Shared Logic Unleashed: Multiple-Node Boolean Optimization for Next-Gen Synthesis

Alessandro Tempia Calvino
Synopsys
Sunnyvale, CA, USA

Anubhaw Xess
Synopsys
Bengaluru, India

Anika Prasad
Synopsys
Sunnyvale, CA, USA

Jacob Minz
Synopsys
Bengaluru, India

Eleonora Testa
Walter Lau Neto
Synopsys
Sunnyvale, CA, USA

Alan Mishchenko
UC Berkeley
Berkeley, CA, USA

Giovanni De Micheli
EPFL
Lausanne, Switzerland

Patrick Vuillod
Synopsys
Montbonnot, France

Luca Amaru
Synopsys
Sunnyvale, CA, USA

Abstract

We present a high-effort Boolean optimization framework that restructures multiple nodes in combinational logic simultaneously, enabling area reductions beyond the reach of conventional single-output transformations. Our method generalizes Boolean resubstitution on both nodes and edges to operate across shared logic while leveraging don't-care conditions. Unlike traditional techniques, our approach supports coordinated removal and resynthesis of shared logic regions and expands the optimization scope to multiple-output windows. When deployed in an industrial standard-cell design flow, our framework achieves up to 6.79% area reduction (1.7% average) and up to 7.85% switching power reduction (1.8% average) post-synthesis on 87 designs. Additionally, it establishes 19 new best results in the EPFL synthesis competition on LUT networks, breaking a plateau in traditional synthesis techniques.

ACM Reference Format:
Alessandro Tempia Calvino, Anubhaw Xess, Anika Prasad, Jacob Minz, Eleonora Testa, Walter Lau Neto, Alan Mishchenko, Giovanni De Micheli, Patrick Vuillod, and Luca Amaru. 2026. Shared Logic Unleashed: Multiple-Node Boolean Optimization for Next-Gen Synthesis. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804422

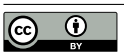
1 Introduction

The continued scaling of *Complementary Metal-Oxide-Semiconductor* (CMOS) technology has historically enabled significant advancements in computational performance, energy efficiency, and integration density. However, as physical scaling slows and design complexity increases, the role of *Electronic Design Automation* (EDA) becomes increasingly critical. Further improvements in *Power, Performance, and Area* (PPA) require sustained innovation in synthesis algorithms and optimization techniques, which must evolve to meet the demands of modern digital systems. Improving PPA, particularly area, is especially valuable in today's landscape, where the cost of chip design continues to rise [1].

Logic synthesis is a key phase of EDA tools aimed at improving the implementation cost of integrated circuits, especially in terms of PPA. In this context, optimization takes place at two abstraction levels: technology independent and technology dependent. Technology-independent synthesis occurs prior to technology mapping, relying on simplified models to guide optimization [2]. Technology-dependent synthesis follows technology mapping, leveraging more accurate and library-specific estimations [3].

We focus on Boolean techniques for area optimization of both technology-independent and technology-dependent combinational networks. Traditional area optimization algorithms, such as Boolean resubstitution [2, 4, 5], rewriting [6], refactoring, and balancing, typically operate on single-output cones of logic. Only a few methods in logic synthesis are inherently designed to simultaneously optimize across multiple nodes. Among these, the most adopted in academic and industrial flows is cube and kernel *extraction*, which identifies and extracts common logic from multiple nodes [7]. Recently, *And-Inverter Graph* (AIG)-rewriting was extended to multiple-output windows [8]. Other efforts explored simultaneous optimization of compatible gates [9], remapping through Boolean matching on multiple-output Boolean functions [10], and exact synthesis on multiple-output sub-circuits [11]. Nevertheless, multiple-node optimization remains a challenging and open problem. In general, simultaneous optimization on multiple nodes sharing common logic exposes additional opportunities for logic restructuring. This is because the *Maximum Fanout-Free Cones* (MFFCs), which represent the potential saving of a transformation, expand into *Maximum Fanout-Free Windows* (MFFWs) when considered jointly, and these windows are typically larger. Additionally, coordinated optimization enables the restructuring of shared logic.

