

# HiM: An Autonomous Hardware Accelerator for Solving Boolean Satisfiability Problem with a Heuristic-in-Macro Engine

Shin Han\*, Minhyeok Jeong\*, and Yoonmyung Lee†

Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea

\*Equally Contributed Author, †Corresponding Author, [yoonyung@skku.edu](mailto:yoonyung@skku.edu)

**Abstract**—Boolean Satisfiability (SAT), an NP-complete problem central to EDA and AI, has motivated hardware acceleration to overcome its exponential complexity. Early approaches focused on speeding up incomplete solvers, but their inherent algorithmic limitations made them unsuitable for correctness-critical tasks. Consequently, the focus shifted to hardware accelerators for complete solvers based on the DPLL/CDCL framework, which concentrated on accelerating the primary bottleneck: the Boolean Constraint Propagation (BCP) operation. However, performance is ultimately dominated by branching heuristics. Existing designs either omit heuristics, suffering large penalties, or offload them to CPUs, incurring prohibitive overhead. This work presents Heuristic-in-Macro (HiM), the first fully autonomous SAT accelerator integrating both an efficient BCP engine and a hardware-embedded MOMs branching heuristic in a single macro, eliminating CPU dependence. A high-throughput parallel processing architecture replaces traditional serialized clause scans with a tiled multi-macro execution, achieving 8.78× acceleration. At the circuit level, physical efficiency is enhanced through a compact 16T unit cell that merges logic and storage, thereby reducing area and energy. Proposed HiM-based solver achieves 100% SAT/UNSAT solvability, 172.1× speedup in algorithmic performance compared to designs without heuristics. When matched against a CPU-offloaded hybrid system, HiM is 305.6× faster and 1.99×10<sup>6</sup>× more energy-efficient. Compared to the widely used MiniSAT software solver, HiM delivers 26.7× speedup and 3.09×10<sup>6</sup>× efficiency, while reducing time- and energy-to-solution by up to 94% and 83% versus state-of-the-art ASIC accelerators.

**Keywords**—Boolean satisfiability, combinatorial optimization problems, complete solver, heuristic-in-macro, in-memory computation, domain-specific accelerator

## I. INTRODUCTION

Boolean Satisfiability (SAT) problem is an NP-complete problem that serves as a core engine for a wide range of applications in Electronic Design Automation (EDA) and Artificial Intelligence (AI) [1]-[6]. Due to its computational complexity, significant research efforts have been dedicated to developing high-performance SAT solvers. To overcome the performance limits of software solvers on von Neumann architectures, various hardware accelerators on both ASIC(Application-Specific Integrated Circuit) and FPGA(Field Programmable Gate Array) platforms have been proposed to mitigate the exponential complexity bottleneck.

These hardware-based SAT solvers can be broadly categorized into incomplete and complete solvers. Many early hardware efforts focused on accelerating incomplete algorithms due to their relative implementation simplicity. For instance, accelerators based on analog computation have been explored, but they often suffer from limitations in scalability, fixed solving algorithms, and low solvability for complex problems, restricting them to incomplete solvers [7]-[10].

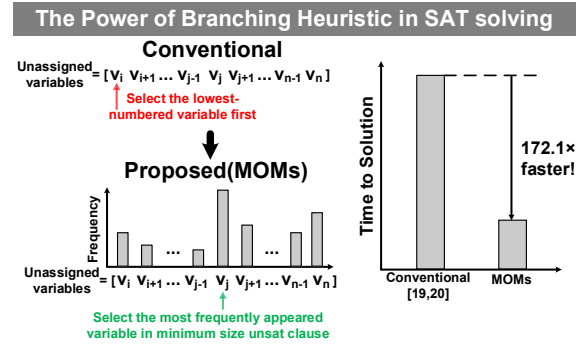


Fig. 1. The benefit of branching heuristics illustrated with an example of MOMs (Maximum Occurrence in clauses of Minimum size) across various benchmarks.

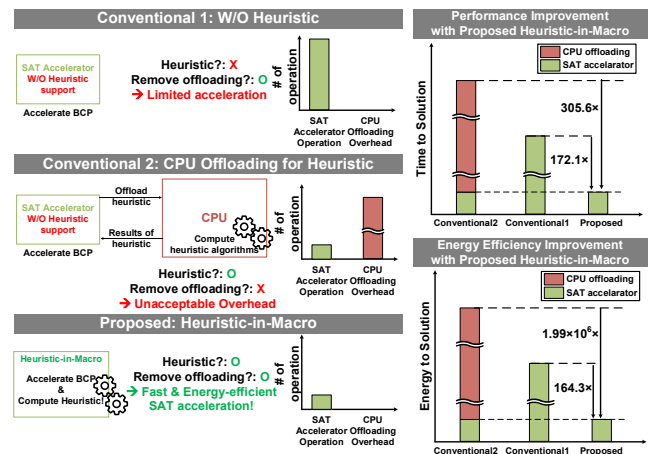


Fig. 2. Limitation of conventional SAT accelerator without heuristic support in 2 scenarios (upper left, middle left), proposed Heuristic-in-Macro (lower left), and Performance/Energy-Efficiency improvement (right).

