Node Mergers in the Presence of Don't Cares

Stephen M. Plaza, Kai-hui Chang, Igor L. Markov, Valeria Bertacco
EECS Department, University of Michigan, Ann Arbor, MI 48109-2121
{splaza, changkh, imarkov, valeria}@umich.edu

Abstract— SAT sweeping is the activity of merging two or more functionally equivalent nodes in a circuit by selecting one of them to represent the entire equivalence class. This provides significant advantages in synthesis because it can reduce circuit size and provides additionaly flexibility in technology mapping which could be crucial in post-synthesis optimizations. In addition, it is also critical in verification because it can reduce the complexity of the netlist to be analyzed in equivalence checking. Most algorithms available to this end do not exploit observability don't cares (ODCs) since they do not lend themselves to symmetric transformations. Although a few recent approaches have proposed solutions that can exploit ODCs by overcoming this limitation, they limit their analysis to just a few levels of surrounding logic due to the elevated computational complexity.

We develop an ODC-based node merging algorithm that performs efficient global ODC analysis (considering the entire netlist) through simulation and SAT. Our contributions which enable global ODC-based optimizations are: (1) a fast ODC-aware simulator and (2) an incremental verification strategy that limits computational complexity. In addition, our techniques operate on arbitrarily mapped netlists, allowing for powerful post-synthesis optimizations. We show that global ODC analysis discovers up to 60% more (and 25% on average) node-merging opportunities than current state-of-the-art solutions based on local ODC analysis.

I. Introduction

Merging equivalent circuit nodes is a popular and effective technique to reduce the area of a logic circuit. It scales to very large circuits, but, unlike BDD-based techniques, requires non-trivial algorithms to identify potential mergers. Such algorithms were first developed in the context of formal verification to detect possible cut-points in equivalence checking [5, 7]. To this end, the work in [6, 9] uses a combination of SAT and simulation. Candidate nodes for merging are first selected by checking whether their outputs correspond when stimulated with random patterns applied to the design's inputs and then their actual equivalence can be verified using SAT. The simulation is refined through counterexamples generated by SAT which reduces the number of checks resulting in non-equivalence. Rather than finding equivalent nodes as post-processing step, the work in [9] improves equivalence checking by merging equivalent nodes while constructing the circuit.

Observability don't cares (ODCs) occur when internal circuit node values for certain input patterns are irrelevant to the outputs of the design and hence they require knowledge of this downstream logic to be detected. Satisfiable don't cares (SDCs), also known as controllability don't cares, represent internal values that can never be stimulated from the design's inputs. Node merging through simulation and SAT-based analysis inherently exploits SDCs because unsatisfiable combinations never arise during simulation. On the other hand, incremental approaches such as [9] do not allow for the detection of ODCs because no information of the downstream logic is maintained. We believe that by taking into account ODCs, additional node mergers should be possible.

Several related algorithms have recently been developed [16, 10, 11, 12, 4] to simplify Boolean networks and improve the efficiency of SAT solvers. However, techniques that determine ODCs often restrict the computation to a subset of them [12] or consider small windows [11] of the circuit due to their lack of scalability. A recent work [16] uses bounded-depth simulation of And-Inverter Graphs (AIGs) [6] to extract local ODCs. Their work uses local ODCs to efficiently improve SAT sweeping resulting in a considerable reduction in the size of the underlying AIG.

Because of the computational complexity involved in deriving ODCs, previous work tends to emphasize local computation as a synthesis optimization before technology mapping. This emphasis is well justified for AIGs which have a much larger number of internal nodes, and thus possible mergers, than mapped circuits. However, our intended applications are in physical synthesis, where technology mapping can significantly affect circuit delay, and the placement of standard cells is crucial. In this context, fewer nodes are exposed, and one must search for additional don't cares not found by existing techniques from verification and not exploited by existing techniques from synthesis. Thus, the goal of our work is to quickly identify nodes equivalent up to global don't cares, efficiently verify their equivalence, and use these techniques to simplify existing circuits. Additionally, our implementation can operate on mapped designs without requiring costly netlist conversions which otherwise lead to a loss in physical information and delay estimates.

To enable global analysis, we develop a fast simulator that identifies don't care conditions and quickly produces signatures for each node. Extending the work in [16], we use this simulator to consider ODCs from any depth. After simulation, we can identify candidates for node merging using an incremental verification approach. We determine the downstream logic necessary to verify a node merger through simulation. If simulation under-approximates the downstream logic necessary to validate a merger, we refine the simulation and expand the amount downstream logic considered. In the case when only a few levels of logic are required to prove equivalence, our verification approach will consider only this logic. Our framework is flexible in that it is not limited to SAT-based verification engines and ATPG engines can be easily used instead. By means of our contributions we can: 1) find on average 25% additional mergers compared to [16] and 2) prove when two given nodes are unmergeable.

