Area-Oriented Optimization After Standard-Cell Mapping

Andrea Costamagna
EPFL
Lausanne, Switzerland
andrea.costamagna@epfl.ch

Alessandro Tempia Calvino
EPFL
Lausanne, Switzerland
alessandro.tempiacalvino@epfl.ch

Alan Mishchenko
UC Berkeley
Berkeley, California
alanmi@berkeley.edu

Giovanni De Micheli
EPFL
Lausanne, Switzerland
giovanni.demicheli@epfl.ch

ABSTRACT

We address the problem of minimizing the area of circuits mapped to a technology library, with or without delay constraints. While traditional methods optimize first a technology-independent representation and then perform technology mapping to a library, this paper explores the potential for further optimizations through technology-dependent algorithms. We propose an optimization engine for mapped circuits that relies on a database of mapped sub-networks for efficient resynthesis. Experimental results on the EPFL benchmarks after area-oriented optimization and mapping show that the proposed method leads to average area improvements of 5.47% without degrading the delay.

CCS CONCEPTS

• **Hardware** → **Combinational circuits**.

KEYWORDS

Resynthesis, Mapped circuits, Standard cells

ACM Reference Format:
Andrea Costamagna, Alessandro Tempia Calvino, Alan Mishchenko, and Giovanni De Micheli. 2025. Area-Oriented Optimization After Standard-Cell Mapping. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3658617.3697722

1 INTRODUCTION

Dennard scaling of transistors motivated area-minimization of digital circuits for over fifty years, resulting in higher performance and reduced power consumption [13]. While the demand for better performance continues to grow, financial and physical limitations hinder delivering higher performance by transistor scaling alone [12]. As a result, further improvements in computing power requires higher optimization effort at the design level [15].

State-of-the-art logic synthesis tools optimize technology independent representations and map them to a technology library [24, 14]. In this case, reducing the number of gates in the technology-independent representation is considered an effective heuristic for reducing area after mapping [30, 7]. However, high-effort optimization of the technology-independent representation does not always correlate with the quality of a mapped circuit [21].

Technology-aware logic synthesis has recently gained momentum as a way to improve the quality of mapped circuits [3, 1, 6, 32]. This approach aims to integrate technology-dependent information into logic optimization, thereby improving the quality of the mapped circuits. In line with this approach, this paper presents a technology-aware logic synthesis algorithm for area-oriented optimization, with or without delay constraints.

The algorithms discussed in this paper are implemented in the first open-source engine for optimizing circuits mapped with a library of *standard cells*. Our approach takes a mapped netlist and replaces circuit sub-portions with high-quality mapped sub-networks stored in a database. We rely on the dependency theory [39, 34] to identify optimization opportunities not available to the technology mapper, as they involve non-local restructuring of the netlist. Furthermore, we leverage the *don't cares* during resynthesis to increase the optimization quality.

Experiments on the EPFL and IWLS benchmarks confirm that aggressive area optimization of the technology-independent representation does not maximize area reductions after mapping. This justifies the need for area optimization of mapped networks. We apply our technology-aware resubstitution algorithm to mapped designs after area-oriented technology-independent optimization and mapping. On the EPFL benchmarks, our method achieves an additional 5.47% average area reduction without delay degradation.

2 BACKGROUND

Given a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, and a technology library L, we consider the problem of synthesizing a network of gates from L that implements f while minimizing circuit area, with optional timing constraints. This problem is addressed by logic synthesis.

2.1 The Two-Steps Approach to Logic Synthesis

State-of-the-art logic synthesis follows a two-step approach [7]. First, the functional specifications are represented as a simple circuit named *subject graph*, which is optimized for area and/or delay. A commonly used subject graph is the *and-inverter graph* (AIG), where each node is a two-input and-gate and complemented edges denote signal negation. Subsequently, the *subject graph* is covered with

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ASPDAC '25, January 20–23, 2025, Tokyo, Japan
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697722

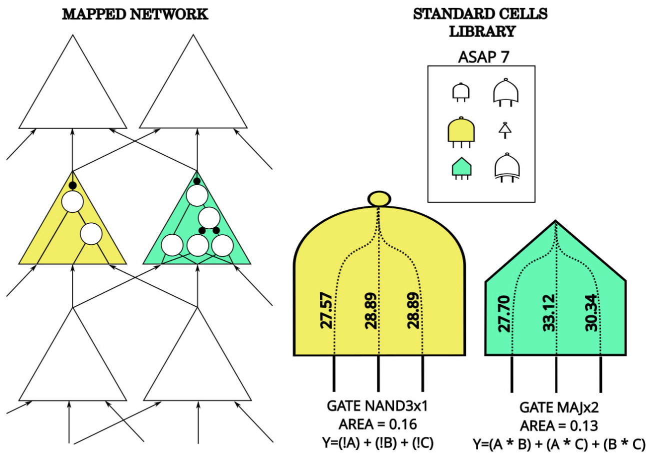

Figure 1: Representation of a mapped network as the cover of a subject graph with gates from a standard cell library. In the figure, each cell is characterized by its functionality, area information, and pin-to-pin propagation delays.

cells from a *technology library* during *technology mapping*. This process is represented in Figure 1.

To obtain a mapped circuit optimized according to a cost function, AIG optimization must be guided by heuristic criteria estimating the correlation between AIG optimization steps and improvements after mapping. A common assumption for area-oriented optimization is that reducing the number of nodes in the *subject graph* correlates with reduced area after mapping [30, 7]. This assumption relies on a property named *structural bias*: the structure of the mapped netlist strongly depends on the subject graph [8].