In this paper, we introduce a high-effort Boolean optimization framework that simultaneously restructures multiple nodes in combinational logic, enabling area reductions beyond the reach of conventional single-output transformations. In particular, our method generalizes Boolean resubstitution to operate across shared logic regions, coordinating transformations on both nodes and edges while



DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804422

leveraging controllability and observability don't-care conditions. Our work expands conventional single-output transformations by introducing a framework that has shared logic as a key optimization objective, enabling coordinated removal and resynthesis of common substructures. A scalable candidate selection strategy is proposed, which identifies shared regions while maintaining manageable computational complexity. Furthermore, we expand the optimization scope from MFFCs to multiple-output windows, unlocking new opportunities for logic untangling.

To demonstrate the effectiveness of our method, we integrate our multiple-node resubstitution into an optimization flow that interleaves logic optimization with technology mapping, targeting both standard-cell and FPGA designs. We implemented this algorithm within an industrial synthesis tool for standard-cell flows and evaluated our experiments post-synthesis (after technology mapping and delay-driven optimization) on 87 designs, including 22 OpenCore designs [12] and 65 industrial designs. On OpenCore designs, our method achieves up to 6.07% area reduction (2.11% average) and up to 7.09% switching power reduction (2.44% average), while on industrial designs it exhibits up to 6.79% area saving (1.50% average) and up to 7.85% switching power reduction (1.60% average). Across all 87 designs, average reductions are 1.7% in area and 1.8% in switching power. These improvements are obtained within 6% runtime increase and maintain comparable or better WNS and TNS. Furthermore, leveraging the proposed optimization flow, we achieve several new best results in the EPFL synthesis competition on *LookUp Table* (LUT) networks [13]. Notably, we find 9 new best results for area and 10 new best results for levels.

2 Background

2.1 Notations and Definitions

A *Boolean function* is a mapping from an n-dimensional Boolean space into a 1-dimensional one: $\{0,1\}^n \rightarrow \{0,1\}$. The points were the function is not specified are called *don't-care* conditions. Don't care conditions arise when input combinations never occur, known as *controllability conditions* (CDCs), or when the output value does not matter for some input combinations, known as *observability conditions* (ODCs). Don't cares may emerge when a Boolean function is within a larger system, such as Boolean networks, and are crucial in logic synthesis because they offer additional flexibility when performing Boolean simplification and optimization.

A *Boolean network* is modeled as a *Directed Acyclic Graph* (DAG) where nodes represent Boolean functions. The sources of the graph are the *Primary Inputs* (PIs) of the network, the sinks are the *Primary Outputs* (POs). For any node n, the *fanins* (FI) of n is a set of nodes driving n, i.e. nodes that have an outgoing edge towards n. Similarly, the *fanouts* (FO) of n is a set of nodes that are driven by node n, i.e., nodes that have an incoming edge from n. If there is a path from node a to node b, then a is in the *Transitive Fanin* (TFI) of b, and b is said to be in the *Transitive Fanout* (TFO) of a. The TFI of b includes node b and the nodes in its TFI, including the PIs. The TFO of b includes b and all the nodes in its TFO including the POs.

The *Maximum Fanout-Free Cone* (MFFC) of a node n is the largest subset of the transitive fanin of n such that every path from the nodes in the MFFC to the POs passes through n. Informally, the MFFC of a node contains the node itself and all the logic exclusively used by the node. Hence, when a node is removed (or substituted) the logic in the MFFC can also be removed. The *Maximum Fanout-Free Window* (MFFW) over a set of nodes S is a generalization of the MFFC such that every path from the nodes in the MFFW to the POs passes through nodes in S [14].

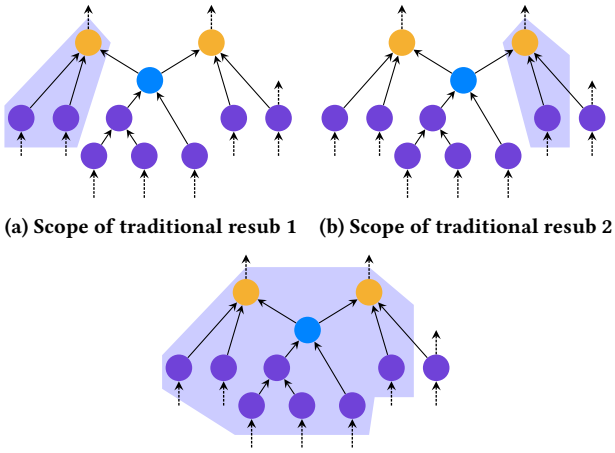
2.2 Boolean resubstitution

Resubstitution, abbreviated as *resub*, is a transformation technique that (re)expresses the functionality of a node n using other nodes, called *divisors*, that already exist in the network. The transformation is accepted if the new implementation of n improves a target metric (e.g., number of gates), compared to its current implementation based on its immediate fanins. This approach generalizes to *k-resubstitution*, which adds k new nodes while removing at least $k+1$ nodes from the MFFC of n. The added nodes derive their functionality from a library of primitives used for resubstitution. In the *And-Inverter Graph* (AIG) implementation, these primitives are 2-input ANDs with optional input/output inversion [4]. To enhance scalability, *windowing* is commonly employed to restrict the scope of resubstitution to a localized sub-network built around the target node. Windows also facilitate the collection of divisors and the computation of don't-care conditions.