Other approaches have accelerated the incomplete WalkSAT algorithm[11] but inherently cannot prove unsatisfiability and may require external memory access to calculate break values[12], [13]. More recently, a parallel solving method using a pre-partitioning technique was introduced, but it suffers from significant CPU overhead for the required pre-processing and remains an incomplete solver [14]. While valuable for certain applications, the inherent limitations of these incomplete solvers make them unsuitable for correctness-critical EDA tasks such as formal verification [2]. Therefore, this work focuses on accelerating complete solvers, which can definitively prove unsatisfiability.

Modern high-performance complete solvers are predominantly based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm[4] and its advanced successor, the Conflict-Driven Clause Learning (CDCL) framework [1], [15]. Profiling studies of these algorithms consistently show that the Boolean Constraint Propagation (BCP) operation consumes the vast majority (80-90%) of the total execution

time [16]. Consequently, a significant body of research on both ASIC and FPGA platforms has focused on creating dedicated BCP accelerators. Early efforts on FPGAs demonstrated the feasibility of offloading BCP to reconfigurable hardware but left more complex control flow, such as heuristic decisions, to software [17], [18]. More recent ASIC-based approaches utilized custom memory architectures like CAM and SRAM arrays to further accelerate BCP [16]. This line of research culminated in [19], [20], the first complete solver on an ASIC, which introduced a bidirectional macro highly optimized the BCP process. While these efforts successfully accelerated the BCP operation, they did not address the rest of the DPLL algorithm.

For fast and efficient SAT solving, accelerating BCP alone is not sufficient, as it exposes the next critical performance bottleneck: the **branching heuristic**. The branching heuristic is a vital component of the CDCL/DPLL search that strategically selects the next variable for assignment, thereby actively and dramatically pruning the exponential search space [1], [15], [21]-[23]. As illustrated in Fig 1, without an effective heuristic, the solver must explore a significantly larger number of branches, leading to significantly longer solution times and higher energy consumption. However, highlighted in Fig 2, existing complete hardware solvers face a dilemma:

- 1) Using a trivial on-chip heuristic (e.g., selecting the first unassigned variable [19],[20]) avoids off-chip communication but incurs a severe performance penalty.
- 2) Offloading heuristic computation to a host CPU enables more intelligent search decisions but introduces substantial communication and computation overhead.

Sophisticated heuristics such as MOMs (Maximum Occurrence in clauses of Minimum size) [23] are highly data-intensive, requiring repeated scans of the clause database. As shown in Fig 2, offloading such computation to a host CPU introduces significant overhead, not only from the high latency of data transfer but also from the thousands of CPU cycles required for each heuristic evaluation. Consequently, conventional approaches remain fundamentally constrained—either relying on inefficient search without a branching heuristic or depending on a slow and power-hungry software brain.

To address this critical bottleneck, this work introduces Heuristic-in-Macro, HiM, the first fully autonomous, end-to-end complete SAT solver that tightly integrates an efficient BCP engine with a hardware-embedded branching heuristic into a single macro. The key contributions of this work can be summarized as follows:

1. A fully autonomous Heuristic-in-Macro (HiM) architecture: We present the first on-chip integration of the complete Decide-Propagate-Analyze loop, embedding a sophisticated MOMs heuristic to aggressively prune the search space. The HiM eliminates the CPU involvement in decision making, removing communication and computation overhead. This achieves an average  $172.1\times$  algorithmic speedup over trivial decision strategies, yielding  $305.6\times$  overall acceleration and  $1.99\times 10^6$  times improvement in energy efficiency across diverse benchmarks.
2. A high-throughput parallel processing scheme: To accelerate both BCP and data-intensive heuristic

evaluation, we introduce a fixed-width parallel processing architecture. This approach overcomes the sequential limitations of conventional backward-indexing, enabling simultaneous BCP and heuristic computations, and delivers an average  $8.78\times$  speedup.

3. Superior physical efficiency and performance: By integrating storage and computation into a compact unit cell requiring only 16 transistors for BCP operations, HiM achieves significant reductions in macro area and energy per cycle. Parallel operation within the macro further enhances throughput, surpassing prior ASIC accelerators in both physical efficiency and performance

## II. BACKGROUND AND MOTIVATION

### A. Complete Solver Algorithm: DPLL/CDCL Framework

The foundation of modern complete SAT solvers lies in the DPLL algorithm [4]. As a backtrack search algorithm, DPLL systematically explores the entire solution space through an iterative loop of three core operations: In the Decide step, an unassigned variable is selected and assigned a truth value, branching the search. In Propagate step, BCP deduces the forced assignments of other variables. Finally, in Analyze & Backtrack step, conflicts trigger clause analysis and backtracking to an earlier decision point.

