This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

m-Inductive Property of Sequential Circuits

Hamid Savoj, Alan Mishchenko, and Robert Brayton, Fellow, IEEE

Abstract—**This paper introduces *m*-inductiveness over a set of nodes *S* in sequential circuits. The *m*-inductive property can be used for equivalence-checking or improved sequential optimization. It allows the behavior of many next-state functions (not in *S*) to be changed while maintaining correctness at the primary outputs of a circuit. As such, it creates flexibility that can be used for sequential optimization. It is shown that the number of nodes in *S* is reduced as *m*, the parameter for the inductiveness, increases. We provide an algorithm for finding a minimal set *S*, as well as one for using *m*-inductiveness in optimization. We give examples of such optimized circuits and show that *m*-inductive based optimization can result in significant area reduction when applied to industrial designs.**

***Index Terms*—Formal verification, sequential synthesis, synthesis for low power, and logic synthesis.**

I. Introduction

A sequential circuit, depicted in Figure 1 is composed of a combinational logic part (A^1) and a set of memory elements (flip-flops (FF), or flops), a set of primary inputs (*PI*), and a set of primary outputs (*PO*). A^1 is a Boolean network which has inputs composed of the *PIs* plus the current states (*CS*) of the FFs, and outputs composed of the *POs* plus the next-state functions (*NS*). The values of the *NS* functions are stored in the FFs on the next clock edge.

The combinational part A^1 is a directed acyclic graph composed of Boolean nodes. Each Boolean node has a local Boolean function f_i in terms of its fanins and a Boolean variable y_i such that $y_i = f_i$. Each Boolean node also has a global function in terms of PIs and current state variables. The Boolean nodes that fan out to sequential elements are *next-state nodes* and their global Boolean functions are *next-state functions* (*NS*). The Boolean nodes that fan to the outside of the circuit are *POs*. All other Boolean nodes in A^1 are referred to as *intermediate nodes*.

We assume that the FFs start in a known initial state. Two sequential circuits, *A* and *B*, are said to be equivalent (denoted $A = B$) if, starting from their respective initial states, for any given (infinite) sequence of *PIs*, the generated (infinite) sequences of the two associated *POs* are identical. A Boolean node of *A* is equal to a Boolean node of *B* for *m* cycles if logic values computed at those two nodes are equal for all allowed Boolean values of *PIs* for *m* cycles.

If only combinational don't cares are used for optimization to convert a circuit A^1 to B^1, sequential equivalence checking (A=B) consists of just checking the equivalence of their combinational parts ($A^1 = B^1$). In contrast, sequential synthesis may use information that can change the behavior and number of the NS functions, but always maintains the observed behavior at the POs. This information can lead to (a) using the unreachable states as don't cares, (b) using the equivalence of two states, (c) using a different state-encoding for the reachable states, (d) retiming the FF's, and (e) merging two signals if they have the same values in all reachable states.

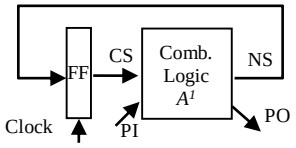


Figure 1: A Sequential Circuit *A*

In this paper, we introduce the *m-inductiveness* property between two sequential machines. This property is similar to general induction but lacks the initial condition constraint. Two machines are *m-inductive* when equality of a set of Boolean node pairs for *m* cycles, implies equality of that same set of node pairs in the next cycle. In order to achieve sequential equivalence, the paired nodes always include all of the primary outputs (*POs*) of the two sequential machines. A subset *S* of the next state nodes or intermediate nodes might also be paired. There are cases where *POs* suffice and *S* is an empty set.

This definition of *m-inductiveness* property and its application for verification and optimization is a generalization of the results in [18], where it is required that all next-state nodes be in *S*. The concept of *m*-inductiveness should not be confused with a *k*-step inductive proof [22]. It is merely a local property and not an unbounded proof of anything.

The *m-inductive* property can be used to verify that two sequential machines are equal. This is done either by induction where *m-inductiveness* and the initial conditions are used to prove equivalency. Or, it can be used to convert a sequential equivalence checking (SEC) problem into a combinational one (CEC).

The *m-inductiveness* property can also be used to compute larger don't care sets for logic optimization, and to change the global function of many next-state nodes while preserving the correct behavior at the *POs*. One of the two sequential machines used in the definition of *m-inductiveness* is typically the template or the "golden" circuit (specification). When applying logic optimization, the initial starting circuit is copied from the template circuit. After optimization is applied, the *m-inductive* property of the *POs* and nodes in *S* between the

Hamid Savoj is a visiting scholar with the Electrical Engineering and Computer Sciences Department, University of California at Berkeley, Berkeley, CA 94720 (phone: (650) 218-3019; e-mail: hamid@savojsolutions.com).

Alan Mishchenko is with the Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, CA 94720 (e-mail: alanmi@eecs.berkeley.edu).

Robert Brayton is with the Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, CA 94720 (e-mail: brayton@eecs.berkeley.edu).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

optimized and template circuit must hold. The next state functions that are not in *S* can change in a way that makes them combinationally different from the template circuit. This new flexibility extracted from induction has not been explored before and it results in better logic optimization.

The organization of this paper is as follows. Section II discusses combinational and sequential don't cares. This section is provided to distinguish our work from previous publications where unreachable state information is computed and used as external don't cares to optimize a sequential circuit. The unreachable state information is usually computed using inductive algorithms but this is different from the *m*-inductiveness concept introduced in this paper.

Section III defines *m*-inductiveness, and proves two theorems, one gives a sufficient condition for *m*-inductiveness and the other states how it can be used for proving sequential equivalence. Section IV provides an algorithm for finding a minimal set *S* of next state nodes for a circuit to be *m-inductive* under *S*. The set *S* is minimal in the sense that all nodes in *S* are needed to maintain the *m-inductive* property. In other words, any redundant node that can be removed has been removed already. The algorithm removes redundant nodes in *S* one at a time and is order-dependent. Finding the smallest possible set *S* is useful because all the next state nodes that are not in *S* can be changed within the allowed *m-inductive* flexibility and are not restricted to have their original global Boolean functions. The minimal set *S* is not unique and there may be many possibilities for choosing *S*. Our algorithm provides an approach for finding a set *S* that results in a good sequential optimization. In our approach for finding *S*, global optimality can be claimed when *S* is empty and thus *m-inductiveness* depends only on *POs*.

Section V discusses applications of *m*-inductiveness. In particular, it shows how *m*-inductiveness can (a) be used to extend the work in [18] in converting SEC to CEC, (b) result in larger sequential observability don't cares, (c) be used in optimizing pipelines, and (d) lead to transformations which result in retiming.

Finally, Section VI shows two sets of experimental results applied to ISCAS and industrial designs. In the first, we apply the algorithm of Section IV to find a minimal set *S* of FFs for *1*-inductiveness (*m* = 1), and discuss the resulting sizes of *S* relative to the number of FFs. In the second, we use the additional flexibility provided by *S* for sequential synthesis to derive smaller sequential machines. Section VII concludes the paper.

II. DON'T CARE METHODS FOR SEQUENTIAL SYNTHESIS

A. Combinational Don't Cares

Combinational synthesis changes only the combinational circuit A^1 of a sequential machine, *A*, but keeps the *PO*s and *NS* functions, as functions of *PIs* and *CS* variables, unchanged. The Boolean network A^1 might be completely restructured as a result of combinational synthesis. Don't care conditions in A^1 can be used during synthesis to find a better implementation of *A*. Don't cares are classified as satisfiability don't cares (SDCs), observability don't cares (ODCs), and external don't cares (EDCs) [2][19][17]. SDCs and ODCs are related to the structure of A^1 while EDCs are computed either from the sequential behavior of *A* (e.g. unreachable states), extracted from a hierarchy, or supplied by the user as a model of the environment in which the circuit is to be used [20]. The *NS* functions and *POs* can change if external don't cares are used during optimization.

Definition 1: *A Boolean Network A^1 is a directed acyclic graph (DAG) such that for each node in A^1 there is an associated representation of a Boolean function f_i, and a Boolean variable y_i, where $y_i = f_i$. There is a directed edge (i, j) from y_i to y_j if f_j explicitly depends on y_i. The global function g_i at y_i is the function at the node expressed in terms of PI and CS variables.*

Definition 2 (SDCs): *If y_i is the variable at an intermediate node of A^1 and f_i its logic function, then $y_i = f_i$; therefore, we don't care if $y_i \neq f_i$ (i.e. $y_i \oplus f_i$ is don't care). The expression $SDC = \sum_{i=1}^{p} (y_i \oplus f_i)$ is the satisfiability don't care set of A^1 where p is the number of intermediate nodes.*

Simulation of A^1 with Boolean values applied to each of PI and CS variables in the Boolean space $\mathbf{B^n}$, where *n* is the total number PI and CS variables (inputs of A^1), uniquely sets the values of the *p* intermediate nodes to 0 or 1. Some combinations of values on the intermediate variables are not possible. The SDC of A^1 contains all the impossible combinations in the Boolean space $\mathbf{B^{n+p}}$.