In the literature, multiple flavours of Boolean resubstitution have been proposed. In this paper, we classify them into two main types: *node resubstitution*, where the node itself is reconstructed, and *edge resubstitution*, where the node's fanins are selectively disconnected or replaced.

- *Node resub*: is the classical approach, commonly adopted to restructure AIGs or mapped networks [4, 5, 15–17]. In this type, the target node and its MFFC are replaced with a new implementation built from divisor nodes. In *window-based resub*, exhaustive simulation is performed within a window to derive local functions of the target node and its divisors in terms of the window inputs. Then, local functions are used to reconstruct the target node from its divisors. In *simulation-based resub* [15], *Automatic Test Pattern Generation* (ATPG) is employed to compute an approximation of the functionality of each node in the network with respect to the PIs. Then, the approximated functions are used to reconstruct the target node from its divisors, and functional equivalence is verified using SAT before committing.
- *Edge resub*: the simplest form of edge resub is *redundancy removal* [18, 19], which attempts to replace fanin edges with constant values under don't-care conditions. More advanced variants have been applied to *LookUp Table* (LUT) networks [20]. Here, each immediate fanin of a target node n is evaluated for potential removal or replacement with a divisor. During this process, the functionality of node n may be adjusted to accommodate the change. This method often relies on interpolation to determine whether a valid substitution exists and compute the updated logic function.

In this work, we propose a multiple-node optimization framework built upon these two flavors of resubstitution, with the objective to remove portions of logic shared by multiple nodes, unlocking further area savings as compared to the standard resub variants.



(a) Scope of traditional resub 1 (b) Scope of traditional resub 2

(c) Scope of multiple-node resub

Figure 1: Example of the scope of the resynthesis problem in traditional resubstitution (a) and (b) operating on one node at a time.Conversely, multiple-node optimization (c) includes the shared logic.

3 Multiple-Node Optimization

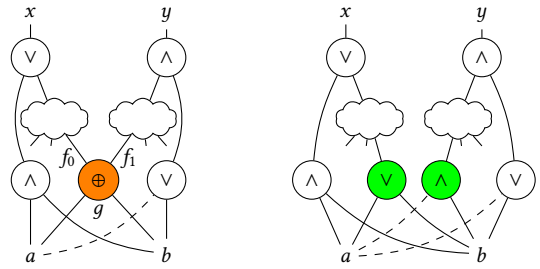
This section motivates the need for multiple-node optimization and introduces two algorithms designed to coordinate transformations across nodes, enabling the removal or resynthesis of shared logic. Since don't-care conditions are crucial for exploiting functional flexibility and unlocking additional optimization opportunities, our approach builds upon advanced variants of Boolean resubstitution.

Traditional resubstitution techniques operate on a single node at a time, resynthesizing its MFFC. However, after several rounds of incremental optimization, the structure of Boolean networks tends to crystallize: MFFCs become too small to enable further improvements through single-node transformations, as each move must reduce the number of nodes or literals. Figure 1 illustrates the limitations of single-output optimization and the benefits of multiple-node optimization within a sub-circuit. In Figures 1a and 1b, a target node (highlighted in yellow) along with its MFFC (in purple) is optimized. In both cases, the shared logic, including the node in blue and its MFFC, remains untouched and cannot be removed or resynthesized. Conversely, Figure 1c shows the broader optimization scope of multiple-node resubstitution. In particular, this offers two key advantages: (i) it unlocks additional resynthesis opportunities by costing the removal of shared logic during each resubstitution sub-problem (at the yellow nodes), and (ii) it enables coordinated restructuring or elimination of the shared logic through joint optimization across the target nodes (in yellow).