While *structural bias* is inevitable during technology mapping, it is important to mitigate it in order to improve the quality of mapped networks [8]. Since AIG optimization is agnostic of the properties of the gates in the library that are only available during mapping, resynthesis after mapping, as described in this paper, helps unlock optimization opportunities that cannot be obtained at the AIG level.

2.2 Boolean Networks Terminology

Let us consider a Boolean network. If there is a path from a node x_i to a node x, then x_i is in the *transitive fanin* (TFI) of x. The primary inputs (PIs) are nodes without fanins in the network and the primary outputs (POs) are nodes without fanouts in the network.

The *maximum fanout free cone* (MFFC) of node x is the subset of nodes in the TFI of x such that every path from a node in the subset to the POs passes through x. The MFFC of a node contains the portion of the circuit used exclusively to compute the functionality of x. When removing a node, its MFFC can also be removed.

A *structural cut* C of a Boolean network is a pair $(x, \mathcal{L})$, where x is a node, called *root*, and $\mathcal{L}$ is a set of nodes, called *leaves*, such that 1) every path from any *primary input* (PI) to node x passes through at least one leaf and 2) for each leaf $v \in \mathcal{L}$, there is at least one path from a PI to x passing through v and not through any other leaf.

Given a cut $C = (x, \mathcal{L})$, the paths connecting the leaves to the root identify a sub-network synthesizing a Boolean function named *cut functionality*. If all the leaves are PIs of a n-inputs network, the cut functionality is the *global function* of the node. A p-bit *simulation signature* for a node x is a Boolean vector approximating the global function of node x obtained by simulating the network with a set of p bit-level simulation patterns assigned at the PIs [29].

2.3 Peephole Optimization

Peephole optimization is an optimization strategy that involves improving structural sub-portions of the target network, named *windows*, by minimizing a local cost function [28, 19]. When optimizing a Boolean network for area, peephole optimization consists in *resynthesizing* the function of a node using a set of candidate nodes [26]. The transformation is accepted if the resynthesis sub-circuit makes redundant a larger area sub-circuit, which can thus be removed.

We focus on three well established peephole optimization strategies, briefly discussed in the following. While these transformations are commonly applied to the subject graph, this paper extends their use to mapped netlists.

Cut-based rewriting [25, 22] is an optimization strategy which involves enumerating a set of structural cut for each node of the network. Next, the functionality of the cut is extracted, and the minimum size circuit is synthesized and used to replace the sub-circuit currently present in the netlist if it has a smaller area. This method is made scalable by precomputing optimum sub-circuits for small Boolean functions and storing them in a database.

In *window-based resubstitution*, a *window* is a sub-network that is structurally built around the target node. The nodes in a window that are not in the MFFC are named divisors [28]. Each window is exhaustively simulated, resulting in the *local function* of each divisor. Next, heuristics attempt resynthesizing the function of the target node with a circuit whose area is smaller than the MFFC.

In *simulation-guided resubstitution*, rather than using exhaustive window simulation, the nodes' functions in the window are represented using *simulation signatures*. This approach has the key advantage of using global functional information. However, the signatures are approximations of the global functions. Their use requires equivalence checking in the end to verify the correctness of each transformation [20, 40].

2.4 Resynthesis and Dependency Cuts

Most state-of-the-art resubstitution engines use on-the-fly decomposition heuristics, simultaneously identifying useful divisors and adding them to the resynthesis sub-network. In contrast to on-the-fly decomposition, alternative approaches divide the process into two phases: divisor selection and synthesis [40, 11].

During the divisor selection phase, the goal is to identify a subset of nodes $C = (x, \mathcal{L})$ that is not (necessarily) a structural cut in the current topology but has the potential to become one. Named *dependency cut*, this subset exists if there is a function $h : \mathbb{B}^{|\mathcal{L}|} \rightarrow \{0, 1, *\}$ such that $x = h(\mathcal{L})$. The synthesis phase then generates a sub-network implementing this function.

Given a Boolean function $x : \mathbb{B}^n \mapsto \{0, 1, *\}$, its *set of pairs of functions to be distinguished* (SPFD) is a mathematical construct $\Upsilon_x : \mathbb{B}^n \times \mathbb{B}^n \rightarrow \mathbb{B}$ encoding the Boolean function's ability to distinguish the points of its input space into *onset* and *offset* [39, 34, 17].

The SPFD of a function x is the set of minterm pairs for which the function has different values, i.e., $\Upsilon_x = \{(M_1, M_2) \text{ s.t. } x_{M_1} \neq x_{M_2}\}$.

In the rest of the paper, we will refer to the node's SPFD as its *functional information*. A set of nodes $\mathcal{L}$ is a *dependency cut* for target node x, $C = (x, \mathcal{L})$, if the functional information of the target node is contained in that of the nodes belonging to $\mathcal{L}$ [16, 39, 11]:

$$\Upsilon_x \subseteq \bigcup_{y \in \mathcal{L}} \Upsilon_y. \quad (1)$$

Recent work [11] has identified an SPFD manipulation technique allowing for solving Eq. 1 with simulation signatures whose dimension is up to 2^{12}. This technique provides a way to identify how many minterm pairs of the SPFD remain after removing the minterm pairs present in the SPFD of the divisor y: $|\Upsilon_x \setminus \Upsilon_y|$. However, the authors did not exploit the full potential of the method since they relied on decomposition for resynthesis [11]. Furthermore, their *dependency cut* selection is stochastic in nature, which is often regarded as undesirable for EDA development, and we address this issue by proposing a deterministic version of their algorithm.

