

Heterogeneous Synergy: Fusing SAT Solvers and GPU Simulation for CEC

Yang Sun
The Chinese University of
Hong Kong
Hong Kong, China
ysun22@cse.cuhk.edu.hk

Tianji Liu*
Institute of Computing
Technology, CAS
Beijing, China
liutianji@ict.ac.cn

Martin D.F. Wong
Hong Kong Baptist
University
Hong Kong, China
mdfwong@hkbu.edu.hk

Evangelina F.Y. Young
The Chinese University of
Hong Kong
Hong Kong, China
fyyoung@cse.cuhk.edu.hk

Abstract

Combinational equivalence checking (CEC) is a critical task in hardware verification whose computational cost increasingly challenges modern tools. Existing approaches predominantly rely on either SAT solving or simulation, yet each has inherent limitations. Simulation can verify large numbers of equivalent pairs quickly but stalls on structurally complex ones, while SAT solvers can formally resolve hard pairs but lack massive parallelism. We present the first heterogeneous CEC engine that couples exhaustive-simulation proving and SAT in a closed feedback loop. A dynamic scheduler monitors verification progress and selects whether to run GPU-based simulation and CPU-based SAT solving cooperatively or competitively based on the characteristics of remaining unresolved pairs at each stage. Two algorithmic contributions augment this framework: iterative sieving, which progressively allocates SAT solver effort to maximize proof throughput, and adaptive AIG resynthesis, which restructures stalled instances to expose new verification opportunities. Our engine achieves average speedups of $12.29\times$ over ABC, $47.81\times$ over the commercial tool Conformal, and $3.19\times$ over the state-of-the-art simulation-based CEC tool, solving all benchmark instances without external engines. On the most challenging cases, verification time drops from hours to minutes, with one benchmark completed in under 5 minutes, compared to 15.1 hours with ABC and 6.7 hours with the simulation-based tool.

ACM Reference Format:

Yang Sun, Tianji Liu, Martin D.F. Wong, and Evangelina F.Y. Young. 2026. Heterogeneous Synergy: Fusing SAT Solvers and GPU Simulation for CEC. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26)*, November 08–12, 2026, San Jose, CA, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3831252.3834152>

1 Introduction

Combinational equivalence checking (CEC) is a fundamental task in the realization flow of modern digital designs. Its objective is to formally prove the functional equivalence of two netlists intended to implement the same combinational logic.

The difficulty of CEC becomes clear when contrasted with other core EDA tasks. Much of the EDA toolchain, including logic optimization, technology mapping, floorplanning [16, 27], partitioning [17], placement, and routing, can be formulated as optimization problems over area, delay, or wirelength, where heuristic algorithms find high-quality solutions within practical runtime budgets and scale to millions of gates. By contrast, CEC demands an exact answer and must be solved to completion, leaving no room for approximation and making it inherently harder to scale.

CEC is co-NP-complete [18], so no single algorithm can efficiently handle all design types. The field has evolved from Binary Decision Diagrams (BDDs) [7, 11] to Boolean Satisfiability (SAT) solving [19, 20], now the industry-standard technique for CEC, driven by decades of improvements in solver performance [3, 21]. Despite this, growing design scale increasingly exposes the scalability limits of SAT-based verification, motivating new acceleration approaches. One such approach is GPU simulation-based CEC [15], which verifies large volumes of equivalence candidates through exhaustive pattern simulation in a single parallel sweep, achieving significant speedups over conventional SAT-based tools. Its effectiveness, however, depends on the existence of compact common cuts between candidate pairs. When such cuts cannot be found, the simulation engine has no path to a proof and must fall back to a SAT-based checker. These simulation-resistant pairs reveal a fundamental complementarity: SAT solvers resolve them through constraint reasoning but lack the data parallelism GPU simulation exploits. Neither paradigm alone is sufficient, yet their failure modes are largely non-overlapping.

We present a heterogeneous CEC engine that, to our knowledge, is the first to combine GPU-based exhaustive-simulation proving with multi-threaded CPU SAT solving in a closed feedback loop. Unlike prior approaches that treat the two as independent stages, the output of each directly guides the other. A dynamic scheduler classifies candidate pairs and allocates them to the appropriate processor in real time, switching between cooperative and competitive execution as verification progresses. Iterative sieving sweeps easy proofs first under a small conflict limit and then raises it progressively, letting the SAT solver concentrate on increasingly harder pairs. When verification stalls, adaptive AIG resynthesis restructures the miter, creating proof opportunities that neither engine could exploit on the original circuit. Because all three mechanisms operate on the same miter and each simplifies it for the others, their combined effect exceeds the sum of their individual contributions.

The engine resolves all benchmark instances without external verification tools, whereas the prior simulation-based approach requires a fallback SAT checker. It achieves average speedups of

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License. ICCAD '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2873-0/2026/11

<https://doi.org/10.1145/3831252.3834152>

12.29 $\times$ (up to 189 $\times$) over ABC [6], 47.81 $\times$ (up to 164 $\times$) over the multi-threaded commercial tool Conformal, and 3.19 $\times$ (up to 83 $\times$) over the state-of-the-art simulation-based CEC tool from DAC'25 [15]. On the hardest benchmarks the improvement is transformative, with *voter_10xd* completed in under 5 minutes versus 15.1 hours (ABC) and 6.7 hours (DAC'25), and *sqr_10xd* in about 25 minutes versus around 6 hours in ABC. Ablation studies confirm that iterative sieving and adaptive AIG resynthesis each contribute independently and combine effectively.