Section II gives background on signatures, SAT, and recent advances in ODC computation. Section III explains a representation scheme for ODCs, and in Section IV, we describe an efficient simulator and a strategy for dynamically generating simulation patterns. Section V introduces the SAT engine that verifies the merger. Finally, in Section VI, we give results that show the number of ODC-based mergers performed for several benchmarks.

II. Background

In this section, we first discuss prior work in signature-based equivalence checking [6, 9] and then strategies for computing ODCs [10, 12, 16].

A. Simulation and Satisfiability

A given node F in a Boolean network can be characterized by its signature, S_F, for K input vectors $X_1 \cdots X_K$.

Definition 1 $S_F = \{F(X_1), \ldots, F(X_K)\}$ *where* $F(X_i) \in \{0, 1\}$ *indicates the output of* F *for a given input vector.*

Vectors X_i can be generated at random and used in bit-parallel simulation to compute a signature for each node in the circuit. For a network with N nodes, the time complexity of generating signatures for the whole network is $O(NK)$. Nodes can be distinguished by the following formula: $S_F \neq S_G \Rightarrow F \neq G$. Therefore, equivalent signatures can be used to efficiently identify potential node equivalences in a circuit by deriving a hash index for each signature [6]. Since

TABLE I
COMPARISONS BETWEEN RECENT STRATEGIES FOR NODE MERGERS AND DERIVING DON'T CARES.

Property	Simulation-guided SAT [5, 9]	Window-based ODC+SDC [10]	Local SAT-sweep [16]	Our global analysis
Don't cares computed	global SDCs	local SDCs & local ODCs	global SDCs & local ODCs	global SDCs & global ODCs
Computational engines	simulation + SAT	primarily SAT	simulation + SAT	simulation + SAT
Complexity limited by	SAT engine	windowing strategy	levels of downstream logic	moving-dominator incremental SAT
Primary application domain	verification	synthesis	verification	verification; logic & physical synthesis

$S_F = S_G$ does not imply that $F = G$, this potential equivalence must be verified, e.g., using SAT, as explained below.

The efficiency of the frameworks in [6, 9] is dependent on the underlying engines for formally verifying the equivalence of nodes with equivalent signatures. Recent advances in SAT such as learning and non-chronological backtracking [14] have made SAT a more scalable alternative to BDDs in applications like equivalence checking. The equivalence of two nodes, F and G, in a network can be determined by constructing an XOR-based miter [2] between them and asserting the output to 1 as shown in the following formula:

$$(F = G) \Leftrightarrow (\forall i \; F(X_i) \oplus G(X_i) \neq 1) \tag{1}$$

where $\bigcup_i X_i$ is the set of all possible inputs.

In [6], input vectors are generated from the counter-examples derived from SAT checks that prove $F \neq G$. These counter examples improve the quality of the signatures by eliminating situations where $S_F = S_G$ despite $F \neq G$.

B. Observability Don't Cares

Figure 1(left) shows examples of satisfiability don't cares (SDCs) and observability don't cares (ODCs). ODCs occur when the value of an internal node does not affect the outputs of the circuit because of limited observability. For the circuit on the right, when $a = 0$ and $b = 0$, the output value of F is a don't care. An SDC occurs when certain input combinations do not arise due to limited controllability. For example, the combination of $x = 1$ and $y = 0$ cannot occur for the circuit shown on the left in Figure 1(left). SDCs are implicitly handled when using SAT for equivalence checking because this combination cannot occur for any satisfying assignment.

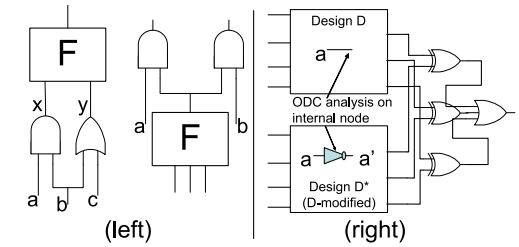

Fig. 1. (left) The left circuit shows examples of SDCs, and the right circuit shows examples of ODCs. (right) Identifying an ODC for an internal node a in a network by constructing a miter for each output and inverting a in a modified copy of the network. The set of inputs where the miter is 1 corresponds to the care-set of that node.