In this paper we observe that the true potential of *dependency cut* selection lies in combining the best features of resubstitution and database-powered rewriting. This observation has been the key motivation of our research.

2.5 Post Mapping Optimization: Previous Works

One of the first approaches for optimizing networks after technology mapping was proposed by Benini et al. [4]. This method extracts windows of two or three cells and remaps them at the same time using *generalized matching*. While the method can leverage Boolean *don't cares* and even Boolean relations, it only performs structural cell substitutions without resynthesis.

Kravets et al. devised an engine for resubstitution after technology mapping [18]. Their method is *window*-based, and can be formulated as a dependency cut-selection problem followed by resynthesis. However, their method uses gate decomposition from the library, while we rely on database rewriting, enhancing scalability and reducing computation. Additionally, our engine combines window-based resubstitution with structural cut enumeration and signature-based cut selection, increasing the variety of rewiring opportunities. Furthermore, our synthesis strategy also exploits don't cares during database look-up to reduce area.

More recently, Amarú et al. developed advanced filtering techniques to increase the scalability of Boolean methods [2]. They propose a resubstitution algorithm, capable of inferring complex gates during resynthesis. The authors mention that this method can be used for optimizing mapped netlists. Their algorithm is a window-based resubstitution which uses precomputed gate structures, which are often single gates from a technology library. Our method, on the other hand, is more general than this approach in that our resynthesis sub-networks are not limited to single-gate netlists. The use of different support selection strategies gives our method additional restructuring capabilities.

3 POST-MAPPING RESYNTHESIS

In this section, we introduce novel contributions enabling scalable post-mapping design space exploration. The algorithms we developed rely on the observation that *dependency cuts* enable combining the global restructuring capabilities of *resubstitution* with database rewriting. Given a technology library, a database can be generated using the proposed method. Overall, the contribution of this paper is to extend technology-dependent optimization with engineering solutions that were previously used for technology-independent representations. By combining them in the proposed optimization engine, we develop a novel optimizer for mapped netlists.

3.1 Database Generation

We begin by considering a simplified version of the area minimization problem, when the function $f : \mathbb{B}^4 \to \mathbb{B}$ has one output and at most four inputs. The number of inputs is chosen to ensure efficient truth table manipulation while considering non-trivial structures, like the one in Figure 2. Minimum node AIGs for four-input functions can be easily computed, and mapping them to technology leads to near-optimal mapped netlists.

The database we construct contains a mapped netlist for each four-input function. We limit the size of the database taking functional symmetries into account. In particular, two functions belong to the same *P-class* if one function is equivalent to the other under some input permutation [33]. Since permutations have zero cost during technology mapping, we synthesize only the 3984 functions representing all *P-classes* of four-input functions. The *representative* functions are those having the lexicographically smallest truth tables among all the functions belonging to an *P-class*.

3.2 Boolean Matching with Don't Cares

Let $C = (x, \mathcal{L})$ be a *dependency cut*, and $f : \mathbb{B}^4 \to \{0, 1, *\}$ be the cut functionality, possibly specified with *don't cares*. During resynthesis, we want to identify the best sub-netlist synthesizing the functionality, and the *don't cares* offer a degree of freedom: various assignments of them yield resynthesis candidates with different area. Previous research investigated algorithms for *Boolean matching with don't cares*, that is the problem of performing database look-up exploiting this flexibility [23, 38, 5]

The number of matches in the database grows exponentially with the number of *don't cares*. However, we empirically verify that, for the strategies adopted in this work, there are on average two don't care minterms, corresponding to four completely specified functions per cut. Due to the small number of *don't cares*, enumerating through all the compatible completely specified functions is feasible. Hence, given the cut functionality, we access the database by assigning *don't cares* in all possible ways and identifying the *P*-representative for each function. Next, database look-up allows us to obtain the pre-computed netlist in constant time.

3.3 Structural Network Exploration

We represent mapped networks as those where each node is associated with a gate identifier from the library. As a consequence, structural cut enumeration algorithms can be extended to identify structural cuts in a mapped network. Figure 2 shows an example of structural cuts enumeration. Each node is a gate from the *asap*7 [10] technology library, characterized by area and functionality.

For each enumerated cut, we extract the cut functionality and compute its *P representative*. Next, we use the representative truth table to look up the precomputed structure stored in the database.

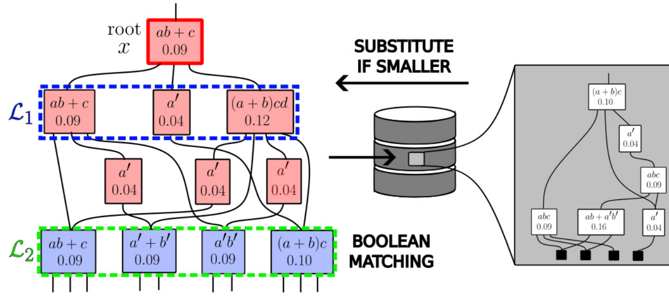

Figure 2: Two structural cuts $C_1 = (x, \mathcal{L}_1)$ and $C_1 = (x, \mathcal{L}_2)$ identified during network exploration and database-based rewriting. The figure represents a portion of a larger design.

If the area of the stored network is smaller than the area of the sub-circuit contained in the cut, we perform the substitution.