Definition 3 (ODCs): *The observability don't cares (ODCs) at each intermediate node y_i of a multi-level network are input minterms (PI and CS valuations) under which y_i can be toggled between* 0 *and* 1 *without causing any of the POs or NS functions to toggle.*

The ODC of an intermediate node y_i can be expressed also in terms of intermediate nodes of the network. For example, the ODC at node y_2 in Figure 2 is $ODC_2 = y_1y_3 + \bar{y}_1\bar{y}_3$. This means that whenever y_1 and y_3 are both 1 or both 0, the value of y_2 has no effect at any of the outputs. The same ODC in terms of inputs is $ODC_2 = x_1x_2 + \bar{x}_1\bar{x}_2 + \bar{x}_3$. The ability to express the ODCs in terms of various sets of intermediate variables is important to node optimization.

An interesting fact is that the ODC at a node can intersect the SDC of the network [19]. In Figure 2, $y_1y_2y_3 + y_1\bar{y}_2y_3$ is in ODC_2 while $y_1y_2y_3$ is also in the SDC because y_1 and y_2 can't be 1 at the same time under any input combination.

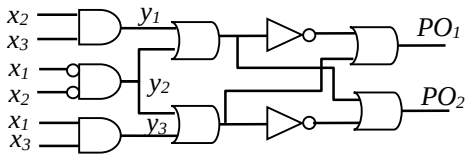


Figure 2: ODCs intersect SDC

Definition 4 (EDCs): *The external don't care (EDC) set for each PO of A^1 captures all input minterms (valuations) for which the value of that output is not important.*

EDCs can come from many sources. If circuit A^1 is cut out of a larger circuit as shown in Figure 3 (e.g. a module in a hierarchical design), some input minterms may not be possible because they are in the SDCs of the larger circuit. In this case,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

inputs of A^1 are intermediate nodes of the larger circuit and they may not produce all combinations of minterms.

The EDCs can come from the ODCs of the larger circuit. Each PO of A^1 is an intermediate node in the larger circuit and a change in that PO may not be visible in the outputs of the larger circuit for certain input minterms of *c*. If EDCs of A^1 come from the ODCs of intermediate nodes of a larger circuit, they may not be *compatible*. Incompatible don't cares are usually not valid when there is a modification in the circuit A^1 and must be recomputed.

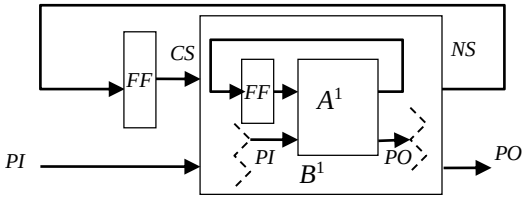


Figure 3: Extracting EDCs

B. *Don't Care Methods for Sequential Synthesis*

Our discussions use the following notation:

1. $\mathbf{B^r}$ *denotes the full Boolean space of the r state variables in a machine A.*
2. T_0^A *denotes a subset of* $\mathbf{B^r}$ *and* T_k^A *denotes its image after k cycles of machine A, i.e.* $s \in T_k^A \Leftrightarrow \exists s_0 \in T_0^A \wedge \exists PI$ *sequences applied to A that reaches s in k cycles.*
3. $[A=B]_I$ *denotes that machine A is equivalent to B for all initial states* $s \in I$.

The unreachable states of a machine *A* is the set of states that are not reachable starting from any initial state. Since such states are never reached, they can be used as don't cares to optimize A^1 into B^1, and hence into an equivalent sequential machine *B*. They can be viewed as external don't cares (EDCs) of A^1.

One way to compute the unreachable state set is to start from the initial state set ($I \equiv T_0^A$) and recursively compute the images T_i^A under *A* until no new states are reached, i.e. $\sum_{i=0}^{k+1} T_i^A = \sum_{i=0}^{k} T_i^A$. However, the number of iterations *m* needed to achieve this fixed point and the size of the reached state set can be exponentially large, so this method is rarely used in practice.

A superset of reachable states is provided by an "invariant" $T_1^A \subseteq T_0^A$ where $T_0^A \subset \mathbf{B^r}$. The condition $T_1^A \subseteq T_0^A$ is enough to guarantee that $T_{k+1}^A \subseteq T_k^A, \forall k \geq 0$ as shown by Lemma 1.

Lemma 1: *If* $T_1^A \subseteq T_0^A$ **,** *then* $T_{k+1}^A \subseteq T_k^A, \forall k \geq 0$ **.**

Proof: Clearly, the base case holds, $T_1^A \subseteq T_0^A$. Assume $T_k^A \subseteq T_{k-1}^A$, but $T_{k+1}^A \not\subseteq T_k^A$. Then $\exists s \in T_k^A$ whose image $s' \notin T_k^A$. But $s \in T_{k-1}^A$ and therefore $s' \in T_k^A$, a contradiction. Therefore, by induction, $T_{k+1}^A \subseteq T_k^A, \forall k \leq 0$. **QED**

Therefore, if T_0^A includes all initial states and $T_1^A \subseteq T_0^A$, then T_0^A is superset of the reachable states and its complement can be used as EDCs to simplify A^1.

If one starts with $T_0^A = \mathbf{B^r}$ (the complete Boolean space), Lemma 1 applies and $T_{k+1}^A \subseteq T_k^A, \forall k \geq 0$. As long as T_n^A includes the initial states *I*, it is a superset of the reachable states and can be used to optimize *A.* One way to achieve this optimization is to build the unrolling A^{n+1}, and use the first *n* copies of A^1 to optimize the last copy of A^1 as shown in Figure 4. The set of the SDCs at the *CS* inputs of the last copy of A^1 is the complement of T_n^A . But, it is still necessary that $I \subset T_n^A$.

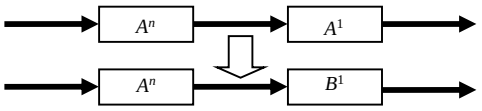


Figure 4: Optimization using SDCs from A^n to simplify A^1 resulting in B^1.

A known approach (and very effective way) for computing a superset of reachable states is "signal correspondence" [22] [3]. It effectively computes such a set by *k*-step induction. Machine *A* is simulated for many clock cycles starting from an initial state. Signals with equal, complementary, or constant values throughout the simulations are put in equivalence classes and postulated to be always in the same relation on the set of all reachable states. These are checked for the first *k* steps as the base case for induction. For the inductive case, they are assumed to hold for *k* cycles. Then an attempt is made to prove that all hold on the next cycle. If not all can be proved, the set is trimmed to only those that could be proved. This is iterated until a fixed point is found, i.e. no more equalities can be disproved. The inductive step can be done efficiently using SAT, hence this method is practical for large machines. Note that a superset of reachable states is given implicitly as the set of states on which the equalities hold. Optimization of the sequential circuit is done by merging the signals proven equal, setting the constant nodes to their values, and propagating the constants.

A more recent method, discussed in [18], is to optimize a sequential circuit, *A*, using a *k*-step unrolling process as indicated by C_1 in Figure 5. $C_1 = A^k$ is a combinational circuit, which has *k* copies of A^1. The outputs of C_1 are the *k* PO sets of *A*, one set for each time-frame plus the final *NS* nodes at the output of the k^{th} frame. The inputs of C_1 are the *k* PI sets, one set for each frame, plus the initial *CS* signals of the first frame.

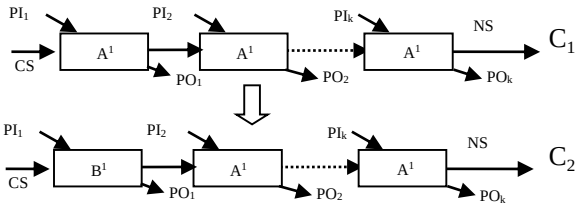


Figure 5: *k*-step unrolling, ODC optimization of A^1

Then *combinational* synthesis is applied to the logic of the first copy of A^1 and the other *k*-1 copies are left untouched, resulting in the combinational circuit $C_2 = B^1A^{k-1}$ (bottom of Figure 5). $C_2 = B^1A^{k-1}$ is a combinational circuit, which has one copy of B^1 followed by *k-1* copies of A^1. Although the global functions at the *NS* outputs of the *final* copy are necessarily preserved by the combinational synthesis, the internal *NS* signals of B^1 as well as the successive *NS* signals of the *k*-2 copies of A^1 in C_2 may be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

functionally different from the corresponding internal *NS* signals in C_1.

The purpose of the last *k*-1 copies of A^1 in this scenario is to produce Observability Don't Cares (ODCs) used for transforming the first copy of A^1 into B^1, which will form the combinational part of a new sequential circuit *B*. This sequential circuit is shown in Figure 6.

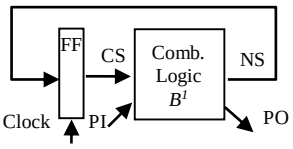


Figure 6: Derived Sequential Circuit *B*

Since the changes in B^1 are not observable at any of outputs of C_1, therefore, $B^1A^{k-1} = A^k$. It is proved in [18] that this combinational equivalence implies the sequential equivalence $[A = B]$, for any initial state $s_0 \in \mathbf{B^r}$.