Furthermore, multiple-node resubstitution can exploit additional observability don't-care (ODC) conditions to simplify shared logic. For a node g with multiple fanouts, its ODC is computed as:

$$\mathrm{ODC}(g) = \bigcap_{f \in \mathrm{FO}(g)} \left(\mathrm{ODC}(f) \cup \overline{\frac{\partial f}{\partial g}} \right), \tag{1}$$

where the intersection combines the ODCs of all fanouts with an additional term capturing observability flexibility introduced



(a) (b)

Figure 2: Optimization of shared logic with multiple-node resub leveraging orthogonal ODCs.

by each fanout. This formulation shows that the ODC set must hold for all fanouts simultaneously in order to be used. In contrast, multiple-node optimization relaxes this constraint by treating fanouts individually:

$$\mathrm{ODC}(f_i) \supseteq \bigcap_{f \in \mathrm{FO}(g)} \mathrm{ODC}(f), \tag{2}$$

thereby potentially expanding the ODC space for logic restructuring. Additionally, the term $\overline{\frac{\partial f}{\partial g}}$ in Equation 1 is naturally addressed during resynthesis of f since f and g are resynthesized together (g is in the MFFW of f for node resub). Figure 2 illustrates this concept with an example. In Figure 2a, a XOR node g (in orange) drives two logic clouds. Dashed edges represent negation. The ODC of g is empty because the ODCs coming from its fanouts, respectively ($ODC(f_0) = ab$ and $ODC(f_1) = a\bar{b}$), are orthogonal, resulting in an empty intersection. Consequently, traditional resubstitution cannot exploit ODCs to reduce area at node g, and node g cannot be resynthesized within its fanout as it lies outside their MFFC. In contrast, multiple-node resubstitution treats each fanout individually, using separate ODC sets. Figure 2b shows the optimized result, where the XOR is replaced by two simpler nodes (AND and OR), typically reducing area compared to the original XOR.

In this work, we propose two Boolean resubstitution variants, node-based and edge-based, formulated to operate on multiple nodes. The node-based one resynthesizes multiple nodes without reusing their shared logic, which enables removal and resynthesis of the shared region. The edges-based one replaces fanin connections to the shared region with alternative nodes or constants to disconnect it.

3.1 Multiple-Node Resubstitution Formulation

A fully general formulation of multiple-output optimization is computationally prohibitive, as it may need to explore up to $|N|^m$ resubstitution problems, where $|N|$ indicates the number of nodes in the network and m is the maximum number of nodes optimized simultaneously. In practice, only nodes that are closely related, meaning that they share portions of the TFI, are suitable candidates for multiple-node optimization. To address this, we propose a *multiple-node approach* to guide the candidate selection rather than exhaustively exploring all the combinations. Our strategy focuses on a small set of nodes (typically two or three) that share

common logic. That is because considering too many nodes would (i) drastically increase the complexity and (ii) reduce the likelihood of a successful optimization, as the resubstitution must be feasible for every candidate. Therefore, instead of starting from nodes, we invert the process: we first detect shared logic and then extract the roots for optimization from the fanout.

Candidate extraction begins by identifying a node g with multiple fanouts. The fanout count serves as a selection filter and is computed across inverters and buffers in the TFO of g. Inverters and buffers are excluded as target nodes or divisors since they provide no meaningful functional contribution to the network. Candidates are then selected by traversing the TFO and identifying the first nodes with multiple fanins. As don't-care-based Boolean optimization is only possible when *reconvergent paths* are present, our method applies an additional filter to ensure reconvergence exists in the TFI or TFO of the candidate nodes (or in both).

Algorithm 1 outlines the high-level procedure for multiple-node resubstitution. For each node g in the network, the algorithm first attempts a single-output resubstitution. If unsuccessful, it proceeds with multiple-node resubstitution. Nodes are processed in reverse topological order to ensure that their TFO and corresponding MFFCs have already been optimized. Candidate nodes are collected by traversing the TFO of g as previously described. If the number of collected candidate nodes is fewer than two or exceeds a given limit m, the algorithm skips to the next node. Otherwise, a window is constructed around node g to include nodes in T, their *Maximum Fanout-Free Window* (MFFW), and surrounding divisor nodes, prioritizing the ones on reconvergent paths. This window is further expanded by a few levels in both the TFI and TFO, as defined by the parameter *dcl*, to enable controllability and observability don't care computation. An additional optional filter ensures that each node in T has a reconvergence path either in the its TFI or TFO within the window. The combined MFFW of nodes in T represents the maximum potential gain of the multiple-node resubstitution problem (as in Figure 1c), and includes node g and its MFFC. Next, for each node in T, a resubstitution candidate g' is computed from local divisors, leveraging controllability and observability don't cares. This step, detailed in the next subsections, aims to remove node g. The potential gain serves as a budget limit for the complexity of the resubstitution candidate. Additionally, the effort parameter k saturates the search to k-resub. If the computation succeeds, the candidate replacement is committed and appended to the replacement vector R. Then, the gain is updated to reflect the change, and the algorithm proceeds to the next fanout node in T. If the resubstitution computation fails or the gain is not positive, the algorithm reverts the changes and moves to the next candidate.