2 Preliminaries

2.1 Background

In logic synthesis, a combinational circuit is usually represented as a *Boolean network*. A Boolean network is a *directed acyclic graph* (DAG), where the nodes symbolize logic functions or gates, and the edges denote signals or nets between the nodes. The sources and sinks of the DAG are the *primary inputs* (PIs) and *primary outputs* (POs) of the Boolean network. For any node n , the *fanins* of n are a set of nodes that have an outgoing edge towards n , i.e., driving n . Similarly, the *fanouts* of n are a set of nodes that have an incoming edge from n , i.e., driven by node n . The predecessors of a node are the *transitive fanins* (TFIs) of the node, and collectively, these predecessors constitute the *transitive fanin cone* of the node. A node's *transitive fanouts* (TFOs) and *transitive fanout cone* can be defined similarly. The function of a node expressed in terms of PIs is the *global function* of the node. A *cut* C_n of a node n in a Boolean network is a set of nodes such that every path from any PI to n contains a node in the set. The *local function* of a node is the function of a node in terms of its cut nodes.

An *and-inverter graph* (AIG) [12] is a specific type of Boolean network where each node is an AND gate with exactly two fanins, and the signals represented by the edges may be inverted as needed. The *level of a node* denotes the length of the longest path from a PI to the node, while the *level of the network* is the maximum level among its POs. AIG is one of the most common circuit representations in modern logic synthesis and verification tools. In this work, we also focus on equivalence checking on AIGs, but the framework is applicable to other representations.

Logic synthesis is a fundamental design step that optimizes a circuit's netlist to improve key metrics like area and delay. Window-based resubstitution [5] is one of the most powerful local optimization techniques, which simplifies a target node by resynthesizing its logic using existing nodes near the target.

Satisfiability don't care (SDC) refers to certain input combinations (also called patterns) that cannot be generated due to the limited controllability of the network. For example, in the network with nodes $n_1 = x + y$, $n_2 = yz$ and $n_3 = n_1n_2$, the pattern $\{(n_1 = 0, n_2 = 1)\}$ is one of n_3 's SDCs. The local function of n can be changed without affecting the global function of n under the SDC assignments [24].

2.2 SAT Sweeping

A typical CEC engine checks the equivalence of two circuits by combining them into a single circuit called *miter* [4], constructed by pairwise connecting the PIs and pairwise combining the POs using XOR gates. The problem is then to check whether all POs

of the miter are constant zeros, meaning that all PO pairs of the original circuits are equivalent.

The *Boolean satisfiability problem* (SAT problem) determines whether some assignment to the input variables makes a Boolean formula true. If such an assignment exists the formula is *satisfiable* (SAT); otherwise it is *unsatisfiable* (UNSAT).

The process of proving equivalence using the miter and SAT is called *mitering* [19]. The miter is encoded into a conjunctive normal form (CNF) instance by Tseitin transformation [26], and the variables of the miter POs are asserted to be constant ones. If the solver returns SAT, at least one PO pair is non-equivalent; if UNSAT, the circuits are proven equivalent.

However, direct mitering is inefficient, so a common strategy is to simplify the circuits first. SAT sweeping [12, 20] identifies and merges functionally equivalent internal nodes: node pairs are checked in topological order and merged if a SAT proof confirms their equivalence, reducing the size and complexity of the final miter. Previous attempts to accelerate CEC have explored a hybrid CPU-GPU framework [8] and multi-core CPU parallelism [23], but these often suffer from limited scalability or dependency on circuit partitioning strategies.

2.3 Simulation-based Parallel Sweeping

While SAT sweeping has become the dominant paradigm in CEC, recent work [15] introduced a CEC engine built around exhaustive simulation on GPUs, leveraging their data-parallel architecture to rapidly verify equivalences.

This engine is structured as a multi-phase flow consisting of a PO checking phase (P), a global function checking phase (G), and repeated local function checking phases (L). Phase *P* proves equivalence at simulatable miter POs in terms of their global functions, reducing the miter early so that checking efforts on internal pairs in the removed portion can be saved. Phase *G* verifies simulatable internal node pairs whose support sizes fall below a given threshold by exhaustive simulation of their global functions, while collecting counter-examples to refine equivalence classes. If the miter is not fully resolved, Phase *L* is conducted repeatedly: each pass performs cut generation and exhaustive local function checking over candidate pairs using a cut enumeration algorithm that generates multiple high-quality common cuts per pair, increasing the probability of proving equivalence without being affected by SDCs. Since miter reduction between passes changes the AIG structure, new cuts can be generated that resolve pairs that previously failed.

This approach achieves 4.89 $\times$ over a standalone ABC checker, validating GPU-based simulation as a complement to SAT solving and motivating our tightly integrated heterogeneous framework.

3 Heterogeneous CEC Engine

This section presents our heterogeneous CEC engine. Section 3.1 gives an architectural overview. Section 3.2 describes the dynamic task scheduling strategy that allocates candidates between simulation and SAT solving. Section 3.3 introduces two techniques for breaking verification stalls. Section 3.4 presents the complete control flow and concludes with a discussion of soundness and completeness.

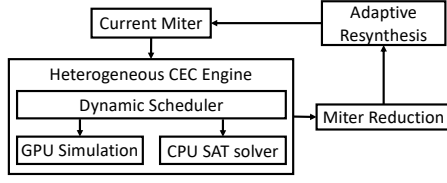


Figure 1: Block diagram of the proposed heterogeneous CEC framework.

