Scalable Sequential Logic Synthesis Using Observability Don't Care Conditions

Dewmini Sudara Marakkalage*, Eleonora Testa†, Giulia Meuli‡, Walter Lau Neto†, Alan Mishchenko§, Giovanni De Micheli*, and Luca Amarù†
* Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
† Synopsys Inc., Design Group, Sunnyvale, California, USA
‡ Synopsys Inc., Design Group, Agrate Brianza, Lombardy, Italy
§ Department of EECS, University of California, Berkeley, USA
Emails: dewmini.marakkalage@epfl.ch, eleanora.testa@synopsys.com, giulia.meuli@synopsys.com, walter.launeto@synopsys.com, alanmi@berkeley.edu, giovanni.demicheli@epfl.ch, luca.amaru@synopsys.com

***Abstract*—Sequential logic synthesis expands the solution space compared to combinational logic synthesis by reasoning about the reachable states of memory elements, leading to better Power-Performance-Area (PPA) outcomes. As gate costs continue to rise in advanced technologies, sequential logic synthesis is gaining significant traction within the EDA community as a powerful alternative. This paper introduces a scalable algorithm for don't-care-based sequential logic synthesis, leveraging sequential k-step induction to perform redundancy removal and resubstitution under Sequential Observability Don't Cares (SODCs). SODCs generalize Observability Don't Cares (ODCs) by explicitly considering reachable states, making SODC-based optimization a challenging problem due to dependencies and alignment issues between the base case and inductive case in k-step induction. Our approach overcomes these challenges, fully utilizing the potential of SODCs without limiting the solution space. We rigorously prove the correctness of our approach, discuss some limitations arising from bounded-step induction, and analyze how our approach can effectively be used in practice to exploit obscure optimization opportunities. Implemented as part of an industrial tool, our algorithm achieves an average -6.9% area improvement after technology mapping compared to state-of-the-art sequential synthesis methods, and further provides 3.16% and 1.06% reductions in combinational and sequential areas, respectively, in post place-and-route results. Furthermore, all optimizations are efficiently verified using industrial sequential verification tools.**

***Index Terms*—Sequential redundancy removal, sequential circuits, observability don't cares, sequential k-step induction**

I. Introduction

Logic synthesis optimizes logic networks under various metrics, such as area, power, and delay. It plays a crucial role in modern *Electronic Design Automation* (EDA) flows. Combinational logic synthesis focuses on optimizing logic networks while maintaining combinational equivalence. Even if a logic network has sequential elements, combinational logic synthesis can still be applied by treating the register inputs/outputs as primary outputs/inputs, ignoring any constraints on reachable states.

Sequential logic synthesis, in contrast, specifically targets the optimization of logic networks with sequential elements. Since it can account for the fact that not all register value combinations are reachable, it offers a more powerful form of logic synthesis. It is well-known that sequential logic synthesis explores a broader solution space and generally achieves better *power, performance, and area* (PPA) [1]. These PPA benefits become increasingly critical as the cost of chip design continues to rise [2].

Sequential logic synthesis has been studied in the past considering various approaches [1], [3]–[10]. One such approach is to integrate combinational optimizations together with retiming [4], [11], which can exploit optimization opportunities arising due to structural properties across register boundaries [8]. The key idea behind this line of work is to move registers across combinational logic while optimizing the resulting combinational logic segments with existing combinational optimization techniques. A powerful yet scalable state-of-the-art approach is given by the *sequential SAT-sweeping* (SSW) algorithm from [1]. The basic idea of SSW is to merge sequentially equivalent nodes, where it uses Bounded Model Checking (BMC), Boolean Satisfiability (SAT), and sequential induction [12]–[14] to prove the validity of such merge candidates.

In this work, we present a novel scalable algorithm for don't-cares-based sequential logic synthesis. Our approach, by design, can work with *dependencies* among observability don't cares (ODCs), which is a challenging problem that has not been addressed in prior approaches. Our method is based on sequential induction and is orthogonal to the approach presented in [1]. In fact, our new method integrates both redundancy removal and resubstitution, leveraging *sequential observability don't cares* (SODCs). Moreover, our method can be enhanced to work with multi-step induction and use of assumptions to detect additional optimization opportunities. The latter is a version of sequential induction where the validity of candidate logic transformations in the inductive case are verified assuming they are already present in prior frames.

In combinational logic synthesis, ODCs—i.e., input patterns where the value of a wire is not observed at the outputs—can be utilized to uncover better optimization opportunities [15]–[20]. However, employing ODCs for optimization poses inherent challenges, as the remaining ODCs can change after applying an ODC-based optimization. These challenges are even more

pronounced in sequential optimizations, where it is particularly difficult to account for the sequential nature of circuits while managing ODC dependencies. (It is worth noting that applying an ODC-based optimization can also alter the set of reachable states.) Although the dependency issues can be mitigated by restricting the analysis to *compatible ODCs* (CODCs) [21], i.e., ODCs that can be used independently at each node, this approach can result in missed optimization opportunities.

Nevertheless, the method we propose inherently handles ODC dependencies, and hence utilizes the full power of ODCs in sequential optimizations. This is achieved by using an inductive approach that takes the reachable states into account and performs simultaneous, in-place optimization of two networks (base case and inductive case) using ODC-based combinational optimization methods that are built on Boolean Satisfiability (SAT) [22]. To make it scalable, our approach uses windowing so that the SAT-problem sizes remain manageable.

The simultaneous optimization of the derived networks enables our method to naturally identify valid ODC-based sequential optimizations that are compatible with one another, without restricting the search space to CODC-based optimizations. As a result, it achieves superior optimizations, particularly in circuits with sequential feedback, which can pose challenges for traditional retiming-based techniques. In essence, our method uncovers sequential optimizations that were previously unexplored by other approaches, while maintaining scalability. Additionally, since our approach avoids moving registers over combinational logic, the verification of the optimized networks is more likely to succeed, making it more suitable for use in industrial tools. In contrast, retiming-based methods often present challenges for verification tools due to the potential loss of anchor points essential for verification.

While an initial version of this work was presented in [23], this paper extends the work in several aspects.

1) Building upon the proof outline presented in [23], we provide the complete proof of correctness which is applicable to the multi-step induction as well as the setting with the use of assumptions.
2) We provide a detailed analysis of some obscure SODC-based optimization opportunities and explain how such cases can be effectively addressed in practice.
3) We expand the experimental evaluation of our approach, comparing different settings of the overall algorithm. Namely, we compare and contrast the effects of redundancies/resubstitutions, multi-step/single-step induction, enabling/disabling assumptions, and the use of small/large window sizes.
4) Integrating our approach to the full optimization flow of an industrial tool, we also present post place & route optimization results for industrial designs.

Through optimization of technology independent logic, we have demonstrated that each of the extensions improves the quality of results compared to the base version. Moreover, our approach is shown to achieve a 6.9% average reduction in area after technology mapping on top of state-of-the-art sequential optimization methods (e.g., SSW), with a 3.16% reduction of combinational area and a 1.06% of sequential area post place & route on industrial designs. All designs were verified using state-of-the-art sequential verification tools [24].

The organization of the paper is as follows: Section II discusses some background and relevant prior work. Section III presents a motivating example for sequential synthesis with ODCs followed by our novel scalable sequential logic synthesis approach, together with the proofs of correctness for different versions of the algorithm. Section IV presents our experimental results, and finally, Section V concludes with a brief discussion of the results and future work.

II. Background

In this section, we provide some background that will be useful to better understand the rest of the paper and briefly describe some state-of-the-art prior work in sequential synthesis.

A. Boolean Network

A Boolean network is a *directed acyclic graph* (DAG) representation of a logic network where the nodes correspond to logic gates and edges represent the connections between gates. A node function can be arbitrary, and usually encoded using its *sum-of-products* (SOPs) representation or its truth table with respect to node inputs. The source nodes of the DAG correspond to *primary inputs* (PIs) or *register outputs* (ROs) while the sinks correspond to *primary outputs* (POs) or *register inputs* (RIs). The corresponding RI/RO pairs are usually stored in a separate data structure together with the respective initial values of the registers. The fanins (fanouts) of a node n refer to the set of nodes that drives (driven by) n. I.e., the fanins of n have directed edges from them to n and the fanouts have directed edges from n to them. The *transitive fanin* (TFI) cone of a node n is the set of all nodes from which n is reachable via a directed path. Similarly, the *transitive fanout* (TFO) cone is the set of all nodes reachable from n via a directed path.

B. Sequential Logic Optimizations

In this section, we briefly introduce two important concepts that we use in our proposed optimization approach: *sequential redundancy* and *sequential resubstitution*.

1) Sequential Redundancy: In logic synthesis, a redundancy is a node or a wire whose value is stuck at a constant in all observable input (PI/RO) patterns. A redundant wire can be optimized away by removing the origin node of the wire from the fanin set of the destination node and modifying the destination node's function accordingly. A node is redundant if all its outgoing wires are redundant. Sequential redundancy is a generalization of a redundancy where the stuck-at-constant property holds considering all *observable input patterns* and *reachable states*.

2) Sequential Resubstitution: In logic synthesis, resubstitution refers to replacing a node n with a different node $m \neq n$. In general, m can be any existing node that is not in the TFO cone of n, or it can be a new node constructed by combining several other non-TFO nodes (called divisors). The Boolean

resubstitution refers to equivalence preserving resubstitutions that are computed considering Boolean properties such as *don't cares* (DCs). Additionally, when the implicit restrictions on reachable states of a sequential logic network are considered during resubstitution, we refer to it as sequential resubstitution.

C. Don't Cares in Logic Networks

In logic synthesis, a *don't care* (DC) is an input pattern (i.e., similar to a minterm) for which the output value of a node is not important [25]. Such patterns can be specified either with respect to the primary inputs or any cut of the node. Note that the latter is a generalization of the former since the set of PIs is a valid cut for any node. Don't cares have been extensively used in logic optimizations [26]–[28].