Figure 1(right) shows a strategy for identifying ODCs for a node a. First, the circuit D is copied, and a is inverted in the circuit D^*. Then miters are constructed between the outputs of the two circuits and the care set, denoted as $C(a)$, can be derived as follows:

$$C(a) = \bigcup_{i:D(X_i) \neq D^*(X_i)} X_i \tag{2}$$

A SAT solver can derive C by adding constraints called **blocking clauses** that invalidate previous satisfying assignments to the miter in Figure 1(right) [10]. The ODC of a is therefore:

$$ODC(a) = \bigcup_i X_i - C(a) \tag{3}$$

This approach can be computationally expensive and therefore unscalable, particularly when the miters are far from a. In [10], this complexity is managed by examining only small windows of logic around each node being optimized. The don't cares extracted are used to reduce the circuit literal counts. In [16], a very efficient methodology is developed to merge nodes using local don't cares through simulation and SAT. Their complexity is limited by considering only a few levels of downstream logic for each node. Our technique enhances [16] by efficiently performing node mergers using global don't cares which enables us to potentially find several more mergers. To do this, we develop a fast simulator that can quickly extract global don't care information. Also, we develop an incremental verification engine that adjusts to the complexity of the particular merger being examined. Our work, along with [10, 16], is not limited to compatibility ODCs (CODCs), which are a subset of ODCs and are easier to compute because of their convenient properties [13, 12]. Although CODCs have the nice property that optimizations involving one node's CODCs do not affect the CODCs of another, this is not necessary in our work because we examine one node at a time. We summarize the comparison between our work and [5, 9, 10, 16] in Table I.

III. IDENTIFYING ODC-BASED NODE MERGERS

In this section, we develop the theory involved behind ODC-based node merging and describe the use of signatures to identify candidate mergers. A similar discussion to the theory described in this section can be found in [16]. Then, in the next section, we explain how we enhance [16] by extracting global don't cares.

A. ODC-substitutability

Traditionally, a node merger can occur between a and b when they are functionally equivalent. We define node mergers between a and b in the presence of ODCs when a is **ODC-substitutable** to b.

Definition 2 *a is ODC-substitutable to b if* $ONSET(a) \cup ODC(b) = ONSET(b) \cup ODC(b)$.

When a is ODC-substitutable to b, a merger between a and b means that a can be substituted for b. Because the ODCs of only one node are considered, ODC-substitutability is not symmetric as b might not be ODC substitutable to a.

As in [16], our strategy of merging nodes in the presence of ODCs first uses signatures to find candidate ODC-substitutabilities. However, because the candidate merger depends on which node's ODCs are considered, more powerful signature matching techniques [16] are required than the $O(1)$-time signature hashing in [9]

B. Reasoning About ODCs in Signatures

Each node in the circuit maintains a signature S as defined in Definition 1. In addition, an **ODC mask** S_f^* is maintained for node f:

Definition 3 $S_f^* = \{X_1 \notin ODC(f), \; \ldots \; , X_K \notin ODC(f)\}$ *When an input vector X_i is in the set $ODC(f)$, that bit position is denoted by a* 0.

Set operations can be efficiently executed on these signatures through bit-wise manipulations. The following shows how the $\subseteq$ relation is defined using the signatures of two nodes, S_a and S_b:

Definition 4 *$S_b \subseteq S_a$ if and only if $S_b | S_a = S_a$ where $|$ represents bit-wise OR.*

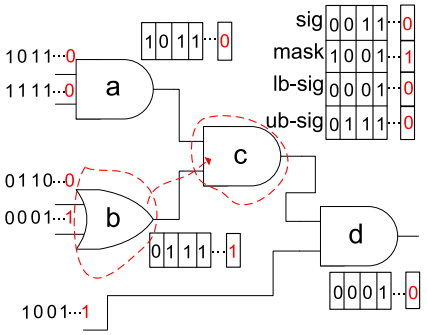

Fig. 2. An example showing how ODCs are represented in a circuit. For clarity, the example only shows ODC information for node c. The other internal nodes show only signatures S. When examining the first four simulation patterns, node b is a candidate for merging with c. Further simulation indicates that an ODC merger is not possible.

Figure 2 shows a circuit with signatures for each node and a mask for node c. Each ODC for a node is marked by a 0 in the ODC mask. In our framework, we express the flexibility of a given node by maintaining an **upper-bound** S^{hi} and **lower-bound signature** S^{lo}.

Definition 5 *$S^{lo} = S\&S^*$ where $\&$ represents bit-wise AND and $S^{hi} = S|\neg(S^*)$.*

S^{lo} and S^{hi} of node f correspond to range of Boolean function $[f^{lo}, f^{hi}]$ that are ODC-substitutable to f because the differences between the range of functions are a subset of $ODC(f)$.

After simulation populates the different signatures, merger candidates can be found. In the example in Figure 2, after the first four simulation patterns, node b is depicted as a **candidate** for ODC-substitutability with c.

Definition 6 *Node b is a candidate for ODC-substitutability with node c if and only if $(S_b \oplus S_c) \subseteq \neg S_c^*$. This can be re-expressed as $S_b \subseteq [S_c^{lo}, S_c^{hi}]$, in other words, S_b is contained within the range of signatures defined by S_c^{lo} and S_c^{hi}.*

Therefore, by simple application of S_c^*, it can be determined that b is an ODC-substitutable candidate with c. However, in this example, further simulation reveals that the candidate merger is incorrect. Similar to Definition 2, if b is an ODC-substitutable candidate with c, it does not imply that c is an ODC-substitutable candidate with b.