III. *M*-INDUCTIVE PROPERTY

Figure 7 shows two sequential circuits *A* and *B* and a set *S* of corresponding node pairs in machines *A* and *B*. Both circuits have the same PIs and POs but can have a different number of flip-flops. Nodes in *S* can be next-state or intermediate nodes. Figure 7 also shows two *m*+1 step unrolling's of *A* and *B* for cycles *n-m* to *n*. The starting states x_{n-m} and y_{n-m} at time *n-m* may be different in the two unrollings. Let the corresponding nodes in the set *S* and the POs in both unrolling's be equal at cycles *n-m to n-1* for any valid combinations of PIs. A Boolean node of *A* is equal to a Boolean node of *B* for *m* cycles if logic values computed at those two nodes are equal for all valid *PIs* for *m* cycles. The *m*-inductive property of *A* and *B* under *S* states that the corresponding nodes in *S* and POs at time *n* are also equal for any valid PIs (i.e. independent of the initial states).

Definition 5 (*m-Inductiveness*): *Given two machines A and B with the same primary inputs and outputs, let S be a set of paired nodes (one of a pair from A and the other one from B), which may include intermediate as well as NS nodes. A and B have the* m-inductive property under *S, if the equality of the corresponding nodes of S and POs in A and B at cycles n-m to n-1 imply those node and PO pairs are also equal at time n.*

Typically *A* and *B* are related, e.g. *B* might be a synthesized version of *A*. If *A* and *B* are combinationally equivalent, then they are 1-inductive under the set of all *NS* node pairs. To see this, suppose all the *NS* and PO pairs of the two machines *A* and *B*, are equal at time *n-1*. Since the machines have the same set of inputs, all signals feeding *A* and *B* at cycle *n* are equal. Since the two machines are combinationally equivalent, the *NS* pairs and POs at time *n* are also equal.

Notice that there is no restriction on the initial states of the two machines. Therefore, two copies of the same machine *A* starting from different initial states are 1-inductive under the set of all *NS* node pairs. In this case we say that *A* is 1-inductive to itself under *S*, or just *A* is 1-inductive under S.

It is usually desirable that *S* is as small as possible. Finding the smallest possible set *S* is useful because all the next state nodes that are not in *S* can be changed within the given *m-inductive* flexibility and are not restricted to have their original global Boolean functions.

There are three ways to reduce the size of *S* but maintain the *m-inductive* property. The first is to look for a subset of *NS* node pairs. In Section IV, an algorithm is presented to find a minimal set of next-state nodes that maintains the *m*-inductive property of a machine *A* to itself. The second way is to try to replace some or all of the next-state nodes by intermediate nodes. The third is to use $m > 1$ cycles in the inductive property.

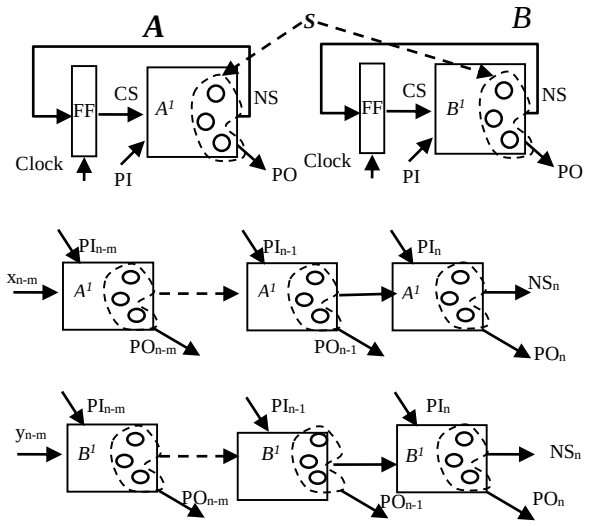


Figure 7: *m*+1 unrolling of *A* and *B* with different incoming states.

Example 1: Figure 8 shows a circuit where equality of the output *q* at time *n*-1 for two copies of Machine *A* implies the equality of output *q* for two copies at time *n*. This is because q_n can be expressed in terms of q_{n-1} and the PIs; there is no dependency on the current state variables:

$$q_n = r_{n-1}\left(e_{n-1}s_{n-1}t_{n-1} + \bar{e}_{n-1}CS1_{n-1}CS2_{n-1}\right) = r_{n-1}\left(e_{n-1}s_{n-1}t_{n-1} + \bar{e}_{n-1}\, q_{n-1}\right)$$

In this case, *S* is empty; hence this machine is 1-*inductive* under the empty set.

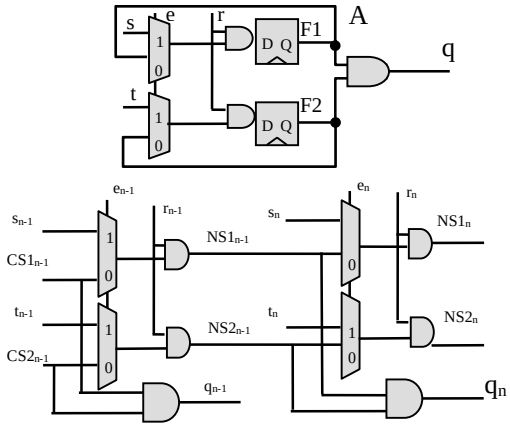


Figure 8: 1-inductive when *S* is an empty set.

A sufficient condition for *m*-inductiveness under *S* is stated by Theorem 1.

Theorem 1: *Machine A is m-inductive under set S if the nodes in S and POs at time n only depend on a) the values of the nodes in S and POs at times n-m to n-1, and b) the PIs at times n-m to n.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Proof: Suppose we have two copies of A, unrolled for *n* cycles. These copies share the same PI inputs. When the corresponding nodes in *S* and outputs in the two copies are equal from times *n*-*m* to *n*-1. Then those nodes are also equal at time *n* because those nodes depend only on the values of nodes in *S* and outputs from times *n*-*m* to *n*-1 and the PI inputs from times *n*-*m* to *n*. Therefore, *A* is *m*-inductive under *S*. **QED**

Note that the theorem holds if there are two different machines, *A* and *B*, but the stated dependencies are the same for both.

Example 2: Figure 9 shows an *m*-stage pipeline circuit without feedback loops. This may be a sub-circuit of a larger Boolean circuit. Each stage of the pipeline has its own PIs and POs. There is a final set of POs after Stage *m*. If extracted from a larger circuit, all the loops and all signals coming from or going to the rest of the circuit have been cut to form new primary inputs and outputs. In this case, the Boolean functions of the outputs of the pipeline at time *n* can be described in terms of PI's from cycles *n*-*m*+1 to *n*. This description is independent of the state variables. Therefore using Theorem 1, machine *A* is *m-inductive*. *S* is an empty set and *m-inductiveness* is based on equality of *POs*.

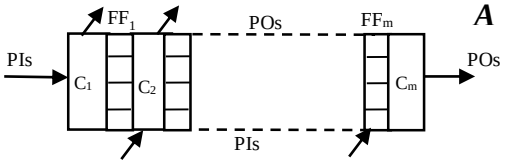


Figure 9: *m*-stage pipeline

Theorem 2: *Given two machines A and B that are m-inductive under the corresponding nodes S, if all corresponding nodes in S and POs of the two machines are also equal for the first m cycles starting from their respective initial states, then the two machines are sequentially equivalent.*

Proof: The proof is by induction. The two machines are equal for the first *m* cycles starting from the respective initial states. Also, equality of corresponding nodes in *S* and POs for *m* cycles produces equivalence of nodes in *S* and outputs for both machines on the next cycle. Thus, the machines are equivalent by induction. **QED**

Example 3: In this example, we show how to prove sequential equivalence between two machines by analyzing the dependency structures and using *m*-inductiveness with *m* = 4. Figure 10 shows two implementations of a counter where both produce the sequence 000100010001… at the *q* outputs.

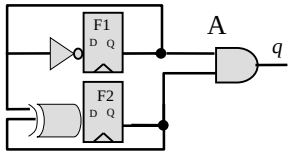
Machine *A* is a two-bit counter initialized at *F*1=0 and *F*2=0. The 4-stage unrolling of machine *A* is shown in the expressions below. There are two different formulas for writing the output *q* in terms of outputs in previous cycles. These are obtained by repeatedly expanding each variable using its logic expression. Using $F1_n = \overline{F1_{n-1}}$ and $F2_n = F1_{n-1} \oplus F2_{n-1}$ we obtain,

$$q_n = F1_n F2_n = \overline{F1_{n-1}}(F1_{n-1} \oplus F2_{n-1}) = \overline{F1_{n-1}} F2_{n-1}$$
$$= F1_{n-2}(F1_{n-2} \oplus F2_{n-2}) = F1_{n-2}\overline{F2_{n-2}} = \overline{F1_{n-3}}(F1_{n-3} \overline{\oplus} F2_{n-3}) = \overline{F1_{n-3}}\ \overline{F2_{n-3}}$$
$$= F1_{n-4}(F1_{n-4} \overline{\oplus} F2_{n-4}) = F1_{n-4} F2_{n-4} = q_{n-4}$$

Thus it is 4-inductive and *S* is an empty set.

Machine *B* is a ring counter initialized at *F*1=1, *F*2=0, *F*3=0, and *F*4=0. A 4-stage unrolling of machine *B* shows that it is 4-inductive where *S* is an empty set and $q_n = q_{n-4}$. Given the equal formulae $q_n = q_{n-4}$ for both machines, if the two machines produce similar outputs in the first 4 cycles, then by Theorem 2 they produce equal outputs thereafter, i.e. *A*=*B*. This can be checked in Figure 10.

cs2	cs1	ns2	ns1	q
0	0	0	1	0
0	1	1	0	0
1	0	1	1	0
1	1	0	0	1



cs4	cs3	cs2	cs1	ns4	ns3	ns2	N1	q
0	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	1

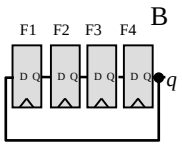


Figure 10: *4-inductive* circuit where *S* is an empty set.