Figure 2 shows an example encountered while optimizing the EPFL benchmarks, in which a sub-network of area $0.56\mu m^2$ can be replaced with a sub-network of area $0.52\mu m^2$. Structural cut enumeration is extremely fast, so that identifying optimizations based on this transformation should be preferred due to its scalability. However, computing structural cuts limits optimization to the transitive fanin of the node, which is also explored by the mapper. The proposed approach, however, is more general because it explores additional resynthesis opportunities arising due to non-local transformations outside of the TFI cone.

3.4 Window Construction and Simulation

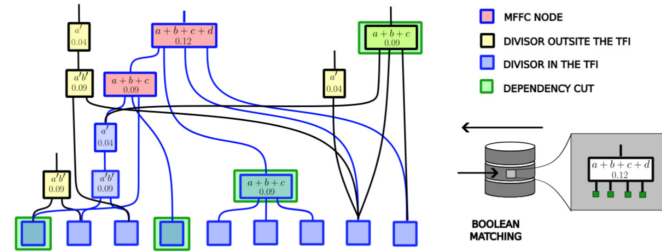

Figure 3: Example of a small window encountered while optimizing the EPFL benchmarks.

Inspired by the recent work on *simulation-guided resubstitution* [19, 20], we build windows using the notion of reconvergence driven cuts. Given a target node, we first identify a large structural cut, corresponding to the blue nodes at the bottom of Figure 3. The sub-circuit between this cut and the target node can be divided into two parts: the MFFC nodes and the divisors in the node's TFI.

The MFFC nodes are the nodes for which each path from their output to a primary output passes through the target node. Upon successful resubstitution, these nodes can be removed and their area can be saved. The second group of nodes cannot be removed after successful resubstitution because there are paths connecting their fanout to the POs without passing through the target node, i.e., they are used for synthesizing other nodes in the design. These nodes are named *divisors*, and are nodes that can potentially be used for resynthesizing the target node.

We enlarge the set of divisors by including some nodes outside of the TFI of the target node, but with fanins in the divisor set. These nodes are highlighted in yellow in Figure 3. The extended set of divisors is used as input to the *dependency cut* selection algorithm. A resulting dependency cut in Figure 3 is highlighted in green.

We associate each node x in the window with two Boolean functions. The local Boolean function $\varphi_n : \mathbb{B}^k \rightarrow \{0,1\}$ is obtained by exhaustively simulating the window for all possible window input patterns. The global function $\psi_n : \mathbb{B}^n \rightarrow \{0,1,?\}$ is the *simulation signature*, a partially specified Boolean function defined over the primary inputs of the design. The value of the function is unknown (?) for some circuit's input patterns, but it contains global information characterizing the node.

3.5 Deterministic Dependency Cut Selection

Let $\mathcal{W} = (x, \mathcal{D}, \mathcal{M})$ be a window, where x is the target node, $\mathcal{D}$ is the set of divisors, and $\mathcal{M}$ is the MFFC. Also, let ϕ_{x_i} be the functional information of a node $x_i \in \mathcal{W}$. This function can be φ_{x_i} or ψ_{x_i}. Algorithm 1 describes the proposed deterministic algorithm for identifying *dependency cuts* using functional information.

Algorithm 1: Deterministic *dependency cut* selection

```
Data: A window W = (x, D, M), functional information of
      each node φ_{x_i}, maximum cut size k, and maximum
      number of attempts T
Result: A dependency cut C = (x, L) with |L| ≤ k if found
D̃ ← sort the divisors based on |Υ_{φ_x} \ Υ_{φ_{x_i}}|;
attempts ← 0;
for x_i ∈ D̃ and attempts++ ≤ T do
    L ← x_i;
    Υ ← Υ_{φ_x} \ Υ_{φ_{x_i}};
    while |L| ≤ k do
        x* = arg min_{x_j ∈ D} {|Υ \ Υ_{x_j}|};
        L ← L ∪ {x*};
        Υ ← Υ \ Υ_{φ_{x*}};
        if Υ = ∅ then
            return C = (x, L);
return cut does not exist;
```

At the beginning, we sort the divisors by the amount of information they share with the target node [17, 34]. Next, we try building a solution by forcing it to contain one of the first T divisors, where T is a parameter chosen by the user. The algorithm performs a greedy set covering to find a set of divisors that contains the complete functional information of the target node.

This approach is effective because the greedy set covering is provably the best polynomial time approximation algorithm for set cover [9]. Since a solution to set covering might not include the set that covers most elements [35], this algorithm considers different starting points. We empirically observed that this approach is effective for detecting dependency cuts. Furthermore, this algorithm is deterministic, which is a desirable feature in the EDA applications.

It is important to note that an enumeration-based exact solution is computationally infeasible for industrial-scale designs, necessitating the use of approximation algorithms, such as the one we developed. However, in addition to the aforementioned computational advantages of limiting the support size to 4, this choice also results in a small optimality gap between a solution to Eq. 1 found with a greedy-like heuristic, and the exact one [37].