There are different types of don't cares such as *Controllability Don't Care* (CDC) and *Observability Don't Care* (ODC), which are described below. Note that, in addition to CDC and ODCs, there is also the notion of *Satisfiability Don't Care* (SDC) [29] which are Boolean value combinations that never occur considering an internal wire. SDCs are primarily used in the exhaustive computation of ODCs and CDCs.

1) Controllability Don't Cares: The CDCs for a node n with respect to a cut I are the Boolean value combinations for I which are impossible to occur. When the considered cut contains some internal nodes, CDCs can occur due to the structure of the network. For example, if a node n as two inputs x and y where $x = a \wedge b$ and $y = a \vee b$, the value combination $x = 1$ and $y = 0$ can never occur. Thus, considering the cut $\{x, y\}$, the value combination $x \wedge \neg y$ is a CDC for n. (Consequently, if n computes $x \wedge \neg y$, a CDC based optimization algorithm might optimize it away and replace it with the constant 0.) A logic network might also have some impossible PI patterns due to external constraints, and such patterns are called *external CDCs*.

2) Observability Don't Cares: The ODCs for a node n with respect to a cut I are the Boolean value combinations for I for which the output of n is not observed at any PO. For example, in Fig. 1 (a), consider the node w_1 with respect to the cut $\{a, b, d\}$. For any pattern where $a = 1, b = 0$, w_1 is not observed at the output o_1 because the output gate's second fanin, g_2, will be 0 under such a pattern. Thus $a \wedge \neg b$ is an ODC for w_1.

The ODCs of nodes in a logic network can have dependencies among themselves. Namely, if an optimization with respect to an ODC is performed for a particular node, it can change the ODCs of other downstream nodes, and hence, ODCs will have to be recomputed for those nodes. This added complexity can be avoided by using a less powerful version of ODCs, called *Compatible ODCs* (CODCs) [21], [30], that do not have dependencies among themselves. In other words, CODC-based optimizations are a subset of ODC-based optimizations that can be applied in parallel, without interfering with each other.

D. Prior Work on Sequential Synthesis

A common optimization approach in early works on sequential synthesis is to use retiming together with logic transformations [8], [10]. In this approach, first, the registers are moved around, then the resulting circuit is optimized using combinational methods, and finally retiming is performed again to minimize the register count. During retiming, if some reconvergent paths have varying numbers of registers, the usual practice is to remove such paths by duplicating the shared nodes, considering small blocks of logic. In contrast, our SODC-based approach does not move registers, thus it avoids the duplication requirement of shared logic. Moreover, our approach scales well to much larger logic blocks.

Another prominent sequential optimization method is *sequential SAT-sweeping (SSW)*, which is a generalization of combinational SAT-sweeping [31], [32] to the sequential setting, where the idea is to merge sequentially-equivalent nodes. If two nodes m and n are equivalent under all observable input patterns of n in all reachable states, n can be merged with m by transferring the fanouts of n to m, without changing the overall output function of the network. An efficient SSW algorithm is proposed in [1] where the sequential equivalences among nodes are proven using *bounded model checking* (BMC) [33] and SAT together with induction [12]–[14]. Once the equivalence classes are identified, all nodes in a class are merged into a chosen representative node and the dangling nodes are removed. Despite its practical success, SSW misses many optimizations made possible due to SODCs. Notably, SSW cannot optimize the simple sequential logic network in Fig. 2(a) into the one in Fig. 2(b).

Additionally, Case et al. [7] considered a simulation-based approach to find merge candidates considering ODCs. Namely, the network is simulated with random bit patterns to identify node pairs a, b such that for each simulated pattern, either a and b are equal or all paths from b to combinational outputs are non-controlling. Then a new network is created with all candidates merged, the equivalence of the new and original network is proven/disproven using SAT, and if disproved, the merge candidates are refined. However, this approach does not scale well to large networks due to large miters used in equivalence checking and hence misses many optimization opportunities. In contrast, we use a window-based approach and check the validity of each optimization in isolation; hence the SAT-based validity checks are scalable.

The method we propose in the next section is able to find optimizations that were never found by the prior approaches.

III. Scalable Sequential Optimization

In this section, we first give a brief motivation for our proposed method and introduce sequential induction. Then, we discuss our novel sequential optimization approach in detail with a formal proof of correctness. Lastly, we analyze some of the limitations of the proposed method and possible workarounds.

A. Motivation

Consider the purely combinational logic network shown in Fig. 1(a) and observe that the wires g_1 and g_2 can never be 1 at the same time. This implies that whenever $g_2 = 1$, g_1 must be 0. Since w_2 is observed at output o_1 only when $g_2 = 1$,

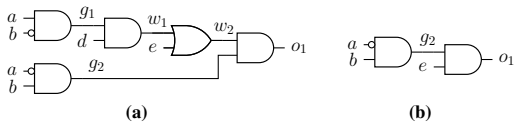

Fig. 1. A combinational logic network (a) and its optimized version (b).

one can simplify the circuit by assuming that g_1 is stuck at 0, which, in turn, implies that w_1 is also stuck at 0. This leads to the optimized circuit in Fig. 1(b).

Now consider the sequential circuit in Fig. 2(a) which is similar to the one in Fig. 1(a) except for the two registers at g_1 and g_2. If we consider this as a combinational network (i.e., disregard the registers, consider g_1, g_2 to be POs, and consider lo_1, lo_2 to be PIs), the previous reasoning no longer applies; lo_1, lo_2 can take arbitrary values, and hence lo_1 is observed even when it is 1. However, if we additionally know that the initial values of lo_1, lo_2 are $(0, 0)$ (or any combination of values different from $(1, 1)$), the optimization is still possible. This is because, by design, lo_1 and lo_2 can never be 1 at the same time in the subsequent clock cycles. This observation yields the optimized circuit shown in Fig. 2(b). The state-of-the-art sequential optimization routines such as *scorr*, *lcorr*, *scl*, and *retime* of the logic synthesis tool ABC [34] are unable to find this optimization. This is because the existing routines fail to consider the reachability of states in conjunction with the observability don't care conditions.

The goal of the proposed method is to identify this kind of optimization opportunities in sequential logic networks in a scalable way. We remark that, while a retiming-based optimization method might be able to optimize the example above, such methods perform poorly especially when there is sequential feedback (e.g., finite state machines) or varying numbers of registers along different reconvergent logic paths.

B. Sequential ODCs

The optimization in the example holds due to two facts:

1) (*Reachability*) Not all states (value combinations for the sequential elements) are reachable.
2) (*Observability*) In all reachable states, the optimization is valid due to the ODCs.

These two facts together form a notion of *sequential ODCs* (SODCs), a generalization of ODCs in combinational logic networks into the sequential setting.

Recall that a sequential network can be optimized by considering it as a combinational network, using combinational synthesis algorithms, where the register inputs are considered as primary outputs and the register outputs are considered as primary inputs. In this setting, suppose that the complete set of unreachable states are somehow known beforehand. Then, we can input such unreachable states as external CDCs, and use an external-CDC-based synthesis algorithm to optimize the network, and this would effectively perform sequential synthesis. The ODCs that exists when such unreachable states are considered as external CDCs are called SODCs.

In practice, the set of unreachable states is not known beforehand; but are implicitly defined by the structure of the network and the initial states. Thus, to consider SODCs in optimizations, an algorithm has to somehow reason about the reachable/unreachable states, and completely characterizing the set of SODCs is a computationally hard problem. In what follows, we present a framework that can be used to approximate SODCs of sequential networks.

C. Framework Definition

To use SODCs in optimization, we first take the *reachability* of states into account. To this end, a widely used technique is to use the so-called sequential induction [12], [13] which leverages two combinational networks called the base case network and the inductive case network that are obtained using k-step unrolling as defined below in Definition 1.

Definition 1 (k-Step Unrolling)**.** *For a sequential logic network N, the k-step base case network N^b is the combinational network obtained by*

1) *taking k copies of N (referred to as frames),*
2) *connecting the RIs of each frame to the corresponding ROs of the subsequent frame,*
3) *replacing the ROs of the first frame with the respective register initial values, and*
4) *designating RIs of the last frame as POs.*

The k-step inductive case network N^i is similarly defined except with the following changes:

1) *it has $k + 1$ frames, and*
2) *the ROs of the first frame are designated as PIs.*

For the example network of Fig. 2(a), the base case and the inductive case networks for 1-step (i.e., for $k = 1$) sequential induction are shown in Fig. 2(c) and Fig. 2(d) respectively. Note that in all figures, wires between frame inputs and gates are implicit, i.e., a frame input $x^{(i)}$ is connected to gate pins denoted by x. As shown in Fig. 2 (d),the RIs of the first frame (namely, li_1 and li_2) are connected to ROs of the second frame (namely, li_1 and li_2). In all frames, a, b, c, d, and e are PIs. The behavior of N^b is the same as that of the original network N for the initial k clock cycles and the behavior of N^i is the same as $k + 1$ consecutive clock cycles of N for any initial state. As formally stated in Theorem 1, if an optimization is valid in all frames of the base case and the last frame of the inductive case, then it is a valid sequential optimization for the original network.