The more advanced CDCL framework extends DPLL with clause learning and non-chronological backtracking, yet the fundamental Decide-Propagate-Analyze loop remains the core computational structure [1], [15]. Within this framework, the overall performance is governed by the Decide step, particularly the branching heuristic. A well-designed heuristic intelligently guides the search, actively pruning the exponential search space by selecting variables that trigger cascades of unit propagations. Each such cascade deterministically assigns multiple variables without requiring further decisions, thereby reducing the number of free variables and dramatically shrinking the search space.

### B. The Role for Branching Heuristics

A variety of branching heuristics have been proposed to maximize the effectiveness of the Decide step. For example, the DLIS (Dynamic Largest Individual Sum) heuristic, used in GRASP [1], [15], selects the literal that occurs most frequently in the unsatisfied clauses (UNSAT clause). The intuition is that assigning this highly constrained literal has the greatest likelihood of producing new unit clauses, thereby triggering unit propagations. The MOMs [23] heuristic refines this strategy further by restricting attention to the variables in the *shortest* UNSAT clauses. Since these minimal-sized clauses are the closest to becoming unit, they impose the strongest constraints. Selecting a variable from this critical subset further increases the probability of immediate and extensive unit propagations, making MOMs even more effective at pruning the search space than DLIS. While these heuristics significantly accelerate solving, they are also data-intensive, require repeated scans of the clause database. This imposes a substantial computational burden on CPU, particularly for large or complex SAT instances, and highlights the need for specialized hardware support.

In fact, the most successful VSIDS (Variable State Independent Decay Sum) heuristic in Chaff [21] and MiniSAT [24], [25], was specifically designed to mitigate the

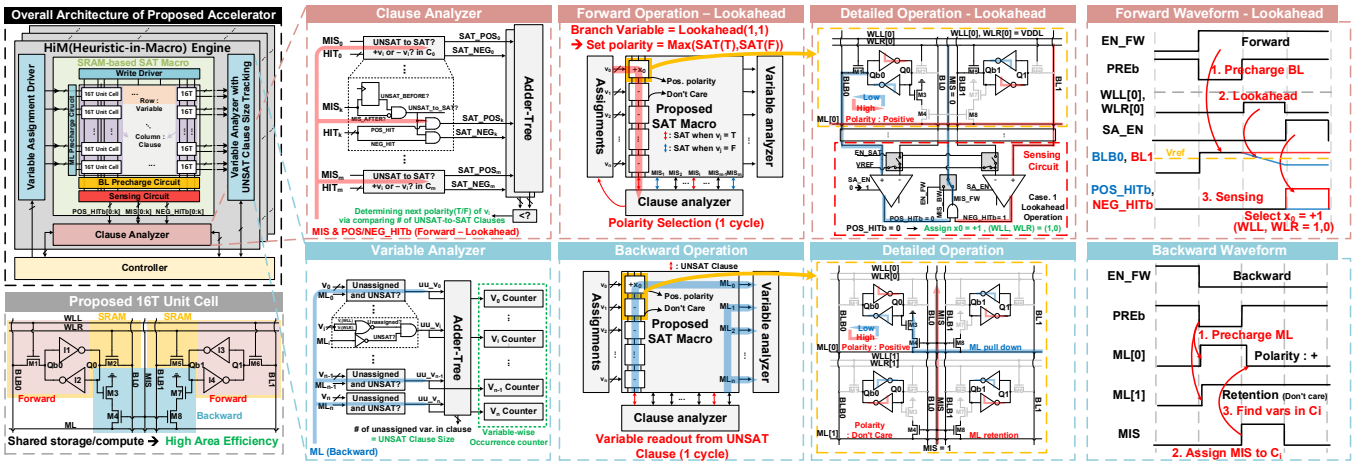


Fig. 3. Overall architecture and Forward/Backward operation of the proposed accelerator: (upper left) Overall architecture; (lower left) proposed 16T unit cell; (columns 2–5) Forward (upper) / Backward (lower) —Variable/Clause Analyzer, Operation block diagram, Circuit-level detail, and transient waveforms.

overhead of full clause-database scans by tracking literal activity only in recently learned conflict clauses. This highlights that the computational cost of the branching heuristic itself is a primary limiter of solver performance. While this is a bottleneck for CPUs, the database-scanning nature of heuristics like MOMs is inherently amenable to massive parallelization in custom hardware macros, an opportunity directly leveraged in this work.

### C. Quantifying the CPU offloading Bottleneck

To evaluate the performance impact of offloading the branching heuristic to a host CPU, we quantify its computational cost under a conservative best-case model. Specifically, we assume an ideal accelerator-CPU interface with zero data transfer latency and infinite bandwidth, thereby isolating the bottleneck to the CPU’s computation alone. In reality, practical implementations would suffer even greater overheads due to limited I/O bandwidth and memory latency, but this assumption provides a definitive lower bound on the offloading cost. Executing the MOMs heuristic on a CPU requires a sequence of operations for each decision: it must first scan the entire clause database to find the minimum length of UNSAT clauses, then scan these specific clauses to identify their literals, count the occurrences of each variable, and finally search for the variable with the highest count.