3.6 Functionality Extraction and Its Validation

Let $C = (x, \mathcal{L})$ be a *dependency cut*, ϕ_x the functional information of the target node x, and ϕ_{x_i} the functional information of a leaf node $x_i \in \mathcal{L}$. The key implication of the dependency theorem, expressed in Equation 1, is that, given a dependency cut there exist a function $h : \mathbb{B}^{|\mathcal{L}|} \rightarrow \mathbb{B}$ such that $\phi_x = h(\{\phi_{x_i}\}_{x_i \in \mathcal{L}})$. The truth table of the function h can be easily extracted by identifying the value of ϕ_x for each pattern in $\{\phi_{x_i}\}_{x_i \in \mathcal{L}}$. Not all the minterms in $\mathbb{B}^{|\mathcal{L}|}$ necessarily appear, and the missing patterns are the *don't cares* of the function h. This function gives the specification for the substitution candidate, synthesizable using a database look-up.

It is essential to note that a *dependency cut* obtained with $\phi_x = \varphi_x$ is guaranteed to be a dependency cut for the global function of the nodes, but when $\phi_x = \psi_x$ this is not guaranteed because simulation signatures only contain partial information. In the latter case, equivalence checking is needed to verify that a transformation preserves the functional equivalence of the design, as shown below.

3.7 The Optimization Engine

Algorithm 2 illustrates our engine. The algorithm allows the user to set a required time at the outputs τ^R_{max}, which is desirable when preserving timing constraints is critical.

The algorithm samples P input patterns of the design and extracts the simulation signatures ψ for each node of the network. Next, the network is explored one node at the time. For each node, we extract a window having a maximum input size C_w, and we exhaustively simulate it to obtain the functional information φ. We then do structural cut enumeration and try computing one *dependency cut.* After extracting the functionalities of theses cuts, we match them against the database. If any of these netlists induces an area reduction, we perform the substitution.

If none of the previous approaches yields an area advantage, it is still possible that using more global information will result in a larger number of *don't cares*, whose exploitation might yield improvements. We therefore attempt to identify a *dependency cut* using simulation signatures. However, if the cut is found and the area is improved, we must verify the correctness of the transformation. We do so by performing SAT-based equivalence checking of the target node with the resynthesis sub-network.

We prioritize *window-based resubstitution* and *cut rewriting* since they are more scalable. To save the runtime of equivalence checking when the proposed method uses *simulation signatures*, we perform it only if other possibilities fail.

If equivalence checking is successful, we update the timing and move on to the next node. Otherwise, we store the counter example found by the SAT solver. When the number of derived counter examples reaches the size of a machine word (e.g., 64 bits) we use them to re-simulate the network and replace the oldest bits in the signatures. This approach minimizes memory reallocation and ensures computational efficiency.

Algorithm 2: Technology-Aware Resubstitution

```
Data: A mapped circuit with required time τ^R_max
Result: A new mapped circuit optimized for area
B_P ← Sample P patterns at random from B^n;
ψ_P ← Functional simulation of G using B_P;
for x ∈ G do
    Build a window of input size C_w for node x;
    φ ←Exhaustive window simulation;
    C_s, A_s ← Find the best structural cut;
    C_d, A_d ← Find a Dependency cut φ in T attempts;
    if (A(C_s) ≤ A(M) or A(C_d) ≤ A(M)) then
        Resynthesize the best cut and substitute x;
    else
        C_d, A_d ← Find a candidate dependency cut;
        if A_d < A(M) and τ^A_{x_new} ≤ τ^R_x preserved then
            x_new ←Resynthesize C_d;
            if x_new and x are globally equivalent then
                Substitute x with x_new;
            else
                Save the counter-example in B_cex;
                if |B_cex| is equal to a word length then
                    σ_64 ← Simulate G using B_cex;
                    Replace the oldest signatures;
                    B_cex ← ∅ ;
    if successful resubstitution and τ^R_max < ∞ then
        Update the arival times and the required times;
return the optimized circuit;
```

4 EXPERIMENTS

This section presents experimental results using the 7*nm* standard cell library `asap7`[10]. For technology mapping, we utilize the state-of-the-art mapper `emap` [31] , implemented in `Mockturtle` [36].

Direct comparison with prior work is hindered by limitations in reproducibility and outdated benchmarking standards, such as reliance on proprietary code, closed-source libraries [4], resynthesis engine evaluations solely on outdated cell libraries and focused on delay-optimization under area constraints [18], and the lack of standard cell mapping evaluations [2]. To address these issues, we establish a solid baseline by implementing high-effort, technology-independent optimization followed by area-oriented technology mapping, ensuring robust and practical benchmarking.

4.1 Technology-Independent Assumptions

In this experiment, we investigate the correlation between node count in the *subject graph* and the area after technology mapping. We use an aggressive optimization flow running the following area-oriented commands and scripts in `ABC`: `rw`, `rs`, `rf`, `resyn2rs`, and `compress2rs`. These are common area-oriented optimization algorithms: `rw`, which performs cut-based rewriting [25, 22]; `rs`, which applies window-based resubstitution [28]; and `rf`, a variation

Table 1: The worst-case analysis of the improvement of technology-aware resubstitution. The benchmarks are those with less than 200K nodes from the EPFL and IWLS benchmark suites. The benchmarks are optimized with high-effort AIG optimization before mapping. The best possible delay is used as the delay constraint.