3.1 Architecture Overview

SAT sweeping is the dominant methodology in modern CEC, yet no single reasoning engine is universally efficient. GPU simulation verifies large batches of candidate pairs with high throughput, but its effectiveness depends on finding compact common cuts and degrades when satisfiability don't cares are present. SAT solving formally resolves structurally complex pairs regardless of cut quality, but is inherently sequential and becomes a bottleneck when many pairs remain. These complementary strengths motivate a heterogeneous engine that dynamically distributes work between the two.

The flow begins with Phases *P* and *G* for global function checking. Phase *P* checks simulatable miter POs, reducing the miter before internal pairs are examined. Phase *G* applies global function checking to internal node pairs whose support sizes do not exceed a threshold, while collecting counter-examples to refine the equivalence classes. These two phases follow [15] and are not the focus of this paper.

When unresolved pairs remain, the engine enters Phase *L*, which performs local function checking through a heterogeneous iterative loop. As shown in Figure 1, at its center is a GPU simulation engine and a CPU SAT engine, coordinated by a dynamic scheduler that classifies candidate pairs and allocates them to the appropriate processor. Proven equivalences feed back into a miter reduction step that simplifies the current miter before the next iteration. When both engines stall, adaptive resynthesis restructures the AIG globally, creating fresh proof opportunities.

Early iterations contain many pairs that the GPU resolves quickly, but as these are eliminated, the remaining pairs increasingly resist simulation and require formal reasoning. Phase *L* therefore operates in two modes. Cooperative mode divides work so that each processor handles the pair types best suited to its architecture: the GPU processes pairs amenable to simulation while the CPU formally resolves the rest through SAT solving. When simulation yields diminishing returns, the engine switches to competitive mode, where the CPU takes over all remaining pairs while the GPU works in parallel as an accelerator. The scheduler manages mode selection, pair classification, and task dispatch across all available threads, as detailed in Section 3.2.

Even with effective scheduling, two bottlenecks can stall progress. The SAT solver may exhaust its time budget on a few intractable pairs while leaving many solvable ones untouched, and certain local structures may remain opaque to both engines regardless of effort spent. To address these, the framework incorporates iterative sieving and adaptive AIG resynthesis, described in Section 3.3.

3.2 Dynamic Task Scheduling

3.2.1 Cooperative Mode. Cooperative mode is designed to augment the conventional level-wise flow of simulation-based CEC by integrating formal methods of SAT solvers. The proposed heterogeneous flow is tightly pipelined between the GPU and CPU engines to maximize resource utilization, as shown in Figure 2(a).

The process begins with the standard level-wise cut generation on the GPU. At each enumeration level [15], the GPU engine identifies candidate pairs and attempts to find feasible common cuts for them. However, a subset of pairs inevitably fails this process. Instead of deferring these simulation-resistant pairs to subsequent passes, which is a common strategy in the purely simulation-based approach, cooperative mode immediately dispatches them to the CPU-based SAT engine. This forms the core of the cooperative strategy: the GPU proceeds with the high volume of simulation-friendly pairs, while the CPU concurrently begins the formal analysis of the simulation-resistant ones. This process is pipelined to maximize concurrency: as soon as cut generation for level k completes on the GPU, the resulting simulation-resistant pairs are enqueued for the CPU, allowing the CPU to work on level k while the GPU moves on to level $k + 1$.

Once the GPU has completed its simulation checking phase for all levels, it signals the CPU engine that no new tasks will be dispatched for the current iteration. Upon receiving this signal, the SAT solvers do not terminate immediately. Instead, they enter a pending task resolution phase, where they continue to process the tasks already in their queue. This phase ensures that CPU cycles are not wasted and allows progress to continue on the most difficult pairs while the GPU waits. Finally, upon completion of the pending task resolution phase, the SAT solvers report the status of all pairs they have processed. These results are then merged with the results from the GPU engine to update the global equivalence classes before the next CEC pass begins.

3.2.2 Competitive Mode. In contrast to the division of labor in cooperative mode, competitive mode employs a race-based strategy designed to minimize the verification latency for individual candidate pairs. Figure 2(b) illustrates the flow of competitive mode. All candidate pairs are dispatched to the CPU-based SAT engine at the very beginning of the verification pass, while the GPU is responsible for the simulatable pairs.

Task scheduling and result merging are straightforward. When the GPU completes its simulation workload, it signals the CPU. The SAT solvers then transition into a pending task resolution phase, completing any ongoing proofs before halting. Specifically, a proof of equivalence from either the GPU or the CPU is accepted, while a proof of non-equivalence can only be formally established by the SAT solver due to SDCs in local function checking. The results from both sides are merged in parallel on the GPU.

The primary advantage of this approach is its efficient use of parallel processing. Instead of waiting for the GPU, the CPU's SAT solvers start formal analysis at the beginning. This establishes two parallel, complementary workflows: a high-throughput proof on the GPU and a formal proof process on the CPU. This concurrent model ensures that all compute resources are productive from the start, significantly reducing the total time to reach a definitive result.

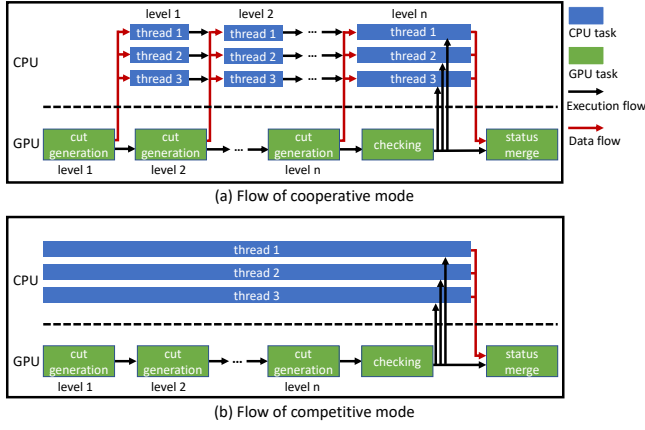


Figure 2: Execution flow of two modes in task scheduling.