TABLE I  
ESTIMATED HEURISTIC CPU OFFLOADING OVERHEAD

Heuristics	Average CPU cycles per decision	Energy per decision* [mJ]	Latency per decision* [μs]	Average number of decision to solution	Geomean of energy overhead	Geomean of latency overhead
DLIS[1,15]	$7.372 \times 10^5$	4607.4	368.6	427.5	$2.356 \times 10^5 \times$	350.7*
MOMs[23]	$5.345 \times 10^5$	3340.7	267.3	165.4	$2.002 \times 10^5 \times$	300.5*
MOMs + Lookahead	$6.081 \times 10^5$	3800.5	304.0	137.8	$1.674 \times 10^5 \times$	244.3*
VSIDS[21]	$3.249 \times 10^5$	20.31	1.625	301.9	$1.288 \times 10^4 \times$	2.839*

\*Estimated with Intel Xeon CPU’s Specification [26](single core, 2GHz operation. Energy/cycle = 0.25nJ)

TABLE I summarizes the estimated CPU cycles required to execute a single decision for various branching heuristics. We implemented all branching heuristic in C++ and tested under single core of Intel(R) Xeon(R) Silver 4410Y CPU(2GHz)[26] on 46,400 instances selected from the SATLIB benchmark suite[27]. The analysis reveals that even a single decision can demand thousands to hundreds of thousands of CPU cycles. In stark contrast, hardware-based BCP completes in only a few cycles, making the CPU’s workload for each decision step the dominant system-level bottleneck. This imbalance undermines the purpose of hardware acceleration, as the system is forced to wait on the CPU despite a fast BCP engine. The table also highlights the intrinsic trade-off of heuristics such as VSIDS: although VSIDS reduces per-decision cost by more than 200×

compared to data-intensive heuristics like MOMs, this efficiency comes at the cost of a higher total decision count, demonstrating the CPU-offloaded heuristic’s limited effectiveness.

## III. PROPOSED DESIGN AND FEATURES

### A. Overall Architecture of the Proposed Accelerator

As illustrated in the upper part of column 1 of Fig. 3, the proposed accelerator is composed of multiple tiled Heuristic-in-Macro (HiM) Engines, which a global controller manages. This tiled structure ensures that performance can be scaled linearly with the problem size. Each HiM engine integrates several key components to execute the complete SAT-solving loop autonomously.

As illustrated in column 3 of Fig. 3, the SRAM-based SAT macro supports both a forward operation (upper) and a backward operation (lower). The forward operation evaluates the result of all clauses based on the current variable assignment. In a single cycle, it outputs SAT for satisfied clauses(SAT clause) and UNSAT for unsatisfied clauses(UNSAT clause), allowing for an immediate assessment of the current assignment’s implications.

A key feature of this forward operation is the Lookahead operation, also shown in the upper part of column 3 in Fig. 3. This function supports the simultaneous temporary assignment of both True and False polarities to a single variable, enabling the system to evaluate which assignment satisfies more clauses in one cycle. This capability is used to determine the polarity ordering for a branching heuristic or to identify the correct polarity for a unit clause assignment, both within a single cycle. The backward operation, in contrast, outputs the variables belonging to a specified clause based on its stored literals. This function is essential for performing clause scans.

The Clause Analyzer, shown in the upper part of column 2 in Fig. 3, is used to analyze the results from the Lookahead operation. It uses the MIS signal, which indicates whether a clause is UNSAT (MIS=1) or SAT (MIS=0), to compare the clause’s result before and after the Lookahead assignment. From this comparison, it generates an UNSAT\_to\_SAT signal, which identifies only the clauses that transitioned from UNSAT to SAT. This serves to filter out the results from already-satisfied clauses, preventing them from influencing the Lookahead result. Subsequently, the POS\_HIT and

NEG\_HIT signals, which indicate whether the literal in the clause corresponding to the variable is positive or negative, are combined with the UNSAT\_to\_SAT signal. The result of this AND operation determines if the newly satisfied clause contains a positive (SAT\_POS) or negative (SAT\_NEG) literal of the variable under test. These SAT\_POS and SAT\_NEG signals are then fed into respective adder-trees to sum the total count for each polarity. By comparing these counts, the analyzer determines which polarity satisfies more clauses, enabling a 1-cycle decision for polarity selection during a heuristic branch or unit clause assignment.

The Variable Analyzer, depicted in the lower part of column 2 in Fig. 3, functions based on the output of the backward operation. By comparing the ML (match line) vector from a specified clause with the current state of variable assignments, it identifies which variables are both unassigned and present in that clause. These identified unassigned variables are summed by an adder-tree to calculate the number of unassigned variables in the clause (i.e., its size).