Our algorithm unifies the area cost modeling for both technology-independent and technology-dependent synthesis. The cost of each transformation and its associated gain is evaluated using the following equation:

$$gain = \sum_{g \in MFFW(T)} (FFLC(g) - 1) - \sum_{g' \in T'} (FFLC(g') - 1), \quad (3)$$

where T is the set of candidate nodes, T' is the set of replacement nodes produced by resubstitution, and $FFLC(g)$ denotes the *Factored Form Literal Count* (FFLC) of a node g, i.e., the number of literals in the factored form representation of its Boolean function [2, 21]. The first term measures the potential gain from removing logic in the MFFW, quantified by the number of two-input AND/OR operations in the factored forms, while the second term captures the cost of the new implementation using the same metric. In the context of AIGs, this formulation effectively counts the number of nodes. Furthermore, inverters and buffers are excluded from the costing, as they do not contribute to functional complexity and can be re-optimized at a later stage during technology mapping.

In the next subsections, we detail the two types of resubstitution employed by our framework: *node-based resubstitution* and *edge-based resubstitution*.

Algorithm 1: Multiple-Node Resubstitution

Input: Network N, Number of max fanout m, Resub effort k, DC levels *dcl*, Reconvergence filter R

```
foreach node g ∈ N in rev topological order do
    g' ← try_single_output_resub(N, g, k)
    if g' found then
        continue                          ▷ Resub successful
    T ← immediate_fanouts_roots(N, g)
    if |T| < 2 or |T| > m then
        continue
    R ← ∅
    W ← compute_window(g, T, dcl)
    if R and dont_reconverge(W, T) then
        continue
    gain ← mffw_gain(W, T)
    foreach node t ∈ T in topological order do
        D ← compute_divisors(W, t, g)
        g' ← resub(t, D, W, g, gain, k)
        if g' not found then
            break                         ▷ Resub failed
        append(R, {t, g'})
        gain ← update_gain(gain, {t, g'})
    if |R| ≠ |T| or gain ≤ 0 then
        foreach node pair {t, g'} ∈ R in rev order do
            undo_resub(N, t, g')
```

3.2 Node-Based Multiple-Node Resubstitution

We implement our node-based resubstitution algorithm by extending simulation-guided resubstitution, which supports larger windows compared to traditional window-based resubstitution. The objective of this approach is to rebuild the nodes in T and their MFFW without reusing node g along with its MFFC. For a node $t \in T$, we compute a set of divisors while explicitly excluding its MFFC, the shared node g, and the MFFC of g. This ensures that the reconstruction avoids reusing the shared logic. The new implementation of node t is computed leveraging the state-of-the-art AIG-based resynthesis algorithm [22], which incrementally adds AND nodes in a bottom-up fashion until either a solution is found or the limits on k or gain are reached. Because this method leverages

ATPG-based simulations, controllability don't cares require no additional computation, as unreachable patterns never appear in the simulation. In contrast, observability don't cares must be computed by traversing the node's TFO for a few levels (*dcl* in Algorithm 1). In practice, less than 5 levels is a good target to balance quality-of-results and runtime. These don't cares are stored as a simulation signature, which, combined with the functional signature, enables more effective resynthesis. The correctness of each transformation is verified using circuit-based SAT solving at the window output, similarly to SAT sweeping [23]. To preserve scalability, both the number of SAT conflicts and variables are constrained by predefined limits. Upon successful computation of a resubstitution, the replacement is added into the network and subsequently becomes available as a potential divisor for the remaining nodes in T.