On top of the reachability criterion, we consider the *observability* to identify sequential optimization opportunities. It seems straightforward to consider the sets of ODC-based optimizations in the base case and inductive networks and then take the intersection of the two sets as the final set of optimizations. However, as discussed in Section I, this approach only works with CODCs which do not have dependencies among them. Unfortunately, using CODCs in place of ODCs leads to many missed optimization opportunities. The regular ODCs can have dependencies in them, and cause this simple

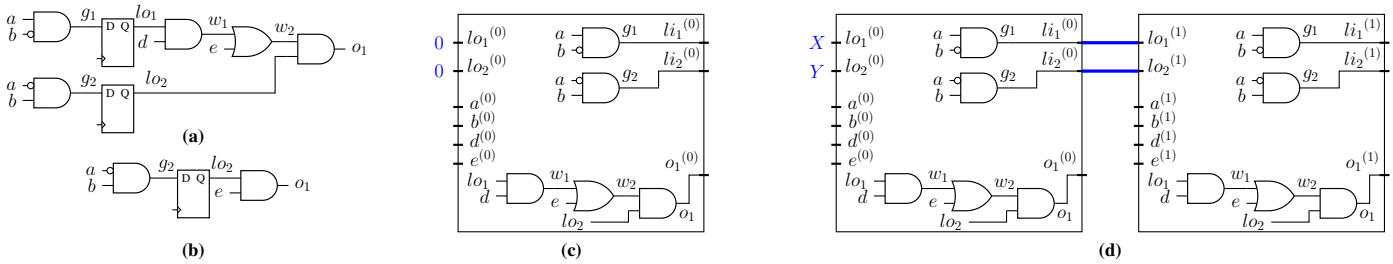

Fig. 2. A sequential logic network (a), its optimized version (b), and its base case network (c) and the inductive case network (d) for 1-step sequential induction.

algorithm to fail. In the remainder of this section, we present an algorithm that, by design, avoids dependency issues of regular ODCs without falling back to CODCs.

D. Proposed Method

Our proposed algorithm is based on sequential induction and it can fully utilize ODCs by simultaneously optimizing base case and inductive case networks.

Namely, we start by constructing the 1-step unrolled base case and inductive case networks for sequential induction. Then, considering one node at a time, we check if there is a valid optimization for that node in both the base case and the inductive case networks. If so, we immediately update both the derived networks as well as the original network by applying the optimization. This approach allows the algorithm to find subsequent optimizations for the remaining nodes that may depend on the already applied optimizations. Thus it avoids any dependency issues that would arise if we were to use the simple approach we stated at the end of Section III-C with regular ODCs. Hence, the algorithm computes compatible sequential optimizations without limiting to CODCs.

To find optimizations, the algorithm considers fanin redundancies for each gate n in the network. Namely, for each fanin f of n we check whether f is effectively stuck at 0 or 1 in

1) the the base case network, and
2) the last frame of the inductive case network.

Since both the base case and inductive case networks are purely combinational, it is possible to use any combinational redundancy check for this purpose. To this end, let n' be the node obtained by fixing fanin f of n at the target constant value. In our implementation, we check if replacing n with n' is valid using a SAT problem.

To formulate the SAT problem, we consider the miter for comparing the outputs of the derived network, with and without the candidate optimization. This miter is represented as a CNF formula, whose satisfiability is then checked with a SAT solver. Note that this formulation implicitly considers both observability don't cares and reachability constraints, i.e., SODCs, as explained at the end of Section III-E.

If the problem is unsatisfiable (UNSAT), then the optimization is valid. To make the overall algorithm scalable, we optimize the SAT formulation not to consider all POs and RIs, but instead consider the leaf nodes of a small TFO cone rooted at n. If the optimization is shown to be valid for both the base and inductive case networks, then we apply it in both the networks as well as in the original network (see Section III-E for details).

As an illustrative example, consider Fig. 2(a). We can prove, that w_1 is stuck at 0, in both the base case and the inductive case networks, which will result in the optimized network in Fig. 2(b) (assuming all registers are initially 0).

We consider three enhancements on our proposed method which enables it to find more optimization opportunities.

Enhancement 1: We extend our algorithm to use k-step sequential induction where the base case network has $k \geq 1$ frames and the inductive case network has $k+1$ frames as defined below:

In this case, we check if the target Δ redundancy is valid in

1) all k frames of the base case network, and
2) the last frame of the inductive case network.

If the considered redundancy is valid in both cases, then we apply it in all frames of the two derived networks as well as in the original network.

Fig. 3(a) shows an example sequential network which can be optimized to the one in Fig. 3(b) with 2-step sequential induction (assuming all registers are initially 0). In the last frame of the inductive network (Fig. 3(c)), lo_3, lo_4 are fed by the gates g_1, g_2 of the first frame, so lo_3, lo_4 of the last frame are never 1 at the same time. Thus the algorithm is able to prove that w_1 of the last frame is stuck at zero. In contrast, if 1-step induction were to be used, we only get the first two frames of Fig. 3(c), and the second frame's lo_3, lo_4 are driven by two arbitrary inputs from the first frame. Hence, all value combinations are possible, so w_1 of the second frame is not stuck at zero.

Enhancement 2: Our approach is not limited to fanin redundancies but also extends to resubstitutions under ODCs. Namely, for a considered node n, we consider a subset D (called divisors) of nodes that are not in the TFO cone of n. Then, we consider all versions of n obtained by replacing one of its fanins with one of the nodes in D as resubstitution candidates for n. As with the redundancies, for each resubstitution candidate n', we use SAT to check if some window output would differ when n is replaced with n'.

Enhancement 3: We further improve our method by considering redundancy assumptions in the base case and the inductive case networks. To elaborate, suppose that we found a valid optimization Δ for the first frame of the base case. Then,

we check whether Δ is also valid for the subsequent frames, *assuming* that the all preceding frames are already updated with Δ. Namely, for $i > 1$, we check if Δ is valid for frame i of the base case, assuming all frames $1, \ldots, i-1$ are transformed with Δ. Once the validity of Δ is confirmed in all frames of the base case network, update the first k frames of the inductive case network with Δ and check for its validity in the last frame. At any point, if we find Δ is not valid for the considered frame, we undo it in all previous frames.

Fig. 4(a) shows a simple sequential network with feedback whose output is always zero provided that the initial state of the register is zero. With assumptions, our proposed method is able to prove this. For the base case, it is clear that $g1$ is stuck at zero. For the inductive case (shown in Fig. 4(b)), if we assume g_1 of the first frame is stuck at zero, then so is g_1 in the second frame. *This is also an example of a sequential network that is not optimized by retiming-based methods.*

Note that the use of assumptions as an enhancement is not to be confused with the inductive hypothesis we use in our proofs. To elaborate, while we use mathematical induction to show the equivalence of the original network and the optimized network, the sequential induction is a construction we use to find eligible optimizations. In the context of enhancements, assumptions are considered in sequential induction. Namely, we assume the candidate optimization is already applied in the first k frames of the inductive case combinational network before checking its validity in the last frame. To formalize the difference between having assumptions and not having assumptions, we define the *validity of a candidate optimization Δ in the inductive case network* as follows: Let $N^i(\Delta, \{j_1, \ldots, j_t\})$ denote the inductive case network modified with transformation Δ applied in frames $j_1, \ldots, j_t$. When assumptions are not used, to check the validity of Δ in the inductive-case network, we check whether N^i is combinationally equivalent to $N^i(\Delta, \{k+1\})$. When assumptions are used, we check whether $N^i(\Delta, \{1, \ldots, k\})$ is combinationally equivalent to $N^i(\Delta, \{1, \ldots, k, k+1\})$.

E. Complete Algorithm

The high-level pseudocode of our method without assumptions (i.e., with the first and second enhancements above) is presented in Algorithm 1 whereas the Algorithm 2 shows the pseudocode considering all three enhancements.

In both our algorithms, we use the following definition of dangling registers:

Algorithm 1: High-level pseudocode of sequential optimization with k-step induction without assumptions.

```
input : Input network N. Number of frames k.
N^b ← N unrolled into k frames and first frame ROs replaced with initial states.
N^i ← N unrolled into k + 1 frames.
Let N^{b,j}, N^{i,j} denote the j-th frame of N^b, N^i respectively.
for each gate g ∈ N do
    for each candidate optimization Δ for g do
        if Δ is not valid in all frames of N^b then
            Continue loop.
        if Δ is invalid for g in N^{i,k+1} then
            Continue loop.
        Apply Δ in N^{b,1}, ..., N^{b,k}.
        Apply Δ in N^{i,1}, ..., N^{i,k+1}.
        Apply Δ in N.
Recursively remove all dangling registers and their MFFCs from N.
return N
```

Definition 2 (Dangling Registers)**.** *We say that a register r in a logic network N is non-dangling if there is a combinational logic path from the RO of r to either*

1) any PO, or
2) the RI of any other non-dangling register.

All remaining registers are called dangling registers.

In Algorithm 1, after constructing the two derived networks N^b and N^i, the gates of the input network are processed one at a time. For each gate, the algorithm iterates of candidate optimizations Δ and checks if Δ is valid for all frames of the base case network N^b. If so, it checks if Δ is also valid for the last frame of the inductive network N^i. If both checks succeed, the algorithm applies Δ in all frames of the two derived networks as well as in the original network N.

Algorithm 2 is an enhanced version of Algorithm 1 which additionally support temporary application, and if necessary, undoing of unconfirmed candidate optimizations. In Line 4 of Algorithm 2, the algorithm iterates over all gates in the original network, and in Line 5, it considers different optimization

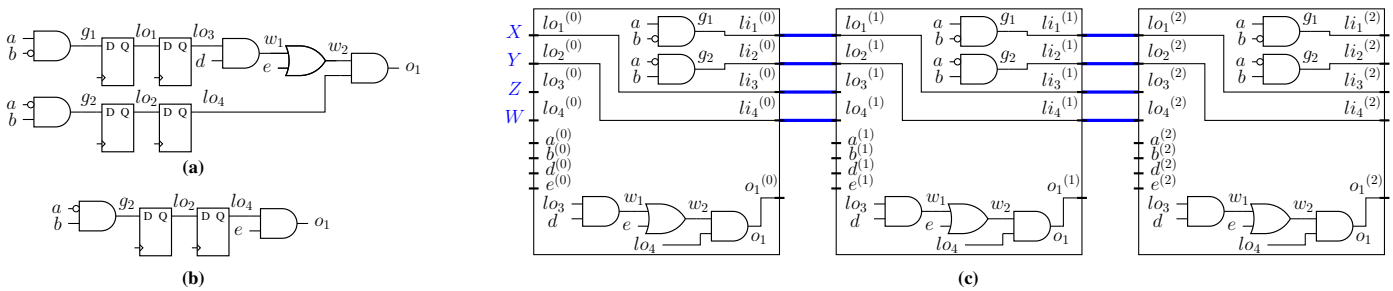

Fig. 3. A sequential logic network (a), its optimized version (b), and its inductive case network (c) for 2-step sequential induction.