Algorithm 1 Scheduling of Heterogeneous Engine

```

Input: Miter miter, number of threads threads, mode mode
1: buf  $\leftarrow \emptyset$  ▷ allocate common cut buffer
2: Compute enumeration levels el( $\cdot$ ) of all nodes
3: Initialize SAT solvers for each thread
4:  $l_m \leftarrow \max_{n \in \text{miter}} el(n)$ 
5: for each  $l = 1, \dots, l_m$  do
6:   enum  $\leftarrow \{n \in \text{miter} : el(n) = l, repr(n) \neq n\}$ 
7:   if mode = 2 then satPairs[l]  $\leftarrow \text{enum}$ 
8:   procedure GPUTASK[l]
9:     parallel generate valid common cuts of pairs as cmn
10:     $p \leftarrow \{(repr(n), n) : n \in \text{enum}, cuts(n) \in \text{cmn}\}$ 
11:    buf  $\leftarrow \text{buf} \cup \text{cmn}$ 
12:    if mode = 1 then satPairs[l]  $\leftarrow \text{enum} - p$ 
13:  for each  $t = 1, \dots, \text{threads}$  do
14:    procedure CPU TASK[l, t]
15:      satPairs[l, t]  $\leftarrow \text{allocateSatPairs}(\text{satPairs}[l], t)$ 
16:      CallSatSolver(satPairs[l, t], t)
17:      if mode = 1 then Precede(GPUsat[l], CPUtask[l, t])
18:      if  $l > 1$  then Precede(CPUtask[ $l-1$ , t], CPUtask[l, t])
19:      if  $l > 1$  then Precede(GPUsat[ $l-1$ ], GPUsat[l])
20:  procedure GPU CHECKING
21:    if  $|\text{buf}| > 0$  then ▷ check the final batch
22:      Local function checking using cuts in buf
23:      terminateSat  $\leftarrow \text{true}$ 
24:    Precede(GPUsat[ $l_m$ ], GPUchecking)
25:    RunTaskflow()
26:    Merge status of proven pairs from GPU and CPU ▷ in parallel
27:    if proven equivalent pairs then
28:      mode  $\leftarrow 1$ 
29:    else mode  $\leftarrow 2$ 
    
```

3.2.3 Dynamic Mode Switching. The framework incorporates a heuristic for dynamic mode switching to optimize the verification strategy across multiple passes. The choice of mode is directly tied to the progress made in each verification pass, which is measured by whether new equivalent pairs are found.

The default operational state is cooperative mode. This mode is employed as long as the verification process is making progress, defined as identifying at least one new equivalent pair in a pass. However, if a verification pass concludes without identifying any

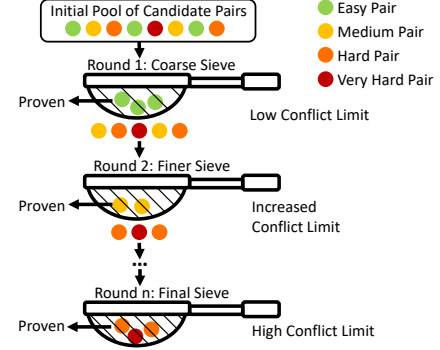


Figure 3: Illustration of iterative sieving.

new equivalent pairs, the AIG remains unchanged. This state of stagnation indicates that the remaining candidate pairs are too difficult for the current strategy. In response to this lack of progress, the framework automatically switches to competitive mode for the next verification pass. The reason is that the parallel, race-based approach of competitive mode may be more effective at resolving these difficult pairs and breaking the impasse. Since the AIG remains unchanged, the SAT solver can directly reuse candidate pairs from the previous pass. The system reverts to the default cooperative mode as soon as progress resumes. If any equivalent pairs are found during a pass, the framework will use cooperative mode for the subsequent pass to continue efficiently simplifying the miter.

Algorithm 1 orchestrates the scheduling using Taskflow [10]. After computing enumeration levels (line 2), the algorithm iterates over levels: a GPU task searches for cuts (lines 8–12) and CPU tasks dispatch SAT solvers (lines 14–16). In cooperative mode the GPU precedes the CPU (line 17), while in competitive mode both run concurrently on the full pair set. A final GPU task checks the accumulated cut buffer and triggers early termination (lines 22–23). Because CPU and GPU results never conflict, the status merge runs in parallel (line 26) and determines the mode for the next pass (lines 27–29).

3.3 Techniques for Breaking Verification Stalls

3.3.1 Iterative Sieving for the SAT Solver. To optimize the allocation of computational resources for the SAT solver, we introduce a strategy called iterative sieving. This approach prevents the solver from becoming stalled on computationally expensive pairs in the verification passes, ensuring that easy proofs across the entire problem set are resolved first. The strategy is governed by two key parameters. The first parameter, conflict limit, defines the maximum computational effort (number of conflicts) the solver may expend on a single candidate pair. If this limit is exceeded, the proof attempt is aborted, the pair's status is marked as undetermined, and it is deferred until a future pass. The second parameter, minimum processed pair count, acts as a work quota for the CPU engine. After receiving the signal that the GPU checking phase has finished, the SAT solvers do not terminate immediately but continue working until either all allocated pairs have been attempted, or this minimum count of processed pairs has been met.