Unlike checking for equivalence with signatures, $O(1)$-time complexity hashing cannot be used to identify ODC-substitutability candidates. Each node needs to apply its mask to every other node to find candidates. The result is that for N nodes, finding all ODC-substitutability candidates for the design requires $O(N^2K)$-time complexity assuming that applying a mask is an $O(K)$-time operation. We now introduce a strategy that significantly reduces computation in practice.

First, all of the signatures, S, in the design are sorted by the value obtained by treating each K-bit signature as a single K-bit number. This operation requires $O(NKlgN)$-time. Then, for a given node c, candidates can be found by performing two binary searches with S_c^{lo} and S_c^{hi} to obtain a lower and upper bound on the sorted S, an $O(KlgN)$-time operation. Searching for complemented candidates can be accomplished by simply complementing S_c^{lo} and using this to derive an upper bound and similarly complementing S_c^{hi} and using this to derive a lower bound. We expect binary searches on this contiguous data structure to involve less pointer chasing and be faster, in practice, than the binary trie used in [16]. The following formula defines the set of signatures S_x that will be checked for candidacy (ignoring the case of negation for simplicity): $\bigcup_x S_x \; if \; num(S_c^{lo}) \leq num(S_x) \leq num(S_c^{hi})$ where num represents the K-bit number of the signature. This set is linearly traversed to find any candidates according to Definition 6.

IV. Global ODC Analysis

Below, we describe an efficient simulator that generates ODC information by considering downstream logic, whose complexity is comparable to non-ODC signature generation without considering downstream logic.

Generating ODC masks S^* efficiently is integral to maintaining the scalability of our framework. Whereas each node's S can be computed from its immediate fanin, computing each node's S^* often requires all downstream logic.

The S^* can be computed for each node by determining the care-set using Equation 2 where the X_i are the random simulation vectors. This approach requires circuit simulation of each X_i for each node. For K simulation vectors and N nodes the time-complexity is $O(N^2K)$. Although the simulation can be confined to just the fanout cone of the node, the procedure is computationally expensive.

A. Approximate ODC Simulator

We now describe our approximate ODC simulator whose complexity is only $O(n'K)$ where n' is the number of nets or wires in the design. The algorithm for generating masks using the approximate simulator is shown in Figure 3.

```
void gen_odc_mask(Nodes N){
  set_output_S*(N);
  reverse_levelize(N);
  for_each node ∈ N
  {
    node.S* = 0;
    for_each output in node.fanout
    {
      temp_S* = get_local_ODC(node, output);
      temp_S* = temp_S* & output.S*;
      node.S* |= temp_S*;
    }
  }
}
```

Fig. 3. Efficiently generating ODC masks for each node.

The function `set_output_`S^* initializes the masks of nodes directly connected to the input of a latch or primary output to all 1s. The nodes are then ordered and traversed in reverse topological order as generated by `reverse_levelize`. The immediate fanout of each $node$ is then examined. The function `get_local_ODC` performs ODC analysis for every simulation vector for $node$ as defined by Equation 2 except only the sub-circuit defined by $node$ and $output$ is considered. This local ODC mask is bitwise-ANDed with $output$'s S^* and is subsequently ORed with $node$'s S^*.

The algorithm requires only a traversal of all the nets given by the two for_each loops and the K simulation vectors considered for each net in `get_local_ODC`, resulting in the $O(n'K)$ complexity. This algorithm enables our global ODC simulator to be more efficient than simply extending the local observability calculations in [16] to perform global ODC analysis.

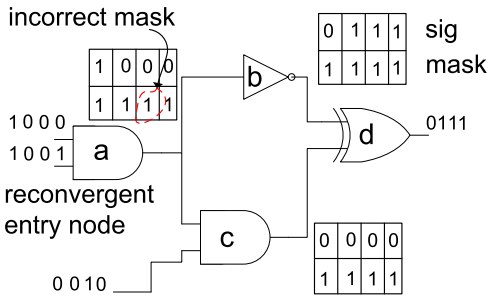

Fig. 4. The ODC information produced by approximate ODC simulation. Sometimes reconvergence can cause ODC simulation to produce incorrect ODC masks. S^* and S are shown for the internal nodes, and only S are shown for the inputs and outputs.