IV. FINDING M-INDUCTIVE SETS

We describe a method for finding a minimal subset *S* of next-state functions of a sequential circuit to make it *m-inductive* (with itself) under *S*. The set *S* is minimal in the sense that all nodes in *S* are needed to maintain the *m-inductive* property, i.e. no redundant nodes remain. The algorithm removes redundant nodes in *S* greedily, one at a time, and is order-dependent. The minimal set *S* obtained this way is not unique; there may be many possibilities for choosing *S*. Our algorithm finds a set *S* that results in a good sequential optimization. When *S* is empty and *m-inductiveness* depends only on the POs, the Boolean function of all FFs can change and only POs are maintained.

Consider Figure 7 showing the unrolling of two copies of a sequential circuit *A* for *m*+1 cycles (copies are called *A* and *B* in this discussion). This unrolling illustrates the construction of an instance of Boolean satisfiability (SAT), which is used to compute *S*. To construct the SAT instance, $2\times(m+1)$ copies of the Conjunctive Normal Form (CNF) are used for the combinational logic of the sequential circuit, representing *m*+1 cycles of the unrolling of the two identical sequential circuits. Each FF is interpreted as a pair of nodes, one being the FF input, the other the FF output. The algorithm finds a set *S* which is a subset of the FF inputs.

We also use an auxiliary variable associated with the next-state function of each FF. It is 1 if and only if the corresponding next-state function belongs to *S*. In addition, the following "glue" constraints are added to the SAT instance.

1. Clauses forcing equality of FF outputs and FF inputs for both copies of *A*. These force equality of FF outputs at cycle *i*+1 with flop inputs at cycle *i*, $1 \le i \le m$.
2. Clauses forcing equality of corresponding primary outputs in *A* and *B* for any cycle *i*, $1 \le i \le m$.
3. Clauses forcing equality of corresponding nodes in *S* for *A* and *B* for any cycle *i*, $1 \le i \le m$.
4. Clauses forcing non equality of at least one pair of primary outputs or nodes in *S* for *A* and *B* in cycle *m*+1.

The "glue" constraints are shown in Figure 11 for the case of *m*=1, with the number in parentheses indicating the constraint

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

type. In Figure 11, *NS(S)* represents the next state functions of FFs in *S* while *NS(FFs-S)* represents the next state functions of FFs not in *S*. The bottom construction in Figure 11 stands for the cycle where equality of POs and next-state nodes belonging to *S* is assumed while the top construction stands for the cycle where that equality is checked. To reduce the clutter, the auxiliary variables are not shown. They are used to enable equality constraints. The same variable is used for each FF in all timeframes.

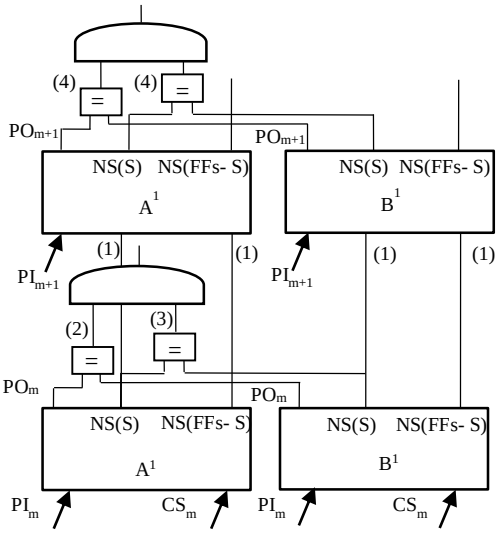


Figure 11: Construction of SAT instance for m-inductiveness.

Auxiliary variables are used only in constraint types 3 and 4. Specifically in constraint type 3, each equality of the corresponding pair of *S*-nodes (n_A, n_B) from *A* and *B* at time *i* is written as two implications $n_A \Rightarrow n_B$ and $n_B \Rightarrow n_A$ with the additional enabling auxiliary variable *h*, resulting in two clauses: $(\bar{n}_A + n_B + \bar{h})(n_A + \bar{n}_B + \bar{h})$. Thus, the implications are forced to hold if $h = 1$; otherwise, these clauses are immediately satisfied and the equality of the pair of *S*-nodes is not required. A similar construction is used in generating constraints 4.

This SAT instance can detect if a given set *S* of next-state functions is *m-inductive*. For this, we use a set of assumptions, which forces a given set of next-state functions to be included in *S*. The assumptions are derived as follows: each auxiliary variable is assigned value 1 (value 0) if the corresponding next-state function is included in (excluded from) *S*. The resulting SAT instance is UNSAT if and only if the corresponding set *S* is *m*-inductive.

Note that the same SAT instance can be used to check *m-inductiveness* of different sets of flops in *S*. For this, only the set of assumptions needs to be changed. This observation is used to efficiently implement the algorithm shown in Figure 12 for finding a minimal *m-inductive* set of next-state functions.

This algorithm can be used also to find *m-inductiveness* of two different sequential machines where the one-to-one correspondence of FFs is known. All corresponding FFs are constrained to be in set *S* at the start and then removed one at time until a minimal set *S* is found. The case where the correspondence between the FFs of the two machine is not known is more difficult and needs further research.

Input: Sequential design, value of *m*
Outputs: A minimal set of next-state functions S
Initialize *S* to contain all next-state functions.
Create SAT instance for given *S* and *m*, as described above.
Assert that this *S* is *m*-inductive (should be always true).
for each node *s* belonging to *S* {
  S = *S* – *s*
  If (the resulting *S* is not *m*-inductive)
    S = *S* + *s*
}
return *S*

Figure 12: *Pseudo-code to compute a minimal m-inductive set.*

Notice that it is a much harder problem to find *S* for verification of two machines where correspondence between FFs is not known. One cannot start from all FFs and delete them one at a time since the behavior of many FFs will not match. A different algorithm is needed to find the subset of FFs that can be in *S*.

V. *Using M*-inductive Sets

This section generalizes the work in [18] resulting in a stronger verification technique and a larger set of sequential ODCs for the optimization of sequential circuits. In [18], it is shown that machine *A* is sequentially equivalent to machine *B* when the combinational equality $A^k = B^1 A^{k-1}$ holds. In such a verification, it is required that all outputs (POs and final *NS* outputs) of $B^1 A^{k-1}$ to be equal to the corresponding outputs of A^k. The work of [18] can be extended in three ways:

The first and most important is that in many circuits equality of only a subset of *NS* outputs and all POs are sufficient to have *A* = *B*. This results in a more powerful verification because it is enough to prove a subset of *NS* functions are equal in A^k and $B^1 A^{k-1}$. Also, since there is no need to preserve all *NS* outputs of A^k, a larger set of ODCs are computed to optimize the first copy of A^1 in $A^1 A^{k-1}$.

The second extension is that the subset of *NS* outputs that needs to be preserved, can be replaced partially or fully by intermediate nodes. These intermediate nodes are in the same copy of A^1 which has the final *NS* outputs. As before, the preserved subset of intermediate nodes and *NS* outputs are referred to as the subset *S*.

The third extension is to require equality of a subset *S* of intermediate and *NS* nodes plus all PO's in the last *m* copies of A^1 in $B^1 A^{k-1}$ and A^k. The nodes in *S* are the same in all *m* copies of A^1. There is an inverse correlation between the number of copies *m* and the size of *S*. An increase in *m* reduces *S* in the last *m* copies of A^1 whose Boolean functions must be preserved. When *m* is increased by 1, the same set *S* still works for *(m+1)-inductiveness*. However, some of the nodes in *S* may become redundant. At some points all signals in *S* are necessary, no matter how large *m* is.

The following subsection takes advantage of *m*-inductiveness of *A* (with itself) to loosen the requirement $A^k = B^1 A^{k-1}$ for implying *A* = *B*. The next discussion shows how to use *m*-inductiveness during ODC optimization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

A. Verification of m-Inductive Circuits

Definition 6 ($[B^1A^{k-1} = A^k]^{S^m}$ Notation): *Let Machine A be m-inductive under set S. Suppose two machines A and B have the same PIs and POs and the same number of FFs. Use some 1-1 mapping between the FF's of A and B to form B^1A^{k-1} as shown in Figure 5. The notation $[B^1A^{k-1} = A^k]^{S^m}$ for $m \le k-1$ means that all the corresponding POs of B^1A^{k-1} are equal to those of A^k, and the nodes in S are equal in the last m copies of A^1 in B^1A^{k-1} and A^k (note $k > m$).*

In this context, *k* is the number of times *A* is unrolled. The parameter *m* is the number of consecutive cycles POs and nodes in *S* are equal in the mentioned unrolling of the two machines *A* and *B*. In almost all practical cases, we assume $k = m+1$. This assumption may make it easier to follow the proof of the following Lemma.