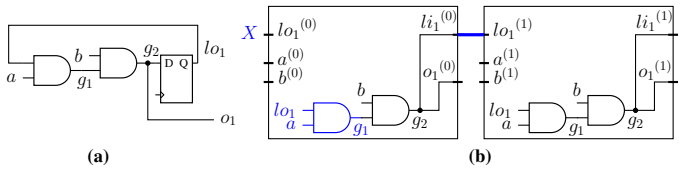

Fig. 4. A sequential logic network with feedback (a) and its inductive case network (b) for 1-step sequential induction before applying assumptions in the first frame.

Algorithm 2: High-level pseudocode of sequential optimization with k-step induction with assumptions.

input : Input network N. Number of frames k.
1 $N^b \leftarrow$ N unrolled into k frames and first frame ROs replaced with initial states.
2 $N^i \leftarrow$ N unrolled into $k+1$ frames.
3 Let $N^{b,j}, N^{i,j}$ denote the j-th frame of N^b, N^i respectively.
4 **for** *each gate* $g \in N$ **do**
5 **for** *each candidate optimization* Δ *for* g **do**
6 **for** $j = 1, \ldots, k$ **do**
7 **if** Δ *is invalid for* g *in* $N^{b,j}$ **then**
8 Undo Δ in all frames $N^{b,1}, \ldots, N^{b,j-1}$.
9 Continue outer loop.
10 Apply Δ in $N^{b,j}$.
11 Apply Δ in $N^{i,1}, \ldots, N^{i,k}$.
12 **if** Δ *is invalid for* g *in* $N^{i,k+1}$ **then**
13 Undo Δ in $N^{b,1}, \ldots, N^{b,k}$ and $N^{i,1}, \ldots, N^{i,k}$.
14 Continue loop.
15 Apply Δ in $N^{i,k+1}$.
16 Apply Δ in N.
17 Recursively remove all dangling registers and their MFFCs from N.
18 return N

candidates Δ. We use the letter Δ to denote a simple logic transformation such as a fanin redundancy or a resubstitution. Then, for each frame of the base case network, the algorithm checks if Δ is valid in that frame; if it is valid, then the algorithm applies Δ in that frame (Line 6-Line 10). If it is valid in all frames of the basecase network, then it applies Δ in the first k frames of the inductive case network (Line 11) and checks for the validity in the last frame (Line 12) of the inductive case network. If it is valid, the algorithm applies Δ in the last frame (Line 15) as well as in the original network (Line 16). At any point, if Δ is invalid for the considered frame, it undoes all preceding applications of it (Line 8 and Line 13).

This undoing is necessary to keep the two derived networks, N^b and N^i, consistent with the current version of the sequential network N. The consistency between the derived networks and the original network is crucial for the correctness of the algorithm.

In Lines 7 and 12, to check for the validity of a target optimization Δ, the algorithm first constructs a window around the target node in the respective network. (Note that the window is not restricted to the considered frame; to take the reachable states into consideration, the window should span to all previous frames in general.) Then it encodes the following as a SAT problem:

> *Is there an input pattern (for the window) that would make at least one output differ for the window with and without the candidate optimization?*

To put differently, we construct a miter to compare the window outputs, with and without the candidate optimization, the miter is represented as a CNF formula using the Tseytin transformation [35]. The satisfiability of the miter is then checked with a SAT solver.

If it is UNSAT, then the target optimization is valid. The size of the window and the conflict limit for the SAT solver are configurable parameters.

Our algorithm is able to find SODC-based optimizations instead of merely settling for ODC-based optimizations (or CODC-based optimizations, which are a subset of ODC-based optimizations). This is facilitated by two crucial aspects of our method: First, by considering unrolled networks over 2 or more clock cycles, the algorithm is able to conclude that certain state combinations are never reachable. To see this, note that the logic values of the wires corresponding to ROs of the last frame are not arbitrary, but are determined by the combinational logic in the previous frames. Second, the SAT formulation we use implicitly considers the observability of signals, as the miter is satisfiable only if the candidate optimization changes the at least one output of the network.

F. Correctness of the Proposed Approach

In this section, we show the correctness of our algorithms, both with and without the use of assumptions. To this end, we consider sequential networks in the following setting:

1) All sequential elements in the network are positive-edge-triggered D flip-flops,
2) PIs are set on the negative edge of the clock, and
3) The clock edge that immediately following the reset is a negative edge.

We label the positive clock edges that follows reset by non-negative integers $0, 1, 2, \ldots$. Unless explicitly mentioned otherwise, we use the following notations in our theorems and proofs: we denote the input sequential logic network by N, and assume that it has m PIs, n POs, and ℓ registers. Let For $x \in \mathbb{B}^m$ and $y \in \mathbb{B}^\ell$, let $\mathrm{PO}(N, x, y) \in \mathbb{B}^n$ and $\mathrm{RI}(N, x, y) \in \mathbb{B}^\ell$, respectively, denote the PO and RI values of N when PIs are set to x and register states (ROs) are set to y. We use $y_0 \in \mathbb{B}^\ell$ to denote the initial state of the registers. Let $\{x\}_0^T = x_0, x_1, \ldots, x_T$ where $x_i \in \mathbb{B}^m$ for all $i = 0, 1, \ldots, T$ be a sequence of Boolean vectors. Given a sequence $\{x\}_0^T$ and initial state $y \in \mathbb{B}^\ell$, suppose that registers are set to y at reset and that x_t is set as PI values at t-th negative clock edge. We observe the PO and RI values just before the T-th positive clock edge. Let $\mathrm{PO}(N, \{x\}_0^T, y) \in \mathbb{B}^n$ and $\mathrm{RI}(N, \{x\}_0^T, y) \in \mathbb{B}^\ell$, respectively, denote the resulting PO values and RI values.

For two networks N_1, N_2, we define $\mathrm{EQ}(N_1, N_2, T)$ to be the proposition that for any sequence of T PI vectors, the PO values of the two networks are the same, when started, respectively, from y^1 and y^2 as the initial state. Formally, $\mathrm{EQ}(N_1, N_2, T, y^1, y^2) :=$ for all sequences $\{x\}_0^T \in \mathbb{B}^{m\times T}$, $\mathrm{PO}(N_1, \{x\}_0^T, y^1) = \mathrm{PO}(N_2, \{x\}_0^T, y^2)$.

Similarly, we define $\mathrm{EQ}^\star(N_1, N_2, T, y)$ to be the proposition that for any sequence of T PI vectors, the PO and RI values of the two networks are the same, when started from y as the initial state. Formally, $\mathrm{EQ}^\star(N_1, N_2, T, y) :=$ for all sequences $\{x\}_0^T \in \mathbb{B}^{m\times T}$, $\mathrm{PO}(N_1, \{x\}_0^T, y) = \mathrm{PO}(N_2, \{x\}_0^T, y)$ and $\mathrm{RI}(N_1, \{x\}_0^T, y) = \mathrm{RI}(N_2, \{x\}_0^T, y)$. Note that, here, both networks are initialized with the same initial state y.

We now define the notion of sequential equivalence and strong sequential equivalence.

Definition 3 (Sequential Equivalence)**.** *Suppose that N_1 and N_2 are two sequential networks with m PIs and n POs, and suppose that N_1 has ℓ_1 registers and N_2 has ℓ_2 registers. Let $y_0^1 \in \mathbb{B}^{\ell_1}$ and $y_0^2 \in \mathbb{B}^{\ell_2}$ be the initial states of the registers of N_1 and N_2, respectively. We say that two sequential networks N_1 and N_2 are sequentially equivalent if $\mathrm{EQ}(N_1, N_2, T, y_0^1, y_0^2)$ is true for all $T \in \mathbb{N}_0$.*

When $\ell_1 = \ell_2$ and $y_0^1 = y_0^2$, we say that N_1 and N_2 are strongly sequentially equivalent if $\mathrm{EQ}^\star(N_1, N_2, T, y_0^1)$ is true for all $T \in \mathbb{N}_0$.

Our goal is to show that the output of Algorithm 2 is sequentially equivalent to the input network. Outline of our proof is as follows:

1) Show that the intermediate network we get before removing dangling registers, i.e., the network N just before Line 17, is sequential equivalent to the input network.
2) Show that removing the dangling registers in Line 17 does not affect the sequential equivalence.

For step 1 above, we in fact show the stronger result that the intermediate network is strongly sequentially equivalent to the input network. To this end, we first start with the following lemma, which essentially yields an alternative definition of sequential equivalence.

Lemma 1. *Let N_1, N_2 be two logic networks with the same initial state y_0, and let $S \subseteq \mathbb{B}^\ell$ be the set of all reachable states of N_1. If $\mathrm{PO}(N_1, x, y) = \mathrm{PO}(N_2, x, y)$ and $\mathrm{RI}(N_1, x, y) = \mathrm{RI}(N_2, x, y)$ for all $x \in B^m$ and $y \in S$, then N_1 and N_2 are strongly sequentially equivalent.*

Proof. The proof is relatively straightforward. We include it in the appendix for completeness. □

With Lemma 1, we now show that the strong sequential equivalence holds with respect to a single valid transformation Δ. Namely, we show the following lemma:

Lemma 2. *Consider a logic network N and its k-step base case and inductive versions N^b and N^i. Let Δ be a logic transformation and let $N_\Delta, N_\Delta^b, N_\Delta^i$, respectively, be the networks obtained by applying Δ to N, to all frames of N^b, and to the last frame of N^i. If N^b and N^i, respectively, are combinationally equivalent to N_Δ^b and N_Δ^i, then N and N_Δ are sequentially equivalent.*

Proof. Let S be the set of reachable states, and note that S can be decomposed as the countable union $S = S_0 \cup S_1 \cup S_2 \ldots$, where S_0 is the set with only the initial state, S_1 is the set of reachable states after the first clock cycle, S_2 is the set of reachable states after the second clock cycle, and so on.