The sieving process spans multiple verification passes, as illustrated in Figure 3. In the initial pass, the conflict limit is set to a low

value and the minimum processed pair count is set larger than the number of pairs assigned to each SAT solver, forcing every solver to make a quick attempt on every pair. In subsequent passes, both parameters increase adaptively, each starting at an initial value of 1. The conflict limit doubles when undetermined pairs exceed resolved ones (SAT and UNSAT), allowing deeper reasoning on harder pairs. The minimum processed pair count doubles when the GPU proves no equivalences in the current iteration, ensuring that SAT solvers are given sufficient working time rather than being cut short by the GPU's early completion. By starting with a low conflict limit and raising it gradually, the CPU resolves easy pairs before investing effort in harder ones. Meanwhile, the minimum processed pair count guarantees that SAT-based progress is never interrupted by the GPU's early completion.

3.3.2 Adaptive AIG Resynthesis. Adaptive AIG resynthesis is a crucial heuristic triggered when verification gets stuck, specifically after failing to identify new equivalent pairs for three consecutive passes. This impasse indicates the current AIG structure is too complex for the solver algorithms. To overcome this, the computationally intensive procedure applies global logic resynthesis to rewrite the entire AIG, creating a new structural representation that is potentially easier to solve and can enable further progress.

The main benefit of this strategy is that it provides a new path forward for the verification engine. By fundamentally changing the AIG's topology, it presents a functionally identical but structurally distinct problem. This transformation can resolve complex structural patterns that previously limited the solvers' effectiveness, allowing the standard iterative verification cycle to resume finding new pairs. This adaptive intervention is therefore fundamental to achieving convergence on challenging problems, providing a method to systematically alter the problem itself when the current approach is not productive.

3.4 Overall Flow of Heterogeneous CEC Engine

The preceding subsections introduced the individual components of the heterogeneous engine. We now analyze how they interact as a closed-loop system and argue that the engine is both sound and complete. Soundness means that the engine never produces an incorrect equivalence or non-equivalence result. Completeness means that it terminates with a definitive result for every candidate pair.

3.4.1 Detailed Flow. The overall workflow of our heterogeneous CEC engine, illustrated in Figure 4, is a multi-phase process designed to efficiently solve the CEC problem. The flow begins with two GPU-parallel processing stages, PO checking phase *P* and global function checking phase *G*, before entering the main iterative local function checking phase *L*. The details of these initial phases are not the focus of this paper and can be found in [15]. These initial phases aim to quickly resolve the easiest equivalences and simplify the miter before engaging the more computationally intensive heterogeneous engine.

If the miter is not fully proven after these initial phases, the engine enters the core iterative loop of Phase *L*. This phase deploys the heterogeneous engine, which alternates between two operational modes: cooperative mode (Mode I; Section 3.2.1) and competitive

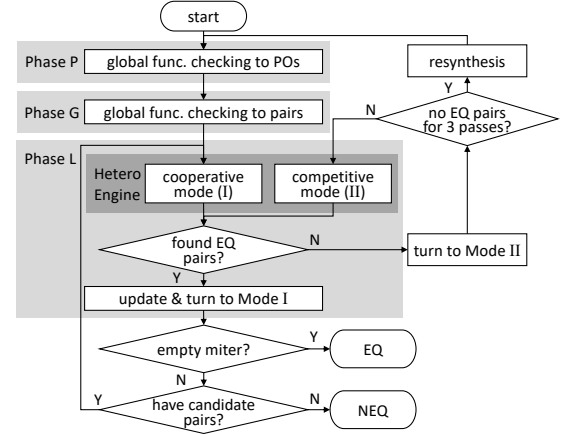


Figure 4: Flow of CEC with heterogeneous engine.

mode (Mode II; Section 3.2.2). The logic for mode-switching and stall handling is central to Phase *L*.

After each run of the heterogeneous engine, the flow checks for new equivalent pairs: if found, the miter is updated and the engine returns to cooperative mode; otherwise it switches to competitive mode. Three consecutive passes without new equivalences trigger adaptive AIG resynthesis, which rewrites the AIG and restarts from Phase *P*. Termination follows: an empty miter proves EQ, while an unreducible non-empty miter with no candidate pairs declares NEQ.

3.4.2 Soundness. The engine accepts equivalence proofs from two sources. A GPU-based proof exhaustively simulates the local function over all $2^{|C|}$ input patterns of a common cut *C*. This constitutes a complete truth-table comparison and is therefore sound. A CPU-based proof is established when a SAT solver returns UNSAT on the corresponding CNF instance, providing a formal certificate of equivalence.

Simulation on the local function can only prove equivalence, not disprove it, because a satisfying assignment of the local miter may fall in the satisfiability don't care space of the global function. Therefore, the GPU simulation engine only accepts pairs that are proven equivalent. Any pair for which simulation finds a potential counterexample is forwarded to the CPU SAT engine for formal verification. The SAT solver can both confirm and refute equivalence, so all non-equivalence conclusions are backed by formal proofs. This asymmetry is enforced in both cooperative and competitive modes, as described in Section 3.2.2.

Adaptive AIG resynthesis applies functionality-preserving transformations and therefore cannot alter the equivalence relation between any pair of nodes. Results obtained before and after resynthesis remain mutually consistent, and merging GPU and CPU results introduces no unsoundness.