Lemma 2: *Let Machine A be m-inductive under set S. Consider two sequential circuits A and B that have the same PIs and POs and the same number of FFs. When using some 1-1 mapping between the FF's of A and B to form B^1A^{k-1} and A^k, if $[B^1A^{k-1} = A^k]^{S^m}$ for $m < k$, then $[B^pA^{k-1} = A^{p+k-1}]^{S^m}$ for any $p > 0$.*

Proof: We use induction on *p* to prove this. For the base case, this holds for $p = 1$ as part of our initial hypothesis. For the inductive case, assume it is true for some *p*, i.e. $[B^pA^{k-1} = A^{p+k-1}]^{S^m}$. Apply A^1 on the right to each side of this equation. Equality under S^m still holds since the corresponding nodes in *S* and outputs are equal in the last *m* copies of A^1 on both sides and *A* has the *m-inductive* property on *S*. Thus, $[B^pA^{k-1}A^1 = A^{p+k-1}A^1]^{S^m}$. Regroup A^1's, $[B^pA^1A^{k-1} = A^{p+k}]^{S^m}$. Replace $[A^1A^{k-1}]^{S^m}$ with $[B^1A^{k-1}]^{S^m}$ to get $[B^pB^1A^{k-1} = A^{p+k}]^{S^m}$. Regroup the B^1's, $[B^{p+1}A^{k-1} = A^{p+1+k-1}]^{S^m}$, and by induction, $[B^pA^{k-1} = A^{p+k-1}]^{S^m}$ for any $p > 0$. **QED**

Theorem 3: *Let A have the m-inductive property under set S. Suppose machines A and B have the same PIs and POs. Use some 1-1 mapping between the FF's of A and B to form B^1A^{k-1} and A^k. If $[B^1A^{k-1} = A^k]^{S^m}$, then A = B, when both are initialized with the same arbitrary initial state.*

Proof: Suppose the theorem is false, meaning that $[B^1A^{k-1} = A^k]^{S^m}$ but $A \neq B$. Then, there exists a state s_0, an integer *l*, and a sequence of PIs, $\hat{I} = \{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_l, \ldots\}$ such that at cycle *l*, one of the POs of *A* differs from its corresponding PO of *B*, when both *A* and *B* are given the same starting state s_0. When $[B^1A^{k-1} = A^k]^{S^m}$, also $[B^pA^{k-1} = A^{p+k-1}]^{S^m}$ according to Lemma 2, meaning outputs of *A* and *B* are equal for any $p > 1$. Choose *p* so that $p > l$. This contradicts the existence of the sequence $\hat{I}$ that causes a difference in outputs. Thus, $A = B$. **QED**

When *S* includes all *NS* outputs of a machine *A*, *A* is *1-inductive* with itself under *S*. As a result, Theorem 1 in [18] is a special case of Theorem 3 in this paper where *m*=1 and *S* is the set of all *NS* nodes.

B. Optimization of m-Inductive Circuits

Consider a sequential circuit *A* which has the *1-inductive* property under set *S*. This circuit is to be optimized using a *k*-step unrolling process where the first copy of A^1 in A^k is optimized using ODCs from the last *k*-1 copies of A^1. The optimized circuit is denoted as B^1. According to Theorem 3, if $[B^1A^{k-1} = A^k]^{S^1}$, then *A* = *B*. This means as long as A^k equals B^1A^{k-1} for all the outputs and all pairs *S* in the last copy of the A^1's on both sides, then *A* = *B*. As a result, one only needs to preserve the behavior of all the POs and the nodes in *S*. In particular, the *NS* functions that are not in *S* can be changed.

Now, let *A* be *m-inductive* under *S*. In this case, one needs to preserve nodes of *S* in the last *m* copies of A^1. In order to get a larger don't care set, one can create the unrolling A^k, $k > m$. After optimization of the first A^1 in A^k, A^1 is transformed into B^1 and $[B^1A^{k-1} = A^k]^{S^m}$. Notice that the parameter *m* needed for *m*-inductiveness is different from *k*. The size of *k* in unrolling A^k can be increased or decreased independent of *m*, as long as $k > m$.

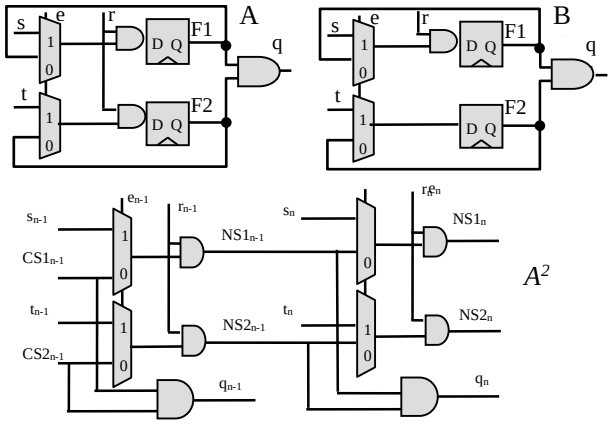


Figure 13: 1-inductive when S is an empty set.

Example 4: The state machine in Figure 13 shows a circuit *A* that is *1-inductive* under the empty set *S*. This was proved in Example 1. A *2-step* unrolling of this circuit, A^2, is also shown (the unrolling is for cycles *n-1* and *n*). The optimization needs to maintain only the POs at time *n*-1 and *n* (not the *next-state* outputs of the last frame) while synthesizing the first copy of A^1. The input r_{n-1} appear in two AND gates while computing output q_n. One of the AND nodes is redundant. Therefore, one instance of the AND gate can be removed. The new optimized machine *B* is also shown in Figure 13. Notice that the next-state function at FF F_2 has changed and is independent of input *r*. This optimization cannot be done using previous sequential optimization techniques that only rely on an unrolling like the ones in [9].

Lemma 3: *Let Machine A be m-inductive under set S. The ODCs computed at any node in the first A^1 under the unrolling A^{k+1} are greater than or equal to the ODCs computed in the*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

corresponding node of the first A^1 under the unrolling A^k, where $k > m$ and all ODCs are computed to maintain m-inductiveness under set S.

Proof: Let N_i be any node in the first A^1 under the unrolling A^k (A^k is shown in circuit C^1 in Figure 5). Let g_i be the global function of that node in terms of the PIs and the *CS* variables. Let D_i be the ODC of that node in terms of inputs and *CS* variables (Global functions and ODCs are further explained in [19]). Any new function $\tilde{g}_i$ can replace g_i as long as $(g_i - D_i) \subseteq \tilde{g}_i \subseteq (g_i + D_i)$. This change in g_i will not be observable at the POs and the subset *S* in the last *m* copies of A^1 in A^k. Since this change is not observable at the last *m* copies of A^1 in A^k, it will not be observable at the POs and the subset *S* of A^{k+1} given the *m*-inductiveness of *A*. As a result the ODC of N_i computed from A^k must be a subset of the ODC of N_i computed from A^{k+1}. **QED**

Lemma 4: *Let Machine A be m-inductive and the set S be empty. The ODCs for any node in the first A^1 in the unrolling A^{m+1} are equal to those computed in the unrolling A^m.*

Proof: Let N_i be any node in the first A^1 in the unrolling A^{m+1}, and g_i and D_i be the global function and ODC respectively of that node in terms of the PIs and *CS* variables. Let $\tilde{g}_i$ be any function such that $(g_i - D_i) \subseteq \tilde{g}_i \subseteq (g_i + D_i)$. If this change is not observable at the outputs of A^{m+1}, it also not observable at the outputs of A^m since outputs of A^m are also outputs of A^{m+1}. This proves ODC computed from A^{m+1} is a subset of ODC computed from A^m. We know from Lemma 3 that ODC computed from A^m is also a subset of the ODCs from A^{m+1}. Thus the sequential ODCs of A^m and A^{m+1} are equal when *S* is an empty set. **QED**

Lemma 4 states that if *S* is an empty set, there is no need to unroll a circuit *A* that is *m-inductive* more than *m* times; additional unrolling does not result in a larger ODC set, e.g. in Figure 8, unrolling beyond A^2 does not give more ODCs. This is interesting because it says Machine *A* has no more ODCs.

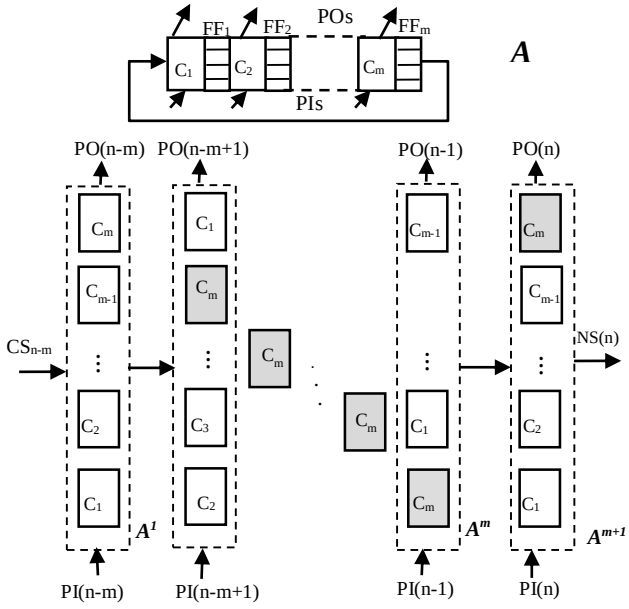


Figure 14: m-stage pipeline with feedback loop

Example 5: Machine *A* in Figure 14 is a pipeline circuit with a feedback loop, where the next-state functions of FF_1 are dependent on the state variables of FF_m. Each stage of the pipeline has its own inputs and outputs as shown. The bottom of Figure 14 shows A^{m+1}, the (*m*+1)-*step* unrolling of *A*. Let *S* be all the next-state nodes in C_m. Observe that *NS* of any C_i at time *n* and all POs at time *n* can be written as functions of the PIs at cycles *n*-*m* to *n* and *NS* of C_m, at cycles *n*-*m* to *n*-1. Therefore by Theorem 1 this circuit is *m-inductive* with itself where the set *S* includes only next-state nodes of stage *m* (FF_m). Thus, an optimization of the first copy of A^1 in the unrolling A^{m+1} only needs to maintain the global next-state functions of FF_m in the last *m* copies of A^1 in A^{m+1} (*NS* of grey boxes in Figure 14) in addition to the POs.