If the clause size is one, it is identified as a unit clause, which in turn triggers a Lookahead operation to execute unit propagation. Otherwise, with each backward operation, the analyzer updates variable-wise counters that track the number of UNSAT clauses each unassigned variable appears in. This output can be used directly to implement the DLIS heuristic by selecting the variable with the highest count. Alternatively, the MOMs heuristic is implemented by adding a condition: the counters are only updated for clauses that match the current minimum size. If a clause with a new, smaller minimum size is found, all counters are cleared, and the counting process is restarted to consider only variables in clauses of this new minimum size.

The lower part of column 1 bottom of Fig. 3 presents the schematics of the unit cell and the sensing circuit. Each unit cell integrates two split-WL 6T SRAM bitcells for literal storage and bitline-based computation, together with two stacked NMOS devices for evaluation, producing MIS (clause unsatisfied) and ML (match line) outputs for a total of 16T. Unlike prior ASIC-based accelerators [19], [20], which relied on SRAM solely for storage and added separate compute logic, incurring area overhead and reduced density, the proposed unit cell leverages the split-WL 6T SRAM pair for both storage and computation, requiring only four additional transistors for SAT. The sensing circuit likewise reuses conventional SRAM sense amplifiers: when EN\_SAT is enabled, BLB0/BL1 are compared against VREF to generate POS/NEG\_HITb, which are ANDed to form MIS. MIS is also fed back to the unit cell so that variable-clause matches can be observed on ML.

### B. Detailed Operation of Proposed SAT Macro

Fig. 3 (column 4, upper) illustrates the Forward-Lookahead operation for a single-clause. During Lookahead (polarity test), both polarities of the candidate variables are temporarily applied by simultaneously driving WLL/WLR. The sense circuit then evaluates POS/NEG\_HITb and MIS to determine which polarity satisfies more clauses. In this example, the unit cell in Row 0 stores a positive literal (Qb0, Q0, Qb1, Q1 = 0, 1, 0, 1). Driving (WLL[0], WLR[0]) = (1, 1) (under-drive) causes BLB0 to discharge while BL1 remains. With SA\_EN enabled, the sense path reports OUTb0

TABLE II  
MACRO OPERATION BASED ON NODE CONDITIONS

Variable		Polarity				→		After Sensing			
WLL	WLR	Qb0	Q0	Qb1	Q1	BLB0	BL1	POSb	NEGb	MIS	ML
-	-	Don't Care (1,0,0,1)				R	R	1	1	1	1
+1	-1	Positive (0,1,0,1)				D/R	R	0/1	1	0/1	1/0
(1,0)	(0,1)	Negative (1,0,1,0)				R	R/D	1	1/0	1/0	0/1
Unassigned (0,0)		Positive / Negative				R	R	1	1	1	0
Lookahead (1,1)		Positive / Negative				D/R	R/D	0/1	1/0	0	1

R : Retention (High-Z), D : Discharge, MIS<sub>init</sub> = 0, POSb/NEGb = POS/NEG\_HITb

low and OUT1 high, indicating that the clause is satisfied for +1. Conversely, the opposite drive condition results in OUT1 low, representing the negative-literal case. The macro then fixes the assignment (e.g., (WLL[0], WLR[0]) = (1, 0)) and proceeds.

Fig. 3 (column 4, lower) shows the backward operation for a single-clause. The unit cell in Row 0 stores the positive literal +x0 (Qb0, Q0, Qb1, Q1 = 0, 1, 0, 1), while Row 1 encodes a don't-care entry (1, 0, 0, 1) for a variable not present in the clause. After PREB precharges ML, asserting the clause's MIS signal activates a pull-down path through stacked NMOS M3–M4 in Row 0, discharging ML only for the row that contains the literal. In contrast, for the non-included variable in Row 1, M3/M7 remain off, leaving ML charged. The resulting ML outcome is then processed as in Fig. 3 (column 4, lower) to identify unit clauses and update per-variable occurrence counts.

TABLE II summarizes the functional behavior of the proposed SAT macro by mapping stored literals and split-WL drive conditions to the resulting bitline/sense outputs. For don't-care rows, ML remains high and MIS is asserted regardless of assignment. If the polarity matches the stored literal, the clause is satisfied and the MIS is low. If a mismatch occurs, the MIS becomes high and outputs the variables included in the clause through ML.

### C. High Throughput Parallel Scheme

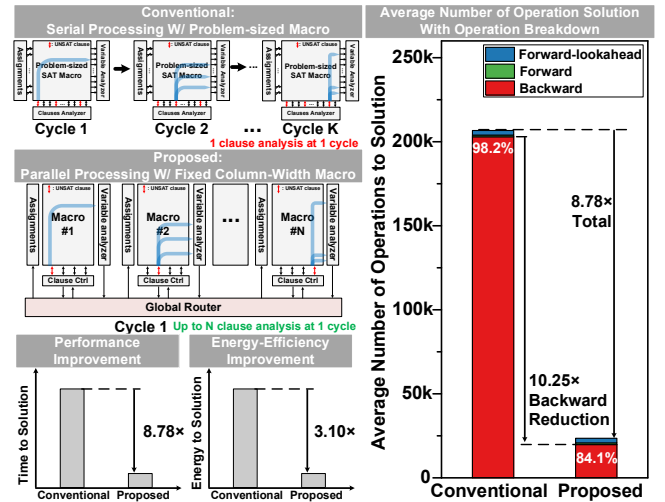


Fig. 4. Architectural comparison between conventional(upper left) and proposed(middle left), number of operation breakdown in conventional and proposed(right), and Performance/Energy-Efficiency improvement(lower left).