3.3 Edge-Based Multiple-Node Resubstitution

We implement our edge-based resubstitution framework by extending redundancy removal and interpolation-based resubstitution techniques. The transformation objective is to attempt to disconnect node g from every node in T, either by replace it with a constant or with another divisor node. The potential gain from this type of resubstitution comes from the removal of node g and its MFFC, as well as the change in functionality for the nodes in T. As in the node-based approach, we adopt factored-form literal costing to align with a standard-cell flow. In FPGA flows, transformations targeting LUT reduction do not need costing as the change in functionality does not impact the cost. For a node t, we first attempt to replace its edge to g by a constant. We use SAT-based interpolation on the window to check if node t can be resynthesized with the new support and to compute its new functionality if one exists. If the process fails, we try to replace the connection to g with divisors and compute an interpolant. If this procedure succeeds for every node in T and cost improves, node g is successfully removed along with the shared logic.

4 Multiple-Node Optimization Framework

This section introduces a multiple-node optimization (MNO) framework for mapped networks targeting both standard-cell and FPGA designs. The proposed framework integrates the techniques presented in this paper with technology mapping and Boolean simplification. While we discuss the flow for mapped netlists, natural extensions to technology-independent optimization are possible.

Algorithm 2 describes an engine for area optimization. For brevity, some parameters have been omitted from the pseudocode. The algorithm receives as input a mapped network, an effort level controlling the number of iterations, a gain threshold, and a technology library specifying the target technology and the costing method. For standard-cell designs, we use factored-form literal costing during optimization. For LUT networks, LUT count and edge count costing is applied in edge-based resub, while factored-form literal costing is used for the rest of optimization. Optimization is performed on a duplicated network, which is compared against the original one before returning to guarantee no degradation. The iterative loop is controlled by a budget ε, which is decreased after each iteration. We follow a waterfall model, selecting the first successful optimization group that succeeds, providing a good trade-off between QoR and runtime. We employ three optimization groups, each with increasing effort. The first block performs quick node and edge resub by running the algorithms in Section 3 without the multiple-node mode followed by incremental remapping. The second block performs deterministic circuit randomization (by reordering PIs and POs), multiple-node resub on nodes with fanout of 2, and incremental remapping. The third block performs stronger optimization employing deterministic circuit randomization, multiple-node resub on nodes with fanout size up to 3, logic simplification using don't cares [24], and incremental remapping. Until the gain exceeds the threshold T, the algorithm applies low-effort optimization. Once exceeded, stronger optimizations are attempted.

In our experiments, we generally employ an effort of $\varepsilon = 5$ and gain threshold $T = 0.3\%$ as empirically it shows a good balance between QoR and runtime. Additionally, we added a bailout threshold of 0.1% after the third iteration.

Algorithm 2: Multiple-Node Optimization (MNO) Flow

```
Input: Mapped Network N, Effort level ε, Gain threshold T,
       Technology library L
Output: Optimize Network N'
N' ← duplicate(N)
while ε > 0 do
    gain ← update_gain()
    if gain > T then
        mapped_node_resub(N')          ▷ Low effort
        mapped_edge_resub(N')
        map(N', L)      ▷ LUT or standard cell mapping
        ε ← ε − 1
        continue
    det_randomize(N')          ▷ Stronger optimization
    mapped_node_resub_mf(N', 2)
    mapped_edge_resub_mf(N', 2)
    map(N', L); ε ← ε − 1
    if gain > T then
        continue
    det_randomize(N')          ▷ Stronger optimization 2
    mapped_node_resub_mf(N', 3)
    mapped_edge_resub_mf(N', 3)
    simplify_dc(N'); sat_sweep(N'); map(N', L)
    ε ← ε − 1
if N better than N' then
    return N
return N'
```

5 Experimental Results

In this section, we present experimental results for our multiple-node optimization flow. First, we integrate the flow in a state-of-the-art industrial EDA flow, and show sensible Quality of Results (QoR) gains post synthesis. Second, we evaluate our approach to improve the best results in the EPFL benchmark suite [13].

Table 1: Post-Synthesis Results on 22 OpenCore Designs.

Flow	Area	Sw. Power	WNS	TNS	Runtime
Baseline [25]	1	1	1	1	1
SNO flow	-1.01%	-1.17%	+0.26%	+0.28%	+2%
MNO flow	-2.11%	-2.44%	+0.37%	-0.73%	+4%

Table 2: Post-Synthesis Results on 65 Industrial Designs.