We first claim the following: For $i = 0, 1, 2, \ldots$, for any $x \in \mathbb{B}^m$ and $y \in S_0 \cup S_1 \cup S_2 \ldots$, we have that $\mathrm{PO}(N, x, y) = \mathrm{PO}(N_\Delta, x, y)$ and $\mathrm{RI}(N, x, y) = \mathrm{RI}(N_\Delta, x, y)$.

To see this, first fix any $i \in \{0, \ldots, k-1\}$, and let $y \in S_i$. Let $x_0, \ldots, x_{i-1}$ be the sequence of PI vectors that resulted in state y after i clock cycles. Now, for networks N^b and N_Δ^b, set the j-th frame PIs to x_j for $j = 0, \ldots, i-1$, set i-th frame PIs to x, and set the remaining PIs arbitrarily (but same for both networks). By design of N^b, we must have its $i-1$-th frame RIs set to y, and by combinational equivalence, the same holds for N_Δ^b. Thus, considering the combinational equivalence for i-th frame POs and RIs, the claim above holds for i-th frame.

Now, fix any $i \geq k$, let $y \in S_i$ and let $x_0, x_1 \ldots x_{i-1}$ be the sequence of PI vectors that resulted in state y after i clock cycles as before. This time, we use the equivalence of N^i and N_Δ^i, and for this, we set ROs of the 0-th frame to state we get on sequence $x_0, \ldots, x_{i-k}$, and we set j-th frame PIs to x_{i-k+j} for $j = 0, \ldots, k-1$, and k-th frame PIs to x. Then, by the combinational equivalence of N^i and N_Δ^i, considering the last frame POs and RIs, we have that the claim holds for i-th frame. □

With Lemma 2, we now proceed to show that the input network of Algorithm 2 is sequentially equivalent to the output network.

Theorem 1. *Let N be the input network of Algorithm 1 and let $N^\star$ be the output network. Then N and $N^\star$ are sequentially equivalent.*

Proof. Let N^{int} denote the intermediate network produced by Algorithm 1 just before removing dangling registers, i.e., just before Line 17. Note that N^{int} is obtained from N by applying a sequence of valid transformations, and each transformation preserves the strong sequential equivalence due to Lemma 2. Thus N^{int} and N are strongly sequentially equivalent, which is a special case of being sequentially equivalent.

Observe that removing dangling registers and their MFFCs does not affect the value of any PO or any remaining non-dangling register. Thus, N^{int} and $N^\star$ are sequentially equivalent. Finally, since sequential equivalence is transitive by definition, we have that N and $N^\star$ are sequentially equivalent. □

G. Characterizing SODC-Optimizable Transformations

As proven above, our approach only finds correct SODC-based optimizations, so it has the soundness property. However, due to the limitations of sequential induction, the proposed algorithm is not complete, i.e., it is unable to prove all valid SODC-based fanin redundancies/resubstitutions.

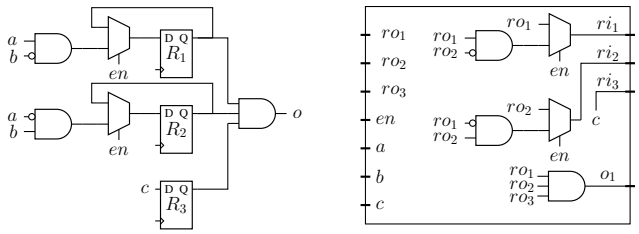

Fig. 5. An example sequential circuit and its frame representation.

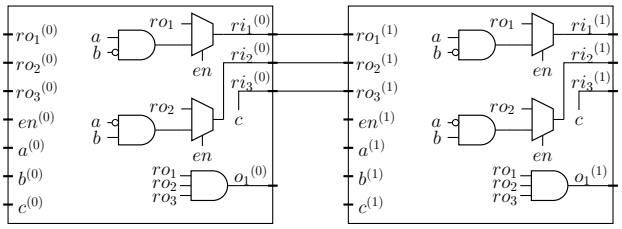

Fig. 6. The inductive network for the sequential circuit in Figure 5.

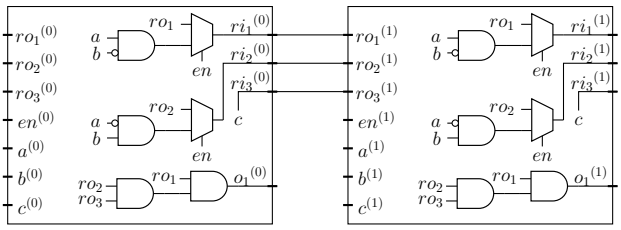

Fig. 7. The inductive network for the sequential network in Figure 5, where the decomposition of the 3-input AND is unfavorable to the proposed algorithm.

In the remainder of this section, we examine in what situations the proposed method can find the SODC-based optimizations and propose workarounds for situations where it may struggle to do so.

1) Networks without sequential feedback: Consider a sequential network with no sequential feedback. This means that for any register, its future (not necessarily the immediate next state) state does not depend its current state. For such a network N, let d be the sequential depth, which is the maximum number of registers in any path from a PI to a PO. If our proposed methods is used to optimize N using d-step sequential induction, then our method is capable of finding any valid SODC-based optimization. This is because, the reachable states approximated by the inductive case network will be the same as the actual reachable states after d clock cycles.

2) Networks with sequential feedback: In networks with sequential feedback, the future state of a register can depend on its current state. In such networks, the number of sequential induction steps required to characterize all reachable states can be exponential, and hence it is impractical. Moreover, unless assumptions are used, it may not be possible to find all valid SODC-based optimizations with sequential induction.

Consider the following example sequential circuit (Fig. 5) and its inductive case network for sequential induction (Fig. 6). Suppose that all registers are initially set to 0, and observe that R_1 and R_2 can never be 1 at the same time. Thus, the output o is stuck at 0.

We analyze under what conditions our proposed method is able to find the above optimization.

Note that in the 1-step inductive case network shown in Fig. 6, the ro_1 and ro_2 of the initial frame are considered as PIs, thus they can take arbitrary values. When en is 0, these values can propagate to the next frame, and if ro_1, ro_2, and c of initial frame are all 1 and $en = 0$, then the output o_1 of the last frame is 1. Thus, this setup is unable to prove the the stuck-at-0 property of o_1, unless we use assumptions. This remains the case even if use k-step induction with any $k > 1$ since the values of ro_1 and ro_2 of the initial frame can propagate to the last frame as long as en remain 0.

However, from Fig. 5, we clearly see that the values of both R_1 and R_2 can never be 1 at the same time if the initial states are 0. This is because, if R_1 and R_2 are not 1 in the current clock cycle, then they are not 1 in the next clock cycle as well. Thus, one would hope to find the optimization considering the assumptions. The challenge for our algorithm is to find the right assumptions to make.

In the proposed method, we consider a simplified set of assumptions. Namely, as assumptions, we always use the candidate redundancy (or resubstitution) property that we are trying to prove. Consequently, there arise cases where these simplified assumptions are not sufficient as-is to prove the property.

To illustrate, consider again the sequential network in Fig. 5 or a different version of the same network as shown in Fig. 7 where the 3-input AND gate is decomposed into two 2-input AND gates. Let us assume that o_1 is 0 in the initial frame.

With the goal of proving that o is stuck at 0 in the last frame, suppose that o is 0 in the initial frame. Unfortunately, the condition of $o_1 = 0$ in the initial frame does not prevent ro_1 and ro_2 from being 1 at the same time in the first frame, because $o_1 = 0$ is possible with $ro_1 = 1$, $ro_2 = 1$, and $ro_3 = 0$. Thus, with this assumption, the algorithm is unable to prove the stuck-at-0 property of o_1.

Alternatively, consider the case where the 3-input AND gate is decomposed into two 2-input AND gates in a different manner, as shown in Fig. 8. In contrast to the previous case, in this network, the algorithm is able to prove the stuck-at-0 property of o_1, by first proving that gate g_1 of the decomposition is stuck at 0. Namely, the algorithm first assumes that g_1 is stuck at 0 in the first frame. This implies that ro_1 and ro_2 are not 1 at the same time in the first frame. For the second frame, the value of g_1 is either AND of ro_1 and ro_2 values from the first frame, or its the and of $a \wedge \neg b$ and $\neg a \wedge b$, which can never be 1 at the same time.

Considering the two scenarios described above, it is clear that the proposed method's ability to find the SODC-based optimizations depends on the network structure. To mitigate this issue, we propose two approaches.

a) Equivalent Structural Transformations: One straightforward way to address the issue is to consider different equivalent structures for different portions of the network and different decompositions of complex gates. This will allow

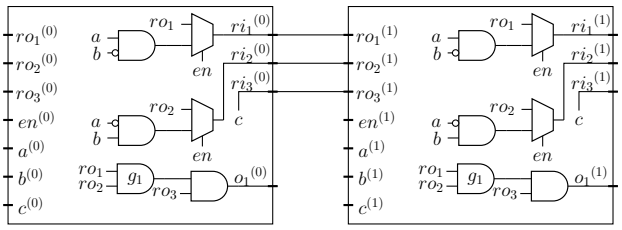

Fig. 8. The inductive network for the sequential network in Figure 5, where the decomposition of the 3-input AND is conducive to the proposed algorithm.