TABLE II
EFFICIENCY/QUALITY OF THE APPROXIMATE ODC SIMULATOR.

bench	runtime(s)			#cands	%pos	%neg
	sim	simodc	approx			
ac97_ctrl	1	6	1	63758	0.0	0.0
aes_core	2	79	1	315917	0.1	0.0
des_perf	9	410	7	296095	0.0	0.0
ethernet	4	76	2	8852009	0.3	0.8
mem_ctrl	1	119	1	867145	1.0	1.4
pci_bdge32	1	28	1	1158654	0.2	0.4
spi	0	39	0	156291	0.0	3.1
systemcaes	1	48	1	285189	0.2	0.2
systemcdes	0	24	0	5288	2.8	0.7
tv80	1	130	1	1348277	1.5	9.0
usb_funct	1	11	1	1685374	2.2	1.8
wb_conmax	3	69	4	1904773	0.0	0.0

B. Accuracy of Approximate Simulator

Because we do not consider logic interactions that occur because of reconvergence, it is possible for the algorithm in Figure 3 to incorrectly produce 0s (**false positives**) or 1s (**false negatives**) in S^*. For the example in Figure 4, node a misses a don't care (false negative) in the third bit of S_a^*. Notice that node b and c do not have any ODCs and no local ODCs exist between a and b or a and c, resulting in no ODCs being detected by the approximate simulator. However, the reconvergence of downstream logic makes the third value of node a a don't care. In a similar manner, false positives can occur when the interaction of multiple signals with local ODCs at a reconvergent node produces a flip at its output.

Incorrect simulation does not affect the correctness of the overall algorithm. When false negatives are produced, merger opportunities can be missed, resulting in less optimization. When false positives are produced, incorrect merger candidates will be later disproven by an equivalence checking tool, resulting in increased runtime. Our experiments show that these situations are rare.

C. Performance of Approximate Simulator

In Table II, the quality of the approximate ODC simulator is assessed. The first column indicates the benchmarks examined. The second column, *sim*, gives the time required to generate only signatures S for each node. We use this as a baseline to assess the cost of generating masks. The third column, *simodc*, shows the time required to generate S^* for each node using Equation 2. The fourth column, *approx*, shows the time to compute S^* using the approximate simulator. The last few columns show the number of ODC-substitutability candidates identified by the approximate simulator and the percent of candidates incorrectly found due false positives or missed due to false negatives.

The results indicate that the approximate simulator is comparable to that of *sim* and is much faster than *sim_odc*. In addition, the number of false positives and negatives is typically a small percentage of the number of opportunities identified. These results were generated by running 2048 random simulation vectors.

D. Refining Simulation

The quality of the simulation patterns used is integral in limiting candidates found to those that are valid mergers. We can prune several candidates that are later proven invalid by refining our simulation dynamically based on counter-examples derived from SAT similar to the approach in [16, 9].

When a merger is performed, the signatures in the fanout cone of the node being replaced could be different from the previous values, resulting in inaccurate don't care information. However, since signatures are used to find candidates that are later proven by a SAT solver, incorrect signatures can never lead to an incorrect merger and updates are not necessary.

V. INCREMENTAL VERIFICATION OF MERGER CANDIDATES

In this section, we outline a new approach to ODC-aware verification of node mergers which dynamically increases the logical depth of downstream logic so as to avoid unnecessarily large miters.

Figure 1(right) shows how ODCs can be identified for a given node in a network. We can prove whether b is ODC-substitutable to a in a similar manner. Instead of using a' in the modified circuit D^*, b is substituted for a and miters are constructed at the outputs. If the care-set determined by Equation 2 is null, b is ODC-substitutable to a and a merging opportunity exists. The whole care-set does not need to be derived as a single satisfiable solution proves non-equivalence.

For large circuits, proving ODC-substitutability could be prohibitive because all downstream logic is considered. We introduce a dynamic SAT framework that attempts to determine ODC-substitutability by considering a small portion of downstream logic. We can use the complexity of the verification procedure to explicitly limit the mergers considered.

Consider Figure 5, where b is a candidate of ODC-substitutability with a. If a miter is constructed across a and b instead of the primary outputs as shown in part a), a set of differences between a and b that results in satisfying assignments is given by $DIFFSET(a,b) = ONSET(a) \cap OFFSET(b) \cup OFFSET(a) \cap ONSET(b)$. A satisfying solution here indicates the non-equivalence for the given section of logic considered. If the satisfying solution is simulated for the remaining downstream logic and discrepancies between the two circuits exist at the primary outputs, then non-equivalence for the whole circuit is proven. If the $DIFFSET$ is null, substitutability is proven.

However, if a and b are very different, the $DIFFSET$ could result in a prohibitive amount of simulation. To reduce the size of the $DIFFSET$, we construct miters further from the merger site while reducing the amount of downstream logic considered. We introduce the notion of a **dominator set** to define where we place the miters.