Flow	Area	Sw. Power	WNS	TNS	Runtime
Baseline [25]	1	1	1	1	1
SNO flow	-0.52%	-0.38%	-0.13%	-2.05%	+3%
MNO flow	-1.50%	-1.60%	-0.24%	-2.22%	+6%

5.1 Evaluation on a Commercial Flow

We present experimental results for 87 standard cell designs, including 22 OpenCore [12] designs[1] and 65 industrial designs[2]. To highlight the added value of our approach, we compare against a strong baseline, the LUT-based engine flow [25], which applies advanced Boolean optimization, including single-node resub (both node-based and edge-based), on both AIGs and mapped networks. We run the proposed MNO flow (Algorithm 2) on top of this baseline, as described in Section 4. Additionally, we include a variant, named SNO, which runs only single-node optimization (group 1 of Algorithm 2). All results are measured post-synthesis, including technology mapping and post-mapping optimization.

Table 1 reports the results for 22 OpenCore designs. The proposed framework achieves an average area reduction of 2.11% (up to 6.07%) and a switching power reduction of 2.44% (up to 7.09%), while maintaining similar or improved timing and increasing the execution time within 4%. Compared to the SNO variant, our method improves area and power by over 1% on average, with reductions of up to 5.28% in area and and 7.29% in switching power.

Table 2 reports the results for 65 industrial designs. Our framework achieves an average area reduction of 1.50% (up to 6.79%) and a switching power reduction of 1.60% (up to 7.85%), again with similar or improved timing and a runtime increase within 6%. Compared to SNO, our method delivers over 1% improvement in both metrics, with maximum reductions reaching 6.02% in area and 8.84% in switching power on a particularly challenging design.

5.2 EPFL Best Results

In Table 3, we show that the proposed optimization flow can improve heavily optimized LUT networks. Our method achieves new best known results for 19 designs in the EPFL suite [26], with improvements highlighted in green and matches to the current best shown in blue. Previous best results were obtained using state-of-the-art heavy logic optimization (on AIGs, LUT networks, etc.) and *design-space exploration* (DSE) techniques, providing a strong baseline to highlight the additional gains of our multiple-node flow.

Table 3: New Best Area Results for the EPFL Suite [26].

Competition	Design	I/O	LUTs	Level
LUT-6 count	Divisor	128/128	3083	1094
	Hypotenuse	256/128	36476	4487
	Log2	32/32	6005	208
	Multiplier	128/128	4310	165
	Sine	24/25	1011	96
	Square	64/128	2926	154
	Arbiter	256/129	259	93
	Mem ctrl	1204/1231	1677	13
	Voter	1001/1	1161	27
LUT-6 level	Divisor	128/128	21810	173
	Hypotenuse	256/128	146037	473
	Log2	32/32	9413	51
	Max	512/130	1007	6
	Multiplier	128/128	6432	25
	Sine	24/25	666529	10
	Square	64/128	3695	10
	i2	147/142	192	3
	Mem ctrl	1204/1231	1724	5
	Voter	1001/1	1328	11

We use the flow in Algorithm 2 with effort level $\varepsilon = 5$ and gain threshold $T = 0.5\%$ on the previous best results. In the best LUT-6 count competition, our approach achieves 9 new best results. Additionally, most designs exhibit a significant reduction in logic levels. Compared to the currently known best results, our flow reduces the LUT-6 count by up to 1% and decreases LUT-6 levels by as much as 20%. In the best LUT-6 level competition, we configure Algorithm 2 to preserve logic levels. Our method finds 10 new best results with the same level count while reducing the LUT-6 count by up to 4.4%.

6 Conclusion

We presented a Boolean optimization framework that generalizes resubstitution to operate across multiple nodes, unlocking optimizations that go beyond the scope of traditional single-output transformations. To enable this, we proposed two advanced resubstitution variants, node-based and edge-based, formulated to restructure shared logic while leveraging controllability and observability don't-care conditions. Our approach expands the optimization scope from MFFCs to multiple-output windows, unlocking new optimization opportunities. Integrated into an optimization flow, the proposed method delivered measurable improvements in both academic and industrial settings. On 87 standard-cell designs, it achieved reductions up to 6.79% in area (1.7% average) and 7.85% in switching power (1.8% average) after synthesis. Moreover, our framework established 19 new best results in the challenging EPFL synthesis competition on LUT networks. These findings position multiple-node optimization as a powerful new direction for logic synthesis and motivate future research on scalable multiple-node transformations within advanced EDA flows.

[1]The OpenCore designs are ac97_ctrl, pci_bridge32, aes_core, pc_conf_cyc_ addr_dec, des_area, pci_spoci_ctrl, des_perf, sasc, ethernet, simple_spi, mem_ctrl, spi, ss_pcm, steppermotordrive, systemcaes, systemcdes, tv80, usb_funct, usb_phy, vga_lcd, wb_conmax, wb_dma.