3.4.3 Completeness. AIG resynthesis and miter updates never increase the number of AIG nodes, so the node count is finite and non-increasing across iterations. Since the number of candidate pairs is bounded by the number of nodes, the candidate pool remains finite at every iteration. Within the iterative sieving process, the conflict limit and the minimum processed pair count both rise across consecutive stalled passes. The growing conflict limit ensures

Table 1: Statistics of benchmarks

Benchmarks	#PIs	#POs	#Nodes	Levels
hyp_7xd	32768	16384	45881216	24801
multiplier_10xd	131072	131072	52600832	274
sqrt_10xd	131072	65536	44978176	5058
square_10xd	65536	131072	33442816	250
voter_10xd	1025024	1024	21862400	70
ac97_ctrl_10xd	2307072	2299904	22685696	12
vga_lcd_5xd	549248	549856	6337536	24
leon2	149507	149577	1470132	58
leon3	185134	185196	1506859	59
netcard	97899	97888	1231949	40

that the SAT solver eventually has a sufficient budget to resolve any finite CNF instance [22], while the growing minimum processed pair count ensures the solver is not terminated prematurely. Every candidate pair is therefore resolved after finitely many passes, without relying on any external verification tool.

4 Experimental Results

The heterogeneous CEC engine is implemented on top of the GPU-based logic synthesis and verification tool CULS¹. All the experiments are performed on a Linux machine with Intel Xeon Platinum 8369B CPU and NVIDIA A100 SXM4 80 GB GPU. For all experiments, our heterogeneous engine utilizes 16 SAT solvers in parallel. To ensure a fair comparison, our engine is built upon the same SAT solver as that used in the ABC &cec checker. GPU-parallel resubstitution [25] is employed in our implementation as AIG resynthesis.

Our evaluation uses benchmarks from the EPFL Combinational Benchmark Suite [2] and the IWLS 2005 Benchmarks [1], covering both arithmetic and control designs. Following prior work on parallel logic synthesis and verification [13, 14, 25], we scale the benchmarks by repeatedly applying the ABC command `double`. For each benchmark, we construct a CEC instance by pairing the original circuit with an optimized version produced by the ABC `resyn2` script, which performs several passes of balancing, rewriting, and refactoring. This setup is identical to the one used in [15]. Of the 9 cases considered there, we use the 7 whose equivalence is not directly established by global simulation on POs. The remaining two cases, *log2_10xd* and *sin_10xd*, are resolved during global simulation on POs and therefore fall outside the scope of this work. We additionally include *leon2*, *leon3*, and *netcard* to broaden the coverage.

Table 1 summarizes the benchmark and miter statistics. “_nxd” means that the benchmark is generated by enlarging the original one using ABC `double` n times. The PI and PO counts refer to the original/optimized circuits, while the node and level counts refer to the miters (excluding the output XOR gates).

4.1 Comparison with ABC and simulation-based CEC

We evaluate our method against three baselines: ABC &cec, a single-threaded yet high-performance checker based on SAT sweeping; Cadence Conformal LEC (version 19.20), a commercial tool that supports parallel checking with up to 16 CPU threads; and the

simulation-based CEC engine from DAC’25 [15]. No open-source parallel CEC tool is publicly available to our knowledge, making Conformal the only accessible multi-threaded baseline. Our heterogeneous CEC engine runs 16 SAT solvers concurrently, matching the thread budget allocated to Conformal. The simulation-based engine was configured with the default parameters specified in its original publication. In their work, the authors employed a two-stage process using a simulation-based engine (runtime in subcolumn “GPU (s)”) followed by the ABC checker (runtime in subcolumn “ABC (s)”). The middle subcolumn “Reduced” shows the percentage of miter size reduction they achieved with the simulation engine.

Table 2 reveals a significant performance advantage for our heterogeneous engine. It achieved the fastest execution time on 8 of the 10 benchmarks. On the remaining two, our performance was highly competitive with the DAC’25 tool and still substantially better than ABC’s traditional approach. The most striking result was on the *voter_10xd* benchmark, where ABC took 54422 seconds and the DAC’25 tool took 23980 seconds. In contrast, our method finished in only 288.16 seconds, achieving a remarkable $188.86\times$ speedup over ABC and $83.22\times$ over the DAC’25 baseline. This improvement is attributed to our engine’s ability to dynamically switch verification strategies.

Overall, the results show that our engine is $12.29\times$ faster than ABC, $47.81\times$ faster than Conformal, and $3.19\times$ faster than DAC’25. This consistent performance gain validates our core hypothesis: a heterogeneous approach that intelligently combines verification techniques can overcome the limitations of any single method, providing a more scalable and practical solution for modern hardware verification.

The benchmarks above are equivalent pairs; NEQ instances terminate at the first differing primary output and are typically resolved faster [8]. On the ICCAD-2015 EC/ECO suite [9], all eleven non-equivalent instances are correctly reported as NEQ, matching both ABC &cec and Conformal.

4.2 Ablation Study

To evaluate our proposed techniques, we conducted an ablation study with four engine configurations: a basic version without resynthesis or iterative sieving, one with resynthesis (w/ rs), one with iterative sieving (w/ sv), and a final version with both. The results are shown in Table 3. Against DAC’25, our basic engine showed a $1.53\times$ speedup. Resynthesis boosted the geomean speedup to $2.29\times$ and iterative sieving to $2.40\times$, both being particularly effective on difficult benchmarks like *voter_10xd*.