Design	$A_i[\mu m^2]$	$\delta A_i^1[\%]$	$\delta A_i^\infty[\%]$	$A_e[\mu m^2]$	$\delta A_e^1[\%]$	$\delta A_e^\infty[\%]$	$D_e[ps]$	$\delta D_e^1[\%]$	$\delta D_e^\infty[\%]$	$t_e^1[s]$	$t_e^\infty[s]$
div	3914.60	−14.35	−23.28	1296.90	−7.81	−9.30	60248.23	0.00	0.01	7.09	38.40
sqrt	1372.25	−12.44	−16.30	1171.15	−3.16	−5.22	78957.63	0.00	−1.21	4.09	50.83
arbiter	557.84	−13.87	−41.31	557.84	−12.01	−51.81	999.95	0.00	−39.57	1.59	17.38
mem_ctrl	2547.32	−7.87	−18.47	2063.01	−5.86	−12.79	1649.46	−5.82	−13.00	18.41	183.16
aes_core	1198.55	−4.18	−5.80	1106.60	−1.07	−1.81	434.52	−2.29	−2.05	11.41	165.45
ethernet	4411.85	−2.99	−4.35	3123.35	−0.96	−3.68	588.34	0.00	0.00	54.60	214.07
iwls05_i2c	66.32	−8.56	−12.26	49.96	−1.46	−2.00	288.07	0.00	0.00	0.19	0.58
RISC	4145.01	−8.10	−8.10	3172.54	−1.03	−1.03	1304.74	0.00	0.00	350.90	350.90
sasc	40.46	−1.29	−1.83	31.72	−1.07	−1.32	191.00	0.00	0.00	0.16	0.48
simple_spi	54.77	−4.78	−6.50	41.58	−0.53	−1.39	287.00	0.00	0.00	0.17	1.06
spi	205.45	−4.16	−9.05	167.59	−1.52	−2.08	489.49	0.00	0.00	0.43	3.66
systemcaes	611.55	−4.10	−5.75	530.89	−3.25	−3.62	784.00	0.00	0.00	1.23	11.28
systemcdes	172.17	−8.11	−13.69	142.22	−1.21	−2.38	530.29	0.00	0.00	0.39	2.22
tv80	512.39	−7.43	−12.29	356.38	−0.70	−1.48	907.86	0.00	0.00	1.03	6.84
usb_funct	887.05	−4.16	−7.41	702.45	−0.70	−0.98	722.00	−2.65	−2.65	1.64	12.45
usb_phy	28.82	−6.45	−10.58	23.97	−0.83	−1.42	179.06	−7.79	−8.51	0.15	0.61
		−5.75%	**−10.21%**		**−1.21%**	**−2.81%**		**−0.48%**	**−1.81%**	17.62	47.15

of rewriting that, for each node, computes a large structural cut and seeks to replace the current AIG structure with a factored form of the cut function [26]. Additionally, the flows `resyn2rs` and `compress2rs` are widely accepted in the community, devised through designer expertise and practical insights.

Before applying any optimization, the AIG is functionally reduced using command `fraig` [29]. This step ensures that no two nodes in the AIG have the same functionality, thereby eliminating trivial optimization opportunities and strengthening the baseline.

As soon as one heuristic successfully reduces the number of AIG nodes, we map the AIG and plot the number of AIG nodes and the area. We start with one-pass commands `rw`, `rs`, `rf` to slow down convergence, which helps articulate the correlation between technology-independent optimization and area after mapping. We iterate this procedure as long as there are changes in the subject graph.

Figure 4 shows the typical trends observed on the EPFL and IWLS benchmarks. For most benchmarks, at least in the first few optimization steps, we observe a good correlation between AIG minimization and area reduction. For instance, in highly non-optimized AIGs, such as `mem_ctrl` (top-left of Figure 4), the correlation between AIG minimization and area reduction is consistently strong, without significant fluctuations. However, there are benchmarks in which improvements in the AIG size can have an adverse effect, resulting in increases of area after mapping by several percent.

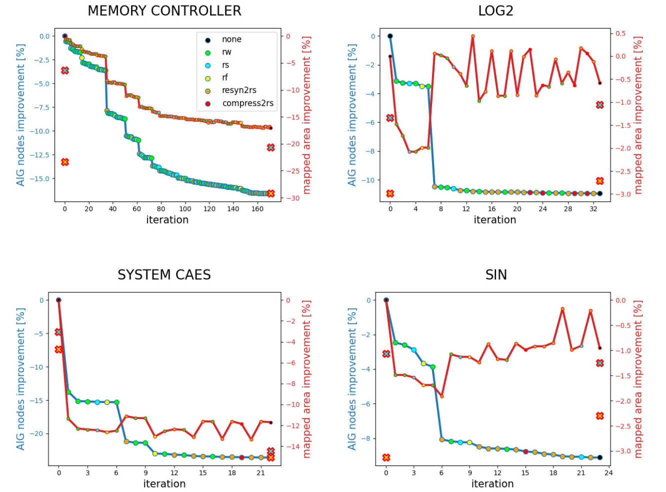

Figure 4: Correlation the subject graph size and the area after mapping, reflecting typical trends for the EPFL and IWLS benchmarks. The crosses show the results of post-mapping optimization, applied once and until convergence, for the original AIG and the optimized AIG.

4.2 Post-Mapping Area Optimization

Each part of Figure 4 contains four crosses, which show the result of optimizing two versions of the subject graph (the initial AIG and the AIG that could not be further optimized) using two types of resynthesis (by a single traversal of the graph and by iteratively traversing the graph until convergence).

In the case of `mem_ctrl`, `log2`, `system_caes`, and `sin`, despite the high-effort AIG optimization, the proposed algorithm can achieve further area reduction. Furthermore, it is interesting to observe that in `log2` and `sin`, the best optimization result is not achieved with the two-step process, but rather by directly optimizing the network mapped using the unoptimized subject graph.