Since the proposed accelerator adopt DPLL-based complete algorithm with a MOMs heuristic, most operations involve BCP and MOMs heuristic evaluation—both of which are highly data-intensive, requiring clause scans. As shown in Fig. 4, conventional problem-sized macros rely on serialized clause scans (Backward operation), which account for 98.2% of total operation time. To address this bottleneck, we introduce a parallel architecture based on a fixed column-width macro. Unlike prior ASIC SAT accelerators [12] - [14],

[19], [20], that scale macros to problem size—thereby serializing backward processing at one cycle per UNSAT clause—the proposed design tiles multiple tiled fixed-size macros, enabling parallel Backward operation. Specifically, if a single macro requires  $k$  cycles to process  $k$  UNSAT clauses, an array of  $N$  macros can process up to  $N$  UNSAT clauses per cycle, effectively reducing latency to approximately  $k$  over  $N$ . This maximizes throughput while preserving small, reusable macro blocks. While compatible with various sizes of macro, we demonstrate this parallel architecture with 128-variable/32-clause sized macro. As shown in Fig. 4, it achieves an average  $8.78\times$  speedup in time-to-solution and an average  $3.10\times$  energy-efficiency improvement in energy-to-solution compared with a single problem-sized macro baseline across all tested benchmark.

#### IV. EVALUATION RESULTS

##### A. Evaluation Setup

The proposed design is evaluated in 28-nm CMOS technology, with detailed analysis of area, latency, and power consumption. Energy efficiency is derived through post-layout simulations and Monte-Carlo analysis. Cycle-level throughput and scalability are also assessed against prior ASIC SAT accelerator and a CPU baseline. Schematic entry and layout were performed using Cadence Virtuoso, while parasitic extraction was carried out with Synopsys StarRC. Post-layout and Monte-Carlo simulations were conducted in Synopsys HSPICE and PrimeSim, from which macro-level latency and power values were obtained. For algorithmic evaluation, macro-side operation counts were measured through a Python-based macro simulator, while the CPU baseline was implemented in C++ and executed on a single core Intel(R) Xeon(R) Silver 4410Y CPU(2GHz)[26]. Both hardware and CPU baselines were tested on the same benchmark set to ensure fair comparison. The benchmarks used in our experiments were drawn from the SATLIB[27]. Specifically, we selected 46,400 instances that all problems with from 20 to 125 variable sizes to ensure a diverse and thorough evaluation.

##### B. Macro-level Comparison

Fig. 5 shows the layout of a single macro from the proposed design, along with its area/energy breakdown. The macro consists of the SAT array and the peripheral circuits—write drivers, sensing circuit, and bitline precharge circuit—as well as the Variable Analyzer and Clause Analyzer. Both pre-simulation and post-layout simulation with parasitic RC extraction, including Monte Carlo analysis, were performed to validate stability and performance. In terms of area breakdown, the array accounts for 59%, the peripheral circuits 26%, and the Variable/Clause analyzers together 15%. The reported

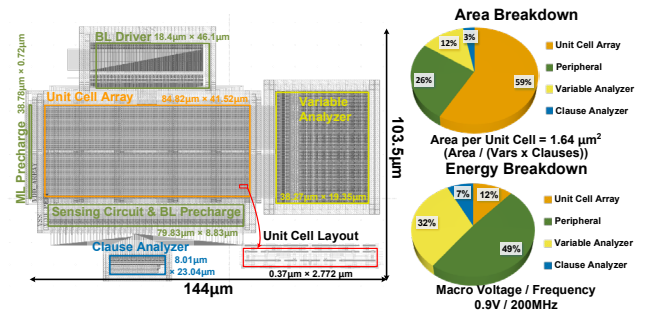


Fig. 5. Layout of single macro (left), Area Breakdown (right upper) and Energy Breakdown (right lower)

area per unit cell, defined as the total macro area (excluding the power ring) divided by the number of unit cells, is  $1.64 \mu\text{m}^2$ , representing a compact footprint compared with prior SAT macros. The energy profile is reported as energy per cycle, averaged over 1,000 instances of uf100-430 [27]. At 0.9 V and 200 MHz, a single macro consumes 0.87 pJ per cycle, highlighting both energy efficiency and suitability for large-scale integration.

