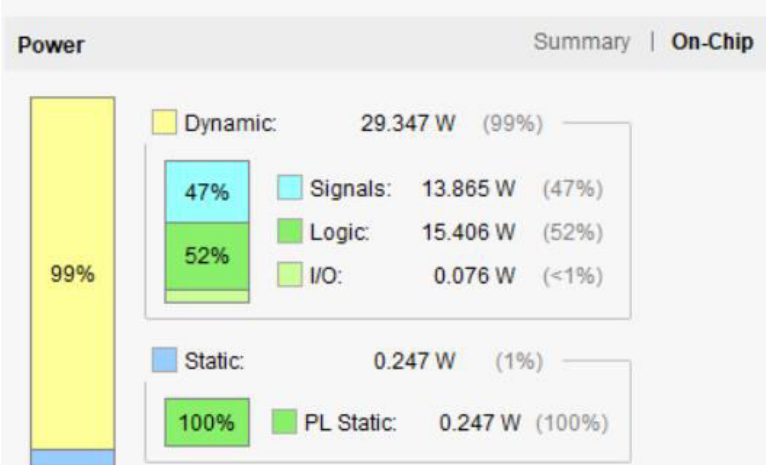

Figure. 6: Proposed power summary

Figure 7 presents the setup timing analysis of the proposed PSU-DTM architecture. The timing report identifies 10 critical setup paths, each containing approximately 375–376 logic levels, 311–312 routing segments, and a high fanout of 68. The longest setup path exhibits a total delay of 206.188 ns, comprising 101.426 ns logic delay and 104.762 ns net delay. The remaining paths show total delays ranging from approximately 205.011 ns to 206.188 ns. Compared with the existing architecture, which exhibited setup delays exceeding 292 ns, the proposed architecture significantly reduces the critical path delay by nearly 86 ns. The balanced contribution of logic delay and routing delay indicates an efficient hardware structure with reduced computational complexity and optimized interconnect utilization. The reduction in setup delay directly improves the achievable operating frequency and enhances real-time learning performance, making the proposed PSU-DTM architecture more suitable for latency-sensitive intelligent applications.

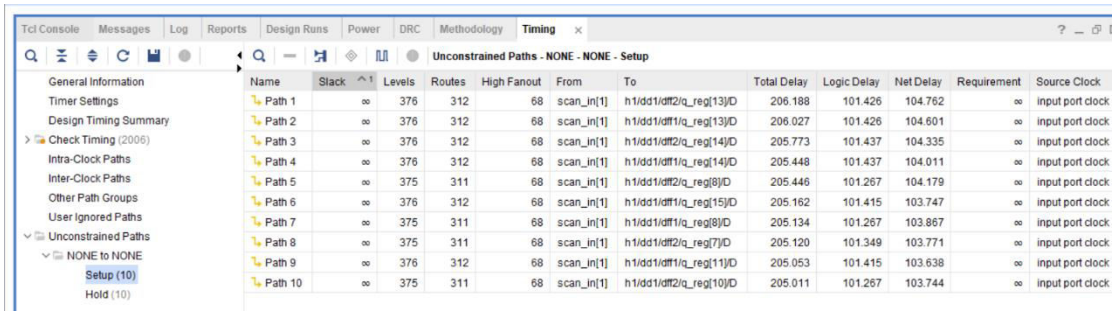

Figure. 7: Proposed setup delay outcome

Figure 8 illustrates the hold timing analysis of the proposed PSU-DTM architecture. The report identifies 10 hold paths, each consisting of only 1 logic level and 1 routing segment, demonstrating simple and efficient register-to-register data transfers. The observed total hold delays range from 0.241 ns to 0.269 ns, with corresponding logic delays varying between 0.177 ns and 0.203 ns and net delays ranging from 0.061 ns to 0.077 ns. The shortest hold path records a delay of 0.241 ns, while the maximum observed hold delay is 0.269 ns. The consistent and low hold delay values indicate excellent timing stability and confirm the absence of significant hold-time constraints. The minimal variation across all hold paths demonstrates that the probabilistic state-update mechanism does not introduce timing irregularities or synchronization issues. Consequently, the proposed PSU-DTM architecture achieves reliable sequential operation while maintaining high-speed data propagation and timing robustness across the entire learning and inference process.