The most compelling result was the synergistic effect of combining both, with the full configuration achieving a $3.19\times$ geomean speedup. This synergy was especially clear on *voter_10xd*, where the combined approach was over $7.0\times$ faster than resynthesis alone and $7.9\times$ faster than sieving alone. Excluding this benchmark, the geomean speedups over the remaining nine cases are $1.72\times$ (Basic), $1.91\times$ (w/ rs), $2.04\times$ (w/ sv), and $2.22\times$ (w/ rs and sv), confirming that the gains are broadly distributed rather than driven by a single outlier. The combined configuration does not dominate on every benchmark, as the two techniques may interact unfavorably on certain instances, but it consistently achieves the best geomean.

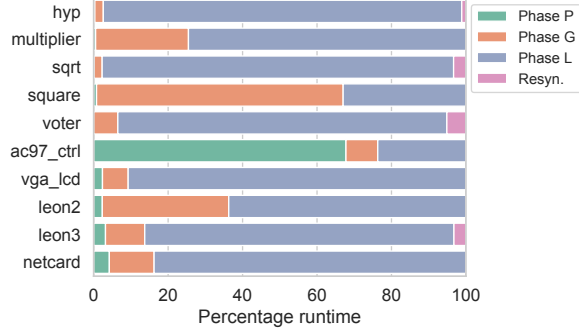
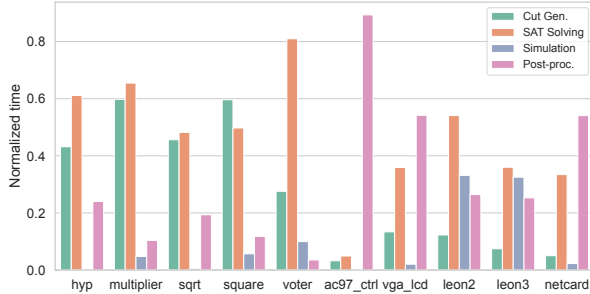
¹<https://github.com/cuhk-eda/CULS>. We obtained the code of simulation-based CEC from the authors of [15].

Table 2: Runtime comparison with ABC and DAC'25

Benchmarks	ABC & cec Runtime (s)	Cfm (16 CPUs) Runtime (s)	DAC'25 (1 GPU+ABC)				Our Runtime (s)	Speedup vs.		
			GPU (s)	Reduced	ABC (s)	Total (s)		ABC	Cfm	DAC'25
hyp_7xd	8066.09	336982	2753.83	40.2	310.07	3063.90	2573.51	3.13×	130.94×	1.19×
multiplier_10xd	2680.06	2795	85.04	100.0	-	85.04	85.52	31.34×	32.68×	0.99×
sqrt_10xd	21533.78	24790	36.37	0.7	19809.50	19845.87	1515.34	14.21×	16.36×	13.10×
square_10xd	1096.47	2331	39.88	100.0	-	39.88	41.76	26.26×	55.82×	0.95×
voter_10xd	54422.25	1793	28.93	43.5	23951.67	23980.60	288.16	188.86×	6.22×	83.22×
ac97_ctrl_10xd	250.27	1258	22.82	98.9	3.24	26.06	25.88	9.67×	48.61×	1.01×
vga_lcd_5xd	111.18	707	7.58	20.1	35.26	42.84	7.17	15.51×	98.61×	5.97×
leon2	21.78	1107	5.71	6.6	10.77	16.48	6.73	3.24×	164.49×	2.45×
leon3	27.14	651	6.97	13.3	15.01	21.98	6.64	4.09×	98.04×	3.31×
netcard	34.78	202	2.39	23.4	8.79	11.18	6.08	5.72×	33.22×	1.84×
Geomean								12.29×	47.81×	3.19×

Table 3: Ablation study of resynthesis and iterative sieving

Benchmarks	DAC'25 Runtime (s)	Our Runtime (s)			
		Basic	w/ rs	w/ sv	w/ rs and sv
hyp_7xd	3063.90	2423.61	2688.76	2702.41	2573.51
multiplier_10xd	85.04	82.69	82.38	83.50	85.52
sqrt_10xd	19845.87	4067.91	1029.02	4199.03	1515.34
square_10xd	39.88	42.39	41.57	41.42	41.76
voter_10xd	23980.60	45221.84	2016.93	2274.66	288.16
ac97_ctrl_10xd	26.06	40.09	42.16	25.78	25.88
vga_lcd_5xd	42.84	29.99	28.67	8.85	7.17
leon2	16.48	6.04	5.38	5.00	6.73
leon3	21.98	6.41	7.48	5.47	6.64
netcard	11.18	4.49	6.11	6.21	6.08
Geomean speedup vs. DAC'25		1.53×	2.29×	2.40×	3.19×

**Figure 5: CEC runtime breakdown by phases.****Figure 6: Normalized time decomposition of Phase L.**

4.3 Runtime Breakdown and Analysis

We further profile the runtime of our heterogeneous CEC engine on the same enlarged benchmarks as above (the “_nxd” suffixes

are omitted in figures for brevity). Figure 5 shows the percentage runtime breakdown by different phases, and Figure 6 shows the normalized component times within Phase *L*.