Definition 7 *The dominator set for node-a is a set of nodes in the circuit such that every path from node-a to a primary output contains a member in the dominator set and where for each dominator member there exists at least one path from node-a to a primary output that contains only that member. Multiple dominator sets can exist for a given node.*

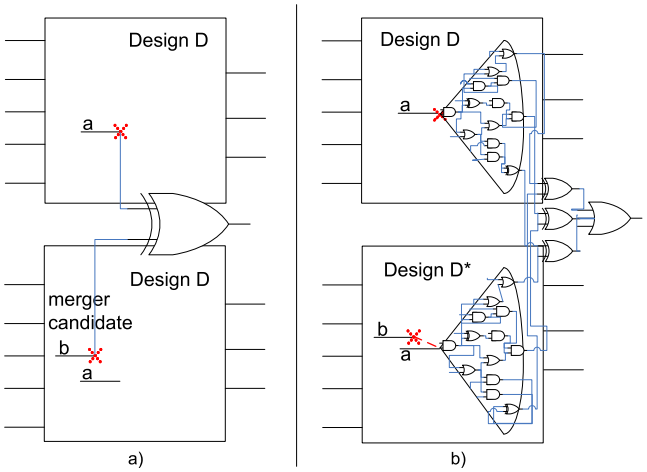

Fig. 5. An example that shows how to prove that b is ODC-substitutable to a. a) A miter is constructed between a and b to find test vectors that are incompatible. b) A dominator set can be formed in the fanout cone of a and miters can be placed across the dominators.

Algorithm: in part b) of Figure 5, we show miters constructed for a dominator set of a. Dominator sets close to the candidate merger will result in less complex SAT instances but potentially more downstream simulation to check whether satisfying assignments prove non-substitutability. We devise a strategy that dynamically moves the dominator set closer to the primary outputs depending on the number of satisfying assignments generated. Our moving-dominator algorithm is outlined in Figure 6.

```
bool odc_substitutable(a, b){
  current_dom = calculate_initial_doms();
  while(dom_Sat(miter(current_dom, a, b)) == SAT){
    if(simulation(satisfying_solution){
      return false;
    }
    else{
      current_dom = calculate_new_dom();
    }
  }
  return true;
}
```

Fig. 6. Determining the substitutability of two nodes in the presence of ODCs.

The moving-dominator algorithm starts by deriving a dominator set that is close to the merger site given by `calculate_initial_dom`. The details of this function will be given later. Then `dom_SAT` solves an instance where miters are placed across the current dominator set. An UNSAT solution means ODC-substitutability and the procedure exits. Otherwise, the satisfying solution is checked by simulating all downstream logic. If simulating the satisfying assignment does not result in an ODC at a, b is not ODC-substitutable to a. Otherwise, a new dominator set is generated as determined by `calculate_new_dom`. `calculate_new_dom` increases the logic considered and will be defined later.

With each invocation of the SAT solver, we add additional constraints indicated by the current dominator set. By incrementally building the SAT instance each time the dominator set is moved, we can reuse learned information and other useful heuristics between several SAT calls.

ATPG techniques can be easily substituted for the SAT-engine described in the previous algorithm. By placing a MUX with a dangling select input between the two nodes in the potential merger, we can generate test patterns for single-stuck-at faults (SSF) on the MUX select input. If a test pattern cannot be generated, the merger can take place because both nodes have the same effect on the outputs. The circuit considered can be limited by the dominator set, and a test pattern counter-example can be used to refine this dominator set. In our experiments, we work exclusively with SAT for two reasons. First, we have access to the API of a highly optimized SAT solver, but not an ATPG library. Second, the ability to share clauses over multiple SAT calls by incrementally building the instance is an advantage over ATPG approaches which typically do not share information between multiple calls.

Calculating Dominators: using simulation, we calculate a dominator set that tries to minimize the amount of downstream logic necessary to prove a merger. In general, we check the downstream logic required to prove specific ODCs for certain input combinations and use that to determine an initial dominator set. We, then, take counter-examples produced from the SAT solver to refine the dominator set. Details of this approach are given in the following.

In Figure 1(right), ODC(a) is derived by examining observability at the primary outputs. However, by placing miters along a cut defined between a and the primary outputs, it is possible to calculate an ODC-set for a, $ODC_{cut}(a)$, where $ODC_{cut}(a) \subseteq ODC(a)$. Previously, we defined this cut as the dominator set. An ideal dominator set would be the closest cut to the merger site sufficient to prove substitutability. We define the minimal dominator set as follows:

Definition 8 *The minimal dominator set D_{min} for proving that b is ODC-substitutable to a is the closest cut to a such that $DIFFSET(a,b) \subseteq ODC_{Dmin}(a)$.*

`calculate_initial_dom` is used to calculate an initial dominator set. We randomly select several input vectors X_i and approximately generate D_{min} using Definition 8 by constructing the sets, $DIFFSET(a,b)$ and $ODC(a)$, from the X_is. Since not all input vectors are considered, it is possible that the cut is an under approximation and results in the SAT solver reporting non-substitutability. To improve the approximation, `calculate_new_dom` extends the cut further from a for every satisfying assignment found by `dom_Sat`.