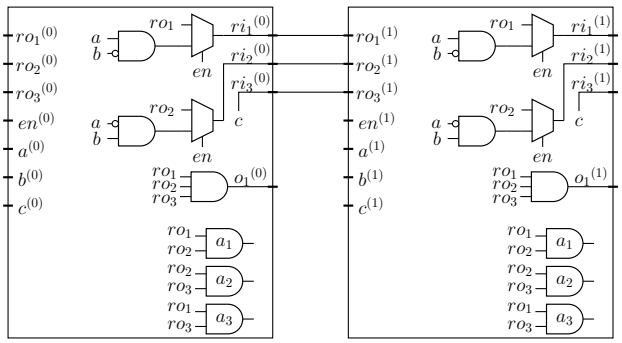

Fig. 9. The inductive network for the sequential network in Figure 5, where shadow nodes are added to reflect the assumptions on the reachable states of registers.

the algorithm to consider different anchor nodes and explore assumptions on them.

b) External Assumptions: The other approach is to expand the possible types of assumptions used by the algorithm. The current method uses the candidate redundancy property as the only assumption. However, this is strictly not necessary, and the algorithm may consider any arbitrary assumption on the reachable states of registers. Such general assumptions can be proven by the current algorithm, simply by introducing additional shadow nodes to serve as anchors for the assumptions.

To elaborate on external assumptions, suppose that the algorithm is faced with the network in Fig. 7. From the initial conditions, we know that o_1 is stuck at zero. If no decomposition is preformed, the algorithm will again construct the inductive network as shown in Fig. 6. However, instead of assuming that the 3-input AND gate of the first frame gate is stuck at 0 and then trying to prove the same for the second frame, the algorithm may perform the following steps:

1) Find the registers on which the value of o_1 depends. This can be done by traversing the transitive fanin cone of o_1. (In general, such registers can be efficiently computed for all gates, using a topological traversal from inputs to outputs.) In this case, the registers are ro_1, ro_2, and ro_3.
2) Explore different subsets of value combinations for these registers which would make o_1 stuck at 0. The number of such subsets grows doubly-exponentially with the number of registers (e.g., for r registers, there are 2^r possible states, so there are 2^{2^r} many subsets of states). Since exploring all possible subsets is impractical, the algorithm may use heuristics to select a subset of states to explore. For example, the algorithm may choose states where a pair of registers are not both 1 at the same time.
3) For each considered subset S of states, construct a new temporary node n_S in the network which is 0 if and only if the register value combination is in S. E.g., for o_1 of the network in Fig. 5, the algorithm may create new temporary AND gates $ro_1 \wedge ro_2$, $ro_2 \wedge ro_3$, and $ro_2 \wedge ro_3$ as shown in Fig. 9. The gate a_1 denotes the assumption that the state $(ro_1 = 1, ro_2 = 1, ro_3 = *)$ is not reachable, gate a_2 denotes the assumption that the state $(ro_1 = 1, ro_2 = *, ro_3 = 1)$ is not reachable, and gate a_3 denotes the assumption that the state $(ro_1 = *, ro_2 = 1, ro_3 = 1)$ is not reachable.
4) Assuming that n_S is stuck at 0 in the first frame, check whether the n_S is stuck at 0 in the second frame. If so, assuming that n_S is stuck at 0 in the second frame, check whether the n_S is stuck at 0 in the third frame, and so on. If n_S is stuck at 0 in all frames, then conclude that the assumption is valid, and hence o_1 is stuck at 0 in all reachable states.

Once the algorithm has proven the stuck at 0 property of o_1, it proceeds to remove the redundant gates and the shadow nodes.

To summarize, while the proposed algorithm always finds correct SODC-based optimizations, the limited horizon of sequential induction and the implementation choices for accommodating assumptions can lead to missed optimization opportunities. These drawbacks can be mitigated by using symmetry-breaking logic transformations and adding support for more elaborate assumptions.

IV. Experimental Results

In this section, we present and discuss the experimental results obtained using the proposed approach which is implemented as part of a commercial EDA tool. For the evaluations, we consider a subset of OpenCores [36] and some industrial designs as the sequential benchmark design suite.

We proceed in three steps. First, we compare the effects of different configurations of our algorithm by optimizing technology-independent logic, using And-Inverter Graphs (AIGs) as the logic representation. Next, we evaluate the performance of the model on technology-mapped designs, and finally we present the results obtained on industrial designs after place and route.

A. Comparison of Different Configurations

Recall that, on top of our base algorithm, we also proposed several extensions, namely, multi-step sequential induction, assumptions, and support for resubstitutions. To compare the effectiveness of these extensions, we optimize a subset of OpenCores designs using different configurations of our algorithm.

As the baseline for comparisons, we consider a state-of-the-art sequential optimization flow [1] together with combinational rewriting (commands *scorr* and *rewrite* in ABC [34]). The two optimizations are interleaved and run until saturation, i.e., no further reduction is observed. In the experimental flow, we additionally run our algorithm (without and with respective extensions) on top of the baseline.

TABLE I
COMPARISON OF THE PROPOSED METHOD AGAINST THE BASELINE (REDUNDANCY REMOVAL VS. REDUNDANCY REMOVAL + RESUBSTITUTION)

	Baseline			**Our Method (redundancy removal)**					**Our Method (redundancy removal + resub)**				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2%	NAND2	Lev	FF	Time (s)	NAND2%
aes_core	22026	32	530	21576	32	530	22.7	-2.04	21535	32	530	49.2	-2.23
des_area	4611	37	64	4611	37	64	0.7	0.00	4611	37	64	1.7	0.00
des_perf	77288	23	8808	76691	23	8808	302.7	-0.77	76534	23	8808	592.6	-0.98
ethernet	168	13	47	166	13	47	0	-1.19	166	13	47	0	-1.19
i2c	931	24	126	891	24	126	0.1	-4.30	891	24	126	0.3	-4.30
mem_ctrl	7097	31	1050	7003	31	1050	1.9	-1.32	6998	31	1050	3.7	-1.39
pci_bridge32	17656	32	3198	17403	32	3198	10.2	-1.43	17403	32	3198	20.9	-1.43
pci_spoci_ctrl	704	20	60	678	20	60	0.2	-3.69	674	20	60	0.5	-4.26
sasc	597	10	117	568	10	117	0.1	-4.86	568	10	117	0.1	-4.86
simple_spi	779	12	131	772	12	131	0.1	-0.90	772	12	131	0.2	-0.90
spi	3621	31	229	3590	31	229	0.5	-0.86	3592	31	229	0.9	-0.80
ss_pcm	464	9	87	399	9	87	0	-14.01	399	9	87	0	-14.01
steppermotordrive	138	17	25	125	17	25	0	-9.42	125	17	25	0	-9.42
systemcaes	11106	42	670	11105	42	670	4.3	-0.01	11105	42	670	8.2	-0.01
systemcdes	2696	36	190	2692	34	190	2	-0.15	2687	34	190	5.7	-0.33
tv80	7740	58	359	7553	58	359	2.5	-2.42	7527	58	359	6.2	-2.75
usb_funct	13910	27	1722	13560	26	1721	8	-2.52	13557	26	1721	16.8	-2.54
usb_phy	457	12	98	407	11	98	0	-10.94	407	11	98	0.1	-10.94
vga_lcd	89555	27	17032	89392	27	17032	286.8	-0.18	89385	27	17032	477.3	-0.19
wb_conmax	47026	32	770	42491	32	770	43.4	-9.64	40734	32	770	84.9	-13.38
wb_dma	3283	19	521	3258	19	521	0.6	-0.76	3257	19	521	1.2	-0.79
Average								-3.40					-3.65

1) Redundancy removal vs. redundancy removal + resubstitution: In Table I, we present the results obtained by running our method with only redundancy removals and with both redundancy removals and resubstitutions. The columns 'NAND2', 'Lev', and 'FF' show the number of two-input NAND-gates, combinational logic levels, and flip-flops, respectively. The last two columns for each experimental setting show the runtime of the experimental flow in seconds[1] and the percentage NAND2 reduction over the baseline. Here we used 1-step sequential induction with windowing, where the window size is limited to 500 nodes with at most 16 levels in the transitive fanout cone of a target node. For the resubstitution, we limit the divisor count to 100 nodes. We set a tight control on the level count to prevent increasing it during resubstitution.

As seen in Table I, the redundancy removals alone lead to an average reduction of 3.40% in the number of NAND2 gates. When resubstitutions are allowed on top of redundancies, the average reduction increases to 3.65%.

As a final remark, note that all testcases have been verified using sequential verification (*dsec*) in ABC [34] where the verification time is below 13 seconds for all the benchmarks.

2) Single-step vs. multi-step induction: In Table II, we present the results obtained by running our method with 1-step and 2-step sequential induction. In both cases, we use the same settings as in the previous experiment with both redundancy removals and resubstitutions enabled, but with a window size of 5000. Note that we use an increased window size to support larger number of inductive steps. To effectively find potential optimizations in the unrolled inductive-case network, the window need to contain logic from all unrolled frames. When 2-step induction is used, the average reduction in the number of NAND2 gates increases to 3.97% from 3.65%. As before, all testcases have been verified with ABC's *dsec*, where the maximum observed verification time is 74 seconds.

3) Without assumptions vs. with assumptions: When assumptions are allowed, we observed improvements in the NAND2 reduction for specific benchmarks. Namely, when our algorithm is used with 1-step sequential induction allowing both redundancy removals and resubstitutions, reductions of -4.94%, -2.98%, and -11.16% were observed, respectively, for benchmarks `ethernet`, `i2c`, and `usb_phy`.

4) Redundancy removal + resubstitution with different window sizes: When using both redundancy removals and resubstitutions with 1-step sequential induction, we observed that increasing the window size from 500 to 50 000 nodes produced even better results as shown in Table III. The reduction in the number of NAND2 gates increased from 3.65% to 4.10% on average. In this case, the maximum observed sequential verification time is 27 seconds.

B. Technology Mapped Results

In Table IV, we present the results obtained after area-oriented technology mapping for the same OpenCores designs we used in the previous experiments. In this experiment, the baseline is an industrial synthesis flow that does not use any sequential logic optimizations, and Flow1 uses two iterations of sequential SAT-sweeping for mapped networks on top of the baseline. Flow2 runs Flow1 followed by our proposed method, with 1-step sequential induction, with both redundancies and resubstitutions enabled, and with a window size of 50 000.