4.3 Area Under Delay Constraints

This experiment shows a detailed analysis on the EPFL and IWLS benchmarks using the subject graphs that are unoptimized and highly optimized. High-effort optimization is achieved by iteratively applying `resyn2rs` and `compress2rs` until the *subject graph* cannot be reduced, and merging equivalent nodes using command `fraig` before applying any script. After the high-effort optimization, we map the network using area-oriented mapping and optimize the network without allowing delay to increase. The reported runtimes include also the time needed to run the SAT solver during signature-based resubstitution, showing the scalability of the method.

Table 1 shows the following: A_i is the mapping area using an unoptimized AIG. A_e is the mapping area using an optimized AIG. D_e is the mapping delay using an optimized AIG. Indicating with Q a quantity, that can be area A or delay D, $\delta Q^1_{i,e}$ is the improvement after one optimization round and $\delta Q^{\infty}_{i,e}$ is the improvement after iterative optimization. We use fixed *simulation signatures* of size 2^{10}, and the windows are limited to at most 10 inputs and 256 divisors.

Table 1 reports the results for the EPFL and IWLS benchmarks. We consider 39 designs whose *subject graphs* initially have less than 200K nodes due to the high runtime of the flow including both AIG optimization and mapping. The average values at the bottom of the tables are computed for all 39 benchmarks. Table 1 shows that post-mapping optimization leads to noticeable reductions in area while the worst-case delay is not increased.

4.4 Design Space Exploration

The last experiment performs design space exploration for mapped circuits. Table 2 shows the results for selected EPFL benchmarks while the average is for all of them. The subject graphs are optimized using two iterations of the script "`fraig; compress2rs; resyn2rs`". Next, the subject graph is mapped and optimized under stringent delay constraints. In one case (column δA^{∞}), we iterate optimization until convergence. In the other case (column $\delta A^{5\times5}$), after 5 iterations of technology-aware resubstitution, we unmap the network, remove redundancies in the resulting AIG and balance it [27], in order to move to a different region of the design space. Finally, we remap it and restart area-oriented optimization. We report the best encountered results where each benchmark also satisfies the delay constraints. This experiment shows that our engine effectively uses logic restructuring for mapped networks, which directly benefits design space exploration.

5 CONCLUSION

This work proposes a *technology-aware* area optimization for mapped networks, motivated by the reduced correlation between technology-independent and technology-dependent optimization after a few iterations of the former. The method is customized to work for *standard cell* designs. Experiments on the EPFL and IWLS benchmarks show that applying our method after aggressive logic minimization and area-oriented technology mapping further reduces area by 2.81% on average and up to 51.81% for some test cases without delay penalty. The method helps reduce area by 5.47% when used as part of design space exploration. These encouraging results suggest the importance of further investigating Boolean methods in technology-aware optimization.

Table 2: Mapped design space exploration.

Design	$A[\mu m^2]$	$\delta A^{\infty}[\%]$	$\delta A^{5\times5}[\%]$	$t^{\infty}[s]$	$t_e^{5\times5}[s]$
bar	149.13	−0.04	−3.24	0.50	7.88
div	1302.07	−9.96	−15.54	91.97	75.89
sin	289.47	−0.24	−1.41	2.58	35.02
sqrt	1171.15	−3.99	−5.98	28.14	89.16
arbiter	557.84	−45.59	−55.49	11.74	19.71
cavlc	34.53	−0.96	−1.13	0.85	6.90
ctrl	5.90	−2.71	−3.90	0.29	4.92
i2c	59.23	−0.95	−1.08	0.62	6.09
mem_ctrl	2164.98	−13.96	−11.38	187.81	256.89
priority	27.66	−0.29	−2.75	0.30	5.34
		−4.23%	−5.47%		

ACKNOWLEDGEMENTS

This research was supported in part by the SRC Contract 3173.001, "Standardizing Boolean transforms to improve the quality and runtime of CAD tools," and in part by Synopsys.

REFERENCES

[1] Luca Amarú, Mathias Soeken, Patrick Vuillod, Jiong Luo, Alan Mishchenko, Pierre-Emmanuel Gaillardon, Janet Olson, Robert Brayton, and Giovanni De Micheli. 2017. Enabling exact delay synthesis. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 352–359.

[2] Luca Amarú, Mathias Soeken, Patrick Vuillod, Jiong Luo, Alan Mishchenko, Janet Olson, Robert Brayton, and Giovanni De Micheli. 2018. Improvements to boolean resynthesis. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 755–760.

[3] Luca Amarú, Patrick Vuillod, Jiong Luo, and Janet Olson. 2017. Logic optimization and synthesis: trends and directions in industry. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 1303–1305.

[4] Luca Benini, Patrick Vuillod, and Giovanni De Micheli. 1998. Iterative remapping for logic circuits. *IEEE transactions on computer-aided design of integrated circuits and systems*, 17, 10, 948–964.

[5] Alessandro Tempia Calvino and Giovanni De Micheli. 2024. Scalable logic rewriting using don't cares. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[6] Alessandro Tempia Calvino, Alan Mishchenko, Herman Schmit, Ethan Mahintorabi, Giovanni De Micheli, and Xiaoqing Xu. 2023. Improving standard-cell design flow using factored form optimization. In *Proc. DAC*. DOI: 10.1109/DAC56929.2023.10247905.

[7] Satrajit Chatterjee. 2007. *On algorithms for technology mapping*. University of California, Berkeley.