C. Retiming

In this section, **w**e illustrate how the *m-inductive* property can be used to move flip flops across combinational logic in any circuit to retime it. We focus on the movement of FFs across one level of combinational logic. FFs can be moved forward or backwards in a 3-step transformation process applied to a sub-circuit of a larger design as shown in Figure 15. The sub-circuit *A* is transformed to *B*, then *C*, and finally *D*. The sub-circuit *A* is a 1-stage pipeline without loops, which includes the FFs to be retimed and the one level combinational logic gates involved in the retiming. For simplicity, it is assumed that the circuit is composed of NAND gates (Circuit *A* in Figure 15), but these transformations apply to any gate.

In Section III, it was shown that any *m*-stage pipeline without loops is *m*-inductive under the empty set. Thus, circuit *A* is 1-inductive under the empty set, which is key to the three transformations. The first transformation, shown at the top of Figure 15, duplicates FF's in sub-circuit *A* to reach a new sub-circuit *B* such that the output of each flip flop goes to exactly one NAND gate. The duplicated FF's have the same initial state as the parent flop.

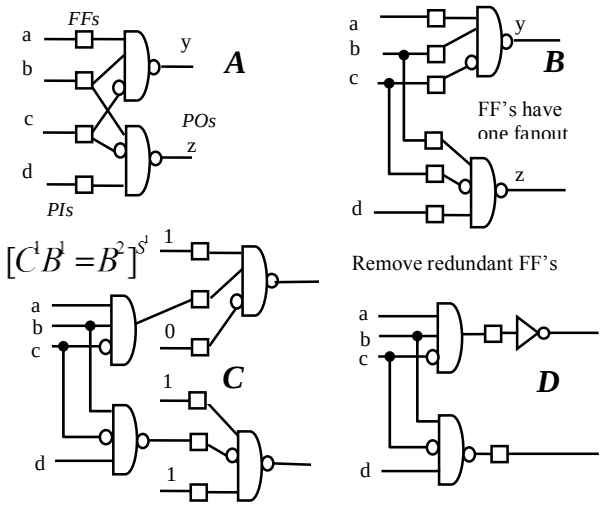


Figure 15: Retiming using 1-inductive Property

Proposition 1*: Sub-circuits A and B in are equivalent.*

Proof: Notice that POs at time *n* are only dependent on inputs at time *n*-1. Both circuits have the same inputs and outputs. If the POs of *A* and *B* are equal at time *n*-1, they are also equal at time *n*. Therefore, sub-circuits *A* and *B* are 1-inductive under

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

the empty set *S*. Also, outputs of *A* and *B* are equal during the first cycle given the proper initialization of FF's. Therefore, using Theorem 2, *A*=*B*. **QED**

The second transformation converts the circuit *B* into *C* as shown in Figure 15. The particular transformation, applied to get circuit *C* from *B*, copies each template NAND gate (original NAND gate) in circuit *B* to the input of one of the FFs feeding it while adjusting the polarity of that NAND gate as explained later on. The template NAND gates are copied to the inputs of the FFs to allow retiming. It also sets the inputs to the rest of the FF's feeding the template NAND gate to non-controlling constants as shown in circuit *C* at the bottom of Figure 15.

Figure 16 justifies the conversion from *B* to *C* in Figure 15. It shows the unrolling B^2 and C^1B^1. The changes in C^1 are done such that the global functions of POs at time *n+1* in terms of PIs at time *n*, are preserved. The global functions of POs in the unrolling B^2 are $y_{n+1}=\overline{\left(a_n b_n \overline{c_n}\right)}$ and $z_{n+1}=\overline{\left(b_n \overline{c_n} d_n\right)}$. When a NAND gate is copied to the input of the FF fed by *b*, the output bubble of the copied gate is removed and non-controlling constants are added to preserve the overall function $y_{n+1}=\overline{\left([1]\left[a_n b_n \overline{c_n}\right][0]\right)}$. The other NAND gate is copied to the FF fed by *c*, the output bubble of the copied gate is retained and non-controlling constants are added to get $z_{n+1}=\overline{\left([1]\left[\overline{\left[b_n \overline{c_n} d_n\right]}\right][1]\right)}$.

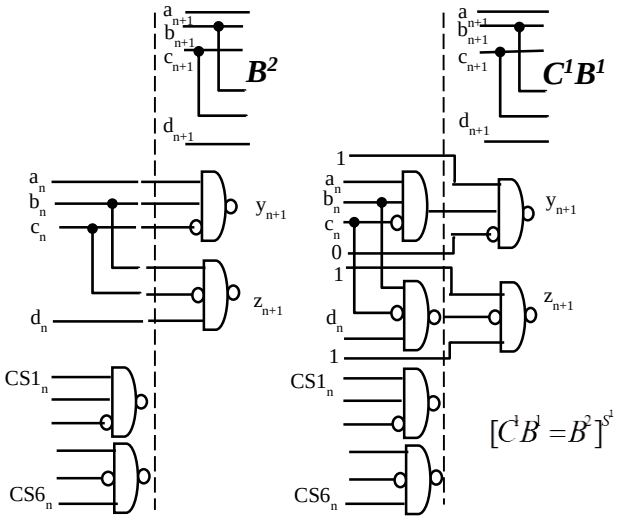


Figure 16: Unrolling B^2 and C^1B^1

Proposition 2: *Sub-circuits B and C in are equivalent.*

Proof: Assume the corresponding outputs of the two circuits are equal at time *n*-1. Because the outputs at time *n*, are only dependent on the inputs at time *n*-1, and by construction the Boolean functions of *B* and *C* in terms of inputs at time *n*-1 are equal, the corresponding outputs are equal at time *n*. Sub-circuits *B* and *C* are 1-inductive under the empty set *S*. Also, outputs of *B* and *C* are equal during the first cycle since there was no change to the initial state of the FFs and the gates fed by them. Therefore, using Theorem 2, *B*=*C*. **QED**

The final transformation in Figure 15 removes FF's whose initial states match the constant feeding them. The constant is propagated to the NAND gate and the NAND gate is simplified to get circuit *D*. Removing the constant FF's is also justified by the 1-inductiveness of circuits *C* and *D* under the empty set.

After the three-step transformation, the FF's have been moved forward through one stage of the NANDs. This process can be repeated many times in the forward or backward direction. In the backward direction, the transformations are done in reverse order.

In the construction of circuit *C*, there can be two cases where the constant feeding a FF is different from its initial value; the input is constant 1 but the initial state is 0 or the input is constant 0 but the initial state is 1. These two cases can happen in multiple places and as a result the affected FFs cannot be removed. All FF whose input is constant 1 and initial state is 0 have similar behavior and are duplicates. The duplicate FFs can be removed by keeping one FF and using it to replace other FFs by adding fanouts to its *Q* output as necessary. An inverter added in front of the *Q* of the FF captures the case where the constant feeding the FF is 0 and the initial state is 1. Fanouts from this invertor can replace all FFs whose input is 0 and initial state is 1.

VI. EXPERIMENTAL RESULTS

The proposed algorithm for finding a minimal set *S* of next-state nodes for *m*-inductiveness of a circuit with respect to itself was implemented in ABC [4] and run on multiple ISCAS, MCNC, and industrial benchmark circuits.

Table 1 shows experimental results on 10 designs. The first column shows the names of the circuits. The second column shows the number of flip flops *FF*, primary inputs *I*, and primary outputs *O* for each design. Columns 3 to 7 show *s*, the cardinality of set *S*, computed by the algorithm for $1 \le m \le 5$, *V* the number of variables used to set up the SAT problems, and *T* the runtimes in seconds to compute *S*.