The table shows the average improvements over the baseline. Our flow achieves an 18.3% area reduction, compared to the baseline, and a 6.9% reduction, compared to Flow1 with

[1]Runtime statistics are not expected to be reproducible.

TABLE II
COMPARISON OF THE PROPOSED METHOD AGAINST THE BASELINE (k-STEP INDUCTION)

	Baseline			**Our Method with 1-step induction**					**Our Method with 2-step induction**				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2%	NAND2	Lev	FF	Time (s)	NAND2%
aes_core	22026	32	530	21535	32	530	49.2	-2.23	21132	31	530	288.9	-4.06
des_area	4611	37	64	4611	37	64	1.7	0.00	4604	37	64	37.7	-0.15
des_perf	77288	23	8808	76534	23	8808	592.6	-0.98	76507	23	8808	1509	-1.01
ethernet	168	13	47	166	13	47	0	-1.19	166	13	47	0.1	-1.19
i2c	931	24	126	891	24	126	0.3	-4.30	891	24	126	0.9	-4.30
mem_ctrl	7097	31	1050	6998	31	1050	3.7	-1.39	6976	31	1048	28.6	-1.70
pci_bridge32	17656	32	3198	17403	32	3198	20.9	-1.43	17376	32	3197	153.1	-1.59
pci_spoci_ctrl	704	20	60	674	20	60	0.5	-4.26	670	20	60	1.5	-4.83
sasc	597	10	117	568	10	117	0.1	-4.86	568	10	117	0.2	-4.86
simple_spi	779	12	131	772	12	131	0.2	-0.90	772	12	131	0.5	-0.90
spi	3621	31	229	3592	31	229	0.9	-0.80	3586	31	229	31.9	-0.97
ss_pcm	464	9	87	399	9	87	0	-14.01	399	9	87	0.1	-14.01
steppermotordrive	138	17	25	125	17	25	0	-9.42	125	17	25	0.1	-9.42
systemcaes	11106	42	670	11105	42	670	8.2	-0.01	11087	42	670	41.7	-0.17
systemcdes	2696	36	190	2687	34	190	5.7	-0.33	2685	34	190	12.5	-0.41
tv80	7740	58	359	7527	58	359	6.2	-2.75	7419	58	359	37.6	-4.15
usb_funct	13910	27	1722	13557	26	1721	16.8	-2.54	13541	26	1721	49.7	-2.65
usb_phy	457	12	98	407	11	98	0.1	-10.94	403	11	98	0.2	-11.82
vga_lcd	89555	27	17032	89385	27	17032	477.3	-0.19	89352	27	17032	1633.5	-0.23
wb_conmax	47026	32	770	40734	32	770	84.9	-13.38	40375	32	770	344.3	-14.14
wb_dma	3283	19	521	3257	19	521	1.2	-0.79	3257	19	521	3.2	-0.79
Average								-3.65					-3.97

TABLE III
COMPARISON OF THE PROPOSED METHOD AGAINST THE BASELINE (INCREASED WINDOW SIZE)

	Baseline			**Our Method**				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2 %
aes_core	22026	32	530	21061	31	530	1404.9	-4.4
des_area	4611	37	64	4594	37	64	71.6	-0.4
des_perf	77288	23	8808	76053	23	8808	811.2	-1.6
ethernet	168	13	47	166	13	47	0	-1.2
i2c	931	24	126	889	24	126	1	-4.5
mem_ctrl	7097	31	1050	6961	31	1048	27	-1.9
pci_bridge32	17656	32	3198	17292	32	3198	151.8	-2.1
pci_spoci_ctrl	704	20	60	671	20	60	2.4	-4.7
sasc	597	10	117	568	10	117	0.2	-4.9
simple_spi	779	12	131	772	12	131	0.6	-0.9
spi	3621	31	229	3583	31	229	86.4	-1.1
ss_pcm	464	9	87	399	9	87	0.1	-14.0
steppermotor	138	17	25	125	17	25	0.1	-9.4
systemcaes	11106	42	670	11070	42	670	137.7	-0.3
systemcdes	2696	36	190	2685	34	190	22.7	-0.4
tv80	7740	58	359	7396	57	359	208.9	-4.4
usb_funct	13910	27	1722	13506	26	1721	38.2	-2.9
usb_phy	457	12	98	403	11	98	0.2	-11.8
vga_lcd	89555	27	17032	89294	27	17032	1993.1	-0.3
wb_conmax	47026	32	770	40184	32	770	391.8	-14.5
wb_dma	3283	19	521	3257	19	521	4	-0.8
Average								-4.1

sequential SAT-sweeping, at a cost of 20% increase in runtime. The results confirm that the two methods, the sequential SAT-sweeping and our proposed method, are orthogonal; our method finds new optimization opportunities on top of state-of-the-art sequential optimizations. All benchmarks were equivalence-checked using existing sequential verification tools.

TABLE IV
RESULTS AFTER TECHNOLOGY MAPPING FOR OPENCORES DESIGNS

Flow	Comb. Area	Seq. Area	# Cells	Runtime
Baseline	1	1	1	-
Flow1 (SSW)	-12.2%	-4.8%	-9.6%	1
Flow2 (SSW + new method)	-18.3%	-4.8%	-14.9%	+20%

TABLE V
RESULTS AFTER PLACE AND ROUTE FOR INDUSTRIAL DESIGNS

	Comb. Area	Seq. Area	WNS	TNS	Tot. Power
Design 1	-6.11%	0.06%	-1.34%	-0.29%	-2.09%
Design 2	-1.66%	-2.79%	-0.40%	2.02%	-1.81%
Design 3	-2.59%	-2.78%	-4.13%	-1.32%	-5.22%
Design 4	-1.39%	-0.31%	0.00%	-0.06%	-2.44%
Design 5	-1.19%	-0.62%	0.00%	0.00%	-1.04%
Design 6	-5.99%	0.10%	-1.85%	-1.59%	-2.42%
Design 7	-2.31%	-2.56%	-1.70%	-0.43%	-2.13%
Average	-3.16%	-1.06%	-1.29%	-0.21%	-2.50%

C. Post Place & Route Results on Industrial Designs

In Table V, we present the results obtained after place and route for 7 industrial benchmarks, where the baseline does not use any sequential logic optimizations and the experimental flow combines our method and sequential SAT-sweeping.

Our flow achieves a 3.16% reduction in combinational area, a 1.06% reduction in sequential area, a 1.29% improvement in worst negative slack (WNS), a 0.21% improvement in total negative slack (TNS), and a 2.5% reduction in total power. These results confirm that our method provides significant improvements in both area and timing metrics, even after place

and route. This further demonstrates that our method scales well such that it can be effectively used on large industrial benchmarks. We note that, on average, 42.26% (max: 69.2%, min: 10.2%) of the total runtime is spent on the SAT solver.

All 7 designs have been verified with an industrial sequential verification tool which uses an state-of-the-art sequential verification flow [24].

V. Conclusion

In this work, we introduced a scalable sequential optimization method based on multi-step induction, extending the concept of Observability Don't Cares (ODCs) to the sequential domain through Sequential Observability Don't Cares (SODCs), which explicitly account for reachability constraints. By simultaneously optimizing two derived combinational networks—representing the base and inductive cases—our approach effectively addresses the dependency issues inherent in traditional ODC-based optimizations. Candidate optimizations are verified using a SAT-based approach that encodes ODC constraints, with a windowing technique employed to maintain scalability. Leveraging SODCs, our method uncovers optimization opportunities that prior approaches were unable to detect.

Experimental results demonstrate the scalability of our method across large industrial designs, yielding significant area improvements in both technology-mapped circuits and post place-and-route designs.

While the method incurs non-negligible runtime, it proves valuable as a high-effort sequential optimization, particularly for area-critical applications. Our analysis shows that most of the runtime is spent on SAT solving, and we anticipate substantial speedups by reducing the number of equivalence-checking SAT calls. Techniques such as randomized or counter-example-guided simulation could efficiently filter out invalid optimizations early. Additionally, exploring heuristics for optimization candidate ordering may expose further opportunities for improvement. As outlined in Section III, applying symmetry-breaking logic transformations and more advanced reachability assumptions could also reveal additional optimization potential. These enhancements are left for future work.

Moreover, our method's integration into a state-of-the-art industrial sequential optimization flow yielded promising results, maintaining significant reductions in both combinational and sequential areas, even after place-and-route. We hope that the enhanced sequential optimization offered by this method will inspire further research in the field, driving efforts to better approximate reachable states and uncover more optimization opportunities.

Acknowledgments

This work was supported by the SNF Grant "Supercool: Design Methods and Tools for Superconducting Electronics" (Grant 200021-1920981), the SRC Contract 3173.001 "Standardizing Boolean transforms to improve quality and runtime of CAD tools," and Synopsys Inc..

Appendix

The proof of Lemma 1 is as follows:

Proof. Suppose that for all Boolean vectors $x \in B^m$ and $y \in S$, it holds that $\mathrm{PO}(N_1, x, y) = \mathrm{PO}(N_2, x, y)$ and $\mathrm{RI}(N_1, x, y) = \mathrm{RI}(N_2, x, y)$. We show that, for any sequence $\{x\}_0^T$, $\mathrm{PO}(N_1, \{x\}_0^T, y_0) = \mathrm{PO}(N_2, \{x\}_0^T, y_0)$ and $\mathrm{RI}(N_1, \{x\}_0^T, y_0) = \mathrm{RI}(N_2, \{x\}_0^T, y_0)$ using mathematical induction.

To this end, fix any sequence $\{x\}_0^T$. We use the notation $\{x\}_0^t$ to denote the subsequence $x_0, x_1, \ldots, x_t$. Considering the initial clock cycle, we have $\mathrm{PO}(N_1, \{x\}_0^0, y_0) = \mathrm{PO}(N_1, x_0, y_0) = \mathrm{PO}(N_2, x_0, y_0) = \mathrm{PO}(N_2, \{x\}_0^0, y_0)$ and $\mathrm{RI}(N_1, \{x\}_0^0, y_0) = \mathrm{RI}(N_1, x_0, y_0) = \mathrm{RI}(N_2, x_0, y_0) = \mathrm{RI}(N_2, \{x\}_0^0, y_0)$.

