2024 Design, Automation & Test in Europe Conference (DATE 2024) – Best Paper Award Candidate

Scalable Sequential Optimization Under Observability Don't Cares

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

***Abstract*—Sequential logic synthesis can provide better Power-Performance-Area (PPA) than combinational logic synthesis since it explores a larger solution space. As the gate cost in advanced technologies keeps rising, sequential logic synthesis provides a powerful alternative that is gaining momentum in the EDA community. In this work, we present a new scalable algorithm for don't-care-based sequential logic synthesis. Our new approach is based on sequential k-step induction and can apply both redundancy removal and resubstitution transformations under Sequential Observability Don't Cares (SODCs). Using SODC-based optimizations with induction is a challenging problem due to dependencies and alignment of don't cares among the base case and the inductive case. We propose a new approach utilizing the full power of SODCs without limiting the solution space. Our algorithm is implemented as part of an industrial tool and achieves a 6.9% average area improvement after technology mapping when compared to state-of-the-art sequential synthesis methods. Moreover, all the new sequential optimizations can be verified using state-of-the-art 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 (e.g., area, power, and delay). It is an integral part of modern *Electronic Design Automation* (EDA) flows. Combinational logic synthesis optimizes logic networks while preserving the combinational equivalence. Even if a logic network has sequential elements, combinational logic synthesis can operate on them by considering register inputs/outputs as primary outputs/inputs, while disregarding constraints on the reachable states. Sequential logic synthesis aims instead at optimizing logic networks with sequential elements and can be seen as a stronger type of logic synthesis, since it can take into account the fact that not all combinations of register values are reachable in general. It is widely known that sequential logic synthesis is able to explore a larger solution space and generally provides better *power, performance, and area* (PPA) [1]. Such PPA opportunities become even more important to grasp as today's chip design cost continues to increase [2].

Previous approaches have been considered for sequential logic synthesis [1], [3]–[10]. One such approach is to integrate combinational optimizations together with retiming (i.e., moving registers over combinational nodes) [4], [11], which can exploit optimization opportunities arising due to structural properties across register boundaries [8]. A powerful yet scalable state-of-the-art approach is given by the *sequential SAT-sweeping* (SSW) algorithm from [1]. This method makes use of SAT and sequential induction [12]–[14] to prove the validity of logic transformations (i.e., merging sequentially equivalent nodes).

In this work, we present a novel scalable algorithm for don't cares-based sequential logic synthesis. Our method is based on k-step induction, and is orthogonal to the one in [1]. As a matter of fact, our new method applies both redundancy removal and resubstitution while using *sequential observability don't cares* (SODCs). In combinational logic synthesis, *observability don't cares* (ODCs), i.e., the input patterns for which the value of a wire is not observed at outputs, can be used to find better optimization opportunities [15]–[18]. The use of ODCs for optimizations comes with inherent challenges as ODCs can change after applying an ODC-based optimization. In sequential optimizations, the challenges are more prominent as it is highly non-trivial to consider the sequential nature of circuits while dealing with ODC dependencies. Although the dependency issues can be avoided by considering only *compatible ODCs* (CODCs) [19], i.e., ODCs that can be used independently at each node, it can lead to many missed optimization opportunities. Nevertheless, the method we propose 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 an ODC-based, combinational optimization method built on Boolean Satisfiability (SAT) [20], which uses windowing for scalability. Essentially, our method naturally finds valid ODC-based sequential optimizations that are compatible with each other, without limiting the search space to CODC-based optimizations. Thus, it may find better optimizations, especially in circuits with sequential feedback that can be problematic for traditional retiming-based methods. In essence, our method finds sequential optimizations that were never covered by prior approaches in a scalable manner. Moreover, as our approach does not move registers over combinational logic, the verification of the optimized networks is more likely to succeed, and hence our approach is more attractive for deployment in industrial tools.

The proposed method has been implemented in an industrial tool and shown to achieve a 6.9% average reduction in area

979-8-3503-4859-0/DATE24/© 2024 EDAA

after technology mapping on top of state-of-the-art sequential optimization methods (e.g., SSW). All designs were verified using state-of-the-art sequential verification tools.