NAME	FF/I/O	M=1	M=2	M=3	M=4	M=5
s13207	FF=669 I=31 O=121	s=392 V=11602 T=3s	s=256 V=15944 T=2s	s=218 V=20286 T=2s	s=197 V=24628 T=2s	s=188 V=28970 T=2s
s9234	FF=211 I=36 O=39	s=176 V=4622 T=1s	s=148 V=6472 T=1s	s=140 V=8322 T=1s	s=130 V=10172 T=2s	s=129 V=12022 T=1s
s35932	FF=1728 I=35 O=320	s=1728 V=34376 T=33	s=848 V=47788 T=276s	s=699 V=61200 T=191	s=499 V=74612 T=367s	s=470 V=88024 T=847s
s38417	FF=1636 I=28 O=106	s=1432 V=31040 T=92s	s=1079 V=43182 T=115s	s=1023 V=55324 T=177s	s=978 V=67466 T=256s	s=969 V=79608 T=334s
key	FF=228 I=258 O=193	s=76 V=7562 T=2s	s=39 V=10694 T=2s	s=39 V=13826 2s	s=39 V=16958 T=3s	s=39 V=20090 T=3s
Min-max32	FF=96 I=35 O=32	s=32 V=2872 T=1s	s=32 V=4084 T=2s	s=32 V=5296 T=2s	s=32 V=6508 T=5s	s=32 V=7702 T=4s
Cpb.-32.4	FF=32 I=33 O=33	s=0 V=1066 T=0.1s	s=0	s=0	s=0	s=0
s641	FF=19 I=35 O=23	s=15 V=678 T=0.01s	s=8 V=956 T=0.01s	s=8 V=1234 T=0.01s	s=8 V=1512 T=0.01s	s=8 V=1790 T=0.01s
s1423	FF=74 I=17 O=5	s=73 V=1540 T=0.1s	s=71 V=2157 T=0.1s	s=69 V=2774 T=0.1s	s=68 V=3391 T=0.1s	s=68 V=4008 T=0.2s
s38584	FF=1452 I=12 O=278	s=1239 V=31136 T=38s	s=1096 V=43522 T=59s	s=1035 V=55908 T=190s	s=981 V=68294 T=298s	s=947 V=80680 T=552s
Total FF	6145	s=5163 84%	s=3577 58%	s=3263 53%	s=2932 48%	s=2850 46%

Table 1: *Finding a minimal S for different values of m*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

All the runtimes are on a single thread of a 2.2 GHZ Intel i7 machine.

The computation of a minimal set *S* starts with *m*=1. Once found, it is used as a starting set for *m*=2. This iteration continues until *m*=5. The last row in the table shows the total number of flip flops in all designs and the total number of elements in the set *S* for each value of *m*. The number of elements in *S* is 16% smaller than the total number of flops when *m* = 1. The set *S* is empty for circuit Cpb-32.4 when *m* = 1. We did not increase *m* beyond 1 for this circuit. The largest percentage decrease in *NS* nodes is when *m* is increased from 1 to 2. In this case, the number of elements in *S* is 42% smaller than the number of FFs. When *m* goes from 4 to 5, the percentage reduction in size of *S* is only 2%. Using values greater than 5 for *m* does not materially reduce the size of *S*. As shown in the case of data paths, there is no unique solution for *S* and some minimal sets can be smaller than the others. The table shows the computation for obtaining only one minimal set per design; it does not enumerate all minimal solutions.

In another experiment, 1-inductive properties of a design are used to optimize its combinational logic. First a minimal set *S* is computed for a design *A* so that it is 1-inductive under *S*. Each design *A* is then unrolled twice to form A^2. A logic optimization algorithm is applied to optimize the first copy of A^1 in A^2 to form B^1A^1. The optimization algorithm maintains the logic functions of all outputs of A^2 plus those next-state functions in *S* of the last copy of A^1 in A^2. The next-state functions not in *S* in the last copy are unconstrained. (One can think of these as floating outputs or consider their external don't care to be the full Boolean space.) The optimization algorithm utilizes existing algorithms in ABC to use ODCs from the second copy of A^1 in A^2 to optimize the first copy. This creates a new design B^1. B^1 is the combinational logic of a new sequential machine *B*. It is sequentially equivalent to *A* because $[B^1A^1 = A^2]^S$.

The algorithm is implemented in ABC and was applied to a large set of ISCAS and industrial designs. To judge the additional power of this optimization, we compared a) the literal count (in factored form) of each design when optimized combinationally, to b) the case when the design was optimized using 1-inductiveness. On average, 25% of designs showed good improvement when 1-inductiveness was used.

NAME	S/FF	INIT	COMB/T	1-IND/T	%LIT-R
s13207	392/669	4007	3789/0.5s	3338 / 6s	11.9
s9234	176/211	2582	2402/0.5s	2140 / 1s	12.4
s38417	1432/1636	13418	13234/1.5s	13166/212s	0.5
Ind1	598/1140	8863	7083/1.6s	6589 / 181s	7
Ind2	257/669	19517	19384/7.8s	19017 / 65s	1.9
Ind3	150/402	15040	14880/2.9s	14513 / 36s	2.5
Ind4	514/594	5455	5322/1.6s	5188/7s	2.5
Ind5	514/594	5425	5292/1.6s	5158 / 7s	2.5
Ind6	349/426	3474	3458/1.1s	3376 / 3s	2.4
Ind7	125/129	1312	1312/0.21s	1292 / 0s	1.5

Table 2: *Optimizing 1-inductive circuits*

Table 2 shows some of the designs where good improvement was obtained. The number of FFs in *S* and the total number of FFs for each design are shown in the second column of Table **2** (labeled *S/FF)*. Column 3 (labeled *INIT*) shows the initial literal count of each circuit measured in factored form. Column 4 (labeled *COMB*) shows the literal count and runtimes when only combinational optimization is used and Column 5 shows the corresponding values for optimizations using 1-inductiveness. The percentage improvement in literal count (1-IND vs COMB) is shown in Column 6. Some industrial designs showed significant reduction in literal count when 1-inductiveness was used, e.g. Ind1 shows 7% reduction in literal count. Notice that this extra sequential optimization is obtained independent of the initial states of the design. Also note that the circuits that show the most improvement in literal count also show big reductions in the sizes of *S* relative to the total FFs in the design.

When the first copy of A^1 in A^2 is optimized, some flexibility in optimization may come from ODCs in the second copy of A^1 and some from the fact that *S* is a subset of the FFs. To determine the contribution from ODCs, we set *S* to be all the flops (all NS functions are maintained in A^2) and re-ran the optimization on all designs. Since the resulting final literal count of each design was very close to the combinational results, we conclude that most of the improvements in literal count shown in Table 2 are due to the reduction in the size of *S* through the use of 1-inductiveness.

NAME	FF	S1	1-IND	S2	2-IND
s13207	669	392	3338	256	3347
s9234	211	176	2140	148	2134
s38417	1636	1432	13166	1096	12977
Ind1	1140	598	6589	596	6596
Ind2	669	257	19017	5	19019
Ind3	402	150	14513	9	14513
Ind4	594	514	5188	389	5173
Ind5	594	514	5158	398	5133
Ind6	426	349	3376	241	3376
Ind7	129	125	1292	108	1292

Table 3: *Comparing 1-inductiveness and 2-inductiveness*

Another set of experiments was run to find the sets *S* when 2-inductiveness was used to optimize the first copy of A^1 in A^3. In this case, the next-state functions in *S* of *both* the second and third copies of A^1 in A^3 need to be maintained. The comparison of 1-inductive and 2-inductive optimization is shown in Table 3. S1 and 1-IND show the number of FFs in S and literal count when 1-inductive optimization is performed while S2 and 2-IND show the number of FFs in S and literal count when 2-inductive optimization is performed. Although, one can build special circuits where only 2-inductive optimization reduces literal count, we did not see much improvement compared to 1-inductive optimization in the set of designs we tested. We do not expect any additional improvements for *m* values greater than 2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2015.2481860, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

VII. Relation to Previous Work

In the synthesis area, there has been a lot of work done on exploiting the flexibility available in implementing combinational and sequential logic networks. A good survey for early work covering a broad set of methods can be found in [5]. More recent work has been on how to compute or represent and use don't cares efficiently [15]. Combinational methods were developed in [23] and [16] to use "observable" simulation vectors to find candidate node pairs which are always equal or always not equal under such inputs. Each such pair, if proved by SAT, can be merged to simplify the circuit.

Combinational methods have been extended for use in sequential circuits using basically two approaches 1) deriving and using unreachable state information as external don't cares, and 2) unrolling a sequential circuit a fixed number of time-frames and using combinational methods on the unrolled combinational circuit. The first method depends on a given set of initial states while the second depends only on the immediate surrounding circuit. These approaches will be further discussed in the following.

Approach 1. Examples of this are the various ways used to compute some unreachable state information. Early work used BDDs to compute the exact set of reachable states or to compute an over-approximation. Later methods were based on the improvements in SAT solving. Many used simulations starting at the initial states to find candidate pairs of nodes in a circuit where they are always equal or always not equal. Thus, if such a pair equivalence can be proved to hold forever, say by *k*-step induction[14][22], then the nodes can be merged, effectively simplifying the circuit using don't cares based on unreachable state information. The set of states where the pair equalities hold is an invariant, which is an over-approximation of the set of all reachable states. The trouble with finding the candidate pairs is that it is done by simulation, which cannot detect non-observability.

The other problem with this is that the usual proof that a set of pair-equalities holds is almost always done by *k*-step induction. This checks that a set of equalities hold for the initial *k*-cycles, and assumes them to hold at some arbitrary time *t* for the previous *k* time-frames. Then it tries to prove they all hold at time *t*+1. However, this can have inductive "leaks" where an equality can be valid but can't be proved by induction. A way to enhance the ability to prove pair-equalities by induction is to use a history of synthesis [10]. A method not based on induction is called *speculation* where an initial set of candidate pairs is derived and used to simplify a circuit creating a speculated circuit [11][12]. Each candidate pair becomes a proof obligation. If all these can be proved somehow, then the speculated circuit is a valid simplification. Otherwise, pairs that can't be proved are thrown out and a new speculated circuit is tried.