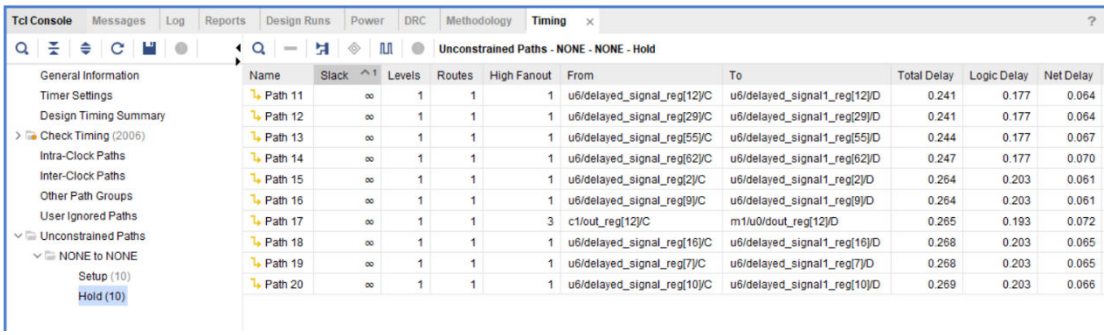

Figure. 8: Proposed hold delay outcome

4.1 Comparative Evaluation

Table 1 presents the FPGA resource utilization comparison between the existing DTM architecture and the proposed PSU-DTM. The existing design utilizes 3,676 LUTs, whereas the proposed architecture requires only 1,637 LUTs, achieving a significant 55.47% reduction in combinational logic resources. Similarly, the existing architecture employs 1 DSP block, while the proposed design completely eliminates DSP utilization, resulting in a 100% improvement through optimized probabilistic computation mechanisms. The number of input/output resources decreases from 24 IOs in the existing design to 23 IOs in the proposed architecture, providing a 4.17% improvement. Both architectures utilize 2 BUFG clock buffers, resulting in 0% change in clocking resources. These results demonstrate that the proposed PSU-DTM substantially reduces hardware resource consumption while maintaining equivalent functionality, thereby improving FPGA scalability and implementation efficiency.

Table 1. Area utilization comparison between existing and proposed architectures

Resource	Existing Design	Proposed PSU-DTM	Improvement (%)
LUT	3676	1637	55.47
DSP	1	0	100.00
IO	24	23	4.17
BUFG	2	2	0.00

Table 2. Setup delay comparison between existing and proposed architectures

Parameter	Existing Design (ns)	Proposed PSU-DTM (ns)	Improvement (%)
Total Delay	292.254	206.188	29.45
Logic Delay	145.437	101.426	30.26
Net Delay	146.817	104.762	28.64
Logic Levels	650	376	42.15
Routes	584	312	46.58
High Fanout	72	68	5.56

Table 2 compares the setup timing performance of the existing and proposed architectures. The existing design exhibits a total setup delay of 292.254 ns, whereas the proposed PSU-DTM reduces this delay to 206.188 ns, corresponding to a 29.45% improvement. The logic delay decreases from 145.437 ns to 101.426 ns, achieving a 30.26% reduction, while the net delay decreases from 146.817 ns to 104.762 ns, resulting in a 28.64% improvement. Furthermore, the number of logic levels is reduced from 650 to 376, corresponding to a substantial 42.15% reduction, indicating lower computational complexity. The routing complexity is also significantly decreased, with the number of routes dropping from 584 to 312, yielding a 46.58% improvement. Additionally, the high fanout value is reduced from 72 to 68, resulting

in a 5.56% improvement. These reductions collectively shorten the critical path length and enhance the operating speed of the proposed architecture.

Table 3 summarizes the hold timing performance comparison between the existing DTM and the proposed PSU-DTM architectures. The minimum hold delay remains unchanged at 0.241 ns for both designs, resulting in 0% improvement, indicating that the shortest register-to-register paths remain stable. The maximum hold delay decreases from 0.343 ns in the existing architecture to 0.269 ns in the proposed design, providing a 21.57% improvement. Likewise, the maximum logic delay remains identical at 0.203 ns, resulting in 0% change, while the maximum net delay is significantly reduced from 0.150 ns to 0.077 ns, corresponding to a 48.67% improvement. The substantial reduction in net delay demonstrates that the proposed architecture optimizes routing efficiency and interconnect management, thereby improving timing stability and reducing the likelihood of hold-time violations during high-speed operation.