##### C. Performance/Energy-Efficiency Improvement Analysis

TABLE III compares the proposed accelerator with prior ASIC-based accelerators. Unlike earlier designs that duplicated compute logic in each unit cell, our macro shares logic resources and reuses SRAM structure for both storage and computation. This optimization yields a compact unit cell with an area efficiency of  $1.82 \mu\text{m}^2/\text{unit cell}$ . By integrating the branching heuristic inside the macro (Heuristic-in-Macro), the accelerator eliminates the need for CPU offloading. As a result, it functions as a fully autonomous complete SAT solver, achieving 100% solvability on bot SAT and UNSAT benchmarks from SATLIB.

The performance gains are substantial. On SAT benchmarks uf50-218/uf100-430, HiM approach reduces time-to-solution (TTS) by 41%/94% (to  $10.1\mu\text{s}$  /  $116.8\mu\text{s}$ ), compared to [19, 20]. For uf50-218, the energy-to-solution (ETS) improves by 83%, dropping to 9.56 nJ. On UNSAT benchmarks (uuf50-218 and uuf100-430), HiM delivers TTS reductions of 33%/92% (to  $28.3\mu\text{s}$  /  $366.1\mu\text{s}$ ), while uuf50-218 achieves ETS improvement of 82%, reduced to 25.7 nJ.

Fig. 6 compares ETS and TTS across configurations (with/without heuristic, with/without CPU offloading) on SATLIB. Using MOMs with Lookahead operation, the proposed HiM accelerator outperforms a heuristic-free ASIC accelerator by  $137.5\times/144.2\times$  in GeoMean ETS/TTS, and by  $164.4\times/172.1\times$  in Average ETS/TTS, demonstrating the benefit of embedding heuristics within the macro. Moreover, because the heuristic is on-chip, i.e. no CPU offload, the proposed accelerator surpasses a CPU-offloaded heuristic

TABLE III  
Comparison Table with State of the Arts ASIC based Accelerators

	CICC'22 [7]	ISSCC'23 RNN+PIM [8]	ISSCC'23 Snap [12]	SOVC'24 [9]	ISSCC'24 [14]	SOVC'25 [13]	ISSCC'25 [19], [20]	This Work		
Technology	65nm	65nm	65nm	65nm	65nm	55nm	28nm	28nm		
Computing Domain	Analog	Mixed	Digital	Analog	Digital	Mixed	Digital	Digital		
Supply Voltage[V]	1-1.2	1.2	0.7-1.2	N.R	1-1.4	1.3	0.65-0.9	0.7-1.0		
Core Area [mm <sup>2</sup> ]	4.41	0.4	0.93	0.37	1.12	0.544	0.2	0.015 per macro		
Area Efficiency* [μm <sup>2</sup> /(unit cell)]	590	12.2	6.03	193.6	102	49.9	8.49	1.82		
Heuristic Acceleration	X	X	X	X	X	X	X	0 Geomean 144.2x, Avg. 172.1x		
Off-chip Operation	Not required	Not required	WalkSAT	Not required	Pre-partition	Not required	Not required	Not required		
Solver Type	Incomplete	Incomplete	Incomplete	Incomplete	Incomplete	Incomplete	Complete	Complete		
Solvability	~16%	46.5%	72%	100%	98%	98%	100%	100%		
Tested Problem Clauses/Variables	30/128	60/252	60/258	20/91	50/218	50/218	50/218 & 100/430		50/218 & 100/430	
							SAT	UNSAT	SAT	UNSAT
Time to Solution	0.9ms	>11.25ms	0.71ms	6.6μs	18.7μs	45μs	17.1μs / 2.12ms*	42.1μs / 4.59ms*	10.1μs / 116.8μs**	28.3μs / 366.1μs**†
Energy to Solution	24.3μJ	N.A.	1.1μJ	5nJ	20.8nJ	518nJ	58.0nJ / N.R*	142.7nJ / N.R*	9.56nJ / 234.0nJ**	25.7nJ / 711.0nJ**†

\*Area Efficiency = (Total Area) / (Variables × Clauses) †uf50-218 / uf100-430 (UNSAT for uuf50-218 / uuf100-430) ‡V<sub>DD</sub>=0.9V, f<sub>CLK</sub>=200MHz condition