From Figure 5, Phase *L* dominates the total runtime for most benchmarks, confirming that local equivalence checking is the primary bottleneck of the CEC flow and motivating the design of our engine. Figure 6 further decomposes Phase *L* into four components: GPU-based cut generation, CPU-based SAT solving, GPU-based simulation checking, and post-processing (including status merging and miter reduction). Since the GPU and CPU operate in parallel in both modes, the component fractions can sum to greater than 1.0, indicating effective CPU-GPU overlap. For most benchmarks, the total fractions are well above 1.0, demonstrating that our engine keeps both CPU and GPU resources highly utilized throughout Phase *L*. For *leon2* and *leon3*, noticeable simulation fractions show that competitive mode contributes meaningfully when cooperative mode SAT encounters harder instances, further validating the benefit of the dual-mode design.

5 Conclusion

This paper presents, to the best of our knowledge, the first heterogeneous CEC engine that tightly integrates exhaustive simulation, AIG resynthesis, and SAT solving into a feedback-driven loop, where each strategy continuously simplifies the problem structure for the others. This stands in contrast to conventional approaches that rely on a single dominant reasoning mode and inevitably hit structural bottlenecks on certain circuit families. Our engine achieves average speedups of 12.29× over ABC, 47.81× over the multi-threaded commercial tool Conformal, and 3.19× over the state-of-the-art simulation-based CEC tool. On the hardest cases, verification time drops from hours to minutes: *voter_10xd* is completed in under 5 minutes versus 15.1 hours (ABC) and 6.7 hours (DAC'25). Ablation studies confirm these gains arise from the interplay among heterogeneous strategies, not any single component in isolation.

Acknowledgments

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Reference Number: T46-415/25-R).

References

- [1] Christoph Albrecht. 2005. IWLS 2005 Benchmarks. <https://iwls.org/iwls2005/benchmarks.html>.
- [2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Proc. IWLS*.
- [3] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleys, and Florian Pollitt. 2024. CaDiCaL 2.0. In *International Conference on Computer Aided Verification (CAV)*. Springer, 133–152.
- [4] Daniel Brand. 1993. Verification of large synthesized designs. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. 534–537.
- [5] Robert Brayton and Alan Mishchenko. 2006. Scalable logic synthesis using a simple circuit structure. In *Proc. IWLS*, Vol. 6. 15–22.
- [6] Robert Brayton and Alan Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification (CAV)*. Springer, 24–40.
- [7] Randal E Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.* C-35, 8 (1986), 677–691.
- [8] Debapriya Chatterjee and Valeria Bertacco. 2010. EQUIPE: Parallel equivalence checking with GP-GPUs. In *2010 IEEE International Conference on Computer Design*. 486–493.
- [9] Chih-Jen Hsu, Chi-An Wu, Wei-Hsun Lin, and Kei-Yong Khoo. 2015. ICCAD-2015 CAD Contest in Large-Scale Equivalence Checking and Function Correction and Benchmark Suite. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 916–920.
- [10] Tsung-Wei Huang, Dian-Lun Lin, Chun-Xun Lin, and Yibo Lin. 2021. Taskflow: A lightweight parallel and heterogeneous task graph computing system. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (2021), 1303–1320.
- [11] Andreas Kuehlmann and Florian Krohm. 1997. Equivalence checking using cuts and heaps. In *Proceedings of the 34th annual Design Automation Conference*. 263–268.
- [12] A. Kuehlmann, V. Paruthi, F. Krohm, and M.K. Ganai. 2002. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21, 12 (2002), 1377–1394.
- [13] Tianji Liu, Yang Sun, Lei Chen, Xing Li, Mingxuan Yuan, and Evangeline FY Young. 2025. A unified parallel framework for LUT mapping and logic optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 1 (2025), 214–226.
- [14] Tianji Liu and Evangeline FY Young. 2023. Rethinking AIG resynthesis in parallel. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
- [15] Tianji Liu and Evangeline FY Young. 2025. Simulation-based Parallel Sweeping: A New Perspective on Combinational Equivalence Checking. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7.
- [16] Qiang Ma and Evangeline F. Y. Young. 2010. Multivoltage Floorplan Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 4 (2010), 607–617.
- [17] Wai-Kei Mak and Evangeline F. Y. Young. 2002. Temporal Logic Replication for Dynamically Reconfigurable FPGA Partitioning. In *Proceedings of the International Symposium on Physical Design (ISPD)*. 190–195.
- [18] Y. Matsunaga. 1996. An efficient equivalence checker for combinational circuits. In *33rd Design Automation Conference Proceedings*. 629–634. doi:10.1109/DAC.1996.545651
- [19] Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, and Niklas Een. 2006. Improvements to combinational equivalence checking. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design (ICCAD)*. 836–843.
- [20] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, and Robert K Brayton. 2005. *FRAIGs: A unifying representation for logic synthesis and verification*. Technical Report. ERL Technical Report.
- [21] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*. 530–535.
- [22] Knot Pipatsrisawat and Adnan Darwiche. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence* 175, 2 (2011), 512–525. doi:10.1016/j.artint.2010.10.002
- [23] Vinicius N Possani, Alan Mishchenko, Renato P Ribas, and Andre I Reis. 2019. Parallel combinational equivalence checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 3081–3092.
- [24] Heinz Riener, Siang-Yun Lee, Alan Mishchenko, and Giovanni De Micheli. 2022. Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 395–402.
- [25] Yang Sun, Tianji Liu, Martin DF Wong, and Evangeline FY Young. 2024. Massively Parallel AIG Resubstitution. In *61st ACM/IEEE Design Automation Conference (DAC)*. 1–6.
- [26] Grigori S Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*. Springer, 466–483.
- [27] Linfu Xiao, Sharad Sinha, Jiang Xu, and Evangeline F. Y. Young. 2010. Fixed-Outline Thermal-Aware 3D Floorplanning. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. 561–567.