Table 3. Hold delay comparison between existing and proposed architectures

Parameter	Existing Design (ns)	Proposed PSU-DTM (ns)	Improvement (%)
Minimum Hold Delay	0.241	0.241	0.00
Maximum Hold Delay	0.343	0.269	21.57
Maximum Logic Delay	0.203	0.203	0.00
Maximum Net Delay	0.150	0.077	48.67

Table 6.4. Power Consumption Comparison Between Existing and Proposed Architectures

Power Metric	Existing Design (uW)	Proposed PSU-DTM (uW)	Improvement (%)
Dynamic Power	79.380	29.347	63.03
Signal Power	37.006	13.865	62.53
Logic Power	41.516	15.406	62.89
DSP Power	0.778	0.000	100.00
I/O Power	0.079	0.076	3.80
Static Power	1.235	0.247	80.00
Total Power	80.615	29.594	63.29

Table 4 compares the power consumption characteristics of the existing and proposed architectures. The dynamic power consumption decreases from 79.380 µW in the existing design to 29.347 µW in the proposed PSU-DTM, achieving a significant 63.03% reduction. Similarly, signal power decreases from 37.006 µW to 13.865 µW, resulting in a 62.53% improvement, while logic power decreases from 41.516 µW to 15.406 µW, providing a 62.89% reduction. The proposed architecture eliminates DSP power consumption, reducing it from 0.778 µW to 0 µW, which corresponds to a 100% improvement. The I/O power is slightly reduced from 0.079 µW to 0.076 µW, yielding a 3.80% improvement. Furthermore, the static power consumption decreases from 1.235 µW to 0.247 µW, achieving an 80.00% reduction. As a result, the total power consumption is reduced from 80.615 µW to 29.594 µW, corresponding to a substantial 63.29% improvement. These results confirm that the probabilistic learning and adaptive state-update mechanisms of the PSU-DTM significantly enhance energy efficiency while maintaining high-performance operation.

5. Conclusion

The performance evaluation demonstrates that the proposed PSU-DTM architecture significantly enhances hardware efficiency, timing performance, and energy optimization compared with the conventional deterministic Dynamic Tsetlin Machine (DTM). The proposed design reduces LUT

utilization from 3,676 to 1,637, achieving a 55.47% improvement, while completely eliminating DSP usage and slightly reducing I/O resource requirements. Timing analysis reveals substantial reductions in critical path complexity, with total setup delay decreasing from 292.254 ns to 206.188 ns, logic delay decreasing by 30.26%, and routing complexity reduced by 46.58%. Hold timing performance is also improved through a 21.57% reduction in maximum hold delay and a 48.67% reduction in net delay. Furthermore, the probabilistic learning mechanism contributes to remarkable power savings, reducing dynamic power by 63.03%, static power by 80.00%, and total power consumption by 63.29%. These improvements confirm that the incorporation of probabilistic state updates, adaptive clause selection, and optimized feedback generation enables a highly scalable, low-latency, and energy-efficient hardware learning accelerator suitable for FPGA-based intelligent systems, edge AI devices, and real-time adaptive computing applications.

References

[1]. G. Mao et al., "Dynamic Tsetlin Machine Accelerators for On-Chip Training Using FPGAs," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 72, no. 11, pp. 6962-6975, Nov. 2025, doi: 10.1109/TCSI.2025.3564875.

[2]. Mao, Gang, Tousif Rahman, Sidharth Maheshwari, Bob Pattison, Zhuang Shao, Rishad Shafik, and Alex Yakovlev. "Dynamic Tsetlin Machine Accelerators for On-Chip Training at the Edge using FPGAs." arXiv preprint arXiv:2504.19797 (2025).