The organization of the paper: 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. Finally, Section IV presents experimental results and 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 refers 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 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. A sequential redundancy is a generalization of a redundancy where the stuck-at-constant property holds considering all *observable input patterns* and *reachable states*.

C. 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.

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 [21], [22] 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) [23] 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.

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 $g_2 = 1$, one can simplify the circuit by assuming that g_1 is stuck at 0,

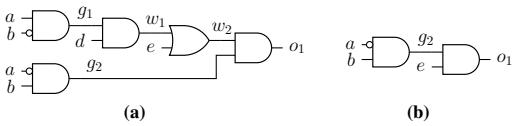

Fig. 1. A combinational logic network (a) and its optimized version (b).

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, as by design, lo_1 and lo_2 can never be 1 at the same time in the subsequent clock cycles. The optimized circuit is 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 [24] cannot find this optimization.

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.

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.

B. 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 considers two combinational networks that are derived from the original network according to Definition 1.

Definition 1. *For a network N, the k-step base case network N^b is the combinational network obtained by taking k copies of N (referred to as frames), connecting the RIs of each frame to the corresponding ROs of the subsequent frame, replacing the ROs of the first frame with the respective register initial values, and designating RIs of the last frame as POs. The k-step inductive case network N^i is similarly defined except with $k + 1$ frames and considering ROs of the first frame as PIs.*

For the example network of Fig. 2(a), the base case and the inductive case networks for 1-step 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. The behavior of N^b is the same as that of the original network N for the initial clock cycle and the behavior of N^i is the same as two consecutive clock cycles of N for any initial state. 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 (see Theorem 1).

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 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.

C. 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 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. This approach allows the algorithm to find subsequent optimizations for the remaining nodes that may depend on the already applied optimizations, and it avoids any dependency issues that would arise if we were to use the simple approach we stated at the end of Section III-B with regular ODCs. Hence, the algorithm computes compatible sequential optimizations without limiting to CODCs.

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 the base case network and 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. 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-D for details).

As an illustrative example, consider Fig. 2(a). We can prove, for example, 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.

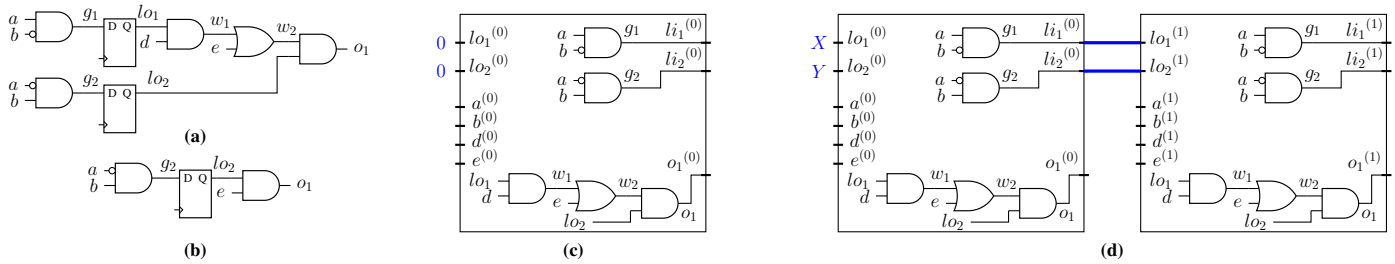

Fig. 2. A sequential logic network (a), its optimized version (b), and its base case network (c) and the inductive case network (d) for 1-step sequential induction.

1) Enhancement 1: We extend our algorithm to use k-step sequential induction where the base case network has k frames and the inductive case network has $k+1$ frames. In this case, we check if the target redundancy is valid in all k frames of the base case and the last frame of the inductive case. The validity in all frames of the base case implies that the original network would behave the same for the first k clock cycles (starting with the initial state). The validity in the last frame of the inductive case implies that the original network will behave the same for any $k+1$ clock cycles, starting from any state.