[2]Details of the industrial designs cannot be disclosed due to proprietary information, including libraries and constraints.

References

[1] "TSMC's New 3nm Chip Wafers Priced at $20,000. [Online Nov 2025]." https://www.siliconexpert.com/blog/tsmc-3nm-wafer.

[2] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.

[3] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[4] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Int'l Workshop on Logic and Synthesis*, pp. 15–22, 2006.

[5] L. Amarù, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, J. Olson, R. Brayton, and G. De Micheli, "Improvements to Boolean resynthesis," in *Design, Automation and Test in Europe*, pp. 755–760, 2018.

[6] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting a fresh look at combinational logic synthesis," in *Design Automation Conference*, pp. 532–535, 2006.

[7] R. K. Brayton and C. T. McMullen, "The decomposition and factorization of Boolean expressions," in *Int'l Symp. on Circuits and Systems*, pp. 49–54, 1982.

[8] X. Zhu, R. Tang, L. Chen, X. Li, X. Huang, M. Yuan, W. Sheng, and J. Xu, "A database dependent framework for k-input maximum fanout-free window rewriting," in *Design Automation Conference*, pp. 1–6, 2023.

[9] M. Damiani, J.-Y. Yang, and G. De Micheli, "Optimization of combinational logic circuits based on compatible gates," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1316–1327, 1995.

[10] L. Benini, P. Vuillod, and G. De Micheli, "Iterative remapping for logic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 948–964, 1998.

[11] F.-X. Reichl, F. Slivovsky, and S. Szeider, "eSLIM: circuit minimization with SAT based local improvement," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, vol. 305, pp. 23:1–23:14, 2024.

[12] "Opencores benchmark suite. [Online Nov 2025]." https://opencores.org.

[13] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Int'l Workshop on Logic and Synthesis*, 2015.

[14] R. Tang, X. Zhu, L. Chen, X. Li, X. Huang, M. Yuan, and J. Xu, "Maximum fanout-free window enumeration: Towards multi-output sub-structure synthesis," in *Design, Automation and Test in Europe*, pp. 1–7, 2025.

[15] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A simulation-guided paradigm for logic synthesis and verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2573–2586, 2022.

[16] A. Costamagna, A. Tempia Calvino, A. Mishchenko, and G. De Micheli, "Area-oriented optimization after standard-cell mapping," in *Asia and South Pacific Design Automation Conference*, (New York, NY, USA), Association for Computing Machinery, 2025.

[17] A. Costamagna, A. Tempia Calvino, A. Mishchenko, and G. De Micheli, "Area-oriented resubstitution for networks of look-up tables," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 44, no. 7, pp. 2571–2584, 2025.

[18] F. Brglez, D. Bryan, J. Calhoun, G. Kedem, and R. Lisanke, "Automated synthesis for testability," *IEEE Trans. on Industrial Electronics*, vol. 36, no. 2, pp. 263–277, 1989.

[19] D. Bryan, F. Brglez, and R. Lisanke, "Redundancy identification and removal," *Int'l Workshop on Logic and Synthesis*, 1991.

[20] A. Mishchenko, R. K. Brayton, J. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 4, no. 4, pp. 34:1–34:23, 2011.

[21] A. Tempia Calvino, A. Mishchenko, H. Schmit, E. Mahintorabi, G. D. Micheli, and X. Xu, "Improving standard-cell design flow using factored form optimization," in *Design Automation Conference*, pp. 1–6, 2023.

[22] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *Asia and South Pacific Design Automation Conference*, pp. 395–402, 2022.

[23] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko, "A circuit-based SAT solver for logic synthesis," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–6, 2021.

[24] L. Amarú, V. Possani, E. Testa, F. Marranghello, C. Casares, J. Luo, P. Vuillod, A. Mishchenko, and G. De Micheli, "LUT-based optimization for ASIC design flow," in *Design Automation Conference*, pp. 871–876, 2021.

[25] W. L. Neto, L. Amarú, V. Possani, P. Vuillod, J. Luo, A. Mishchenko, and P.-E. Gaillardon, "Improving lut-based optimization for asics," in *Design Automation Conference*, DAC '22, p. 421–426, Association for Computing Machinery, 2022.

[26] "EPFL Synthesis Competition Best Results [Online Nov 2025]." https://github.com/lsils/benchmarks/tree/v2025.1/best_results.