TABLE III
THE PERCENTAGE OF MERGERS PROVABLE WITH K LEVELS OF LOGIC.

benchmarks	K=1	K=2	K=3	K=4	K=5
i2c	36.7	53.3	60.0	66.7	80.0
pci_spoci_ctrl	21.6	51.5	67	84.5	93.8
alu4	13.2	26.9	35.2	42.6	50.1
dalu	9.9	14.3	19.8	31.9	38.5
i10	15.8	28.9	60.5	71.1	86.8
spi	60.9	82.6	91.3	95.7	100
systemcdes	26.1	38.7	60.4	74.8	86.5
C5315	87.5	87.5	87.5	87.5	87.5
C7552	36.0	64	64	68	72.0
s9234	0	0	20	20	40.0
tv80	11.9	23.4	38	49	56.3
systemcaes	21.6	45.8	70.5	72.8	73.9
s13207	13.3	46.7	60	80	93.3
ac97_ctrl	63.2	88.1	93.5	96.8	97.8
mem_ctrl	26.5	43	55.4	68.3	77.0
usb_funct	42.5	69.4	81.7	87.6	91.4
pci_bridge32	45.1	54.9	68.3	78	87.8
s38584	17.3	55.3	70.7	82	85.3
aes_core	9.7	15.4	22.9	31.6	42.3
s38417	12.1	54.5	78.8	100	100
wb_conmax	7.9	16.5	26	36.5	48.5
b17	21.4	30.4	35.7	42.4	44.2
ethernet	31.1	48.9	68.9	77.8	84.4
des_perf	16.8	27.4	39.4	55.7	74.0
average	27.0	44.5	57.3	66.7	74.6

VI. EXPERIMENTS

We implemented our algorithms in C++. The SAT engine was developed by modifying MiniSAT [3]. Random simulation patterns are used to generate the initial ODC signatures. The circuits are from the IWLS 2005 suite produced from OpenCores designs generated by a quick synthesis run of the Cadence RTL Compiler [15]. We perform our ODC-based node merging algorithm by examining each node in a circuit in one topological traversal. The tests were run on Pentium-4 3.2 GHz machines.

For combinational simulation and equivalence checking, we treat the latches as inputs. Every internal node with an ODC-set that is not null is examined. If an ODC-substitutability is detected for the node, a merger is made. We ignore mergers that increase the number of logic levels in the design. The results were verified using the equivalence checking tool in ABC [1].

A. ODC Locality

In this section, we show that several levels of downstream logic are often needed to prove ODC-substitutability. Because of our efficient simulation and incremental verification technique, we can enhance the local ODC analysis performed in [16] by considering node mergers of unbounded depth.

In Table III, we show the percentage of mergers that can be proven using K levels of downstream logic. The benchmarks were synthesized using ABC [1] without performing technology mapping. The results indicate that most mergers are proven using only a few levels of logic. However, on average, 25% of mergers are missed by limiting the window of computation to five levels which the maximum considered in [16].

We expect that more levels of logic in a design will increase the importance of computing non-local don't cares. In bounded model checking, circuit simplification is important to increase the efficiency of verification. In physical synthesis, finding sequential don't cares for more aggressive optimization could require unrolling a circuit several times. We therefore re-examine the strategy pursued in [16] that bounds the depth of ODCs to a few levels of downstream logic. A

TABLE IV
THE PERCENTAGE OF MERGERS PROVABLE USING K=5 LEVELS OF LOGIC FOR CIRCUITS UNROLLED 1-5 TIMES.

benchmarks	1	2	3	4	5
i2c	80.0	57.0	42.8	43.1	43.2
s9234	40.0	51.4	42.0	38.2	42.9
pci_spoc_ctrl	93.8	87.8	86.5	84.8	84.5
systemcdes	86.5	85.3	88.7	86.2	86.3
spi	100	70.7	71.7	64.6	67.5
ac97_ctrl	97.8	83.2	64.2	46.6	38.9
average	83	72.6	66.0	60.6	60.6

TABLE V
AREA REDUCTIONS ACHIEVED BY PERFORMING THE ODC MERGING ALGORITHM AFTER THE ABC REWRITING ALGORITHM [8]. OUR ALGORITHM HAS A TIMEOUT OF 5000 SECONDS AND IS DENOTED BY ☹.

benchmarks	Orig(s)	+ODC(s)	#merge	%area_reduct
i2c	0	3	30	3.2%
pci_spoci_ctrl	0	6	97	9.2%
alu4	1	64	469	22.9%
dalu	0	10	91	12.0%
i10	1	12	38	1.3%
spi	1	84	23	1.3%
systemcdes	1	9	111	4.7%
C5315	0	2	8	0.7%
C7552	1	8	25	3.4%
s9234	0	8	10	1.2%
tv80	3	1445	606	7.1%
systemcaes	4	360	518	3.8%
s13207	1	17	15	1.8%
ac97_ctrl	3	188	185	2.0%
mem_ctrl	5	738	1797	18.0%
usb_funct	5	681	186	1.4%
pci_bridge32	6	1134	82	0.1%
s38584	2	223	150	0.8%
aes_core	9	1620	2144	8.6%
s38417	2	275	33	1.0%
wb_conmax	19	☹	2433	6.2%
b17	6	☹	224	1.6%
ethernet	28	☹	45	1.4%
des_perf	50	☹	3148	3.7%
average				4.9%

key question is by how much it reduces the number of possible mergers. Table IV shows the percentage of mergers proven using five levels of logic for six different circuits unrolled up to five times. We, again, simplify each unrolled circuit using rewriting before checking for potential mergers. For a few of the benchmarks, the percentage of mergers found using local ODC computation decreases significantly as a function of the number of times the circuit is unrolled as with *ac97_ctrl*.