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 is used, we only get the first two frames of Fig. 3(c), and the second frame's lo_3, lo_4 are fed 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.

2) 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. Then we consider as resubstitution candidates all versions of n obtained by replacing a fanin of n with a divisor. 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'.

3) 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.*

D. Complete Algorithm

The high-level pseudocode of our method is presented in Algorithm 1 including all enhancements. In Line 4, the algorithm iterates over all gates in the original network, and in Line 5, it considers different optimization candidates Δ, namely, different fanin redundancies and resubstitutions. Then, for each frame of the base case network, the algorithm checks if Δ is valid and if so, applies Δ in that frame (Line 6-Line 10). If it is valid in all frames, then it applies Δ in the first k frames of the inductive case network (Line 6) and checks for the validity in the last frame (Line 12). If valid, it applies Δ in the last frame

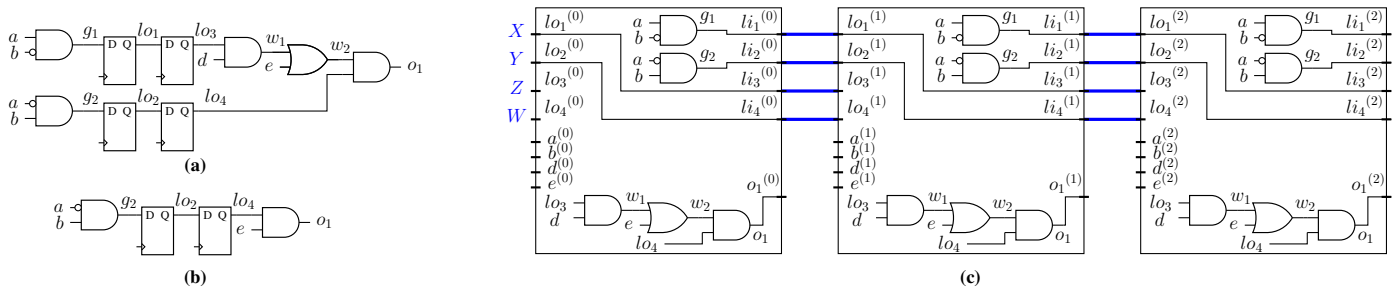

Fig. 3. A sequential logic network (a), its optimized version (b), and its inductive case network (c) for 2-step sequential induction.

Algorithm 1: High-level pseudocode of sequential optimization with k-step induction and assumptions.

```
   input : Input network N. Number of frames k.
1  N^b ← N unrolled into k frames and first frame ROs replaced
     with initial states.
2  N^i ← 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 ∈ N do
5      for each candidate optimization Δ for g do
6          for j = 1, ..., k do
7              if Δ is invalid for g in N^{b,j} then
8                  Undo Δ in all frames N^{b,1}, ..., N^{b,j-1}.
9                  Continue outer loop.
10             Apply Δ in N^{b,j}.
11         Apply Δ in N^{i,1}, ..., N^{i,k}.
12         if Δ is invalid for g in N^{i,k+1} then
13             Undo Δ in N^{b,1}, ..., N^{b,k} and N^{i,1}, ..., N^{i,k}.
14             Continue loop.
15         Apply Δ in N^{i,k+1}.
16         Apply Δ in N.
17 return N
```

as well and also in the original network (Line 15-Line 16). At any point, if Δ is invalid for the considered frame, it undoes all preceding applications of it (Line 8, Line 13).

In Lines 7 and 12, to check for the validity of a target optimization, 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 window output differ if the candidate optimization is applied?* If the problem is UNSAT, then the target optimization is valid. The size of the window and the conflict limit for the SAT solver are configurable parameters.

E. Correctness of the Proposed Approach

In this section, we show the correctness of our algorithm considering the simple case of 1-step sequential induction with no assumptions. The proof can easily be extended to the case of k-step induction and to the case with assumptions.