As the inductive hypothesis, assume that for a positive integer t such that $T \geq t > 0$, $\mathrm{RI}(N_1, \{x\}_0^{t-1}, y_0) = \mathrm{RI}(N_2, \{x\}_0^{t-1}, y_0)$. Note that $\mathrm{RI}(N_1, \{x\}_0^{t-1}, y_0)$ is also in S. Then we have $\mathrm{PO}(N_1, \{x\}_0^t, y_0) = \mathrm{PO}\left(N_1, x_t, \mathrm{RI}(N_1, \{x\}_0^{t-1})\right) = \mathrm{PO}\left(N_2, x_t, \mathrm{RI}(N_2, \{x\}_0^{t-1})\right) = \mathrm{PO}(N_2, \{x\}_0^t, y_0)$ and $\mathrm{RI}(N_1, \{x\}_0^t, y_0) = \mathrm{RI}\left(N_1, x_t, \mathrm{RI}(N_1, \{x\}_0^{t-1})\right) = \mathrm{RI}\left(N_2, x_t, \mathrm{RI}(N_2, \{x\}_0^{t-1})\right) = \mathrm{RI}(N_2, \{x\}_0^t, y_0)$.

Thus, by induction, it follows that the claim holds for $\{x\}_0^T$. □

References

[1] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *ICCAD*, 2008.

[2] "TSMC's New 3nm Chip Wafers Priced at $20,000.[Online July 2023] https://www.siliconexpert.com/blog/tsmc-3nm-wafer/."

[3] E. M. S. K. J. Singh, L. L. C. M. R. Murgai, and R. K. B. A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," *University of California, Berkeley*, vol. 94720, p. 4, 1992.

[4] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1-6, pp. 5–35, 1991.

[5] H. Savoj, A. Mishchenko, and R. Brayton, "m-Inductive Property of Sequential Circuits," *IEEE TCAD*, vol. 35, no. 6, pp. 919–930, 2015.

[6] M. Case, J. Baumgartner, H. Mony, and R. Kanzelman, "Optimal redundancy removal without fixedpoint computation," in *FMCAD*, 2011, pp. 101–108.

[7] M. L. Case, V. N. Kravets, A. Mishchenko, and R. K. Brayton, "Merging nodes under sequential observability," in *DAC*, 2008, pp. 540–545.

[8] R. K. Brayton and A. Mishchenko, "Sequential Rewriting and Synthesis," in *IWLS*, 2007.

[9] V. N. Kravets and A. Mishchenko, "Sequential logic synthesis using symbolic bi-decomposition," in *DATE*, 2009, pp. 1458–1463.

[10] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE TCAD*, vol. 10, no. 1, pp. 63–73, 1991.

[11] A. P. Hurst, A. Mishchenko, and R. K. Brayton, "Fast minimum-register retiming via binary maximum-flow," in *FMCAD*, 2007, pp. 181–187.

[12] P. Bjesse and K. Claessen, "SAT-Based Verification without State Space Traversal," in *FMCAD*, 2000, p. 372–389.

[13] C. van Eijk, "Sequential equivalence checking based on structural similarities," *IEEE TCAD*, vol. 19, no. 7, pp. 814–819, 2000.

[14] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, "Exploiting suspected redundancy without proving it," in *DAC*, 2005, pp. 463–466.

[15] M. Damiani and G. De Micheli, "Observability don't care sets and boolean relations." in *ICCAD*, vol. 90, 1990, pp. 502–505.

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

[17] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure," in *IWLS*, 2006, pp. 15–22.

[18] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Sat sweeping with local observability don't-cares," in *DAC*, 2006, pp. 229–234.
[19] E. Testa, L. Amaru, M. Soeken, A. Mishchenko, P. Vuillod, P.-E. Gaillardon, and G. De Micheli, "Extending Boolean Methods for Scalable Logic Synthesis," *IEEE Access*, vol. 8, 2020.
[20] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A Simulation-Guided Paradigm for Logic Synthesis and Verification," *IEEE TCAD*, vol. 41, no. 8, pp. 2573–2586, 2022.
[21] N. Saluja and S. P. Khatri, "A robust algorithm for approximate compatible observability don't care (CODC) computation," in *DAC*, 2004, pp. 422–427.
[22] J. P. Marques-Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *DAC*, 2000, pp. 675–680.
[23] D. S. Marakkalage, E. Testa, W. L. Neto, A. Mishchenko, G. De Micheli, and L. Amarù, "Scalable sequential optimization under observability don't cares," in *DATE*, 2024, pp. 1–6.
[24] E. Testa, D. S. Marakkalage, M. Quayle, S. Kundu, A. Kumar, D. Ghosh, G. Meuli, G. De Micheli, and L. Amaru, "Enabling scalable sequential synthesis and formal verification in an industrial flow," in *IWLS*, 2024.
[25] D. Brand, "Redundancy and don't cares in logic synthesis," *IEEE Transactions on Computers*, vol. 100, no. 10, pp. 947–952, 1983.
[26] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multilevel logic minimization using implicit don't cares," *IEEE TCAD*, vol. 7, no. 6, pp. 723–740, 1988.
[27] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *ASP-DAC*. IEEE, 2022, pp. 395–402.
[28] A. T. Calvino and G. De Micheli, "Scalable logic rewriting using don't cares," in *DATE*. IEEE, 2024, pp. 1–6.
[29] G. D. Micheli, *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.
[30] R. Brayton, "Compatible observability don't cares revisited," in *ICCAD IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, 2001, pp. 618–623.
[31] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE TCAD*, vol. 21, no. 12, pp. 1377–1394, 2002.
[32] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," UC Berkeley, Tech. Rep., 2005.
[33] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *TACAS*. Springer, 1999, pp. 193–207.
[34] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification*, 2010, pp. 24–40.
[35] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 466–483. [Online]. Available: https://doi.org/10.1007/978-3-642-81955-1_28
[36] "Opencores: https://opencores.org."

Dewmini Sudara Marakkalage is a Ph.D. student at the Integrated Systems Laboratory, EPFL, Lausanne, Switzerland. She received a B.Sc. in Engineering from the Department of Electronic and Telecommunication Engineering, University of Moratuwa, Sri Lanka, in 2016 and a M.Sc. in Computer Science from the School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland, in 2020. Her research interests include logic synthesis and design automation for emerging technologies.

Eleonora Testa (Member, IEEE) is a Manager, R&D in the Design Technology Group of Synopsys Inc., Sunnyvale, CA, USA where she leads an international team focusing on logic synthesis. Dr. Testa obtained her Ph.D. degree in Computer Science from EPFL, Lausanne, Switzerland, in 2020. She has received the EDAA Outstanding Dissertation Award (2021). She has been serving as a TPC member for several conferences, including DAC, DATE, and ICCAD and is an Associate Editor for IEEE Transactions on CAD.

Giulia Meuli received the Ph.D. degree in electrical engineering from the Swiss Federal Institute of Technology Lausanne (EPFL), Lausanne, Switzerland, in 2020. She received the EPFL/EDEE best Doctoral Thesis Award in Electrical Engineering and the IBM Research Award 2023. She is currently R&D Staff Engineer in the Design Group, Synopsys Inc., Agrate Brianza, Lombardy, Italy. She is reviewer for several conferences and journal, including TCAD. She served as TPC member for conferences including DATE, and in the Organizing Committee of IWLS. Her research interests include logic synthesis, EDA for emerging technologies and quantum computing.

Walter Lau Neto received the M.S. degree in Microelectronics from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2018. In 2022, he received the Ph.D. degree in Computer Engineering from the University of Utah, Salt Lake City, UT, USA. Currently, he is a Senior Staff R&D engineer at Synopsys Inc., Sunnyvale, CA, USA, where he works on developing novel logic synthesis techniques. His research interests include logic synthesis, machine learning for logic synthesis, and electronic design automation.

Alan Mishchenko received the M.S. degree from the Moscow Institute of Physics and Technology, Moscow, Russia, in 1993 and the Ph.D. degree from the Glushkov Institute of Cybernetics, Kiev, Ukraine, in 1997. In 2002, he joined the EECS Department, University of California at Berkeley, Berkeley, CA, USA, where he is currently a Full Researcher. His current research interests include computationally efficient logic synthesis, formal verification, and machine learning.

Giovanni De Micheli is Professor and Director of the EcoCloud center at EPFL Lausanne, Switzerland. Prof. De Micheli is a Fellow of ACM, AAAS, and IEEE, a member of the Academia Europaea, and an International Honorary member of the American Academy of Arts and Sciences. His current research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis for emerging technologies. He is the author of Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994, co-author and/or co-editor of ten other books, and of over 1000 technical publications. His citation h-index is above 100 according to Google Scholar. He is a member of the Scientific Advisory Board of IMEC (Leuven, Belgium) and STMicroelectronics. Prof. De Micheli is the recipient several awards, including the 2025 IEEE Kirchhoff Award, 2022 ESDA-IEEE/CEDA Phil Kaufman Award, and the 2019 ACM/SIGDA Pioneering Achievement Award.

Luca Amarú is an Executive Director, R&D in the Design Technology Group of Synopsys Inc., Sunnyvale, CA, USA, where he is responsible for designing the next generation of logic synthesis technologies. When not coding, Dr. Amaru leads an exceptional team of R&D engineers focusing on logic synthesis. Dr. Amaru received his PhD degree in Computer Science from EPFL, Lausanne, Switzerland (2015). He received the ACM/IEEE Design Automation Conference Under-40 Innovator Award (2022), the IEEE TCAD Donald O. Pederson Best Paper Award (2018), the EDAA Outstanding Dissertation Award (2016) and other best paper awards and nominations. Dr. Amaru is an Associate Editor for IEEE Transactions on CAD. He is a Senior member of the IEEE.