Approach 2. In the second approach, there is no dependence on the initial states, just the environment of the node to be minimized [7][8][9][21][3] is used. In these works, the sequential machine is unrolled a few frames and the result is treated as a combinational circuit. The nodes to be minimized are usually those in the first time-frame. The resulting combinational logic of the first frame is then used as the combinational logic of a new sequential machine. Thus methods like redundancy removal [1], redundancy insertion and removal [6], and local windowing [15] can be used. Improved methods in combinational equivalence [13] can be used to enhance these methods."

In [16], their combinational approach of using ODC's in addition to SDC's to find candidate node merges, is applied to sequential circuits by unrolling a few cycles. They do not say how they manage sequential equivalence of the synthesized logic, but probably do this by synthesizing B^1 in $B^1A^{k-1} = A^k$ for a *k*-step unrolling. Otherwise, they would have to unroll to get A^k, do a single node merge across *k* frames forming $\tilde{B}^k$, and then repeat this for the next pair of nodes to merge. Either way, as in [18], they assume that the observable outputs of the unrolling are all POs for *k*-steps plus the final flop inputs at the k^{th} step.

As far as we can see, none the methods using Approach 2 seem to be concerned about the legitimacy of this, i.e. the old and new machines should be sequentially equivalent, which is proved in [18].

All of these combinational methods extended to a sequential circuit by an unrolling are compatible with the methods of this paper. Namely *m*-inductiveness affects the definition of what is observable. An unrolled circuit's outputs are redefined. The current paper says that, for example, if the circuit has the 1-inductiveness property, then only a subset of the flop inputs of the final frame need be considered as observable (plus of course all POs of each frame).

A result from [18], is that if $B^nA^k = A^{n+k}$ holds, then the synthesized sequential circuit *B* derived from B^1 is a safe replacement [21] for the original one *A*. The concept of *m*-inductiveness redefines what it means for $B^nA^k = A^{n+k}$ to hold. Safe replaceability is useful because it implies that every synchronization sequence for the original design also synchronizes the replacement circuit.

VIII. Conclusions and Future work

A new concept, *m*-inductiveness over a set of nodes *S*, is introduced. The set *S* is related to the structure of the circuit. The proposed SAT-based algorithm finds a minimal set of next-state nodes *S* that exhibit *m*-inductiveness property in a given circuit. Experimental results show that for *m*=1, the size of *S* is about 16% smaller than the total FFs. The *m*-inductiveness property can be used for synthesis as well as verification of sequential machines. It is shown, that an optimization algorithm that uses the flexibility of *m*-inductiveness significantly reduces the area of a design on a good fraction of the designs tested. It can remove redundancies that could not be removed otherwise. When there is one to one correspondence between FFs of two sequential machines, new sufficient formulations are provided to convert the SEC of the two machines to CEC. A general algorithm for finding *m-inductiveness* for two machines where the correspondence between FF's is unknown is currently under investigation.

References

[1] Abramovici M., Iyer M.A., "One-Pass Redundancy Identification and Removal," Test Conference, 1992. Proceedings., International Sept. 20-24 1992 Page(s):807

[2] K.A. Barlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, R. Rudell, A. Sangiovannni-Vincentelli, and A. Wang, "Multilevel Logic Minimization Using Implicit Don't Cares", IEEE Trans. Computer-Aided Design, vol. CAD-7, pp. 723-739, June 1988

[3] P. Bjesse and K. Claessen. "SAT-based verification without state space traversal". Proc. FMCAD'00. LNCS, Vol. 1954, pp. 372-389.

[4] R.K. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", in Proc. CAV, 2010, pp.24-40.

[5] Robert K. Brayton, Ellen M. Sentovich: Network Hierarchies and Node Minimization. IEICE Transactions 78-D(3): 199-208 (1995)

[6] Shih-Chieh Chang, M. Marek-Sadowska, **"**Perturb And Simplify: Multi-level Boolean Network Optimizer**,"** International Conference on Computer-Aided Design, 1994, November 6-10, 1994 Page(s):2 - 5 .

[7] M. A. Iyer, D. E. Long, and M. Abramovici, "Identifying sequential redundancies without search," *Proc. DAC'96*, pp. 457-462.

[8] M. A. Iyer, D. E. Long, and M. Abramovici, ''Surprises in sequential redundancy identification,'' *Proc. EDTC'96*.

[9] A. Mehrotra, S. Qadeer, V. Singhal, R. K. Brayton, A. Aziz, and A. L. Sangiovanni-Vincentelli. "Sequential optimization without state space exploration". *Proc. ICCAD'97*, pp. 208-215.

[10] A. Mishchenko and R. K. Brayton, "Recording synthesis history for sequential verification", *FMCAD'08*, pp. 27-34.

[11] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, "Exploiting suspected redundancy without proving it". *Proc. DAC'05*.

[12] Hari Mony, Jason Baumgartner, Alan Mishchenko, Robert K. Brayton: "Speculative reduction-based scalable redundancy identification." DATE 2009: 1674-1679

[13] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking", *Proc. ICCAD '06*, pp. 836-843.

[14] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", *Proc. ICCAD'08*, pp. 234-241.

[15] Alan Mishchenko, and Robert K.Brayton, "SAT-Based Complete Don't-Care Computation for Network Optimization." CoRR abs/0710.4695 (2007)

[16] Stephen Plaza, Kai-Hui Chang, Igor L. Markov, Valeria Bertacco: Node Mergers in the Presence of Don't Cares. ASP-DAC 2007: 414-419

[17] A. Saldanha, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Multi-level logic optimization using don't cares and filters", *Proc. DAC'89.*

[18] H. Savoj, A. Mishchenko, and R. Brayton, "Combinational techniques for sequential equivalence checking". *Transactions on Computer Aided Design of Integrated Circuits Systems (Volume 33 Issue 2),* Feb. 2014, pp 305-317.

[19] H. Savoj, "Don't Cares in Multi-Level Network Optimization". PhD Thesis, University of California Berkeley.

[20] H. Savoj and R. K. Brayton, "The Use of Observability and External Don't Cares for the Simplification of Multiple-level Networks," in Proc. Design Automation Conference, June 1990.

[21] V. Singhal and C. Pixley. "The verification problem for safe replaceability", *Proc. CAV'94*, LNCS, Vol.818, pp. 311-3

[22] K. van Eijk, C.A.J.: Sequential equivalence checking without state space traversal. In: Proceedings of the conference on Design, automation and test in Europe (DATE). pp. 618–623. IEEE (1998)

[23] Qi Zhu, Nathan Kitchen, Andreas Kuehlmann, Alberto L. Sangiovanni-Vincentelli: SAT sweeping with local observability don't-cares. DAC 2006: 229-234

ACKNOWLEDGMENT

This work was partly supported by SRC contract 1875.001 and NSA grant 'Enhanced equivalence checking in crypto-analytic applications'. We also thank industrial sponsors of BVSRC: Altera, Atrenta, Cadence, IBM, Intel, Jasper, Microsemi, Real Intent, Synopsys, Tabula, and Verific for their continued support.

Hamid Savoj received a B.S. degree in EE from Caltech in 1987 and the Ph.D. degree in EECS from UC-Berkeley in 1992. He cofounded Magma Design Automation, an Electronic Design Automation company, in 1997 and was Senior Vice President of Engineering at Magma until 2006. From 2006 to 2010, he worked on low power IP and software solutions at Envis where he was the CTO. He cofounded Reflektion, an ecommerce company, in 2012. He is currently a visiting scholar at the University of California in Berkeley. He has published some 30 technical papers and has been awarded 9 patents. Dr Savoj received Artur Major Prize in Engineering from California Institute of Technology in May 1987 and Earle C. Anthony Fellowship from University of California at Berkeley for the academic year 1987-1988. His research interests are in logic synthesis, formal verification, and machine learning.

Alan Mishchenko graduated from Moscow Institute of Physics and Technology (Moscow, Russia) in 1993 with MS and received his PhD from the Glushkov Institute of Cybernetics (Kiev, Ukraine) in 1997. From 1998 to 2002 he was an Intel-sponsored visiting scientist at Portland State University. Since 2002, he has been an Associate Research Engineer in the EECS Department at UC Berkeley. Dr. Mishchenko shared the D.O. Pederson TCAD Best Paper Award in 2008 and the SRC Technical Excellence Award in 2011 for work on ABC. His research interests are in developing computationally efficient methods for synthesis and verification.

Robert Brayton received his Ph.D. degree in mathematics from MIT in 1961. He was a member of the Mathematical Sciences Department of the IBM T. J.Watson Research Center until he joined the EECS Department at Berkeley in 1987. He is a Fellow of the IEEE and a member of the National Academy of Engineering. Notable awards include: IEEE Emanuel R. Piore (2006); ACM Kanallakis (2006); European DAA Lifetime Achievement (2006); EDAC/CEDA Phil Kaufman (2007); D.O. Pederson best paper in Trans. CAD (2008); ACM/IEEE A. Richard Newton Technical Impact in EDA (2009); Iowa State University Distinguished Alumnus (2010); SRC Technical Excellence (2011); and the ACM/SIGDA Pioneering Achievement (2011). Prof. Brayton held the Buttner Chair and the Cadence Distinguished Professorship of Electrical Engineering and is currently a Professor in the Graduate School at Berkeley.