[3]. S. Kundu, S. S. Patkar, S. M. Mishra, G. Trivedi and F. Merchant, "A Comprehensive Review of Tsetlin Machines: Concepts, Applications, Analysis, and the Future," in IEEE Internet of Things Journal, vol. 13, no. 10, pp. 20105-20127, 15 May15, 2026, doi: 10.1109/JIOT.2026.3665145.

[4]. Ghazal, Omar, Wei Wang, Shahar Kvatinsky, Farhad Merchant, Alex Yakovlev, and Rishad Shafik. "Impact: In-memory computing architecture based on y-flash technology for coalesced tsetlin machine inference." Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 383, no. 2288 (2025).

[5]. Shtaiwi, S. and Mustafa, D., 2025. Towards Secure and Adaptive AI Hardware: A Framework for Optimizing LLM-Oriented Architectures. Computers, 15(1), p.10.

[6]. Sivasubramani, Santhosh, C. Kishore, Amit Acharyya, Vihar Georgiev, Rishad Shafik, and Themis Prodromakis. "Design methodologies for skyrmion-based circuits and systems in AI-driven applications: bi-directional integration [feature]." IEEE Circuits and Systems Magazine 25, no. 3 (2025): 30-55.

[7]. Marot, Victor, Manu Bala Krishnan, Mukesh Kumar Kulsreshath, Elliott Worsey, Roshan Weerasekera, and Dinesh Pamunuwa. "Nanoelectromechanical Binary Comparator for Edge-Computing Applications." In 2025 Design, Automation & Test in Europe Conference (DATE), pp. 1-7. IEEE, 2025.

[8]. Padder, Shahid Gulzar, Jayesh Ambulkar, Atul Banotra, Sudhakar Modem, Sidharth Maheshwari, Kolleboyina Jayaramulu, and Chinmoy Kundu. "Data-driven approaches for estimation of EV battery SoC and SoH: A review." Ieee Access 13 (2025): 35048-35067.

[9]. Y. He et al., "Ultra-Low Voltage Reconfigurable FETs in 22nm FDSOI Technology Enabling Dynamic Circuit Obfuscation for Embedded Security," in IEEE Journal of the Electron Devices Society, doi: 10.1109/JEDS.2026.3654187.

[10]. Mejail, M., and B. K. Nestares. "Self-Optimizing VLSI Systems for Sustainable Embedded Computing." Annals of Energy-Efficient VLSI Architectures (2026): 28-33.

[11]. Saqlain, Maniyar Mohammed, and Sakthivel Ramachandran. "Enhanced Power Estimation of Digital VLSI Circuits Using Machine Learning." In 2025 IEEE 9th International Test Conference India (ITC India), pp. 1-4. IEEE, 2025.

[12]. Malarkodi, K., S. Hemalatha, M. Amanullah, S. Irin Sherly, Gnanajeyaraman Rajaram, and D. Sheela. "SVM-Based Framework is Designed for Precise On-Chip Power Analysis in CMOS VLSI Circuits." In 2025 8th International Conference on Trends in Electronics and Informatics (ICOEI), pp. 147-152. IEEE, 2025.

[13]. Shin, Jae Hwan, Hyunbeen Kim, Jin Hwan Park, and Young-woo Lee. "Scalable Multi-Site Test Architecture for Chiplet-based Systems on ATE Platforms." IEEE Transactions on Semiconductor Manufacturing (2025).

[14]. Mathur, Tanuj, Mohan Vamsi Musunuru, Naveen Kumar Siripuram, Prabhu Muthusamy, and Bhargav Kumar Konidena. "Generative Diffusion Models for Test Pattern Synthesis in VLSI: Enhancing Fault Coverage and Silicon Lifecycle Management." In 2025 IEEE East-West Design & Test Symposium (EWDTS), pp. 1-14. IEEE, 2025.

[15]. Darko, E.N., Karimpour, S. and Chen, D., 2025, April. High-Accuracy, Cost-Effective Built-In Self-Test Approach for High-Resolution Data Converters. In 2025 IEEE 43rd VLSI Test Symposium (VTS) (pp. 1-7). IEEE.

[16]. Mujahid, Md Manan, and Deepa Jose. "A new quantum AI-performance analysis of low-power VLSI circuits in IoT devices using quantum machine learning techniques." Quantum Information Processing 25, no. 4 (2026): 126.