Theorem 1. *Consider a logic network N and its 1-step base case and inductive versions N^b and N^i. Let Δ be a logic transformation and let $N_\Delta, N^b_\Delta, N^i_\Delta$, 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^b_Δ and N^i_Δ, then N and N_Δ are sequentially equivalent.*

Proof. Let m, n, ℓ be, respectively, the PI, PO, and register counts of N. Let $s_0 \in \mathbb{B}^\ell$ be the initial state, and let $S \subseteq \mathbb{B}^\ell$ be the set of all reachable states after the initial clock cycle. For any $x \in \mathbb{B}^m$, let $N^b(x) \in \mathbb{B}^\ell \times \mathbb{B}^m$ be the RI/PO pattern of N^b on input x. Similarly, let $N(x, y)$ and $N^i(x, y)$ be the RI/PO pattern of N and N^i on PI pattern $x \in \mathbb{B}^m$ and RO pattern $y \in \mathbb{B}^\ell$. It is sufficient to show that, for any PI pattern $x \in \mathbb{B}^m$ and state $y \in \{s_0\} \cup S$, it holds that $N(x, y) = N_\Delta(x, y)$.

Note that $N^b(x) = N(x, s_0)$ and $N^b_\Delta(x) = N_\Delta(x, s_0)$ by construction of N^b, which implies the claim for $y = s_0$ because we have $N^b(x) = N^b_\Delta(x)$ due to combinational equivalence of N^b and N^b_Δ. Now let S_i be the subset of reachable states of N after i clock cycles where $S_0 = \{s_0\}$. Note that $S = \cup_{i=1}^{\infty} S_i$. We show that the claim holds for all $y \in S_{i+1}$ assuming it holds for all $y \in \cup_{j=0}^{i} S_j$. For contrary, suppose that there exists some $y \in S_{i+1}$ and $x \in \mathbb{B}^m$ such that $N(x, y) \neq N_\Delta(x, y)$. Let x', y' be the state and input patterns that led to the state y. Then, if we input x', x, y' to N^i and N^i_Δ, their outputs should differ, which is a contradiction because we assumed they are combinationally equivalent. □

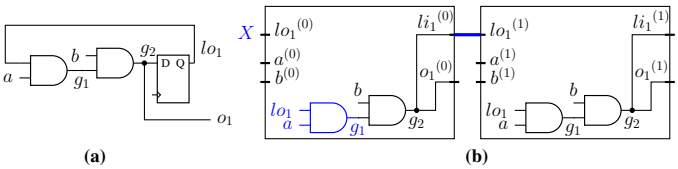

Fig. 4. A sequential logic network with feedback (a) and its inductive case network (b) for 1-step sequential induction before applying assumptions in the first frame.

IV. Experimental Results

This section shows experimental results obtained using the proposed approach. Presented are both results for *and-inverter graphs* (AIGs) and after technology mapping within an industrial flow. The approach was evaluated on a sequential benchmark suite from [25].

First, we present results for AIGs. In the evaluation baseline, we consider an optimization flow consisting of a state-of-the-art sequential optimization [1] together with combinational rewriting (commands *scorr* and *rewrite* in ABC [24]). These two are interleaved and run until saturation, i.e., no further reduction is observed. In the experimental flow, we additionally run the proposed method on top of the baseline, using 1-step sequential induction. Our setup limits the window size to 50k nodes, the divisor count during resubstitution to 100 nodes, and the TFO of a node to 16 levels. The proposed method performs both redundancy removal and resubstitution. We also set a tight control on the level count to prevent increasing it during resubstitution.

The results are presented in Table I where columns 'NAND2', 'Lev', and 'FF' show the number of two-input NAND-gates, combinational logic levels, and flip-flops, respectively. The last two columns show the runtime of the experimental flow in seconds and the NAND2 reduction over the baseline. The experimental flow leads to a 4.1% average area reduction over the baseline, evaluated as the NAND2 count reduction. It is worth noting that different trade-offs can be achieved between area and runtime. For instance, a runtime decrease of 43% leads to an average gain of 3.6%. Larger runtime can lead to more improvements. All testcases have been verified using sequential verification (*dsec*) in ABC [24].