[8] Satrajit Chatterjee, Alan Mishchenko, Robert K Brayton, Xinning Wang, and Timothy Kam. 2006. Reducing structural bias in technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25, 12, 2894–2903.

[9] Vasek Chvatal. 1979. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4, 3, 233–235.

[10] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. Asap7: a 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53, 105–115. DOI: https://doi.org/10.1016/j.mejo.2016.04.006.

[11] Andrea Costamagna, Alan Mishchenko, Satrajit Chatterjee, and Giovanni De Micheli. An enhanced resubstitution algorithm for area-oriented logic optimization. Accepted at the *International Symposium On Circuits And Systems (ISCAS)*, (2024).

[12] Robert H Dennard, Jin Cai, and Arvind Kumar. 2018. A perspective on today's scaling challenges and possible future directions. In *Handbook of Thin Film Deposition*. Elsevier, 3–18.

[13] Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. 1974. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of solid-state circuits*, 9, 5, 256–268.

[14] Gary D Hachtel and Fabio Somenzi. 2005. *Logic synthesis and verification algorithms*. Springer Science & Business Media.

[15] Ajey P Jacob, Ruilong Xie, Min Gyu Sung, Lars Liebmann, Rinus TP Lee, and Bill Taylor. 2017. Scaling challenges for advanced cmos devices. *International Journal of High Speed Electronics and Systems*, 26, 01n02, 1740001.

[16] Jie-Hong R Jiang and Robert K Brayton. 2004. Functional dependency for verification reduction. In *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings 16*. Springer, 268–280.

[17] Lech Józwiak. 1999. Information relationships and measures in application to logic design. In *Proceedings 1999 29th IEEE International Symposium on Multiple-Valued Logic (Cat. No. 99CB36329)*. IEEE, 228–235.

[18] Victor N Kravets and Prabhakar Kudva. 2004. Implicit enumeration of structural changes in circuit optimization. In *Proceedings of the 41st Annual Design Automation Conference*, 438–441.

[19] Siang-Yun Lee and Giovanni De Micheli. 2023. Heuristic logic resynthesis algorithms at the core of peephole optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[20] Siang-Yun Lee, Heinz Riener, Alan Mishchenko, Robert K Brayton, and Giovanni De Micheli. 2021. A simulation-guided paradigm for logic synthesis and verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41, 8, 2573–2586.

[21] Yingjie Li, Mingju Liu, Mark Ren, Alan Mishchenko, and Cunxi Yu. 2024. Dag-aware synthesis orchestration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[22] Shiju Lin, Jinwei Liu, Tianji Liu, Martin DF Wong, and Evangeline FY Young. 2022. Novelrewrite: node-level parallel aig rewriting. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 427–432.

[23] Frédéric Mailhot and Giovanni De Micheli. 1990. Technology mapping using boolean matching and don't care sets. In *EURO-DAC*. Vol. 90, 212–216.

[24] Giovanni De Micheli. 1994. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education.

[25] A. Mishchenko, S. Chatterjee, and R. Brayton. 2006. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In *Proc. DAC*.

[26] Alan Mishchenko and Robert Brayton. 2006. Scalable logic synthesis using a simple circuit structure. In *Proc. IWLS*. Vol. 6, 15–22.

[27] Alan Mishchenko, Robert Brayton, Stephen Jang, and Victor Kravets. 2011. Delay optimization using SOP balancing. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 375–382.

[28] Alan Mishchenko, Robert Brayton, Jie-Hong R Jiang, and Stephen Jang. 2011. Scalable don't-care-based logic optimization and resynthesis. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 4, 4, 1–23.

[29] Alan Mishchenko, Roland Jiang, Satrajit Chatterjee, and Robert Brayton. 2004. Fraigs: functionally reduced and-inv graphs. In *International Conference on Computer Aided Design*.

[30] Yukio Miyasaka. 2024. Transduction method for aig minimization. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 398–403.

[31] Gianluca Radi, Alessandro Tempia Calvino, and Giovanni De Micheli. 2024. In medio stat virtus: combining boolean and pattern matching. In *29th Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[32] André Inácio Reis and Jody MA Matos. 2018. Physical awareness starting at technology-independent logic synthesis. *Advanced Logic Synthesis*, 69–101.

[33] Tsutomu Sasao and Jon Butler. 2022. *Progress in Applications of Boolean Functions*. Springer Nature.

[34] Subarnarekha Sinha. 2002. *SPFDs: A new approach to flexibility in logic synthesis*. University of California, Berkeley.

[35] Petr Slavík. 1996. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 435–441.

[36] Mathias Soeken et al. 2018. The epfl logic synthesis libraries. *arXiv preprint arXiv:1805.05121*.

[37] Vijay V Vazirani. 2001. Approximation algorithms. (2001).

[38] Zile Wei, Donald Chai, Andreas Kuehlmann, and A Richard Newton. 2006. Fast boolean matching with don't cares. In *7th International Symposium on Quality Electronic Design (ISQED'06)*. IEEE, 6–pp.

[39] Yu-Shen Yang, Subarna Sinha, Andreas Veneris, and Robert K Brayton. 2007. Automating logic rectification by approximate SPFDs. In *2007 Asia and South Pacific Design Automation Conference*. IEEE, 402–407.

[40] Jin S Zhang, Subarna Sinha, Alan Mishchenko, Robert K Brayton, and Malgorzata Chrzanowska-Jeske. 2005. Simulation and satisfiability in logic synthesis. *computing*, 7, 14.

Received 12 July 2024; accepted 8 September 2024