Because of the scalability and accuracy of our simulator and incremental verification, we can quickly find mergers requiring few levels of downstream logic, while still being able to find more complicated mergers prevalent in larger designs.

B. Post-Synthesis Optimization

In this section, we show that our global ODC analysis discovers node mergers even after full-fledged synthesis optimizations [1, 8]. These additional reductions can be easily performed in conjunction with layout information to help achieve design closure.

In Table V, we evaluate our ODC merging algorithm on unmapped circuits after synthesizing them using local rewriting through the resyn2 script defined in the synthesis package ABC [1, 8]. The column, `area_reduct`, gives an estimated area reduction for ODC merging after rewriting by mapping the final result. Despite the quality of rewriting, we see that benchmarks can still be optimized with improvement over 10% in some cases. These results illustrate that our strategy is sufficiently strong for post-synthesis optimizations.

VII. CONCLUSIONS

The increasing impact of interconnect on design performance necessitates aggressive physical synthesis optimizations. Current state-of-the-art synthesis strategies tend to be local in nature. Although significant optimization is reported in [16] that considers up to 5 levels of logic in AIGs, this improvement would typically correspond to fewer logic levels in mapped circuits resulting in less optimization in the physical synthesis domain. Therefore, we present a node merging strategy that can operate directly on mapped netlists. Unlike the work in [16], our techniques pursue global ODCs and are successfully evaluated against logic synthesis transformations. By exploiting global don't cares we identify several node mergers even after extensive synthesis optimizations, resulting in up to 23% area reduction. Furthermore, our techniques are not restricted to mapped circuits and can be used directly on AIGs in sequential verification applications. In this context, global ODC analysis becomes more important because of the greater depth in unrolled circuits.

A key contribution of our work is our strategy to avoid unnecessary computation of ODCs in logical synthesis, verification, and physical synthesis while maximizing optimization strength. This is accomplished through the efficient use of simulation-based signatures and on-demand SAT-based equivalence checking which considers only as much downstream logic as necessary. These observations enable scalable ODC analysis of unbounded depth that discovers up to 60% more node mergers than local ODC analysis.

REFERENCES

[1] Berkeley Logic Synthesis and Verification Group, "ABC: a system for sequential synthesis and verification", `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[2] D. Brand, "Verification of large synthesized designs", *ICCAD '93*, pp. 534-537.

[3] N. Een and N. Sorensson, "An extensible SAT-solver", *SAT '03*, `http://www.cs.chalmers.se/~een/Satzoo`.

[4] Z. Fu, Y. Yu, and S. Malik, "Considering circuit observability don't cares in cnf satisfiability", *DATE '05*, pp. 1108-1113.

[5] E. Goldberg, M. Prasad, and R. Brayton, "Using SAT for combinational equivalence checking", *DATE '01*, pp. 114-121.

[6] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification", *IEEE Trans. CAD '02*, pp. 1377-1394.

[7] F. Lu, L.-C. Wang, K.-T. Cheng, J. Moondanos, and Z. Hanna, "A signal correlation guided circuit-SAT solver", *J. UCS 10(12) '04*, pp. 1629-1654.

[8] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", *DAC '06*, pp. 532-536.

[9] A. Mishchenko, S. Chatterjee, R. Jiang, and R. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification", *ERL Technical Report*, Berkeley. `http://www.eecs.berkeley.edu/~alanmi/publications/`.

[10] A. Mishchenko and R. Brayton, "SAT-based complete don't care computation for network optimization", *DATE '05*, pp. 412-417.

[11] A. Mishchenko et al, "Using simulation and satisfiability to compute flexibilities in Boolean networks", *TCAD '06*, pp. 743-755.

[12] N. Saluja and S. Khatri, "A robust algorithm for approximate compatible observability don't care computation", *DAC '04*, pp. 422-427.

[13] H. Savoj and R. Brayton, "The use of observability and external don't-cares for the simplification of multi-level networks", *DAC '90*, pp. 297-301.

[14] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability", *IEEE Trans. Comp '99*, pp. 506-521.

[15] `http://iwls.org/iwls2005/benchmarks.html`.

[16] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't cares", *DAC '06*, pp. 229-234.