In Table II, we present the results obtained in an industrial flow after the initial synthesis and area-oriented technology

TABLE I
COMPARISON OF THE PROPOSED METHOD AGAINST THE BASELINE

	Baseline			Our Method				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2 %
aes_core	22026	32	530	21061	31	530	2419.6	-4.4
des_area	4611	37	64	4594	37	64	108	-0.4
des_perf	77288	23	8808	76053	23	8808	1433.5	-1.6
ethernet	168	13	47	166	13	47	0.1	-1.2
i2c	931	24	126	889	24	126	1.5	-4.5
mem_ctrl	7097	31	1050	6961	31	1048	41.2	-1.9
pci_bridge32	17656	32	3198	17292	32	3198	228.1	-2.1
pci_spoci_ctrl	704	20	60	671	20	60	3.8	-4.7
sasc	597	10	117	568	10	117	0.3	-4.9
simple_spi	779	12	131	772	12	131	0.9	-0.9
spi	3621	31	229	3583	31	229	118.3	-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	204	-0.3
systemcdes	2696	36	190	2685	34	190	34.4	-0.4
tv80	7740	58	359	7396	57	359	310.1	-4.4
usb_funct	13910	27	1722	13506	26	1721	61	-2.9
usb_phy	457	12	98	403	11	98	0.3	-11.8
vga_lcd	89555	27	17032	89294	27	17032	3273.5	-0.3
wb_conmax	47026	32	770	40184	32	770	655.1	-14.5
wb_dma	3283	19	521	3257	19	521	6.4	-0.8
Average								-4.1

TABLE II
RESULTS AFTER TECHNOLOGY MAPPING FOR OPENCORES DESIGNS

Flow	Combo. 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%

mapping. The baseline is an industrial flow that does not use any sequential logic synthesis. Flow1 runs two iterations of sequential SAT-sweeping of a mapped network. Flow2 runs Flow1, followed by the proposed method. The results are shown as 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 sequential SAT-sweeping. Flow2 leads to a 20% increase in runtime, compared to Flow1. These results confirm that the two methods are orthogonal and the proposed method finds new optimization opportunities on top of state-of-the-art sequential optimizations (i.e., sequential SAT-sweeping). All benchmarks were equivalence-checked using existing sequential verification tools.

V. CONCLUSION

In this work, we propose a new scalable sequential optimization method based on induction where ODCs can be effectively used to find optimization opportunities without limiting don't-cares to be compatible (CODCs). Our method uses ODCs in the context of windowing while employing SAT-based checks to identify optimization opportunities and it is able to find optimization opportunities that were never found before by prior approaches. Despite the non-negligible running time of the proposed method, it is useful as a high-effort sequential optimization for area-critical applications. Our analysis indicates that most of it is spent on SAT solving. We envision that significant running time reductions are achievable by drastically reducing the number of equivalence-checking SAT calls using randomized/counter-example-guided simulation techniques to reject invalid optimization early. Moreover, it is also worth exploring different heuristics for ordering the candidate optimizations, as it can uncover more optimization opportunities. We leave these improvements for future work.

We also remark that the use of our method in a state-of-the-art industrial sequential optimization flow shows promising results. While the observed gains are achieved by naively running the new method after the baseline optimization flow, it remains to be seen where in the flow the new method is the most useful. We hope that the improved sequential optimization offered by the proposed method will inspire further research.

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] 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.

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

[17] 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.

[18] 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.

[19] N. Saluja and S. P. Khatri, "A robust algorithm for approximate compatible observability don't care (CODC) computation," in *DAC*, 2004, pp. 422–427.

[20] J. P. Marques-Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *DAC*, 2000, pp. 675–680.

[21] 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.

[22] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," UC Berkeley, Tech. Rep., 2005.

[23] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *TACAS*. Springer, 1999, pp. 193–207.

[24] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*, 2010, pp. 24–40.

[25] "Opencores: https://opencores.org."