## REFERENCES

- [1] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 506–521, May 1999.
- [2] Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
- [3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 1020 states and beyond," *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, Jun. 1992.
- [4] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962.
- [5] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Annu. ACM Symp. Theory Comput.*, 1971, pp. 151–158.
- [6] E. Boros and P. L. Hammer, "Pseudo-Boolean optimization," *Discrete Appl. Math.*, vol. 123, nos. 1–3, pp. 155–225, Nov. 2002.
- [7] M. Chang, X. Yin, Z. Toroczka, X. Hu and A. Raychowdhury, "An Analog Clock-free Compute Fabric base on Continuous-Time Dynamical System for Solving Combinatorial Optimization Problems," 2022 IEEE Custom Integrated Circuits Conference (CICC), Newport Beach, CA, USA, 2022, pp. 1-2.
- [8] D. Kim, N. M. Rahman and S. Mukhopadhyay, "29.1 A 32.5mW Mixed-Signal Processing-in-Memory-Based k-SAT Solver in 65nm CMOS with 74.0% Solvability for 30-Variable 126-Clause 3-SAT Problems," 2023 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2023, pp. 28-30.
- [9] Q. Zhang et al., "A Stochastic Analog SAT Solver in 65nm CMOS Achieving 6.6 $\mu$ s Average Solution Time with 100% Solvability for Hard 3-SAT Problems," 2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits), Honolulu, HI, USA, 2024, pp. 1-2.
- [10] E. Dikopoulos, Y. Hsu, L. Wormald, W. Tang, Z. Zhang and M. P. Flynn, "25.1 A Physics-Inspired Oscillator-Based Mixed-Signal Optimization Engine for Solving 50-Variable 218-Clause 3-SAT Problems with 100% Solvability and 31.7 $\mu$ s Solution Time," 2025 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2025, pp. 01-03.
- [11] B. Selman, H. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," *Cliques, Coloring, Satisfiability*, vol. 26, pp. 521-531, Oct. 1996.
- [12] S. Xie et al., "29.2 Snap-SAT: A One-Shot Energy-Performance-Aware All-Digital Compute-in-Memory Solver for Large-Scale Hard Boolean Satisfiability Problems," 2023 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2023, pp. 420-422.
- [13] T. Bhattacharya, D. Kwon, G. H. Hutchinson, X. Zhang, I. Rozada and D. Strukov, "A Fully Integrated Mixed-Signal Compute-In-Memory Accelerator for Solving Arbitrary Order Boolean Satisfiability Problems," 2025 Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits), Kyoto, Japan, 2025, pp. 1-3.
- [14] C. Shim, J. Bae and B. Kim, "30.3 VIP-Sat: A Boolean Satisfiability Solver Featuring 5 $\times$ 12 Variable In-Memory Processing Elements with 98% Solvability for 50-Variables 218-Clauses 3-SAT Problems," 2024 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2024, pp. 486-488.
- [15] J. P. Marques Silva and K. A. Sakallah, "GRASP-A new search algorithm for satisfiability," *Proceedings of International Conference on Computer Aided Design (ICCAD)*, San Jose, CA, USA, 1996, pp. 220-227..
- [16] S. Park, J. -W. Nam and S. K. Gupta, "HW-BCP: A Custom Hardware Accelerator for SAT Suitable for Single Chip Implementation for Large Benchmarks," 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 2021, pp. 29-34.
- [17] J. D. Davis, Zhangxi Tan, Fang Yu and Lintao Zhang, "A practical reconfigurable hardware accelerator for boolean satisfiability solvers," 2008 45th ACM/IEEE Design Automation Conference, Anaheim, CA, USA, 2008, pp. 780-785.
- [18] J. Thong and N. Nicolici, "FPGA acceleration of enhanced boolean constraint propagation for SAT solvers," 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 2013, pp. 234-241.
- [19] Z. Wu et al., "37.5 SKADI: A 28nm Complete K-SAT Solver Featuring Dual-Path SRAM-Based Macro and Incremental Update with 100% Solvability," 2025 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2025, pp. 614-616
- [20] Z. Wu et al., "SKADI: A 28-nm Complete K-SAT Solver Featuring Bidirectional In-Memory Deduction and Incremental Updating," in *IEEE Journal of Solid-State Circuits (Early Access)*.
- [21] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: engineering an efficient SAT solver," *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, Las Vegas, NV, USA, 2001, pp. 530-535.
- [22] R. G. Jeroslow and J. Wang, "Solving propositional satisfiability problems," *Annals of Mathematics and Artificial Intelligence*, vol. 1, pp. 167–187, 1990.
- [23] C.M. Li and Anbulagan, "Heuristics based on unit propagation for satisfiability problems," In *Proceedings of the 15th international joint conference on Artificial intelligence (IJCAI)*, San Francisco, CA, USA, 1997, pp. 366–371.
- [24] N. E'én and N. S'orensson, "An extensible SAT solver," in *Proc. Int. Conf. Theory Appl. Satisfiability Test. (SAT)*. Cham, Switzerland: Springer, 2003, pp. 502–518.
- [25] N. S'orensson, N. E'én, "MiniSat 2.1 and MiniSat++ 1.0 — SAT Race 2008."
- [26] Intel. "Intel® Xeon® Silver 4410Y Processor (30M Cache, 2.00 GHz) Specifications." Intel.com. Accessed: Sep. 15, 2025. [Online]. Available: <https://www.intel.com/content/www/us/en/products/sku/232376/intel-xeon-silver-4410y-processor-30m-cache-2-00-ghz/specifications.html>
- [27] H. H. Hoos, "SATLIB: Benchmark Problems for SAT Solvers," University of British Columbia. Accessed: Apr. 15, 2025. [Online]. Available: <https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html>.